Zero to Cloud-Native with IBM Cloud

Part 9A: Creating a Tekton Pipeline

Kevin Collins

Technical Sales Leader

IBM Cloud Enterprise Containers – Americas

Kunal Malhotra

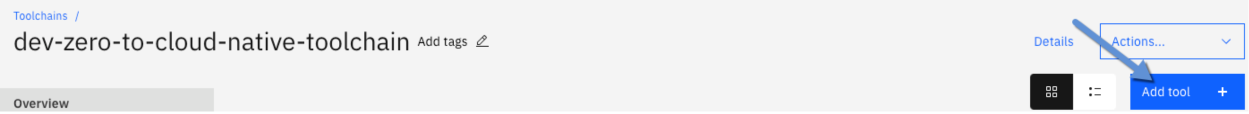
Cloud Platform Engineer

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.



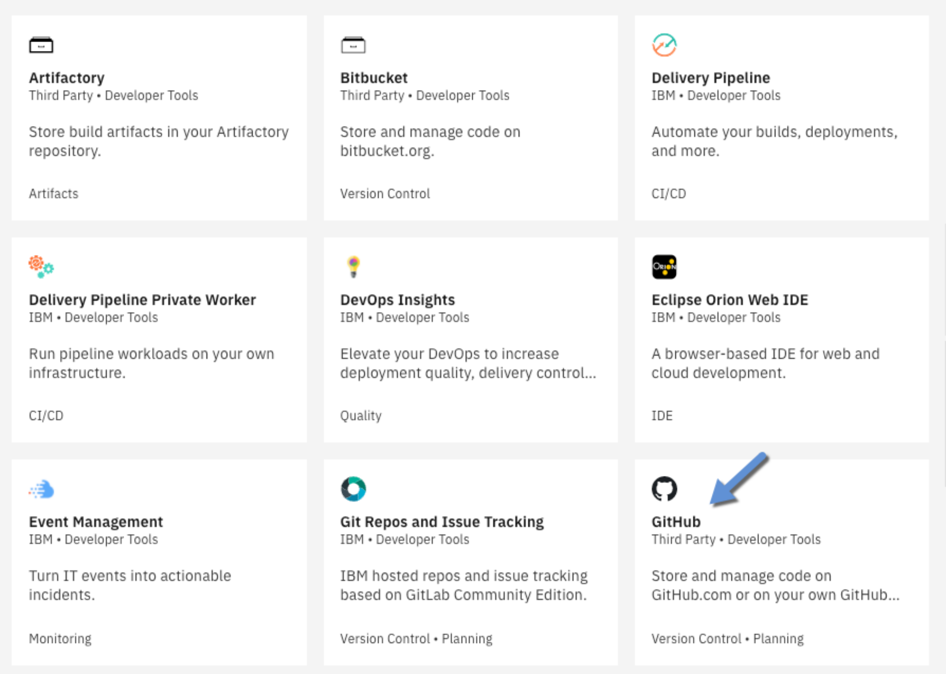
1 – 1 GitHub Integration

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

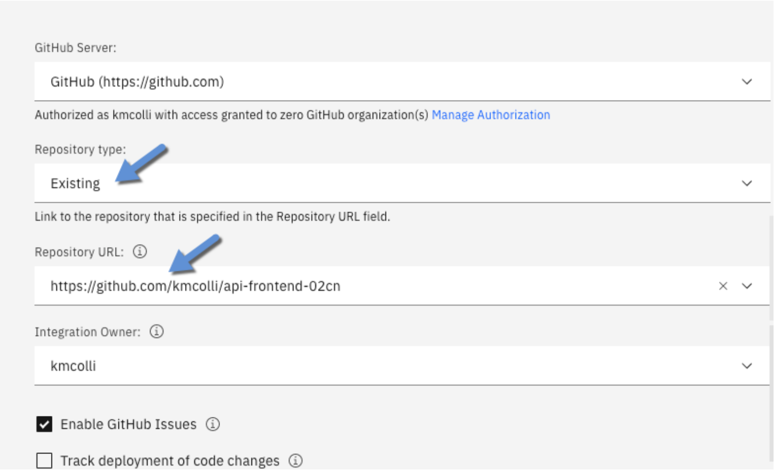
Eclipse Orion Web IDE
IBM • Developer Tools
A browser-based IDE for web and cloud development.
IDE

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.



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

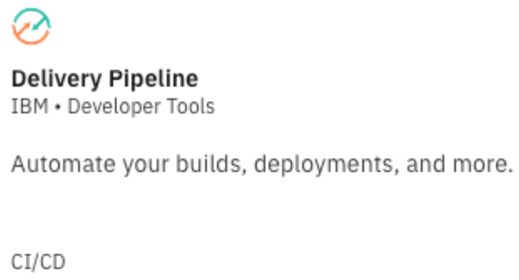
Integration Owner: ⓘ
kmcolli

☑ 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**.

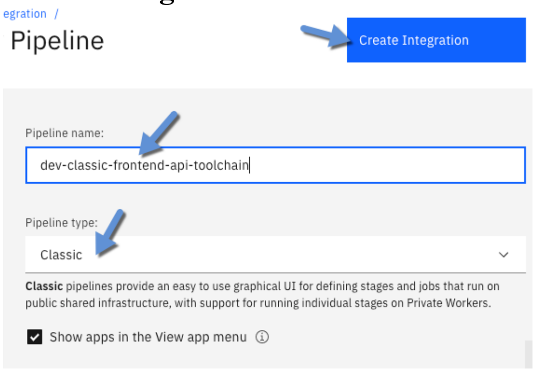


Delivery Pipeline
IBM • Developer Tools

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**.



Pipeline name: dev-classic-frontend-api-toolchain

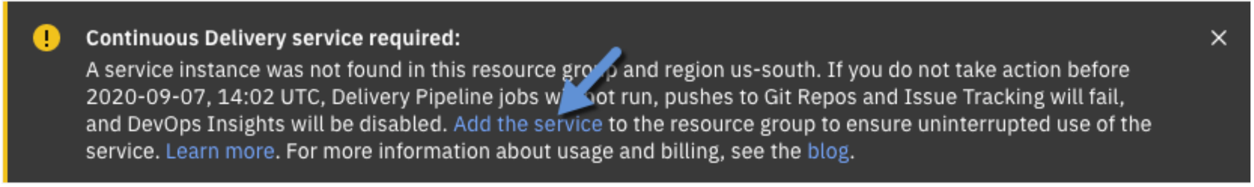
Pipeline type: Classic

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.

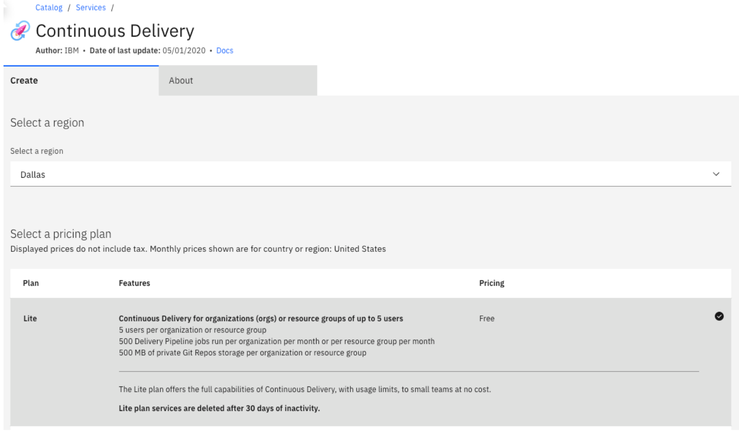


Continuous Delivery service required:

A service instance was not found in this resource group and region us-south. If you do not take action before 2020-09-07, 14:02 UTC, Delivery Pipeline jobs will not run, pushes to Git Repos and Issue Tracking will fail, and DevOps Insights will be disabled. Add the service to the resource group to ensure uninterrupted use of the service. Learn more. For more information about usage and billing, see the blog.

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**.



Continuous Delivery

Author: IBM • Date of last update: 05/01/2020 • Docs

Select a region

Select a region: Dallas

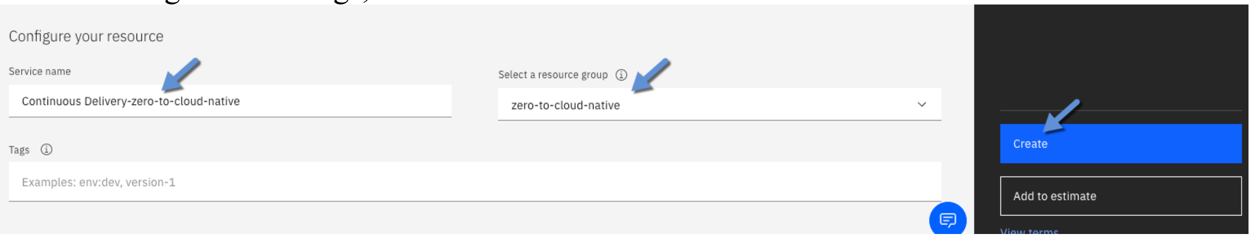
Select a pricing plan

Displayed prices do not include tax. Monthly prices shown are for country or region: United States

Plan	Features	Pricing
Lite	**Continuous Delivery for organizations (orgs) or resource groups of up to 5 users** 5 users per organization or resource group 500 Delivery Pipeline jobs run per organization per month or per resource group per month 500 MB of private Git Repos storage per organization or resource group The Lite plan offers the full capabilities of Continuous Delivery, with usage limits, to small teams at no cost. **Lite plan services are deleted after 30 days of inactivity.**	Free

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**.



