PI 14 Reference Architecture Working Doc

Overall plan

- Phase 1 - Complete Ref Arch and Deliver Implementation Architecture
- Phase 2 - Build the foundation for V2
- Phase 3 - Consolidate and Release

Objectives of this PI:

- Agreed Reference Architecture
- Agreed Implementation Architecture
- Build all the breaking changes starting this PI, to release a beta with the final structure ASAP
- Build Plan for next PI

Relevant docs:

- This doc has all notes and will later serve as the main documentation source
- Miro board is: https://miro.com/app/board/uXjVPpKjqlw=/ (since 8th Deb 2023), original is https://miro.com/app/board/o9J_lJyA1TA=/?fromRedirect=1
- Google spreadsheet with build plans and interface matrix: https://docs.google.com/spreadsheets/d/1ITmAesHjRZICC0EUNV8vUVV8VDnKLjbSKu_dzhEa5Fw/edit#gid=1810993431
- PI14 Convening presentation slides: https://docs.google.com/presentation/d/1oZz8wwKVKK4dGSP-9yjMrMKBJovowBFndA7UOXToRkM/edit#slide=id.gd35b442fa0_0_192
- Michael Wagener proper version of the Ref Arch Documentation (to be replaced by VuePress doc site) - https://docs.google.com/document/d/1gmgJ3MnfaET3-mXM4OwzqIAMos_qQ4K7qtZxYifscHk/edit?usp=sharing
- VuePress documentation repository (Later we'll decide if we keep this separate from the main Mojaloop doc repo or merge this wth it): https://github.com/mojaloop/reference-architecture-doc
    - This is being automatically published here: https://ref-arch-docs.moja-lab.live/ (use this if that doesn't work http://mojaloop-ref-arch-docs.s3-website.eu-west-2.amazonaws.com/)
- Scalability and performance PoC report from Nov2020- https://docs.google.com/document/d/1fWSh1zu-I_BC8ki7gfDiNrz9sJiSMpiKG-xo0Kc56HQ/edit#heading=h.lnt20jsfjvz6
- Zenhub project board with all work items: https://github.com/mojaloop/project/pulls#workspaces/mojaloop-project-59edee71d1407922110cf083/board?epics=116650553_2482&filterLogic=any&repos=116650553

- Slack channel for this workstream: https://mojaloop.slack.com/archives/C01P7PFD9RN

Future repository for the published version of the Reference Architecture

Repo name: “reference-architecture-doc” -> here

Documentation tracker:

(20 Sept 2021 - this is not being used anymore, now we have zenhub: link at the top)

Bounded contexts (last updated 6th Sept):

- Missing BC's in the list
  - TBD
- Missing content in the docs
  - Security (critical for initial release)
  - Monitoring
  - Scheduling
- WIP
  - Notifications and alerts - Miguel started this 6 Set
  - Reporting - Nel started this 16 Sep
- Pending review
  - 3rd party (Lewis did the review, PR has some changes missing, Michael W working on it - posted updated draft 09/17/2021 - ready for review - 09/20/2021)
  - Participant Life cycle Management (In PR, needs review) - Nel did this by 6 Sep
  - Accounts and balances (review needed)
  - Settlements (review needed)
  - Account Lookup And Discovery BC (review needed)
  - Quoting/Agreements BC (review needed)
  - FSP Interop API - Nel did this by 14 Sep (In PR, needs review)
  - Auditing (critical for initial release) - Michael Wagener working on it - 09/20/2021
  -
- Done and reviewed
  - Transfers
  - Logging - Nel did this by 16 Sep (In PR, needs review, update: PR merged)

Phase 1 (active) plan:

1. Reference architecture

    a. Finish the main use cases of the main Bounded Contexts (happy path sequence diagrams)
        i. Transfer flow (reviewed 5May2021)
        ii. Participant Lifecycle flow (reviewed 5May2021)
        iii. Agreements (completed 10 May 2021)
        iv. Settlements (mostly complete in 11 May 2021)
        v. Liquidity cover (mostly complete in 11 May 2021)
        vi. Bulk Flow BC (mostly complete in 12 May)
        vii. Third Party-API (mostly complete in 12 May)
        viii. ~~Fee and Interchange Calculation BC~~ (not needed anymore / complete 17/18 May)
        ix. Transaction Request Flow (Payee initiated transfer Completed 19 May)
        x. Scheduling (discuss timeouts here)
    b. Detailed documentation of BC's interfaces, including non-happy path use cases
        i. We need an owner per Bounded Contexts
        ii. Document Bounded Context interfaces (and events)
        iii. Important to flag any issues that invalidate the overall Ref Arch
        iv. This task Depends on a) being completed
    c. Comms patterns (this can be done in parallel with b)
    d. Cross-cutting concerns (platform config, security,)
2. Implementation architecture
3. Resource discussion for build phase

Next up:

- Monday 17th
    - Fee and Interchange Calculation BC
    - Transaction Request Flow
    - Bulk Flow BC
    - Scheduling (discuss timeouts here)

TODOs:

- Clean up the transfers flow - remove the FRMS integration and put it somewhere else
- Link the use cases from the source docs to avoid duplication
    - FSPIOP - https://docs.mojaloop.io/mojaloop-specification/fspiop-api/documents/Use%20Cases.html#3-use-case-summaries
- Personal Identifiable Information storage (get requirements and decide method)
- Platform monitoring and health checking (WIP)
    - Define service levels (SLA's, SLO's and SLI's)
    - Complete the platform automation section
    - Distributed tracing
- Yes they are, solved - ~~Confirm the transactions are simple as documented - simple request to pay that forwards the transaction request to the Payer DFSP, and everything else is "just" a quote and a transfer~~
- 30 Jun - Discussion: Relationship between settlements, transfers and liquidity checks
    - Problem is where and how to decide which accounts to credit/debit when executing a transfer depending on the settlement model.
    - Solution:
        - A settlements BC event handler will listen to the "TransferPrepareRequested" event, decide which accounts should be used (based on settlements config and logic) and send out an event that has the data in the original event plus the accounts to use;
        - this 2nd event will be then used by the transfers BC as the starting point;
        - The accounts map is persisted with the transfer state so the fulfil leg can also use the predetermined accounts
        - This solution maintains separation of concerns, as all settlement logic stays in settlements BC AND is performant, because the settlements event handler requires only read data to make decisions and can be scaled without big partitioning concerns.
- Endpoint simplification - can we have this in the spec? A CCB discussion is required - see the TODOs on Participants section
- Liquidity Threshold Warning Notifications
    - Should be moved to the Accounts & Balances BC
    - Threshold warning events will be picked up by the Participant BC and notification would be carried out from there.
- Throttle participant transaction amounts
    - Each bounded context needs to have Scheme Rules that define the throttle limits and send out an event based on the trespass, other BCs

will have rules based on this event and perform actions based on that event E.g.
        - Notifications BC - Send Notification
        - Participant Life Cycle BC - Change the state of a participant
- Forensic Logging
    - The “audit” operations that we are doing should be renamed to “Forensic Logging”? ← This is to avoid confusion between the Org “Audit” function and the capability to capture crypto-signed events that cannot be tampered.
    - Should “Audit BC” be a new separated BC?
- Account closing
    - Who closes them?
    - When can we close them? ← only when they are at 0?
    - What happens to their existing balances?

Bounded Contexts Ownership

Updated list in table format: Mojaloop 2.0 Reference Architecture

Short list:

- Transfer & Transaction - Miguel (moved from Greg)
- Accounts & Balances - Jason (moved from Adrian)
- Participant Life-cycle Management - Sam/Nel (moved from Greg)
- Account Lookup and Discovery - Michael
- Agreements/Quotes – Sam/Miguel
- Settlements – Michael
- 3rd Party Initiated Payments - Lewis
- Notifications & Alerts - Miguel
- Scheduling - Justus
- Reporting - ?? Pedro Temp
- DEPRECATED - Bulk - Sam
- DEPRECATED - Liquidity Cover Monitoring & Settlement Risk Management - Michael

Transfer & Transaction

Owner(s):

- Miguel
- (was Greg)

Team:

- Justus
- Godfrey
- ???

Participant Life-cycle Management

Owner(s):
- Johann Nel, Greg (confirmed)
- Sam

Team:

- Jean-Pierre Neil (JP), Johan Nel (Nel), Len, Roland
- Sam, Dorota, Paul Makin, Paul Baker
- Godfrey

Account Lookup and Discovery

Owner(s):
- Michael (confirmed)

Team:
- Godfrey
- Miguel
- ???

