Create Helm Chart From Scratch

To get hands-on with helm chart creation, let's **create an Nginx helm chart** from scratch.

Execute the following command to create the chart boilerplate. It creates a chart with the name `nginx-chart` with default files and folders.

```
helm create nginx-chart
```

If you check the created chart, it will have the following files and directories.

```
nginx-chart
│   ├── Chart.yaml
│   ├── charts
│   ├── templates
│   │   ├── NOTES.txt
│   │   ├── _helpers.tpl
│   │   ├── deployment.yaml
│   │   ├── hpa.yaml
│   │   ├── ingress.yaml
│   │   ├── service.yaml
│   │   ├── serviceaccount.yaml
│   │   └── tests
│   │       └── test-connection.yaml
│   └── values.yaml
```

Let's cd into the generated chart directory.

```
cd nginx-chart
```

We'll **edit the files one by one** according to our deployment requirements.

Chart.yaml

As mentioned above, we put the details of our chart in `Chart.yaml` file. Replace the default contents of `chart.yaml` with the following.

```
apiVersion: v2
name: nginx-chart
description: My First Helm Chart
type: application
version: 0.1.0
appVersion: "1.0.0"
maintainers:
- email: contact@devopscube.com
  name: devopscube
```

apiVersion: This denotes the chart API version. v2 is for Helm 3 and v1 is for previous versions.

name: Denotes the name of the chart.

description: Denotes the description of the helm chart.

Type: The chart type can be either '**application**' or '**library**'. Application charts are what you deploy on Kubernetes. Library charts are re-usable charts that can be used with other charts. A similar concept of libraries in programming.

Version: This denotes the chart version.

appVersion: This denotes the version number of our application (Nginx).

maintainers: Information about the owner of the chart.

We should increment the `version` and `appVersion` each time we make changes to the application. There are some other fields like dependencies, icons, etc.

templates

There are multiple files in `templates` directory created by Helm. In our case, we will work on simple Kubernetes Nginx deployment.

Let's remove all default files from the template directory.

```
rm -rf templates/*
```

We will add our Nginx YAML files and change them to the template for better understanding.

