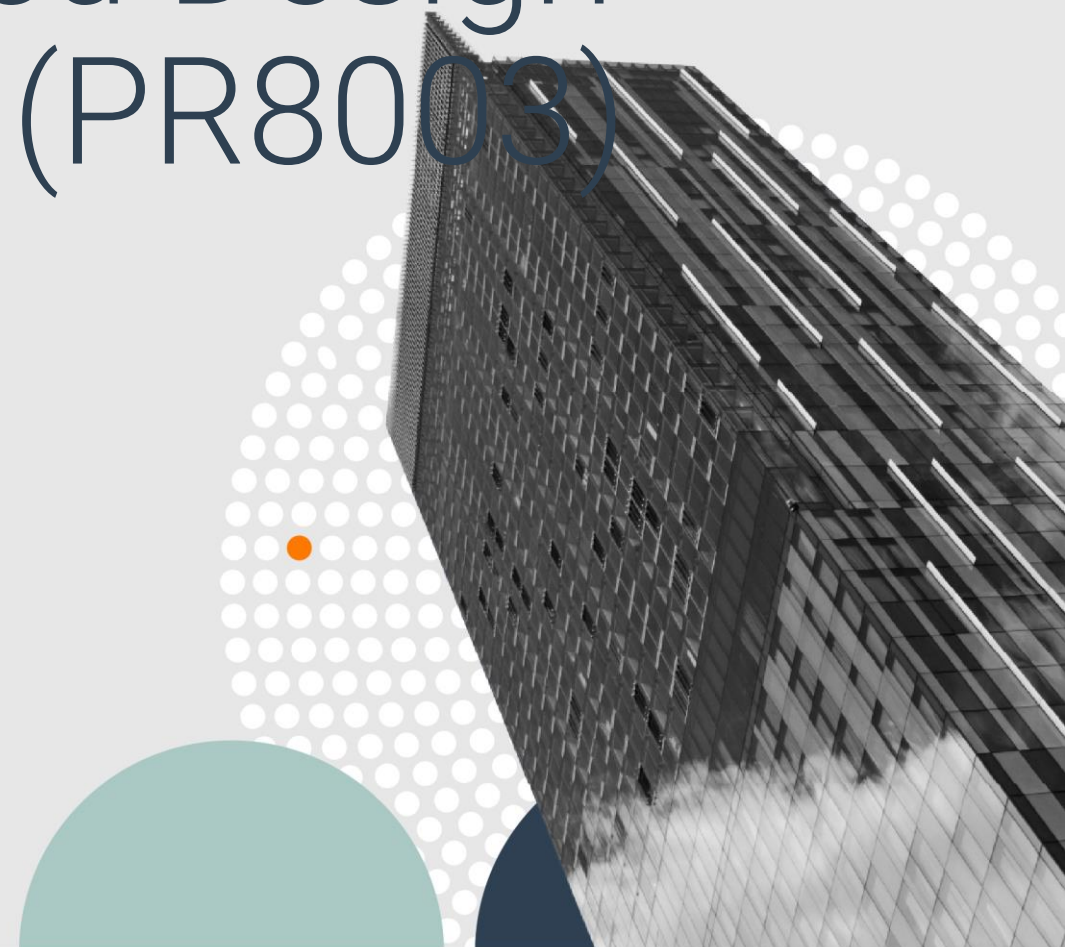
Landmark Solutions

# Message Regeneration Detailed Design Stage (PR8003)



Final

## Solution Architecture Design Document

**13th June 2025 / Ferguson / V.1.01 (Final)**

# Contents

## Contents

**Contents** ..... **2**
Revision history ..... 6
Review ..... 6
Approval ..... 6
**1 Introduction** ..... **7**
**1.1 Audience** ..... **7**
**1.2 Forward Looking Statement** ..... **7**
**1.3 Document References** ..... **8**
**1.4 Purpose of the Solution Architecture Design Document** ..... **8**
**1.5 Solution Design Principles** ..... **9**
**1.6 Terminology** ..... **10**
**1.7 Solution Scope** ..... **12**
1.7.1 PR8003 Statement of Deliverables ..... 12
1.7.2 PR8003 Statement of Work ..... 12
1.7.3 Design Objective ..... 12
1.7.4 Boundaries & Considerations ..... 12
1.7.5 Requirements ..... 13
1.7.6 BCDR Exercise Recommendations for Consideration ..... 14
1.7.7 Risks ..... 14
1.7.8 Assumptions ..... 14
1.7.9 Constraints ..... 15
1.7.10 Dependencies ..... 15
1.7.11 Decisions ..... 15
**2 Solution Concepts and Capabilities** ..... **16**
**2.1 Background Understanding** ..... **16**
2.1.1 Existing High Level Request Flow ..... 17
**2.2 Conceptual Solution Overview** ..... **18**
2.2.1 Recovery Point Ledger ..... 19
2.2.2 Recovery Point Orchestrator ..... 19
2.2.3 Recovery Point Service ..... 19
2.2.4 Message Queue Replacement ..... 19
2.2.5 Existing Azure Blob Storage Statement ..... 20
2.2.6 Refactored Function using Recovery Point Ledger ..... 21
2.2.7 Refactoring an Operation to use the Recovery Point Service ..... 22
**3 Solution Architecture Building Blocks** ..... **23**
**3.1 Information Architecture** ..... **23**

3.1.1 Overview ..... 23
3.1.2 Data Domains and Entities ..... 23
3.1.3 Data Flow and Lifecycle ..... 24
3.1.4 Storage and Access ..... 24
3.1.5 Data Governance and Security ..... 24
3.1.6 Integration with CSS Data Model ..... 24
**3.2 Application Architecture ..... 25**
3.2.1 Overview ..... 25
3.2.2 Recovery Point System Solution Building Blocks ..... 26
3.2.3 RPSys Reporting and Dashboard Building Blocks ..... 28
3.2.4 Design Patterns and Rationale ..... 29
3.2.5 Scalability and Resilience ..... 29
3.2.6 Security and Observability ..... 29
**3.3 Data Architecture ..... 30**
3.3.1 Overview ..... 30
3.3.2 Data Storage Architecture ..... 30
3.3.3 Data Flow and Access Patterns ..... 30
3.3.4 Data Integration and Analytics (Dashboards and Reports) ..... 30
3.3.5 Data Governance and Security ..... 31
3.3.6 Scalability and Performance ..... 31
3.3.7 Recovery Points ..... 31
**3.4 Integration Architecture ..... 32**
3.4.1 Overview ..... 32
3.4.2 Integration Components ..... 32
3.4.3 Refactoring of Existing Integration Components ..... 34
3.4.4 Integration Patterns ..... 38
3.4.5 Security and Access Control ..... 38
3.4.6 Monitoring and Observability ..... 38
**3.5 System Architecture ..... 39**
3.5.1 Overview ..... 39
3.5.2 Error Handling Strategy ..... 39
3.5.3 Logging and Monitoring ..... 39
3.5.4 Caching Strategy ..... 39
**3.6 Infrastructure Architecture ..... 40**
3.6.1 Azure Cosmos DB (Multi-Region Write) ..... 40
3.6.2 Azure Synapse & SQL Pool with Power BI Integration ..... 40
3.6.3 Azure Service Bus (Premium Tier) ..... 40
3.6.4 Azure Networking and Resource Group ..... 40
3.6.5 Azure Function App – Recovery Point Orchestrator ..... 41
3.6.6 Scalability and High Availability ..... 41
3.6.7 Security and Compliance ..... 41
3.6.8 Integration with Existing Systems ..... 41
3.6.9 High Availability Infrastructure Design ..... 42

**3.7 Security & Compliance Architecture ..... 43**
3.7.1 Overview ..... 43
3.7.2 Data Protection ..... 43
3.7.3 Access Control and Identity Management ..... 43
3.7.4 Compliance Standards ..... 43
3.7.5 Risk Management and Threat Mitigation ..... 44
3.7.6 Incident Response and Monitoring ..... 44
3.7.7 Continuous Compliance and Governance ..... 44
**4 Implementation ..... 45**
**4.1 Development ..... 45**
4.1.1 Development Phases ..... 45
4.1.2 Technology Stack ..... 46
4.1.3 Integration Points ..... 46
4.1.4 Testing Strategy Activities ..... 46
4.1.5 Deployment Procedures ..... 46
4.1.6 Roles and Responsibilities ..... 46
4.1.7 PR8003 PoC Slices Evaluated ..... 46
**4.2 Testing ..... 47**
4.2.1 Environment Consideration ..... 47
4.2.2 Execution Strategy for Automated Tests ..... 47
4.2.3 Privilege Requirements for Automated Testing ..... 47
4.2.4 Testing Strategy and Tool Alignment ..... 48
4.2.5 External System Integrator (SI) Verification Testing ..... 48
4.2.6 Explore Synthetic Test Data in Production Environment ..... 48
4.2.7 Consideration for adopting an integrated cross-team testing framework ..... 49
4.2.8 PIT Environment Load and Performance Testing ..... 49
4.2.9 Use of Azure Chaos Studio for Testing ..... 49
4.2.10 BCDR Testing ..... 50
**4.3 DevOps ..... 51**
4.3.1 Pipeline Configuration and Deployment Automation ..... 51
4.3.2 Monitoring and Alerting ..... 51
4.3.3 Observability and Diagnostics ..... 51
4.3.4 Resilience and Failover Readiness ..... 51
4.3.5 Configuration and Feature Flag Management ..... 51
4.3.6 Security and Compliance ..... 51
4.3.7 Incident Readiness ..... 52
**4.4 Deployment Considerations ..... 52**
4.4.1 Pre-Requisites ..... 52
4.4.2 Deployment Process ..... 52
4.4.3 Rollback Strategy ..... 52
4.4.4 Post-Deployment Activities ..... 52
**4.5 Data Usage & Licensing (COTS, Libraries and Packages) ..... 53**
**4.6 Operational Management ..... 53**

**4.7 Service Levels** ..... **54**
4.7.1 Objectives ..... 54
4.7.2 Monitoring and Alerting ..... 54
4.7.3 Reporting and Compliance ..... 55
4.7.4 Compliance and Audit Readiness ..... 55
4.7.5 Initial Service Level Impact Assessment ..... 55
**4.8 Business Continuity and Disaster Recovery** ..... **56**
4.8.1 Recover Point System BCDR Actions ..... 56
4.8.2 Recovery Point System Resources Fail ..... 56
4.8.3 Cold Start Behaviour ..... 56
4.8.4 Participant Behaviour During BCDR Event ..... 57
**4.9 Change Governance** ..... **57**
**4.10 Costs** ..... **58**
4.10.1 Azure Cosmos DB (Multi-Region Write + Analytical Store) ..... 58
4.10.2 Azure Service Bus (Premium Tier) ..... 58
4.10.3 Azure Functions (Premium Plan) ..... 58
4.10.4 Monitoring and Logging ..... 58
4.10.5 Power BI and Synapse Analytics ..... 59
4.10.6 Cost Governance Recommendations ..... 59
**5 Non-Functional Requirements** ..... **60**
**5.1 Performance** ..... **60**
**5.2 Scalability** ..... **60**
**5.3 Availability and Reliability** ..... **60**
**5.4 Security** ..... **61**
**5.5 Compliance** ..... **61**
**5.6 Data Integrity and Backup** ..... **61**
**5.7 Usability and Supportability** ..... **61**
**5.8 Monitoring, Logging, and Maintenance** ..... **61**
**5.9 Service Recovery Point Non-Functional Mitigation Strategies** ..... **62**
**6 Appendix** ..... **63**
**6.1 PR8003 Statement of Work Activities** ..... **63**
**6.2 Future Features to Enhance R0096** ..... **64**
6.2.1 ML/AI for Augmented Notifications and Operational Triage ..... 64
6.2.2 ML-Based Stall Prediction Through Clustering and Time Series Analysis ..... 64
6.2.3 Short-Term Remediation Interface for Unresolved Stalls ..... 64
**6.3 Existing Non-Functional Service, Resiliency and Recovery Patterns** ..... **64**
6.3.1 Azure Functions ..... 64
6.3.2 Cosmos Database ..... 65
6.3.3 Azure Storage (Blob & Queue) ..... 65
6.3.4 Address Service ..... 66
6.3.5 Event Notification Retry Strategy ..... 66
6.3.6 Message Stage Lifecycle ..... 66

| | |
|---|---|
| Prepared For: | Smart DCC |
| Document Author: | Steven Ferguson |
| Last Modified: | 13 June 2025 |
| Version: | Final |
| Filename: | CSS Message Regeneration Detailed Design (R0096) |
| Sensitivity: | Confidential |

## Revision history

| Date | Author | Version | Reason |
|---|---|---|---|
| 28 May 2025 | Ferguson | 0.1 | Draft for Review |
| 30 May 2025 | Ferguson | 0.2 | Internal Feedback |
| 30 May 2025 | Ferguson | 1.0 | Final |
| 13 June 2025 | Ferguson | 1.01 | DCC/NC Feedback, see Link (R0096 collated feedback from DCC/SI) for details. Section 3.8 corrected to 4.0 Implementation and therefore subsequent section numbers updated. |

## Review

| Name | Role | Organisation |
|---|---|---|
| **Aishwarya Chourasia** | Principle QA Engineer | Landmark Solutions |
| **Isabelle North** | DevOps Lead | Landmark Solutions |
| **Jason Worlin** | Solution Architect | Landmark Solutions |
| **Claire Webster** | Business Analyst | Landmark Solutions |
| **Tom Owens** | Software Engineering Lead | Landmark Solutions |

## Approval

| Name | Role | Signature | Date |
|---|---|---|---|
| Ian Coe | Senior Technical Consultant | | 30 May 2025 |
| Steven Ferguson | Solution Architect | | 13 June 2025 |

# 1 Introduction

This Solution Architecture Design Document (SADD) for PR8003 (R0096 Detailed Design Stage), provides an end-to-end view of the solution, its purpose is to communicate the detailed design to all stakeholders. Providing views of the design to address core service design principles, traceability to the requirements and how the service will meet functional and non-functional demands.

Requirements for R0096 [Table 1. DC-01] and the proposed Recovery Point System (RPSys) detailed herein emerged from the detailed analysis and remediation recommendations following the Business Continuity / Disaster Recovery (BCDR) event in July 2023 where storage replication latency and data recovery issues were encountered.

This analysis determined that during a service interruption, there is a risk that dependent message files may not be fully replicated to the recovery site. This creates a scenario where the content required to generate outbound events may be unavailable upon recovery, potentially leading to incomplete notifications, it was further recommended that all operations should be capable of re-entrant recovery, not just notifications.

To address this, the scope of the Recovery Point System (RPSys) has been defined to ensure that any market participant requests which either fail to start or are interrupted during execution are capable of recovery, permitting such requests to safely restart from a known, consistent point, thereby preserving data integrity, message order and ensuring continuity of operations post-recovery.

## 1.1 Audience

The target audience for this document comprises individuals well-versed in the responsibilities of the CSS service, as well as the platform's functional and non-functional capabilities including the multi-region architecture.

## 1.2 Forward Looking Statement

This Solution Architecture Design Document represents the proposed architecture at the time of publication of Version 1.0 and is intended to provide a comprehensive and well-considered foundation for the implementation of the Recovery Point System within the Central Switching Service Registration capability. While every effort has been made to ensure that the design is aligned with best practices, is operationally sustainable, and leverages current Azure platform services appropriately, it is acknowledged that the detailed impact assessment phase may identify changes necessary to meet emerging requirements, updated standards, or unforeseen constraints.

Furthermore, given that a period of 9 to 12 months may elapse between the approval of this design and the commencement of development activities, it is recognised that technologies, services, or architectural patterns may evolve in that time. As such, components and design decisions outlined in this document must be reviewed and validated prior to implementation to ensure continued relevance, optimal performance, cost-effectiveness, and alignment with the most current standards.

This document should therefore be considered a baseline subject to refinement as part of ongoing solution assurance, and not a fixed or final implementation specification.

## 1.3 Document References

This document must be read in conjunction with the documents listed in Table 1.

| DOC-ID | Description | Link / File Name |
|---|---|---|
| DC-01 | REC Change Request R0096 | REC Portal |
| DC-02 | Smart DCC Requirements | R0096_Requirements Catalogue v1.0 |
| DC-03 | Landmark Statement of Work (PR8003) | SoW PR8003 R0096 Detailed Design Stage - CSS Message Regeneration Functionality v1.0 |
| DC-04 | CSS Interface Design Specification | Category 3 Document on REC Portal |

**Table 1.** Document References

## 1.4 Purpose of the Solution Architecture Design Document

The objective of this document is to define the detailed design for the REC Change Request R0096 Message Regeneration for the Central Switching Service (CSS) as set out in PR8003 Statement of Work. The intention is to provide sufficient details to inform the Detailed Impact Assessment (DIA) stage.

