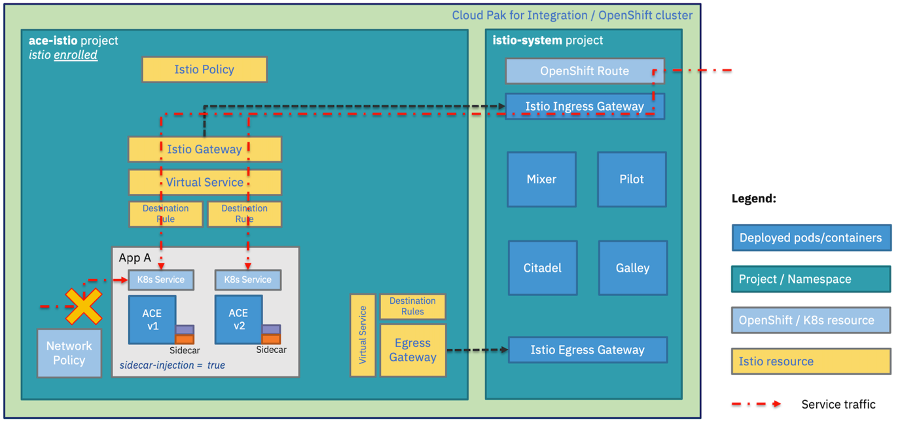
**Istio implementation for ACE on CP4I**

Using ACE and OSSM in CP4I

This post and the accompanying GitHub repository will provide instructions on one example use case for Istio - how to use Istio to enable A/B testing when rolling out a new integration. Other use cases would typically require the same steps to put Istio in place:

* Deploy the Red Hat OpenShift Service Mesh to an OpenShift cluster running the IBM Cloud Pak for Integration
* Create an *istio-enabled*project to run ACE servers
* Work with the out-of-the-box Network Policies created when deploying ACE
* Route traffic to ACE using the correct Istio resources
* Deploy new version of the ACE servers and perform A/B testing through Istio

OSSM and ACE in CP4I

Red Hat OpenShift Service Mesh

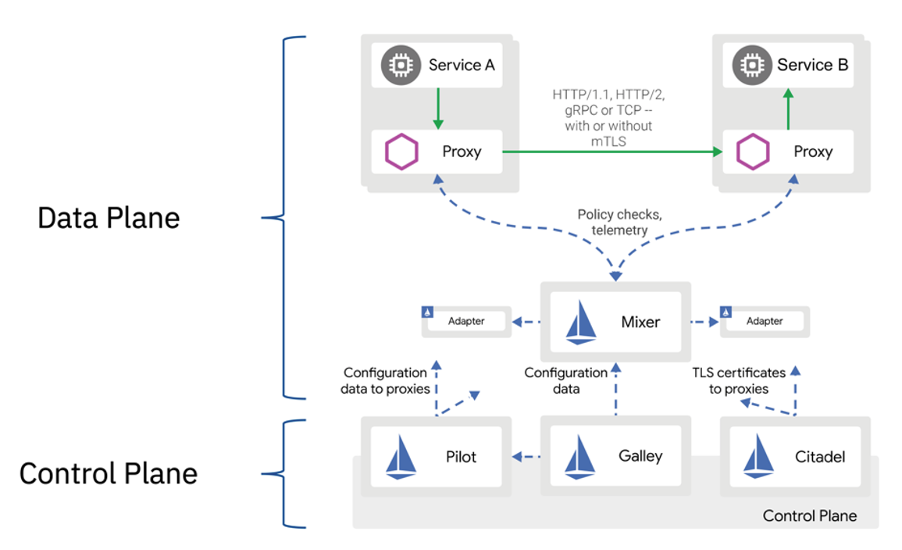
The Red Hat OpenShift Service Mesh is based on an opinionated distribution of Istio called [Maistra](https://maistra.io/" \t "_blank), which combines Kiali, Jaeger, ElasticSearch and Prometheus into a platform managed via [operators](https://kubernetes.io/docs/concepts/extend-kubernetes/operator/). This brings a number of benefits, especially to those new to service mesh implementation. Examples include

