Zero to Cloud-Native with IBM Cloud

Part 9A: Creating a Classic Pipeline

Kevin Collins
kevincollins@us.ibm.com
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Kunal Malhotra
kunal.malhotra3@ibm.com
Cloud Platform Engineer
IBM Cloud MEA

1 – IBM Cloud Continuous Delivery – Classic Pipeline

Note: This tutorial will assume you have already setup a toolchain as describe in Part 9 – Section 4.

This tutorial will go step through creating is a classic pipeline for the API frontend microservice. To get started, navigate to the zero-to-cloud-native-toolchain you created in part 9 for this series.
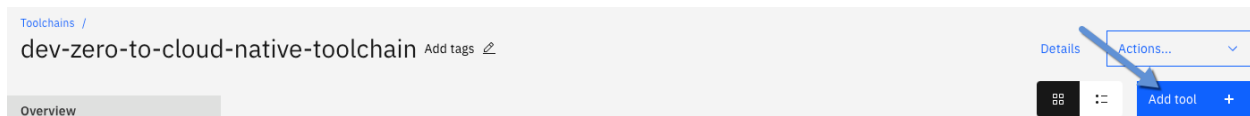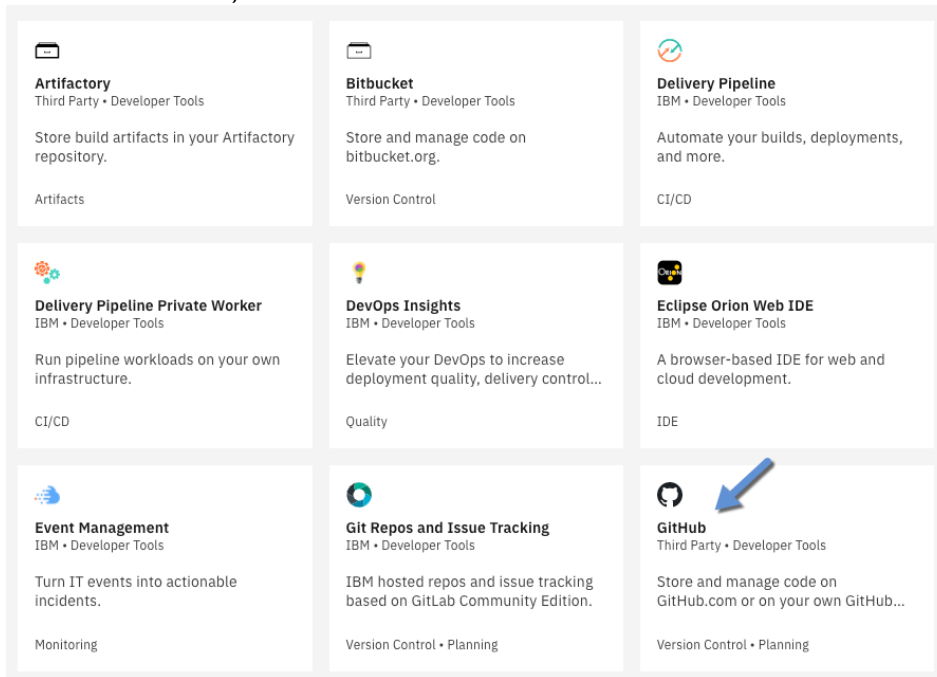

1 – 1 GitHub Integration

The first thing you need to do is connect your SCM (Source Code Management) repository with the toolchain. You do this by clicking add tool and then selecting SCM (GitHub, Bitbucket, GitLab, or GitHub Enterprise Whitewater).

Note: Repositories can be Public or Private. In this tutorial, I will be the public repository that I've shared and that you clone in part 8. By default, I recommend setting your repositories to private unless you plan to share them publicly. This way, if you make a mistake and upload something like an API key, you are still protected.

Note: if this is the first time you are creating a toolchain with a GitHub repository, you will first need to Authorize IBM Cloud to access GitHub. You can follow the instructions here if needed: https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-integrations#github

On the next screen, select **GitHub**.

Artifactory Third Party • Developer Tools Store build artifacts in your Artifactory repository. Artifacts	**Bitbucket** Third Party • Developer Tools Store and manage code on bitbucket.org. Version Control	**Delivery Pipeline** IBM • Developer Tools Automate your builds, deployments, and more. CI/CD
Delivery Pipeline Private Worker IBM • Developer Tools Run pipeline workloads on your own infrastructure. CI/CD	**DevOps Insights** IBM • Developer Tools Elevate your DevOps to increase deployment quality, delivery control... Quality	**Eclipse Orion Web IDE** IBM • Developer Tools A browser-based IDE for web and cloud development. IDE
Event Management IBM • Developer Tools Turn IT events into actionable incidents. Monitoring	**Git Repos and Issue Tracking** IBM • Developer Tools IBM hosted repos and issue tracking based on GitLab Community Edition. Version Control • Planning	**GitHub** Third Party • Developer Tools Store and manage code on GitHub.com or on your own GitHub... Version Control • Planning

Select **Existing** as the repository type and enter the repository URL for your git repository that you setup in Part 8 of this tutorial series. Click **Create Integration** after entering these settings.

Create Integration

GitHub Server:

GitHub (https://github.com)

Authorized as kmcolli with access granted to zero GitHub organization(s) Manage Authorization

Repository type:

Existing

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

Repository URL: ⓘ

https://github.com/kmcolli/api-frontend-02cn

Integration Owner: ⓘ

kmcolli

☑ Enable GitHub Issues ⓘ

☐ Track deployment of code changes ⓘ

1 – 2 Create Delivery Pipeline

The next tool we need to add is a Delivery Pipeline. Click on **add tool** and then click on the **Delivery Pipeline tile**.

Delivery Pipeline
IBM • Developer Tools

Automate your builds, deployments, and more.

