

## 121Kubernetes 系列（一一五）使用 Skupper 进行 Kubernetes 多集群负载均衡

在本文中，您将了解如何利用 Skupper 在多个 Kubernetes 集群上运行的应用程序实例之间实现负载平衡。我们将使用 Kind 在本地创建一些 Kubernetes 集群。然后我们将使用 Skupper 连接它们。

Skupper集群互连工作在第7层（应用层）。这意味着无需创建任何 VNP 或特殊的防火墙规则。Skupper 按照虚拟应用网络 (VAN) 方法进行工作。因此，它可以连接不同的 Kubernetes 集群并保证服务之间的通信，而无需将它们暴露在互联网上。您可以在 Skupper 文档中(<https://skupper.io/docs/overview/index.html>)阅读有关其背后概念的更多信息。

### 源代码

如果您想自己尝试一下，可以随时查看源代码 (<https://github.com/piomin/sample-spring-kotlin-microservice.git>)。这次我们将使用命令行工具 ( skupper CLI) 完成几乎所有操作。该存储库仅包含一个示例应用程序 Spring Boot 以及 Kubernetes Deployment 清单和 Skaffold 配置。📄

### 使用 Kind 创建 Kubernetes 集群

第一步，我们将使用 Kind 创建三个 Kubernetes 集群。我们需要给它们不同的名称： `c1` 、 `c2` 和 `c3` 。因此，它们可以在上下文名称下使用： `kind-c1` 、 `kind-c2` 和 `kind-c3` 。

```
$ kind create cluster --name c1
$ kind create cluster --name c2
$ kind create cluster --name c3
```

默认情况下，Skupper 将自身公开为 Kubernetes LoadBalancer Service 。因此，我们需要在 Kind 上启用负载均衡器。为此，我们可以安装 MetalLB。您可以在此处(<https://kind.sigs.k8s.io/docs/user/loadbalancer/>)的 Kind 文档中找到完整的安装说明。首先，我们切换到 `c1` 集群：

```
$ kubectlx kind-c1
```

然后，我们必须应用以下 YAML 清单：

```
$ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.7/config/manifests/metallb-native.yaml
```

您应该对其他两个集群重复相同的过程： `c2` 和 `c3` 。然而，这还不是全部。我们还需要设置负载均衡器使用的地址池。为此，我们首先检查 Kind 使用的 Docker 网络上的 IP 地址范围。对我来说是 `172.19.0.0/16 172.19.0.1` 。

```
$ docker network inspect -f '{{.IPAM.Config}}' kind
```

根据结果，我们需要为所有三种集群选择正确的 IP 地址。然后我们必须创建 IPAddressPool 对象，其中包含 IP 范围。以下是 c1 集群的 YAML 清单：

```
c1-address-pool.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example
  namespace: metallb-system
spec:
  addresses:
    - 172.19.255.200-172.19.255.250
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: empty
  namespace: metallb-system
```

下面是c2 集群池配置。重要的是，地址范围不应与其他两个 Kind 集群中的范围冲突。

```
c2-address-pool.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example
  namespace: metallb-system
spec:
  addresses:
    - 172.19.255.150-172.19.255.199
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: empty
  namespace: metallb-system
```

最后是 c3 集群配置：

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example
  namespace: metallb-system
spec:
  addresses:
    - 172.19.255.100-172.19.255.149
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: empty
  namespace: metallb-system

```

使用 `kubectl apply -f` 命令应用 YAML 清单后，我们可以继续下一部分。

## 在 Kubernetes 上安装 Skupper

我们可以通过两种不同的方式在 Kubernetes 上安装和管理 Skupper：使用 CLI 或通过 YAML 清单。Skupper 文档中的大多数示例都使用 CLI，因此我认为这是一种更好的方法。因此，在开始使用 Kubernetes 之前，我们需要安装 CLI。您可以在[此处的 Skupper 文档](https://skupper.io/docs/cli/index.html#installing-cli)中找到安装说明。安装后，只需使用以下命令验证它是否有效：

```
$ skupper version
```

之后，我们就可以继续操作 Kubernetes 集群了。我们将在所有三个集群中创建相同的命名空间 `interconnect`。为了简化我们即将进行的练习，我们还可以为每个上下文设置默认命名空间（或者您可以使用 `kubectl config set-context --current --namespace interconnect` 命令来完成此操作）。

```
$ kubectl create ns interconnect
$ kubens interconnect
```

然后，我们切换到 `kind-c1` 集群。我们将保持在这种背景下直到我们的练习结束。

```
$ kubectx kind-c1
```

最后，我们将在 Kubernetes 集群上安装 Skupper。为此，我们必须执行 `skupper init` 命令。幸运的是，它允许我们使用 `-c` 参数设置目标 Kubernetes 上下文。在 `kind-c1` 集群内，我们还将启用 Skupper UI 仪表板（`--enable-console` 参数）。使用 Skupper 控制台，我们可以可视化 Skupper 网络中所有目标的流量。

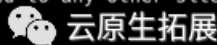
```
$ skupper init --enable-console --enable-flow-collector
$ skupper init -c kind-c2
$ skupper init -c kind-c3
```

让我们验证 Skupper 安装的状态：

