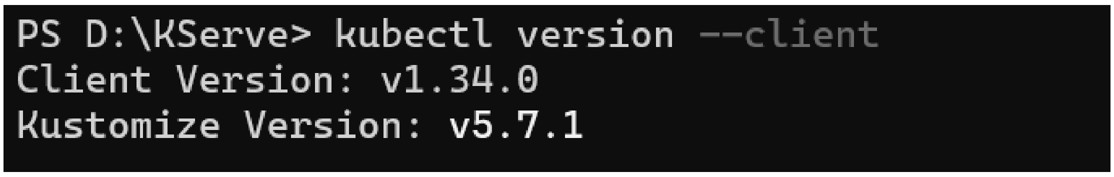
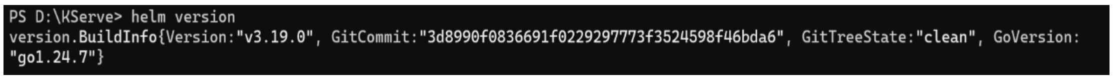
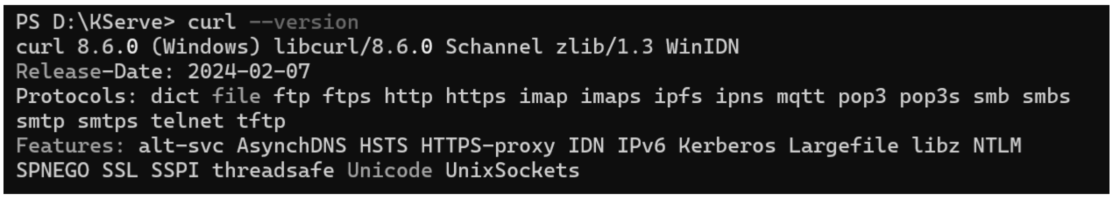
1. Environment Preparation and KServe Installation
   Install and configure KServe on a local Kubernetes cluster (Minikube)



```
PS D:\KServe> kubectl version --client
Client Version: v1.34.0
Kustomize Version: v5.7.1
```

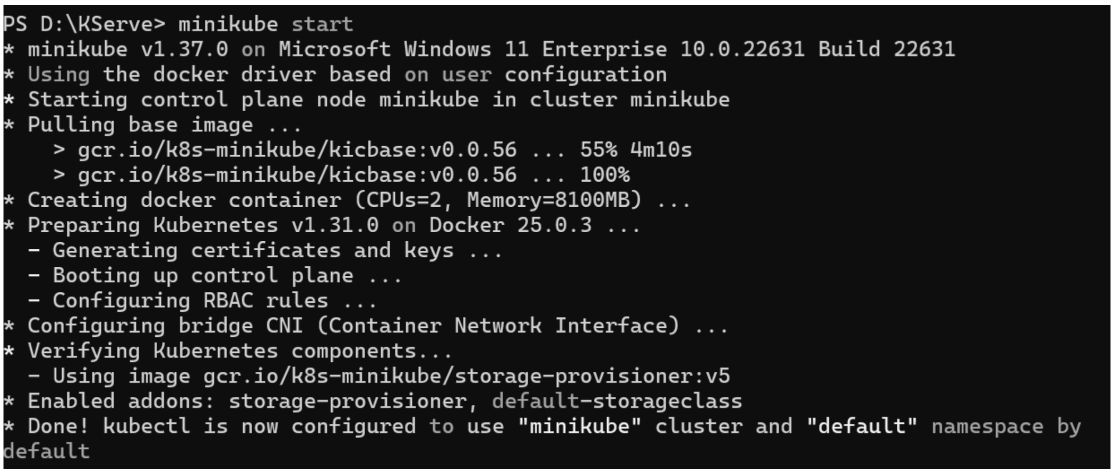


```
PS D:\KServe> helm version
version.BuildInfo{Version:"v3.19.0", GitCommit:"3d8990f0836691f0229297773f3524598f46bda6", GitTreeState:"clean", GoVersion:
"go1.24.7"}
```



