BLOCKCHAIN PROVIDER PLUGIN HANDLEIDING

VOOR DE PERFORMANCE TESTING TOOL

Lex Felix

28 November 2019

INHOUDSOPGAVE

INTRODUCTIE

In deze handleiding wordt beschreven hoe een blockchain provider plugin geschreven wordt voor de performance testing tool (PTT). Er is een git repository beschikbaar voor het performance testing tool project. Hierin zit ook documentatie voor de werking van de performance testing tool, een handleiding voor het schrijven van een testrunner plugin en een module (empty-blockchainprovider) in het performance testing tool maven project die gebruikt wordt als startpunt voor deze handleiding.

DEPENDENCIES

Een blockchain provider plugin **moet** gebruik maken van een aantal libraries:

- PF4J 3.1.0 – Plugin framework
- Akka 2.5.26 (classic) – Concurrency framework
- API module PTT – De klassen die overeen komen
- SLF4J – Logging framework
- Akka Testkit 2.5.26 – Voor het testen van de providers
- Junit 5.5.2 – Voor het testen van de providers

Hiernaast kunnen er naar eigen dependencies toegevoegd worden zolang dat deze dependencies werken met de performance testing tool die gecompileerd wordt voor Java 11. Een aantal geadviseerde dependencies staan in al in de pom van de start module.

Akka is de belangrijkste library om te begrijpen. Interactie me je plugin gebeurt via Akka. Akka is een concurrency framework gebaseerd op het Actor Model. Hierin worden berichten naar de mailbox van Actoren gestuurd. Elke Actor heeft zijn eigen mailbox en handelt de berichten synchroon af. Doordat Actoren parallel kunnen draaien op aparte Threads wordt zo concurrency verkleint doordat alleen nagedacht hoeft te worden aan het uitwisselen van berichten. Meer over Akka is te lezen in hun documentatie. **Er wordt gebruik gemaakt van Akka classic binnen de performance testing tool.**

BOUWEN

De plugin wordt via maven als een fat jar gebouwd. In de start module hoeft niets aan de build in de pom.xml veranderd te worden. Wel moeten wat properties veranderd worden zodat de plugin gebruikt kan worden in de PTT.

