# Practice Lab: Autoscaling and Secrets Management
# 实践实验室：自动扩展和密钥管理



This practice lab is designed to provide hands-on experience with Kubernetes, focusing on vertical and horizontal pod autoscaling and secrets management.
本实践实验室旨在提供 Kubernetes 的动手经验，重点关注垂直和水平 Pod 自动扩展和密钥管理。

## Objectives 目标

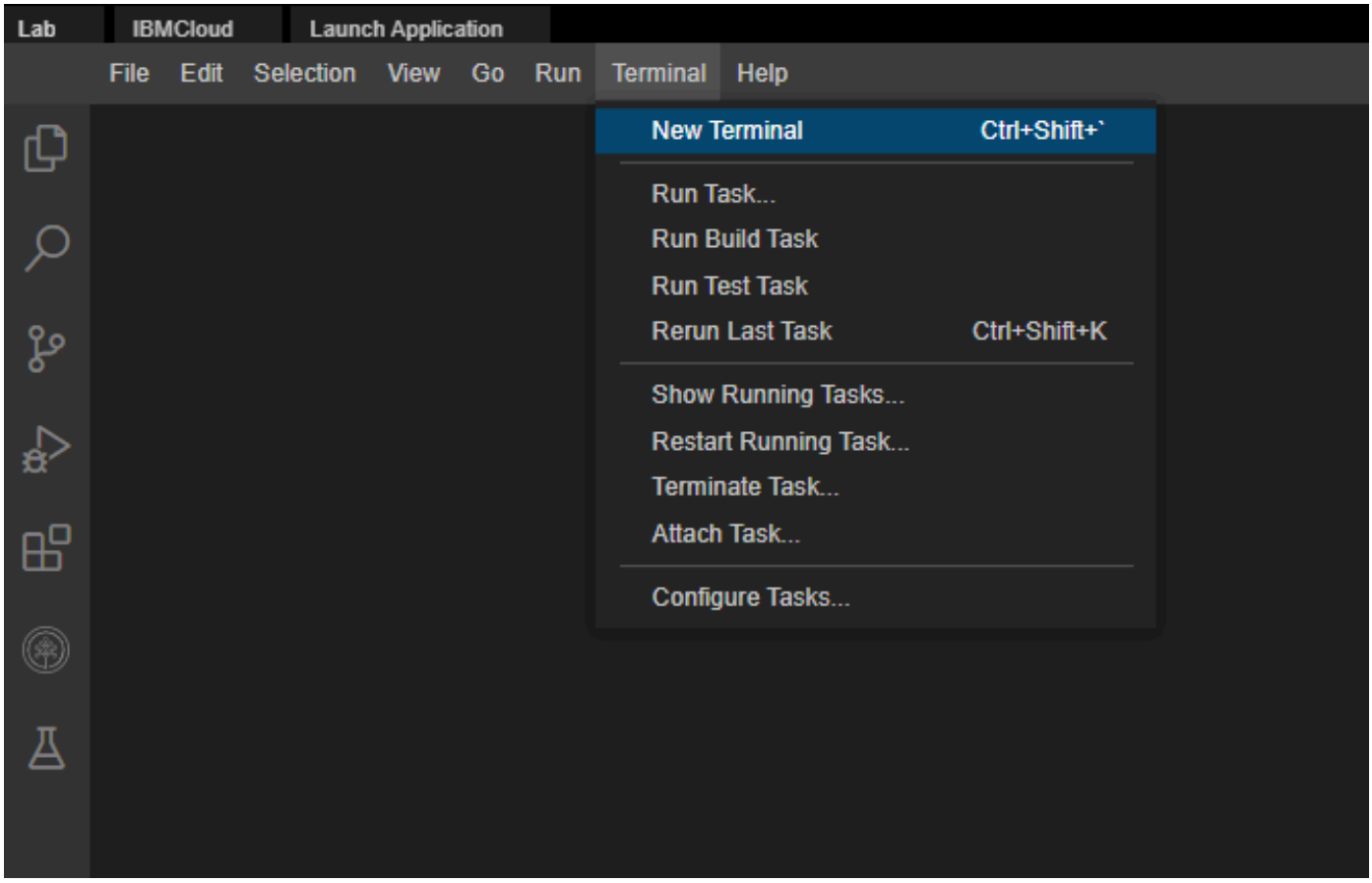
In this practice lab, you will:
在本实践实验室中，您将：

