**Practical Exercise: Run WSO2 Micro Integrator on Containers using K8s-APIM Operator**

Training Objective

Understand the basics of using WSO2 Micro Integrator on Kubernetes and try out a few simple use cases using the K8s-APIM operator.

High Level Steps

* Create the integration artifacts.
* Create a Docker/Kubernetes project.
* Build and run the Docker image.

Detailed Instructions

K8s Deployment Sample 1: Hello World Scenario

Let's define a basic Hello World scenario using WSO2 Micro Integrator and deploy it on your Kubernetes environment.

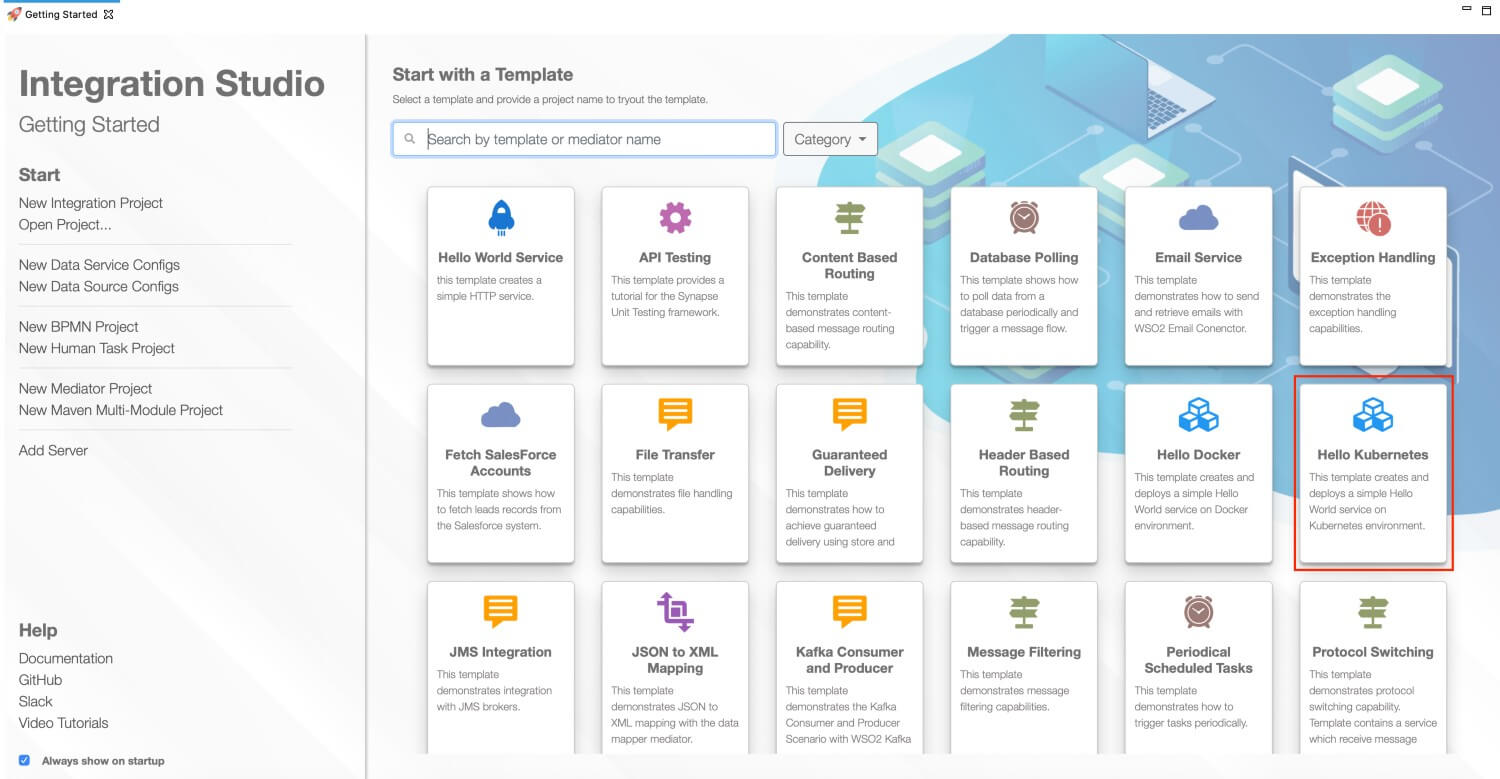
Prerequisites

* Install and set up [WSO2 Integration Studio](https://apim.docs.wso2.com/en/4.1.0/integrate/develop/installing-wso2-integration-studio).
* Install a [Kubernetes](https://kubernetes.io/docs/setup/" \t "_blank) cluster and **v1.11+** client. Alternatively, you can [run Kubernetes locally via Minikube](https://kubernetes.io/docs/setup/learning-environment/minikube/).
* Install [Docker](https://docs.docker.com/" \t "_blank).
* Install the [Kubernetes API Operator](https://apim.docs.wso2.com/en/4.1.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/install).

