



# Red Hat Application Interconnect 1.0

## Creating a service network with OpenShift

For Use with Application Interconnect 1.0 LIMITED AVAILABILITY



# Red Hat Application Interconnect 1.0 Creating a service network with OpenShift

---

For Use with Application Interconnect 1.0 LIMITED AVAILABILITY

## Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This tutorial describes how to create Application Interconnect sites on OpenShift to build a service network.

## Table of Contents

PREFACE .....	3
CHAPTER 1. INSTALLING THE SKUPPER CLI .....	4
CHAPTER 2. CONFIGURING TERMINAL SESSIONS .....	5
CHAPTER 3. INSTALLING THE SERVICE NETWORK ROUTER IN BOTH CLUSTERS .....	6
CHAPTER 4. CONNECTING NAMESPACES TO CREATE A SERVICE NETWORK .....	7
CHAPTER 5. CREATING THE FRONTEND SERVICE .....	8
CHAPTER 6. CREATING THE BACKEND SERVICE AND MAKING IT AVAILABLE ON THE SERVICE NETWORK .....	9
CHAPTER 7. TEARING DOWN THE SERVICE NETWORK .....	11
CHAPTER 8. CREATING SITES USING A CUSTOM CERTIFICATE AUTHORITY ON OPENSIFT .....	12



# PREFACE

This tutorial demonstrates how to connect a frontend service on a OpenShift cluster with a backend service on a OpenShift cluster using the **skupper** command-line interface (CLI).

See [Introduction to Application Interconnect](#) for an introduction to Application Interconnect.

## Prerequisites

- Access to projects in two OpenShift clusters, **cluster-admin** access is not required.
- One of the OpenShift clusters must be addressable from the other cluster.
- **kubectrl** and **oc** CLI. Many commands can be performed using **oc**, however this tutorial shows the **kubectrl** options.

This tutorial shows how to connect the following namespaces:

- **west** - runs the frontend service and is typically a public cluster.
- **east** - runs the backend service.

## CHAPTER 1. INSTALLING THE **SKUPPER** CLI

Installing the **skupper** command-line interface (CLI) provides a simple method to get started with Application Interconnect.

### Procedure

1. Ensure your subscription has been activated and your system is registered.
2. Subscribe to the required repositories:

#### Red Hat Enterprise Linux 8

```
$ sudo subscription-manager repos --enable=application-interconnect-1-for-rhel-8-x86_64-rpms
```

3. Use the **yum** or **dnf** command to install the **skupper** package:

```
$ sudo yum install skupper-cli
```

4. Verify the installation.

```
$ skupper version  
client version 1.0.2-redhat-1
```





## CHAPTER 2. CONFIGURING TERMINAL SESSIONS

This procedure describes how to configure your terminal sessions to use configurations to avoid problems as you configure Application Interconnect on different clusters.

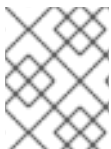
The following table shows how you might set up your terminal sessions.

**Table 2.1. Terminal sessions**

west terminal session	east terminal session
 \$ kubectl get pods	 \$ kubectl get pods

### Prerequisites

- The OpenShift CLI is installed. See the [OpenShift CLI](#) documentation for more instructions on how to install **oc**.



### NOTE

In OpenShift 4.6 and later, you can use the web terminal to perform the following procedure, as described in the [web terminal](#) documentation.


### Procedure

1. Start a terminal session to work on the **west** namespace and set the **KUBECONFIG** environment variable:

```
 $ export KUBECONFIG=$HOME/.kube/config-west
```

This session is referred to later as the *west* terminal session.

2. Start a terminal session to work on the **east** namespace and set the **KUBECONFIG** environment variable:

```
 $ export KUBECONFIG=$HOME/.kube/config-east
```

This session is referred to later as the *east* terminal session.

3. In each terminal session, log into the OpenShift cluster.

## CHAPTER 3. INSTALLING THE SERVICE NETWORK ROUTER IN BOTH CLUSTERS

1. In the west terminal session:

a. Create the **west** project (namespace):

```
$ kubectl create namespace west  
$ kubectl config set-context --current --namespace west
```

b. Create the service network router:

```
$ skupper init
```

c. Check the site status:

```
$ skupper status
```

The output should be similar to the following:

```
Skupper enabled for namespace 'west'. It is not connected to any other sites.
```

2. In the east terminal session:

a. Create the **east** project (namespace):

```
$ kubectl create namespace east  
$ kubectl config set-context --current --namespace east
```

b. Create the service network router:

```
$ skupper init
```

c. Check the site status:

```
$ skupper status
```

The output should be similar to the following:

```
Skupper enabled for namespace 'east'. It is not connected to any other sites.
```

## CHAPTER 4. CONNECTING NAMESPACES TO CREATE A SERVICE NETWORK

With the service network routers installed, you can connect them together securely and allow service sharing across the service network.

