



Red Hat Application Interconnect 1.0

Using the Application Interconnect CLI

For Use with Application Interconnect 1.0 LIMITED AVAILABILITY

Red Hat Application Interconnect 1.0 Using the Application Interconnect CLI

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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 guide describes how to install, configure, and manage Application Interconnect sites to build a service network.

Table of Contents

PREFACE	3
CHAPTER 1. INSTALLING THE SKUPPER CLI	4
CHAPTER 2. CREATING A SITE USING THE CLI	5
CHAPTER 3. CUSTOM SITES	6
CHAPTER 4. LINKING SITES	7
CHAPTER 5. EXPOSING SERVICES ON THE SERVICE NETWORK FROM A NAMESPACE	9
5.1. EXPOSING SIMPLE SERVICES ON THE SERVICE NETWORK	9
5.2. EXPOSING COMPLEX SERVICES ON THE SERVICE NETWORK	10
CHAPTER 6. EXPOSING SERVICES ON THE SERVICE NETWORK FROM A LOCAL MACHINE	11
6.1. EXPOSING SIMPLE LOCAL SERVICES TO THE SERVICE NETWORK	11
6.2. WORKING WITH COMPLEX LOCAL SERVICES ON THE SERVICE NETWORK	12
6.3. CREATING A GATEWAY AND APPLYING IT ON A DIFFERENT MACHINE	13
6.4. GATEWAY YAML REFERENCE	15
CHAPTER 7. EXPLORING A SERVICE NETWORK	18
CHAPTER 8. SECURING A SERVICE NETWORK	20
8.1. RESTRICTING ACCESS TO SERVICES USING NETWORK-POLICY	20
8.2. APPLYING TLS TO HTTP2 TRAFFIC ON THE SERVICE NETWORK	20
CHAPTER 9. SUPPORTED STANDARDS AND PROTOCOLS	22
CHAPTER 10. CLI OPTIONS FOR WORKING WITH DIFFERENT CLUSTERS	23
CHAPTER 11. USING SKUPPER TOKENS	24
11.1. CREATING CLAIM TOKENS	24
11.2. CREATING CERT TOKENS	25
11.3. REVOKING ACCESS TO A SITE	25
CHAPTER 12. TROUBLESHOOTING A SERVICE NETWORK	27
12.1. CHECKING SITES	27
12.2. CREATING A SKUPPER DEBUG TARBALL	28

PREFACE

Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).



NOTE

This Limited Availability release is not available to all customers. Contact [Red Hat Sales](#) if you are interested in learning more about Application Interconnect.

Using the **skupper** command-line interface (CLI) allows you to create and manage Application Interconnect sites from the context of the current namespace.

A typical workflow is to create a site, link sites together, and expose services to the service network.

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. CREATING A SITE USING THE CLI

A service network consists of Application Interconnect sites. This section describes how to create a site using the default settings.

Prerequisites

- The **skupper** CLI is installed.
- You are logged into the cluster.
- The services you want to expose on the service network are in the active namespace.

Procedure

1. Create a default site:

```
$ skupper init
```

2. Check the site:

```
$ skupper status
```

Skupper is enabled for namespace "west" in interior mode. It is not connected to any other sites.



NOTE

The default message above is displayed when you initialize a site on a cluster that does not have the policy system installed. If you install the policy system as described in [Securing a service network using policies](#), the message becomes **Skupper is enabled for namespace "west" in interior mode (with policies)**.

The default site settings include:

- console - The Skupper console is provisioned with a single user. The password for the **admin** user is stored in the **skupper-console-users** secret. For more information on the console, see [Using the Skupper console](#).
- site name - The site name defaults to the namespace name, for example, **west**.
- ingress - An ingress is created to support linking to the site you are creating. By default, a route is created. You can specify **--ingress none** if you do not want other sites to link to this site. Other ingress options are not supported.

CHAPTER 3. CUSTOM SITES

The default **skupper init** creates sites that satisfy typical requirements.