* Pre-defined installations of Kiali for observability and Jaeger for tracing
* All components are managed by [operators](https://kubernetes.io/docs/concepts/extend-kubernetes/operator/) to simplify installation and ongoing administration
* Multi-tenancy for simplified management of multiple service-mesh ecosystems within a single cluster

An installation of Red Hat OpenShift Service Mesh differs from upstream Istio community installations [in multiple ways](https://www.redhat.com/en/topics/microservices/why-choose-openshift-service-mesh):

* OpenShift Service Mesh installs a multi-tenant control plane by default
* OpenShift Service Mesh extends Role Based Access Control (RBAC) features
* OpenShift Service Mesh replaces BoringSSL with its general-purpose equivalent: OpenSSL
* Kiali and Jaeger, visualisation tools for Istio, are enabled by default in OpenShift Service Mesh

Istio v1.4.8 neatly breaks down the Service Mesh into a Control Plane and a Data Plane, as the diagram below from [Istiooldie](https://istio.io/v1.3/docs/concepts/what-is-istio/" \t "_blank) shows.

Istio architecture

## Install OpenShift Service Mesh for App Connect Enterprise

The base of this implementation is a CP4I 2020.3 installation (on OpenShift 4.4), which has the ACE Dashboard, ACE Designer, and Operations Dashboard deployed in a project called ace.

For more information on how to install and configure CP4I 2020.3, please refer to the knowledge center [documentation](https://www.ibm.com/support/knowledgecenter/SSGT7J_20.3/install/install.html).

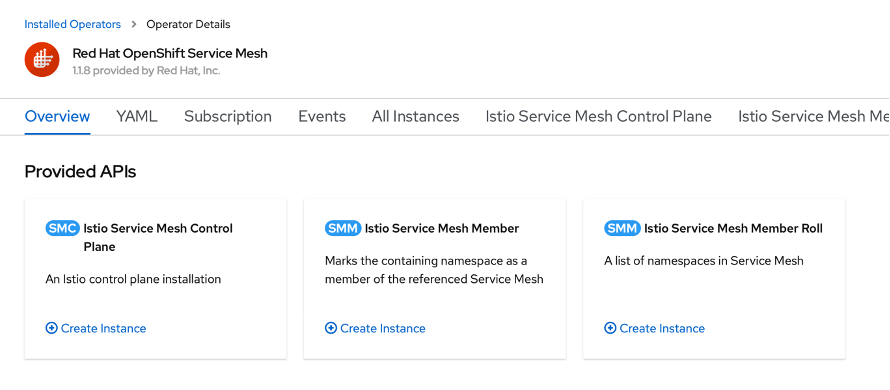
From that starting point, the next step is to install and deploy the OpenShift Service Mesh. More detailed instructions are provided in the [GitHub repository](https://github.com/ot4i/CP4I-OSSM) accompanying this post, which summarises some of the key steps that help to understand what is happening.

The OpenShift Service Mesh is a supported feature in OpenShift 4, so it should come as no surprise that it is deployed via Kubernetes operators. As noted, it has dependencies on three other supported features, Elasticsearch, Kiali and Jaeger.

Istio operators

* The OSSM control plane then requires creation of a unique project, such as istio-system, where instances of the operators are installed.
* All the operators mentioned above must be installed in the specified order and in all OpenShift projects that will use the mesh.

Once all operators are installed, a control plane can be [deployed](https://github.com/ot4i/CP4I-OSSM/blob/main/Readme.md#control-plane-deployment) in the istio-system project from the Installed Operator view.

Red Hat OpenShift Service Mesh

With the Control Plane in place, the projects which will be part of the data plane can be added to the Service Mesh member roll - which lists the namespaces to be istio-enabled – from the same view.

