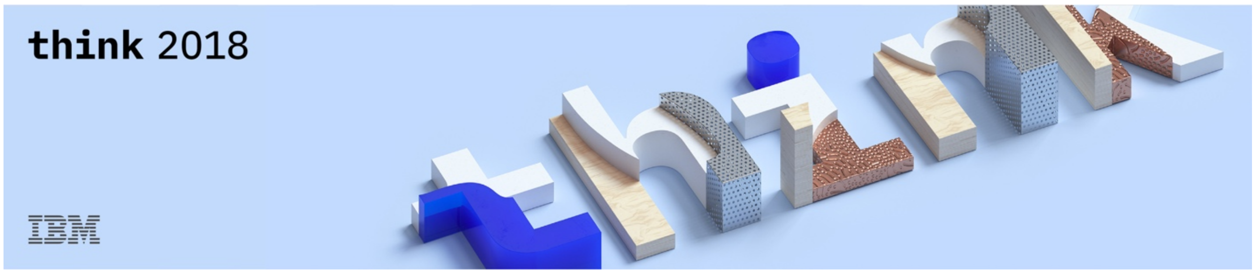


Lab Center – Hands-on Lab

Session 3149

Develop and Deploy Your First Private Cloud Application Using Microservice Builder and IBM Cloud Private

Kyle Miller, IBM

Jose Ortiz, IBM

Table of Contents

Disclaimer 3

Introduction 5

Pre-requisites 6

GitHub setup 7

Creating a GitHub ID 7

Creating a GitHub organization 8

Creating a GitHub personal access token 9

Adding OAuth integration in GitHub 10

Creating a starter application 12

Fork IBM's sample application 12

Explore the sample application 13

Deploy Microservice Builder Pipeline on IBM Cloud Private 14

Set up a registry secret 14

Log in to the IBM Cloud Private web console 15

Deploy Microservice Builder Pipeline using a Helm chart 16

Build and update your application 20

Log in to your new Jenkins environment 20

Build your sample application 20

Test your running sample application 21

Modify and redeploy the sample application 22

Conclusion 24

We Value Your Feedback! 25

Disclaimer

Introduction

IBM Cloud Private (ICP) is an application platform for developing and managing on-premises, containerized applications. It is an integrated environment for managing containers that includes the container orchestrator Kubernetes, a private image repository, a management console, and monitoring frameworks. IBM Cloud Private also includes a graphical user interface which provides a centralized location from where you can deploy, manage, monitor, and scale your applications.

Microservice Builder greatly accelerates and simplifies your software development and deployment process by providing a turnkey continuous delivery pipeline for cloud-native applications and services. Integration with IBM Cloud Private enables delivery of releases to a production-ready, enterprise-quality private cloud in your own data center in minutes instead of days or weeks.

In this hands-on lab designed for new users, you will learn to build a basic microservice-based application, then deploy and update that service to an ICP environment with lightning-fast speed. You will achieve by following these basic steps:

- Creating the pre-requisite setup in GitHub
- Forking and investigating IBM’s sample node-js application in GitHub
- Configuring and deploying Microservice Builder Pipeline using the default Helm chart that is available in IBM Cloud Private
- Viewing the application that was automatically deployed to your IBM Cloud Private cluster from your GitHub repository
- Updating the application and initiating a new build event, which will rebuild the application based on your changes, and then automatically redeploy the new version to your IBM Cloud Private cluster

Pre-requisites

This exercise requires that you have the following available to you, which are provided here at THINK 2018 by the virtual machines that have been provisioned for you:

- An IBM Cloud private 2.10.1 environment
- Outbound access to the internet from the IBM Cloud Private environment

This exercise also requires you to be able to access repositories on github.com using your GitHub ID and a GitHub organization. Instructions for creating both are provided if you do not already have them.

The user credentials for the virtual machine you are using are:

Username: skytap

Password: A1rb0rn3

The administrative credentials for the IBM Cloud Private cluster installed on the virtual machine are:

Username: admin

Password: admin

GitHub setup

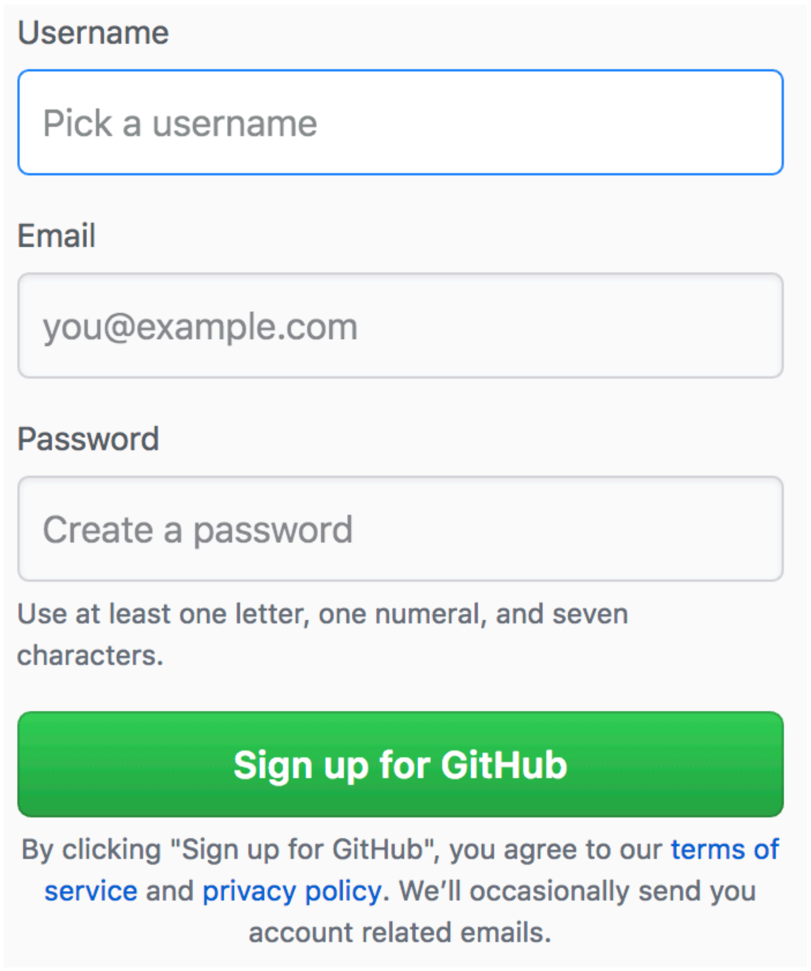
Microservice Builder on IBM Cloud Private is designed to integrate with a git repository management system, such as GitHub, GitHub Enterprise, or GitLab. In this exercise, you will integrate Microservice Builder with GitHub, so you will need a GitHub ID, a GitHub organization, and some security credentials configured to complete the lab.