If you require a custom configuration, note the following options:

- Creating a site without a console:

```
$ skupper init --enable-console false
```

- Configuring console authentication. There are a number of **skupper** options regarding authentication for the console:

--console-auth <authentication-mode>

Set the authentication mode for the console:

- **openshift** – Use OpenShift authentication, so that users who have permission to log into OpenShift and view the Project (namespace) can view the console.
- **internal** – Use Application Interconnect authentication, see the **console-user** and **console-password** options.
- **unsecured** – No authentication, anyone with the URL can view the console.

--console-user <username>

Username for the console user when authentication mode is set to **internal**. Defaults to **admin**.

--console-password <password>

Password for the console user when authentication mode is set to **internal**. If not specified, a random password is generated.

- Configuring service access

```
$ skupper init --create-network-policy
```



NOTE

All sites are associated with a namespace, called the *active namespace* in this procedure.

Services in the active namespace may be accessible to pods in other namespaces on that cluster by default, depending on your cluster network policies. As a result, you can expose services to pods in namespaces not directly connected to the service network. This setting applies a network policy to restrict access to services to those pods in the active namespace.

For example, if you create a site in the namespace **projectA** of **clusterA** and link that site to a service network where the **database** service is exposed, the **database** service is available to pods in **projectB** of **clusterA**.

You can use the **--create-network-policy** option to restrict the **database** service access to **projectA** of **clusterA**.

CHAPTER 4. LINKING SITES

A service network consists of Application Interconnect sites. This section describes how to link sites to form a service network.

Linking two sites requires a single initial directional connection. However:

- Communication between the two sites is bidirectional, only the initial linking is directional.
- The choice of direction for linking is typically determined by accessibility. For example, if you are linking an OpenShift Dedicated cluster with a CodeReady Containers cluster, you must link from the CodeReady Containers cluster to the OpenShift Dedicated cluster because that route is accessible.

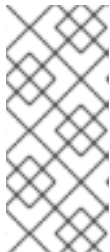
Procedure

1. Determine the direction of the link. If both clusters are publicly addressable, then the direction is not significant. If one of the clusters is addressable from the other cluster, perform step 2 below on the addressable cluster.
2. Generate a token on the cluster that you want to link to:

```
$ skupper token create <filename>
```

where **<filename>** is the name of a YAML file that is saved on your local filesystem.

This file contains a key and the location of the site that created it.



NOTE

Access to this file provides access to the service network. Protect it appropriately.

For more information about protecting access to the service network, see [Using Skupper tokens](#).

3. Use a token on the cluster that you want to connect from:
 - a. Create a link to the service network:

```
$ skupper link create <filename> [-name <link-name>]
```

where **<filename>** is the name of a YAML file generated from the **skupper token create** command and **<link-name>** is the name of the link.

- b. Check the link:

```
$ skupper link status
Connection for link1 not active
```

In this example no **<link-name>** was specified, the name defaulted to **link1**.

4. If you want to delete a link:

```
$ skupper link delete <link-name>
```

-

where **<link-name>** is the name of the link specified during creation.

CHAPTER 5. EXPOSING SERVICES ON THE SERVICE NETWORK FROM A NAMESPACE

After creating a service network, exposed services can communicate across that network.

The **skupper** CLI has two options for exposing services that already exist in a namespace:

- **expose** supports simple use cases, for example, a deployment with a single service. See [Section 5.1, “Exposing simple services on the service network”](#) for instructions.
- **service create** and **service bind** is a more flexible method of exposing services, for example, if you have multiple services for a deployment. See [Section 5.2, “Exposing complex services on the service network”](#) for instructions.

5.1. EXPOSING SIMPLE SERVICES ON THE SERVICE NETWORK

This section describes how services can be enabled for a service network for simple use cases.

Procedure

1. Create a deployment, some pods, or a service in one of your sites, for example to create the backend service for the [tutorial](#):

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

This step is not Application Interconnect-specific, that is, this process is unchanged from standard processes for your cluster.

