**Introduction**

Customize is a configuration management tool. It simplifies the declaration of Kubernetes manifest files by providing features such as patches, generators, transformers, and validators. It introduces an alternative to manifest template management solutions like Helm. Version **1.14** onwards Kustomize comes built into the kubectl command line tool as apply -k. It helps transverse a Kubernetes manifest to add, remove or update configuration options.

Kustomize enables you to make selective patches to a base manifest file to create new variants such as staging, development, production, and so on with different configuration options. It follows the same declarative approach as Kubernetes itself. The native integration with the kubectl command-line tool makes it easier for you to implement Kustomize on your Vultr Kubernetes Engine (VKE) cluster to build and manage application configurations.

This article walks you through the configuration of standard settings like namespace or name prefixes, the configuration of base and overlays to set up different variants, and the configuration of overlay patches. It also explains the installation of the kustomize individual binary that allows previewing plain manifests generated using Kustomize.

**Prerequisites**

* [Deploy a Vultr Ubuntu 20.04 instance](https://www.vultr.com/servers/ubuntu) to use as a management workstation.
* [Deploy a Kubernetes cluster at Vultr](https://www.vultr.com/docs/vultr-kubernetes-engine/#How_to_Deploy_a_VKE_Cluster).

On the management workstation:

* Install [Kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/).
* Download your VKE configuration and [configure Kubectl](https://www.vultr.com/docs/vultr-kubernetes-engine/#How_to_Manage_a_VKE_Cluster).

You must perform the rest of the steps in this guide from your management workstation.

**Set Up the Environment**

Kustomize comes built into the kubectl command line tool, but the integration between the two does not allow previewing the configuration. The individual kustomize binary allows building the configuration without applying it to the cluster, and you can use it to generate a plain Kubernetes manifest to preview the generated configuration.

Download the Kustomize installation script.

# wget https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install\_kustomize.sh

The above command downloads the installation script, which determines the operating system and the system architecture of the management workstation to download the latest pre-complied kustomize binary file.

Run the Kustomize installation script.

# bash install\_kustomize.sh

The above command downloads the kustomize binary file into the working directory.

Move the kustomize binary file to the /usr/local/bin directory.

# mv kustomize /usr/local/bin

The above command moves the kustomize binary file to the /usr/local/bin directory to be available globally.

Create and enter a new directory.

# mkdir /kustomize-demo

# cd /kustomize-demo

The above command creates and enters a directory named **kustomize-demo** in the /root directory.

Using a text editor, create a new manifest file named **deployment.yaml**.

# nano deployment.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx-kustomize

template:

metadata:

labels:

app: nginx-kustomize

spec:

containers:

- image: nginx

name: nginx

The above manifest file defines a basic Deployment resource using the nginx image.

Create a new manifest file named **service.yaml**.

# nano service.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

apiVersion: v1

kind: Service

metadata:

name: nginx-svc

spec:

selector:

app: nginx-kustomize

ports:

- protocol: TCP

port: 80

targetPort: 80

The above manifest file defines a basic Service resource which by default creates a ClusterIP service resource to expose the connection to pods matching with the defined selector to the cluster.

Create a new manifest file named **kustomization.yaml**.

# nano kustomization.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

resources:

- deployment.yaml

- service.yaml

The above manifest enables using the features of Kustomize, as demonstrated in the next sections. The **kustomization.yaml** file is manifest that specifies a list of resources, patches to apply, configMap and secret generators, and so on. You specify the manifest files under the resources attribute, which merge when you build the configuration.

Build and preview the configuration.

# kustomize build .

The above command outputs the merged configuration of the manifest files listed under the resources attribute.

Apply the configuration.

# kubectl apply -k .

The above command uses the -k flag, which uses the **kustomization.yaml** file in the given directory to generate a plain manifest and apply changes to the cluster.

Verify the deployment.

# kubectl get all

Output.

NAME READY STATUS RESTARTS AGE

pod/nginx-74cd6f44f6-l9brg 1/1 Running 0 30s

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

service/kubernetes ClusterIP 10.245.0.1 <none> 443/TCP 10d

service/nginx-svc ClusterIP 10.245.232.0 <none> 80/TCP 30s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/nginx 1/1 1 1 30s

NAME DESIRED CURRENT READY AGE

replicaset.apps/nginx-74cd6f44f6 1 1 1 30

Delete the deployment.

# kubectl delete -k .

The above command deletes the resources from the cluster.

