Practice Lab: Autoscaling and Secrets Management

This practice lab is designed to provide hands-on experience with Kubernetes, focusing on vertical and horizontal pod autoscaling and secrets management.

Objectives

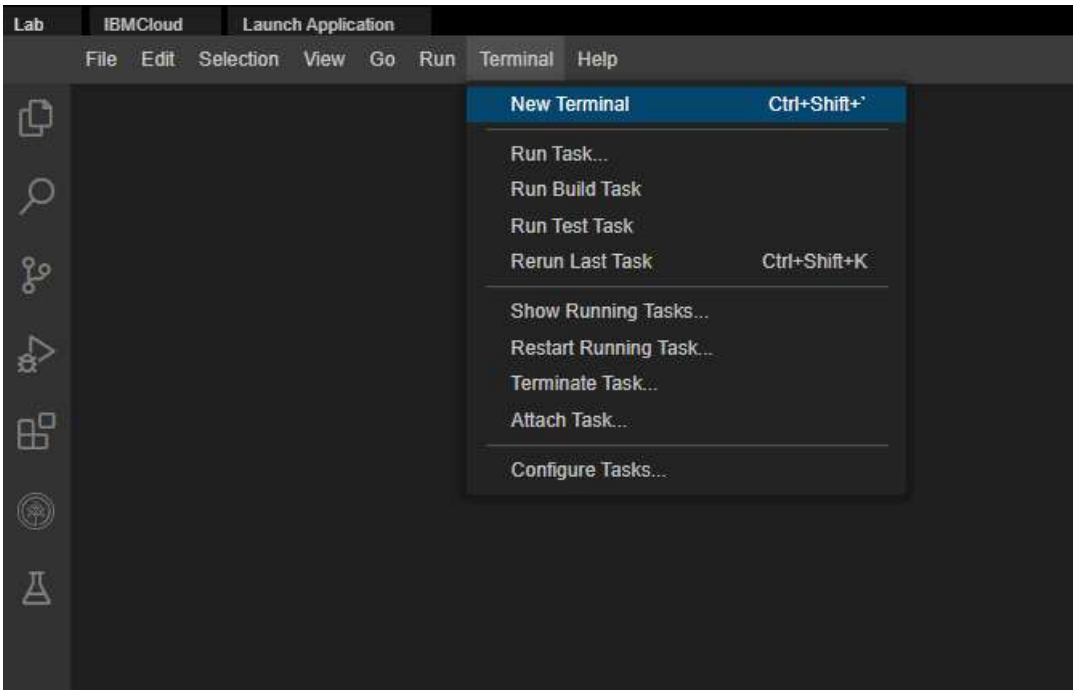
In this practice lab, you will:

- Build and deploy an application to Kubernetes
- Implement Vertical Pod Autoscaler (VPA) to adjust pod resource requests/limits
- Implement Horizontal Pod Autoscaler (HPA) to scale the number of pod replicas based on resource utilization
- Create a Secret and update the deployment for using it

> **Note: Kindly complete the lab in a single session without any break because the lab may go in offline mode and cause errors. If you face any issues/errors during the lab process, please logout from the lab environment. Then, clear your system cache and cookies and try to complete the lab.**

Setup the environment

On the menu bar, click `Terminal` and select the `New Terminal` option from the drop-down menu.



> Note: If the terminal is already open, please skip this step.

Step 1: Verify kubectl version

Before proceeding, ensure that you have kubectl installed and properly configured. To check the version of kubectl, run the following command:

```
kubectl version
```

You should see the following output, although the versions may be different:

```
theia@theiadocker-ksundararaja:/home/project$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short.  Use --outp
 the full version.
Client Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.6", GitCommit:"741c8db18a52787d734cbe4795f0b4ad860906d6
an", BuildDate:"2023-09-13T09:21:34Z", GoVersion:"go1.20.8", Compiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v5.0.1
Server Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.14+IKS", GitCommit:"8db9c4804f1f37994e83aa5326700063697
:"clean", BuildDate:"2024-05-15T17:52:02Z", GoVersion:"go1.21.9", Compiler:"gc", Platform:"linux/amd64"}
theia@theiadocker-ksundararaja:/home/project$
```

Step 2: Clone the project repository

Clone the repository with the starter code to commence the project.

```
git clone https://github.com/ibm-developer-skills-network/k8-scaling-and-secrets-mgmt.git
```

Exercise 1: Build and deploy an application to Kubernetes

The Dockerfile in this repository already has the code for the application. You are just going to build the docker image and push it to the registry.

You will be giving the name `myapp` to your Kubernetes deployed application.