## 1.5 Solution Design Principles

This section outlines the fundamental guidelines and best practices for the solution design.

| AP-ID | (AP) Architectural Principle(s) |
|---|---|
| AP-01 | **Contractual and Schedule-Driven Compliance**<br>The solution will be designed, implemented and operated in full alignment with the governing contract framework and associated technical and operational schedules. |
| AP-02 | **Security by Design**<br>Security will be embedded at every layer of the architecture using the principles of least privilege, role-based access control, encryption (in transit and at rest), secure defaults, and defence in depth. |
| AP-03 | **Cloud-Native First**<br>Microsoft Azure-native services will be used to maintain compatibility, scalability and operational effectiveness. Justification will be provided for using non-cloud-native components. |
| AP-04 | **Resilience and Recoverability**<br>The solution will handle transient faults, partial failures, and service disruptions through built-in resilience mechanisms such as retries, exponential backoff, circuit breaker, and high availability architecture. |
| AP-05 | **Scalability and Performance**<br>The solution will scale to meet both average and peak usage demands. Architecture will favour stateless, horizontally scalable components and implement performance optimisation patterns such as partitioning. |
| AP-06 | **Modularity and Separation of Concerns**<br>Architectural components will be logically and physically separated by responsibility. Enabling independent development, testing, deployment, and maintenance. |
| AP-07 | **Observability and Traceability**<br>The solution will implement end-to-end observability including logging, metrics, and distributed tracing. Logs and telemetry will aid in root cause analysis, service level validation, and compliance auditing. |
| AP-08 | **Idempotency and Retry Safety**<br>All invoked and retriable operations will be idempotent. This prevents duplication or inconsistent states during retries, and reprocessing during recovery. |
| AP-09 | **Data Privacy and Regulatory Compliance**<br>Data will be handled in compliance with applicable laws and regulations such as GDPR, the UK Data Protection Act, and sector-specific obligations (e.g., Code of Connection, REC standards). Data classification and handling rules will be clearly defined. |
| AP-10 | **Alignment with Enterprise Standards**<br>The solution will follow Landmark Information Group (LIG) enterprise architecture standards, naming conventions, security baselines, and governance models to ensure interoperability and ease of transition into support. |
| AP-11 | **Testability and Automation**<br>All components will be designed for testability. This includes support for unit tests, automated unit tests, integration, and regression activities. CI/CD pipelines will be implemented to enforce build, test, and deployment automation using Infrastructure as Code (IaC) and Configuration as Code (CaC). |
| AP-12 | **Operational Readiness and Supportability**<br>The architecture will support operational readiness through clear monitoring, alerting, deployment automation, configuration management, and runbooks. Operational roles will be clearly defined and documented. |
| AP-13 | **Versioning and Change Management**<br>All APIs, data schemas, and shared components will be versioned to prevent breaking changes. Change management procedures will be in place to handle evolution in a controlled manner. |

**Table 2.** Architecture Principles

## 1.6 Terminology

The terminology in Table 3 is in line with industry terminology used within the REC Code of Connection schedules or specific to the internal functionality of the CSS implementation.

For a full list of terminology used in CSS refer to the Interface Design Specification: CSS Interface Design Specification located on the Retail Energy Code portal in the Code Products section.

| Term | Description | Explanation |
|---|---|---|
| Workflow | A collection of operations where each operation results in a notification and a resulting decision can determine the next operation | A successfully completed switch from one supplier to another is an example of a collection of operations. |
| CorrelationId | Links together all steps in a request-response transaction chain across multiple systems | Generated by the CSS within the VerifyInboundRequest function.<br><br>Propagated unchanged through all downstream systems and services.<br><br>Included in:<br>• API headers (e.g., X-Correlation-Id)<br>• Event metadata<br>• Logs and error responses |
| Operation | An operation is an asynchronous workflow responsible for processing incoming API requests and messages from queues | Distinct stages within a workflow will have their own operation, e.g. a create switch workflow will have (not extensive):<br><br>CreateSwitchOperation (Processes the initial create switch API request)<br><br>RegistrationInterventionOperation (Processes objection and no objection API requests)<br><br>RegistrationObjectionWindowOperation (Processes a queue message indicating the closure of the objection window) |
| OperationId | These are OperationId's from the Open API Specification Portal | The OperationId as specified in the OpenAPI specification, e.g.<br>• createGasRmp<br>• createRegistration<br>• createSwitch |
| Event Type | The type of event the message is related to, e.g. RMPEventSynchronisation | Primarily used to identify the event type the CSS is sending to a market participant |
| EventId | Unique Id for each inbound and outbound message | A CorrelationId may be linked to one or many EventId's e.g., the identities of operations within a workflow |
| Task | An action completed as part of an operation | A task is a distinct action in an operation e.g., perform business rules |
| MessageStage | The stage in an operation a task was logged | The message stage provides a logged checkpoint of specific tasks in an operation |

| Term | Description | Explanation |
|---|---|---|
| Stall / Stalled | Where an operation has been unable to run to completion | A stall may occur as the result of a BCDR event or a dependency failure |
| Persistent Stall | A stall where the number of attempts by the RPO has exceeded a threshold | A persistent stall that fails to resolve when the Registration Service is in a healthy state. |
| Recovery Point Ledger | A record of all recovery points | Contains one or more recovery points for an inflight operation |
| Recovery Point (RP) | The point in an operation where a recovery point is committed | Recovery points are created at the start, during and the completion of an operation |
| Recovery Point Orchestrator (RPO) | The service responsible for initiating recovery and alerting | A service that continuously monitors and responds to unresolving stall conditions |
| Recovery Point Service (RPS) | The service responsible for creating recovery points and recovering operations that have stalled | The service can initiate self-recovery from a stalled event or recover an operation detected by the recovery point orchestrator |
| Recovery Point System (RPSys) | The overall service. | The name for the collection of resources required to deliver the R0096 Message Regeneration. |

**Table 3.** Terminology

## 1.7 Solution Scope

The scope of the proposed solution is to introduce a robust recovery pattern that ensures uninterrupted message generation by the Central Switching Service for market participants. This solution is designed to automatically regenerate messages that experience a stall either during normal operations or because of a Business Continuity and Disaster Recovery (BCDR) event.

Specifically, the scope of PR8003 as set out in Table 4 is to produce the detailed design artefacts and evaluate the impact of the design on the existing service to inform the detailed impact assessment (DIA).

### 1.7.1 PR8003 Statement of Deliverables

| DEL-ID | Deliverable |
|---|---|
| DEL-01 | Demonstrable Proof of Concept (PoC) to gather greater understanding of the 'How' this change will be implemented. |
| DEL-02 | Design artifacts to be used for code modifications/development based on the activities outlined above. |
| DEL-03 | Document outlining the intricacies of the testing strategy or why intervention testing might be an appropriate testing strategy within SIT including evidence that similar work has been successfully completed in the past. |

**Table 4.** PR8003 Deliverables

### 1.7.2 PR8003 Statement of Work

See appendix 7.1 for PR8003 statement of work activity carried out.

### 1.7.3 Design Objective

The primary design objective is to minimise disruption to market participants in the event that CSS Operations stall. This involves ensuring that the CSS Operations can either continue seamlessly or be recovered swiftly to a known state with minimal impact.

By introducing Operation Recovery Points and implementing a robust Recovery Point Ledger to record recovery points, the aim is to improve operational resilience, maintain stakeholder confidence, and support the integrity and stability of operations during adverse events, minimising the impact of disruptions in the switching process and the need for manual intervention.

### 1.7.4 Boundaries & Considerations

This detailed design defines the boundaries of the recovery pattern to ensure a focused and effective delivery and consider adjacent impacts. In determining the criteria for the recovery pattern all known activities, systems, and processes directly related to the identification, mitigation, and recovery from failures in dependencies, resources, or Business Continuity and Disaster Recovery (BCDR) that could impact workflow operations have been considered.

In scope are the operations executed by market participants, the supporting infrastructure, key third-party services, and internal protocols essential to maintain operational continuity. See Table 5 for a list of in-scope items.

| Ref. | Description (In-Scope) |
|---|---|
| IS-01 | ALL CSS Registration Market Participant Operations and Resulting Notifications as described in Category 3 CSS Interface Design Specification ( See Table 1 DC-04 ) |
| IS-02 | Impact on the CSS Azure Infrastructure and High Availability Architecture |
| IS-03 | Incorporation of BCDR Exercise Recommendations |
| IS-04 | Evaluation of compliance with contracted Service Levels |

| Ref. | Description (In-Scope) |
|---|---|
| IS-05 | Refactoring of Circuit Breaker and Poison Queue patterns |
| IS-06 | Refactoring of Logging, Insights, Monitoring and Alerting |
| IS-07 | Refactoring of Error and Exception Handling |

**Table 5.** In-scope Capabilities

Out of scope are unrelated business functions, non-critical systems, and any enhancements or optimisations not directly tied to failure response or resilience. See Table 6 for a list of out-of-scope items.

| Ref. | Description (Out-of-Scope) |
|---|---|
| OS-01 | The CSS Address Service and Reporting Service |
| OS-02 | Refactoring the CSS to prioritise operations by type |
| OS-03 | Market participant systems or processes beyond the interface with the CSS |
| OS-04 | Resolution of persisting stalls resulting from misuse by market participants |
| OS-05 | Resolution of any unresolved gaps in CSS Registration business rules e.g., R0178 or similar |
| OS-06 | Prevention of stalls caused by malicious content via future unknown exploit |

**Table 6.** Out-of-Scope Capabilities

## 1.7.5 Requirements

The detailed functional requirements were agreed with Smart DCC as the target baseline for R0096. Both Smart DCC and Landmark acknowledge that these are the aspirational requirements, and the outcome of the detailed design stage may propose amendments, additions and removal of specific requirements.

### 1.7.5.1 Non-Functional Requirements

Although no explicit non-functional requirements have been specified, the detailed design target is to uphold the overarching contracted service levels as a baseline standard. Where there is any potential risk to meeting these service levels, whether due to architectural decisions, system limitations, or operational constraints—such risks will be clearly identified and addressed in this design document.

### 1.7.5.2 Requirements Traceability Matrix

Listed in Table. 7 are the requirements by ID the **[R0096-XXX]** acts as reference in the Application Architecture section of this document where a Recovery Point System Solution Building Blocks satisfies one or more requirements.

| REQ-ID | Requirement Title | Satisfied By |
|---|---|---|
| R0096-010 | Workflow Identification - Target Workflow | RPS, Reports |
| R0096-020 | Workflow Identification - Outbound Messages | RPS, Reports |
| R0096-030 | Workflow Identification - Impacted Parties | RPS, Reports |
| R0096-040 | Definitive Final Outcome for Request | RPS, RPO |
| R0096-050 | Initiate Workflow for Request | RPS, RPL |
| R0096-060 | Monitor Request Processing | RPO, RPS |
| R0096-070 | Failed Operation Resolution | RPS, RPO |
| R0096-080 | Execute Corrective Actions | RPS, RPL, RPO, Reports |
| R0096-090 | Raise Failed Operation Notification | RPO, RPL |
| R0096-100 | Maintain Set of Known Failed Operations | RPS, RPL, Reports |
| R0096-110 | Reporting Frequency of Failed Operations | Reports, RPL |

**Table 7.** Requirements Matrix (Smart DCC Requirements References)

### 1.7.6 BCDR Exercise Recommendations for Consideration

The BCDR Design Improvements (See Section 5.5 of April 2024 BCDR Exercise Report) listed in Table. 8 were reviewed as design considerations based on promoting an expedited recovery where requests stall in volume due to a BCDR event.

| BCDR ID | Observations and Lessons Learned | Included in Scope |
|---|---|---|
| SD-01 | Optimise Storage; Explore options to optimise the Storage Accounts to reduce the operating working set from the current 4.4 TB (2 years of data) to a working dataset that contains a rolling 35 days of active data to satisfy the scheduler demand to manage a rolling 28-day switch window. The reduced size would decrease the creation of the read replica. | No, this is not included in the detailed design for PR8003 as it can be implemented independently. |
| SD-02 | Use Cosmos DB for Inbound Message Storage; Explore the use of Cosmos for the storage of workflow operations dependencies, removing the reliance on Storage Accounts which do not have an RPO SLA. | No, this is not included. See section 2.2.5 for explanation. |
| SD-03 | Store Messages in Queues; Explore the use of Azure Event Grid, Service Bus or other queuing mechanism that has the capacity to store the message in the queue, removing the write delay to Blob Storage and subsequent queue update with the pointer to the message. | Yes, Azure Service Bus will replace Azure Storage Queues. |

**Table 8.** BCDR Recommendations

### 1.7.7 Risks

This section outlines key risks identified during the design process.

| ID | Risks |
|---|---|
| RK-01 | Recovery patterns must not affect service performance and continuity |
| RK-02 | Stalled operation recovery time needs to consider how it will be measured where the CSS service is degraded or recovering from a BCDR event. Consideration should be given to independent measurement of stall recovery metrics versus normal operational service level metrics. |
| RK-03 | Project start date and schedule may overlap with significant Operational Releases (.NET10 Upgrade as an example). |
| RK-04 | Project will need to account for changes applied to CSS prior to build commencement. |
| RK-05 | Inclusion and benefit of a Recovery Point Service stall injection framework for Testing. |
| RK-06 | Maintaining resource continuity across Organisations participating in project. |
| RK-07 | Long running project plan impact with branching and releasing and feature flag. |
| RK-08 | Establishing robust pattern for executing a collection of transaction commits to more than 1 container or storage type. |
| RK-09 | Extending existing K6 Load & Performance Testing Capability. |

### 1.7.8 Assumptions

This section outlines key assumptions made during the design process, including technology, user requirements, external dependencies, and project scope.

| ID | Assumptions |
|---|---|
| AS-01 | It is assumed that R0096 will go-live after REC Mod R0080 Improvements for 'Failed to Deliver' CSS Messages has been implemented and deployed. |
| AS-02 | The Address Service and Reporting Service are out of scope of R0096. |
| AS-03 | Deploy R0096 before R0169 End of Gate Closure |
| AS-04 | Deploy R0096 using Feature Flag to enable / disable the whole service |

| ID | Assumptions |
|---|---|
| AS-05 | R0096 deployed post MHHS Migration Completion |
| AS-06 | Use of K6 Load Testing Framework in PIT, Dev and LE environments |
| AS-07 | Implementation of shared testing framework in Dev, QA and DevOps processes. |

### 1.7.9 Constraints

This section outlines limitations impacting the solution, including technical, operational, and regulatory constraints, budget restrictions, timelines, resource availability, and dependencies on external systems or services.

| ID | Constraints |
|---|---|
| CN-01 | SI will be unable to create non-functional based stalls during testing without Azure Chaos inclusion. |
| CN-02 | Dual region environment usage and release scheduling if long running use of Pre-Production is required. Should the SIT environment for the SI extend to dual region on a permanent basis |

### 1.7.10 Dependencies

This section lists all external and internal dependencies, including infrastructure, software, libraries, services, and third-party components, essential for the solution's development, deployment, and operation.

| ID | Dependencies |
|---|---|
| D-01 | Azure Service Bus for Queues |
| D-02 | Azure Fabric (Power BI) for Reports and Dashboards |
| D-03 | Azure Synapse with SQL Serverless Pool |
| D-04 | Azure PIM (Privileged Identity Management) |
| D-05 | Azure Chaos Framework |
| D-06 | SI Testing Coordination Activities |
| D-07 | K6 Load & Performance Testing Framework |

### 1.7.11 Decisions

This section captures key architectural decisions, detailing choices, rationale, alternatives considered, and the implications.

| ID | Decisions |
|---|---|
| DE-01 | Remove existing Message Stage function within Operations in favour of Recovery Point System. |
| DE-02 | The first recovery point will only occur post certificate validation and schema validation of the request payload. |
| DE-03 | Use of Saga Pattern in preference to Outbox Pattern for Recovery Point Service |

# 2 Solution Concepts and Capabilities

For the purposes of illustration, only the components explicitly impacted are included in any diagrams and content, where applicable a reference anchor to a peripheral service will be included i.e., cloud platform services such as logging or API management.

## 2.1 Background Understanding

The Recovery Point System, particularly the Recovery Point Service (RPS) building block, necessitates deep integration with the CSS service architecture, an event-driven cloud platform. To effectively embed the RPS into operational workflows, the system must undergo core refactoring. This foundational change is essential to ensure that recovery capabilities are seamlessly aligned with the platform's event-driven nature and can support resilient, re-entrant request processing.

The following content describes the existing system patterns to provide a background understanding of Operations, using the 'CreateRegistrationOperation' which is performed in response to a POST to the API '/registrations' as an example throughout this design document.

### 2.1.1 Existing High Level Request Flow

The CSS on receipt of a market participant API request as illustrated in Figure 1 will perform synchronous operations (Inbound Request) to authenticate and authorise the request, schema validate the message, store and queue the resulting work (Scheduled Work Queue).

The asynchronous process (Process Scheduled Work) is triggered by the presence of work in the Scheduled Work Queue and performs the operation requested by the market participant. The operation performs tasks (actions) to retrieve the data required, apply business rule validation, execute the operation specific action (e.g., create registration), store and queue the resulting event notifications for market participants for immediate delivery or future based events (e.g., Gate Closure).

Each task (action) in an operation reads and writes to the Azure Cosmos Database and the resulting event messages stored and queued in Azure Storage Accounts.

