Kubernetes 3

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Kubectl:The Kubernetes CLI

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. **Define Kubectl and the command structure**
2. **List the three command types, their features, and advantages**
3. **List commonly used commands with specific examples**

Kubectl

- Kubectl is the **Kubernetes command line interface (or CLI).**
- Kubectl stands for **Kube Command Tool Line.**
- Kubectl is one of the key tools for working with Kubernetes and it helps users **deploy applications, inspect and manage cluster resources, view logs, and more.**
- It provides many features for users who work with Kubernetes clusters and manage running cluster workloads.

Three Kubectl command types are:

- ❑ **Imperative commands**
- ❑ **Imperative object configuration**
- ❑ **Declarative object configuration.**

Kubectl commands Structure

- Kubectl[command][type][name][flags]

- Command means **any operation to be performed**, like ‘**create’, ‘get’, ‘apply’, or ‘delete’,**
- type means **resource type**, like ‘**pod’, ‘deployment’, or ‘ReplicaSet**’,
- name means **resource name** (if applicable), and
- Flags means **special options or modifiers** that override default values.

Imperative Commands

•Imperative commands allow you to **create, update, and delete live objects directly.**

- ❑ Operations should be specified to the command as arguments or flags.
- ❑ Imperative commands are easy to learn and run. For example, to **create a pod with a specific container,** simply run the command as shown:

  Kubectl run nginx –image nginx

- ❑ They **don't provide an audit trail**, which is important for tracking changes.
- ❑ They **aren't flexible** since options are limited, they **don't use templates**, and they **cannot integrate with change review processes**.
- ❑ They are **ideal for development and test environments**.

Imperative Command Limitations

•Suppose a developer runs a command to deploy an application. Another developer wants to deploy that same application, but they cannot because there is **no configuration file**. ***The second developer must check with the first developer for the exact command to deploy and then run it.***

•It would be best if both developers **used a template for the deployment** since it overcomes the limitations of working with imperative commands.

Imperative Object Configuration

- In imperative object configuration, the **kubectl command specifies required operations, optional flags, and at least one file name.**
- The specified **configuration file must contain a full definition of the objects** in YAML or JSON format.
- To create the objects defined in the file, run the command

  'kubectl create -f nginx.yaml'.

Advantages

- It may be stored in a **source control system like Git**.
- It can **integrate with change processes**.
- It **provides audit trails and templates for creating new objects**.

Disadvantages

- It requires **understanding of the object schema**.
- It requires **writing a YAML or JSON file**.

Imperative Object Configuration Limitations

- You **need to specify all necessary command operations.**

For example, if a developer ***performs an update operation that isn’t merged into the configuration file, then another developer cannot use the updated configuration in future deployments***. The second developer instead uses the original or previous configuration.

- It is better to define the **desired state in a shared configuration file, then when you deploy**, Kubernetes automatically determines the necessary operations. This is known as ***declarative object configuration***

Declarative Object Configuration

- Declarative object configuration **stores configuration data in files.**
- **Operations are identified by Kubectl instead of being specified by the user**. This works on directories or individual files.

  For example, the ***'kubectl apply –f nginx'*** command mentions a directory, then applies configuration data to all files in that directory.
- **The user is not required to perform any operations since they are performed by the system automatically.**
- Configuration files **define a desired state, and Kubernetes actualizes that state.**
- This approach is **the ideal method for production systems.**

Declarative Object Configuration Benefits

A developer performs updates to a running application. Since **configuration data is stored in the shared template, there is still one source of truth for the configuration of this object**. Now, even if another developer misses several of these updates, all they need to do is apply the current configuration template to ensure the deployed object is as expected. Kubernetes automatically determines and performs the necessary operations to match the current state to the desired state.

Kubectl Commonly used commands

Command	Description
get	Accesses a file, container, or any other resource.
delete	Delete command deletes a file or container.
autoscale	Applies the autoscaling process to the selected file or container.
apply	Apply/change configuration of a resource using a file or stdin
create	Create one or more resources using file or stdin
describe	Describe or detail a file or container
edit	Make changes to a file or a container
exec	Execute a command on a container in a specific pod
expose	Make a running file/container available
label	Apply a label to a file or a container

Examples of Kubectl commands usage

Kubectl ‘get’ commands allow you to list:

- **services in a current namespace**
- **pods in all namespaces**
- **a particular deployment**
- **and pods in the current namespace**

```
kubectl get services
kubectl get pods --all-namespaces
kubectl get deployment my-dep
```