Step 1: Build the Docker image

1. Navigate to the project directory.

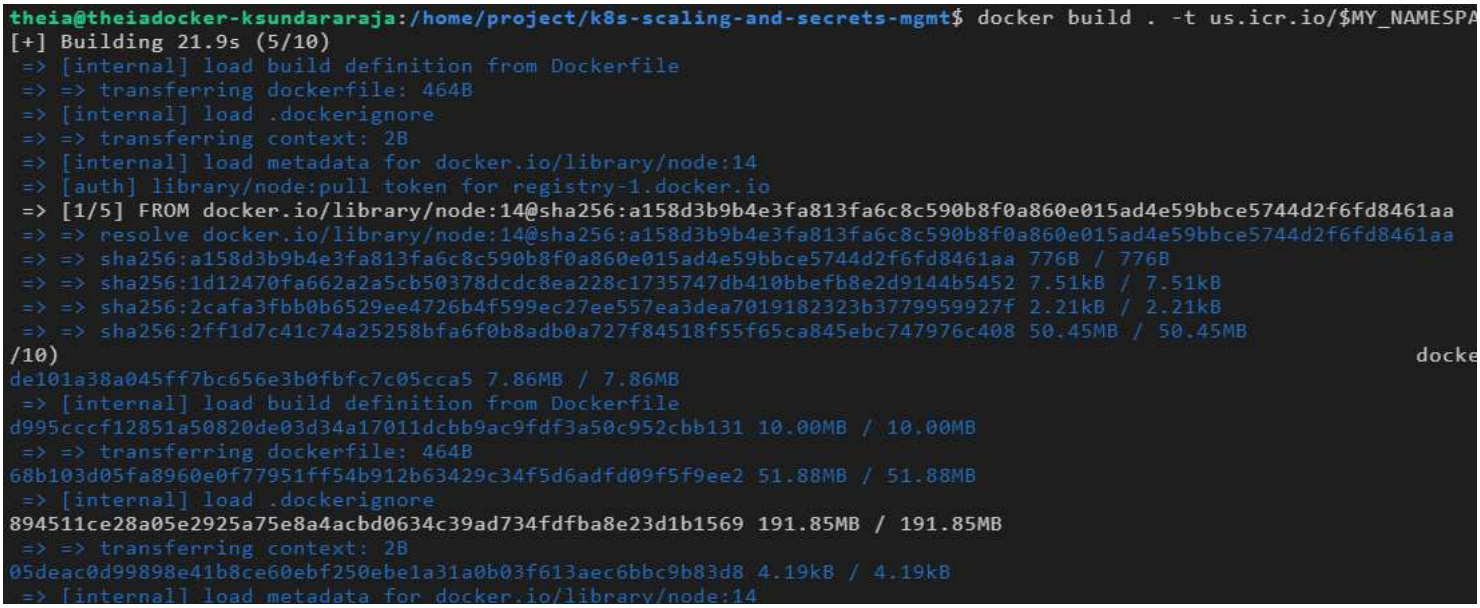
   ```
   cd k8-scaling-and-secrets-mgmt
   ```

2. Export your namespace.

   ```
   export MY_NAMESPACE=sn-labs-$USERNAME
   ```

3. Build the Docker image.

   ```
   docker build . -t us.icr.io/$MY_NAMESPACE/myapp:v1
   ```



Step 2: Push and list the image

1. Push the tagged image to the IBM Cloud Container Registry.

   ```
   docker push us.icr.io/$MY_NAMESPACE/myapp:v1
   ```

2. List all the images available. You will see the newly created `myapp` image.

   ```
   ibmcloud cr images
   ```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ ibmcloud cr images
Listing images...

Repository                                                                 Tag
          Digest          Namespace              Created          Size     Security status
us.icr.io/sn-labs-ksundararaja/myapp                                       v1
          28d591aa8284    sn-labs-ksundararaja   2 minutes ago    350 MB   -
us.icr.io/sn-labsassets/categories-watson-nlp-runtime                      latest
          6b01b1e5527b    sn-labsassets          2 years ago      3.1 GB   -
us.icr.io/sn-labsassets/classification-watson-nlp-runtime                  latest
          dbd407898549    sn-labsassets          2 years ago      4.0 GB   -
us.icr.io/sn-labsassets/concepts-watson-nlp-runtime                        latest
          1e4741f10569    sn-labsassets          2 years ago      3.2 GB   -
us.icr.io/sn-labsassets/custom-watson-nlp-runtime                          latest
          f6513e19a33d    sn-labsassets          2 years ago      6.5 GB   -
us.icr.io/sn-labsassets/detag-watson-nlp-runtime                           latest
          38916c2119fc    sn-labsassets          2 years ago      2.7 GB   -
us.icr.io/sn-labsassets/emotion-watson-nlp-runtime                         latest
          1c9de1d27318    sn-labsassets          2 years ago      4.0 GB   -
us.icr.io/sn-labsassets/entity-mentions-bert-watson-nlp-runtime            latest
          57d92957214f    sn-labsassets          2 years ago      3.8 GB   -
us.icr.io/sn-labsassets/entity-mentions-bilstm-watson-nlp-runtime          latest
```

Step 3: Deploy your application

1. Open the `deployment.yaml` file located in the main project directory. It's content will be as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: myapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: us.icr.io/<your SN labs namespace>/myapp:v1
        imagePullPolicy: Always
        name: myapp
        ports:
        - containerPort: 3000
          name: http
        resources:
          limits:
            cpu: 50m
          requests:
            cpu: 20m
```