CI/CD

On the next screen, give you delivery pipeline a name. I will use **dev-classic-frontend-api-toolchain**. For this part of the tutorial, keep **Classic** selected as the Pipeline type, and click **Create Integration**.

egration /

Pipeline   Create Integration

Pipeline name:

dev-classic-frontend-api-toolchain

Pipeline type:

Classic   ⌄

Classic pipelines provide an easy to use graphical UI for defining stages and jobs that run on public shared infrastructure, with support for running individual stages on Private Workers.

☑ Show apps in the View app menu ⓘ

1 − 2 − 1 Add the Continuous Delivery Service

On the next screen, you will see all the tools we have added to our toolchain. The next step is to configure the delivery pipeline we just created. You will also see the following warning notice.

You have 5 days to use the service for free. If you want to use the service more than 5 days, click on **Add the service**.

The first step on the next page is to select a region and plan. Keep the region as **Dallas**, like all our other resources, and you can also keep the plan selected as **Lite**.

Scrolling down, you need to give you service a name and indicate which resource group to place it in. I will use **Continuous Delivery-zero-to-cloud-native** as the name and select **zero-to-cloud-native** as the resource group.
After entering those settings, click on **Create**.

1 – 3 Configure your Delivery Pipeline

Navigate back to the toolchain you just created by clicking on the IBM Cloud hamburger menu and **select DevOps**.
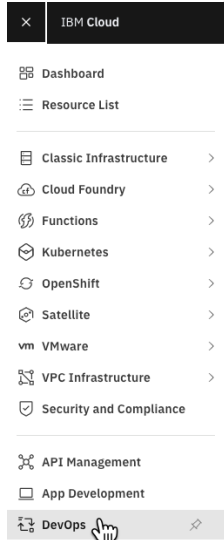
On the next screen, select **Dallas** as the location your toolchain is in and then click on the **toolchain name**.
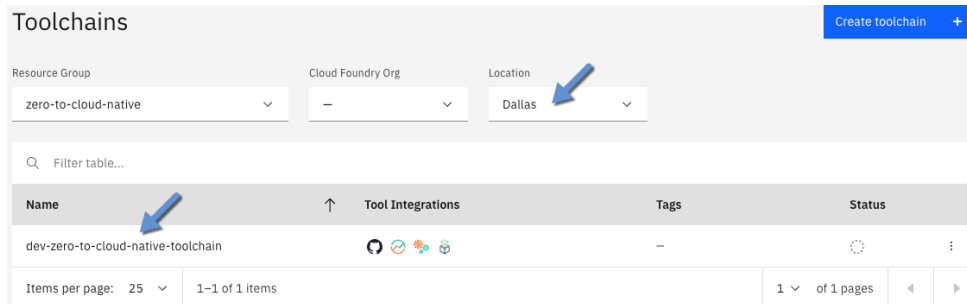
On the next screen, click on the **Delivery Pipeline** tile that you just created.
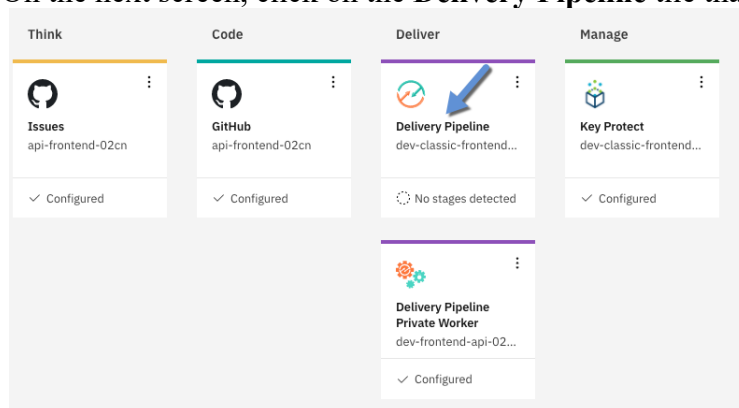
1 – 3 - 1 Build Stage
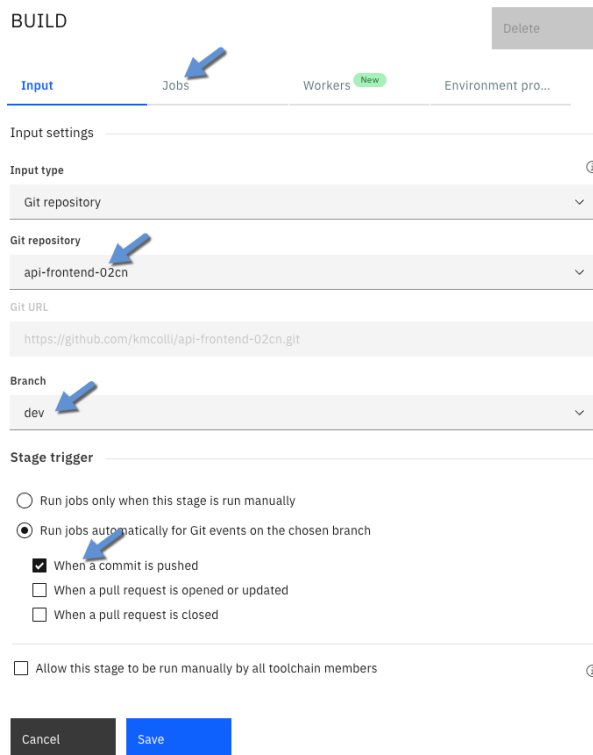
This will bring you to what looks like an empty screen as your pipeline does not have any stages in it yet. Click on **Add Stage**.

dev-classic-frontend-api-toolchain | Delivery Pipeline

Add Stage   +   Actions...   ∨
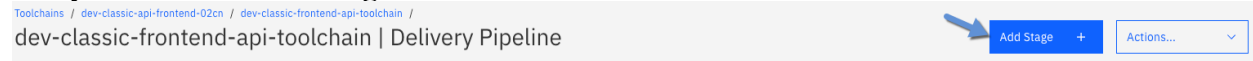
Start by naming the first stage **Build**.