Step 1 - Create the integration solution

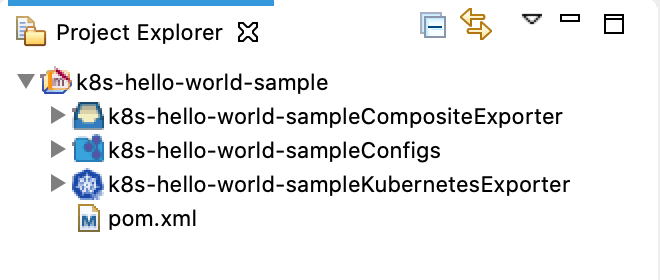
Let's use an integration template in WSO2 Integration Studio to generate a sample integration solution that returns a 'Hello World' response when invoked.

1. Open WSO2 Integration Studio.
2. In the **Getting Started** view, select the **Hello Kubernetes** template.



1. Give a project name and click **Finish**.

This generates the complete integration project with the 'Hello World' solution, which is ready to be deployed in Kubernetes.



Step 2 - Build and Push the Docker image

**Note**

Be sure to start your Docker instance before building the image. If Docker is not started, the build process will fail.

There are two ways to build a Docker image of the integration solution and push it to your Docker registry:

* Using **Maven**:

**Before you begin**

You need **Maven 3.5.2** or a later version when you build the Docker image manually (without using WSO2 Integration Studio).

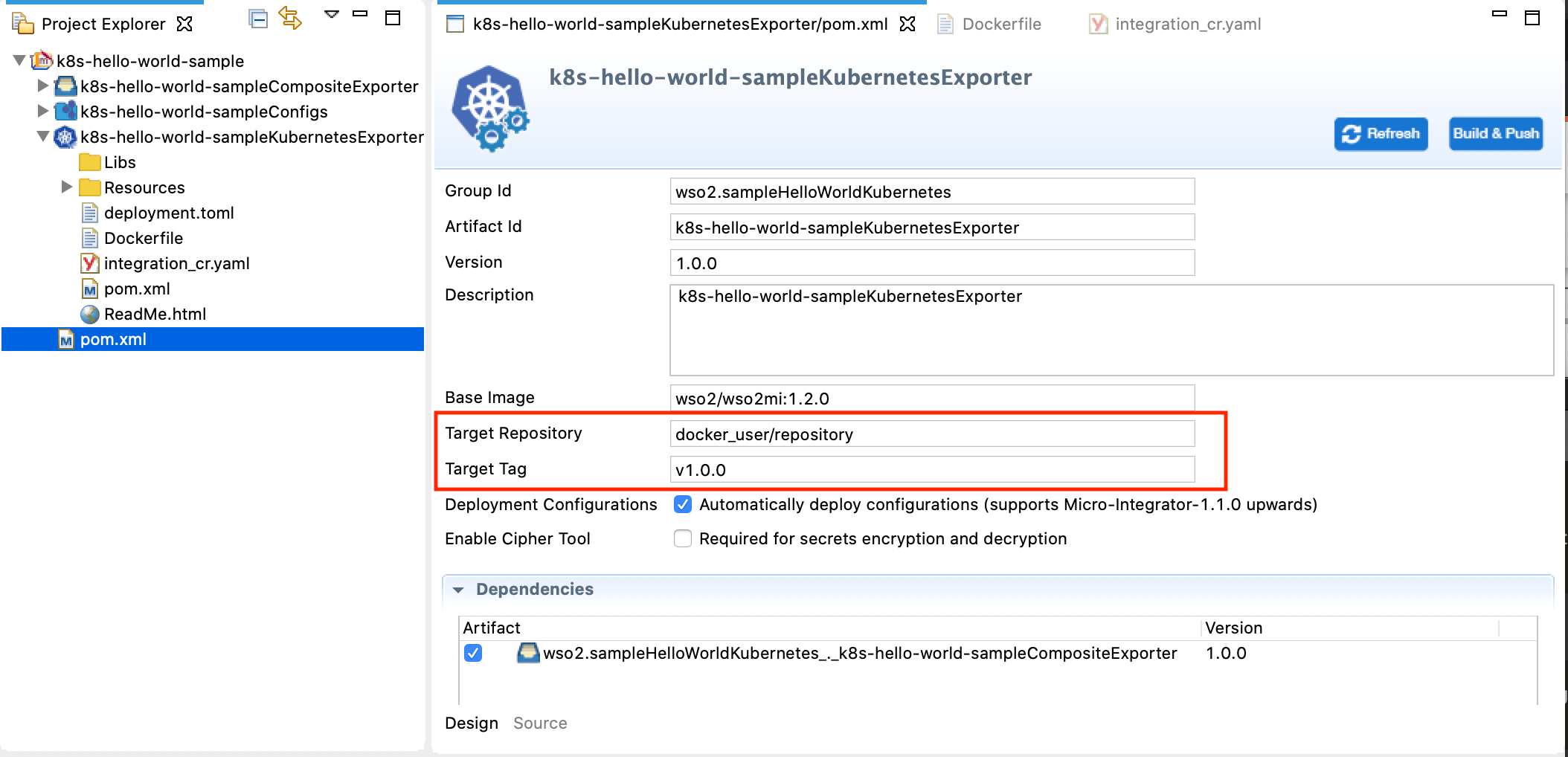
* 1. Open a terminal and navigate to the integration project.
  2. Execute the following command.