Settlements

Owner(s):
- Michael (confirmed)

Team:
- Godfrey
- Justus
- Valentin ???

What about settlement on demand?

Liquidity Cover Monitoring & Settlement Risk Management

Owner(s):
- Michael (confirmed)

Team:
- Godfrey
- ???

Bulk

Owner(s):
- Sam (confirmed)

Team:
- Paul M
- Michael R
- Greg M
- Istvan M

Notifications & Alerts

Owner(s):
- Miguel (confirmed)

Team:
- ???

Scheduling

Owner(s):
- Justus (confirmed)
- Pedro (confirmed)

Team:
- ???

Agreements/Quotes

Owner(s):
- Miguel (confirmed)
- Sam (confirmed)

Team:
- Miguel
- Godfrey
- Someone from Sybrin and Coil???
- Someone from Mowali

3rd Party Initiated Payments

Owner(s):
- Lewis

Team:

- Godfrey
- Michael

Accounts & Balances

Owner(s):

- Jason Bruwer (Confirmed)

Team:

- Pedro Barreto
- Concalo Garcia

Bounded Context breakout sessions Objectives:

- Starting with the existing use diagrams, document the non-happy path use cases and sequence flows
- Document Bounded Context interfaces:
    - What this BC provides - Endpoints provided and events published by “this” BC
    - What this BC needs - Endpoints, RPC calls, events consumed by “this” BC
    - This is very high level, callout what is important, no need to detail structures.
- Important to flag any issues that invalidate the overall Ref Arch, especially the bounded context structure (no logic should be implemented by multiple BCs)

Ways of work:

- Specific use case discussions should preferably happen at the same schedule as the normal Reference Architecture Sessions, using the zoom breakout room functionality.
- The last hour of each session will be used for “Use Case” owners to report progress - timeboxed to 5 mins each.

Use cases

Ref:
https://docs.mojaloop.io/mojaloop-specification/documents/Generic%20Transaction%20Patterns.html

Transfer flow discussion

- Transfers BC doesn't know or deal with accounts or balances - if it ever needs this info it asks the Accounts and Balances BC
- Transfers BC controls transfer timeouts
- Where in the process should signatures and conditions be checked?
    - Message Signatures (JWS) in the ext API part (FSPIOP[1] or ISO, etc.)
    - ILP Condition check is performed by the Transfers BC

NOTE: there is a Settlements event handler between TransferPrepareRequested and the Transfers BC initial step

[1] FSPIOP = Financial Services Provider Interoperability Specification

Settlements (2022 version)

Use cases:

1. Assign transfer to settlement batch (a set if transfer characteristics defined in a settlement model)
    - Instead of assigning a transfer to the current open settlement window, the Settlement vNext would be responsible for allocating the transfer itself
    - Transfers-BC / Central-Ledger: At time of fulfil, produce an event to be consumed eventually by Settlement v3
    - Settlement-BC would then be responsible for allocating a transfer to a settlement batch and settlement model, independently from other components
2. Produce a settlement matrix based on settlement criteria:
    - Retrieve CLOSED settlement batches based on supported settlement models for a specific time period (including other such as currency, channel etc.)
    - Producing the settlement matrix based on the accumulated batches
    - Closing settlement batches are driven by generating the settlement matrix
    - Late settlement transactions will be allocated to a newly created batch, but how do we close those batches?
3. Return settlement model based on a Transfer
4. Record deposit to settlement account
    - TODO
5. Record withdrawal from settlement account
    - TODO

Central-Services Touch Points

1. Settlement model logic will be contained within a library
2. Settlement Account Creation
    a. There will not be participant settlement accounts created, those will remain in Central-Ledger
    b. Settlement batch accounts however will be created in Settlement v3
    c. The settlement batch account is derived from the a Batch (*only a single batch 1..1*), Participant & Currency and created as transfers are allocated to settlement batches
3. Transfer Reservation / Prepare
    a. Settlement models are cached by the Transfer-BC / Central-Ledger (liquidity handler)
    b. A new liquidity handler will be developed in order to perform the liquidity check
4. Transfer Completion / Fulfilment
    a. When Settlement v3 is enabled, the fulfilment event will be consumed by the Settlement v3. The transfer will not be allocated to a settlement window by Central-Ledger.

    b. The necessary account and balance journal entries will be created to debit the position (immediate gross settlement) back to Central-Ledger ????. This can be achieved through either an event/api call.
    c. What goes into the event back to Central-Ledger:
        i. transferId
        ii. Credit account
        iii. Debit account
5. An exposed API from Settlement v3 would need to:
    a. Create a settlement matrix from inputs;
        i. Date range
        ii. Settlement model
6. Reporting:
    a. Same settlement API exposed for reporting
    b. List transfers for a settlement batch
    c. List settlement accounts for a settlement batch

High Level Requirements

1.

Settlements (original from v1 ref arch)

Use cases:

1. Assign transfer to settlement batch
2. Return contents of settlement batch(es)
3. Mark batch as settling
4. Settle
5. Manually close batch
6. Record deposit to settlement account
7. Record withdrawal from settlement account

Interfaces

Consumes

1. Information on completed transfer (TransferCommittedFulfiled from transfers BC)
2. Definition of settlement models (from Platform configuration BC)
3. Document content (from API BC)
4. Request for information on settlement batch(es)
5. Settlement requests.
6. Batch closure request.
7. Request to record deposit to settlement account.
8. Request to record withdrawal from settlement account.

Emits

1. Update batch content (AnB Create Account and Insert Journal entry)
2. Content of settlement batch(es)
3. Settle (AnB Insert Journal Entry)
4. Record deposit (AnB Insert Journal Entry)
5. Record withdrawal (AnB Insert Journal Entry)

TODO:

-

19 Oct 2021 - settlements change to use batches instead of windows

Notable differences:

- No external creation of batches (windows); batches are created automatically when transfer fulfilled events arrive (according to the configured per batch time span)
- Batches include properties according to:
    - the settlement model they belong to
    - optional settlement model data (ex: in the case of FX, the currency pair)
    - the start date of the batch
    - open/closed flag
- There can be multiple similar batches for the same model, model detail and start time, however, only one should be open

Todo:

- Cross-cutting throttling events

Transfers/Transactions BC

TODO:

- ~~**Discuss the relationship between Accounts and Participants. Can we use Accounts to validate participants or must we validate participants similar to the Quoting Service use-cases?** ← this may impact most of the use-case flows.~~
- ~~**Follow-up discussion on the "Single Writer Principle" design pattern, i.e. combine the Transfers BC and Accounts BC.**~~

- ~~Discuss where Fulfilment Conditions are validated~~
- Continue working on Non-happy paths
- Document Interfaces (provides + needs)
- Design/Document canonical model for Transfers
- Discuss bulk trasfers
- ...
- ~~How do we handle participants being disabled?~~ ← We validate participants from the Participant BC
    - ~~What happens if the transfers BC validation check on the Fulfil leg fails due to this?~~ ← All transfers that are already “prepared” should continue, all new transfers coming in should not be accepted.

Notes

- ..

Use cases:

- Perform Transfers
- Perform Transfers with Payee Confirmation -
- Query (GET) Transfer -
- Perform Transfer - Duplicate POST (Resend) -
- Perform Transfer - Duplicate POST (Ignore) -

Non-happy path variations:

- Perform Transfer - PayeeFSP Rejects Transfer -
- Perform Transfer - Timeout (Prepare) -
- Perform Transfer - Timeout (Pre-Committed) -
- Perform Transfer - Timeout (Post-Committed) -
- Perform Transfer - Duplicate POST (None Matching) -
- Perform Transfer - Payer FSP Insufficient Liquidity -
- Perform Transfer - Transfer Prepare Validation Failure (Invalid Payer Participant) -
- Perform Transfer - Transfer Prepare Validation Failure (Invalid Payee Participant) -
- Query (GET) Transfer - Validation Failure (Invalid Payer Participant)
- Query (GET) Transfer - Validation Failure (Invalid Payee Participant)
- Query (GET) Transfer - Validation Failure (Transfer Identifier Not Found)

Conclusions:

- Payer FSP should not be allowed to unilaterally time-out a transfer (irrespective of the expiration time) but respect the Switch’s decision (i.e. wait for notification)
- Validation of crypto condition & fulfilment would be handled by the Transfers BC as it is a fundamental aspect of the “transfer” itself (i.e. not specific to the FSPIOP “language”).

