运维服务

Peer 和 orderer 管理一个 HTTP 服务器，该服务器提供 RESTful “运维” API。此API与Fabric网络服务无关，旨在供运维人员使用，而非网络管理员或“用户”。

该 API 提供了以下功能：

- 日志级别管理
- 健康检查
- （当API被配置时）运维指标数据的Prometheus目标

配置运维服务

运维服务需要两层基本的配置

- 要监听的 **地址** 和 **端口** 。
- 用于身份验证和加密的 **TLS（传输层安全协议）证书** 和 **密钥** 。

注意，**这些证书必须由另外一个专门的CA（证书授权中心）来生成**，不能由已为某通道上的组织生成过证书的CA来生成。

节点

对每个 peer 来说，可在 `core.yaml` 部分的 `operations` 中配置运维服务器：

```
operations:
  # host and port for the operations server
  listenAddress: 127.0.0.1:9443

  # TLS configuration for the operations endpoint
  tls:
    # TLS enabled
    enabled: true

    # path to PEM encoded server certificate for the operations server
    cert:
      file: tls/server.crt

    # path to PEM encoded server key for the operations server
    key:
      file: tls/server.key

    # most operations service endpoints require client authentication when TLS
    # is enabled. clientAuthRequired requires client certificate authentication
    # at the TLS layer to access all resources.
    clientAuthRequired: false

    # paths to PEM encoded ca certificates to trust for client authentication
    clientRootCAs:
      files: []
```

`listenAddress` 键定义了运维服务器将执行监听的主机和端口。如果服务器要监听所有地址，则可以忽略主机部分。

`tls` 部分用于指明是否为运维服务启用了TLS，还指明了该服务的证书和私钥的位置以及客户端身份验证应该信任的证书颁发机构根证书的位置。当 `enabled` 是真实的，多数运维服务端点会要要求客户验证，因此必须设定 `clientRootCAs.files` 。当 `clientAuthRequired` 是 `true` ，TLS层将需要客户在每次请求时都提供一份证书以供验证。参考下面运维安全部分的内容来获取更多信息。

排序节点

对每个orderer来说，运行服务器都可配置在的 `orderer.yaml` 的 *Operations* 部分。

```
Operations:
  # host and port for the operations server
  ListenAddress: 127.0.0.1:8443

  # TLS configuration for the operations endpoint
  TLS:
    # TLS enabled
    Enabled: true

    # PrivateKey: PEM-encoded tls key for the operations endpoint
    PrivateKey: tls/server.key

    # Certificate governs the file location of the server TLS certificate.
    Certificate: tls/server.crt

    # Paths to PEM encoded ca certificates to trust for client authentication
    ClientRootCAs: []

    # Most operations service endpoints require client authentication when TLS
    # is enabled. ClientAuthRequired requires client certificate authentication
    # at the TLS layer to access all resources.
    ClientAuthRequired: false
```

`ListenAddress` 键定义了运维服务器将监听的主机和端口。如果服务器要监听所有地址，则可忽略主机部分。

`TLS` 部分用于指明是否为运维服务启用了TLS，还指明了该服务的证书和私钥的位置以及客户端身份验证应该信任的证书授权中心根证书的位置。当 `Enabled` 是真实的，多数运维服务端点会要要求客户验证，因此必须设定 `RootCAs` 。当 `clientAuthRequired` 是 `true` ，TLS层将需要客户在每次请求时都提供一份证书以供验证。参考下面运维安全部分的内容来获取更多信息。

运维安全

由于运维服务专注于运维，与 Fabric 网络无关，因此它不是用MSP（成员服务提供者）来进行访问控制，而是完全依赖于具有客户端证书身份验证功能的双向 TLS。

禁用TLS后，授权将被绕过，这样一来，任何能连接到运行端点的客户端都可以使用API（应用程序编程接口）。

启用TLS后，除非下面另有说明，否则必须提供有效的客户端证书才能访问所有资源。

若同时启用了 clientAuthRequired 时，无论访问的是什么资源，TLS 层都将需要有效的客户端证书。

日志级别管理

运维服务提供了 `/logspec` 资源，运维人员可用该资源来管理peer或orderer的活跃日志记录规范。该资源是常规的REST资源，支持 `GET` 和 `PUT` 请求。

当运维服务接收到 `GET /logspec` 请求时，它将使用包含当前日志记录规范的 JSON 有效负载进行响应：

```
{"spec":"info"}
```

当运维服务接收到 `PUT /logspec` 请求时，它将把 body 读取为 JASON 有效负载。有效负载必须包含名为 `spec` 的单个属性。

```
{"spec":"chaincode=debug:info"}
```

