Introduction to Cloud Computing Report

Lab 7: Istio Installation, Gateway Configuration and Traffic Management

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. Objective

The objective of this experiment is to deploy and configure the Istio service mesh on a Kubernetes cluster, understand its core architecture, and verify traffic management and observability capabilities using the Bookinfo sample application.

Through this experiment, the following objectives are achieved:

- Install the Istio control plane and ingress gateway components in a Kubernetes environment.
- Deploy a microservices-based application into the Istio service mesh with automatic sidecar injection.
- Configure Gateway API resources to expose services to external traffic.
- Diagnose and resolve common gateway deployment issues in a local Kubernetes environment.
- Verify service-to-service communication, traffic routing, and observability within the service mesh.

II. Content

This experiment focuses on deploying and validating the Istio service mesh in a Kubernetes environment using the official Bookinfo sample application.

The main experimental contents include:

- Installing Istio control plane components and enabling the Ingress Gateway using the demo profile.
- Deploying a microservices-based application (Bookinfo) into the Istio service mesh with automatic sidecar injection.
- Configuring Kubernetes Gateway API resources to expose services to external traffic.
- Diagnosing and resolving Gateway deployment issues caused by service type limitations in a local Kubernetes environment.
- Implementing Istio traffic management rules to split traffic between multiple service versions (Reviews 50/50).
- Verifying service mesh observability through the Kiali dashboard, including topology visualization and traffic flow analysis.

Through these experimental steps, core Istio capabilities such as traffic routing, gateway management, and service mesh visualization were systematically verified.

III. Results

1. Downloading and Preparing Istio

The experiment started by downloading the official Istio release package for Windows. After extraction, the directory structure was inspected to confirm the presence of the `bin` directory, which contains the `istioctl` executable.

