Contents

Bookinfo Application

7 minute read page test[3]

This example deploys a sample application composed of four separate microservices used to demonstrate various Istio features.

If you installed Istio using the Getting Started[4] instructions, you already have Bookinfo installed and you can skip most of these steps and go directly to Define the service versions.

The application displays information about a book, similar to a single catalog entry of an online book store. Displayed on the page is a description of the book, book details (ISBN, number of pages, and so on), and a few book reviews.

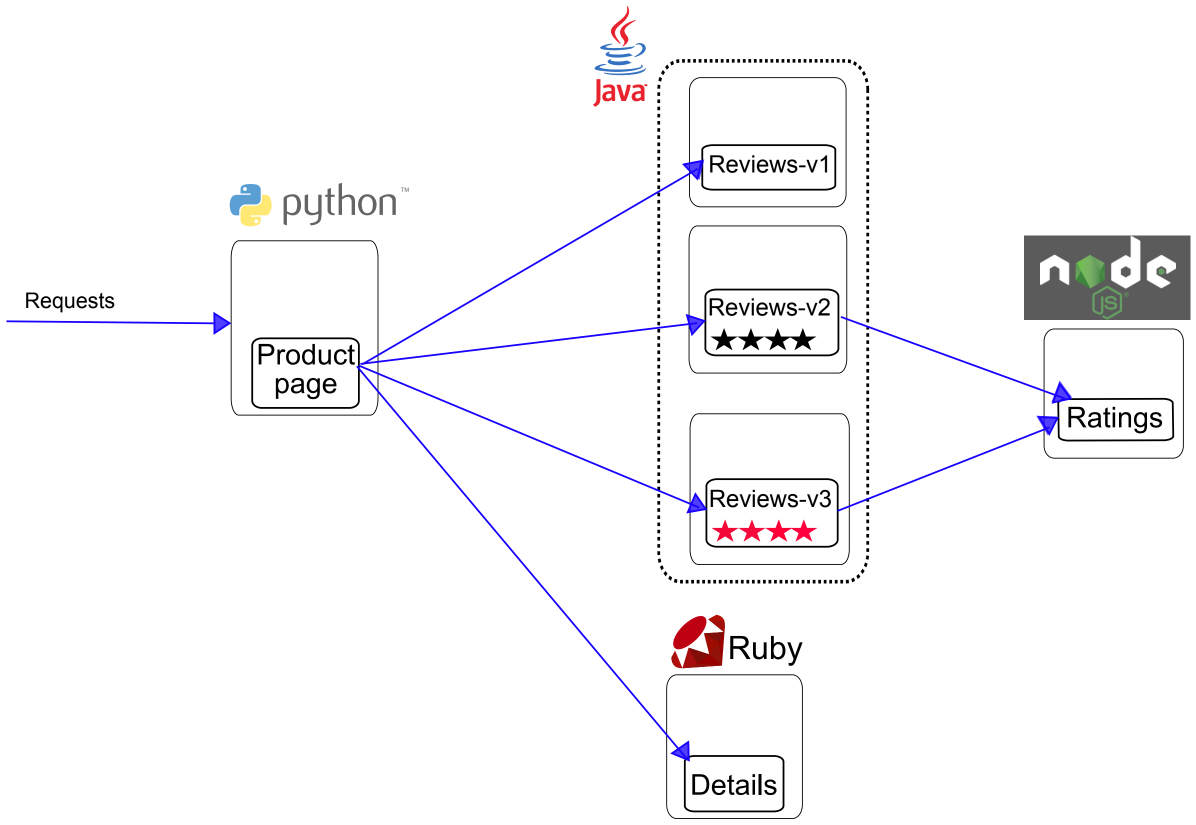
The Bookinfo application is broken into four separate microservices:

- `productpage`. The `productpage` microservice calls the `details` and `reviews` microservices to populate the page.
- `details`. The `details` microservice contains book information.
- `reviews`. The `reviews` microservice contains book reviews. It also calls the `ratings` microservice.
- `ratings`. The `ratings` microservice contains book ranking information that accompanies a book review.

There are 3 versions of the `reviews` microservice:

- Version v1 doesn't call the `ratings` service.
- Version v2 calls the `ratings` service, and displays each rating as 1 to 5 black stars.
- Version v3 calls the `ratings` service, and displays each rating as 1 to 5 red stars.

The end-to-end architecture of the application is shown below.



Bookinfo Application without Istio

This application is polyglot, i.e., the microservices are written in different languages. It's worth noting that these services have no dependencies on Istio, but make an interesting service mesh example, particularly because of the multitude of services, languages and versions for the `reviews` service.

Before you begin

If you haven't already done so, setup Istio by following the instructions in the installation guide[5].