Your git hub repository should already to populated as we have already created Git integration.

Next, we will be building a dev environment, select the **dev branch** that you created in part 8 of this tutorial series.

Make sure to keep the '**When a commit is pushed'** option selected. This will automatically start our toolchain once a commit is pushed to GitHub.

BUILD

Delete

Input   Jobs   Workers New   Environment pro...

Input settings

Input type

Git repository   ∨

Git repository

api-frontend-02cn   ∨

Git URL

https://github.com/kmcolli/api-frontend-02cn.git

Branch

dev   ∨

Stage trigger

○ Run jobs only when this stage is run manually

◉ Run jobs automatically for Git events on the chosen branch

☑ When a commit is pushed

☐ When a pull request is opened or updated

☐ When a pull request is closed

☐ Allow this stage to be run manually by all toolchain members

Cancel   Save

Don't click Save quite yet, instead click on the **Jobs** tab. A stage in a classic toolchain consists of a number of jobs. The first job we will create is fetching the code from GitHub. Click on **Add Job** and select **Build**.

⊕

ADD JOB

Select job type

Build

Deploy

Test
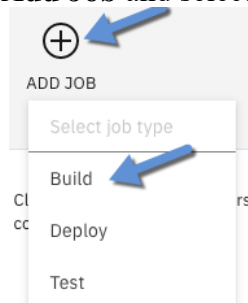
Change the name of the Job to **Fetch Code**, the builder type to **shell script** and enter
> `source ./pipeline/scripts/fetch_code.sh`

as the Build script.

BUILD   Delete

Input   **Jobs**   Workers New   Environment pro...

Fetch Code   ⊕ ADD JOB

Fetch Code   Remove

Build configuration

Builder type   ⓘ

Shell Script   ⌄

Pipeline image version   ⓘ

Inherited from Configure Pipeline (latest)   ⌄

Build script   ⓘ

source ./pipeline/scripts/fetch_code.sh

Note: you can view all the scripts used in this tutorial from the **pipeline/scripts** directory in each git repository:

After entering these settings, click on **Add Job** and **Test** and name the job **Unit Test**. This job will make sure your code repository was successfully cloned.
Keep the job as **simple** and enter, set the builder type to **shell script** and enter
> source ./pipeline/scripts/unit_test.sh

for Test script.

BUILD   Delete

Input   **Jobs**   Workers New   Environment pro...

Fetch Code ＞ Unit Test   ⊕ ADD JOB

Unit Test   Remove

Test configuration

Tester type   ⓘ

Simple   ⌄

Pipeline image version   ⓘ

Inherited from Configure Pipeline (latest)   ⌄

Test script   ⓘ

source ./pipeline/scripts/unit_test.sh

To configure your pipeline with the private worker you created, click on the **Workers tab** and select the private worker you previously created.
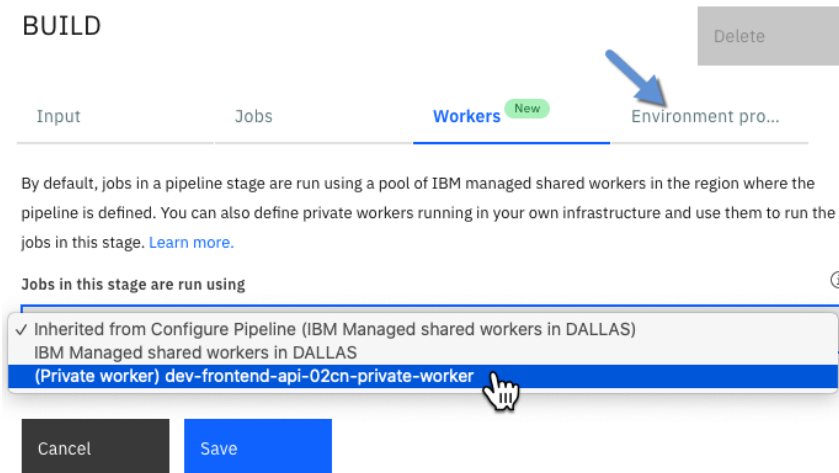
After selecting your private worker, the last step for the Build stage is to set an environment variable. Click on the **Environment properties** tab, click **Add Property** and select **Text Property**. Enter **IMAGE_NAME** as the property name and the name of the image for your code to be deployed as the value. For this example, we are building a CI/CD pipeline for the api frontend, so I will enter **api-frontend-02cn** as for image name. Click **Save** to save the Build Stage.
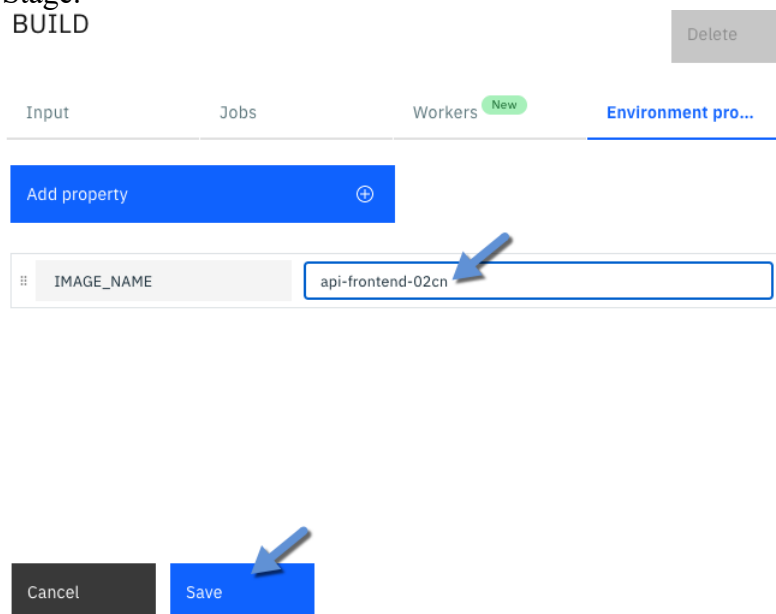
That completes the build stage.