- Transfers BC will follow the same validation pattern as used by the Quoting and Party BC to validate Participants, as the ability of an Account to transact or if a Participant is enabled is mutually exclusive.
- Transfers BC is the “source of truth” for Transfers, and is responsible for persisting the Transfer’s respective state/s.
- Disabling participants should not hinder the processing of transfers that are already in a “prepared” state, only new transfers being received by the Transfers BC via TransferPrepareAccountAllocated events should not be accepted.

Interfaces:

What this BC provides:

- Async Events Produced:
    - TransferPrepared -
    - TransferPrepareAccepted -
    - TransferCommittedFulfiled -
    - TransferReservedFulfiled -
    - TransferQueryResponse -
    - TransferRejectRequestProcessed -
    - TransferPrepareRequestTimedout -
    - TransferFulfilCommitRequestTimedout -
    - TransferPrepareDuplicateCheckFailed -
    - TransferPrepareLiquidityCheckFailed -
    - TransferPrepareRequestRejected -
    - TransferRejectRequestProcessed -
    - TransferPrepareInvalidPayerCheckFailed -
    - TransferPrepareInvalidPayeeCheckFailed -

What this BC needs:

- Async Events Consumed:
    - TransferPrepareAccountAllocated - Settlement BC -
    - TransferPrepareCallbackReport - Interop API BC -
    - TransferFulfilCommittedRequested - Interop API BC -
    - TransferFulfilCommittedCallbackReport - Interop API BC -
    - TransferFulfilReservedRequested - Interop API BC -
    - TransferFulfilReservedCommittedCallbackReport - Interop API BC -
    - TransferQueryReceived - Interop API BC -
    - TransferQueryResponseCallbackReport - Interop API BC -
    - TransferRejectRequested - Interop API BC -
    - TransferRejectRequestProcessedCallbackReport - Interop API BC -
    - TransferPrepareRequestTimedoutCallbackReport - Interop API BC -
    - TransferFulfilCommitRequestTimedoutCallbackReport - Interop API BC -
    - TransferPrepareDuplicateCheckFailedCallbackReport - Interop API BC -
    - TransferPrepareLiquidityCheckFailedCallbackReport - Interop API BC -
    - TransferRejectRequestProcessedCallbackReport - Interop API BC

        - TransferPrepareInvalidPayerCheckFailedCallbackReport - Interop API BC-
        - TransferPrepareInvalidPayeeCheckFailedCallbackReport - Interop API BC-
- Sync API Requests
    - CreateLedgerEntryPair (i.e. Debig, Credit) - Accounts & Balances BC
    - GetParticipantsTransfersData - Participant Lifecycle Management BC -
    - GetParticipantNotificationData - Admin API BC

What is the canonical model:

- Transfer
    - transferId
    - transferType
    - quoteId (optional)
    - settlementModelId
    - Participants
        - Payer
            - participantId
            - Accounts
                - Debit
                    - accountId
                    - accountType
                    - currency
                - Credit
                    - accountId
                    - accountType
                    - currency
        - Payee
            - participantId
            - Accounts
                - Debit
                    - accountId
                    - accountType
                    - currency
                - Credit
                    - accountId
                    - accountType
                    - currency
    - Amount (amount to transfer)
        - value (number)
        - currency (ISO currency code)
    - expiration (ISO dateTime)
    - ilpPacket
    - Extensions

What is the Bulk canonical model:

- Transfers

- bulkId
- bulkQuoteId
- Transfers[]
    - Transfer* (see above)
- ~~expiration (ISO dateTime)~~
- Extensions

Quoting/Agreements

TODO:

- ~~Decide if this should be a specific request to validate and return a bool response, OR if it should return a status flag and the Agreements BC can make the decision~~ - decided, we need more participant information, so a bespoke request is required
- Discuss Bulk Quotes
- On the normal “Calculate Quote” we need to map the scenario here the a fully duplicated request ina final state is considered a get request
    - This will be discussed at DA level - DA to propose a solution

Notes

- If/when we need to implement rules that depend on the information about party account limits (volume or quantity), we can add an additional component that listens to the transfer events, calculates these limits and sends out events (or makes this info available for quick query)
- Initial happy path mapped on 10 May 2021

Use cases:

- Calculate quote
- Get quote

Non-happy path variations:

- Calculate Quote - Invalid Quote Request
- Calculate Quote - Invalid FSPs
- Calculate Quote - Invalid Scheme Rules detected in request
- Calculate Quote - Invalid Scheme Rules detected in response
- Calculate Quote - Timeout
- Are there more?

Conclusions:

- No red flags regarding overall BC and Ref arch design
- Need to understand/clarify the get via post pattern:
    - Should a get be a simple RESTful get or do we need to support the get from duplicate posts?
    - Do we need to serve get requests that include FSP later,

Interfaces:

What this BC provides:

- Events this BC publishes:
    - QuoteRequestAccepted
    - QuoteResponseAccepted
    - QuoteQueryResponse
    - InvalidQuoteRequest
    - InvalidFSPInQuoteRequest
    - RuleViolatedInQuoteRequest
    - RuleViolatedInQuoteResponse
- Other endpoints/calls we provide:
    - (Nothing mapped now)

What this BC needs:

- Events this BC consumes:
    - QuoteRequestReceived
        - Quote Id
        - Transaction Id
        - Participant info (per participant)
            -
    - QuoteReceivedCallbackReport
    - QuoteResponseReceived
    - QuoteQueryReceived
- Other endpoints/calls that we consume:
    - GetParticipantsQuoteData - this an RPC call provided by the Participants BC, should return participant data that is relevant for the Quoting BC and the status of the participant (enum?)
        - Input params: array of participant IDs
        - Returns: return relevant participant data, including status flags
    - Query Participant Notification Callbacks URIs - this call should return the participant callback endpoint, and is **not required by the Quoting BC**, but by the Notifications BC

What is the canonical Quote (what we store in the quoting BC):

- Quote ID
- Transaction ID
- Participants
    - payerId
    - payeeId
- Payer
    - Participant
        - participantId
        - roleType ← payer
    - Amount request (initial amount)

        - Value (number)
        - Currency (ISO currency code)
    - Amount to send (including fees, etc)
        - Value (number)
        - Currency (ISO currency code)
- Payee(s) (one or more - should all be added up to the “Amount to send”)
    - #
        - Participant
            - participantId
            - roleType ← identify why this “payee” is receiving this amount, i.e. fee, recipient, etc
        - reason
        - Amount to receive
            - value (number)
            - currency (ISO currency code)
- Extesions

Liquidity Cover Monitoring & Settlement Risk Management BC

NOTE: Not needed anymore, its functions are performed by the Participant Lifecycle Management BC - revisit if we need use cases that don’t belong or make sense in the participants BC

Main use cases

- Participant deposit or withdrawal
- Threshold liquidity definition (what can be used as liquidity cover) -> covered with the “reservations” use cases

Bulk

Important note: We don’t need a Bulk Bounded Context, we have decided to split this in two parts:

- **Interop APIs** - which forward the request using the internal events (canonical) t the destination BC (agreements, transfers, etc)
- **Target BC** - that knows how to handle the bulk canonical requests, when necessary do the split and join operations so per target DFSP bulk requests can be sent. The target BC is also responsible to interpret any all/nothing flags that come in the original request.

Main use cases:

- Bulk quotes (done 12th May)
- Bulk transfers (with or without scheduling) (done 12th May)
- All or nothing bulk transfers (with or without scheduling)
    - same as normal bulk transfers (needs property/flag)
- Bulk lookup??
    - Do we need to support this use case?
        - There is demand for this, we should support it in the next version
- Bulk association of parties to participants (DFSP)

-

This design allows the target BC to make all decisions regarding their requests.

