



# Red Hat Service Interconnect 1.8

例

CLI および YAML を使用したサービスネットワークチュートリアル



## Red Hat Service Interconnect 1.8 例

---

CLI および YAML を使用したサービスネットワークチュートリアル

## 法律上の通知

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

Red Hat Service Interconnect は、オープンソースの Skupper プロジェクトの Red Hat ビルドです。この Skupper ドキュメントは、参考用として複製されたものです。

---

## 目次

第1章 サンプルの概要 .....	3
第2章 SKUPPER HELLO WORLD .....	4
第3章 SKUPPER を使用した ACTIVEMQ へのアクセス .....	9
第4章 SKUPPER CAMEL インテグレーションの例 .....	15
第5章 SKUPPER を使用した FTP サーバーへのアクセス .....	21
第6章 IPERF .....	26
第7章 SKUPPER を使用した KAFKA へのアクセス .....	32
第8章 患者ポータル .....	39
第9章 TRADE ZOO .....	45



# 第1章 サンプルの概要

## 重要な機能

### 2章 *Skupper Hello World*

サイト全体にデプロイされる最小限のマルチサービス HTTP アプリケーション。

### 8章 *患者ポータル*

サイト全体にデプロイされるデータベースベースの Web アプリケーション。

### 9章 *Trade Zoo*

サイト全体にデプロイされた Kafka ベースの取引アプリケーション。

## メッセージング

### 3章 *Skupper を使用した ActiveMQ へのアクセス*

ActiveMQ メッセージブローカーにアクセスする。

### 7章 *Skupper を使用した Kafka へのアクセス*

Kafka クラスターにアクセスする。

## プロトコル

### 5章 *Skupper を使用した FTP サーバーへのアクセス*

FTP サーバーにアクセスする。

### 6章 *iPerf*

iPerf3 を使用して、リアルタイムのネットワークスループット測定を実行する。

## その他

### 4章 *Skupper Camel インテグレーションの例*

Camel からのプライベートオンプレミスデータにアクセスする。

## 第2章 SKUPPER HELLO WORLD

Skupper を使用して Kubernetes クラスター全体にデプロイされた最小限の HTTP アプリケーション

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例は、Skupper を使用して Kubernetes クラスター全体にデプロイされた非常にシンプルなマルチサービス HTTP アプリケーションです。

これには 2 つのサービスが含まれます。

- `/api/hello` エンドポイントを公開するバックエンドサービス。これは、**Hi, <your-name>. I am <my-name> (<pod-name>)** という形式のあいさつを返します。
- フロントエンドサービス。バックエンドにあいさつを送信し、応答として新しい挨拶を取得します。

Skupper を使用すると、バックエンドを 1 つのクラスターに配置し、フロントエンドを別のクラスターに配置して、バックエンドをパブリックインターネットに公開することなく、2 つのサービス間の接続を維持できます。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも 1 つの Kubernetes クラスターにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [クラスターを設定する](#)
- [フロントエンドとバックエンドをデプロイする](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [バックエンドを公開する](#)
- [フロントエンドにアクセスする](#)
  1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
  2. Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを 1 回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

### 3. クラスターを設定します。

Skupper は複数の Kubernetes クラスターで使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、**kubeconfig** と現在のコンテキストを使用して、動作するクラスターと名前空間を選択します。

**kubeconfig** はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの **kubeconfig** は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の **kubeconfig** を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



#### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [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. フロントエンドとバックエンドをデプロイします。

この例では、フロントエンドとバックエンドを異なるクラスター上の別々の Kubernetes 名前空間で実行します。

**kubectl create deployment** を使用して、フロントエンドを West に、バックエンドを East にデプロイします。

**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. サイトを作成します。

Skupper サイトは、アプリケーションのコンポーネントが実行される場所です。サイトは相互にリンクされ、アプリケーションのネットワークを形成します。Kubernetes では、サイトは名前空間に関連付けられます。

名前空間ごとに、**skupper init** を使用してサイトを作成します。これにより、Skupper ルーターとコントローラーがデプロイされます。次に、**skupper status** を使用して結果を確認します。

**West:**

```
skupper init  
skupper status
```

## 出力サンプル

```
$ 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
```

## 出力サンプル

```
$ 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.
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

#### 6. サイトをリンクします。

Skupper リンクは、2つのサイト間の通信チャネルです。リンクは、アプリケーションの接続と要求のトランスポートとして機能します。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリンクを作成します。



#### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、West で **skupper token create** を使用してトークンを生成します。次に、East で **skupper link create** を使用して、サイトをリンクします。

#### West:

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

#### 出力サンプル

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

#### East:

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

#### 出力サンプル