1 – 3 – 2 Containerize Stage

The next stage we will create is the containerization stage. Click on **Add Stage** to create a new stage.
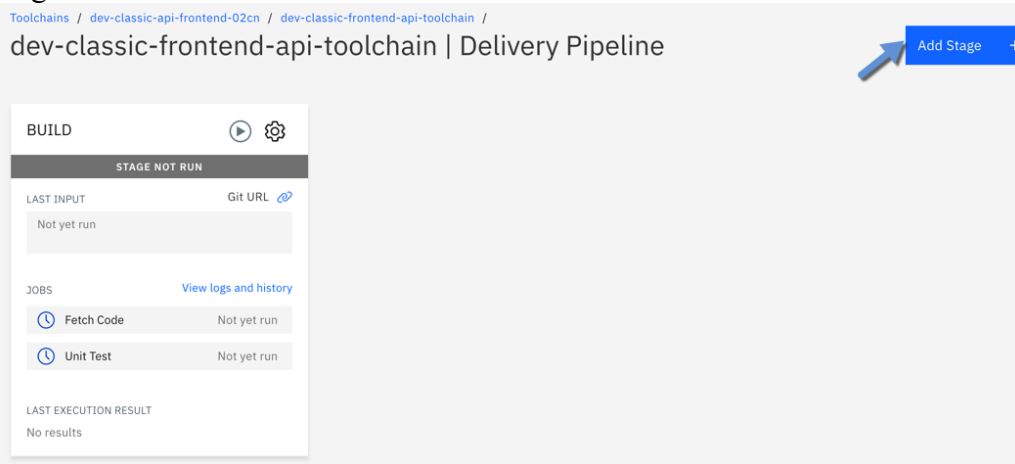
Change the name to **CONTAINERIZE**, leave the rest of the Input step as-is. On the **input tab**, **Build artifacts** should be the input type, **BUILD** as the stage and **Fetch Code** for the job. This will pass the code that was cloned from the git branch so we can create an image of the code in this step.
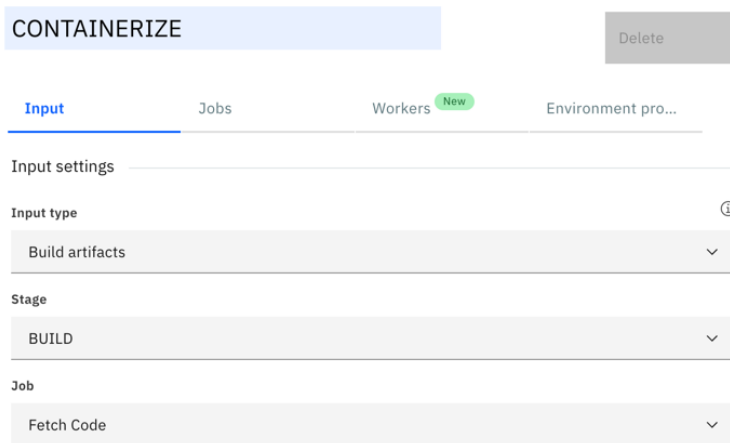
Next, we need to create the jobs that will build our container. The first job is making sure the Dockerfile is formatted correctly. Click on **Add Job** and select **Test**. Click on the **Jobs** tab then click **Add Job** and select **Test.**
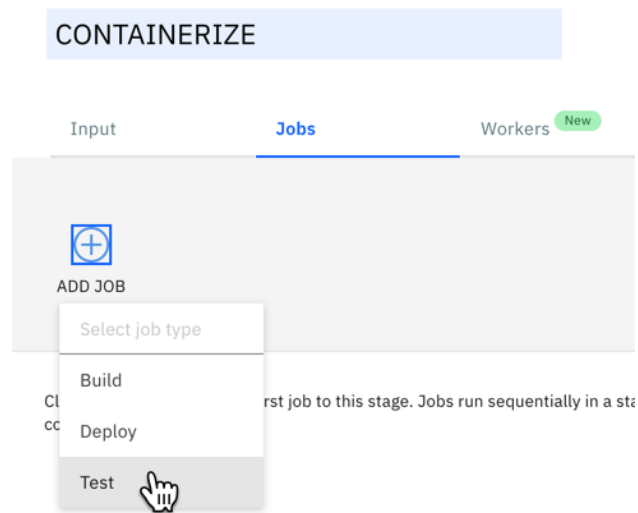
Change the name to **Check Dockerfile**.
Keep the tester type as Simple and for Test script enter:
source ./pipeline/scripts/check_dockerfile.sh
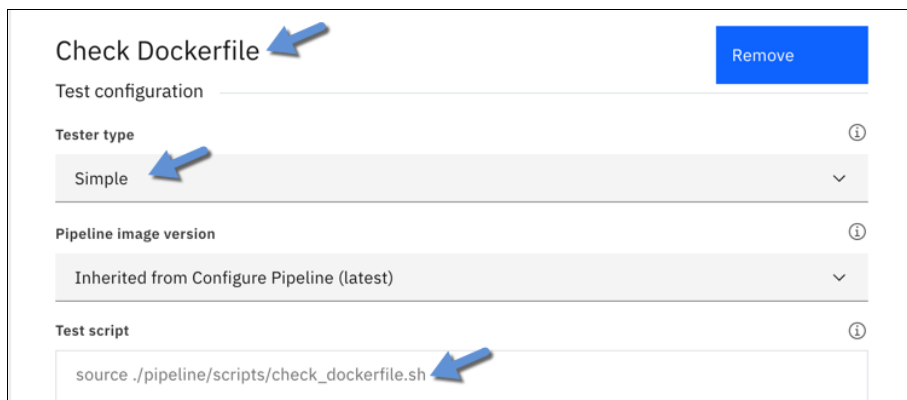
Create a second job by clicking **Add Job** and select **build**.
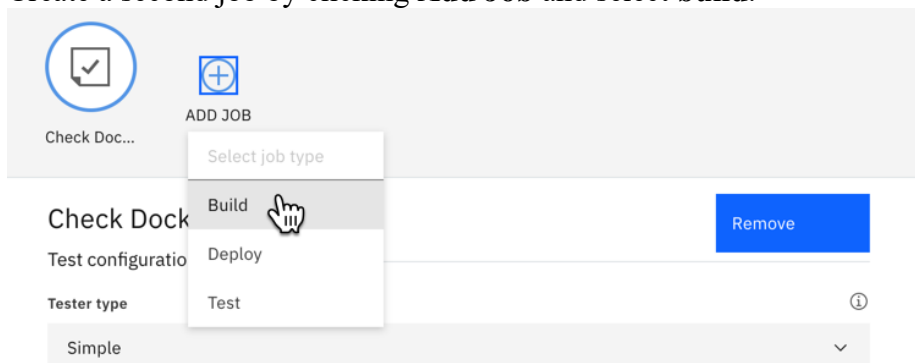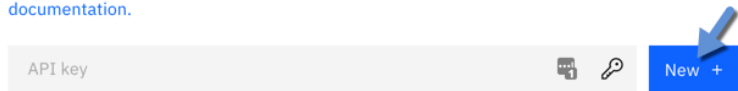
Name the job **Check Container Registry**. This job will make sure you have the appropriate permissions and quota to add new images to your IBM Cloud Registry.

