# Skupper

vlatest

2023-08-10

# Preview homepage

This is a preview of skupperproject/skupper-docs: Documentation for the Skupper project

This documentation is built using Antora and contains the following Antora modules:

- ROOT - Metadata like this page and images
- cli - Kubernetes and Podman CLI procedures
- cli-reference - Kubernetes CLI reference (generated)
- console - Console docs
- declarative - Using Skupper with YAML
- overview - Skupper concepts
- operator - Using Skupper with a Kubernetes operator
- policy - Control Skupper usage in a cluster using the policy CRD
- podman-reference - Podman CLI reference (generated)
- troubleshooting - Resolving problems in your service network

> ℹ This ROOT module is never published but contains partials that are used in various documents.

# <strong>Basics</strong>

# Overview

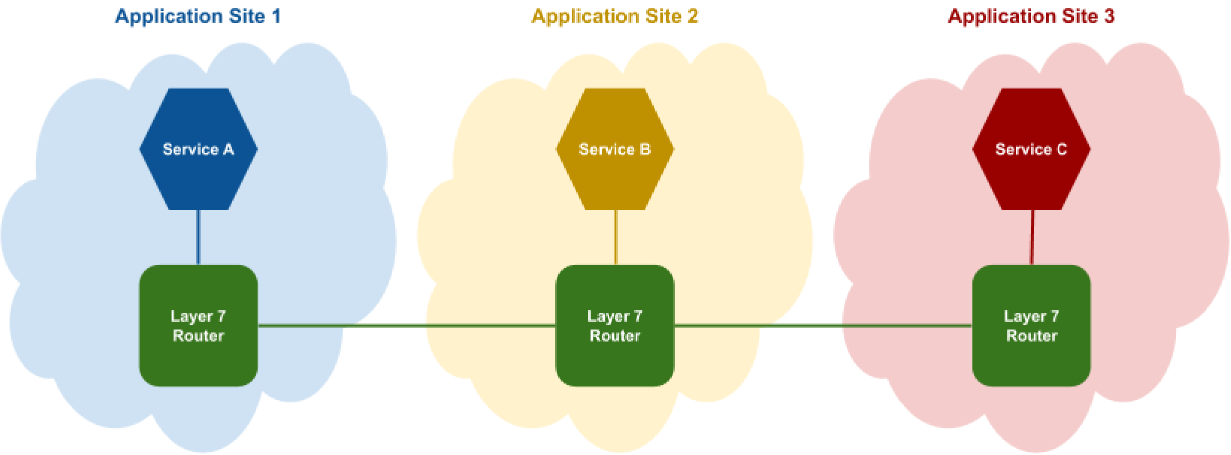
Hybrid clouds enable organizations to combine on-premises, private cloud, and public cloud resources. While such a solution provides many benefits, it also presents a unique challenge: enabling these resources to communicate with each other.

Skupper provides a solution to this challenge with a Virtual Application Network that simply and securely connects applications running in different network locations.

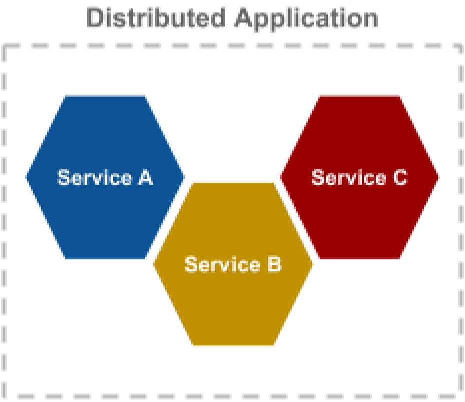
# Virtual Application Networks

Skupper solves multi-cluster communication challenges through something called a Virtual Application Network (VAN). To understand the value of Skupper, it is helpful to first understand what a VAN is.

A VAN connects the applications and services in your hybrid cloud into a virtual network so that they can communicate with each other as if they were all running in the same site. In this diagram, a VAN connects three services, each of which is running in a different cloud:



In essence, the VAN connects the services in a distributed application with a microservice architecture.



VANs are able to provide connectivity across the hybrid cloud because they operate at Layer 7 (the application layer). They use **Layer 7 application routers** to route communication between **Layer 7 application addresses**.

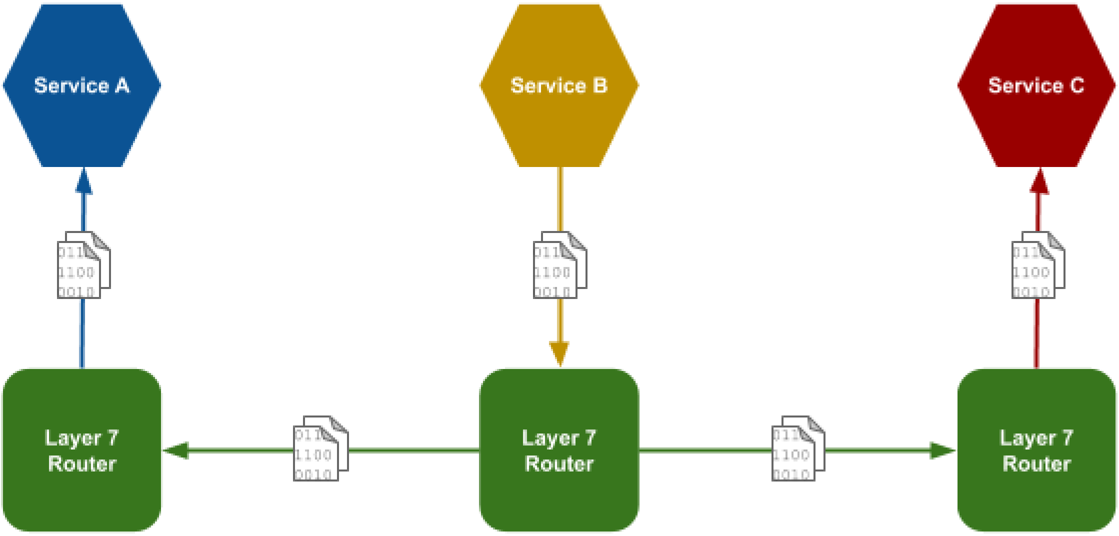
## Layer 7 application routers

Layer 7 application routers form the backbone of a VAN in the same way that conventional network routers form the backbone of a VPN. However, instead of routing IP packets between network endpoints, Layer 7 application routers route messages between application endpoints (called Layer 7 application addresses).

## Layer 7 application addresses

A Layer 7 application address represents an endpoint, or destination in the VAN. When an application sends a communication to an address, the Layer 7 application routers distribute the communication to any other application in the VAN that has the same address.

For example, in this diagram, **Service B** sends a message with an application address to its local application router. **Service A** and **Service C** are subscribed to the same address, so the application router routes copies of the message through the VAN until they arrive at each destination.

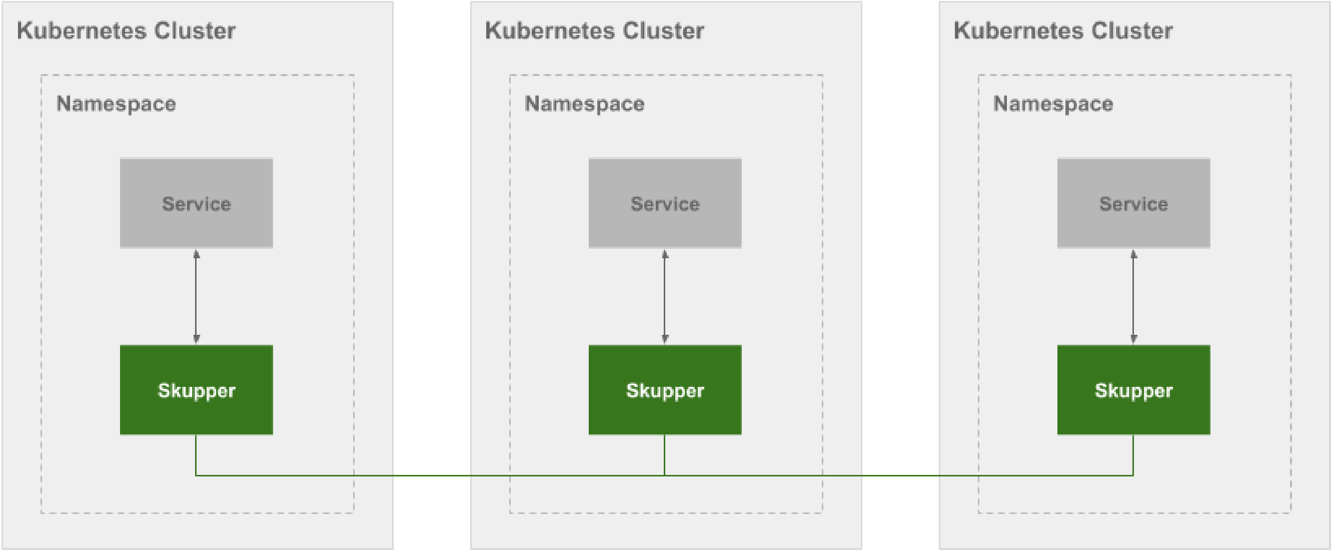


VANs provide multiple routing patterns, so communications can be distributed in anycast (balanced or closest) or multicast patterns.

## Skupper

Skupper is an open source tool for creating VANs in Kubernetes. By using Skupper, you can create a distributed application consisting of microservices running in different Kubernetes clusters.

This diagram illustrates a Skupper network that connects three services running in three different Kubernetes clusters:



In a Skupper network, each namespace contains a Skupper instance. When these Skupper instances connect, they continually share information about the services that each instance exposes. This means that each Skupper instance is always aware of every service that has been exposed to the Skupper network, regardless of the namespace in which each service resides.

Once a Skupper network is formed across Kubernetes namespaces, any of the services in those namespaces can be exposed (through annotation) to the Skupper network. When a service is exposed, Skupper creates proxy endpoints to make that service available on each namespace in the Skupper network.

