Red Hat Service Interconnect 1.8

Examples

Service network tutorials with the CLI and YAML

Last Updated: 2024-11-26

Red Hat Service Interconnect 1.8 Examples

Service network tutorials with the CLI and YAML

Legal Notice

Abstract

Red Hat Service Interconnect is a Red Hat build of the open source Skupper project. This Skupper documentation is reproduced for reference.

Table of Contents

CHAPTER 1. INTRODUCTION TO EXAMPLES

Featured

Chapter 2, *Skupper Hello World*

A minimal multi-service HTTP application deployed across sites.

Chapter 8, *Patient Portal*

A database-backed web application deployed across sites.

Chapter 9, *Trade Zoo*

A Kafka-based trading application deployed across sites.

Messaging

Chapter 3, *Accessing ActiveMQ using Skupper*

Access an ActiveMQ message broker.

Chapter 7, *Accessing Kafka using Skupper*

Access a Kafka cluster.

Protocols

Chapter 5, *Accessing an FTP server using Skupper*

Access an FTP server.

Chapter 6, *iPerf*

Perform real-time network throughput measurements using iPerf3.

Other

Chapter 4, *Skupper Camel Integration Example*

Accessing private on-prem data from Camel

CHAPTER 2. SKUPPER HELLO WORLD

A minimal HTTP application deployed across Kubernetes clusters using Skupper

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example is a very simple multi-service HTTP application deployed across Kubernetes clusters using Skupper.

It contains two services:

- A backend service that exposes an **/api/hello** endpoint. It returns greetings of the form **Hi, <your-name>. I am <my-name> (<pod-name>)**.
- A frontend service that sends greetings to the backend and fetches new greetings in response.

With Skupper, you can place the backend in one cluster and the frontend in another and maintain connectivity between the two services without exposing the backend to the public internet.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your clusters
- Deploy the frontend and backend
- Create your sites
- Link your sites
- Expose the backend
- Access the frontend

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.
2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

   See the Installation for details about installing the CLI. For configured systems, use the following command:

   ```
   sudo dnf install skupper-cli
   ```

3. Set up your clusters
   Skupper is designed for use with multiple Kubernetes clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the cluster and namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

   NOTE

   The login procedure varies by provider. See the documentation for yours:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service
   - OpenShift

   West:

   ```
   export KUBECONFIG=~/.kube/config-west
   # Enter your provider-specific login command
   kubectl create namespace west
   kubectl config set-context --current --namespace west
   ```

   East:

   ```
   export KUBECONFIG=~/.kube/config-east
   # Enter your provider-specific login command
   kubectl create namespace east
   kubectl config set-context --current --namespace east
   ```

4. Deploy the frontend and backend
   This example runs the frontend and the backend in separate Kubernetes namespaces, on different clusters.

   Use **kubectl create deployment** to deploy the frontend in West and the backend in East.

   West:

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

East:

```
kubectl create deployment backend --image quay.io/skupper/hello-world-backend --replicas 3
```

5. Create your sites
   A Skupper site is a location where components of your application are running. Sites are linked together to form a network for your application. In Kubernetes, a site is associated with a namespace.

   For each namespace, use **skupper init** to create a site. This deploys the Skupper router and controller. Then use **skupper status** to see the outcome.

   West:

   ```
   skupper init
   skupper status
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'west'.  Use 'skupper status' to get more information.

   $ skupper status
   Skupper is enabled for namespace "west". It is not connected to any other sites. It has no exposed services.
   ```

   East:

   ```
   skupper init
   skupper status
   ```

   Sample output:

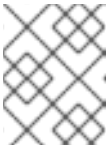
   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'east'.  Use 'skupper status' to get more information.

   $ skupper status
   Skupper is enabled for namespace "east". It is not connected to any other sites. It has no exposed services.
   ```

   As you move through the steps below, you can use **skupper status** at any time to check your progress.

6. Link your sites
   A Skupper link is a channel for communication between two sites. Links serve as a transport for application connections and requests.

Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.



NOTE

The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

First, use **skupper token create** in West to generate the token. Then, use **skupper link create** in East to link the sites.

West:

```
skupper token create ~/secret.token
```

Sample output:

```
$ skupper token create ~/secret.token
Token written to ~/secret.token
```

East:

```
skupper link create ~/secret.token
```

Sample output:

```
$ skupper link create ~/secret.token
Site configured to link to https://10.105.193.154:8081/ed9c37f6-d78a-11ec-a8c7-04421a4c5042 (name=link1)
Check the status of the link using 'skupper link status'.
```

If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

7. Expose the backend
We now have our sites linked to form a Skupper network, but no services are exposed on it. Skupper uses the **skupper expose** command to select a service from one site for exposure in all the linked sites.

Use **skupper expose** to expose the backend service in East to the frontend in West.

East:

```
skupper expose deployment/backend --port 8080
```

Sample output:

```
$ skupper expose deployment/backend --port 8080
deployment backend exposed as backend
```

8. Access the frontend
   In order to use and test the application, we need external access to the frontend.

   Use **kubectl port-forward** to make the frontend available at **localhost:8080**.

   West:

   ```
   kubectl port-forward deployment/frontend 8080:8080
   ```

   You can now access the web interface by navigating to http://localhost:8080 in your browser.

CHAPTER 3. ACCESSING ACTIVEMQ USING SKUPPER

Use public cloud resources to process data from a private message broker

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example is a simple messaging application that shows how you can use Skupper to access an ActiveMQ broker at a remote site without exposing it to the public internet.

It contains two services:

- An ActiveMQ broker running in a private data center. The broker has a queue named "notifications".
- An AMQP client running in the public cloud. It sends 10 messages to "notifications" and then receives them back.

For the broker, this example uses the Apache ActiveMQ Artemis image from ArtemisCloud.io. The client is a simple Quarkus application.

The example uses two Kubernetes namespaces, "private" and "public", to represent the private data center and public cloud.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your namespaces
- Deploy the message broker
- Create your sites
- Link your sites
- Expose the message broker
- Run the client

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.
2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

See the Installation for details about installing the CLI. For configured systems, use the following command:

```
sudo dnf install skupper-cli
```

3. Set up your namespaces
   Skupper is designed for use with multiple Kubernetes namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

   NOTE

   The login procedure varies by provider. See the documentation for yours:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service
   - OpenShift

   Public:

   ```
   export KUBECONFIG=~/.kube/config-public
   # Enter your provider-specific login command
   kubectl create namespace public
   kubectl config set-context --current --namespace public
   ```

   Private:

   ```
   export KUBECONFIG=~/.kube/config-private
   # Enter your provider-specific login command
   kubectl create namespace private
   kubectl config set-context --current --namespace private
   ```

4. Deploy the message broker
   In Private, use the **kubectl apply** command to install the broker.

   Private:

   ```
   kubectl apply -f server
   ```

Sample output:

```
$ kubectl apply -f server
deployment.apps/broker created
```

5. Create your sites
   A Skupper site is a location where components of your application are running. Sites are linked together to form a network for your application. In Kubernetes, a site is associated with a namespace.

   For each namespace, use **skupper init** to create a site. This deploys the Skupper router and controller. Then use **skupper status** to see the outcome.

   Public:

   ```
   skupper init
   skupper status
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'public'.  Use 'skupper status' to get more information.

   $ skupper status
   Skupper is enabled for namespace "public". It is not connected to any other sites. It has no exposed services.
   ```

   Private:

   ```
   skupper init
   skupper status
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'private'.  Use 'skupper status' to get more information.

   $ skupper status
   Skupper is enabled for namespace "private". It is not connected to any other sites. It has no exposed services.
   ```

   As you move through the steps below, you can use **skupper status** at any time to check your progress.

6. Link your sites
   A Skupper link is a channel for communication between two sites. Links serve as a transport for application connections and requests.

Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.

NOTE

The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

First, use **skupper token create** in site Public to generate the token. Then, use **skupper link create** in site Private to link the sites.

Public:

```
skupper token create ~/secret.token
```

Sample output:

```
$ skupper token create ~/secret.token
Token written to ~/secret.token
```

Private:

```
skupper link create ~/secret.token
```

Sample output:

```
$ skupper link create ~/secret.token
Site configured to link to https://10.105.193.154:8081/ed9c37f6-d78a-11ec-a8c7-04421a4c5042 (name=link1)
Check the status of the link using 'skupper link status'.
```

If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

7. Expose the message broker
In Private, use **skupper expose** to expose the broker on the Skupper network.

Then, in Public, use **kubectl get service/broker** to check that the service appears after a moment.

Private:

```
skupper expose deployment/broker --port 5672
```

Sample output:

```
$ skupper expose deployment/broker --port 5672
deployment broker exposed as broker
```

Public:

```
kubectl get service/broker
```

Sample output:

```
$ kubectl get service/broker
NAME     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
broker   ClusterIP   10.100.58.95   <none>        5672/TCP   2s
```

8. Run the client
In Public, use **kubectl run** to run the client.

Public:

```
kubectl run client --attach --rm --restart Never --image quay.io/skupper/activemq-example-client --env SERVER=broker
```

Sample output:

```
$ kubectl run client --attach --rm --restart Never --image quay.io/skupper/activemq-example-client --env SERVER=broker
  ____  __  _____   ___  __ ____  ____
 --/ __ \/ / / _ | / _ \/ //_/ / / / /
 -/ /_/ / /_/ / __ |/ , / ,< / // /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/_/
2022-05-27 11:19:07,149 INFO  [io.sma.rea.mes.amqp] (main) SRMSG16201: AMQP broker configured to broker:5672 for channel incoming-messages
2022-05-27 11:19:07,170 INFO  [io.sma.rea.mes.amqp] (main) SRMSG16201: AMQP broker configured to broker:5672 for channel outgoing-messages
2022-05-27 11:19:07,198 INFO  [io.sma.rea.mes.amqp] (main) SRMSG16212: Establishing connection with AMQP broker
2022-05-27 11:19:07,212 INFO  [io.sma.rea.mes.amqp] (main) SRMSG16212: Establishing connection with AMQP broker
2022-05-27 11:19:07,215 INFO  [io.quarkus] (main) client 1.0.0-SNAPSHOT on JVM (powered by Quarkus 2.9.2.Final) started in 0.397s.
2022-05-27 11:19:07,215 INFO  [io.quarkus] (main) Profile prod activated.
2022-05-27 11:19:07,215 INFO  [io.quarkus] (main) Installed features: [cdi, smallrye-context-propagation, smallrye-reactive-messaging, smallrye-reactive-messaging-amqp, vertx]
Sent message 1
Sent message 2
Sent message 3
Sent message 4
Sent message 5
Sent message 6
Sent message 7
Sent message 8
Sent message 9
Sent message 10
2022-05-27 11:19:07,434 INFO  [io.sma.rea.mes.amqp] (vert.x-eventloop-thread-0) SRMSG16213: Connection with AMQP broker established
2022-05-27 11:19:07,442 INFO  [io.sma.rea.mes.amqp] (vert.x-eventloop-thread-0) SRMSG16213: Connection with AMQP broker established
```

```
2022-05-27 11:19:07,468 INFO  [io.sma.rea.mes.amqp] (vert.x-eventloop-thread-0)
SRMSG16203: AMQP Receiver listening address notifications
Received message 1
Received message 2
Received message 3
Received message 4
Received message 5
Received message 6
Received message 7
Received message 8
Received message 9
Received message 10
Result: OK
```

CHAPTER 4. SKUPPER CAMEL INTEGRATION EXAMPLE

Twitter, Telegram and PostgreSQL integration routes deployed across Kubernetes clusters using Skupper

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

In this example we can see how to integrate different Camel integration routers that can be deployed across multiple Kubernetes clusters using Skupper.

The main idea of this project is to show a Camel integration deployed in a public cluster which searches tweets that contain the word 'skupper'. Those results are sent to a private cluster that has a database deployed. A third public cluster will ping the database and send new results to a Telegram channel.

In order to run this example you will need to create a Telegram channel and a Twitter Account to use its credentials.

It contains the following components:

- A Twitter Camel integration that searches in the Twitter feed for results containing the word **skupper** (public).
- A PostgreSQL Camel sink that receives the data from the Twitter Camel router and sends it to the database (public).
- A PostgreSQL database that contains the results (private).
- A Telegram Camel integration that polls the database and sends the results to a Telegram channel (public).

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later
- The **skupper** command-line tool, the latest version
- Access to at least one Kubernetes cluster, from any provider you choose
- **Kamel** installation to deploy the Camel integrations per namespace.

