How to Mount a Secrets to a Pod as Volumes

Secrets

Les Secrets dans Kubernetes sont des objets qui stockent des données sensibles, comme les mots de passe, les tokens OAuth, ou les clés SSH, pour être utilisés par les pods. Ils permettent de gérer ces informations confidentielles de manière sécurisée sans les exposer dans la configuration de l'application.

Volume

Un volume dans Kubernetes est un espace de stockage attaché aux pods, permettant de conserver des données au-delà de la durée de vie de ces pods. Il offre une solution pour le stockage persistant des données, nécessaire pour des applications comme les bases de données. Les volumes peuvent être hébergés localement sur l'hôte, ou provenir de sources externes comme des disques en réseau.

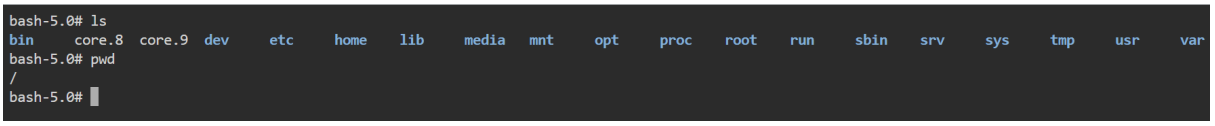
Task Details

1. Launching Lab Environment.
2. SSH into the EC2 instance through the Whiz terminal
3. Create a secret as YML file
4. Create a Deployment
5. Mount the secret into the deployment using the config file
6. Verify the access of the secret within the pod.

Lab Steps

Task 1: Launching Lab Environment

1. Launch the lab environment by clicking on the **Start Lab** button.
2. Please wait until the lab environment is provisioned. It will take around 2 minutes to provision the lab environment which contains the single node Kubernetes cluster. The unique Username, Public IP address, and the Password are displayed on the right corner of the screen.
3. Alongside the EC2 instance, the Whizlabs terminal is also being provisioned.



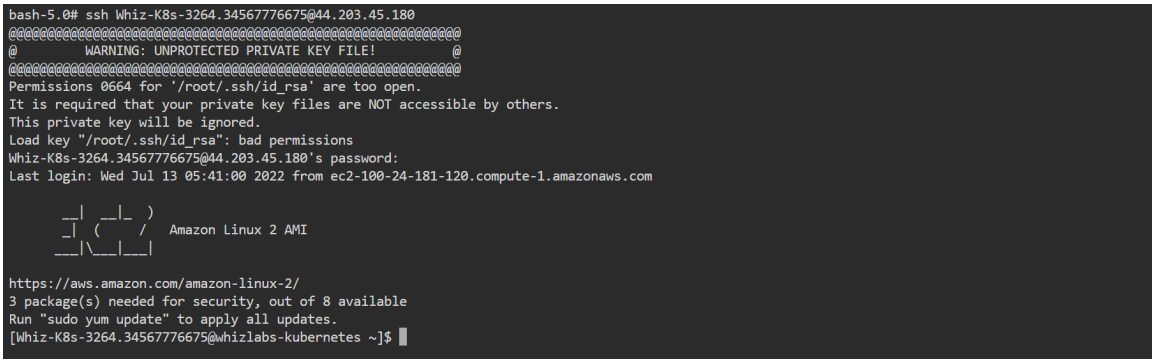
Note: If you face any issues, please go through FAQs and Troubleshooting for Labs.

Task 2 : SSH into the EC2 instance through the Whiz terminal:

1. Copy the IP address and the Password for SSH to the EC2 instance configured with the Kubernetes cluster.
2. Run the following command to SSH into the system:

```
ssh username@ip_address
```

1. Replace the unique username and the IP address with the details of your own lab environment.
2. Type yes if prompted for the SSH permission for the first time. Then, Insert the password in the prompt as it is.



3. Now switch the user to root with the command below:

sudo su root

---

1. This ec2 instance is configured with Kubernetes packages and all prerequisites are fulfilled. This is a single node cluster and we will use the same node as the client node to communicate and control the containers.