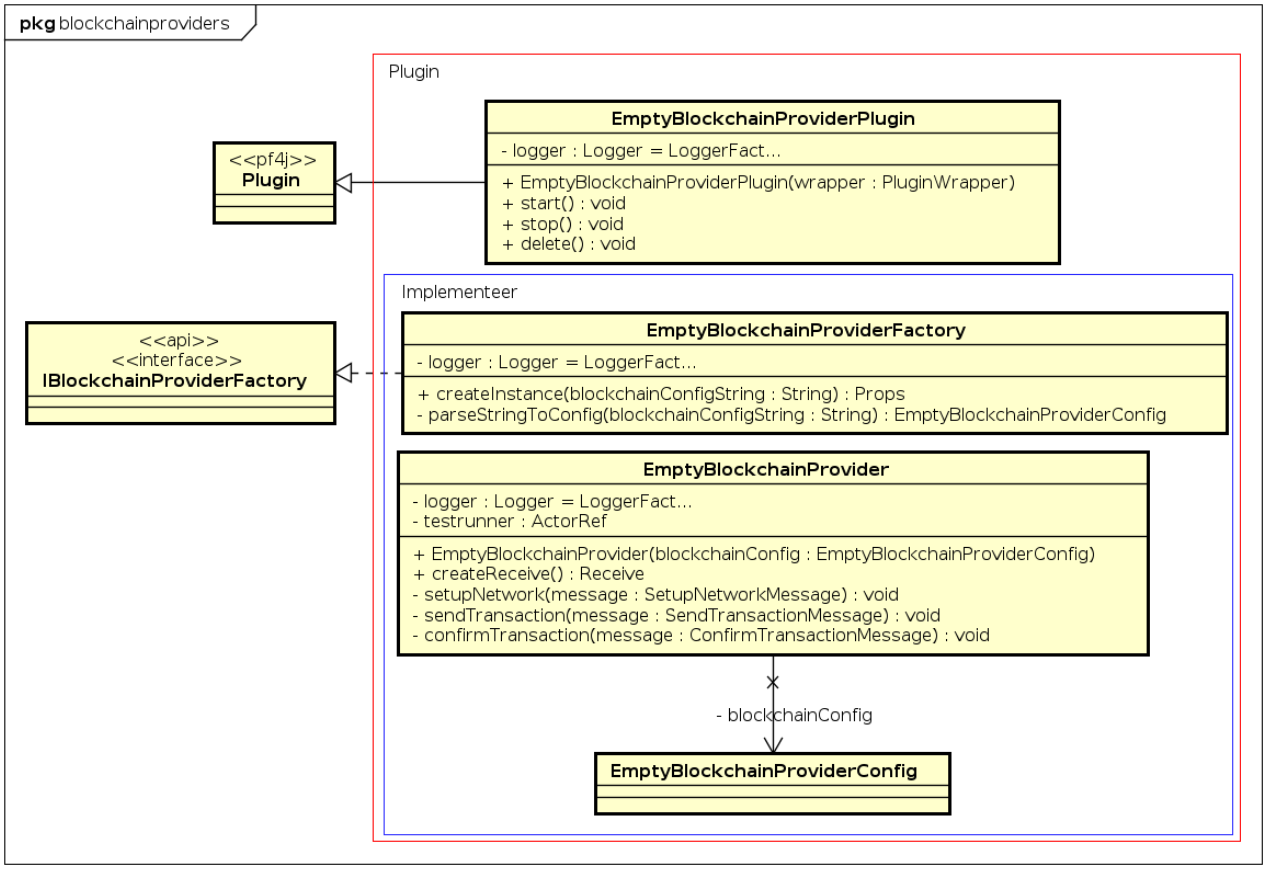
artifactId	ArtifactId van het project
name	Naam van het project
jar.name	Uiteindelijke naam van de plugin jar
plugin.class	Volledige klasse naam van de EmptyBlockchainProviderPlugin
plugin.id	De unique identifier van de plugin, wordt gebruikt als type in het testscenario bestand
plugin.version	Versie van de plugin (Semantic Versioning Specification)
plugin.description	Beschrijving van de plugin (optioneel)
plugin.provider	Auteur van de plugin (optioneel)
plugin.license	Licentie van de plugin (optioneel)

Tabel 1: Pom.xml variabelen veranderen

Nadat de plugin gebouwd is, is het te vinden in de target map zoals normaal. De plugin kan gebruikt worden in de PTT door het in een folder genaamd “plugins” te stoppen die naast de ptt.jar staat en testscenario bestand te schrijven die gebruik maakt van je plugin.

START MODULE

De start module in de git repository ziet er als volgt uit.



Figuur 1: Klassen overzicht start module

Alles in de blauwe box moet geïmplementeerd worden en alles in de rode box is minimaal nodig in het project voor een werkende plugin. De EmptyBlockchainProviderPlugin hoeft dus niets aan veranderd te worden. Dit is namelijk een klasse die het plugin framework PF4J gebruikt voor het inladen van alle extensies. Het implementeren van de EmptyBlockchainProviderFactory is te vinden in het hoofdstuk Factory en voor de EmptyBlockchainProvider is het hoofdstuk Provider. De klasse EmptyBlockchainProviderConfig is een POJO voor je configuratie variabelen, meer informatie in het hoofdstuk Configuratie. Het is mogelijk om zelf nieuwe klassen toe te voegen die gebruikt worden in deze klassen.

FACTORY

De EmptyBlockchainProviderFactory klasse is een implementatie van de IBlockchainproviderFactory. Deze factory wordt in een abstract factory pattern gebruikt binnen de PTT voor het instantiëren van een blockchain provider. De PTT

kan een instantie aanvragen door de createInstance methode aan te roepen van je factory. Deze methode heeft de taak om een instantie van je EmptyBlockchainProvider terug te geven die gebruikt kan worden zoals beschreven in het Hoofdstuk Provider. Om dit voor elkaar te krijgen moet gezorgd worden dat er een Props object gemaakt wordt met daarin een valide EmptyBlockchainProviderConfig object. Een Props object is nodig omdat, binnen de PTT gebruikt wordt gemaakt van Akka, een concurrency framework. Hierdoor worden blockchain providers actoren in een Akka systeem en verandert het instantiëren. Waar normaal gesproken een nieuw object aangemaakt moet worden moet voor een actor een Props object aangemaakt worden.

```
Props blockchainProvider = Props.create(EmptyBlockchainProvider.class, config);
```

Het aanmaken verschilt niet veel van het gebruiken van een normale constructor. Binnen de parameters moet de eerste de klasse zijn en de rest moeten overeen komen met parameters van een constructor in EmptyBlockchainProvider. Hierdoor kan Akka zelf de actoren aanmaken op andere threads wanneer dat nodig is.

