

mojaloop

Mojaloop Phase3

Supporting Adoption & Deployment

Phase-3 kick-off

Jan 2019

ModusBox for BMGF

Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment



Phase-3 Kick-off Agenda

1. Phase-2 Summary (focus on PI-4)
2. PI-4 Features
 - a. Settlements
 - b. QA Regression testing framework
 - c. CEP, email-notifier
3. PI-4 Operational Monitoring Capability
4. Performance
5. Phase-3 RoadMap

Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment

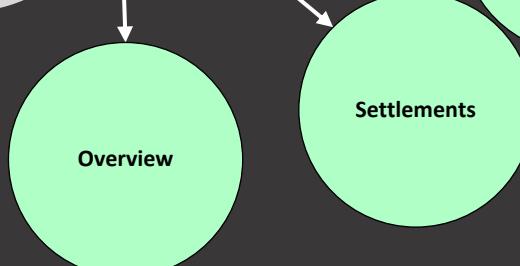
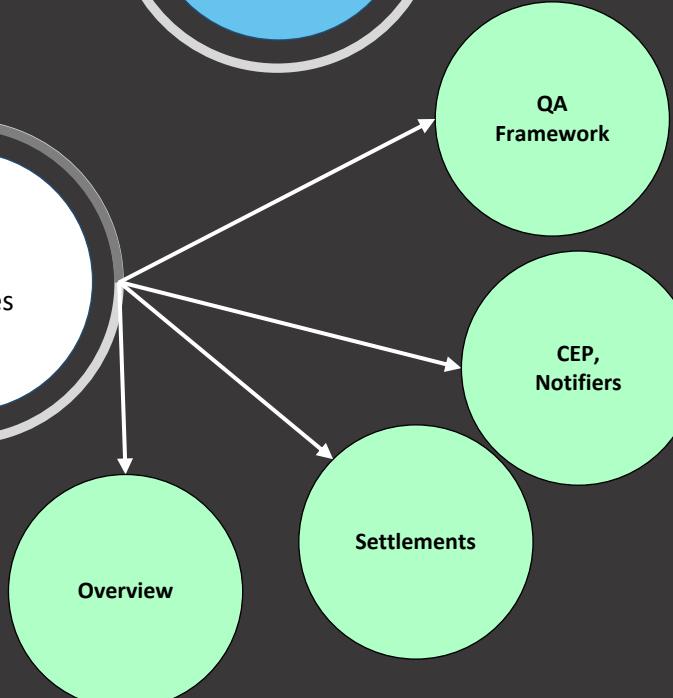
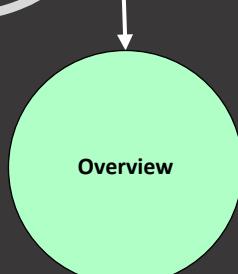
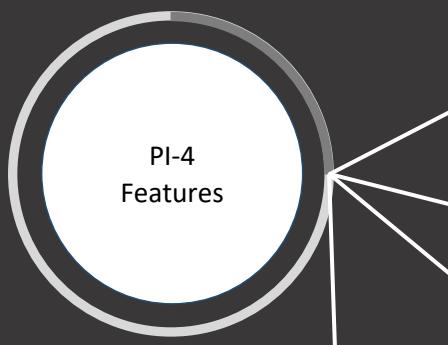


Timeline Overview

Timeline	Summary
Phase-1	<p>Level One Project</p> <ul style="list-style-type: none">• Reference Implementation• 6 Program Increments (PIs) (2016 - 17)
Phase-2	<p>Road To Productionization: Phase-2 (2018)</p> <ul style="list-style-type: none">• PI – 1 (Feb - April)• PI - 2 (April - June)• PI - 3 (June - August)• PI – 3.5 (September)• PI – 4 (November-December 2018): Performance, Settlements, CEP, QA Framework, Operational Monitoring, Managed backlog for Phase-3
Phase-3	<p>Supporting Adoption & Deployment – (2019 Jan - June)</p> <ul style="list-style-type: none">• PI-5: Merchant Payments, Account lookup, PoC for automated DFSP handler provisioning.• PI-6: Error, Event handling frameworks, PoC for Bulk payments, Comprehensive QA Framework, streamlined CI/CD process

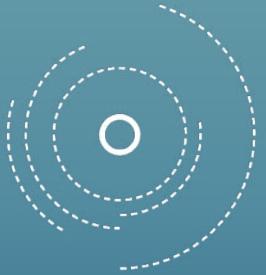
Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment



Phase-2 Summary: PI-4

PI	Topic	Summary
PI-4	1. Performance Improvements	1. Performance issues investigated, several high priority issues addressed to improve performance.
	3. QA Regression testing framework	2. A Regression testing framework is developed that can be run on a given Mojaloop Switch deployment for the features covered. Currently this is run daily against the dev AWS envt. and result reported.
	3. Settlements support	3. Settlements Functionality evolved to support more key features
	4. Operational Monitoring Capability	4. Instrumentation for metrics, Grafana dashboards developed. Support for ELK added
	5. Managed backlog for phase-3	5. Backlog for Phase-3 using stories onboard provided for discussion
	6. Bug Fixes, Community support	6. High priority bugs fixed based on QA support by partners, internal QA, performance testing and other sources. Support for community questions provided.



mojaloop

PI-4 Features Overview

PI-4 Features: Overview

PI	Topic	Summary
PI-4	1. QA Regression testing framework	A functional /regression testing framework is available that can be run periodically on a given deployment and send reports.
	2. Settlements support	Support for commercial Settlements activity provided; Functionality for integration of accounts with Banks added. Features to activate, de-activate FSPs, accounts provided. Reconciliation of positions and handling of aborted settlements provided.
	3. Central Event Processor (CEP), Notifier	A central event processing framework developed to monitor, respond to events based on Rules and send alarms/notifications as configured

Switch Operations [Resources] - Overview of changes

Participants

- [●] POST - Create
- [●] GET - Query
- [●] *PUT - Update*

Participants Limits

- [●] POST - Set initial Position
- [●] *Manage Limits*
- [●] POST - Set Limits
- [●] *PUT - Update Limits & Thresholds*
- [●] GET - Query Limits

Positions

- [●] GET - Query by FSP

Participants Callback

- [●] POST - Set Callback URIs
- [●] *PUT - Update Callback URIs*
- [●] GET - Query Callback URIs

Settlement Windows

- [●] POST - Open new window, and close previous
- [●] GET - Query

Settlements

- [●] POST – Create/trigger new Settlement with associated Windows
- [●] *PUT - Receive Acknowledgments from Settlement Providers*
- [●] Process successful Settlement acknowledgements
- [●] *Reconcile Positions based on successful Settlements*
- [●] GET – Query

Accounts to support Bank integrations

- [●] *Funds In (Collateral deposited into an a/c, e.g.: FSP)*
- [●] *Funds Out*

Key

- [●] Fully implemented
- [●] Legacy Code
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope

Switch Operations [Use Cases]: Overview of changes

Participants

- [●] *Manage Participants*
 - [●] Create Initial Value
 - [●] Query
 - [●] *Update*
- [●] *Manage Participant Limits*
 - [●] Create Initial Value
 - [●] Query
 - [●] *Update*
- [●] Manage Callback URLs
 - [●] Create Initial Value
 - [●] Query
 - [●] Update
- [●] *Manage Thresholds for Limits*
 - [●] *Create Initial Value*
 - [●] *Query*
 - [●] *Update*

Onboarding, Auditing, Security

- [○] To be defined/prioritized

Settlements

- [●] Open, close Settlement Windows
- [●] Query Settlement Windows
- [●] Query Settlement Report
- [●] Create/Trigger Settlement with Windows
- [●] *Process successful Settlement Acknowledgements*
- [●] *Reconcile Positions based on successful Settlements*
- [●] *Process failed Settlement Acknowledgements*
- [●] *Sending notification for Position Adjustment*
- [○] Link Funds In/Out to Limits

Key

- [●] Fully implemented
- [●] Legacy Code
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope

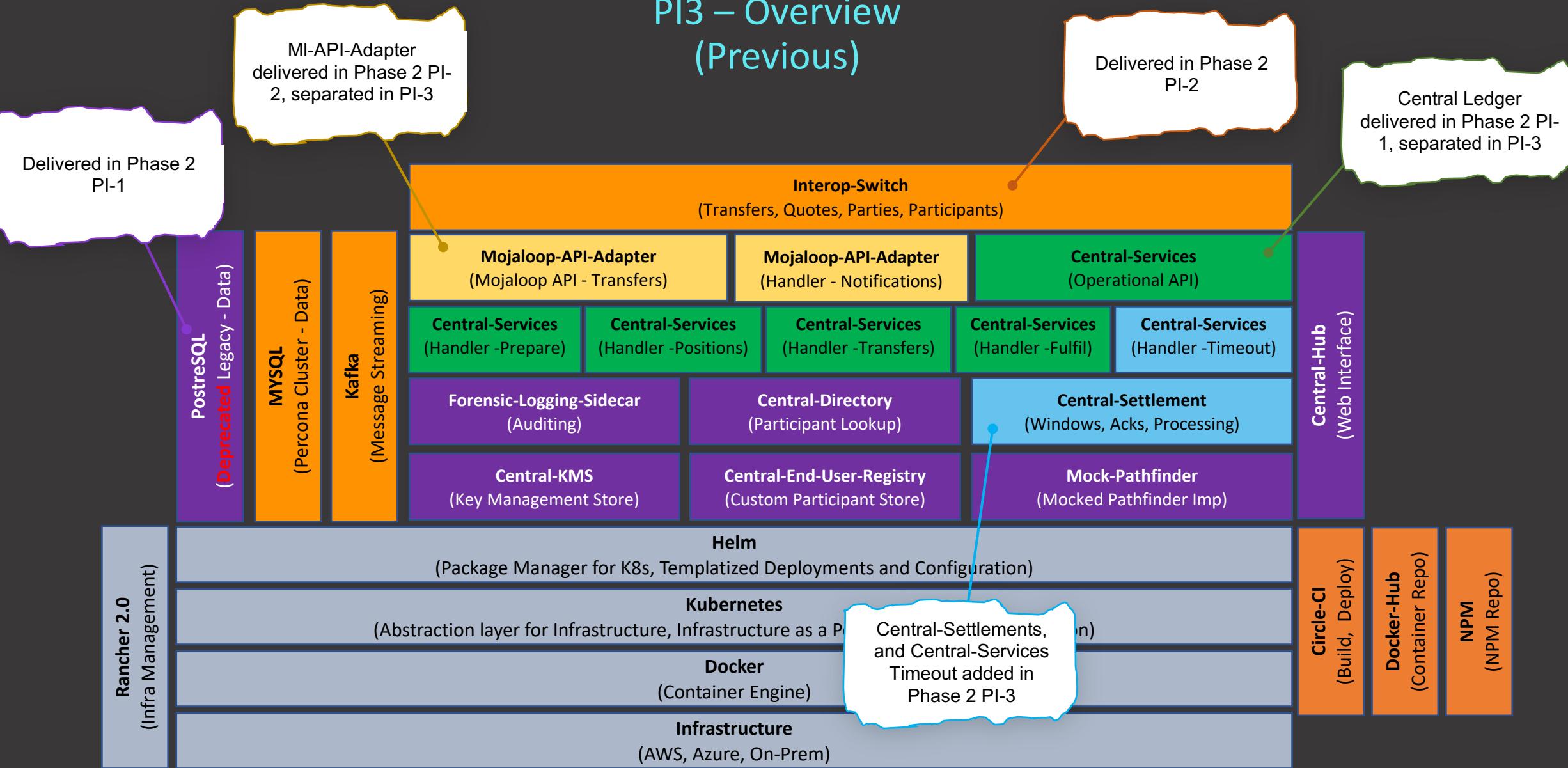
Positions

- [●] Query Positions
- [●] Manage Positions
- [●] Create Initial Value

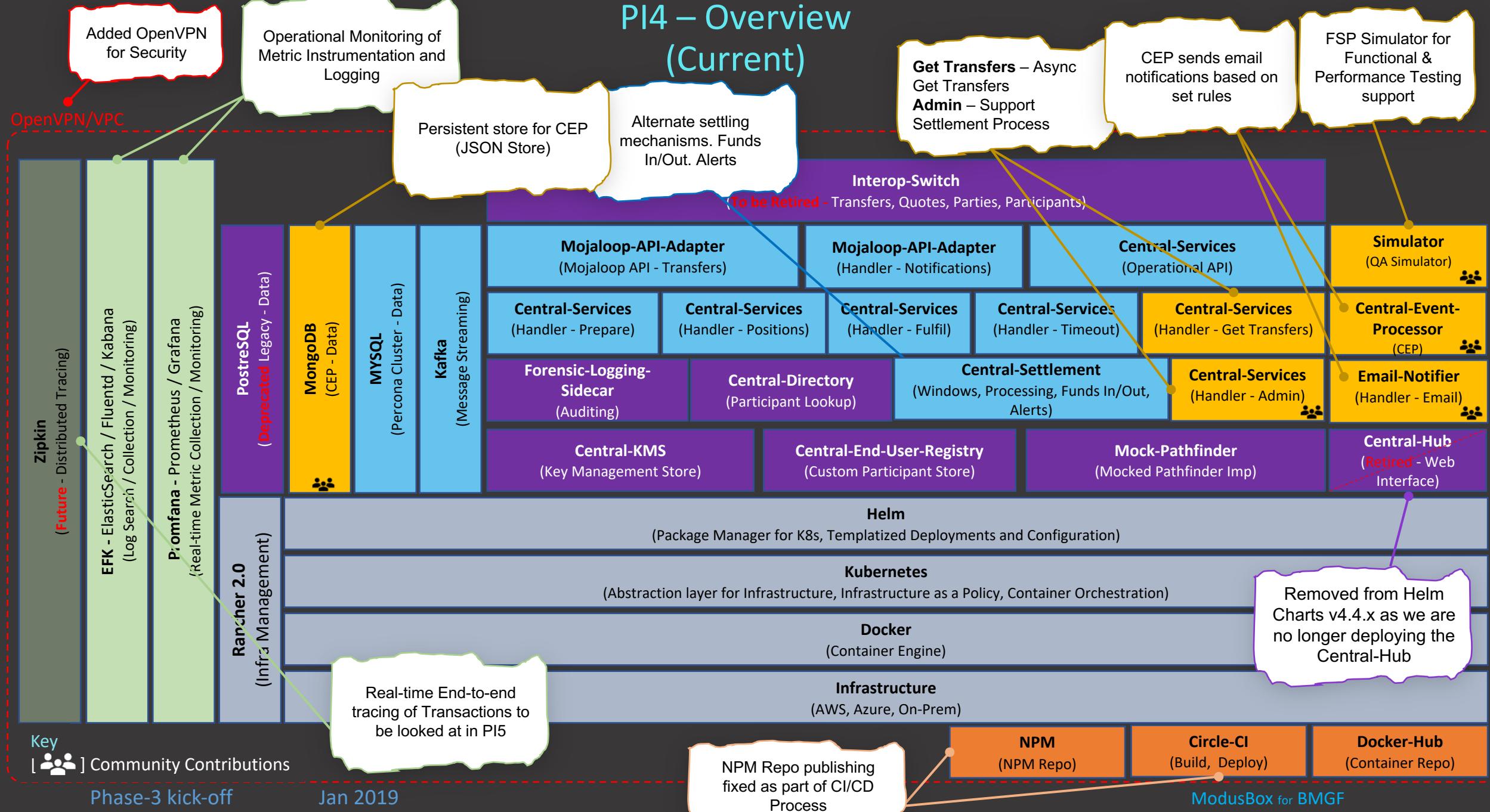
Monitoring

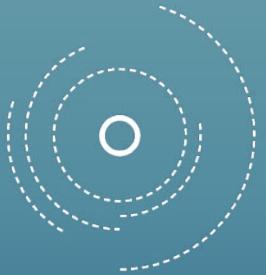
- [●] *Instrumentation for metrics*
- [●] *Real-time Metric Monitoring via Promfana*
- [●] *Log Monitoring via EFK (ELK)* (Dashboards pending)
- [●] *Instrumentation for metrics* (only done for Golden Path Process) ModusBox for BMGF
- [○] ZipKin

PI3 – Overview (Previous)



PI4 – Overview (Current)