Enter **Container Registry** of the Builder Type. This job requires an IBM Cloud API Key in order to access the **IBM Cloud Container Registry Service**. Input your **IBM Cloud API Key** or we will be following best practices and use **Key Protect** to store our API Key. Since this is the first time we will be using an IBM Cloud API Key in the toolchain service we will create a new API key which we will store in Key Protect. Click the **New** icon.

Enter an API key

This API key will be used to access the container registry and its namespaces.

Note: To maintain the strongest security posture you should use a service ID API key that has been configured to have only the access you need. For more information, see the IAM documentation.

| API key | | | New + |

| Cancel | Authenticate |

On the next screen, give the key a meaningful name. I will name the key **API Key for zero to cloud native toolchain.**
Check the box **Save this key in secrets for reuse.**
Under Provider, select the **Key Protect** instance we created in Part 3.
Finally, provide a **Secret Name** for the secret that will be stored in Key Protect. I will use **dev-classic-api-frontend-02cn** as my secret name.
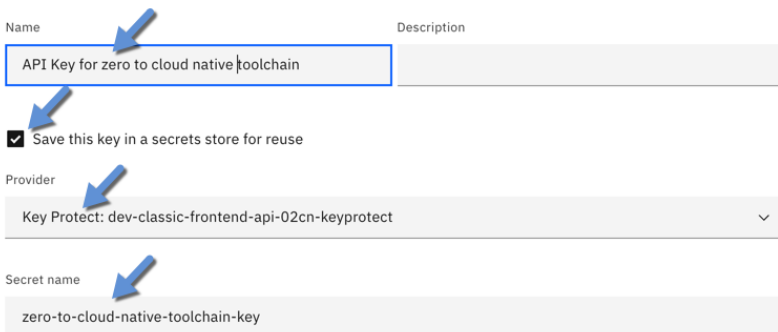
Create a new API key with full access

anything you could do. You can improve your security posture by using the IAM UI to create a service ID API key that limits access to only what your pipeline requires, and then pasting that into the template UI instead. For more information on API keys and access see the IAM documentation.

Name
API Key for zero to cloud native toolchain

Description

☑ Save this key in a secrets store for reuse

Provider
Key Protect: dev-classic-frontend-api-02cn-keyprotect

Secret name
zero-to-cloud-native-toolchain-key

| Cancel | OK |

After entering all of these settings click OK.

On the next screen, click Authenticate.

Enter an API key

This API key will be used to access the container registry and its namespaces.

Note: To maintain the strongest security posture you should use a service ID API key that has been configured to have only the access you need. For more information, see the IAM documentation.

| .... | 🗗 | 🗐 | 🔑 | New + |

| Cancel | Authenticate |

Change the **IBM Cloud Region** to US South which is where our container registry is deployed. Select the **container registry namespace** you created earlier. The container registry namespace I created was zero-to-cloud-native.

For the **docker image**, enter **api-frontend-02cn** since this pipeline is building the frontend api microservice.

Finally, for the build script, enter:
```
source ./pipeline/scripts/unit_test.sh
```

Container Registry   Remove

Build configuration

Builder type   ⓘ

| Container Registry | ⌄ |

Pipeline image version   ⓘ

| Inherited from Configure Pipeline (latest) | ⌄ |

API key   ⓘ

| API Key for zero to cloud native toolchain |

IBM Cloud region   ⓘ

| US South | ⌄ |

Account Name   ⓘ

| kevin collins's Account | ⌄ |

Container Registry namespace   ⓘ

| zero-to-cloud-native | ✕ ⌄ |

Docker image name   ⓘ

| api-frontend-02cn |

Build script   ⓘ

| source ./pipeline/scripts/unit_test.sh |

Don't click Save quite yet, we need to create one more job for this stage.