### Procedure

1. In the west terminal session, create a connection token to allow connection to the west namespace:

```
$ skupper token create $HOME/secret.yaml
```

This command creates the **secret.yaml** file in your home directory, which you can use to create the secure connection.

2. In the east terminal session, use the token to create a connection to the west namespace:

```
$ skupper link create $HOME/secret.yaml
```

3. Check the site status from the west terminal session:

```
$ skupper status
```

The output should be similar to the following:

```
Skupper is enabled for namespace "west" in interior mode. It is connected to 1 other site. It
has no exposed services.
The site console url is: https://<skupper-url>
The credentials for internal console-auth mode are held in secret: 'skupper-console-users'
```

## CHAPTER 5. CREATING THE FRONTEND SERVICE

The frontend service is a simple Python application that displays a message from the backend application.

### Procedure

Perform all tasks in the west terminal session:

1. Deploy the frontend service:

```
$ kubectl create deployment hello-world-frontend --image quay.io/skupper/hello-world-frontend
```

2. Expose the frontend deployment as a cluster service:

```
$ kubectl expose deployment hello-world-frontend --port 8080 --type LoadBalancer
```

3. Create a route for the frontend:

```
$ kubectl expose svc/hello-world-frontend
```

4. Check the frontend route:

- a. Get the route details:

```
$ oc get routes
```

The output should be similar to the following:

```
NAME           HOST/PORT
hello-world-frontend <frontend-url>
```

- b. Navigate to the **<frontend-url>** value in your browser, you see a message similar to the following because the frontend cannot communicate with the backend yet:

```
Trouble! HTTPConnectionPool(host='hello-world-backend', port=8080): Max retries
exceeded with url: /api/hello (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at 0x7fbcdf0d1d0>:
Failed to establish a new connection: [Errno -2] Name or service not known'))
```

To resolve this situation, you must create the backend service and make it available on the service network.

## CHAPTER 6. CREATING THE BACKEND SERVICE AND MAKING IT AVAILABLE ON THE SERVICE NETWORK

The backend service runs in the **east** namespace and is not available on the service network by default. You use the **skupper** command to expose the service to all namespaces on the service network. The backend app is a simple Python application that passes a message to the frontend application.

### Procedure

1. Deploy the backend service in the east terminal session:

```
$ kubectl create deployment hello-world-backend --image quay.io/skupper/hello-world-backend
```

2. Expose the backend service on the service network from the east terminal session:

```
$ skupper expose deployment hello-world-backend --port 8080 --protocol tcp
```

3. Check the site status from the west terminal session:

```
$ skupper status
```

The output should be similar to the following:

```
Skupper is enabled for namespace "west" in interior mode. It is connected to 1 other site. It has 1 exposed service.
```

The service is exposed from the **east** namespace.

4. Check the frontend route in the west terminal session:

- a. Get the route details:

```
$ oc get routes
```

The output should be similar to the following:

```
NAME          HOST/PORT
hello-world-frontend <frontend-url>
```

- b. Navigate to the **<frontend-url>** value in your browser, you see a message similar to the following:

```
Hi, <name>. I am Mathematical Machine (backend-77f8f45fc8-mnrdp).
```

If you click **Say hello** again, a different backend process responds showing how Application Interconnect balances the requests.

This shows how the frontend calls the backend service over the service network from a different OpenShift cluster.

### Additional resources

- [Using the Skupper console](#)
- [Configuring Application Interconnect sites using the CLI](#)

## CHAPTER 7. TEARING DOWN THE SERVICE NETWORK

This procedure describes how to remove the service network you created.

1. Delete the **west** namespace from the west terminal session:

```
$ kubectl delete namespace west
```

2. Delete the **east** namespace from the east terminal session:

```
$ kubectl delete namespace east
```

## CHAPTER 8. CREATING SITES USING A CUSTOM CERTIFICATE AUTHORITY ON OPENSIFT

By default, Application Interconnect creates certificates to establish links between sites using mutual TLS. These certificates are stored as secrets in the namespace when you create a site using **skupper init**. If you want to use your own certificates, you can populate the a set of secrets with the appropriate certificates before creating the site as described in this section. This set of secrets provides Application Interconnect with the configuration required to create a site.

The following certificates are required:

### **skupper-claims-server**

Used for linking sites with claim type tokens.

