mojaloop

Defining and executing rules in Mojaloop
July 2020

mojaloop

What problem do we need to solve?

Specific requirements

- TIPS need to calculate and record interchange fees which are applicable to some types of transfer
- There will be additional requirements to charge fees based on types of transfer
- Mowali need to categorise transfers according to their own rules

The general version of the problem:

- We need to evaluate a rule for each transaction being processed in the switch.
- We need to take some action if the rule is passed.
- If possible, we'd like this solution to be:
  1. Efficient
  2. Generic.
  3. Capable of being controlled by administrators, not software developers

So what do we want?

1. A way of executing a rule-based evaluation of (initially) an interchange fee
2. A framework to enable us to execute a variable number of such evaluations
   - (and to change them while the switch is running…)
   - (and, ideally, to locate them at various points in the end-to-end transfer process)
3. A way of representing an evaluation in human-readable text form
4. A deterministic way of turning 3 into 1
5. A user interface to allow administrators to create and modify these evaluations
6. A deterministic way of turning 5 into 3

Critical success factors

1. A way of executing a rule-based evaluation of (initially) an interchange fee
   - Efficiency in operation
2. A framework to enable us to execute a variable number of such evaluations
   - Statelessness
   - Efficiency in operation
3. A way of representing an evaluation in human-readable text form
   - Clarity
   - Concision
4. A deterministic way of turning 3 into 1
5. A user interface to allow administrators to create and modify these evaluations
   - Ease of use
   - Responsiveness to changes in data structures and requirements
6. A deterministic way of turning 5 into 3

Solving the specific problem

1. A way of executing a rule-based evaluation of (initially) an interchange fee
2. A framework to enable us to execute a variable number of such evaluations
   - (and to change them while the switch is running…)
   - (and, ideally, to locate them at various points in the end-to-end transfer process)

We restrict ourselves to these requirements

The specific TIPS solution

1) Defining and evaluating a rule

What is the TIPS requirement?

The rules for TIPS interchange fees are:

- If the transaction is a wallet-to-wallet P2P transaction, then the receiver pays the sender 0.6% of the amount of the transaction.
- No interchange fees are levied for on-us transactions.
- Interchange fees are recorded by the switch.
- Net obligations will be aggregated by the switch and communicated to participants on a regular cycle.
- Participants are not required to maintain liquidity cover for interchange fee obligations

Assumptions

- All information required is contained in the Transaction object (however that is defined.)
- We hold the account type for Payer and Payee in the extension list of the Transaction object.
  - These items have the keys “PayerAccountType” and “PayeeAccountType”.
  - The values can be set to “WALLET” or “ACCOUNT”.

Evaluating a rule

- Question: What's the most efficient way of evaluating a rule?
- Answer: direct code

The actual implementation of the TIPS rule

```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId


if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === 'TRANSFER' &&
    transfer.transactionType.initiator === 'PAYER' &&
    transfer.transactionType.initiatorType === 'CONSUMER')) {
  log(`Adding an interchange fee for Wallet to Wallet from ${payerFspId} to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE', // Ledger account type Id
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2),
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

The actual implementation of the TIPS rule

```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId

if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === 'TR
    transfer.transactionType.initiator === 'PAY
    transfer.transactionType.initiatorType ===
  log(`Adding an interchange fee for Wallet to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE', // Ledger account type Id
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2),
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

First, we exclude on-us transfers

The actual implementation of the TIPS rule

```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId

if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === 'TRANSFER' &&
    transfer.transactionType.initiator === 'PAYER' &&
    transfer.transactionType.initiatorType === 'CONSUMER')) {
  log(`Adding an interchange fee for Wallet to Wallet ${payerFspId} to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE'
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2)
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

Next, we include only wallet-to-wallet transfers

The actual implementation of the TIPS rule

```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId


if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.extensionList.extension,
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === 'TRANSFER' &&
    transfer.transactionType.initiator === 'PAYER' &&
    transfer.transactionType.initiatorType === 'CONSUMER')) {
  log(`Adding an interchange fee for Wallet to Wallet fr  ${payerFspId} to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE', // Ledg      unt type Id
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2),
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

