Lab 10 – Deploying a Predictive Model with KServe on Minikube

Experiment Objective

The objective of this lab is to gain hands-on experience with KServe for deploying and serving machine learning models on Kubernetes. In this experiment, KServe is installed and configured on a local Kubernetes cluster using Minikube. A predictive model is deployed through a KServe InferenceService. The deployment process is monitored, common issues are examined, and the inference service is accessed from outside the cluster. Finally, inference requests are sent to the deployed model and the prediction results are interpreted.

Experimental Environment

This experiment is performed on a local Kubernetes cluster created using Minikube. KServe is installed following the official KServe documentation. All cluster operations are executed using kubectl. A pre-trained sklearn model provided by the KServe examples is used as the predictive model in this experiment.

Installing and Configuring KServe on Minikube

The experiment begins by starting a local Kubernetes cluster using Minikube. After Minikube is started, the cluster status is verified to ensure that the Kubernetes control plane is running correctly.

The cert-manager component is required by KServe to manage certificates. Cert-manager is installed into the cluster, and its pods are checked to confirm that they are in the Running state. Once cert-manager is ready, KServe is installed by applying the official KServe deployment manifests. The KServe controller, webhook, and related components are verified by checking the pods in the kserve namespace.

At this stage, all KServe-related pods should be running successfully, indicating that KServe has been installed and configured correctly on the Minikube cluster.

Deploying a Predictive Model Using InferenceService

After KServe is successfully installed, a predictive model is deployed using a KServe InferenceService. In this experiment, a simple sklearn-based model is used. The model is defined using an InferenceService custom resource, which specifies the predictor type and the storage location of the model artifacts.

An InferenceService YAML file is created that defines the model name, predictor framework, and storage URI. The InferenceService is then applied to the Kubernetes cluster using kubectl. After applying the configuration, the InferenceService resource is created, and KServe begins pulling the model image and initializing the inference container.

The status of the InferenceService is checked to confirm that the resource has been created successfully.

Screenshot note:
This section should include screenshots of the InferenceService YAML configuration and the output showing the InferenceService resource being created.

Monitoring Service Status and Debugging

Once the InferenceService is deployed, its status is monitored to determine whether the service becomes ready. The readiness status is checked using kubectl commands, which display the current state of the InferenceService. During the initial deployment, the service may show a non-ready state while the model container is starting.

If the service does not become ready, logs from the predictor pod are examined to identify common issues such as image pull errors, insufficient resources, or incorrect model storage paths. By checking pod logs and pod status, potential deployment problems can be identified and resolved.

After the model container is successfully initialized, the InferenceService status changes to Ready, indicating that the predictive model is now available to serve inference requests.

Screenshot note:
This part should include screenshots showing the InferenceService status transitioning to Ready, as well as example logs or pod status outputs used for debugging.

Accessing the InferenceService from Outside the Cluster

After the InferenceService becomes ready, it needs to be accessed from outside the Kubernetes cluster. In this experiment, Minikube’s networking features are used to

expose the service externally. The service URL is obtained using kubectl commands, and port-forwarding or Minikube service access is used to enable external access.

Once the external access method is configured, the inference service endpoint becomes reachable from the local machine.

Screenshot note:
This section should include screenshots showing how the service URL is obtained and how the service is exposed for external access.

Sending Inference Requests and Interpreting Results

With the inference service accessible, inference requests are sent to the deployed model using HTTP requests. A sample JSON payload is constructed based on the expected input format of the sklearn model. The request is sent to the inference endpoint using a command-line HTTP client.

The service responds with prediction results, which are returned in JSON format. The response includes the predicted class or value generated by the model. The prediction results are examined to verify that the model is functioning correctly and that the inference service is responding as expected.

Screenshot note:
This part should include screenshots showing the inference request command and the corresponding prediction response returned by the service.

