

**In this article I will describe how to use Azure DevOps to:**

- build a docker image of your application,
- push this image to Azure Container Register,
- release your kubernetes deployment to Azure Kubernetes Service.

If you don't have any basic knowledge about docker, kubernetes, and Azure CLI please check out my previous article [Deploying .NET Core Application to Azure Kubernetes Cluster](#).

## **The Benefits of Automating Continuous Integration and Delivery Processes**

Automating CI/CD processes allows you to:

- save a lot of time
- eliminate bugs that happen when you do the same job over and over again
- deliver your product much faster

Azure DevOps can help you with that.

### **Azure DevOps**

Azure DevOps can automate your Continuous Integration and Delivery processes. It can get access to your git repository (Azure Repos Git, GitHub, and other git repositories). It can automatically react to your activity in your repository:

- run tests when you create a pull request
- build a docker image when you merge your pull request to a selected branch and push it to Azure Container Register (ACR)
- when everything is ok, it can apply changes to your Azure Kubernetes Service (AKS)

You can use docker images from your ACR to create as many release configurations as you need. For example, one for dev, test, stage and production environment and decide when you want to release them.

If you don't have your Azure Container Register and Azure Kubernetes Service, you can use necessary Azure CLI commands from my previous article:

### **Azure setup**

```
#Create temp variables:
$projectName="shkube"
$argName=$projectName+"RG"
$acrName=$projectName+"ACR"
$aksName=$projectName+"AKS"
$region="northeurope"
```

We are going to use them to create ACR and AKS in Azure Resource Group (ARG).

Login to azure:

```
az login
```

And lets create everyting you will need later:

```
# Create resource group
az group create -n $argName -l $region

# Create azure container register
az acr create -n $acrName -g $argName --sku standard

# Create azure kubernetes service
az aks create -n $aksName -g $argName --generate-ssh-keys --node-count 1 --node-vm-size Standard_B2s --enable-addons monitoring

# Get AKS Client Id and AKS Id
$CLIENT_ID=$(az aks show -g $argName -n $aksName --query "servicePrincipalProfile.clientId" --output tsv)
$ACR_ID=$(az acr show --name $acrName --resource-group $argName --query "id" --output tsv)

# Give AKS access to ACR
az role assignment create --assignee $CLIENT_ID --role acrpull --scope $ACR_ID

# Get credential to your AKS
az aks get-credentials -g $argName -n $aksName
```

Now we can begin to work with your CI/CD.

## Prerequisites

If you are reading this article, then you probably already have your application up and running inside Docker Container or even Kubernetes. If not, then you need three additional files to do that:

- Dockerfile
- docker-compose.yml
- deployment.yml

If you don't have them you can use the ones from this project.

# Azure DevOps

Go to your [Azure DevOps website](#) and create a new project.


## Create new project ✕

Project name \*

SHKube ✓


Description

Visibility



Public

Anyone on the internet can view the project. Certain features like TFVC are not supported.



Private

Only people you give access to will be able to view this project.

Create a new pipeline:

I'm going to use a classic editor and select Github repository. Pick your repository and a branch that you are going to use to build a container **from this project**.

Select a source

If you see Docker Compose on a list, you can use it, but I will pick **Empty job**.

## Select a template

Or start with an [Empty job](#)

 Search

### Configuration as code



#### YAML

Looking for a better experience to configure your pipelines using YAML files? Try the new YAML pipeline creation experience. [Learn more](#)

### Featured



#### .NET Desktop

Build and test a .NET or Windows classic desktop solution.



#### Android

Build, test, sign, and align an Android APK.



#### ASP.NET

Build and test an ASP.NET web application.



#### Azure Web App for ASP.NET

Build, package, test, and deploy an ASP.NET Azure Web App.




#### Docker container

Build a Docker image and push it to a container registry.

Use + button to add the first step of your pipeline! In this step we build a new docker image with your application. We use **Docker Compose** step that will use your **docker-compose.yml** file.

Add tasks

 Refresh

 **docker compose**



#### Docker Compose

Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container registry.



#### Service Fabric Compose deploy

Deploy a Docker Compose application to an Azure Service Fabric cluster

Marketplace 



#### Docker build task

Adds a build task that enables Docker actions.