2. Replace `<your SN labs namespace>` with your actual SN lab's namespace.

▶ Click here for the ways to get your namespace

3. Apply the deployment.

```
kubectl apply -f deployment.yaml
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl apply -f deployment.yaml
deployment.apps/myapp created
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

4. Verify that the application pods are running and accessible.

```
kubectl get pods
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
myapp-6cc7f9ffcf-2xnm6   1/1     Running   0          29s
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

Step 4: View the application output

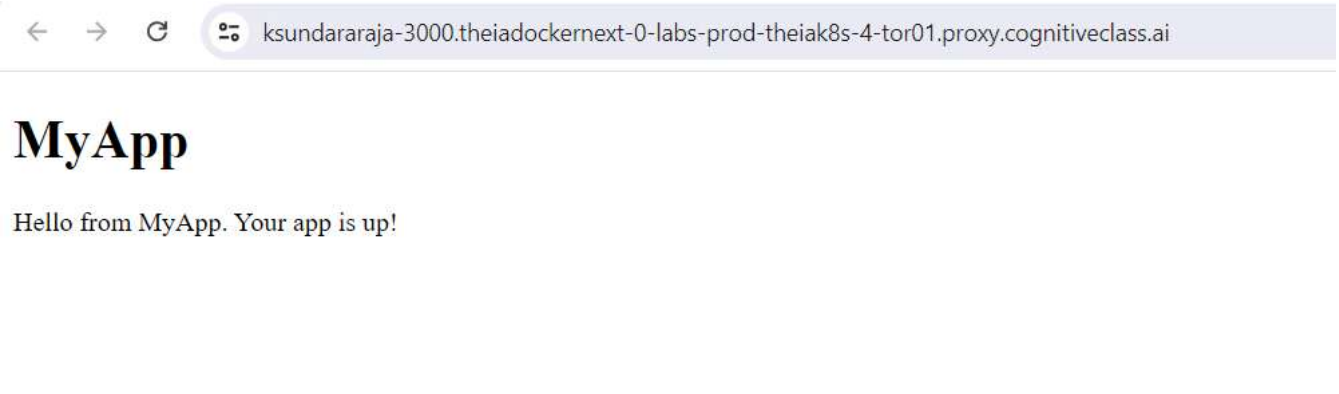
1. Start the application on port-forward:

```
kubectl port-forward deployment.apps/myapp 3000:3000
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl port-forward deployment.apps/m
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

2. Launch the app on Port `3000` to view the application output.

3. You should see the message `Hello from MyApp. Your app is up!.`



4. Stop the server before proceeding further by pressing `CTRL + C`.

5. Create a ClusterIP service for exposing the application to the internet:

```
kubectl expose deployment/myapp
```

```
^Ctheia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl expose deployment/myapp
service/myapp exposed
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

Exercise 2: Implement Vertical Pod Autoscaler (VPA)

Vertical Pod Autoscaler (VPA) helps you manage resource requests and limits for containers running in a pod. It ensures pods have the appropriate resources to operate efficiently by automatically adjusting the CPU and memory requests and limits based on the observed resource usage.

Step 1: Create a VPA configuration

You will create a Vertical Pod Autoscaler (VPA) configuration to automatically adjust the resource requests and limits for the `myapp` deployment.