- Build and deploy an application to Kubernetes
  构建并部署应用程序到 Kubernetes
- Implement Vertical Pod Autoscaler (VPA) to adjust pod resource requests/limits
  实现垂直 Pod 自动伸缩（VPA）以调整 Pod 资源请求/限制
- Implement Horizontal Pod Autoscaler (HPA) to scale the number of pod replicas based on resource utilization
  实现水平 Pod 自动伸缩（HPA）根据资源利用率扩展 Pod 副本数量
- Create a Secret and update the deployment for using it
  创建一个 Secret 并更新部署以使用它

> **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.**
> **注意：请在一个会话中完成实验，不要中断，因为实验可能会进入离线模式并导致错误。如果在实验过程中遇到任何问题/错误，请从实验环境中注销。然后清除系统缓存和 Cookie，并尝试完成实验。**

# Setup the environment 设置环境

On the menu bar, click `Terminal` and select the `New Terminal` option from the drop-down menu.
在菜单栏中，点击 `Terminal` 并从下拉菜单中选择 `New Terminal` 选项。




Note: If the terminal is already open, please skip this step.
注意：如果终端已经打开，请跳过此步骤。

## Step 1: Verify kubectl version
## 步骤 1：验证 kubectl 版本

Before proceeding, ensure that you have kubectl installed and properly configured. To check the version of kubectl, run the following command:
在进行下一步之前，请确保已安装 kubectl 并正确配置。要检查 kubectl 的版本，请运行以下命令：

```
kubectl version
```

You should see the following output, although the versions may be different:
你应该看到以下输出，尽管版本可能不同：

```
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 --output=yaml|json to get
 the full version.
Client Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.6", GitCommit:"741c8db18a52787d734cbe4795f0b4ad860906d6", GitTreeState:"cle
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:"8db9c4804f1f37994e83aa532670006369716b8d", GitTreeState
:"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
## 步骤 2：克隆项目仓库

Clone the repository with the starter code to commence the project.
使用启动代码克隆仓库以开始项目。

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

# Exercise 1: Build and deploy an application to Kubernetes
# 练习 1：构建并部署应用程序到 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.
此存储库中的 Dockerfile 已经包含了应用程序的代码。您只需构建 Docker 镜像并将其推送到注册中心。

You will be giving the name `myapp` to your Kubernetes deployed application.
您将给您的 Kubernetes 部署的应用程序命名为 `myapp` 。

## Step 1: Build the Docker image
## 步骤 1：构建 Docker 镜像

1. Navigate to the project directory.
   导航到项目目录。

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

2. Export your namespace. 导出您的命名空间。

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

3. Build the Docker image. 构建 Docker 镜像。

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

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ docker build . -t us.icr.io/$MY_NAMESPACE/myapp:v1
[+] Building 21.9s (5/10)                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                 0.2s
 => => transferring dockerfile: 464B                                                                                 0.0s
 => [internal] load .dockerignore                                                                                    0.1s
 => => transferring context: 2B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:14                                                           0.7s
 => [auth] library/node:pull token for registry-1.docker.io                                                          0.0s
 => [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa    20.9s
 => => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa      0.0s
 => => sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa 776B / 776B                           0.0s
 => => sha256:1d12470fa662a2a5cb50378dcdc8ea228c1735747db410bbefb8e2d9144b5452 7.51kB / 7.51kB                       0.0s
 => => sha256:2cafa3fbb0b6529ee4726b4f599ec27ee557ea3dea7019182323b3779959927f 2.21kB / 2.21kB                       0.0s
 => => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca845ebc747976c408 50.45MB / 50.45MB                     [+] Building 22.0s
/10)                                                                                  docker:default53aeafeaa7e0671bb60008d
de101a38a045ff7bc656e3b0fbfc7c05cca5 7.86MB / 7.86MB
 => [internal] load build definition from Dockerfile                                                                 0.2s22
d995cccf12851a50820de03d34a17011dcbb9ac9fdf3a50c952cbb131 10.00MB / 10.00MB
 => => transferring dockerfile: 464B                                                                                 0.0se7
68b103d05fa8960e0f77951ff54b912b63429c34f5d6adfd09f5f9ee2 51.88MB / 51.88MB
 => [internal] load .dockerignore                                                                                    0.1sa8
894511ce28a05e2925a75e8a4acbd0634c39ad734fdfba8e23d1b1569 191.85MB / 191.85MB
 => => transferring context: 2B                                                                                      0.0s51
05deac0d99898e41b8ce60ebf250ebe1a31a0b03f613aec6bbc9b83d8 4.19kB / 4.19kB
 => [internal] load metadata for docker.io/library/node:14                                                           0.7sg
```