```
$ 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'.
```

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用後、または作成後 15 分後に期限切れになります。

#### 7. バックエンドを公開します。

現在、サイトはリンクされて Skupper ネットワークを形成していますが、そのネットワーク上でサービスは公開されていません。Skupper は、**skupper expose** コマンドを使用して、リンクされたすべてのサイトで公開するサービスを1つのサイトから選択します。

**skupper expose** を使用して、East のバックエンドサービスを West のフロントエンドに公開します。

East:

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

出力サンプル

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

8. フロントエンドにします。  
アプリケーションを使用およびテストするには、フロントエンドへの外部アクセスが必要です。

**kubectl port-forward** を使用して、フロントエンドを **localhost:8080** で利用できるようにします。

West:

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

ブラウザで <http://localhost:8080> に移動すると、Web インターフェイスにアクセスできるようになります。

## 第3章 SKUPPER を使用した ACTIVEMQ へのアクセス

パブリッククラウドリソースを使用してプライベートメッセージブローカーからのデータを処理します。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例は、Skupper を使用して、パブリックインターネットに公開せずにリモートサイトの ActiveMQ ブローカーにアクセスする方法を示したシンプルなメッセージングアプリケーションです。

これには2つのサービスが含まれます。

- プライベートデータセンターで実行されている ActiveMQ ブローカー。ブローカーには、"notifications" という名前のキューがあります。
- パブリッククラウドで実行している AMQP クライアント。"notifications" に10個のメッセージを送信し、再度受信します。

この例では、ブローカーとして [ArtemisCloud.io](#) の [Apache ActiveMQ Artemis](#) イメージを使用します。クライアントはシンプルな [Quarkus](#) アプリケーションです。

この例では、プライベートデータセンターとパブリッククラウドを表すために、"private" と "public" という2つの Kubernetes 名前空間を使用します。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも1つの Kubernetes クラスタにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [名前空間を設定する](#)
- [メッセージブローカーをデプロイする](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [メッセージブローカーを公開する](#)
- [クライアントを実行する](#)
  1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
  2. Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを1回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

### 3. 名前空間を設定します。

Skupper は、通常は異なるクラスター上の複数の Kubernetes 名前空間で使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、[kubeconfig](#) と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



#### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)
- [IBM Kubernetes Service](#)
- [OpenShift](#)

#### パブリック:

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

#### プライベート:

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

### 4. メッセージブローカーをデプロイします。

プライベートでは、**kubectl apply** コマンドを使用してブローカーをインストールします。

#### プライベート:

```
kubectl apply -f server
```

出力サンプル

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

##### 5. サイトを作成します。

Skupper サイトは、アプリケーションのコンポーネントが実行される場所です。サイトは相互にリンクされ、アプリケーションのネットワークを形成します。Kubernetes では、サイトは名前空間に関連付けられます。

名前空間ごとに、**skupper init** を使用してサイトを作成します。これにより、Skupper ルーターとコントローラーがデプロイされます。次に、**skupper status** を使用して結果を確認します。

**パブリック:**

```
skupper init
skupper status
```

出力サンプル

```
$ 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.
```

**プライベート:**

```
skupper init
skupper status
```

出力サンプル

```
$ 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.
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

##### 6. サイトをリンクします。

Skupper リンクは、2つのサイト間の通信チャンネルです。リンクは、アプリケーションの接続と要求のトランスポートとして機能します。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリンクを作成します。



### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、パブリックサイトの **skupper token create** を使用してトークンを生成します。次に、プライベートサイトで **skupper link create** を使用して、サイトをリンクします。

#### パブリック:

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

#### 出力サンプル

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

#### プライベート:

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

#### 出力サンプル

```
$ 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'.
```

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用後、または作成後15分後に期限切れになります。

#### 7. メッセージブローカーを公開します。

プライベートでは、**skupper expose** を使用して、ブローカーを Skupper ネットワーク上に公開します。

次に、Public で **kubectl get service/broker** を使用して、しばらくするとサービスが表示されることを確認します。

#### プライベート:

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



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

## 第4章 SKUPPER CAMEL インテグレーションの例

Skupper を使用して Kubernetes クラスター全体に Twitter、Telegram、PostgreSQL インテグレーションルートをデプロイします。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例では、Skupper を使用して複数の Kubernetes クラスターにデプロイできるさまざまな Camel インテグレーションルーターを統合する方法を説明します。

このプロジェクトは、パブリッククラスターにデプロイされ、'skupper' という単語を含むツイートを検索する Camel インテグレーションを示すことが主な目的です。これらの結果は、データベースがデプロイされているプライベートクラスターに送信されます。3 番目のパブリッククラスターはデータベースに ping を実行し、新しい結果を Telegram チャンネルに送信します。

この例を実行するには、Telegram チャンネルとその認証情報を使用する Twitter アカウントを作成する必要があります。

これには、以下のコンポーネントが含まれます。

- Twitter フィード内で **skupper** (public) という単語を含む結果を検索する Twitter Camel インテグレーション。
- Twitter Camel ルーターからデータを受信してデータベース (パブリック) に送信する PostgreSQL Camel シンク。
- 結果を格納する PostgreSQL データベース (プライベート)。
- データベースをポーリングし、その結果を Telegram チャンネル (パブリック) に送信する Telegram Camel インテグレーション。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降
- **skupper** コマンドラインツール、最新バージョン
- 任意のプロバイダーから少なくとも 1 つの Kubernetes クラスターにアクセスできる
- 名前空間ごとに Camel インテグレーションをデプロイするための **Kamel** インストール

```
kamel install
```

