UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Paralelni i distribuirani algoritmi i strukture podataka

ms Nebojša Horvat

Zimski semestar 2019/2020.

Studijski program: Računarstvo i automatika

Modul: Računarstvo visokih performansi

Hyperledger Fabric

Chaincode

- Pametni ugovor
  - Biznis logika koja se izvršava u Fabric-u
  - Paket shim
  - Chaincode je zadužen za pisanje podataka u ledger, kao i za čitanje istih
  - Može se pisati u više jezika
    - Golang (podrazumevani)
    - Javascript
    - Java
    - ...

Struktura chaincode-a

- Chaincode ima funkciju koja ga pokreće
  - Main
- Init
- Invoke
- ~~Query~~
  - Više ne postoji, već se i za upite koristi Invoke funkcija

Init

- Funkcija koja se poziva prilikom instanciranja i upgrade-a chaincode-a
- Koristi se da se inicijalizuje World State početnim vrednostima
- Često se kreira posebna funkcija koja inicijalizuje World State, radi lakšeg testiranja

Init

```
type SmartContract struct {
    contractapi.Contract
}

type Asset struct {
    ID             string `json:"ID"`
    Color          string `json:"color"`
    Size           int    `json:"size"`
    Owner          string `json:"owner"`
    AppraisedValue int    `json:"appraisedValue"`
}

func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
    assets := []Asset{
        {ID: "asset1", Color: "blue", Size: 5, Owner: "Tomoko", AppraisedValue: 300},
    }
    for _, asset := range assets {
        assetJSON, err := json.Marshal(asset)
        if err != nil {
            return err
        }
         err = ctx.GetStub().PutState(asset.ID, assetJSON)
         if err != nil {
            return fmt.Errorf("failed to put to world state. %v", err)
        }
    }
    return nil
}
```

Invoke

- Do verzije 2.0 postojale su samo Invoke i Init funkcije. Unutar Invoke funkcije je developer morao ručno da izabere funkciju koju želi da pozove i prihvati je preko prvog argumenta.
- Od verzije 2.0 nije potrebno implementirati Invoke funkciju. Moguće je diretno pozvati funkciju implementiranu chaincode-u.
- Od verzije 2.0 ne dobijamo stub objekat direktno, već ga zatražimo preko ctx-a (Transaction Context).

Upis u ledger

```
// CreateAsset issues a new asset to the world state with given details.
func (s *SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, id string, color string, size int, owner string, appraisedValue int) error {
	exists, err := s.AssetExists(ctx, id)
	if err != nil {
		return err
	}
	if exists {
		return fmt.Errorf("the asset %s already exists", id)
	}

	asset := Asset{
		ID:             id,
		Color:          color,
		Size:           size,
		Owner:          owner,
		AppraisedValue: appraisedValue,
	}
	assetJSON, err := json.Marshal(asset)
	if err != nil {
		return err
	}

	return ctx.GetStub().PutState(id, assetJSON)
}
```

Primer upita

```
// ReadAsset returns the asset stored in the world state with given id.
func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (*Asset,
error) {
    assetJSON, err := ctx.GetStub().GetState(id)
    if err != nil {
        return nil, fmt.Errorf("failed to read from world state: %v", err)
    }
    if assetJSON == nil {
        return nil, fmt.Errorf("the asset %s does not exist", id)
    }


    var asset Asset
    err = json.Unmarshal(assetJSON, &asset)
    if err != nil {
        return nil, err
    }


    return &asset, nil
}
```

Shim paket

- Shim paket se koristi za rad sa ledger-om
  - Pisanje podataka na ledger
  - Čitanje podataka

SHIM paket

- GetState(key) - Vraća vrednost svojine iz Fabric World State-a
- PutState(key,value) - Postavi vrednost svojine u Fabric World State
- GetHistoryForKey(key) - Upit koji vraća sve transakcije nad odgovarajućom svojinom
- DelState(key) // Stavlja “deleted” flag u World State za odgovarajuću svojinu
- GetQueryResult(query) //Vraća iterator na rezultat pretrage specificiran query stringom
- GetStateByRange(from,to) //Vraća iterator na rezultat pretrage specificiran opsegom ključceva (from, to)
- Kompletna dokumentacija
  - https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim

JSON kao vrednost u World State-u

```
func (tx *Transaction) FromJson(txJson []byte) (*Transaction, error) {

    error:= json.Unmarshal(txJson, &tx)

    if error != nil {
        return nil, errors.New(err.Error())
    }

    return tx, nil
}

func (tx *Transaction) ToJson() ([]byte, error) {

    txJson, err or:= json.Marshal(tx)

    if error != nil {
        logger.Error("Error while marshaling")
        return nil, errors.New(err.Error())
    }

    return txJson, nil
}
```

Instaliranje chaincode-a

- Lifecycle system chaincode (LSCC) mora biti iskorišćen da bi se instalirao chaincode na peer-ovima

```
peer lifecycle chaincode install ${CC_NAME}.tar.gz >&log.txt
```

Zadatak

Izmeniti postojeći asset-transfer-basic chaincode, tako da se prilikom poziva Transfer asset funkcije dodaje dodatni argument koji određuje koliko je cena asset-a porasla posle transfera.

Zadatak

Izmeniti postojeći asset-transfer-basic chaincode, tako da se dodaje nova funkcija koja menja boju asset-a.

NodeJS chaincode- primer

```
class SecondaryMarket {


    async Init(stub) {
    }
    async Invoke(stub) {
        let functionAndParams = stub.getFunctionAndParameters();
        let functionName = functionAndParams.fcn;
        let functionParams = functionAndParams.params;


        logger.debug("Invoking " + functionName + "with args: " + functionParams);
        ....
        let fcn = this[functionName];
        return await fcn.call(this, stub, functionParams);
    }
    async createTx(stub, args) {
        let something= args[0]     ...
         // some validation

        let a= args[1];
        let receivedVal = stub.getState(a)
        //use receivedVal
        let val= JSON.parse(something)
        //change val
        await stub.putState(txId, Buffer.from(val));
        return shim.success(Buffer.from("Transaction saved !"));
    }
```