```
$ skupper status
$ skupper status -c kind-c2
$ skupper status -c kind-c3
```

以下是在 kind-c1 集群中运行的 Skupper 的状态：

```
skupper status
Skupper is enabled for namespace "interconnect" in interior mode. It is not connected to any other sites.
It has no exposed services.
The site console url is: https://172.19.255.201:8010
The credentials for internal console-auth mode are held in secret: 'skupper-console-users'
```



我们还可以显示 interconnect 命名空间中正在运行的 Skupper Pod 的列表：

```
$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
skupper-prometheus-867f57b89-dc4lq	1/1	Running	0	3m36s
skupper-router-55bbb99b87-k4qn5	2/2	Running	0	3m40s
skupper-service-controller-6bf57595dd-45hvw	2/2	Running	0	3m37s

现在，我们的目标是将 **c2** 和 **c3** Kind 集群与 **c1** 集群连接起来。在 Skupper 命名法中，我们必须在源集群和目标集群中的命名空间之间创建链接。在创建链接之前，我们需要生成一个 secret 令牌，表示有权创建链接。该令牌还带有链接详细信息。我们在目标集群上生成两个令牌。每个令牌都存储为 YAML 文件。第一个用于 kind-c2 集群 ( skupper-c2-token.yaml )，第二个用于 kind-c3 集群 ( skupper-c3-token.yaml )。

```
$ skupper token create skupper-c2-token.yaml
$ skupper token create skupper-c3-token.yaml
```

我们将考虑使用不同参数创建链接的几种场景。在此之前，让我们在 kind-c2 和 kind-c3 集群上部署示例应用程序。

## 使用 Skaffold 在 Kubernetes 上运行示例应用程序

克隆示例应用程序存储库后，转到主目录。您可以使用以下命令轻松构建应用程序并将其部署到 kind-c2 和 kind-c3 ：

```
$ skaffold dev --kube-context=kind-c2
$ skaffold dev --kube-context=kind-c3
```

部署应用程序 skaffold 后会自动打印所有日志，如下所示。这对我们下一步的练习会有帮助。

```
[sample-spring-kotlin-microservice] 2023-08-03T23:03:41.122Z INFO 1 --- [ restartedMain] o.apache.catalina.core.StandardEngine :  
Starting Servlet engine: [Apache Tomcat/10.1.11]  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:41.643Z INFO 1 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] :  
Initializing Spring embedded WebApplicationContext  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:41.662Z INFO 1 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext :  
Root WebApplicationContext: initialization completed in 12951 ms  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:47.312Z INFO 1 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer :  
LiveReload server is running on port 35729  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:47.334Z INFO 1 --- [ restartedMain] o.s.b.a.e.web.EndpointLinksResolver :  
Exposing 14 endpoint(s) beneath base path '/actuator'  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:47.648Z INFO 1 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer :  
Tomcat started on port(s): 8080 (http) with context path ''  
[sample-spring-kotlin-microservice] 2023-08-03T23:03:47.722Z INFO 1 --- [ restartedMain] pleSpringKotlinMicroserviceApplicationKt :  
Started SampleSpringKotlinMicroserviceApplicationKt in 21.351 seconds (process running for 24.258)
```