Now there are a few things to do:

- select your Azure Subscription
- select Azure Container Register

- put a path to your `docker-compose.yml` file
- change Action to `Build service image`

and put `$(Build.BuildId)` as Additional Image Tags (without this, we won't be able to determine which image version to deploy later).

Display name \*

`Build service`

Container Registry Type \* ⓘ

Azure Container Registry

Azure subscription ⓘ | [Manage](#) ⌵

Visual Studio Enterprise — MPN (15d3cb86-4161-46a4-adf6-f123b1ff28ec)

ⓘ Scoped to subscription "Visual Studio Enterprise — MPN"

Azure Container Registry ⓘ

`shkubaACR`

Docker Compose File \* ⓘ

`docker-compose.yml`

Additional Docker Compose Files ⓘ

Environment Variables ⓘ

Project Name ⓘ

`$(Build.Repository.Name)`

☒ Qualify Image Names ⓘ

Action \* ⓘ

`Build service images`

Additional Image Tags ⓘ

`$(Build.BuildId)`

Great! Now we have to push this image to our ACR. Let's add second `Docker Compose` step. The only difference is the Action field: now pick `Push service image`.

Display name \*

Push services

Container Registry Type \* ⓘ

Azure Container Registry

Azure subscription ⓘ | [Manage](#) ↗

Azure Container Registry ⓘ

Docker Compose File \* ⓘ

docker-compose.yml

Additional Docker Compose Files ⓘ

Environment Variables ⓘ

Project Name ⓘ

\$(Build.Repository.Name)

☒ Qualify Image Names ⓘ

Action \* ⓘ

Push service images

The third step is optional but recommended. Lock an image version or a repository so that it can't be deleted or updated.

As before, add **Docker Compose** step. The only difference is the Action field. Now pick **Lock image service**. **Output Docker Compose File** will fill automatically.

Display name \*

Container Registry Type \* ⓘ

Azure subscription ⓘ | [Manage](#) ⓘ  
 ⓘ

ⓘ Scoped to subscription 'Visual Studio Enterprise — MPN'

Azure Container Registry ⓘ

Docker Compose File \* ⓘ

Additional Docker Compose Files ⓘ

Environment Variables ⓘ

Project Name ⓘ

☒ Qualify Image Names ⓘ

Action \* ⓘ

☐ Remove Build Options ⓘ

Base Resolve Directory ⓘ

Output Docker Compose File \* ⓘ

There are two more steps to follow.

Add **Copy Files** step.

- In **Contents** put a name or a path to your **deployment.yml** file (we will use this file during Release)
- In **Target folder** put **\$(Build.ArtifactStagingDirectory)**



Copy files ⓘ

Task version 2.\* ▼

Display name \*

Copy Files

Source Folder ⓘ

Contents \* ⓘ

deployment.yml

Target Folder \* ⓘ

\$(Build.ArtifactStagingDirectory)

Advanced ▼

Control Options ▼

Output Variables ▼

And **Publish build artifacts** step. Leave it as it is.

Publish build artifacts ⓘ

Task version 1.\* ▼

Display name \*

Publish Artifact: drop

Path to publish \* ⓘ

\$(Build.ArtifactStagingDirectory)

