Kustomize (optional)

Kustomize allows you to manage Kubernetes manifest files using declarative "kustomization" files. It provides the ability to express "base" manifests for your Kubernetes resources and then apply changes using composition, customization and easily making cross-cutting changes across many resources.

For example, take a look at the following manifest file for the `checkout` Deployment:

~/environment/eks-workshop/base-application/checkout/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: checkout
  labels:
    app.kubernetes.io/created-by: eks-workshop
    app.kubernetes.io/type: app
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: checkout
      app.kubernetes.io/instance: checkout
      app.kubernetes.io/component: service
  template:
    metadata:
      annotations:
        prometheus.io/path: /metrics
        prometheus.io/port: "8080"
        prometheus.io/scrape: "true"
      labels:
        app.kubernetes.io/name: checkout
        app.kubernetes.io/instance: checkout
        app.kubernetes.io/component: service
        app.kubernetes.io/created-by: eks-workshop
    spec:
      serviceAccountName: checkout
      securityContext:
        fsGroup: 1000
      containers:
        - name: checkout
          envFrom:
            - configMapRef:
                name: checkout
          securityContext:
            capabilities:
              drop:
              - ALL
            readOnlyRootFilesystem: true
          image: "public.ecr.aws/aws-containers/retail-store-sample-
checkout:0.4.0"
          imagePullPolicy: IfNotPresent
          ports:
```

```
        - name: http
          containerPort: 8080
          protocol: TCP
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
        initialDelaySeconds: 30
        periodSeconds: 3
      resources:
        limits:
          memory: 512Mi
        requests:
          cpu: 250m
          memory: 512Mi
      volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
  volumes:
    - name: tmp-volume
      emptyDir:
        medium: Memory
```

This file has already been applied in the previous Getting Started lab, but let's say we wanted to scale this component horizontally by updating the `replicas` field using Kustomize. Rather than manually updating this YAML file, we'll use Kustomize to update the `spec/replicas` field from 1 to 3.

To do so, we'll apply the following kustomization.

- The first tab shows the kustomization we're applying
- The second tab shows a preview of what the updated `Deployment/checkout` file looks like after the kustomization is applied
- Finally, the third tab shows just the diff of what has changed

- Kustomize Patch

- Deployment/checkout
- Diff

~/environment/eks-workshop/modules/introduction/kustomize/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: checkout
spec:
  replicas: 3
```

You can generate the final Kubernetes YAML that applies this kustomization with the `kubectl kustomize` command, which invokes `kustomize` that is bundled with the `kubectl` CLI:

```
~$kubectl kustomize ~/environment/eks-workshop/modules/introduction/kustomize
```

This will generate a lot of YAML files, which represents the final manifests you can apply directly to Kubernetes. Let's demonstrate this by piping the output from `kustomize` directly to `kubectl apply`:

```
~$kubectl kustomize ~/environment/eks-workshop/modules/introduction/kustomize | kubectl apply -f -
namespace/checkout unchanged
serviceaccount/checkout unchanged
configmap/checkout unchanged
service/checkout unchanged
service/checkout-redis unchanged
deployment.apps/checkout configured
deployment.apps/checkout-redis unchanged
```

You'll notice that a number of different `checkout`-related resources are "unchanged", with the `deployment.apps/checkout` being "configured". This is intentional — we only want to apply changes to the `checkout` deployment. This happens because running the previous command actually applied two files: the Kustomize `deployment.yaml` that we saw above, as well as the following `kustomization.yaml` file which matches all files in the `~/environment/eks-workshop/base-application/checkout` folder. The `patches` field specifies the specific file to be patched:

~/environment/eks-workshop/modules/introduction/kustomize/kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../../base-application/checkout
patches:
- path: deployment.yaml
```

To check that the number of replicas has been updated, run the following command:

```
~$kubectl get pod -n checkout -l app.kubernetes.io/component=service
NAME                        READY   STATUS    RESTARTS   AGE
checkout-585c9b45c7-c456l   1/1     Running   0          2m12s
checkout-585c9b45c7-b2rrz   1/1     Running   0          2m12s
checkout-585c9b45c7-xmx2t   1/1     Running   0          40m
```

Instead of using the combination of `kubectl kustomize` and `kubectl apply` we can instead accomplish the same thing with `kubectl apply -k <kustomization_directory>` (note the `-k` flag instead of `-f`). This approach is used through this workshop to make it easier to apply changes to manifest files, while clearly surfacing the changes to be applied.

Let's try that:

```
~$kubectl apply -k ~/environment/eks-workshop/modules/introduction/kustomize
```

To reset the application manifests back to their initial state, you can simply apply the original set of manifests:

```
~$kubectl apply -k ~/environment/eks-workshop/base-application
```

Another pattern you will see used in some lab exercises looks like this:

```
~$kubectl kustomize ~/environment/eks-workshop/base-application \
  | envsubst | kubectl apply -f-
```

This uses `envsubst` to substitute environment variable placeholders in the Kubernetes manifest files with the actual values based on your particular environment. For example in some manifests we need to reference the EKS cluster name with `$EKS_CLUSTER_NAME` or the AWS region with `$AWS_REGION`.

Now that you understand how Kustomize works, proceed to the Fundamentals module.

To learn more about Kustomize, you can refer to the official Kubernetes documentation.