我们的应用程序部署在 sample-spring-kotlin-microservice 名称下。

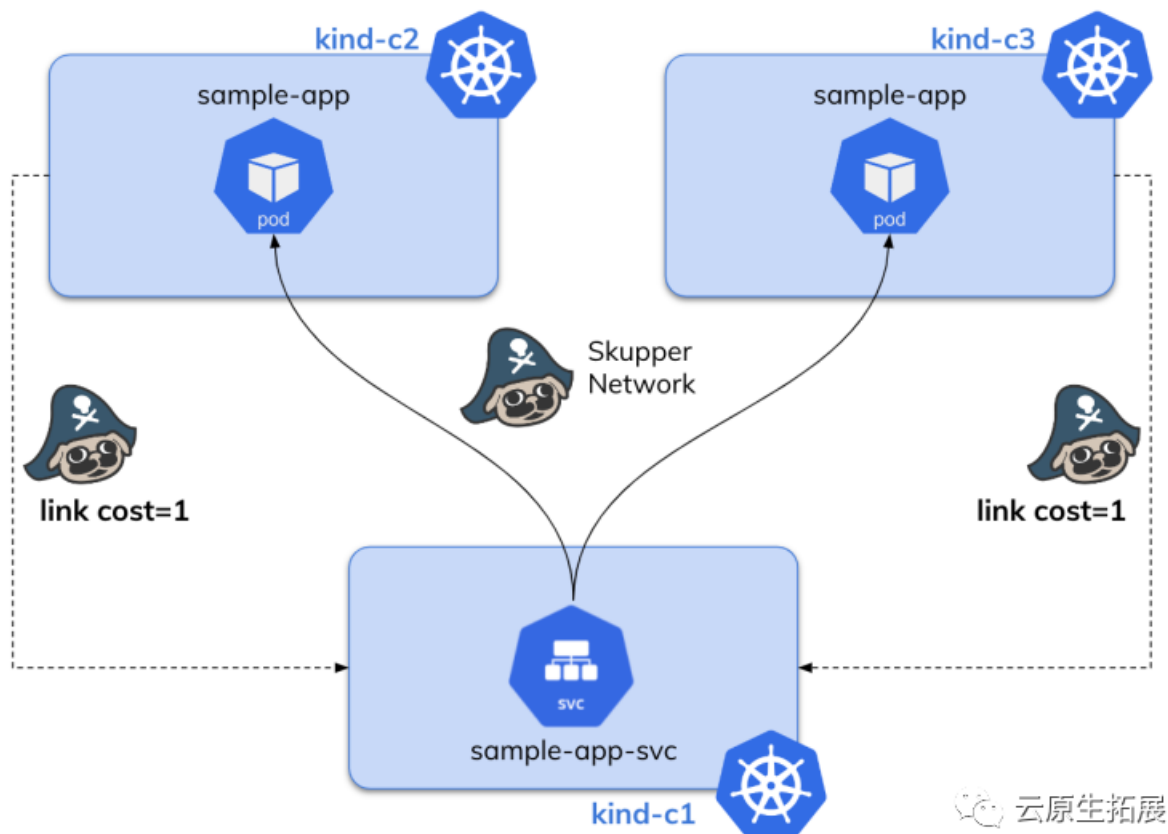
```
kubectl get deploy
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
sample-spring-kotlin-microservice	1/1	1	1	3m16s
skupper-router	1/1	1	1	9m59s
skupper-service-controller	1/1	1	1	9m59s

## 使用 Skupper 进行负载均衡 - 场景

### 场景一：相同的 Pod 数量和链路成本

让我们从最简单的场景开始。我们的应用程序有一个 pod 在 kind-c2 和 kind-c3 集群上运行。在 Skupper 中，我们还可以为每个链接分配一个成本来影响流量。默认情况下，新链接的链接成本设置为 1。在服务网络中，路由算法尝试使用从客户端到目标服务器总成本最低的路径。现在，我们将保留默认值。这是第一个场景的可视化：



让我们使用之前生成的令牌创建指向 c1 Kind 集群的链接。

```
$ skupper link create skupper-c2-token.yaml -c kind-c2
$ skupper link create skupper-c3-token.yaml -c kind-c3
```

如果一切顺利，您应该会看到类似的消息：

```
skupper link create skupper-c3-token.yaml -c kind-c3
Site configured to link to https://172.19.255.201:8081/ac219768-3251-11ee-8d00-acde4604122 (name=link1)
Check the status of the link using 'skupper link status'.
```

我们还可以通过执行以下命令来验证链接的状态：


```
$ skupper link status -c kind-c2
$ skupper link status -c kind-c3
```

这意味着现在 c2 和 c3 Kind 集群与 c1 集群在同一个 Skupper 网络中“工作”。下一步是将在 c2 和 c3 集群中运行的应用程序公开到 c1 集群中。Skupper 在第 7 层工作，默认情况下，它不会连接应用程序，除非我们不会为特定应用程序启用该功能。为了将我们的应用程序公开给 c1 集群，我们需要在 c2 和 c3 集群上运行以下命令。

```
$ skupper expose deployment/sample-spring-kotlin-microservice \
--port 8080 -c kind-c2
$ skupper expose deployment/sample-spring-kotlin-microservice \
--port 8080 -c kind-c3
```

让我们看一下目标 ( kind-c1 ) 集群发生了什么。如您所见，Skupper 创建了 sample-spring-kotlin-microservice Kubernetes 服务，将流量转发到 skupper-router pod。Skupper 路由器负责跨属于 Skupper 网络一部分的 Pod 进行负载均衡请求。