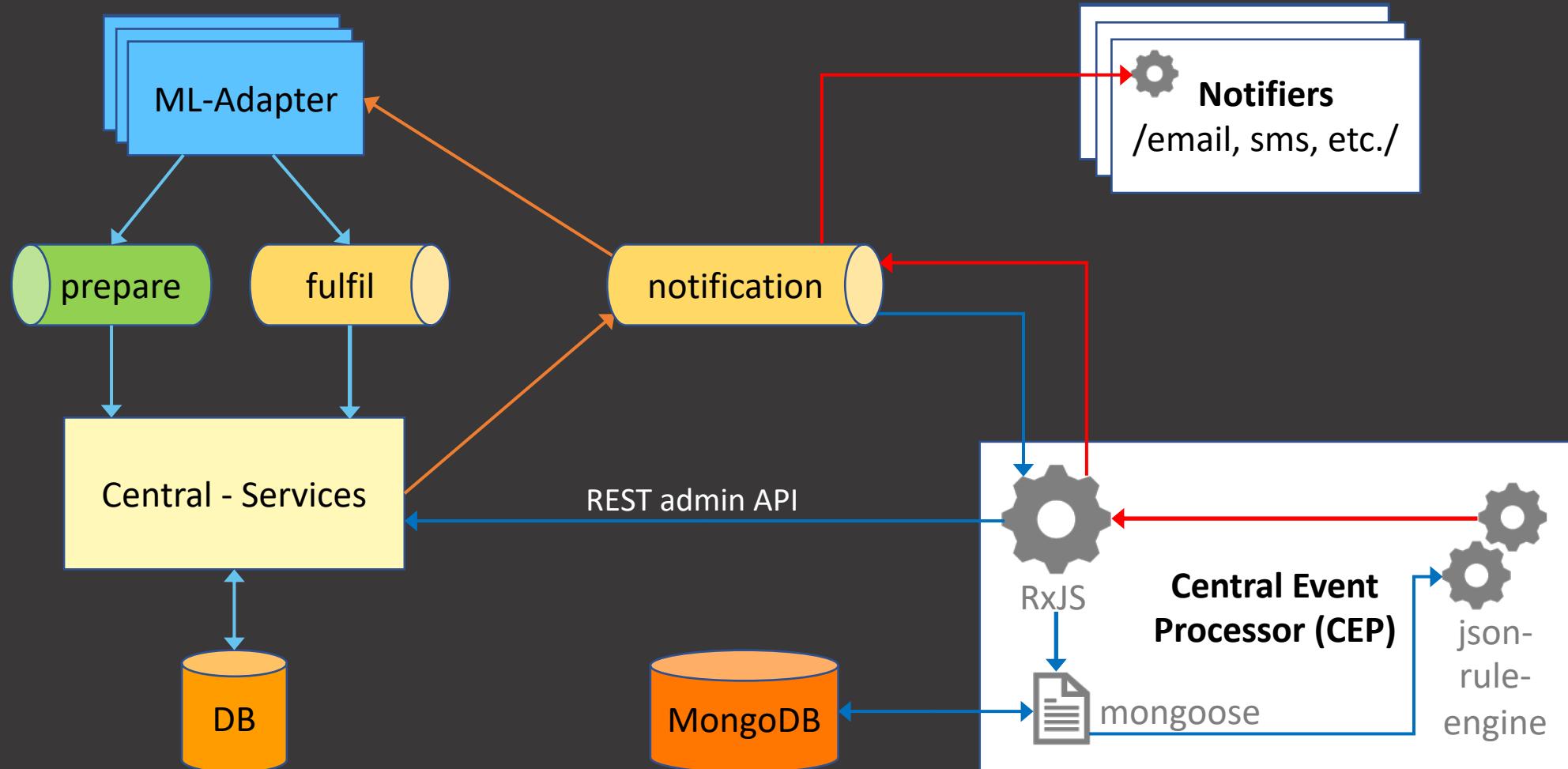
mojaloop



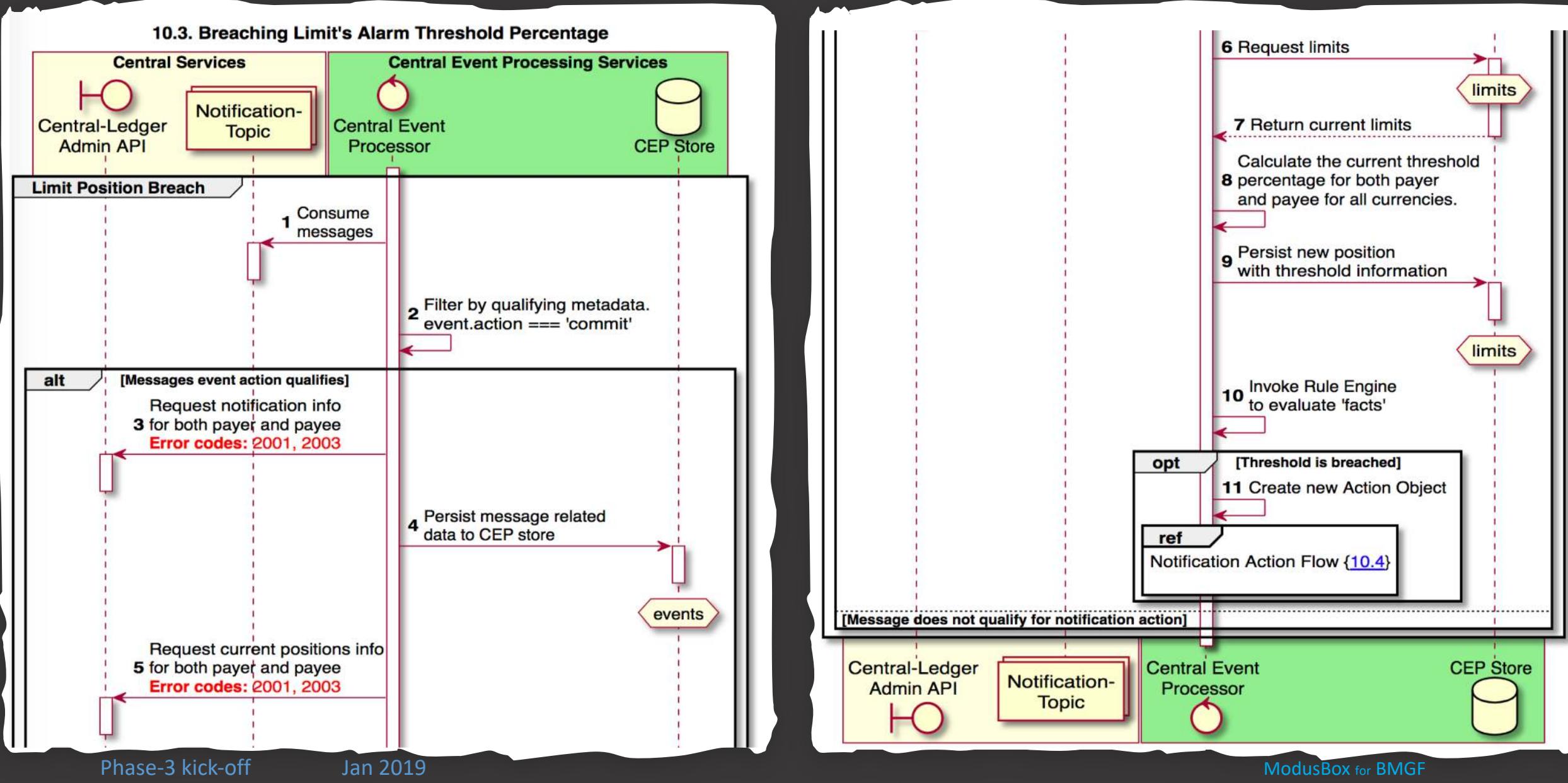
PI-4 Features

Event Processing, Notifications

CEP Architecture and Technologies Overview



Example Central Event Processor Action



CEP & Email Notifier - Demo Index

1. Net Debit Cap (NDC) Threshold Percentage Breach Alarm
 - a. Set participant (dfsp1) NDC limit
 - b. Set notification email endpoints for Hub and dfsp1
 - c. Execute a transfer – *prepare & fulfil (to achieve the result)*
2. Net Debit Cap Change Notification
 - a. Set notification email endpoints
 - b. Adjust participant limit



mojaloop

PI-4 Features QA Regression Testing Framework

Regression Testing

is typically characterized by

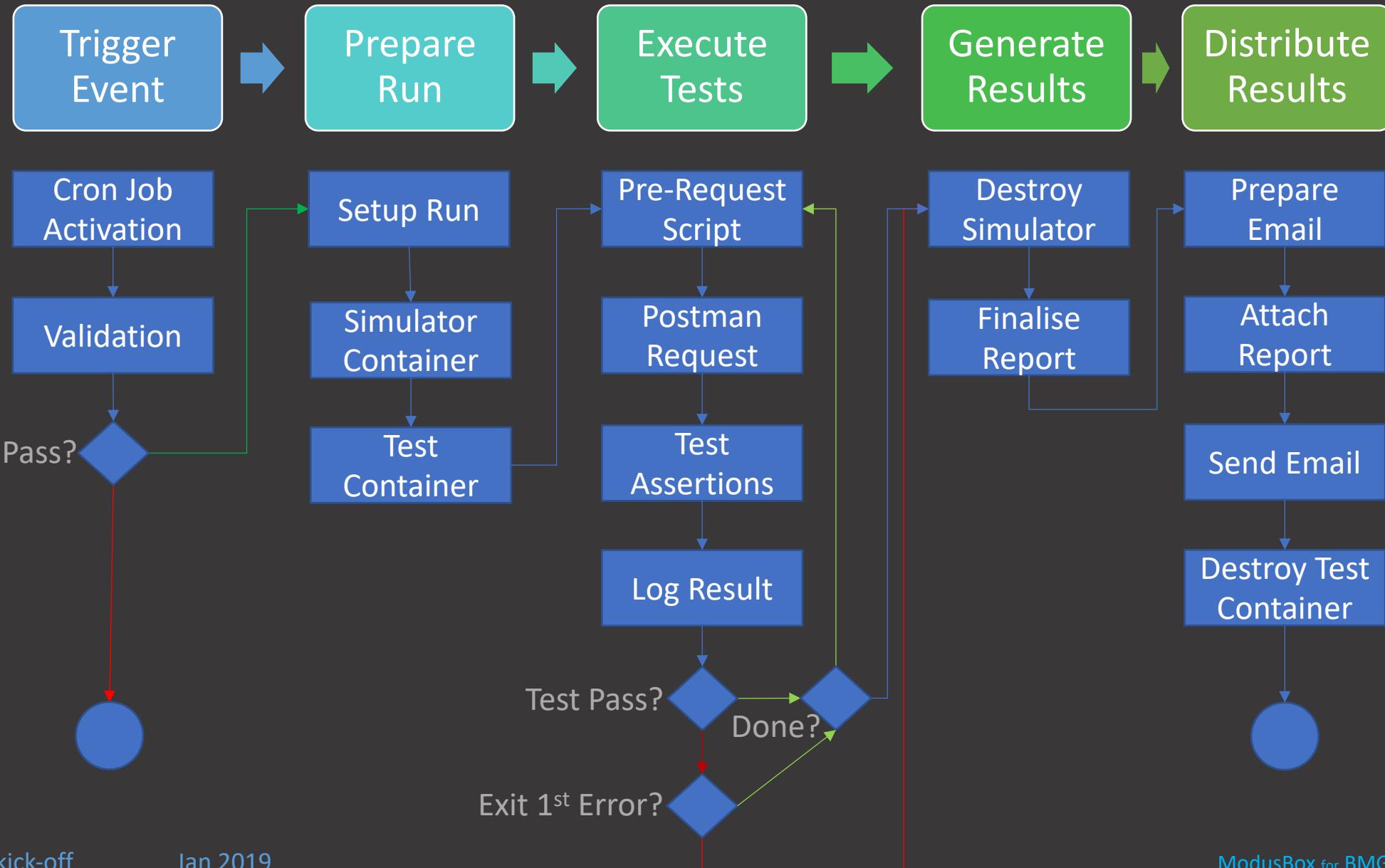
- Ensuring a specified deployed environment performs and produces consistent results
- Can be triggered manually or automatically by linking it with an event
- Ideally could be executed many as many times as required - idempotent
- Automatically sends out alerts in case of any failure or unexpected result
- Communicated results can be used to pinpoint anomalies
- Executes the entire range of functionality of the system under test (FR / NFR)

Regression Testing

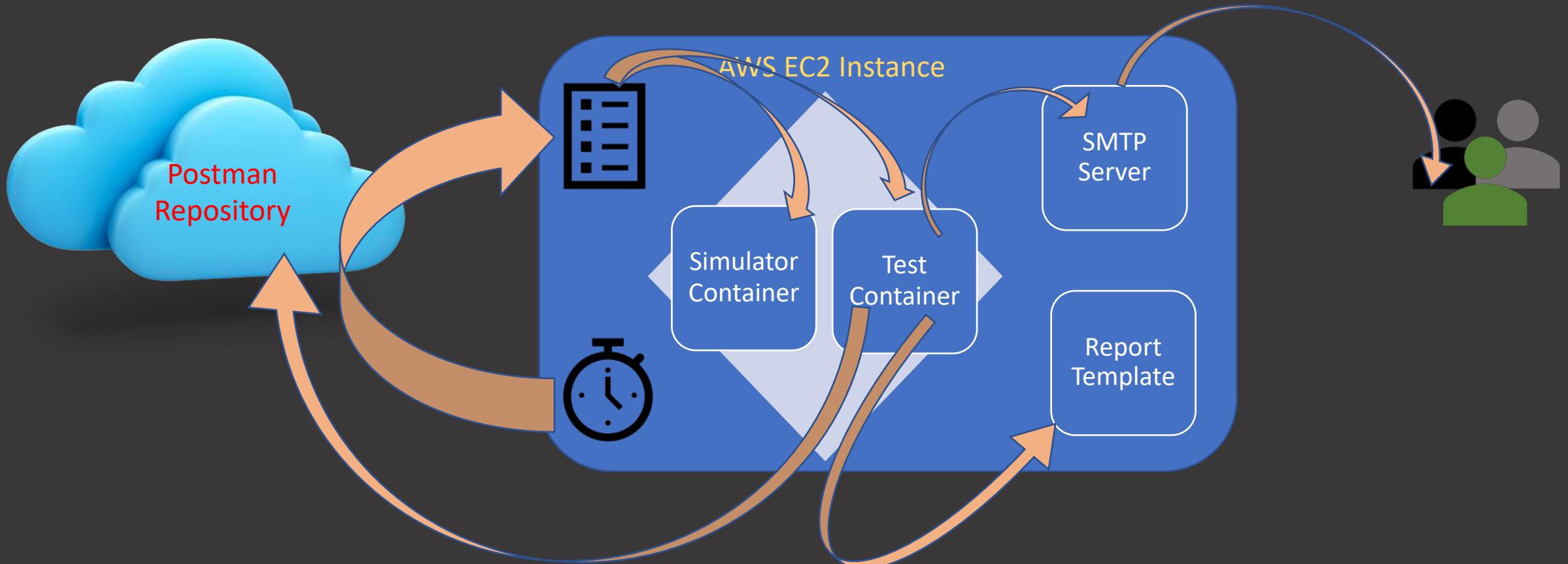
is not

- Unit testing
- Coverage testing
- Integration testing
- Specific functionality (use case) testing
- Testing or checking the health of a deployed environment

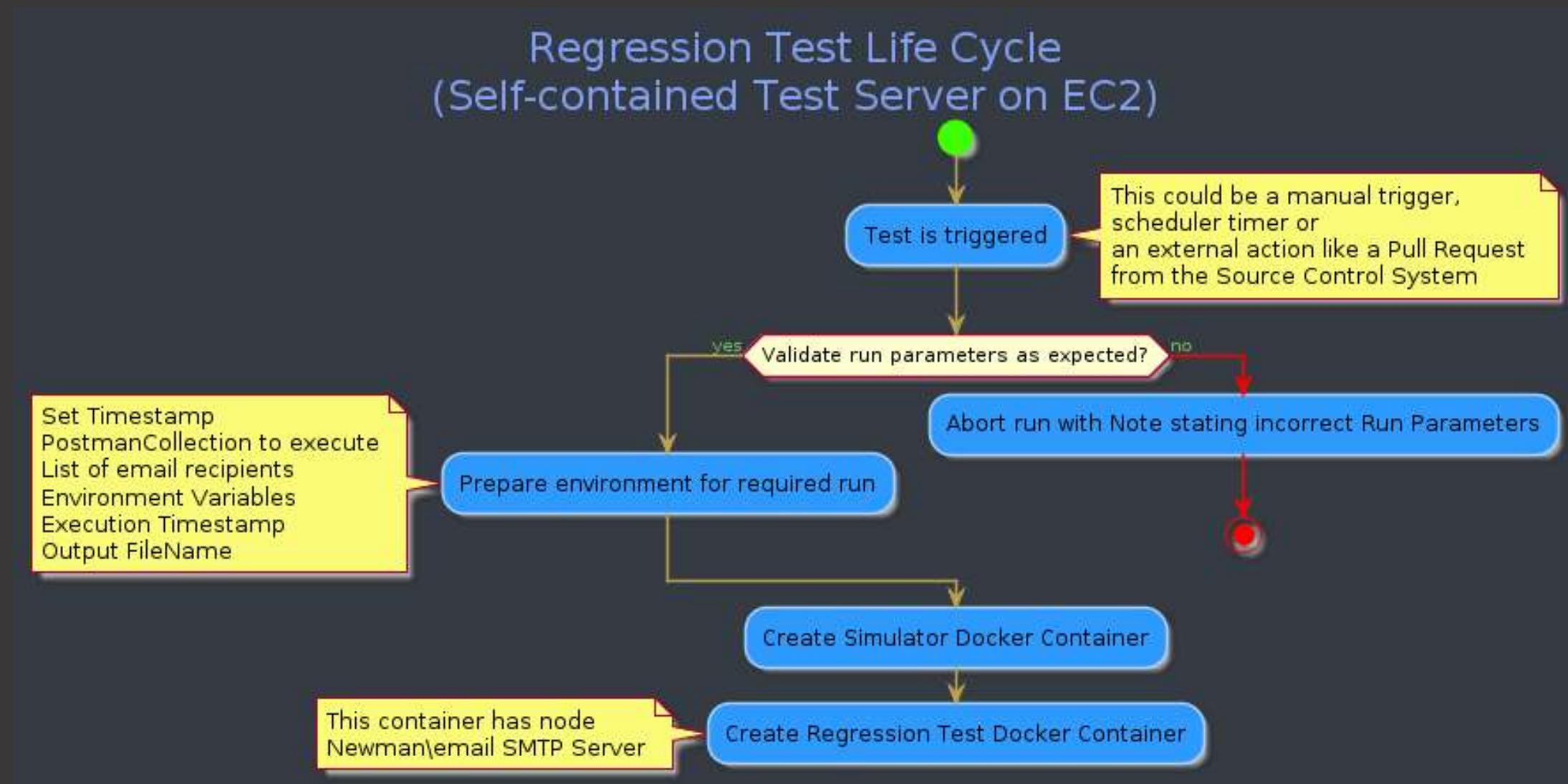
Regression Testing – Instance Life-Cycle



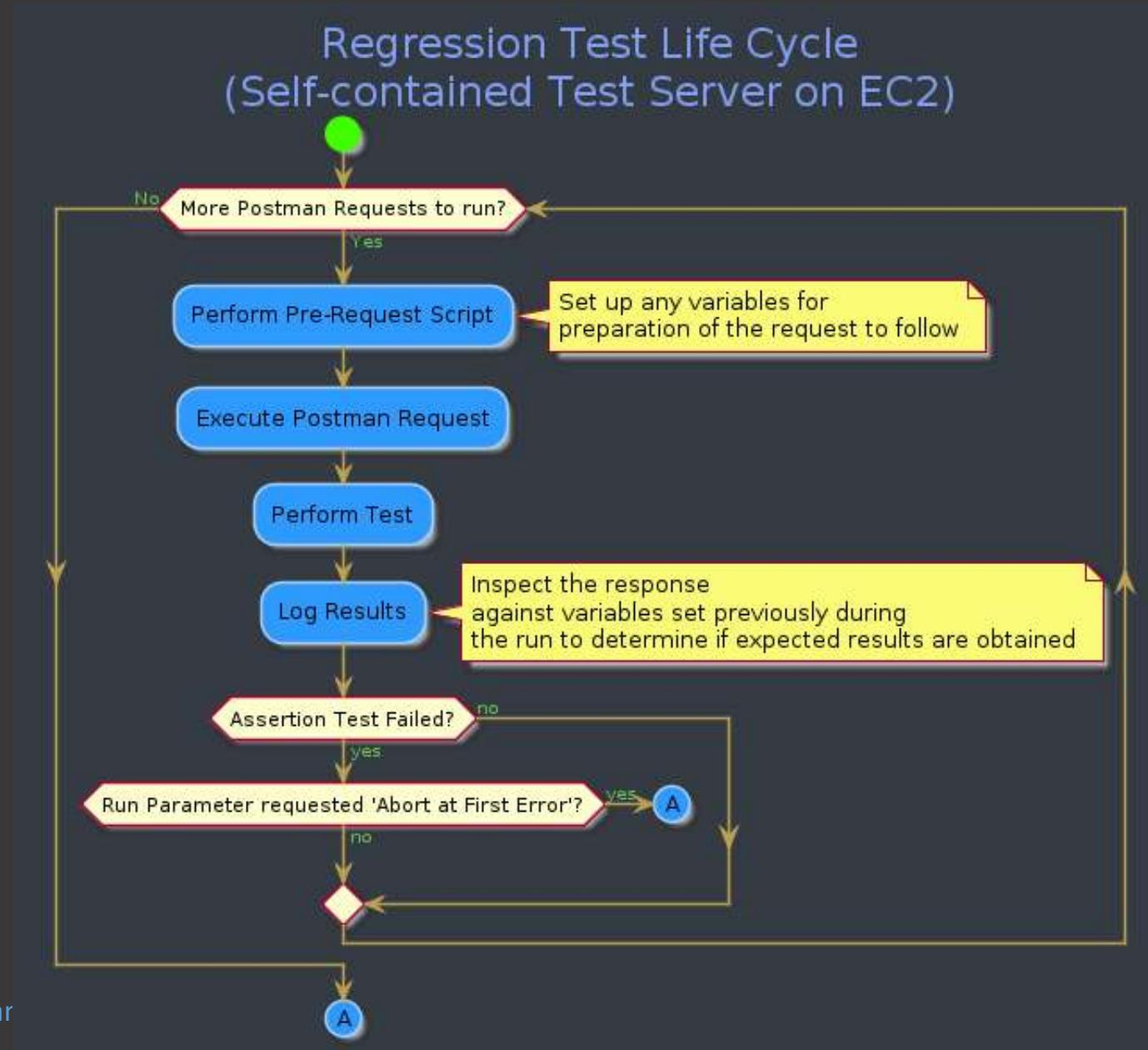
Regression Testing Info-graphic



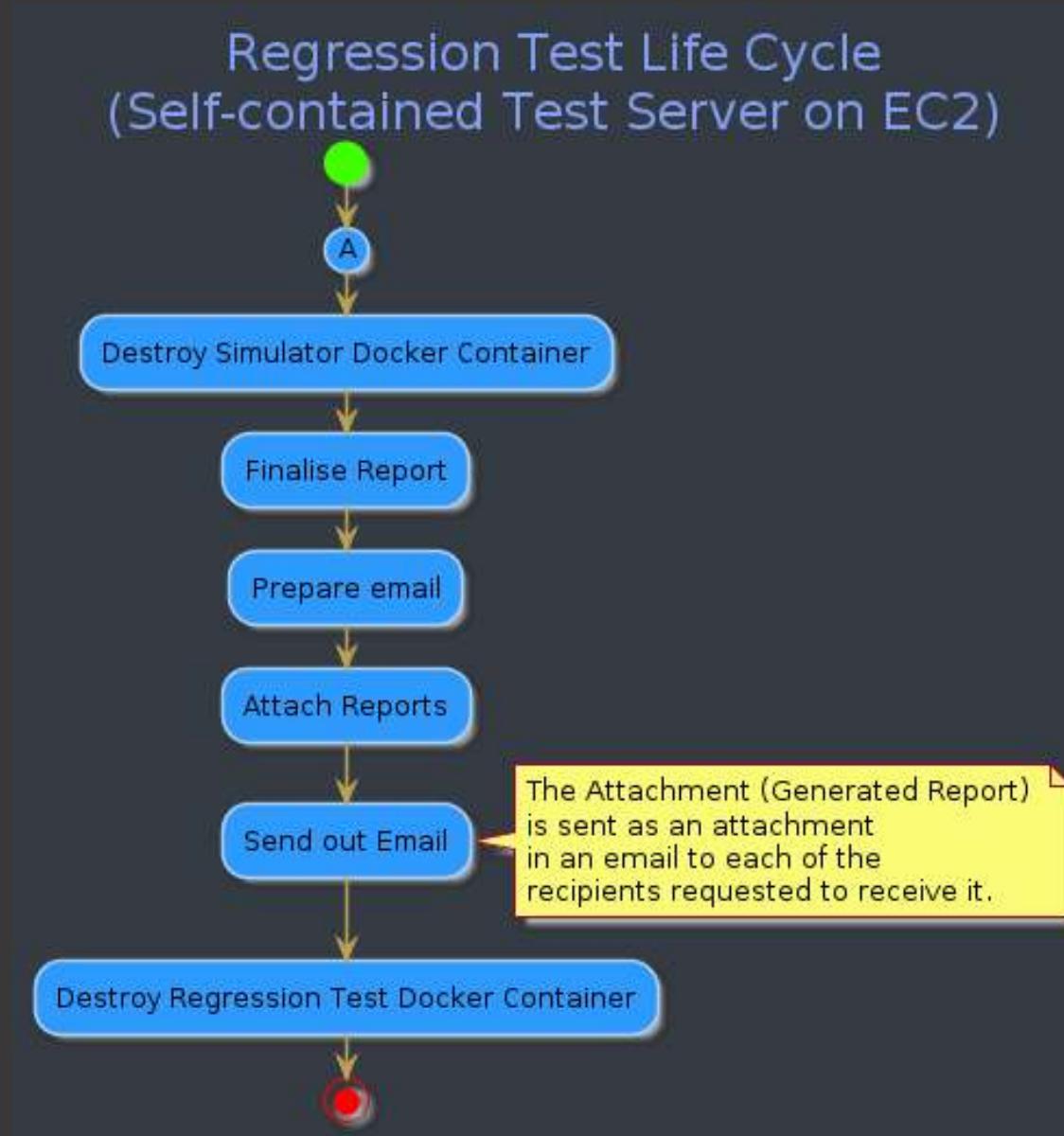
Regression Testing Process Flow – Part 1