## Step 2: Push and list the image
## 步骤 2：推送并列出镜像

1. Push the tagged image to the IBM Cloud Container Registry.
   将标记的镜像推送至 IBM Cloud 容器注册中心。

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

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ docker push us.icr.io/$MY_NAMESPACE/myapp:v1
The push refers to repository [us.icr.io/sn-labs-ksundararaja/myapp]
d60490235730: Pushed
003de62710da: Pushed
306c0ccb34b4: Pushed
769169bec673: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Pushed
f8a91dd5fc84: Pushed
cb81227abde5: Pushed
e01a454893a9: Pushed
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushed
v1: digest: sha256:28d591aa82841c98be17f9d0f04bc9d56df6e3cce36b43320b64e5747cee2078 size: 3042
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$
```

2. List all the images available. You will see the newly created `myapp` image.
   列出所有可用的图像。您将看到新创建的 `myapp` 图像。

   ```
   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
## 步骤 3：部署您的应用程序

1. Open the `deployment.yaml` file located in the main project directory. It's content will be as follows:
   打开位于主项目目录中的 `deployment.yaml` 文件。其内容如下：

```
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.
   将 `<your SN labs namespace>` 替换为您的实际 SN 实验室的命名空间。

▼ Click here for the ways to get your namespace
点击此处获取获取命名空间的途径

1. Run the command `oc project`, and use the namespace mapped to your project name
   运行命令 `oc project` ，并使用与您的项目名称映射的命名空间

```
theia@theiadocker-ksundararaja:/home/project$ oc project
Using project "sn-labs-ksundararaja" from context named "ksundararaja-context" on server "https://c1.us-east.containers.cloud.ibm.com:22916".
```

2. Run the command `ibmcloud cr namespaces` and use the one which shows your sn-labs-username
   运行命令 `ibmcloud cr namespaces` 并使用显示您的 sn-labs-username 的那个

```
theia@theiadocker-ksundararaja:/home/project$ ibmcloud cr namespaces
Listing namespaces for account 'QuickLabs - IBM Skills Network' in registry 'us.icr.io'...

Namespace
sn-labs-ksundararaja
sn-labsassets

OK
theia@theiadocker-ksundararaja:/home/project$
```

3. Apply the deployment. 应用部署。

```
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.
   验证应用程序 Pod 是否正在运行且可访问。

```
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$
```

