Flomesh 软件负载均衡平台部署手册

Version 1.0

2023.02

support@flomesh.cn

目录

1 前提

Flomesh 软件负载均衡(FLB)平台包含控制平面管理控制台(flomesh-gui)、控制平面配置中心（Pipy repo)、数据代理（pipy proxy）和 Kubernetes Pipy Ingress Controller，以及用于持久化配置存储的关系型数据和用于日志存储的非关系型数据库。

用于部署 Kubernetes Ingress Controller 的集群，要求：

- Kubernetes Cluster v1.19+

用于部署控制平面配置中心的机器要求：

- Red Hat Enterprise Linux 7/8 / CentOS 7/8
- Ubuntu 20.04 或以上
- 8GB 或以上内存
- 4 CPU 或以上
- 100GB 磁盘空间或以上

1.1 软件信息

表 1 软件信息*

软件名称	软件版本	用途	备注
flomesh-gui	0.1.0-12	管理控制台	镜像
Pipy	0.90.0	反向代理软件	镜像/RPM
ClickHouse	21.6.3.14	日志数据库	RPM
MySQL	5.7/8.0	控制台数据持久化	如有提供,可直接使用
PostgreSQL**	14.2	控制台数据持久化	

表 2 k8s resources limit 信息

Pod 名称	k8s resource request	k8s resource limit	备注
flomesh-gui	CPU: 1000m Mem: 500Mi	CPU: 2000m Mem: 2000Mi	
pipy-repo	CPU: 1000m Mem: 200Mi	CPU: 1000m Mem: 2000Mi	
fsm-manager	CPU: 200m Mem: 200Mi	CPU: 2000m Mem: 2000Mi	
fsm-ingress	CPU: 500m Mem: 128Mi	CPU: 2000m Mem: 1000Mi	
fsm-repo	CPU: 100m Mem: 20Mi	CPU: 1000m Mem: 500Mi	

注：

* Flomesh 软件负载均衡平台持续更新，本文档适用于当前（2023 年 02 月）版本。

** 管理控制台需要关系型数据库作为配置持久化存储，目前支持 MySQL 与 PostgreSQL，可根据情况选择其一。

1.2 总体架构

Flomesh 软件负载均衡平台分为控制平面与数据平面。

1) 控制平面

- 控制平面 flomesh-gui 提供 REST 和 GQL 接口
- 控制平面数据存储在关系型数据库中（mysql/postgresql/sqlite3）
- flomesh-gui 将配置和脚本推送到 Pipy Repo 节点
- flomesh-gui 将 Ingress 配置推送到 Operator
- Operator 将 flomesh-gui 的配置和监听到的配置推送到 Pipy repo 节点

2）数据平面

- Pipy 代理节点从 Pipy repo 节点拉取配置和脚本
- Pipy 代理节点收集和产生日志、跟踪、指标、事件信息，发送给 NoSQL 数据库（clickhouse、elasticsearch）

1.3 物料清单

- flomesh-gui 容器镜像（traffic-guru:0.1.0-12）

flomesh/traffic-guru:0.1.0-12

- 代理软件容器镜像（pipy-0.90.0）

  flomesh/pipy:0.90.0-185

- 代理软件安装包（pipy-0.90.0）

https://pipy-oss-1255617643.cos-website.ap-beijing.myqcloud.com/repo/pipy/x86_64/binary/pipy-0.90.0-185-generic_linux-x86_64.tar.gz

- ClickHouse >= 22.3

https://packages.clickhouse.com/rpm/stable/clickhouse-client-22.3.20.29.x86_64.rpm

https://packages.clickhouse.com/rpm/stable/clickhouse-common-static-22.3.20.29.x86_64.rpm

https://packages.clickhouse.com/rpm/stable/clickhouse-keeper-22.3.20.29.x86_64.rpm

- MySQL/PostgreSQL 数据库
- 控制平面软件部署文件（flomesh-gui.yml）
- node_exporter

https://pipy-oss-1255617643.cos.ap-beijing.myqcloud.com/repo/infra/node_exporter_1.5.0_linux_amd64.tar.gz