*Additional information*

- Security
- Routing
- Connectivity

# Security

Skupper securely connects your services with TLS authentication and encryption. See how Skupper enables you to deploy your application securely across Kubernetes clusters.

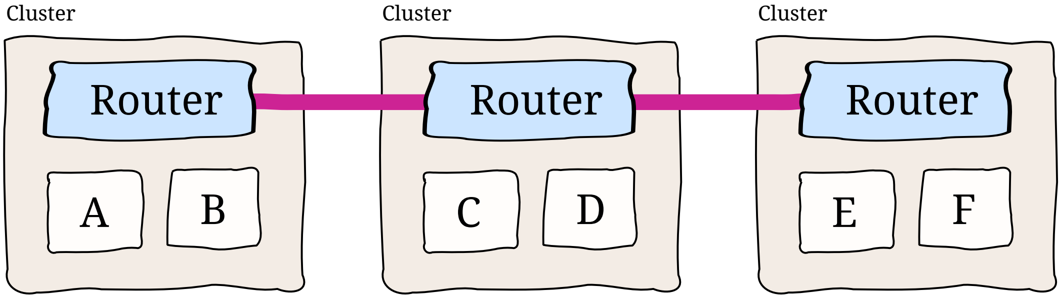
## Security challenges in the cloud

Moving an application to the cloud raises security risks. Either your services must be exposed to the public internet, or you must adopt complex layer 3 network controls like VPNs, firewall rules, and access policies.

Increasing the challenge, layer 3 network controls do not extend easily to multiple clusters. These network controls must be duplicated for each cluster.

## Built-in network isolation

Skupper provides default, built-in security that scales across clusters and clouds. In a Skupper network, the connections between Skupper routers are secured with mutual TLS using a private, dedicated certificate authority (CA). Each router is uniquely identified by its own certificate.



This means that the Skupper network is isolated from external access, preventing security risks such as lateral attacks, malware infestations, and data exfiltration.

*Additional information*

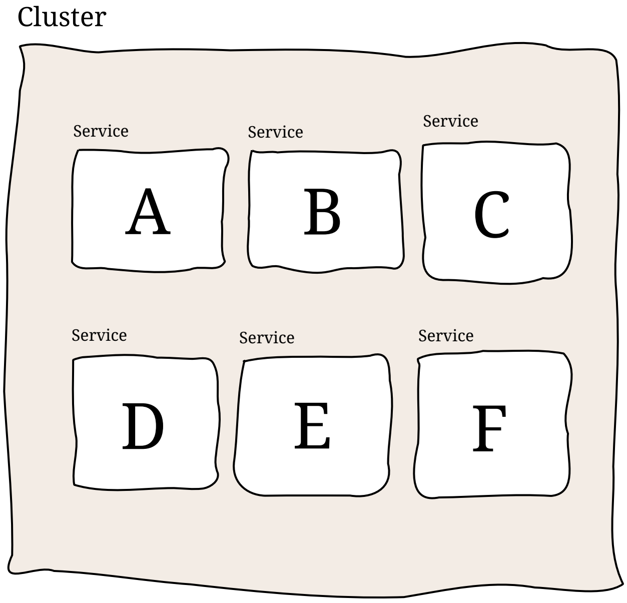
- Overview
- Connectivity
- Routing

# Connectivity

Skupper represents a new approach to connecting services across multiple Kubernetes clusters. See how Skupper can give you the flexibility to deploy your services where you need them.

## One cluster

Kubernetes **services** provide a virtual network address for each element of your distributed application. Service "A" can contact service "B", "B" can contact "C", and so on.

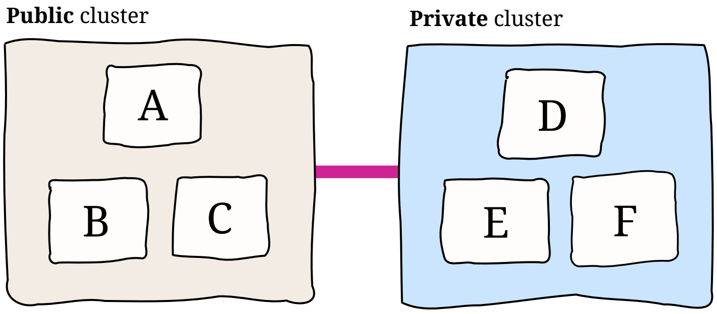


But if you want to deploy your application across multiple clusters, your options are limited. You have to either expose your services to the public internet or set up a VPN.

Skupper offers a third way. It connects clusters to a secure layer 7 network. It uses that network to forward local service traffic to remote clusters.

## Secure hybrid cloud communication

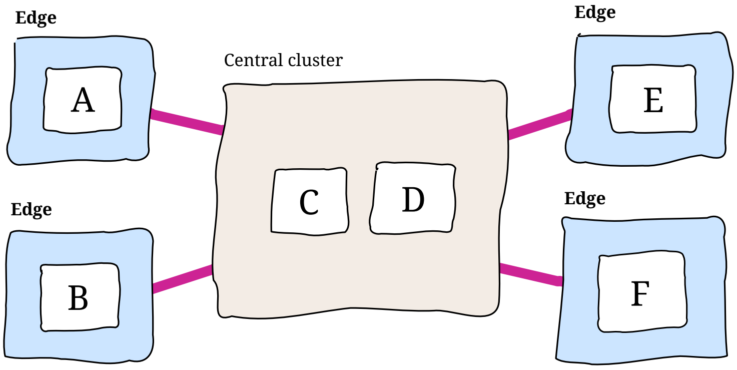
Deploy your application across public and private clusters.



You can host your database on a private cluster and retain full connectivity with services running on the public cloud. All communication is secured by mutual TLS authentication and encryption.

# Edge-to-edge connectivity

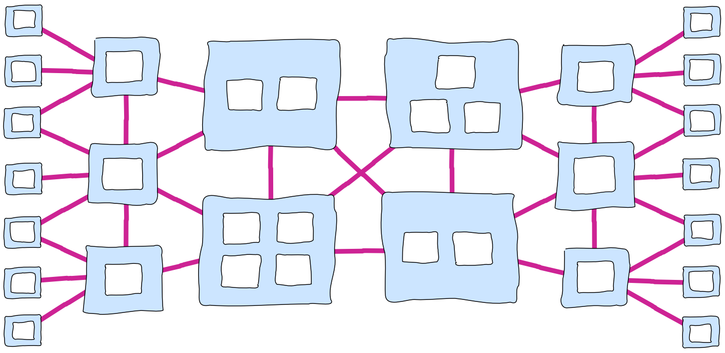
Distribute application services across geographic regions.



You can connect multiple retail sites to a central office. Once connected, each edge location can contact any other edge. You can add and remove sites on demand.

# Scale up and out

Build large, robust networks of connected clusters.



*Additional information*

- Overview
- Routing
- Connectivity

# Routing

Skupper uses layer 7 addressing and routing to connect services. See how the power of application-layer addressing can bring new capabilities to your applications.

## Multi-cluster services

Deploy a single logical service across multiple clusters.

Skupper can route requests to instances of a single service running on multiple clusters. If a provider or data center fails, the service instances running at unaffected sites can scale to meet the need and maintain availability.

## Dynamic load balancing

Balance requests across clusters according to service capacity.

The Skupper network has cross-cluster visibility. It can see which services are already loaded and which have spare capacity, and it directs requests accordingly.

You can assign a cost to each inter-cluster connection. This enables you to configure a preference for one resource over another. If demand is normal, you can keep all traffic on your private cloud. If demand peaks, you can dynamically spill over to public cloud resources.

## Reliable networks

Skupper uses redundant network paths and smart routing to provide highly available connectivity at scale.

*Additional information*

- Overview
- Security
- Connectivity

# <strong>Kubernetes</strong>

# Getting started

# CLI guide

Using the `skupper` command-line interface (CLI) allows you to create and manage Skupper 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.

## Installing the Skupper CLI

Installing the `skupper` command-line interface (CLI) provides a simple method to get started with Skupper.

*Procedure*

1. Install the `skupper` command-line interface.

   For Linux:

   ```
   $ curl -fL
   https://github.com/skupperproject/skupper/releases/download/1.4.2/skupper-cli-
   1.4.2-linux-amd64.tgz | tar -xzf -
   ```

   For MacOS:

   ```
   $ curl -fL
   https://github.com/skupperproject/skupper/releases/download/1.4.2/skupper-cli-
   1.4.2-mac-amd64.tgz | tar -xzf -
   ```

2. Copy the `skupper` executable to a directory in your $PATH:

   ```
   $ mkdir -p $HOME/bin
   $ export PATH=$PATH:$HOME/bin
   $ mv skupper $HOME/bin
   ```

3. Verify the installation.

   ```
   $ skupper version
   client version 1.4
   ```

## Creating a site using the CLI

A service network consists of Skupper 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
   ```

   Starting with Skupper release 1.3, the console is not enabled by default. To use the new console, which is a preview feature and may change, see Console.

2. Check the site:

   ```
   $ skupper status

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

   > 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)`.

By default, the site name defaults to the namespace name, for example, `west`.

# Custom sites

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

Starting with Skupper release 1.3, the console is not enabled by default. To use the new console, which is a preview feature and may change, see Console.

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

- Configuring console authentication. There are several `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 Skupper 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 passwords is generated.

- Configuring service access

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

> 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`.

# Linking sites

A service network consists of Skupper 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.