Create a `deployment.yaml` file and copy the following contents.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: "nginx:1.16.0"
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
```

If you see the above YAML file, the values are static. The idea of a helm chart is to template the YAML files so that we can reuse them in multiple environments by

dynamically assigning values to them.

To template a value, all you need to do is add the **object parameter** inside curly braces as shown below. It is called a **template directive** and the syntax is specific to the **Go templating**

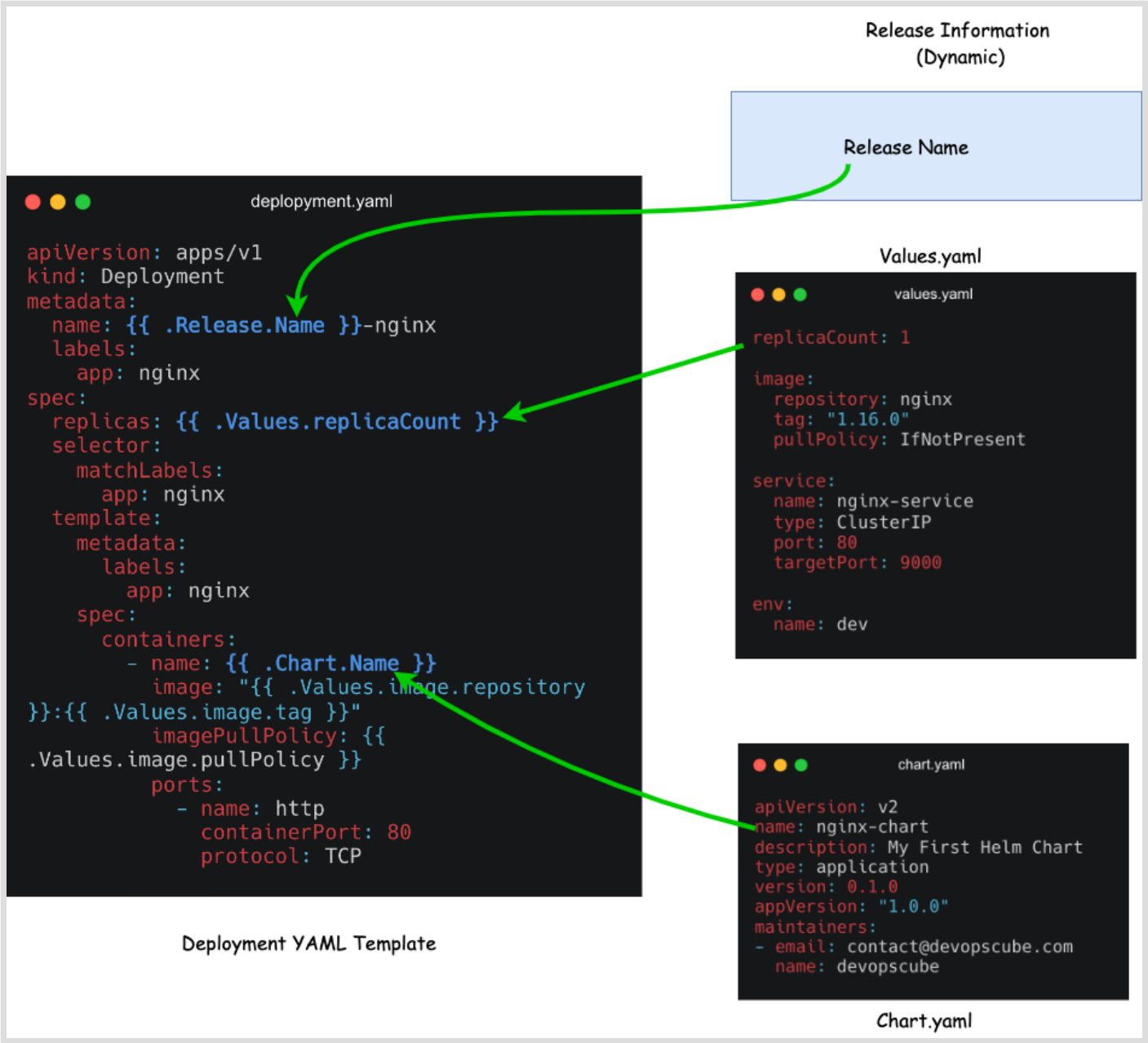
```
{{ .Object.Parameter }}
```

First Let's understand what is an Object. Following are the three Objects we are going to use in this example.

> **Release**: Every helm chart will be deployed with a release name. If you want to use the release name or access **release-related dynamic values** inside the template, you can use the release object.
>
> **Chart**: If you want to use any values you mentioned in the **chart.yaml**, you can use the chart object.
>
> **Values**: All parameters inside **values.yaml** file can be accessed using the Values object.

To know more about supported Objects check the Helm Builtin Object document.

The following image shows how the built-in objects are getting substituted inside a template.



First, you need to figure out what values could change or what you want to templatize. I am choosing **name**, **replicas, container name, image,** and **imagePullPolicy** which I have highlighted in the YAML file in bold.

> **name:** `name: {{ .Release.Name }}-nginx` : We need to change the deployment name every time as Helm does not allow us to install releases with the same name. So we will templatize the name of the deployment with the release name and interpolate **-nginx** along with it. Now if we create a release using the name **frontend**, the deployment name will be **frontend-nginx**. This way, we will have guaranteed unique names.

container name: `{{ .Chart.Name }}` : For the container name, we will use the Chart object and use the chart name from the **chart.yaml** as the container name.

Replicas: `{{ .Values.replicaCount }}` We will access the replica value from the **values.yaml** file.

image: `"{{ .Values.image.repository }}:{{ .Values.image.tag }}"` Here we are using multiple template directives in a single line and accessing the repository and tag information under the image key from the Values file.

Similarly, you can templatize the required values in the YAML file.

Here is our final `deployment.yaml` file after applying the templates. The templated part is highlighted in bold. Replace the deployment file contents with the following.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-nginx
  labels:
    app: nginx
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
```

Create `service.yaml` file and copy the following contents.