- Twitter API を使用するための **Twitter Developer Account** (**config.properties** ファイルに認証情報を追加する必要があります)
- メッセージを公開するために **Telegram** ボットとチャンネルを作成する (**config.properties** ファイルに認証情報を追加する必要があります)

### 手順

- [個別のコンソールセッションを設定する](#)

- クラスターにアクセスする
- 名前空間を設定する
- 名前空間に Skupper をインストールする
- 名前空間のステータスを確認する
- 名前空間をリンクする
- プライベートクラスターにデータベースをデプロイして公開する
- ツイートを保存するテーブルを作成する
- パブリッククラスターに Twitter Camel インテグレーションをデプロイする
- パブリッククラスターに Telegram Camel インテグレーションをデプロイする
- アプリケーションをテストする

#### 1. 個別のコンソールセッションを設定する

Skupper は、通常は異なるクラスター上の複数の名前空間で使用するよう設計されています。**skupper** コマンドは、**kubeconfig** と現在のコンテキストを使用して、動作する名前空間を選択します。

**kubeconfig** はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの **kubeconfig** は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の **kubeconfig** を作成する必要があります。

各名前空間のコンソールセッションを開始します。各セッションで **KUBECONFIG** 環境変数を異なるパスに設定します。

private1 のコンソール:

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

public1 のコンソール:

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

public2 のコンソール:

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

#### 2. クラスターにアクセスする

クラスターにアクセスする方法は、Kubernetes プロバイダーによって異なります。選択したプロバイダーの手順を確認し、それを使用して各コンソールセッションのアクセスを認証および設定します。詳細は、次のリンクを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)

- [Google Kubernetes Engine \(GKE\)](#)
  - [IBM Kubernetes Service](#)
  - [OpenShift](#)
  - [その他のプロバイダー](#)
3. 名前空間を設定する
- kubectl create namespace** を使用して、使用する名前空間を作成します (または既存の名前空間を使用します)。 **kubectl config set-context** を使用して、各セッションの現在の名前空間を設定します。

private1 のコンソール:

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

public1 のコンソール:

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

public2 のコンソール:

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

4. 名前空間に Skupper をインストールする
- skupper init** コマンドは、現在の名前空間に Skupper ルーターとサービスコントローラーをインストールします。各 namespace で **skupper init** コマンドを実行します。

private1 のコンソール:

```
skupper init
```

public1 のコンソール:

```
skupper init
```

public2 のコンソール:

```
skupper init
```

5. 名前空間のステータスを確認する
- 各コンソールで **skupper status** を使用して、Skupper がインストールされていることを確認します。

private1 のコンソール:

```
skupper status
```

public1 のコンソール:

```
skupper status
```

public2 のコンソール:

```
skupper status
```

名前空間ごとに次のような出力が表示されます。

```
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'
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

## 6. 名前空間をリンクする

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモート名前空間で、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成した名前空間へのリンクを作成します。



### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、名前空間にリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、1つの名前空間で **skupper token create** を使用してトークンを生成します。次に、もう1つで **skupper link create** を使用してリンクを作成します。

public1 のコンソール:

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

public2 のコンソール:

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

private1 のコンソール:

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

コンソールセッションが異なるマシン上にある場合は、トークンを転送するために **scp** または同様のツールを使用する必要がある場合があります。

7. プライベートクラスターにデータベースをデプロイして公開する  
**kubectl apply** を使用して、**private1** にデータベースをデプロイします。次に、デプロイメントを公開します。

private1 のコンソール:

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

8. ツイートを保存するテーブルを作成する

private1 のコンソール:

```
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 (),sighthing
VARCHAR(255),created TIMESTAMP default CURRENTTIMESTAMP,PRIMARY
KEY(id));
```

9. パブリッククラスターに Twitter Camel インテグレーションをデプロイする  
まず、**kamel** を使用して **TwitterRoute** コンポーネントを Kubernetes にデプロイする必要があります。このコンポーネントは、5000 ミリ秒ごとに Twitter をポーリングして、**skupper** という単語を含むツイートを検索します。その後、**postgresql-sink** に結果を送信します。この **postgresql-sink** は、同じクラスター内にインストールしておく必要があります。**kamelet** シンクは、結果を **postgreSQL** データベースに挿入します。

public1 のコンソール:

```
src/main/resources/scripts/setupPublic1Cluster.sh
```

10. パブリッククラスターに Telegram Camel インテグレーションをデプロイする  
このステップでは、**TelegramRoute** コンポーネントで使用するために、データベース認証情報を含むシークレットを Kubernetes にインストールします。その後、**Kamel** を使用して **TelegramRoute** を Kubernetes クラスターにデプロイします。このコンポーネントは3秒ごとにデータベースをポーリングし、最後の3秒間に挿入された結果を収集します。

public2 のコンソール:

```
src/main/resources/scripts/setupPublic2Cluster.sh
```

11. アプリケーションをテストする

全体の流れを確認するには、**skupper** という単語を含むツイートを投稿する必要があります。その後、Telegram チャンネルに **New feedback about Skupper** というタイトルの新しいメッセージが表示されます。

private1 のコンソール:

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

## 出力サンプル

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

public1 のコンソール:

```
kamel logs twitter-route
```

## 出力サンプル

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

## 第5章 SKUPPER を使用した FTP サーバーへのアクセス

リモート Kubernetes クラスター上の FTP サーバーに安全に接続します。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例では、Skupper を使用して、ある Kubernetes クラスター上の FTP クライアントを別の Kubernetes クラスター上の FTP サーバーに接続する方法を示します。

FTP などのマルチポートサービスでの Skupper の使用法を示します。これは、パッシブモードの FTP (最近ではより一般的) と [制限されたポート範囲](#) を使用して、Skupper の設定を簡素化します。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも1つの Kubernetes クラスターにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [名前空間を設定する](#)
- [FTP サーバーをデプロイする](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [FTP サーバーを公開する](#)
- [FTP クライアントを実行する](#)
  1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
  2. Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを1回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

3. 名前空間を設定します。  
Skupper は、通常は異なるクラスター上の複数の Kubernetes 名前空間で使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、[kubeconfig](#) と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)
- [IBM Kubernetes Service](#)
- [OpenShift](#)

### パブリック:

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

### プライベート:

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

4. FTP サーバーをデプロイします。  
プライベートでは、**kubectl apply** を使用して FTP サーバーをデプロイします。

### プライベート:

```
kubectl apply -f server
```

### 出力サンプル

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

5. サイトを作成します。

Skupper サイトは、アプリケーションのコンポーネントが実行される場所です。サイトは相互にリンクされ、アプリケーションのネットワークを形成します。Kubernetes では、サイトは名前空間に関連付けられます。

名前空間ごとに、**skupper init** を使用してサイトを作成します。これにより、Skupper ルーターとコントローラーがデプロイされます。次に、**skupper status** を使用して結果を確認します。

#### パブリック:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

#### プライベート:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

#### 6. サイトをリンクします。

Skupper リンクは、2つのサイト間の通信チャンネルです。リンクは、アプリケーションの接続と要求のトランスポートとして機能します。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリン

クを作成します。



### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、パブリックサイトの **skupper token create** を使用してトークンを生成します。次に、プライベートサイトで **skupper link create** を使用して、サイトをリンクします。

#### パブリック:

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

#### 出力サンプル

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

#### プライベート:

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

#### 出力サンプル

```
$ 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'.
```

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用後、または作成後15分後に期限切れになります。

#### 7. FTP サーバーを公開します。

プライベートでは、**skupper expose** を使用して、リンクされたすべてのサイトで FTP サーバーを公開します。

#### プライベート:

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

#### 出力サンプル

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

#### 8. FTP クライアントを実行します。

パブリックでは、**kubectl run** と **curl** イメージを使用して、FTP の put および get 操作を実行します。

#### パブリック:

```
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
```

#### 出力サンプル

```
$ 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
```

## 第6章 IPERF

iPerf3 を使用しながら、リアルタイムのネットワークスループット測定を実行します。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

このチュートリアルでは、iperf3 ツールを使用して Kubernetes 全体でリアルタイムのネットワークスループット測定を実行する方法を説明します。このチュートリアルでは、次のことを行います。

- iperf3 を 3 つの別々のクラスターにデプロイする
- iperf3 クライアントテストインスタンスを実行する

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降
- パフォーマンスを確認するために 3 つのクラスターにアクセスできる。たとえば、3 つのクラスターは、以下のような構成になります。
- ローカルマシン上で実行されるプライベートクラウドクラスター (**private1**)
- パブリッククラウドプロバイダーで実行されている 2 つのパブリッククラウドクラスター (**public1** と **public2**)

### 手順

- [この例のリポジトリのクローンを作成する](#)
- Skupper コマンドラインツールをインストールする
- 個別のコンソールセッションを設定する
- クラスターにアクセスする
- 名前空間を設定する
- 名前空間に Skupper をインストールする
- 名前空間のステータスを確認する
- 名前空間をリンクする
- iperf3 サーバーをデプロイする
- 各名前空間から iperf3 を公開する
- クラスター全体でベンチマークテストを実行する
  1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
  2. Skupper コマンドラインツールをインストールします。

**skupper** コマンドラインツールは、Skupper をインストールおよび設定するためのエントリーポイントです。開発環境ごとに **skupper** コマンドを1回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

Windows およびその他のインストールオプションについては、[Installing Skupper](#) を参照してください。

### 3. 個別のコンソールセッションを設定します。

Skupper は、通常は異なるクラスター上の複数の名前空間で使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、**kubeconfig** と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

各名前空間のコンソールセッションを開始します。各セッションで **KUBECONFIG** 環境変数を異なるパスに設定します。

**public1** のコンソール:

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

**public2** のコンソール:

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

**private1** のコンソール:

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

### 4. クラスターにアクセスします。

Kubernetes クラスターにアクセスする手順はプロバイダーによって異なります。[選択したプロバイダーの手順](#)を確認し、それを使用して各コンソールセッションのアクセスを認証および設定します。

### 5. 名前空間を設定します。

**kubectl create namespace** を使用して、使用する名前空間を作成します (または既存の名前空間を使用します)。**kubectl config set-context** を使用して、各セッションの現在の名前空間を設定します。

**public1** のコンソール:

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

**public2 のコンソール:**

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

**private1 のコンソール:**

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

- 名前空間に Skupper をインストールします。

**skupper init** コマンドは、現在の名前空間に Skupper ルーターとコントローラーをインストールします。各 namespace で **skupper init** コマンドを実行します。

**public1 のコンソール:**

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

**public2 のコンソール:**