Regression Testing Process Flow – Part 2

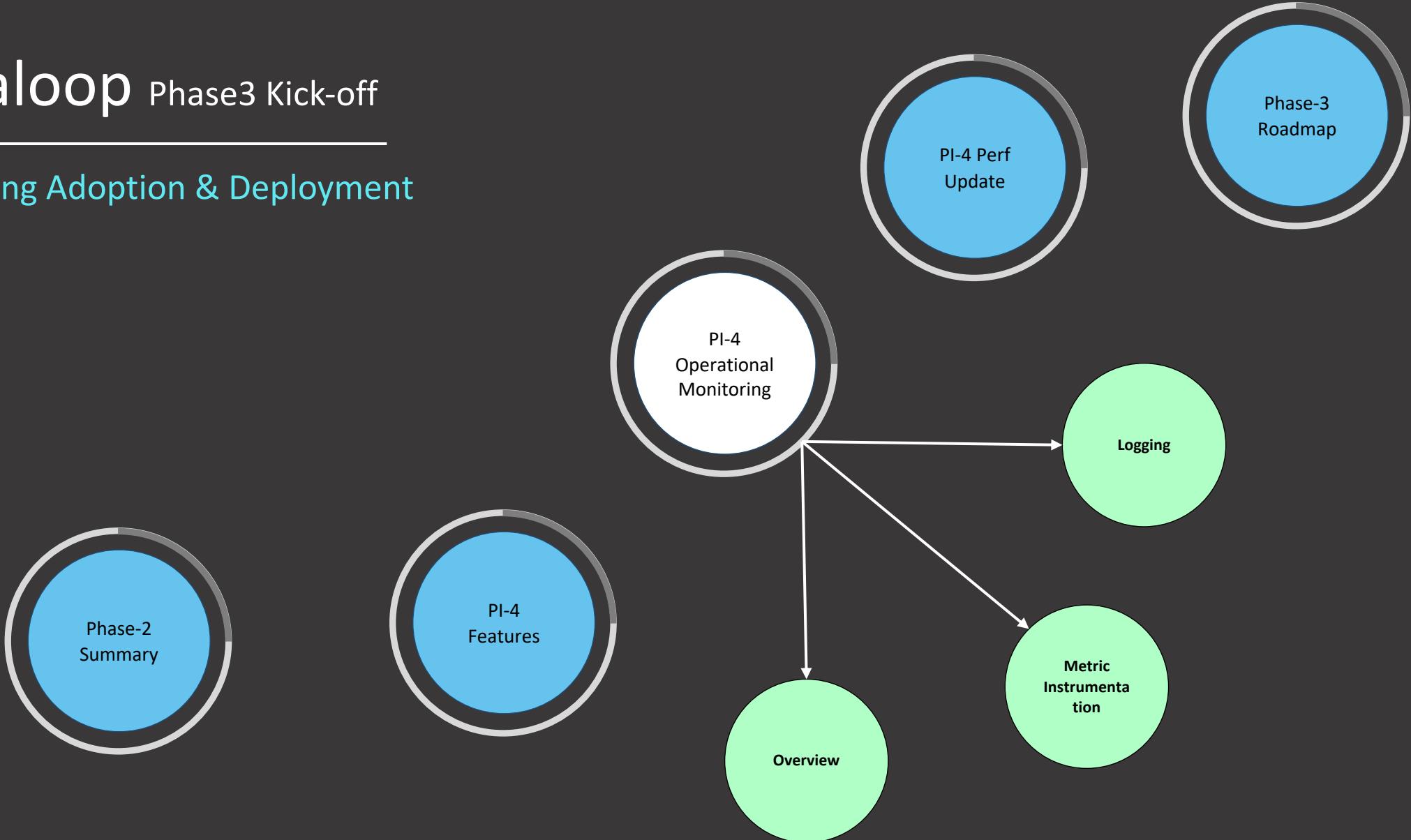


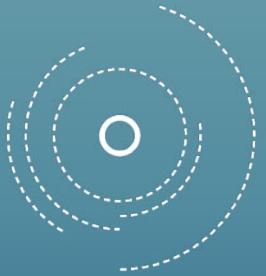
Regression Testing Process Flow – Part 3



Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment



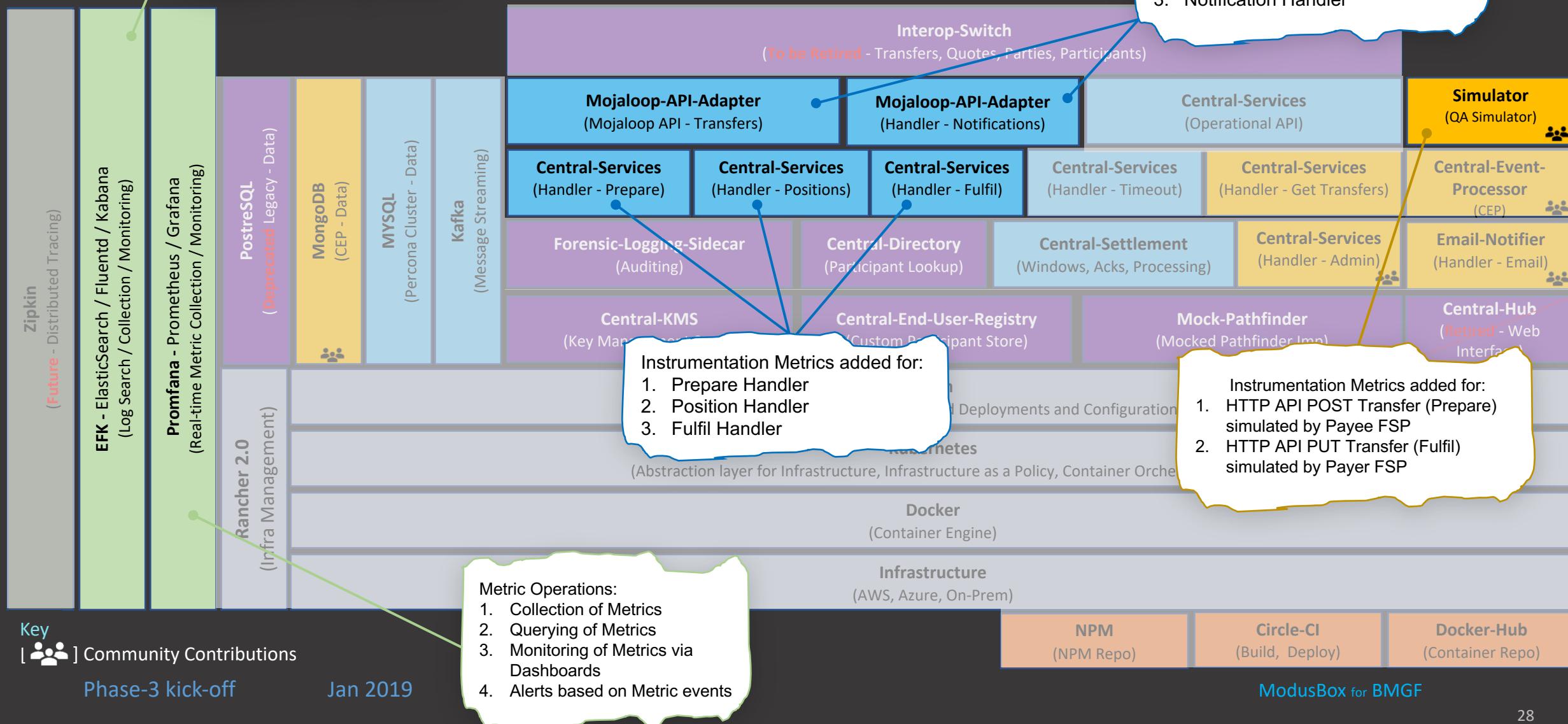


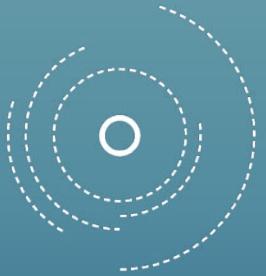
mojaloop



Operational Monitoring Overview

Operational Monitoring - Overview





mojaloop



Operational Monitoring Metric Instrumentation

Operational Monitoring - Metric Instrumentation

- 1. Overview
 - a. Motivation
 - b. Technologies
- 2. Architecture
- 3. Dashboards

Ops Metric Instrumentation - Overview

Ref: <http://prometheus.io>, <http://Grafana.com>



What is Metric Instrumentation?

Real-time operational visibility for:

- Performance
- Health
- Alerts

Why Promfana?

- Metric Instrumentation for Mojaloop
- Low overhead on nodejs (histograms + pull metric end-point)
- Real-time metric visualization for Performance and Health monitoring of the Mojaloop Stack

What is Promfana?



Leading open-source instrumentation solution for monitoring



Dimensional data
Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.



Powerful queries
PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.



Great visualization
Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.



Efficient storage
Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.



Simple operation
Each server is independent for reliability, relying only on local storage. Written in Go, all binaries are statically linked and easy to deploy.



Precise alerting
Alerts are defined based on Prometheus's flexible PromQL and maintain dimensional information. An alertmanager handles notifications and silencing.



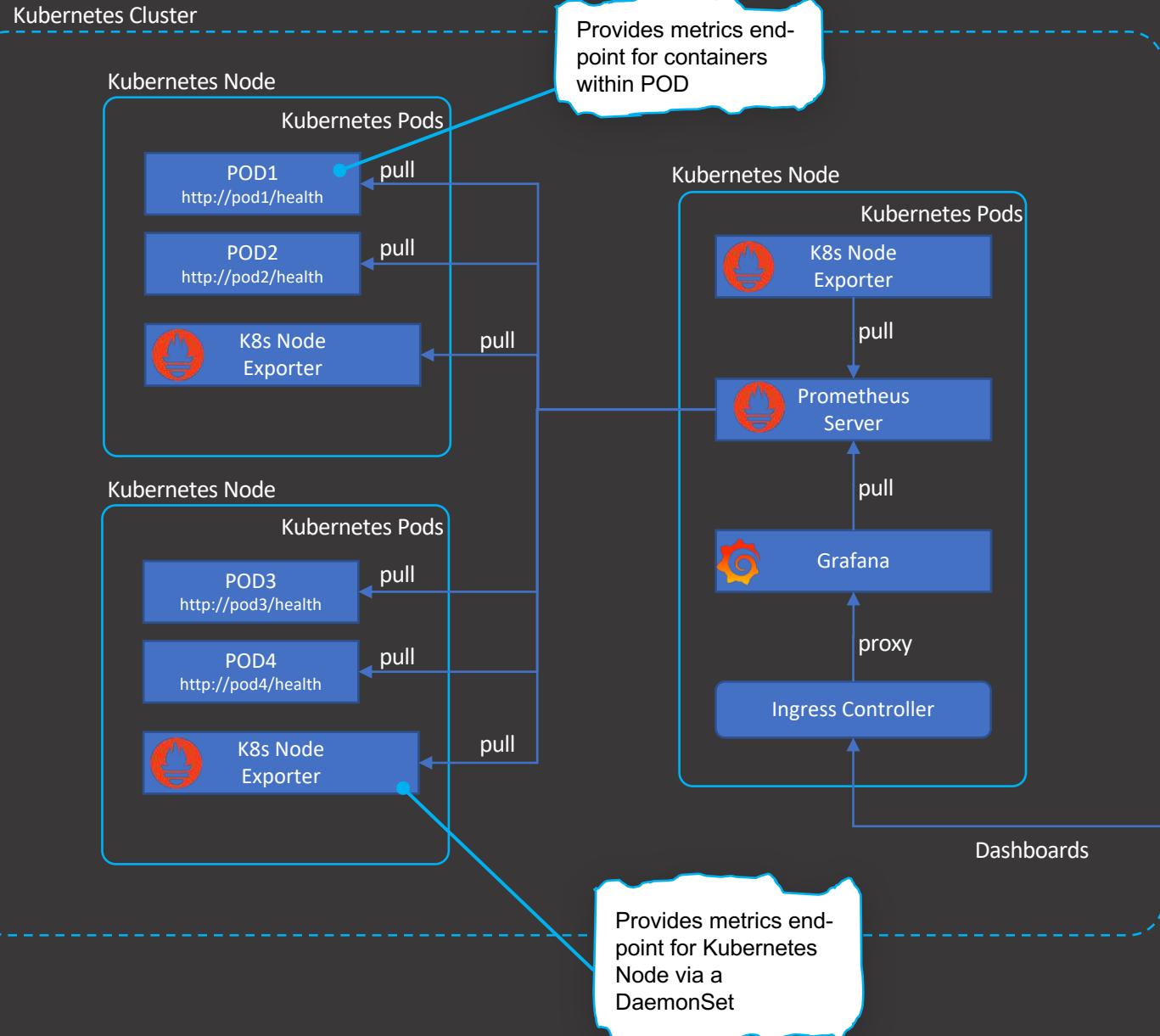
Many client libraries
Client libraries allow easy instrumentation of services. Over ten languages are supported already and custom libraries are easy to implement.



Many integrations
Existing exporters allow bridging of third-party data into Prometheus. Examples: system statistics, as well as Docker, HAProxy, StatsD, and JMX metrics.



Ops Metric Instrumentation – Architecture



Requirements:

- Each pod to expose a HTTP API for /metrics operation to provide the instrumentation

Enabling Collection of Metrics for a Pod

- Add annotation to Pod during deployment

annotations:

prometheus.io/port: "8080" <-- Specifies Port for /metrics collection
prometheus.io/scrape: "true" <-- Informs Prom Server that Pod is a target for collection

- This can be configured in each Helm chart by enabling Metrics

metrics:

enabled: true <-- By default this has been 'disabled'

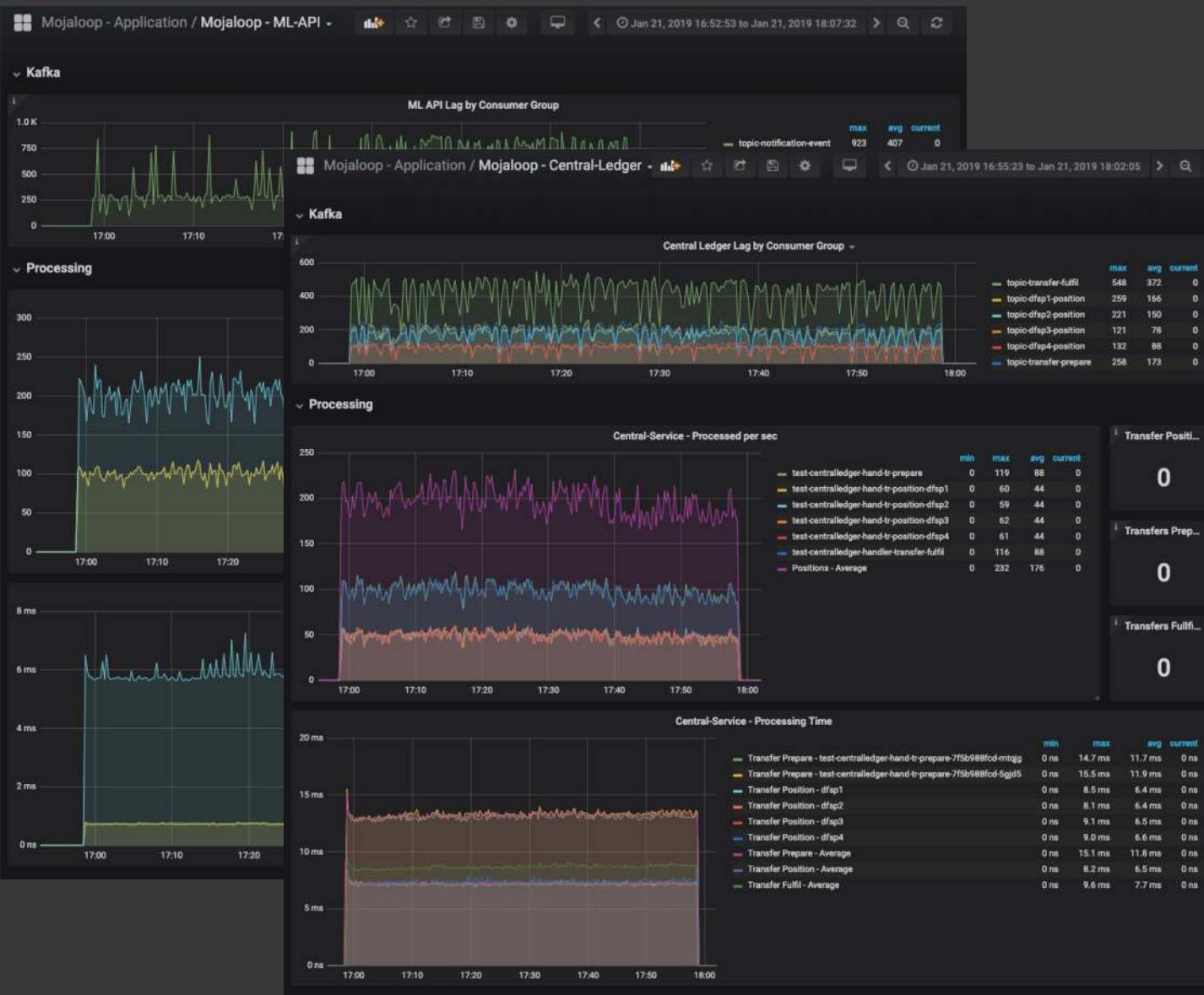
Metrics Supported Components:

Mojaloop-API-Adapter (Transfer Prepare & Fulfil)	Mojaloop-API-Adapter (Handler - Notifications)	
Central-Services (Handler - Prepare)	Central-Services (Handler - Positions)	Central-Services (Handler - Fulfil)
Simulator (QA Simulator)		



Operations

Ops Metric Instrumentation – Dashboards



Phase-3 kick-off

Jan 2019

Documentation

<http://mojaloop.io/helm/monitoring/>

Mojaloop Application

- ML-API-Adapter -- nodejs + application
- Central-Ledger -- nodejs + application
- Simulators -- nodejs + application

Data Store

- MySQL Overview
- PXC Galera Overview
- PXC Galera Graphs

Messaging

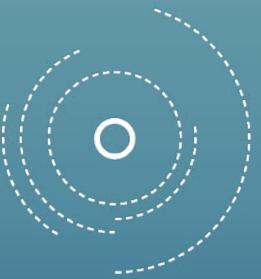
- Kafka Cluster Overview
- Kafka Topic Overview

Kubernetes

- Clusters
- Deployments

DEMO

ModusBox for BMGF



mojaloop



Operational Monitoring Logging

Operational Monitoring - Logging

1. Overview

- a. Components: ElasticSearch, Fluentd, Kibana
- b. Motivation
- c. Usage

2. Deployment Architecture

Ops Logging – Overview

Ref: <http://prometheus.io>, <http://Grafana.com>

What is EFK (aka ELK)?



elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine.



fluentd

Open source data collector for unified logging layer, with ingestions into Elasticsearch.



kibana

Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack.

