



# Create and set up a Helm chart for deployment

10 minutes

Let's set up the environment to deploy a Helm chart to AKS by using GitHub Actions.

In this exercise, to deploy a Helm chart, we'll complete the following tasks:

- Check the Helm installation
- Create a chart
- Configure the chart
- Create a deployment
- Create an empty YAML file
- Add contents to the YAML file
- Create a service
- Create an ingress
- Create a DNS zone name

## Check the Helm installation

1. In Cloud Shell, the Helm CLI is already installed. Sign in to Azure Cloud Shell by using the account you want to deploy resources to.

Azure Cloud Shell

### 📘 Important

We'll run all the scripts with Bash. If you haven't created a Cloud Shell yet, select **Bash** as the running shell.

2. Run `helm version` and check whether the displayed version is greater than 3.
3. To pull the changes you've made to the CI workflow in the preceding units, run this command:


Bash

📋 Copy

```
git pull origin main
```


## Create a chart

1. Run `cd` to go to the repository you forked, and then run `cd` to go to the **kubernetes** directory.
2. Run this command:

Bash	 Copy
<pre>helm create contoso-website</pre>	


The command creates a new directory called **contoso-website** in the **kubernetes** directory.

3. Run `cd` to go to the new directory. Delete the **charts** and **templates** folders in that directory.
4. Run the following command to create a new empty **templates** folder:

Bash	 Copy
<pre>mkdir templates</pre>	

You've created an empty chart. To start building the workloads, you'll use what others have already built. You'll use YAML files that currently aren't in the new directory you created.

5. Move the old **kubernetes** files to the **templates** folder.
6. From inside the **contoso-website** directory, run this command:


Bash	 Copy
<pre>mv ../*.yaml ./templates</pre>	

Completing these steps is all it takes to create a chart. Now, let's configure the chart.


## Configure the chart

1. Run `cd ..` to return to the parent directory. You should be at the root of the repository now.
2. To open the editor in the current directory, run `code .`
3. Open the **chart.yaml** file.

Chart.yaml is the file that names the chart. This file is where Helm looks for information about the chart itself. You should have a file that looks like this example:

YAML	 Copy
<pre>apiVersion: v2 name: contoso-website description: A Helm chart for Kubernetes  # A chart can be either an application or a library chart. # # Application charts are a collection of templates that can be packaged # into versioned archives to be deployed. # # Library charts provide useful utilities or functions for the chart devel- # oper. They're included as # a dependency of application charts to inject those utilities and func- # tions into the rendering # pipeline. Library charts do not define any templates and therefore cannot # be deployed. type: application  # This is the chart version. This version number should be incremented each # time you make changes # to the chart and its templates, including the app version. # Versions are expected to follow semantic versioning (https://semver.org # /). version: 0.1.0  # This is the version number of the application that's being deployed. This # version number should be # incremented each time you make changes to the application. Versions are # not expected to # follow semantic versioning. They should reflect the version the applica- # tion is using. appVersion: 1.16.0</pre>	

Remove all the comments and unused keys. Leave only the required options, and edit them to look like this example:

YAML	 Copy
<pre>apiVersion: v2</pre>	


```
name: contoso-website
description: Chart for the Contoso company website
version: 0.1.0
```

4. Save and close the file.

## Create a deployment


1. In the left menu, go to the **kubernetes** folder. Find the **deployment.yaml** file in the templates folder.

The file should look like this example:

YAML	 Copy
<pre>apiVersion: apps/v1 kind: Deployment metadata:   name: contoso-website spec:   selector:     matchLabels:       app: contoso-website   template:     metadata:       labels:         app: contoso-website     spec:       containers:         - image: !IMAGE!           name: contoso-website           resources:             requests:               cpu: 100m               memory: 128Mi             limits:               cpu: 250m               memory: 256Mi           ports:             - containerPort: 80               name: http</pre>	

Next, we add templating for this deployment, beginning with the namespace and name keys.


2. In the `metadata` section, add a new key called `namespace`. The key should have the following configuration:

YAML	 Copy
<pre>apiVersion: apps/v1 kind: Deployment metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }} spec:   selector:     matchLabels:       app: contoso-website   template:     metadata:       labels:         app: contoso-website     spec:       containers:         - image: !IMAGE!           name: contoso-website           resources:             requests:               cpu: 100m               memory: 128Mi             limits:               cpu: 250m               memory: 256Mi       ports:         - containerPort: 80           name: http</pre>	

By default, you deploy this resource to the `staging` namespace. But, if the installation has a `namespace` option, use that instead.