Be sure to specify the user name and password of the correct Docker registry.

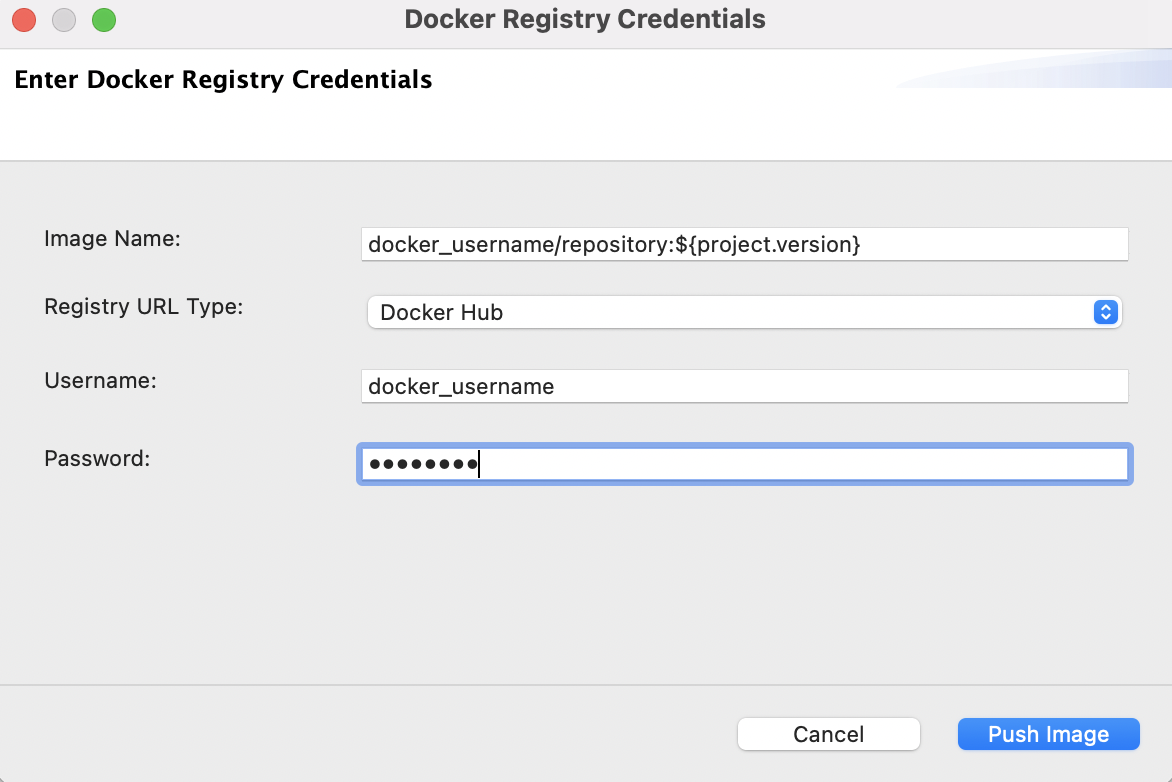
mvn clean install -Dmaven.test.skip=true -Ddockerfile.username={username} -Ddockerfile.password={password}

This will build the Docker image and then push it to the specified Docker registry.

* Using **WSO2 Integration Studio**:
  1. Open the **pom.xml** file in the Kubernetes project as shown below.



* 1. Ensure that the composite exporter is selected under **Dependencies**.
  2. In the **Target Repository** field, enter the name of the Docker registry to which you will push a Docker image.
  3. Click **Build & Push** to build the image and push to the Docker registry.
  4. In the dialog box that opens, enter the credentials of your Docker registry to which the image should be pushed.