```
skupper init
```

**private1 のコンソール:**

```
skupper init
```

## 出力サンプル

```
$ 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.
```

- 名前空間のステータスを確認します。

各コンソールで **skupper status** を使用して、Skupper がインストールされていることを確認します。

**public1 のコンソール:**

```
skupper status
```

**public2 のコンソール:**

```
skupper status
```

**private1 のコンソール:**

```
skupper status
```

## 出力サンプル

```
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'
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

8. 名前空間をリンクします。  
リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモート名前空間で、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成した名前空間へのリンクを作成します。



### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、名前空間にリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、1つの名前空間で **skupper token create** を使用してトークンを生成します。次に、もう1つで **skupper link create** を使用してリンクを作成します。

public1 のコンソール:

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

public2 のコンソール:

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

private1 のコンソール:

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

コンソールセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用後、または作成後15分後に期限切れになります。

9. iperf3 サーバーをデプロイします。  
アプリケーションルーターネットワークを作成したら、各名前空間に **iperf3** をデプロイします。

private1 のコンソール:

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

public1 のコンソール:

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

public2 のコンソール:

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

10. 各名前空間から iperf3 を公開します。  
 名前空間間の接続を確立し、**iperf3** をデプロイしました。パフォーマンスをテストする前に、各 namespace から **iperf3** にアクセスする必要があります。

private1 のコンソール:

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

public1 のコンソール:

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

public2 のコンソール:

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

11. クラスター全体でベンチマークテストを実行します。  
 iperf3 サーバーをプライベートクラウドクラスターとパブリッククラウドクラスターにデプロイすると、仮想アプリケーションネットワークでは、サーバーが別々のクラスターで実行されていても通信が可能になります。

private1 のコンソール:

```
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
```

public1 のコンソール:

```
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
```

public2 のコンソール:

```
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
```

## 第7章 SKUPPER を使用した KAFKA へのアクセス

パブリッククラウドリソースを使用したプライベート Kafka クラスターからのデータの処理

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例は、Skupper を使用して、パブリックインターネットに公開せずにリモートサイトの Kafka クラスターにアクセスする方法を示したシンプルな Kafka アプリケーションです。

これには 2 つのサービスが含まれます。

- プライベートデータセンターで実行されている "cluster1" という名前の Kafka クラスター。クラスターには "topic1" という名前のトピックがあります。
- パブリッククラウドで実行される Kafka クライアント。"topic1" に 10 個のメッセージを送信し、再度受信します。

この例では、Kafka クラスターをセットアップするために、[Strimzi](#) プロジェクトの Kubernetes Operator を使用します。Kafka クライアントは、[Quarkus](#) を使用して構築された Java アプリケーションです。

この例では、プライベートデータセンターとパブリッククラウドを表すために、"private" と "public" という 2 つの Kubernetes 名前空間を使用します。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも 1 つの Kubernetes クラスターにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [名前空間を設定する](#)
- [Kafka クラスターをデプロイする](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [Kafka クラスターを公開する](#)
- [クライアントを実行する](#)
  1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
  2. Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを 1 回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

### 3. 名前空間を設定します。

Skupper は、通常は異なるクラスター上の複数の Kubernetes 名前空間で使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、[kubeconfig](#) と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



#### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)
- [IBM Kubernetes Service](#)
- [OpenShift](#)

#### パブリック:

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

#### プライベート:

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

### 4. Kafka クラスターをデプロイします。

プライベートでは、リストされた YAML ファイルを使用して **kubectl create** コマンドと **kubectl apply** コマンドを使用し、Operator をインストールしてクラスターとトピックをデプロイします。

## プライベート:

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

## 出力サンプル

```
$ 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
```

## 注記:

デフォルトでは、Kafka ブートストラップサーバーは、ドメイン名に Kubernetes 名前空間が含まれるブローカーアドレスを返します。この例のように、Kafka クライアントが Kafka クラスターとは異なる名前空間で実行されている場合、クライアントは Kafka ブローカーを解決できなくなります。

Kafka ブローカーにアクセスできるようにするには、各ブローカーの **advertisedHost** プロパティを、Kafka クライアントがリモートサイトで解決できるドメイン名に設定します。この例では、次のリスナー設定を使用してこれを行います。