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-service
spec:
  selector:
    app.kubernetes.io/instance: {{ .Release.Name }}
  type: {{ .Values.service.type }}
  ports:
    - protocol: {{ .Values.service.protocol | default "TCP" }}
      port: {{ .Values.service.port }}
      targetPort: {{ .Values.service.targetPort }}
```

In the **protocol template directive**, you can see a pipe `( | )` . It is used to define the default value of the protocol as TCP. So that means we won't define the protocol value in `values.yaml` file or if it is empty, it will take TCP as a value for protocol.

Create a `configmap.yaml` and add the following contents to it. Here we are replacing the default Nginx **index.html** page with a custom HTML page. Also, we added a template directive to replace the environment name in HTML.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-index-html-configmap
  namespace: default
data:
  index.html: |
    <html>
    <h1>Welcome</h1>
    </br>
    <h1>Hi! I got deployed in {{ .Values.env.name }} Environment using Helm
Chart </h1>
    </html
```

values.yaml

The `values.yaml` file contains all the values that need to be substituted in the template directives we used in the templates. For example, `deployment.yaml` template contains a template directive to get the image repository, tag, and pullPolicy from the **`values.yaml`** file. If you check the following **`values.yaml`** file, we have repository, tag, and pullPolicy key-value pairs nested under the image key. That is the reason we used **`Values.image.repository`**

Now, replace the default **`values.yaml`** content with the following.

```
replicaCount: 2

image:
  repository: nginx
  tag: "1.16.0"
  pullPolicy: IfNotPresent

service:
  name: nginx-service
  type: ClusterIP
  port: 80
  targetPort: 9000

env:
  name: dev
```

Now we have the Nginx helm chart ready and the final helm chart structure looks like the following.

```
nginx-chart
├── Chart.yaml
├── charts
├── templates
│   ├── configmap.yaml
│   ├── deployment.yaml
│   └── service.yaml
└── values.yaml
```

Validate the Helm Chart

Now to make sure that our chart is valid and, all the indentations are fine, we can run the below command. Ensure you are inside the chart directory.

```
helm lint .
```

If you are executing it from outside the **`nginx-chart`** directory, provide the full path of **`nginx-chart`**

```
helm lint /path/to/nginx-chart
```

If there is no error or issue, it will show this result

```
==> Linting ./nginx
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

To validate if the values are getting substituted in the templates, you can render the templated YAML files with the values using the following command. It will generate and display all the manifest files with the substituted values.

```
helm template .
```

We can also use `--dry-run` command to check. This will pretend to install the chart to the cluster and if there is some issue it will show the error.

```
helm install --dry-run my-release nginx-chart
```

If everything is good, then you will see the manifest output that would get deployed into the cluster.

Deploy the Helm Chart

When you deploy the chart, Helm will read the chart and configuration values from the `values.yaml` file and generate the manifest files. Then it will send these files to the Kubernetes API server, and Kubernetes will create the requested resources in the cluster.

Now we are ready to install the chart.

Execute the following command where **`nginx-release`** is release name and **`nginx-chart`** is the chart name. It installs **`nginx-chart`** in the default namespace

```
helm install frontend nginx-chart
```

You will see the output as shown below.