For future reference, these GitHub setup requirements are documented in the IBM Knowledge Center here: https://www.ibm.com/support/knowledgecenter/SS5PWC/pipeline.html

Creating a GitHub ID

If you do not already have a GitHub ID, you will need to create one.

1. Open a web browser and navigate to www.github.com



Use at least one letter, one numeral, and seven characters.

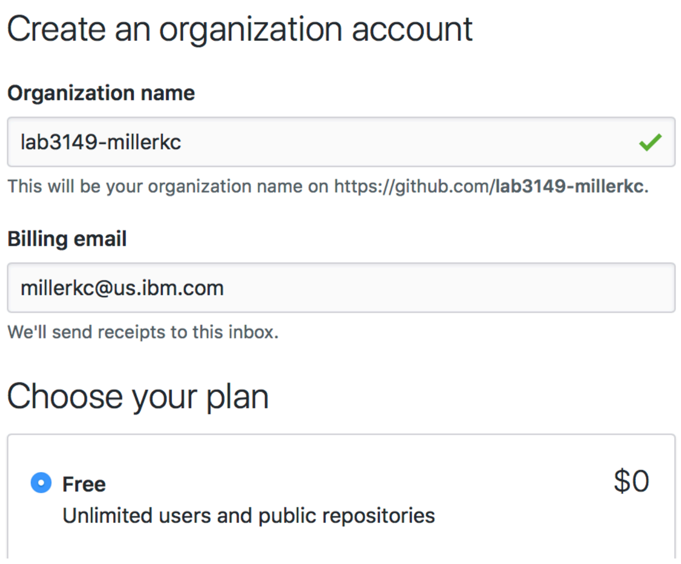
By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.

2. Enter a unique username of your choice
3. Enter your email address
4. Enter a password of your choice
5. Click the **Sign up for GitHub** button
6. Check your email for an email titled “[GitHub] Please verify your email address.”
7. Click the link provided in the email to verify your account

Creating a GitHub organization

You will need to have access to an “organization” in GitHub to integrate with Microservice Builder. To create a new organization:

1. Expand the user menu, represented by your user icon in the top right corner (this may be a generic icon if you just created an account) and click **Settings**.
2. Click **Organizations** in the menu on the left.
3. Click the **New organization** button near the top right.
4. Enter a name for your organization in the **Organization name** field, such as *lab3149-<username>,* where <username> is your github username.
5. Enter your email address in the **Billing email** field. Note that no billing will occur, since you will be choosing a free organization.
6. Leave the **Free** option selected in the **Choose your plan** section.


Create an organization account

Organization name

lab3149-millerkc ✓

This will be your organization name on https://github.com/**lab3149-millerkc**.

Billing email

millerkc@us.ibm.com

We'll send receipts to this inbox.

Choose your plan

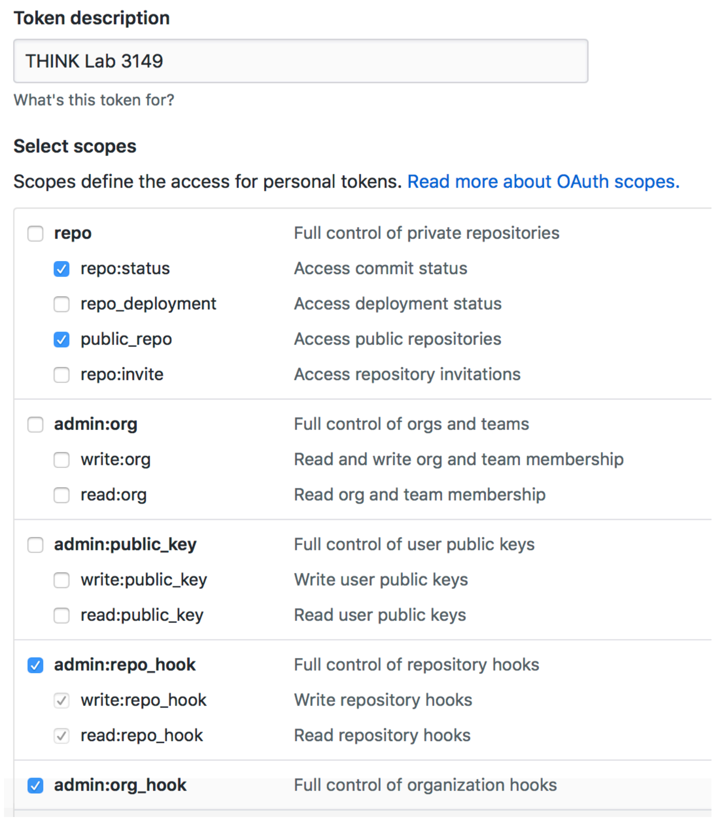
● **Free** $0
Unlimited users and public repositories

7. Click the **Create organization** button.
8. You do not need to add any members to your organization, since you are a member by default.

Creating a GitHub personal access token

A personal access token enables you to create automated integrations with GitHub, so that you don't need to interactively provide a username and password.

1. Create a text file in your VM to take note of the credentials you are about to create. You will need to revisit them later in the exercise, and some values, such as your personal access token, can only be viewed once.
    a. You can create a new text document by selecting **Applications > Accessories > Text Editor** from the menu, or by using the console-based text editor of your choice. vim, nano, and pico are available in the terminal.
2. In the web browser window where you are logged in to GitHub, use the user menu near the top right to select **Settings** again.
3. Click **Developer settings** in the left menu.
4. Click **Personal access tokens** in the left menu.
5. Click the **Generate new token** button.
6. You will be prompted to confirm your password. Enter your GitHub password and click **Confirm password** to continue.
7. Enter "THINK Lab 3149" in the **Token Description** field.
8. Check the boxes next to **repo:status**, **public_repo**, **admin:repo_hook**, and **admin:org_hook** in the "Select scopes" table.