OLD design (pre 9Jun2021):
~~**Bulk request splitter and joiner that is inside the Interop API BC** (one per language FSPIOP/ISO) and takes care of splitting the external bulk requests into internal per target DFSP bulk requests that are sent internally to the correspondent BC; later keeps track of all responses and once all responses are received sends the final single response to the original requester.~~

- ~~Per target DFSP bulk implementation that is provided by any BC that requires bulk processing - agreements and transfers (maybe later lookups)~~

~~This design pushes the split/join tasks to the Interop API's, where specific knowledge about the requests exists, and allows the other BC's to handle their own bulk use cases, again keeping specific domain knowledge inside each BC.~~

(*Note: Check ISO20022 ability to support bulk requests)*

Bulk Quotes Use Case

The design we're proposing assumes that internally we support multiple destinations, using the following structure:

```
Parent quote request (proposed)
- source dfsp (only one of this)
        - quotes (many)
        - dest dfsp
        - rest of quote data
```

TODO:

- 3rd party BC requires bulk lookup

Third Party Initiated Payments BC

Main use cases:

- Link Accounts **(done 13 May)**
- Unlink from DFSP side + PISP side (can be captured in one sequence - updated 28th June,)
- Third Party Payment Initiation (done)
- **(done 13 May)**
- DFSP paying a PISP **(Done 29th June)**
    - *2 main options are available, neither of which impact the ref architecture*

- Bulk version of Request to pay (PISP requesting DFS to pay) (started 22 June - updated 28th June, Lewis to present to team)
- DFSP Authorized Thirdparty Payment **(done 29th June)**
    - (ie. not using the on-device FIDO credential)
    - Uses the existing Third Party Payment Initiation - just a slight change to the verification step. It won't affect the reference architecture.

- PISP as a marketplace
    - Paying to a PISP use case mostly gets us there
    - We are stuck on how PISPs can deduct fees without disclosing them to the Payer or Payee
    - **blocked by split payments**

TODO:

- Help with names of messages… I just made some up
- For bulk cases, how does the Authz+Authn work? What will be signed?
- Pick up conversation about Split Payments in Bulk
- Talk about consent registration and 3p tx request timeouts - can they be enforced by the DFSP? Is that fine?

Notes

- Good with DFSP paying a PISP use case. Architecture holds up. A few quirks to be ironed out. Assumptions:
    - more responsibility on linked DFSP
    - is it ok for the Payer DFSP to know about the Payee DFSP?
- Error scenarios for Paying to a PISP - if the tx doesn't continue, no notification needs to be sent to the PISP (depending on the above assumptions)
- *DFSP Authorized Thirdparty Payment* - seems fine
- *PISP as a marketplace* design needs more thought and work… nothing currently is an issue, but we need more thinking

Non-happy path variations:

- Link Accounts - account discovery failed
- Link Accounts - DFSP rejects consentRequest
- Link Accounts - DFSP rejects entered authToken or OTP after out of band user auth
- Link Accounts - Credential Registration Error
- Unlink Accounts - can't find CONSENT
- Unlink Accounts - Downstream auth service error
- Transfers - Bad Payee Lookup
- Transfers - Bad thirdparty Transaction request
- Transfers - Downstream FSPIOP Error (e.g. in quotes or transfers)
- Transfers - Bad signed challenge
- Transfers - User rejected authorization
- Transfers - Thirdparty Transaction Request Timeout

Conclusions:

- Added a new BC to separate out the 3P API messages from the canonical Mojaloop messages for 3rd party
- Most of the 3P use cases are wrappers around DFSPs using the FSPIOP API, so there's not much work being done at the centre
    - It's just messages being passed around from one participant to another
- The only other dependencies this BC has are upon Parties + Participants, and Participant Lifecycle Management

Interfaces:

What this BC provides:

- Events this BC publishes:
    - [TODO - waiting to discuss event names with the team]
- Other endpoints/calls we provide:
    - None

What this BC needs:

- Events this BC consumes:
    - [TODO - waiting to discuss event names with the team]

- Other endpoints/calls that we consume:
    - Query Participant Notification Callbacks URIs - this call should return the participant callback endpoint

References *(not explicitly used by this BC, but referred to by DFSPs and PISPs)*

- GET /parties (Thirdparty-API implementation?)
- Party/Participant Associate

- Party/Participant Disassociate
- POST /quotes
- POST /transfers
- Bulk GET /parties (Thirdparty-API implementation?)
- POST /bulkQuotes
- POST /bulkTransfers

Fee and Interchange Calculation BC

Not needed anymore

Main use cases:

- Fee calculation - this happens in the other flows: quote, transfer or settlement
- Input configuration of the calculation, which can include dynamic rules
- Query the system for applied fees (this is more of a reporting feature than a transactional one)

Participant Lifecycle Management

TODOs:

- Discuss Global Configurations,
    - Criticality defines who decides the default values
        - E.g BC assumes default config or throw error that config is not set
    - Review configurations section
- Operations API – should be included in all the flows and diagrams.
- Should we need the update position use case?
    - Is it relevant or do we disregard this and document the correct procedure for **recon/refund.**
- Update Endpoints – Update architecture to use a single hostname and port registration, **the routes/paths will be defined in the documentation including the liveness check**.

Notes:

- All write operations must have maker/checker protection (only read only ops can forgo maker/checker)
- A participant cannot have more than 1 account of the same type/currency pair
- Resolved - Do we need the *Liquidity Cover Monitoring & Settlement Risk Management BC*, looks like its functions are per participant and could be served by the participant lifecycle management BC?
- For now we removed the Liquidity BC, all its use cases are now handled by the participants - revisit if needed

Main use cases:

- Create Participant (including initial account, to be sent to the Accounts & balances BC)
    - Could we possibly merge update endpoints as an optional flow if the data is provided?
    - Show single-step registration flow - define the order & flow
    - Single-step registration - Consider the failure flows - map required vs optional steps and data
    - Single Step Registration
- Add accounts (type/currencies)
    - We need to do a lookup to determine what currencies are allowed by the switch
    -
- Disable / Enable accounts
    - Should behave similar to the below TODO: JoNel
- Disable / Enable Participant
    - Should be called Update Participant State
        - State management should be defined - What states a participant can move to-and-from TODO: JoNel
        - States should be defined at bootstrapping and the configuration for validation and application of the state change.
        - Default (System) States should not be defined, all states are dictated by the scheme rules.
        - Roles should be mapped to the Participant State, as a consequence the restrictions are enforced by those roles.
    - The request should take additional parameters to allow two scenarios:
        - Should fully shutdown the participant and not allow any features, regardless of the config that was setup
        - Should only disable some features based on the config provided at bootstrap – This should probably be done using scheme rules, to disable a participant could be just to assign scheme rules to that participant. Thus this use ase is complete shut down rather than mixing logic
    - We need to notify the relevant participant that they were disabled/enabled
    - We are limiting only some features of the participant by allowing the operator to design what features and actions should be allowed.
        - The limitations should be defined by the HUB operator when defining the config on the bootstrapping phase
- Update Endpoints (Callback URL)
    - We should probably do a liveliness check of some sort to ensure the endpoints are reachable
    - Single endpoint for all FSP-IOP callbacks, all the callbacks should be defined in the specification including health check, we need to go to CCB to add the health check

    - Remove duplicate checks, there should be warnings and the certs should throw an error if it is invalid.
- Update Limits (NDC) and thresholds for warnings
    - Should this be in this BC? Or should it be moved to Settlements or Accounts and Balances
        - If the settlement models need to be taken into account for the threshold notification then it should form part of the Settlements BC
        - If it is purely on liquidity per currency then it should form part of the Accounts and Balances BC
        - If Participant BC needs to maintain this it should listen to events from the A&B BC or Settlement BC when there are liquidity changes. The events in this case should then be defined. Known Issues with this approach:
            - Other BC's need to know what the threshold limits are and only send notifications when breached
            - Other BC should send the information regarding liquidity with the event and then the Participant BC checks that information on every event to determine that the threshold was exceeded and if it was then send the notification.
    - Is there a possible maximum/ minimum threshold period?
    - We should be able to configure a threshold and action(s) based on the scheme rules.
    - Threshold exceed warnings should be a system notification (not to be confused with a system event notification), so the BC needs to provide the notification details, additional notifications (based on system events) configured should remain within the notifications BC and it would react on those events separately (asynchronously)
- Update Position
    - We will **no longer support** this use case
- FundsIn / FundsOut (deposits and withdrawals)
    - There should be an external reference ID & Source
    - Should there be any business rules applied to limit maximum and minimum values - Scheme Rules (Platform Config)
    - How do we capture any fees associated with the deposit/withdrawal?
        - We should discuss whether this is a valid transfer or just an artificial accounting entry
            - Decision: It should purly be a artificial Accounts and Balances BC update on the account
        - We know fees could play a role here and we need to determine when/where this will be done
            - Fees should be defined by Scheme Rules that stipulate an action i.e. a new transaction that would be the fees for the deposit/withdraw
- Liquidity Cover Reservations
    - Convert flow on miro to mirror checker/maker process
    - Absolute amount, never changing
        - Insurance taken on an amount
    - % Should be fixed from the time of handling the request