And this is a P2P payment, obviously…

mojaloop

The actual implementation of the TIPS rule

```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId

if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensi
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.extens
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === 'TRANSFER' &&
    transfer.transactionType.initiator === 'PAYER' &&
    transfer.transactionType.initiatorType === 'CONSUMER')) {
  log(`Adding an interchange fee for Wallet to Wallet from ${payerFspId} to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE', // Ledger account type Id
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2),
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

We log the fact that we're including an interchange fee…

The actual implementation of the TIPS rule

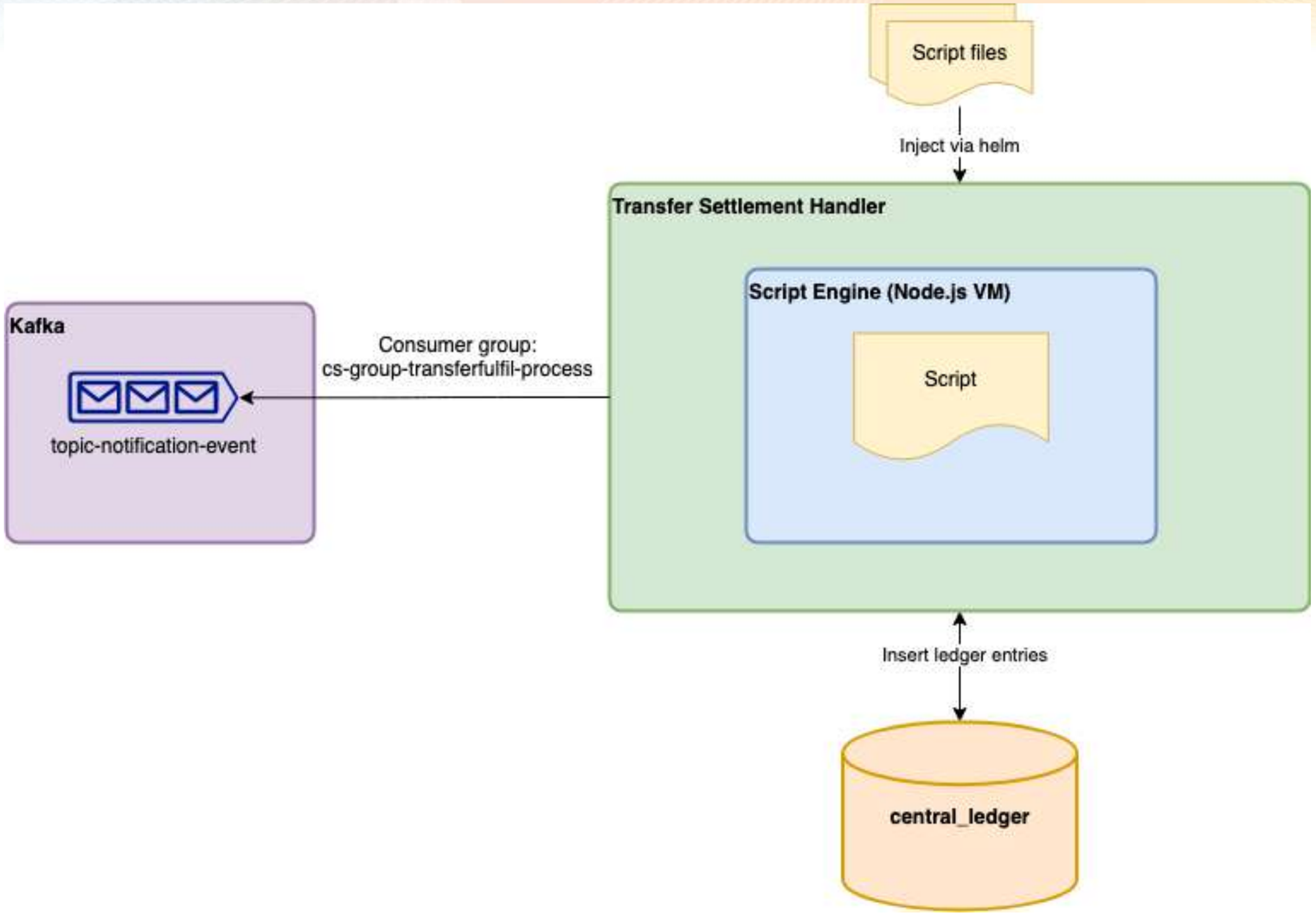
```
const payerFspId = transfer.payer.partyIdInfo.fspId
const payeeFspId = transfer.payee.partyIdInfo.fspId