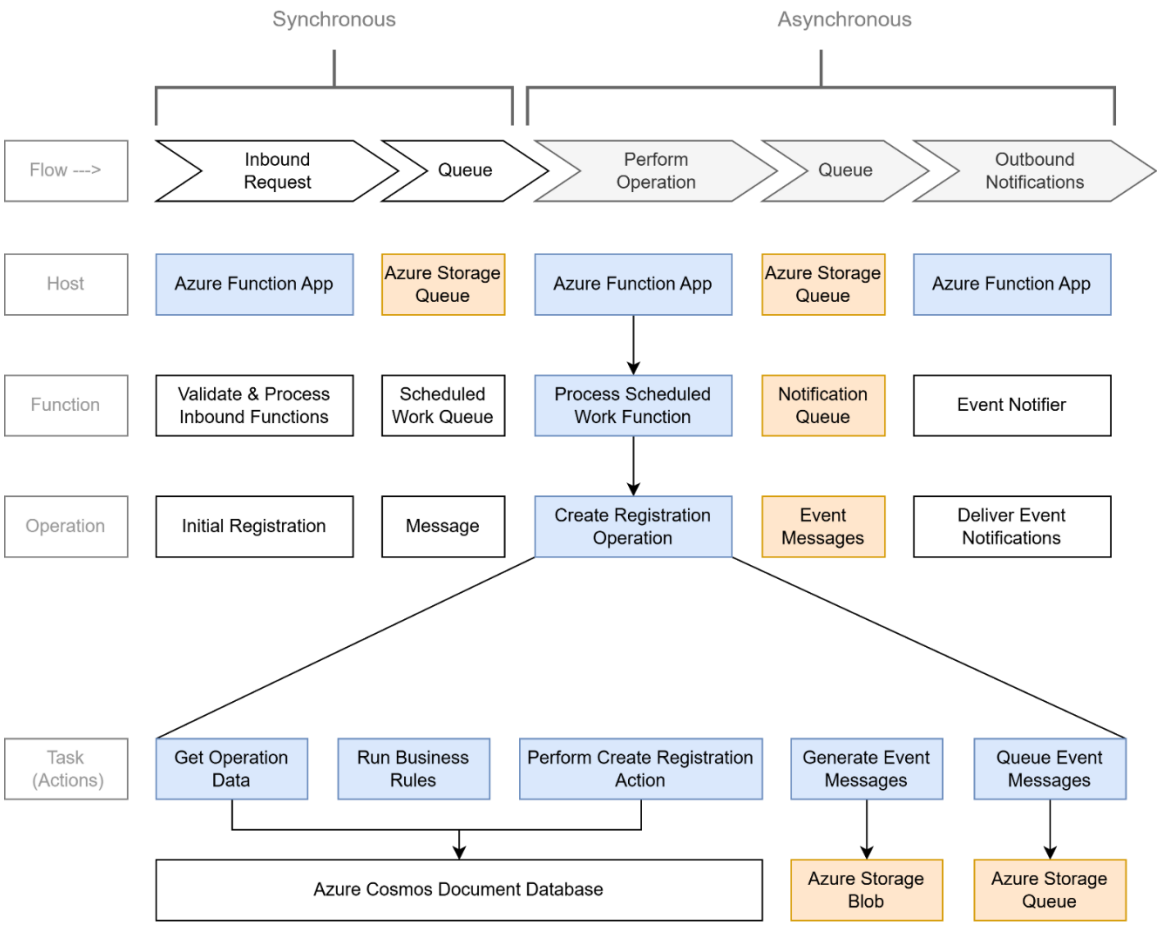



**Fig.1** Existing High-Level Request Flow CSS Service Design

## 2.2 Conceptual Solution Overview

As outlined by the high-level request flow in Figure. 1 a mechanism for recovering an Operation is required and it must be capable of restarting the operation from the last successful recovery point to ensure:

- Duplicate records are not created.
- Business Rules are not violated (e.g., multiple pending registrations for the same MPxN).
- Participants do not receive duplicate notifications they may not be able to tolerate.
- Re-processing is minimised to promote recovery time.

The Recovery Point System (RPSys) is the solution proposed to meet the requirements and is designed to enhance the resilience, observability, recoverability of stalled operations within the Central Switching Service (CSS) and is independent to the CSS. It introduces a structured mechanism for tracking, persisting, and recovering the state of in-progress operations across both synchronous and asynchronous workflows.

The system is composed of three decoupled but cooperative components:

- The **Recovery Point Ledger (RPL)**, responsible for storing recovery points for in progress operations and their dependent data.

- The **Recovery Point Orchestrator (RPO)**, responsible for re-queuing stalled operations, monitoring and alerting.

- The **Recovery Point Service (RPS)**, responsible for creating and updating recovery points within ANY Operation (as published in the Category 3 CSS Interface Design Specification 8.8.2*) whether synchronous or asynchronous and recovering a stalled operation.

The Recovery Point Ledger and Orchestrator will operate independently of the Central Switching Service both functionally and non-functionally. The Recovery Point Service will be integrated as a shared class library in all CSS operations with a fail-safe to permit an Operation to continue should the RPSys be unavailable.

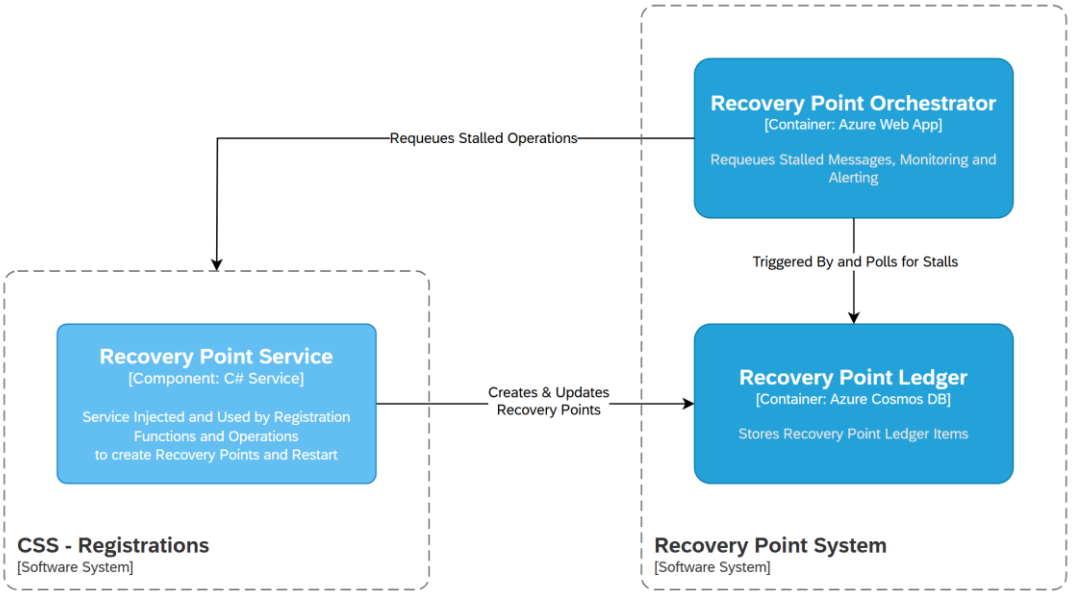



**Fig. 2** Recovery Point System Building Blocks

*Or the latest version of this specification at the point of deployment of the R0096 solution release candidate. Any effort for additional Operations or modifications to existing Operations will be accounted for in the related impact assessment for that change.

### 2.2.1 Recovery Point Ledger

The Recovery Point Ledger will provide the durable storage for Recovery Points (including payload), the RPL will use Azure Cosmos DB with multi-region writes, where any region will accept a write.

The Recovery Point Service will use a preferred region configuration so writes remain in the region context the RPS instance is executing within unless a failure occurs at which point the write is automatically routed to the available region.

The Analytical Store will be enabled on the RPL Azure Cosmos DB to provide near real-time dashboards and reporting, presented using Azure Fabric (Power BI).

### 2.2.2 Recovery Point Orchestrator

The Recovery Point Orchestrator responds to a Recovery Point entering a 'stalled' state, triggered either directly via the Cosmos Change Feed or a result of a message type exceeding a timer threshold. The trigger and the threshold must account for operations that are in a 'wait' state and therefore not stalled but delayed and account for known and unknown burst conditions, where a known condition is gate closure and a message could be waiting for up to 35 mins to complete processing.

The Recovery Point Orchestrator is responsible for alerting and detecting persistent stalls. In addition, the RPO will be aware of overall system health to determine if re-queuing can be attempted and the same health information used to augment alert notification content.

### 2.2.3 Recovery Point Service

The Recovery Point Service will closely follow the published 'Saga' pattern, which manages transactions as a sequence of 'local transactions'. Each Task (or Action) in an Operation is a local transaction in this context.

The benefits of combining Saga + Recovery Points

- Resilience: If a step fails, the system can resume from the last successful point.
- Observability: Each step's status is persisted and queryable.
- Auditability: Full history of what happened and when.
- Decoupling: Each step is logically isolated and can be retried or compensated independently.
- Scalability: Works well in distributed systems (e.g., cloud-native apps).

Alternative patterns were explored (notably the Outbox pattern, where all write events in operation are executed as a collection of transactions by separate service) and deemed too complex, requiring additional platform resources and transaction commit rollback. However, the Recovery Point Service implementation for writing asynchronously to the Recovery Point Ledger is similar in principle to the Outbox pattern.

### 2.2.4 Message Queue Replacement

Introducing the Recovery Point System requires a message queue service that can detect duplicates to ensure that re-entrant recovery doesn't send a duplicate notification to a market participant or re-process at gate-closure. Additionally, the ability to query future dated messages (e.g., delayed switch start date), count queue items, inspect in-queue stalls and trigger compensating actions or alerts are also beneficial to real-time Operation recovery.

As a consequence, Azure Storage Queues will be replaced by Azure Service Bus Premium, it should be noted that this still relies on Azure Storage Accounts, however this is mitigated by the RPL using Cosmos for durable storage and the Recovery Orchestrator re-queuing the message from the recovery point ledger item.

### 2.2.5 Existing Azure Blob Storage Statement

As part of enhancing the Central Switching Service Registration process with the new Recovery Point System, a consideration was made regarding the continued use of Azure Blob Storage for storing operational data used during registration steps. While other services, such as Cosmos DB, were reviewed, it is recommended that Azure Blob Storage remain in use for this purpose.

This recommendation is based on a balance of practical and technical factors. Blob Storage is already integrated into the existing solution, and refactoring to another service would require significant development effort to refactor all functions and operations consuming the storage, extensive testing, and validation, without offering substantial benefit in this context, where transactional storage is isolated from the durable recovery point storage. Moreover, the new Recovery Point System now captures a copy of each operational message directly within the Recovery Point Ledger, which is securely stored in Cosmos DB with multi-region resilience.

This means that if a storage file is temporarily unavailable or lost due to a cross-region replication delay, the Recovery Point Ledger retains the necessary data to allow the operation to resume from its last known successful step. This adds an important layer of protection and redundancy, reducing dependency on any single data store for operational continuity.

Retaining Azure Blob Storage allows the solution to continue as is while leveraging the Recovery Point System for improved reliability and recoverability, without introducing unnecessary complexity or risk.

It is further recommended that the working storage dataset for Registrations is reduced to a rolling 35 days of message transactions and older transactions are put into an archive storage tier, improving the replication failback recovery to a known size range and predictable replication time.

## 2.2.6 Refactored Function using Recovery Point Ledger

Figure 3. Illustrates the introduction of a ‘get ledger data’ method that is invoked if the event Id of a message has a corresponding recovery point ledger item, when true the ledger item data is retrieved and injected into the Operation and used if the system is unable to retrieve from its primary storage location.

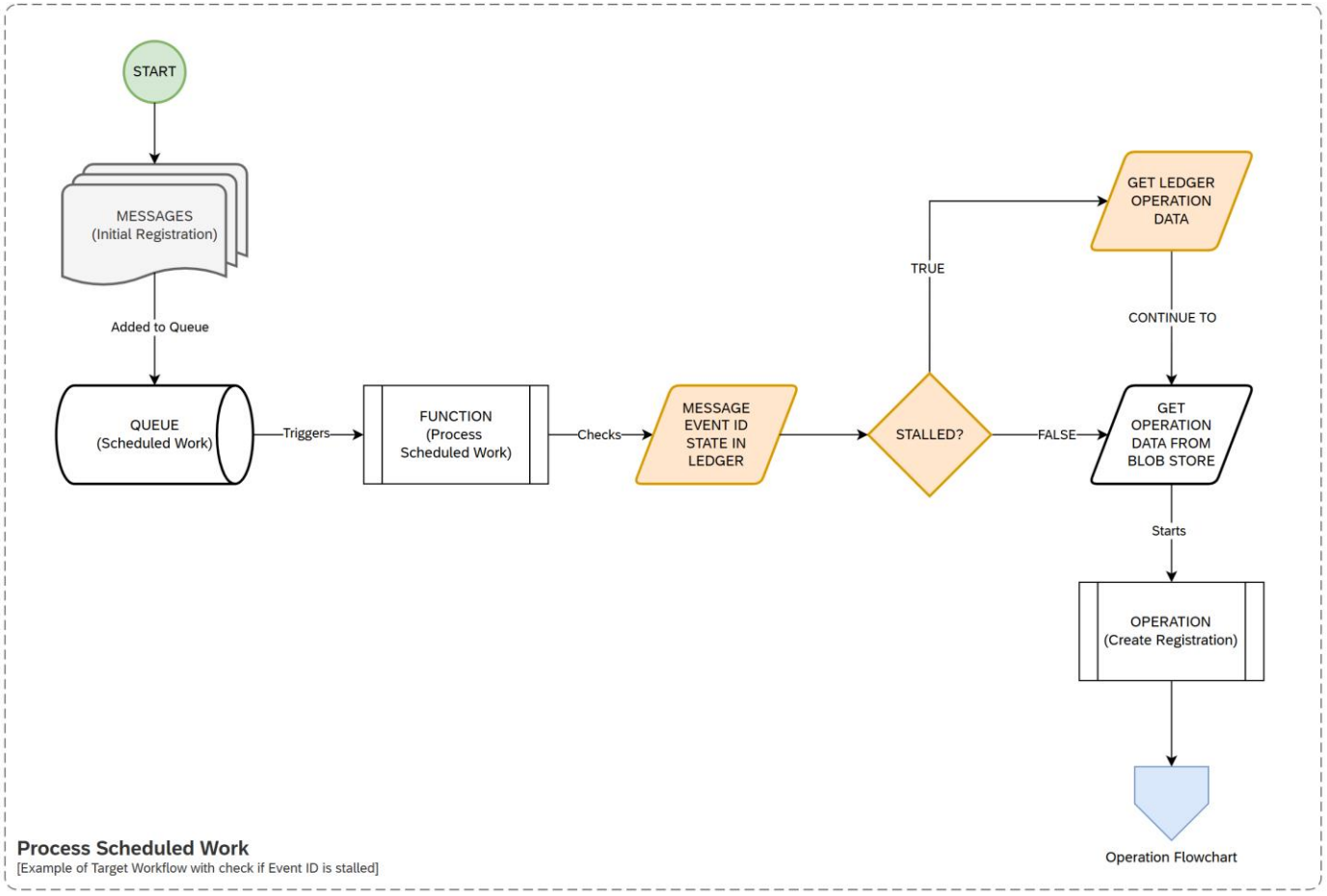



**Fig. 3** Example of refactoring an Azure Function App to respond to re-entrant messages

## 2.2.7 Refactoring an Operation to use the Recovery Point Service

Figure. 4 illustrates how an Operation responds to the injection of data into the Operation where a Recovery Point Id is present in the injected payload.

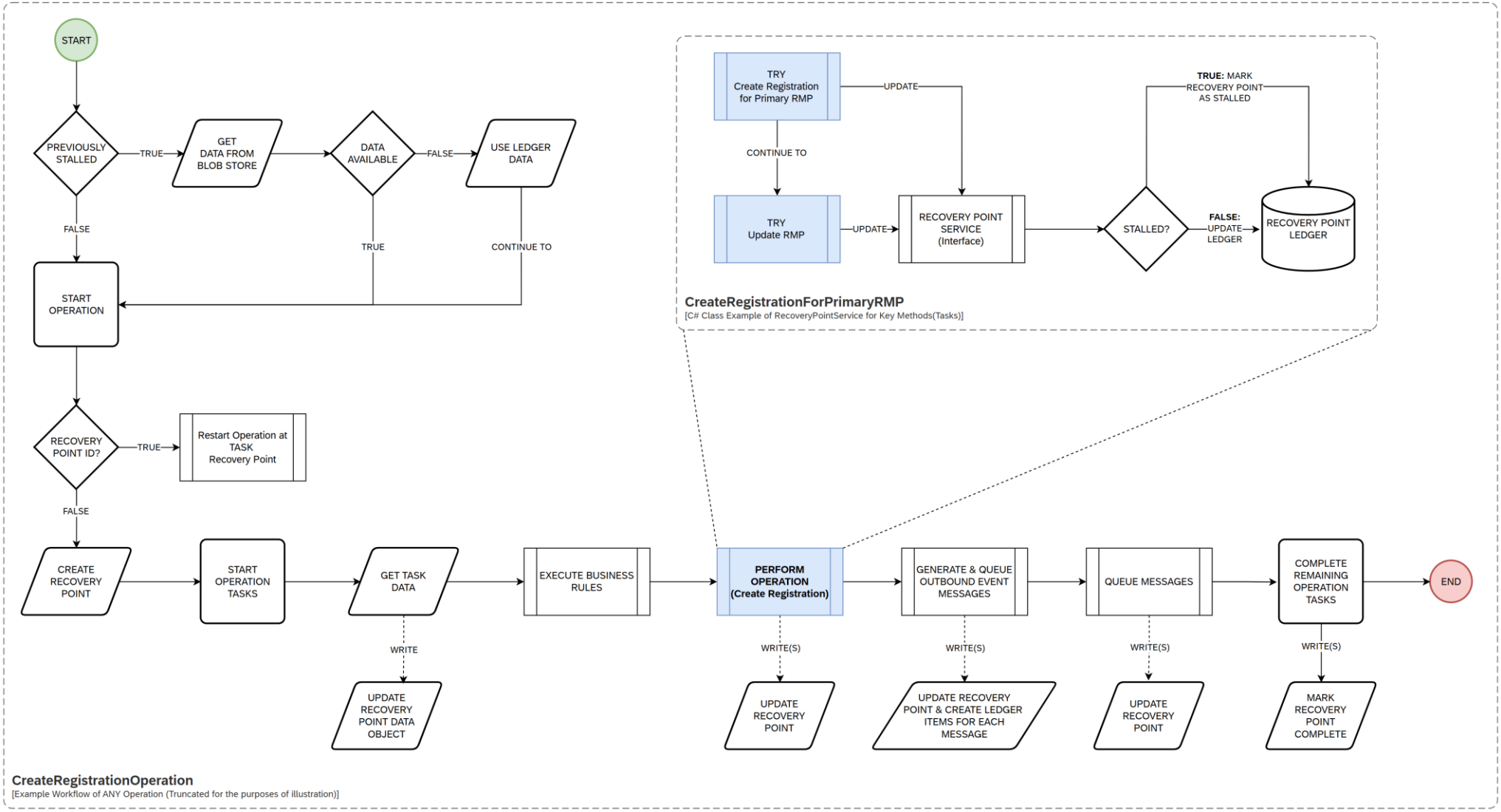



**Fig. 4** Example of refactoring an Operation to create recovery points and resuming from a recovery point.

# 3 Solution Architecture Building Blocks

This section outlines the detailed design, components, integration, and design principles of the Recovery Point System, detailing how it meets the requirements.