Token description

THINK Lab 3149

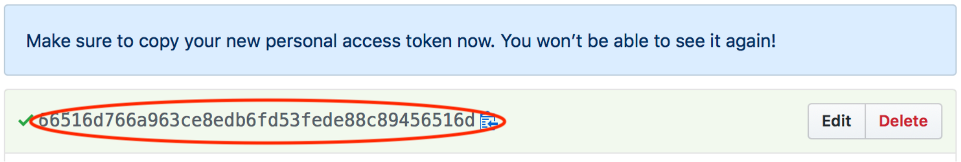
What's this token for?

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☐ **repo**	Full control of private repositories
✓ repo:status	Access commit status
☐ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
☐ repo:invite	Access repository invitations
☐ **admin:org**	Full control of orgs and teams
☐ write:org	Read and write org and team membership
☐ read:org	Read org and team membership
☐ **admin:public_key**	Full control of user public keys
☐ write:public_key	Write user public keys
☐ read:public_key	Read user public keys
✓ **admin:repo_hook**	Full control of repository hooks
✓ write:repo_hook	Write repository hooks
✓ read:repo_hook	Read repository hooks
✓ **admin:org_hook**	Full control of organization hooks

9. Click **Generate token**.


Make sure to copy your new personal access token now. You won't be able to see it again!

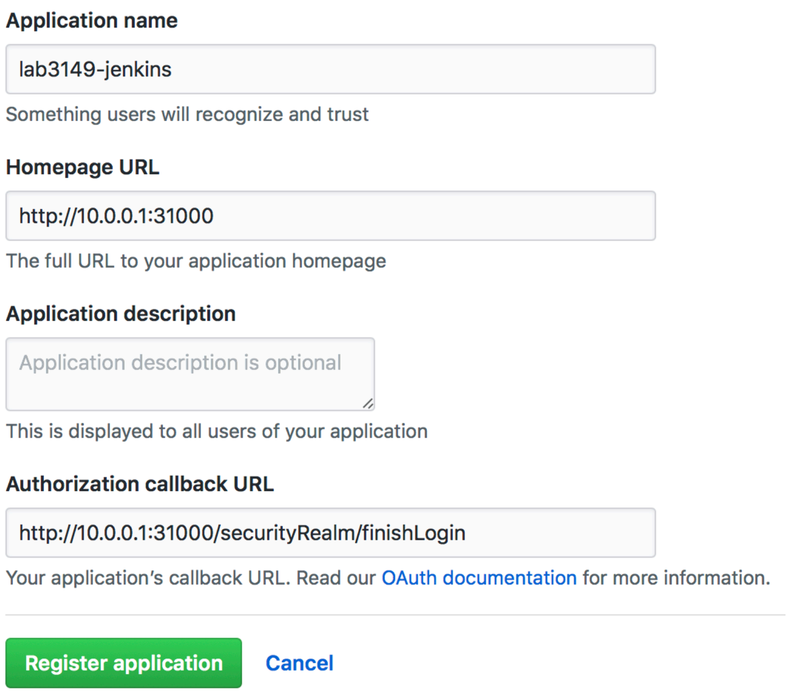
66516d766a963ce8edb6fd53fede88c89456516d

10. Copy the string that is now displayed in the green field and paste it into the text document that you created previously, so it will be available for reference later.

Adding OAuth integration in GitHub

Adding OAuth integration will allow you to log in to the Jenkins instance deployed by Microservice Builder using your GitHub credentials.

1. In the web browser window where you are logged in to GitHub, click **OAuth Apps** in the left menu. (If you do not see OAuth apps in the left menu, navigate back to Settings > Developer settings.)
2. Enter "lab3149-jenkins" in the **Application Name** field.
3. Enter "http://10.0.0.1:31000" in the **Homepage URL** field. 10.0.0.1 is the IP address of your IBM Cloud Private instance, and 31000 is the default port that the Jenkins environment deployed by Microservice Builder listens on.
4. Enter "http://10.0.0.1:31000/securityRealm/finishLogin" in the **Authorization callback URL** field. This is the URL used by OAuth to complete your login to the local Jenkins environment.


Application name

lab3149-jenkins

Something users will recognize and trust

Homepage URL

http://10.0.0.1:31000

The full URL to your application homepage

Application description

Application description is optional

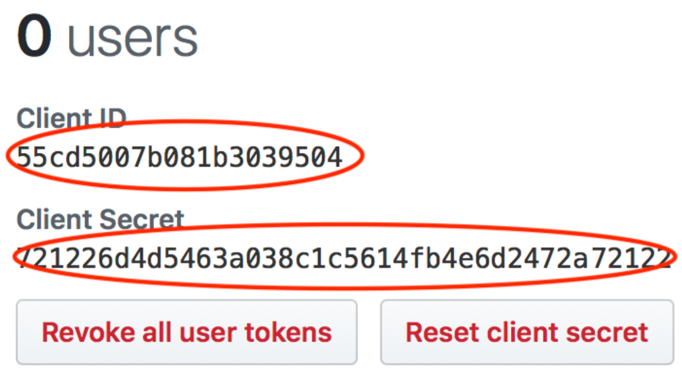
This is displayed to all users of your application

Authorization callback URL

http://10.0.0.1:31000/securityRealm/finishLogin

Your application's callback URL. Read our OAuth documentation for more information.

5. Click **Register application**.
6. GitHub will display your OAuth **Client ID** and **Client Secret**. Copy both strings and paste them into the text document that you created previously, so you can refer to them later.



0 users

Client ID
55cd5007b081b3039504

Client Secret
721226d4d5463a038c1c5614fb4e6d2472a72122

7. Your GitHub account is now prepared to integrate with Microservice Builder. In a real-world environment, it is possible to further automate the integration by creating a GitHub web hook, so that Microservice Builder can start a new build process any time changes are committed to your project in GitHub. That capability requires inbound access to your IBM Cloud Private Environment from the public internet, which is not available in this lab environment. For more information on configuring web hooks, see the previously referenced Knowledge Center documentation at https://www.ibm.com/support/knowledgecenter/en/SS5PWC/pipeline.html

Creating a starter application

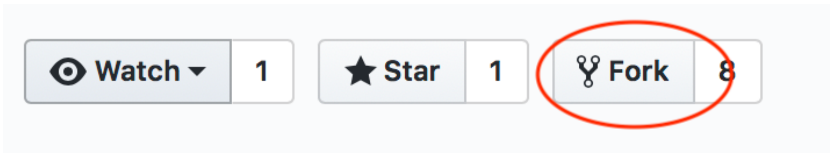
There are several ways you can create the skeleton of a microservices-based application to begin a project. The simplest is to use the IBM command-line developer tools to create a basic skeleton (for example **bx dev create** as described here: https://console.bluemix.net/docs/cloudnative/idt/commands.html)

For the purposes of this exercise, you will start your project by forking IBM's sample Node.js application, which already exists in GitHub, since it will accomplish the goal of creating a repository in GitHub and a sample application at the same time.