if ((payeeFspId !== payerFspId) &&
  (getExtensionValue(transfer.payee.partyIdInfo.extensi
'accountType') === 'Wallet' &&
    getExtensionValue(transfer.payer.partyIdInfo.ext
'accountType') === 'Wallet') &&
  (transfer.transactionType.scenario === '          ER' &&
    transfer.transactionType.initiat       = 'PAYER' &&
    transfer.transactionType.i      iatorType === 'CONSUMER')) {
  log(`Adding an interchange fee for Wallet to Wallet from ${payerFspId} to
${payeeFspId}`)
  addLedgerEntry(payload.id, 'INTERCHANGE_FEE', // Ledger account type Id
    'INTERCHANGE_FEE', // Ledger entry type Id
    multiply(transfer.amount.amount, 0.006, 2),
    transfer.amount.currency,
    payerFspId,
    payeeFspId)
```

And now we call a generic function to add the matching ledger entries…

The specific TIPS solution

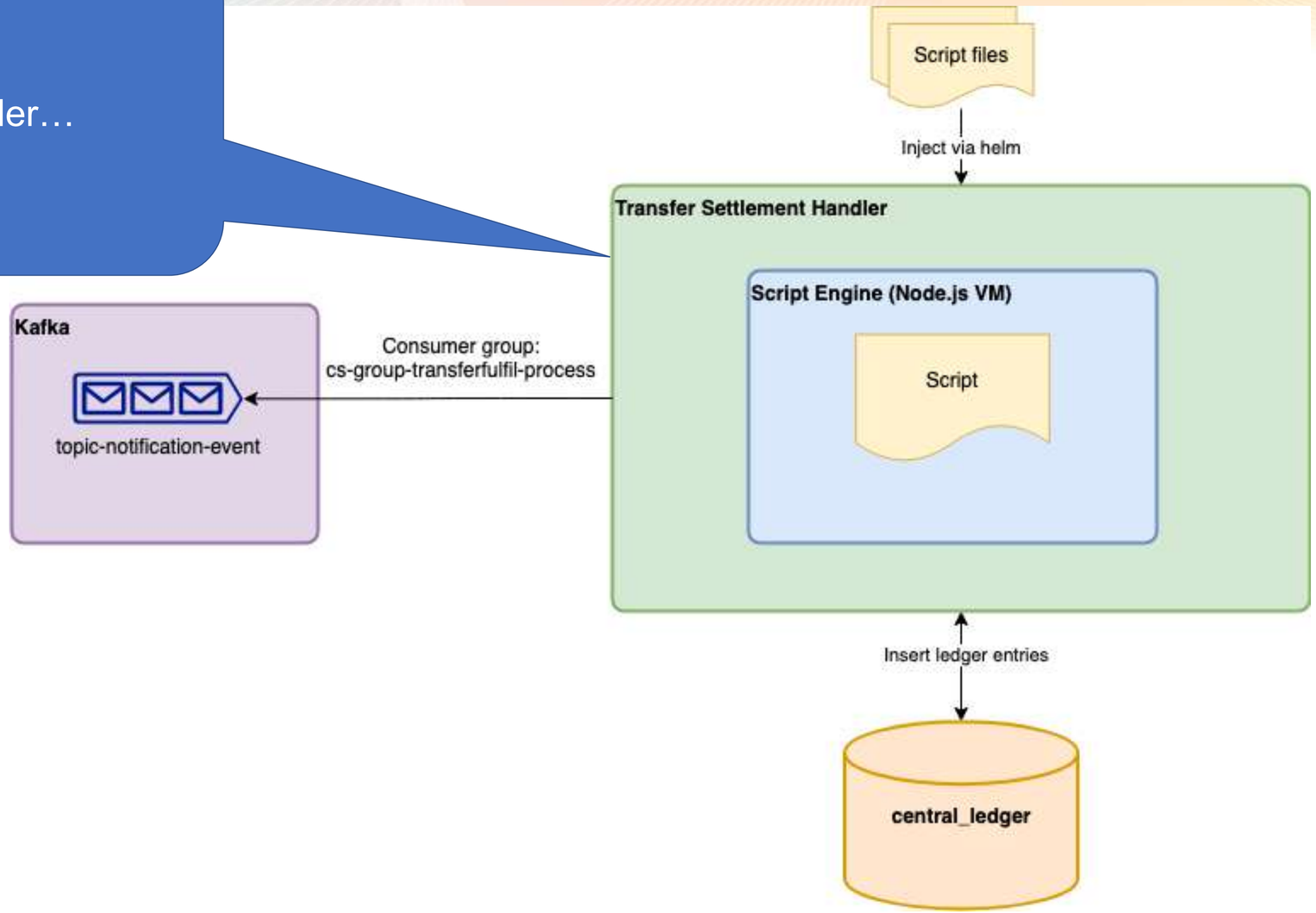
2) Executing and adding a rule

Connecting a rule to the system



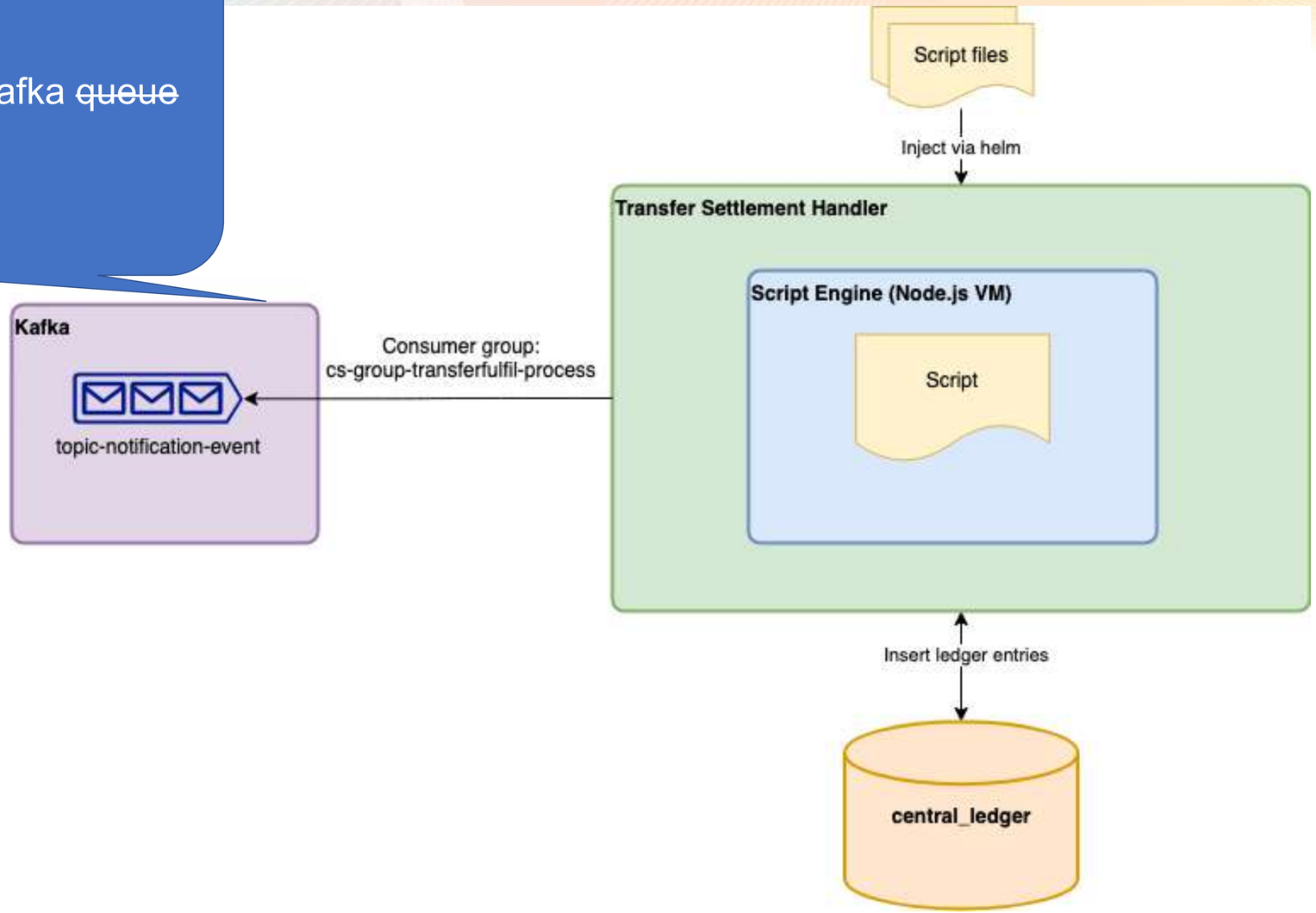
Connecting a rule to the system