In het code voorbeeld is config een valide EmptyBlockchainProviderConfig object. Dit object is een POJO met alle configuratie variabelen die in de configuratie blok van je blockchain provider staan (Hoofdstuk Configuratie voor meer informatie). Dit object kan verkregen worden door de blockchainConfigString, die als parameter wordt meegegeven, te parsen naar een EmptyBlockchainProviderConfig object en deze vervolgens te valideren. Vergelijkbare functionaliteit wordt binnen de PTT gedaan met een aantal jackson libraries deze staan in de start module onder geadviseerde dependencies. Wanneer de configuratie niet valide blijkt te zijn moet er een BlockchainConfigException worden gethrowed. Deze exception wordt gebruikt om aan de gebruiker door te geven wat er precies fout is gegaan, wees zo expliciet mogelijk in de message.

PROVIDER

De EmptyBlockchainProvider wordt in de PTT gebruikt als actor in een AkkaSystem. Hierdoor extend de EmptyBlockchainProvider van de AbstractActor klasse of een klasse die AbstractActor extend zoals AbstractActorWithTimers.

Aangezien Akka gebruikt wordt kan de PTT geen interface gebruiken om specifieke methoden aan te roepen van de EmptyBlockchainProvider. Om deze reden is er ook geen interface voor de EmptyBlockchainProvider beschikbaar gesteld. In de plaats van een interface zijn een set aan bericht klassen opgesteld waar een functionaliteit aan gekoppeld is. Het afhandelen van deze berichten en de bijbehorende functionaliteit uit te voeren is de taak van de EmptyBlockchainProvider.

Bericht klassen moeten gekoppeld worden aan een methode in de createReceive method.

```
@Override
public Receive createReceive() {
    return receiveBuilder()
            .match(SetupNetworkMessage.class, this::setupNetwork)
            .match(SendTransactionMessage.class, this::sendTransaction)
            .build();
}
```

Figuur 2: createReceive methode

De blockchain provider kan een tweetal berichten ontvangen:

- SetupNetworkMessage
- SendTransactionMessage

SETUPNETWORKMESSAGE

Bij een SetupNetworkMessage moet de blockchain provider het netwerk klaarzetten voor testen. Wat hiervoor gedaan moet worden is per blockchain provider verschillend. Gedacht moet worden aan het klaarzetten van een contract (ethereum) of het opzetten van een channel (hyperledger fabric). Nadat het klaarzetten klaar is zijn er een aantal variabelen verkregen, denk aan een contract adres in ethereum. Deze variabelen kunnen worden gedeeld met andere instanties van je blockchain provider. Daarom moeten deze variabelen in een string verwerkt worden en in een SetupNetworkMessage terug worden gestuurd.