We need to create one more job to create the container image. To do so, click on **Add Job** and select **Build**.
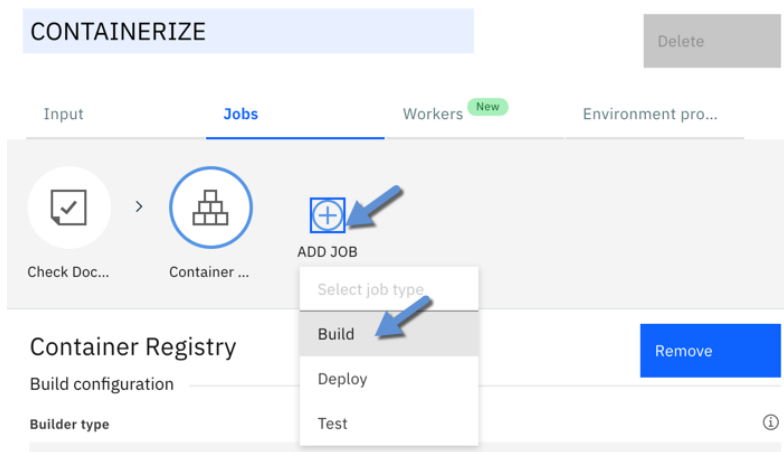
Enter **Build Container Image** as the name and **Container Registry** as the **Builder type**. With the previous job we added an API key for our toolchain service to our Key Protect instance which we will be able to use. Click on the **api key** box, and then this time click on the **key icon**.

On the next screen, select the **Key Protect instance** you created and the **Secret Name** we create in the previous step. Then click **OK**.

On the next screen, the api key will be populated. Click **Authenticate**.

Just like we did on the previous job, you will need to change the **IBM Cloud Region** to US South which is where our container registry is deployed.

Select the **container registry namespace** you created earlier. The container registry namespace I created was zero-to-cloud-native.

For the **docker image**, enter **api-frontend-02cn** since this pipeline is building the frontend api microservice.

For the build script, enter:

```
source ./pipeline/scripts/build_container.sh
```


This finishes setting up the jobs, but there are a couple others things we need to do before saving the stage.

The next thing we need to do is to configure the containerize stage to use the private worker we created.
Click on the **wokers** tab:
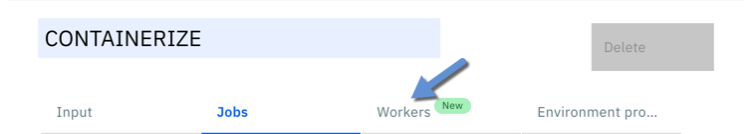
Next, select the **private worker** you created.
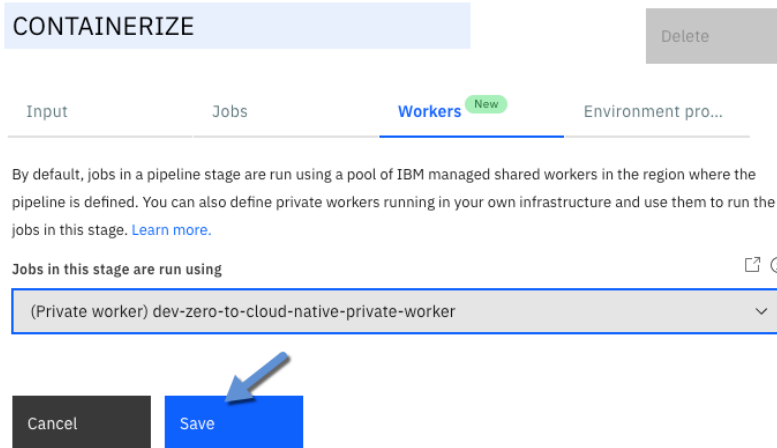
Before clicking save, we need to first set a couple environment properties.
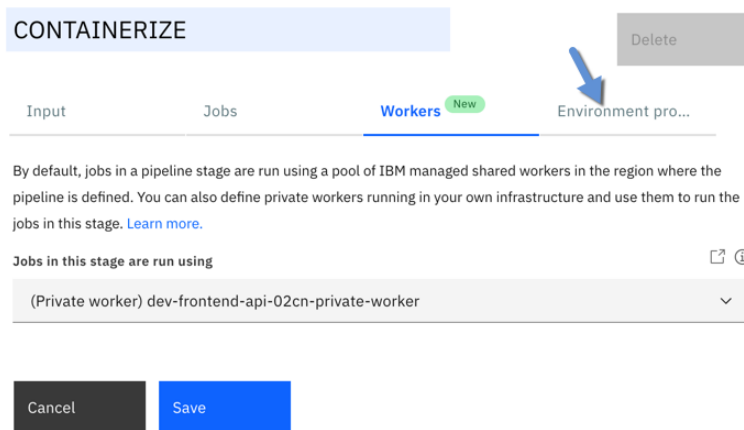
Click on the Environment Properties tab.

There are three Environment Properties we will need to add. To add a property, click on Add Property and enter the property type.
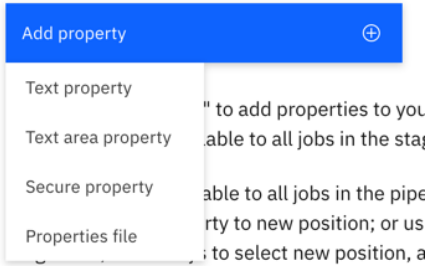
Referring to the table below, create the following environment properties.

Property Type	Property Name	Value
Text Property	DOCKER_FILE	Dockerfile
Text Property	DOCKER_ROOT	.
Properties File	buildprops	build.properties

After entering all of these properties, click Save.
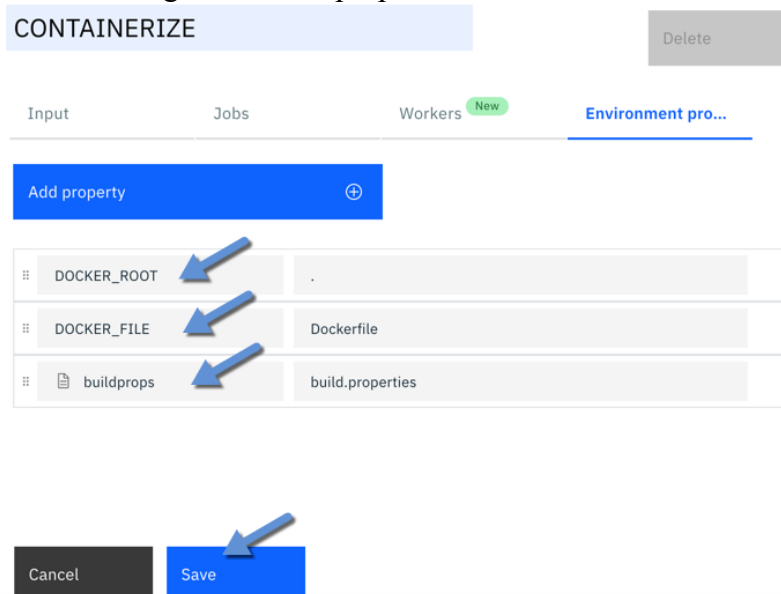
This completes the containerize job.