Once the OpenShift Service Mesh is created, a couple of [extra configuration steps](https://github.com/ot4i/CP4I-OSSM/blob/main/Readme.md#control-plane-configuration) need to be applied. In particular, for ease of use, enabling [Automatic route creation](https://docs.openshift.com/container-platform/4.3/service_mesh/service_mesh_day_two/ossm-auto-route.html) is recommended. Automatic route creation creates OpenShift routes for each new virtual service exposed via the Istio Ingress, without the need of creating additional resources.

## Configure App Connect Enterprise to leverage the service mesh

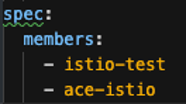
### **Istio-enable ACE project**

Once the OpenShift Service Mesh is deployed and configured, the next step is to create a project (e.g. ace-istio) and istio-enable it by enrolling it as a service mesh member.

When a project is enrolled in the Service Mesh, a network policy is automatically created to ensure that no direct access to Kubernetes services in that project is allowed, unless traffic-allowing Network Policies are present. Both ACE Designer and ACE Dashboard deploy such policies, hence remain accessible even after the project is enrolled in the service mesh.

On the other hand, a new container deployment would not be directly accessible.

To enroll the ace-istio project into the service mesh, it needs to be added as a member of the Service Mesh Member Roll, which can be found in Installed Operators > Red Hat OpenShift Service Mesh > Istio Service Mesh Member Roll, in the istio-system project.

Service Mesh Member Roll

With the Red Hat Open Shift Service Mesh, enrolling the ace-istio project does not in itself result in the Istio sidecars (i.e. Envoy Proxys) being injected in pods in that namespace.

### **Enable sidecar injection when deploying ACE servers**

For the sidecars to be injected, existing and new Kubernetes deployments need to be explicitly annotated: the annotation sidecar.istio.io/inject: 'true' needs to be present in the deployment spec's template metadata.

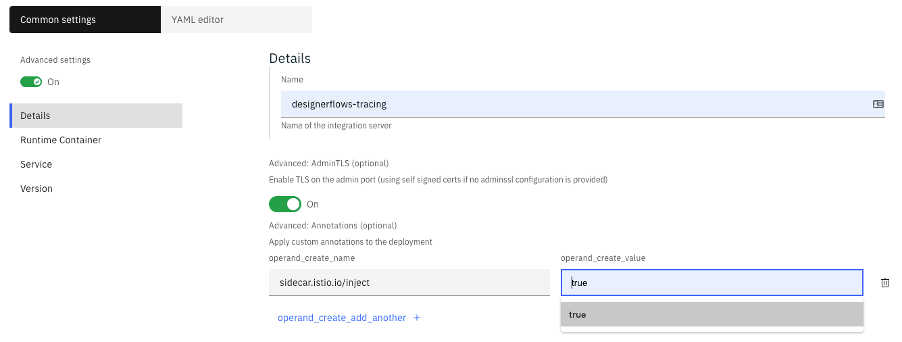
Only if that annotation is present, the sidecars will be injected automatically into the workload pods.

With Cloud Pak for Integration 2020.3.1, ACE server deployments are performed by operators. The operator allows for custom annotations – like the one needed to enable sidecar injection - to be easily added prior to the deployment, via either Command Line or User Interface.  
  
Let’s use one of the ACE flows in this test case as an example: <https://github.com/ot4i/CP4I-OSSM/tree/main/ace/designerflows-tracing>.

This test case uses an [ACE flow](https://github.com/ot4i/CP4I-OSSM/blob/main/ace/testAPIs/simpleAPI.bar) implementing a simple REST API designed in ACE Designer, which will be deployed from the ACE Dashboard UI, with open tracing enabled.