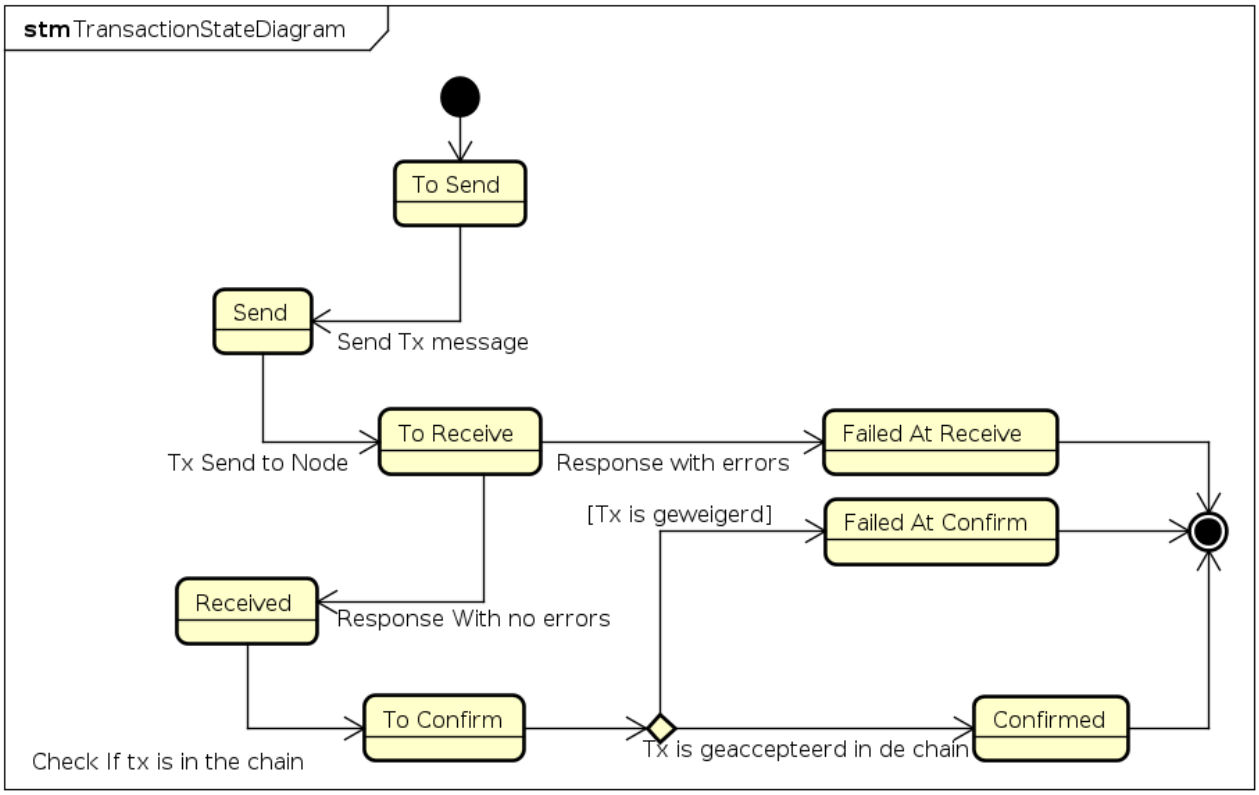
```
message.setSetupConfigString("variabelen");
getSender().tell(message, getSelf());
```

Figuur 3: Verstuur bericht naar sender

Dit bericht wordt dan naar andere instanties van je blockchain provider gestuurd zodat ze deze variabelen kunnen gebruiken. De methode die aan SetupNetworkMessage is gekoppeld moet dus kijken of hij een setup moet uitvoeren of variabelen moet opslaan. Dit kan gedaan worden door te kijken of getSetupConfigString null terug geeft, dit betekend dat er een setup uitgevoerd moet worden. Bij het opslaan van variabelen is het niet nodig om een SetupNetworkMessage terug te sturen

SEND TRANSACTION

Bij een SendTransactionMessage moet de blockchain provider een transactie versturen naar het netwerk en hier data over verzamelen. De transactie die verstuurd moet worden is bepaald door de unique identifier functionId in het bericht. De data die verzameld moet worden staat in de klasse TransactionResult. Een object van deze klasse moet aan het begin gemaakt worden voor een specifieke transactie. Een TransactionResult object mag dus geen data van een andere transactie bevatten. Er wordt vanuit gegaan dat een transactie zich in de volgende staten kan begeven.



Figuur 4: Transactie State diagram

Wanneer een transactie zich in een Send, Receive of Confirmed begeeft dan moet de tijd opgeslagen worden in het TransactionResult object. Wanneer de transactie niet goed ontvangen of niet geconfirmeerd wordt bereikt de transactie een Failed staat. Wanneer een van deze staten zijn bereikt dan moet er een TransactionError object worden toegevoegd aan het TransactionResult object. Hierdoor kan de PTT zien dat de transactie gefaald is en door de informatie in het object kan in het test rapport verwerkt worden hoeveel transacties er om dezelfde reden zijn gefaald en wanneer een transactie gefaald is. Het is belangrijk dat de code variabele uniek is voor alle blockchain providers (denk aan provider afhankelijke voorvoegsels of achtervoegsels). Hierdoor kan het zijn dat codes niet expliciet genoeg zijn verdere uitleg kan geplaatst worden in de message variabele. Wanneer de transactie een geldige eind staat heeft bereikt en dus klaar is moet het TransactionResult object terug worden gestuurd naar de testrunner. Geldige eind staten zijn Failed At Receive, Failed At Confirm en Confirmed.