Artifact name \* ⓘ

drop

Artifact publish location \* ⓘ

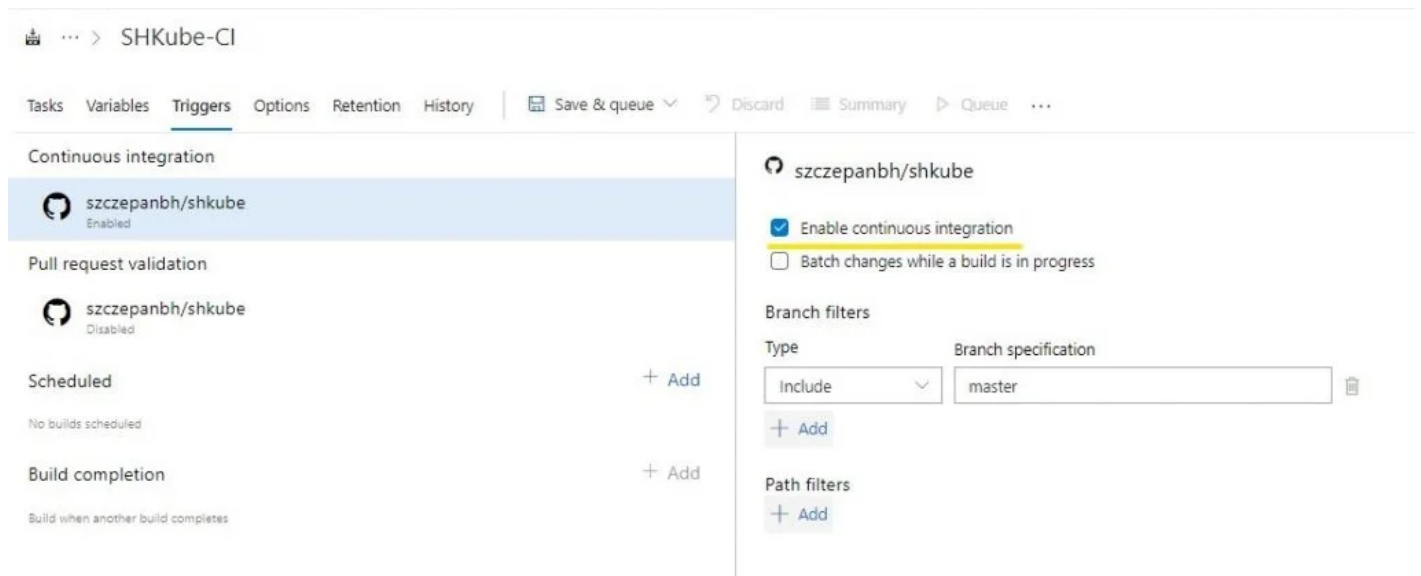
Azure Pipelines

Advanced ▼

Control Options ▼

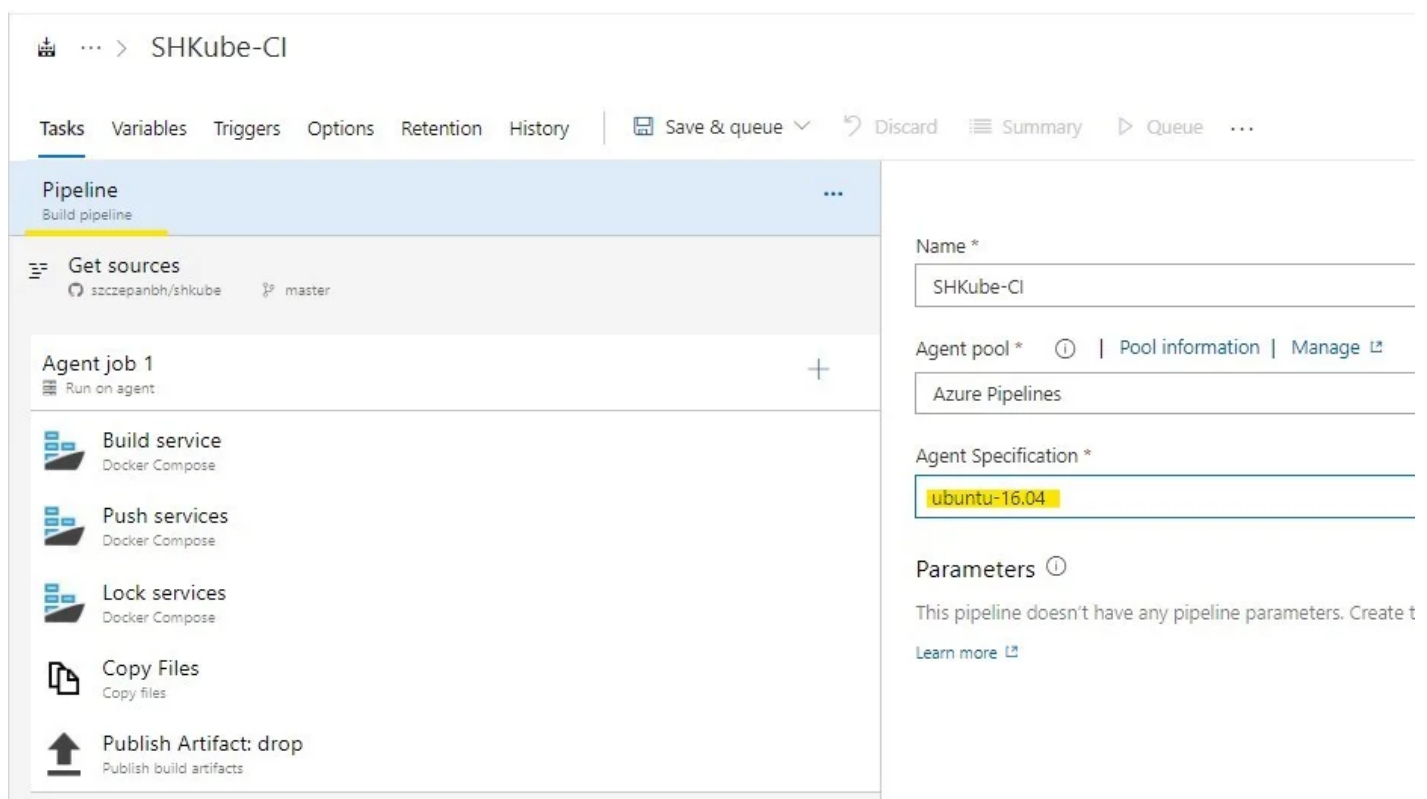
Output Variables ▼

We will use **Artifact name** during Release. If you want your pipeline to trigger automatically after each merge, go to **Triggers** tab and select **Enable continuous integration**.



The screenshot shows the 'Triggers' tab for a pipeline named 'SHKube-CI'. The left sidebar lists trigger categories: 'Continuous integration' (with a sub-item 'szczepanbh/shkubc' marked 'Enabled'), 'Pull request validation' (with a sub-item 'szczepanbh/shkubc' marked 'Disabled'), 'Scheduled' (with '+ Add' and 'No builds scheduled'), and 'Build completion' (with '+ Add' and 'Build when another build completes'). The main panel on the right is for 'Continuous integration' and shows 'Enable continuous integration' checked, 'Batch changes while a build is in progress' unchecked, and a 'Branch filters' section with 'Type' set to 'Include' and 'Branch specification' set to 'master'. There is also a 'Path filters' section with '+ Add'.

Click on **Build pipeline** and change **Agent Specification** to use Ubuntu (if you prefer Windows then you will have to change all paths to match Windows).