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

詳細は、[Advertised addresses for brokers](#) を参照してください。

#### 5. サイトを作成します。

Skupper サイトは、アプリケーションのコンポーネントが実行される場所です。サイトは相互にリンクされ、アプリケーションのネットワークを形成します。Kubernetes では、サイトは名前空間に関連付けられます。

名前空間ごとに、**skupper init** を使用してサイトを作成します。これにより、Skupper ルーターとコントローラーがデプロイされます。次に、**skupper status** を使用して結果を確認します。

#### パブリック:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

#### プライベート:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

#### 6. サイトをリンクします。

Skupper リンクは、2つのサイト間の通信チャンネルです。リンクは、アプリケーションの接続と要求のトランスポートとして機能します。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリンクを作成します。



#### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、パブリックサイトの **skupper token create** を使用してトークンを生成します。次に、プライベートサイトで **skupper link create** を使用して、サイトをリンクします。

#### パブリック:

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

#### 出力サンプル

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

#### プライベート:

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

#### 出力サンプル

```
$ 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'.
```

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要があります。デフォルトでは、トークンは1回の使用后、または作成後15分後に期限切れになります。

## 7. Kafka クラスターを公開します。

プライベートでは、**--headless** オプションを指定した **skupper expose** を使用して、Kafka クラスターを Skupper ネットワーク上のヘッドレスサービスとして公開します。

次に、パブリックで **kubectl get service** コマンドを使用して、しばらくすると **cluster1-kafka-brokers** サービスが表示されることを確認します。

プライベート:

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

出力サンプル

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

パブリック:

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

出力サンプル

```
$ 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. クライアントを実行します。

**kubectl run** コマンドを使用して、パブリックでクライアントプログラムを実行します。

パブリック:

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

出力サンプル

```
$ 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
[...]
```

クライアントコードを確認するには、このプロジェクトの [クライアントディレクトリー](#) を参照してください。

## 第8章 患者ポータル

パブリッククラウドで実行され、データをプライベートデータベースに保存する、シンプルなデータベースの Web アプリケーションです。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例は、Skupper を使用して、パブリックインターネットに公開せずにリモートサイトのデータベースにアクセスする方法を示す、データベースをバックアップしたシンプルな Web アプリケーションです。

これには、3つのサービスが含まれます。

- プライベートデータセンター内のベアメタルまたは仮想マシン上で実行される PostgreSQL データベース。
- プライベートデータセンターの Kubernetes 上で実行される支払い処理サービス。
- パブリッククラウドの Kubernetes 上で実行される Web フロントエンドサービス。PostgreSQL データベースと支払い処理サービスを使用します。

この例では、**private** と **public** という2つの Kubernetes 名前空間を使用して、プライベートデータセンター内の Kubernetes クラスターとパブリッククラウド内のクラスターを表します。Podman を使用してデータベースを実行します。

### 前提条件

- **kubectl** コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも1つの Kubernetes クラスターにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [Kubernetes 名前空間を設定する](#)
- [Podman ネットワークを設定する](#)
- [アプリケーションのデプロイ](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [アプリケーションサービスを公開する](#)
- [フロントエンドにアクセスする](#)

1. この例のリポジトリをクローンします。<https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。

- Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを1回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

- Kubernetes 名前空間を設定します。  
Skupper は、通常は異なるクラスター上の複数の Kubernetes 名前空間で使用するよう設計されています。**skupper** コマンドと **kubectl** コマンドは、[kubeconfig](#) と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)
- [IBM Kubernetes Service](#)
- [OpenShift](#)

### パブリック:

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

### プライベート:

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

4. Podman ネットワークを設定します。

新しいターミナルウィンドウを開き、**SKUPPERPLATFORM** 環境変数を **podman** に設定します。これにより、このターミナルセッションの Skupper プラットフォームが Podman に設定されます。

**podman network create** を使用して、Skupper が使用する Podman ネットワークを作成します。

**systemctl** を使用して Podman API サービスを有効にします。

Podman:

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

**systemctl** コマンドが機能しない場合は、代わりに **podman system service** コマンドを試してください。

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

5. アプリケーションをデプロイします。

**kubectl apply** を使用して、フロントエンドと支払いプロセッサを Kubernetes にデプロイします。**podman run** を使用して、ローカルマシンでデータベースを起動します。



#### 注記

Skupper がサービスにアクセスするために作成するコンテナとの衝突を避けるために、実行中のコンテナに **--name** を使用して名前を付けることが重要です。



#### 注記

**podman run** コマンドでは **--network skupper** を使用する必要があります。

パブリック:

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

プライベート:

```
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. サイトを作成します。

パブリック:

```
skupper init
```

プライベート:

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

Podman:

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

## 7. サイトをリンクします。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリンクを作成します。



### 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、パブリックサイトの **skupper token create** を使用してトークンを生成します。次に、プライベートサイトで **skupper link create** を使用して、サイトをリンクします。