Task 3: Create a secret as a YML file

- First, let's create a YML file using vim command:

```
vim creds.yml
```

- Now paste the following code to create a secret resource where we will store a username and a password which will eventually be stored inside a secret in encoded form.

```
apiVersion: v1
kind: Secret
metadata:
  name: credentials
stringData:
  password: w#izl@b$
  username: whizlabs
type: Opaque
```

- Now let's create the resource using the kubectl command

```
kubectl apply -f creds.yml
```

- Verify the resource is created using the get query with the kubectl command

```
kubectl apply -f creds.yml
```

- To retrieve only the username, we can run the below addon command

```
kubectl get secret credentials -o jsonpath="{.data.username}"
```

Task 4: Create a Deployment

1. In this step, we will create a deployment wherein we will specify the volume configuration and mention the secret resource details in it so that the deployment and the underlying pod has access to the secret and the data stored in it. Create a file using vim command

```
vim busybox.yml
```

1. Paste the following code in the created file and save it. Here, we are taking a busybox container image that is capable of performing tiny executables. With the keyword *volume,* we are going to specify the secret created in the previous stage.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
spec:
  replicas: 1
  selector:
    matchLabels:
      name: busybox
  template:
    metadata:
      labels:
        name: busybox
    spec:
      containers:
```

```
        - name: my-container
          image: busybox
          command: ["/bin/sh","-c"]
          args: ["while true; do sleep 9999999; done"]
      volumes:
      - name: credentials-volume
        secret:
          secretName: credentials
```

1. Now, apply the changes and provision the deployment using the kubctl command

kubectl apply -f busybox.yml

1. Cross-verify that the pod is up and running by the following command. Make sure the pod state is RUNNING before proceeding to the next step.

kubectl get pods

Task 5: Mount the secret into the deployment using the config file

1. Now append the below configuration in the **busybox.yml** after the volume file which will mount the volume to a directory inside a container. Till now, we have attached the volume to the pod but the container will not be able to access the data until a native directory is mounted with it. Using the volumeMounts keyword block, we are attaching the created secret as a volume to the mentioned directory at mountPath of the container. As an outcome, we will be able to retrieve the data stored inside the secret from the container. Also, the keyword readOnly will restrict the user to edit the content of the mounted volume.

The resultant busybox.yml file will look like this.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      name: busybox
  template:
    metadata:
      labels:
        name: busybox
    spec:
      containers:
        - name: my-container
          image: busybox
          command: ["/bin/sh","-c"]
          args: ["while true; do sleep 9999999; done"]
          volumeMounts:
          - mountPath: /mnt/credentials
            name: credentials-volume
            readOnly: true
      volumes:
      - name: credentials-volume
        secret:
          secretName: credentials
```

```
=== this part is for reference only ===
volumeMounts:
- mountPath: /mnt/credentials
  name: credentials-volume
  readOnly: true
```

Now, apply the changes of the updated file and add the mount configuration to the existing deployment with the following command:

```
kubectl apply -f busybox.yml
```

Now the deployment container will be able to access the data through /mnt/credentials directory.

Task 6: Verify the access of the secret within the pod.

First, set the pod as an environmental variable by using the command below so that we don’t have to remember the ID now and then.

```
POD=`kubectl get pod -l name=busybox -o jsonpath="{.items[0].
```

1. Now, run the command below to read the data inside the /mnt/credentials directory inside the container. Here, the data will be called through the mounted secret. Since we haven’t passed the credentials anywhere while launching the pod, the output data indicates that it is being called from the secret.

```
kubectl exec $POD -it -- cat /mnt/credentials/username
```

In the same way, we can call the password value by replacing the username in the command above.

This is how we can manage sensitive data separately and natively within the Kubernetes scope and independent from the pod lifecycle.

Completion and Conclusion

1. You have successfully provisioned a secret resource.
2. You have successfully logged into the cluster through the Whiz terminal and launched a busybox deployment inside it.
3. You have successfully mounted the secret as a volume and read the data from outside the container.

End Lab

1. Exit from the terminal SSH session.
2. You have successfully completed the lab.
3. Once you have completed the steps, click on **End Lab** from your Whizlabs dashboard.