The screenshot shows the 'Build pipeline' tab for the 'SHKube-CI' pipeline. The left sidebar shows the pipeline steps: 'Get sources' (with 'szczepanbh/shkubc' and 'master'), 'Agent job 1' (with 'Run on agent'), 'Build service' (with 'Docker Compose'), 'Push services' (with 'Docker Compose'), 'Lock services' (with 'Docker Compose'), 'Copy Files' (with 'Copy files'), and 'Publish Artifact: drop' (with 'Publish build artifacts'). The main panel on the right shows the 'Name' field set to 'SHKube-CI', the 'Agent pool' set to 'Azure Pipelines', and the 'Agent Specification' set to 'ubuntu-16.04'. There is also a 'Parameters' section with a note that the pipeline doesn't have any parameters.

Now we can test our pipeline! Hit **Save & queue**.




The screenshot shows the 'Triggers' tab for the 'SHKube-CI' pipeline. The 'Save & queue' button is highlighted in yellow in the top right corner of the main panel.

Go to Pipelines, select the pipeline you have created and pick the newest Run. If it's green, everything is ok.

## ✓ #184 Merge pull request #1 from szczepanbh/sb\_deployment

on SHKube-CI ↗ Retained

Summary Code Coverage

Manually run by  Szczepan Błaszkiwicz

Repository and version

🔗 szczepanbh/shkuba

📁 master 📌 ba4a169

Time started and elapsed

📅 Today at 22:18

🕒 1m 41s

### Jobs

Name

✓ Agent job 1

If not, then you have to check which step went wrong and fix it.