2. Create a service that can communicate on the service network:

```
$ skupper expose [deployment <name>|pods <selector>|statefulset <statefulsetname>|service <name>]
```

where

- **<name>** is the name of your deployment
- **<selector>** is a pod selector
- **<statefulsetname>** is the name of a statefulset

For the example deployment in step 1, you create a service using the following command:

```
$ skupper expose deployment/hello-world-backend --port 8080
```

Options for this command include:

- **--port <port-number>::** Specify the port number that this service is available on the service network. NOTE: You can specify more than one port by repeating this option.
- **--target-port <port-number>::** Specify the port number of pods that you want to expose.
- **--protocol <protocol>** allows you specify the protocol you want to use, **tcp**, **http** or **http2**

**NOTE**

If you do not specify ports, **skupper** uses the **containerPort** value of the deployment.

5.2. EXPOSING COMPLEX SERVICES ON THE SERVICE NETWORK

This section describes how services can be enabled for a service network for more complex use cases.

Procedure

1. Create a deployment, some pods, or a service in one of your sites, for example to create the backend service for the [tutorial](#):

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

This step is not Application Interconnect-specific, that is, this process is unchanged from standard processes for your cluster.

2. Create a service that can communicate on the service network:

```
$ skupper service create <name> <port>
```

where

- **<name>** is the name of the service you want to create
- **<port>** is the port the service uses

For the example deployment in step 1, you create a service using the following command:

```
$ skupper service create hello-world-backend 8080
```

3. Bind the service to a cluster service:

```
$ skupper service bind <service-name> <target-type> <target-name>
```

where

- **<service-name>** is the name of the service on the service network
- **<target-type>** is the object you want to expose, **deployment**, **statefulset**, **pods**, or **service**.
- **<target-name>** is the name of the cluster service
- **--protocol <protocol>** allows you specify the protocol you want to use, **tcp**, **http** or **http2**

For the example deployment in step 1, you bind the service using the following command:

```
$ skupper service bind hello-world-backend deployment hello-world-backend
```

CHAPTER 6. EXPOSING SERVICES ON THE SERVICE NETWORK FROM A LOCAL MACHINE

After creating a service network, you can expose services from a local machine on the service network.

For example, if you run a database on a server in your data center, you can deploy a front end in a cluster that can access the data as if the database was running in the cluster.

6.1. EXPOSING SIMPLE LOCAL SERVICES TO THE SERVICE NETWORK

This section shows how to expose a single service running locally on a service network.

Prerequisites

- A service network. Only one site is required.
- Access to the service network.
- **skrouterd** for the default **service** type gateways

Procedure

1. Run your service locally.
2. Log into your cluster and change to the namespace for your site.
3. Expose the service on the service network:

```
$ skupper gateway expose <service> localhost <port>
```

- <service> - the name of the service on the service network.
- <port> - the port that runs the service locally.



NOTE

You can also expose services from other machines on your local network, for example if MySQL is running on a dedicated server (with an IP address of **192.168.1.200**), but you are accessing the cluster from a machine in the same network:

```
$ skupper gateway expose mysql 192.168.1.200 3306
```

4. Check the status of Skupper gateways:

```
$ skupper gateway status
```

Gateway Definition:

```
└─ machine-user type:service version:1.18.0
   └─ Bindings:
      └─ mydb:3306 tcp mydb:3306 127.0.0.1 3306
```

This shows that there is only one exposed service and that service is only exposing a single port (BIND). There are no ports forwarded to the local host.

The URL field shows the underlying communication and can be ignored.

6.2. WORKING WITH COMPLEX LOCAL SERVICES ON THE SERVICE NETWORK

This section shows more advanced usage of skupper gateway.

1. Create a Skupper gateway:

```
$ skupper gateway init --type <gateway-type>
```



NOTE

The default service type gateway requires that **skrouterd** is running.