```
kubectl describe svc sample-spring-kotlin-microservice
Name:                sample-spring-kotlin-microservice
Namespace:           interconnect
Labels:              <none>
Annotations:         internal.skupper.io/controlled: true
Selector:             application=skupper-router,skupper.io/component=router
Type:                ClusterIP
IP Family Policy:    SingleStack
IP Families:         IPv4
IP:                  10.96.216.233
IPs:                 10.96.216.233
Port:                port8080 8080/TCP
TargetPort:          1024/TCP
Endpoints:           10.244.0.6:1024
Session Affinity:    None
Events:              <none>
```



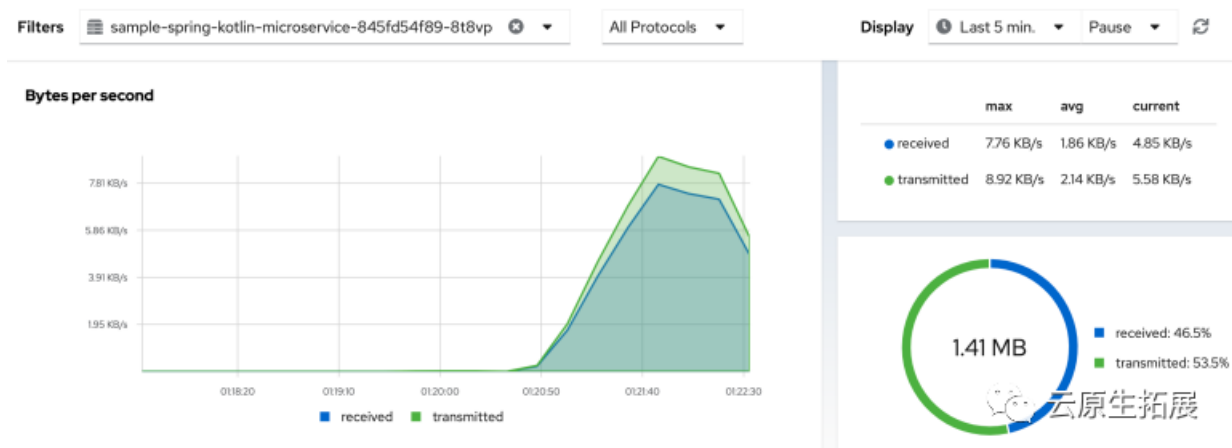
为了简化我们的练习，我们将为上面可见的 Service 启用端口转发。