## 3.1 Information Architecture

### 3.1.1 Overview

This section describes the structure, flow, and lifecycle of information within the Recovery Point System (RPSys). It outlines how recovery point data is modelled, persisted, queried, and secured across distributed components. The design ensures traceability, consistency, and recoverability of Operations within the Central Switching Service (CSS).

### 3.1.2 Data Domains and Entities

#### 3.1.2.1 RecoveryPoint

The RecoveryPoint entity represents the state of a step within an operation.

| Field | Content | Description |
|---|---|---|
| recoveryPointType | [Operation, Notification] | Type of Recovery Point |
| recoveryPointId | unique_id | Unique Identifier |
| correlationId | correlation_id | Correlation Id |
| eventId | event_id | Unique Event Id |
| operationId | operation_id | Name of Operation |
| messageStage | number | Message stage enum reference |
| recoveryPointStage | number | Recovery Point stage enum reference |
| status | [Completed, Running, Stalled, ReQueued] | Status of Operation |
| created | ISO 8601 Timestamp | Timestamp |
| lastUpdated | ISO 8601 Timestamp | Timestamp |
| lastUpdatedBy | Entity Type | Default value of “RecoveryPoint” |
| recoveryModel | {{ET:Alliance...}, {ET:Participant}} | Array of Data Objects used in Operation and Corresponding Notifications |
| participantEvents | [{mpId, role, eventType}] | List of Participants by Role and Event Notification Type |
| processQueueName | Queue Name | Specific queue configured for the market participants notifications. |
| queueVisibilityTime | DateTime when the item should be processed | Required to give the queue time to process the original message |
| requeueCount | number | Count of number of retries since stall |
| stalledReason | Storage Timeout | Reason for stall returned by local transaction |
| errorMessage | null | Error Message if present |
| metadata | {mpxn:mpxn_value} | Additional data for reporting dashboards |

**Table 8.** Recovery Point Schema

### 3.1.3 Data Flow and Lifecycle

- **Creation:** Recovery Points are created at the start of each Operation step.
- **Update:** Status is updated to "Completed" or "Stalled" based on outcome.
- **Query:** The orchestrator and operations  query the ledger to determine the last successful step.
- **Retention:** Recovery Points will be retained for a configurable period (e.g., 7* days) for audit and replay within Cosmos, and 90* days in the Analytical Store for Dashboards and Reports.
- **Archiving/Deletion:** Records will be deleted based on contracted data governance policies.
- **Triage:** Recovery Points that fail to resolve and exceeds the re-queue threshold will be moved to a dedicated container for resolution.

*Periods to be agreed to ensure optimal query performance return latency.

### 3.1.4 Storage and Access

- **Storage Platform:** Azure Cosmos DB (multi-region, multi-write)
- **Partitioning:** Partitioned by operationId for efficient access and scalability
- **Indexing:** Indexed on status, timestamp, eventId and operationId for fast lookups
- **Access Patterns:**
    - QueryStalledOperations()
    - GetRecoveryPoint(eventId)
- **Containers:** RecoveryPoints, RecoveryPointsTriage

### 3.1.5 Data Governance and Security

- **Access Control:** Enforced via Azure RBAC and managed identities.
- **Encryption:** All data encrypted at rest and in transit
- **Audit Logging:** All writes logged for traceability (Azure Log Analytics).
- **Compliance:** Designed to meet data retention and audit requirements for regulated environments.

### 3.1.6 Integration with CSS Data Model

- Recovery Points reference CSS domain entities (e.g., Registration, Rmp) via serialised payloads in recoveryModel.
- No direct coupling to existing CSS data stores; all integration is via Operation context and payload snapshots.
- Enables replay and debugging without requiring access to live CSS database container(s).

## 3.2 Application Architecture

### 3.2.1 Overview

This section details the application's architecture, covering the key components, their interactions, and design patterns. It includes an overview of the system's structure, technology, modules, and dependencies. Diagrams illustrate the flow of data and control. Key architectural decisions, rationale, and trade-offs included for reference. Figure 5 illustrates the key components and interactions.

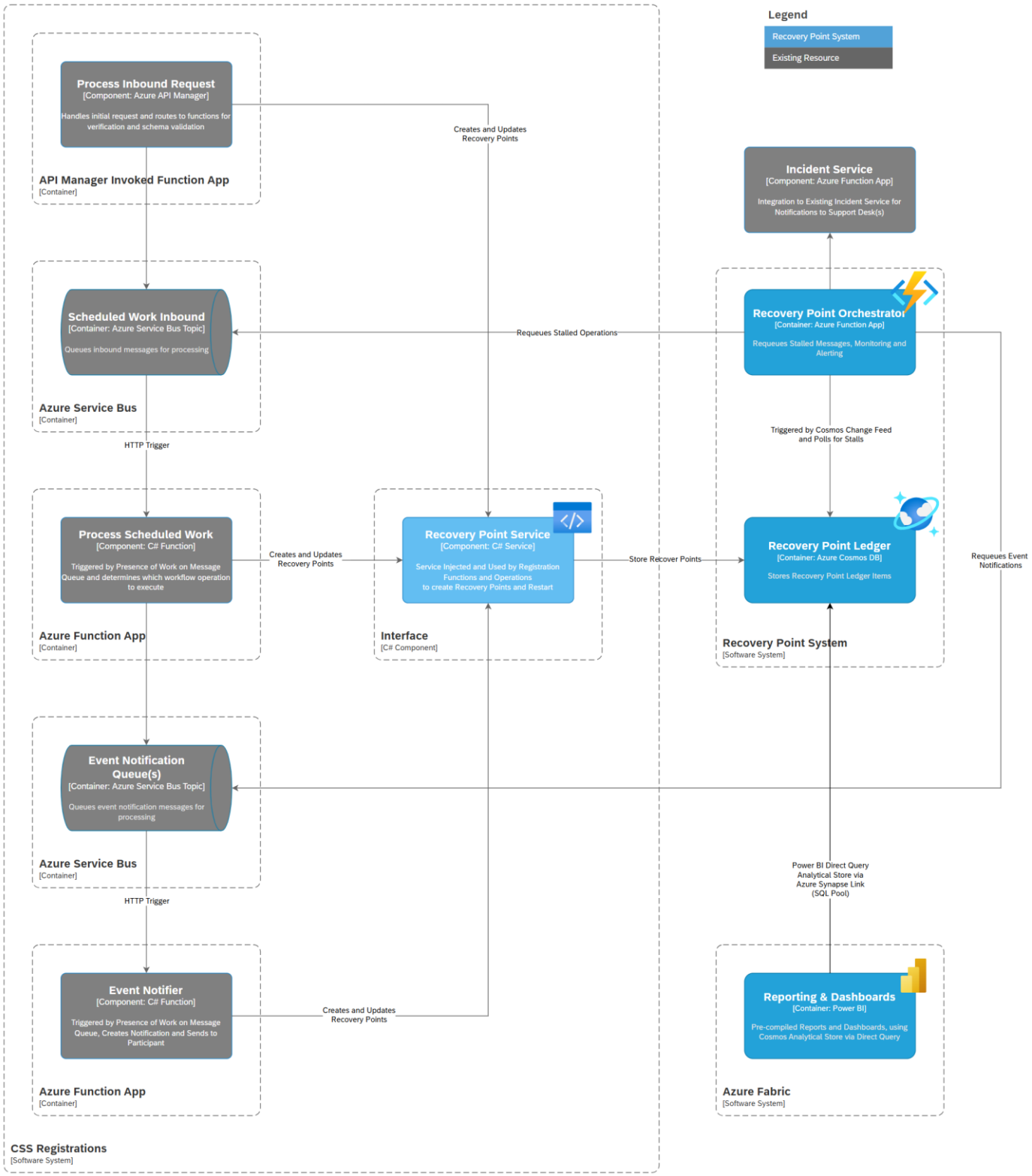



**Fig. 5** Recovery Point System integration to CSS Registrations System

## 3.2.2 Recovery Point System Solution Building Blocks

### 3.2.2.1 Recovery Point Ledger (RPL)

**[R0096-050, 080, 090, 100, 110]**

- **Purpose:** Acts as the durable store for all recovery points.
- **Technology:** Azure Cosmos DB Container, multi-region, multi-write enabled for high availability and low latency writes.
- **Responsibilities:**
  - Persist recovery point records for each operation step and triage of persistent stall items
  - Store metadata such as operation ID, task name, status, timestamp, and payload snapshot.
  - Support querying for last successful or stalled steps.
  - Provides partitioned shared change feed lease container to trigger RPO regional Function Apps to ensure stalls are only requeued once.

### 3.2.2.2 Recovery Point Orchestrator (RPO)

**[R0096-040, 060, 070, 080, 090]**

- **Purpose:** Is triggered by and monitors the ledger for stalled operations and orchestrates their recovery.
- **Technology:** Azure Function(s)
- **Responsibilities:**
  - Detect stalled Operations based on recovery point status.
  - Re-queue Operations for retry into existing queues.
  - Checks RecoveryPointTriage queue for ‘Pending’ items and re-injects to RPL
  - Emit alerts and metrics for operational visibility.
  - Conducts periodic health checks of dependent registration resources e.g., Azure Service Bus Queue availability and Last Processed Datetime.
  - Can suspend recovery of items on detection of dependency failure.

### 3.2.2.3 Recovery Point Orchestrator (RPO) Functionality

When a recovery item enters the “stalled” state the RPO will add the message back onto the queue specified by the ProcessQueueName field.

A Recovery Point item is marked as “stalled” by the Operation when all ‘local’ strategies are exhausted, or an error or exception has occurred. The Recovery Point Orchestrator will mark a Recovery Point as stalled only when a configurable time period has elapsed as part of a scheduled check.

**Requeue Step Outline:**

- The RPO will query the message store for the message to requeue (to ensure the latest data is used if available)
- If unable to find the message in storage, the RPO will use the data in the ledger and write the data from the ledger to the message store.
- The RPO will not check whether the message is already in the queue before adding a message to the Service Bus instance which will detect duplicates and remove.
- The RPO will update the Status to “ReQueued”
- Add finally increment the ReQueueCount by 1

**Persistent Stall State:**

Where a Recovery Point item for an Operation is re-queued by the RPO and the Operation continues to stall at the same point (Task/Action) the following steps will apply.

- If the reQueueCount exceeds reQueueCountThreshold value and the registration service is Healthy then:
    - Create an alert notification
    - Move the Recovery Point item into RecoveryPointTriage Cosmos container, retaining the 'stalled' status

**Recovery from Persistent Stall State:**

To re-instate a Recovery Point item that has entered a persistent stall state the following steps apply.

- Given the Recovery Point item is in the RecoveryPointTriage container and the underlying stall condition has been resolved then:
    - Update the Recovery Point status to 'Pending'*

*The RPO will be triggered via the Cosmos Change Feed to move the item into the RPL for re-injection into the appropriate queue for processing.

#### 3.2.2.4 Recovery Point Service (RPS)

**[R0096-010, 020, 030, 040, 050, 060, 070, 080]**

- **Purpose:** A shared service integrated into all CSS operations to manage recovery point lifecycle.
- **Responsibilities:**
  - Create and update recovery points during operation execution.
  - Abstracts persistence and orchestration logic from Operation.
  - Maintains latest Cosmos _etag for later recovery point writes.
  - Can be by-passed in the event that the RPL is not available in either region.
- **Integration with CSS Operations:**
  - The RPS is injected into all CSS operation classes (e.g., CreateRegistrationOperation) as a shared dependency.
  - Each operation step is wrapped with recovery logic that:
    - Saves the Market Participant mpId, Role and Event Notification(s)
    - Saves a "Started" recovery point before execution.
    - Saves a "Completed" recovery point upon success.
    - Saves a "Waiting" recovery point when the condition exists.
    - Saves a "Stalled" recovery point with error details on failure.
    - On restart*, an operation queries the RPL to determine the last successful step and resumes from that point.
  - Fail-safe method to prevent execution of subsequent Operation tasks should the RPL not be available within 200ms.
    *Applicable to Functions hosted in Azure Function Apps.

### 3.2.3 RPSys Reporting and Dashboard Building Blocks

This section describes the solution building blocks involved in enabling reporting and operational insight for the Recovery Point System.

**[R0096-010, 020, 030, 080, 100, 110]**

#### 3.2.3.1 Cosmos DB Analytical Store

- **Purpose:** The Cosmos DB Analytical Store is used to enable near real-time querying and reporting on operational data stored in the Recovery Point Ledger, without impacting transactional performance.
- **Responsibilities:**
  - Automatically synchronise data written to the Recovery Point Ledger (Cosmos DB transactional store). Max. 2 min delay.
  - Maintain a columnar, read-optimised version of the data for analytical workloads.
  - Support continuous, serverless data extraction to Azure Synapse.
- **Technology:**
  - Azure Cosmos DB with Analytical Store enabled.
  - Multi-region write support for resilience.
  - Serverless integration with Synapse via Synapse Link.
- **Integration Points:**
  - Feeds Azure Synapse Analytics through Synapse Link, enabling Power BI reporting.
  - Integrated into the existing Azure monitoring and security framework (e.g., RBAC, Private Link).

#### 3.2.3.2 Azure Synapse Analytics (Dedicated SQL Pool)

- **Purpose:** Azure Synapse provides the central analytical layer to query, model, and shape data from the Cosmos Analytical Store before exposing it to visualisation tools like Power BI.
- **Responsibilities:**
  - Query and aggregate recovery point data for trend analysis and reporting.
  - Provide a performant, scalable SQL interface over semi-structured data.
  - Serve as a transformation layer for creating reusable datasets.
- **Technology:**
  - Azure Synapse Analytics with dedicated SQL pool.
  - Serverless Synapse Link connection to Cosmos DB Analytical Store.
  - Native integration with Azure Data Lake (for historical data retention).
- **Integration Points:**
  - Ingests data from the Cosmos DB Analytical Store via Synapse Link.
  - Exposes cleansed and transformed views to Power BI datasets.
  - Integrated with enterprise security (e.g., Azure Entra Id, managed identities, and data masking).

#### 3.2.3.3 Power BI (via Azure Fabric or Office 365 Tenant Service Integration)

- **Purpose:** Power BI provides the visual layer for business users (Landmark/Smart DCC) and operational teams to monitor the health, performance, and reliability of the Recovery Point System.
- **Responsibilities:**
  - Present interactive dashboards and reports based on Synapse views.
  - Enable tracking of operation status trends, recovery volumes, failure rates, and response times.
  - Facilitate role-based reporting and executive summaries.
- **Technology:**
  - Power BI Pro and Power BI Premium* (embedded).
  - Integration with Azure Synapse datasets via DirectQuery.
- **Integration Points:**
  - Connects to Azure Synapse views or datasets as the reporting source.
  - Supports secure access using Azure Entra ID authentication.

*Azure Fabric optional if adopted for unified data access and governance.

### 3.2.4 Design Patterns and Rationale

- **Saga Pattern with Persistence:** Each operation is treated as a saga, with recovery points acting as durable checkpoints.
- **Outbox Pattern (implicit):** Recovery points serve as a form of outbox for tracking side effects.
- **Separation of Concerns:** RPL and RPO are decoupled from CSS, enabling independent scaling, monitoring, and evolution.

### 3.2.5 Scalability and Resilience

- Cosmos DB's multi-region write capability ensures high availability and low-latency persistence.
- Azure Service Bus provides deduplication, dead-lettering, and retry policies for robust orchestration.
- Stateless orchestrator and service components allow horizontal scaling.

### 3.2.6 Security and Observability

- All recovery point writes are authenticated and authorised via managed identities.
- Recovery point data is encrypted at rest and in transit.
- Logs, metrics, and alerts are emitted to Azure Monitor and integrated with dashboards.

## 3.3 Data Architecture

### 3.3.1 Overview

This section describes the physical and logical structure of data storage and movement within the Recovery Point System (RPSys). It outlines how data is partitioned, replicated, queried, and integrated with analytics platforms, ensuring durability, scalability, and accessibility across regions and services.

### 3.3.2 Data Storage Architecture

- **Primary Store:** Azure Cosmos DB
  - Containers: RecoveryPoints, RecoveryPointsTriage
  - Partition Key: operationId
  - Indexing: Enabled on status, timestamp, operationId, eventId
  - Consistency Level: Session (for balance between performance and consistency)
  - Multi-Region Writes: Enabled for high availability and write resilience.
  - Analytical Store: Enabled for historical querying via Synapse/Power BI
  - Change Feed: Enabled to trigger requeue on stall event.