Configure your resource

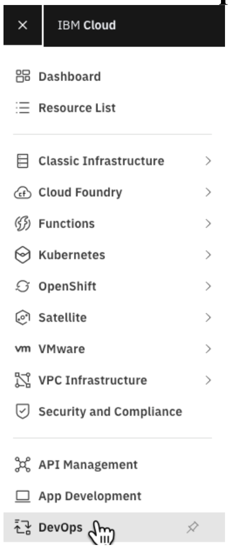
Service name: Continuous Delivery-zero-to-cloud-native

Select a resource group: zero-to-cloud-native

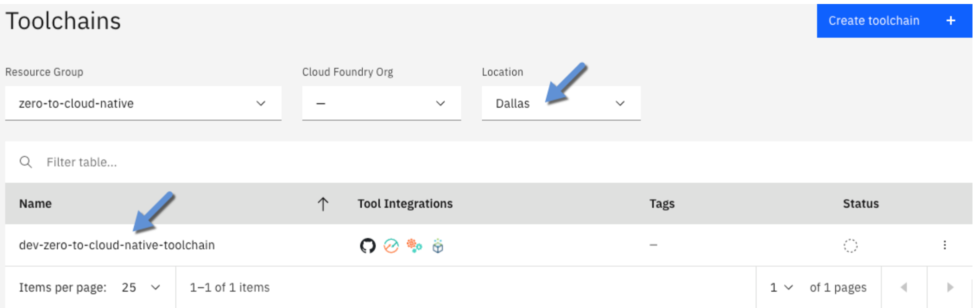
Tags: Examples: env:dev, version-1

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**.



Resource Group: zero-to-cloud-native

Cloud Foundry Org: –

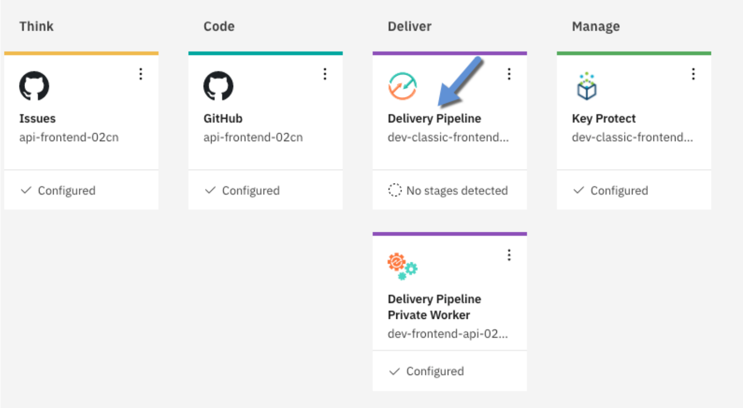
Location: Dallas

Filter table...

Name	Tool Integrations	Tags	Status
dev-zero-to-cloud-native-toolchain		–	

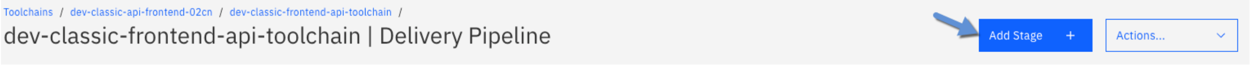
Items per page: 25 1–1 of 1 items 1 of 1 pages

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**.



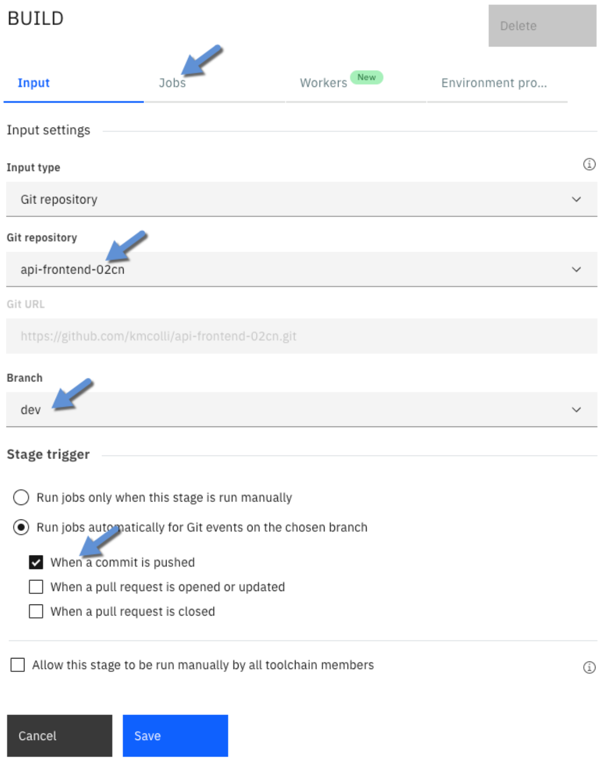
Toolchains / dev-classic-api-frontend-02cn / dev-classic-frontend-api-toolchain /

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

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

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

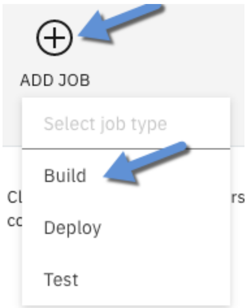
☑ When a commit is pushed

□ 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

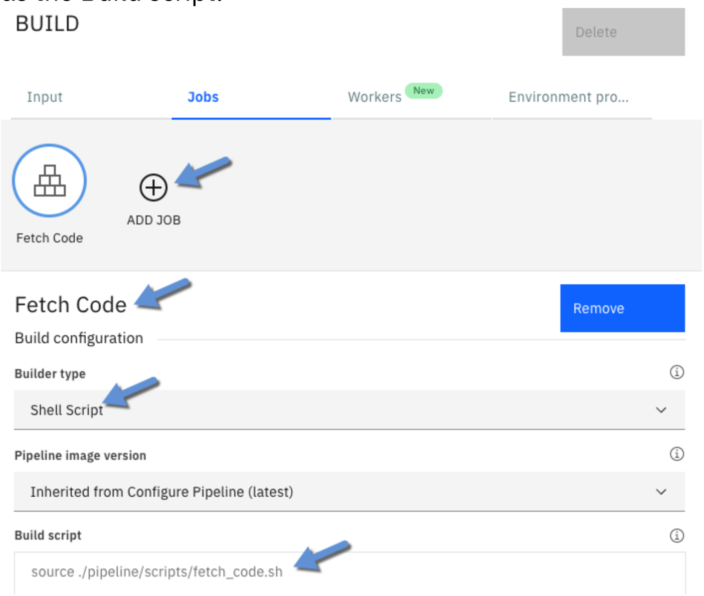
Don't click Save quite yet, instead click on the **Jobs** tab. A stage in a classic toolchain consists of a number of jobs. The first job we will create is fetching the code from GitHub. Click on **Add Job** and select **Build**.



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

Fetch Code

Build configuration

Builder type: Shell Script

Pipeline image version: Inherited from Configure Pipeline (latest)

Build script: source ./pipeline/scripts/fetch_code.sh

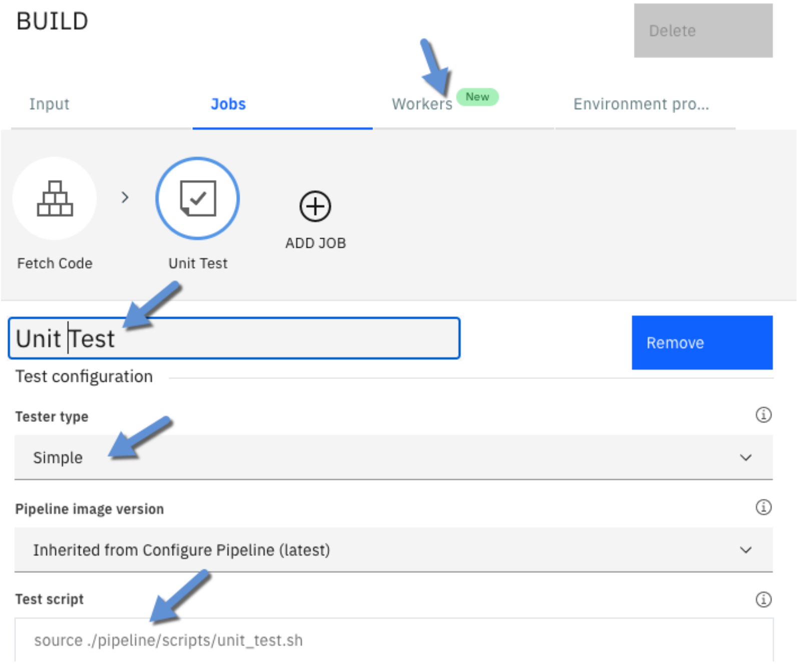
Note: you can view all the scripts used in this tutorial from the **pipeline/scripts** directory in each git repository:

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

Unit Test

Test configuration

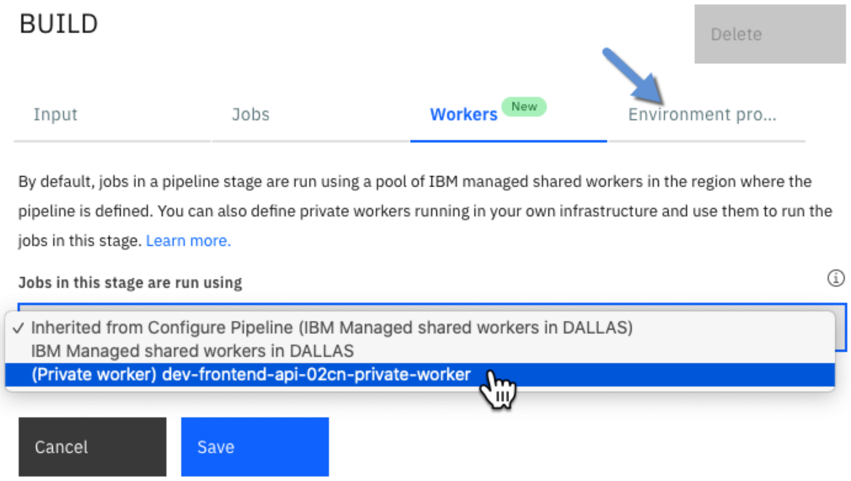
Tester type: Simple

Pipeline image version: Inherited from Configure Pipeline (latest)

Test script: source ./pipeline/scripts/unit_test.sh

Best Practice: You will note that we are using scripts stored in our git repository to perform the various tasks for stage jobs versus entering the script code directly in the Job. It is much easier to maintain and develop these scripts in a single place outside of the toolchain itself. Furthermore, once we commit changes to our scripts, the commit will invoke our pipeline with the new scripts.

To configure your pipeline with the private worker you created, click on the **Workers tab** and select the private worker you previously created.

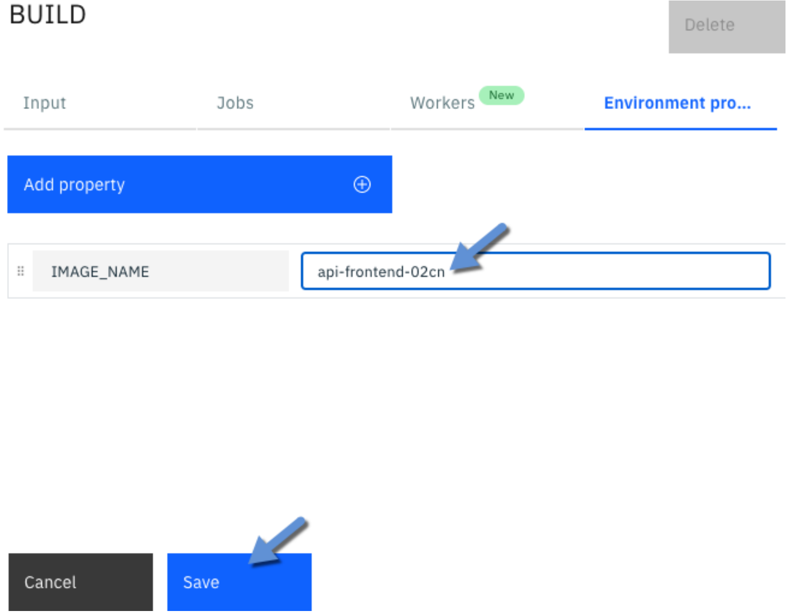


By default, jobs in a pipeline stage are run using a pool of IBM managed shared workers in the region where the pipeline is defined. You can also define private workers running in your own infrastructure and use them to run the jobs in this stage. Learn more.

Jobs in this stage are run using

✓ Inherited from Configure Pipeline (IBM Managed shared workers in DALLAS)
IBM Managed shared workers in DALLAS
(Private worker) dev-frontend-api-02cn-private-worker

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.

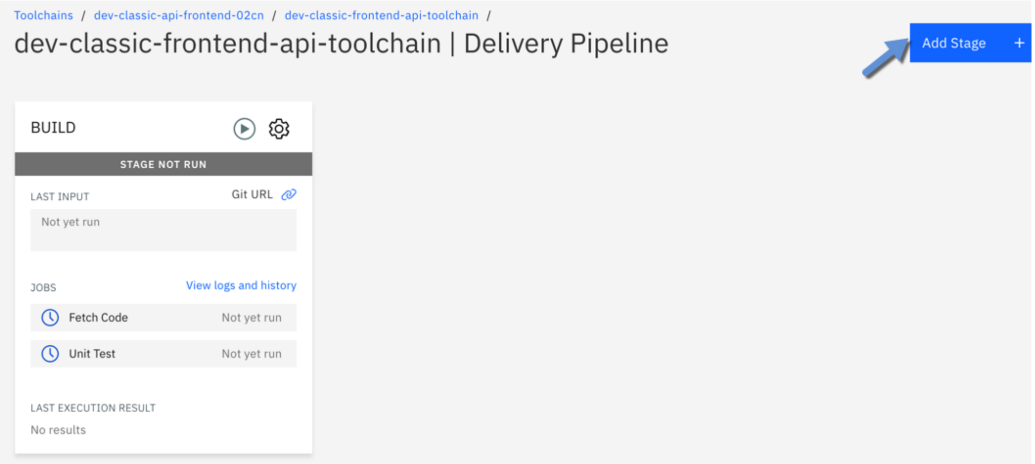


IMAGE_NAME api-frontend-02cn

That completes the build stage.

1 – 3 – 2 Containerize Stage

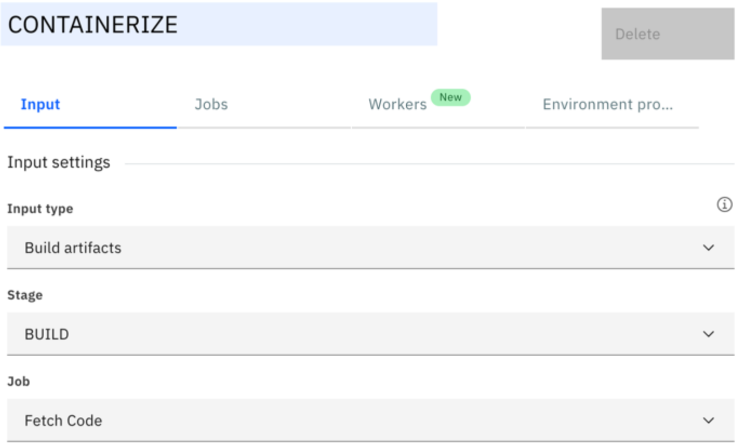
The next stage we will create is the containerization stage. Click on **Add Stage** to create a new stage.



Toolchains / dev-classic-api-frontend-02cn / dev-classic-frontend-api-toolchain /

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

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.



CONTAINERIZE

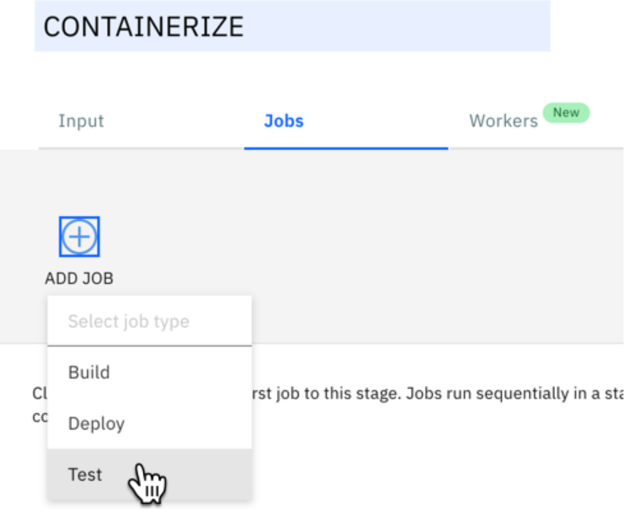
Input settings

Input type: Build artifacts

Stage: BUILD

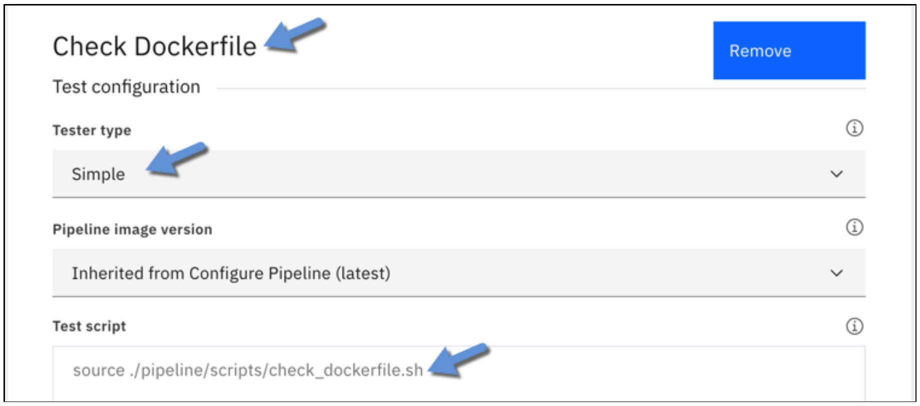
Job: Fetch Code

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



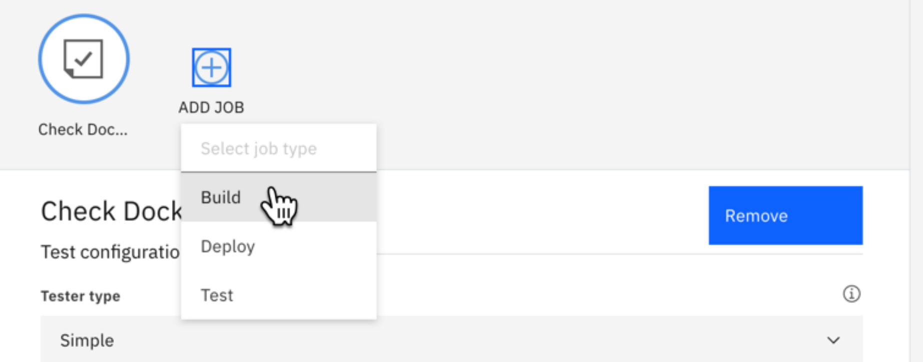
Check Dockerfile

Tester type: Simple

Pipeline image version: Inherited from Configure Pipeline (latest)