All commands can be found at Kubernetes.io

Examples of Kubectl commands usage

- Kubectl ‘apply’ commands **create resources using YAML or JSON files.**
- They use extensions like **.yaml, .yml, or .json**.
- You can use ‘apply’ commands to create resources:
  - **from multiple files**
  - **or from a URL.**

```
kubectl apply –f ./my1.yaml ./my2.yaml
kubectl apply –f https://git.io/vPieo
```

All commands can be found at Kubernetes.io

Examples of Kubectl commands usage

- Kubectl ‘scale’ **commands scale the number of replicas.** You can use ‘scale’ commands to scale:
  - a ReplicaSet named ‘foo’ to 3.
  - a resource in ‘foo.yaml’ to 3.

kubectl scale –replicas=3 rs/foo

kubectl scale –replicas=3 –f foo.yaml

All commands can be found at [Kubernetes.io]()

Create a resource(cmd+output)

Let’s create a **deployment with three replicas of the nginx image.**

- Create the deployment using an **‘apply’** command.
- The output confirms the deployment creation.
- The **‘get deployment**’ command provides the specific ‘my-dep’ deployment details.
- The output confirms the creation of three replicas: ready, up-to-date, and available.

```
kubectl apply –f nginx.yaml
deployment.apps/nginx-deployment created

kubectl get deployment my-dep
NAME              READY     UP-TO-DATE     AVAILABLE     AGE
Nginx-deployment   3/3          3              3          73s
```

ReplicaSet

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. Define a ReplicaSet.
2. Explain how a ReplicaSet works.
3. List the benefits of using a ReplicaSet.

Single Pod Deployment Limitations

•If an application is deployed on a single pod, the pod will be unable to perform certain actions if requests increase manifold or outages occur.

Single-pod deployments cannot:

- Accommodate growing demands of the application and load balancing across pods.
- Handle outages by eliminating a single point of failure.
- Minimize downtime and service interruptions by providing high availability through redundant pods
- Automatically restart deployments if something goes wrong.

ReplicaSet

•A ReplicaSet ensures the right number of pods are always up and running. It always tries to match the actual state of the replicas to the desired state.

A ReplicaSet:

- Adds or deletes pods for scaling and redundancy, which helps maintain availability.
- It replaces failing pods or deletes additional pods to maintain the desired state.
- It supersedes a ReplicaController and should be used instead.
- A ReplicaSet is created for you when you create a deployment in your cluster. Deployments manage ReplicaSets.

Replicaset | Deployments | Pods

- Kubernetes is designed to keep object types independent.
- ReplicaSet does not own any of the pods.
- It uses pod labels to decide which pods to acquire when bringing a deployment to the desired state.

```
apiVersion:apps/v1
kind: Deployment
metadata:
       name: hello-kubernetes
        labels:
          app:nginx
 spec:
        replicas: 3
        selector:
        matchLabels:
             app:hello-kubernetes
   template:
        metadata:
         labels:
             app:hello-kubernetes
    spec:
        containers:
           -name: hello-kubernetes
             image:avitakatal/hello-kubernetes:1.5
              ports:
               -containerPort: 8080
```

Create ReplicaSet from Deployment

- A ReplicaSet is automatically created for you when you create a deployment.

```
kubectl get deployment my-dep
NAME                 READY      UP-TO-DATE      AVAILABLE      AGE
hello-kubernetes     1/1             1               1         13m
kubectl get replicaset
NAME                            DESIRED      CURRENT        READY        AGE
 hello-kubernetes-56557b546f8    1            1              1           14m
```

ReplicaSet: Describe the pod

- Additionally, if you describe the pod, you will see the pod’s details and that it is “controlled by:” the same ReplicaSet

kubectl describe pod hello-kubernetes-56557b546f8-ntxgk

```
Name:
Namespace
Priority
Node:
StartTime::
Labels:
Annotations:
Stats:
IP:
IPs:
  IP:
Controlled by: Replicsaet/hello-kubernetes-56557b546f8
```

Create ReplicaSet from Scratch

To create a ReplicaSet from scratch:

•Apply a yaml file with the kind attribute set to “ReplicaSet” as shown. If you define the number of replicas as 1, you get 1 pod. This is similar to creating a default without specifying the number of replicas in your yaml file.

Let’s step through creating a ReplicaSet from scratch:

•Use the ‘create ReplicaSet’ command.