> 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
   Link link1 not connected
   ```

   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.

# Specifying link cost

When linking sites, you can assign a cost to each link to influence the traffic flow. By default, link cost is set to `1` for a new link. In a service network, the routing algorithm attempts to use the path with the lowest total cost from client to target server.

- If you have services distributed across different sites, you might want a client to favor a particular target or link. In this case, you can specify a cost of greater than `1` on the alternative links to reduce the usage of those links.

  > The distribution of open connections is statistical, that is, not a round robin

system.

- If a connection only traverses one link, then the path cost is equal to the link cost. If the connection traverses more than one link, the path cost is the sum of all the links involved in the path.
- Cost acts as a threshold for using a path from client to server in the network. When there is only one path, traffic flows on that path regardless of cost.

> If you start with two targets for a service, and one of the targets is no longer available, traffic flows on the remaining path regardless of cost.

- When there are a number of paths from a client to server instances or a service, traffic flows on the lowest cost path until the number of connections exceeds the cost of an alternative path. After this threshold of open connections is reached, new connections are spread across the alternative path and the lowest cost path.

*Prerequisite*

- You have set your Kubernetes context to a site that you want to link *from*.
- A token for the site that you want to link *to*.

*Procedure*

1. Create a link to the service network:

```
$ skupper link create <filename> --cost <integer-cost>
```

where `<integer-cost>` is an integer greater than 1 and traffic favors lower cost links.

> If a service can be called without traversing a link, that service is considered local, with an implicit cost of `0`.

For example, create a link with cost set to `2` using a token file named `token.yaml`:

```
$ skupper link create token.yaml --cost 2
```

2. Check the link cost:

```
$ skupper link status link1 --verbose

 Cost:          2
 Created:       2022-11-17 15:02:01 +0000 GMT
 Name:          link1
 Namespace:     default
 Site:          default-0d99d031-cee2-4cc6-a761-697fe0f76275
 Status:        Connected
```

3. Observe traffic using the console.

   If you have a console on a site, log in and navigate to the processes for each server. You can view the traffic levels corresponding to each client.

   > ℹ If there are multiple clients on different sites, filter the view to each client to determine the effect of cost on traffic. For example, in a two site network linked with a high cost with servers and clients on both sites, you can see that a client is served by the local servers while a local server is available.

# 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 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 Exposing complex services on the service network for instructions.

## 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:

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

   This step is not Skupper-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`

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

## 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:

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

   This step is not Skupper-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

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
```

## Exposing services from a different namespace to the service network

This section shows how to expose a service from a namespace where Skupper is not deployed.

Skupper allows you expose Kubernetes services from other namespaces for any site. However, if you want to expose workloads, for example deployments, you must create a site as described in this section.

*Prerequisites*

- A namespace where Skupper is deployed.
- A network policy that allows communication between the namespaces
- cluster-admin permissions if you want to expose resources other than services

1. Create a site with cluster permissions if you want to expose a workload from a namespace other than the site namespace:

   ```
   $ skupper init --enable-cluster-permissions
   ```

2. Expose the service on the service network:

   > The site does not require the extra permissions granted with the `--enable-cluster-permissions` to expose a Kubernetes service.

   a. If you want to expose a Kubernetes service from a namespace other than the site namespace:

      ```
      $ skupper expose service <service>.<namespace> --address <service>
      ```

      - <service> - the name of the service on the service network.
      - <namespace> - the name of the namespace where the service you want to expose runs.

For example, if you deployed Skupper in the `east` namespace and you created a `backend` Kubernetes service in the `east-backend` namespace, you set the context to the `east` namespace and expose the service as `backend` on the service network using:

```
$ skupper expose service backend.east-backend --port 8080 --address backend
```

b. If you want to expose a workload and you created a site with `--enable-cluster -permissions`:

```
$ skupper expose <resource> --port <port-number> --target-namespace
<namespace>
```

- <resource> - the name of the resource.
- <namespace> - the name of the namespace where the resource you want to expose runs.

For example, if you deployed Skupper in the `east` namespace and you created a `backend` deployment in the `east-backend` namespace, you set the context to the `east` namespace and expose the service as `backend` on the service network using:

```
$ skupper expose deployment/backend --port 8080 --target-namespace east-
backend
```

# 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.

## 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.

*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.

> ℹ 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
   ▯── Bindings:
      ▯── mydb:3306 tcp mydb:3306 localhost 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.

## 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>
```

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
   └── Bindings:
      └── mydb:3306 tcp mydb:3306 localhost 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.

## 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 ②
         host: localhost ③
         service:
           address: database:5432 ④
           protocol: tcp ⑤
           ports:
               - 5432 ⑥
         target_ports:
           - 5432 ⑦
   qdr-listeners:
       - name: amqp
         host: localhost
         port: 5672
   ```

   ① Gateway name, useful for reference only.

   ② Binding name, useful to track multiple bindings.

   ③ Name of host providing the service you want to expose.

④ Service name and port on service network. You created the service in a previous step.

⑤ The protocol you want to use to expose the service, `tcp`, `http` or `http2`.

⑥ The port on the service network that you want this service to be available on.

⑦ 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
   ```

   Use `./launch.py -t podman` or `./launch.py -t docker` to run the Skupper 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
```

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 Definition:
└─ machine-user type:service version:1
   └─ Bindings:
      └─ mydb:3306 tcp mydb:3306 localhost 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.

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.

## Gateway YAML reference

The 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.

> ℹ 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.

> ℹ 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`.

# Exploring a service network

Skupper 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:

```
Sites:
 ├── [local] 4dba248 - west  ①
 │   URL: 10.96.146.236 ②
 │   name: west ③
 │   namespace: west
 │   version: 0.8.6 ④
 │   └── Services:
 │       └── name: hello-world-backend ⑤
 │           address: hello-world-backend: 8080 ⑥
 │           protocol: tcp ⑦
└── [remote] bca99d1 - east ⑧
    URL:
    name: east
    namespace: east
    sites linked to: 4dba248-west ⑨
    version: 0.8.6
    └── Services:
        └── name: hello-world-backend
            address: hello-world-backend: 8080
            protocol: tcp
            └── Targets:
                └── name: hello-world-backend-7dfb45b98d-mhskw ⑩
```

① The unique identifier of the site associated with the current context, that is, the `west` namespace

② 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.

③ 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 Skupper 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 `skupper update`.

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.

> The URL for the east site has no value because that site was initialized without ingress using the following command:
>
> ```
> $ skupper init --ingress none
> ```

# Securing a service network

Skupper 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.

## 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.

## Applying TLS to TCP or 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 TCP or HTTP2, the traffic between the pod and the router pod can be encrypted using TLS.

*Prerequisites*

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

*Procedure*

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

   For example, if you want to expose a TCP service:

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

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

1. 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`.

2. 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
```

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

# Supported standards and protocols

Skupper 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.

# CLI options

For a full list of options, see the Kubernetes and Podman reference documentation.

> ⚠ When you create a site and set logging level to `trace`, you can inadvertently log sensitive information from HTTP headers.
>
> ```
> $ skupper init --router-logging trace
> ```

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.

**`--context <context-name>`**

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

# CLI reference

*Options*

```
-c, --context string      The kubeconfig context to use
-h, --help                help for skupper
--kubeconfig string   Path to the kubeconfig file to use
-n, --namespace string    The Kubernetes namespace to use
--platform string     The platform type to use [kubernetes, podman]
```

*See also*

- skupper completion - Output shell completion code for bash
- skupper debug - Debug skupper installation
- skupper delete - Delete skupper installation
- skupper expose - Expose a set of pods through a Skupper address
- skupper gateway - Manage skupper gateway definitions
- skupper init - Initialise skupper installation
- skupper link - Manage skupper links definitions
- skupper network - Show information about the sites and services included in the network.
- skupper revoke-access - Revoke all previously granted access to the site.
- skupper service - Manage skupper service definitions
- skupper status - Report the status of the current Skupper site
- skupper token - Manage skupper tokens
- skupper unexpose - Unexpose a set of pods previously exposed through a Skupper address
- skupper update - Update skupper installation version
- skupper version - Report the version of the Skupper CLI and services

# YAML configuration reference

Using YAML files to configure Skupper allows you to use source control to track and manage Skupper network changes.

## Installing Skupper using YAML

Installing Skupper using YAML provides a declarative method to install Skupper. You can store your YAML files in source control to track and manage Skupper network changes.

*Prerequisites*

- Access to a Kubernetes cluster

*Procedure*

1. Log into your cluster. If you are deploying Skupper to be available for all namespaces, verify you have `cluster-admin` privileges.
2. Deploy the site controller:
   - To install Skupper into the current namespace deploy the site controller using the following YAML:

     ```
     kubectl apply -f deploy-watch-current-ns.yaml
     ```

     where the contents of `deploy-watch-current-ns.yaml` is specified in the YAML for watching current namespace appendix.
   - To install Skupper for all namespaces:
     a. Create a namespace named `skupper-site-controller`.
     b. Deploy the site controller using the following YAML:

        ```
        kubectl apply -f deploy-watch-all-ns.yaml
        ```

        where the contents of `deploy-watch-all-ns.yaml` is specified in the YAML for watching all namespaces appendix.
3. Verify the installation.

   ```
   $ oc get pods
   NAME                                        READY   STATUS    RESTARTS   AGE
   skupper-site-controller-84694bdbb5-n8slb    1/1     Running   0          75s
   ```

## Creating a Skupper site using YAML

Using YAML files to create Skupper sites allows you to use source control to track and manage Skupper network changes.

*Prerequisites*

- Skupper is installed in the cluster or namespace you want to target.
- You are logged into the cluster.

*Procedure*

1. Create a YAML file to define the site, for example, `my-site.yaml`:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: skupper-site
   data:
   ```

```
  name: my-site
  console: "true"
  console-user: "admin"
  console-password: "changeme"
  flow-collector: "true"
```

The YAML creates a site with a console and you can create tokens from this site.