### **skupper-console-certs**

Used by the Skupper console.

### **skupper-local-client** and **skupper-local-server**

Used by the Skupper router.

### **skupper-site-server**

Used for all inter-router connections, and for headless services.

### **skupper-service-client**

Used for services exposed over TLS.

### Prerequisites

- Access to an OpenShift cluster with sufficient permission to run **skupper init**.
- Access to create certificates using your certificate authority.

### Procedure

1. Create one or more certificates for a site.

There are several alternative approaches to this step:

- Reissue an existing certificate with a set of Subject Alternative Names (SANs) for the site.
- Create a new certificate with a set of SANs for the site.
- Create a new certificate for each item relating to the site.

You require a certificate for each of the following secrets:

- **skupper.<namespace>**
- **skupper-router.<namespace>**
- **skupper-router-local**
- **skupper-router-local.<namespace>.svc.cluster.local**
- **claims-<namespace>.<clustername>.<domain>**
- **skupper-<namespace>.<clustername>.<domain>**



- **skupper-edge-`<namespace>`.`<clustername>`.`<domain>`**
- **skupper-inter-router-`<namespace>`.`<clustername>`.`<domain>`**

where:

- **`<namespace>`** is the name of the namespace where you want to create a site.
- **`<clustername>`** is the name of the cluster.
- **`<domain>`** is the domain name for the cluster.

Using a specific certificate authority technology is beyond the scope of this guide. However, the following commands show how to create a certificate authority on Linux and create a single certificate that you can use to populate the secrets.

- a. Create a **ca** directory and create a certificate authority certificate:

```
$ mkdir ca
$ cd ca
$ ssh-keygen -t rsa -m PEM -f tls.key -q -N ""
$ openssl req -x509 -nodes -days 365 -key tls.key -out tls.crt
```

- b. Given the certificate authority created **tls.crt** and **tls.key** files, you can create a certificate for the site as follows:

```
$ cd ..
$ mkdir certificate
$ cd certificate
$ openssl req -nodes -newkey rsa:4096 -x509 -CA ../ca/tls.crt -CAkey ../ca/tls.key -out
tls.crt -keyout tls.key -addext "subjectAltName = DNS:skupper.<namespace>,
DNS:skupper-router.<namespace>, DNS:skupper-router-local, DNS:skupper-router-
local.<namespace>.svc.cluster.local,DNS:claims-<namespace>.<clustername>.
<domain>, DNS:skupper-<namespace>.<clustername>.<domain>, DNS:skupper-edge-
<namespace>.<clustername>.<domain>, DNS:skupper-inter-router-<namespace>.
<clustername>.<domain>"
```

You should now have a root certificate in the **ca** directory and another certificate in the **certificate** directory that you can use with a site.

## 2. Create secrets for the site

- a. Change to the parent directory of the **certificate** directory:

```
$ cd ..
```

- b. Populate the **ca** related secrets using the certificate from the **ca** directory:

```
$ kubectl create secret tls skupper-site-ca --cert=ca/tls.crt --key=ca/tls.key
$ kubectl create secret tls skupper-service-ca --cert=ca/tls.crt --key=ca/tls.key
```

```
$ kubectl create secret tls skupper-local-ca --cert=ca/tls.crt --key=ca/tls.key
```

- c. Populate the other secrets and modify them into the format required by **skupper**:

```
$ kubectl create secret tls skupper-claims-server --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-claims-server -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-site-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

```
$ kubectl create secret tls skupper-console-certs --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-console-certs -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-local-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

```
$ kubectl create secret tls skupper-local-client --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-local-client -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-local-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

```
$ kubectl create secret tls skupper-local-server --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-local-server -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-local-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

```
$ kubectl create secret tls skupper-site-server --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-site-server -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-site-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

```
$ kubectl create secret tls skupper-service-client --cert=certificate/tls.crt --key=certificate/tls.key
```

```
$ kubectl patch secret skupper-service-client -p="{\"data\":{\"ca.crt\": \"$( kubectl get secret skupper-service-ca -o json -o=jsonpath='{.data.tls.crt}')}\"}"
```

3. Create the site using the following command:

```
$ skupper init
```

On OpenShift, **skupper** defaults to use the **route** ingress, which is the equivalent of **skupper init --ingress route**.

To verify your site, check the status:

```
$ skupper status
```

You can also verify the OpenShift routes are created using:

```
$ oc get routes
```

Finally, use the following command to check for errors relating to incorrect certificates:

```
$ skupper debug events
```

*Revised on 2022-10-19 18:01:27 UTC*