← Jobs in run #184

SHKube-CI

Jobs

✓	Agent job 1	1m 37s
✓	Initialize job	2s
✓	Checkout szczepanbh/shkuba@m...	2s
✓	Build service	1m 15s
✓	Push services	14s
✓	Lock services	2s
✓	Copy Files	<1s
✓	Publish Artifact: drop	<1s
✓	Post-job: Checkout szczepanbh/...	<1s
✓	Finalize Job	<1s

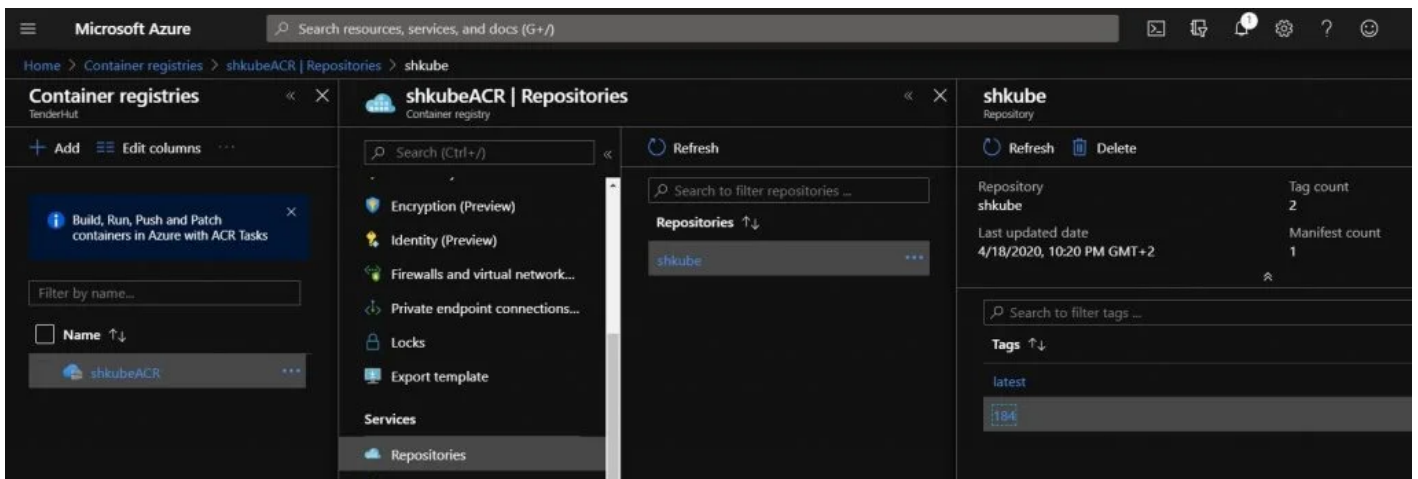
✓ Agent job 1

```

1 Pool: Azure Pipelines
2 Image: ubuntu-16.04
3 Agent: Hosted Agent
4 Started: Today at 22:18
5 Duration: 1m 37s
6
7 ▶ Job preparation parameters
8 ▶ ✖ 1 queue time variable used
9 📦 1 artifact produced