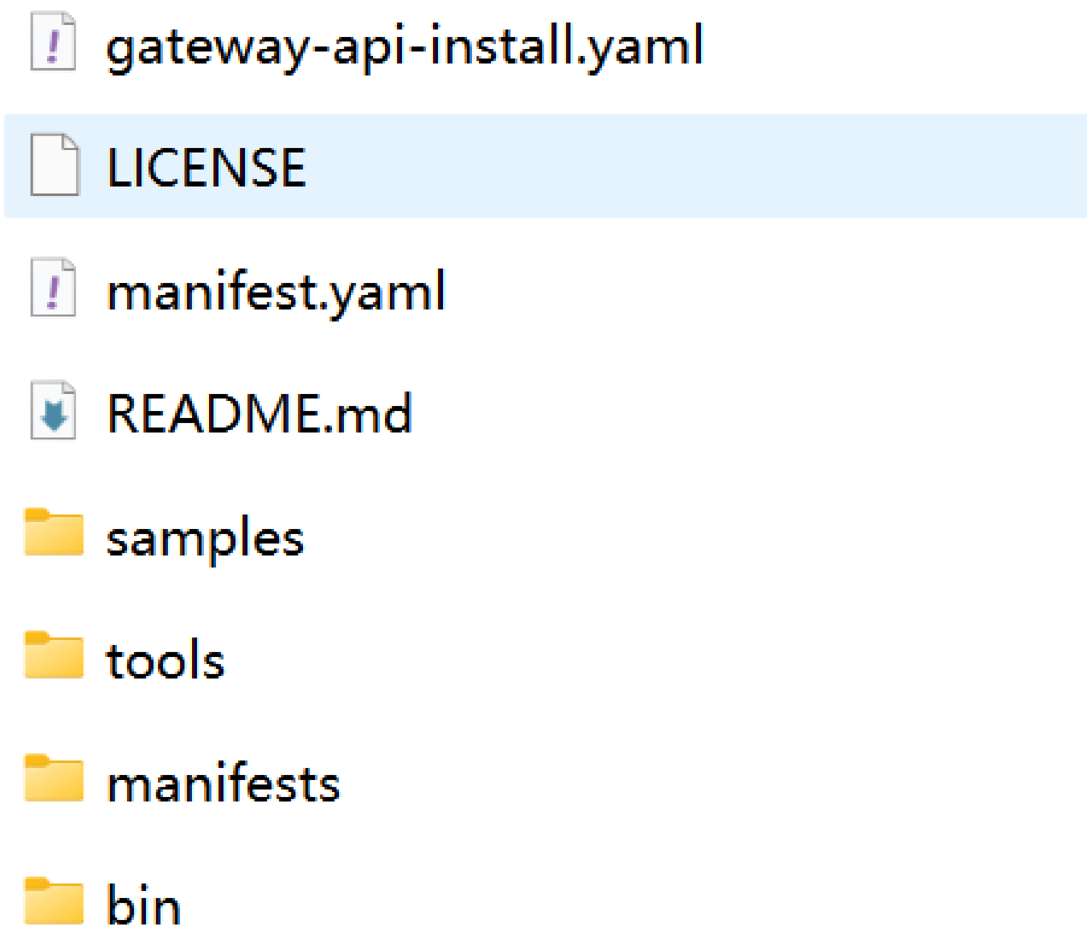


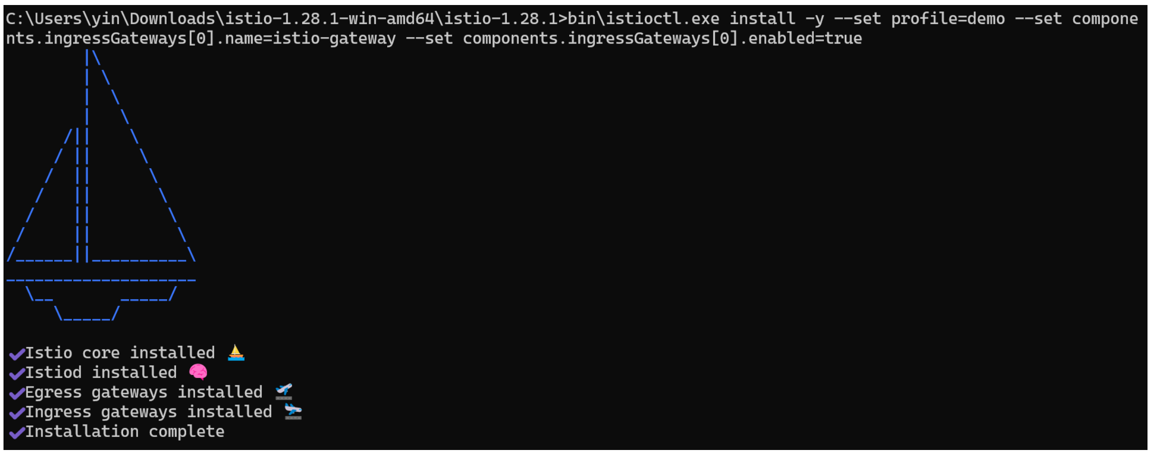
Figure 1: Istio installation directory structure on Windows

2. Installing Istio Control Plane and Gateway

Istio was installed using the demo profile with an explicitly enabled Ingress Gateway component. The following command was executed:

```
bin\istioctl.exe install -y --set profile=demo
--set components.ingressGateways[0].name=istio-gateway
--set components.ingressGateways[0].enabled=true
```

The installation output confirmed successful deployment of Istio core components, Istiod, and Gateway resources.

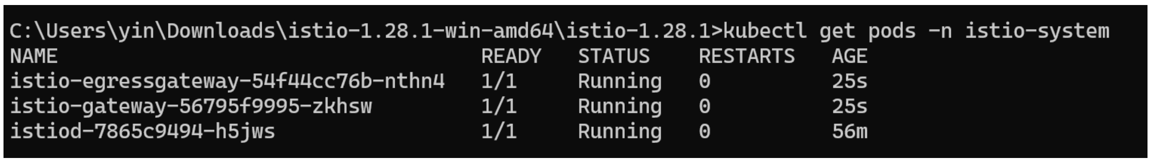


```
C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>bin\istioctl.exe install -y --set profile=demo --set compone
nts.ingressGateways[0].name=istio-gateway --set components.ingressGateways[0].enabled=true
        |\
        | \
        |  \
        |   \
      /||    \
     / ||     \
    /  ||      \
   /   ||       \
  /    ||        \
 /     ||         \
/______||__________\
____________________
  \__       _____/
     \_____/

✔Istio core installed ⛵
✔Istiod installed 🧠
✔Egress gateways installed 🛫
✔Ingress gateways installed 🛬
✔Installation complete
```