Zipkin
(Future - Distributed Tracing)

EFK - Elasticsearch / Fluentd / Kibana
(Log Search / Collection / Monitoring)

Promfana - Prometheus / Grafana
(Real-time Metric Collection / Monitoring)

Why EFK?

- Central location and storage of all Mojaloop log files
- Management of log data (persistence, long-term storage, etc)
- Management of alert/events based on log data
- Assist with tracing & trouble shooting Mojaloop's distributed micro-service logs
- Produces real-time metrics or alerts to Prometheus

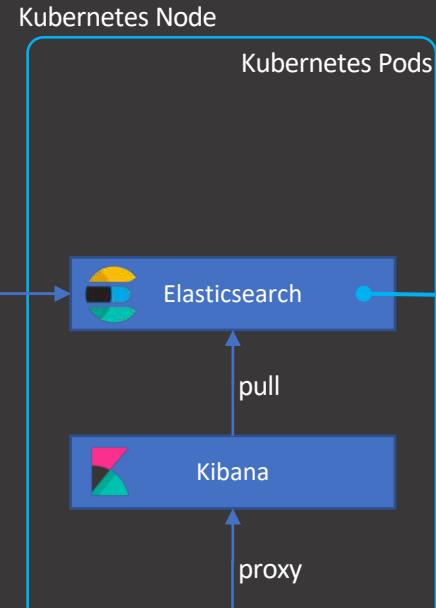
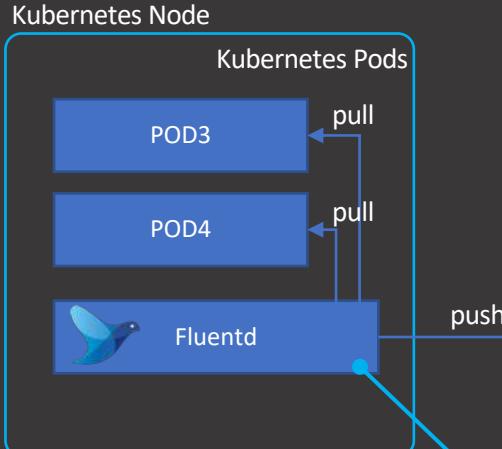
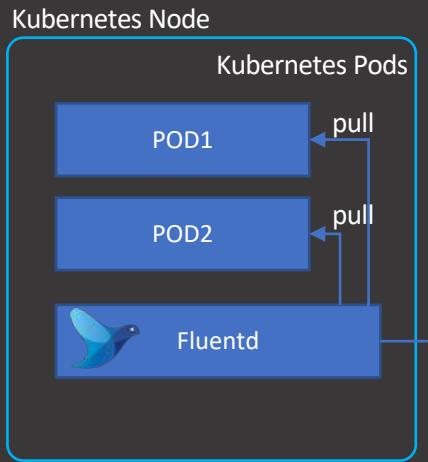
Why E-F-K and not E-L-K?

- **Fluentd** is used instead of **Logstash** due to its support & seamless integration for Kubernetes (k8s).
- K8s Pods/Containers are easily collected by **Fluentd** and ingested into Elasticsearch using the underlying K8s logging architecture.



Ops Logging – Architecture

Kubernetes Cluster



Phase-3 kick-off

Jan 2019

Fluentd collects logs
using the Kubernetes
API for each of their
respective Nodes

Requirements:

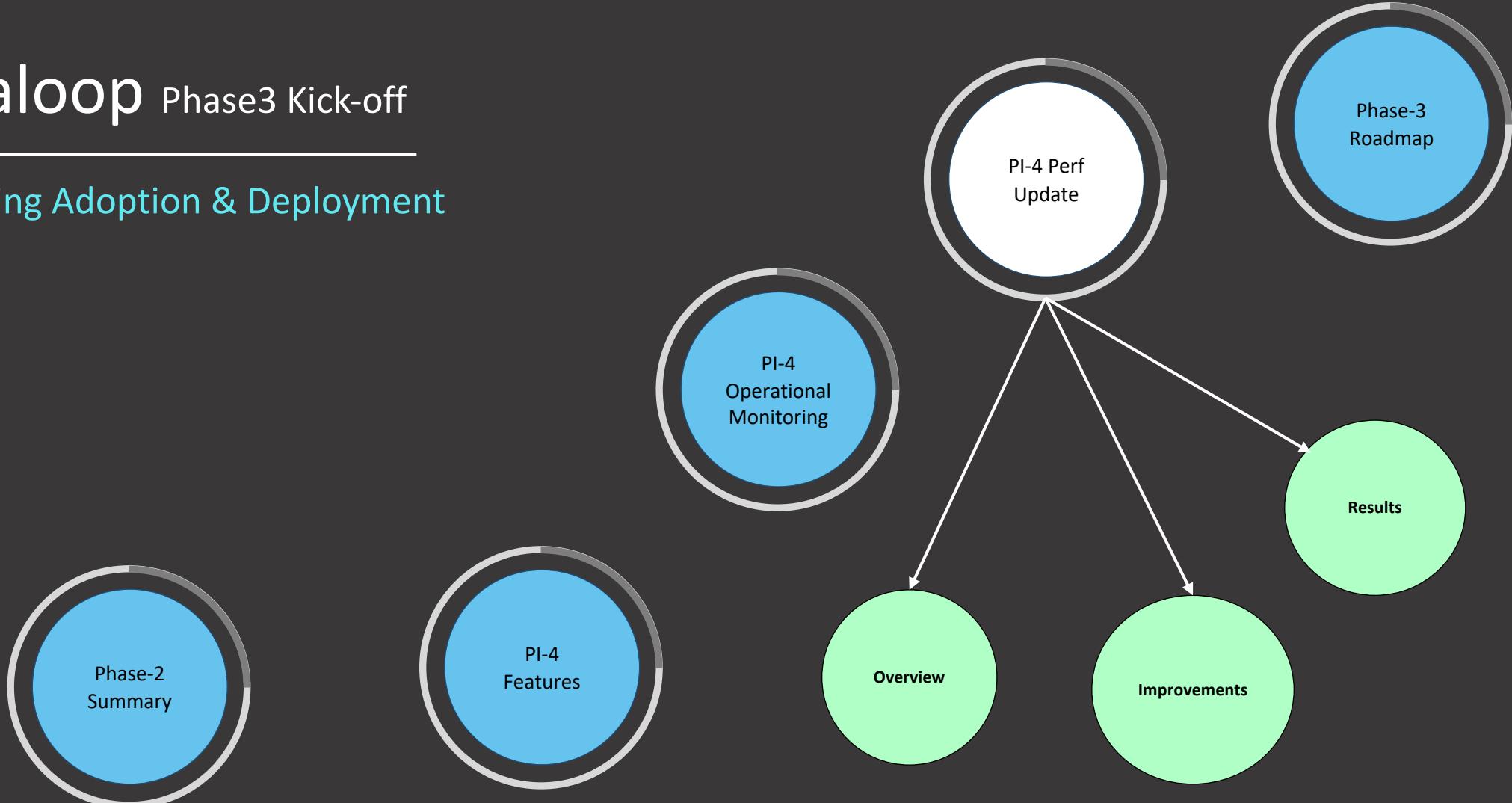
- Each pod must write logs to `stdout` so that it is accessible via the Kubernetes Log API

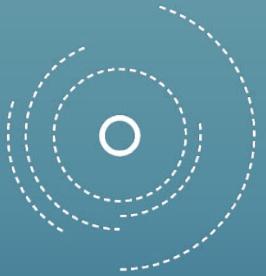
Logs are ingested, and
parsed into a
meaningful
representation which
allows for easier
searching & filtering.



Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment





mojaloop



Performance Update Overview

Phase-3 kick-off

Jan 2019

ModusBox for BMGF



39

Performance Improvements

1. PI4 Performance Summary

- a. Approach
- b. Scenarios
- c. Numbers
- d. Optimizations

2. PI4 Next Steps

PI4 Performance Summary

Key
[●] Completed in PI-4
[●] Partially completed in PI-4
[●] To be considered in Future

Approach:

- [●] Set a base-line on PI-3 Code-base
- [●] Production grade environment
- [●] Identified & fix stability issues due to failure on load
- [●] Optimized Kafka Client (Producers, Consumers) for Performance
- [●] Scenarios with Multiple FSPs (4x)
- [●] Increase multi-threaded processing of non-¹EOS Kafka Consumers (Prepare, Fulfil, Notification)
- [●] Optimized Handlers with reduced CPU Usage
- [●] Optimized Handlers with reduced IO
- [●] Tested against an alternative hosted Kafka service²
- [●] Increased scalability of solution
- [●] Optimized Handlers with PRISM

Comparison of Selected Results (in seconds):

Metric	PI2	PI3	PI4 Switch
Number of Messages	100	100	358 201
Prepare Process	✓	✓	✓
Fulfil Process	✗	✗	✓
Transactions Per Second	6.51	22.18	207
Average Transaction Time	8.21	1.65	0.101
Shortest Response Time	2.594	0.942	0.32
Longest Response Time	19.746	2.589	3.047
Standard Deviation	n/a	n/a	0.119
% of requests taking longer than 1 second	n/a	n/a	0.002%
Duration	15 seconds	4.5 seconds	1 hour

¹ EOS – Exactly Once Semantic,

² Mojaloop Kafka deployment on Rancher K8s-Phoenix Environment was comparable and better at times,

³ Average over 3 runs – individual results in the appendix

PI4 Performance Next Steps

Key
[●] Completed in PI-4
[●] Partially completed in PI-4
[●] To be considered in Future

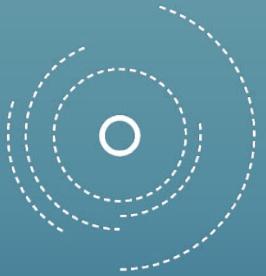
Next Steps:

- [●] Test Different Kafka Consumer Modes & Libraries (Producers, Consumers) for Performance (only tested end-to-end in recursive mode)
- [●] Test new version of Node-rdkafka for Performance (released Dev 2018 - v2.5)
- [●] Optimize Kafka Producers for Local Queue Issue
- [●] Optimize Handlers with PRISM¹ (or similar approaches)
- [●] Test/Optimize scalability against current baseline (4xFSPs)
- [●] Test/Optimize scalability against 4+ DFSPs
- [●] Implement full Batch Processing of Kafka Consumers
- [●] Automate Performance Tests into CI/CD pipeline
- [●] Re-factor Kafka Consumers once EOS is natively supported
- [●] Validate performance results against selected Mojaloop implementations

On-going:

- [●] Configure Kafka Client (Producers, Consumers) for Performance
- [●] Optimize Kafka Server performance

¹ PRISM – Persisted Replicated Internal State Messaging



mojaloop



Performance Update Improvements

Phase-3 kick-off

Jan 2019

ModusBox for BMGF



43

Performance Improvements

1. Optimized Prepare Handlers
2. Optimized Position Handlers
3. Optimized IO
4. Optimized CPU Usage on Handlers
5. PRISM Optimization

Optimized Prepare Handlers

Key

API Adapter

Kafka Consumer

Kafka Topic

PI3

- Ordering was important for prepares

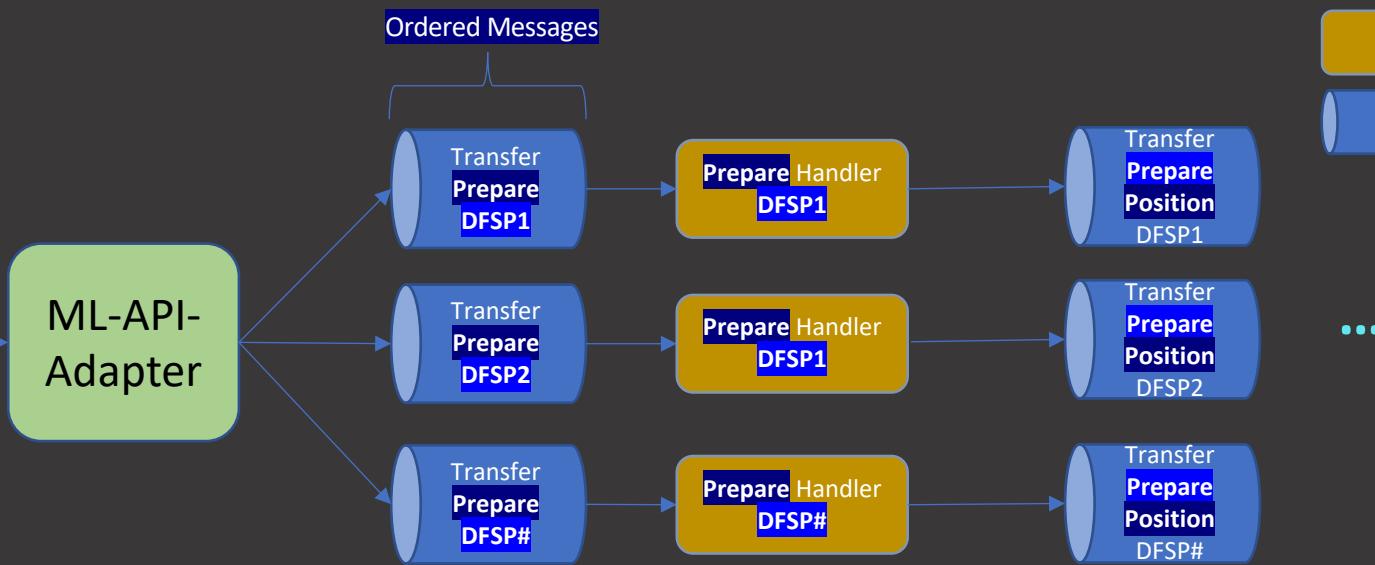
Transfer
Prepare



PI4

- Ordering not important for prepares
- Reduced CPU usage
- Easier to maintain
- Improved scalability

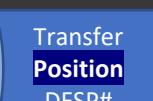
Transfer
Prepare



Phase-3 kick-off

Jan 2019

Ordered Messages



...

ML-API-Adapter

Transfer
Prepare

Prepare Handler

Optimized Position Handlers

Key

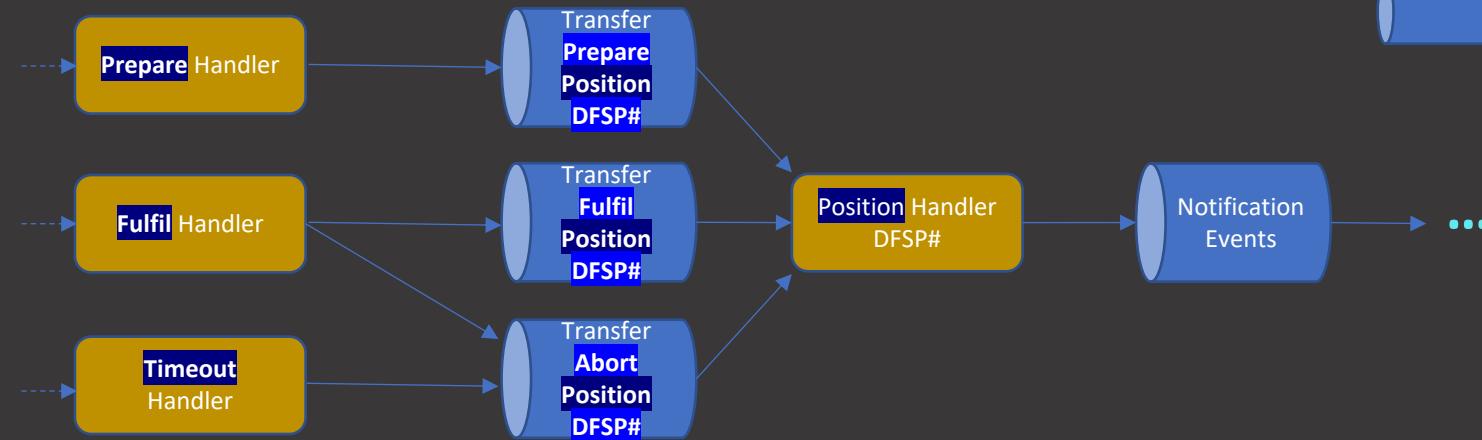
API Adapter

Kafka Consumer

Kafka Topic

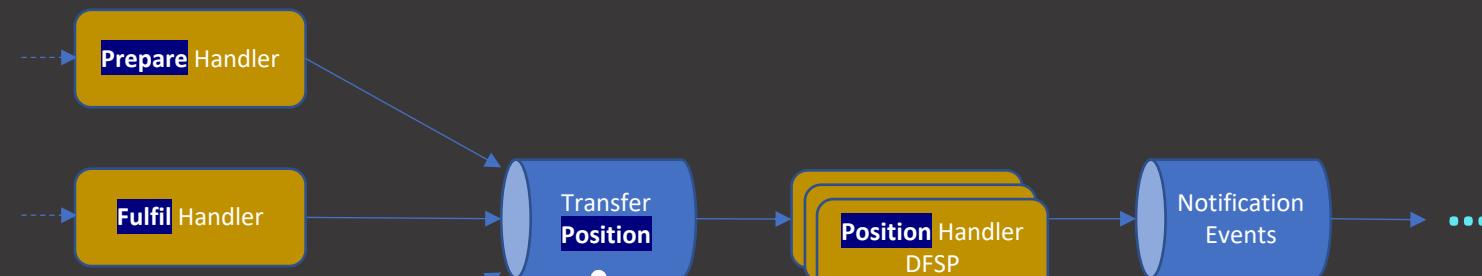
PI3

- Messages for each DFSP and Action (Prepare, Fulfil, Abort) is routed to a specific Topic
- Each Position Handler internally uses 3 Consumers for each DFSP Topic within the same thread Pool = 3x CPU Usage



PI4

- Message keys are used to route each message to a specific Partition ← This guarantees ordering and the integrity of the FSPs transfer process
- CPU usage reduced by three-fold
- Easier to maintain
- Improved scalability
- *Possible issue/limitation*:
 - Increasing partitions on Kafka can only be done when there no active consumers or the ordering will be an issue.
 - Mitigation:
 - Create more than required Partitions up-front for future DFSP growth
 - Create versioned topics



Message Key (DFSP Id) is used to assign a message for each DFSP to a specific partition

Optimized IO

Description:

- Slow response times observed on Handlers due to high IO

Resolution:

- Handlers:
 - Manual Commits were causing excessive high IO thread contention which was drastically reduced by enabling auto-commits
 - Optimized Kafka configurations to reduce unnecessary IO
 - Loggers were moved into an async process

(20% improvement)

Considerations:

- Additional logic will need to be added in Handlers as the auto-commits allows for re-processing of messages. The handlers will not re-apply any business logic as the “state” of the transactions will have moved on, however it is possible that there will be unwanted notifications sent to the FSPs. Thus we will have to store the Kafka index for each message to validate if the message is a re-request from FSP or a re-processing of a previous message.