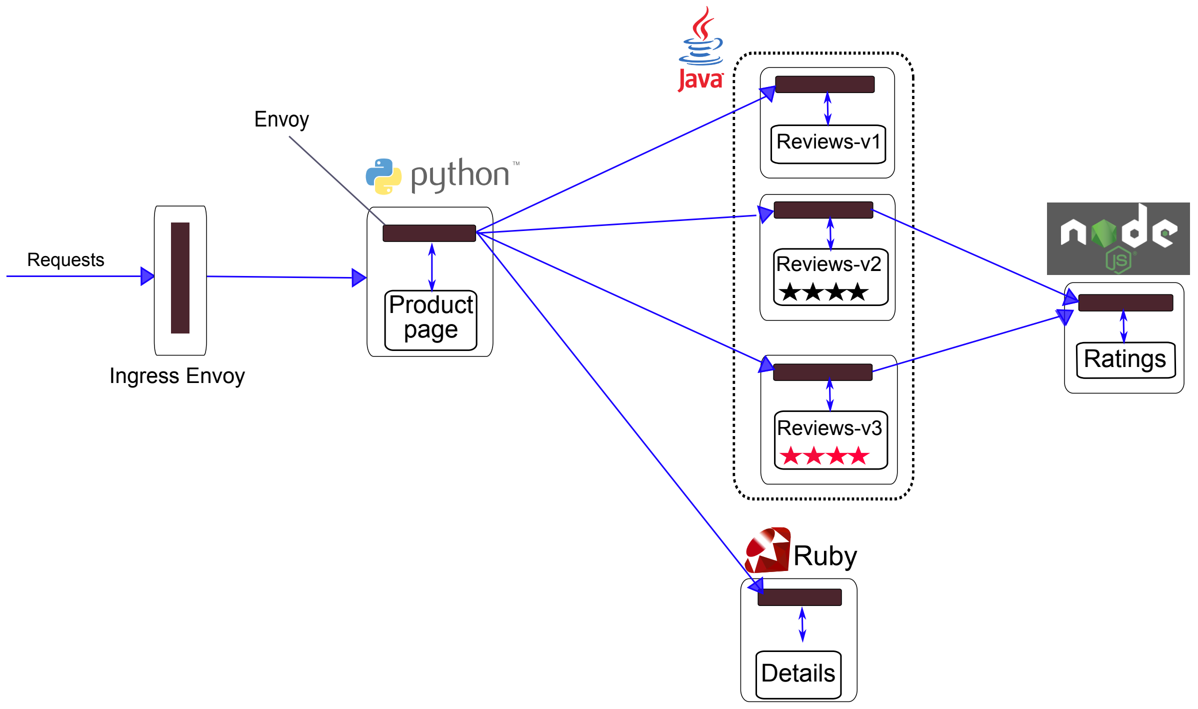
Istio includes beta support for the Kubernetes Gateway API[6] and intends to make it the default API for traffic management in the future[7]. The following instructions allow you to choose to use either the Gateway API or the Istio configuration API when configuring traffic management in the mesh. Follow instructions under either the `Gateway API` or `Istio APIs` tab, according to your preference.

Note that the Kubernetes Gateway API CRDs do not come installed by default on most Kubernetes clusters, so make sure they are installed before using the Gateway API:

```
$ kubectl get crd gateways.gateway.networking.k8s.io &> /dev/null || \
  { kubectl kustomize "github.com/kubernetes-sigs/gateway-api/config/crd?ref=v1.0.0" | kubectl apply -f -; }
```

Deploying the application

To run the sample with Istio requires no changes to the application itself. Instead, you simply need to configure and run the services in an Istio-enabled environment, with Envoy sidecars injected along side each service. The resulting deployment will look like this:



Bookinfo Application

All of the microservices will be packaged with an Envoy sidecar that intercepts incoming and outgoing calls for the services, providing the hooks needed to externally control, via the Istio control plane, routing, telemetry collection, and policy enforcement for the application as a whole.

Start the application services

If you use GKE, please ensure your cluster has at least 4 standard GKE nodes. If you use Minikube, please ensure you have at least 4GB RAM.

1. Change directory to the root of the Istio installation.
2. The default Istio installation uses automatic sidecar injection. Label the namespace that will host the application with `istio-injection=enabled`:

```
$ kubectl label namespace default istio-injection=enabled
```

If you use OpenShift, make sure to give appropriate permissions to service accounts on the namespace as described in

OpenShift setup page.

3. Deploy your application using the `kubectl` command:

```
$ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml[8]
```

If you disabled automatic sidecar injection during installation and rely on manual sidecar injection, use the `istioctl kube-inject` command to modify the `bookinfo.yaml` file before deploying your application.

```
$ kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/bookinfo.yaml[8])
```

The command launches all four services shown in the `bookinfo` application architecture diagram. All 3 versions of the reviews service, v1, v2, and v3, are started.

In a realistic deployment, new versions of a microservice are deployed over time instead of deploying all versions simultaneously.