We have our own handler…



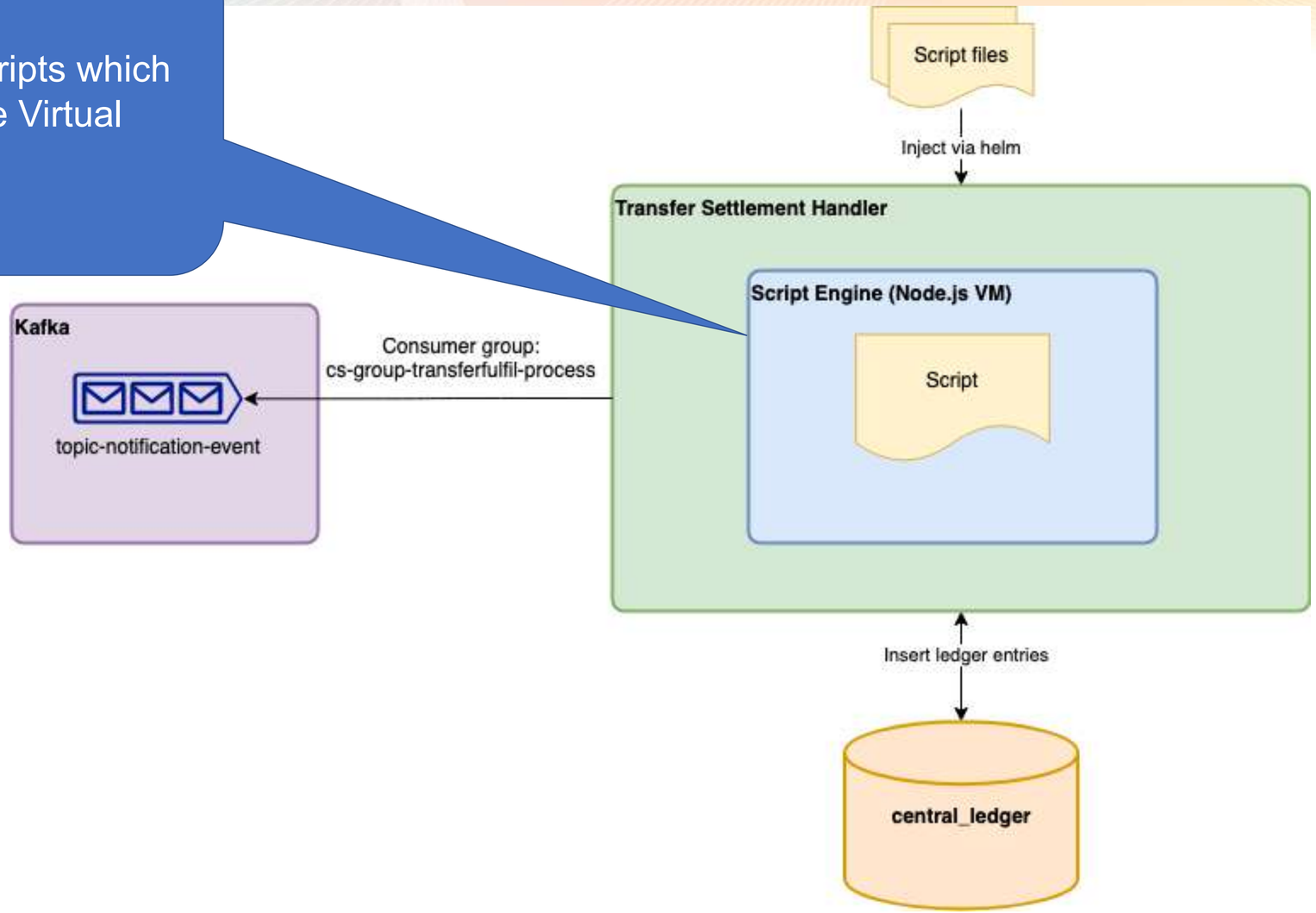
Connecting a rule to the system

Which listens to an existing Kafka ~~queue~~ stream…



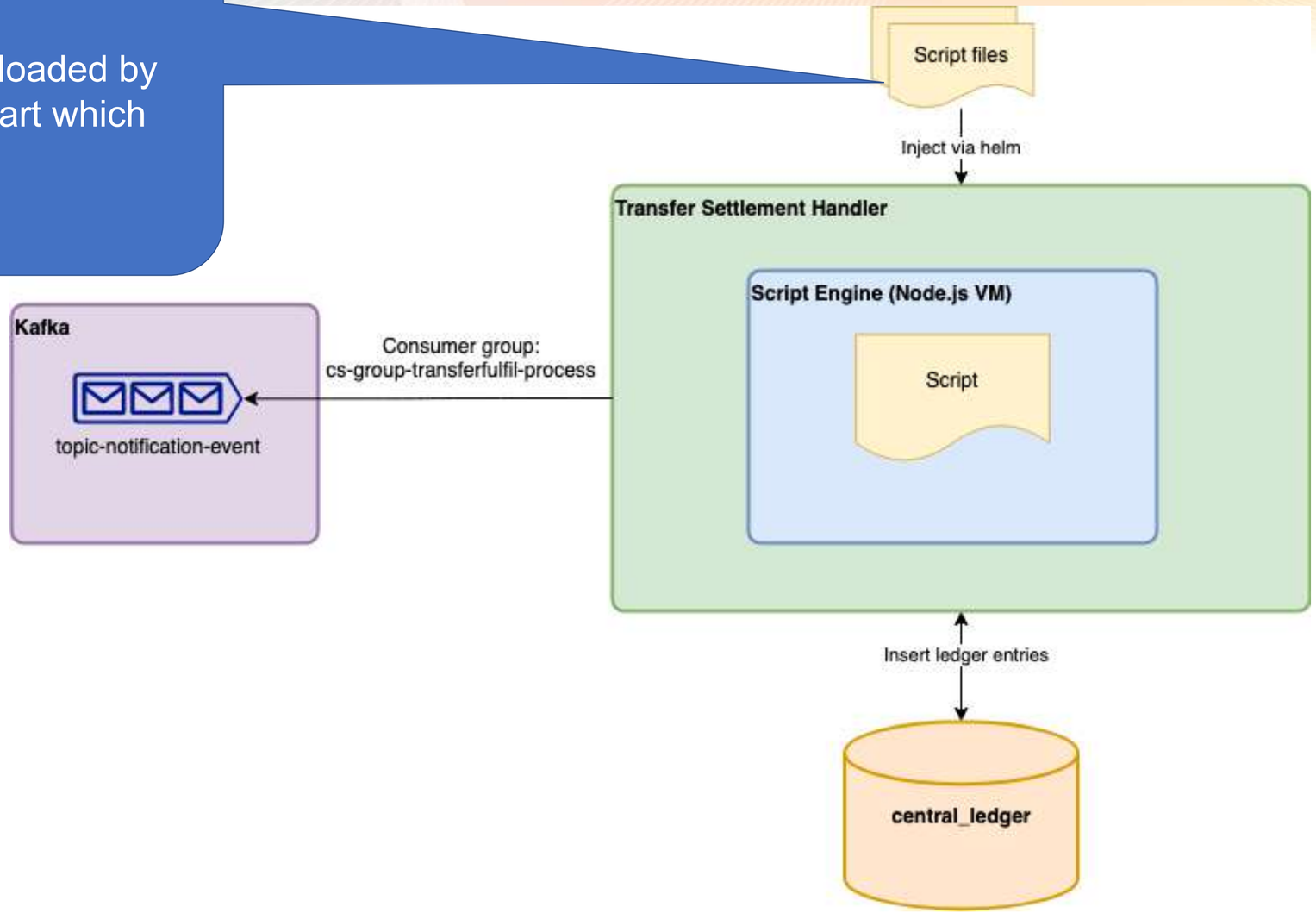
Connecting a rule to the system

The rules are contained in scripts which are executed in a separate Virtual Machine…



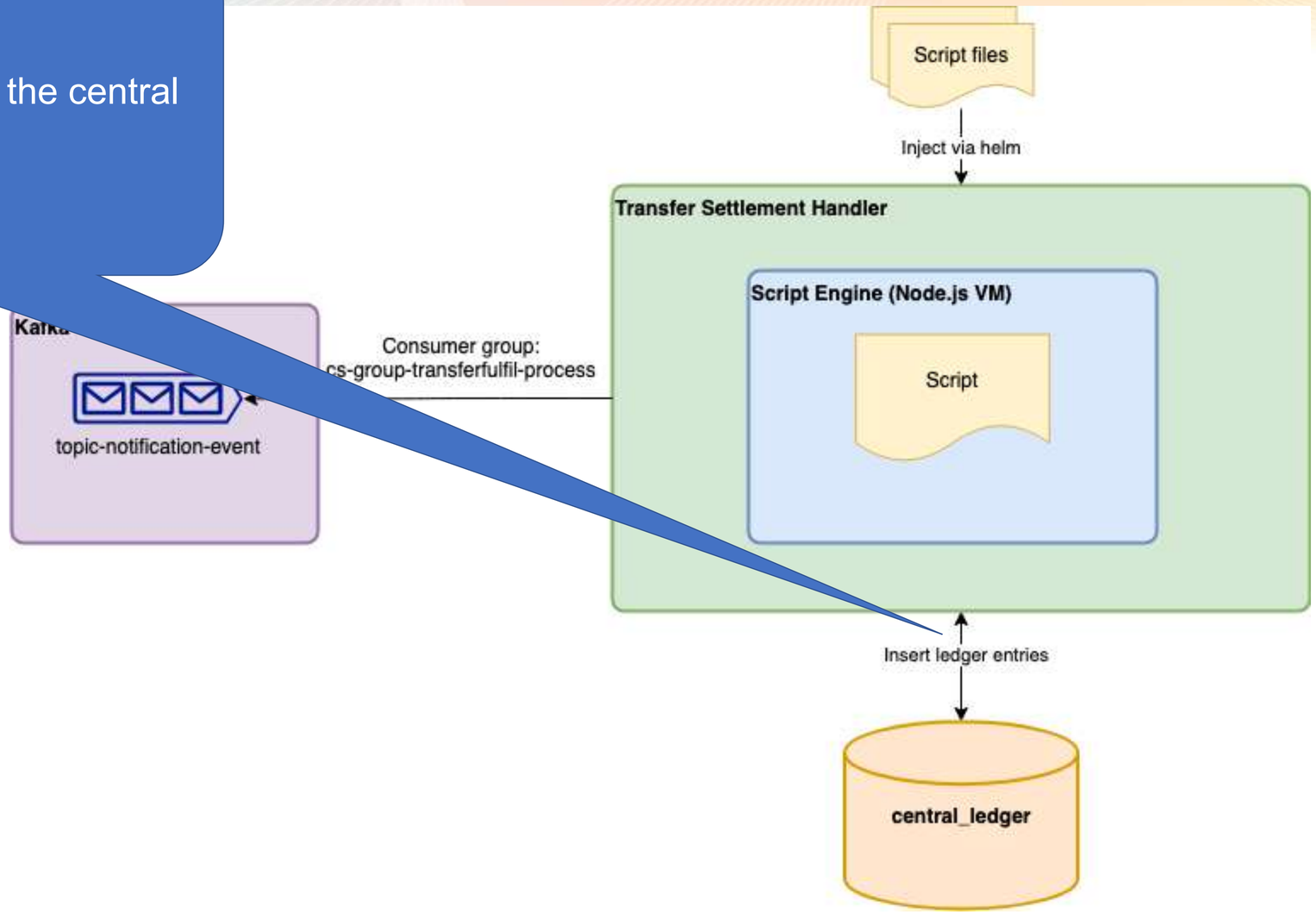
Connecting a rule to the system

New and changed rules are loaded by adding them to the Helm chart which defines the VM



Connecting a rule to the system

… and the rule scripts update the central ledger database



Drawing breath…

We have the following:

- A structure for defining a rule
  - Defining when the rule should fire
  - Defining the action to take when the rule fires
- A way of executing and updating rules
  - Rules listen to Kafka streams
  - They can be attached to multiple handlers
  - They run in Virtual Machines
  - They are loaded via Helm chart definitions
