将链码作为外部服务

Fabric v2.0支持将链码在Faric外部部署与执行以方便用户独立管理每个节点的代码运行环境。这有助于Kubernetes等Fabric云上部署链码。不再需要在每个节点构建和运行，链码现在可以作为一个服务运行，其生命周期在Fabric之外进行管理。此功能利用了Fabric v2.0外部构建器和启动器功能，使操作员能够通过程序扩展对等方来构建、启动和发现链码。在阅读本主题之前，您应该熟悉外部构建器和启动器 内容。

在外部构建器可用之前，链码包内容要求是一组特定语言的源代码文件，可以作为链码二进制文件构建和启动。新的外部构建和启动程序功能现在允许用户有选择地定制构建过程。为了将链码作为外部服务运行，构建过程允许您指定正在运行链码的服务器的端点信息。因此，该包只包含外部运行的链码服务器端点信息和用于安全连接的TLS构件。TLS是可选的，但强烈建议用于除简单测试环境以外的所有环境。

本主题的其余部分介绍如何将链码配置为外部服务：

- 打包链码
- 设置一个节点来执行外部链码
- 外部构建器和启动器示例脚本
- 编写作为外部服务的链码
- 部署链码
- 将链码作为外部服务执行

注意: 这是一个需要定制封装peer镜像的高级功能。例如，接下来的示例需要用到的`jq`和`bash`并不包含在当前的`fabric-peer`官方镜像中。

打包链码

在Fabricv2.0的链码生命周期中，链码是被打包为`.tar.gz`的格式并安装的。下述`myccpackage.tgz`文件将展示所需要的结构。

```
$ tar xvfz myccpackage.tgz
metadata.json
code.tar.gz
```

链码包应向外部构建器和启动器进程提供两个信息

- 标记链码是否为外部服务。`bin/detect`章节描述了使用`metadata.json`的要求。
- 在`connection.json`中提供链码端点信息并放置在发布目录中。`bin/run`章节描述了`connection.json`文件。

收集上述信息可以有很多灵活的方法。[外部构建器和运行器示例脚本]示例脚本中展示了收集这些信息的简单方法。 作为一个灵活性的例子，考虑打包couchdb索引文件（参考将索引添加至你的链码文件夹)。下列示例脚本展示了打包文件到 code.tar.gz的方法。

```
tar cfz code.tar.gz connection.json metadata
tar cfz $1-pkg.tgz metadata.json code.tar.gz
```

设置一个节点来执行外部链码

在本章节中文名将介绍所需的配置

- 检测链码是否被标记为外部链码服务
- 在发布目录中创建`connection.json`文件

修改 peer core.yaml文件以包含externalBuilder（外部构建器）

假设peer上的脚本被如下所示放置在`bin`文件夹下

```
    <fully qualified path on the peer's env>
    └── bin
        ├── build
        ├── detect
        └── release
```

修改peer的`core.yaml`中`chaincode`段落以包含`externalBuilders`（外部构建器）配置元素：

```
externalBuilders:
     - name: myexternal
       path: <fully qualified path on the peer's env>
```

外部构建器和启动器示例脚本

为了帮助理解外部服务链码需要包含的每个脚本，本章节包含`bin/detect` `bin/build`，`bin/release`，和`bin/run`脚本。

注意: 本示例使用了`jq`命令来分析json。你可以运行`jq --version`来检查你是否已安装。如果没有，则安装`jq`或适当调整脚本。

bin/detect

`bin/detect script`负责确定是否应使用buildpack来生成链码包并启动它。对于作为外部服务的链码，示例脚本在`metadata.json`文件中设置`type`属性为`external`：

```
{"path":"","type":"external","label":"mycc"}
```

Peer调用detect需要两个参数:

```
bin/detect CHAINCODE_SOURCE_DIR CHAINCODE_METADATA_DIR
```

`bin/detect` 示例脚本包含：

```
#!/bin/bash

set -euo pipefail

METADIR=$2
#check if the "type" field is set to "external"
if [ "$(jq -r .type "$METADIR/metadata.json")" == "external" ]; then
    exit 0
fi

exit 1
```

bin/build

作为一个外部服务的链码，示例构建脚本假设链码包 `code.tar.gz` 的文件内的 `connection.json` 仅简单复制 `BUILD_OUTPUT_DIR` 。节点需要三个参数调用构建脚本:

```
bin/build CHAINCODE_SOURCE_DIR CHAINCODE_METADATA_DIR BUILD_OUTPUT_DIR
```

`bin/build` 示例脚本包含：

```
#!/bin/bash

set -euo pipefail

SOURCE=$1
OUTPUT=$3

#external chaincodes expect connection.json file in the chaincode package
if [ ! -f "$SOURCE/connection.json" ]; then
    >&2 echo "$SOURCE/connection.json not found"
    exit 1
fi

#simply copy the endpoint information to specified output location
cp $SOURCE/connection.json $OUTPUT/connection.json

if [ -d "$SOURCE/metadata" ]; then
    cp -a $SOURCE/metadata $OUTPUT/metadata
fi

exit 0
```

bin/release

作为外部服务的链码， `bin/release` 脚本负责将 `connection.json` 放置在 `RELEASE_OUTPUT_DIR` 以提供给peer。 `connection.json` 文件包含下列JSON结构

- **address** - 可供peer访问的链码服务端点. 必须写为 ":" 格式.
- **dial_timeout** - 等待连接完成的间隔。指定为带有时间单位的字符串（例如，“10s”、“500ms”、“1m”）。如果未指定，默认值为“3s”。
- **tls_required** - true 或 false. 如果为 false, “client_auth_required”, “client_key”, “client_cert” 和 “root_cert” 则不需要填写. 默认 “true”.