4. Confirm all services and pods are correctly defined and running:

```
$ kubectl get services

NAME          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
details       ClusterIP   10.0.0.31     <none>        9080/TCP   6m
kubernetes    ClusterIP   10.0.0.1      <none>        443/TCP    7d
productpage   ClusterIP   10.0.0.120    <none>        9080/TCP   6m
ratings       ClusterIP   10.0.0.15     <none>        9080/TCP   6m
reviews       ClusterIP   10.0.0.170    <none>        9080/TCP   6m
```

and

```
$ kubectl get pods

NAME                                 READY     STATUS    RESTARTS   AGE
details-v1-1520924117-48z17          2/2       Running   0          6m
productpage-v1-560495357-jk1lz       2/2       Running   0          6m
ratings-v1-734492171-rnr5l           2/2       Running   0          6m
reviews-v1-874083890-f0qf0           2/2       Running   0          6m
reviews-v2-1343845940-b34q5          2/2       Running   0          6m
reviews-v3-1813607990-8ch52          2/2       Running   0          6m
```

5. To confirm that the Bookinfo application is running, send a request to it by a `curl` command from some pod, for example from `ratings`:

```
$ kubectl exec "$(kubectl get pod -l app=ratings -o jsonpath='{.items[0].metadata.name}')" -c ratings -- curl -sS productpage:9080/productpage

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

Determine the ingress IP and port

Now that the Bookinfo services are up and running, you need to make the application accessible from outside of your Kubernetes cluster, e.g., from a browser. A gateway is used for this purpose.

1. Create a gateway for the Bookinfo application:

   Istio APIs Gateway API

   Create an Istio Gateway using the following command:

   ```
   $ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml⁹

   gateway.networking.istio.io/bookinfo-gateway created
   virtualservice.networking.istio.io/bookinfo created
   ```

   Confirm the gateway has been created:

   ```
   $ kubectl get gateway

   NAME               AGE
   bookinfo-gateway   32s
   ```

   Follow these instructions to set the `INGRESS_HOST` and `INGRESS_PORT` variables for accessing the gateway. Return here, when they are set.

2. Set `GATEWAY_URL`:

   ```
   $ export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
   ```

Confirm the app is accessible from outside the cluster

To confirm that the Bookinfo application is accessible from outside the cluster, run the following `curl` command:

```
$ curl -s "http://${GATEWAY_URL}/productpage" | grep -o "<title>.*</title>"

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

You can also point your browser to `http://$GATEWAY_URL/productpage` to view the Bookinfo web page. If you refresh the page several times, you should see different versions of reviews shown in `productpage`, presented in a round robin style (red stars, black stars, no stars), since we haven't yet used Istio to control the version routing.

Define the service versions

Before you can use Istio to control the Bookinfo version routing, you need to define the available versions.

Istio APIs | Gateway API

Istio uses *subsets*, in destination rules, to define versions of a service. Run the following command to create default destination rules for the Bookinfo services:

```
$ kubectl apply -f samples/bookinfo/networking/destination-rule-all.yaml[12]
```

The `default` and `demo` configuration profiles[13] have auto mutual TLS enabled by default. To enforce mutual TLS, use the destination rules in `samples/bookinfo/networking/destination-rule-all-mtls.yaml`.

Wait a few seconds for the destination rules to propagate.

You can display the destination rules with the following command:

```
$ kubectl get destinationrules -o yaml
```

What's next

You can now use this sample to experiment with Istio's features for traffic routing, fault injection, rate limiting, etc. To proceed, refer to one or more of the Istio Tasks[15], depending on your interest. Configuring Request Routing[16] is a good place to start for beginners.

Cleanup

When you're finished experimenting with the Bookinfo sample, uninstall and clean it up using the following command:

```
$ samples/bookinfo/platform/kube/cleanup.sh[17]
```

Links

1. https://istio.io/latest/docs/
2. https://istio.io/latest/docs/examples/
3. https://github.com/istio/istio.io/tree/master/README.md#testing-document-content
4. https://istio.io/latest/docs/setup/getting-started/
5. https://istio.io/latest/docs/setup/
6. https://gateway-api.sigs.k8s.io/
7. https://istio.io/latest/blog/2022/gateway-api-beta/
8. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/platform/kube/bookinfo.yaml

9. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/networking/bookinfo-gateway.yaml
10. https://gateway-api.sigs.k8s.io/api-types/gateway/
11. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/gateway-api/bookinfo-gateway.yaml
12. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/networking/destination-rule-all.yaml
13. https://istio.io/latest/docs/setup/additional-setup/config-profiles/
14. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/platform/kube/bookinfo-versions.yaml
15. https://istio.io/latest/docs/tasks
16. https://istio.io/latest/docs/tasks/traffic-management/request-routing/
17. https://raw.githubusercontent.com/istio/istio/release-1.20/samples/bookinfo/platform/kube/cleanup.sh