```
NAME: frontend
LAST DEPLOYED: Tue Dec 13 10:15:56 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Now you can check the release list using this command:

```
helm list
```

Run the kubectl commands to check the deployment, services, and pods.

```
kubectl get deployment
kubectl get services
```

```
kubectl get configmap
kubectl get pods
```

We can see the deployment `frontend-nginx` , `nginx-service` and pods are up and running as shown below.

```
→  helm-tutorial git:(main) ✗ kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
frontend-nginx  2/2     2            2           30s
→  helm-tutorial git:(main) ✗ kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
kubernetes      ClusterIP   172.20.0.1       <none>        443/TCP
nginx-service   ClusterIP   172.20.119.119   <none>        80/TCP
→  helm-tutorial git:(main) ✗ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
frontend-nginx-74fd5b8d46-f2jjz   1/1     Running   0          47s
frontend-nginx-74fd5b8d46-rwblp   1/1     Running   0          47s
→  helm-tutorial git:(main) ✗
```

We discussed how a single helm chart can be used for multiple environments using different `values.yaml` files. To install a helm chart with an external `values.yaml` file, you can use the following command with the `--values` flag and path of the values file.

```
helm install frontend nginx-chart --values env/prod-values.yaml
```

When you have Helm as part of your CI/CD pipeline, you can write custom logic to pass the required values file depending on the environment.

Helm Upgrade & Rollback

Now suppose you want to modify the chart and install the updated version, we can use the below command:

```
helm upgrade frontend nginx-chart
```

For example, we have changed the replicas from 2 to 1. You can see the revision number is 2 and only 1 pod is running.

```
→  helm-tutorial git:(main) ✗ helm upgrade frontend nginx-chart
Release "frontend" has been upgraded. Happy Helming!
NAME: frontend
LAST DEPLOYED: Tue Dec 13 10:21:09 2022
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
→  helm-tutorial git:(main) ✗ helm list
NAME            NAMESPACE       REVISION        UPDATED
frontend        default         2               2022-12-13 10:21:09.01
→  helm-tutorial git:(main) ✗ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
frontend-nginx-74fd5b8d46-rwblp 1/1     Running   0          7m45s
→  helm-tutorial git:(main) ✗
```

Now if we want to roll back the changes that we have just done and deploy the previous one again, we can use the rollback command to do that.

```
helm rollback frontend
```

The above command will roll back the helm release to the previous one.

```
→  helm-tutorial git:(main) ✗ helm rollback frontend
Rollback was a success! Happy Helming!
→  helm-tutorial git:(main) ✗ kubectl get pods
NAME                            READY   STATUS    RESTARTS
frontend-nginx-74fd5b8d46-pbh2d 1/1     Running   0
frontend-nginx-74fd5b8d46-rwblp 1/1     Running   0
→  helm-tutorial git:(main) ✗
→  helm-tutorial git:(main) ✗
→  helm-tutorial git:(main) ✗ helm list
NAME            NAMESPACE       REVISION        UPDATED
frontend        default         3               2022-12-13 10
→  helm-tutorial git:(main) ✗
```

After the rollback, we can see 2 pods are running again. Note that Helm takes the rollback as a new revision, that’s why we’re getting the revision as 3.

If we want to roll back to the specific version we can put the revision number like this.

```
helm rollback <release-name> <revision-number>
```

For example,

```
helm rollback frontend 2
```

Uninstall The Helm Release

To uninstall the helm release use uninstall command. It will remove all of the resources associated with the last release of the chart.

```
helm uninstall frontend
```

We can package the chart and deploy it to Github, S3, or any other platform.

```
helm package frontend
```

Debugging Helm Charts

We can use the following commands to debug the helm charts and templates.

> **`helm lint:`** This command takes a path to a chart and runs a series of tests to verify that the chart is well-formed.

`helm get values:` This command will output the release values installed to the cluster.

`helm install --dry-run:` Using this function we can check all the resource manifests and ensure that all the templates are working fine.

`helm get manifest:` This command will output the manifests that are running in the cluster.

`helm diff:` It will output the differences between the two revisions.

```
helm diff revision nginx-chart 1 2
```

Helm Chart Possible Errors

If you try to install an existing Helm package, you will get the following error.

```
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
```

To update or upgrade the release, you need to run the upgrade command.

If you try to install a chart from a different location without giving the absolute path of the chart you will get the following error.

```
Error: non-absolute URLs should be in form of repo_name/path_to_chart
```

To rectify this, you should execute the helm command from the directory where you have the chart or provide the absolute path or relative path of the chart directory.