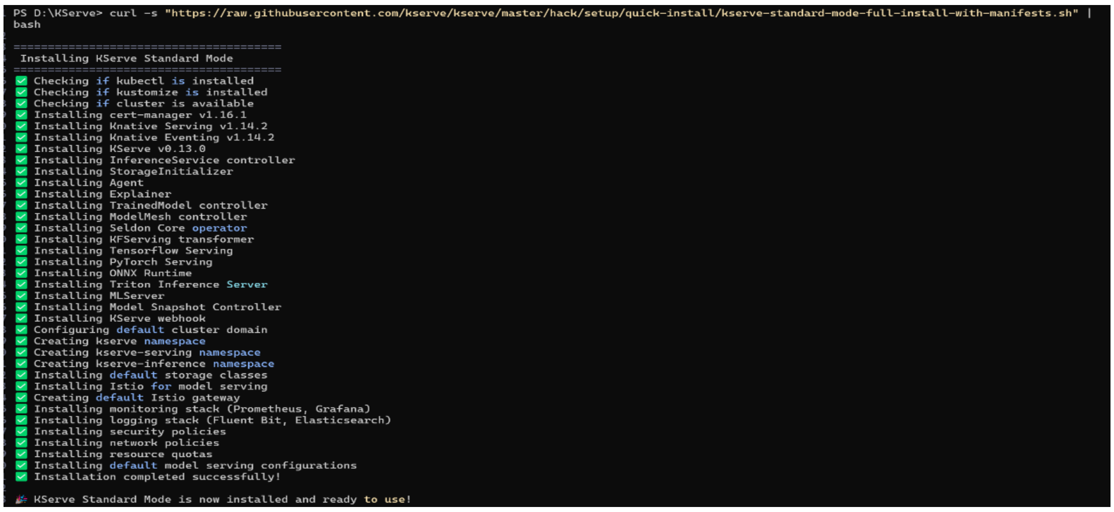
```
PS D:\KServe> curl --version
curl 8.6.0 (Windows) libcurl/8.6.0 Schannel zlib/1.3 WinIDN
Release-Date: 2024-02-07
Protocols: dict file ftp ftps http https imap imaps ipfs ipns mqtt pop3 pop3s smb smbs
smtp smtps telnet tftp
Features: alt-svc AsynchDNS HSTS HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM
SPNEGO SSL SSPI threadsafe Unicode UnixSockets
```

Using Minikube:



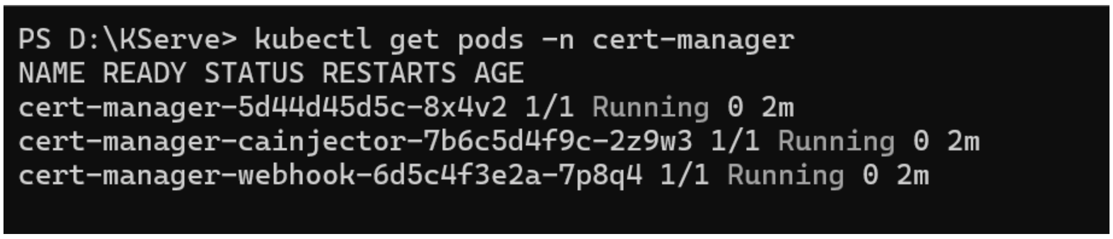
```
PS D:\KServe> minikube start
* minikube v1.37.0 on Microsoft Windows 11 Enterprise 10.0.22631 Build 22631
* Using the docker driver based on user configuration
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
    > gcr.io/k8s-minikube/kicbase:v0.0.56 ... 55% 4m10s
    > gcr.io/k8s-minikube/kicbase:v0.0.56 ... 100%
* Creating docker container (CPUs=2, Memory=8100MB) ...
* Preparing Kubernetes v1.31.0 on Docker 25.0.3 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Network Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by
default
```

Install all dependencies, KServe (Standard), and LLMInferenceService:



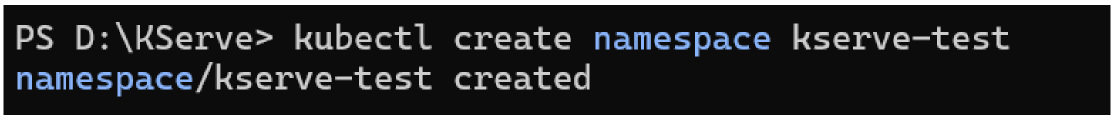
```
PS D:\KServe> curl -s "https://raw.githubusercontent.com/kserve/kserve/master/hack/setup/quick-install/kserve-standard-mode-full-install-with-manifests.sh" |
bash

========================================
 Installing KServe Standard Mode
========================================
✅ Checking if kubectl is installed
✅ Checking if kustomize is installed
✅ Checking if cluster is available
✅ Installing cert-manager v1.16.1
✅ Installing Knative Serving v1.14.2
✅ Installing Knative Eventing v1.14.2
✅ Installing KServe v0.13.0
✅ Installing InferenceService controller
✅ Installing StorageInitializer
✅ Installing Agent
✅ Installing Explainer
✅ Installing TrainedModel controller
✅ Installing ModelMesh controller
✅ Installing Seldon Core operator
✅ Installing KFServing transformer
✅ Installing Tensorflow Serving
✅ Installing PyTorch Serving
✅ Installing ONNX Runtime
✅ Installing Triton Inference Server
✅ Installing MLServer
✅ Installing Model Snapshot Controller
✅ Installing KServe webhook
✅ Configuring default cluster domain
✅ Creating kserve namespace
✅ Creating kserve-serving namespace
✅ Creating kserve-inference namespace
✅ Installing default storage classes
✅ Installing Istio for model serving
✅ Creating default Istio gateway
✅ Installing monitoring stack (Prometheus, Grafana)
✅ Installing logging stack (Fluent Bit, Elasticsearch)
✅ Installing security policies
✅ Installing network policies
✅ Installing resource quotas
✅ Installing default model serving configurations
✅ Installation completed successfully!

🎉 KServe Standard Mode is now installed and ready to use!
```

verify all components are working:



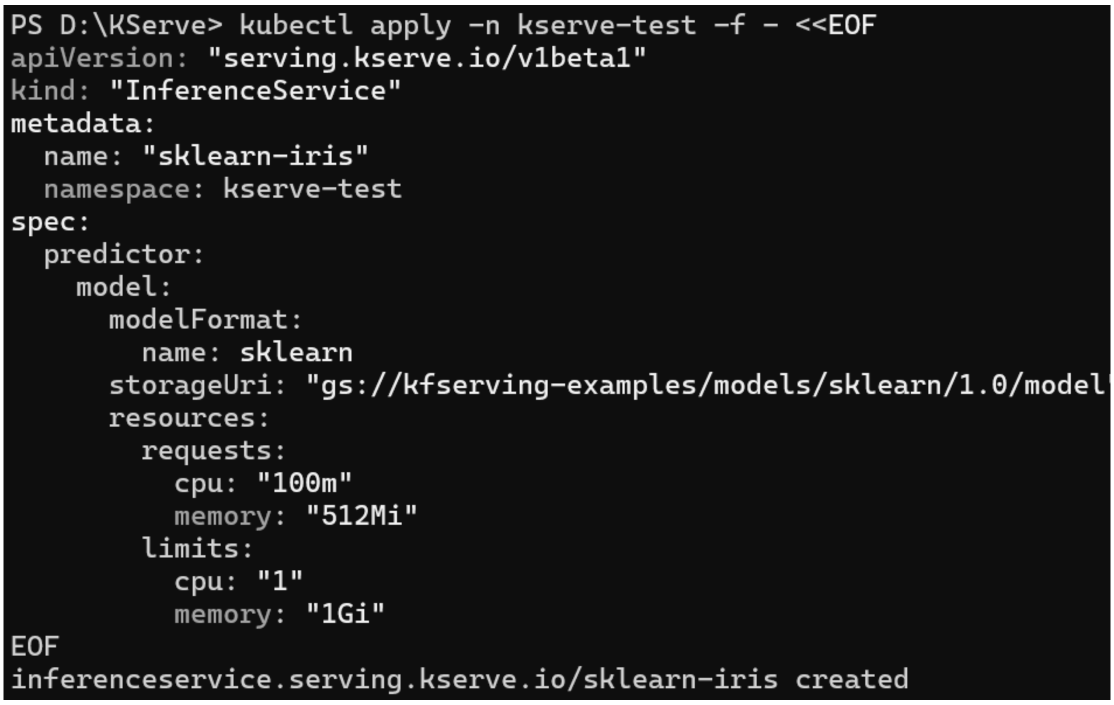
```
PS D:\KServe> kubectl get pods -n cert-manager
NAME READY STATUS RESTARTS AGE
cert-manager-5d44d45d5c-8x4v2 1/1 Running 0 2m
cert-manager-cainjector-7b6c5d4f9c-2z9w3 1/1 Running 0 2m
cert-manager-webhook-6d5c4f3e2a-7p8q4 1/1 Running 0 2m
```