- Liquidity Cover Queries
- Get Participant – Get participant of the party in question
- Query Participants (search with criteria for operators) – Get information on a particular participant by providing identifier information.
- Listen to system events and execute actions based on rules:
    - ALS Throttle Event
    - Quote Throttle Event
    - Transfers Throttle Event
    - GetParticipantData Event

Negative Use Cases:

- General - Error handling upon approvals/Depen dency BCs failed
- General - Invalid Permissions
- General - Invalid Requests
- General - Approval gets rejected with reason
- General - Maker/Checker process retry count - To be discussed
- Create Participant - Duplicate Participant/Already Exists
- Deposit/Withdraw Funds - Duplicate Request based on linked Tran ID, send notifications, and adds additional approval
    - Pending parked discussion
- Deposit/Withdraw Funds - Withdraw, amount more than available amount
- Deposit/Withdraw Funds - Limits based on business rules
    - E.g. Switch owners have a max amount withdraw-able in a given day?
    - Both Participant and HUB operator should be defined in scheme rules
-
- Enable/Disable Participant - Disable of participant while transactions are still awaiting settlement
    - Enable/Disable Accounts - Flow also required
    - Settlement should continue with transactions that have been approved but future transactions should be halted
- Notify Threshold Exceed - - Repeated reset on limit notifications, could cause notification overload
    - E.g. The $1 see-saw effect
    - We should take into account the time frame of how long ago this notification was already sent before
    - Configuration setting on how long we should wait to resend a notification
- Add Accounts - Duplicate Account
- Add Accounts - Invalid Currency Provided based on business rules
- Update Threshold - Threshold lower than minimum business requirement
- Update Position - No references were provided - Reject Action
- Update Position - Request exceeds maximum update amount allowed
- Update Endpoints - Failed to do liveness check(s)

Conclusions

- Update Position – Flow has been disregarded

- Maker/Checker Operations - Retry count have **no** effect on the way that we process/re-process requests.

Scheme Rules

Below listed are scheme rules that would be required for this BC, the rules themselves will be sourced through different means i.e. Platform Configuration or Privileges

- Available Currencies – Loaded from Platform Configuration BC
- Limit Thresholds - To be determined
- Deposit/WithdrawFundsAmountLimit
    - Limits should stipulate:
        - % or static amount
        - Date related restrictions i.e.
            - Daily
            - Weekly
            - Monthly
            - Annually
            - Weekdays
            - Public Holidays
- Participant State & Role mappings
    - Assign Roles to States
    - States dictates roles to a participant when that Participant is in that state
- Threshold event configuration
    - Could cause a change to the Participant State

Interfaces:

- Events Provided:
    - ParticipantCreated
    - ParticipantAccountCreated
    - ParticipantStateChanged
    - Participant
- Events Consumed:
    - AccountLookupThrottleLimitExceeded
    - QuoteThrottleLimitExceeded
    - TranserThrottleLimitExceeded
    - GetParticipantData
- Admin/Operational API Provided:
    - ParticipantsPendingApproval
    -
- Admin/Operational API Consumed:
    - CreateParticipant
    - ApproveParticipant

What is a canonical Participant

- Participant
  - id
  - participantAlias
  - endpointURL
  - state
  - Accounts
    - accountID
    - ledgerAccountType
    - accountCurrency
    - isActive
    - warningThreshold
    - limit
      - type
      - value

What do we need:

Accounts and Balances

Notes

- Accounts and Balances BC is the “central ledger” for the system.
- Primary interactions are with the transfers and settlements BCs
- The accounts & balances BC is a directed sub-system. It is a dependency of the transfers and settlements BCs which use it as a system of record for their financial accounting
- There is no Mojaloop business logic in the accounts and balances BC. The only logic is to ensure that when an external BC attempts to create new journal entries these do not cause an account balance to exceed its limits).

Use cases:

- Create Account
- Close Account
- Query Account
- Insert Journal Entry

Non-happy path variations:

- Create Account - Invalid Parameters
- Create Account - Duplicate
- Close Account - Unknown Account
- Close Account - Invalid State?
- Query Account - Unknown Account
- Insert Journal Entry - Unknown Debit Account
- Insert Journal Entry - Unknown Credit Account
- Insert Journal Entry - Invalid Debit Account Currency
- Insert Journal Entry - Invalid Credit Account Currency
- Insert Journal Entry - Credit Account Limit Exceeded
- Insert Journal Entry - Debit Account Limit Exceeded

Interfaces:

Link to TigerBeetle client interface
Commands:

- CreateAccount
    - Create a new account in the ledger
- CloseAccount
    - Close an account in the ledger and prevent new journal entries impacting it
    - Drain balance to another account atomically?
- QueryAccount
    - Get the current state of the account
- InsertJournalEntry
    - Insert a new journal entry into the ledger specifying the debit and credit accounts
    - Respond with changed balance

Comms patterns

These should apply to communications to and from the outside and the switch; and communications between Bounded Contexts.
Intra Bounded-context communications don't have to follow these rules, still have to be discussed and approved by the DA.

Classification as to external access (maybe one way??)

- External comms - ports available to the outside of the switch
- Internal comms - ports not available to the outside of the switch

Classification as to pattern:

- Async
    - Async/Ack - No response expected, just ack
    - Async/Callback - Response will be received later using a different connection (assumes some form of ack to the initial send)
- Sync (with or without resp)
    - Response will be received immediately using the same "connection" (doesn't need to be a data response, can be only an ack)

Classification as to medium:

- Event based
    - Using Kafka Topics
- RPC based (remote procedure call)
    - HTTP/Rest
    - gRPC/Protobuf

Classification as to write guarantees of messages to the messaging component:

- Critical - requires the minimum write guarantee to acknowledge the write with recommended replication factor (see kafka recommendations)
    - Business logic related and audits
- Medium - requires write confirmation but does not need to wait for replication
    - Logs
- Fire and forget - doesn't require write confirmation or replication, as fast as possible
    - Notifications like progress details

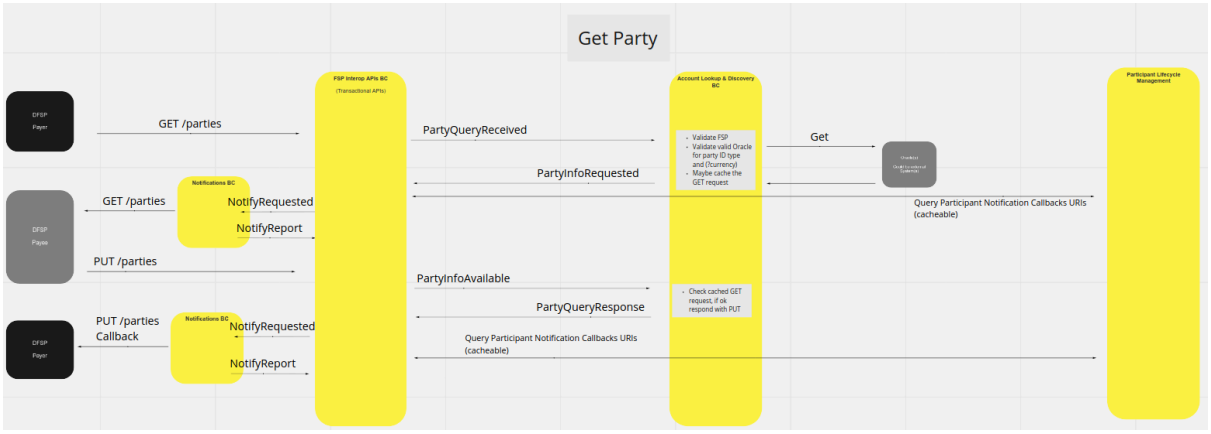
TODO

- Discuss how to protect PII in events - options: encrypt partial/fields, kafka topic access restrictions, ??
- PII Protection areas - TBC
    - Kafka Topics - Victor to bring our Kafka security standard here
    - Database field - tokenize or encrypt
    - Http payload Calls - encrypt payload using JWS

- Log files - Victor to reference our logging standard - Signing and encryption. The current audit log protection should be sufficient for the protection of PII datasets

Get Parties example:

Use case step	External/Internal	Sync/Async	Event/RPC
Initial req From FSP	External	Async/Callback	RPC/Rest
From FSPIOP API	Internal	Async/Ack	Event
From Lookup BC to Oracle	External	Sync	RPC/Rest
From Lookup BC to FSPIOP API	Internal	Async/Ack	Event
FSPIOP API Get payee endpoint from Participants BC	Internal	Sync	RPC?
FSIOP API to Notifications BC	Internal	Async/Ack	Event
Notifications BC Callback to Payee FSP	External	Async/Callback	RPC/Rest
Payee FSP Response	External	Async/Ack	RPC/Rest
From FSPIOP API - PartyInfoAvailable	Internal	Async/Ack	Event
Lookup BC to FSIOP API - PartyQueryResponse	Internal	Async/Ack	Event
FSPIOP API Get payer endpoint from Participants BC	Internal	Sync	RPC?
FSIOP API to Notifications BC	Internal	Async/Ack	Event
Notifications BC Callback to Payer FSP	External	Async/Ack	RPC/Rest



Recommended technology for patterns:

- Event based - Kafka
- External RPC - Rest'ish (FSPIOP example)
- Internal RPC - gRPC with protobuf
    - These .proto files should be in github

Internal communication recommendations for event based:

- Bounded contexts send out events when something happened, these events should be specific to the producer BC and carry relevant data (not private or sensitive data);
- Other BCs react to the events and can create internal commands as a consequence; these commands are internal to that BC and cannot be consumed/read by other BCs.
- These rules are meant to reduce coupling, i.e., senders of events don't know who is doing work as a consequence of those events (no need to know target address), and don't need to know the format the recipient is expecting (no need to know target format).
- Producing BC's are responsible for publishing and maintaining event schemas
- Consuming BC's are responsible for notifying producer BC's teams when consuming their events

Other Rules:

- For distributed transaction flows, favour the event based choreographed/saga like comms pattern (DESCRIBE)
- Every time a bounded context does work that other BC might be interested in known, it should publish an event
- Event names are past tense
- Topic rules - BCs put events in their own topics; consume from any other topics
    - Exceptions to this can exist for cross-cutting concerns, like notifications, platform configuration, auditing, etc. For these cases, BC A can publish a NotifyRequest to the notifications topic.
- Event Sourcing events should be private to the owner BCs
- Put here the event types from the PoC document (make sure it fits)

- Callbacks to external systems must use the notifications BC flow (NotifyRequested/NotifyReport). This will ensure consistent comms and the usage of the pluggable and reliable notifications mechanisms

Platform Configuration

Approach is to have a central component(s) that is used to store and serve all configurations. This central component is queried by other BCs for the specific, or generic configurations, at startup.
BCs can opt to store their local configurations, all configs will have a monotonic version number.
Any configuration received by the platform configuration BC will send a "ConfigurationsChanged" event.
The components that make-up the platform configuration

Types of configuration

- Specific to a single Bounded Context - the schema for this configuration is provided by a BC (any BC can use it)
- Switch wide configuration - the schema for this configuration is provided by the platform configuration itself (any BC can use it)

Properties of a configuration set:

- Name
- Description (like a label to be presented in a UI)
- Type (system wide or BC specific)
- Owner (if BC specific)
- Version number (monotonic and relative to the data being changed)
- Schema
    - Schema version number
    - List of configuration items - using JSON schemas (https://json-schema.org/learn/miscellaneous-examples.html)

Access control TODO

- We need an access control matrix BC owners and system-wide admins(to be established in line with the access policy) to be enforced using role based access.
    - Who can change/create schemas?
        - Ideally use MTLS to authenticate own BCs
    - Who can change configuration values?
    - Who can read/query configuration values?
    - Can we use this for the "Scheme Rule/Patterns Management" / "Scheme-mandated transaction patterns"?
    - Audit logging BC integration

    - Define segregation of duties between BC and System-wide admins

Secret management

- This is done in a similar fashion as the platform configuration, so we are extending Platform Configuration to include secret management.

Scheduling

Main use cases are:
- Client BC requests a reminder to be created
- Client BC requests a reminder to be deleted
- Client BC requests a reminder to be updated
- Client BC queries own reminders
- Platform Admin queries all reminders
- Scheduling BC executes reminder trigger when time comes

Reminders must have:
- ID / name (some identifier)
- Cron definition - recurring? Time interval? 5 secs in the future?
- Transport for trigger - http callback or event
    - Callback url or event topic
- Special payload (opaque for the scheduling BC)
- Recovery conditions (retry, reschedule, abort, abandon)
- Alerts/notifications/logging on exceptions
- Register of automatable/schedulable BC processes

Requirements:
- Must trigger reminders only once
- Must keep track of what was triggered
- Must keep track of Create, Read, Update, and Delete (CRUD) actions
- ~~Workflow (maker-checker)~~
- Job batchesSchedule
- Multiple interface options (gRPC, REST, HTTP, etc)
- Reminders should be sent with an HTTP callback, not gRPC, or to a specific topic.
- No ability to process logic external to the Scheduling BC itself
- Use Linux timestamps in UTC only to avoid synchronisation issues
- Assumption: The underlying system will keep perfect time
- Define access requirements for the scheduling BC

Exceptions
- Malformed instructions
    - Invalid date/time, including times in the past
    - Invalid BC or command

- Failed execution (identified through call-back)
- Insufficient authority (CRUD)
- Failure to process/execute reminder
- ~~BC synchronisation issues (system time loss/mis-match)~~

Questions

- After the scheduled job has been kicked off, is the scheduler responsible for tracking its progress?
    - No.
- Is the scheduler the “user” of a scheduled task in another BC, or is the “user” the person who set up the scheduled task in the scheduler? (Whose ID gets stamped on the process audit trail)
    - This is the responsibility of the client bounded context to determine, based on its action on receipt of the notification

Reporting and auditing BC

TODO:

- See main TODO for comment on **Forensic Logging**

Auditing vs Logging - Logging should be enabled and the system must have sufficient capacity to generate and store logs. Audit related entries can be extracted from the logs. Other uses of logs could include:

- Troubleshooting (performance related)
- Debugging functionality
- Security incident/event detection
- Data analysis etc

Logs are hashed and signed using a private key at the generation BC to detect tampering to assure integrity and should have access controls to ensure restriction. To ensure confidentiality,logs can be encrypted.

All cryptographic functions to support this will be provided by Cryptographic Processing Module (CPM).

Auditing

Requirements:

- The auditing system must be self sufficient, i.e., for auditing/regulatory purposes the information stored by this BC must be enough
- Audit entries cannot be changed ie, the audit BC persisted store should be immutable, should only support append and not changes or deletes.
- Consider having client BCs signing audit entries
- Audit entries, should have a pair of fields to store encrypted data and the encryption key id - these are provided by the submitting BCs
- Querying capabilities:

        - Auditor wants to see all activity in the last X days
        - Auditor wants to see all activity of action type Y, or from a certain BC
        - Auditor should be able to request the decryption of encrypted data to an operator
- Access to the audit store must be securable, so whatever tech is chosen must implement access control mechanisms (ideally that can connect to our IAM provider/connector)
- Users can interact with audit logs via a front end application where role based access controls can be enforced.

Implementation detail:

- We should probably use the same tech for the storage as the logs, in our case using ElasticSearch as the audit store makes perfect sense. This store need to be separate from the log store

Audit entry structure:

- Original Timestamp (unix)
- Persistence Timestamp (unix)
- Function / Transaction
- Source BC System ID (eg participant ID, PISP ID)
- Source BC id/name (we assume this will include UserID)
- Source BC signature
- Source BC key id (any encrypted data uses this key)
- Source BC network identifiers (e.g .IP addresses)
- Security Context
    - List of actors
        - Token (e.g. user or system)
        - Role (e.g. payer, payee, maker-checker-approval, etc)
- Action type (enum from each BC)
- Status (Success/Failure) - consider more universal names (OK/NOK ?)
- Meta tracking info (owned by the platform monitoring BC)
    - trace id
    - parent id
    - span id
- Label list (this is for data that can be easily filtered)
    - Key string + Value string + encryption key id...
- Payload (opaque to the audit BC)
    - Do we really need this? It can lead to abuse and storing of unneeded info

NOTE: the tracing data structure is owned by the platform monitoring BC - distributed tracing below.