Explore the `vpa.yaml` file, which has the following content:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: myvpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: myapp
  updatePolicy:
    updateMode: "Auto"  # VPA will automatically update the resource requests and limits
```

Explanation

This YAML file defines a VPA configuration for the `myapp` deployment. The `updateMode: "Auto"` setting means that VPA will automatically update the resource requests and limits for the pods in this deployment based on the observed usage.

Step 2: Apply the VPA

Apply the VPA configuration using the following command:

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

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl apply -f vpa.yaml
verticalpodautoscaler.autoscaling.k8s.io/myvpa created
```

Step 3: Retrieve the details of the VPA

1. Retrieve the created VPA:

```
kubectl get vpa
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get vpa
NAME    MODE   CPU   MEM       PROVIDED   AGE
myvpa   Auto   25m   262144k   True       29s
```

This output shows that:

- The VPA named `myvpa` is in Auto mode, recommending 25 milli-cores of CPU and 256 MB of memory for the pods it manages.
- It has been created 29 seconds ago and has been providing these recommendations since then.

2. Retrieve the details and current running status of the VPA.

```
kubectl describe vpa myvpa
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl describe vpa myvpa
Name:         myvpa
Namespace:    sn-labs-ksundararaja
Labels:       <none>
Annotations:  <none>
API Version:  autoscaling.k8s.io/v1
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2024-06-25T15:17:04Z
  Generation:          1
  Resource Version:    287538855
  UID:                 57f5fac3-8720-4340-b877-3831490fb03f
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         myapp
  Update Policy:
    Update Mode:  Auto
Status:
  Conditions:
    Last Transition Time:  2024-06-25T15:17:20Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  myapp
      Lower Bound:
        Cpu:     25m
        Memory:  262144k
      Target:
        Cpu:     25m
        Memory:  262144k
      Uncapped Target:
        Cpu:     25m
        Memory:  262144k
      Upper Bound:
        Cpu:     60m
        Memory:  262144k
```

Explanation

The output of `kubectl describe vpa myvpa` is providing recommendations for CPU and memory:

Resource	**Definition**
Lower Bound	Minimum resources the VPA recommends.
Target	Optimal resources the VPA recommends.
Uncapped Target	Target without any predefined limits.
Upper Bound	Maximum resources the VPA recommends.

Resource	**CPU**	**Memory**
Lower Bound	25m	256MiB (262144KiB)
Target	25m	256MiB
Uncapped Target	25m	256MiB
Upper Bound	671m	1.34GiB (1438074878KiB)

These recommendations indicate that the VPA is functioning correctly and is providing target values based on observed usage.

You can stop the Kubernetes proxy and load generation commands on the other two terminals by pressing `CTRL + C` before continuing further.

Exercise 3: Implement Horizontal Pod Autoscaler (HPA)

Horizontal Pod Autoscaler (HPA) automatically scales the number of pod replicas based on observed CPU/memory utilization or other custom metrics. VPA adjusts the resource requests and limits for individual pods. However, HPA changes the number of pod replicas to handle the load.

Step 1: Create an HPA configuration

You will configure a Horizontal Pod Autoscaler (HPA) to scale the number of replicas of the `myapp` deployment based on CPU utilization.

Explore the `hpa.yaml` file, which has the following content:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: myhpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1         # Minimum number of replicas
  maxReplicas: 10        # Maximum number of replicas
  targetCPUUtilizationPercentage: 5  # Target CPU utilization for scaling
```

Explanation

This YAML file defines a Horizontal Pod Autoscaler configuration for the `myapp` deployment. The HPA will ensure that the average CPU utilization across all pods remains close to 5%. If the utilization is higher, HPA will increase the number of replicas, and if it's lower, it will decrease the number of replicas within the specified range of 1 to 10 replicas.

Step 2: Configure the HPA

Apply the HPA configuration:

```
kubectl apply -f hpa.yaml
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/myhpa created
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

Step 3: Verify the HPA

Obtain the status of the created HPA resource by executing the following command:

```
kubectl get hpa myhpa
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get hpa myhpa
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myhpa   Deployment/myapp   0%/5%     1         10        1          61s
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

This command provides details about the current and target CPU utilization and the number of replicas.

Step 4: Start the Kubernetes proxy

Open another terminal and start the Kubernetes proxy:

```
kubectl proxy
```

```
theia@theiadocker-ksundararaja:/home/project$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Step 5: Spam and increase the load on the app

Open another new terminal and enter the below command to spam the app with multiple requests for increasing the load:

```
for i in `seq 100000`; do curl -L localhost:8001/api/v1/namespaces/sn-labs-$USERNAME/services/myapp/proxy; done
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple App - v1</title>
    <link rel="stylesheet" href="./style.css">
</head>
<body>
    <h1>MyApp</h1>
    <p>Hello from MyApp. Your app is up!</p>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple App - v1</title>
    <link rel="stylesheet" href="./style.css">
</head>
<body>
    <h1>MyApp</h1>
    <p>Hello from MyApp. Your app is up!</p>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple App - v1</title>
    <link rel="stylesheet" href="./style.css">
</head>
<body>
    <h1>MyApp</h1>
    <p>Hello from MyApp. Your app is up!</p>
</body>
</html>
```

Proceed with further commands in the new terminal.