1 − 3 − 3 Deploy Stage

The final stage we need to create is the **deploy stage** where we will deploy the container into our OpenShift. Click **Add Stage.**
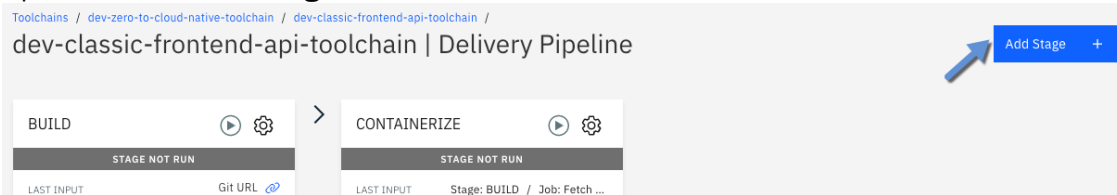
On the next screen name the stage **DEPLOY**. While on the **Input tab**, select **CONTAINERIZE** as the Stage to get inputs from and select the job **Build Container Image**. This will pass the container image that we will deploy to the deploy stage.

Next we need to add two more jobs. The first one to deploy the image and the second to make sure the deployment was successful. To create the Deploy to OpenShift Job, click on **Add Job**, select **Deploy.**
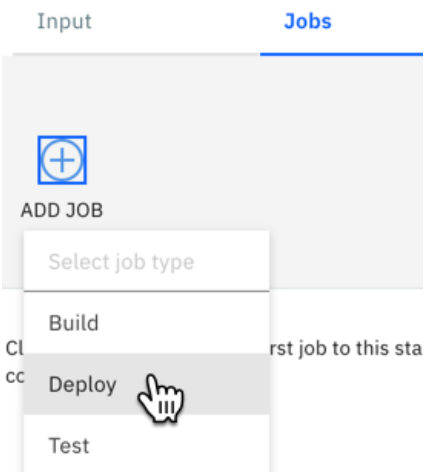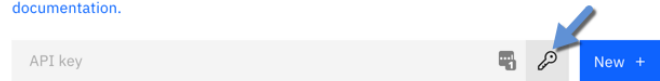
On the next screen, name the job **Deploy to OpenShift**. Select **Kubernetes** as the Deployer. Just like we did on the **Containerize Job**, click on the API Key text box, and then click on the Key Protect key icon.

Enter an API key

This API key will be used to access the cluster where your application will be deployed.

Note: To maintain the strongest security posture you should use a service ID API key that has been configured to have only the access you need. For more information, see the IAM documentation.

| API key | | | New + |

| Cancel | Authenticate |

On the next screen, enter your **Key Protect instance** and **Secret Name** for the toolchain service and click **OK**.

API key

Select a secret for this field from a secrets store using the options below.

Provider

Key Protect: dev-classic-frontend-api-02cn-keyprotect

Secret name

zero-to-cloud-native-toolchain-key

| Cancel | OK |

The next screen will populate the API Key automatically, click **Authenticate**.

Enter an API key

This API key will be used to access the cluster where your application will be deployed.

Note: To maintain the strongest security posture you should use a service ID API key that has been configured to have only the access you need. For more information, see the IAM documentation.

| •• | | New + |

| Cancel | Authenticate |

Scrolling down, enter US South as the **IBM Cloud Region**. Select the **Resource Group** where your cluster is created which should be **zero-to-cloud-native**. Next, selected your **cluster name** which should be **zero-to-cloud-native**.

Finally, for the deploy script, enter:

```
source ./pipeline/scripts/deploy_to_kubernetes.sh
```

Deploy to OpenShift

Deploy configuration

Deployer type

Kubernetes

Pipeline image version

Inherited from Configure Pipeline (latest)

API key

API Key for zero to cloud native toolchain

IBM Cloud region

US South

Account Name

kevin collins's Account

Resource Group

zero-to-cloud-native

Cluster name

zero-to-cloud-native

Deploy script

source ./pipeline/scripts/deploy_to_kubernetes.sh

Remove

The last step in our pipeline is to validate that the deployment was successful. To do so, scroll back to the top of the page and click **Create Job** and select **Deploy.**

DEPLOY

Input   **Jobs**

Deploy to ...   ADD JOB

Deploy to O

Deploy configura

Deployer type

Select job type

Build

Deploy

Test

Change the name to **Check Deployment Health** and deployment type to **Kubernetes**. Select your API Key from Key Protect the same way as you did above.

Scrolling down, enter US South as the **IBM Cloud Region**. Select the **Resource Group** where your cluster is created which should be **zero-to-cloud-native**. Next, selected your **cluster name** which should be **zero-to-cloud-native**.

For the deploy script, enter:

```
source ./pipeline/scripts/check_health.sh
```


After entering all the information, click on the Workers tab.

On the next screen, select the Private Worker that we created.

The final step for this job is to set environment properties. Click on the Environment properties tab.

There are three Environment Properties we will need to add. To add a property, click on Add Property and enter the property type.

Referring to the table below, create the following environment properties.

Property Type	Property Name	Value
Text Property	CLUSTER_NAMEPACE	zero-to-cloud-native
Text Property	DEPLOYMENT_FILE	./deployments/deployment.yaml
Properties File	buildprops	build.properties

After entering all of these properties, click Save.