* 1. Click **Push Image**.

Run the docker image ls command to verify that the Docker image is created.

Step 3 - Deploy the solution in K8s

**Info**

**Before you begin**, the [API Kubernetes Operator](https://apim.docs.wso2.com/en/4.1.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/install) should be installed in your Kubernetes environment.

Follow the steps given below.

1. Open the integration\_cr.yaml file from the Kubernetes exporter in WSO2 Integration Studio.
2. See that the **integration** details of the hello-world solution are updated. **Be sure** to add the image name in the following format: docker\_user/repository:tag

apiVersion: "wso2.com/v1alpha2"

kind: "Integration"

metadata:

name: "hello-world"

spec:

image: "<Docker image for the Hello World Scenario>"

deploySpec:

minReplicas: 1

1. Open a terminal and start the Kubernetes cluster.
2. Navigate to the location of your integration\_cr.yaml file and execute the following command to deploy the integration solution in the Kubernetes cluster:

kubectl apply -f integration\_cr.yaml

When the integration is successfully deployed, it should create the hello-world integration, hello-world-deployment, hello-world-service, and ei-operator-ingress as follows:

**Tip**

The ei-operator-ingress is not created if you have [disabled the ingress controller](https://apim.docs.wso2.com/en/4.1.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/manage-integrations/integration-deployments/#disable-ingress-controller-creation).

kubectl get integration

NAME STATUS SERVICE-NAME AGE

hello-world Running hello-service 2m

kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

hello-world-deployment 1/1 1 1 2m

kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

hello-world-service ClusterIP 10.101.107.154 <none> 8290/TCP 2m

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 2d

k8s-api-operator ClusterIP 10.98.78.238 <none> 443/TCP 1d

kubectl get ingress

NAME HOSTS ADDRESS PORTS AGE

api-operator-ingress wso2ei.ingress.wso2.com 10.0.2.15 80, 443 2m

Step 4 - Test the deployment

Let's invoke the service without going through the ingress controller.

1. Apply port forwarding as shown below. This will allow you to invoke the service without going through the Ingress controller:

kubectl port-forward service/hello-world-service 8290:8290

1. Invoke the service as follows:

curl http://localhost:8290/HelloWorld

You will receive the following response:

{"Hello":"World"}%

K8s Deployment Sample 2: Content Based Routing

Let's define a content-based routing scenario using WSO2 Micro Integrator and deploy it on your Kubernetes environment.