Een blockchain provider moet tijdens het uitvoeren van het testscenario zo snel mogelijk reageren op SendTransactionMessage berichten. De testrunner bepaald wanneer een transactie moet worden verstuurd zodat het testscenario goed verloopt. Dit betekend dat de blockchain provider niet kan wachten tot een transactie geconfirmeerd is aangezien dit een lange tijd kan duren. Om dit te voorkomen kan de blockchain provider ook berichten naar zichzelf sturen om een staat verandering te checken. Deze berichten kan je zelf definiëren en zorgen ervoor dat nieuwe SendTransactionMessages in de mailbox afgehandeld kunnen worden. Hierbij moet je wel gedacht worden dat je de link tussen TransactionResult object en transactie niet kwijt raakt, dit kan gedaan worden door het object mee te sturen in het bericht. Ook verlies je bij deze berichten de referentie naar de testrunner in de method getSender. Hiervoor kan je de referentie opslaan in de blockchain provider (er is een één op één relatie tussen blockchain provider en testrunner) of ook meesturen in het bericht. Als je staat veranderingen periodiek wilt controleren, als je bijvoorbeeld een confirmatie aanvraag moet doen bij het blockchain netwerk, dan kan je gebruik maken van

timers of scheduled messages (in de start module staat een voorbeeld met confirmTransaction en ConfirmTransactionMessage).

CONFIGURATIE

Om te kunnen communiceren met het blockchain netwerk is het nodig om informatie te krijgen van de gebruiker. Welke informatie nodig is, is heel verschillend en moet je zelf bepalen. Het is dus belangrijk dat de gebruiker duidelijk kan lezen welke informatie gegeven moet worden.

De informatie wordt in een configuratie blok in het test scenario bestand gezet door de gebruiker.

```yaml
# this is just an example of a testscenario file your blockchain provider will only receive the config value block
# for testing put the config values inside testBlockchainProviderConfigBlock.yml file
runners:
  testrunner1:
    &testrunner
    type: testrunner
    providerId: blockchainprovider1
    #  testrunner config block BEGIN

    #  testrunner config block END
  testrunner2:
    <<: *testrunner
    providerId: blockchainprovider2
useProvidersForSetup:
  blockchainprovider1:
    passthroughSetupDataTo:
      - blockchainprovider2
providers:
  blockchainprovider1: &blockchainprovider
    type: empty-blockchainprovider
    functions:
      - function1
      - function2
    #  blockchain provider config block BEGIN

    #  blockchain provider config block END
  blockchainprovider2:
    <<: *blockchainprovider
```

Figuur 5: Test scenario configuratie bestand voorbeeld

Het test scenario bestand is een yaml bestand met extra ondersteuning voor merge keys waardoor configuratie blokken makkelijk gedupliceerd kunnen worden. De enige eis is dus dat het configuratie blok geldige yaml is zodat het door de PTT geparsed kan worden. Na het parsen wordt het configuratie blok eruit gehaald en geconverteerd naar een string. Als de createInstance methode

wordt aangeroepen van je EmptyBlockchainProviderFactory wordt deze string gebruikt.

TESTING

Om de EmptyBlockchainProvider goed te testen is het TestKit van Akka nodig. Dit is een klasse om een actor binnen een Akka systeem te mocken en te gebruiken. Hiermee kan een testrunner gesimuleerd worden. Voor de EmptyBlockchainProvider zijn al drie testen bijgevoegd die de happy flow van de blockchain provider testen. Hiermee kan getest worden of je blockchain provider aan de minimale eisen voldoen. Errors handling of dat transacties daadwerkelijk verstuurd worden, worden hiermee niet getest. Dit moet zelf getest worden.