That completes setting up our classic toolchain with a private worker for the api frontend! Now, let's test it.

4 – 6 Testing the Classic Delivery Pipeline

To test our pipeline, all we have to do is make a change to the source code and commit it to GitHub as we have configured our pipeline to automatically start after a GitHub commit to our repository.

To make a change, start Visual Studio Code, and open the workspace you created in step 5. Navigate to the api-frontend-02cn folder and click on the Dockerfile. Simply add a new line anywhere in the file.
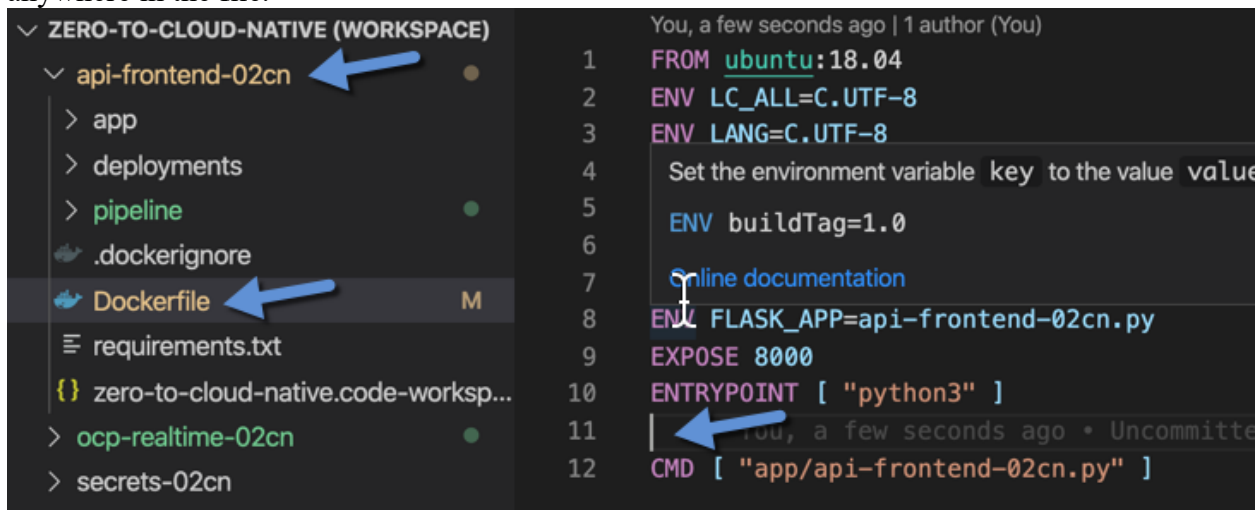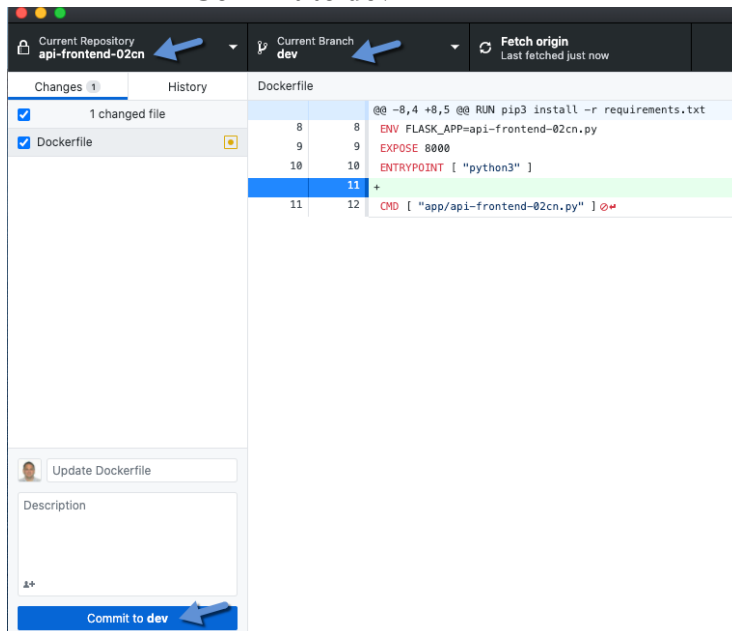

Next, we will need to push the 'change' to GitHub. Start your **GitHub Desktop**.
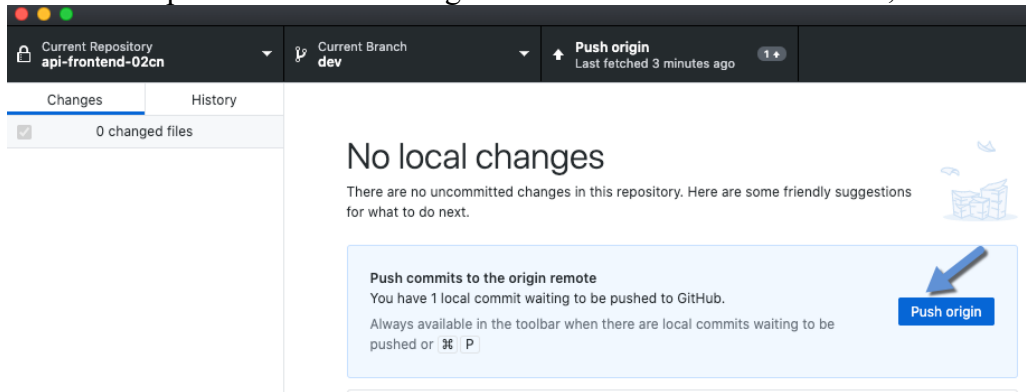Change to the **api-frontend-02cn** as the repository.
Make sure the **current branch** is dev.

You will notice the highlighted changes which should only be adding the newline.
Then click on **Commit to dev**.

The final step is the Push the changes to GitHub. On the next screen, click on Push origin.

Now, navigate back to your browser with the api frontend pipeline and watch it run!

3B – IBM Cloud Continuous Delivery – Tekton Pipeline