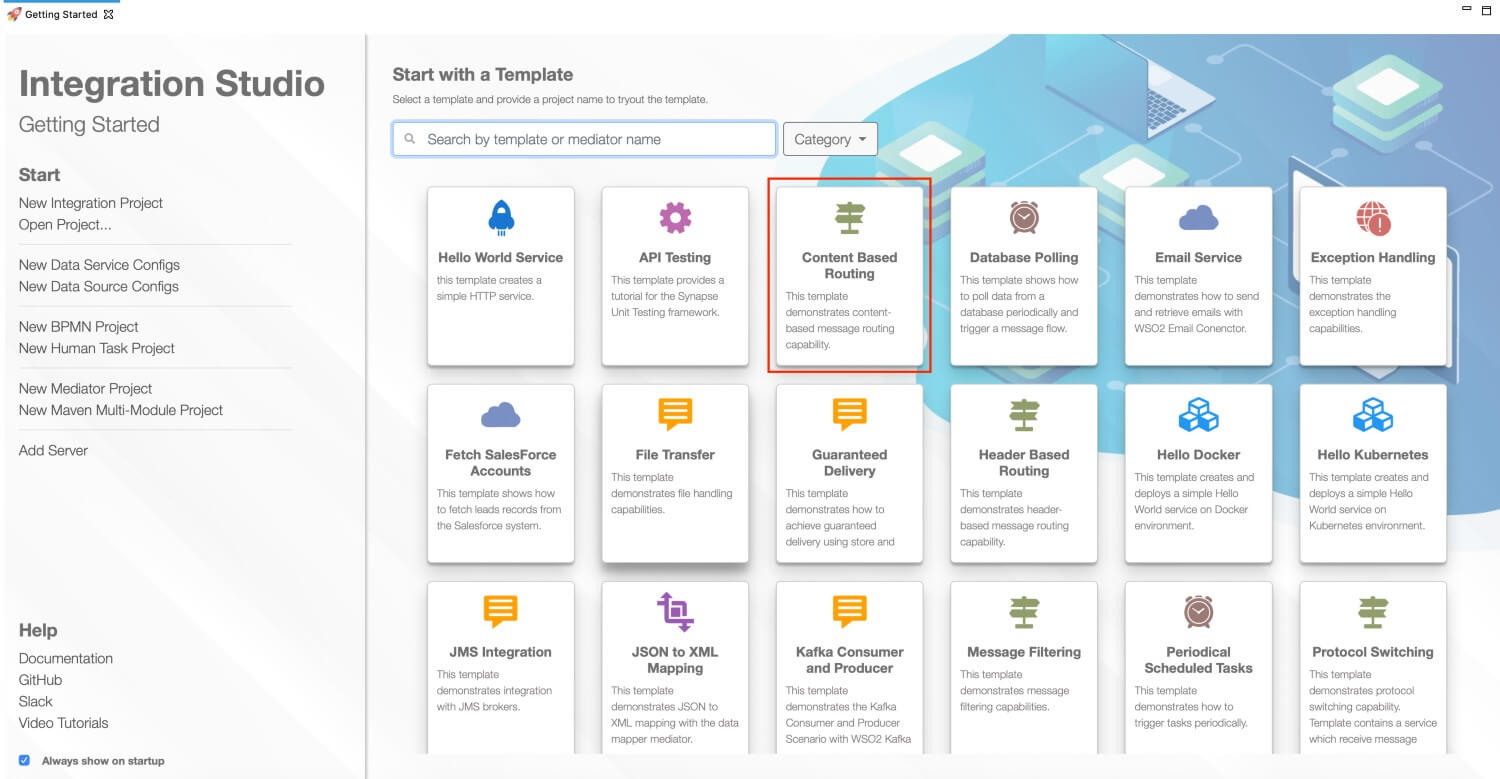
Prerequisites

* Install and set up [WSO2 Integration Studio](https://apim.docs.wso2.com/en/4.0.0/integrate/develop/installing-wso2-integration-studio).
* Install a [Kubernetes](https://kubernetes.io/docs/setup/" \t "_blank) cluster and **v1.11+** client. Alternatively, you can [run Kubernetes locally via Minikube](https://kubernetes.io/docs/setup/learning-environment/minikube/).
* Install [Docker](https://docs.docker.com/" \t "_blank).
* Install the [Kubernetes API Operator](https://apim.docs.wso2.com/en/4.0.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/install).

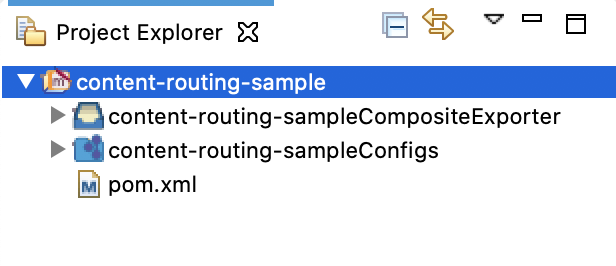
Step 1 - Create the integration solution

Let's use the **Content Routing** integration template in WSO2 Integration Studio:

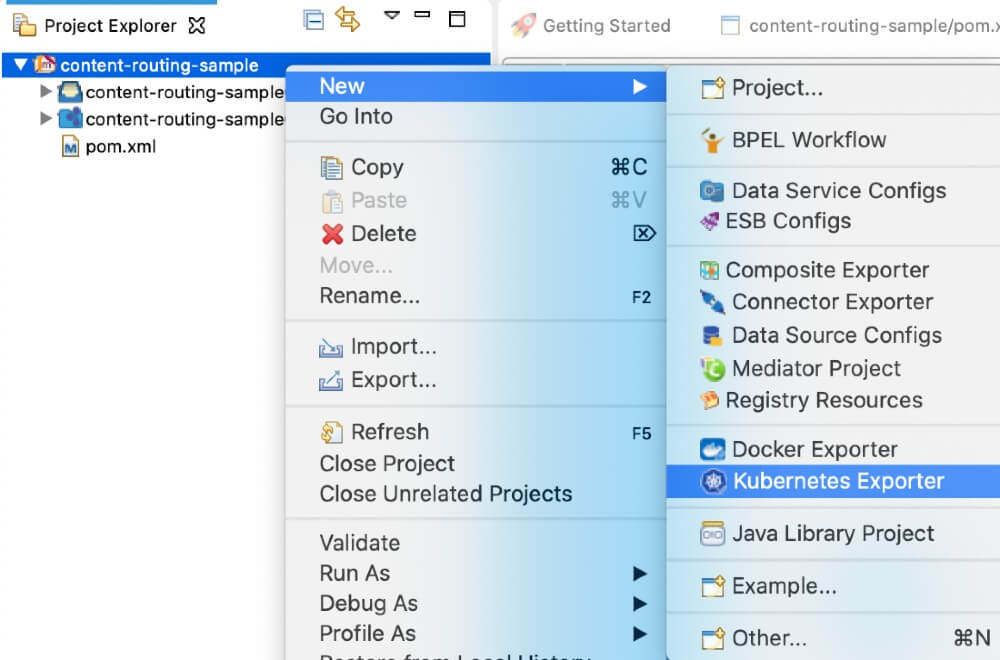
1. Open WSO2 Integration Studio.
2. In the **Getting Started** view, select the **Content Based Routing** template.