Figure 2: Successful installation of Istio control plane and gateways

Verification of running Pods in the `istio-system` namespace showed that all components were in the `Running` state.



```
C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl get pods -n istio-system
NAME                                    READY   STATUS    RESTARTS   AGE
istio-egressgateway-54f44cc76b-nthn4    1/1     Running   0          25s
istio-gateway-56795f9995-zkhsw          1/1     Running   0          25s
istiod-7865c9494-h5jws                  1/1     Running   0          56m
```

Figure 3: Istio system Pods running successfully

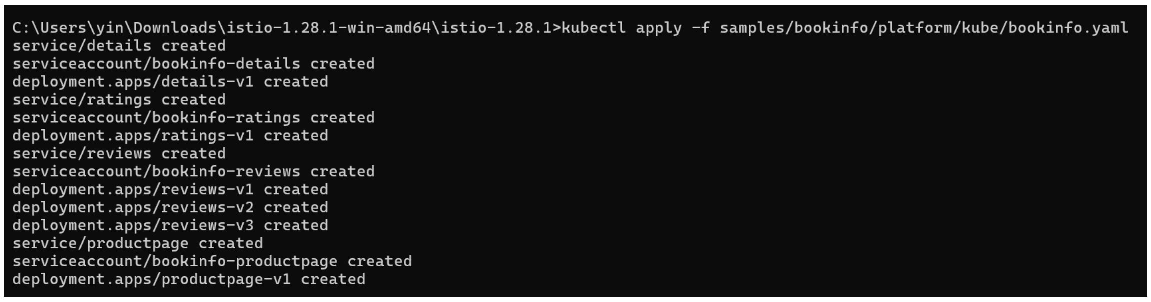
3. Deploying the Bookinfo Sample Application

The Bookinfo microservices application was deployed into the Kubernetes cluster using the provided manifest files.

```
kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

Automatic sidecar injection was enabled for the default namespace to ensure that each Pod received an Envoy proxy. The successful injection was verified by inspecting the running Pods, where each Bookinfo Pod contained both the application container and the `istio-proxy` sidecar.

Once deployed, all Bookinfo Pods reached the `Running` state.



```
C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
deployment.apps/productpage-v1 created
```

Figure 4: Bookinfo application Pods running successfully with Istio sidecar proxies injected

4. Configuring Gateway API and Exposing the Application

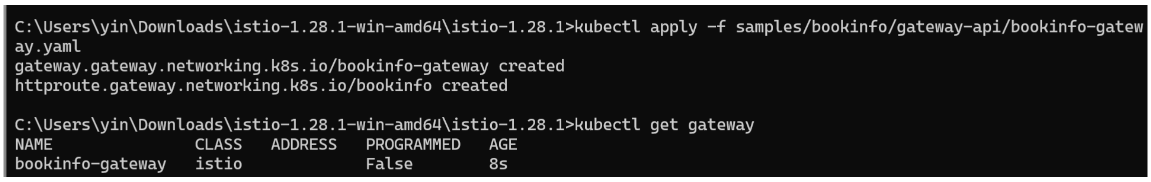
To expose the Bookinfo application externally, the Gateway API resources provided by Istio were applied.

```
kubectl apply -f samples/bookinfo/gateway-api/bookinfo-gateway.yaml
```

After creation, the Gateway resource was listed:

```
kubectl get gateway
```

At this stage, the Gateway status showed `PROGRAMMED=False`, indicating that the Gateway had not yet been fully configured.