#### 3.3.2.1 Data Model

- JSON document structure aligned with the RecoveryPoint entity.
- Flattened fields for analytics (e.g., recoveryPointStage, recoveryModel)
- Serialised payloads stored in payloadJson for replay and debugging.

### 3.3.3 Data Flow and Access Patterns

| Flow | Source | Destination | Method |
|---|---|---|---|
| Operation step execution | CSS Operation | Cosmos DB | RecoveryPointService.SaveAsync() |
| Recovery monitoring | Cosmos DB (Change Feed) | Recovery Point Orchestrator | Triggered Function |
| Analytics | Cosmos Analytical Store | Power BI / Synapse | DirectQuery via Synapse Link |
| Resumption logic | Cosmos DB | CSS Operation | GetLastSuccessfulStepAsync() |

**Table 9**. Flow and Access Patterns

### 3.3.4 Data Integration and Analytics (Dashboards and Reports)

#### 3.3.4.1 Power BI + Synapse Link

- **Integration:** Cosmos DB Analytical Store → Synapse Views → Power BI
- **Query Mode:** DirectQuery for near real-time dashboards.
- **Data Model:** Flattened views with calculated fields (e.g., duration, retry count).
- **Use Cases:**
  - Stalled operation trends.
  - Step-level success/failure rates.
  - Recovery time analysis.

### 3.3.5 Data Governance and Security

- **Access Control:** Azure RBAC and managed identities for all Cosmos DB and Synapse access
- **Encryption:** Data encrypted at rest and in transit
- **Auditing:** Access logs and change tracking via Azure Monitor
- **Retention Policy:**
  - Operational data: 35 days (configurable)
  - Analytical data: 90 days (configurable)
- **Compliance:** Aligned with GDPR and data handling policies as defined in the contract.

### 3.3.6 Scalability and Performance

- **Horizontal Scaling:** Cosmos DB configured with auto scale throughput (RU/s) per region.
- **Partitioning Strategy:** Ensures even distribution of writes and queries.
- **Query Optimisation:**
  - Uses indexed fields for filtering and aggregation.

### 3.3.7 Recovery Points

#### 3.3.7.1 Synchronous Operation recovery points

(Default Type: Operation)

- On Start
- Unsigned Payload Stored
- Message Queued (Type: Queue)
- Audit Log
- Store Response
- Log Metrics
- On Completion

#### 3.3.7.2 Asynchronous Operation recovery points

(Default Type: Operation)

- On Start
- GetOperationData()
- PerformOperationAsync()
- RunRulesAsync()
- GenerateAndStoreEventMessage()
- QueueOutboundMessages()
- GenerateAndStoreScheduledWorkMessages()
- QueueScheduledMessages()
- On Completion

## 3.4 Integration Architecture

### 3.4.1 Overview

This section outlines how the Recovery Point System (RPSys) integrates with the broader Central Switching Service (CSS) platform and supporting Azure services. It describes the communication flows, integration points, and technologies used to ensure reliable, secure, and scalable interactions between components.

### 3.4.2 Integration Components

#### 3.4.2.1 Recovery Point Service (RPS)

- **Integration Type:** Shared service
- **Consumers:** All CSS operation classes (e.g., CreateRegistrationOperation)
- **Responsibilities:**
  - Called synchronously within operations to persist recovery points.
  - Retrieves last successful steps to support resumption logic.
- **Interface:** IRecoveryPointService
- **Data Format:** JSON-serialised payloads

#### 3.4.2.2 Recovery Point Ledger (RPL)

- **Integration Type:** Azure Cosmos DB (multi-region, multi-write)
- **Accessed By:** RPS and RPO
- **Responsibilities:**
  - Stores recovery point documents.
  - Supports querying by operationId, correlationId, eventId, and status.
- **Partitioning Strategy:** Partitioned by operationId
- **Consistency Model:** Session/Strong

#### 3.4.2.3 Recovery Point Orchestrator (RPO)

- **Integration Type:** Event-driven.
- **Triggers:**
  - Recovery Point Status == “Stalled”
  - Recovery Point Status == “Waiting” + QueueVisibilityTime > Current Data/Time + 5mins*
  - Recovery Point Status ==”Pending” + In RecoveryPointsTriage container
  - Health Status
  - Circuit Breaker Status
- **Actions:**
  - On Cosmos Change Feed item containing “Stalled” or “Pending” in RecoveryPointsTriage container
  - Scheduled query to verify status and change to “Stalled” if stall condition detected e.g., “Waiting” + QueueVisibilityTime > Current Data/Time + 5mins*
  - Re-queues operations to Registration Message Queues (Service Bus).
  - Sends alerts to Azure Monitor and Service Now via Incident service.
  - Suspends recovery if circuit breaker is open or health of dependency degraded.
- **Message Format:** JSON with operation metadata and recovery context.

*Adjustable based on NFRs

#### 3.4.2.4 Service Bus

- **Integration Type:** Event-driven.
- **Triggers:**
  - Azure Function App to respond to new messages.
- **Actions:**
  - Initiates asynchronous Operations.
  - Delayed message delivery
  - Moves message exceeding MaxDeliveryCount to dead-letter-queue (DLQ)
- **Message Format:** JSON.

#### 3.4.2.5 Cosmos Analytical Store

- **Integration Type:** Continuous Synchronisation (Serverless, Near Real-Time)
- **Triggers:**
  - Updates to Recovery Point Ledger items within the Cosmos DB transactional container (e.g. changes to status, save points, or metadata).
  - Automatically reflects inserts and updates from the operational store into the Analytical Store without manual ETL.
- **Actions:**
  - Makes updated data queryable by Azure Synapse through Synapse Link, supporting ad hoc analysis, trend evaluation, and dashboard reporting.
  - Enables near real-time analytics and insights with no performance impact on the transactional workload.
  - Facilitates scalable, serverless data access for Power BI via Azure Synapse Link.
- **Message Format:**
  - Data is stored and made available in JSON document format, preserving the structure of the original Recovery Point item for consistency and traceability.
- **Operational Characteristics:**
  - No data movement or transformation required—Analytical Store is a columnar, automatically updated representation of the operational data.
  - Supports schema-on-read, allowing flexible access to semi-structured data through SQL queries in Azure Synapse.

#### 3.4.2.6 Azure Synapse (SQL Serverless Pool)

- **Integration Type:** On-Demand Query Execution
- **Triggers:**
  - Analyst or system-initiated queries via Power BI.
  - Scheduled or ad hoc access to Cosmos DB Analytical Store via Synapse Link for real-time operational reporting.
- **Actions:**
  - Executes T-SQL queries over JSON-based Recovery Point data exposed from the Cosmos DB Analytical Store.
  - Generates aggregations, summaries, and views of operational metrics (e.g. operation durations, failure rates, recovery retries).
  - Serves structured datasets and SQL views as data sources for Power BI dashboards.
  - Supports role-based filtering and access control through integration with Azure Entra ID security.
- **Message Format:**
  - Reads and queries JSON documents from the Analytical Store in their native format.
  - Outputs tabular result sets in standard SQL formats for integration with BI tools.

- **Operational Characteristics:**
  - Serverless and scalable, allowing cost-effective use with no need for dedicated infrastructure.
  - Optimised for read-heavy, analytical workloads across large volumes of semi-structured data.
  - Automatically adapts to the schema of the JSON documents from the source without pre-defining fixed table structures.

### 3.4.3 Refactoring of Existing Integration Components

| Component | Count | Description |
|---|---|---|
| Functions | 15 | Functions within CSS Platform hosted with Azure Function App(s) |
| Operations | 53 | Market Participant Operations |
| Tasks per Operation | 4-18 | Tasks executed within an Operation |
| Recovery Points | 3-12 (Average 8) | Recovery Points (After Data Gets and After Data Writes) |
| Storage Queues | 12 | Processing and Outbound Notification Queues |
| Poison Queue | 12 | Poison Queue Processing and Incident Service Integration |
| Circuit Breaker | 1 | Writes to uncommitted transactions to a temporary location when an unresolving event is in progress |
| Incident Service | 1 | Raise incidents from events into ZenDesk and onwards into Service Now |

**Table 10.** Resource requiring refactoring.

#### 3.4.3.1 CSS Operations

Outline of refactoring activity:

- Implement the RecoveryPointService Interface for each Task in an Operation
- Removal of messageStage service and interfaces
- New method which sets the ledger data within the operation (Operation data is either from storage or ledger, not a merge of the two).
- Try-Catch blocks refactored to ensure all tasks in an operation can mark a task as “Stalled”.
- Update all Tasks for use retry logic for transient failures
- Update all Tasks to use structured logging in try-catch blocks
- Updates the status in the ledger (via RecoveryPointService) with “Waiting”, “Running”, “Completed”.
- Outbound Message Handling, when an outbound message is enqueued:
  - A new Recovery Point Ledger entry is created via RecoveryPointService with at minimum the fields in Table 13. (See Information Architecture section for full schema)

| Field | Description |
|---|---|
| Status | Set to "waiting" to indicate a pending future operation. |
| ProcessQueueName | The target queue, typically " event-notification". |
| QueueVisibilityTime | Scheduled time + 5 minutes* — allows the queue time to process the original message before RPO acts. (*Adjustable based on NFRs) |
| notificationModel | A snapshot of the outbound payload (e.g., CssUnprocessedMessage, outbound message content). |

**Table 11.** Outbound Recovery Point for Queued Message

#### 3.4.3.2 Process Scheduled Work Function

**Objective:** Integrate the RecoveryPointService to manage durable checkpoints when handling scheduled work function operations.

Steps to implement:

- Integrate RecoveryPointService Interface
  - Ensure the Operation function is injected with and uses the shared RecoveryPointService.
- Query the Recovery Point Ledger (RPL)
  - On function trigger, use RecoveryPointService to retrieve the recovery point ledger entry for the operation.
- Evaluate Ledger Entry
  - If recovery data is found:
    - Call the new operation method SetRestorePointData to initialize the operation with ledger context.
- If no ledger data exists:
  - Create an initial recovery point for the operation.
- Update Ledger Pre-Post-Processing
  - After completing GetOperation:
  - Refresh or upsert the ledger entry with the latest operation data, regardless of whether an entry previously existed.
- Implement Azure Service Bus endpoint and remove Azure Storage Queue references.

#### 3.4.3.3 Process Outbound Queue Function (Queue Recovery)

**Objective:** Handle outbound messages that were previously scheduled by ensuring their recovery point is up-to-date and initiating their processing through the appropriate service.

**Pre-requisites:** When an outbound message is created during a ProcessScheduledWork Operation e.g., createRegistrationOperation, a corresponding Recovery Point Ledger (RPL) entry must exist.

Steps to implement:

- The ProcessOutboundQueue function uses the RecoveryPointService to manage execution flow:
  - Retrieve Recovery Point
    - Use RecoveryPointService to fetch the existing ledger item for the outbound message.
- Update Status
  - If the current status is not "Running", update it to "Running" to reflect active processing.
  - Set LastUpdated to the current timestamp.
  - Set LastUpdatedBy to the current process
- Restore Context & Process Message
  - Invoke ProcessEventNotificationService:SetRestorePointData to initialize operation context from the recovery point.
  - Invoke ProcessEventNotificationService:ProcessEventNotification to handle the outbound message processing logic.
- Implement Azure Service Bus endpoint and remove Azure Storage Queue references.

#### 3.4.3.4 Process Event Notification

**Objective:** Enhance the function to reliably resume from recovery points using the RecoveryPointService, ensuring robustness during retries or failure recovery.

Steps to implement:

- Check for Existing Restore Point
  - Use the RecoveryPointService to inspect the ledger item.
  - If a restore point (RP) is defined:
    - Resume execution from the corresponding operation step.
  - If no restore point is present:
  - Begin processing from the start of the function.
- Data Retrieval
  - Retrieve the necessary processing data (e.g., CssUnprocessedMessage, event notification details) from Cosmos DB and/or Blob Storage.
  - This applies whether the function is resuming from a restore point or starting fresh.
- Fallback to Ledger Data
  - If either Cosmos DB or Blob Storage is unavailable, attempt to use the data embedded in the ledger entry (if present) to continue processing.
- Ledger Update
  - When data is successfully retrieved from Cosmos or Storage, update the ledger with this latest payload.
  - Immediately create a new Recovery Point to persist this initialisation state.
- Progressive Recovery Points
  - Continue to create recovery points at key stages throughout the remainder of the processing flow.
  - Each recovery point captures status, operation step, and relevant metadata to enable precise restart if needed.
- Implement Azure Service Bus endpoint and remove Azure Storage Queue references.

#### 3.4.3.5 Future Dated Event Handling (Queue Recovery)

When a future-dated event is created (typically during execution of ProcessScheduledWork), a corresponding Recovery Point Ledger (RPL) entry must be written to ensure the event can be recovered and replayed reliably, even in failure scenarios.

The ledger entry must include the following:

- Event Payload
  - The data required to initiate the future-dated event (e.g., a CssUnprocessedMessage).
  - While this payload is also written to Blob Storage (MessageStore), the ledger holds a duplicate copy to provide resilience in the event of storage lag or cross-region replication failure (e.g., following a BCDR event).
- Status
  - The initial ledger Status must be set to "Waiting", indicating the event is scheduled but not yet processed.
- Queue Visibility Time
  - A timestamp that defines when the event is expected to become visible for processing. This is used by the Recovery Point Orchestrator (RPO) to determine if an event has stalled.
  - The exact value depends on the nature of the event:
    - Immediate Processing:
      - Set to the message enqueue timestamp plus 5 minutes* (to allow initial processing time before retry is considered).
    - Future Scheduled Processing (e.g., Gate Closure events):
      - Set to the expected execution time plus 5 minutes*

*Configurable Function App Timeout to ensure rogue Operations are terminated.

#### 3.4.3.6 Azure Dashboards

With the refactoring of the Operations to remove the current messageStage tracking reference and replacing with the recoveryStage tracking reference, then all existing dashboards accessed by Landmark and Smart DCC where the messageStage is displayed must be updated to display the recoveryStage instead.

#### 3.4.3.7 Azure Service Bus Migration and Poison Queue Handling

Migrating from Azure Storage Queues to Azure Service Bus, the mechanism for handling failed messages (previously via a poison queue) must be refactored to align with Service Bus's built-in Dead-Letter Queue (DLQ) functionality. Unlike Storage Queues, where a separate poison queue is created and monitored programmatically with a custom process, Service Bus automatically moves messages that exceed a defined MaxDeliveryCount into a dedicated DLQ tied to each queue. This ensures failed messages are isolated for investigation.

To accommodate this change, a dedicated Azure Function will be implemented to monitor and process messages in the DLQ (e.g., ProcessDeadLetteredMessages). This function can log or archive failed messages, extract error metadata (such as DeadLetterReason and DeadLetterErrorDescription), and support reprocessing. Operational monitoring will include DLQ metrics and alerts to detect and act on failure patterns.

The migration to Azure Service Bus, if completed as part of the R0096 scope, will also expedite and simplify the implementation of R0169 End of Gate Closure Notification, as the queue can be interrogated and provide accurate message counts and metrics on message statuses that will satisfy in part the requirements for R0169.

#### 3.4.3.8 Circuit Breaker Function

Refactoring the existing circuit breaker pattern to integrate with the Recovery Point System (RPS) is required to enhance the system's ability to manage failure scenarios in a structured, state-aware, and recoverable manner. The existing circuit breaker monitors failure thresholds and temporarily blocks calls to unstable downstream dependencies to prevent cascading failures. By integrating with the Recovery Point Ledger (RPL), Recovery Point Orchestrator (RPO), and Recovery Point Service (RPS), the circuit breaker can coordinate more effectively with the broader operation recovery lifecycle.

When a circuit breaker transitions to an "open" state, operations relying on the affected dependency can use the RPS to persist their current state as a "Stalled" recovery point. This ensures no work is lost and enables operations to resume from the last successful checkpoint once the dependency recovers. The RPO, which continuously monitors the RPL for stalled operations, can respect the circuit breaker's status by deferring recovery or reprocessing attempts until the dependency is considered healthy again. This prevents unnecessary retries, reduces system load, and allows recovery efforts to be more targeted and timelier.

Telemetry from the circuit breaker (e.g., failure rates, state transitions) within be combined with RPL data (e.g., frequency and status of stalled operations) to provide improved observability into the health of the services.

Integrating the circuit breaker in the Recovery Point System permits proactive detection—such as when a spike in stalled recovery points correlates with backend instability. Once the circuit breaker resets (closes), the RPO can resume orchestrated recovery, requeuing affected operations from their recorded checkpoints using the RPS.

#### 3.4.3.9 Incident Service

The Recovery Point Orchestrator (RPO) will integrate with the existing incident management function to automate the creation of support tickets in Zendesk and ServiceNow when failures occur during Operation execution. The integration ensures that recurring issues such as operations repeatedly marked as "Stalled" or exceeding a defined retry threshold are automatically escalated for investigation and onward resolution.

When the RPO encounters a persistent failure (e.g., due to downstream service unavailability, validation failure, or circuit breaker activation), it can trigger the incident function with the relevant recovery point context. This context includes the operation ID, last completed step, error details, affected market participant or system component, and a link to the recovery point ledger entry for traceability. The incident function then raises tickets in both Zendesk and ServiceNow to ensure visibility across internal IT support systems.

### 3.4.4 Integration Patterns

| Pattern | Description | Applies To |
|---|---|---|
| Shared Service | RPS is injected into all operations | CSS operations ↔ RPS |
| Event-Driven | RPO reacts to changes in RPL | RPL ↔ RPO |
| Command Queue | RPO re-queues operations for retry | RPO ↔ Service Bus |
| Change Feed | Detects stalled operations in real time | Cosmos DB ↔ RPO |
| Idempotency | Ensures recovery points are not duplicated | RPS writes to RPL |
| Retry with Backoff | Retry logic for transient failures | Service Bus delivery policies |

**Table 12.** Integration Patterns

### 3.4.5 Security and Access Control

- **Authentication:** Managed identities for Azure resources.
- **Authorisation:** Role-based access control (RBAC) on Cosmos DB and Re-Entrant access to queues.
- **Data Protection:** All data in transit and at rest is encrypted using Azure-managed keys.