### 3. Go to the `image` key.

It's a good practice to split up the `registry`, `tag`, and `image` parts of the image name. Add three new template variables to this section of the file:

YAML	 Copy
<pre>apiVersion: apps/v1 kind: Deployment metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }} spec:   selector:     matchLabels:       app: contoso-website   template:</pre>	

```
metadata:
  labels:
    app: contoso-website
spec:
  containers:
    - image: {{ .Values.image.registry }}/{{ .Values.image.name }}:{{
default "latest" .Values.image.tag }}
      name: contoso-website
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      ports:
        - containerPort: 80
          name: http
```

In this section, you split the three sections of the image so you can work with them more easily.

4. Save and close the file.

## Create an empty YAML file

1. In the root of the contoso-website directory, open the **values.yaml** file.
2. Delete all contents in the file, so you have an empty YAML file.

## Add contents to the YAML file

Now, let's add your content to the empty file.


You saw earlier that you used `{{ .Release.Namespace }}`, so `Release` is a *variable scope*. Each variable scope has different default values and variables.

Helm uses the `values.yaml` file to retrieve all the template values that start with `{{ .Values }}`. The `values.yaml` file is another variable scope.

This file should have the same structure of the file you use to call variables. Let's take a quick look in the `deployment.yaml` file you edited to see the structure.

Notice that you used `.Values.image.registry`, `.Values.image.name`, and `.Values.image.tag` in the `deployment.yaml` file.

1. Create the **values.yaml** file so that it looks like this example:


YAML	 Copy
<pre>image:   registry: &lt;your-acr-name&gt;   name: contoso-website   tag: latest</pre>	

These values are the *default* values if you don't pass a new value as a parameter by using the `--set` option of the Helm CLI.

2. Save and close the file.

## Create a service

1. Find and open the **service.yaml** file.
2. In the `metadata` section of the file, add a new key called `namespace`. Use the same value that you used in the `deployment.yaml` file.


YAML	 Copy
<pre>apiVersion: v1 kind: Service metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }} spec:   ports:     - port: 80       protocol: TCP       targetPort: http       name: http   selector:     app: contoso-website   type: ClusterIP</pre>	

3. Save and close the file.


## Create an ingress

1. Find and open the **ingress.yaml** file.
2. In the `metadata` section of the file, add a new key called `namespace`. Use the same

value that you used in the deployment.yaml file.


YAML	 Copy
<pre>apiVersion: v1 kind: Ingress metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }}   annotations:     kubernetes.io/ingress.class: addon-http-application-routing spec:   rules:     - host: contoso.!DNS!       http:         paths:           - backend:               serviceName: contoso-website               servicePort: http             path: /</pre>	

3. Go to the `host` key. You create separate hosts for staging and production deployments. (You don't want users to access the `staging` namespace by using production URLs.)
4. Concatenate the namespace in the host name. The HTTP application routing add-on in the AKS cluster handles name resolution.

YAML	 Copy
<pre>apiVersion: v1 kind: Ingress metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }}   annotations:     kubernetes.io/ingress.class: addon-http-application-routing spec:   rules:     - host: contoso-{{ default "staging" .Release.Namespace }}.!DNS!       http:         paths:           - backend:               serviceName: contoso-website               servicePort: http             path: /</pre>	

5. Add a new template variable, which will be your DNS zone name:




YAML	 Copy
<pre>apiVersion: v1 kind: Ingress metadata:   name: contoso-website   namespace: {{ default "staging" .Release.Namespace }}   annotations:     kubernetes.io/ingress.class: addon-http-application-routing spec:   rules:     - host: contoso-{{ default "staging" .Release.Namespace }}.{{ .Values.dns.name }}       http:         paths:           - backend:               serviceName: contoso-website               servicePort: http             path: /</pre>	



6. Save and close the file.

## Create a DNS zone name

1. Open the `values.yaml` file.
2. Add the `dns.name` key, so the file looks like this example:


YAML	 Copy
<pre>image:   repository: &lt;acr-name&gt;   name: contoso-website   tag: latest  dns:   name: &lt;your-dns-zone-name&gt;</pre>	

To get your DNS zone name, run this Azure CLI query:

Azure CLI	 Copy	 Try It
<pre>az aks show -g {resource-group-name} -n {aks-cluster-name} -o tsv --query addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName</pre>		

3. Save and close the file.

4. To push all the changes to the fork, run this command:

Bash	 Copy
<pre>git push -u origin main</pre>	

---

## Next unit: Create the deployment pipeline

Continue >