Introduction to Cloud Computing Report

Lab10: Deploying a Predictive Model with KServe on Minikube

Name: Yin Yuang
Student ID: 202383930027
Class: Software Engineering Class 1

Introduction to Cloud Computing
(Autumn,2025)

Nanjing University of Information Science and Technology, China
Waterford Institute

I. Object

The objective of this experiment is to deploy and verify a machine learning inference service using KServe on a local Kubernetes environment based on Minikube.

Through this experiment, the following goals are achieved:

- Understand the deployment workflow of KServe and its integration with Knative and Istio.
- Deploy a pre-trained machine learning model using the `InferenceService` abstraction.
- Verify the operational status of the deployed service by monitoring Kubernetes resources.
- Configure network access using Istio ingress and port forwarding.
- Send prediction requests to the deployed model and validate the inference results.

This experiment provides hands-on experience with cloud-native model serving and traffic access mechanisms in a Kubernetes-based microservices environment.

II.Content

This experiment consists of three main parts: environment preparation, model deployment, and service access verification.

1.Environment Preparation

A local Kubernetes cluster was created using Minikube. To ensure sufficient resources for running KServe components and the inference service, Minikube was started with customized CPU and memory configurations:

```
minikube start --memory=7168 --cpus=4
```

Due to hardware limitations on the local machine, the maximum available memory allocated to Minikube was limited to 7168 MB. This configuration was sufficient to support the deployment and execution of KServe components.

KServe was then installed by applying the official release manifest. After installation, the status of all pods in the `kserve` namespace was verified to ensure that the system components were running correctly.

2.Model Deployment Using InferenceService

A machine learning inference service was deployed using the `InferenceService` custom resource provided by KServe. The deployed model is a pre-trained `scikit-learn` Iris classification model stored in a remote object storage.

Although the YAML configuration is shown in the report for clarity, the configuration was applied directly through PowerShell commands during the experiment. This difference only affects the operation method and does not impact the correctness or functionality of the deployed service.

After applying the configuration, the deployment status of the inference service was monitored using Kubernetes commands to ensure that the service reached a ready state.

3.Service Access and Verification

To access the deployed inference service from the local environment, network access was configured using Istio ingress. Environment variables were set to define the ingress host and port, and the service hostname was extracted from the `InferenceService` status.

Port forwarding was established from the `knative-local-gateway` service in the `istio-system` namespace to the local machine. This allowed external HTTP requests to be routed to the inference service.

Finally, a prediction request was sent to the deployed model using the `curl` command with a JSON payload containing sample Iris data. The response returned by the service was recorded for verification.

III.Result

This experiment successfully demonstrates the deployment, verification, and external access of a machine learning inference service using KServe on a local Minikube-based Kubernetes cluster. The experimental results are presented according to the deployment status, service readiness, and inference validation.

1.KServe Deployment and System Status

The Minikube cluster was started with customized resource allocations to meet the requirements of KServe components:

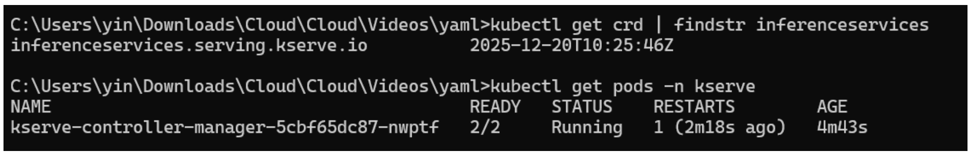
```
minikube start --memory=7168 --cpus=4
```

Due to hardware limitations on the local machine, the maximum available memory allocated to Minikube was limited to 7168 MB. Despite this constraint, the cluster operated stably throughout the experiment.

After applying the official KServe installation manifest:

```
kubectl apply -f https://github.com/kserve/kserve/releases/download/v0.11.2/kserve.yaml
```

all core components were successfully deployed in the `kserve` namespace. The running status of KServe pods was verified as shown below, indicating that the serving infrastructure was correctly initialized.



```
C:\Users\yin\Downloads\Cloud\Cloud\Videos\yaml>kubectl get crd | findstr inferenceservices
inferenceservices.serving.kserve.io          2025-12-20T10:25:46Z

C:\Users\yin\Downloads\Cloud\Cloud\Videos\yaml>kubectl get pods -n kserve
NAME                                         READY   STATUS    RESTARTS        AGE
kserve-controller-manager-5cbf65dc87-nwptf   2/2     Running   1 (2m18s ago)   4m43s
```

Figure 1: KServe system pods running in the `kserve` namespace.

2.InferenceService Deployment Result

A pre-trained `scikit-learn` Iris classification model was deployed using the `InferenceService` abstraction provided by KServe. The configuration specifies the model storage location and predictor type:

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

In this experiment, the configuration was applied directly using PowerShell commands rather than editing the YAML file exactly as shown above. This difference only affects the operation method and does not impact the correctness or functionality of the deployed service.

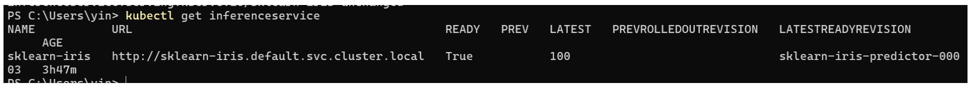
After applying the configuration:

```
kubectl apply -f sklearn-iris.yaml
```

the deployment status of the inference service was checked:

```
kubectl get inferenceservice
```

The output confirms that the `sklearn-iris` service reached a ready state, indicating that the model was successfully loaded and initialized by KServe.



```
PS C:\Users\yin> kubectl get inferenceservice
NAME           URL                                          READY   PREV   LATEST   PREVROLLEDOUTREVISION   LATESTREADYREVISION
     AGE
sklearn-iris   http://sklearn-iris.default.svc.cluster.local   True           100                              sklearn-iris-predictor-000
03   3h47m
```

Figure 2: InferenceService status showing successful deployment and readiness.