TODOs

- Do we need an explicit list of audit action types or can this be fetched implicitly from existing data later. I.e., does the Audit BC need to know all possible audit action types, even the ones that never happened? This will require some bootstrap

mechanism, where the source BC sends all possible action types to the audit BC. Is this necessary?
- Consider the audit entry size and the capacity of the persistence layer (in years)
    - Consider a size and count limit for the label list
- Define audit views or reports to deliver “Forensic Event Logging & KMS” and “Forensic Event Log Access & Management”
- How CPM will be used for signing and encryption of audit log events

Logs

Requirements:

- Logs are technicalities - should we lose logs there should be no consequence other than losing technical ability to understand how the system behaved when losing logs
- Log all system activities and make it available for Audit BC to run queries against existing log data.

Implementation details:

- Abstraction layer is the event structure received by the Logging BC, which in turn will be used to persist the actual log to the store
- Source BC have to publish the log events with the agree format - using whatever mechanisms (fluentd stdout scraping, direct publish of msg, etc)

NOTE: the tracing data structure is owned by the platform monitoring BC - distributed tracing below.

TODOs

- We need to define standard log entry structures and standard logging endpoints/transports.
- Consider using audit logging structures as minimum structure and add performance related information for debugging

Reporting

Purpose is to deliver a reporting only data store, that is automatically updated by a switch provided component.

- This data store will be write-only from the switch and should be read-only by external components
- The data model on the reporting data store can be different from the internal operational data models that the switch uses
- The component provided by the switch will be translating internal events and internal data models to the external data store models - This component can be replaced

TODO:

- Decide which initial reports and dashboards should be included as part of the base reporting functionality
- Chose an open source reporting and dashboarding tools to deliver this base functionality
- Compliance / Assurance Reporting - define these reports (KYC, AML)
- Discuss “Process Monitoring (and SLA's)” and decide if this can be done on top of the reporting layer (the definition of the critical numbers, SLI’s & SLO’s, is done via platform configuration)

Reporting strategies:

- Event based - Preferred way - On the reporting BC sits a component (event handler) that is listening to relevant event from its correspondent BC and transforms those events into reporting data store entries - this component should 1 per source BC, and should be the only one responsible for that data;
- Push - The source BC will call the reporting API to send data, this API will be transforming the data and persisting it to the reporting store (1:1 with the source BC, ie, there should be a mini API per source BC)
- Pull - On the reporting BC sits a component (timer based) that calls an API at the source BC to retrieve data, that is then persistent to the reporting store (1:1 with the source BC)

Rules:

- Only the reporting data store can be used for reporting and dashboarding - reporting or any other external components are forbidden access to bounded contexts data stores.
- We can never have the source data ending up directly in the reporting store - There must be a translation between the source data structure and the reporting data structure, even if there are no structure changes. Objective is to not have a dependency on the source BC data structure on the reporting side.
- For performance critical BC we should always try to use the event driven reporting strategy.

Security

TODO:

- Investigate the possibility of using OpenID with OAuth2 instead of just OAuth2 flows with custom fields (inside the JWT, for userId, appId, roles, etc).

Authentication

- An internal authentication service is responsible for processing authentication requests (logins)
    - This login is forwarded to an IAM provider, which will perform the actual authentication and return an access token and an optional refresh token
    - The returned token must include the necessary information like user/app ID and the claims/roles (requested by the authentication service)

- The central authentication services will serve/proxy the list of users, groups and applications, as well as their associations - the data will be provided by an IAM implementation
- The central authentication services will not own any information about users, groups
- All requests to the system must include the access token
- Between the authentication service and the external IAM implementation an adapter component (anti corruption layer) must exist to guarantee the decoupling between the internal authentication service and the external (multiple) IAM implementation

Identity management

- Identity management services are provided by external IAM implementations and served internally by authentication service (that proxies requests to the IAM)
- IAM implementations must support:
    - user, group and roles functionality, as well as their relations/mapping
    - Application provisioning and credentials (clientId + client secret) oauth2 client credentials flow
- Explore WS02, Ory and Keycloak option for the OSS community reference implementation

Authorization

We're using a multi tier approach, where the roles are global and privileges are BC specific.

- Each bounded context is responsible for defining their own privileges, and by publishing those privileges to the central Authorization services at bootstrap, upgrade, etc
- The security bounded context also provides the system wide privileges, for example the privilege to login to the system, as well as the privileges to manage the authorization aspects (roles/privilege association)
- The central Authorization services will maintain a matrix with the association of Roles with a list of granular privileges (from the BCs)

Service instances should publish a metric about when they last fetched authorization data from the central Authorization services.

Access Policy Management

- This is the matrix composed by:
    - Roles and privileges association
    - Users/Groups/apps and roles association

Application Management & security

- From the authentication and authorization perspective Applications are equivalent to users. They will be mapped to roles and privileges just like users.
- The IAM implementation must allow the setup/provisioning of applications and applications credentials, typically client ID and client secret.

- This should permit the creation of application security principles from the ML platform. These requests would have to be sent to the IAM provider for the principal provisioning.

Personal Identifiable Information storage and transmission (PII)
We need to address this, how to store it and how to protect it in the cases where this is being transmitted in events.

- Encrypt/sign all events with PII
- Authenticate BC's accessing events with with PII
- Isolate PI event so they could be delete in case where the right to be forgotten is exercised
- All access the PII events should be audible and monitored as well if possible

Platform monitoring & health checks

WIP

Health checks

- Services must provide more than just up/down, they should provide a liveness check and readiness check endpoints
    - Alive - I'm responding to the health checks, am up and have resources to work
    - Ready - besides being alive, I can serve/do work, all dependencies are also ok (this can be false in the case of a dependency not being available, services not ready should not be restarted)
    - TO DECIDE LATER - This ready state could be a non-boolean state, ex:
        - Ready - all ok
        - Ready - but resources close to limits (this might be just a metics interpretation)
        - Not ready - Draining/Terminating
        - Not ready - Maintenance mode

- All services must have these two health capabilities and use separate ports for the liveness probe and readiness probe

Metrics and performance requirements

- All services have to provide standard metrics endpoint
- Implementers are required to input thresholds that can be interpreted by the *platform* to make scaling or alerting decisions - these inputs can be gathered by executing the load tests in the production env
- Platform must be able to interpret the liveness, readiness and metrics to make those decisions (restart, scale out/in, or alert)

Distributed Tracing

TODOs:

- Define the event structure for the tracing events;
- Define the tracing product to be used for storing and querying.
- Define the standard trace state key pairs (ex: BC name, service name, service instance id, etc)

All requests, internal and external, should include distributed tracing information that must be recorded in logs and audits. This structure should be the same used in logging and auditing.

We use an event based strategy to record the trace events.

1. Producers or tracing information have to publish the tracing events to a TBD topic;
2. On the Platform Monitoring BC there is a tracing event handler that is responsible for the processing of these events: sending them to the correct external tracing system (OSS to be decided)
3. Tracing system will store and make the tracing information available for consultation

Reference from the current implementation:

- https://github.com/mojaloop/mojaloop-specification/blob/master/fspiop-api/documents/Tracing%20v1.0.md#table-1-%E2%80%93-data-model-of-http-header-fields-for-Tracing
- https://www.w3.org/TR/trace-context-1 – Trace Context - Level 1 - W3C Recommendation 06 February 2020
- https://w3c.github.io/trace-context – Trace Context - W3C Editor's Draft

We're using the same structure as today - see table 1 of: table 1

Minimum tracing structure proposal:

- parent id
- trace id (child of parent id)
- Trace state key-pair list
    - For vendor="moja" we will encode the following key-pairs as part of the trace state:
        - trace id
        - service id
        - environment id
        - labels (optional)

Storage and querying:

- Trace events will be captured into Kafka by each BC
- The Platform Monitoring BC will ingest trace events from Kafka

Recommendations for DFSPs:

- Include the tracing headers in follow-up requests that are connected (or consequence) to responses sent from the Switch.

Load and scale tests

The platform must include a set of load tests that can be executed on a production environment, prior to its operation, to understand its limits and provide those thresholds to the platform monitoring system.

These tests should assess the deployment in a comparable way, i.e., it should output numbers that describe the performance characteristics of all the main components of the platform, even the infrastructure components like kafka, DBs or network latency.

TODO, define the details of what to measure and how

This should also characterise the required resources per pod/service; so we use these limits/constraints at infra/K8s level. Maybe some of these can be provided by the dev team, as they are not deployment related (ex: max ram of a certain pod can be calculated at dev time).

Resilience requirements

Service calls to other resources should be assumed to fail sometimes.