2 参考架构

2.1 容器与 VM 混合模式参考架构

以一个容器集群与物理机/虚拟机混合环境为例，部署架构如下：

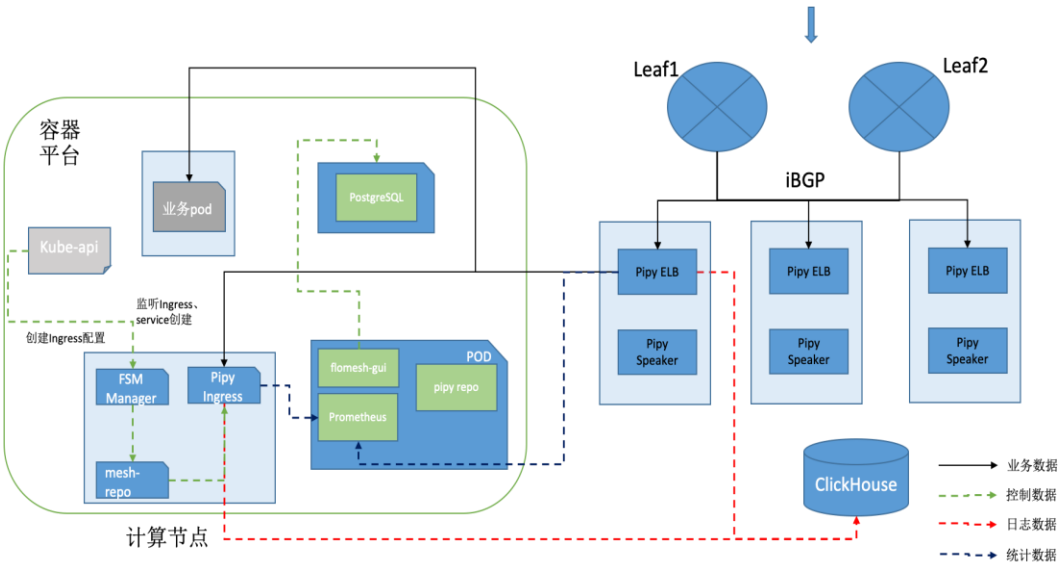


图 2-1 架构图

- 控制平面管理控制可选择部署在容器集群中，部署内包括 flomesh-gui、内置 Prometheus[注 1]。【3.6 部署 Flomesh GUI】
- 部署 Pipy-Operator。包含 Operator Manager，用于容器 Ingress 和 Sidecar 注入的控制平面配置中心——Mesh-Repo。【3.5 部署 Flomesh Ingress】
- 部署 Pipy Ingress。包含作为 Ingress 的 Pipy Ingress Controller。【3.5 部署 Flomesh Ingress】
- Pipy repo，用于 API 网关和反向代理的控制平面配置中心，选择虚拟机部署。【3.3 部署 Pipy Repo】
- 部署用于反向代理或者 API 网关的代理节点，选择虚拟机或者物理机部署。【3.7 部署反向代理】

> **注 1**：Prometheus 用于代理性能指标收集和控制平面查询，如果环境中存在全局 Prometheus，可直接采用。

2.2 依赖关系

如 1.2 总体架构 中描述，Flomesh 软件负载均衡平台需要关系型数据库和用于日志记录的非关系型数据库。

关系型数据支持：

- MySQL 5.7 / MySQL 8.0 或以上
- PostgreSQL 10 或以上，如需 PostgreSQL 高可用架构，建议使用 PostgreSQL 14 或以上版本。

日志数据库：

- ElasticSearch 7.0 或以上
- ClickHouse 22.3 或以上（推荐使用 ClickHouse）
- Splunk
- OpenSearch 1.2 或以上

关于多集群部署以及数据库高可用方案，请与我们（support@flomesh.cn）联系。

3 安装步骤

3.1 准备工作

在部署 Flomesh 软件负载均衡平台前，需要进行以下准备工作：

1. 确认环境连通性

需要确认 Kubernetes/OpenShift 集群与 MySQL/PostgreSQL 数据库和 ClickHouse 数据库所在节点的连通性。默认情况下，MySQL 数据库使用 3306 端口，PostgreSQL 数据库使用 5432 端口，ClickHouse 使用 8123 端口。

2. 创建 OpenShift/Kubernetes 集群中的项目/Namespace

1) 创建一个名为“flomesh”的项目/Namespace 用于部署控制平面；

2) 通常情况下，为保证服务质量，需要将 Ingress 部署在固定的节点上。可以通过“kubectl label node/<node name> <label>”给节点打上标签。例如：

```
node-role.kubernetes.io/router=true
```

3. 准备数据库

创建一个名为“flomesh”的数据库，创建用户“flomesh”并设置用户密码，为用户授权相应权限，设置字符集为 utf8mb4。【注】

注：为方便测试，附录中提供 MySQL 的容器化部署文件。

4. 上传镜像到镜像仓库

上传所需镜像到镜像仓库，所需镜像参考《1.3 物料清单》。参考命令：

```
# docker tag flomesh-gui:1.12-105 registry.flomesh.io/flomesh/flomesh-gui:2.0.0-82
# docker push registry.flomesh.io/flomesh/flomesh-gui:2.0.0-82
```

请根据当前环境的容器镜像地址修改上述命令。

5. **准备部署文件**

包括以下文件（见附录）：

1) 用于部署控制平面管理控制台：

- flomesh-gui-prod.yaml，

2) 用于部署独立 Pipy 代理：

- pipy-proxy-node-prod.yaml，

3) 用于部署 Kubernetes Pipy Operator：

- fsm.yaml

3.2 部署 Mysql 数据库

3.2.1 容器化部署

1. 参考附录中的文件 mysql.yml，通过修改环境变量对 Mysql 进行配置

```
kubectl apply -f mysql.yml -n flomesh
```

3.3 部署 ClickHouse 数据库

3.3.1 虚拟机/物理机 部署

1. 通过 rpm 命令安装 clickhouse 并启动：

```
$ sudo rpm -ivh clickhouse-*.rpm
$ sudo systemctl start clickhouse-server.service
```

2. 通过配置文件设置默认用户 default 密码

```
$ sudo vim /etc/clickhouse-server/user.xml
```

```
    <!-- Users and ACL. -->
    <users>
        <!-- If user name was not specified, 'default' user is used. -->
        <default>
            <!-- See also the files in users.d directory where the password can be overridden.

                 Password could be specified in plaintext or in SHA256 (in hex format).
......
            <password></password>
......
```

要给用户配置登录密码，就要在 defaul 用户下修改<passsword>标签。

password 用于设置登录密码，支持明文、SHA256 加密和 double_sha1 加密三种形式，我们可以根据实际需求，任选一种进行配置。

（1）明文密码：直接通过 password 标签定义

例如：<password>123</password>　注意：当 password 标签为空时，代表免密码登录。

（2）SHA256 加密：SHA256 加密算法的密码，需要使用<password_sha256_hex>

```
$ echo -n 123456 | openssl dgst -sha256
(stdin)=8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c
923adc6c92
```

