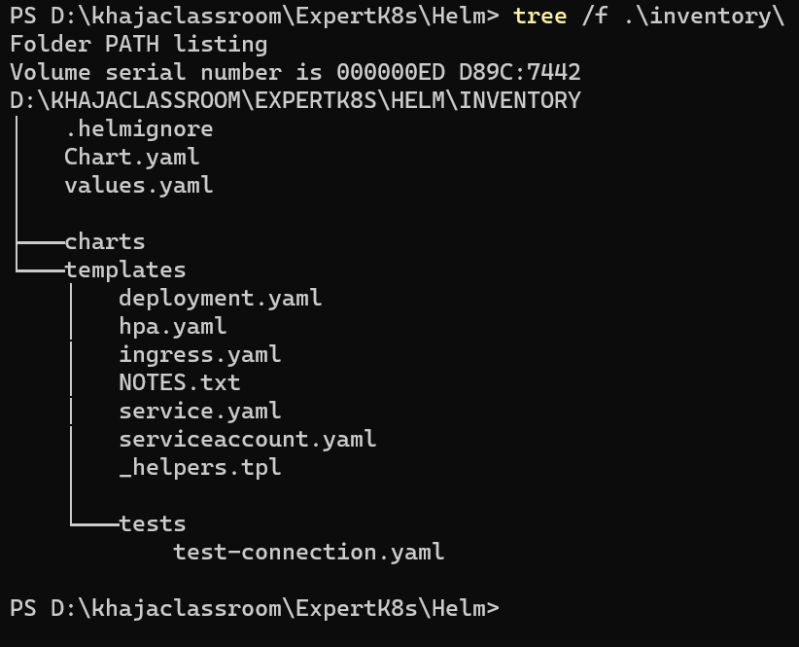
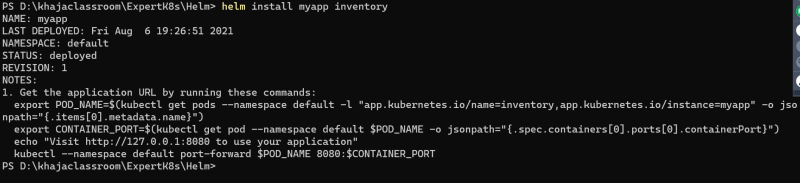
Helm Chart Creation Command

* Helm includes create command to make it easy for us to create charts
* This command creates a new **Nginx chart** with name of our choice 
* Lets try to install the chart which we have create 
* The output from the command is

NAME: myapp

LAST DEPLOYED: Fri Aug 6 19:26:51 2021

NAMESPACE: default

STATUS: deployed

REVISION: 1

helm install [NAME] [CHART]

NOTES:

1. Get the application URL by running these commands:

export POD\_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=inventory,app.kubernetes.io/instance=myapp" -o jsonpath="{.items[0].metadata.name}")

export CONTAINER\_PORT=$(kubectl get pod --namespace default $POD\_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")

echo "Visit http://127.0.0.1:8080 to use your application"

kubectl --namespace default port-forward $POD\_NAME 8080:$CONTAINER\_PORT

* Now refer the charts.yaml <https://helm.sh/docs/topics/charts/> for the offical docs
* In this charts.yaml lets try to focus on some name value pairs
  + apiVersion: v2: This tells helm what structure of chart we are using. An apiVersion of v2 is designed for Helm3
  + name: inventory: The name used to identify the chart
  + version: 0.1.0: Charts can have many versions. Helm uses the version information to order and identify charts
* Charts.yaml also contain descriptive information
  + home: URL of chart or projects
  + icon: an image in the form of URL
  + maintainers: contains list of maintainers
  + keywords: can hold list of keyworkds about the project
  + sources: list of URLs for the source code for project or chart
* Refer below for the sample chart.yaml



Link: <https://github.com/bitnami/charts/blob/master/bitnami/mysql/Chart.yaml>

Modifying Templates

* Helm is written in Go programming language and Go includes template packages. Helm leverages the text template package as the foundation for its templates Refer <https://pkg.go.dev/text/template>
* {{ and }} are the opening and closing brackets to enter and exit the template logic
* Sample

product: {{ .Values.product | default "kubernetes" | quote }}

* There are three parts to the template logic sepearted by a |. This is called as pipeline and works exactly in the sameway as pipeline in Unix/Linux Based system. The value or output of a function on the left is passed as a last argument to the next item in pipeline.
* .Values.product This comes for the data passed in when the templates are rendered
* This value is passed as last argument to the default function
* The default is the helm function and output of default is passed to the quoted
* The . at the start .Values.production is considered as root object in the current scope

Developing Templates

* Helm uses the Go text template engine provided as part of standard Go Libarary
* Actions:
  + Logic, control structures and data evaluations are wrapped by {{ and }}. These are called as actions.
  + Anything outside of actions is copied to output
  + When the curly braces are used to start and stop actions they can be accompanies by a – to remove leading or trailing white spaces.
* {{ "Hello" -}}, {{- "World" }}, {{- "of Helm" -}}
* # generated Output Hello,World,of Helm

Information Helm Passes to Templates

* When Helm renders a template it passes a single data object to the template with information you can access.
* Inside the template that object is representeed as . (i.e period)
* The properties on .values are specific to each chart based entirely on values in values.yaml
* What values should be present in values.yaml have no specific structure or schema.
* In addition to values, information about the release can be access as properties of .Release. This information includes
  + .Release.Name: name of the release
  + .Release.Namespace: Contains the namespace the chart is being released to
  + .Release.IsInstall: Set to true when relase is workload being installed
  + .Release.IsUpgrade: Set to true when the release is upgrade or rollback
  + .Release.Service: Lists the Service performing the release. when Helm installs a chart this value would be Helm
* The information in Chart.yaml can alos be found the data object at .Chart
  + .Chart.Name
  + .Chart.Version
  + .Chart.AppVersion
  + .Chart.Annotations
* Note the Names differ as names in Charts.yaml start with lowercase but Start with Uppercase later when they properties of .Chart object
* If you want to pass the custom information from the Chart.yaml to the template, you need to use annotations
* Helm also provides some data about the capabilities of the K8s cluster as properties of .Capabilities.
  + .Capabilities.ApiVersions: Contains the API Versions and resource types available in your cluster
  + .Capabilities.KubeVersion.Version: Full Kubernetes Version
  + .Capabilities.KubeVersion.Major: Contains major K8s version
  + .Capabilities.KubeVersion.Minor: The minor version of K8s being used in cluster
* The final piece of data passed into the template is details about the current template being executed.
  + .Template.Name: Contains the namespaced filepath to the template (inventory/templates/deployment.yaml)
  + .Template.BasePath: Contains the namespaced Path of Templates directory (inventory/templates)