1. Give a project name and click **Finish**.



1. Create a **Kubernetes Project** inside the integration project.
   1. Right-click the **content-routing-sample** project, go to **New -> Kubernetes Exporter**:

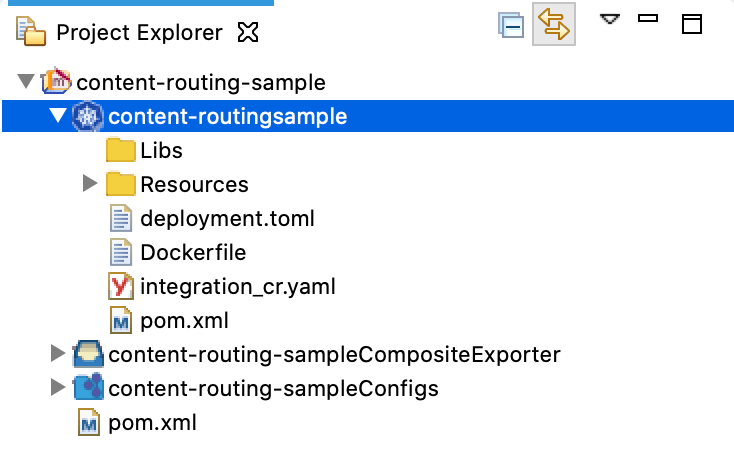


* 1. In the **Kubernetes Exporter Information for K8s EI Operator** dialog box that opens, enter the following details:

|  |  |
| --- | --- |
| Parameter | Description |
| Kubernetes Exporter Name | Give a unique name for the project. |
| Integration Name | This name will be used to identify the integration solution in the kubernetes custom resource. Let's use content-routing as the integration name for this example. |
| Number of Replicas | Specify the number of pods that should be created in the kubernetes cluster. |
| Base Image Repository | Specify the base Micro Integrator Docker image for your solution. For this example, let's use the Micro Integrator docker image from the WSO2 public docker registry: **wso2/wso2mi**.  Note that the image value format should be 'docker\_user\_name/repository\_name'. |
| Base Image Tag | Give a tag name for the base Docker image. |
| Target Image Repository | The Docker repository to which the Docker image will be pushed: 'docker\_user\_name/repository\_name'. |
| Target Image Tag | Give a tag name for the Docker image. |

* 1. Click **Finish**.

Your integration project with the **content routing** sample is now ready to be deployed in Kubernetes.



Step 2 - Build and Push the Docker image

**Note**

Be sure to start your Docker instance before building the image. If Docker is not started, the build process will fail.

There are two ways to build a Docker image of the integration solution and push it to your Docker registry:

* Using **Maven**:

**Before you begin**

You need **Maven 3.5.2** or a later version when you build the Docker image manually (without using WSO2 Integration Studio).