Test script: source ./pipeline/scripts/check_dockerfile.sh

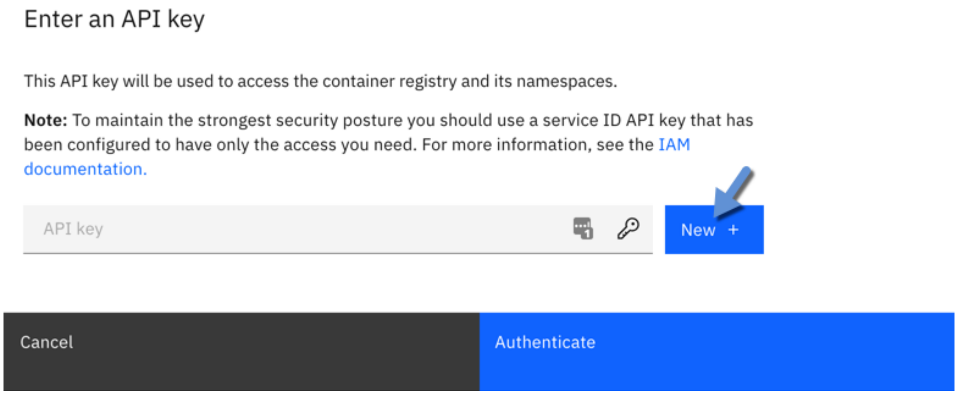
Create a second job by clicking **Add Job** and select **build**.



Tester type: Simple

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.



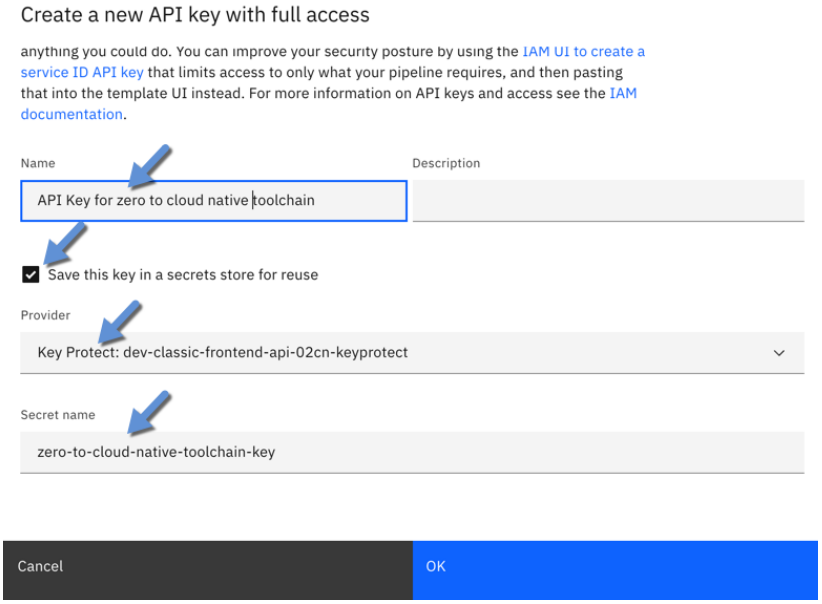
Enter an API key

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

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:

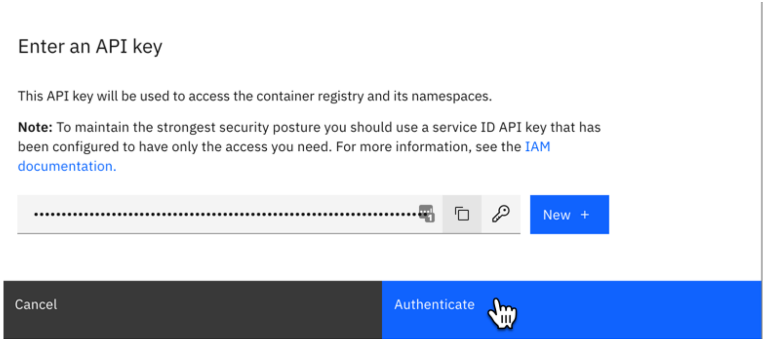
☑ Save this key in a secrets store for reuse

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

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

After entering all of these settings click OK.

On the next screen, click Authenticate.



Enter an API key

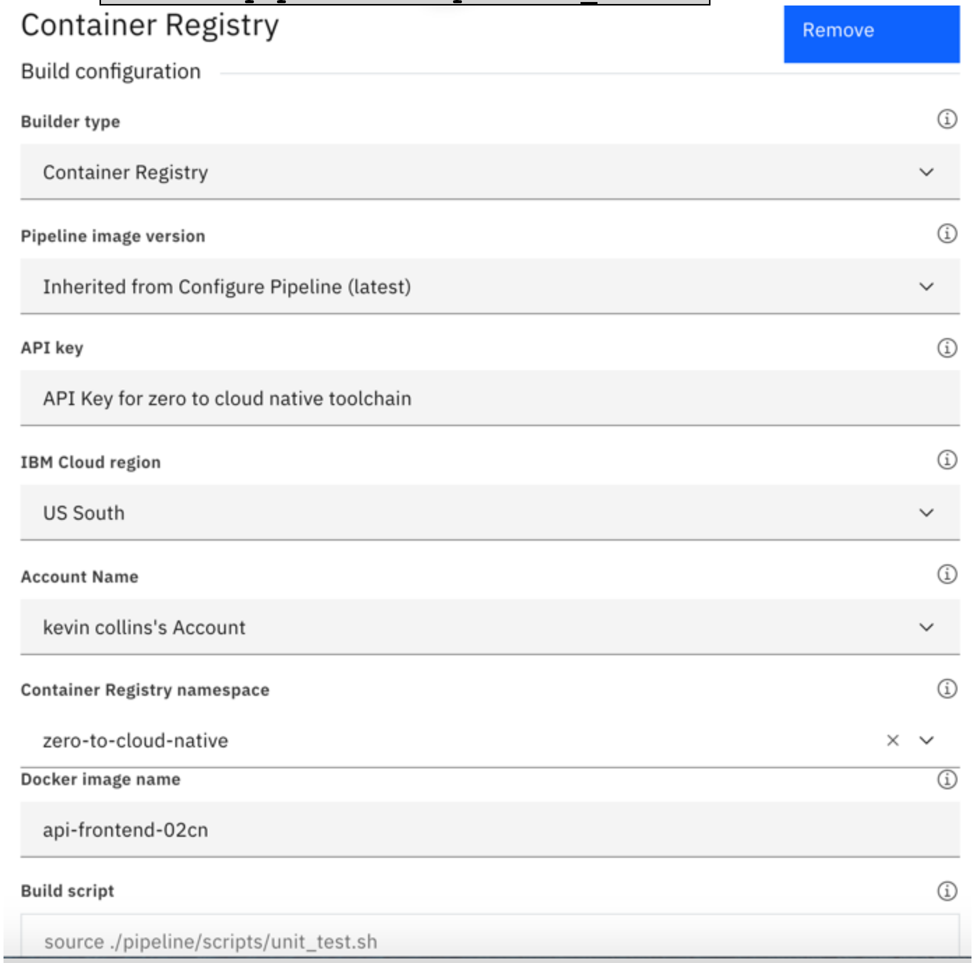
This API key will be used to access the container registry and its namespaces.

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.

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

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

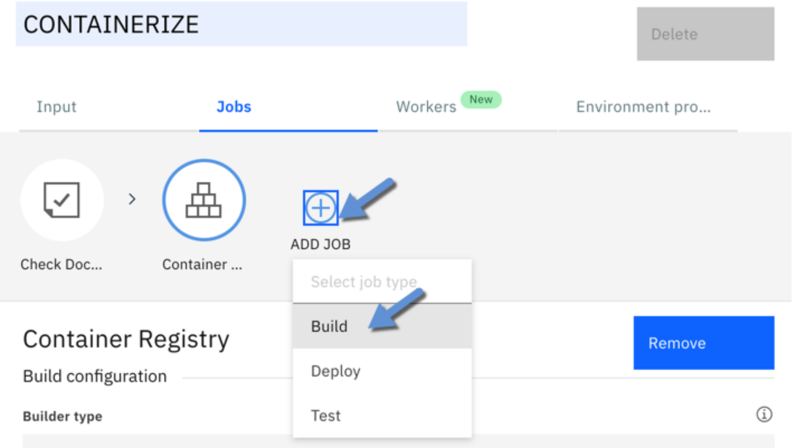
Container Registry namespace: zero-to-cloud-native

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**.