```

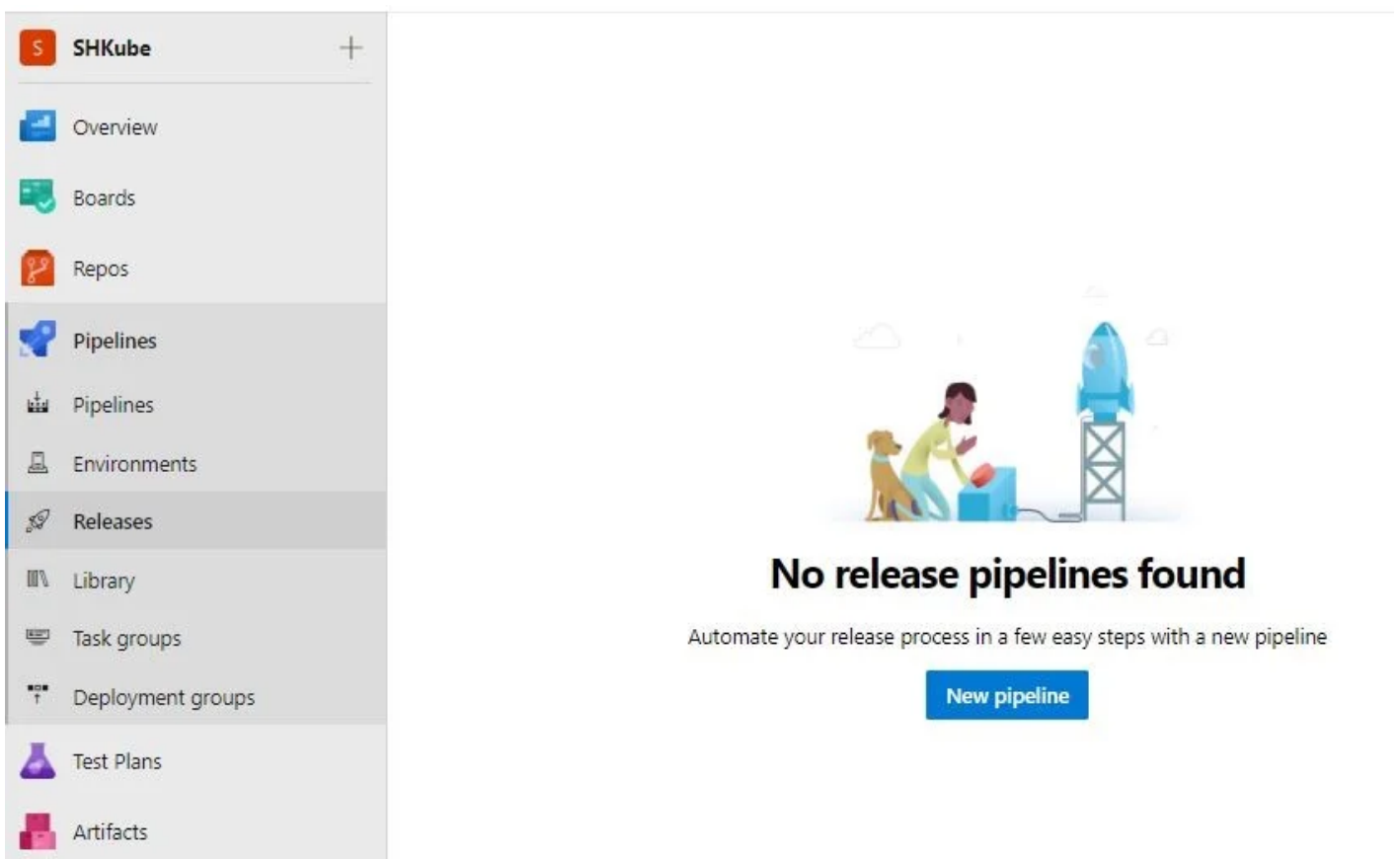
Congratulations! We're halfway through. We have an image and it's available on your ACR. Let's check it out. Please login to your Azure Portal and check if it's there (please notice that it's also tagged by its build number).



Now we can configure new Release.

## Azure Releases

Go to **Release** tab and create a new one.



Similarly to pipeline configuration, I have select **Empty job** . Now click on **Add an artifact** , select your project and source.

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

Stages | + Add

Stage 1  
1 job, 0 task

Add an artifact

Schedule not set

Add an artifact

Source type

Build Azure Repos ... GitHub TFVC

5 more artifact types

Project \* SHKube

Source (build pipeline) \* SHKube-CI

Default version \* Latest

Source alias \* \_SHKube-CI

The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **SHKube-CI** published the following artifacts: **drop**.

Add

Go to **Tasks** tab add **Kubectl** task.

- change Service connection type,
- select Azure subscription,
- select resource group,
- select Kubernetes cluster,
- pick Apply Command,
- check Use configuration.