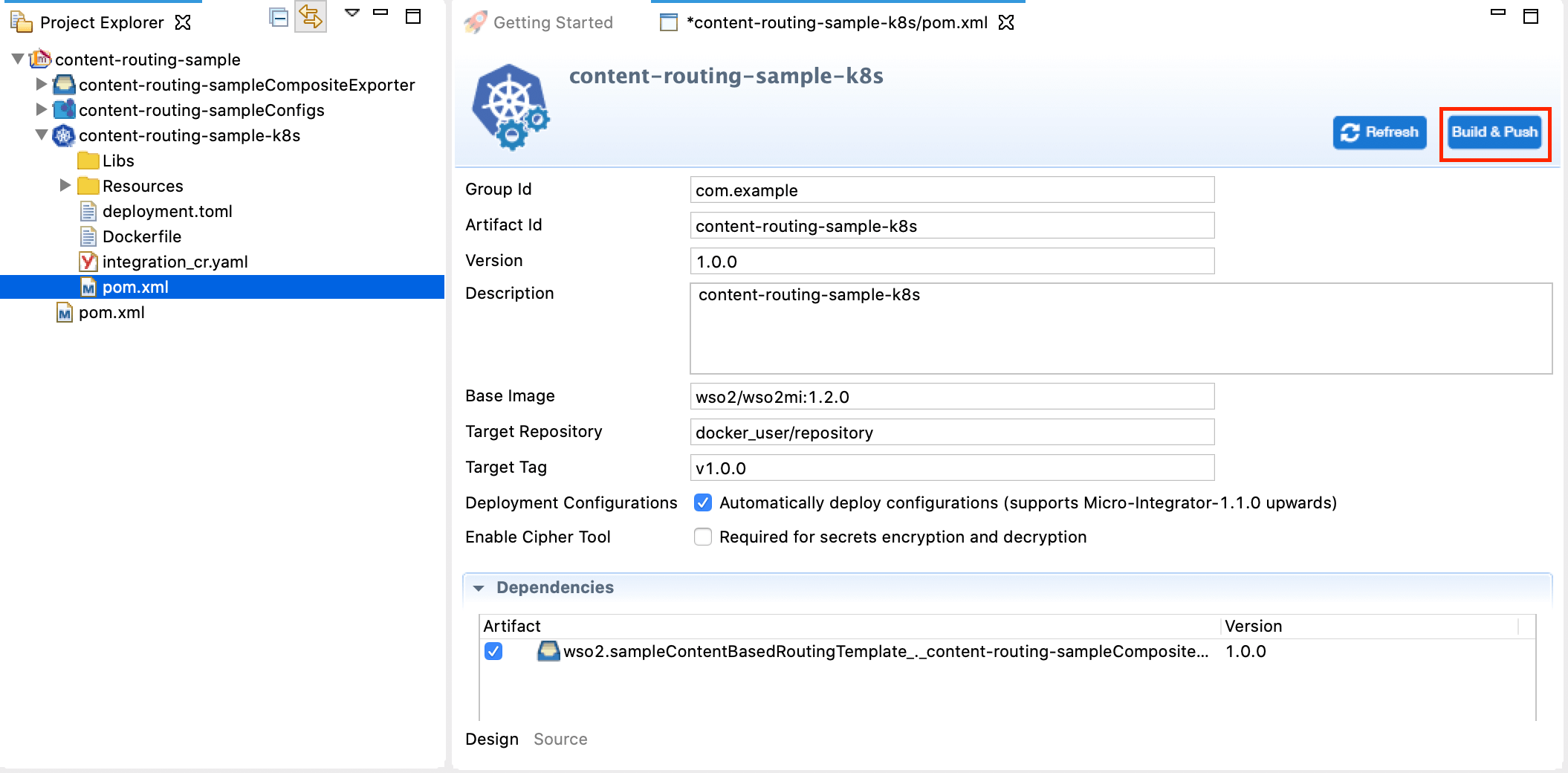
* 1. Open a terminal and navigate to the integration project.
  2. Execute the following command.

Be sure to specify the user name and password of the correct Docker registry.

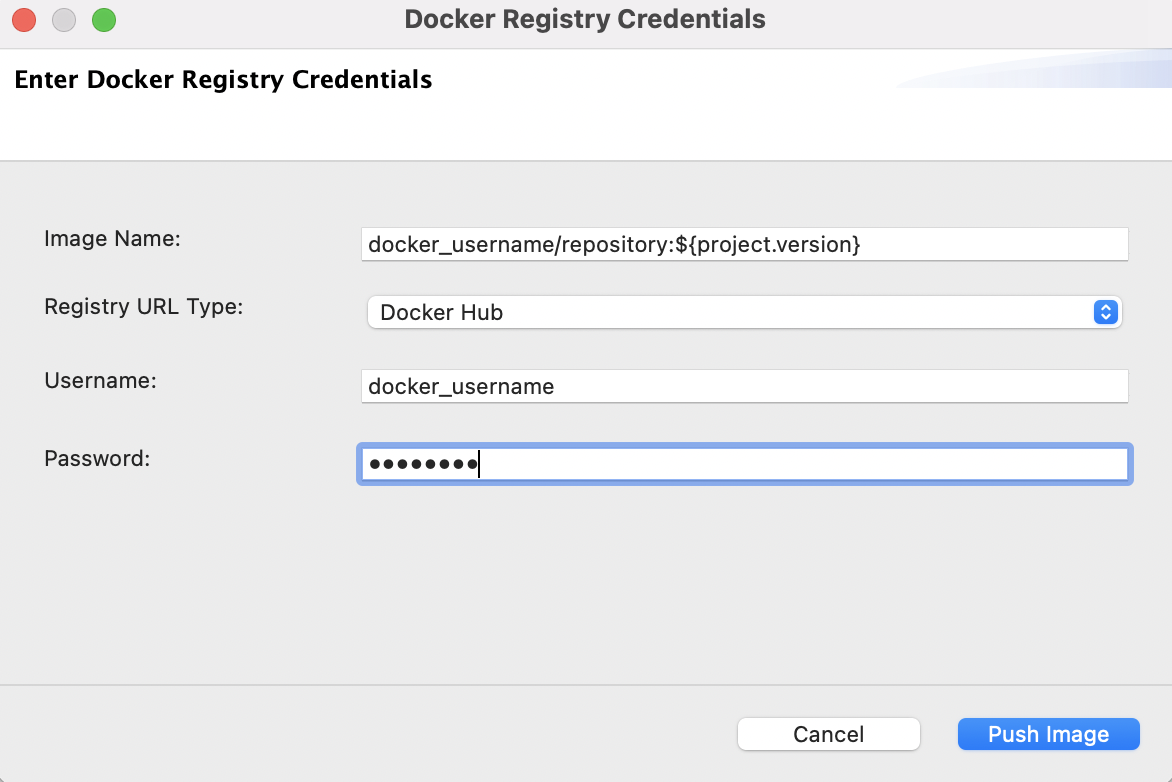
mvn clean install -Dmaven.test.skip=true -Ddockerfile.username={username} -Ddockerfile.password={password}