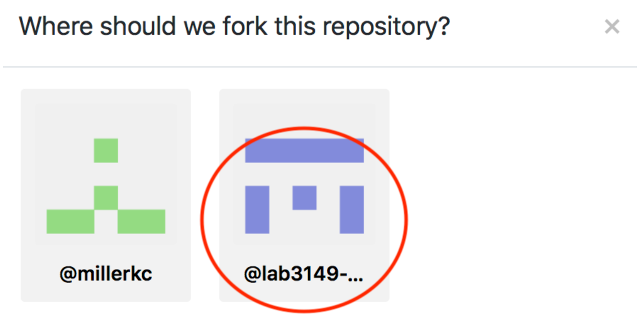
Fork IBM's sample application

IBM provides a sample Node.js application that can be used as a starting point for Node.js-based projects. Similar samples are available for Java-based applications.

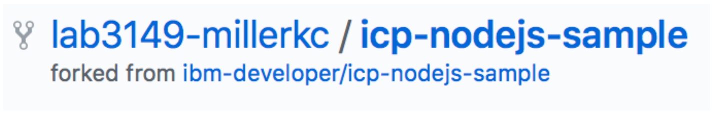
1. In your web browser, navigate to https://github.com/ibm-developer/icp-nodejs-sample
2. Click the **Fork** button at the top-right of the page.



3. When the "Where should we fork this repository?" panel appears, choose the organization that you created earlier in this exercise.



4. After a little bit of time, you should see your own copy of the project in your web browser. For example:


lab3149-millerkc / icp-nodejs-sample
forked from ibm-developer/icp-nodejs-sample

think 2018

Explore the sample application

The sample application includes everything that is necessary to build, deploy, and run a Node.js-based application on IBM Cloud Private. Take some time to browse around the project to get an idea of both the structure and the contents.

- The **chart** directory contains YAML files that define a Helm chart, which automates deployment of containerized applications on a Kubernetes platform, such as IBM Cloud Private. Later in this exercise, you will see this very chart in the IBM Cloud Private Helm chart repository.
- The **docker-6** directory contains a Dockerfile that defines how to automatically build a Docker image using IBM's Node.js v6 runtime.
- The **docker-8** directory contains a Dockerfile that defines how to automatically build a Docker image using IBM's Node.js v8 runtime.
- The **multiarch-manifests** directory contains the "fat manifest" files that enable the sample application to be built for multiple hardware architectures. This application supports the amd64, ppc64le, and s390x platforms
- The **server** directory contains the JavaScript code that implements the HTTP server for the sample application
- The **views** directory contains the various web pages that are part of the sample application.
- **Dockerfile** is the default file that contains defines the steps to automatically build the Docker image.
- **Jenkinsfile** is the file that contains instructions to load the Microservice Builder library to define the Jenkins build pipeline

Deploy Microservice Builder Pipeline on IBM Cloud Private

IBM Cloud Private is configured by default to display a catalog of containerized IBM software that is available in IBM's public Helm chart repositories. Helm charts provide a standardized way of packaging, versioning, and managing container-based workloads on a Kubernetes platform.

Microservice Builder is one of the workloads available in the catalog by default, and provides a turnkey solution for building a Jenkins-based build-deploy-test pipeline that is directly integrated with the IBM Cloud Private platform.

Set up a registry secret

You will need to set up a Docker registry secret that will allow the Microservice Builder Pipeline to securely access IBM Cloud Private's built-in Docker registry

1. Use the IBM Cloud Private command-line utility to log in to IBM Cloud Private
    a. Open a command-line terminal window by navigating to **Applications > System tools > Terminal**
    b. Log in to your cluster by entering the following command (the –skip-ssl-validation flag is needed because the system used in this exercise is using a self-signed SSL certificate.)

       bx pr login –a https://bluedemocluster.icp:8443 --skip-ssl-validation

    c. Enter **admin** at the "Username>" prompt
    d. Enter **admin** at the "Password>" prompt
    e. Enter **1** at the "Enter a number>" prompt to select the only account that is configured in this environment

2. Configure the kubernetes command-line tools to communicate with your IBM Cloud Private Cluster
    a. Enter the following command in the terminal window:

       bx pr clusters

    b. This command will return a list of clusters. You should see only one cluster, named "bluedemocluster"

   c. Enter the following command to configure the kubernetes command-line tools to communicate with that cluster:

      bx pr cluster-config bluedemocluster

   d. You should see the result “Cluster bluedemocluster configured successfully”

3. Use the “kubectl” command-line client to create a secret and patch the default service account to use that secret.
   a. Run the following script, which has been provided on your virtual machine, since you may not be able to easily copy and paste long commands from this document:

      /home/skytap/createSecret.sh

   b. This script runs two commands, which are described in the IBM Knowledge Center documentation for setting up the Microservice Builder pipeline. The first creates a kubernetes secret that contains your docker username and password.

```
kubectl create secret docker-registry admin.registrykey --docker-
server=https://bluedemocluster.icp:8500 --docker-username=admin --
docker-password=admin --docker-email=null
```

   c. The second command patches the default service account with the registry key you just created, using the *jq* command to parse and manipulate the JSON data.

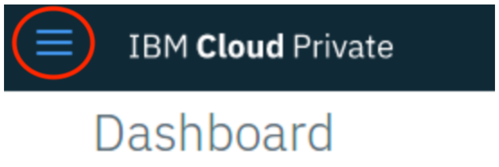
```
kubectl get serviceaccounts default -o json |
   jq  'del(.metadata.resourceVersion)' |
   jq 'setpath(["imagePullSecrets"];[{"name":"admin.registrykey"}])' |
   kubectl replace serviceaccount default -f -
```

      (To learn more about this step if you are new to Kubernetes, see “Configure Service Accounts for Pods” in the Kubernetes documentation.)