  - The functions available to a rule will depend on the handler in which it is executed.

The immediate future

- Adding execution locations as required
- Adding rule actions when required

Further down the line…

- Representing rules to administrators
- Defining a rules API to convert administrator rules into actually executing rules

What might a user interface look like?

- For example, a simple Excel-based one…

We start with some enumerations to map the underlying JSON types onto simpler text types…

Transaction	Transaction Reference
Transaction ID	Transaction.transactionId
Quote ID	Transaction.quoteId
Payee Identifier Type	Transaction.payee.partyIdInfo.partyIdType
Payee Identifier	Transaction.payee.partyIdInfo.partyIdentifier
Payee DFSP	Transaction.payee.partyIdInfo.fspId
Payee First Name	Transaction.payee.personalInfo.complexName.firstName
Payee Last Name	Transaction.payee.personalInfo.complexName.lastName
Payee Account Type	Transaction.extensionList["payeeAccountType"]
Payer Identifier Type	Transaction.payer.partyIdInfo.partyIdType
Payer Identifier	Transaction.payer.partyIdInfo.partyIdentifier
Payer DFSP	Transaction.payer.partyIdInfo.fspId
Payer First Name	Transaction.payer.personalInfo.complexName.firstName
Payer Last Name	Transaction.payer.personalInfo.complexName.lastName
Payer Account Type	Transaction.extensionList["payerAccountType"]
Amount	Transaction.amount.amount
Currency	Transaction.amount.currency
Transaction Type Scenario	Transaction.transactionType.scenario
Transaction Type Initiator	Transaction.transactionType.initiator
TransactionType Initiator Type	Transaction.transactionType.initiatorType
Note	Transaction.note

Result	Result Item
Journal entry type	result.ledgerEntryType
Currency	result.currency
Payer DFSP	result.payer.dfspId
Payer Amount	result.payer.amount
Payee DFSP	result.payee.dfspId
Payee amount	result.payee.amount

Test	Test symbol
Equals	EQ
Not equal to	NEQ
Greater than	GT
Less than	LT
Greater than or equal to	GE