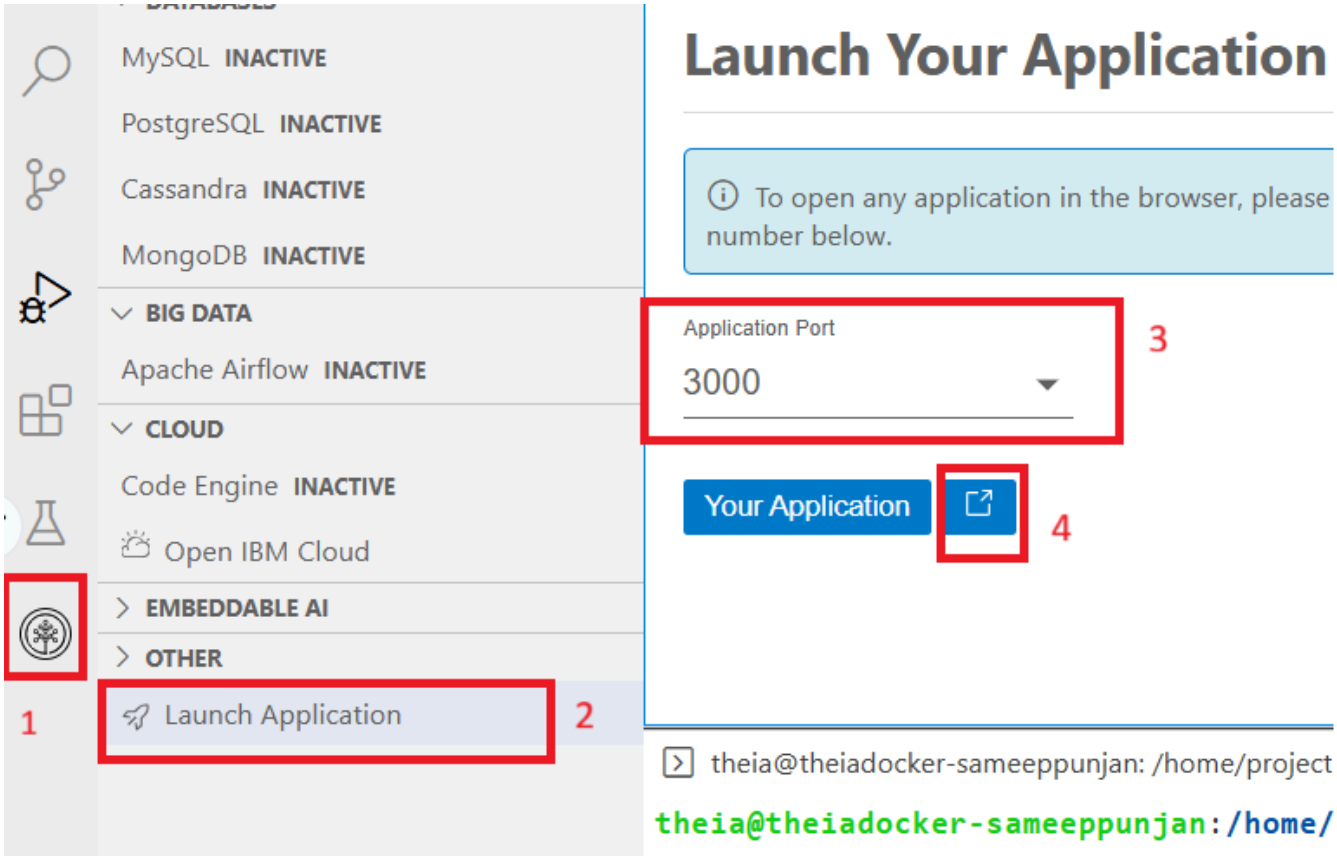
## Step 4: View the application output
## 步骤 4：查看应用程序输出