PoC Producer - Commit = Manual

Number of unique matched entries: **10000**
Total difference of all requests in milliseconds: 1308
Shortest response time in millisecond: 0
Longest response time in millisecond: 3
Mean/The average time a transaction takes in millisecond: **0.1122**
Standard **deviation** in milliseconds: **0.3206**
Number of entries that took longer than a second: 0
% of entries that took longer than a second: 0.00%
Average **transactions per second**: **7645.26**



PoC Producer - Commit = Auto

Number of unique matched entries: **10000**
Total difference of all requests in milliseconds: 1234
Shortest response time in millisecond: 0
Longest response time in millisecond: 3
Mean/The average time a transaction takes in millisecond: **0.1023**
Standard **deviation** in milliseconds: **0.3080**
Number of entries that took longer than a second: 0
% of entries that took longer than a second: 0.00%
Average **transactions per second**: **8103.73**

PoC Consumer - Commit = Manual

Number of unique matched entries: **10000**
Total difference of all requests in milliseconds: 16154
Shortest response time in millisecond: 0
Longest response time in millisecond: 3
Mean/The average time a transaction takes in millisecond: **0.1116**
Standard **deviation** in milliseconds: **0.3187**
Number of entries that took longer than a second: 0
% of entries that took longer than a second: 0.00%
Average **transactions per second**: **619.04**



PoC Consumer - Commit = Auto

Number of unique matched entries: **10000**
Total difference of all requests in milliseconds: 8852
Shortest response time in millisecond: 0
Longest response time in millisecond: 3
Mean/The average time a transaction takes in millisecond: **0.1018**
Standard **deviation** in milliseconds: **0.3063**
Number of entries that took longer than a second: 0
% of entries that took longer than a second: 0.00%
Average **transactions per second**: **1129.69**

Optimized CPU Usage on Handlers

Description:

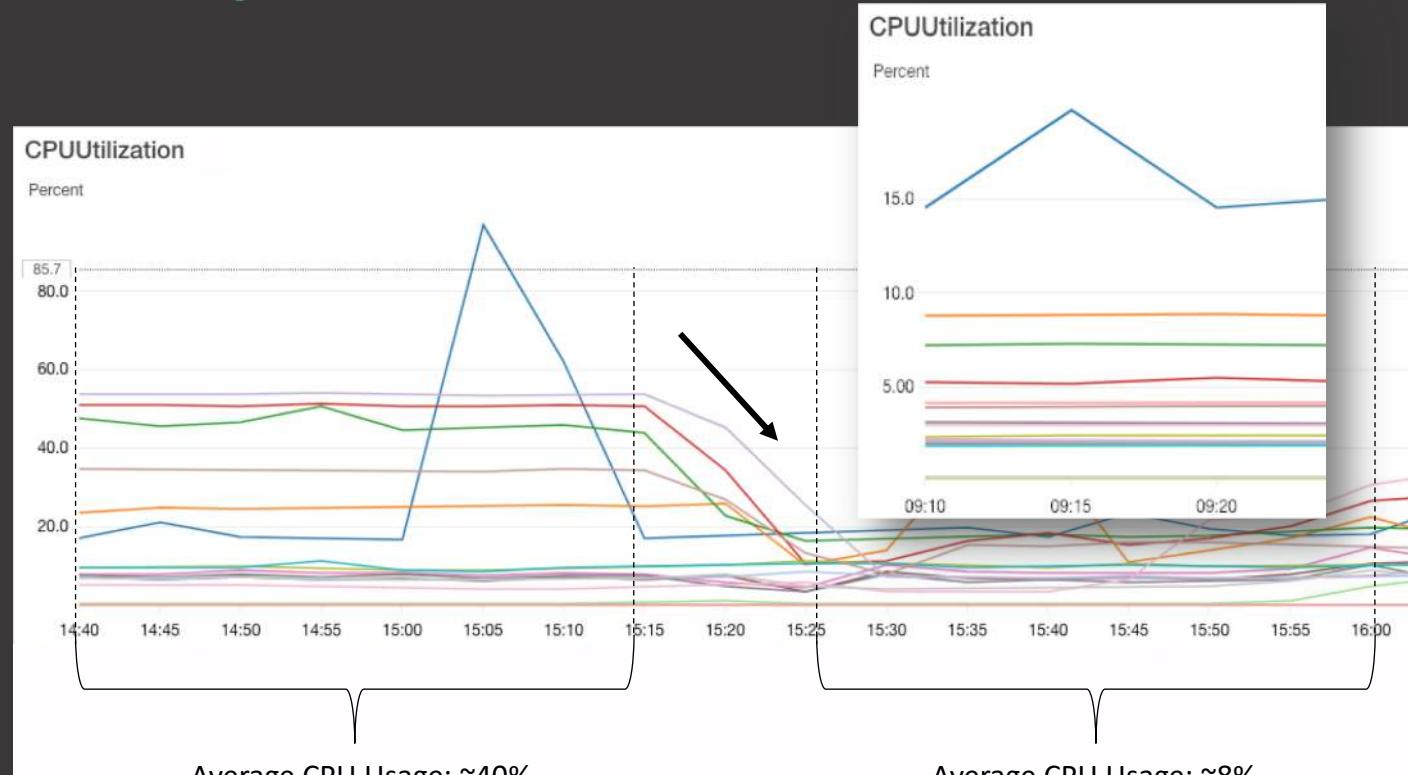
- Excessive CPU usage on handlers, specifically the position handlers was observed consistently

Resolution:

- All nodejs processors
 - Async logging (Max CPU ~40% -> ~20%)
- Position Handlers
 - Single Topic – Combined topics Abort, Prepare and Fulfil topics
 - Applied optimized config for Producers
 - Applied optimized config for Consumers
- Prepare Handlers
 - Combined separate Topics for each DFSP into a single topic to allow for multiple partitions
 - Applied optimized config for Producers
 - Applied optimized config for Consumers
- Fulfil Handlers
 - Applied optimized config for Producers
 - Applied optimized config for Consumers
- Notification Handlers
 - Applied optimized config for Producers
 - Applied optimized config for Consumers

Phase-3 kick-off

Jan 2019



Key

- [●] Completed in PI-4
- [●] Partially completed in PI-4
- [●] To be considered in Future

ModusBox for BMGF

PRISM Optimization

Meaning:

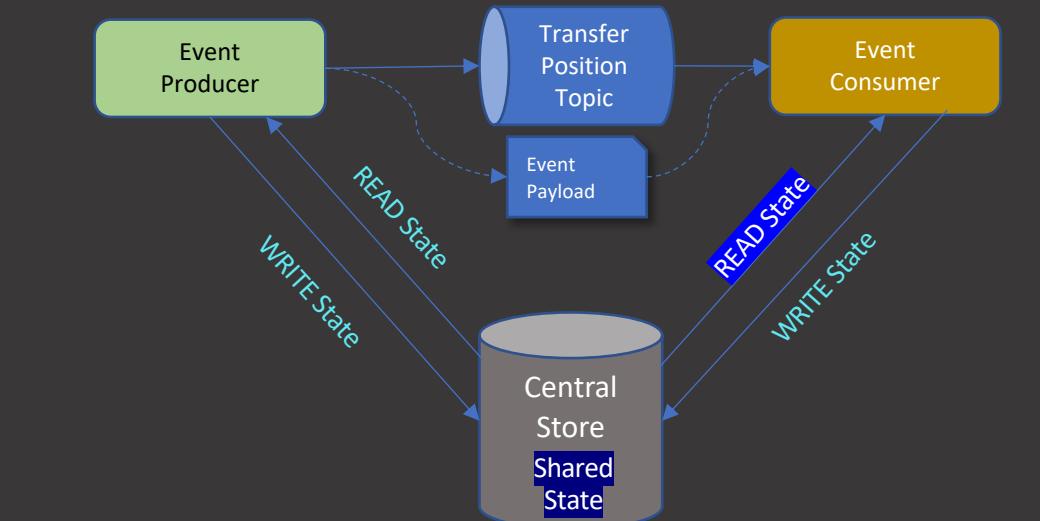
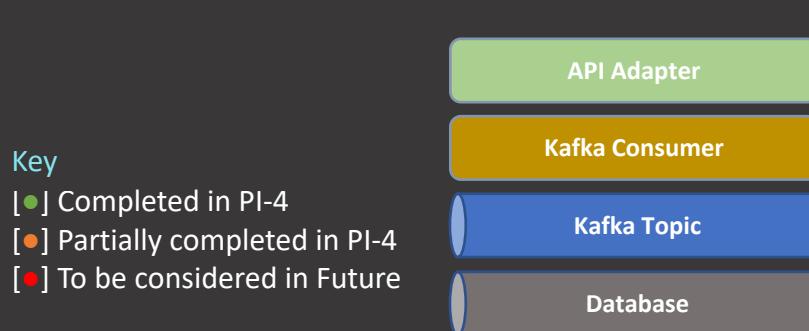
- PRISM - Persisted Replicated Internal State Messaging

Description:

- Kafka messaging system is used for stateful enrichment of messages to down-stream handlers

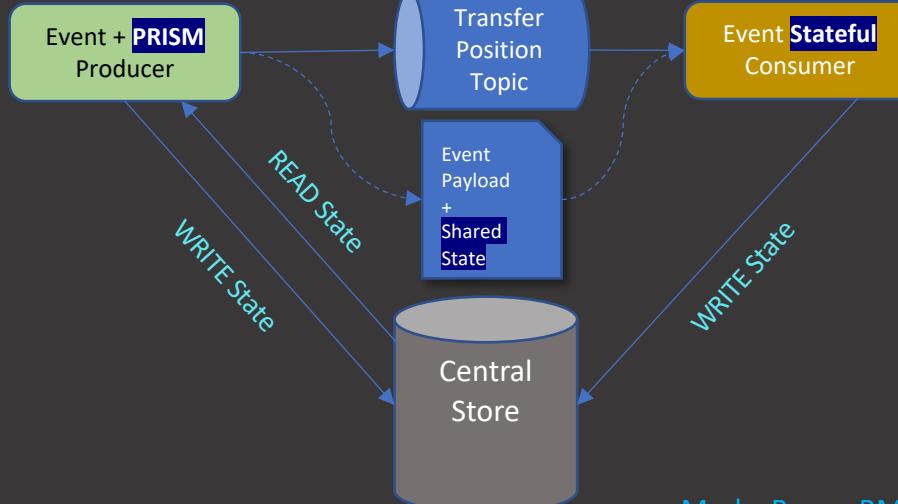
Considerations:

- We can minimize unnecessary locks between the Prepare, Position, Fulfil and Timeout Handlers by using PRISM to pass the necessary information to the down-stream handlers
- Timeout handler logic will need to be moved to the Position Handler ← “Position Handler” should perhaps be renamed to the “Transfer Handler” as it handled events for position change, rejection core-logic, etc.

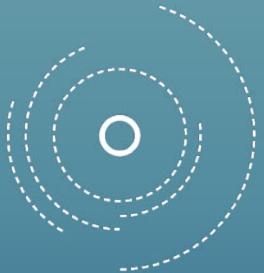


Previous PI3

PI4 / Future



ModusBox for BMGF



mojaloop

PI-4 Features Settlements Update

Settlement Business Challenges

1. Movement within the Switch
2. Understanding Accounting Principles
3. Managing the risk in the system
4. Changes Undertaken
5. Checking NDC for a sender
6. Delayed Settlement Risk
7. Key Learnings

Movement within the Switch

- Funds need to be moved in the various accounts within the switch – safe in the knowledge that funds are available to support the transfer
- The following Controls are in place
 - Net Debit Cap
 - Position – how much a DFSP may owe or be owed
 - An increase in the position reflects the increase in the Liability of the Payer
 - A decrease in the position reflects the decrease in the Liability of the Payee

Accounting Principles

- The Mojaloop platform accounts for the financial position of the switch and therefore any movements are recorded from that perspective
- We also need to be careful with the language we use when talking about accounts. An Increase and a Decrease depends on the type of account that is being reviewed. An increase is not synonymous with more positive (Dr) or negative (Cr) transactions.
- The DFSP sees an increase in their liability when recording the transaction in their systems, whilst the Switch records the transaction as an increase in their Asset. For the switch the transaction should be immediately balanced with an increase in the liability to the receiving DFSP once the transfer is committed.

Depending on an Account – Increase means...

- The mnemonic **DEADCLIC** is used to help remember the effect of debit or credit transactions on the relevant accounts. **DEAD**: Debit to increase Expense, Asset and Drawing accounts and **CLIC**: Credit to increase Liability, Income and Capital accounts.
- The account types are related as follows:
current equity = sum of equity changes across time (increases on the left side are debits, and increases on the right side are credits, and vice versa for decreases)

current equity = Assets - Liabilities
- sum of equity changes across time = owner's investment (Capital above) + Revenues - Expenses
- From <https://en.wikipedia.org/wiki/Double-entry_bookkeeping_system>

	Debit	Credit
Asset	Increase	Decrease
Liability	Decrease	Increase
Income (revenue)	Decrease	Increase
Expense	Increase	Decrease
Capital	Decrease	Increase

Understanding Accounting Principles

- Standard Process in accounting Software
 - Debits (Drs) are Positive
 - Credits (Crs) are Negatives.
- When adding up
 - if the net position is positive (More Drs) they are treated as an asset
 - if the net position is negative (More Crs) they are treated as a liability.
- The Switch has a different perspective to the DFSPs but follows the same rules:
 - Funds in - the cash (asset) increases, is a debit;
 - The change in the customer's account balance (liability) is a credit.
 - When a transfer takes place, the Dr is applied to the sender - it will lead to a reduction in the amount that is owed back to the DFSP (reduced liability).

New Accounts in the System

- The purpose of the new accounts is as follows
 - Position Account – Facilitates the Debtors book per currency per DFSP
 - Hub Multi-Lateral Net Settlement (HMLNS) Account – Used to balance Debtors settlements
 - DFSP Settlement account – Reconciliation account for external movements of funds
 - Hub Reconciliation Account – Control account – Net Position across all DFSP settlement accounts
- Certain types of transfers allowed in each account
 - P2P Transfers – Position account Only
 - Settlement Transfers – Position Account and HMLNS
 - Fund Transfers (in and Out) - DFSP Settlement account and Hub Reconciliation Account

Managing the risk in the system

- Dr reduces the switches liability, so it will always be applied in the Reservation ring fenced first. Applying this to the System Treatment of a P2P Transfer
 - Reserve state - the Position of the Payer DFSP is Debited (Dr)
 - Commit state - the Position of the Payee DFSP is Credited (Cr) – failure in receiving the funds leads to liability issues.
- Standard approach - confirmation expected before funds are released also lead to consistency in reporting, reconciliation or roll back.
- This principle is applied to all transfers, potential loss of funds due to change reviewed as part of the flow – assessing if transactions that may occur whilst the process is underway do not expose the switch to a financial risk.

Changes Undertaken

- We have introduced options on the relationship with real money flows
 - Real time settlement
 - Delayed settlement
- Which required the addition of 3 more account types
 - DFSP Settlement Ledgers – to mirror funds in the real bank account
 - Hub Multilateral Settlement Ledger
 - Hub Reconciliation Ledger
- Each have their own risk which has been factored in Design

Checking NDC for a Sender

Activity	DFSP1 Transfer Ledger			DFSP1 Settlement Ledger	
	Transfer	Position	NDC	Transaction	Balance
1 Settlement Window Closed		+350	+800		-1000
2 DFSP1 Settlement Processed	DR	+350	+800		-1000
5 DFSP1 Position Reset	CR	-350	0	+800	-1000
6 Funds Transfer For Settlement - DFSP 1			0	+800	+350 -650

Step 1 is the closure of the Window

Steps 2-5 are the steps to settle

Steps 6 and 7 are the real movement of funds.

However when the money flows the NDC is now higher than the Settlement Ledger balances – Automation of these steps requires a change in the NDC

The NDC adjustment needs to be applied before any processing of the settlement window – to ensure the control is in place before any positions are reset. This could lead to a temporary block on transfers, but eliminates the risk that transfers are allowed before the settlement is processed

Note that the balance of transfers is done in the Hub Multilateral Settlement Ledger to ensure reconciliation, but it also allows for the settlement amounts to be handled differently

In delayed settlement a new risk is introduced

Net Receiver becomes Net Sender

Activity	DFSP2 Transfer Ledger			DFSP2 Settlement Ledger	
	Transfer	Position	NDC	Transaction	Balance
1 Settlement Window Closed		-350	+800		-1000
2 DFSP2 → DFSP1 \$600	Allowed	900	550	+800	-1000
5 DFSP2 Position Reset	DR	+350	900	+800	-1000
6 DFSP2 Settlement Processed	CR		900	+800	-350
					-1350

If the time between Step 1 and Step 6 is large, the likelihood of this impact is increased.

- Envisioned scenario is a Natural disaster over the weekend that reverses the normal flows before the settlement can be processed
- The settlement Windows is closed on Friday Evening, and the settlement and resetting of the Position does not happen until the following Monday or Tuesday (in the case of a Bank Holiday weekend)
- Before the position is reset, the DFSP had sufficient funds, but the reset would immediately block their ability to continue sending

Key Learning

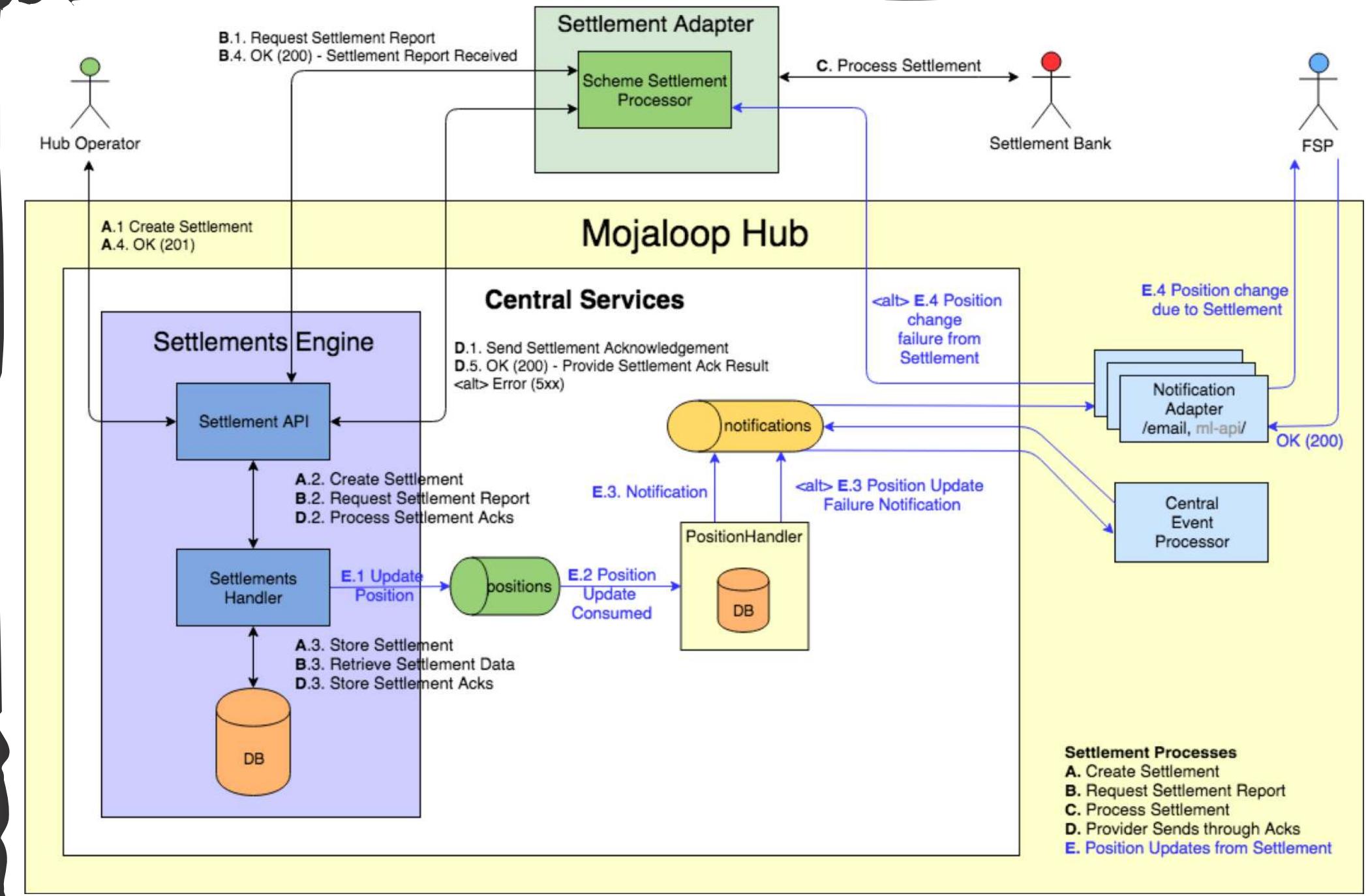
- Get a good basic understanding of Double Entry book keeping
- Model any transfers or movements via T accounts first
- Then extend the model to a spreadsheet
- Work through the transfers and put in some rogue transactions to ensure you have adequate

Settlements Functionality

1. Phase-2 Overview
 - a. Settlement Windows
 - b. Settlements API
2. PI-4 Update
3. Settlements Demo Index

Settlements Solution Design

Phase-3



Settlement Windows

Usage benefits:

1. Continuity
2. Isolation
3. Manageability

Note: No changes were made to the settlement windows states flow.