### 3.4.6 Monitoring and Observability

- **Telemetry:** Logs and metrics emitted by RPS and RPO to Azure Log Analytics
- **Dashboards:** Azure Fabric (Power BI) dashboards for recovery point status and retry rates.
- **Alerts:** Triggered on thresholds (e.g., count of an operation persistently stalling).

## 3.5 System Architecture

### 3.5.1 Overview

This section outlines how the Recovery Point System (RPSys) integrates with the underlying system components. It describes key components including persistence, error handling, logging, and caching.

### 3.5.2 Error Handling Strategy

- **Granular Task-Level Recovery:** Each operation step is checkpointed independently.
- **Refactor:**
  - Try-Catch for tasks to ensure exceptions trigger a stall state.
  - Retry logic to ensure failure of final retry triggers a stall state.
  - Addition of structured logging for refactored error handling when RPS implemented.
- **Stalled Detection:**
  - RPO identifies operations that have not progressed within a defined time window.
  - RPS identifies tasks that exhausted local retry strategies.
- **Safe Resumption:** Operations resume from the last Completed checkpoint, ensuring idempotency.
- **Fallbacks:**
  - Alerts triggered for unrecoverable failures.
  - Service Bus Triggers

### 3.5.3 Logging and Monitoring

- **Structured Logging:**
  - Operation Id, event Id, recovery point state, recovery point Id, status, and timestamps are logged.
  - Logs are centralised via Azure Log Analytics Workspace.
  - Log data presented via existing Azure Data Explorer Dashboards.*
- **Telemetry:**
  - Metrics on recovery point creation.
- **Alerting:**
  - Configurable thresholds for stalled operations.
  - Integration with Azure Monitor and PagerDuty for Operational Support.

*Additions to existing SLA dashboards to be agreed (for the purposes of estimating then no more than 5 ADX tiles should be required)

### 3.5.4 Caching Strategy

- **Purpose:** Reduce latency and Cosmos DB read costs.
- **Approach:**
  - Recover Point Service compute No ainstance in-memory Function App cache for Cosmos _etag used for optimistic concurrency control of recovery points (e.g., prevent older recovery point overwriting newer)

## 3.6 Infrastructure Architecture

The Recovery Point System for the Central Switching Service Registration capability will be built entirely on Azure-native services to ensure a scalable, resilient, and secure platform. The infrastructure will span two UK regions, UK South (preferred primary region with 3 x Availability Zones) and UK West to support geo-redundancy and disaster recovery while ensuring low-latency performance.

See Figure 6 below for design.

### 3.6.1 Azure Cosmos DB (Multi-Region Write)

Cosmos DB will be deployed as multi-region write enabled across UK South and UK West to support high availability and disaster recovery. The Recovery Point Ledger is stored in this container and leverages the change feed to detect updates in real time. The Analytical Store is enabled to support near real-time insights without impacting transactional performance. The lease container for the change feed is partitioned and readable by the Recovery Point Orchestrator instances in UK South and UK West without conflict.

### 3.6.2 Azure Synapse & SQL Pool with Power BI Integration

Azure Synapse accesses the Cosmos DB Analytical Store via a linked connection, enabling analytics and historical reporting on recovery operations. A dedicated SQL pool supports structured querying at scale for the reporting and notification requirements. Data is visualised in Power BI, to permit tracking recovery trends, latency, and success metrics via secure dashboards.

### 3.6.3 Azure Service Bus (Premium Tier)

A geo-paired Azure Service Bus Premium instance will be deployed in both UK South and UK West, with Meta-DR configured. This ensures messaging services (e.g., operation initiation, completion, and recovery triggers) are resilient to regional outages and automatically failover when necessary.

The Meta-DR as of May 2025 only supports meta data replication to the secondary site, therefore the Recovery Point Orchestrator must re-queue stalled notifications when the secondary is promoted to primary.

Full replication of meta data and message data is in public preview (Geo-DR) and will be available ahead of current proposed schedules. However, VNET integration on the product roadmap does not have a delivery date.

[Link: https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-geo-replication ]

### 3.6.4 Azure Networking and Resource Group

A dedicated Azure Resource Group hosts all Recovery Point components. All compute and data services are deployed within an Azure Virtual Network (VNET) segmented into subnets with tightly scoped Network Security Groups (NSGs). Service Endpoints are used for secure access to Azure services within the VNET, while Private Endpoints enable secure, region-spanning access between UK South and UK West without traversing the public internet.

### 3.6.5 Azure Function App – Recovery Point Orchestrator

The Recovery Point Orchestrator is an Azure Function App hosted in a Web App Plan EP1 tier with regional failover support. It is triggered by the Cosmos DB change feed and a time-triggered scheduler to check for stalled operations (based on item timestamps). The orchestrator also integrates with:

- Incident Management Service for issue tracking and escalation,
- Azure Monitor for operational metrics and logs,
- Log Analytics for long-term retention and correlation of telemetry,
- Azure Alerts for proactive notifications of processing delays or recovery failures.

### 3.6.6 Scalability and High Availability

All services are deployed using PaaS components that auto-scale based on demand. Cosmos DB and Service Bus scale throughput automatically; the Azure Functions hosting the orchestrator scale out on execution volume. Multi-region deployment ensures 99.99% availability and meets contracted RTO/RPO under disaster recovery scenarios.

### 3.6.7 Security and Compliance

The infrastructure uses Azure Private Link, role-based access control (RBAC), Azure Key Vault for secrets management, and encryption at rest and in transit. All resources are monitored using Azure Defender and configured to comply with contract data privacy and compliance requirements.

### 3.6.8 Integration with Existing Systems

The Recovery Point System integrates seamlessly with the existing Central Switching Service via well-defined endpoints, messaging interfaces on Service Bus, and telemetry pipelines connected to existing support monitoring tools. The system is designed to extend rather than replace the current operational model, ensuring minimal disruption and full traceability across platforms.

### 3.6.9 High Availability Infrastructure Design

For the purpose of illustration ONLY the Registration Service Resources that the Recovery Point System integrates with are shown.

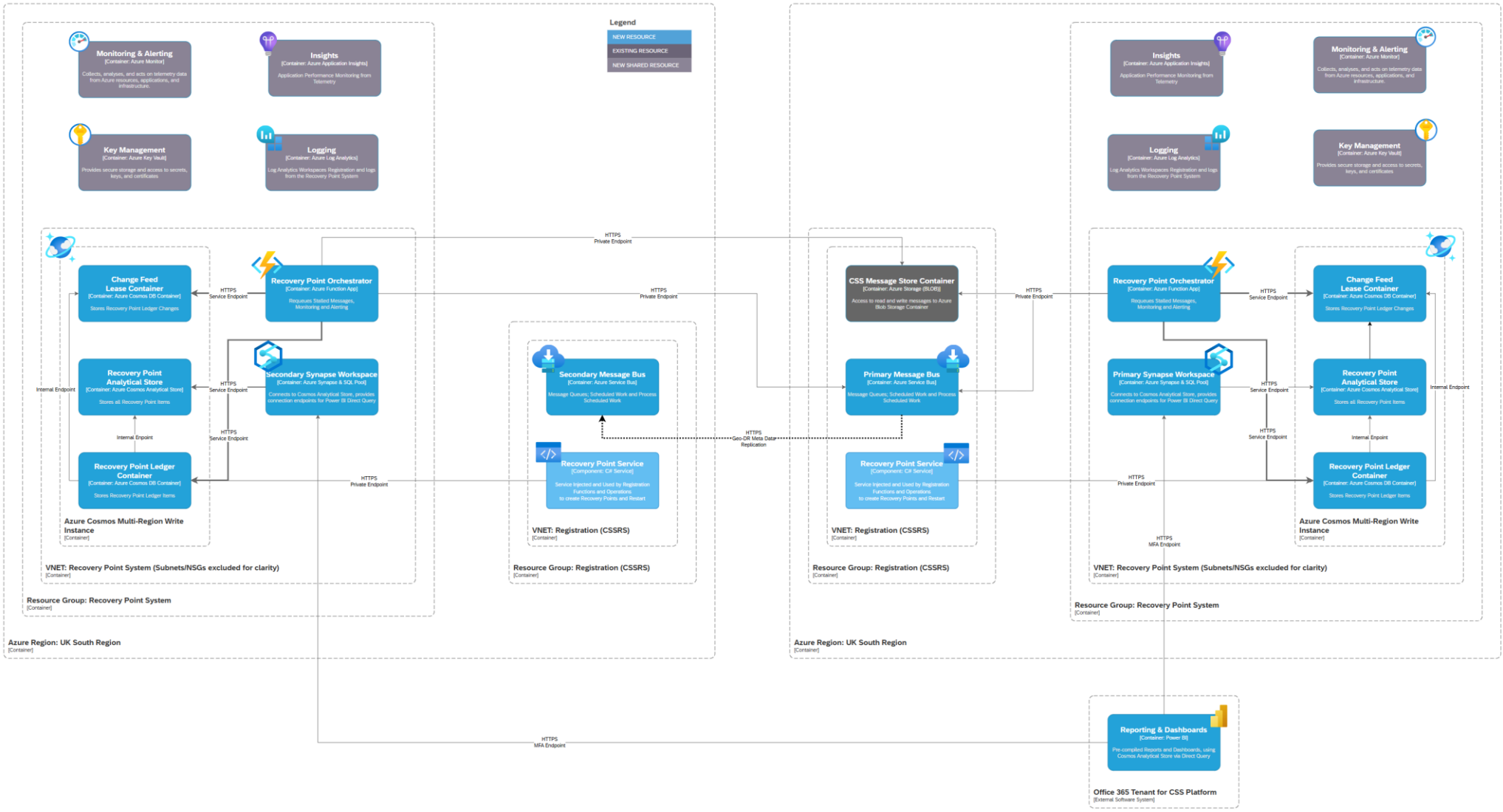



**Fig. 6** Target Infrastructure Design for Recovery Point System.

## 3.7 Security & Compliance Architecture

### 3.7.1 Overview

The Recovery Point System (RPSys) is designed with security and compliance as foundational principles. It ensures the confidentiality, integrity, and availability of recovery data across all CSS operations. This section outlines the security architecture, access controls, encryption protocols, compliance alignment, and operational safeguards that govern the system.

### 3.7.2 Data Protection

- **Encryption at Rest:** All recovery point data stored in the Recovery Point Ledger (Cosmos DB) is encrypted using Azure-managed keys (AES-256).
- **Encryption in Transit:** All communication between CSS operations, the Recovery Point Service (RPS), and the Recovery Point Ledger (RPL) is secured via TLS 1.2+ and Azure Private Endpoints
- **Payload Redaction:** Sensitive fields in recovery point payloads will not be redacted or encrypted before persistence to expedite performance, so access to personally identifiable information (PII) or commercially sensitive data will be restricted in accordance with existing procedures.

### 3.7.3 Access Control and Identity Management

- **Role-Based Access Control (RBAC):**
  - Only the Recovery Point Service and Orchestrator have write access to the Recovery Point Ledger.
  - Read access is restricted to authorised services for the Recovery Point Service and monitoring purposes.
- **Managed Identities:**
  - Azure Managed Identities are used for secure, credential-free authentication between services.
  - The existing Azure Key Vault instance(s) will continue to provide secure storage of credentials and configuration parameters.
- **Least Privilege Principle:**
  - Each component is granted only the minimum permissions required to perform its function.

### 3.7.4 Compliance Standards

The CSS service is governed by contract schedule 2.7 (Standards), for the purposes of the Recovery Point System (RPSys) the following is stated.

- **GDPR:**
  - Supports data minimisation and purpose limitation by storing only essential recovery data.
  - Enables data subject rights through traceable and deletable recovery point records.
- **ISO/IEC 27001:**
  - Aligns with controls for access management, data protection, and incident response.
- **Audit Logging:**
  - All recovery point operations are logged with timestamps, operation Ids, and event Ids.

### 3.7.5 Risk Management and Threat Mitigation

- **Threat Modeling:**
  - Assessment of the Recovery Point System (RPSys) will be governed within existing operational processes.
- **Data Integrity Checks:**
  - Recovery points will not include any specific mechanism to detect tampering or corruption.
- **Redundancy:**
  - Refer to BCDR Section in this document.

### 3.7.6 Incident Response and Monitoring

- **Stalled Operation Detection:**
  - The Recovery Point Orchestrator (RPO) continuously monitors for stalled operations and triggers alerts on the Recovery Point Service marking a Recovery Point as stalled.
- **Alerting and Escalation:**
  - Integrated with Azure Monitor and PagerDuty for real-time incident response, with incidents raised via the Incident Service to Zendesk and Service Now.
- **Audit Trails:**
  - Immutable logs are maintained for all recovery-related actions to support forensic analysis and compliance audits.

### 3.7.7 Continuous Compliance and Governance

- **Policy Enforcement:**
  - Azure Policy and Defender for Cloud are used to enforce security baselines and detect misconfigurations.
- **Penetration Testing:**
  - Annual security testing is conducted to validate the robustness of the system.
  - Pre-release penetration test audit the Recovery Point System (RPSys) will be executed with critical and high recommendations implemented prior to Production deployment.
- **Change Management:**
  - All changes to RPSys components will be subject to formal change control and security review processes.

# 4 Implementation

This section outlines the considerations for implementing the solution, including system configurations, deployment procedures, integration points, and operational guidelines to ensure smooth and effective execution.

## 4.1 Development

This section details the development of the Recovery Point System following a structured, phased approach to ensure controlled delivery, maintainable design, and seamless integration with the existing Central Switching Service Registration capability. Agile methodology will be used, with work delivered and validated through phases, automated testing, and feedback.

### 4.1.1 Development Phases

**Phase 1** – RecoveryPointService Integration

- Refactor a representative operation (CreateRegistrationOperation) to implement the RecoveryPointService interface.
- Implement SaveAsync() to persist progress and status to the Recovery Point Ledger (Cosmos DB).
- Refactor ProcessScheduledWork to retrieve and resume processing from the last saved point.
- Validate the integration pattern and document reusable design guidance.

**Phase 2** – Performance and Load Testing

- Use K6 to simulate concurrent operations and assess write throughput, Cosmos DB RU consumption, and Service Bus reliability.
- Validate ledger scalability under realistic peak load scenarios.
- Refine retry logic, concurrency handling, and operation timeouts based on insights.

**Phase 3** – RecoveryPointService Integration

- Refactor all Operations and Functions to use RecoveryPointService*
- Repeat use of K6 to simulate concurrent operations and assess write throughput, Cosmos DB RU consumption, and Service Bus reliability.

*In the event that Phase 2 highlights that the In-bound synchronous and asynchronous response time are within 25% of the upper contracted service levels for average and peak volume response times then the RPS SaveAsync() _await pattern will be refactored to the ‘fire-forget with _etag check’ pattern.

**Phase 4** – Recovery Point Orchestrator

- Develop the orchestrator as an Azure Function App triggered by:
  - Cosmos DB Change Feed for real-time detection of stalled operations.
  - Timer Trigger to identify overdue items based on last modified timestamp.
- Integrate with existing Incident Management System for escalation and alerting.
  - Create notification templates and content injection

**Phase 5** – Reporting and Dashboards

- Enable analytical access to the Recovery Point Ledger via Azure Synapse.
- Develop Power BI reports and dashboards for operational metrics, recovery trends, and exception monitoring.
- Integrate dashboard access with existing role-based security models.

### 4.1.2 Technology Stack

- Languages: .NET (C#), JavaScript (K6)
- Services: Azure Cosmos DB, Azure Functions, Azure Service Bus, Azure Synapse, Power BI
- CI/CD: Azure DevOps Pipelines with Infrastructure-as-Code (ARM)
- Testing: Azure Chaos Studio
- Monitoring: Azure Monitor, Log Analytics, Application Insights

### 4.1.3 Integration Points

- Cosmos DB for state persistence.
- Azure Service Bus for operational messaging.
- Incident and alerting tools for support integration.
- Existing Central Switching Endpoints and workflows.

### 4.1.4 Testing Strategy Activities

- Unit Testing: Core logic, ledger integration, recovery logic.
- Integration Testing: End-to-end flow validation in test environments.
- Performance Testing: High-volume scenarios using K6.
- Azure Chaos Studio

### 4.1.5 Deployment Procedures

- Infrastructure deployed via IaC templates to both UK South and UK West.
- Code deployed via CI/CD pipelines with approval gates.
- Blue-green strategies considered for Function App rollouts.
- Validation checklists before production promotion.

### 4.1.6 Roles and Responsibilities

- Solution Architect: Oversee design, ensure alignment with CSS solution architecture.
- Developers: Implement features, write unit and integration tests, and contribute to documentation.
- QA Engineers: Own test strategy, run K6 performance tests, coordinate releases.
- DevOps Engineers: Manage pipelines, IaC, and deployment consistency.
- SRE/Support/Operations: Review documentation and participate in training on new recovery features.