Less than or equal to	LE
Case insensitive equals	EQCI
Case insensitive not equal	NECI

mojaloop

We build those up into rules with comprehensible names…

Name	Variable	Test	Value
Transaction is not an on-us transfer			
	Payee DFSP	Not equal to	Payer DFSP
Transaction is a wallet-to-wallet transfer			
	Payee Account Type	Case insensitive equals	"Wallet"
AND	Payer Account Type	Case insensitive equals	"Wallet"
Transaction is a P2P transaction			
	Transaction Type Scenario	Case insensitive equals	"TRANSFER"
AND	Transaction Type Initiator	Case insensitive equals	"PAYER"
AND	TransactionType Initiator Type	Case insensitive equals	"CONSUMER"

Then we make the whole thing into a comprehensive statement

Name	Connector	Test	Item	Reference	Value
Interchange fee test					
		Transaction is not an on-us transfer			
	AND	Transaction is a wallet-to-wallet transfer			
	AND	Transaction is a P2P transaction			
	THEN		Journal entry type		"INTERCHANGE_FEE"
			Currency	Currency	
			Payer DFSP	Payer DFSP	
			Payer Amount	Amount	*0.6%
			Payee DFSP	Payee DFSP	
			Payee amount	Amount	*0.6%*-1

And the converted result (input to the interface) is something like:

```
/*  Interchange fee test    */
if(
/*  Transaction is not an on-us transfer    */
(
(
Transaction.payee.partyIdInfo.fspId NEQ
Transaction.payer.partyIdInfo.fspId
)
)
AND
/*  Transaction is a wallet-to-wallet transfer  */
(
(
Transaction.payee.accountType EQCI "Wallet"
)
AND
(
Transaction.payer.accountType EQCI "Wallet"
)
)
```

```
AND
/*  Transaction is a P2P transaction    */
(
(
Transaction.transactionType.scenario EQCI "TRANSFER"
)
AND
(
Transaction.transactionType.initiator EQCI "PAYER"
)
AND
(
Transaction.transactionType.initiatorType EQCI
"CONSUMER"
)
)
)
{
};
```

mojaloop

But this is just a quick hack…

- You could do it much better and slicker, of course.
- But to get going, all you need to have is a clear syntax of how to construct a rule definition and a process for executing it.

Any questions?