By default a *service* type gateway is created, however you can also specify:

- **podman**
- **docker**

2. Create a service that can communicate on the service network:

```
$ skupper service create <name> <port>
```

where

- **<name>** is the name of the service you want to create
- **<port>** is the port the service uses

For example:

```
$ skupper service create mydb 3306
```

3. Bind the service on the service network:

```
$ skupper gateway bind <service> <host> <port>
```

- **<service>** - the name of the service on the service network, **mydb** in the example above.
- **<host>** - the host that runs the service.
- **<port>** - the port the service is running on, **3306** from the example above.

4. Check the status of Skupper gateways:

```
$ skupper gateway status
Gateway Definitions Summary
```

```
Gateway Definition:
```



```
└─ machine-user type:service version:1.18.0
   └─ Bindings:
      └─ mydb:3306 tcp mydb:3306 127.0.0.1 3306
```

This shows that there is only one exposed service and that service is only exposing a single port (BIND). There are no ports forwarded to the local host.

The URL field shows the underlying communication and can be ignored.

You can create more services in the service network and bind more local services to expose those services on the service network.

5. Forward a service from the service network to the local machine.

```
$ skupper gateway forward <service> <port>
```

where

- **<service>** is the name of an existing service on the service network.
- **<port>** is the port on the local machine that you want to use.

6.3. CREATING A GATEWAY AND APPLYING IT ON A DIFFERENT MACHINE

If you have access to a cluster from one machine but want to create a gateway to the service network from a different machine, you can create the gateway definition bundle on the first machine and later apply that definition bundle on a second machine as described in this procedure. For example, if you want to expose a local database service to the service network, but you never want to access the cluster from the database server, you can use this procedure to create the definition bundle and apply it on the database server.

Procedure

1. Log into your cluster from the first machine and change to the namespace for your site.
2. Create a service that can communicate on the service network:

```
$ skupper service create <name> <port>
```

where

- **<name>** is the name of the service you want to create
- **<port>** is the port the service uses

For example:

```
$ skupper service create database 5432
```

3. Create a YAML file to represent the service you want to expose, for example:

```
name: database 1
bindings:
```

```

- name: database ❷
  host: localhost ❸
  service:
    address: database:5432 ❹
    protocol: tcp ❺
    ports:
      - 5432 ❻
  target_ports:
    - 5432 ❼
  qdr-listeners:
    - name: amqp
      host: localhost
      port: 5672

```

- ❶ Gateway name, useful for reference only.
- ❷ Binding name, useful to track multiple bindings.
- ❸ Name of host providing the service you want to expose.
- ❹ Service name and port on service network. You created the service in a previous step.
- ❺ The protocol you want to use to expose the service, **tcp**, **http** or **http2**.
- ❻ The port on the service network that you want this service to be available on.
- ❼ The port of the service running on the host specified in point 3.

4. Save the YAML file using the name of the gateway, for example, **gateway.yaml**.
5. Generate a bundle that can be applied to the machine that hosts the service you want to expose on the service network:

```
$ skupper gateway generate-bundle <config-filename> <destination-directory>
```

where:

- <config-filename> - the name of the YAML file, including suffix, that you generated in the previous step.
- <destination-directory> - the location where you want to save the resulting gateway bundle, for example **~/gateways**.

For example:

```
$ skupper gateway generate-bundle database.yaml ./
```

This bundle contains the gateway definition YAML and a certificate that allow access to the service network.

6. Copy the gateway definition file, for example, **mylaptop-jdoe.tar.gz** to the machine that hosts the service you want to expose on the service network.
7. From the machine that hosts the service you want to expose:

■

```
$ mkdir gateway
$ tar -xvf <gateway-definition-file> --directory gateway
$ cd gateway
$ sh ./launch.py
```

**NOTE**

Use **`./launch.py -t podman`** or **`./launch.py -t docker`** to run the Application Interconnect router in a container.

Running the gateway bundle uses the gateway definition YAML and a certificate to access and expose the service on the service network.