Step 6: Observe the effect of autoscaling

1. Run the following command to observe the replicas increase in accordance with the autoscaling:

```
kubectl get hpa myhpa --watch
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get hpa myhpa --watch
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myhpa   Deployment/myapp   95%/5%    1         10        5          3m26s
myhpa   Deployment/myapp   45%/5%    1         10        10         3m30s
myhpa   Deployment/myapp   25%/5%    1         10        10         4m16s
myhpa   Deployment/myapp   19%/5%    1         10        10         4m31s
```

2. You will see an increase in the number of replicas, which shows that your application has been autoscaled.

3. Terminate this command by pressing `CTRL + C`.

Step 7: Observe the details of the HPA

1. Run the following command to observe the details of the horizontal pod autoscaler:

```
kubectl get hpa myhpa
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get hpa myhpa
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
myhpa   Deployment/myapp   16%/5%    1         10        10         5m17s
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

2. You will notice that the number of replicas has increased now.

3. Stop the proxy and the load generation commands running in the other two terminals by pressing `CTRL + C`.

Exercise 4: Create a Secret and update the deployment

Kubernetes Secrets lets you securely store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. Secrets are base64-encoded and can be used in your applications as environment variables or mounted as files.

Step 1: Create a Secret

Explore the content of the file `secret.yaml`:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: myapp-secret
type: Opaque
data:
  username: bXl1c2VybmFtZQ==
  password: bXlwYXNzd29yZA==
```

Explanation

This YAML file defines a secret named `mysecret` with two key-value pairs: `username` and `password`. The values are base64-encoded strings.

Step 2: Update the deployment to utilize the secret

Add the following lines at the end of `deployment.yaml`:

```
env:
- name: MYAPP_USERNAME
  valueFrom:
    secretKeyRef:
      name: myapp-secret
      key: username
- name: MYAPP_PASSWORD
  valueFrom:
    secretKeyRef:
      name: myapp-secret
      key: password
```

Explanation

- `name:` - Defines the environment variables: 'MYAPP_USERNAME' and 'MYAPP_PASSWORD', respectively.
- `valueFrom:` - Specifies that the value of the environment variable should be sourced from another location rather than being hardcoded.
- `secretKeyRef:` - Indicates that the value of the environment variable should come from a Kubernetes secret.
- `name: myapp-secret` - Specifies the name of the secret 'myapp-secret', from which to retrieve the value.
- `key:` - Specifies which key within the secret is to be used for the value of the 'MYAPP_USERNAME' and 'MYAPP_PASSWORD' environment variables, respectively.

With these updates, the `myapp` application can now read these environment variables to get the required credentials, making it more secure and flexible.

Step 3: Apply the secret and deployment

1. Apply the secret using the following command:

   ```
   kubectl apply -f secret.yaml
   ```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl apply -f secret.yaml
secret/mysecret created
```

2. Apply the updated deployment using the following command:

   ```
   kubectl apply -f deployment.yaml
   ```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl apply -f deployment.yaml
deployment.apps/myapp configured
```

Step 4: Verify the secret and deployment

You will now verify if the secret and the deployment using it have been applied.

1. Run the following command to retrieve the details of `myapp-secret` showing its name, type, and creation timestamp:

   ```
   kubectl get secret
   ```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl get secret
NAME           TYPE                                  DATA   AGE
dh             kubernetes.io/dockerconfigjson        1      12m
icr            kubernetes.io/dockerconfigjson        1      12m
myapp-secret   Opaque                                2      16s
```

2. Run the following command to show the status of the deployment, including information about replicas and available replicas.

   ```
   kubectl get deployment
   ```

```
theia@theiadocker-ksundararaja:/home/project/k8-scaling-and-secrets-mgmt$ kubectl get deployment
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
myapp   5/10    1            5           6m6s
theia@theiadocker-ksundararaja:/home/project/k8-scaling-and-secrets-mgmt$
```

Conclusion

In this lab, you began by building and deploying an application called `myapp` on Kubernetes.
Following this, you configured a Vertical Pod Autoscaler (VPA) to automatically adjust resource requests and limits for the `myapp` deployment.

Subsequently, you implemented a Horizontal Pod Autoscaler (HPA) to scale the number of replicas for the `myapp` deployment based on CPU utilization. Finally, you created a Secret and updated the `myapp` deployment to utilize it.

Author(s)

Nikesh Kumar