```
$ kubectl port-forward svc/sample-spring-kotlin-microservice 8080:8080
```

因此，我们不必配置 Kubernetes Ingress 来调用该服务。现在，我们可以通过本地主机发送一些测试请求，例如 siege：

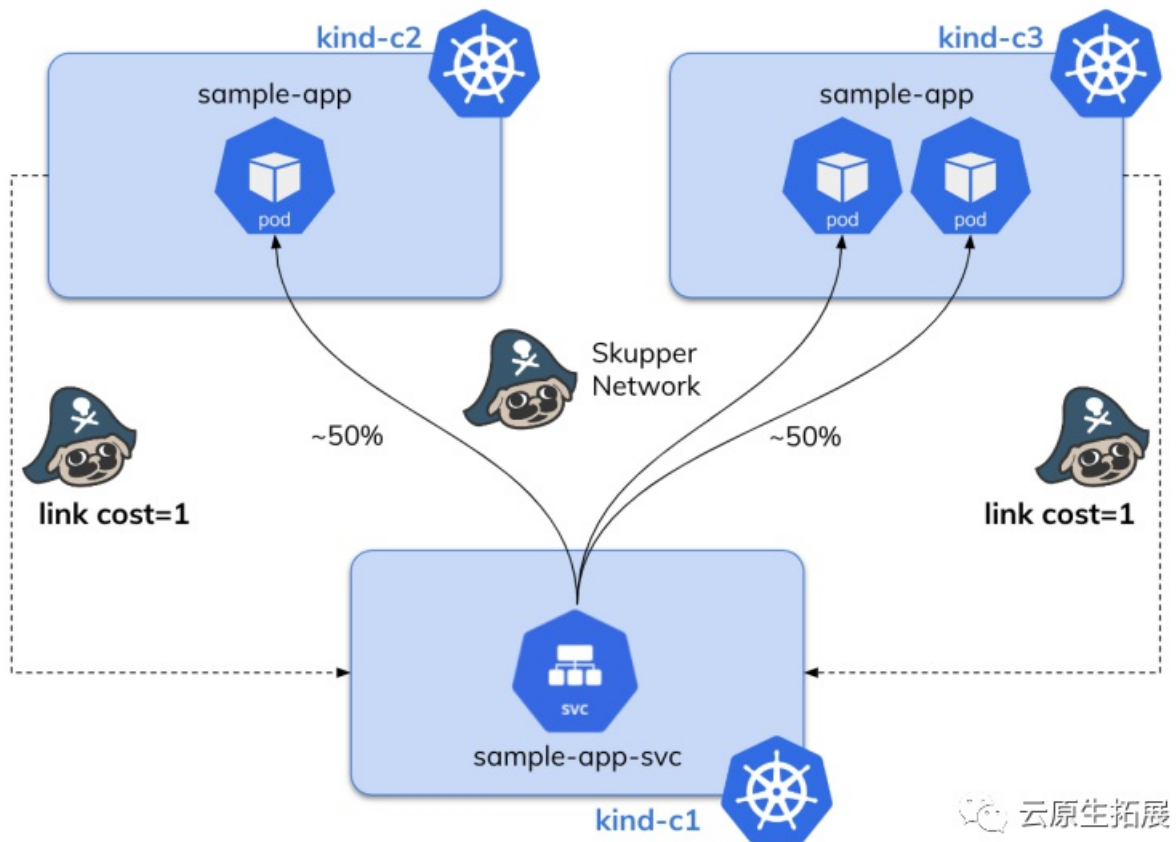
```
$ siege -r 200 -c 5 http://localhost:8080/persons/1
```

通过查看日志，我们可以轻松验证流量是否传入 kind-c2 和 kind-c3 上运行的 Pod。或者，我们可以转到 Skupper 控制台并查看流量可视化：



## 场景二：不同数量的 Pod 和相同的链路成本

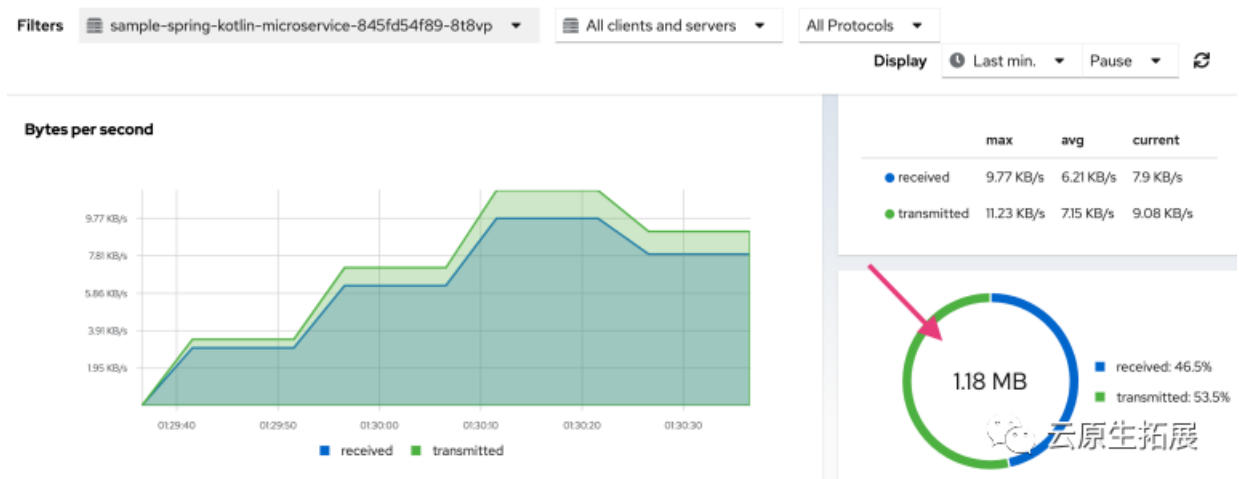
在下一个场景中，我们不会更改 Skupper 网络配置中的任何内容。我们将仅在 kind-c3 集群中运行应用程序的第二个 Pod。现在，kind-c2 集群中运行一个 pod，kind-c3 集群中运行两个 pod。这是我们的架构。



再次，我们可以使用 `siege` 命令向之前测试的 Kubernetes Service 发送一些请求：