```
C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl apply -f samples/bookinfo/gateway-api/bookinfo-gatew
ay.yaml
gateway.gateway.networking.k8s.io/bookinfo-gateway created
httproute.gateway.networking.k8s.io/bookinfo created

C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl get gateway
NAME               CLASS   ADDRESS   PROGRAMMED   AGE
bookinfo-gateway   istio             False        8s
```

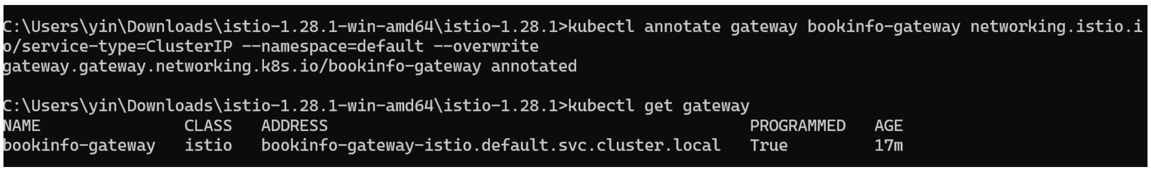
Figure 5: Gateway initially showing PROGRAMMED=False

The issue occurred because, in a local Kubernetes environment, the default Gateway Service type (`LoadBalancer`) cannot obtain an external IP address. As a result, the Gateway controller was unable to mark the Gateway as fully programmed.

To resolve this issue, the Gateway was explicitly annotated to use the `ClusterIP` service type:

```
kubectl annotate gateway bookinfo-gateway networking.istio.io/service-type=ClusterIP
--namespace=default --overwrite
```

After applying this annotation, the Gateway status was updated successfully.



```
:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl annotate gateway bookinfo-gateway networking.istio.i
o/service-type=ClusterIP --namespace=default --overwrite
gateway.gateway.networking.k8s.io/bookinfo-gateway annotated

:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl get gateway
NAME               CLASS   ADDRESS                                            PROGRAMMED   AGE
bookinfo-gateway   istio   bookinfo-gateway-istio.default.svc.cluster.local   True         17m
```

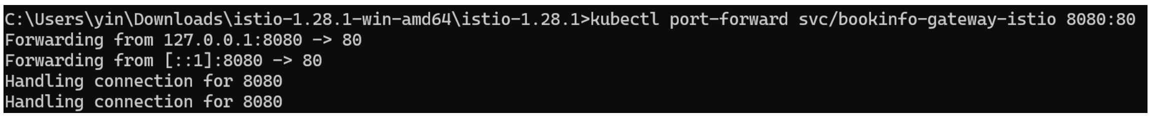
Figure 6: Gateway successfully programmed after annotation

5. Verifying External Access

Since the Gateway Service type was set to `ClusterIP`, port forwarding was used to expose the application locally.

```
kubectl port-forward svc/bookinfo-gateway-istio 8080:80
```

Accessing the URL `http://localhost:8080/productpage` successfully returned the Bookinfo application page, confirming that external traffic was correctly routed through the Istio Gateway.