### 4.1.7 PR8003 PoC Slices Evaluated

| PoC Type | Description |
|---|---|
| Logical | Ledger Schema |
| Logical | Cosmos DB as the ledger technology with multi-write in dual region |
| Logical | Refactor Storage Queues to Service Bus |
| Logical | Future Dated Event (Queue Recovery) |
| Develop/Demo | Recovery Point Service (Save) |
| Develop/Demo | Recovery Point Orchestrator |
| Develop/Demo | Recovery Point Service (Resume) |
| Develop/Demo | Refactor Error / Exception Handling (eg. Mark an operation as stalled) |
| Logical | Evaluate Refactor Blob Storage to use Cosmos |
| Develop/Demo | Dashboards (Power BI to Cosmos Analytical Store) |
| Logical | Monitoring and Alerting |

**Table 13.** PoC Slices Evaluated to Inform Detailed Design

## 4.2 Testing

The introduction of the Recovery Point System to the Central Switching Service Registration capability introduces a new architectural pattern and failure-handling behaviours that requires an extension to the existing test strategy that is targeted and coordinated across disciplines and SI governed environments.

Testing will need to validate both functional correctness and non-functional resilience, particularly in scenarios involving stalled operations and automatic recovery. The following areas represent the key pillars of the test strategy concerns further assessment in the detailed impact analysis phase.

### 4.2.1 Environment Consideration

Simulating stall conditions and degraded processing states, necessary to validate recovery behaviours has the potential to introduce instability into shared environments. As such, these tests must not be executed within release environments (e.g., PIT-A, SIT-A and above) where multiple teams operate concurrently.

To mitigate this risk, all testing associated with Recovery Point System scenarios (e.g., R0096 and related recovery patterns) should be executed within a dedicated test dual region environments. This will require cross-team collaboration between Development QA, DevOps, and SRE teams to provision and maintain the environment. Isolating these tests ensures system stability elsewhere while enabling controlled, repeatable validation of complex failure scenarios.

### 4.2.2 Execution Strategy for Automated Tests

Recovery scenarios will require longer execution durations and introduce processing complexity that is potentially unsuited to the existing regression pipelines. Therefore, R0096 specific automated tests will be:

- Tagged with a unique identifier (e.g., @RecoveryPointTests).
- Executed independently in an ad hoc CI/CD stage.
- Designed to run in parallel, where possible, to reduce end-to-end test time.

A recurring scheduling mechanism (e.g., via pipeline triggers or Jira automation) will ensure that these tests are executed periodically as part of ongoing quality assurance cycles, maintaining continuous validation without degrading routine build or deployment efficiency.

### 4.2.3 Privilege Requirements for Automated Testing

Certain test cases, such as those simulating system stalls via network restrictions or storage access issues will require elevated permissions. The automated test execution accounts must be able to:

- Modify Azure Blob Storage container content.
- Disable network rules or simulate outage scenarios.
- Manipulate Cosmos DB resource states and content.

To manage this securely, these elevated permissions will be scoped to the dedicated test environment using Azure Privileged Identity Management (PIM). This ensures privileges are tightly controlled, auditable, and granted only when needed, minimising risk to other environments and production workloads.

### 4.2.4 Testing Strategy and Tool Alignment

The Recovery Point System introduces new integration-level testing needs that span multiple components and resources, including message-based communication, data persistence, and recovery orchestration. These requirements overlap with existing end-to-end (E2E) testing frameworks and processes already in place.

To maintain consistency and reduce redundancy, a unified test automation strategy will be considered, integrating Recovery Point scenarios into a common test harness alongside existing E2E and non-functional test suites. This approach promotes:

- Cross-team alignment on tools, data, and reporting.
- Consolidated test coverage across functional and recovery logic.
- Efficient use of infrastructure and automation resources.

### 4.2.5 External System Integrator (SI) Verification Testing

The external System Integrators (SIs) will be unable to simulate stalls directly given the elevated privilege required to conduct non-functional stalls, test execution must consider a coordinated approach using detailed runbooks that define the types of stall conditions to simulate (e.g., non-functional timeouts, functional logic stalls) R0178 OFAF issue of multiple MPxN's is an example of condition the SI could simulate.

The runbooks must also consider how stall simulation coordination will be executed during a production release to ensure the Recovery Point System is operating as expected.

### 4.2.6 Explore Synthetic Test Data in Production Environment

As part of the Recovery Point System test strategy, the use of synthetic test data in the production environment could be explored to validate observability, stall recovery, and end-to-end orchestration in real-world conditions without impacting live customer operations.

Production is the only environment where service telemetry, network conditions, load characteristics, and regional configurations are fully representative. Introducing controlled synthetic operation designed to simulate common failure scenarios (e.g., stalled states, retries, dead-letter recovery) can provide assurance that the Recovery Point Orchestrator behaves correctly under live operational conditions.

Synthetic data must be clearly identifiable, non-disruptive, and isolated from customer-facing logic. Test records should be flagged using metadata and excluded from reporting pipelines and analytics dashboards. Additionally, synthetic tasks should trigger notifications and logs identical to real tasks, allowing validation of monitoring, alerting, and incident workflows.

In summary, introducing synthetic data into production under strict control may provide critical assurance that recovery mechanisms function as expected in the environments they are meant to protect.

Introducing synthetic domain and registration data may also assist with future migrations and external participant systems.

### 4.2.7 Consideration for adopting an integrated cross-team testing framework

To support R0096 maintainability and cross-team activities, adoption of a shared integration testing framework is recommended. Successful testing of R0096 relies on disrupting the underlying non-functional resources to verify the solution. Therefore, It's important to assess how a shared testing framework will benefit long term maintenance and assist the development and QA roles. The assessment should include evaluating the learning curve and whether the framework fits naturally into existing workflows. Shared ownership must also be clearly defined—establishing who is responsible for writing and maintaining different types of tests, such as integration versus end-to-end (E2E), to avoid duplication and ensure accountability.

In addition, the framework must integrate seamlessly with the existing tools like Jira, Git, Azure DevOps, CI/CD pipelines, and reporting systems. Standardising how R0096 test scripts and data are structured, stored, and version-controlled is key to ensuring consistency and reducing rework.

### 4.2.8 PIT Environment Load and Performance Testing

Introducing K6 Load & Performance testing during the Pre-Integration Testing (PIT) phase enables early validation of the Recovery Point System's performance and scalability under realistic and peak-load conditions.

- Key benefits of early load testing include:
  - Proactive performance validation before shared environments or full-scale integration
  - Identification of bottlenecks in Cosmos DB RU/s consumption, Service Bus throughput, and Function cold start times
  - Validation of retry and stall handling logic under sustained pressure
  - Reduction in performance-related 'surprises' during later stages (e.g., SIT or UAT)
  - Ability to right-size resource provisioning (e.g., Cosmos throughput, Premium Function plans) based on data

### 4.2.9 Use of Azure Chaos Studio for Testing

Azure Chaos Studio will be explored and considered to determine if it can play a role in validating the resilience and recovery capabilities of the Recovery Point System by introducing controlled fault injections into the system's dependent components and infrastructure. This would be non-functional, to proactively identify how the system responds under failure conditions, ensuring that the recovery orchestration, retry logic, and fault tolerance mechanisms behave as designed.

- Scenarios
  - Injecting failures into resource dependencies e.g., Cosmos DB, Service Bus, Azure Functions, and Blob Storage.
  - Resource throttling, network latency, service outages, and regional failures
    - Cosmos DB availability drops or RU/s throttling
  - Service Bus delivery delays or DLQ pressure
  - Azure Function execution interruptions or latency
  - Storage access failures or permission misconfigurations
  - Network partitioning scenarios across UK South and UK West

- Environments
  - PIT-x Dual Region, SIT-x Dual Region and Pre-Production

- Third Party Access
  - Delegate third party System Integrator access using the Azure Chaos Experimenter Contributor Role with resource group restrictions.

### 4.2.10 BCDR Testing

Not withstanding the use of Azure Chaos to instigate a DR scenario, a BCDR test must be conducted in each Dual Region environment (listed in 3.9.9) at least once in each and repeated in Pre-Production when the final release candidate is agreed. See section 3.15 for BCDR details.

## 4.3 DevOps

The Recovery Point System, introduces several responsibilities for the DevOps function across the deployment, monitoring, and maintenance lifecycle.

### 4.3.1 Pipeline Configuration and Deployment Automation

- Implement infrastructure-as-code (IaC) templates to deploy:
  - Cosmos DB with Analytical Store and multi-region configuration
  - Azure Functions for Recovery Point Orchestrator
  - Service Bus namespaces and queues with DLQ support
  - Integrate Recovery Point Services into the existing CI/CD pipeline, with gating to ensure health checks and telemetry readiness before release progression.

### 4.3.2 Monitoring and Alerting

- Configure Azure Monitor, Log Analytics, and Application Insights to track:
  - Latency and throughput of SaveAsync() and GetLastSuccessfulStepAsync()
  - Function App execution time, success/failure rates, and retries
  - Service Bus queue depth, processing delays, and DLQ entries
  - Set up alerts for Cosmos DB write throttling, failed recovery point writes, or excessive stall detections.

### 4.3.3 Observability and Diagnostics

- Ensure logging includes:
  - Correlation IDs across operation, orchestrator, and functions
  - Maintain dashboards to provide operational transparency to internal and external support teams.

### 4.3.4 Resilience and Failover Readiness

- Validate regional failover procedures and test recovery workflows when one region (UK South or UK West) is offline or degraded.
- Coordinate release failover exercises and simulate function or queue failures to test retry logic and queue processing integrity.

### 4.3.5 Configuration and Feature Flag Management

- Use feature flags to toggle between sync and async recovery point write modes without redeployment.
- Manage application settings for:
  - Cosmos DB
  - Service Bus connection strings and retry policies
  - Scheduled Recovery Point Orchestrator intervals

### 4.3.6 Security and Compliance

- Ensure managed identities and RBAC roles are correctly assigned and scoped to least privilege.
- Monitor use of elevated permissions (via PIM) for deployment and runtime access.

### 4.3.7 Incident Readiness

- Develop and maintain runbooks for:
  - Recovery Point Orchestrator failures
  - Service Bus backlog or DLQ triage
  - Cosmos DB unavailability or write conflicts
  - Support root cause analysis with access to full operational telemetry and trace data.

## 4.4 Deployment Considerations

The deployment of the Recovery Point System for the Central Switching Service Registration capability follows a structured and controlled process to ensure stability, traceability, and minimal risk to live operations. The deployment strategy is aligned with the existing dual region release approach, with UK West updated first and UK South remaining as the live traffic endpoint during initial rollout.

### 4.4.1 Pre-Requisites

- Confirm infrastructure provisioning via Infrastructure-as-Code (ARM) is available for:
  - Cosmos DB with Analytical Store enabled
  - Azure Function Apps (Recovery Point Orchestrator)
  - Service Bus, monitoring resources, and networking
- Ensure configuration of:
  - Environment-specific settings (e.g., Cosmos DB connection strings, function timeouts)
  - Feature flags for Recovery Point System activation
- Validate permissions are in place for deployment automation, monitoring, and runtime access
- CI/CD pipelines are fully integrated and reviewed

### 4.4.2 Deployment Process

- Deploy to UK West (including deployment of all UK South Recovery Point System non-functional resources)
- Deploy Recovery Point Service components to UK West
- Perform smoke tests and execute recovery using synthetic data*
- Pre-Switchover Verification
- Run integration and functional tests to validate Recovery Point orchestration and ledger updates
- Validate telemetry, alerting, and logging integrations via Application Insights and Log Analytics
- Simulate stalled messages to confirm orchestrator reactivation and recovery flow
- Switch percentage of traffic to UK West (via Azure Traffic Manager)
- Monitor Recovery Point System under real-time operational load
- If stable, proceed to deploy updated components to UK South

*Must be approved by RECCo if used in Production

### 4.4.3 Rollback Strategy

- Use of Feature flags to disable Recovery Point processing without service interruption
- Traffic can be redirected back to UK South immediately if issues are detected in initial UK West

### 4.4.4 Post-Deployment Activities

- Monitor system health metrics and orchestrator activity in both regions
- Confirm recovery events and ledger updates are functioning correctly
- Conduct a release review and update operational runbooks and support documentation
- Schedule follow-up validation of reporting dashboards and analytical access in Power BI

## 4.5 Data Usage & Licensing (COTS, Libraries and Packages)

This solution leverages several Azure Platform-as-a-Service (PaaS) components to support the Central Switching Service Registration and Recovery Point System capabilities.

- Azure Cosmos DB is used to store operational metadata and recovery state in the Recovery Point Ledger. This service operates under a provisioned throughput model and is configured with multi-region writes for high availability and resilience.
- Azure Service Bus Premium with Geo-Disaster Recovery (Geo-DR) is used for decoupled, durable messaging across distributed operations, supporting high-throughput, low-latency communication and failover across paired regions.
- Azure Synapse Analytics with a dedicated SQL Pool is used for structured data transformation and querying at scale, supporting operational insight and historical analysis.
- For reporting, Power BI is used to enable secure, role-based access to visual analytics, providing transparency into operational metrics, success rates, and recovery trends.

All Azure services used are subject to Microsoft's standard licensing models. Consumption will be monitored and governed using Azure Cost Management to ensure scalability within financial constraints.

## 4.6 Operational Management

This section outlines the impact of the Recovery Point System on ITIL based support operations with system introducing a structured mechanism for managing operation continuity and recovery within the Central Switching Service Registration process, aligned with ITIL best practices.

From a Service Operation perspective, each Market Participant Operation will implement the Recovery Point Service interface, enabling real-time progress saving and retrieval of the last known good state. This will support Incident Management by allowing failed or stalled operations to be recovered automatically without escalation to second-line support, thereby reducing Mean Time to Restore Service (MTRS) for specific incidents that may have required user intervention.

- **Problem Management:** The Recovery Point Ledger, persisted in Azure Cosmos DB, acts as an audit trail of Operations activity, capturing metadata about execution steps and failure points. This will enable the support teams to identify recurring patterns, isolate root causes, and inform long-term corrective action plans.
- **Change Management:** support teams should be aware that Operational logic has shifted toward state-driven orchestration. Azure Functions like the Recovery Point Orchestrator monitor and act on recovery point data using the Cosmos DB Change Feed and scheduled queries for aged or stalled items. This system autonomously restarts operations or flags persisting issues for intervention via support dashboards or alerts.
- **Monitoring and Event Management:** Delivered through Azure Application Insights and Log Analytics, with integration into existing support alerting platforms Zendesk and Service Now. Telemetry includes operation status, failure reasons, and recovery actions, with correlation IDs to support end-to-end traceability. Support teams may need to act on alerts related to retry loops, persistent failures, or ledger inconsistencies.
- **Service Continuity Management:** Improved by enabling mid-operation recovery, reducing the impact of transient infrastructure or dependency failures. Support staff are not required to manually restart operations but may need to validate system recovery in edge cases or support exception handling where automatic recovery is not possible.

## 4.7 Service Levels

This section defines the proposed target service level objectives for the Recovery Point System, outlining the key performance indicators (KPIs), monitoring strategies, and compliance procedures required to meet contracted service levels for synchronous and asynchronous activities. These objectives are designed to ensure consistent reliability, timely recovery of stalled operations, and visibility into service health.

### 4.7.1 Objectives

- **Availability:**
  - The Recovery Point System will maintain a minimum availability across all core components in-line with contracted requirements. (Cosmos DB 99.999%, Azure Functions 99.95%, Service Bus 99.99% UK South, 99.9% UK West).
  - Region failover strategies (UK South and UK West) ensure continuity during maintenance or service interruptions.
- **Recovery Time Objective (RTO):**
  - A stalled operation should be detected and resumed (automatically or with support intervention) **within 5 minutes of stalling.***
- **Recovery Point Objective (RPO):**
  - The system should not lose more than one operation's worth of progress under normal conditions, as intermediate states persisted using SaveAsync().
- **Azure Cosmos Change Feed:**
  - Azure Cosmos Change Feed Latency 1-5 seconds
- **Function Execution Performance:**
  - SaveAsync() method must complete within 20ms under normal operating conditions. (Microsoft SLA for 99$^{th}$ percentile is <15ms within preferred region)
  - GetLastSuccessfulStepAsync() method should complete within 200ms under normal operating conditions. (Microsoft SLA for 99$^{th}$ percentile is <10ms within preferred region)
  - The Recovery Point Orchestrator must initiate a recovery action within 5 seconds of receiving a stall condition trigger via the Cosmos DB Change Feed. This interval is measured from the time the stall status is recorded in the Recovery Point Ledger to the time the orchestrator begins processing the corresponding recovery logic.

***This is configurable and is mandatory value for the Function App, as only the Function App can terminate rogue Operations, there is no access to App Plan Compute Threads and therefore this value acts as the catch all timeout condition to mark items as stalled where the LastUpdatedTime exceeds this value.**

### 4.7.2 Monitoring and Alerting

- Azure Monitor and Log Analytics will track key metrics such as:
  - Function execution success/failure
  - Number of operations in "Stalled" status
  - DLQ message volume and retry count
- Alerts will be configured for:
  - Breaches of recovery latency thresholds
  - Failure spikes
  - Missing or overdue orchestrator runs
- Alert Priority and Content Format
  - Unless specified otherwise P2 for the count of persistent stalls exceeding threshold and / or the service degraded / unhealthy
  - P3 for all other alerts.

### 4.7.3 Reporting and Compliance

- Weekly service health summaries and SLA compliance reports will be generated using:
  - Power BI dashboards connected to Synapse queries
  - Log Analytics reports for error and alert history
- Reports will be reviewed by the service owner and made available for assurance and trend analysis.

### 4.7.4 Compliance and Audit Readiness

- All SLA-affecting events (e.g., failures, escalations, manual interventions) will be logged with correlation IDs for traceability.
- Retention policies will ensure all logs, recovery attempts, and orchestration actions are available in accordance with contracted time periods for retention or at least the maximum supported retention period of the Azure Resource, whichever is less.