8. Check the status of the gateway service:

To check a *service* type gateway:

```
$ systemctl --user status <gateway-definition-name>
```

To check a *podman* type gateway:

```
$ podman inspect
```

To check a *docker* type gateway:

```
$ docker inspect
```

**NOTE**

You can later remove the gateway using **`./remove.py`**.

9. From the machine with cluster access, check the status of Skupper gateways:

```
$ skupper gateway status
Gateway Definitions Summary

NAME   BINDS  FORWARDS  URL
<machine-name>  1      0      amqp://127.0.0.1:5672
```

This shows that there is only one exposed service and that service is only exposing a single port (BIND). There are no ports forwarded to the local host.

**NOTE**

If you need to change the gateway definition, for example to change port, you need to remove the existing gateway and repeat this procedure from the start to redefine the gateway.

6.4. GATEWAY YAML REFERENCE

The [Section 6.3, “Creating a gateway and applying it on a different machine”](#) describes how to create a gateway to apply on a separate machine using a gateway definition YAML file.

The following are valid entries in a gateway definition YAML file.

name

Name of gateway

bindings.name

Name of binding for a single host.

bindings.host

Hostname of local service.

bindings.service

Definition of service you want to be available on service network.

bindings.service.address

Address on the service network, name and port.

bindings.service.protocol

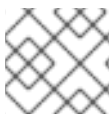
Skupper protocol, **tcp**, **http** or **http2**.

bindings.service.ports

A single port that becomes available on the service network.

bindings.service.target_ports

A single port that you want to expose on the service network.

**NOTE**

If the local service requires more than one port, create separate bindings for each port.

forwards.name

Name of forward for a single host.

forwards.host

Hostname of local service.

forwards.service

Definition of service you want to be available locally.

forwards.service.address

Address on the service network that you want to use locally, name and port.

forwards.service.protocol

Skupper protocol, **tcp**, **http** or **http2**.

forwards.service.ports

A single port that is available on the service network.

forwards.service.target_ports

A single port that you want to use locally.

**NOTE**

If the network service requires more than one port, create separate forwards for each port.

qdr-listeners

Definition of skupper router listeners

qdr-listeners.name

Name of skupper router, typically **amqp**.

qdr-listeners.host

Hostname for skupper router, typically **localhost**.

qdr-listeners.port

Port for skupper router, typically **5672**.

CHAPTER 7. EXPLORING A SERVICE NETWORK

Application Interconnect includes a command to allow you report all the sites and the services available on a service network.

Prerequisites

- A service network with more than one site

Procedure

1. Set your Kubernetes context to a namespace on the service network.
2. Use the following command to report the status of the service network:

```
$ skupper network status
```

For example, the following shows the output for the service network created from the [Creating a service network with OpenShift](#) tutorial from the **west** namespace:

```
Sites:
├─ [local] 4dba248 - west ❶
│  │ URL: 10.96.146.236 ❷
│  │ name: west ❸
│  │ namespace: west
│  │ version: 0.8.6 ❹
│  └─ Services:
│     └─ name: hello-world-backend ❺
│        │ address: hello-world-backend: 8080 ❻
│        │ protocol: tcp ❼
└─ [remote] bca99d1 - east ❽
   │ URL:
   │ name: east
   │ namespace: east
   │ sites linked to: 4dba248-west ❾
   │ version: 0.8.6
   │ └─ Services:
   │    └─ name: hello-world-backend
   │       │ address: hello-world-backend: 8080
   │       │ protocol: tcp
   │       └─ Targets:
   │          └─ name: hello-world-backend-7dfb45b98d-mhskw ❿
```

- ❶ The unique identifier of the site associated with the current context, that is, the **west** namespace
- ❷ The URL of the service network router. This is required for other sites to connect to this site and is different from the console URL. If you require the URL of the console, use the **skupper status** command to display that URL.
- ❸ The site name. By default, skupper uses the name of the current namespace. If you want to specify a site name, use **skupper init --site-name <site-name>**.