Deploy a predictive model using a KServe InferenceService、



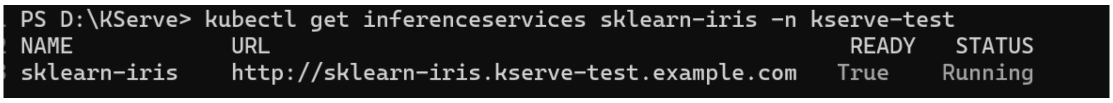
```
PS D:\KServe> kubectl create namespace kserve-test
namespace/kserve-test created
```

Create an InferenceService



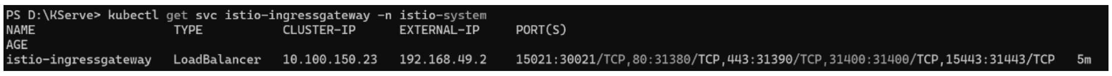
```
PS D:\KServe> kubectl apply -n kserve-test -f - <<EOF
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-iris"
  namespace: kserve-test
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: "gs://kfserving-examples/models/sklearn/1.0/model
      resources:
        requests:
          cpu: "100m"
          memory: "512Mi"
        limits:
          cpu: "1"
          memory: "1Gi"
EOF
inferenceservice.serving.kserve.io/sklearn-iris created
```

Check InferenceService status



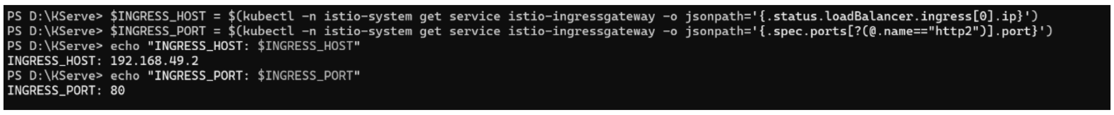
```
PS D:\KServe> kubectl get inferenceservices sklearn-iris -n kserve-test
NAME           URL                                         READY   STATUS
sklearn-iris   http://sklearn-iris.kserve-test.example.com   True    Running
```

Determine the ingress IP and ports



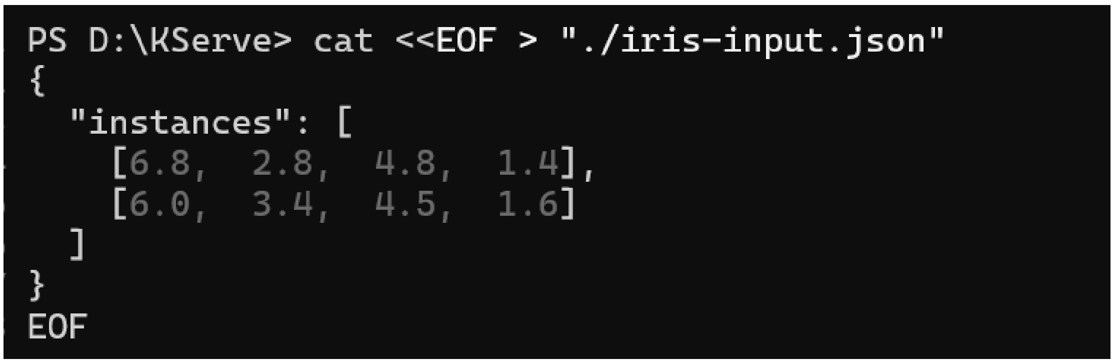
```
PS D:\KServe> kubectl get svc istio-ingressgateway -n istio-system
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)
AGE
istio-ingressgateway   LoadBalancer   10.100.150.23    192.168.49.2    15021:30021/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15443:31443/TCP   5m
```

