Demo Project: Deploy our web application in K8s cluster from private Docker registry

This guide demonstrates how to deploy a web application in a Kubernetes cluster while securely pulling its Docker image from a private Docker registry. It provides two options for creating a Kubernetes Secret for Docker registry credentials and includes deployment files for both.

Step 1: Create a Kubernetes Secret for Docker Registry Credentials

Option 1: Create Secret Using Docker Login and Config.json

1. **Log in to Docker Registry**:
    - Obtain the authentication token for AWS Elastic Container Registry (ECR):

      `aws ecr get-login-password`
    - SSH into the Minikube instance: `minikube ssh`
    - Log in to the private Docker registry:

`docker login --username AWS -p <paste-authentication-token> 038462748802.dkr.ecr.us-east-1.amazonaws.com`

2. **Verify Config File**:
   - Confirm the `config.json` file generated by Docker: `cat .docker/config.json`
3. **Copy Config File from Minikube**:
   - Copy the `.docker/config.json` file from Minikube to your host machine:

`minikube cp minikube:/home/docker/.docker/config.json /users/USERNAME/.docker/config.json`

example: `minikube cp minikube:/home/docker/.docker/config.json /home/eb/.docker/config.json`

   - Confirm: `cat .docker/config.json`
4. **Base64 Encode the Config File (Alternative to Step 5):**:
   - Encode the `.docker/config.json` file to Base64 format:

     `cat .docker/config.json | base64`
   - Create `docker-secret.yaml` and paste the Base64 content within the file.
5. **Create the Kubernetes Secret**:
   - Use the `config.json` file to create the secret (this will create the Base64 content for you):

```
kubectl create secret generic my-registry-key \
--from-file=.dockerconfigjson=.docker/config.json \
--type=kubernetes.io/dockerconfigjson
```

`kubectl create secret generic my-registry-key --from-file=.dockerconfigjson=.docker/config.json --type=kubernetes.io/dockerconfigjson`

6. **Verify the Secret**:
   - Check that the secret was created: `kubectl get secret`
   - To view detailed content including the detail of the base 64 encoded contents:

     `kubectl get secret -o yaml`

Option 2: Create Secret with Login Credentials in One Step

1. **Obtain the authentication token for AWS Elastic Container Registry (ECR):**

   `aws ecr get-login-password`

2. **Create the Secret**:
   - Use `kubectl` to log in and create the secret in a single command:

```
kubectl create secret docker-registry my-registry-key-two \
--docker-server=https://private-repo \
--docker-username=user \
--docker-password=<paste-authentication-token>
```

`kubectl create secret docker-registry my-registry-key-two --docker-server=https://private-repo --docker-username=user --docker-password=<paste-authentication-token>`

Example:

```
kubectl create secret docker-registry my-registry-key-two \
--docker-server=https://038462748802.dkr.ecr.us-east-1.amazonaws.com \
--docker-username=AWS \
--docker-password=<paste-authentication-token>
```

3. **Verify the Secret**:
   - Confirm the secret was created successfully: `kubectl get secret`

When to Use Each Option:

- **Option 1**: Use when multiple private registries need to be accessed, as you can log in to all registries and store their tokens in a single `config.json` file.
- **Option 2**: Convenient for creating a secret for one private registry in a single step.

Step 2: Create deployment component

1. **Verify which secret you will use:** `kubectl get secret`
2. **Create deployment component:**

→ For Option 1 you would create the deployment file like this:

File: `my-app-deployment.yaml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      imagePullSecrets:
      - name: my-registry-key
      containers:
      - name: my-app
        image: 038462748802.dkr.ecr.us-east-1.amazonaws.com/my-app:1.1
        imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

→ For Option 2 you would create the deployment file like this:

File: `my-app-deployment.yaml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-two
  labels:
    app: my-app-two
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app-two
  template:
    metadata:
      labels:
        app: my-app-two
    spec:
      imagePullSecrets:
      - name: my-registry-key-two
      containers:
      - name: my-app-two
        image: 038462748802.dkr.ecr.us-east-1.amazonaws.com/my-app:1.1
        imagePullPolicy: Always
        - containerPort: 3000
```

- **Note: For both cases:** `imagePullSecrets` This references to the secret created earlier, allowing the pod to authenticate to the private registry and pull the image.
- `imagePullPolicy: Always`: This ensures that the latest image version is always pulled, bypassing any cached images.

Step 3: Deploy web application image from our private Docker registry in K8s cluster

1. **Deploy the application:** `kubectl apply -f my-app-deployment.yaml`
2. **Verify the Deployment**:
   - Check pod status: `kubectl get pod`

Important Notes

Namespace Scope:

- Ensure that the secret is created in the same namespace as the deployment.

Summary

1. **Create Secret**:
   - Use Option 1 (with `config.json`) for multiple registries.
   - Use Option 2 (with login credentials) for a single registry.
2. **Configure Deployment**:
   - Reference the secret using `imagePullSecrets` in the deployment.
3. **Deploy Application**:
   - Apply the deployment and verify the pods are running correctly.