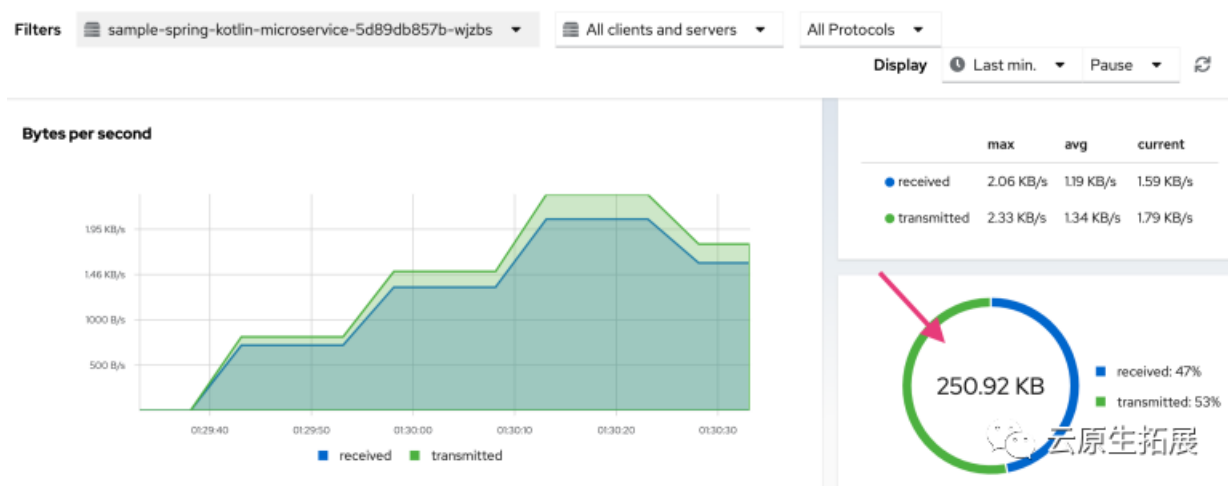
```
$ siege -r 200 -c 5 http://localhost:8080/persons/2
```

让我们看一下 Skupper 仪表板中的流量可视化。我们可以在所有可用的 Pod 之间切换。这是在 kind-c2 集群中运行的 Pod 的图表。



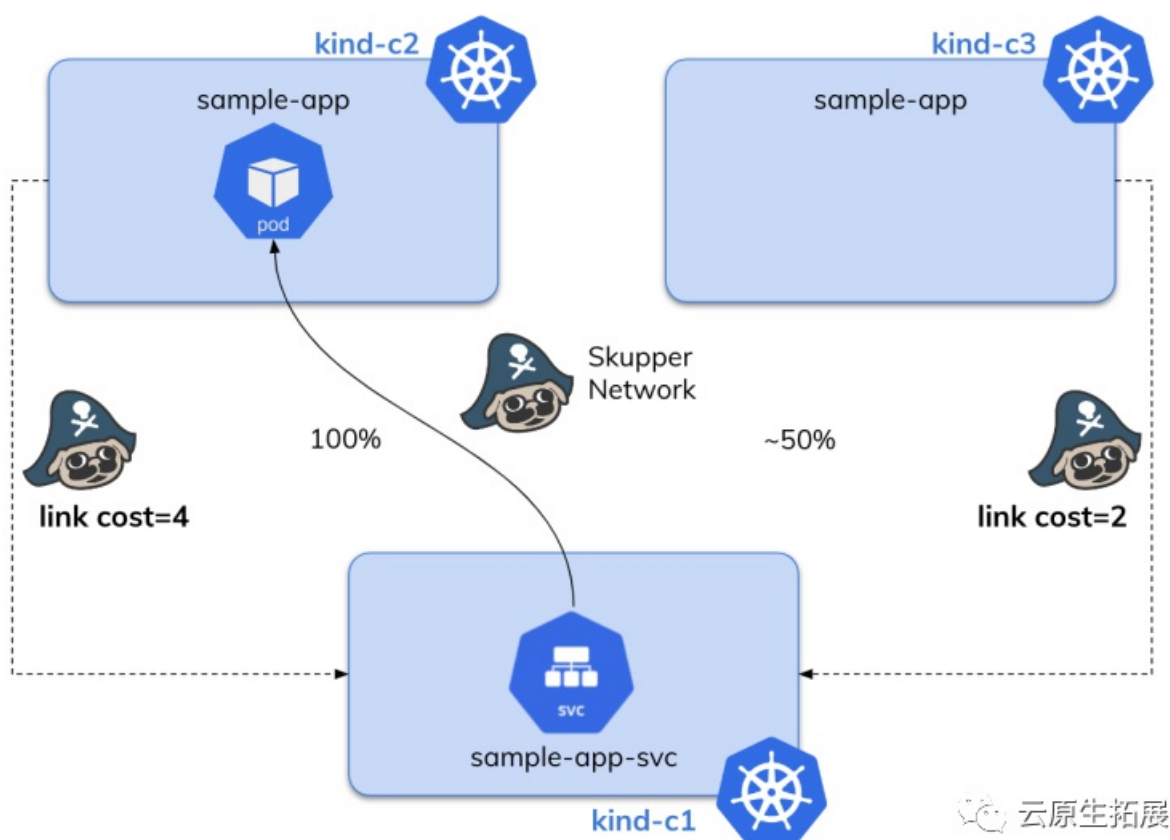


这是在 kind-c3 集群中运行的 Pod 的相同图表。正如您所看到的，它仅接收 kind-c2 集群中 pod 接收到的流量的大约 50%（甚至更少，具体取决于我们想象的 pod）。这是因为 Skupper 在 kind-c3 集群中运行着两个 pod，而 Skupper 仍然平等地平衡集群之间的请求。



### 场景三：只有一个Pod，链路成本不同

在当前场景中，应用程序的单个 Pod 运行在 c2 Kind 集群上。同时，c3 集群上没有 pod（Deployment 存在，但已缩减至零个实例）。这是我们场景的可视化。



这里重要的是 Skupper 首选 c3 集群，因为到它的链接的成本（2）比到 c2 集群的链接（4）更低。所以现在，我们需要删除以前的链接，然后使用以下命令创建一个新链接：

