链滴

HyperLedger Fabric1.4.4 的部署以及测试

作者：itcalvin

原文链接：https://ld246.com/article/1575605275192

来源网站：链滴

HyperLedger Fabric的部署以及测试

一、部署

1、软硬件准备（Mine）

* a、centos 7.x
* b、jdk 1.8
* c、go 1.12.13
* d、node-v12.13.1
* e、python 3.6 （这里我用了anaconda）
* f、docker 18.03.1-ce
* g、docker-compose 1.25.0
* f、git

```
//conda 国内镜像
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --set show_channel_urls yes

// 安装yum管理工具
yum install -y yum-utils
// 添加镜像源
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.
epo
// 查看docker版本
yum list docker-ce --showduplicates|sort -r
//如果上个命令没有查询出来，先执行下面的
curl -o /etc/yum.repos.d/Docker-ce-Ali.repo https://mirrors.aliyun.com/docker-ce/linux/cento
/docker-ce.repo
// 安装
yum install docker-ce-18.03.1.ce-1.el7.centos
//设置docker 的国内镜像
# vi /etc/docker/daemon.json
{
    "registry-mirrors": ["http://hub-mirror.c.163.com"]
}
systemctl restart docker.service

//pip设置源
mkdir ~/.pip
vim pip.conf
cd ~/.pip
vim pip.conf
[global]
index-url=http://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host=mirrors.aliyun.com

//安装 docker-compose
pip install docker-compose
```

2、部署

```
//下载项目文件
git clone https://github.com/hyperledger/fabric.git
//进入scripts目录
cd fabric/scripts
//执行 bootstrap.sh 下载相关镜像，比较慢
bash bootstrap.sh
```

3、生成网络

```
// 如果 first-network目录
cd fabric/scripts/fabric-samples/first-network/
// 生成创世区块
./byfn.sh -m generate -C channel_name_xxxx
//启动网络
./byfn.sh -m up -C channel_name_xxxx
```