1. Start the application on port-forward:
   通过端口转发启动应用程序：

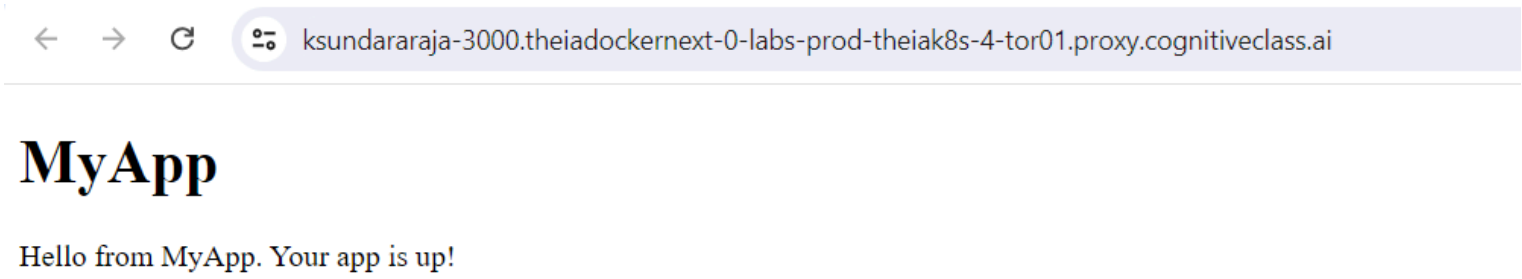
```
kubectl port-forward deployment.apps/myapp 3000:3000
```

```
theia@theiadocker-ksundararaja:/home/project/k8s-scaling-and-secrets-mgmt$ kubectl port-forward deployment.apps/myapp 3000:3000
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.
   在端口 `3000` 上启动应用程序以查看应用程序输出。
   1. Click on the Skills Network Toolbox Icon.
      点击技能网络工具箱图标。
   2. Click on Launch Application.
      点击启动应用程序。
   3. Enter the Port Number. 输入端口号。
   4. Click the launch button as shown in the screenshot.
      按照截图所示点击启动按钮。



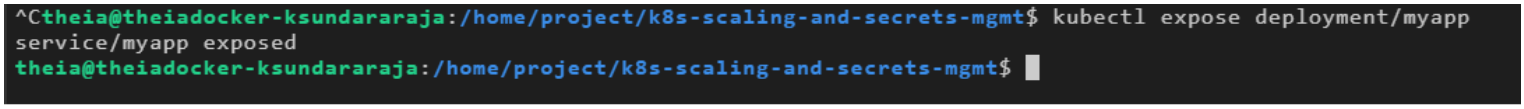

3. You should see the message `Hello from MyApp. Your app is up!.`
   你应该看到消息 `Hello from MyApp. Your app is up!` 。



4. Stop the server before proceeding further by pressing `CTRL + C`.
   在继续之前，通过按 `CTRL + C` 停止服务器。

5. Create a ClusterIP service for exposing the application to the internet:
   为将应用程序暴露给互联网创建一个 ClusterIP 服务：

   ```
   kubectl expose deployment/myapp
   ```




# Exercise 2: Implement Vertical Pod Autoscaler (VPA)
# 练习 2：实现垂直 Pod 自动缩放器（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.
垂直 Pod 自动伸缩器（VPA）帮助您管理 Pod 中运行的容器的资源请求和限制。它通过根据观察到的资源使用情况自动调整 CPU 和内存请求和限制，确保 Pod 拥有适当的资源以高效运行。

### Step 1: Create a VPA configuration
### 步骤 1：创建 VPA 配置

You will create a Vertical Pod Autoscaler (VPA) configuration to automatically adjust the resource requests and limits for the `myapp` deployment.
您将创建一个垂直 Pod 自动伸缩器（VPA）配置，以自动调整 `myapp` 部署的资源请求和限制。

Explore the `vpa.yaml` file, which has the following content:
探索 `vpa.yaml` 文件，其内容如下：

```
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 解释

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.
此 YAML 文件定义了 `myapp` 部署的 VPA 配置。 `updateMode: "Auto"` 设置表示 VPA 将根据观察到的使用情况自动更新此部署中 Pod 的资源请求和限制。

## Step 2: Apply the VPA
## 步骤 2：应用 VPA

Apply the VPA configuration using the following command:
使用以下命令应用 VPA 配置：

```
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
## 步骤 3：获取 VPA 的详细信息

1. Retrieve the created VPA:
   获取已创建的 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:
此输出显示：

- The VPA named `myvpa` is in Auto mode, recommending 25 milli-cores of CPU and 256 MB of memory for the pods it manages.
  名为 `myvpa` 的 VPA 处于自动模式，建议为它管理的 Pod 提供 25 毫核的 CPU 和 256 MB 的内存。
- It has been created 29 seconds ago and has been providing these recommendations since then.
  它是在 29 秒前创建的，并从那时起一直在提供这些推荐。

2. Retrieve the details and current running status of the VPA.
   检索 VPA 的详细信息和当前运行状态。

   ```
   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 解释

The output of `kubectl describe vpa myvpa` is providing recommendations for CPU and memory:
`kubectl describe vpa myvpa` 的输出正在提供 CPU 和内存的推荐：

| Resource 资源 | Definition 定义 |
|---|---|
| Lower Bound 下限 | Minimum resources the VPA recommends.<br>VPA 推荐的最少资源。 |
| Target 目标 | Optimal resources the VPA recommends.<br>VPA 推荐的最优资源。 |
| Uncapped Target 无上限目标 | Target without any predefined limits.<br>无任何预定义限制的目标。 |
| Upper Bound 上限 | Maximum resources the VPA recommends.<br>VPA 推荐的最大资源。 |

| Resource 资源 | CPU | Memory 内存 |
|---|---|---|
| Lower Bound 下限 | 25m | 256MiB (262144KiB) |
| Target 目标 | 25m 25 米 | 256MiB 256 兆字节 |
| Uncapped Target 无限制目标 | 25m 25 米 | 256MiB 256 兆字节 |
| Upper Bound 上界 | 671m 671 米 | 1.34GiB (1438074878KiB) |