**Configure Common Settings**

You can configure common settings such as labels, annotations, namespace, name prefixes, or name suffixes using Kustomize across all the manifest files included in the **kustomization.yaml** file. Configuring common settings using Kustomize reduces the repetitive code among all the other manifest files while making it easier for the operator to keep the settings in sync in case of any changes reducing the risk of human error.

Create a new manifest file named **namespace.yaml**.

# nano namespace.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

apiVersion: v1

kind: Namespace

metadata:

name: test

The above manifest file declares a new namespace resource named **test**. You include this manifest file under the resources attribute in the **kustomization.yaml** file. Kubectl tries to create other resources before creating the namespace resource, which makes creating a new manifest file to define the namespace necessary.

Edit the **kustomization.yaml** manifest file.

# nano kustomization.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

resources:

- deployment.yaml

- service.yaml

- namespace.yaml

namespace: test

namePrefix: test-

commonLabels:

owner: mayank

The above manifest file uses the namespace and the namePrefix attribute, which sets the common namespace and name prefix of the resources defined in the included manifest files. Also, it uses the commonLabels attribute that accepts a list of key-value pairs which set common labels across all the resources. You can also use [nameSuffix](https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/namesuffix/) and [commonAnnotations](https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/commonannotations/) setting up name suffixes and common annotations.

Build and preview the configuration.

# kustomize build .

The above command generates a plain manifest containing all the resources in the Kustomize manifest file modified according to the common settings.

Apply the configuration.

# kubectl apply -k .

Verify the deployment.

# kubectl get all -n test

Output.

NAME READY STATUS RESTARTS AGE

pod/test-nginx-5ff865b54d-lw5vk 1/1 Running 0 18s

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

service/test-nginx-svc ClusterIP 10.245.205.173 <none> 80/TCP 19s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/test-nginx 1/1 1 1 19s

NAME DESIRED CURRENT READY AGE

replicaset.apps/test-nginx-5ff865b54d 1 1 1 19s

Delete the deployment.

# kubectl delete -k .

**Configure Base and Overlays**

You can configure different variants of your existing manifest files to adapt to environments such as staging, development, production, and so on using Kustomize utilizing base and overlays. This section explains the configuration of base and overlays and setting common settings like name prefixes and labels across all the resources included in the **kustomization.yaml** file.

The following is a metadata directory structure inside the project directory for configuring multiple variants based on the main manifest files.

base/

- deployment.yaml

- service.yaml

- kustomization.yaml

overlays/

development/

- kustomization.yaml

production/

- kustomization.yaml

Edit the **kustomization.yaml** file.

# nano kustomization.yaml

Overwrite the file with the following content and save it using CTRL X then ENTER.

resources:

- deployment.yaml

- service.yaml

Delete the **namespace.yaml** file.

# rm -f namespace.yaml

Create a new directory named **base**.

# mkdir base

Move the manifest files to the **base** directory.

# mv \*.yaml base/

The previous section demonstrated the configuration of a common namespace among all the included resources. This section does not include the steps to configure a common namespace for each variant as it adds many repetitive steps, such as creating a **namespace.yaml** file and so on. The steps to configure the common namespace when using base and overlays remain unchanged where you create a **namespace.yaml** file to declare a new Namespace resource and mention that file under the resources attribute in the Kustomize manifest file.

Create a new directory named **overlays**.

# mkdir overlays

Create **development** and **production** sub-directories inside the **overlays** folder.

# mkdir overlays/development overlays/production

Create the Kustomize manifest file for the development variant.

# nano overlays/development/kustomization.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

bases:

- ../../base

namePrefix: dev-

Create the Kustomize manifest file for the production variant.

# nano overlays/production/kustomization.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

bases:

- ../../base

namePrefix: prod-

Build and preview the configurations.

# kustomize build overlays/development

# kustomize build overlays/production

Apply the configurations.

# kubectl apply -k overlays/development

# kubectl apply -k overlays/production

The above commands generate and apply a plain manifest using Kustomize for each variant containing all the resources in the base variant with a name prefix according to the value set to the namePrefix attribute in the **kustomization.yaml**.

Verify the deployment.

# kubectl get all

Output.

NAME READY STATUS RESTARTS AGE

pod/dev-nginx-74cd6f44f6-fmg7r 1/1 Running 0 8s

pod/prod-nginx-74cd6f44f6-szf2p 1/1 Running 0 17s

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

service/dev-nginx-svc ClusterIP 10.245.28.102 <none> 80/TCP 8s

service/kubernetes ClusterIP 10.245.0.1 <none> 443/TCP 9d

service/prod-nginx-svc ClusterIP 10.245.246.9 <none> 80/TCP 17s

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/dev-nginx 1/1 1 1 8s

deployment.apps/prod-nginx 1/1 1 1 17s

NAME DESIRED CURRENT READY AGE

replicaset.apps/dev-nginx-74cd6f44f6 1 1 1 9s

replicaset.apps/prod-nginx-74cd6f44f6 1 1 1 18s

Delete the deployment.