```
$ skupper link create skupper-c2-token.yaml --cost 4 -c kind-c2
$ skupper link create skupper-c3-token.yaml --cost 2 -c kind-c3
```

我们看一下Skupper的网络状态：

#### skupper network status

```
└─ [remote] 91961a6 - interconnect
  URL: 172.19.255.100
  name: interconnect
  namespace: interconnect
  sites linked to: 0a82cd9-interconnect
  version: 1.4.2
  └─ Services:
    └─ name: sample-spring-kotlin-microservice
       address: sample-spring-kotlin-microservice: 8080
       protocol: tcp
└─ [local] 0a82cd9 - interconnect
  URL: 172.19.255.200
  mode: interior
  name: interconnect
  namespace: interconnect
  version: 1.4.2
  └─ Services:
    └─ name: sample-spring-kotlin-microservice
       address: sample-spring-kotlin-microservice: 8080
       protocol: tcp
└─ [remote] a21023a - interconnect
  URL: 172.19.255.150
  name: interconnect
  namespace: interconnect
  sites linked to: 0a82cd9-interconnect
  version: 1.4.2
  └─ Services:
    └─ name: sample-spring-kotlin-microservice
       address: sample-spring-kotlin-microservice: 8080
       protocol: tcp
       └─ Targets:
          └─ name: sample-spring-kotlin-microservice-845fd54f89-8t8vp
```

No Targets

云原生拓展

让我们向公开的服务发送一些测试请求。它可以正常工作，没有任何错误。由于只有一个正在运行的 Pod，因此整个流量都会流向那里：

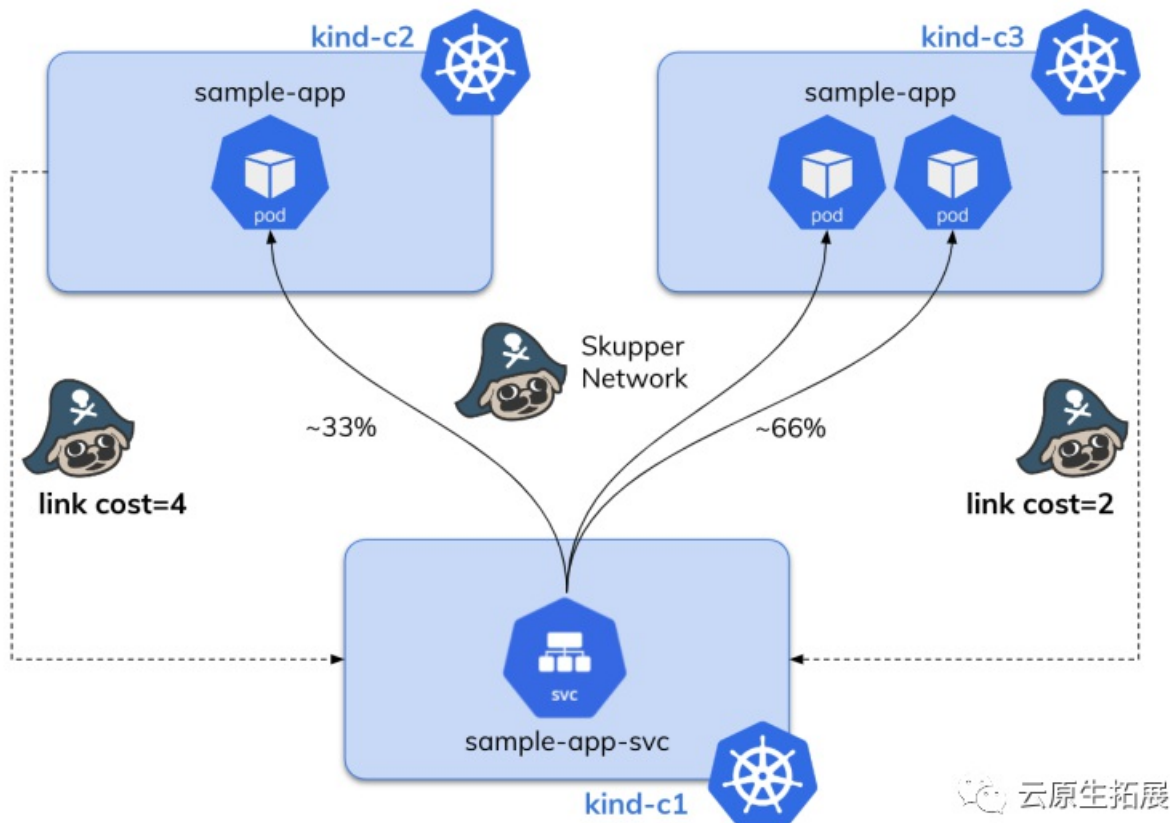
```
siege -r 200 -c 5 http://localhost:8080/persons/100
```