These recommendations indicate that the VPA is functioning correctly and is providing target values based on observed usage.
这些建议表明 VPA 运行正常，并根据观察到的使用情况提供目标值。

# Exercise 3: Implement Horizontal Pod Autoscaler (HPA)
# 练习 3：实现水平 Pod 自动缩放器 (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.
水平 Pod 自动伸缩器（HPA）根据观察到的 CPU/内存利用率或其他自定义指标自动调整 pod 副本的数量。垂直 Pod 自动伸缩器（VPA）调整单个 pod 的资源请求和限制。然而，HPA 通过改变 pod 副本的数量来处理负载。

## Step 1: Create an HPA configuration
## 第一步：创建 HPA 配置

You will configure a Horizontal Pod Autoscaler (HPA) to scale the number of replicas of the `myapp` deployment based on CPU utilization.
您将配置一个水平 Pod 自动伸缩器（HPA），根据 CPU 利用率来伸缩 `myapp` 部署的副本数量。

Explore the `hpa.yaml` file, which has the following content:
探索 `hpa.yaml` 文件，其内容如下：

```
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 解释

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.
这个 YAML 文件定义了 `myapp` 部署的水平 Pod 自动缩放器（HPA）配置。HPA 将确保所有 Pod 的平均 CPU 利用率保持在 5%左右。如果利用率高于 5%，HPA 将增加副本数量；如果低于 5%，则会在 1 到 10 个副本的指定范围内减少副本数量。

## Step 2: Configure the HPA
## 步骤 2：配置 HPA

Apply the HPA configuration:
应用 HPA 配置：

```
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
### 步骤 3：验证 HPA

Obtain the status of the created HPA resource by executing the following command:
通过执行以下命令获取创建的 HPA 资源的状态：

```
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.
该命令提供有关当前和目标 CPU 利用率以及副本数量的详细信息。

### Step 4: Start the Kubernetes proxy
### 步骤 4：启动 Kubernetes 代理

Open another terminal and start the Kubernetes proxy:
打开另一个终端并启动 Kubernetes 代理：

```
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
### 步骤 5：发送垃圾信息并增加应用的负载

Open another new terminal and enter the below command to spam the app with multiple requests for increasing the load:
打开另一个新的终端，输入以下命令向应用发送多个请求以增加负载：

```
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.**
**在新终端中继续执行后续命令。**

## Step 6: Observe the effect of autoscaling
## 步骤 6：观察自动扩展的效果

1. Run the following command to observe the replicas increase in accordance with the autoscaling:
   运行以下命令以观察副本根据自动扩展增加：

   ```
   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.
   您将看到副本数量的增加，这表明您的应用程序已经自动扩展。