•Output shows a ReplicaSet was created.

```
apiVersion:apps/v1
kind: ReplicaSet
metadata:
        name: hello-kubernetes
  spec:
         replicas: 1
         selector:
         matchLabels:
               app:hello-kubernetes
    template:
          metadata:
           labels:
               app:hello-kubernetes
     spec:
         containers:
             -name: hello-kubernetes
               image:avitakatal/hello-kubherntes:1.5
                ports:
                 -containerPort: 8080
```

Create ReplicaSet from Scratch

Let’s step through creating a ReplicaSet from scratch:

•Use the ‘create ReplicaSet’ command. Output shows a ReplicaSet was created.

```
kubectl create  -f replicaset.yaml
```

- Confirm it was created by using the ‘get pods’ command and observe the status as ‘Running’

```
kubectl get pods
```

- And then, use the ‘get rs’ command (rs is short for ReplicaSet). Output shows the name and other details of the newly created ReplicaSet and its pod.

```
kubectl get rs
```

Creating a Deployment that includes a ReplicaSet is recommended over creating a standalone ReplicaSet.

ReplicaSet: Create Deployment

- Before you scale a deployment, you must ensure you have a deployment and pod.

Create a deployment using the ‘create’ command. The output will confirm a deployment was created. The deployment creates a pod by default.

kubectl create –f deployment.yaml

You can confirm this with the ‘get pods’ command. The output shows the pod name and other details.

kubectl get pods

You can check the deployment details with the ‘get deploy’ command.
The output shows the deployment is named ‘hello-Kubernetes’.

kubectl get deploy

ReplicaSet: Scale Deployment

•Now, once the deployment and pod are in place, use the ‘scale’ command to scale the deployment and set the desired number of replicas. Here we set the number of replicas to 3, and the output confirms the deployment is scaled. If you use the ‘get pods’ command, you will see 3 pods running.

```
kubectl scale deploy hello-kubernetes --replicas=3.

kubectl get pods
```

ReplicaSet: Maintain Desired State

•Let’s observe how the ReplicaSet maintains the desired state when a pod is deleted.

•First, use the ‘get pods’ command. Our 3 pods appear in the output, as expected. Now, use the ‘delete pod’ command to delete the pod ending in ‘5mflw’. The pod ending in “5mflw” is deleted, and notice that the desired state does not match the actual state. And that the deleted pod is replaced by a new pod automatically. Now, use the ‘get pods’ command again. The output shows the ReplicaSet immediately created a new pod ending in “6lw4r” to get the total number of pods back to 3.

```
kubectl get pods
kubectl delete pod hello-kubernetes-55655bbf568-5mflw
kubetctl get pods
```

ReplicaSet: Maintain Desired State

- Now let’s observe how the ReplicaSet maintains desired state when a pod is created.
- The ‘get pods’ command shows our 3 pods from before. Use the ‘create pod’ command to create a pod ending in ‘mx9rp’. Then use the ‘get pods’ command again. The output now shows 4 pods. Desired state does not match actual state. Since the ReplicaSet always strives to match the actual state to the desired state, the new pod (ending in “mx9rp”) is marked for deletion and removed automatically. The ‘get pods’ command now shows the total number of pods restored to 3.

```
kubectl get pods
kubectl create pod hello-kubernetes-55655bbf568-mx9rp
kubetctl get pods
```

AutoScaling

Prepared by:
Ms. Avita Katal
Assistant Professor (SG)
School of Computer Science
UPES, Dehradun

Objectives

1. **Define autoscaling**
2. **Explain the three types of autoscalers**
3. **Demonstrate how each autoscaler works**

Introduction

•If an application is deployed on a single pod, the pod will be unable to perform certain actions if requests increase manifold or outages occur.

Single-pod deployments cannot:

- Accommodate growing demands of the application and load balancing across pods.
- Handle outages by eliminating a single point of failure.
- Minimize downtime and service interruptions by providing high availability through redundant pods
- Automatically restart deployments if something goes wrong.

- ReplicaSets provide a good start for scaling, but you don’t always want 10 instances of your resource running. You should be able to scale as needed.

- Kubernetes autoscaling helps optimize resource usage and costs by automatically scaling a cluster in line with demand.

- Kubernetes enables autoscaling at two different layers:
  - ✓ **the cluster or node level**
  - ✓ **and the pod level**.