Log in to the IBM Cloud Private web console

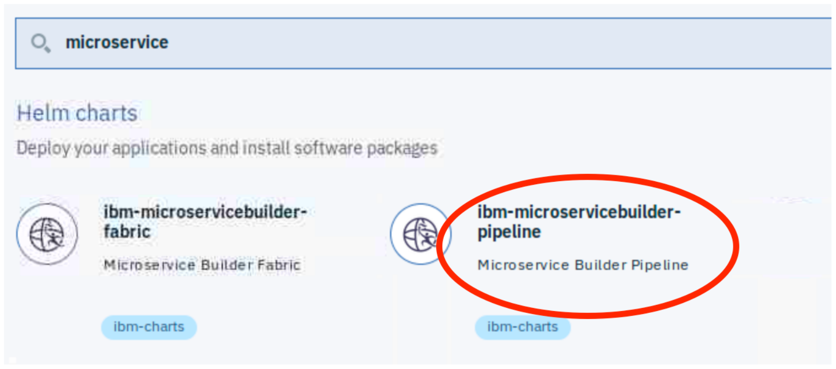
1. Open a web browser on the virtual machine that you have been provided. An icon for Firefox is available on the desktop.
2. Navigate to https://10.0.0.1:8443 to access the IBM Cloud Private web console (this should happen by default on the virtual machine you have been provided.)

3. Enter the username **admin** and the password **admin** (they should be pre-filled on the virtual machine you have been provided) and click **Log in**.
4. You will see the IBM Cloud Private Dashboard view by default. If you are new to IBM Cloud Private, feel free to browse around by using the menu at the top left.



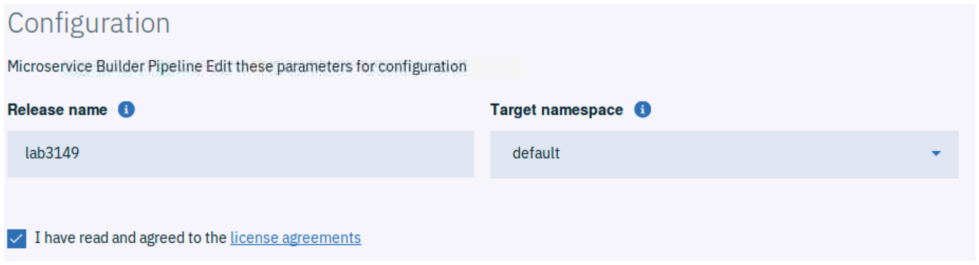
Deploy Microservice Builder Pipeline using a Helm chart

1. Expand the menu at the top left and navigate to **Catalog > Helm charts**. The default IBM Helm chart repository will be displayed.
2. Scroll through the catalog and take note of the various freely deployable development-edition products that are available by default.
3. Scroll back to the top and type **microservice** in the "Search items" field to filter the available charts.
4. Click **ibm-microservicebuilder-pipeline**


microservice
Helm charts
Deploy your applications and install software packages
ibm-microservicebuilder-fabric
Microservice Builder Fabric
ibm-charts
ibm-microservicebuilder-pipeline
Microservice Builder Pipeline
ibm-charts

5. The chart's README file will be displayed. Each Helm chart in the catalog contains a README file that describes the chart's prerequisites, input parameters, and instructions for deploying the chart from the command line. Browse this chart's README file, and note that the prerequisites are the steps that you have completed earlier in this exercise.
6. Click the **Configure** button to begin configuring the chart for deployment.
7. Enter **lab3149** in the "Release name" field (you may need to scroll to the top of the page if you don't see the "Release name" field.
8. Select **default** from the "Target namespace" menu

9. Click the **license agreements** link to read the chart's license agreement, then click **Accept**. You should now see that the box next to "I have read and agreed to the license agreements" is checked.



Configuration

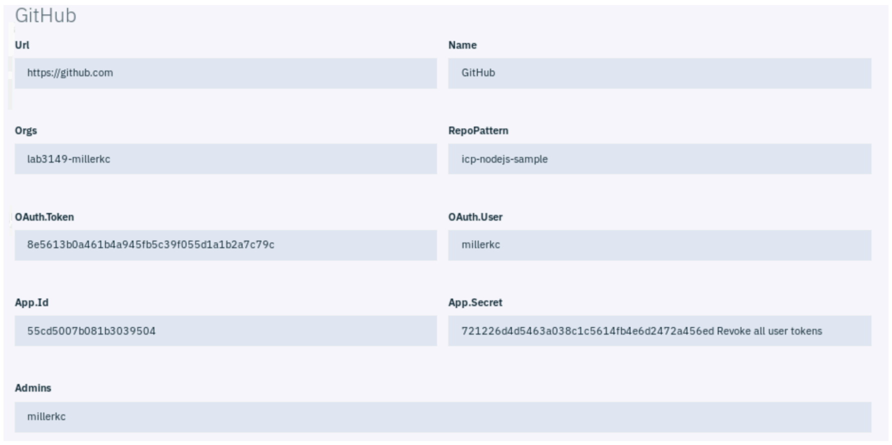
Microservice Builder Pipeline Edit these parameters for configuration

Release name: lab3149

Target namespace: default

☑ I have read and agreed to the license agreements

10. Scroll down to the "GitHub" section and configure Microservice Builder to use the GitHub configuration that you created earlier.
    a. Enter the name of the GitHub organization you created earlier (such as **lab3149-<username>**) in the "Orgs" field.
    b. Enter **icp-nodejs-sample** in the "RepoPattern" field. In a real-world scenario, you can also enter a regular expression here to match multiple repositories in your organization.
    c. Copy your OAuth access token from the text file that you created earlier, and paste it into the "OAuth.Token" field
    d. Enter your GitHub username in the "OAuth.User" field.
    e. Copy your GitHub client ID from the text file that you created earlier, and paste it into the "App.Id" field
    f. Copy your GitHub client ID from the text file that you created earlier, and paste it into the "App.Id" field
    g. Copy your GitHub client secret from the text file that you created earlier, and paste it into the "App.Secret" field
    h. Enter your GitHub username in the "Admins" field