配置完成后重启 clickhouse

```
$ sudo systemctl restart clickhouse-server.service
```

3. 导入 log.sql

```
$ clickhouse-client -mn  -password < log.sql
```

注：sql 文件为提前准备，详细内容见附录

3.3.2 容器化部署

本节介绍的容器化部署 ClickHouse 仅为测试环境使用，。

执行命令（文件参考附录中《ClickHouse 容器化部署》）：

```
kubectl apply -f log4pipy-clickhouse.yml -n flomesh
```

3.4 部署 Pipy Repo

在 k8s 上创建 namespace flomesh-gui 并通过 yaml 部署 pipy repo (pipy-repo.yml 参考附录 pipy-repo.yml)，执行命令：

```
kubectl apply -n flomesh -f pipy-repo.yml
```

3.5 部署 Flomesh GUI

1. 通过配置文件进行数据库及代理参数配置

注：详细配置文件内容见附录中的《flomesh-gui k8s 部署文件》

```
env:
- name: HOST                => 服务主机地址，一般不做更改
  value: '0.0.0.0'
- name: PORT                => 服务开启端口，一般不做更改
  value: "8080"
- name: PIPY_REPO_HOST          => pipy repo 地址
  value: 'pipy-repo.flomesh.svc'
- name: PIPY_REPO_PORT          => pipy repo 端口
  value: '6060'
- name: DATABASE_HOST           => 数据库主机地址
  value: 'mysql.flomesh.svc'
- name: DATABASE_PORT           => 数据库端口
  value: '3306'
- name: DATABASE_NAME           =>database 名称
  value: 'flomesh '
- name: DATABASE_USERNAME       => 数据库用户名称
  value: 'flomesh'
- name: DATABASE_PASSWORD       => 数据库密码
  value: 'xxxxxx'
```

```
kubectl apply -f flomesh-gui.yml -n flomesh
```

3. 访问 GUI 地址 http://<k8s-node>:<node port>，第一次访问 GUI 提示创建管理员用户

3.6 部署 Flomesh Ingress

3.6.1 HELM 部署

1. 添加 Helm repo

```
helm repo add fsm https://flomesh-io.github.io/fsm
```

2. 创建 values-overrides.yml 文件，填入 GUI 相关配置

```
fsm:
  flb:
    enabled: true
    baseUrl: http://<flomesh gui address>:<flomesh gui port>
    username: <admin-user>
    password: "<admin-password>"
    defaultCluster: ft01
    defaultAddressPool: ap01
```

参数说明如下：

- fsm.flb.enabled: 是否启用 k8s FLB
- fsm.flb.baseUrl: Flomesh-gui 地址
- fsm.flb.username: flomesh-gui 管理员用户
- fsm.flb.password: flomesh-gui 管理员密码
- fsm.flb.defaultCluster: 当前安装的 FSM FLB 对应的集群
- fsm.flb.defaultAddressPool: FSM FLB 的 loadBalancer 默认使用的地址池

3. 执行命令：

```
helm install fsm fsm/fsm --namespace flomesh --version 0.2.4 --
create-namespace -f values-override.yaml
```

values-override.yaml 为部署时的参数，例如可以更改 nodeSelector，将 Pipy Ingress Controller 固定在某个或某些节点上。默认值参考：

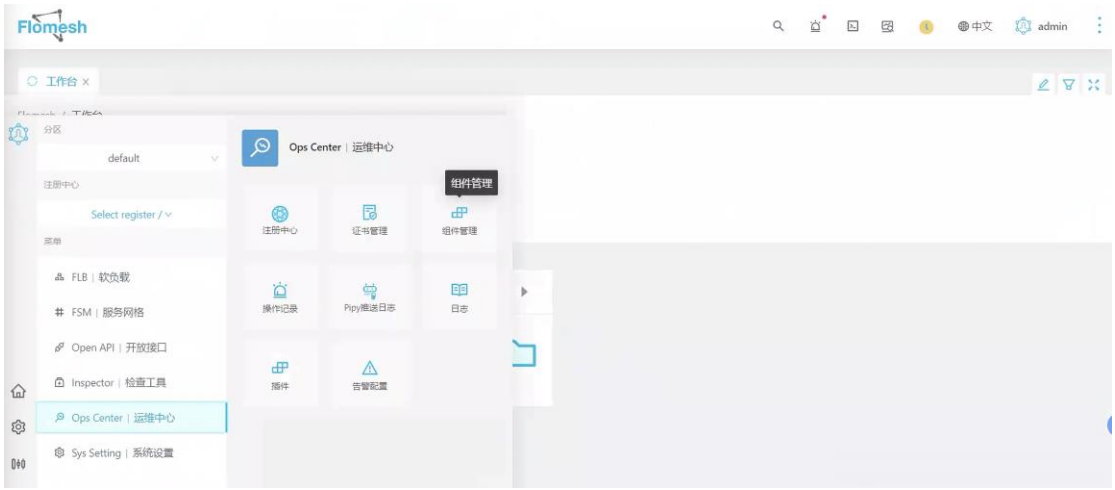
fsm/values.yaml at main • flomesh-io/fsm • GitHub