Kubernetes Autoscalers

Three types of autoscalers are available in Kubernetes:

- **Horizontal Pod Autoscaler (or HPA)**
- **Vertical Pod Autoscaler (or VPA)**
- **Cluster Autoscaler (or CA)**

The **Horizontal Pod Autoscaler (or HPA)** adjusts the number of replicas of an application by **increasing or decreasing the number of pods.**

The **Vertical Pod Autoscaler (or VPA)** adjusts the resource requests and limits of a container **by increasing or decreasing the resource size or speed of the pods.**

And the **Cluster Autoscaler (or CA)** adjusts the number of nodes in the cluster when pods fail to schedule, or **demand increases or decreases in relation to the nodes' capacity**.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

In this example, the HPA will adjust the number of pods to maintain an average CPU utilization of 50%.

How it works:

- **Metrics:** HPA periodically queries metrics from either the resource metrics API (like CPU utilization) or custom metrics API (your application-specific metrics).
- **Target Value:** You set a target value for the metric. For example, you might want to maintain 50% CPU utilization.
- **Scaling Thresholds:** You can define minimum and maximum numbers of pods to ensure you don't scale below or above certain limits.
- **Scaling Algorithm:** Based on the observed metric value compared to the target value, HPA calculates and adjusts the desired number of pods.

```yaml
apiVersion: "autoscaling.k8s.io/v1"
kind: VerticalPodAutoscaler
metadata:
  name: my-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "Auto"
```

How it works:

- **Metrics Collection:** VPA collects historical resource utilization data for pods.
- **Analysis:** Based on the collected data and real-time metrics, VPA analyzes the resource usage patterns of the pods.
- **Adjustment:** VPA adjusts the CPU and memory requests of the pods to match their actual usage, ensuring they have enough resources without over-provisioning.

VPA can work in two modes: "Recommender" mode suggests resource requests without applying them, allowing administrators to review the changes before implementation, and "Updater" mode automatically applies the resource changes.

In this example, the VPA adjusts the resource requests for pods belonging to the specified deployment (my-app) automatically.

How Cluster Autoscaler Works:

1.**Node Utilization Monitoring:** Cluster Autoscaler continuously monitors the utilization of nodes in the cluster, tracking metrics like CPU, memory, and other resources.

2.**Pod Resource Requirements:** When a pod cannot be scheduled due to resource constraints (for example, insufficient CPU or memory), Cluster Autoscaler identifies the node pools that can satisfy the pod's resource requirements.

3.**Node Scaling:** If Cluster Autoscaler finds a node pool where the pod can be scheduled, it communicates with the cloud provider's API to add new nodes to the pool, thereby increasing the cluster size. This ensures that the pod can be scheduled and run without resource constraints.

4.**Node Removal:** Conversely, if nodes are underutilized and all the pods on a node can be safely moved to other nodes, Cluster Autoscaler can remove nodes. This helps in optimizing resource usage and reducing costs.

5.**Constraints:** Cluster Autoscaler allows you to set constraints to prevent certain nodes from being removed. For example, nodes with specific labels indicating special hardware or nodes in specific availability zones can be protected from scaling down.

```
kubectl autoscale deployment <deployment-name> --min=<min-pods> --max=<max-pods> --cpu-percent=<cpu-utilization>
```

<deployment-name> is the name of the deployment you want to autoscale.
<min-pods> is the minimum number of pods to run.
<max-pods> is the maximum number of pods to run.
<cpu-utilization> is the target average CPU utilization across all pods. When the average CPU utilization exceeds this value, the HPA scales up the number of pods.

```
kubectl autoscale deployment my-app-deployment --min=2 --max=10 --cpu-percent=50
```

Here's an example of using kubectl autoscale to create an HPA for a deployment named my-app-deployment with a target CPU utilization of 50%, a minimum of 2 pods, and a maximum of 10 pods:

Configuring Vertical Pod Autoscaler (VPA) and Cluster Autoscaler (CA) typically involves creating YAML manifests and applying them using kubectl commands. Below are examples for both Vertical Pod Autoscaler and Cluster Autoscaler:

Vertical Pod Autoscaler (VPA):
To configure Vertical Pod Autoscaler, you need to create a VPA manifest and apply it using kubectl:

Example VPA Manifest (vpa.yaml):

```
apiVersion: "autoscaling.k8s.io/v1"
kind: VerticalPodAutoscaler
metadata:
  name: my-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "Auto"
```

Apply the VPA manifest using kubectl:

```
kubectl apply -f vpa.yaml
```

This example sets up a VPA for the my-app Deployment, allowing it to automatically adjust its resource requests based on its usage patterns.

UPES
UNIVERSITY OF TOMORROW

Thank You