To create a site that has no ingress:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: skupper-site
data:
  name: my-site
  ingress: "false"
```

2. Apply the YAML file to your cluster:

```
kubectl apply -f ~/my-site.yml
```

*Additional resources*

See the Site ConfigMap YAML reference section for more reference.

# Linking sites using YAML

While it is not possible to declaratively link sites, you can create a token using YAML.

*Prerequisites*

- Skupper is installed on the clusters you want to link.
- You are logged into the cluster.

*Procedure*

1. Log into the cluster you want to link to and change context to the namespace where Skupper is installed. This site must have `ingress` enabled.
2. Create a YAML file named `token-request.yml` to request a token:

```
apiVersion: v1
kind: Secret
metadata:
  labels:
    skupper.io/type: connection-token-request
  name: secret-name
```

3. Apply the YAML to the namespace to create a secret.

   ```
   $ kubectl apply -f token-request.yml
   ```

4. Create the token YAML from the secret.

   ```
   $ kubectl get secret -o yaml secret-name | yq 'del(.metadata.namespace)' >
   ~/token.yaml
   ```

5. Log into the cluster you want to link from and change context to the namespace where Skupper is installed.

6. Apply the token YAML.

   ```
   $ kubectl apply -f token.yml
   ```

7. Verify the link, allowing some time for the process to complete.

   ```
   $ skupper link status --wait 60
   ```

# Configuring services using annotations

After creating and linking sites, you can use Kubernetes annotations to control which services are available on the service network.

## Exposing simple services on a service network using annotations

This section provides an alternative to the `skupper expose` command, allowing you to annotate existing resources to expose simple services on the service network.

*Prerequisites*

- A site with a service you want to expose

*Procedure*

1. Log into the namespace in your cluster that is configured as a site.

2. Create a deployment, some pods, or a service in one of your sites, for example:

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

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

3. Annotate the kubernetes resource to create a service that can communicate on the service

network, for example:

```
$ kubectl annotate deployment backend "skupper.io/address=backend"
"skupper.io/port=8080" "skupper.io/proxy=tcp"
```

The annotations include:

- `skupper.io/proxy` - the protocol you want to use, `tcp`, `http` or `http2`. This is the only annotation that is required. For example, if you annotate a simple deployment named `backend` with `skupper.io/proxy=tcp`, the service is exposed as `backend` and the `containerPort` value of the deployment is used as the port number.
- `skupper.io/address` - the name of the service on the service network.
- `skupper.io/port` - one or more ports for the service on the service network.

When exposing services, rather than other resources like deployments, you can use the `skupper.io/target` annotation to avoid modifying the original service. For example, if you want to expose the `backend` service:

```
$ kubectl annotate service backend "skupper.io/address=van-backend"
"skupper.io/port=8080" \
"skupper.io/proxy=tcp" "skupper.io/target=backend"
```

This allows you to delete and recreate the `backend` service without having to apply the annotation again.

4. Check that you have exposed the service:

```
$ skupper service status
Services exposed through Skupper:
▯── backend (tcp port 8080)
   ▯── Targets:
      ▯── app=hello-world-backend name=hello-world-backend
```

The related targets for services are only displayed when the target is available on the current cluster.

## Understanding Skupper annotations

Annotations allow you to expose services on the service network. This section provides details on the scope of those annotations

### skupper.io/address

The name of the service on the service network. Applies to:

- Deployments

- StatefulSets
- DaemonSets
- Services

**skupper.io/port**

The port for the service on the service network. Applies to:

- Deployments
- StatefulSets
- DaemonSets

**skupper.io/proxy**

The protocol you want to use, `tcp`, `http` or `http2`. Applies to:

- Deployments
- StatefulSets
- DaemonSets
- Services

**skupper.io/target**

The name of the target service you want to expose. Applies to:

- Services

**skupper.io/service-labels**

A comma separated list of label keys and values for the exposed service. You can use this annotation to set up labels for monitoring exposed services. Applies to:

- Deployments
- DaemonSets
- Services

# Appendix A: Site ConfigMap YAML reference

Using YAML files to configure Skupper requires that you understand all the fields so that you provision the site you require.

The following YAML defines a Skupper site:

```
apiVersion: v1
data:
  name: my-site ①
  console: "true" ②
  flow-collector: "true" ③
  console-authentication: internal ④
  console-user: "username" ⑤
  console-password: "password" ⑥
  cluster-local: "false" ⑦
  edge: "false" ⑧
  service-sync: "true" ⑨
  ingress: "true" ⑩
kind: ConfigMap
metadata:
  name: skupper-site
```

① Specifies the site name.

② Enables the skupper console, defaults to `false`. NOTE: You must enable `console` and `flow-collector` for the console to function.

③ Enables the flow collector, defaults to `false`.

④ Specifies the skupper console authentication method. The options are `openshift`, `internal`, `unsecured`.

⑤ Username for the `internal` authentication option.

⑥ Password for the `internal` authentication option.

⑦ Only accept connections from within the local cluster, defaults to `false`.

⑧ Specifies whether an edge site is created, defaults to `false`.

⑨ Specifies whether the services are synchronized across the service network, defaults to `true`.

⑩ Specifies whether the site supports ingress, for example, to create tokens usable from remote sites.

ℹ All ingress types are supported using the same parameters as the `skupper` CLI.

# Appendix B: YAML for watching current namespace

The following example deploys Skupper to watch the current namespace.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: skupper-site-controller
  labels:
    application: skupper-site-controller
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    application: skupper-site-controller
  name: skupper-site-controller
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - pods
  - pods/exec
  - services
  - secrets
  - serviceaccounts
  - events
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
  - patch
- apiGroups:
  - apps
  resources:
  - deployments
  - statefulsets
  - daemonsets
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
- apiGroups:
  - route.openshift.io
  resources:
  - routes
  verbs:
  - get
  - list
  - watch
  - create
  - delete
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  - networkpolicies
  verbs:
  - get
  - list
  - watch
  - create
  - delete
- apiGroups:
  - projectcontour.io
  resources:
  - httpproxies
  verbs:
  - get
  - list
  - watch
  - create
  - delete
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - rolebindings
  - roles
  verbs:
  - get
  - list
  - watch
  - create
  - delete
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    application: skupper-site-controller
  name: skupper-site-controller
subjects:
- kind: ServiceAccount
  name: skupper-site-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: skupper-site-controller
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: skupper-site-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      application: skupper-site-controller
  template:
    metadata:
      labels:
        application: skupper-site-controller
    spec:
      serviceAccountName: skupper-site-controller
      # Please ensure that you can use SeccompProfile and do not use
      # if your project must work on old Kubernetes
      # versions < 1.19 or on vendors versions which
      # do NOT support this field by default
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containers:
      - name: site-controller
        image: quay.io/skupper/site-controller:master
        securityContext:
          capabilities:
            drop:
            - ALL
          runAsNonRoot: true
          allowPrivilegeEscalation: false
        env:
        - name: WATCH_NAMESPACE
          valueFrom:
             fieldRef:
               fieldPath: metadata.namespace
```

# Appendix C: YAML for watching all namespaces

The following example deploys Skupper to watch all namespaces.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: skupper-site-controller
  namespace: skupper-site-controller
  labels:
    application: skupper-site-controller
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    application: skupper-site-controller
  name: skupper-site-controller
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - pods
  - pods/exec
  - services
```

```
    - secrets
    - serviceaccounts
    verbs:
    - get
    - list
    - watch
    - create
    - update
    - delete
  - apiGroups:
    - apps
    resources:
    - deployments
    - statefulsets
    - daemonsets
    verbs:
    - get
    - list
    - watch
    - create
    - update
    - delete
  - apiGroups:
    - route.openshift.io
    resources:
    - routes
    verbs:
    - get
    - list
    - watch
    - create
    - delete
  - apiGroups:
    - networking.k8s.io
    resources:
    - ingresses
    - networkpolicies
    verbs:
    - get
    - list
    - watch
    - create
    - delete
  - apiGroups:
    - projectcontour.io
    resources:
    - httpproxies
    verbs:
    - get
    - list
    - watch
    - create
    - delete
  - apiGroups:
    - rbac.authorization.k8s.io
    resources:
    - rolebindings
    - roles
    verbs:
    - get
    - list
    - watch
    - create
    - delete
    - update
  - apiGroups:
    - rbac.authorization.k8s.io
    resources:
    - clusterrolebindings
    verbs:
    - create
  - apiGroups:
    - rbac.authorization.k8s.io
    resources:
    - clusterroles
    verbs:
    - bind
    resourceNames:
    - skupper-service-controller
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    labels:
      application: skupper-site-controller
    name: skupper-site-controller
  subjects:
  - kind: ServiceAccount
    name: skupper-site-controller
    namespace: skupper-site-controller
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: skupper-site-controller
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: skupper-site-controller
    namespace: skupper-site-controller
  spec:
    replicas: 1
    selector:
      matchLabels:
        application: skupper-site-controller
    template:
      metadata:
        labels:
          application: skupper-site-controller
      spec:
        serviceAccountName: skupper-site-controller
        # Please ensure that you can use SeccompProfile and do not use
        # if your project must work on old Kubernetes
        # versions < 1.19 or on vendors versions which
        # do NOT support this field by default
        securityContext:
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        containers:
        - name: site-controller
          image: quay.io/skupper/site-controller:1.3.0
          securityContext:
            capabilities:
              drop:
              - ALL
            runAsNonRoot: true
            allowPrivilegeEscalation: false
```

# Using the Skupper operator

The Skupper Operator creates and manages Skupper sites in Kubernetes.

You can install the Operator as described in Installing the Operator using the CLI.

> Installing an Operator requires administrator-level privileges for your Kubernetes cluster.

After installing the Operator, you can create a site by deploying a ConfigMap as described in Creating a site using the Skupper Operator

## Installing the Operator using the CLI

The steps in this section show how to use the `kubectl` command-line interface (CLI) to install and deploy the latest version of the Skupper Operator in a given Kubernetes cluster.

*Prerequisites*

- The Operator Lifecycle Manager is installed in the cluster. For more information, see the QuickStart.

*Procedure*

1. Download the Skupper Operator example files, for example:

   ```
   $ wget https://github.com/skupperproject/skupper-
   operator/archive/refs/heads/main.zip
   ```

2. Create a `my-namespace` namespace. NOTE: If you want to use a different namespace, you need to edit the referenced YAML files.

   a. Create a new namespace:

      ```
      $ kubectl create namespace my-namespace
      ```

   b. Switch context to the namespace:

      ```
      $ kubectl config set-context --current --namespace=my-namespace
      ```

3. Create a CatalogSource in the `openshift-marketplace` namespace:

   ```
   $ kubectl apply -f examples/k8s/00-cs.yaml
   ```

4. Verify the skupper-operator catalog pod is running before continuing:

```
$ kubectl -n olm get pods | grep skupper-operator
```

5. Create an OperatorGroup in the `my-namespace` namespace:

   ```
   $ kubectl apply -f examples/k8s/10-og.yaml
   ```

6. Create a Subscription in the `my-namespace` namespace:

   ```
   $ kubectl apply -f examples/k8s/20-sub.yaml
   ```

7. Verify that the Operator is running:

   ```
   $ kubectl get pods -n my-namespace

   NAME                                     READY    STATUS     RESTARTS    AGE
   skupper-site-controller-d7b57964-gxms6   1/1      Running    0           1m
   ```

   If the output does not report the pod is running, use the following command to determine the issue that prevented it from running:

   ```
   $ kubectl describe pod -l name=skupper-operator
   ```

## Creating a site using the Skupper Operator

1. Create a YAML file defining the ConfigMap of the site you want to create.

   For example, create `skupper-site.yaml` that provisions a site with a console:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: skupper-site
     namespace: my-namespace
   data:
     console: "true"
     flow-collector: "true"
     console-user: "admin"
     console-password: "changeme"
   ```

   ℹ The console is a preview feature and may change before becoming fully supported by skupper.io. Currently, you must enable the console on the same site as you enable the flow collector. This requirement may change before the

> console is fully supported by skupper.io.