二、开发链码进行部署测试

参考 https://blog.csdn.net/TripleS_X/article/details/80550401

1、链码脚本

```
// chaincode_student.go
// ====CHAINCODE EXECUTION SAMPLES (CLI) ==================

// ==== 安装、实例化 ====
// peer chaincode install -p chaincodedev/chaincode/Student -n mycc -v 0
// peer chaincode instantiate -n mycc -v 0 -c '{"Args":[]}' -C myc

// ==== 调用链码 ====
// peer chaincode invoke -n mycc -c '{"Args":["initSchool", "schoolId_A", "学校A"]}' -C myc
// peer chaincode invoke -n mycc -c '{"Args":["initSchool", "schoolId_B", "学校B"]}' -C myc

// peer chaincode invoke -n mycc -c '{"Args":["addStudent", "张三", "1", "schoolId_A", "classA"]
' -C myc
// peer chaincode invoke -n mycc -c '{"Args":["addStudent", "李四", "2", "schoolId_A", "classA"]
' -C myc
// peer chaincode invoke -n mycc -c '{"Args":["addStudent", "老王", "3", "schoolId_B", "classC"]
' -C myc

// peer chaincode invoke -n mycc -c '{"Args":["updateStudent", "隔壁老王", "3", "schoolId_B", "
lassC"]}' -C myc

// peer chaincode invoke -n mycc -c '{"Args":["queryStudentByID", "1"]}' -C myc
package main
import (

    "fmt"
    "errors"
    pd "github.com/hyperledger/fabric/protos/peer"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    "encoding/json"
```

```
    "strconv"
)
type StudentChaincode struct {
}
type Student struct {
    UserId      int    `json:"user_id"` //学生id
    Name        string `json:"name"` //姓名
    SchoolId    string `json:"school_id"` //学校id
    Class       string `jsong:"class"`//班级名称
}
type School struct {
    SchoolId    string `json:"id"` //学校id
    School      string `json:"name"` //学校名称
}
// ********** chaincode begin ********** //
func (t *StudentChaincode) Init(stub shim.ChaincodeStubInterface) pd.Response {
    return shim.Success(nil)
}
func (t *StudentChaincode) Invoke(stub shim.ChaincodeStubInterface) pd.Response {

    fn, args := stub.GetFunctionAndParameters()
    fmt.Println("invoke is running " + fn)

    if fn == "initSchool" {
        return t.initSchool(stub, args)
    } else if fn == "addStudent" {
        return t.addStudent(stub, args)
    } else if fn == "queryStudentByID" {
        return t.queryStudentByID(stub, args)
    } else if fn == "deleteSchool" {
        return t.deleteSchool(stub, args)
    } else if fn == "updateStudent" {
        return t.updateStudent(stub, args)
    }

    fmt.Println("invoke did not find func: " + fn)
    return shim.Error("Received unknown function invocation")
}

//初始化学校，学生隶属于学校
func (t *StudentChaincode) initSchool(stub shim.ChaincodeStubInterface, args []string) pd.Re
ponse {
    if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2(school_id, school_name)")
    }

    schoolId := args[0]
    schoolName := args[1]
    school := &School{schoolId, schoolName}

    //这里利用联合主键，使得查询school时，可以通过主键的“school”前缀找到所有school
    schoolKey, err := stub.CreateCompositeKey("School", []string{"school", schoolId})
    if err != nil {
        return shim.Error(err.Error())
```

```
    }

    //结构体转json字符串
    schoolJSONasBytes, err := json.Marshal(school)
    if err != nil {
        return shim.Error(err.Error())
    }
    //保存
    err = stub.PutState(schoolKey, schoolJSONasBytes)
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(schoolJSONasBytes)
}

//删除学校，包括删除所有对应学生信息
func (t *StudentChaincode) deleteSchool(stub shim.ChaincodeStubInterface, args []string) pd.
esponse {
    if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1(schoolid)")
    }
    schoolidAsString := args[0]

    schoolKey, err := stub.CreateCompositeKey("School", []string{"school", schoolidAsString})
    if err != nil {
        return shim.Error(err.Error())
    }

    schoolAsBytes, err := stub.GetState(schoolKey)
    if err != nil {
        return shim.Error("Failed to get school:" + err.Error())
    } else if schoolAsBytes == nil {
        return shim.Error("School does not exist")
    }
    //删除学校
    err = stub.DelState(schoolKey)
    if err != nil {
        return shim.Error("Failed to delete school:" + schoolidAsString + err.Error())
    }
    //删除学校下的所有学生
      queryString := fmt.Sprintf("{\"selector\":{\"school_id\":\"%s\"}}", schoolidAsString)
      resultsIterator, err := stub.GetQueryResult(queryString)//富查询，必须是CouchDB才行
      if err != nil {
        return shim.Error("Rich query failed")
    }
    defer resultsIterator.Close()
    for i := 0; resultsIterator.HasNext(); i++ {
        responseRange, err := resultsIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }
        err = stub.DelState(responseRange.Key)
        if err != nil {
            return shim.Error("Failed to delete student:" + responseRange.Key + err.Error())
```

```
        }
    }
    return shim.Success(nil)
}

//添加学生，需要检查所属学校是否已经初始化
func (t *StudentChaincode) addStudent(stub shim.ChaincodeStubInterface, args []string) pd.R
sponse {
    st, err := studentByArgs(args)
    if err != nil {
        return shim.Error(err.Error())
    }

    useridAsString := strconv.Itoa(st.UserId)

    //检查学校是否存在，不存在则添加失败
    schools := querySchoolIds(stub)
    if len(schools) > 0 {
        for _, schoolId := range schools {
            if schoolId == st.SchoolId {
                goto SchoolExists;
            }
        }
        fmt.Println("school " + st.SchoolId+ " does not exist")
        return shim.Error("school " + st.SchoolId+ " does not exist")
    } else {
        fmt.Println("school " + st.SchoolId+ " does not exist")
        return shim.Error("school " + st.SchoolId+ " does not exist")