Experimental Results and Discussion

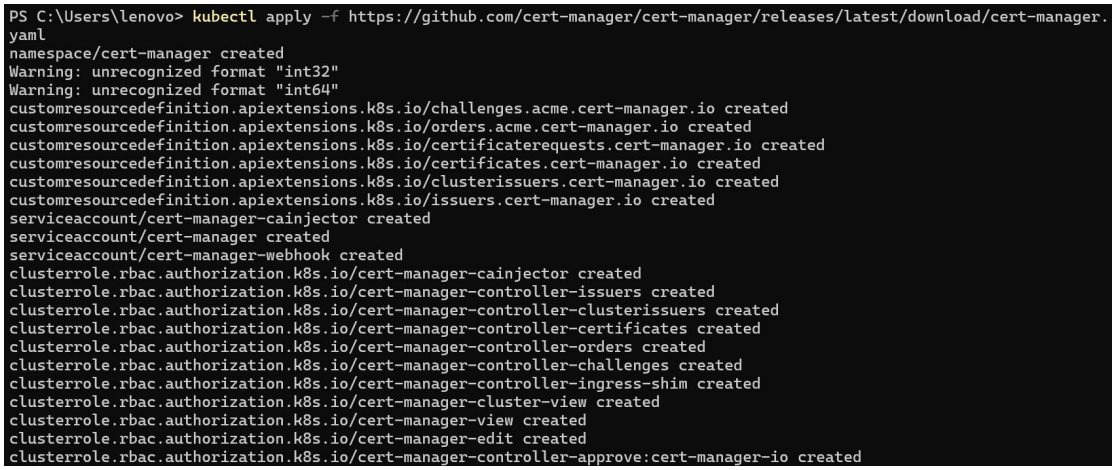
Through this experiment, KServe successfully demonstrates its ability to deploy and manage predictive models on Kubernetes. The InferenceService abstraction simplifies the deployment process by handling container creation, scaling, and networking automatically. The experiment also highlights common troubleshooting steps when deploying machine learning models in a Kubernetes environment.

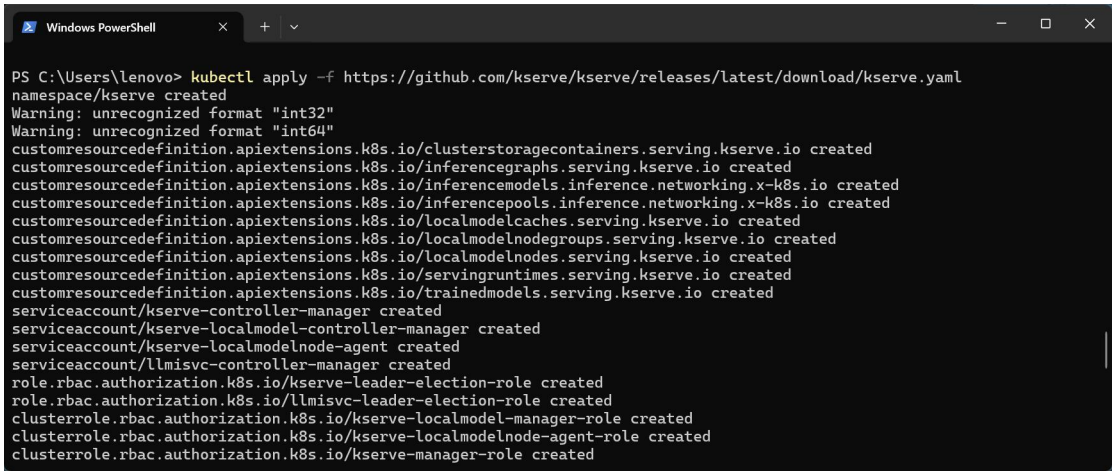
The ability to access the inference service externally and obtain prediction results confirms that the deployed model is operational and correctly integrated with KServe.

Conclusion

This lab provides practical experience with deploying predictive models using KServe on a local Kubernetes cluster. By installing KServe, deploying an InferenceService, monitoring its status, exposing the service, and sending inference requests, a complete end-to-end model serving workflow is achieved. The experiment demonstrates how KServe simplifies machine learning model serving in cloud-native environments.

Screenshots:





D: > ! sklearn-isvc.yaml

```yaml
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: sklearn-iris
spec:
  predictor:
    sklearn:
      storageUri: gs://kserve-samples/models/sklearn/iris
```

```
PS D:\> kubectl get inferenceservice
```

```
PS D:\> kubectl port-forward svc/sklearn-iris-predictor-default 8080:80
```