This will build the Docker image and then push it to the specified Docker registry.

* Using **WSO2 Integration Studio**:
  1. Open the **pom.xml** file in the Kubernetes exporter.
  2. Ensure that the composite exporter is selected under **Dependencies** and click **Build & Push**.



* 1. In the dialog box that opens, enter the credentials of your Docker registry to which the image should be pushed.



* 1. Click **Push Image**.

Run the docker image ls command to verify that the Docker image is created.

Step 3 - Deploy the solution in K8s

**Info**

**Before you begin**, the [API Kubernetes Operator](https://apim.docs.wso2.com/en/4.0.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/install) should be installed in your Kubernetes environment.

Follow the steps given below:

1. Open the integration\_cr.yaml file from the Kubernetes project in WSO2 Integration Studio.
2. See that the **integration** details of the content-routing solution are updated. **Be sure** to add the image name in the following format: docker\_user/repository:tag

apiVersion: "wso2.com/v1alpha2"

kind: "Integration"

metadata:

name: "content-routing"

spec:

image: "<Docker image for the Content-Based Routing Scenario>"

deploySpec:

minReplicas: 1

1. Open a terminal and start the Kubernetes cluster.
2. Navigate to the location of your integration\_cr.yaml file, and execute the following command to deploy the integration solution in the Kubernetes cluster:

kubectl apply -f integration\_cr.yaml

When the integration is successfully deployed, it should create the content-routing integration, content-routing-deployment, content-routing-service, and ei-operator-ingress as follows:

**Tip**

The api-operator-ingress is not created if you have [disabled the ingress controller](https://apim.docs.wso2.com/en/4.0.0/install-and-setup/setup/kubernetes-operators/k8s-api-operator/manage-integrations/integration-deployments/#disable-ingress-controller-creation).

kubectl get integration

NAME STATUS SERVICE-NAME AGE

content-routing 40s

kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

content-routing-deployment 1/1 1 1 2m

kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

content-routing-service ClusterIP 10.101.107.154 <none> 8290/TCP 2m

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 2d

k8s-api-operator ClusterIP 10.98.78.238 <none> 443/TCP 1d

kubectl get ingress

NAME HOSTS ADDRESS PORTS AGE

api-operator-ingress wso2ei.ingress.wso2.com 10.0.2.15 80, 443 2m

Step 4 - Test the deployment

Let's invoke the service without going through the ingress controller.

1. Create a request.xml file as follows:

<**ArithmaticOperation**>

<**Operation**>Add</**Operation**>

<**Arg1**>10</**Arg1**>

<**Arg2**>25</**Arg2**>

</**ArithmaticOperation**>

or

<**ArithmaticOperation**>

<**Operation**>Divide</**Operation**>

<**Arg1**>25</**Arg1**>

<**Arg2**>5</**Arg2**>

</**ArithmaticOperation**>

1. Apply port forwarding as shown below. This will allow you to invoke the service without going through the Ingress controller:

kubectl port-forward service/content-routing-service 8290:8290

1. Execute the following command to invoke the service:

curl -X POST -d @request.xml http://localhost:8290/ArithmaticOperationService -H "Content-Type: text/xml"

You will receive the following SOAP response:

<?xml version="1.0" encoding="UTF-8" ?>

<**SOAP-ENV:Envelope** xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:s='http://www.w3.org/2001/XMLSchema'>

<**SOAP-ENV:Body**><**AddIntegerResponse** xmlns="http://tempuri.org"><**AddIntegerResult**>35</**AddIntegerResult**></**AddIntegerResponse**></**SOAP-ENV:Body**>

</**SOAP-ENV:Envelope**>