    }

    SchoolExists:
    //检查学生是否存在
    studentAsBytes, err := stub.GetState(useridAsString)
    if err != nil {
        return shim.Error(err.Error())
    } else if studentAsBytes != nil {
        fmt.Println("This student already exists: " + useridAsString)
        return shim.Error("This student already exists: " + useridAsString)
    }

    //结构体转json字符串
    studentJSONasBytes, err := json.Marshal(st)
    if err != nil {
        return shim.Error(err.Error())
    }
    //保存
    err = stub.PutState(useridAsString, studentJSONasBytes)
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(studentJSONasBytes)
}
```

```
//删除学生
func (t *StudentChaincode) deleteStudent(stub shim.ChaincodeStubInterface, args []string) pd
Response {
    if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1(userid)")
    }
    useridAsString := args[0]
    studentAsBytes, err := stub.GetState(useridAsString)
    if err != nil {
        return shim.Error("Failed to get student:" + err.Error())
    } else if studentAsBytes == nil {
        return shim.Error("Student does not exist")
    }

    err = stub.DelState(useridAsString)
    if err != nil {
        return shim.Error("Failed to delete student:" + useridAsString + err.Error())
    }
    return shim.Success(nil)
}

//更新学生信息。如果学生不存在，则执行新增学生逻辑
func (t *StudentChaincode) updateStudent(stub shim.ChaincodeStubInterface, args []string) p
.Response {
    st, err := studentByArgs(args)
    if err != nil {
        return shim.Error(err.Error())
    }
    useridAsString := strconv.Itoa(st.UserId)

    //检查学校是否存在，不存在则添加失败
    schools := querySchoolIds(stub)
    if len(schools) > 0 {
        for _, schoolId := range schools {
            if schoolId == st.SchoolId {
                goto SchoolExists;
            }
        }
        fmt.Println("school " + st.SchoolId+ " does not exist")
        return shim.Error("school " + st.SchoolId+ " does not exist")
    } else {
        fmt.Println("school " + st.SchoolId+ " does not exist")
        return shim.Error("school " + st.SchoolId+ " does not exist")
    }

    SchoolExists:
    //因为State DB是一个Key Value数据库，如果我们指定的Key在数据库中已经存在，那么就是修
操作，如果Key不存在，那么就是插入操作。
    studentJSONasBytes, err := json.Marshal(st)
    if err != nil {
        return shim.Error(err.Error())
    }
    //保存
    err = stub.PutState(useridAsString, studentJSONasBytes)
```

```
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(studentJSONasBytes)
}

//根据学生ID查看学生信息
func (t *StudentChaincode) queryStudentByID(stub shim.ChaincodeStubInterface, args []string
 pd.Response {
    if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1(userid)")
    }
    useridAsString := args[0]
    studentAsBytes, err := stub.GetState(useridAsString)
    if err != nil {
        return shim.Error("Failed to get student:" + err.Error())
    } else if studentAsBytes == nil {
        return shim.Error("Student does not exist")
    }

    // 解析json字符串到结构体
    // st := Student{}
    // err = json.Unmarshal(studentAsBytes, &st)
    // if err != nil {
    //     return shim.Error(err.Error())
    // }
    fmt.Printf("Query Response:%s\n", string(studentAsBytes))
    return shim.Success(studentAsBytes)
}

// ********** chaincode end ********** //

// ********** tool fun begin ********** //
//将参数构造成学生结构体
func studentByArgs(args []string) (*Student, error) {
    if len(args) != 4 {
        return nil, errors.New("Incorrect number of arguments. Expecting 4(name, userid, schooli
, classid)")
    }

    name := args[0]
    userId, err := strconv.Atoi(args[1]) //字符串转换int
    if err != nil {
        return nil, errors.New("2rd argument must be a numeric string")
    }
    schoolId := args[2]
    class := args[3]
    st := &Student{userId, name, schoolId, class}

    return st, nil
}

//获取所有创建的学校id
```

```
func querySchoolIds(stub shim.ChaincodeStubInterface) []string {
    resultsIterator, err := stub.GetStateByPartialCompositeKey("School", []string{"school"})
    if err != nil {
        return nil
    }
    defer resultsIterator.Close()

    scIds := make([]string,0)
    for i := 0; resultsIterator.HasNext(); i++ {
        responseRange, err := resultsIterator.Next()
        if err != nil {
            return nil
        }
        _, compositeKeyParts, err := stub.SplitCompositeKey(responseRange.Key)
        if err != nil {
            return nil
        }
        returnedSchoolId := compositeKeyParts[1]
        scIds = append(scIds, returnedSchoolId)
    }
    return scIds
}
// ********** tool fun end ********** //

// ********** main ********** //
func main() {
    if err := shim.Start(new(StudentChaincode)); err != nil {
        fmt.Printf("Error starting student chaincode: %s", err)
    }
}
```

2、开发环境测试链码

将上述的脚本拷贝到 fabric-samples/chaincode 目录

```
//进入链码开发环境目录
cd fabric-samples/chaincode-docker-devmode
```

开启3个终端

```
//终端 1 – 启动网络
docker-compose -f docker-compose-simple.yaml up

//终端 2 – 编译和部署链码
docker exec -it chaincode bash
cd Student
go build
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./Student

//终端3 – 使用链码
docker exec -it cli bash
$ cd ../
//安装链码
```

```
$ peer chaincode install -p chaincodedev/chaincode/Student -n mycc -v 0
//初始化
$ peer chaincode instantiate -n mycc -v 0 -c '{"Args":[]}' -C myc
//调用链码
peer chaincode invoke -n mycc -c '{"Args":["initSchool", "schoolId_A", "学校A"]}' -C myc
peer chaincode invoke -n mycc -c '{"Args":["addStudent", "张三", "1", "schoolId_A", "classA"]}'
C myc
```