3. Terminate this command by pressing `CTRL + C`.
   通过按 `CTRL + C` 终止此命令。

### Step 7: Observe the details of the HPA
### 步骤 7：观察 HPA 的详细信息

1. Run the following command to observe the details of the horizontal pod autoscaler:
   运行以下命令以观察水平 Pod 自动缩放器的详细信息：

   ```
   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.
   您会注意到副本数量现在增加了。

3. Stop the proxy and the load generation commands running in the other two terminals by pressing `CTRL + C`.
   通过按 `CTRL + C` 停止运行在其他两个终端中的代理和负载生成命令。

# Exercise 4: Create a Secret and update the deployment
# 练习 4：创建一个密钥并更新部署

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.
Kubernetes Secrets 允许您安全地存储和管理敏感信息，例如密码、OAuth 令牌和 SSH 密钥。Secrets 是 base64 编码的，可以作为环境变量在您的应用程序中使用，或者作为文件挂载。

### Step 1: Create a Secret
### 第一步：创建一个 Secret

Explore the content of the file `secret.yaml`:
探索文件 `secret.yaml` 的内容：

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

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

## Explanation 解释

This YAML file defines a secret named `mysecret` with two key-value pairs: `username` and `password`. The values are base64-encoded strings.
这个 YAML 文件定义了一个名为 `mysecret` 的密钥，包含两个键值对： `username` 和 `password` 。这些值是 base64 编码的字符串。

## Step 2: Update the deployment to utilize the secret
## 步骤 2：更新部署以使用该密钥

Add the following lines at the end of `deployment.yaml`:
在 `deployment.yaml` 的末尾添加以下行：

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

## Explanation 解释

- `name:` - Defines the environment variables: 'MYAPP_USERNAME' and 'MYAPP_PASSWORD', respectively.
  `name:` - 定义环境变量：'MYAPP_USERNAME' 和 'MYAPP_PASSWORD'，分别对应。
- `valueFrom:` - Specifies that the value of the environment variable should be sourced from another location rather than being hardcoded.
  `valueFrom:` - 指定环境变量的值应从其他位置获取，而非硬编码。
- `secretKeyRef:` - Indicates that the value of the environment variable should come from a Kubernetes secret.
  `secretKeyRef:` - 指示环境变量的值应来自 Kubernetes 密钥。
- `name: myapp-secret` - Specifies the name of the secret 'myapp-secret', from which to retrieve the value.
  `name: myapp-secret` - 指定从名为 'myapp-secret' 的密钥中获取值的名称。
- `key:` - Specifies which key within the secret is to be used for the value of the 'MYAPP_USERNAME' and 'MYAPP_PASSWORD' environment variables, respectively.
  `key:` - 指定在密钥中用于'MYAPP_USERNAME'和'MYAPP_PASSWORD'环境变量的值的是哪个密钥。

With these updates, the `myapp` application can now read these environment variables to get the required credentials, making it more secure and flexible.
通过这些更新， `myapp` 应用程序现在可以读取这些环境变量以获取所需的凭证，使其更加安全和灵活。

## Step 3: Apply the secret and deployment
## 步骤 3：应用密钥和部署

1. Apply the secret using the following command:
   使用以下命令应用密钥：

   ```
   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:
   使用以下命令应用更新后的部署：

   ```
   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
## 步骤 4：验证密钥和部署

You will now verify if the secret and the deployment using it have been applied.
现在，您将验证密钥及其使用的部署是否已应用。

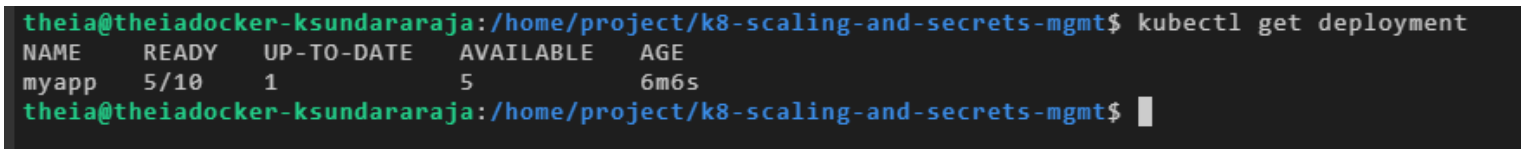
1. Run the following command to retrieve the details of `myapp-secret` showing its name, type, and creation timestamp:
   运行以下命令以检索 `myapp-secret` 的详细信息，显示其名称、类型和创建时间戳：

   ```
   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.
运行以下命令以显示部署状态，包括副本信息以及可用副本信息。

```
kubectl get deployment
```




# Conclusion 结论

In this lab, you began by building and deploying an application called `myapp` on Kubernetes.
在这个实验中，你首先在 Kubernetes 上构建并部署了一个名为 `myapp` 的应用程序。
Following this, you configured a Vertical Pod Autoscaler (VPA) to automatically adjust resource requests and limits for the `myapp` deployment.
随后，你配置了一个垂直 Pod 自动伸缩器（VPA），用于自动调整 `myapp` 部署的资源请求和限制。
Subsequently, you implemented a Horizontal Pod Autoscaler (HPA) to scale the number of replicas for the `myapp` deployment based on CPU utilization.
接着，你实现了一个水平 Pod 自动伸缩器（HPA），根据 CPU 利用率来扩展 `myapp` 部署的副本数量。
Finally, you created a Secret and updated the `myapp` deployment to utilize it.
最后，你创建了一个 Secret，并更新了 `myapp` 部署以使用它。

## Author(s) 作者

Nikesh Kumar 尼基什·库马尔

© IBM Corporation. All rights reserved.