GitHub

Url: https://github.com

Name: GitHub

Orgs: lab3149-millerkc

RepoPattern: icp-nodejs-sample

OAuth.Token: 8e5613b0a461b4a945fb5c39f055d1a1b2a7c79c

OAuth.User: millerkc

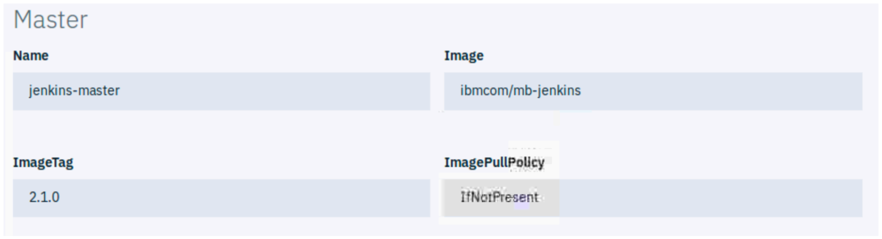
App.Id: 55cd5007b081b3039504

App.Secret: 721226d4d5463a038c1c5614fb4e6d2472a456ed Revoke all user tokens

Admins: millerkc

think
2018

i. Scroll down to the “Pipeline” section and change the value of the “Registry.Url” field to **bluedemocluster.icp:8500/default** to reflect the correct hostname of your cluster.
j. Scroll down to the “Master” section and change the value of the “ImagePullPolicy” field to **IfNotPresent**, taking care to match the capitalization correctly. This tells Docker to use the Jenkins Docker image that is cached locally instead of downloading it from DockerHub over the internet, since everyone in this lab downloading large files simultaneously over a shared connection might add significant time to the exercise.



Master

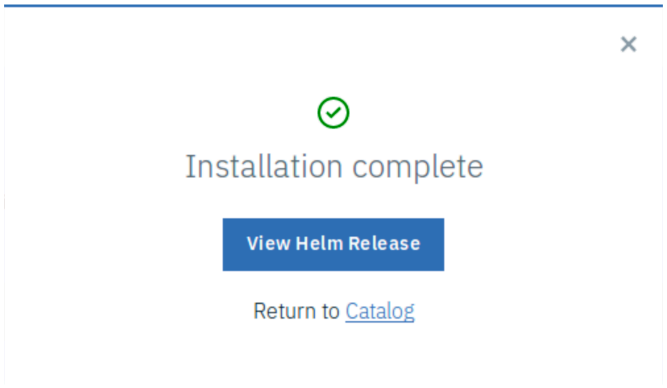
Name: jenkins-master

Image: ibmcom/mb-jenkins

ImageTag: 2.1.0

ImagePullPolicy: IfNotPresent

k. Leave the remaining fields with their default values. In a real-world scenario, you may want to customize some of them. For example, you would want to enable persistence, so that your files that have been written to disk can persist beyond the lifetime of a given pod.
l. Click **Install**. You should see the “Installation complete” message as shown below.



Installation complete

m. Click **View Helm release** to view details about your deployment in the web console.
n. In your terminal window, enter the command below to get the status of your kubernetes deployment:

```
kubectl get pods
```

o. You will very likely see **0/1** displayed in the “READY” column and **Init:0/1** in the “STATUS” column. It can take a few minutes for the pipeline to start in this lab environment.

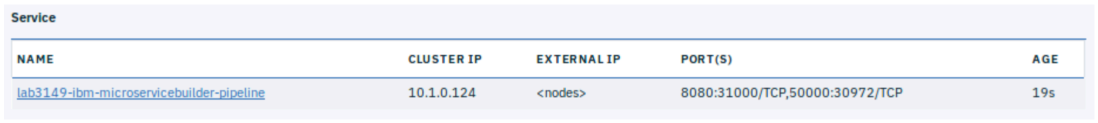
p. Run the command again after a few minutes, and you should see **1/1** in the “READY” column and **Running** in the “STATUS” column

Build and update your application

Now that you have deployed Microservice Builder Pipeline, you can use it to build and deploy your sample application to IBM Cloud Private

Log in to your new Jenkins environment

1. Return to your web browser window, where the status of your Helm release is displayed.
2. Click on the name of your release, **lab3149**.
3. Scroll down to the "Service" section, and click on the name of your service, which begins with **lab3149-ibm-microservice-builder-pipeline**.



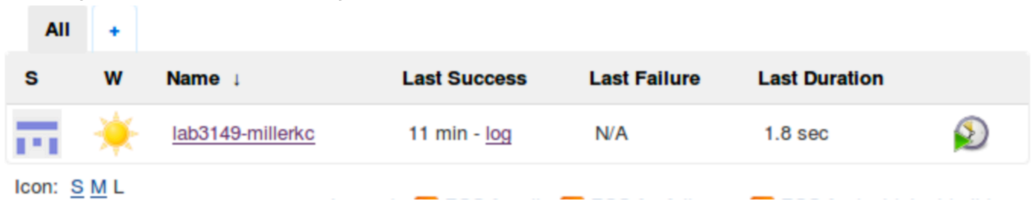
Service

NAME	CLUSTER IP	EXTERNAL IP	PORT(S)	AGE
lab3149-ibm-microservicebuilder-pipeline	10.1.0.124	<nodes>	8080:31000/TCP,50000:30972/TCP	19s

4. Scroll down to the "Node port" section of the table and click **http_31000/TCP**. The Node port is the externally available port where your newly deployed service is listening. This will open a new browser tab to http://10.0.0.1:31000, and then you will be redirected to GitHub to authenticate.
5. Enter your GitHub username and password, then click **Sign in**.
6. Github will display the application that is asking to be authenticated, "lab3149-jenkins", which you created previously. Click **Authorize <username>** to proceed. You will then be redirected back to your local Jenkins console.