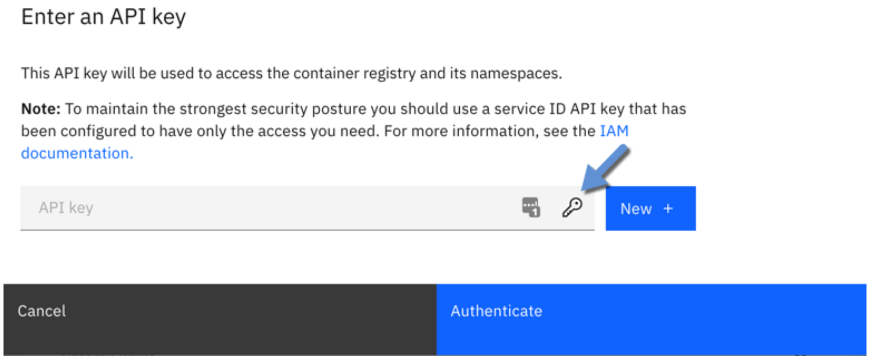


Container Registry

Build configuration

Builder type

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**.

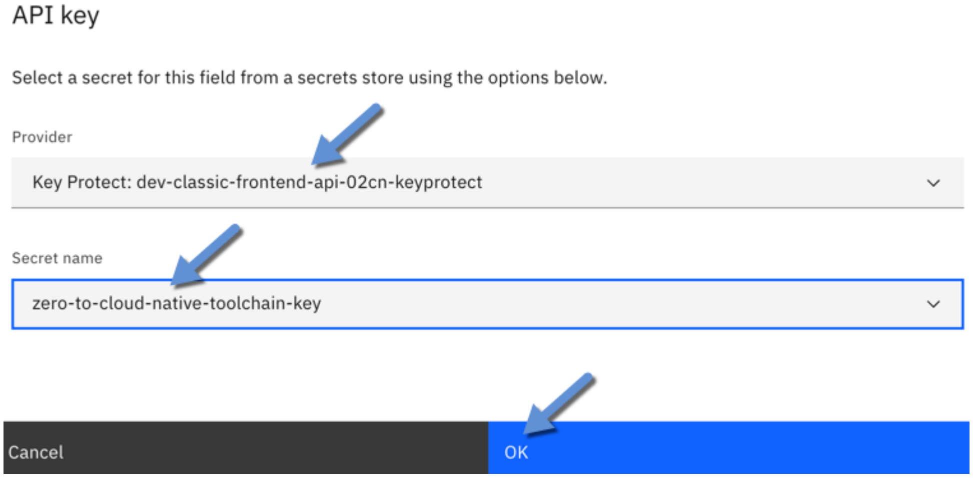


Enter an API key

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.

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



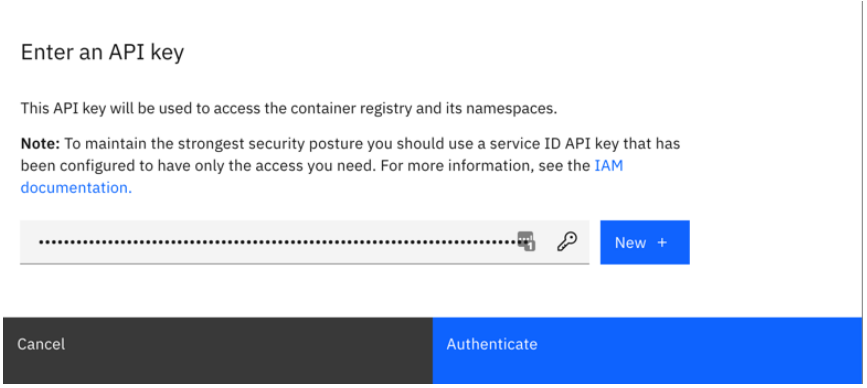
API key

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

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



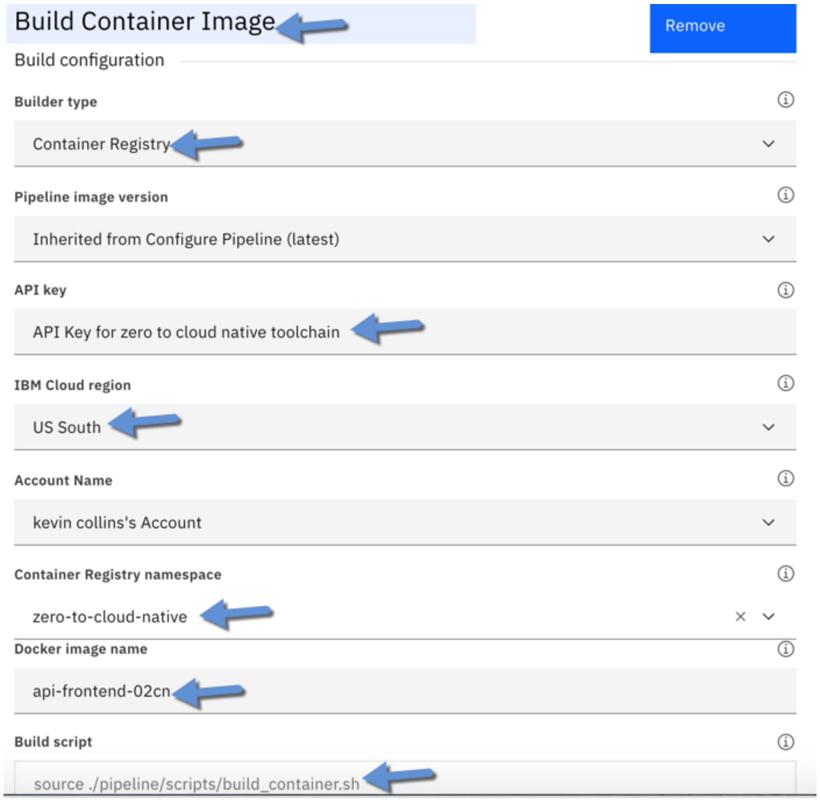
Enter an API key

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.