```
kamel install
```

- A **Twitter Developer Account** in order to use the Twiter API (you need to add the credentials in **config.properties** file)
- Create a **Telegram** Bot and Channel to publish messages (you need to add the credentials in **config.properties** file)

Procedure

- Configure separate console sessions
- Access your clusters

- Set up your namespaces
- Install Skupper in your namespaces
- Check the status of your namespaces
- Link your namespaces
- Deploy and expose the database in the private cluster
- Create the table to store the tweets
- Deploy Twitter Camel Integration in the public cluster
- Deploy Telegram Camel integration in the public cluster
- Test the application

1. Configure separate console sessions
   Skupper is designed for use with multiple namespaces, typically on different clusters. The **skupper** command uses your kubeconfig and current context to select the namespace where it operates.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   Start a console session for each of your namespaces. Set the **KUBECONFIG** environment variable to a different path in each session.

   Console for private1:

   ```
   export KUBECONFIG=~/.kube/config-private1
   ```

   Console for public1:

   ```
   export KUBECONFIG=~/.kube/config-public1
   ```

   Console for public2:

   ```
   export KUBECONFIG=~/.kube/config-public2
   ```

2. Access your clusters
   The methods for accessing your clusters vary by Kubernetes provider. Find the instructions for your chosen providers and use them to authenticate and configure access for each console session. See the following links for more information:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service

- OpenShift
- More providers

3. Set up your namespaces
   Use **kubectl create namespace** to create the namespaces you wish to use (or use existing namespaces). Use **kubectl config set-context** to set the current namespace for each session.

   Console for private1:

   ```
   kubectl create namespace private1
   kubectl config set-context --current --namespace private1
   ```

   Console for public1:

   ```
   kubectl create namespace public1
   kubectl config set-context --current --namespace public1
   ```

   Console for public2:

   ```
   kubectl create namespace public2
   kubectl config set-context --current --namespace public2
   ```

4. Install Skupper in your namespaces
   The **skupper init** command installs the Skupper router and service controller in the current namespace. Run the **skupper init** command in each namespace.

   Console for private1:

   ```
   skupper init
   ```

   Console for public1:

   ```
   skupper init
   ```

   Console for public2:

   ```
   skupper init
   ```

5. Check the status of your namespaces
   Use **skupper status** in each console to check that Skupper is installed.

   Console for private1:

   ```
   skupper status
   ```

   Console for public1:

   ```
   skupper status
   ```

   Console for public2:

```
skupper status
```

You should see output like this for each namespace:

```
Skupper is enabled for namespace "<namespace>" in interior mode. It is not connected
to any other sites. It has no exposed services.
The site console url is: http://<address>:8080
The credentials for internal console-auth mode are held in secret: 'skupper-console-
users'
```

As you move through the steps below, you can use **skupper status** at any time to check your progress.

6. Link your namespaces
Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote namespace, The **skupper link create** command uses the token to create a link to the namespace that generated it.

NOTE

The link token is truly a secret. Anyone who has the token can link to your namespace. Make sure that only those you trust have access to it.

First, use **skupper token create** in one namespace to generate the token. Then, use **skupper link create** in the other to create a link.

Console for public1:

```
skupper token create ~/public1.token --uses 2
```

Console for public2:

```
skupper link create ~/public1.token
skupper link status --wait 30
skupper token create ~/public2.token
```

Console for private1:

```
skupper link create ~/public1.token
skupper link create ~/public2.token
skupper link status --wait 30
```

If your console sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token.

7. Deploy and expose the database in the private cluster
Use **kubectl apply** to deploy the database in **private1**. Then expose the deployment.

Console for private1:

```
kubectl create -f src/main/resources/database/postgres-svc.yaml
skupper expose deployment postgres --address postgres --port 5432 -n private1
```

8. Create the table to store the tweets
   Console for private1:

```
kubectl run pg-shell -i --tty --image quay.io/skupper/simple-pg --
env="PGUSER=postgresadmin" --env="PGPASSWORD=admin123" --
env="PGHOST=$(kubectl get service postgres -o=jsonpath='{.spec.clusterIP}')" -- bash
psql --dbname=postgresdb
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE TABLE tw_feedback (id uuid DEFAULT uuid_generatev4 (),sigthning
VARCHAR(255),created TIMESTAMP default CURRENTTIMESTAMP,PRIMARY
KEY(id));
```

9. Deploy Twitter Camel Integration in the public cluster
   First, we need to deploy the **TwitterRoute** component in Kubernetes by using kamel. This component will poll Twitter every 5000 ms for tweets that include the word **skupper**. Subsequently, it will send the results to the **postgresql-sink**, that should be installed in the same cluster as well. The kamelet sink will insert the results in the postgreSQL database.

   Console for public1:

```
src/main/resources/scripts/setUpPublic1Cluster.sh
```

10. Deploy Telegram Camel integration in the public cluster
    In this step we will install the secret in Kubernetes that contains the database credentials, in order to be used by the **TelegramRoute** component. After that we will deploy **TelegramRoute** using kamel in the Kubernetes cluster. This component will poll the database every 3 seconds and gather the results inserted during the last 3 seconds.

    Console for public2:

```
src/main/resources/scripts/setUpPublic2Cluster.sh
```

11. Test the application
    To be able to see the whole flow at work, you need to post a tweet containing the word **skupper** and after that you will see a new message in the Telegram channel with the title **New feedback about Skupper**

    Console for private1:

```
kubectl attach pg-shell -c pg-shell -i -t
psql --dbname=postgresdb
SELECT * FROM twfeedback;
```

    Sample output:

```
id                                   | sigthning       |          created
--------------------------------------+-----------------+----------------------------
 95655229-747a-4787-8133-923ef0a1b2ca | Testing skupper | 2022-03-10
19:35:08.412542
```

Console for public1:

```
kamel logs twitter-route
```

Sample output:

```
"[1] 2022-03-10 19:35:08,397 INFO  [postgresql-sink-1] (Camel (camel-1) thread #0 - twitter-search://skupper) Testing skupper"
```

CHAPTER 5. ACCESSING AN FTP SERVER USING SKUPPER

Securely connect to an FTP server on a remote Kubernetes cluster

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example shows you how you can use Skupper to connect an FTP client on one Kubernetes cluster to an FTP server on another.

It demonstrates use of Skupper with multi-port services such as FTP. It uses FTP in passive mode (which is more typical these days) and a restricted port range that simplifies Skupper configuration.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your namespaces
- Deploy the FTP server
- Create your sites
- Link your sites
- Expose the FTP server
- Run the FTP client

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.

2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

   See the Installation for details about installing the CLI. For configured systems, use the following command:

   ```
   sudo dnf install skupper-cli
   ```

3. Set up your namespaces
   Skupper is designed for use with multiple Kubernetes namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

NOTE

The login procedure varies by provider. See the documentation for yours:

- Amazon Elastic Kubernetes Service (EKS)
- Azure Kubernetes Service (AKS)
- Google Kubernetes Engine (GKE)
- IBM Kubernetes Service
- OpenShift

Public:

```
export KUBECONFIG=~/.kube/config-public
# Enter your provider-specific login command
kubectl create namespace public
kubectl config set-context --current --namespace public
```

Private:

```
export KUBECONFIG=~/.kube/config-private
# Enter your provider-specific login command
kubectl create namespace private
kubectl config set-context --current --namespace private
```

4. Deploy the FTP server
   In Private, use **kubectl apply** to deploy the FTP server.

   Private:

   ```
   kubectl apply -f server
   ```

   Sample output:

   ```
   $ kubectl apply -f server
   deployment.apps/ftp-server created
   ```

5. Create your sites
   A Skupper site is a location where components of your application are running. Sites are linked together to form a network for your application. In Kubernetes, a site is associated with a namespace.

For each namespace, use **skupper init** to create a site. This deploys the Skupper router and controller. Then use **skupper status** to see the outcome.

Public:

```
skupper init
skupper status
```

Sample output:

```
$ skupper init
Waiting for LoadBalancer IP or hostname...
Waiting for status...
Skupper is now installed in namespace 'public'.  Use 'skupper status' to get more information.

$ skupper status
Skupper is enabled for namespace "public". It is not connected to any other sites. It has no exposed services.
```

Private:

```
skupper init
skupper status
```

Sample output:

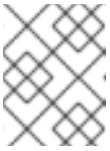
```
$ skupper init
Waiting for LoadBalancer IP or hostname...
Waiting for status...
Skupper is now installed in namespace 'private'.  Use 'skupper status' to get more information.

$ skupper status
Skupper is enabled for namespace "private". It is not connected to any other sites. It has no exposed services.
```

As you move through the steps below, you can use **skupper status** at any time to check your progress.

6. Link your sites
   A Skupper link is a channel for communication between two sites. Links serve as a transport for application connections and requests.

   Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

   The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.



NOTE

The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

First, use **skupper token create** in site Public to generate the token. Then, use **skupper link create** in site Private to link the sites.

Public:

```
skupper token create ~/secret.token
```

Sample output:

```
$ skupper token create ~/secret.token
Token written to ~/secret.token
```

Private:

```
skupper link create ~/secret.token
```

Sample output:

```
$ skupper link create ~/secret.token
Site configured to link to https://10.105.193.154:8081/ed9c37f6-d78a-11ec-a8c7-04421a4c5042 (name=link1)
Check the status of the link using 'skupper link status'.
```

If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

7. Expose the FTP server
   In Private, use **skupper expose** to expose the FTP server on all linked sites.

   Private:

   ```
   skupper expose deployment/ftp-server --port 21100 --port 21
   ```

   Sample output:

   ```
   $ skupper expose deployment/ftp-server --port 21100 --port 21
   deployment ftp-server exposed as ftp-server
   ```

8. Run the FTP client
   In Public, use **kubectl run** and the **curl** image to perform FTP put and get operations.

   Public:

   ```
   echo "Hello!" | kubectl run ftp-client --stdin --rm --image=docker.io/curlimages/curl --restart=Never -- -s -T - ftp://example:example@ftp-server/greeting
   kubectl run ftp-client --attach --rm --image=docker.io/curlimages/curl --restart=Never -- -s ftp://example:example@ftp-server/greeting
   ```

Sample output:

```
$ echo "Hello!" | kubectl run ftp-client --stdin --rm --image=docker.io/curlimages/curl --
restart=Never -- -s -T - ftp://example:example@ftp-server/greeting
pod "ftp-client" deleted

$ kubectl run ftp-client --attach --rm --image=docker.io/curlimages/curl --restart=Never --
-s ftp://example:example@ftp-server/greeting
Hello!
pod "ftp-client" deleted
```

CHAPTER 6. IPERF

Perform real-time network throughput measurements while using iPerf3

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This tutorial demonstrates how to perform real-time network throughput measurements across Kubernetes using the iperf3 tool. In this tutorial you:

- deploy iperf3 in three separate clusters
- run iperf3 client test instances

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later
- Access to three clusters to observe performance. As an example, the three clusters might consist of:
- A private cloud cluster running on your local machine (**private1**)
- Two public cloud clusters running in public cloud providers (**public1** and **public2**)

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Configure separate console sessions
- Access your clusters
- Set up your namespaces
- Install Skupper in your namespaces
- Check the status of your namespaces
- Link your namespaces
- Deploy the iperf3 servers
- Expose iperf3 from each namespace
- Run benchmark tests across the clusters

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.

2. Install the Skupper command-line tool
   The **skupper** command-line tool is the entrypoint for installing and configuring Skupper. You need to install the **skupper** command only once for each development environment.

See the Installation for details about installing the CLI. For configured systems, use the following command:

```
sudo dnf install skupper-cli
```

For Windows and other installation options, see Installing Skupper.

3. Configure separate console sessions
   Skupper is designed for use with multiple namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   Start a console session for each of your namespaces. Set the **KUBECONFIG** environment variable to a different path in each session.

   Console for public1:

   ```
   export KUBECONFIG=~/.kube/config-public1
   ```

   Console for public2:

   ```
   export KUBECONFIG=~/.kube/config-public2
   ```

   Console for private1:

   ```
   export KUBECONFIG=~/.kube/config-private1
   ```

4. Access your clusters
   The procedure for accessing a Kubernetes cluster varies by provider. Find the instructions for your chosen provider and use them to authenticate and configure access for each console session.

5. Set up your namespaces
   Use **kubectl create namespace** to create the namespaces you wish to use (or use existing namespaces). Use **kubectl config set-context** to set the current namespace for each session.

   Console for public1:

   ```
   kubectl create namespace public1
   kubectl config set-context --current --namespace public1
   ```

   Console for public2:

   ```
   kubectl create namespace public2
   kubectl config set-context --current --namespace public2
   ```

   Console for private1:

```
kubectl create namespace private1
kubectl config set-context --current --namespace private1
```

6. Install Skupper in your namespaces
   The **skupper init** command installs the Skupper router and controller in the current namespace. Run the **skupper init** command in each namespace.

   Console for public1:

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

   Console for public2:

   ```
   skupper init
   ```

   Console for private1:

   ```
   skupper init
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace '<namespace>'.  Use 'skupper status' to get more information.
   ```

7. Check the status of your namespaces
   Use **skupper status** in each console to check that Skupper is installed.

   Console for public1:

   ```
   skupper status
   ```

   Console for public2:

   ```
   skupper status
   ```

   Console for private1:

   ```
   skupper status
   ```

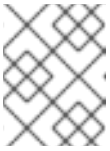
   Sample output:

   ```
   Skupper is enabled for namespace "<namespace>" in interior mode. It is connected to 1 other site. It has 1 exposed service.
   The site console url is: <console-url>
   The credentials for internal console-auth mode are held in secret: 'skupper-console-users'
   ```

   As you move through the steps below, you can use **skupper status** at any time to check your progress.

8. Link your namespaces
   Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

   The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote namespace, The **skupper link create** command uses the token to create a link to the namespace that generated it.

   NOTE

   The link token is truly a secret. Anyone who has the token can link to your namespace. Make sure that only those you trust have access to it.

   First, use **skupper token create** in one namespace to generate the token. Then, use **skupper link create** in the other to create a link.

   Console for public1:

   ```
   skupper token create ~/private1-to-public1-token.yaml
   skupper token create ~/public2-to-public1-token.yaml
   ```

   Console for public2:

   ```
   skupper token create ~/private1-to-public2-token.yaml
   skupper link create ~/public2-to-public1-token.yaml
   skupper link status --wait 60
   ```

   Console for private1:

   ```
   skupper link create ~/private1-to-public1-token.yaml
   skupper link create ~/private1-to-public2-token.yaml
   skupper link status --wait 60
   ```

   If your console sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

9. Deploy the iperf3 servers
   After creating the application router network, deploy **iperf3** in each namespace.

   Console for private1:

   ```
   kubectl apply -f deployment-iperf3-a.yaml
   ```

   Console for public1:

   ```
   kubectl apply -f deployment-iperf3-b.yaml
   ```

   Console for public2:

   ```
   kubectl apply -f deployment-iperf3-c.yaml
   ```

10. Expose iperf3 from each namespace
    We have established connectivity between the namespaces and deployed **iperf3**. Before we can test performance, we need access to the **iperf3** from each namespace.

    Console for private1:

    ```
    skupper expose deployment/iperf3-server-a --port 5201
    ```

    Console for public1:

    ```
    skupper expose deployment/iperf3-server-b --port 5201
    ```

    Console for public2:

    ```
    skupper expose deployment/iperf3-server-c --port 5201
    ```

11. Run benchmark tests across the clusters
    After deploying the iperf3 servers into the private and public cloud clusters, the virtual application network enables communications even though they are running in separate clusters.

    Console for private1:

    ```
    kubectl exec $(kubectl get pod -l application=iperf3-server-a -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-a
    kubectl exec $(kubectl get pod -l application=iperf3-server-a -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-b
    kubectl exec $(kubectl get pod -l application=iperf3-server-a -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-c
    ```

    Console for public1:

    ```
    kubectl exec $(kubectl get pod -l application=iperf3-server-b -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-a
    kubectl exec $(kubectl get pod -l application=iperf3-server-b -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-b
    kubectl exec $(kubectl get pod -l application=iperf3-server-b -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-c
    ```

    Console for public2:

    ```
    kubectl exec $(kubectl get pod -l application=iperf3-server-c -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-a
    kubectl exec $(kubectl get pod -l application=iperf3-server-c -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-b
    kubectl exec $(kubectl get pod -l application=iperf3-server-c -
    o=jsonpath='{.items[0].metadata.name}') -- iperf3 -c iperf3-server-c
    ```

CHAPTER 7. ACCESSING KAFKA USING SKUPPER

Use public cloud resources to process data from a private Kafka cluster

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example is a simple Kafka application that shows how you can use Skupper to access a Kafka cluster at a remote site without exposing it to the public internet.

It contains two services:

- A Kafka cluster named "cluster1" running in a private data center. The cluster has a topic named "topic1".
- A Kafka client running in the public cloud. It sends 10 messages to "topic1" and then receives them back.

To set up the Kafka cluster, this example uses the Kubernetes operator from the Strimzi project. The Kafka client is a Java application built using Quarkus.

The example uses two Kubernetes namespaces, "private" and "public", to represent the private data center and public cloud.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your namespaces
- Deploy the Kafka cluster
- Create your sites
- Link your sites
- Expose the Kafka cluster
- Run the client

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.
2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

See the Installation for details about installing the CLI. For configured systems, use the following command:

```
sudo dnf install skupper-cli
```

3. Set up your namespaces
   Skupper is designed for use with multiple Kubernetes namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

   NOTE

   The login procedure varies by provider. See the documentation for yours:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service
   - OpenShift

   Public:

   ```
   export KUBECONFIG=~/.kube/config-public
   # Enter your provider-specific login command
   kubectl create namespace public
   kubectl config set-context --current --namespace public
   ```

   Private:

   ```
   export KUBECONFIG=~/.kube/config-private
   # Enter your provider-specific login command
   kubectl create namespace private
   kubectl config set-context --current --namespace private
   ```

4. Deploy the Kafka cluster
   In Private, use the **kubectl create** and **kubectl apply** commands with the listed YAML files to install the operator and deploy the cluster and topic.

   Private:

```
kubectl create -f server/strimzi.yaml
kubectl apply -f server/cluster1.yaml
kubectl wait --for condition=ready --timeout 900s kafka/cluster1
```

Sample output:

```
$ kubectl create -f server/strimzi.yaml
customresourcedefinition.apiextensions.k8s.io/kafkas.kafka.strimzi.io created
rolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator-entity-operator-delegation
created
clusterrolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator created
rolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator-topic-operator-delegation
created
customresourcedefinition.apiextensions.k8s.io/kafkausers.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkarebalances.kafka.strimzi.io created
deployment.apps/strimzi-cluster-operator created
customresourcedefinition.apiextensions.k8s.io/kafkamirrormaker2s.kafka.strimzi.io
created
clusterrole.rbac.authorization.k8s.io/strimzi-entity-operator created
clusterrole.rbac.authorization.k8s.io/strimzi-cluster-operator-global created
clusterrolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator-kafka-broker-
delegation created
rolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator created
clusterrole.rbac.authorization.k8s.io/strimzi-cluster-operator-namespaced created
clusterrole.rbac.authorization.k8s.io/strimzi-topic-operator created
clusterrolebinding.rbac.authorization.k8s.io/strimzi-cluster-operator-kafka-client-
delegation created
clusterrole.rbac.authorization.k8s.io/strimzi-kafka-client created
serviceaccount/strimzi-cluster-operator created
clusterrole.rbac.authorization.k8s.io/strimzi-kafka-broker created
customresourcedefinition.apiextensions.k8s.io/kafkatopics.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkabridges.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkaconnectors.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkaconnects2is.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkaconnects.kafka.strimzi.io created
customresourcedefinition.apiextensions.k8s.io/kafkamirrormakers.kafka.strimzi.io
created
configmap/strimzi-cluster-operator created

$ kubectl apply -f server/cluster1.yaml
kafka.kafka.strimzi.io/cluster1 created
kafkatopic.kafka.strimzi.io/topic1 created

$ kubectl wait --for condition=ready --timeout 900s kafka/cluster1
kafka.kafka.strimzi.io/cluster1 condition met
```

NOTE:

By default, the Kafka bootstrap server returns broker addresses that include the Kubernetes namespace in their domain name. When, as in this example, the Kafka client is running in a namespace with a different name from that of the Kafka cluster, this prevents the client from resolving the Kafka brokers.

To make the Kafka brokers reachable, set the **advertisedHost** property of each broker to a domain name that the Kafka client can resolve at the remote site. In this example, this is achieved with the following listener configuration:

```
spec:
  kafka:
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
        configuration:
          brokers:
            - broker: 0
              advertisedHost: cluster1-kafka-0.cluster1-kafka-brokers
```

See Advertised addresses for brokers for more information.

5. Create your sites
   A Skupper site is a location where components of your application are running. Sites are linked together to form a network for your application. In Kubernetes, a site is associated with a namespace.

   For each namespace, use **skupper init** to create a site. This deploys the Skupper router and controller. Then use **skupper status** to see the outcome.

   Public:

   ```
   skupper init
   skupper status
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'public'.  Use 'skupper status' to get more information.

   $ skupper status
   Skupper is enabled for namespace "public". It is not connected to any other sites. It has no exposed services.
   ```

   Private:

   ```
   skupper init
   skupper status
   ```

   Sample output:

   ```
   $ skupper init
   Waiting for LoadBalancer IP or hostname...
   Waiting for status...
   Skupper is now installed in namespace 'private'.  Use 'skupper status' to get more information.
   ```

```
$ skupper status
Skupper is enabled for namespace "private". It is not connected to any other sites. It has
no exposed services.
```

As you move through the steps below, you can use **skupper status** at any time to check your progress.

6. Link your sites
   A Skupper link is a channel for communication between two sites. Links serve as a transport for application connections and requests.

   Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

   The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.

   NOTE

   The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

   First, use **skupper token create** in site Public to generate the token. Then, use **skupper link create** in site Private to link the sites.

   Public:

   ```
   skupper token create ~/secret.token
   ```

   Sample output:

   ```
   $ skupper token create ~/secret.token
   Token written to ~/secret.token
   ```

   Private:

   ```
   skupper link create ~/secret.token
   ```

   Sample output:

   ```
   $ skupper link create ~/secret.token
   Site configured to link to https://10.105.193.154:8081/ed9c37f6-d78a-11ec-a8c7-
   04421a4c5042 (name=link1)
   Check the status of the link using 'skupper link status'.
   ```

   If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

7. Expose the Kafka cluster
   In Private, use **skupper expose** with the **--headless** option to expose the Kafka cluster as a headless service on the Skupper network.

Then, in Public, use the **kubectl get service** command to check that the **cluster1-kafka-brokers** service appears after a moment.

Private:

```
skupper expose statefulset/cluster1-kafka --headless --port 9092
```

Sample output:

```
$ skupper expose statefulset/cluster1-kafka --headless --port 9092
statefulset cluster1-kafka exposed as cluster1-kafka-brokers
```

Public:

```
kubectl get service/cluster1-kafka-brokers
```

Sample output:

```
$ kubectl get service/cluster1-kafka-brokers
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
cluster1-kafka-brokers   ClusterIP   None         <none>        9092/TCP   2s
```

8. Run the client
   Use the **kubectl run** command to execute the client program in Public.

   Public:

   ```
   kubectl run client --attach --rm --restart Never --image quay.io/skupper/kafka-example-client --env BOOTSTRAPSERVERS=cluster1-kafka-brokers:9092
   ```

   Sample output:

   ```
   $ kubectl run client --attach --rm --restart Never --image quay.io/skupper/kafka-example-client --env BOOTSTRAPSERVERS=cluster1-kafka-brokers:9092
   [...]
   Received message 1
   Received message 2
   Received message 3
   Received message 4
   Received message 5
   Received message 6
   Received message 7
   Received message 8
   Received message 9
   Received message 10
   Result: OK
   [...]
   ```

   To see the client code, look in the client directory of this project.

CHAPTER 8. PATIENT PORTAL

A simple database-backed web application that runs in the public cloud but keeps its data in a private database

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example is a simple database-backed web application that shows how you can use Skupper to access a database at a remote site without exposing it to the public internet.

It contains three services:

- A PostgreSQL database running on a bare-metal or virtual machine in a private data center.
- A payment-processing service running on Kubernetes in a private data center.
- A web frontend service running on Kubernetes in the public cloud. It uses the PostgreSQL database and the payment-processing service.

The example uses two Kubernetes namespaces, **private** and **public**, to represent the Kubernetes cluster in the private data center and the cluster in the public cloud. It uses Podman to run the database.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your Kubernetes namespaces
- Set up your Podman network
- Deploy the application
- Create your sites
- Link your sites
- Expose application services
- Access the frontend

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.

2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

See the Installation for details about installing the CLI. For configured systems, use the following command:

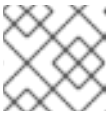
```
sudo dnf install skupper-cli
```

3. Set up your Kubernetes namespaces
Skupper is designed for use with multiple Kubernetes namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

   NOTE

   The login procedure varies by provider. See the documentation for yours:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service
   - OpenShift

   Public:

   ```
   export KUBECONFIG=~/.kube/config-public
   # Enter your provider-specific login command
   kubectl create namespace public
   kubectl config set-context --current --namespace public
   ```

   Private:

   ```
   export KUBECONFIG=~/.kube/config-private
   # Enter your provider-specific login command
   kubectl create namespace private
   kubectl config set-context --current --namespace private
   ```

4. Set up your Podman network
Open a new terminal window and set the **SKUPPERPLATFORM** environment variable to **podman**. This sets the Skupper platform to Podman for this terminal session.

   Use **podman network create** to create the Podman network that Skupper will use.

Use **systemctl** to enable the Podman API service.

Podman:

```
export SKUPPERPLATFORM=podman
podman network create skupper
systemctl --user enable --now podman.socket
```

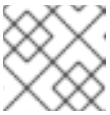
If the **systemctl** command doesn't work, you can try the **podman system service** command instead:

```
podman system service --time=0 unix://$XDGRUNTIMEDIR/podman/podman.sock &
```

5. Deploy the application
Use **kubectl apply** to deploy the frontend and payment processor on Kubernetes. Use **podman run** to start the database on your local machine.

NOTE

It is important to name your running container using **--name** to avoid a collision with the container that Skupper creates for accessing the service.



NOTE

You must use **--network skupper** with the **podman run** command.

Public:

```
kubectl apply -f frontend/kubernetes.yaml
```

Private:

```
kubectl apply -f payment-processor/kubernetes.yaml
```

Podman:

```
podman run --name database-target --network skupper --detach --rm -p 5432:5432
quay.io/skupper/patient-portal-database
```

6. Create your sites
Public:

```
skupper init
```

Private:

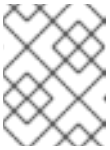
```
skupper init --ingress none
```

Podman:

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

7. Link your sites
   Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

   The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.

   NOTE

   The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

   First, use **skupper token create** in site Public to generate the token. Then, use **skupper link create** in site Private to link the sites.

   Public:

   ```
   skupper token create --uses 2 ~/secret.token
   ```

   Private:

   ```
   skupper link create ~/secret.token
   ```

   Podman:

   ```
   skupper link create ~/secret.token
   ```

   If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

8. Expose application services
   In Private, use **skupper expose** to expose the payment processor service.

   In Podman, use **skupper service create** and **skupper service bind** to expose the database on the Skupper network.

   Then, in Public, use **skupper service create** to make it available.

   NOTE

   Podman sites do not automatically replicate services to remote sites. You need to use **skupper service create** on each site where you wish to make a service available.

   Private:

   ```
   skupper expose deployment/payment-processor --port 8080
   ```

   Podman:

```
skupper service create database 5432
skupper service bind database host database-target --target-port 5432
```

Public:

```
skupper service create database 5432
```

9. Access the frontend
In order to use and test the application, we need external access to the frontend.

Use **kubectl expose** with **--type LoadBalancer** to open network access to the frontend service.

Once the frontend is exposed, use **kubectl get service/frontend** to look up the external IP of the frontend service. If the external IP is **<pending>**, try again after a moment.

Once you have the external IP, use **curl** or a similar tool to request the **/api/health** endpoint at that address.

NOTE

The **<external-ip>** field in the following commands is a placeholder. The actual value is an IP address.

Public:

```
kubectl expose deployment/frontend --port 8080 --type LoadBalancer
kubectl get service/frontend
curl http://<external-ip>:8080/api/health
```

Sample output:

```
$ kubectl expose deployment/frontend --port 8080 --type LoadBalancer
service/frontend exposed

$ kubectl get service/frontend
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
frontend   LoadBalancer   10.103.232.28   <external-ip>   8080:30407/TCP   15s