You can also create a site without a console:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: skupper-site
  namespace: my-namespace
```

2. Apply the YAML to create a ConfigMap named `skupper-site` in the namespace you want to use:

```
$ kubectl apply -f skupper-site.yaml
```

3. Verify that the site is created by checking that the Skupper router and service controller pods are running:

```
$ kubectl get pods

NAME                                        READY   STATUS    RESTARTS   AGE
skupper-router-8c6cc6d76-27562              1/1     Running   0          40s
skupper-service-controller-57cdbb56c5-vc7s2 1/1     Running   0          34s
```

> ℹ If you deployed the Operator to a single namespace, an additional site controller pod is also running.

# Creating sites using a custom certificate authority

By default:

- Network traffic between pods and the Skupper router is not encrypted. To encrypt traffic between pods and the Skupper router see Encrypting traffic from a pod to the Skupper router.
- Skupper creates certificates to establish links between sites using mutual TLS. This ensures that traffic between sites is encrypted.

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 Skupper 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 a Kubernetes 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
   ```

   > On OpenShift, you can also verify routes are created using:
   >
   > ```
   > $ oc get routes
   > ```

4. Use the following command to check for errors relating to incorrect certificates:

   ```
   $ skupper debug events
   ```

# Encrypting traffic from a pod to the Skupper router

This section describes how to apply certificates to encrypt the traffic within a cluster.

By default:

- Skupper creates certificates to establish links between sites using mutual TLS so that traffic between sites is encrypted. To use custom certificates for traffic between sites, see Creating sites using a custom certificate authority.
- Network traffic between pods and the Skupper router is not encrypted.

You can use certificates to encrypt traffic between pods and the Skupper router as follows:

- Exposing services on the service network with Skupper-generated certificates - does not require that you provide certificates.
- Exposing services on the service network with user-provided certificates - requires that the certificate authority, and the signed certificates, are distributed across all sites.

> With both procedures, you reference the certificates when exposing the service on the service network. If you do not reference certificates, the traffic between pods and the Skupper router is unencrypted.

## Exposing services on the service network with Skupper-generated certificates

A Skupper installation includes a certificate authority which can generate certificates that can be used to encrypt traffic from the pod to the Skupper router. This procedure describes how to use those certificates in your service network.

*Prerequisites*

- Access to the Kubernetes site where the service is exposed
- Access to the Kubernetes site where the service is called from
- service sync is enabled on both sites

*Procedure*

1. Expose the service on the service network:
    a. If you use the `expose` option:

    ```
    $ skupper expose  <target-type> <target-name> --generate-tls-secrets
    ```

    For example, to expose a `backend` deployment using `http2`:

```
$ skupper expose deployment backend --port 8080 --protocol http2 --generate-tls
-secrets
```

b. If you use `create` and `bind` options:

```
$ skupper service create <service-name> --generate-tls-secrets
$ skupper service bind <service-name>  <target-type> <target-name>
```

2. Check that the service is available on another site:

```
$ skupper service status
Services exposed through Skupper:
▯── nghttp2tls (http2 port 443)
```

On this site, a secret is created named `skupper-tls-<service-name>`. The secret contains the generated certificates under `data/ca.crt`, `data/tls.crt`, and `data/tls.key`.

3. Configure components that call the exposed service to use the certificates stored in `skupper-tls-<service-name>`.

For example, modify a deployment to mount the secret in a container.

```
    volumes:
    - name: certs
      secret:
        secretName: skupper-tls-nghttp2tls
```

4. Configure the exposed service, which is the component that responds to the request, to use the certificates stored in `skupper-service-client`.

The `skupper-service-client` secret contains the certificate and private key of the Skupper certificate authority. For example, modify a deployment to mount the secret in a container.

```
    volumes:
    - name: certs
      secret:
        secretName: skupper-service-client
```

## Exposing services on the service network with user-provided certificates

You can encrypt traffic from the pod to the Skupper router using certificates provided by a certificate authority.

*Prerequisites*

- Access to the Kubernetes site where the service is exposed
- Access to the Kubernetes site where the service is called from
- Certificate authority access (intermediate certificate is sufficient)

*Procedure*

1. Create a TLS secret from the certificate authority to store the private key and certificate.

   The required format of the secret is:

   `data/ca.crt`
   : CA TLS certificate

   For example, you might name the secret `ca-tls-secret`:

   ```
   $ kubectl create secret generic ca-tls-secret --from-file=ca.crt=rootCA.crt
   ```

2. Create a secret from the signed certificate and private key files:

   The required format of the secret is:

   `data/ca.crt`
   : CA TLS certificate from step 1

   `data/tls.crt`
   : Signed TLS certificate

   `data/tls.key`
   : Signed Private key

   For example, to encrypt a service named `backend`, you might name the secret `user-tls-backend`:

   ```
   $ kubectl create secret tls user-tls-backend --key <key-path> --cert <cert-path>
   $ kubectl patch secret user-tls-backend  -p="{\"data\":{\"ca.crt\": \"$(kubectl get
   secret ca-tls-secret -o json -o=jsonpath="{.data.tls\.crt}")\"}}"
   ```

3. Expose the service on the service network:

   a. If you use the `expose` option, you specify the certificate secret and the CA secret, for example:

      ```
      $ skupper expose deployment backend --port 5432 --protocol http2 --tls-cert
      user-tls-backend --tls-trust ca-tls-secret
      ```

   b. If you use the `create` and `bind` options:

      ```
      $ skupper service create backend 5432 --tls-cert user-tls-backend
      ```

```
$ skupper bind deployment backend  --port 5001  --protocol http2 --tls-trust ca-tls-secret
```

When certificates expire, you need to perform this procedure again with the new certificates.

# <strong>Podman</strong>

# CLI guide

Skupper podman allow you to create a site using containers, without requiring Kubernetes. Typically, you create a site on a Linux host, allowing you to link to and from other sites, regardless of whether those sites are running in podman or Kubernetes.

> This is a preview feature and may change before becoming fully supported by skupper.io.

## About Skupper podman

Skupper podman is available with the following precedence:

`skupper --platform podman <command>`

Use this option to avoid changing mode, for example, if you are working on Kubernetes and podman simultaneously.

`export SKUPPER_PLATFORM=podman`

Use this command to use Skupper podman for the current session, for example, if you have two terminals set to different contexts. To set the environment to target Kubernetes sites:

```
$ export SKUPPER_PLATFORM=kubernetes
```

`skupper switch podman`

If you enter this command, all subsequent command target podman rather than Kubernetes for all terminal sessions.

To determine which mode is currently active:

```
$ skupper switch

podman
```

To switch back to target Kubernetes sites: `skupper switch kubernetes`

## Creating a site using Skupper podman

*Prerequisites*

- The latest `skupper` CLI is installed.
- Podman is installed, see https://podman.io/

1. Set your session to use Skupper podman:

```
$ export SKUPPER_PLATFORM=podman
```

   To verify the `skupper` mode:

   ```
   $ skupper switch

   podman
   ```

2. Create a Skupper site:

   The simplest Skupper site allows you to link to other sites, but does not support linking *to* the current site.

   ```
   $ skupper init --ingress none

   It is recommended to enable lingering for <username>, otherwise Skupper may not
   start on boot.
   Skupper is now installed for user '<username>'.  Use 'skupper status' to get
   more information.
   ```

   If you require that other sites can link to the site you are creating:

   ```
   $ skupper init --ingress-host <machine-address>

   It is recommended to enable lingering for <username>, otherwise Skupper may not
   start on boot.
   Skupper is now installed for user '<username>'.  Use 'skupper status' to get
   more information.
   ```

   For more information, see podman skupper init.

3. Check the status of your site:

   ```
   $ skupper status
   Skupper is enabled for "<username>" with site name "<machine-name>-<username>"
   in interior mode. It is not connected to any other sites. It has no exposed
   services.
   ```

   You can only create one site per user. If you require a host to support many sites, create a user for each site.

# Linking sites using Skupper podman

The general flow for linking podman sites is the same as for Kubernetes sites:

1. Generate a token on one site:

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

2. Create a link from the other site:

```
$ skupper link create <filename>
```

After you have linked to a network, you can check the link status:

```
$ skupper link status
```

# Working with services using Skupper podman

The general flow for working with services is the same for Kubernetes and Podman sites.

> Services exposed on Kubernetes are not automatically available to Podman sites. This is the equivalent to Kubernetes sites created using `skupper init --enable-service-sync false`.

*Example 01: Consuming a service from a Podman site*

In this variation of the hello world example, the `backend` service is exposed on Kubernetes site and a Podman site is linked. You deploy the `frontend` as a container and that container can access the `backend` service.