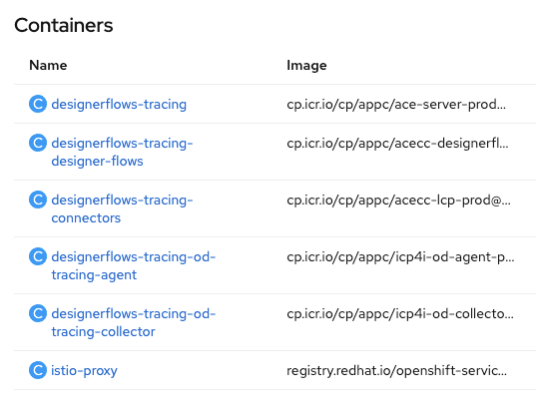
In order to add a custom annotation to deployment template via the operator, it is sufficient to enable Advanced Settings from the ACE server deployment UI, and to add a custom annotation as name/value pair:

* operand\_create\_name: sidecar.istio.io/inject
* operand\_create\_value: true

Enable sidecar injection

In this test case, the operator will create an ACE server deployment, for each pod in the deployment, 6 containers will be present:

* The main ACE server container
* Two sidecar containers to enable designer flows
* Two sidecar containers to enable tracing
* One sidecar container with the Envoy Proxy

Deployed sidecars

### **Deploy second ACE server for A/B testing**

A typical use case for the service mesh is to enable A/B testing: routing traffic to a new version of a service without taking the original version offline.

The test case implements an A/B testing scenario by creating a second ACE server – with the same custom annotation to enable sidecar injection – with a new version of the same API used earlier deployed to it.

To deploy the v2 ACE server, follow the instructions for the first deployment, and simply use this [v2 bar file](https://github.com/ot4i/CP4I-OSSM/blob/main/ace/testAPIs/simpleAPIv2.bar).

## Create Istio resources

At this point the annotated ACE deployments will have an Envoy Proxy sidecars intercepting traffic running next to the ACE Server containers in the same pod. However, the ACE servers are not yet accessible from outside the cluster, and need to be formally exposed via the Istio ingress using further Istio resources:

* An Istio Gateway
* A Virtual Service
* Destination Rules

An Istio gateway is a YAML file referencing Istio ingress, and specifying how the ACE server will be exposed on it. It also defines one or more hosts, which will be used by the Automatic route creation feature to generate an OpenShift route.

A Virtual Service is a YAML file referencing the Istio gateway, and defining the rules to route traffic to the Kubernetes services exposing the two versions of the ACE deployments: v1 and v2.  
Traffic splitting weights can also be defined in the Virtual Service.

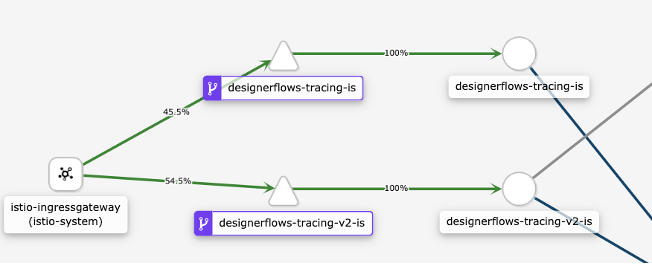
Destination Rules are YAML files which define - in a finer grained detail - policies that apply to traffic intended for a service after routing has occurred via the Virtual Service. Traffic policies in Destination Rules can override Virtual Service policies at subset level.

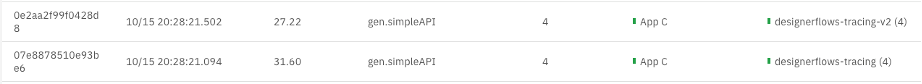
Sample YAML files for these 3 resource types are available in the [GitHub repo for this test case](https://github.com/ot4i/CP4I-OSSM/tree/main/ace/designerflows-tracing).