Display name \*

Create Deployments & Services in AKS

Kubernetes Cluster ^

Service connection type \* ⓘ

Azure Resource Manager

Azure subscription \* ⓘ | Manage ↗

Visual Studio Enterprise — MPN (15d3cb86-4161-46a4-adf6-f123b1ff28ec)

ⓘ Scoped to subscription 'Visual Studio Enterprise — MPN'

Resource group \* ⓘ

shkubeRG

Kubernetes cluster \* ⓘ

shkubeAKS

☐ Use cluster admin credentials ⓘ

Namespace ⓘ

Commands ^

Command ⓘ

apply

☒ Use configuration ⓘ

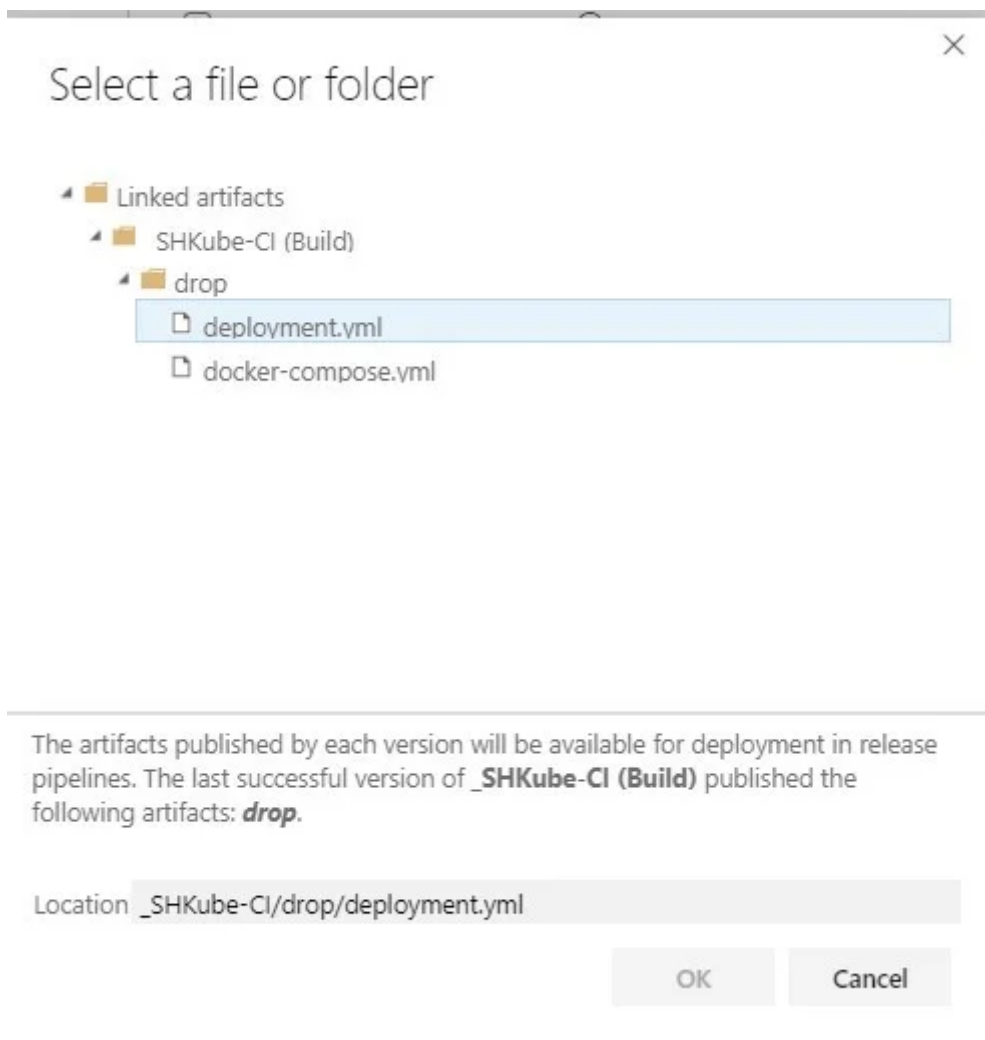
Configuration type ⓘ

☒ File path ☐ Inline configuration

File path \* ⓘ

\$(System.DefaultWorkingDirectory)/\_SHKube-CI/drop/deployment.yml

And pick your `deployment.yml` file in File path.