3.6.2 YAML 部署

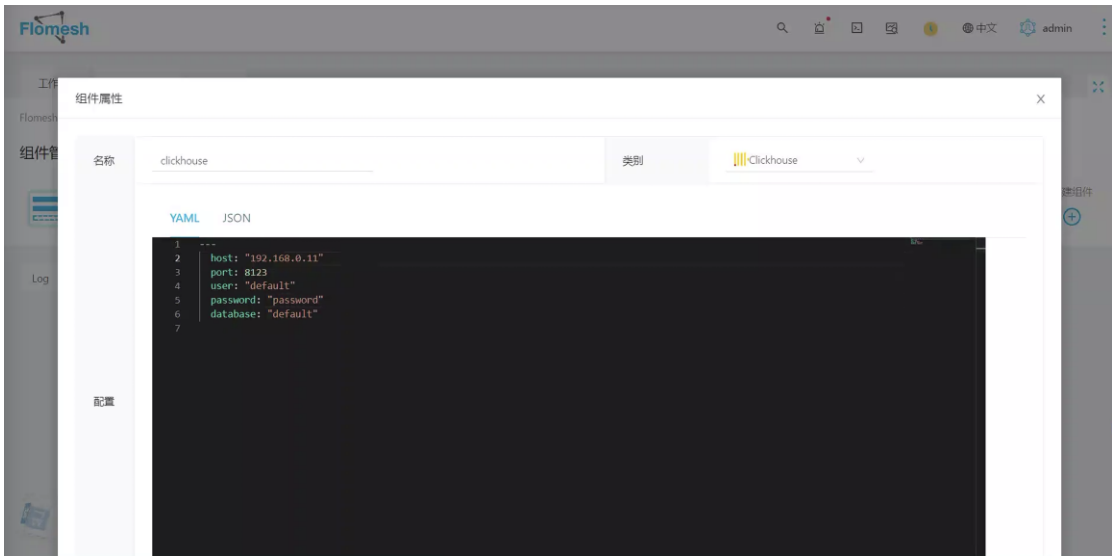
1. 通过 kubectl 命令运行 ingress yaml 配置文件及相关的依赖配置文件

注：配置文件详细内容见附录

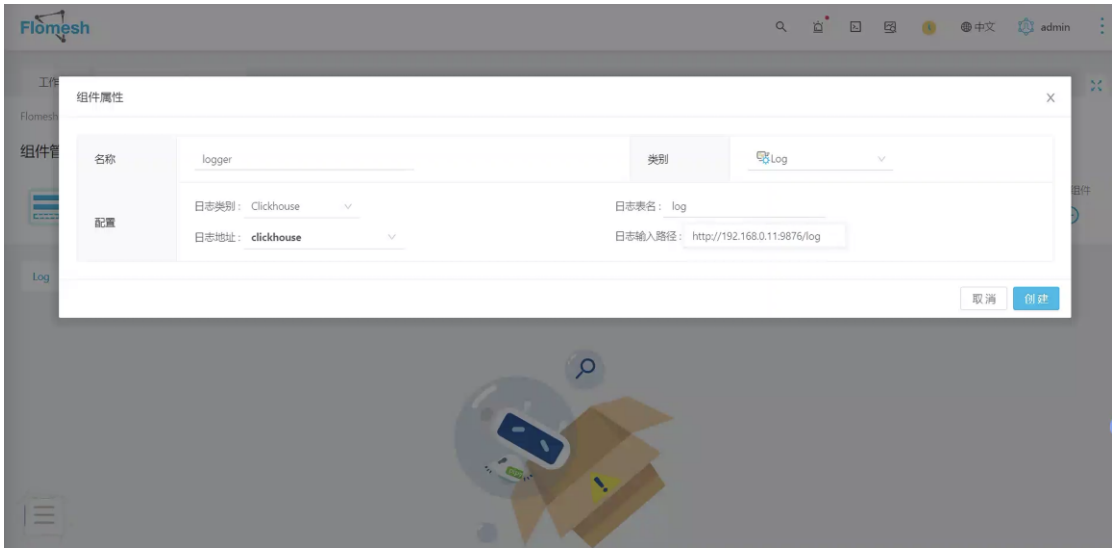
从 GitHub 下载 fsm 的部署文件（https://github.com/flomesh-io/fsm/blob/release-v0.2/deploy/fsm.yaml）