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
```



Build Container Image

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

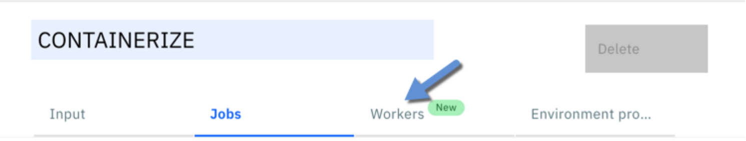
Container Registry namespace: zero-to-cloud-native

Docker image name: api-frontend-02cn

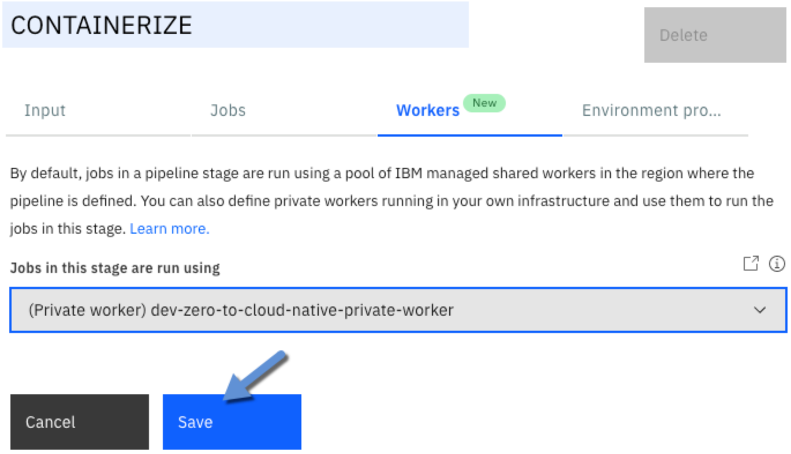
Build script: source ./pipeline/scripts/build_container.sh

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.



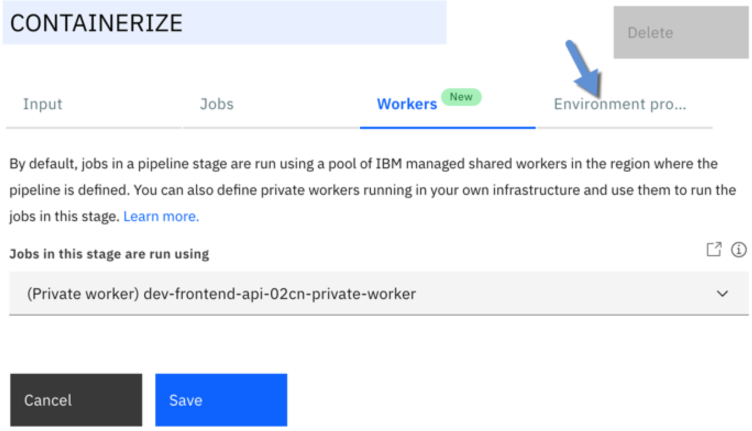
By default, jobs in a pipeline stage are run using a pool of IBM managed shared workers in the region where the pipeline is defined. You can also define private workers running in your own infrastructure and use them to run the jobs in this stage. Learn more.

Jobs in this stage are run using

(Private worker) dev-zero-to-cloud-native-private-worker

Before clicking save, we need to first set a couple environment properties.

Click on the Environment Properties tab.

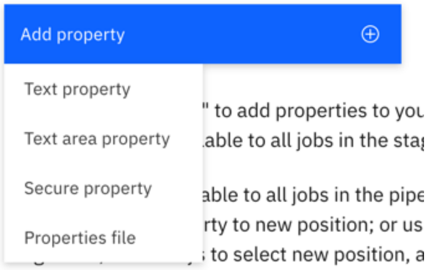


By default, jobs in a pipeline stage are run using a pool of IBM managed shared workers in the region where the pipeline is defined. You can also define private workers running in your own infrastructure and use them to run the jobs in this stage. Learn more.

Jobs in this stage are run using

(Private worker) dev-frontend-api-02cn-private-worker

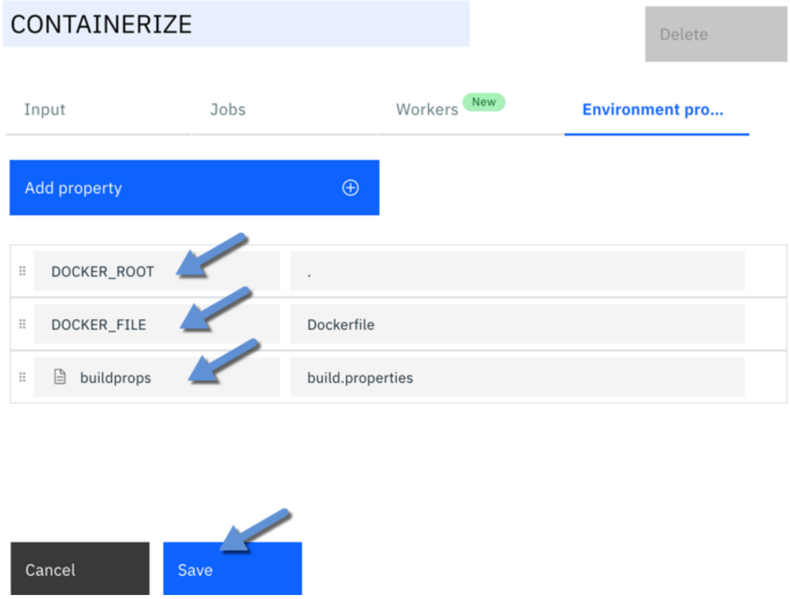
There are three Environment Properties we will need to add. To add a property, click on Add Property and enter the property type.


Add property
Text property
Text area property
Secure property
Properties file
" to add properties to you
able to all jobs in the sta
able to all jobs in the pipe
rty to new position; or us
to select new position, a

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.

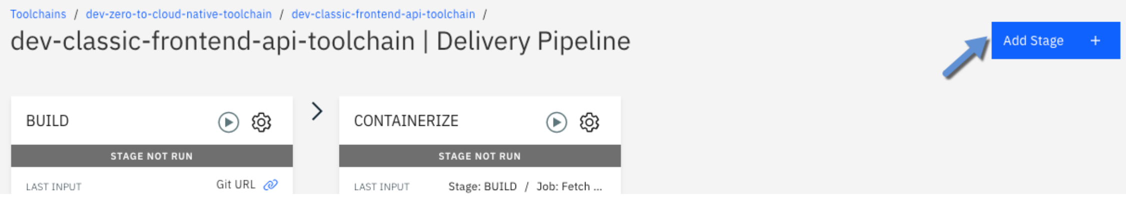


DOCKER_ROOT	.
DOCKER_FILE	Dockerfile
buildprops	build.properties

This completes the containerize job.

1 – 3 – 3 Deploy Stage

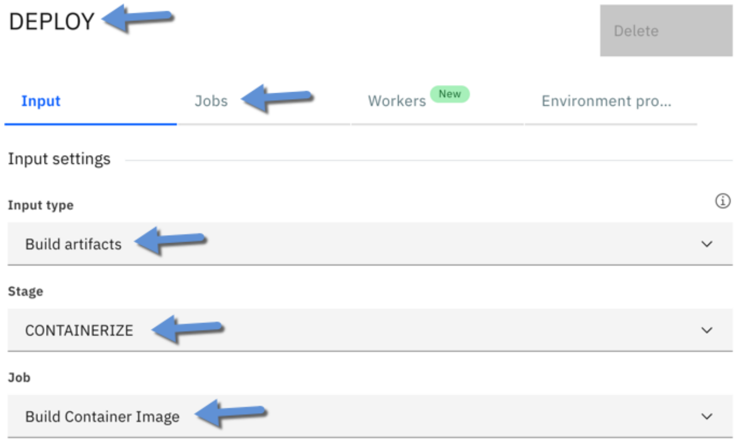
The final stage we need to create is the **deploy stage** where we will deploy the container into our OpenShift. Click **Add Stage.**



Toolchains / dev-zero-to-cloud-native-toolchain / dev-classic-frontend-api-toolchain /

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

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.



DEPLOY

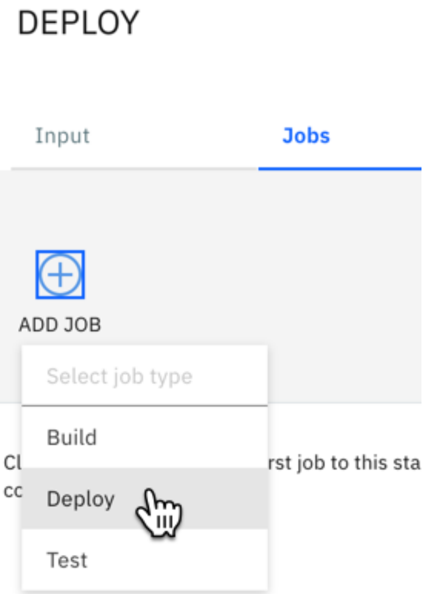
Input settings

Input type: Build artifacts

Stage: CONTAINERIZE

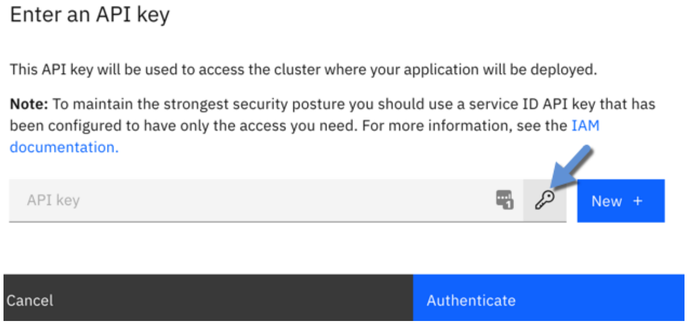
Job: Build Container Image

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.**



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.



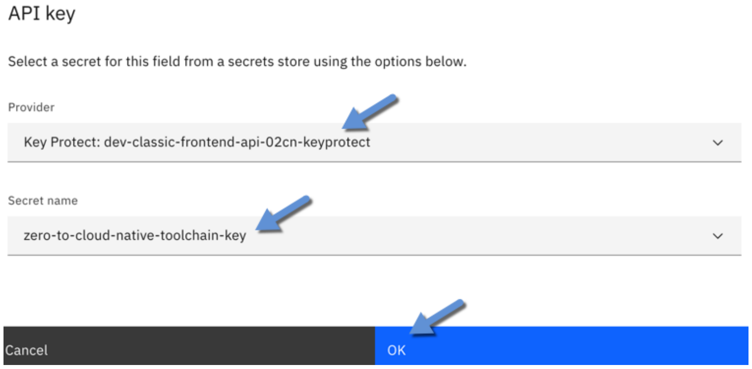
Enter an API key

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

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



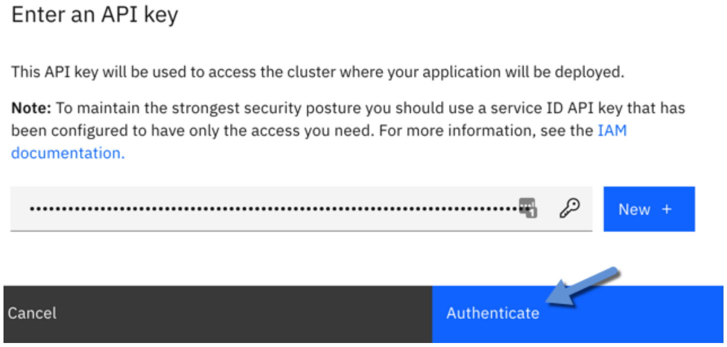
API key

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

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.

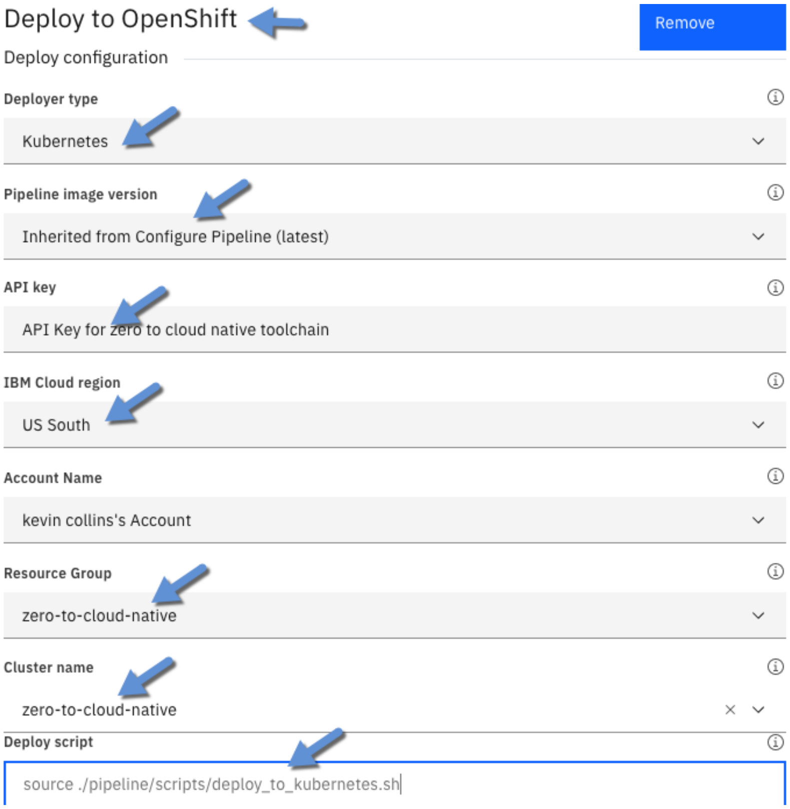
Note: To maintain the strongest security posture you should use a service ID API key that has been configured to have only the access you need. For more information, see the IAM documentation.

••

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

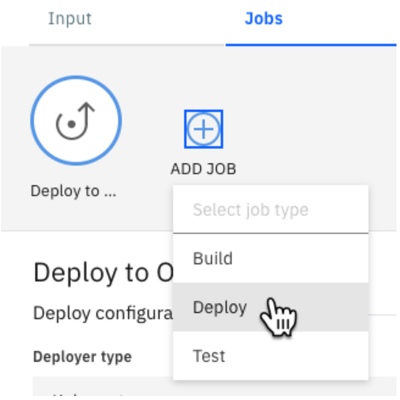
Resource Group: zero-to-cloud-native

Cluster name: zero-to-cloud-native

Deploy script: source ./pipeline/scripts/deploy_to_kubernetes.sh

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

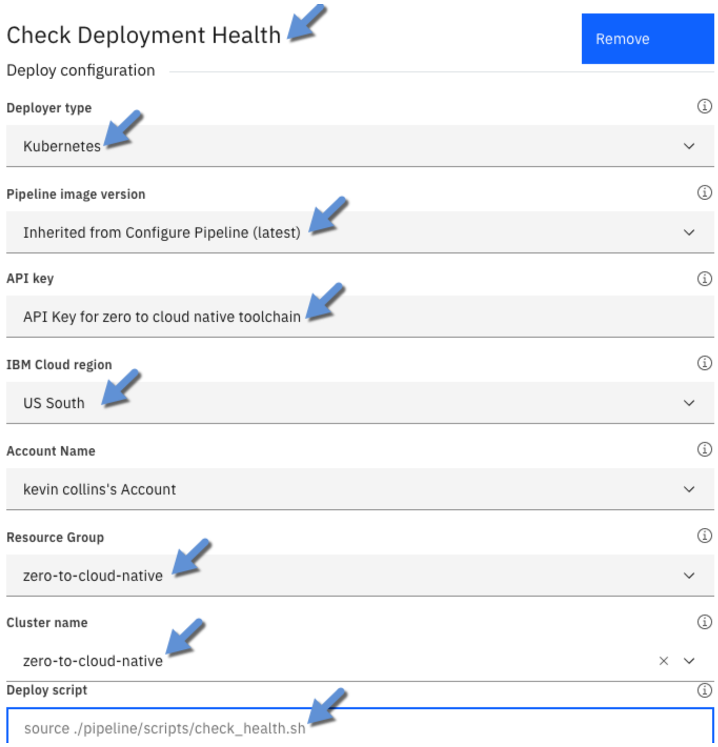


Change the name to **Check Deployment Health** and deployment type to **Kubernetes**. Select your API Key from Key Protect the same way as you did above.

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
```