- **client_auth_required** - 如果为 true, “client_key” and “client_cert” 需要填写. 默认 false. 它将忽略 tls_required 为 false.
- **client_key** - 客户端的私钥PEM 加密字串。
- **client_cert** - 客户端的PEM加密证书字串。
- **root_cert** - 服务器（peer）根节点证书的PEM加密字串。

例如：

```
{
  "address": "your.chaincode.host.com:9999",
  "dial_timeout": "10s",
  "tls_required": "true",
  "client_auth_required": "true",
  "client_key": "-----BEGIN EC PRIVATE KEY----- ... -----END EC PRIVATE KEY-----",
  "client_cert": "-----BEGIN CERTIFICATE----- ... -----END CERTIFICATE-----",
  "root_cert": "-----BEGIN CERTIFICATE---- ... -----END CERTIFICATE-----"
}
```

如 `bin/build` 章节提示的，本示例假设链码包的 `connection.json` 文件直接复制到 `BUILD_OUTPUT_DIR` 。节点使用两个参数调用发布脚本：

```
bin/release BUILD_OUTPUT_DIR RELEASE_OUTPUT_DIR
```

`bin/release` 示例脚本包含：

```
#!/bin/bash

set -euo pipefail

BLD="$1"
RELEASE="$2"

if [ -d "$BLD/metadata" ]; then
   cp -a "$BLD/metadata/"* "$RELEASE/"
fi

#external chaincodes expect artifacts to be placed under "$RELEASE"/chaincode/server
if [ -f $BLD/connection.json ]; then
   mkdir -p "$RELEASE"/chaincode/server
   cp $BLD/connection.json "$RELEASE"/chaincode/server

   #if tls_required is true, copy TLS files (using above example, the fully qualified path for th

   exit 0
fi

exit 1
```

编写作为外部服务的链码

当前，外部服务模式的链码仅支持 GO chaincode shim.在Fabric v2.0中，GO shim API添加了 `ChaincodeServer` 类型使开发者可以使用它来创建一个链码服务。 `Invoke` and `Query` APIs 不受影响。开发者应将其写入 `shim.ChaincodeServer` API,然后构建链码并运行在选择的外部环境中。下面是一个简单的链码程序示例来说明这个模式:

```
package main

import (
        "fmt"

        "github.com/hyperledger/fabric-chaincode-go/shim"
        pb "github.com/hyperledger/fabric-protos-go/peer"
)

// SimpleChaincode example simple Chaincode implementation
type SimpleChaincode struct {
}

func (s *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
        // init code
}

func (s *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
        // invoke code
}

//NOTE - parameters such as ccid and endpoint information are hard coded here for illustration. T
func main() {
       //The ccid is assigned to the chaincode on install (using the “peer lifecycle chaincode in
        ccid := "mycc:fcbf8724572d42e859a7dd9a7cd8e2efb84058292017df6e3d89178b64e6c831"

        server := &shim.ChaincodeServer{
                        CCID: ccid,
                        Address: "myhost:9999"
                        CC: new(SimpleChaincode),
                        TLSProps: shim.TLSProperties{
                                Disabled: true,
                        },
                }
        err := server.Start()
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
        }
}
```

使用`shim.ChaincodeServer`是将链码作为外部服务的关键。新的shim API `shim.ChaincodeServer`服务配置参数的作用如下：

- **CCID** (string)- CCID 应匹配peer节点上链码的包名.这是`CCID`关联链码并使用`peer lifecycle chaincode install <package>`命令获取返回值。这可以在安装后使用`peer lifecycle chaincode queryinstalled`命令获得。
- **Address** (string) - 是链码服务的监听地址
- **CC** (Chaincode) - CC 是初始化（Init）和调用（Invoke）的链码。
- **TLSProps** (TLSProperties) - TLSProps 是发送给链码服务的TLS属性
- **KaOpts** (keepalive.ServerParameters) - KaOpts keepalive 属性, 如果为空，则提供合理的默认值

然后构建适合您的GO环境的链码

部署链码

当GO链码可以被部署时，你可以如打包链码 章节描述的一样打包，并按照Fabric 链码生命周期 中说明的进行部署。

将链码作为外部服务运行

如编写外部服务链码章节描述的创建链码。在您选择的环境中运行构建可执行文件，如Kubernetes或直接在 peer 机器上作为一个进程运行。

以外部服务模式使用链码便不需要再每个节点上安装链码。当链码端点部署并替换运行后，你依然能执行提交通道上定义的链码和调用链码。