在页面右侧选择创建组件，先创建 Clickhouse 组件

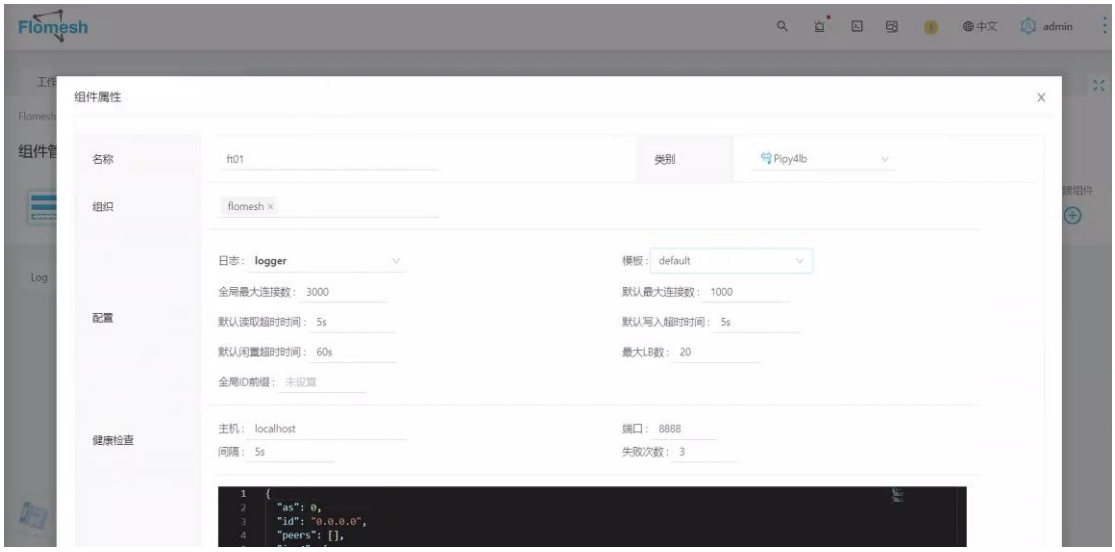


再创建 log 组件，log 组件填写 log4pipy 地址：http://<log4pipy>:9876/log



3.7.3 创建 pipy4lb 集群

在 GUI 运维中心，组件管理，创建 pipy4lb 组件。选择对应的组织以及日志组件，创建 4LB。具体配置项可参考《Flomesh 软件负载管理手册》



3.8 部署 pipy 4LB

3.8.1 下载安装 pipy

下载 pipy 并解压

```
# wget https://pipy-oss-1255617643.cos-website.ap-beijing.myqcloud.com/repo/pipy/x86_64/binary/pipy-0.90.0-185-generic_linux-x86_64.tar.gz

# tar xvf pipy-0.90.0-185-generic_linux-x86_64.tar.gz -C /
```

3.8.2 配置 pipy 4LB

创建配置文件 /etc/pipy-4lb/env.defaults

```
# mkdir /etc/pipy-4lb/

# chown nobody /etc/pipy-4lb

# vim /etc/pipy-4lb/env.defaults
```

```
# The dir of this configuration file
HOME_DIR=/etc/pipy-4lb

# The name for the pipy
PIPY_INSTANCE_NAME=pipy

# The name for the pipy
LB_ID="pipy"

# IP of the node running pipy
NODE_IP=""

# The repo address
REPO="http://localhost:6060/repo/path/to/pipy-repo"

# Set extra args in ARGS if you need more arguments for pipy
#ARGS="--tls-trusted=/etc/pipy/ssl/ca.pem --tls-cert=/etc/pipy/ssl/client.pem --tls-key=/etc/pipy/ssl/client-key.pem"
ARGS=""
```

```
[Unit]
Description = pipy
After = network.target

[Service]
EnvironmentFile=-/etc/pipy-4lb/env.defaults
ExecStartPre=/bin/bash -c 'if [ ! -f ${HOME_DIR}/uuid ]; then uuidgen > ${HOME_DIR}/uuid; fi'
ExecStart=/bin/bash -c "/usr/local/bin/pipy --reuse-port --threads=max --instance-uuid=$(cat ${HOME_DIR}/uuid) --instance-name=${PIPY_INSTANCE_NAME} $ARGS $REPO"
ExecStop=/usr/bin/kill $MAINPID
Restart=on-failure
User=nobody
AmbientCapabilities=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_ADMIN
LimitNOFILE=655360

[Install]
WantedBy = multi-user.target
```

保存后 systemd reload 并启动 pipy

```
# systemctl daemon-reload
# systemctl enable pipy-4lb
# systemctl start pipy-4lb
```

3.8.3 配置 node_exporter

下载 node_exporter 并解压到系统目录下

```
$ wget https://pipy-oss-1255617643.cos.ap-beijing.myqcloud.com/repo/infra/node_exporter_1.5.0_linux_amd64.tar.gz

$ tar xvf https://pipy-oss-1255617643.cos.ap-beijing.myqcloud.com/repo/infra/node_exporter_1.5.0_linux_amd64.tar.gz

$ sudo mv node_exporter_1.5.0_linux_amd64/node_exporter /usr/local/bin/
```

配置 systemd /etc/systemd/system/node_exporter.service 并启动 node_exporter

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable node_exporter
$ sudo systemctl start node_exporter
```

3.8.4 配置 prometheus

prometheus 需要配置两个抓取目标，一个是 pipy repo 聚合的 metrics，另一个是 LB 节点的 node_exporter，LB 的 node_exporter 可以通过 GUI 的 http service discovery 自动发现

```
scrape_configs:
  - job_name: "pipy-sd"
    http_sd_configs:
      - url: http://<flomesh gui addr>:<flomesh gui port>/api/sd

  - job_name: "pipy-repo"
    static_configs:
      - targets:
        - <pipy repo addr>:<pipy repo port>
```

4 安全加固

默认情况下，控制台、Pipy 代理与 Pipy Repo 的交互使用 HTTP 协议。为了提升传输的安全性与身份认证，可以启用 HTTPS 协议，并启用 Pipy Repo 证书认证。Pipi repo 证书认证使用的参数如下所示：

```
$ pipy -h
Usage: pipy [options] [<filename or URL>]