Build your sample application

1. Once you are logged in to Jenkins, you will note that your sample application has already been automatically built!



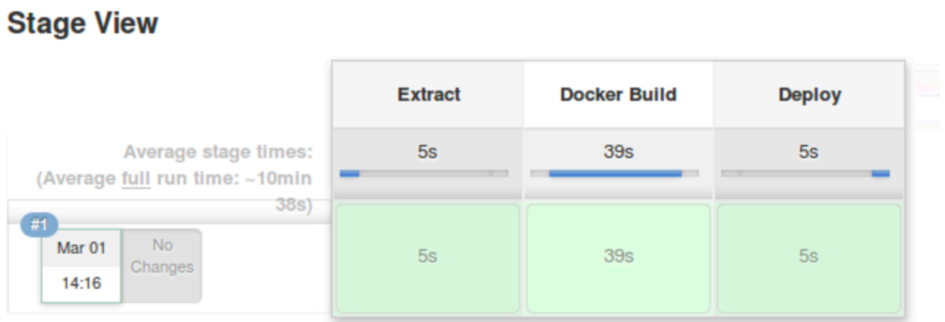
All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
	☀	lab3149-millerkc	11 min - log	N/A	1.8 sec	

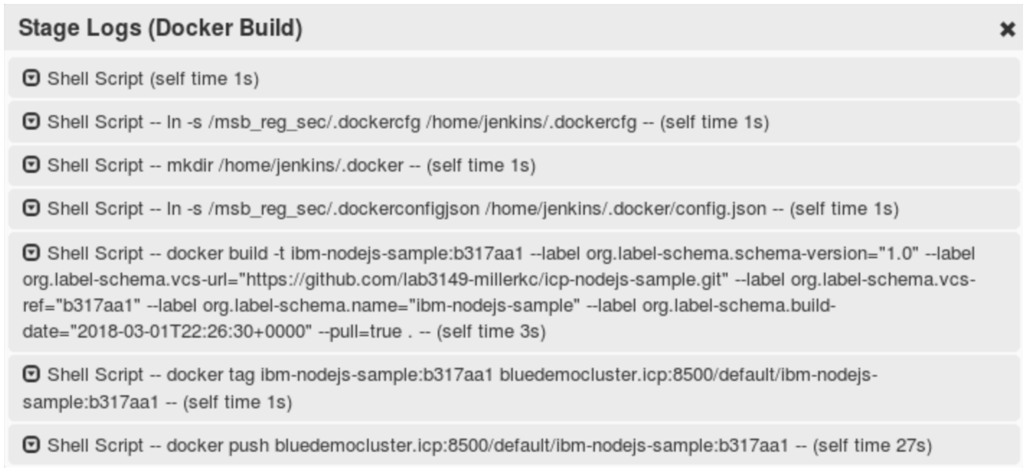
Icon: S M L

2. Click the name of your organization (**lab3149-<username>**).
3. Click the name of your application (**icp-nodejs-sample**).
4. Click **master** to view the results of your build.

5. If you have completed all of the previous steps correctly, the "Stage view" will show green boxes for each of the three steps of your build pipeline.



6. You can hover over any of the boxes and click the **Logs** button to view the detailed steps that were automated, such as the Docker Build process below.

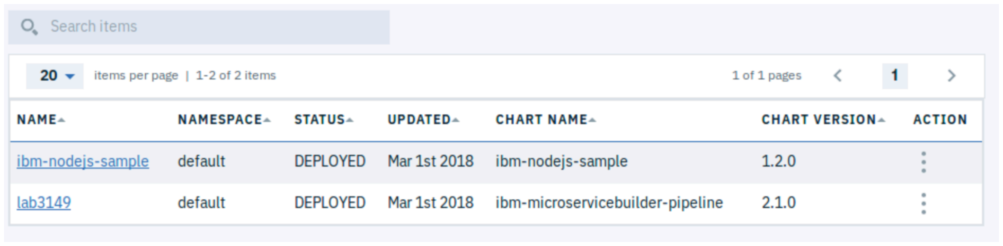


Stage Logs (Docker Build)

```
Shell Script (self time 1s)
Shell Script -- ln -s /msb_reg_sec/.dockercfg /home/jenkins/.dockercfg -- (self time 1s)
Shell Script -- mkdir /home/jenkins/.docker -- (self time 1s)
Shell Script -- ln -s /msb_reg_sec/.dockerconfigjson /home/jenkins/.docker/config.json -- (self time 1s)
Shell Script -- docker build -t ibm-nodejs-sample:b317aa1 --label org.label-schema.schema-version="1.0" --label
org.label-schema.vcs-url="https://github.com/lab3149-millerkc/icp-nodejs-sample.git" --label org.label-schema.vcs-
ref="b317aa1" --label org.label-schema.name="ibm-nodejs-sample" --label org.label-schema.build-
date="2018-03-01T22:26:30+0000" --pull=true . -- (self time 3s)
Shell Script -- docker tag ibm-nodejs-sample:b317aa1 bluedemocluster.icp:8500/default/ibm-nodejs-
sample:b317aa1 -- (self time 1s)
Shell Script -- docker push bluedemocluster.icp:8500/default/ibm-nodejs-sample:b317aa1 -- (self time 27s)
```

Test your running sample application

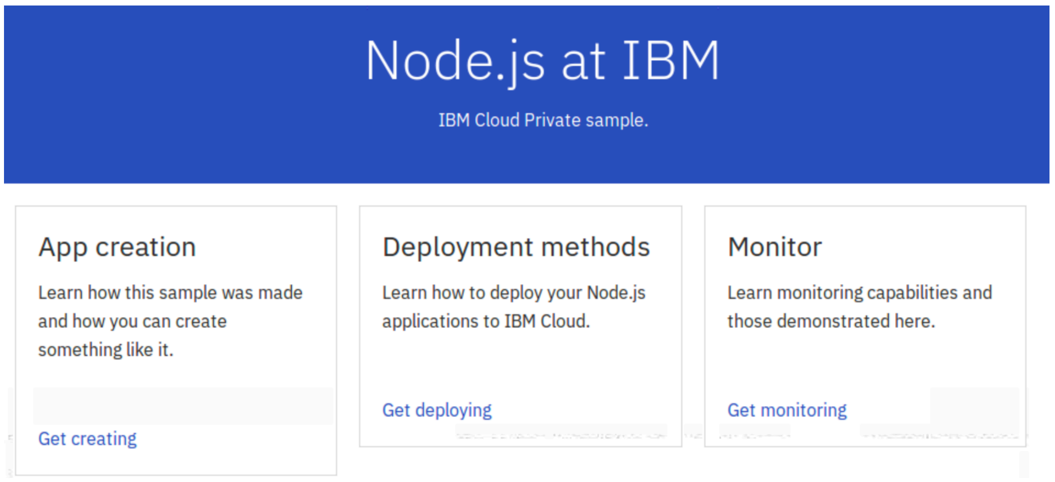
1. Return to the browser window where you are logged in to the IBM Cloud Private web console.
2. Navigate to **Workloads > Helm releases** and note that you now have two releases, "lab3149" and "ibm-nodejs-sample" because Microservice Builder used Helm to deploy your application after it was built.