Calls to dependencies (dbs, infra, other services, etc) must implement known standard patterns: Retries, sensible timeouts, circuit-breakers and bulkheads.

Service levels (SLA's, SLO's and SLI's)

We need describe what this is, copy from google's SRE book https://sre.google/sre-book/table-of-contents/

Requirements

- Enforcement of inbound SLAs/Throtting to be handled by an API Gateway (cross-cutting concern across API BCs).

Transfers

Current SLA is:

- 200 financial transactions per second served under 1 second, with less than 1% above 1 second.
- Uptime agreed with DFSPs is 99%
- Response time - Prepare and fulfil take less than 500 ms

SLOs

- Uptime internal objective is 99.9%
- Response time - Prepare and fulfil take less than 300 ms

SLIs:

- Uptime (%) - who to measure
- Response time - we measure with average of the last 5 mins with a sampling rate of 75%

Agreements/Quotes

Settlements

Auditing and reporting

Other?

Platform automation (scaling, restarting, etc)

Resource allocation

In the case of K8s, pods must have resource constraints on CPU and RAM at least

Scaling decisions

When to scale out and back in

Restart decisions

When should pods/services be restarted?

- Health checks related
- We should restart a service if it hasn't been able to reload the authorization data (ie, it refreshes authZ data every 5 mins, if it can't do that for more than 15 mins, then platform should restart that service instance)

Notification decisions

When should operators be notified and how?

Should not be using the notifications BC service, must use a separate channel for critical platform notifications.

Infrastructure Services Deployment Recommendations

Async messaging / Kafka

(or whatever async msg/event/streaming platform we use)

It need to provide:

- Topics
- Partitioning, ordering guarantees and single reader (per topic/partition pair)
- Automatic reader load balancing per topic (rebalancing)
- Cluster needs to have at least 3 nodes
- Writes need to be replicated to at least 2 nodes for critical data (need to guarantee data is not lost)
- Report health and performance state
- Horizontally scalable
- Security
    - Encryption in transit (TLS/SSL)
    - Encryption at rest
    - PII considerations?!?!
    - Strong Authentication
    - Per topic access (only read, only write or read&write)

Notifications

TODO:

- Was not discussed

General Recommendations and Architecture Principles

Communications with external services

BCs that communicate with external services should implement an Anti-Corruption Layer (ACL) pattern to protect from external dependency and external model corruption. This can be a simple software component that exposes an internal abstraction and communicates with the external component.

Cache invalidation

Recommendations:

- Bounded context should try as much as possible to use events to update and invalidate local cached data
- BC's must define a sensible timeout for all cached data, and the preference should be to get fresh data if local cached data is timed out
- Bounded context should be able to interpret Cache invalidate requests from the outside (events or calls)
- Sensible in this context has to be considered in light of the entity being cached, something that changes every minute should not have a cache greater than a few secs. Something that changes every day or month, can have a cache of up to 5 mins. Usually nothing requires a cache longer than 5 mins.

----- Older stuff below -----

Slides proposed structure:

- ref arch
    - what is a ref arch and why we need it
    - what we did (ddd process)
    - details of the proposed high level arch
        - right level of abstraction
        - bounded contexts and why that level of decoupling is important
        - communications patterns
    - gaps that were identified
        - what needs work, where the as is different implementations
        - missing blocks:
            - security
            - Platform configuration
            - Central auditing, logging and reporting
            - UI
    - examples of how this would support/benefit:
        - License changes = need to replace components - grafana
        - Resiliency - ability to recover - how do services recover individually?
        - Replacing core component - tigerbeetle AHB
        - Replacing core component - notification MdB
        - Extending functionality - frms
        - customising UX/UI - custom web UI using the APIs
        - community contribution to marketplace - slack notification plugin or some other example
- Ideally we have 1 or 2 sub presentations about the examples above:
    - TigerBeetle specific
    - FRMS or community contributed plugin
    - Adding a dispute resolution module MR
- how to execute proposal
    - two options:
        - slow evolution OR
        - working in parallel on v2 (favourite)
    - Including an idea of how many developers
    - Can we have a single team led by me, Adrian and Miguel? So everyone is represented?

Ref Arch Docs / Slides

Main objective of a reference architecture is to provide guidance to the detailed implementation architectures of a future version of a product, and its development.

It captures the essence of the current product and guides the development of the future vision.

Ref Arch is:

- Captures the essence of the technical design principles
- Identifies abstractions and standardisation opportunities
- Proposes solution patterns to common problems
- Provides guidance to the actual implementation architecture
- Provides clarity on future to help strategic and tactical decision making
- Promotes innovation and contribution, by laying forth what can be done and how

What is is not:

- not an actual concrete implementation or implementation architecture
- not a criticism of the current implementation
- not an immutable document

What it needs to be:

- understandable by all parties (not purely technical)
- Up to date and available

What is the right abstraction level?

- It can sometimes go as high as to discuss the requirements
- Can go as low as specific code strategies or tech implementations
- For the most of it, should ideally be focused on the systems and actors level (incl. components)

Ref Arch should propose:

- Common integration patterns to common problems found in all distributed systems, we're doing SOA, but how do we do:
    - service-to-service communication (sync or async rpc; message/event driven)
    - async work and queueing
    - concurrency and parallel processing
    - consistency &&/|| reporting DB sync
    - performance and scalability of critical components
- Common approaches to the whole platform:
    - AuthN & AuthZ (and IAM)
    - Logging and monitoring
    - Discovery
    - Configuration Management
    - Resiliency / Fault Tolerance (retries, circuit breaker)
- Provide a strategy to deal with future concerns:
    - Interoperability with future external systems
    - extensibility (ability to add features without significant change, and enable a marketplace plugins/modules)
    - overall scalability and its relation to cost (ability to continue scaling)

Auditing, Logging and Reporting

--- New Doc below ---

Introduction

What is a ref arch and why we need one

Reference architecture role in next version of the platform

Approach Used and how we worked to get it

DDD, etc

Reference architecture

Principles guiding this architecture

Single responsibility principle, extend don't change, etc (most of SOLID)

Advantages of this design:

- Single responsibility and the internal canonical interfaces, provide the ability to implement ISO without having to change the inner system

Problem space

Problem space identification and map;
Description of types of domain and subdomain

Solution space - bounded contexts

High level description and the context map
High level description of each BC (not the details of the BC's)
High level description of each cross-cutting concern BC

Bounded contexts details

Here we describe better the BC's function and its details
One subsection per BC

Context1

Text goes here

Context2

Text goes here

Context3

Text goes here

Cross cutting concerns

Here we describe better the cross-cutting concerns BC's function and its details
One subsection per BC

Other recommendations

Recommendations or guidance for technical implementation, in regards to patterns, principles and or code structure and defensive coding strategies

Some of this can be required, other things can be just a recommendation.

Comms patterns
Cache invalidation
Other abilities, like DR, resilience, scalability, etc

How to implement this for the next version

Here we need have a quick explanation about how we think is the way to implement this and next steps, linking to a detailed document (this doc should not include the details)

Recommended revision interval - this should be a living doc

Implementation strategy

Phases / releases:

- Alpha
- Beta
- Release Candidate 1 (and others)

Alpha phase is intended to deliver:

- Maintain the current invariants, ex: JWS
- Stable interfaces (canonical/internal)
- FSPIOP and ISO Interop API initial support (TBD)
- Minimum assessment that the design is adequate (needs to be specific)

- Guarantee of minimum performance and scalability (no optimisation at this phase) 1000/1hr/<1% err
- Happy path only
- Fully testable code

Non functional requirements for Alpha phase:

- Minimum auditing (TBD)
- Logging, monitoring and tracing (TBD)
- Platform configuration and secret management
- Basic reporting datastore and ingestion mechanisms (no fancy reporting or dashboarding tools required)
- Authentication
    - central login & IAM integration
    - services able to validate JWTs independently
- Authorization
    - central collection and management of privileges and roles
    - association with users/apps
    - client services/BCs ability to publish their privilege lists at bootstrap
    - Client services' ability to receive role/privilege maps at startup and on intervals (and authorise requests)

Main use cases included in Alpha phase (only happy path):

- Participants
    - Onboarding
    - Management of limits
- Lookups
    - Register identifiers in ALS
    - Get parties (single unique identifier)
- Quotes
- Transfers (with timeouts)
- Settlement (deferred net)
- PISP
    - Link account
    - Request to pay

Glossary

References and further reading