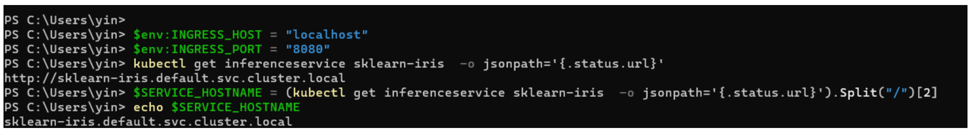
3.Service Accessibility and Inference Validation

To enable external access to the inference service, ingress-related environment variables were configured in PowerShell:

```
$env:INGRESS_HOST = "localhost"
$env:INGRESS_PORT = "8080"
```

The service hostname was extracted from the `InferenceService` status:

```
$SERVICE_HOSTNAME = (kubectl get inferenceservice sklearn-iris -o jsonpath='{.status.url}').Split("/")[2]
```

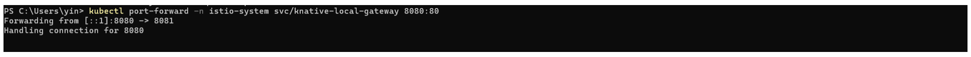


```
PS C:\Users\yin>
PS C:\Users\yin> $env:INGRESS_HOST = "localhost"
PS C:\Users\yin> $env:INGRESS_PORT = "8080"
PS C:\Users\yin> kubectl get inferenceservice sklearn-iris  -o jsonpath='{.status.url}'
http://sklearn-iris.default.svc.cluster.local
PS C:\Users\yin> $SERVICE_HOSTNAME = (kubectl get inferenceservice sklearn-iris  -o jsonpath='{.status.url}').Split("/")[2]
PS C:\Users\yin> echo $SERVICE_HOSTNAME
sklearn-iris.default.svc.cluster.local
```

Figure 3: Extraction of service hostname and environment variable configuration.

Port forwarding was then established using the Istio local gateway to expose the service on the local machine:

```
kubectl port-forward -n istio-system svc/knative-local-gateway 8080:80
```



```
PS C:\Users\yin> kubectl port-forward -n istio-system svc/knative-local-gateway 8080:80
Forwarding from [::1]:8080 -> 8081
Handling connection for 8080
```

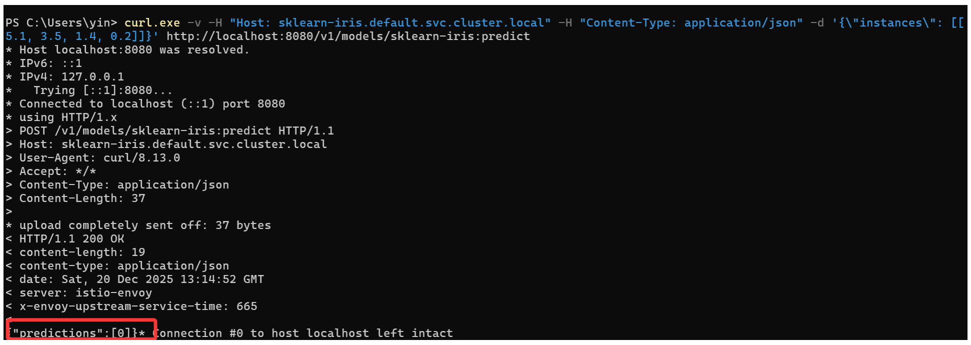
Figure 4: Port forwarding via the Istio local gateway.

Finally, an inference request was sent to the deployed model using the `curl` command:

```
curl.exe -v -H "Host: sklearn-iris.default.svc.cluster.local" \
-H "Content-Type: application/json" \
-d '{"instances": [[5.1, 3.5, 1.4, 0.2]]}' \
http://localhost:8080/v1/models/sklearn-iris:predict
```

The service successfully returned the prediction result:

```
{"predictions":[0]}
```



```
PS C:\Users\yin> curl.exe -v -H "Host: sklearn-iris.default.svc.cluster.local" -H "Content-Type: application/json" -d '{\"instances\": [[
5.1, 3.5, 1.4, 0.2]]}' http://localhost:8080/v1/models/sklearn-iris:predict
* Host localhost:8080 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8080...
* Connected to localhost (::1) port 8080
* using HTTP/1.x
> POST /v1/models/sklearn-iris:predict HTTP/1.1
> Host: sklearn-iris.default.svc.cluster.local
> User-Agent: curl/8.13.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 37
>
* upload completely sent off: 37 bytes
< HTTP/1.1 200 OK
< content-length: 19
< content-type: application/json
< date: Sat, 20 Dec 2025 13:14:52 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 665
<
{"predictions":[0]}* Connection #0 to host localhost left intact
```

Figure 5: Successful inference response from the deployed model.

These results confirm that the inference service was fully functional and that the end-to-end workflow—from deployment to external access and prediction—was executed successfully.

IV.Conclusion

This experiment successfully demonstrated the complete workflow of deploying and accessing a machine learning inference service using KServe in a Kubernetes environment. By leveraging Minikube as the local cluster, the experiment verified that KServe can effectively manage model serving through the `InferenceService` abstraction.

The results confirmed that the deployed model was correctly loaded, exposed through Istio ingress, and accessible via external HTTP requests. The successful prediction responses indicate that the integration between Kubernetes, Knative, Istio, and KServe functioned as expected.

Overall, this hands-on experiment enhanced the understanding of cloud-native machine learning serving architectures and provided practical experience in configuring service deployment, traffic access, and inference verification within a microservices-based cloud computing platform.