Settlements API

1. POST */settlementWindows/{ID}* – current window closure
2. POST */settlements* – trigger a settlement for **CLOSED / ABORTED** windows
3. PUT */settlements/{id}* – update settlement accounts
 - a. PUT */settlements/{id}/participants/{id}/accounts/{id}*
 - b. PUT */settlements/{id}/participants/{id}*
4. GET endpoints provide information about settlements and windows

Settlements Redesign

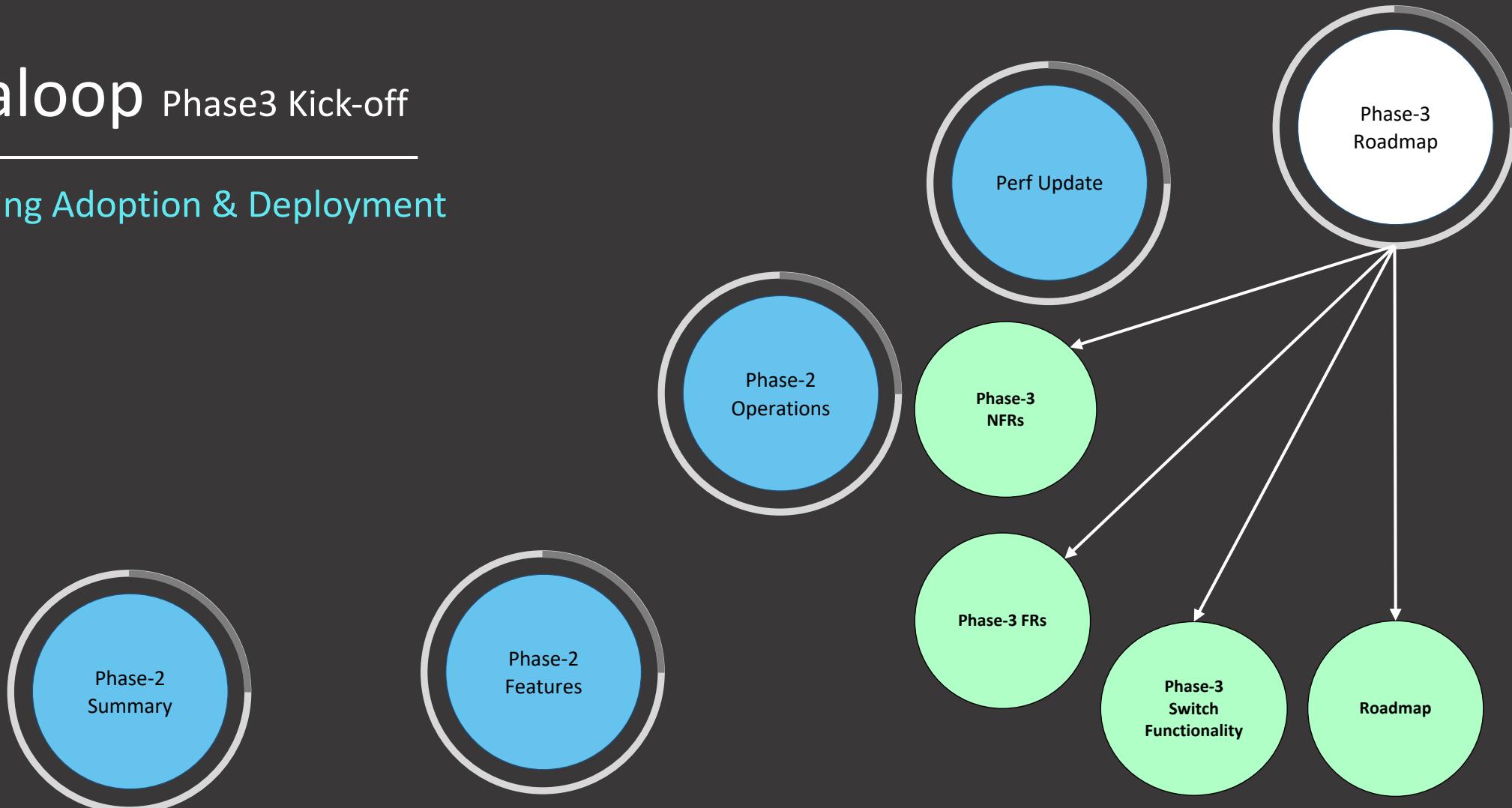
1. Database Schema Changes: One new column
`settlementParticipantCurrency.settlementTransferId` added
2. Enumeration Additions:
 - a. *ledgerAccountType*: `HUB_MULTILATERAL_SETTLEMENT`, `HUB_RECONCILIATION`, `SETTLEMENT`
 - b. *settlementState*: `PS_TRANSFERS_RECORDED`, `PS_TRANSFERS_RESERVED`,
`PS_TRANSFERS_COMMITTED`, `SETTLING`
3. Settlement Process & Settlement Transfer

Settlements Demo Index

1. 10 transfers (\$100 ea.) between 3 participants
2. dfsp1 → dfsp2: 2 transfers, dfsp2 → dfsp3: 5, dfsp3 → dfsp1: 3
3. Balance: dfsp1 -100 (recipient), dfsp2 +300 (sender), dfsp3 -200 (recipient)
4. Golden path for end-to-end settlement
5. Notes:
 - a. Same-state Account Updates and Settlement Abort
 - b. Observables: API GET endpoints and DB queries

Mojaloop Phase3 Kick-off

Supporting Adoption & Deployment



Phase-3 Roadmap: NFRs

PI	Summary
Phase - 3	<ol style="list-style-type: none">1. DFSP Handler Provisioning2. API Gateway3. Implementing Topology Guidelines4. Automated Testing: Integrated Functional tests5. Event Logging Framework6. Error Handling Framework7. DevOps

1. DFSP Handler Provisioning

1. As new DFSPs are on-boarded, it is necessary to
 - a. Provision the Kafka topics
 - b. Position Handlers
 - c. the underlying data structures

This forms the basis for the future horizontal scaling for high-throughput DFSPs.



2. API Gateway

1. To support the onboarding of DFSPs, SIs, and potential Hub operators. A base API gateway (or developer portal) for Mojaloop implementers to customize, including the orchestration for the onboarding provisioning for DFSPs.
2. To support the community by making lab facilities easily and securely accessible; The PoC work from Phase 2 PI-2 and the subsequent commercial partner work using WSO2 will be leveraged to create a developer portal v0.
3. Onboarding of developer/system authentication, service documentation, and exposure of Mojaloop APIs via the portal will allow the deployed labs, such as GSMA's Mojaloop Lab, to provide easy access and demonstrations to their industry SI and DFSPs

3. Implementation Topology Guidelines

1. As commercial partners are experimenting with Mojaloop laboratories and production ready deployments, the community should develop recommended topologies across the on-premise and cloud hub operators that meet security and performance requirements while realizing the benefits of the Mojaloop design principles.
2. This would include workshops with existing implementers and designers to ensure actionable guidelines based on practical experiences are made available



4. Automated Testing

1. A robust set of functional tests is required to ensure that third parties can adopt the code base into their solution. This needs to be executed as a standard component of the Mojaloop Community CI/CD process.
2. Both - for core development of the platform allowing for full automated testing during the development cycle; and a fully automated test suite is required to ensure that organizations can adopt the Mojaloop OSS into their solution without impacting non-functional or functional performance of their solution.
3. This needs to extend across the Mojaloop APIs, Settlement and Operational APIs. The pre-phase 3 effort adopted some of the tests and capability developed by commercial implementers, but this needs to be extended to cover all OSS functionality.



5. Event Logging

1. To ensure that operations can investigate the flow of messages and
2. that injection / tampering can be detected on investigation.



6. Error Handling

1. To ensure that the API defined errors as well as all other error cases are
 - a. handled similarly and
 - b. recorded (logged) in a standardized manner to support diagnostics

7. DevOps

1. The CI/CD pipeline needs to support the parameterized deployment so that the solution can easily be deployed into a development environment, community labs as well as a production system.
2. While the current deployment of the solution is appropriate for an OSS open lab, the rigor and security constraints required for direct deployment into production-grade environments has not yet been reached. This requirement includes the need to incorporate the deployment of the API Gateway to provide secure access for operating labs for the Mojaloop community OSS process and as a baseline for implementation-specific Dev Portals.

Phase-3 Roadmap: FRs

PI	Summary
Phase - 3	<ol style="list-style-type: none">1. Settlement Management2. Central Directory3. Forensic Logging4. Merchant "Request To Pay"5. Bulk Payments



1. Settlement Management

1. Based on the OSS Settlement API analysis and
2. Confirmation of priority from commercial partners, the items identified in the design work will be prioritized into the OSS development stream.



2. Central Directory

1. There are at least two potential avenues that the commercial partners are exploring for supporting multiple identifiers on the platform: the first is the use of multiple existing third party registries; the second is the creation of a centralized merchant registry controlled directly by the hub operator.
2. In both cases there is significant work to be done to ensure that the management of and registration of multiple identities map to the end users' desired account state while avoiding adverse behavior by DFSPs with respect to end-user preferences.



3. Forensic Logging

1. Ensure that, based on the Event and Error logging that the Forensic Logging component is re-installed and functioning as expected.
2. This requires an initial investigation to ensure the correct breadth of data is being logged in a forensically-strong manner.



4. Merchant 'Request to Pay'

1. Prepare support for Request to Pay by extending the DFSP Simulator and tests and
2. Making changes needed to support this use-case based on the outcomes



5. Bulk Payments

1. Support for B2P/G2P payments are being requested, this requires native Mojaloop API support
2. Planning for designing the solution, reviews and a PoC, which can then be adopted/implemented

Switch Functionality – Mojaloop End-points (Current)

Mojaloop v1.0 – API Specification

Transfers:

- [●] POST - Prepare
- [●] PUT - Response
- [●] PUT – Error
- [●] Outgoing
- [●] Incoming
- [●] GET - Query

Parties:

- [●] GET - Request
- [●] PUT - Response
- [○] PUT - Error

Quotes:

- [●] POST - Request
- [●] PUT - Response
- [●] PUT - Error
- [○] GET - Query

Participants:

- [●] POST - Create
- [●] PUT - Response
- [○] POST - Bulk Create
- [○] PUT - Error
- [○] DEL - Delete

Transactions:

- [○] PUT - Response
- [○] GET - Query

TransactionRequests:

- [○] POST - Request
- [○] PUT - Response
- [○] PUT - Error
- [○] GET - Query

Authorizations:

- [○] GET - Request
- [○] PUT - Response
- [○] PUT - Error

BulkTransfers:

- [○] POST - Request
- [○] PUT - Response
- [○] PUT - Error
- [○] GET - Query

BulkQuotes:

- [○] POST - Request
- [○] PUT - Response
- [○] PUT - Error
- [○] GET - Query

Key

- [●] Fully implemented
- [●] Legacy Code
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope for Phase2

Switch Functionality – Mojaloop End-points (Phase-3)

Mojaloop v1.0 – API Specification

Transfers:

- [●] POST - Prepare
- [●] PUT - Response
- [●] PUT – Error
- [●] Outgoing
- [●] Incoming
- [●] GET - Query

Parties:

- [●] GET - Request
- [●] PUT - Response
- [●] PUT - Error

Quotes:

- [●] POST - Request
- [●] PUT - Response
- [●] PUT - Error
- [●] GET - Query

Participants:

- [●] POST - Create
- [●] PUT - Response
- [○] POST - Bulk Create
- [●] PUT - Error
- [○] DEL - Delete

Transactions:

- [○] PUT - Response
- [●] GET - Query

TransactionRequests:

- [●] POST - Request
- [●] PUT - Response
- [●] PUT - Error
- [●] GET - Query

Authorizations:

- [○] GET - Request
- [○] PUT - Response
- [○] PUT - Error

BulkTransfers:

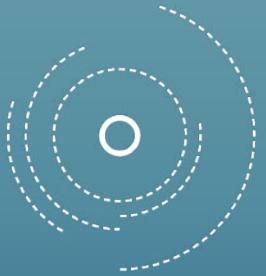
- [○] POST - Request
- [○] PUT - Response
- [○] PUT - Error
- [○] GET - Query

BulkQuotes:

- [○] POST - Request
- [○] PUT - Response
- [○] PUT - Error
- [○] GET - Query

Key

- [●] Fully implemented
- [●] Legacy Code
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope for Phase2



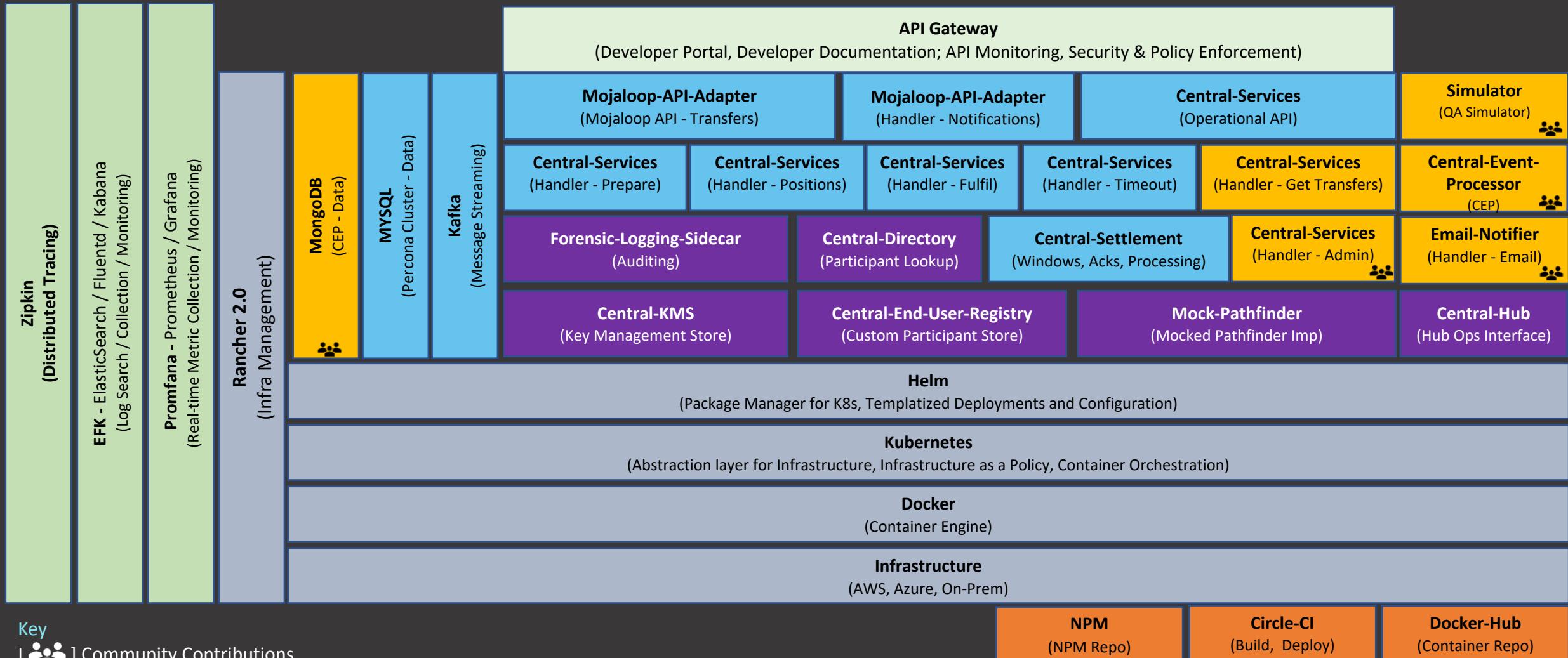
mojaloop

APPENDIX

Appendix - Contents

1. Component Architecture End-state - 87
2. Database Design/Schema - 88
3. End-to-end Flow – 89
4. Transfers Solution Design – 90
5. Deployment Architecture – Helm Overview – 91
6. Deployment Architecture – Kubernetes Overview – 92
7. Deployment Architecture – Rancher Overview – 93
8. Deployment Architecture – CircleCI Overview – 94
9. Deployment Architecture – CI/CD Pipeline – 95
10. Database Design – 96
11. Switch Functionality – Mojaloop – 97
12. Switch Functionality – Mojaloop Use Cases(Phase-2) - 98
13. PI4 Performance Summary – 99
14. Performance: Infrastructure Seutp - 100
15. Results – KafkaPoC – 101
16. Results – Mojaloop End-to-end Transfers– 102
17. Settlements System Treatment of a P2P Transfer – 103
18. Movement within the Switch – 104
19. Understanding accounting Principles – 105
20. Managing risk in the System – 106
21. Changes undertaken – 107
22. Checking NDC for a Sender – 108
23. In delayed Settlement a new risk is... - 109