Options:
   --admin-port=<[[ip]:]port>                Enable administration service on the
specified port
   --admin-tls-cert=<filename>               Administration service certificate
   --admin-tls-key=<filename>                Administration service private key
   --admin-tls-trusted=<filename>            Client certificate(s) trusted by
administration service
   --tls-cert=<filename>                     Client certificate in communication
to administration service
   --tls-key=<filename>                      Client private key in
communication to administration service
   --tls-trusted=<filename>                  Administration service certificate(s)
trusted by client
```

其中，`--admin-*` 选项为 Pipy repo 侧使用的参数，`--tls-*`选项为 Pipy 代理侧使用的参数。

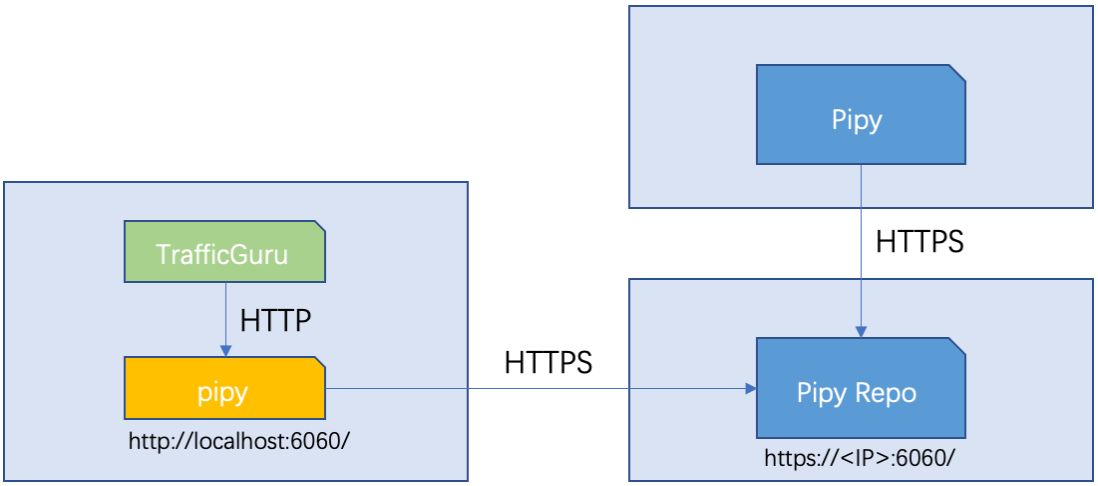


图 4-1 HTTPS 访问模式

4.1 创建 TLS 证书

4.1.1 创建 CA 证书

```
# openssl req -newkey rsa:2048 -nodes -keyout ca.key -x509 -days 365 -out ca.crt
```

4.1.2 创建服务端证书

```
# openssl req -newkey rsa:2048 -nodes -keyout server.key -out server.csr

# openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365
```

```
# kubectl create secret generic -n flomesh-gui pipy-repo-tls-certs --from-file=ca.crt=ca.crt --from-file=server.crt=server.crt --from-file=server.key=server.key
```

修改 pipy-repo deployment，设置 volumes 和 volumeMounts 挂载 secrets 并且 pipy 启动参数加上 --admin-tls-* 相关参数，例如：

4.2.2 GUI 使用 TLS 访问 pipy repo

将 CA 证书公钥，客户端证书公私钥作为 secrets 添加到 k8s 中

```
# kubectl create secret generic -n flomesh-gui pipy-worker-tls-certs --from-
file=ca.crt=ca.crt --from-file=client.crt=server.crt --from-file=client.key=client.key
```

修改 flomesh-gui deployment，增加本地 pipy sidecar，同时 pipy sidecar 通过 TLS 连接 pipy repo，GUI 通过本地的 sidecar (PIPY_REPO_HOST)访问 repo

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: flomesh-gui
  name: flomesh-gui
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flomesh-gui
  template:
    metadata:
      labels:
        app: flomesh-gui
    spec:
      volumes:
      - name: pipy-worker-tls-certs
        secret:
          defaultMode: 420
          secretName: pipy-worker-tls-certs
      containers:
      - image: flomesh/pipy-repo:0.90.0-185
        name: pipy-repo
        args:
        - pipy
        - --tls-trusted=/ssl/ca.crt
        - --tls-cert=/ssl/client.crt
        - --tls-key=/ssl/client.key
        - https://pipy-repo.flomesh-gui.svc.cluster.local:6060
        volumeMounts:
        - mountPath: /ssl
          name: pipy-worker-tls-certs
          readOnly: true
      - image: traffic-guru:1.0.0
        name: flomesh-gui
        ports:
        - name: flomesh-gui-web
          containerPort: 8080
          hostPort: 8080
        env:
        - name: HOST
          value: '0.0.0.0'
        - name: PORT
          value: "8080"
        - name: PIPY_REPO_HOST
          value: '127.0.0.1'
        - name: PIPY_REPO_PORT
          value: '6060'
        - name: DATABASE_HOST
          value: 'mysql-pipy.default.svc'
        - name: DATABASE_PORT
          value: '3306'
        - name: DATABASE_NAME
          value: 'flomesh_gui'
        - name: DATABASE_USERNAME
          value: 'flomesh'
```

4.3 非容器模式

4.3.1 pipy 4LB 使用 TLS 订阅 pipy repo

将 CA 证书公钥，client 证书公私钥拷贝到 pipy 节点，放置在 /etc/pipy-4lb/ssl 下

```
# mkdir /etc/pipy-4lb/ssl
# cp ca.crt client.crt client.key /etc/pipy-4lb/ssl
```

修改 /etc/pipy-4lb/env.defaults 的 ARGS 参数，加入 --tls-* 相关参数，同时修改 REPO 地址，使用 https 协议访问

```
...
REPO="https://<IP>:<Port>/repo/path/to/repo/"

ARGS="--tls-trusted=/etc/pipy-4lb/ssl/ca.crt --tls-cert=/etc/pipy-4lb/ssl/client.crt --tls-key=/etc/pipy-4lb/ssl/client.key"
```

重启 pipy

```
# systemctl restart pipy-4lb
```

4.3.2 prometheus 使用 TLS 抓取 pipy repo metrics

将 CA 证书公钥，client 证书公私钥拷贝到 prometheus 上，配置 prometheus 使用 TLS 访问方式抓取 pipy repo metrics

```
scrape_configs:
  - job_name: "pipy-repo"
    scheme: https
    tls_config:
      insecure_skip_verify: true
      ca_file: /etc/prometheus/ssl/ca.pem
      cert_file: /etc/prometheus/ssl/client.pem
      key_file: /etc/prometheus/ssl/client.key
    static_configs:
      - targets:
        - <pipy repo addr>:<pipy repo port>
```