如果规范成功激活，服务将回复 `204 "No Content"` 。如果出现错误，服务将回复 `400 "Bad Request"` 以及一个错误有效负载：

```
{"error":"error message"}
```

健康检查

运维服务提供了 `/healthz` 资源，运维人员可用该资源来确定 peer 和 orderer 的活跃度及健康状况。该资源是支持GET请求的常规REST资源。它的实现旨在与 Kubernetes 使用的活跃度探针模型兼容，不过还可以在其他场景中进行。

当运维服务收到 `GET/healthz` 请求，它将调用所有已注册的运行状况检查程序来执行该流程。当所有运行状况检查程序都成功返回时，运维服务将以 `200 "OK"` 和 JSON body 进行回应：

```
{
  "status": "OK",
  "time": "2009-11-10T23:00:00Z"
}
```

如果运行状况检查程序中的一个或多个返回错误时，运行服务将响应 `503 "Service Unavailable"` 和一个包含未成功的运行状况检查程序的JASON body：

```
{
  "status": "Service Unavailable",
  "time": "2009-11-10T23:00:00Z",
  "failed_checks": [
    {
      "component": "docker",
      "reason": "failed to connect to Docker daemon: invalid endpoint"
    }
  ]
}
```

在当前版本中，唯一注册的运行状况检查程序是针对Docker的。后期版本将增加额外的运行状况检查程序。

当启用TLS时，不需要提供有效的客户端证书就可以使用该服务，除非 `clientAuthRequired` 被设置为 `true` 。

指标数据（Metrics）

Fabric的peer和orderer的某些组件获取metrics，这些metrics可帮助深入了解系统行为。通过这些信息，运维人员和管理人员可以更好地理解系统随着时间的推移是如何运行的。

配置 Metrics

Fabric提供了两种获取metrics的方法：一种是基于Prometheus的 **拉式** 模型，另一种是基于StatsD的 **推式** 模型。

Prometheus

典型的Prometheus部署通过从已检测目标公开的HTTP端点请求指标来获取指标数据。由于Prometheus负责请求metrics，因此它被看成是一种拉式系统。

当配置完成，Fabric的peer或orderer将在运维服务中展示 /metrics 资源。

节点

通过在 `core.yaml` 部分的 `metrics` 中将metrics获取方式设置为prometheus ， 可对peer进行配置，从而获取 `/metrics` 端点，以供Prometheus使用。

```
metrics:
  provider: prometheus
```

排序节点

通过在 `orderer.yaml` 部分的 `Metrics` 中将 metrics 获取方式设置为bprometheus ， 可对orderer进行配置，从而获取 `/metrics` 端点，以供 Prometheus 使用。

```
Metrics:
  Provider: prometheus
```

StatsD

StatsD是一个简单的统计聚合守护程序。Metrics被发送到 `statsd` 守护程序进行收集、汇总并推送至后端以进行可视化和警报。由于该模型需要辅助型流程来将metrics数据发送至StatsD,因此它被视为一种推式系统。

节点

通过在 `core.yaml` 部分的 `metrics` 中将metrics获取方式设置为 `statsd`，可对节点进行配置，从而使metrics被发送至StatsD. `statsd` 子节必须配置有StatsD守护程序的地址、要使用的网络类型(`tcp` or `udp`)以及发送metrics的频率。通过指定一个可选 `prefix` ，可帮助区分metrics的来源（例如，区分来自不同peer的metrics），这些metrics将被添加到所有已生成的metrics中。

```
metrics:
  provider: statsd
  statsd:
    network: udp
    address: 127.0.0.1:8125
    writeInterval: 10s
    prefix: peer-0
```

排序节点

通过在 `orderer.yaml` 部分的 `Metrics` 中将metrics获取方式设置为 `statsd` ，可对排序节点进行配置，使得metrics被发送至StatsD. `Statsd` 子节必须配置有StatsD守护程序的地址、要使用的网络类型(`tcp` or `udp`)以及发送metrics的频率。通过指定一个可选 `prefix` ，可帮助区分metrics的来源。

```
Metrics:
    Provider: statsd
    Statsd:
      Network: udp
      Address: 127.0.0.1:8125
      WriteInterval: 30s
      Prefix: org-orderer
```

想了解已生成的不同metrics，请参考 Metrics Reference

版本

The orderer and peer both expose a `/version` endpoint. This endpoint serves a JSON document containing the orderer or peer version and the commit SHA on which the release was created.