- 4 The version of Application Interconnect running the site. The site version can be different from the current **skupper** CLI version. To update a site to the version of the CLI, use
- 5 The name of a service exposed on the service network.
- 6 The address of a service exposed on the service network.
- 7 The protocol of a service exposed on the service network.
- 8 The unique identifier of a remote site on the service network.
- 9 The sites that the remote site is linked to.
- 10 The name of the local Kubernetes object that is exposed on the service network. In this example, this is the **hello-world-backend** pod.



NOTE

The URL for the east site has no value because that site was initialized without ingress using the following command:

```
$ skupper init --ingress none
```

CHAPTER 8. SECURING A SERVICE NETWORK

Application Interconnect provides default, built-in security that scales across clusters and clouds. This section describes additional security you can configure.

See [Securing a service network using policies](#) for information about creating granular policies for each cluster.

8.1. RESTRICTING ACCESS TO SERVICES USING NETWORK-POLICY

By default, if you expose a service on the service network, that service is also accessible from other namespaces in the cluster. You can avoid this situation when creating a site using the **--create-network-policy** option.

Procedure

1. Create the service network router with a network policy:

```
$ skupper init --create-network-policy
```

2. 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.
```

You can now expose services on the service network and those services are not accessible from other namespaces in the cluster.

8.2. APPLYING TLS TO HTTP2 TRAFFIC ON THE SERVICE NETWORK

By default, the traffic between sites is encrypted, however the traffic between the service pod and the router pod is not encrypted. For services exposed as HTTP2, the traffic between the pod and the router pod can be encrypted using TLS.

Prerequisites

- Two or more linked sites
- A HTTP2 frontend and backend service

Procedure

1. Deploy your backend service.
2. Expose your backend deployment on the service network, enabling TLS, for example:

```
$ skupper expose deployment <deployment-name> --port 443 --protocol http2 --enable-tls
```

Enabling TLS creates the necessary certificates required for TLS backends and stores them in a secret named **skupper-tls-<deployment-name>**.

3. Modify the backend deployment to include the generated certificates, for example:

```
...
spec:
  containers:
    ...
    command:
      ...
      - "/certs/tls.key"
      - "/certs/tls.crt"
      ...
    volumeMounts:
      ...
      - mountPath: /certs
        name: certs
        readOnly: true
  volumes:
    - name: index-html
      configMap:
        name: index-html
    - name: certs
      secret:
        secretName: skupper-tls-<deployment-name>
```

Each site creates the necessary certificates required for TLS clients and stores them in a secret named **skupper-service-client**.

4. Modify the frontend deployment to include the generated certificates, for example:

```
spec:
  template:
    spec:
      containers:
        ...
        volumeMounts:
          - name: certs
            mountPath: /tmp/certs/skupper-service-client
        ...
      volumes:
        - name: certs
          secret:
            secretName: skupper-service-client
```

5. Test calling the service from a TLS enabled frontend.

CHAPTER 9. SUPPORTED STANDARDS AND PROTOCOLS

Application Interconnect supports the following protocols for your service network:

- TCP – default
- HTTP1
- HTTP2

When exposing or creating a service, you can specify the protocol, for example:

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

where **<protocol>** can be:

- tcp
- http
- http2

When choosing which protocol to specify, note the following:

- **tcp** supports any protocol overlayed on TCP, for example, HTTP1 and HTTP2 work when you specify **tcp**.
- If you specify **http** or **http2**, the IP address reported by a client may not be accessible.
- All service network traffic is converted to AMQP messages in order to traverse the service network.
TCP is implemented as a single streamed message, whereas HTTP1 and HTTP2 are implemented as request/response message routing.

CHAPTER 10. CLI OPTIONS FOR WORKING WITH DIFFERENT CLUSTERS

By default, all **skupper** commands apply to the cluster you are logged into and the current namespace. The following **skupper** options allow you to override that behavior and apply to all commands:

--namespace <namespace-name>

Apply command to **<namespace-name>**. For example, if you are currently working on **frontend** namespace and want to initialize a site in the **backend** namespace:

```
$ skupper init --namespace backend
```

--kubeconfig <kubeconfig-path>

Path to the kubeconfig file - This allows you run multiple sessions to a cluster from the same client. An alternative is to set the **KUBECONFIG** environment variable. See the [tutorial](#) for an example of using kubeconfig files.

--context <context-name>

The kubeconfig file can contain defined contexts, and this option allows you to use those contexts.

CHAPTER 11. USING SKUPPER TOKENS

Skupper tokens allow you to create links between sites. You create a token on one site and use that token from the other site to create a link between the two sites.



NOTE

Although the linking process is directional, a Skupper link allows communication in both directions.

If both sites are equally accessible, for example, two public clouds, then it is not important where you create the token. However, when using a token, the site you link to must be accessible from the site you link from. For example, if you are creating a service network using both a public and private cluster, you must create the token on the public cluster and use the token from the private cluster.

There are two types of Skupper token:

Claim token (default)

A claim token can be restricted by:

- time - prevents token reuse after a specified period.
- usage - prevents creating multiple links from a single token.

All inter-site traffic is protected by mutual TLS using a private, dedicated certificate authority (CA). A claim token is not a certificate, but is securely exchanged for a certificate during the linking process. By implementing appropriate restrictions (for example, creating a single-use claim token), you can avoid the accidental exposure of certificates.

Cert token

You can use a cert token to create a link to the site which issued that token, it includes a valid certificate from that site.

All inter-site traffic is protected by mutual TLS using a private, dedicated certificate authority (CA). A cert token is a certificate issued by the dedicated CA. Protect it appropriately.

11.1. CREATING CLAIM TOKENS

You can use a claim token to create a link to the site which issued that token. It does not include a certificate from that site, but a certificate is passed from the site when the claim token is used. A claim token can be restricted by time or usage.

Procedure

1. Log into the cluster.
2. Change to the namespace associated with the site.
3. Create a claim token, for example:

```
$ skupper token create $HOME/secret.yaml --expiry 30m0s --uses 2 -t claim
```

**NOTE**

Claim tokens are the default, the **-t claim** section of the command is unnecessary.

--expiry

The amount of time the token is valid in minutes and seconds, default **15m0s**.

--uses

The number of times you can use the token to create a link, default **1**.

Additional information

- See the [Configuring Application Interconnect sites using the CLI](#) for information on using the token to create links.

11.2. CREATING CERT TOKENS

A cert token allows you create many links to the service network from different sites without restrictions.

Procedure

1. Log into the cluster.
2. Change to the namespace associated with the site.
3. Create a cert token:

```
$ skupper token create $HOME/secret.yaml -t cert
```

**NOTE**

Cert tokens are always valid and can be reused indefinitely unless revoked as described in [Section 11.3, "Revoking access to a site"](#)

Additional information

- See the [Configuring Application Interconnect sites using the CLI](#) for information on using the token to create links.

11.3. REVOKING ACCESS TO A SITE

If a token is compromised, you can prevent unauthorized use of that token by invalidating all the tokens created from a site.

This option removes all links to the site and requires that you recreate any links to restore the service network.

1. Procedure
2. Log into the cluster.
3. Change to the namespace associated with the site.

4. Check the status of the site:

```
$ skupper status
Skupper is enabled for namespace "west" in interior mode. It is linked to 2 other sites.
```

5. Check outgoing links from the site:

```
$ skupper link status
Link link1 is active
```

In this case, there are two links, and one outgoing link, meaning there is one incoming link.

6. Revoke access to the site from incoming links:

```
$ skupper revoke-access
```

7. Check the status of the site to see the revocation of access:

```
$ skupper status
Skupper is enabled for namespace "west" in interior mode. It is linked to 1 other site.
$ skupper link status
Link link1 is active
```

The output shows that the **skupper revoke-access** command has revoked the incoming links, but outgoing links are still active.

You can remove that link using the **skupper link delete link1** command, but to revoke access, you must follow this procedure while logged into the appropriate cluster.

Additional information

- See the [Configuring Application Interconnect sites using the CLI](#).

CHAPTER 12. TROUBLESHOOTING A SERVICE NETWORK

Typically, you can create a service network without referencing this troubleshooting guide. However, this guide provides some tips for situations when the service network does not perform as expected.

A typical troubleshooting workflow is to check all the sites and create debug tarballs.

12.1. CHECKING SITES

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

Procedure

1. Check the site status:

```
$ skupper status --namespace west
```

Skupper is enabled for namespace "west" in interior mode. It is connected to 2 other sites. It has 1 exposed services.

The output shows:

- A site exists in the specified namespace.
- A link exists to two other sites.
- A service is exposed on the service network and is accessible from this namespace.

2. Check the link status:

```
$ skupper link status --namespace east
```

Link link1 is active

A link exists from the specified site to another site, meaning a token from another site was applied to the specified site.



NOTE

Running **skupper link status** on a connected site produces output only if a token was used to create a link.

3. Check the service network:

```
$ skupper network status --namespace west
```

Sites:

```
|— [local] 05f8c38 - west
|  URL: 10.110.15.54
|  mode: interior
|  name: west
|  namespace: west
|  version: 1.0.2
```

```

└─ Services:
    └─ name: backend
        address: backend: 8080
        protocol: tcp
└─ [remote] 1537b82 - east
    URL: 10.97.26.100
    name: east
    namespace: east
    sites linked to: 05f8c38-west
    version: 1.0.2
    └─ Services:
        └─ name: backend
            address: backend: 8080
            protocol: tcp
            └─ Targets:
                └─ name: backend-77f8f45fc8-smckp
                └─ name: backend-77f8f45fc8-gh6tp
                └─ name: backend-77f8f45fc8-m58tg

```

The output shows:

- There are 2 sites on the service network, **east** and **west**.
- Details for each site, for example the namespace names.
- The original services that are exposed (Targets), in this case the three backend services exposed using the **tcp** protocol.
- The services available on the service network, including the port number. For example, **backend:8080**.

12.2. CREATING A SKUPPER DEBUG TARBALL

The debug tarball contains all the logs from the Application Interconnect components for a site and provides detailed information to help debug issues.

1. Create the debug tarball:

```
$ skupper debug dump my-site
```

Skupper dump details written to compressed archive: `my-site.tar.gz`

2. You can expand the file using the following command:

```
$ tar -xvf kind-site.tar.gz
```

```

k8s-versions.txt
skupper-versions.txt
skupper-router-deployment.yaml
skupper-router-867f5ddcd8-plrcg-skstat-g.txt
skupper-router-867f5ddcd8-plrcg-skstat-c.txt
skupper-router-867f5ddcd8-plrcg-skstat-l.txt
skupper-router-867f5ddcd8-plrcg-skstat-n.txt
skupper-router-867f5ddcd8-plrcg-skstat-e.txt
skupper-router-867f5ddcd8-plrcg-skstat-a.txt

```



```
skupper-router-867f5ddcd8-plrcg-skstat-m.txt
skupper-router-867f5ddcd8-plrcg-skstat-p.txt
skupper-router-867f5ddcd8-plrcg-router-logs.txt
skupper-router-867f5ddcd8-plrcg-config-sync-logs.txt
skupper-service-controller-deployment.yaml
skupper-service-controller-7485756984-gvrf6-events.txt
skupper-service-controller-7485756984-gvrf6-service-controller-logs.txt
skupper-site-configmap.yaml
skupper-services-configmap.yaml
skupper-internal-configmap.yaml
skupper-sasl-config-configmap.yaml
```

These files can be used to provide support for Application Interconnect, however some items you can check:

versions

See ***versions.txt** for the versions of various components.

ingress

See **skupper-site-configmap.yaml** to determine the **ingress** type for the site.

linking and services

See the **skupper-service-controller-*-events.txt** file to view details of token usage and service exposure.

Revised on 2022-08-08 15:36:29 UTC