Check Deployment Health

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

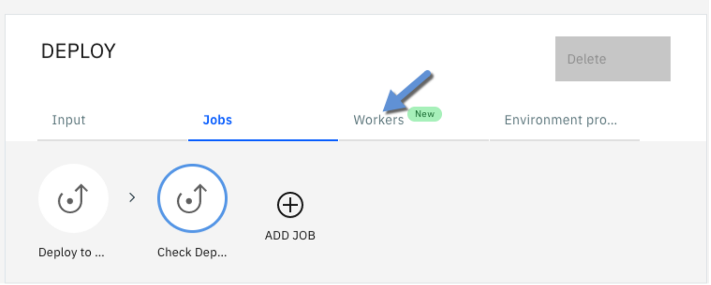
Account Name: kevin collins's Account

Resource Group: zero-to-cloud-native

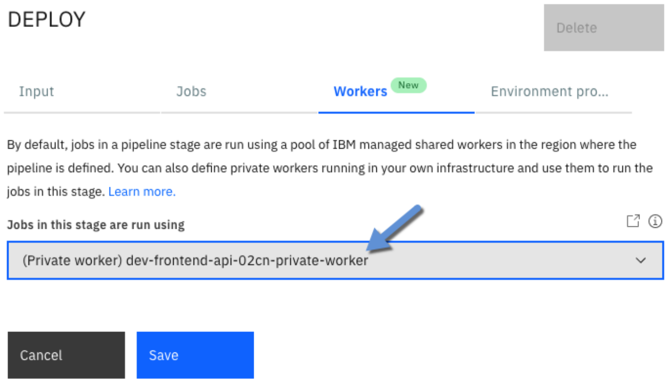
Cluster name: zero-to-cloud-native

Deploy script: source ./pipeline/scripts/check_health.sh

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



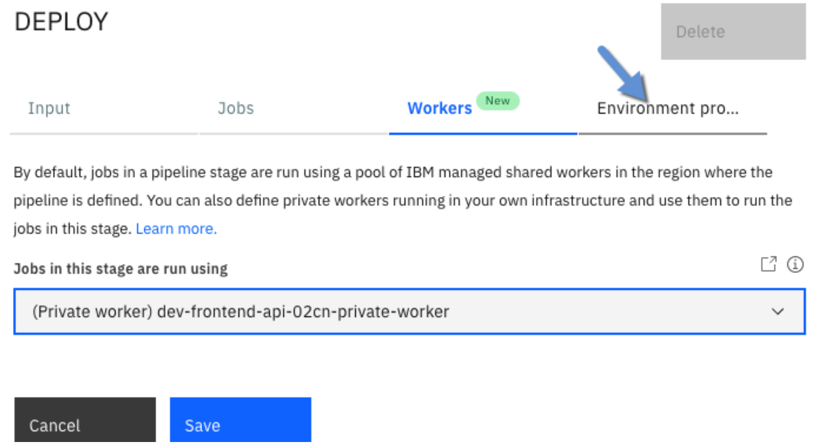
On the next screen, select the Private Worker that we created.



By default, jobs in a pipeline stage are run using a pool of IBM managed shared workers in the region where the pipeline is defined. You can also define private workers running in your own infrastructure and use them to run the jobs in this stage. Learn more.

Jobs in this stage are run using: (Private worker) dev-frontend-api-02cn-private-worker

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

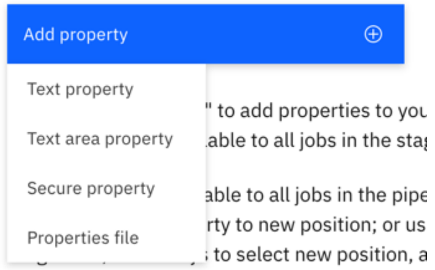


By default, jobs in a pipeline stage are run using a pool of IBM managed shared workers in the region where the pipeline is defined. You can also define private workers running in your own infrastructure and use them to run the jobs in this stage. Learn more.