4.4 调试便利性

在生产环境或者资源受限的情况下，Pipy Repo 会进行轻量化部署，并不会开启 GUI（图形用户界面）；或者在开启 TLS 的情况下，无法访问 GUI。

如果此时浏览器打开 `https://localhost:6060`，无法访问正常访问：

```
2021-12-07 14:19:30 [ERR] [tls] Handshake failed (error = 1)
2021-12-07 14:19:30 [ERR] [tls] error:14094416:SSL
routines:ssl3_read_bytes:sslv3 alert certificate unknown
```

此时就需要使用 Pipy Repo 的客户端模式了：Pipy 启动的时候可以指定一个远程 Repo 的地址（注意不是 codebase 的地址），将远程 Repo 作为服务端，本地只提供 Repo 的 GUI。

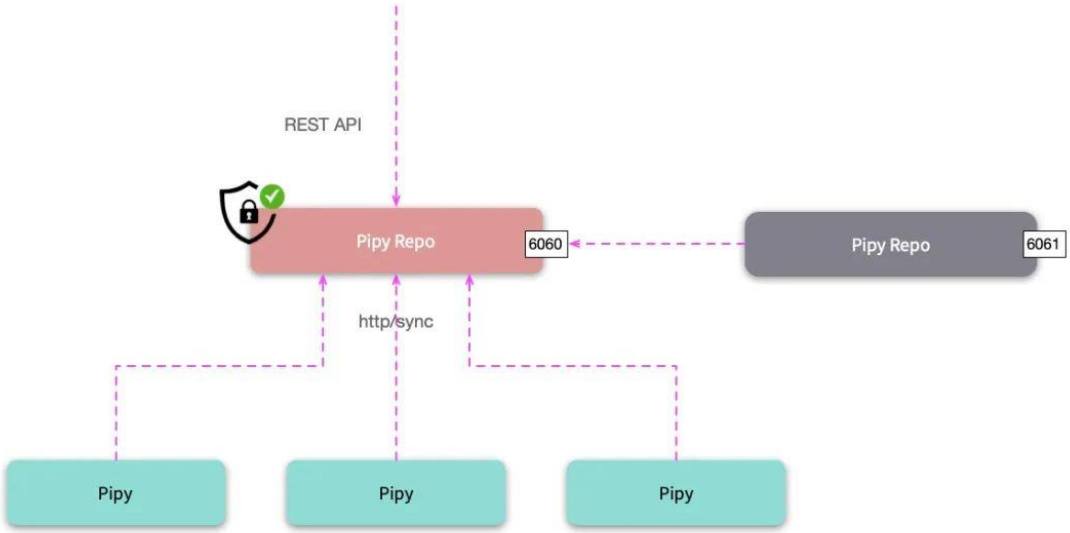


图 4-2 Pipy repo 代理

比如，连接到 `https://localhost:6060/` 。需要根据服务端的情况，使用客户端证书和秘钥启动。请注意，如果两个 pipy repo 位于同一台主机上，避免端口冲突需要指定另外一个端口。

```
$ pipy https://localhost:6060/ --tls-cert=client.cert --tls-key=client.key --admin-port=6061
```

此时在浏览器中打开 `https://localhost:6061/`，可以正常访问。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mysql
---
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - port: 3306
    targetPort: 3306
    name: client
    appProtocol: tcp
  selector:
    app: mysql
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "local-path"
      resources:
        requests:
          storage: 5Gi
  template:
    metadata:
      labels:
        app: mysql
```

ClickHouse 数据库表 - log.sql

```
CREATE TABLE default.log
(
    `rid` UInt64 DEFAULT JSONExtractInt(message,
'rid'),
    `sid` UInt64 DEFAULT JSONExtractInt(message,
'sid'),
    `iid` String DEFAULT JSONExtractString(message,
'iid'),
    `dir` String DEFAULT JSONExtractString(message,
'dir'),
    `proto` String DEFAULT JSONExtractString(message,
'proto'),
    `req` String DEFAULT JSONExtractRaw(message,
'req'),
    `req.id` String DEFAULT JSONExtractString(req,
'id'),
    `req.protocol` String DEFAULT JSONExtractString(req,
'protocol'),
    `req.version` String DEFAULT JSONExtractString(req,
'version'),
    `req.service.version` String DEFAULT JSONExtractString(req,
'service.version'),
    `req.method.name` String DEFAULT JSONExtractString(req,
'method.name'),
    `req.method.type` String DEFAULT JSONExtractString(req,
'method.type'),
    `req.method` String DEFAULT JSONExtractString(req,
'method'),
    `req.path` String DEFAULT JSONExtractString(req,
'path'),
    `req.headers` String DEFAULT JSONExtractRaw(req,
'headers'),
    `req.service.name` String DEFAULT JSONExtractString(req.headers,
'serviceidentity'),
    `req.body` String DEFAULT JSONExtractString(req,
'body'),
    `res` String DEFAULT JSONExtractRaw(message,
'res'),
    `res.protocol` String DEFAULT JSONExtractString(res,
'protocol'),
    `res.type` Int32 DEFAULT JSONExtractInt(res,
'type'),
    `res.value` String DEFAULT JSONExtractString(res,
'value'),
    `res.status` UInt32 DEFAULT JSONExtractInt(res,
'status'),
    `res.statusText` String DEFAULT JSONExtractString(res,
'statusText'),
    `res.headers` String DEFAULT JSONExtractRaw(res,
'headers'),
    `res.body` String DEFAULT JSONExtractString(res,
'body'),
    `reqTime` UInt64 DEFAULT JSONExtractInt(message,
'reqTime'),
    `resTime` UInt64 DEFAULT JSONExtractInt(message,
'resTime'),
    `reqSize` UInt64 DEFAULT JSONExtractInt(message,
'reqSize'),
    `resSize` UInt64 DEFAULT JSONExtractInt(message,
'resSize'),
    `localAddr` String DEFAULT JSONExtractString(message,
'localAddr'),
```

log4pipy k8s 部署文件 - log4pipy.yml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: log4pipy
  name: log4pipy
spec:
  ports:
  - name: app
    nodePort: 30876
    port: 9876
    protocol: TCP
    targetPort: 9876
  selector:
    app: log4pipy
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: log4pipy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: log4pipy
  template:
    metadata:
      labels:
        app: log4pipy
    spec:
      containers:
      - name: log4pipy
        image: flomesh/log4pipy-clickhouse:v0.0.1
        imagePullPolicy: IfNotPresent
        env:
        - name: SERVER_LISTENING
          value: "0.0.0.0:9876"
        - name: DB_SERVER
          value: "192.168.0.11:8123"
        - name: DB_USER
```