Component Architecture – End-state

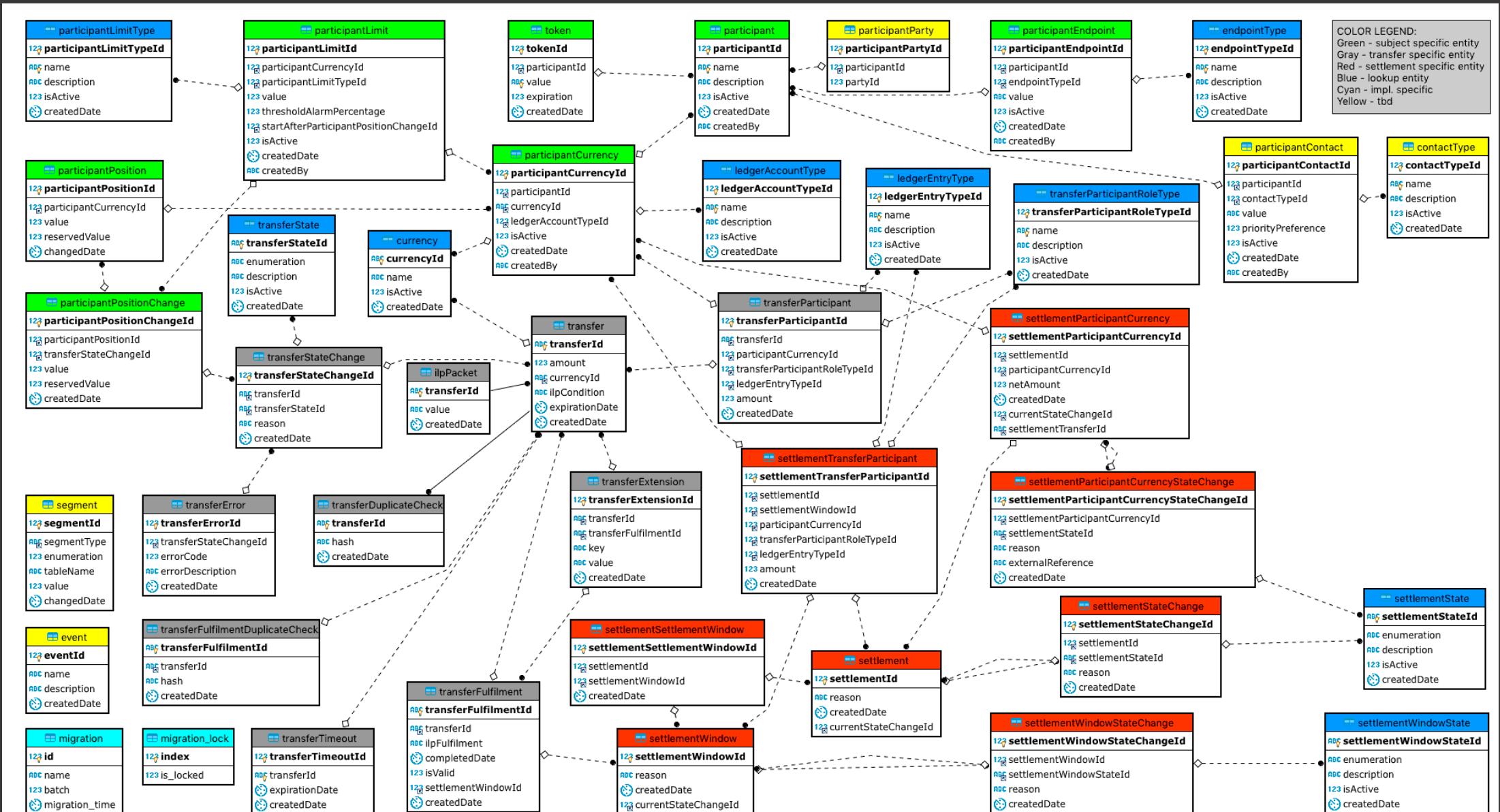


Phase-3 kick-off

Jan 2019

ModusBox for BMGF

Database Design – PI4

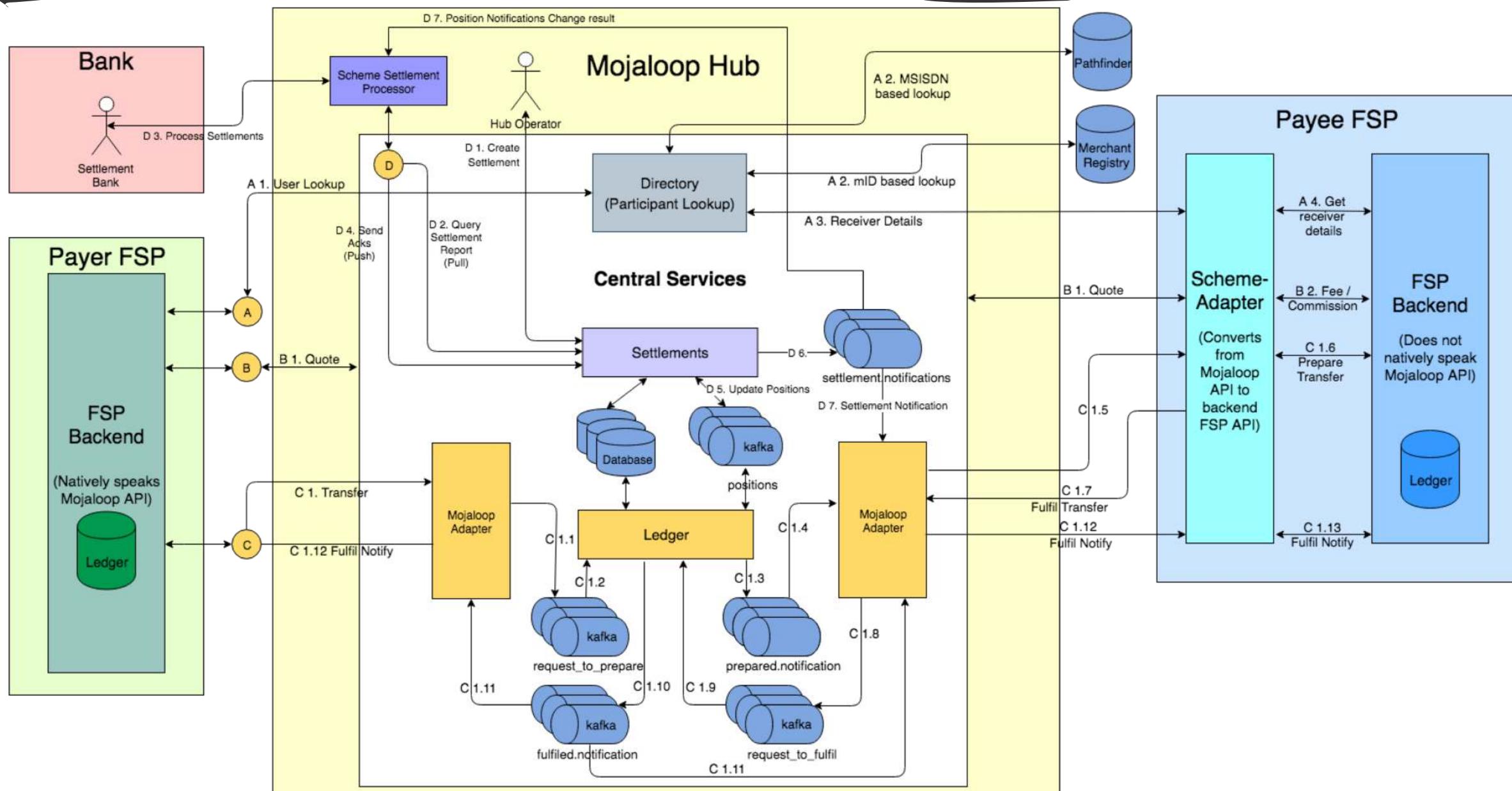


Phase-3 kick-off

Jan 2019

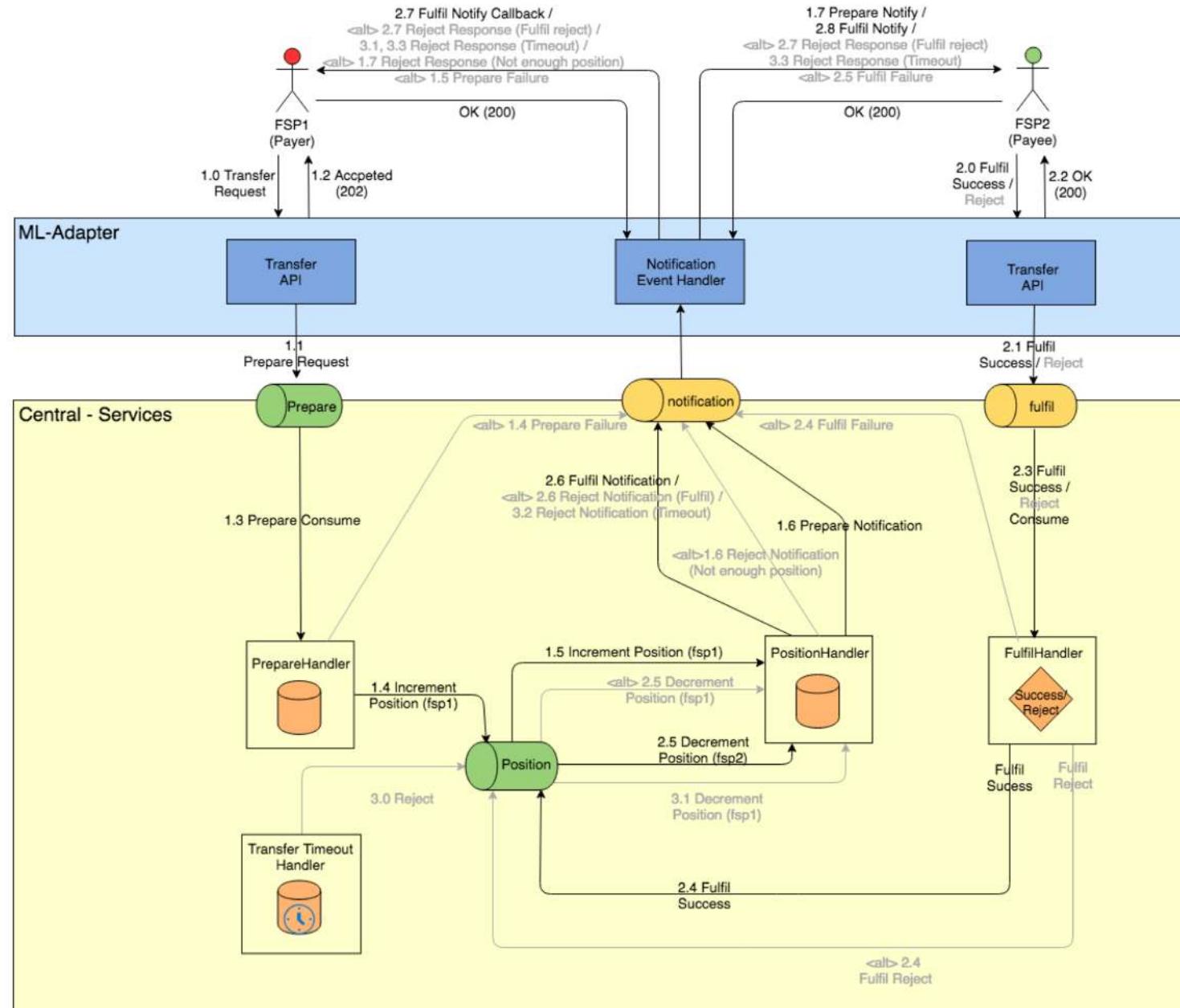
ModusBox for BMGF

End-to-end Flow



Transfers Solution Design

Phase-3 kick-off





Deployment Architecture – Helm Overview

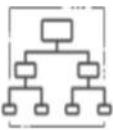
Ref: <http://helm.sh>

What is Helm?

Open Source Package Manager for Kubernetes through the use of Charts.

Charts help you define, install and upgrade releases for Kubernetes deployment via templates and configuration.

Why Helm?



Manage Complexity

Charts describe even the most complex apps; provide repeatable application installation, and serve as a single point of authority.



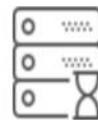
Easy Updates

Take the pain out of updates with in-place upgrades and custom hooks.



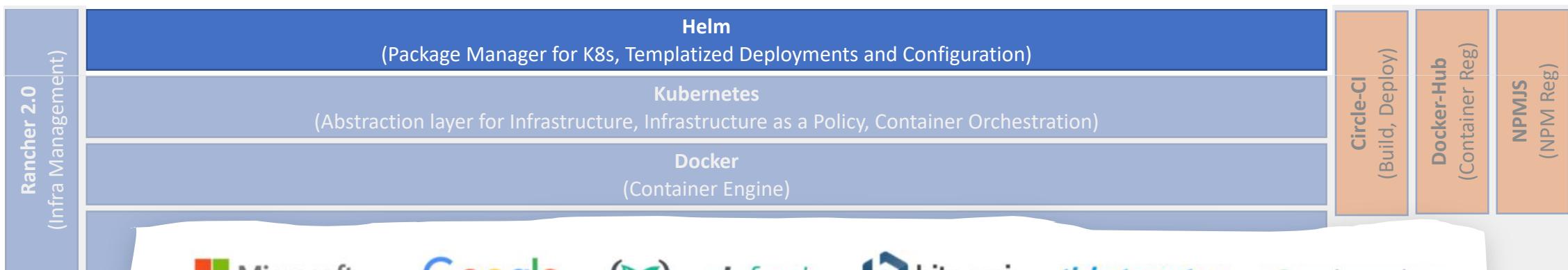
Simple Sharing

Charts are easy to version, share, and host on public or private servers.



Rollbacks

Use `helm rollback` to roll back to an older version of a release with ease.



Deployment Architecture – Kubernetes Overview

What is Kubernetes?

Open-source system for automating deployment, scaling, and management of containerized applications.



kubernetes

Why Kubernetes?



Deploy your applications quickly and predictably

- Infrastructure as a Policy
- Abstraction of Infrastructure (Cloud, On-Prem)



Scale your applications on the fly

- Policy rule based scaling
- Limit hardware & resources by scaling horizontally up/down



Roll out new features seamlessly

- Rolling updates



Discoverability

- Dynamic service resolution via DNS



Durability

- Self-healing
- Auto-[placement, restart, replication, scaling] based on Policies
- Load Balancing



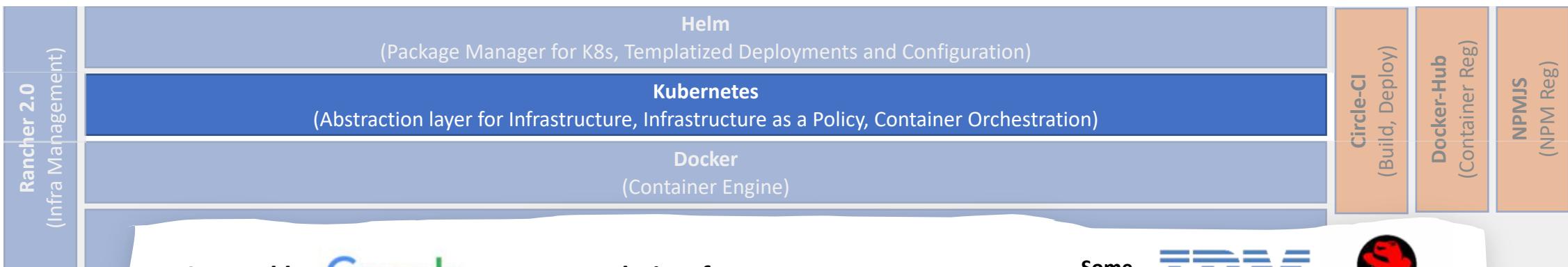
Security

- Isolation through Containers, Network and Namespaces



Operations

- App config & secrets stored in distributed key-value store (etcd)
- Monitoring of containers



Created by **Google** to support their Infrastructure.

Some contributors:





Deployment Architecture – Rancher Overview

Ref: <http://rancher.com>

What is Rancher?

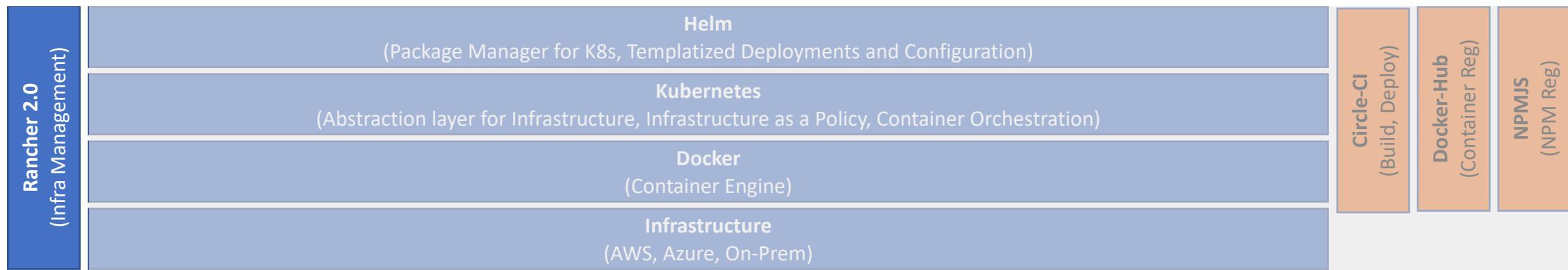
Rancher is enterprise management for Kubernetes.

Every distro. Every cluster. Every cloud.

Why Rancher?

- Kubernetes Management (v1.8 to *v1.9)
- Container Management
- *Access Management (RBAC)
- *Helm Repository Management
- Multi-environment Management (multi k8s clusters, On-prem, Azure, Google, AWS, etc)
- Multi-Provider provisioning (On-prem, Azure, Google, AWS, vSphere)
- Easily scale up/down Kubernetes clusters

* New in Rancher v2.x





Deployment Architecture – CircleCI Overview

What is CircleCI? Cloud based Continuous Integration & Deployment Platform

Ref: <http://circleci.com>

VCS Integration

CircleCI integrates with GitHub, GitHub Enterprise, and Bitbucket. Every time you commit code, CircleCI creates a build.

Automated Testing

CircleCI automatically tests your build in a clean container or virtual machine.

Automated Deployment

Passing builds are deployed to various environments so your product goes to market faster.

Notifications

Your team is notified if a build fails so issues can be fixed quickly.

Why CircleCI?



Workflows for Job Orchestration

Orchestrate customizable job execution (such as build, test, deploy), giving complete control over your development process.



Language-Agnostic Support

Supports any language that builds on Linux or macOS, including C++, Javascript, .NET, PHP, Python, and Ruby.



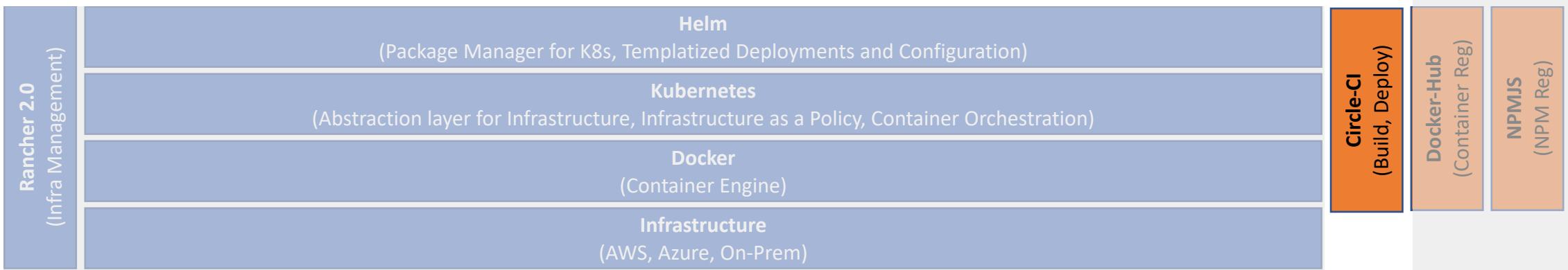
First-Class Docker Support

Run any image from Docker's public/private registry or other common registries. Build Docker images, access Docker layer caching, Compose.



Powerful Caching

Speed up builds with expanded caching options, including images, source code, dependencies, and custom caches. Full control over cache save and restore points for optimal performance.



* Forrester names CircleCI a leader (<https://www2.circleci.com/circleci-forrester-wave-leader-2017.html>)

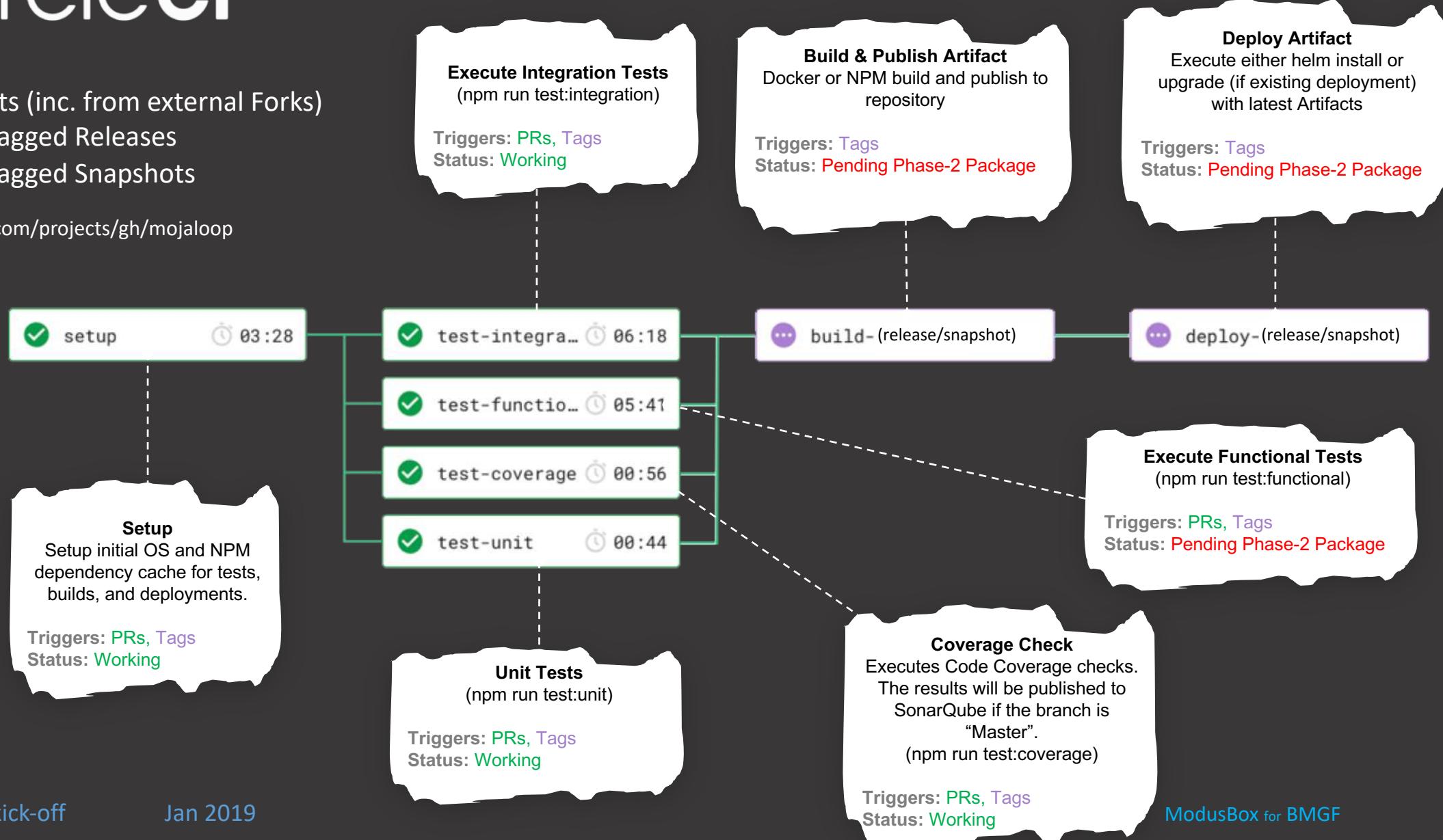
Deployment Architecture – CI/CD Pipeline



Triggers:

- [●] Pull-Requests (inc. from external Forks)
- [●] Publishing tagged Releases
- [●] Publishing tagged Snapshots

Ref: <https://circleci.com/projects/gh/mojaloop>



Phase-3 kick-off

Jan 2019

ModusBox for BMGF

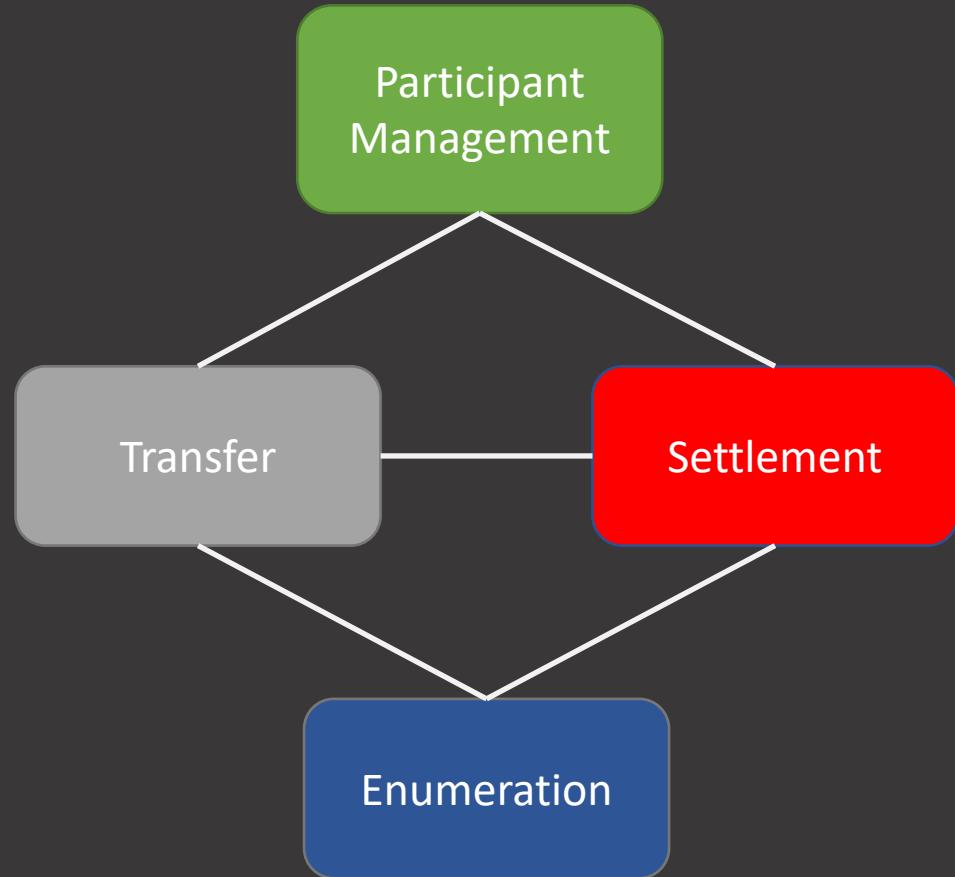
Database Design

Database Design Objectives

1. Align the data model to support to the Mojaloop specification
2. Design for efficiency and reduce locking
3. Data integrity maintained through a idempotency pattern

How was this achieved?