```
C:\Users\yin\Downloads\istio-1.28.1-win-amd64\istio-1.28.1>kubectl port-forward svc/bookinfo-gateway-istio 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
Handling connection for 8080
```

Figure 7: Bookinfo application accessed through Istio Gateway

6. Traffic Management with Istio (Reviews 50/50)

One of the key features of Istio is its fine-grained traffic management capability. In this experiment, traffic splitting was configured for the `reviews` service so that requests would be distributed evenly between two different service versions.

Specifically:

- `reviews-v1`: No star ratings displayed
- `reviews-v3`: Red star ratings displayed

This visual difference allows traffic routing behavior to be directly observed from the browser.

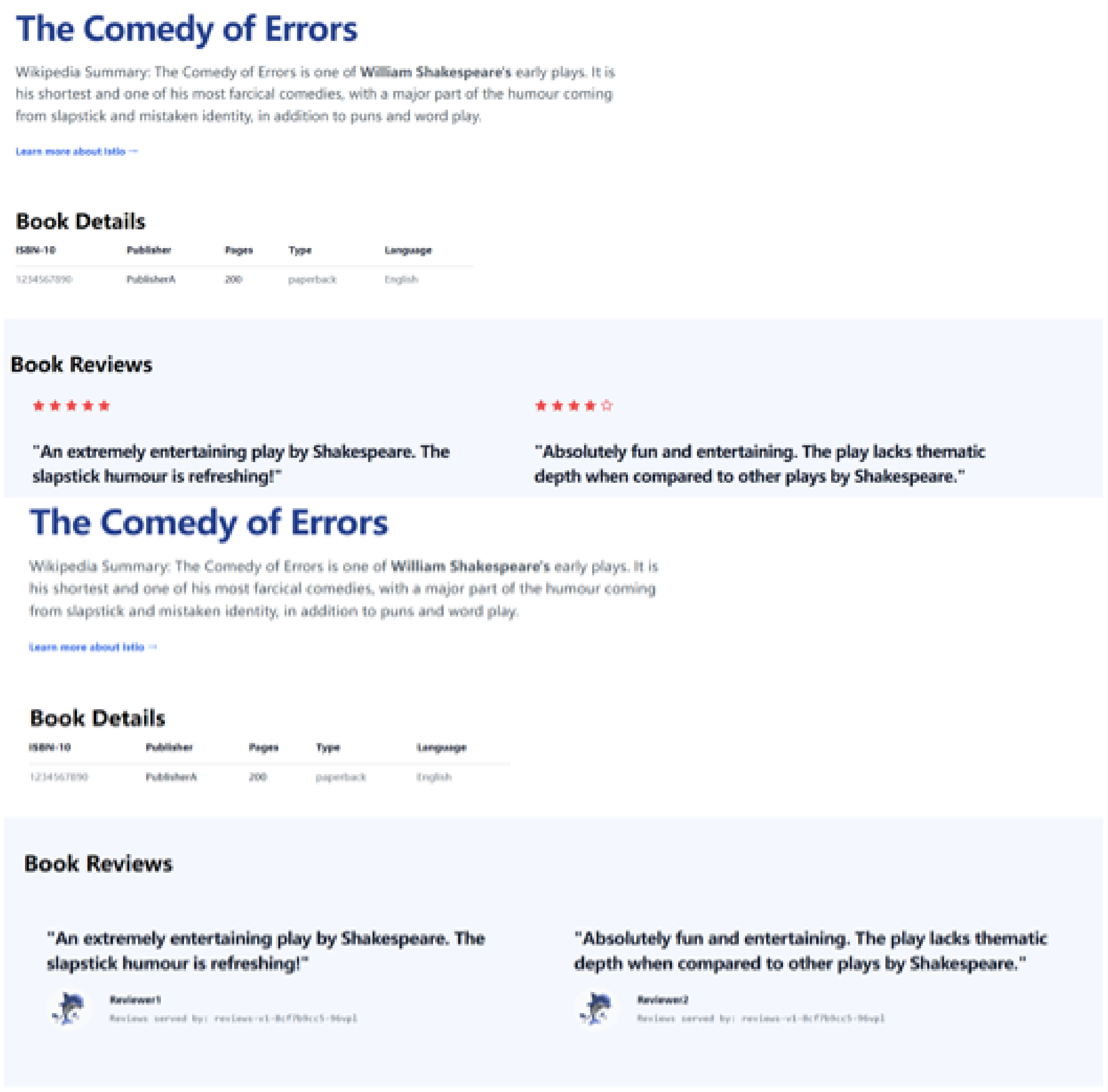
First, a `DestinationRule` was applied to define subsets corresponding to different versions of the reviews service: Next, a `VirtualService` was applied to split traffic evenly between version v1 and v3 of the reviews service:

```
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml
```

After the routing rules were applied, the Bookinfo product page was refreshed multiple times in the browser.
As shown in Figure 8, the reviews section alternated between:

- Pages without star ratings (reviews-v1)
- Pages displaying red star ratings (reviews-v3)

This configuration groups service instances by version labels and enables version-aware routing.



The Comedy of Errors

Wikipedia Summary: The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Learn more about Istio →

Book Details

ISBN-10	Publisher	Pages	Type	Language
1234567890	PublisherA	200	paperback	English

Book Reviews

★★★★★

"An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!"

★★★★☆

"Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare."

The Comedy of Errors

Wikipedia Summary: The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Learn more about Istio →

Book Details

ISBN-10	Publisher	Pages	Type	Language
1234567890	PublisherA	200	paperback	English

Book Reviews

"An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!"

Reviewer1
Reviews served by: reviews-v1-8cf7b9cc5-96vpl

"Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare."

Reviewer2
Reviews served by: reviews-v1-8cf7b9cc5-96vpl

Figure 8: Bookinfo reviews service showing different versions under 50/50 traffic split

7.Verifying Internal Service Mesh Connectivity

To verify that the microservices inside the mesh could communicate with one another, internal traffic tests were executed from within the ratings service Pod.
First, the Pod name was obtained using a label selector:

```
$POD_NAME = kubectl get pod -l app=ratings \
  -o jsonpath='{.items[0].metadata.name}'
```

Then, a request was issued from inside the Pod to the productpage service:

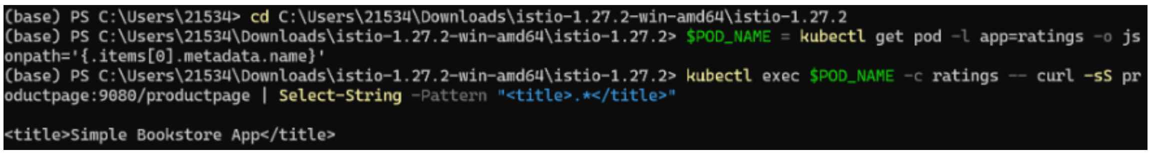
```
kubectl exec $POD_NAME -c ratings -- \
  curl -sS productpage:9080/productpage | \
  Select-String -Pattern "<title>.*</title>"
```

The output extracted the HTML `<title>` tag:

```
<title>Simple Bookstore App</title>
```

Figure 9 captures this verification process. The successful response demonstrates that:

- Istio sidecar injection is functioning correctly,
- Service-to-service communication within the mesh is operational,
- DNS-based service discovery inside the mesh is working as expected.



```
(base) PS C:\Users\21534> cd C:\Users\21534\Downloads\istio-1.27.2-win-amd64\istio-1.27.2
(base) PS C:\Users\21534\Downloads\istio-1.27.2-win-amd64\istio-1.27.2> $POD_NAME = kubectl get pod -l app=ratings -o js
onpath='{.items[0].metadata.name}'
(base) PS C:\Users\21534\Downloads\istio-1.27.2-win-amd64\istio-1.27.2> kubectl exec $POD_NAME -c ratings -- curl -sS pr
oductpage:9080/productpage | Select-String -Pattern "<title>.*</title>"

<title>Simple Bookstore App</title>
```

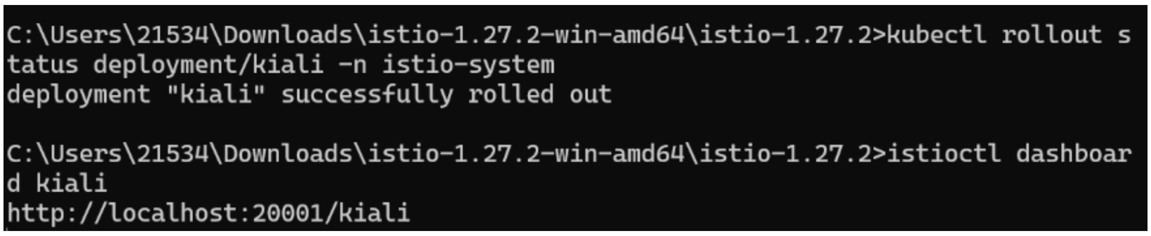
Figure 9: Internal mesh verification using `kubectl exec` and `curl` from the ratings Pod.

8.Viewing the Service Mesh in Kiali Dashboard

Kiali, Grafana, Prometheus, and Jaeger were installed using Istio addons. Launching the dashboard:

```
istioctl dashboard kiali
```

The terminal message confirming dashboard launch is shown in Figure 10.