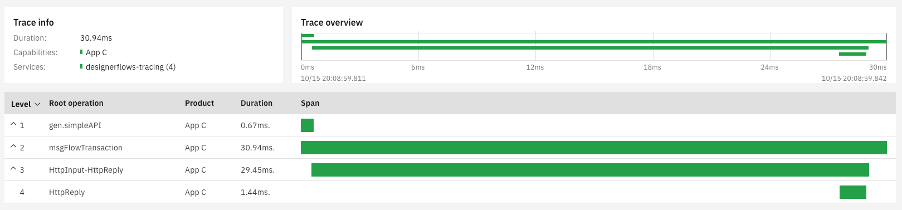
Worth noting that the Destination Rules provided here are only a simplistic example, with no additional policies to the ones defined in the Virtual Service.

## Test ACE and Istio working together

Once all Istio resources have been created and applied to the ace-istio project, both ACE servers will be accessible on the automatically generated route listed in the Istio Gateway.  
  
By calling the service endpoint exposed on the OpenShift route, it is possible to observe how traffic is distributed across the two versions of the ACE server according to the policies defined in the Istio Virtual Service.

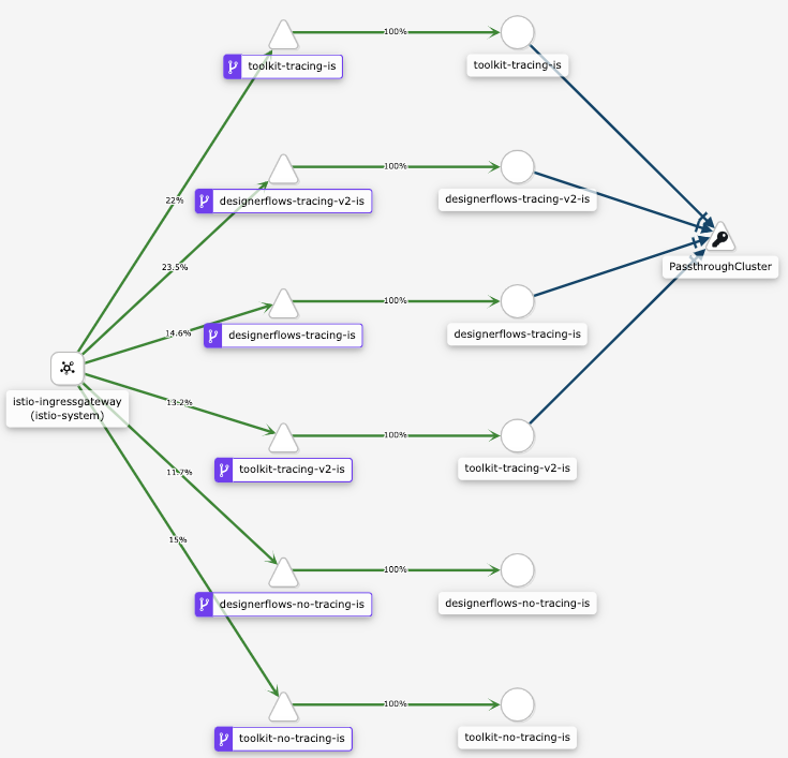
ACE via Istio

As this test case has tracing enabled, the Operations Dashboard can also be used to validate the ACE functionality.  


ACE, Istio, and Operations Dashboard

## Additional configurations

The GitHub repository provides four variations of ACE configurations, with and without ACE Designer and Operations Dashboard, all of which integrate successfully with the service mesh.

OSSM and CP4I

## Conclusion

The Red Hat OpenShift Service Mesh is an opinionated implementation of Istio which runs on OpenShift Container Platform and Cloud Pak for Integration.

In the context of Cloud Pak for Integration, the major difference between Istio and the Red Hat OpenShift Service Mesh is that deployments need to be individually enabled for sidecar injection, even if they are running in an istio-enabled project.

This blog post provides a guide to successfully configure App Connect Enterprise to work with OpenShift Service Mesh in Cloud Pak for Integration 2020.3, running on OpenShift 4.4.

A Git Hub repository with examples and more detailed guidance specific to the test case described in this blog post is available [here](https://github.com/ot4i/CP4I-OSSM).