パブリック:

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

プライベート:

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

Podman:

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

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用后、または作成後15分後に期限切れになります。

## 8. アプリケーションサービスを公開します。

プライベートでは、**skupper expose** を使用して支払い処理サービスを公開します。

Podman では、**skupper service create** と **skupper service bind** を使用して、Skupper ネットワーク上のデータベースを公開します。

次に、パブリックで **skupper service create** を使用して利用できるようにします。



### 注記

Podman サイトは、リモートサイトにサービスを自動的に複製しません。サービスを利用可能にする各サイトで **skupper service create** を使用する必要があります。

#### プライベート:

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

#### Podman:

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

#### パブリック:

```
skupper service create database 5432
```

9. フロントエンドにアクセスします。  
アプリケーションを使用およびテストするには、フロントエンドへの外部アクセスが必要です。

**--type LoadBalancer** を指定した **kubectctl expose** を使用して、フロントエンドサービスへのネットワークアクセスを開放します。

フロントエンドが公開されたら、**kubectctl get service/frontend** を使用して、フロントエンドサービスの外部 IP を検索します。外部 IP が **<pending>** の場合は、しばらくしてからもう一度お試しください。

外部 IP を取得したら、**curl** または同様のツールを使用して、そのアドレスの **/api/health** エンドポイントを要求します。



### 注記

次のコマンドの **<external-ip>** フィールドはプレースホルダーです。実際の値は IP アドレスです。

#### パブリック:

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

#### 出力サンプル

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

$ kubectctl 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
```

すべてが正常であれば、ブラウザーで <http://<external-ip>:8080/> に移動して Web インターフェイスにアクセスできるようになります。

## 第9章 TRADE ZOO

パブリッククラウドで実行され、データはプライベート Kafka クラスターに保存されるシンプルな取引アプリケーションです。

この例は、Skupper を使用してクラウドプロバイダー、データセンター、エッジサイト間でサービスを接続できるさまざまな方法を紹介する [一連の例](#) の一部です。

### 概要

この例は、Skupper を使用して、パブリックインターネットに公開せずにリモートサイトの Kafka クラスターにアクセスする方法を示したシンプルな Kafka アプリケーションです。

これには、4つのサービスが含まれています。

- プライベートデータセンターで実行されている Kafka クラスター。クラスターには、"orders" と "updates" という2つのトピックがあります。
- パブリッククラウドで実行される order プロセッサ。"orders" から消費され、買いと売りのオファーをマッチングさせて取引を行います。新規および更新された注文と取引を "updates" として公開します。
- パブリッククラウドで実行される市場データサービス。完了した取引を調べて最新の価格と平均価格を計算し、それを "updates" として公開します。
- パブリッククラウドで実行される Web フロントエンドサービス。売買注文を "orders" に送信し、"updates" から消費して、何が起きているかを表示します。

この例では、Kafka クラスターをセットアップするために、[Strimzi](#) プロジェクトの Kubernetes Operator を使用します。その他のサービスは小さな Python プログラムです。

この例では、プライベートデータセンターとパブリッククラウドを表すために、"private" と "public" という2つの Kubernetes 名前空間を使用します。

### 前提条件

- [kubectI](#) コマンドラインツール、バージョン 1.15 以降 ([インストールガイド](#))
- [任意のプロバイダー](#) から少なくとも1つの Kubernetes クラスターにアクセスできる

### 手順

- [この例のリポジトリのクローンを作成する](#)
- [Skupper コマンドラインツールをインストールする](#)
- [名前空間を設定する](#)
- [Kafka クラスターをデプロイする](#)
- [アプリケーションサービスをデプロイする](#)
- [サイトを作成する](#)
- [サイトをリンクする](#)
- [Kafka クラスターを公開する](#)

- フロントエンドにアクセスする

1. この例のリポジトリをクローンします。 <https://skupper.io/examples/index.html> から適切な GitHub リポジトリに移動し、リポジトリをクローンします。
2. Skupper コマンドラインツールをインストールします。  
この例では、Skupper コマンドラインツールを使用して Skupper をデプロイします。開発環境ごとに **skupper** コマンドを1回だけインストールする必要があります。

CLI のインストールの詳細は、[インストール](#) を参照してください。設定されたシステムの場合は、次のコマンドを使用します。

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

3. 名前空間を設定します。

Skupper は、通常は異なるクラスター上の複数の Kubernetes 名前空間で使用するよう設計されています。**skupper** コマンドと **kubectrl** コマンドは、[kubeconfig](#) と現在のコンテキストを使用して、動作する名前空間を選択します。

kubeconfig はホームディレクトリーのファイルに保存されます。**skupper** および **kubectrl** コマンドは、**KUBECONFIG** 環境変数を使用してこれを検索します。

1つの kubeconfig は、ユーザーごとにアクティブコンテキストを1つだけサポートします。この演習では複数のコンテキストを一度に使用するため、個別の kubeconfig を作成する必要があります。

名前空間ごとに、新しいターミナルウィンドウを開きます。各ターミナルで、**KUBECONFIG** 環境変数を別のパスに設定し、クラスターにログインします。次に、使用する名前空間を作成し、現在のコンテキストに名前空間を設定します。



### 注記

ログイン手順はプロバイダーによって異なります。以下のドキュメントを参照してください。

- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Azure Kubernetes Service \(AKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)
- [IBM Kubernetes Service](#)
- [OpenShift](#)

### パブリック:

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

### プライベート:

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