# kubectl delete -k overlays/development

# kubectl delete -k overlays/production

**Configure Overlay Patches**

You can patch the existing manifest files using Kustomize to configure advanced settings like memory usage, image name or tag, service type, and so on. This section explains the configuration of overlay patches for configuring the different amounts of replicas, image name or tag, and service type for the resources included in the **kustomization.yaml** file.

**Update Replica Count**

The replicas attribute allows changing the number of replicas spawned for any resource included in the Kustomize manifest file. It accepts the name matching the resource and the updated replica count. Here, you update the replica count for the Deployment resource named **nginx** in the production overlay from 1 to 3.

Edit the Kustomize manifest file.

# nano overlays/production/kustomization.yaml

Make the following changes to the file and save the file using CTRL X then ENTER.

bases:

- ../../base

namePrefix: prod-

replicas:

- name: nginx

count: 3

Build and preview the configurations.

# kustomize build overlays/production

Apply the configurations.

# kubectl apply -k overlays/production

Verify the deployment.

# kubectl get deployment

Output.

NAME READY UP-TO-DATE AVAILABLE AGE

prod-nginx 3/3 3 3 7s

Delete the deployment.

# kubectl delete -k overlays/production

**Update Image Parameters**

The images attribute allows changing the image name or the image tag of any resource included in the Kustomize manifest file. It accepts the name used for matching the resource and values for a new name or new tag using the newName and the newTag attributes. Here, you update the image name for the Deployment resource named **nginx** in the production overlay from **nginx** to **httpd**.

Edit the Kustomize manifest file.

# nano overlays/production/kustomization.yaml

Make the following changes to the file and save the file using CTRL X then ENTER.

bases:

- ../../base

namePrefix: prod-

replicas:

- name: nginx

count: 3

images:

- name: nginx

newName: httpd

Apply the configurations.

# kubectl apply -k overlays/production

Verify the deployment.

# kubectl get deployment -o wide

Output.

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

prod-nginx 3/3 3 3 2m5s nginx httpd app=nginx-kustomize

The output confirms the updated image name. You can also change the image tag of any resource included in the Kustomize manifest file. Refer to the [images documentation](https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/images/) for more information.

Delete the deployment.

# kubectl delete -k overlays/production

**Update Resource Specification**

The patches attribute allows changing any specification like memory allocation, service type, storage class, and any resource included in the Kustomize manifest file. It accepts a list of manifest file names. Kustomize uses the contents of the included files for matching the resource using the metadata.name attribute. Here, you update the service type of the Service resource named **nginx-svc** in the production overlay from ClusterIP to NodePort.

Create a new manifest file named **service-patch.yaml**.

# nano overlays/production/service-patch.yaml

Add the following contents to the file and save the file using CTRL X then ENTER.

apiVersion: v1

kind: Service

metadata:

name: nginx-svc

spec:

type: NodePort

Edit the Kustomize manifest file.

# nano overlays/production/kustomization.yaml

Make the following changes to the file and save the file using CTRL X then ENTER.

bases:

- ../../base

namePrefix: prod-

replicas:

- name: nginx

count: 3

images:

- name: nginx

newName: httpd

patches:

- service-patch.yaml

Apply the configurations.

# kubectl apply -k overlays/production

Verify the deployment.

# kubectl get svc

Output.

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

kubernetes ClusterIP 10.245.0.1 <none> 443/TCP 10d

prod-nginx-svc NodePort 10.245.251.102 <none> 80:31842/TCP 6m42s

The output confirms the updated service type. The patches attribute is not limited to updating the type. It allows you to change any resource specification included in the Kustomize manifest file by creating a manifest file to patch the resource, as demonstrated above.

Delete the deployment.

# kubectl delete -k overlays/production

**Conclusion**

You learned the basic usage of the Kustomize configuration management tool, including the configuration of common settings like namespace or name prefixes, the configuration of base and overlays to set up different variants, and the configuration of overlay patches. You also installed the kustomize individual binary to preview configurations before applying them using kubectl. These operations are a good starting point for integrating Kustomize into your workflow, and you can refer to the [Kustomize documentation](https://kubectl.docs.kubernetes.io/references/kustomize/) for more information.