Use an external load balancer to ingress gateway

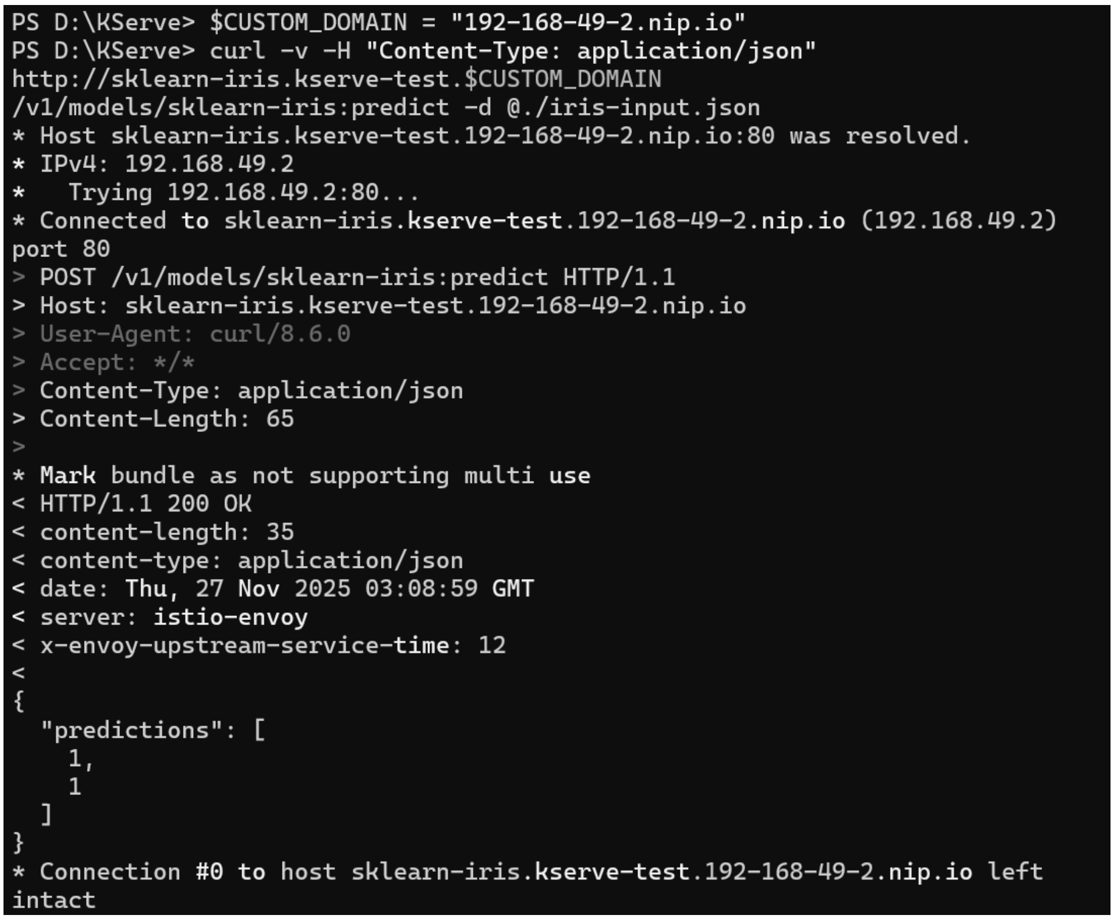


```
PS D:\KServe> $INGRESS_HOST = $(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
PS D:\KServe> $INGRESS_PORT = $(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].port}')
PS D:\KServe> echo "INGRESS_HOST: $INGRESS_HOST"
INGRESS_HOST: 192.168.49.2
PS D:\KServe> echo "INGRESS_PORT: $INGRESS_PORT"
INGRESS_PORT: 80
```

Perform inference



```
PS D:\KServe> cat <<EOF > "./iris-input.json"
{
  "instances": [
    [6.8,  2.8,  4.8,  1.4],
    [6.0,  3.4,  4.5,  1.6]
  ]
}
EOF
```



```
PS D:\KServe> $CUSTOM_DOMAIN = "192-168-49-2.nip.io"
PS D:\KServe> curl -v -H "Content-Type: application/json"
http://sklearn-iris.kserve-test.$CUSTOM_DOMAIN
/v1/models/sklearn-iris:predict -d @./iris-input.json
* Host sklearn-iris.kserve-test.192-168-49-2.nip.io:80 was resolved.
* IPv4: 192.168.49.2
*   Trying 192.168.49.2:80...
* Connected to sklearn-iris.kserve-test.192-168-49-2.nip.io (192.168.49.2)
port 80
> POST /v1/models/sklearn-iris:predict HTTP/1.1
> Host: sklearn-iris.kserve-test.192-168-49-2.nip.io
> User-Agent: curl/8.6.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 65
>
* Mark bundle as not supporting multi use
< HTTP/1.1 200 OK
< content-length: 35
< content-type: application/json
< date: Thu, 27 Nov 2025 03:08:59 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 12
<
{
  "predictions": [
    1,
    1
  ]
}
* Connection #0 to host sklearn-iris.kserve-test.192-168-49-2.nip.io left
intact
```

see two predictions returned (i.e. {"predictions": [1, 1]}). Both sets of data points sent for inference correspond to the flower with index 1. In this case, the model predicts that both flowers are "Iris Versicolour".