Jobs in this stage are run using

(Private worker) dev-frontend-api-02cn-private-worker

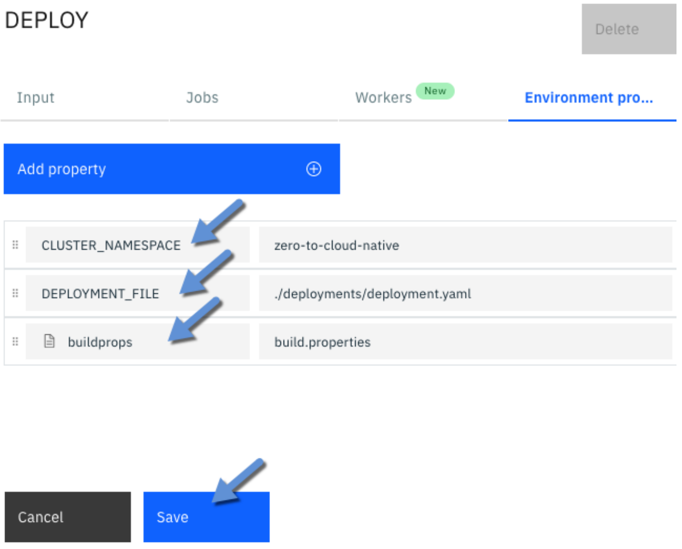
There are three Environment Properties we will need to add. To add a property, click on Add Property and enter the property type.


Add property
Text property
Text area property
Secure property
Properties file
" to add properties to you
able to all jobs in the sta
able to all jobs in the pipe
rty to new position; or us
to select new position, a

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

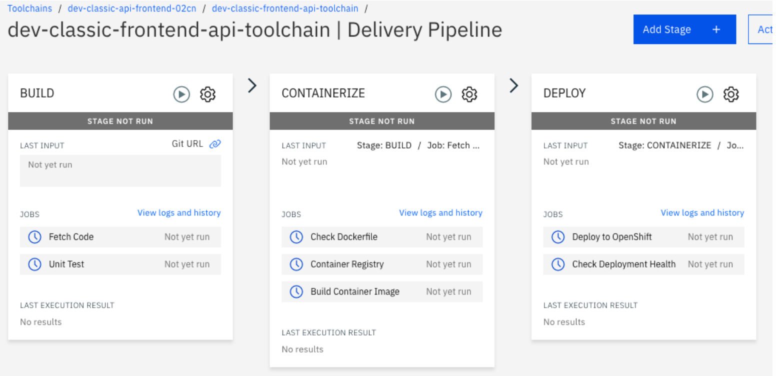
Property Type	**Property Name**	**Value**
Text Property	CLUSTER_NAMEPACE	zero-to-cloud-native
Text Property	DEPLOYMENT_FILE	./deployments/deployment.yaml
Properties File	buildprops	build.properties

After entering all of these properties, click Save.



CLUSTER_NAMESPACE	zero-to-cloud-native
DEPLOYMENT_FILE	./deployments/deployment.yaml
buildprops	build.properties

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



Toolchains / dev-classic-api-frontend-02cn / dev-classic-frontend-api-toolchain /

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

BUILD

STAGE NOT RUN

LAST INPUT Git URL

Not yet run

JOBS View logs and history

Fetch Code	Not yet run
Unit Test	Not yet run

LAST EXECUTION RESULT

No results

CONTAINERIZE

STAGE NOT RUN

LAST INPUT Stage: BUILD / Job: Fetch ...

Not yet run

JOBS View logs and history

Check Dockerfile	Not yet run
Container Registry	Not yet run
Build Container Image	Not yet run

LAST EXECUTION RESULT

No results

DEPLOY

STAGE NOT RUN

LAST INPUT Stage: CONTAINERIZE / Jo...

Not yet run

JOBS View logs and history

Deploy to OpenShift	Not yet run
Check Deployment Health	Not yet run

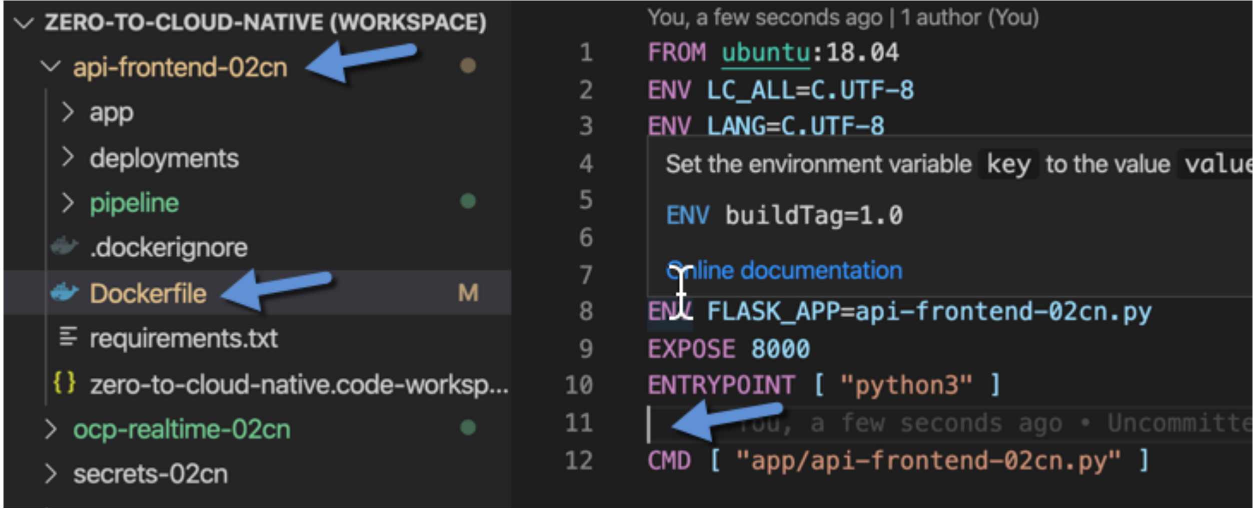
LAST EXECUTION RESULT

No results

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.



You, a few seconds ago | 1 author (You)

```
FROM ubuntu:18.04
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
ENV FLASK_APP=api-frontend-02cn.py
EXPOSE 8000
ENTRYPOINT [ "python3" ]
                      You, a few seconds ago • Uncommitte
CMD [ "app/api-frontend-02cn.py" ]
```

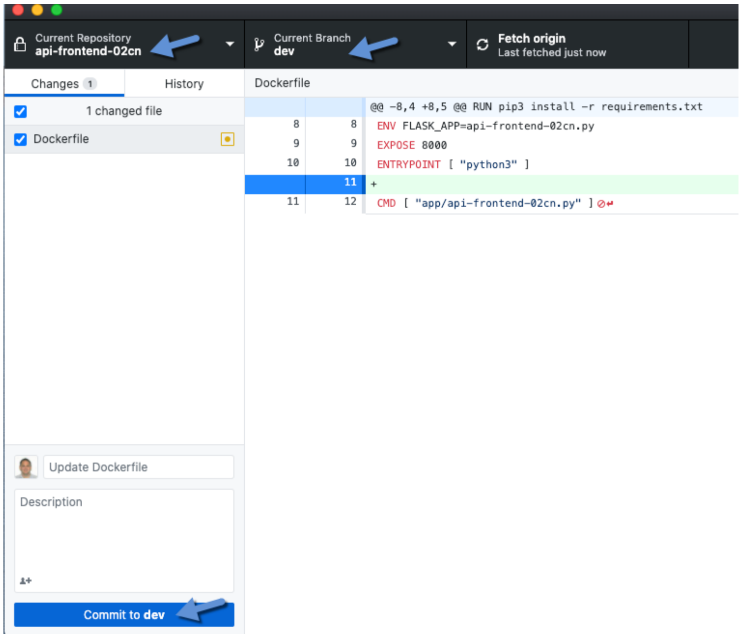
Set the environment variable key to the value value

```
ENV buildTag=1.0
```

Online documentation

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**.

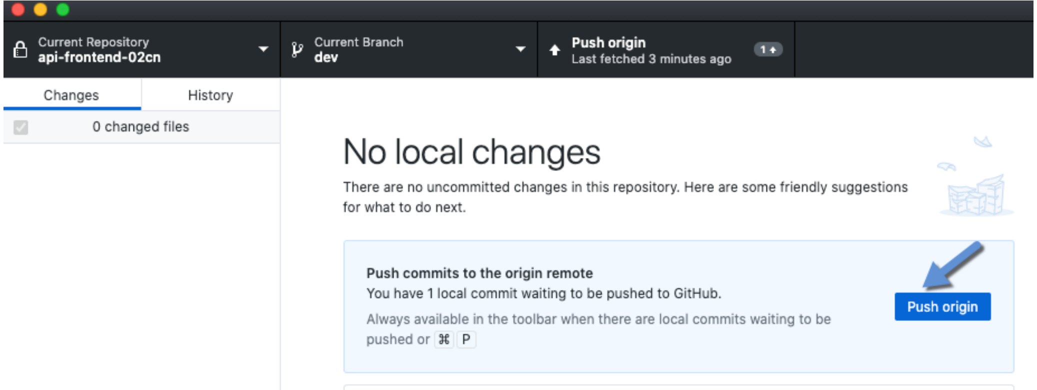


```
@@ -8,4 +8,5 @@ RUN pip3 install -r requirements.txt
 ENV FLASK_APP=api-frontend-02cn.py
 EXPOSE 8000
 ENTRYPOINT [ "python3" ]
+
 CMD [ "app/api-frontend-02cn.py" ]
```

Update Dockerfile

Description

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



No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Push commits to the origin remote

You have 1 local commit waiting to be pushed to GitHub.

Always available in the toolbar when there are local commits waiting to be pushed or ⌘ P

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

3B – IBM Cloud Continuous Delivery – Tekton Pipeline