1. Create a Podman site and link it to a Kubernetes site.
2. Check the service from the Podman site:

```
$ skupper service status

No services defined
```

   This result is expected because services exposed on Kubernetes are not automatically available to Podman sites.

3. Create a service on the Podman site matching the service exposed on the Kubernetes site:

```
$ skupper service create backend 8080
```

4. Validate the service from the Podman site by checking the backend API health URL:

```
$ podman run -it --rm --network=skupper --name=myubi ubi8/ubi curl
backend:8080/api/health
```

```
OK
```

This command runs a container using the `skupper` network and returns the results from `http://backend:8080/api/health`

5. Run the frontend as a container:

```
$ podman run -dp 8080:8080 --name hello-world-frontend --network skupper
quay.io/skupper/hello-world-frontend
```

6. Check your service network is working as expected by navigating to http://localhost:8080 and click **Say hello**.

   Each of the backend replicas respond, for example `Hi, Perfect Parrot. I am Kind Hearted Component (backend-7c84887f9f-wxhxp).`

> In this scenario, running the `skupper service status` command on the Podman site does not provide much detail about the service:
>
> ```
> $ skupper service status
> Services exposed through Skupper:
> ▯── backend (tcp port 8080)
> ```

*Example 02: Exposing a service from a Podman site*

In this variation of the hello world example, the `backend` service is exposed on Podman site and consumed from a `frontend` on a Kubernetes site.

1. Create a Podman site and link it to a Kubernetes site.
2. Create and expose a frontend deployment on the Kubernetes site:

```
$ kubectl create deployment frontend --image quay.io/skupper/hello-world-frontend
$ kubectl expose deployment/frontend --port 8080 --type LoadBalancer
```

3. Run the backend as a container:

```
$ podman run -d --name hello-world-backend --network skupper quay.io/skupper/hello-
world-backend
```

4. Expose the `backend` from the Podman site.

```
$ skupper expose host hello-world-backend --address backend --port 8080
```

5. From the Kubernetes site, create the `backend` service:

```
$ skupper service create backend 8080
```

6. Check your service network is working as expected by navigating to your cluster URL, port 8080, and clicking **Say hello**.

For more information, see podman skupper expose.

# Podman Reference

*Options*

```
-h, --help              help for skupper
--platform string   The platform type to use [kubernetes, podman]
```

*See also*

- skupper delete - Delete skupper installation
- skupper expose - Expose a set of pods through a Skupper address
- skupper init - Initialise skupper installation
- skupper link - Manage skupper links definitions
- skupper revoke-access - Revoke all previously granted access to the site.
- skupper service - Manage skupper service definitions
- skupper status - Report the status of the current Skupper site
- skupper token - Manage skupper tokens
- skupper unexpose - Unexpose a set of pods previously exposed through a Skupper address
- skupper version - Report the version of the Skupper CLI and services

# <strong>Observability</strong>

# Console

The Skupper console provides data and visualizations of the traffic flow between Skupper sites.

> This is a preview feature and may change before becoming fully supported by skupper.io.

## Enabling the Skupper console

By default, when you create a Skupper site, a Skupper console is not available.

When enabled, the Skupper console URL is displayed whenever you check site status using `skupper status`.

*Prerequisites*

- A Kubernetes namespace where you plan to create a site

*Procedure*

1. Determine which site in your service network is best to enable the console.

   Enabling the console also requires that you enable the flow-collector component, which requires resources to process traffic data from all sites. You might locate the console using the following criteria:

   - Does the service network cross a firewall? For example, if you want the console to be available only inside the firewall, you need to locate the flow-collector and console on a site inside the firewall.
   - Is there a site that processes more traffic than other sites? For example, if you have a *frontend* component that calls a set of services from other sites, it might make sense to locate the flow collector and console on that site to minimize data traffic.
   - Is there a site with more or cheaper resources that you want to use? For example, if you have two sites, A and B, and resources are more expensive on site A, you might want to locate the flow collector and console on site B.

2. Create a site with the flow collector and console enabled:

   ```
   $ skupper init --enable-console --enable-flow-collector
   ```

   > Currently, you must enable the console on the same site as you enable the flow collector. This requirement may change before the console is fully supported by skupper.io.

## Accessing the Skupper console

By default, the Skupper console is protected by credentials available in the `skupper-console-users` secret.

*Procedure*

1. Determine the Skupper console URL using the `skupper` CLI, for example:

```
$ skupper status

Skupper is enabled for namespace "west" in interior mode. It is not connected to
any other sites. It has no exposed services.
The site console url is:  https://skupper-west.apps-crc.testing
```

2. Browse to the Skupper console URL. The credential prompt depends on how the site was created using `skupper init`:
   - Using the `--console-auth unsecured` option, you are not prompted for credentials.
   - Using the `--console-auth openshift` option, you are prompted to enter OpenShift cluster credentials.
   - Using the default or `--console-user <user> --console-password <password>` options, you are prompted to enter those credentials.

3. If you created the site using default settings, that is `skupper init`, a random password is generated for the `admin` user. To retrieve the password the `admin` user:

```
$ kubectl get secret skupper-console-users -o jsonpath={.data.admin} | base64 -d

JNZWzMHtyg
```

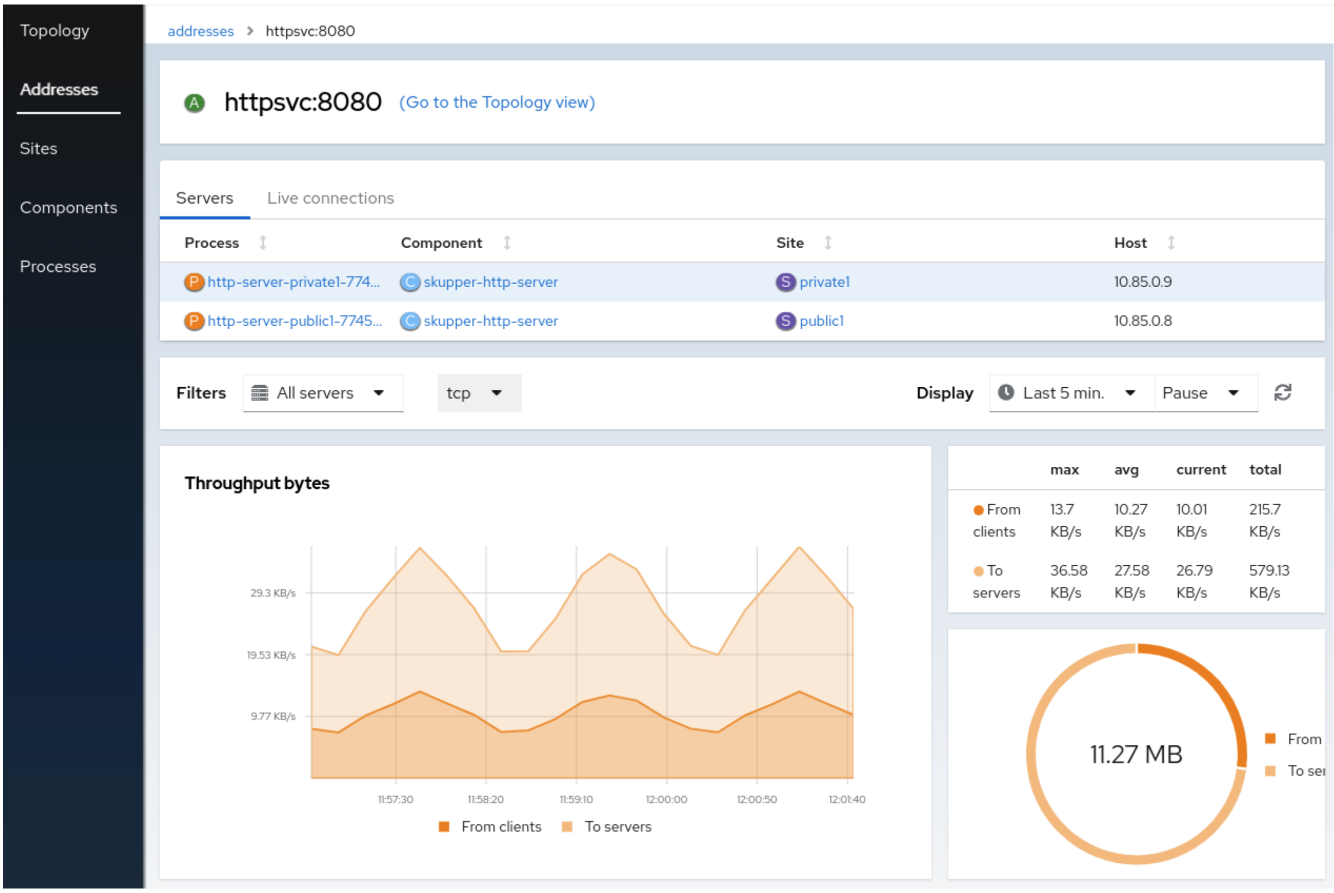
# Exploring the Skupper console

After exposing a service on the service network, you create an *address*, that is, a service name and port number associated with a site. There might be many replicas associated with an address. These replicas are shown in the Skupper console as *processes*. Not all participants on a service network are services. For example, a *frontend* deployment might call an exposed service named *backend*, but that frontend is not part of the service network. In the console, both are shown so that you can view the traffic and these are called *components*.

The Skupper console provides an overview of the following:

- Topology
- Addresses
- Sites
- Components
- Processes

The Skupper console also provides useful networking information about the service network, for example, traffic levels.



A httpsvc:8080

| Process | Component | Site | Host |
|---|---|---|---|
| P http-server-private1-774… | C skupper-http-server | S private1 | 10.85.0.9 |
| P http-server-public1-7745… | C skupper-http-server | S public1 | 10.85.0.8 |

Filters: All servers, tcp  Display: Last 5 min., Pause

| | max | avg | current | total |
|---|---|---|---|---|
| ● From clients | 13.7 KB/s | 10.27 KB/s | 10.01 KB/s | 215.7 KB/s |
| ● To servers | 36.58 KB/s | 27.58 KB/s | 26.79 KB/s | 579.13 KB/s |

1. Check the **Sites** tab. All your sites should be listed. See the **Topology** tab to view how the sites are linked.
2. Check that all the services you exposed are visible in the **Components** tab.
3. Click a component to show the component details and associated processes.
4. Click on a process to display the process traffic.