```
# Enter your provider-specific login command
kubectl create namespace private
kubectl config set-context --current --namespace private
```

#### 4. Kafka クラスターをデプロイします。

プライベートでは、リストされた YAML ファイルを使用して **kubectl create** コマンドと **kubectl apply** コマンドを使用し、Operator をインストールしてクラスターとトピックをデプロイします。

##### プライベート:

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

##### 注記:

デフォルトでは、Kafka ブートストラップサーバーは、ドメイン名に Kubernetes 名前空間が含まれるブローカーアドレスを返します。この例のように、Kafka クライアントが Kafka クラスターとは異なる名前空間で実行されている場合、クライアントは Kafka ブローカーを解決できなくなります。

Kafka ブローカーにアクセスできるようにするには、各ブローカーの **advertisedHost** プロパティを、Kafka クライアントがリモートサイトで解決できるドメイン名に設定します。この例では、次のリスナー設定を使用してこれを行います。

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

詳細は、[Advertised addresses for brokers](#) を参照してください。

#### 5. アプリケーションサービスをデプロイします。

パブリックでは、リストされた YAML ファイルとともに **kubectl apply** コマンドを使用して、アプリケーションサービスをインストールします。

##### パブリック:

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

#### 6. サイトを作成します。

Skupper サイトは、アプリケーションのコンポーネントが実行される場所です。サイトは相互にリンクされ、アプリケーションのネットワークを形成します。Kubernetes では、サイトは名前空間に関連付けられます。

名前空間ごとに、**skupper init** を使用してサイトを作成します。これにより、Skupper ルーターとコントローラーがデプロイされます。次に、**skupper status** を使用して結果を確認します。

#### パブリック:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

#### プライベート:

```
skupper init
skupper status
```

#### 出力サンプル

```
$ 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.
```

以下の手順を実行する場合は、いつでも **skupper status** を使用して進捗状況を確認できます。

#### 7. サイトをリンクします。

Skupper リンクは、2つのサイト間の通信チャンネルです。リンクは、アプリケーションの接続と要求のトランスポートとして機能します。

リンクを作成するには、**skupper token create** と **skupper link create** の2つの **skupper** コマンドを組み合わせて使用する必要があります。

**skupper token create** コマンドは、リンクを作成する権限を示す秘密トークンを生成します。トークンにはリンクの詳細も含まれます。次に、リモートサイトで、**skupper link create** コマンドを実行すると、トークンを使用して、トークンを生成したサイトへのリンクを作成します。



## 注記

リンクトークンは実際にはシークレットです。トークンがある場合は、サイトにリンクできます。信頼できる人だけがアクセスできるようにしてください。

まず、パブリックサイトの **skupper token create** を使用してトークンを生成します。次に、プライベートサイトで **skupper link create** を使用して、サイトをリンクします。

### パブリック:

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

### 出力サンプル

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

### プライベート:

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

### 出力サンプル

```
$ 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'.
```

ターミナルセッションが異なるマシン上にある場合は、トークンを安全に転送するために **scp** または同様のツールを使用する必要がある場合があります。デフォルトでは、トークンは1回の使用后、または作成後 15 分後に期限切れになります。

### 8. Kafka クラスターを公開します。

プライベートでは、**--headless** オプションを指定した **skupper expose** を使用して、Kafka クラスターを Skupper ネットワーク上のヘッドレスサービスとして公開します。

次に、Public で **kubectl get service** を使用して、しばらくすると **cluster1-kafka-brokers** サービスが表示されることを確認します。

### プライベート:

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

### パブリック:

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

### 9. フロントエンドにアクセスします。

アプリケーションを使用およびテストするには、フロントエンドへの外部アクセスが必要です。

プライベートIPアドレスを指定して、kubectl expose を使用して、フロントエンドが公共IPア

**--type LoadBalancer** を指定した **kubecti expose** を使用して、フロントエンドサービスへのネットワークアクセスを開放します。

フロントエンドが公開されたら、**kubecti get service/frontend** を使用して、フロントエンドサービスの外部 IP を検索します。外部 IP が **<pending>** の場合は、しばらくしてからもう一度お試しください。

外部 IP を取得したら、**curl** または同様のツールを使用して、そのアドレスの **/api/health** エンドポイントを要求します。



### 注記

次のコマンドの **<external-ip>** フィールドはプレースホルダーです。実際の値は IP アドレスです。

### パブリック:

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

### 出力サンプル

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

$ kubecti 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
```

すべてが正常であれば、ブラウザで <http://<external-ip>:8080/> に移動して Web インターフェイスにアクセスできるようになります。