flomesh-gui k8s 部署文件 - flomesh-gui.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: flomesh-gui
  name: flomesh-gui
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flomesh-gui
  template:
    metadata:
      labels:
        app: flomesh-gui
    spec:
      containers:
      - image: flomesh/traffic-guru:0.1.0-12
        name: flomesh-gui
        ports:
        - name: flomesh-gui-web
          containerPort: 8080
          hostPort: 8080
        env:
        - name: HOST
          value: '0.0.0.0'
        - name: PORT
          value: "8080"
        - name: PIPY_REPO_HOST
          value: 'pipy-repo.flomesh.svc'
        - name: PIPY_REPO_PORT
          value: '6060'
        - name: DATABASE_HOST
          value: 'mysql.flomesh.svc'
        - name: DATABASE_PORT
          value: '3306'
        - name: DATABASE_NAME
          value: 'flomesh'
        - name: DATABASE_USERNAME
          value: 'flomesh'
        - name: DATABASE_PASSWORD
```

pipy-repo.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: pipy-repo
  name: pipy-repo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pipy-repo
  template:
    metadata:
      labels:
        app: pipy-repo
    spec:
      containers:
      - image: flomesh/pipy-repo:0.90.0-185
        name: pipy-repo
        command:
        - pipy
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: pipy-repo
  name: pipy-repo
spec:
  ports:
  - port: 6060
    protocol: TCP
    targetPort: 6060
    nodePort: 30060
  selector:
    app: pipy-repo
  type: NodePort
```

Flomesh ingress 部署文件 - fsm.yaml

https://github.com/flomesh-io/fsm/blob/release-v0.2/deploy/fsm.yaml

ClickHouse 容器化部署 -- clickhouse.yml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  creationTimestamp: null
  labels:
    app: clickhouse
  name: clickhouse
spec:
  serviceName: clickhouse
  selector:
    matchLabels:
      app: clickhouse
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: clickhouse
    spec:
      volumes:
      - name: clickhouse-pvc
        persistentVolumeClaim:
          claimName: clickhouse-pvc
      containers:
      - image: clickhouse/clickhouse-server:22.3
        name: clickhouse
        volumeMounts:
        - mountPath: /var/lib/clickhouse/
          name: clickhouse-pvc
        resources: {}
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: clickhouse
  name: clickhouse
spec:
  ports:
  - name: 8123-8123
    port: 8123
```

Flomesh Ingress RBAC 权限及 CRD 列表

Group	Resources	Verbs	Comment
admissionregistration.k8s.io	mutatingwebhookconfigurations	get list watch create update patch delete	
	validatingwebhookconfigurations		
	mo		
apps	daemonsets	get list watch create update patch delete	纳管集群-创建
	deployments		
	replicasets		
networking.k8s.io	ingresses	get list watch	代码包含，可开
	ingressclasses		
apiextensions.k8s.io	customresourcedefinitions	get list watch create update patch delete	
coordination.k8s.io	leases	get list watch create update patch delete	选主
batch	jobs	get list watch	helm
core	endpoints	get list	sidecar 注入，监
	pods		

	services	watch	
	secrets	create	证书
	configmaps	update	配置
	volumes	patch	
		delete	
		deletecollection	
core	namespaces	get	
		list	
		watch	
core	events	get	
		list	
		watch	
		create	
		update	
		patch	
flomesh.io	clusters	get	多集群纳管
	proxyprofiles	list	注入
	serviceexports	watch	服务发布
	serviceimports	create	服务订阅
		update	
		patch	
		delete	
flomesh.io	clusters/finalizers	update	自定义资源
	proxyprofiles/finalizers		
flomesh.io	clusters/status	get	
	proxyprofiles/status	patch	
		update	
config.openservicemesh.io	meshconfigs	get	配置 证书
	meshrootcertificates	list	
		watch	
		create	
		update	
		patch	
		delete	
config.openservicemesh.io	meshrootcertificates/status	update	
split.smi-spec.io	trafficsplits	get	流量调度
		list	
		watch	
access.smi-spec.io	traffictargets	get	

		list watch
specs.smi-spec.io	httproutegroups	get list watch
	tcproutes	
policy.openservicemesh.io	egresses	get list watch
	egressgateways	
	ingressbackends	
	accesscontrols	
	retries	
	upstreamtrafficsettings	
policy.openservicemesh.io	ingressbackends/status	update
	accesscontrols/status	
	upstreamtrafficsettings/status	