Expand **Advance** and choose **Version spec** that matches the version of Kubernetes on Azure. You can use this command to check the current one:

```
az aks show -g $argName -n $aksName --output table
```

Advanced ^

Kubectl ⓘ

☒ Version ☐ Specify location

Version spec ⓘ

1.13.2

☐ Check for latest version ⓘ

Working directory ⓘ

\$(System.DefaultWorkingDirectory)

Output format ⓘ

json

Add new `Kubectl` step (we will have to replace the name of your docker image to match the one on ACR).

Set everything as before but change `Command` to `set`, and `Arguments` input to `image deployments/shkube-deployment shkube=shkubeacr.azurecr.io/shkube:$(Build.BuildId)`.

This command is tricky, we have to navigate by name to deployment, and then to container image.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: shkube-deployment
5  labels:
6    app: shkube
7  spec:
8    replicas: 1
9    template:
10     metadata:
11       name: shkube
12       labels:
13         app: shkube
14     spec:
15       containers:
16       - name: shkube
17         image: shkube
18         imagePullPolicy: IfNotPresent
19         restartPolicy: Always
20     selector:
21       matchLabels:
22         app: shkube
23
24 ---
25
```

We specify the image version here. It's a better practice than just using latest because the latest image always points to the newest one (created after each merge). It could be ok for your dev environment but not for the production one.



Display name \*

Update image

Kubernetes Cluster ^

Service connection type \*



Azure Resource Manager

Azure subscription \*



| [Manage](#)

Visual Studio Enterprise — MPN (15d3cb86-4161-46a4-adf6-f123b1ff28ec)

Scoped to subscription 'Visual Studio Enterprise — MPN'

Resource group \*



shkubeRG

Kubernetes cluster \*



shkubeAKS



Use cluster admin credentials



Namespace



Commands ^

Command



set



Use configuration



Arguments



image deployments/shkube-deployment shkube=shkubeacr.azurecr.io/shkube:\$(Build.BuildId)

Click **Save** and **Create release** (top right) and go to Release to check if everything's alright.

↑ New release pipeline > Release-1 ▾

Pipeline Variables History | + Deploy ▾ ⏸ Cancel ↻ Refresh ✎ Edit ▾ ...

### Release

**Manually triggered**  
by Szczepan Błaszkiwi...  
18.04.2020, 23:15

---

Artifacts

\_SHKube-CI  
184  
 master

### Stages

**Stage 1**

**Succeeded**

on 18.04.2020, 23:15

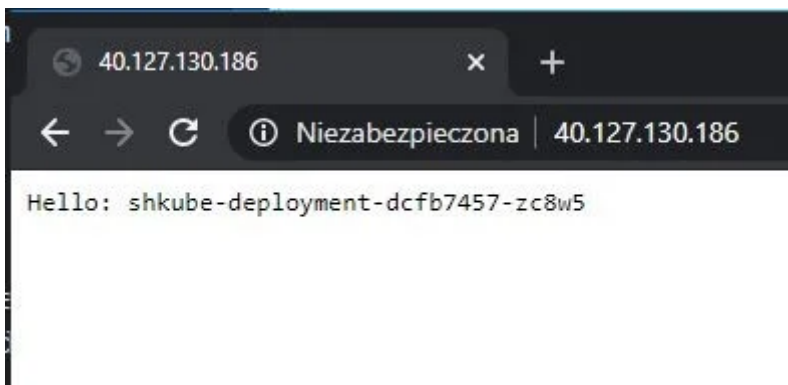
Ok, let's go to console and check your services and pods.

```

szczepq@szczepan-dell /c/Users/szcze/source/repos/blog/SHKube /sb deployment ● ? kubectl get svc
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP     10.0.0.1      <none>         443/TCP          24h
shkube-service LoadBalancer  10.0.212.55   40.127.130.186 80:31779/TCP     14m
szczepq@szczepan-dell /c/Users/szcze/source/repos/blog/SHKube /sb deployment ● ? kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
shkube-deployment-dcfb7457-zc8w5    1/1     Running   0           39s

```

And navigate to IP of your service to check if it's up and running.



It's working! Congratulations!

Your Continuous Integration and Delivery setup is ready. You can now use your pipeline to create another Release configuration for your test, stage, or production environment.