> The process detail displays the associated image, host, and addresses. You can also view the clients that are calling the process.

5. Click **Addresses** and choose an address to show the details for that address. This shows the set of servers that are exposed across the service network.

> To view information about each window, click the **?** icon.

# <strong>Security</strong>

# 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.

> 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.

# Creating claim tokens

You can use a claim token to create a link to the site which issued that token. It does not includes 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
```

> 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 CLI guide for information about using the token to create links.

## 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
```

> Cert tokens are always valid and can be reused indefinitely unless revoked as described in Revoking access to a site

*Additional information*

- See the CLI guide for information about using the token to create links.

## 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 connected
   ```

   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 connected
   ```

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

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

   After performing the `skupper revoke-access` command, the remote site still retains the link information and returns a `Already connected to <site>` message if you try to recreate the link. To recreate the link, you must first delete the link manually from the remote site context.

# Securing a service network

Skupper 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.

## 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.

## Applying TLS to TCP or 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 TCP or HTTP2, the traffic between the pod and the router pod can be encrypted using TLS.

*Prerequisites*

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

*Procedure*

1. Deploy your backend service.

2. Expose your backend deployment on the service network, enabling TLS.

   For example, if you want to expose a TCP service:

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

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

1. 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`.

2. 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
```

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

# Securing a service network using policies

By default, Skupper includes many security features, including using mutual TLS for all service network communication between sites. By default, applying the policy system to a cluster prevents all service network communication to and from that cluster. You specify granular policies to allow only the service network communication you require.

> The policy system is distinct from the `network-policy` option which restricts access to Skupper services to the current namespace as described in CLI guide.

Each site in a service network runs a Skupper router and has a private, dedicated certificate authority (CA). Communication between sites is secured with mutual TLS, so the service network is isolated from external access, preventing security risks such as lateral attacks, malware infestations, and data exfiltration. The policy system adds another layer at a cluster level to help a cluster administrator control access to a service network.

This guide assumes that you understand the following Skupper concepts:

**site**

A namespace in which Skupper is installed.

**token**

A token is required to establish a link between two sites.

**service network**

After exposing services using Skupper, you have created a service network.

# About the policy system

After a cluster administrator installs the policy system using a Custom Resource Definition (CRD), the cluster administrator needs to configure one or more policies to allow *developers* create and use services on the service network.

> In this guide, *developers* refers to users of a cluster who have access to a namespace, but do not have administrator privileges.

A cluster administrator configures one or more of following items using custom resources (CRs) to enable communication:

**Allow incoming links**

Use `allowIncomingLinks` to enable developers create tokens and configure incoming links.

**Allow outgoing links to specific hosts**

Use `allowedOutgoingLinksHostnames` to specify hosts that developers can create links to.

**Allow services**

Use `allowedServices` to specify which services developers can create or use on the service network.

**Allow resources to be exposed**

Use `allowedExposedResources` to specify which resources a developer can expose on the service network.

> ℹ A cluster administrator can apply each policy CR setting to one or more namespaces.

For example, the following policy CR fully allows all Skupper capabilities on all namespaces, except for:

- only allows outgoing links to any domain ending in `.example.com`.
- only allows 'deployment/nginx' resources to be exposed on the service network.

```
apiVersion: skupper.io/v1alpha1
kind: SkupperClusterPolicy
metadata:
  name: cluster-policy-sample-01
spec:
  namespaces:
    - "*"
  allowIncomingLinks: true
  allowedExposedResources:
    - "deployment/nginx"
  allowedOutgoingLinksHostnames: [".*\\.example.com$"]
  allowedServices:
    - "*"
```

> ℹ You can apply many policy CRs, and if there are conflicts in the items allowed, the most permissive policy is applied. For example, if you apply an additional policy CR with the line `allowedOutgoingLinksHostnames: []`, which does not list any hostnames, outgoing links to `*.example.com` are still permitted because that is permitted in the original CR.

`namespaces`

One or more patterns to specify the namespaces that this policy applies to. Note that you can use Label selectors to match the namespaces.

`allowIncomingLinks`

Specify `true` to allow other sites create links to the specified namespaces.

`allowedOutgoingLinksHostnames`

Specify one or more patterns to determine which hosts you can create links to from the specified namespaces.

`allowedServices`

Specify one or more patterns to determine the permitted names of services allowed on the service network from the specified namespaces.

`allowedExposedResources`

Specify one or more permitted names of resources allowed on the service network from the specified namespaces. Note that patterns are not supported.

> Use regular expressions to create pattern matches, for example:
>
> - `.*\\.com$` matches any string ending in `.com`. A double backslash is required to avoid issues in YAML.
> - `^abc$` matches the string `abc`.

If you create another CR that allows outgoing links for a specific namespace, a user can create a link from that namespace to join a service network. That is, the logic for multiple policy CRs is `OR`. An operation is permitted if any single policy CR permits the operation.

# Installing the policy system CRD

Installing the policy system CRD enables a cluster administrator to enforce policies for service networks.

> If there are existing sites on the cluster, see Installing the policy system CRD on a cluster with existing sites to avoid service network disruption.

*Prerequisites*

- Access to a cluster using a `cluster-admin` account
- The Skupper operator is installed

*Procedure*

1. Log in to the cluster using a `cluster-admin` account.
2. Download the CRD:

```
$ wget
https://raw.githubusercontent.com/skupperproject/skupper/1.4/api/types/crds/skupper
_cluster_policy_crd.yaml
```

3. Apply the CRD:

```
$ kubectl apply -f skupper_cluster_policy_crd.yaml

customresourcedefinition.apiextensions.k8s.io/skupperclusterpolicies.skupper.io
created
```

```
clusterrole.rbac.authorization.k8s.io/skupper-service-controller created
```

4. To verify that the policy system is active, use the `skupper status` command and check that the output includes the following line:

   ```
   Skupper is enabled for namespace "<namespace>" in interior mode (with policies).
   ```

## Installing the policy system CRD on a cluster with existing sites

If the cluster already hosts Skupper sites, note the following before installing the CRD:

- All existing connections are closed. You must apply a policy CR to reopen connections.
- All existing service network services and exposed resources are removed. You must create those resources again.

*Procedure*

To avoid disruption:

1. Plan the CRD deployment for an appropriate time.
2. Search your cluster for sites:

   ```
   $ kubectl get pods --all-namespaces --selector=app=skupper
   ```

3. Document each service and resource exposed on the service network.
4. Install the CRD as described in Installing the policy system CRD. This step closes connections and removes all service network services and exposed resources.
5. If Skupper sites exist in the cluster not created by `cluster-admin`, you must grant permissions to read policies to developers to avoid that site being blocked from the service network.

   For each site namespace:

   ```
   $ kubectl create clusterrolebinding skupper-service-controller-<namespace>
   --clusterrole=skupper-service-controller --serviceaccount=<namespace>:skupper
   -service-controller
   ```

   where `<namespace>` is the site namespace.

6. Create policy CRs as described in Creating policies for the policy system
7. Recreate any services and exposed resources as required.

# Creating policies for the policy system

Policies allow a cluster administrator to control communication across the service network from a cluster.

*Prerequisites*

- Access to a cluster using a `cluster-admin` account.
- The policy system CRD is installed on the cluster.

> *Procedure*
>
> Typically, you create a policy CR that combines many elements from the steps below. See About the policy system for an example CR.

1. Implement a policy to allow incoming links
2. Implement a policy to allow outgoing links to specific hosts
3. Implement a policy to allow specific services
4. Implement a policy to allow specific resources

## Implement a policy to allow incoming links

Use `allowIncomingLinks` to enable developers create tokens and configure incoming links.

*Procedure*

1. Determine which namespaces you want to apply this policy to.
2. Create a CR with `allowIncomingLinks` set to `true` or `false`.
3. Create and apply the CR.

For example, the following CR allows incoming links for all namespaces:

```
apiVersion: skupper.io/v1alpha1
kind: SkupperClusterPolicy
metadata:
  name: allowincominglinks
spec:
  namespaces:
    - "*"
  allowIncomingLinks: true
```

## Implement a policy to allow outgoing links to specific hosts

Use `allowedOutgoingLinksHostnames` to specify hosts that developers can create links to. You cannot create a `allowedOutgoingLinksHostnames` policy to disallow a specific host that was previously allowed.

1. Determine which namespaces you want to apply this policy to.

2. Create a CR with `allowedOutgoingLinksHostnames` set to a pattern of allowed hosts.
3. Create and apply the CR.

For example, the following CR allows links to all subdomains of `example.com` for all namespaces:

```
apiVersion: skupper.io/v1alpha1
kind: SkupperClusterPolicy
metadata:
  name: allowedoutgoinglinkshostnames
spec:
  namespaces:
    - "*"
  allowedOutgoingLinksHostnames: ['.*\\.example\\.com']
```

## Implement a policy to allow specific services

Use `allowedServices` to specify which services a developer can create or use on the service network. You cannot create a `allowedServices` policy to disallow a specific service that was previously allowed.

*Procedure*

1. Determine which namespaces you want to apply this policy to.
2. Create a CR with `allowedServices` set to specify the services allowed on the service network.
3. Create and apply the CR.

For example, the following CR allows users to expose and consume services with the prefix `backend-` for all namespaces:

```
apiVersion: skupper.io/v1alpha1
kind: SkupperClusterPolicy
metadata:
  name: allowedservices
spec:
  namespaces:
    - "*"
  allowedServices: ['^backend-']
```

> ℹ When exposing services, you can use the `--address <name>` parameter of the `skupper` CLI to name services to match your policy.

## Implement a policy to allow specific resources

Use `allowedExposedResources` to specify which resources a developer can expose on the service network. You cannot create a `allowedExposedResources` policy to disallow a specific resource that was previously allowed.

*Procedure*

1. Determine which namespaces you want to apply this policy to.
2. Create a CR with `allowedExposedResources` set to specify resources that a developer can expose on the service network.
3. Create and apply the CR.

For example, the following CR allows you to expose an `nginx` deployment for all namespaces:

```
apiVersion: skupper.io/v1alpha1
kind: SkupperClusterPolicy
metadata:
  name: allowedexposedresources
spec:
  namespaces:
    - "*"
  allowedExposedResources: ['deployment/nginx']
```

For `allowedExposedResources`, each entry must conform to the `type/name` syntax.

# Exploring the current policies for a cluster

As a developer you might want to check which policies are enforced for a particular site.

*Procedure*

1. Log into a namespace where a Skupper site has been initialized.
2. Check whether incoming links are permitted:

   ```
   $ kubectl exec deploy/skupper-service-controller -- get policies incominglink

   ALLOWED POLICY ENABLED ERROR
   ALLOWED BY
   false   true           Policy validation error: incoming links are not allowed
   ```

   In this example incoming links are not allowed by policy.

3. Explore other policies:

   ```
   $ kubectl exec deploy/skupper-service-controller -- get policies
   Validates existing policies

   Usage:
     get policies [command]

   Available Commands:
     expose       Validates if the given resource can be exposed
   ```

```
  incominglink Validates if incoming links can be created
  outgoinglink Validates if an outgoing link to the given hostname is allowed
  service      Validates if service can be created or imported
```

As shown, there are commands to check each policy type by specifying what you want to do, for example, to check if you can expose an nginx deployment:

```
$ kubectl  exec deploy/skupper-service-controller -- get policies expose deployment
nginx
ALLOWED POLICY ENABLED ERROR
ALLOWED BY
false   true           Policy validation error: deployment/nginx cannot be exposed
```

If you allowed an nginx deployment as described in Implement a policy to allow specific resources, the same command shows that the resource is allowed and displays the name of the policy CR that enabled it:

```
$ kubectl  exec deploy/skupper-service-controller -- get policies expose deployment
nginx
ALLOWED POLICY ENABLED ERROR
ALLOWED BY
true    true
allowedexposedresources
```

# <strong>Troubleshooting</strong>

# 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.

See Resolving common problems if you have encountered a specific issue using the `skupper` CLI.

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

## Checking sites

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

*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 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
```

If the output is not what you expected, you might want to check links before proceeding.

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`.

3. Check the status of services exposed on the service network:

```
$ skupper service status
Services exposed through Skupper:
▯── backend (tcp port 8080)
   ▯── Targets:
      ▯── app=backend name=backend
```

The output shows the `backend` service and the related target of that service.

The related targets for services are only displayed when the target is available on the current cluster.

4. List the events for a site:

```
$ skupper debug events
NAME                             COUNT
AGE
GatewayQueryRequest              3
9m12s
                                 3      gateway request
9m12s
```

```
SiteQueryRequest             3
9m12s
                             3     site data request
9m12s
ServiceControllerEvent       9
10m24s
                             2     service event for west/frontend
10m24s
                             1     service event for west/backend
10m26s
                             1     Checking service for: backend
10m26s
                             2     Service definitions have changed
10m26s
                             1     service event for west/skupper-router
11m4s
DefinitionMonitorEvent       15
10m24s
                             2     service event for west/frontend
10m24s
                             1     service event for west/backend
10m26s
                             1     Service definitions have changed
10m26s
                             5     deployment event for west/frontend
10m34s
                             1     deployment event for west/skupper-service-
controller     11m4s
ServiceControllerUpdateEvent 1
10m26s
                             1     Updating skupper-internal
10m26s
ServiceSyncEvent             3
10m26s
                             1     Service interface(s) added backend
10m26s
                             1     Service sync sender connection to
11m4s
                                   amqps://skupper-router-
local.west.svc.cluster.local:5671
                                   established
                             1     Service sync receiver connection to
11m4s
                                   amqps://skupper-router-
local.west.svc.cluster.local:5671
                                   established
IpMappingEvent               5
10m34s
                             1     172.17.0.7 mapped to frontend-6b4688bf56-rp9hc
10m34s
                             2      mapped to frontend-6b4688bf56-rp9hc
```

```
10m54s
                             1     172.17.0.4 mapped to
11m4s
                                   skupper-service-controller-6c97c5cf5d-6nzph
                             1     172.17.0.3 mapped to skupper-router-547dffdcbf-
l8pdc     11m4s
TokenClaimVerification       1
10m59s
                             1     Claim for efe3a241-3e4f-11ed-95d0-482ae336eb38
succeeded 10m59s
```

The output shows sites being linked and a service being exposed on a service network. However, this output is most useful when reporting an issue and is included in the Skupper debug tar file.

*Additional information*

- Checking links

# Checking links

You must link sites before you can expose services on the service network.

By default, tokens expire after 5 minutes and you can only use a token once. Generate a new token if the link is not connected. You can also generate tokens using the `-token-type cert` option for permanent reusable tokens.

This section outlines some advanced options for checking links.

1. Check the link status:

```
$ skupper link status --namespace east

Links created from this site:
-------------------------------
Link link1 is connected
```

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

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

If you use this command on a site where you did not create the link, but there is an incoming link to the site:

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

```
Links created from this site:
-------------------------------
There are no links configured or connected

Currently connected links from other sites:
----------------------------------------
A link from the namespace east on site east(536695a9-26dc-4448-b207-519f56e99b71)
is connected
```

2. Check the verbose link status:

```
$ skupper link status link1 --verbose --namespace east

 Cost:          1
 Created:       2022-10-24 12:50:33 +0100 IST
 Name:          link1
 Namespace:     east
 Site:          east-536695a9-26dc-4448-b207-519f56e99b71
 Status:        Connected
```

   The output shows detail about the link, including a timestamp of when the link was created and the associated relative cost of using the link.

   The status of the link must be `Connected` to allow service traffic.

*Additional information*

- Checking sites

# Checking gateways

By default, `skupper gateway` creates a service type gateway and these gateways run properly after a machine restart.

However, if you create a docker or podman type gateway, check that the container is running after a machine restart. For example:

1. Check the status of Skupper gateways:

```
$ skupper gateway status

Gateway Definition:
└─ machine-user type:podman version:1
   └─ Bindings:
      └─ mydb:3306 tcp mydb:3306 localhost 3306
```

   This shows a podman type gateway.

2. Check that the container is running:

```
$ podman ps
CONTAINER ID  IMAGE                                               COMMAND
CREATED         STATUS             PORTS                       NAMES
4e308ef8ee58  quay.io/skupper/skupper-router:1                 /home/skrouterd/b...  26
seconds ago  Up 27 seconds ago                              machine-user
```

This shows the container running.

To view stopped containers, use `podman ps -a` or `docker ps -a`.

3. Start the container if necessary:

```
$ podman start machine-user
```

# Creating a Skupper debug tar file

The debug tar file contains all the logs from the Skupper components for a site and provides detailed information to help debug issues.

1. Create the debug tar file:

```
$  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 Skupper, 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.

## Improving Skupper router performance

If you encounter Skupper router performance issues, you can scale the Skupper router to address those concerns.

Currently, you must delete and recreate a site to reconfigure the Skupper router.

For example, use this procedure to increase throughput, and if you have many clients, latency.

1. Delete your site or create a new site in a different namespace.

   Note all configuration and delete your existing site:

   ```
   $ skupper delete
   ```

   As an alternative, you can create a new namespace and configure a new site with optimized Skupper router performance. After validating the performance improvement, you can delete and recreate your original site.

2. Create a site with optimal performance CPU settings:

   ```
   $ skupper init --router-cpu 5
   ```

3. Recreate your configuration from step 1, recreating links and services.

While you can address availability concerns by scaling the number of routers and somewhat improve performance, for example `skupper init --routers 3`, typically this is not necessary.

# Resolving common problems

The following issues and workarounds might help you debug simple scenarios when evaluating Skupper.

### Cannot initialize skupper

If the `skupper init` command fails, consider the following options:

- Check the load balancer.

  If you are evaluating Skupper on minikube, use the following command to create a load balancer:

  ```
  $ minikube tunnel
  ```

  For other Kubernetes flavors, see the documentation from your provider.

- Initialize without ingress.

  This option prevents other sites from linking to this site, but linking outwards is supported. Once a link is established, traffic can flow in either direction. Enter the following command:

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

  See https://skupper.io/skupper/latest/cli-reference/skupper_init.html for more options.

### Cannot link sites

To link two sites, one site must be accessible from the other site. For example, if one site is behind a firewall and the other site is on an AWS cluster, you must:

1. Create a token on the AWS cluster site.
2. Create the link on the site inside the firewall.

By default, a token is valid for only 15 minutes and can only be used once. See Using Skupper tokens for more information on creating different types of tokens.

### Cannot access Skupper console

Starting with Skupper release 1.3, the console is not enabled by default. To use the new console, which is a preview feature and may change, see Console.

Use `skupper status` to find the console URL.

Use the following command to display the password for the `admin` user:doctype: article

```
$ kubectl get secret/skupper-console-users -o jsonpath={.data.admin} | base64 -d
```

### Cannot create a token for linking clusters

There are several reasons why you might have difficulty creating tokens:

**Site not ready**

After creating a site, you might see the following message when creating a token:

```
Error: Failed to create token: Policy validation error: Skupper is not enabled in
namespace
```

Use `skupper status` to verify the site is working and try to create the token again.

**No ingress**

You might see the following note after using the `skupper token create` command:

```
Token written to <path> (Note: token will only be valid for local cluster)
```

This output indicates that the site was deployed without an ingress option. For example `skupper init --ingress none`. You must specify an ingress to allow sites on other clusters to link to your site.

You can also use the `skupper token create` command to check if an ingress was specified when the site was created.