1. Redesign scheme to reflect the Mojaloop API and support the state transitions.
2. Inserts are used instead of updates to ensure reduced locking, and increased speed of data “updates”
3. Inserts are used to ensure an idempotent pattern for data “updates” is maintained. The result being that all “updates” have a history.
4. Support for batch processing of DFSP position, maintain repeatable and deterministic outcomes of rule processing and “running balance” for positions.



Switch Functionality – Mojaloop

Mojaloop v1.0 – P2P Use-case

Payer-Initiated Transaction

- [●] P2P Transfers – Golden Path
- [●] Prepares, Fulfils
- [●] Rejections, Timeouts
- [●] Error Endpoints

Mojaloop v1.0 – End Points for P2P

Transfers

- [●] POST - Prepare
- [●] PUT - Response
- [●] PUT – Error
 - [●] Outgoing
 - [●] Incoming
- [●] GET - Query

Parties

- [●] GET - Request
- [●] PUT - Response
- [○] PUT - Error

Participants

- [●] POST - Create
- [●] PUT - Response

Quotes

- [●] POST - Request
- [●] PUT - Response
- [●] PUT - Error
- [○] GET - Query

Key

- [●] Fully implemented
- [●] Legacy Code
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope

Switch Functionality – Mojaloop Use Cases (Phase-2)

Mojaloop v1.0 – Use-cases

Payer-Initiated Transaction

- [●] P2P Transfers – Golden Path
- [●] Prepares, Fulfils
- [●] Rejections, Timeouts
- [●] Error Endpoints
- [●] Customer-Initiated Merchant Payment
- [●] Customer-Initiated Cash-out - Receive Amount
- [●] Customer-Initiated Cash-out - Send Amount
- [○] ATM-Initiated Cash-out
- [●] Refund

Bulk Transactions

- [○] Bulk Payments

Payee-Initiated Transaction

- [○] Merchant-Initiated Merchant Payment
- [○] Agent-Initiated Cash-out
- [○] Agent-Initiated Cash-In – Send Amount
- [○] Agent-Initiated Cash-In – Receive Amount

Payee-Initiated Transaction using OTP

- [○] Merchant-Initiated Merchant Payment Authorized on POS
- [○] Agent-Initiated Cash-out Authorized on POS

Key

- [●] Fully implemented
- [●] Supported, not tested
- [●] Partially implemented
- [●] Not implemented
- [○] Out of Scope

PI4 Performance Summary

Key
● Completed in PI-4
● Partially completed in PI-4
● To be considered in Future

Approach:

- [●] Set a base-line on PI-3 Code-base
- [●] Configure Environment for Production-like setup
- [●] Identified & fix stability issues due to failure on load
- [●] Optimized Kafka Client (Producers, Consumers) for Performance
- [●] Scenarios with Multiple FSPs
- [●] Increase multi-threaded processing of non-¹EOS Kafka Consumers (Prepare, Transfer, Fulfil, Notification)
- [●] Optimized Handlers with reduced CPU Usage
- [●] Optimized Handlers with reduced IO
- [●] Tested against an alternative hosted Kafka service²
- [●] Increase scalability of solution
- [●] Optimized Handlers with PRISM

Comparison of Selected Results (in seconds):

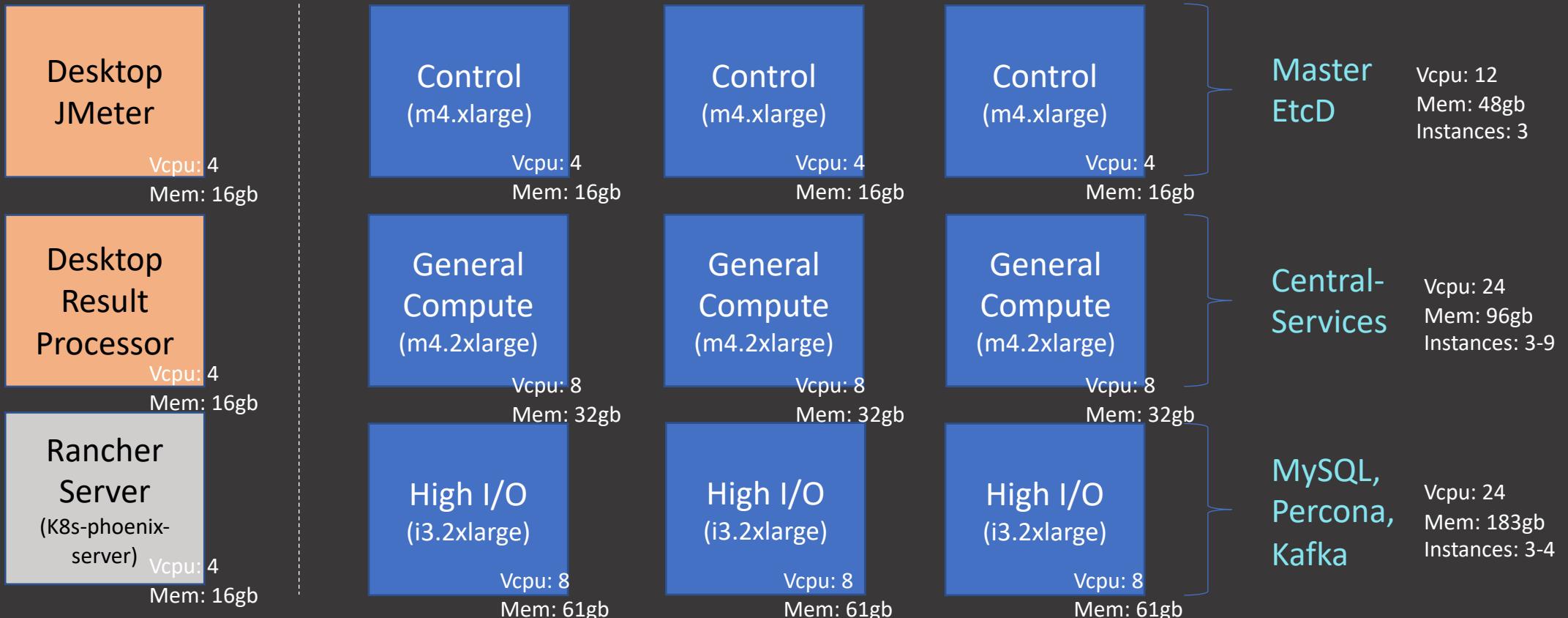
Metric	PI2	PI3	PI4 ³	
Number of Messages	100	100	358	201
Prepare Process	✓	✓	✓	✓
Fulfil Process	✗	✗	✓	✓
Included Mock FSP Simulator Times	✗	✗	✓	✗
Transactions Per Second	6.51	22.18	99	207
Average Transaction Time	8.21	1.65	0.111	0.101
Shortest Response Time	2.594	0.942	0.47	0.32
Longest Response Time	19.746	2.589	3.057	3.047
Standard Deviation	n/a	n/a	0.120	0.119
% of requests taking longer than 1 second	n/a	n/a	0.17%	0.002%
Duration	15 seconds	4.5 seconds	1 hour	1 hour

¹ EOS – Exactly Once Semantic,

² Mojaloop Kafka deployment on Rancher K8s-Phoenix Environment was comparable and better at times,

³ Average over 3 runs – individual results in the appendix

Performance Update – Environment: Infrastructure Setup



Results¹ - KafkaPoC

KafkaPoC - Producer

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	1.2 Seconds
Min Response	0 ms
Max Response	3 ms
Mean/Avg	0.10 ms
Std. Deviation	0.31 ms
% > 1 Second	0.00%
Trans / per Sec	7861

Recursive
Mode

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	1.2 Seconds
Min Response	0 ms
Max Response	2 ms
Mean/Avg	0.10 ms
Std. Deviation	0.31 ms
% > 1 Second	0.00%
Trans / per Sec	7874

PoC Flow
Mode

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	1.2 Seconds
Min Response	0 ms
Max Response	3 ms
Mean/Avg	0.10 ms
Std. Deviation	0.09 ms
% > 1 Second	0.00%
Trans / per Sec	7993

PoC Stream
Mode

KafkaPoC - Consumer

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	2.6 Seconds
Min Response	0 ms
Max Response	1 ms
Mean/Avg	0.08 ms
Std. Deviation	0.27 ms
% > 1 Second	0.00%
Trans / per Sec	3796

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	1.6 Seconds
Min Response	0 ms
Max Response	2 ms
Mean/Avg	0.06 ms
Std. Deviation	0.25 ms
% > 1 Second	0.00%
Trans / per Sec	5938

Date	2018-12-21 @ 15h49
# No. Request	10 000
Time	2.1
Min Response	0 ms
Max Response	1 ms
Mean/Avg	0.06 ms
Std. Deviation	0.24 ms
% > 1 Second	0.00%
Trans / per Sec	4725

¹ All results are available on demand. Only selected results shown here

Results¹ - Mojaloop End-to-end Transfer Process

Calculations including
Simulators (With Mock FSP)

Date	2018-12-14 @ 20h50
# No. Request	359 508
Time	1 Hour
Min Response	48 ms
Max Response	1492 ms
Mean/Avg	111 ms
Std. Deviation	114 ms
% > 1 Second	0.25%
Trans / per Sec	100

Recursive
Mode

Date	2018-12-14 @ 20h50
# No. Request	359 508
Time	1 Hour
Min Response	33 ms
Max Response	1482 ms
Mean/Avg	101 ms
Std. Deviation	112 ms
% > 1 Second	0.25%
Trans / per Sec	197

Date	2019-01-18 @ 20h35
# No. Request	359 839
Time	1 Hour
Min Response	44 ms
Max Response	6913 ms
Mean/Avg	118 ms
Std. Deviation	205 ms
% > 1 Second	0.25%
Trans / per Sec	100

Recursive
Mode

Date	2019-01-18 @ 20h35
# No. Request	359 839
Time	1 Hour
Min Response	23 ms
Max Response	6903 ms
Mean/Avg	107 ms
Std. Deviation	205 ms
% > 1 Second	0.24%
Trans / per Sec	216

Date	2019-01-25 @ 10h48
# No. Request	355 258
Time	1 Hour
Min Response	51 ms
Max Response	768 ms
Mean/Avg	106 ms
Std. Deviation	41 ms
% > 1 Second	0.00%
Trans / per Sec	98

Recursive
Mode

Date	2019-01-25 @ 10h48
# No. Request	355 258
Time	1 Hour
Min Response	41 ms
Max Response	758 ms
Mean/Avg	95 ms
Std. Deviation	41 ms
% > 1 Second	0.00%
Trans / per Sec	210

¹ All results are available on demand. Only selected results shown here

System Treatment of a P2P Transfer

- From a system perspective, any P2P transfer is managed with three separate stages:
 - Prepare
 - Reserve
 - Commit or Abort
- During the prepare stage, the entries are recorded in the database, without affecting the balance of the relevant ledger.
- When moving to the reserve state, the Position of the Payer DFSP is increased to reserve the funds. This ensures the principal that funds are ring-fenced - so cannot be used for another transfer until they are released.
- When moving to the commit state, the Position of the Payee DFSP is decreased allowing them to send more funds to other DFSPs if desired.
- After receiving a quote, the payer DFSP can initiate a transfer. That initial request results in the transfer being prepared, and then the Position increased to reserve the funds. This is referred to as the Prepare leg of the transfer process within Mojaloop. When the Payee DFSP sends a confirmation of the transfer, the transfer is committed, and the Payee DFSPs Position is reduced and since they are now able to use those funds - the committed state means the transfer is irrevocable at this point. This portion is referred to as the Fulfil leg of the transfer process within Mojaloop.
- NEEDS REWORK

Movement within the Switch

- To ensure the safe movement of funds, it is important that any transfers from one account to another are done in an orderly manner. This approach should ensure that funds are ring-fenced, and the switch does not enter a situation where funds are released, before they are confirmed to be available.
- The principle role of the switch is to facilitate transfers between the DFSPs.
 - This is within an agreed limit known as the Net Debit Cap, which from a DFSP perspective is the maximum Debt (Dr balance) THEY - the DFSP - can have with the Switch.
 - Within the Switch, this is the amount that the DFSP owes to other DFSPs - this value is known as the Position.
 - Within the switch, the Position of a DFSP increases the more they owe. From the perspective of the Payer/Sending DFSP it is an increase in their liability, from the Payee/Receiving DFSP it is a reduction in their liability (or increase in their assets if sufficient). So therefore the Payer DFSP sees their position increase, and the Payee sees their position decrease

Understanding Accounting Principles

- Standard practice in accounting software is to treat Debits (Drs) as positives and Credits (Crs) as negatives. That way when you add up the items in a ledger, if the net position is positive (More Drs) they are treated as an asset, and if the net position is negative (More Crs) they are seen as a liability. Debits and Credits can occur in any account. – this has been adopted in MojaLoop
- It is worth noting that the Switch has a different perspective to the DFSPs but follows the same rules: when funds are input the cash (asset) increases, which is a debit; the increase in the customer's account balance (liability from the Switch's perspective) is a credit. The reporting of the switch, follows this process. When a transfer takes place, the Dr is applied to the sender - because when the movement is applied back to the account balance, it would lead to a reduction in the amount that is owed back to the DFSP (reduced liability).

Managing the risk in the system

- As we outlined above in Understanding a Double Entry, a Dr reduces the switches liability, so it will always be applied in the Reservation stage so any impact to the switch is based on ensuring funds have been ring fenced first. Applying this to the System Treatment of a P2P Transfer
 - When moving to the reserve state, the Position of the Payer DFSP is Debited (Dr) to reserve the funds.
 - When moving to the commit state, the Position of the Payee DFSP is Credited (CR) allowing them to send more funds to other DFSPs if desired – but failure in receiving those funds would lead to liability issues.
- Adopting a standard approach where confirmations are expected to take place before funds are realeased also lead to consistency in reporting, reconciliation or roll back.
- This principle must be applied in all transfers, and the potential loss of funds due to any change reviewed as part of the flow – ensuring that transactions that may occur whilst the process is underway do not expose the switch to a financial risk.

Changes Undertaken

- We have introduced options on the relationship with real money flows
 - Real time settlement
 - Delayed settlement
- Which required the addition of 3 more account types
 - DFSP Settlement Ledgers – to mirror funds in the real bank account
 - Hub Multilateral Settlement Ledger
 - Hub Reconciliation Ledger
- Each have their own risk which has been factored in Design

Checking NDC for a Sender

	Activity	DFSP1 Transfer Ledger			DFSP2 Transfer Ledger			DFSP1 Settlement Ledger		DFSP2 Settlement Ledger		Multi Lateral Settlement Ledger		Hub Reconciliation		
		Transfer	Position	NDC	Transfer	Position	NDC	Transaction	Balance	Transaction	Balance	Transaction	Balance	Transaction	Balance	
1	Settlement Window Closed			+350	+800		-350	+800		-1000		-1000		0	+2000	
2	DFSP1 Settlement Processed	DR		+350	+800		-350	+800		-1000		-1000	+350	+350	+2000	
3	DFSP2 Position Reset	DR		+350	+800	+350	0	+800		-1000		-1000	+350	+350	+2000	
4	DFSP2 Settlement Processed	CR		+350	+800		0	+800		-1000		-1000	-350	0	+2000	
5	DFSP1 Position Reset	CR	-350	0	+800		0	+800		-1000		-1000			+2000	
6	Funds Transfer For Settlement - DFSP 1			0	+800		0	+800	+350	-650		-1000		0	-350	+1650
7	Funds Transfer For Settlement - DFSP 2			0	+800		0	+800		-650	-350	-1350		0	+350	+2000

Step 1 is the closure of the Window

Steps 2-5 are the steps to settle

Steps 6 and 7 are the real movement of funds.

However when the money flows the NDC is now higher than the Settlement Ledger balances – Automation of these steps requires a change in the NDC

The NDC adjustment needs to be applied before any processing of the settlement window – to ensure the control is in place before any positions are reset. This could lead to a temporary block on transfers, but eliminates the risk that transfers are allowed before the settlement is processed

Note that the balance of transfers is done in the Hub Multilateral Settlement Ledger to ensure reconciliation, but it also allows for the settlement amounts to be handled differently

In delayed settlement a new risk is introduced

Net Receiver becomes Net Sender

Activity		DFSP1 Transfer Ledger			DFSP2 Transfer Ledger			DFSP1 Settlement Ledger		DFSP2 Settlement Ledger		Hub Reconciliation	
		Transfer	Position	NDC	Transfer	Position	NDC	Transaction	Balance	Transaction	Balance	Transaction	Balance
1	Settlement Window Closed		+350	+800		-350	+800		-1000		-1000		+2000
2	DFSP2 → DFSP1 \$600	Allowed	-900	-550	+800	900	550	+800	-1000	-1000	-1000		+2000
3	Net Debit Cap Adjusted - lower Balance		-550	+650		550	+800		-1000		-1000		+2000
4	DFSP1 Settlement Processed	DR	-550	+650		550	+800	+350	-650		-1000		+2000
5	DFSP2 Position Reset	DR	-550	+650	+350	900	+800		-650		-1000		+2000
6	DFSP2 Settlement Processed	CR	-900	+650		900	+800		-650	-350	-1350		+2000
7	DFSP1 Position Reset	CR	-350	-900	+650		900	+800		-650	-1000		+2000

If the time between Step 1 and Step 6 is large, the likelihood of this impact is large.

- Envisioned scenario is a Natural disaster over the weekend that reverses the normal flows before the settlement can be processed
- The settlement Windows is closed on Friday Evening, and the settlement and resetting of the Position does not happen until the following Monday or Tuesday (in the case of a Bank Holiday weekend)
- Before the position is reset, the DFSP had sufficient funds, but the reset would immediately block their ability to continue sending