```
C:\Users\21534\Downloads\istio-1.27.2-win-amd64\istio-1.27.2>kubectl rollout s
tatus deployment/kiali -n istio-system
deployment "kiali" successfully rolled out

C:\Users\21534\Downloads\istio-1.27.2-win-amd64\istio-1.27.2>istioctl dashboar
d kiali
http://localhost:20001/kiali
```

Figure 10: Launching the Kiali dashboard.

Figure 11 shows the fully rendered Kiali UI, with Bookinfo services displayed in the service mesh topology.

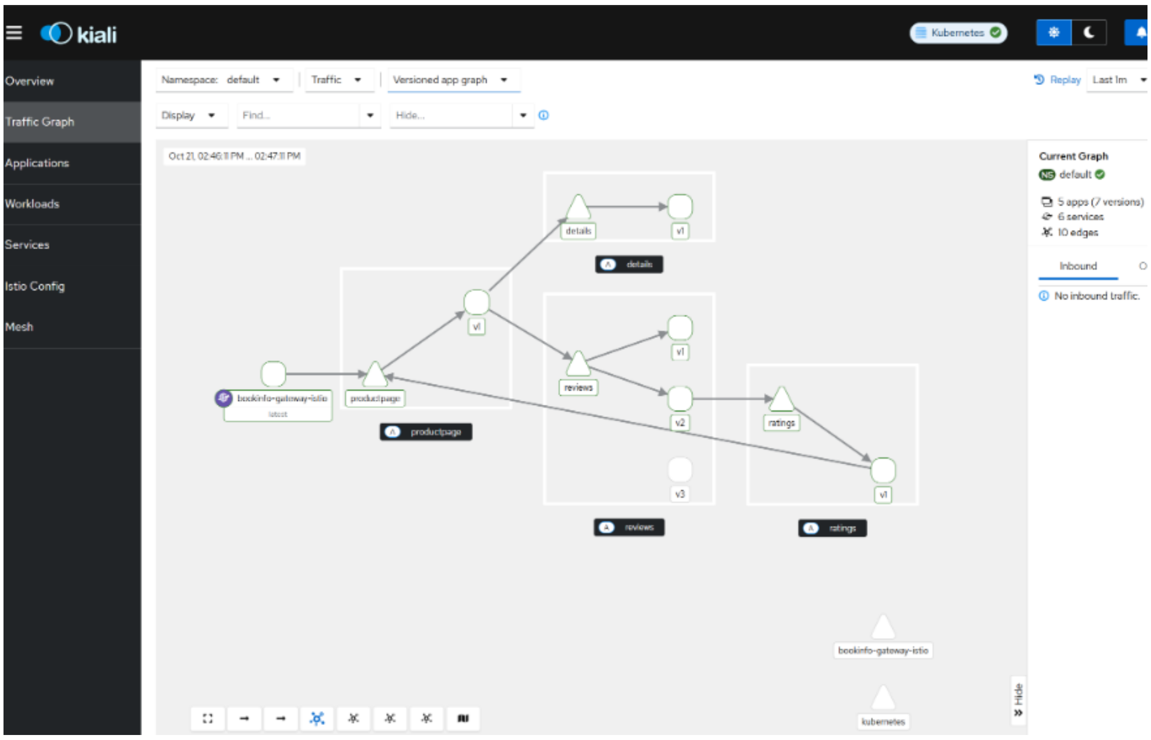


Figure 11: Kiali dashboard visualizing Bookinfo mesh services.

IV. Conclusion

In this experiment, we successfully deployed and configured the Istio service mesh on a Kubernetes cluster, gaining hands-on experience with key Istio components, including the control plane and ingress gateway. Through the deployment of the Bookinfo sample application, we verified the core functionalities of Istio such as traffic management, service-to-service communication, and observability.

Key tasks achieved in this experiment included:

- Installing the Istio control plane and ingress gateway in a local Kubernetes environment.
- Deploying the Bookinfo application with automatic sidecar injection.
- Configuring Gateway API resources to expose the application to external traffic.
- Managing traffic distribution between multiple service versions (reviews-v1 and reviews-v3) using Istio's traffic splitting feature.
- Verifying internal service mesh connectivity and ensuring communication between services within the mesh.
- Using the Kiali dashboard to visualize the service mesh topology and traffic flow.

This experiment provided valuable insights into how Istio enhances the manageability and observability of microservices, particularly in complex Kubernetes environments. The ability to route traffic, secure service-to-service communication, and monitor the performance of services within the mesh makes Istio a powerful tool for managing cloud-native applications. Future work could involve exploring advanced features of Istio such as fault injection, circuit breaking, and enhanced security policies.