$ curl http://<external-ip>:8080/api/health
OK
```

If everything is in order, you can now access the web interface by navigating to **http://<external-ip>:8080/** in your browser.

CHAPTER 9. TRADE ZOO

A simple trading application that runs in the public cloud but keeps its data in a private Kafka cluster

This example is part of a suite of examples showing the different ways you can use Skupper to connect services across cloud providers, data centers, and edge sites.

Overview

This example is a simple Kafka application that shows how you can use Skupper to access a Kafka cluster at a remote site without exposing it to the public internet.

It contains four services:

- A Kafka cluster running in a private data center. The cluster has two topics, "orders" and "updates".
- An order processor running in the public cloud. It consumes from "orders", matching buy and sell offers to make trades. It publishes new and updated orders and trades to "updates".
- A market data service running in the public cloud. It looks at the completed trades and computes the latest and average prices, which it then publishes to "updates".
- A web frontend service running in the public cloud. It submits buy and sell orders to "orders" and consumes from "updates" in order to show what's happening.

To set up the Kafka cluster, this example uses the Kubernetes operator from the Strimzi project. The other services are small Python programs.

The example uses two Kubernetes namespaces, "private" and "public", to represent the private data center and public cloud.

Prerequisites

- The **kubectl** command-line tool, version 1.15 or later (installation guide)
- Access to at least one Kubernetes cluster, from any provider you choose

Procedure

- Clone the repo for this example.
- Install the Skupper command-line tool
- Set up your namespaces
- Deploy the Kafka cluster
- Deploy the application services
- Create your sites
- Link your sites
- Expose the Kafka cluster
- Access the frontend

1. Clone the repo for this example. Navigate to the appropriate GitHub repository from https://skupper.io/examples/index.html and clone the repository.

2. Install the Skupper command-line tool
   This example uses the Skupper command-line tool to deploy Skupper. You need to install the **skupper** command only once for each development environment.

   See the Installation for details about installing the CLI. For configured systems, use the following command:

   ```
   sudo dnf install skupper-cli
   ```

3. Set up your namespaces
   Skupper is designed for use with multiple Kubernetes namespaces, usually on different clusters. The **skupper** and **kubectl** commands use your kubeconfig and current context to select the namespace where they operate.

   Your kubeconfig is stored in a file in your home directory. The **skupper** and **kubectl** commands use the **KUBECONFIG** environment variable to locate it.

   A single kubeconfig supports only one active context per user. Since you will be using multiple contexts at once in this exercise, you need to create distinct kubeconfigs.

   For each namespace, open a new terminal window. In each terminal, set the **KUBECONFIG** environment variable to a different path and log in to your cluster. Then create the namespace you wish to use and set the namespace on your current context.

   NOTE

   The login procedure varies by provider. See the documentation for yours:

   - Amazon Elastic Kubernetes Service (EKS)
   - Azure Kubernetes Service (AKS)
   - Google Kubernetes Engine (GKE)
   - IBM Kubernetes Service
   - OpenShift

   Public:

   ```
   export KUBECONFIG=~/.kube/config-public
   # Enter your provider-specific login command
   kubectl create namespace public
   kubectl config set-context --current --namespace public
   ```

   Private:

   ```
   export KUBECONFIG=~/.kube/config-private
   # Enter your provider-specific login command
   kubectl create namespace private
   kubectl config set-context --current --namespace private
   ```

4. Deploy the Kafka cluster
   In Private, use the **kubectl create** and **kubectl apply** commands with the listed YAML files to install the operator and deploy the cluster and topic.

   Private:

   ```
   kubectl create -f kafka-cluster/strimzi.yaml
   kubectl apply -f kafka-cluster/cluster1.yaml
   kubectl wait --for condition=ready --timeout 900s kafka/cluster1
   ```

   NOTE:

   By default, the Kafka bootstrap server returns broker addresses that include the Kubernetes namespace in their domain name. When, as in this example, the Kafka client is running in a namespace with a different name from that of the Kafka cluster, this prevents the client from resolving the Kafka brokers.

   To make the Kafka brokers reachable, set the **advertisedHost** property of each broker to a domain name that the Kafka client can resolve at the remote site. In this example, this is achieved with the following listener configuration:

   ```
   spec:
     kafka:
       listeners:
         - name: plain
           port: 9092
           type: internal
           tls: false
           configuration:
             brokers:
               - broker: 0
                 advertisedHost: cluster1-kafka-0.cluster1-kafka-brokers
   ```

   See Advertised addresses for brokers for more information.

5. Deploy the application services
   In Public, use the **kubectl apply** command with the listed YAML files to install the application services.

   Public:

   ```
   kubectl apply -f order-processor/kubernetes.yaml
   kubectl apply -f market-data/kubernetes.yaml
   kubectl apply -f frontend/kubernetes.yaml
   ```

6. Create your sites
   A Skupper site is a location where components of your application are running. Sites are linked together to form a network for your application. In Kubernetes, a site is associated with a namespace.

   For each namespace, use **skupper init** to create a site. This deploys the Skupper router and controller. Then use **skupper status** to see the outcome.

   Public:

```
skupper init
skupper status
```

Sample output:

```
$ skupper init
Waiting for LoadBalancer IP or hostname...
Waiting for status...
Skupper is now installed in namespace 'public'.  Use 'skupper status' to get more information.

$ skupper status
Skupper is enabled for namespace "public". It is not connected to any other sites. It has no exposed services.
```

Private:

```
skupper init
skupper status
```

Sample output:

```
$ skupper init
Waiting for LoadBalancer IP or hostname...
Waiting for status...
Skupper is now installed in namespace 'private'.  Use 'skupper status' to get more information.

$ skupper status
Skupper is enabled for namespace "private". It is not connected to any other sites. It has no exposed services.
```

As you move through the steps below, you can use **skupper status** at any time to check your progress.

7. Link your sites
   A Skupper link is a channel for communication between two sites. Links serve as a transport for application connections and requests.

   Creating a link requires use of two **skupper** commands in conjunction, **skupper token create** and **skupper link create**.

   The **skupper token create** command generates a secret token that signifies permission to create a link. The token also carries the link details. Then, in a remote site, The **skupper link create** command uses the token to create a link to the site that generated it.

   NOTE

   The link token is truly a secret. Anyone who has the token can link to your site. Make sure that only those you trust have access to it.

   First, use **skupper token create** in site Public to generate the token. Then, use **skupper link create** in site Private to link the sites.

Public:

```
skupper token create ~/secret.token
```

Sample output:

```
$ skupper token create ~/secret.token
Token written to ~/secret.token
```

Private:

```
skupper link create ~/secret.token
```

Sample output:

```
$ skupper link create ~/secret.token
Site configured to link to https://10.105.193.154:8081/ed9c37f6-d78a-11ec-a8c7-
04421a4c5042 (name=link1)
Check the status of the link using 'skupper link status'.
```

If your terminal sessions are on different machines, you may need to use **scp** or a similar tool to transfer the token securely. By default, tokens expire after a single use or 15 minutes after creation.

8. Expose the Kafka cluster
   In Private, use **skupper expose** with the **--headless** option to expose the Kafka cluster as a headless service on the Skupper network.

   Then, in Public, use **kubectl get service** to check that the **cluster1-kafka-brokers** service appears after a moment.

   Private:

   ```
   skupper expose statefulset/cluster1-kafka --headless --port 9092
   ```

   Public:

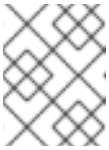
   ```
   kubectl get service/cluster1-kafka-brokers
   ```

9. Access the frontend
   In order to use and test the application, we need external access to the frontend.

   Use **kubectl expose** with **--type LoadBalancer** to open network access to the frontend service.

   Once the frontend is exposed, use **kubectl get service/frontend** to look up the external IP of the frontend service. If the external IP is **<pending>**, try again after a moment.

   Once you have the external IP, use **curl** or a similar tool to request the **/api/health** endpoint at that address.



NOTE

The **<external-ip>** field in the following commands is a placeholder. The actual value is an IP address.

Public:

```
kubectl expose deployment/frontend --port 8080 --type LoadBalancer
kubectl get service/frontend
curl http://<external-ip>:8080/api/health
```

Sample output:

```
$ kubectl expose deployment/frontend --port 8080 --type LoadBalancer
service/frontend exposed

$ kubectl get service/frontend
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
frontend   LoadBalancer   10.103.232.28   <external-ip>   8080:30407/TCP   15s

$ curl http://<external-ip>:8080/api/health
OK
```

If everything is in order, you can now access the web interface by navigating to **http://<external-ip>:8080/** in your browser.