NAME	NAMESPACE	STATUS	UPDATED	CHART NAME	CHART VERSION	ACTION
ibm-nodejs-sample	default	DEPLOYED	Mar 1st 2018	ibm-nodejs-sample	1.2.0	⋮
lab3149	default	DEPLOYED	Mar 1st 2018	ibm-microservicebuilder-pipeline	2.1.0	⋮

3. Click **ibm-nodejs-sample**.

4. Scroll down to the “Service” are, and click the link that begins with **ibm-nodejs-sample**.
5. Scroll down to the “Node port” entry in the table, and click **30277/TCP** (your port number may vary) to open the application in a new browser tab. You should see a screen similar to the one below.



Node.js at IBM

IBM Cloud Private sample.

App creation

Learn how this sample was made and how you can create something like it.

Get creating

Deployment methods

Learn how to deploy your Node.js applications to IBM Cloud.

Get deploying

Monitor

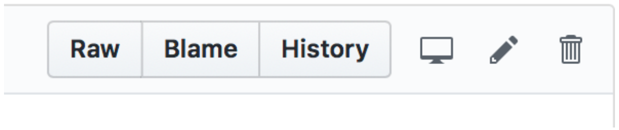
Learn monitoring capabilities and those demonstrated here.

Get monitoring

6. Browse around the sample application, particularly the **Get Creating** link, which will take you through more information about how this sample application was built, and how you can build your own.

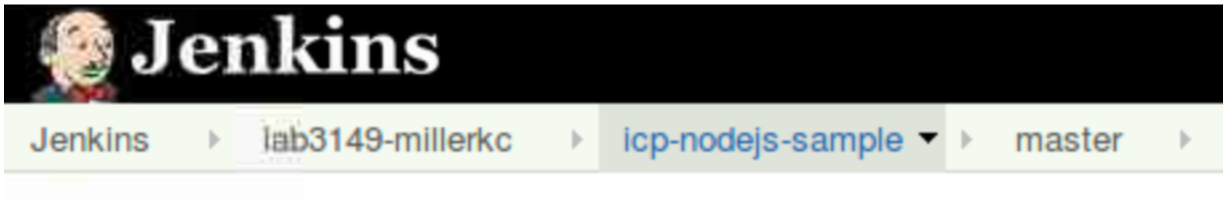
Modify and redeploy the sample application

1. Return to the browser window where you are logged in to GitHub, and navigate to the project that you created (**lab3149-<username>**) if you are not already there.
2. Click **views**, then click **partials**, then click **header.ejs** to open the file **views/partials/header.ejs**. This file defines the shared header section of the sample application.
3. Click the **pencil icon** to edit the file directly in your web browser. In a real-world scenario you would more likely clone this git repository to your local machine, and then edit the file with the editor or IDE of your choice, and then push your changes back to the remote repository, rather than doing everything in one step directly on GitHub.

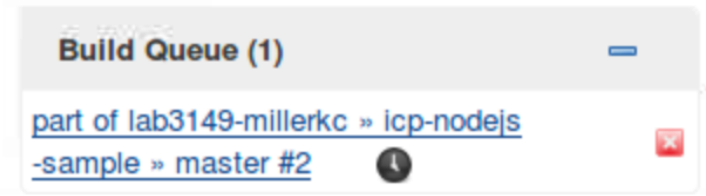


4. Navigate to line **96** and change the background color of the header. Change the text *background-color:#**3151b7*** to *background-color:#**7023bc***

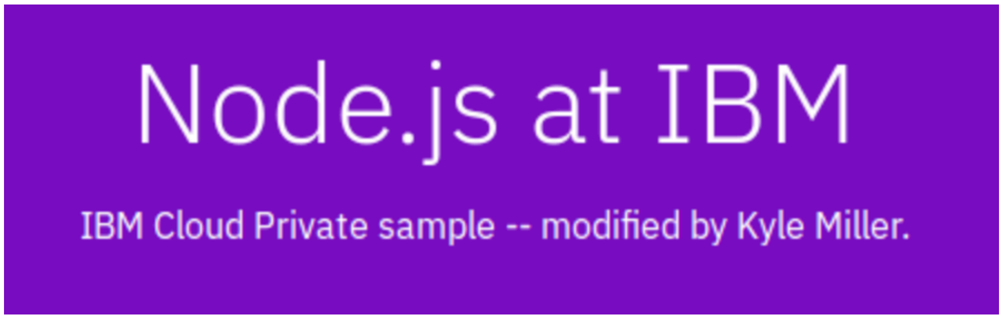
5. Navigate to line **105** and add your name to the header. Change the text *IBM Cloud Private sample* to *IBM Cloud Private sample – **modified by <your name>***.
6. Add a comment in the "Commit changes" section, such as "Changed background color to purple at THINK 2018."
7. Click **Commit changes**.
8. If you had configured a GitHub web hook, committing your code change would have automatically triggered a new build in Microservice Builder. Instead, switch back to the browser window where you are logged in to Jenkins, and click **icp-nodejs-sample** in the top menu.



9. Then click **Scan repository now** in the left menu.
10. The text "Done" will appear briefly, and then you will see a new build appear in your build queue in the lower left.



11. It may take a few minutes for the job to disappear from the job queue, indicating that the new build is complete.
12. Once the build completes, switch back to the browser tab where you were exploring the sample application and reload the page. (If you closed this tab, open a new tab and navigate to https://10.0.0.1:30277, though your application may be listening on a different port, which you can view in the IBM Cloud Private web console.)
13. You should now see the modified version of the application, with the text you modified, and a purple background in place of the blue background.



Conclusion

You have now successfully created a simple Node.js-based application and deployed it to IBM Cloud Private using Microservice Builder. You also used Microservice Builder to automatically rebuild and redeploy your modified application after committing you changed source code to GitHub

We Value Your Feedback!

- Don’t forget to submit your Think 2018 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Access the Think 2018 agenda tool to quickly submit your surveys from your smartphone, laptop or conference kiosk.