```
{
  "transactions": 1000,
  "availability": 100.00,
  "elapsed_time": 9.23,
  "data_transferred": 0.06,
  "response_time": 0.05,
  "transaction_rate": 108.34,
  "throughput": 0.01,
  "concurrency": 4.94,
  "successful_transactions": 1000,
  "failed_transactions": 0,
  "longest_transaction": 0.12,
  "shortest_transaction": 0.02
}
```

云原生拓展

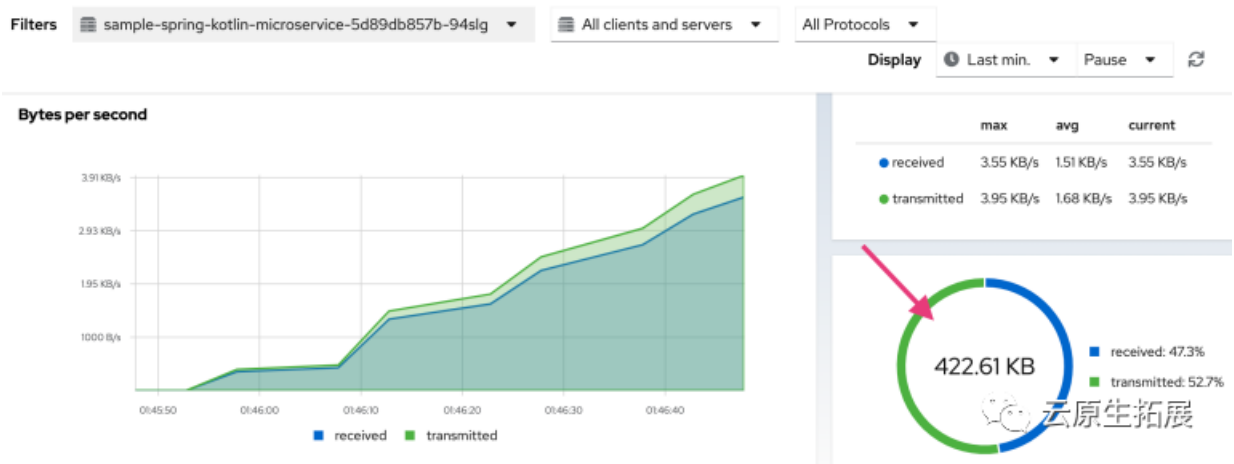
#### 场景四：一个集群中有更多 Pod 且链路成本不同

最后，我们练习中的最后一个场景。我们将使用与场景 3 中相同的 Skupper 配置。但是，这次我们将在 kind-c3 集群中运行两个 pod。

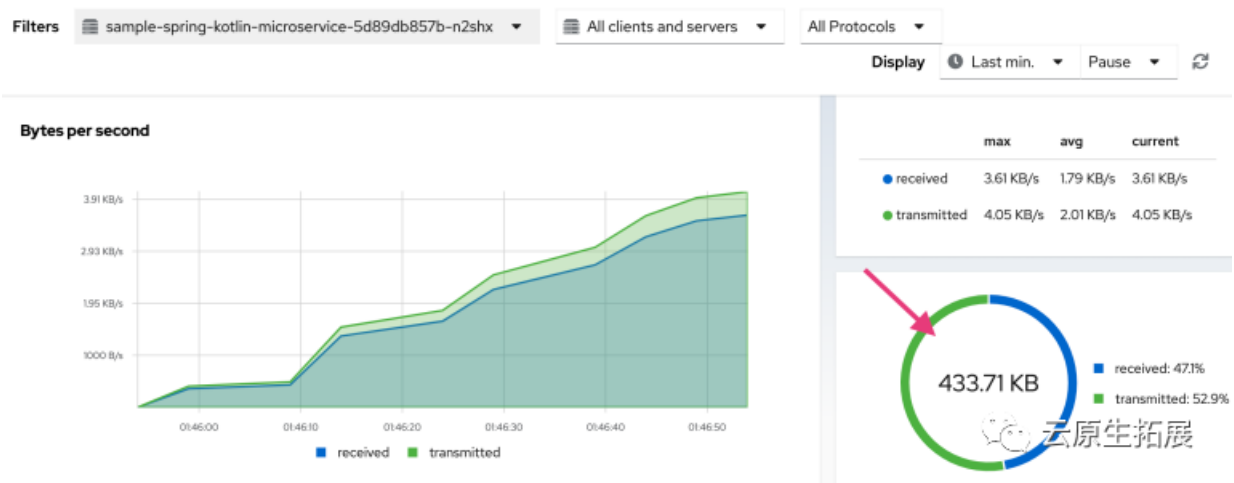


云原生拓展

我们可以再次切换到 Skupper 仪表板。现在，如您所见，所有 Pod 都收到非常相似的流量。这是在 kind-c2 集群上运行的 Pod 的图表。



这是在 kind-c3 集群上运行的 Pod 的类似图表。假设集群上运行的 Pod 数量设置了链路成本后，我能够在两个集群中的所有 Pod 之间平均分配流量。有用。然而，这并不是负载均衡的完美方式。我希望至少有一个选项可以在同一 Skupper 网络中工作的所有 Pod 之间启用循环。除非我们为应用程序启用自动缩放，否则此场景中提供的解决方案将按预期工作。



## 总结

Skupper 引入了一种完全基于第 7 层的 Kubernetes 多集群连接的有趣方法。您可以将其与基于不同层（例如 Submariner 或 Cilium 集群网络）的另一种解决方案进行比较。