### 4.7.5 Initial Service Level Impact Assessment

Figure 7 and 8, illustrate the spread of total transaction times for a Market Participant requests over a 24 hour period and 88K individual inbound request Operations. The purpose of the illustrations is to provide an indication that under normal operating conditions the Recovery Point System pattern will not exceed the safety threshold within the current service level targets for synchronous and asynchronous operations.

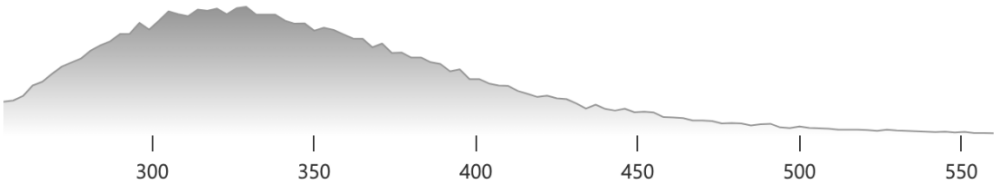



**Fig. 7** Synchronous Profile

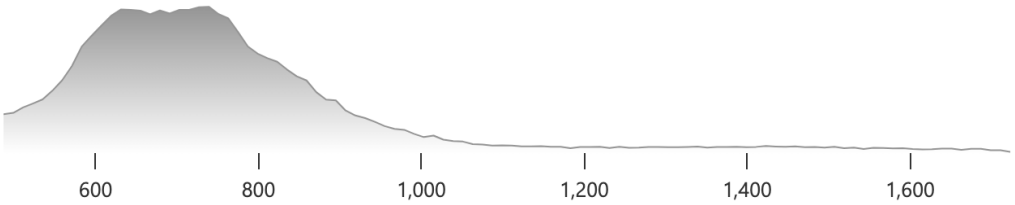



**Fig. 8** Asynchronous Profile

## 4.8 Business Continuity and Disaster Recovery

To meet the requirements outlined in Schedule D of Contract Schedule 8.6, the solution incorporates robust Business Continuity and Disaster Recovery (BCDR) strategies to ensure minimal disruption and data loss in the event of system failures, operational incidents, or wider service outages. The Recovery Point System enhances continuity by enabling operations, such as registrations and event notifications, to be safely paused and resumed without starting over. Save points are recorded during processing, allowing stalled tasks to be automatically recovered and resumed, ensuring service integrity and reducing manual intervention.

The solution uses highly resilient Azure Platform Services, including geo-redundant Azure Cosmos DB for recovery point storage and Azure Service Bus Premium with geo-disaster recovery enabled for messaging, where missing notifications can be re-inserted by the Recovery Point Orchestrator. This ensures critical data and communications are replicated across regions, supporting failover and service availability even in the event of a regional outage. Operations are monitored continuously, and the dedicated Recovery Point Orchestrator function automatically detects and responds to failures, triggering retries, alerts or notifications where necessary.

Data will be backed up continuously and stored in compliance with existing contracted retention policies. All services employ encryption and strict access controls, supporting confidentiality and integrity of data. A design risk assessment has been performed to identify critical components and single points of failure, with mitigations implemented accordingly. In the event of a disaster, the existing predefined communication protocols and escalation paths will be used.

BCDR Runbooks will be updated to include the specific steps required for the Recovery Point System components to continue operating in the event of a region failure or in response to an annual BCDR exercise.

### 4.8.1 Recover Point System BCDR Actions

- Failover of Azure Service Bus Primary to Secondary
- DNS update for Power BI Dashboards and Reports
- Recreation of Azure Service Bus Primary when region available for failback

### 4.8.2 Recovery Point System Resources Fail

If there is a failure of the Recovery Point System where the Recovery Point Service is unable to save or resume recovery the service will retry only once per recovery point and then continue to the next recovery point. Failure of the Recovery Point System in whole or in part will not impact Registrations.

### 4.8.3 Cold Start Behaviour

On a cold start—such as after a system failure, deployment, or intentional shutdown—the Recovery Point System is designed to resume operation safely and predictably, regardless of whether there are stalled operations present.

If there are stalled operations in the ledger, the orchestrator will detect them on startup in two ways:

- Cosmos DB Change Feed Trigger:
    - If the cold start follows a crash or deployment where new stalled items were written just prior to shutdown, the orchestrator will resume consuming the change feed and act on any items flagged for recovery.
- Scheduled Catch-All Query:
    - On cold start, the scheduled function executes a query for stalled items based on

timestamp thresholds. This ensures that items missed during downtime or without change feed activity are still recovered.

- Graceful Shutdown Handling:
  - If the service is shut down cleanly (e.g., via deployment or scaling event), it does not impact state integrity. The Cosmos DB ledger acts as the source of truth. Because recovery point writes are idempotent and version-controlled using _etag, partially completed operations will not be lost or duplicated on restart.

### 4.8.4 Participant Behaviour During BCDR Event

When a system restart occurs due to failure, deployment, or a Business Continuity and Disaster Recovery (BCDR) event, it's essential to consider how and when the Recovery Point Orchestrator (RPO) resumes recovery. There is no guarantee that market participants won't re-send requests, assuming their previous request failed if no response was received within expected timeframes. Therefore, startup of the Recovery Point Orchestrator needs consideration.

#### 4.8.4.1 Duplicate Request Risk

If a participant does not receive a confirmation (due to an incomplete operation or outage), they may re-send the same request, currently this could be rejected by business rules where an RMP may have been updated but a registration has not been linked.

- Proposed Business Rule
  - Implement a business rule, response code and description where:
    - If a new request is received, the system checks the Recovery Point Ledger to determine if stalled entry exists for the same business entity (e.g., MPxN).
    - If such an entry exists, the new request is either rejected or held, with a clear message stating it is pending recovery.

#### 4.8.4.2 Delay Participant Request Processing on Restart

- On system recovery post BCDR event restart, reject market participants until the Recovery Point Orchestrator has processed stalled Operations.

## 4.9 Change Governance

The introduction of the Recovery Point System does not alter the existing business or operational processes defined in Contract Schedule 8.2.

All changes to the solution architecture, including enhancements to the Central Switching Service Registration capability, follow established change management policies to ensure consistency, traceability, and minimal risk.

Any related change requests will be submitted through the formal governance process and undergo thorough impact assessment, including operational, service, and risk considerations. Approval workflows remain unchanged, with clear roles and responsibilities assigned for assessment, approval, and implementation oversight. All changes will be fully documented, including rationale, testing evidence, and rollback plans where applicable. Communication of approved changes is coordinated with support and operational teams to ensure readiness and understanding.

## 4.10 Costs

Whilst the purpose of the DIA is to provide the detailed Azure Cost impact, an indication of the considerations that impact costs are provided to give some perspective to the Azure costs of the Recovery Point System in order to equate this to the downstream costs to industry in the event of data loss.

The Recovery Point System relies on several Azure services that incur consumption or capacity-based charges. While the design prioritises resilience and scalability, it also includes mechanisms to manage and monitor costs. This section outlines the primary cost contributors and considerations.

### 4.10.1 Azure Cosmos DB (Multi-Region Write + Analytical Store)

- **Cost Drivers:**
  - Provisioned throughput (RU/s) for transactional operations
    - Typical RPSSys payloads, 5-15 RU's on Recovery Points , 1 RU per point read, 3-10 RU's on queries by the Recovery Point Orchestrator.
    - Largest Recovery Point Service write (CreateRegiustration) 22K approx. 5-50 RU's.
  - Storage consumption for both transactional and analytical stores
  - Multi-region replication (UK South and UK West)
- **Mitigation Strategies:**
  - Auto-scale RU/s to match operational demand and avoid overprovisioning.
  - Apply custom TTL (time-to-live) policies to manage data retention.
  - Analytical Store used for read-heavy analytics

### 4.10.2 Azure Service Bus (Premium Tier)

- **Cost Drivers:**
  - Number of messaging units (MUs) provisioned.
  - Operations (send, receive, dead-letter)
  - Geo-DR configuration (paired namespace in secondary region)
- **Mitigation Strategies:**
  - Optimise message size and frequency.
  - Monitor usage and scale MUs based on peak vs average load

### 4.10.3 Azure Functions (Premium Plan)

- **Cost Drivers:**
  - Execution time (GB-s) and number of executions
  - Always-ready instances for low-latency processing
- **Mitigation Strategies:**
  - Use Premium Plan auto-scaling and minimum instance configuration

### 4.10.4 Monitoring and Logging

- **Cost Drivers:**
  - Log Analytics ingestion and retention
  - Application Insights telemetry volume
- **Mitigation Strategies:**
  - Configure lossy logging for telemetry
  - Set retention policies for logs aligned to operational and compliance needs

### 4.10.5 Power BI and Synapse Analytics

- **Cost Drivers:**
  - Power BI licensing (Premium per user/capacity & Embed to view only users)
  - Synapse Serverless SQL pool queries against the Analytical Store
- **Mitigation Strategies:**
  - Optimise SQL queries and views to reduce data scan costs
  - Use scheduled refresh intervals aligned with operational needs

### 4.10.6 Cost Governance Recommendations

- Tag all Recovery Point System resources for cost tracking
- Set up budgets and alerts in Azure Cost Management
- Periodically review usage and performance metrics to identify savings opportunities
- Involve DevOps and Finance in reviewing forecasts vs actuals

# 5 Non-Functional Requirements

For the purposes of this design document the following definition of normal operating conditions applies:

Normal operating conditions refer to the expected baseline state in which the system functions as designed, without degradation or external disruptions. Under these conditions, all integrated services, components, and dependencies are available, responsive, and operating within their defined thresholds.

This includes:

- All Azure services (e.g., Cosmos DB, Azure Functions, Service Bus, Blob Storage) are online and meeting their service-level agreements (SLAs).
- No artificial or simulated faults are introduced (e.g., no forced stalls, timeouts, or connectivity issues).
- Network latency is within standard ranges for intra-region and inter-region communication.
- The system is processing data and operations at typical volumes and concurrency levels defined by average or peak business-as-usual demand.
- No maintenance or failover events are in progress, and traffic is routed to the primary region (e.g., UK South).
- All monitoring, alerting, and telemetry systems are functional and receiving accurate data.
- Users and automated processes have appropriate access, and no permissions or security anomalies are occurring.

Normal operating conditions provide the baseline against which exception handling, recovery scenarios, and non-functional performance metrics are measured.

## 5.1 Performance

- The Recovery Point System must support the contract peak of peaks volume of market participant requests.
- The total latency increase from Recovery Points Saves during a single Operation should not exceed 200ms under normal conditions.
- The Recovery Point Orchestrator should act on a record marked as stalled immediately under normal conditions.

## 5.2 Scalability

- The system must automatically scale to handle increased load, using Azure Functions' Premium App Plan(s) and Cosmos DB's auto-scaling capabilities.
- Cosmos DB and Service Bus must be configured to scale horizontally where required.

## 5.3 Availability and Reliability

- The solution must meet existing contracted service level availability, with active-active failover support across UK South and UK West.
- The system must be resilient to partial component failure, such as temporary network loss or cross-region delays.
- Failed messages must follow a structured retry process, and when retries are exhausted, move to the dead-letter queue (DLQ) with notification.

## 5.4 Security

- All Recovery Point operations must use Azure Managed Identity and be restricted by role-based access control (RBAC).
- Data must be encrypted both at rest and in transit.
- Private Endpoints must be enforced for cross region secure communication with Cosmos DB and Azure Blob Storage and Service Endpoints within region.
- All recovery actions and data writes must be logged for auditing.

## 5.5 Compliance

- All data must remain within UK South and UK West Azure regions.
- Azure Policy must be configured for global platform resources to ensure data remains with the UK domicile.

## 5.6 Data Integrity and Backup

- Recovery Point Ledger entries must ensure consistent, idempotent writes.
- Cosmos DB must have continuous backup enabled with a defined retention policy.
- Azure Blob Storage must support lifecycle management to retain or purge data according to policy.

## 5.7 Usability and Supportability

- Power BI dashboards should provide visibility into the status of recovery operations and stalled items.
- Operational support runbooks must be available to guide manual intervention if automated recovery fails.
- All logs should include correlation IDs to support tracing across systems.

## 5.8 Monitoring, Logging, and Maintenance

- Azure Monitor, Log Analytics, and Application Insights must be configured to track:
- Function execution metrics
- Recovery failures and DLQ events
- Alerts for abnormal latency, system errors, or data anomalies
- Scheduled test events should be used to validate orchestrator responsiveness.
- Maintenance tasks, such as purging outdated ledger items, must be automated and auditable.

## 5.9 Service Recovery Point Non-Functional Mitigation Strategies

In the proposed pattern, the Recovery Point Service uses a synchronous “wait-for-commit” pattern to persist operation state to the Recovery Point Ledger (Cosmos DB) using the SaveAsync() method. This ensures that each recovery point is written and confirmed before proceeding to the next task in the operation. In the event this pattern when put under 2 x peak of peaks load during the proposed development phases exceeds the service level objective threshold the following strategies should be considered:

- **Compute:**
  - Scale Up App Plan Compute for Registration Function Apps to reduce the execution time of an Operation.

- **Fire-and-Forget:**
  - The architecture supports a fallback to a “fire-and-forget” pattern, in which:
  - The SaveAsync() call initiates the write to Cosmos DB but does not block the operation from proceeding to the next task.
  - The operation will store the Cosmos DB _etag value (used for optimistic concurrency control) along with the write request to ensure the correct version is being updated or resumed from.
  - If a newer recovery point exists (based on _etag), the system will prevent stale writes, avoiding regression of recovery state.

- **Asynchronous decoupling mechanism:**
  - The Recovery Point Service will enqueue a message onto Azure Service Bus containing the recovery point details and _etag.
  - A dedicated Recovery Point Writer Function will consume these messages and persist them to Cosmos DB independently of the main operation flow.
  - This ensures that recovery state persistence does not interfere with service-level commitments of the primary operation path.

**Summary**

The architecture provides a layered approach to maintain performance while preserving data consistency and recovery reliability, the design ensures the system can dynamically adapt should high volume non-functional performance testing raise concerns.

# 6 Appendix

## 6.1 PR8003 Statement of Work Activities

| ACT-ID | Activity | Role | Status |
|---|---|---|---|
| ACT-01 | Landmark meetings and workshops as necessary to review the requirements and determine an appropriate solution. | Architects | Complete |
| ACT-02 | Assessment/review at a low-level of every synchronous/asynchronous process that may need to be modified to facilitate the completion or restart of a business process that generates outbound messages in the event of execution issues, regardless of the stage at which resumption occurs. | Developers | Complete |
| ACT-03 | Detailed documentation and extensive analysis outlining which and how each of the affected processes will need to be modified and to identify the appropriate points in the process where a checkpoint (backup) should be taken for recovering the process at an appropriate place as required. | Architects | Complete |
| ACT-04 | Highlight dependencies and impacted relationships. | Architects | Complete |
| ACT-05 | Clarification and analysis on what testing will/can be achieved and in what environments. | Testing / QA | Complete |
| ACT-06 | Investigation of a design framework to be applied to every synchronous/asynchronous process to record and manage whether the process is scheduled to run, when it is scheduled to run, and information needed to reschedule the process. | Developers / Architects | Complete |
| ACT-07 | Design the mechanism to detect and reschedule processes that did not start or failed to complete (self-managing). | Developers / Architects | Complete |
| ACT-08 | Creation of detailed documentation covering the proposed new mechanism for the restart of failed business processes. | Architects | Complete |
| ACT-09 | Define epic, features and backlog items for the proof of concept. | Analysts / Architects | |
| ACT-10 | Develop the proof of concept using Azure environment pre-requisites, repositories, and pipelines to assess the impact of the change on the deployment operation and maintenance processes. | Developers / Architects | Complete |
| ACT-11 | Set up the Azure PoC environment. (Local Dev) | Developers | Complete |
| ACT-12 | Deploy the proof of concept application. | DevOps / SRE | N/A |
| ACT-13 | Testing and validation including security testing, performance, and load testing. | DevOps / Developers | Complete |
| ACT-14 | Assess the outputs from monitoring and logging. | Architects / SRE | Complete |
| ACT-15 | Inclusion of interface assessment to peripheral services for reporting, analytics, support, and operations. | Architects | Complete |
| ACT-16 | Gather data and analyse results. | QA / Architects | Assessed |
| ACT-17 | Collect performance metrics, logs, and feedback. | Architects / SRE | Assessed |
| ACT-18 | Prepare a detailed report of the findings, highlighting key insights, challenges encountered, and recommendations to inform the DIA. | Architects | SADD fulfils this action. |

**Table A1.** Statement of Work Activity and Status

## 6.2 Future Features to Enhance R0096

### 6.2.1 ML/AI for Augmented Notifications and Operational Triage

Future iterations of the Recovery Point System could integrate AI/ML capabilities aligned with Energy Code CoCo standards to enrich operational alerts and automated ticket creation. By analysing telemetry data and past incident records, the system can generate intelligent, context-aware notifications and auto-populate support tickets with probable root causes, suggested diagnostic steps, and related historical issues. This reduces triage time, supports shift-left incident handling, and enhances coordination between operations, support, and engineering teams during recovery and escalation scenarios.

### 6.2.2 ML-Based Stall Prediction Through Clustering and Time Series Analysis

An ML model can be introduced to analyse historical stall records using clustering and time series techniques to identify recurring failure types and operational hotspots. By correlating features like time of day, operation type, system load, and participant ID, the system can produce predictive insights into which operations are at higher risk of stalling. These insights can feed into alert thresholds, automated recovery prioritisation, or pre-emptive scaling actions, ultimately increasing the reliability and responsiveness of the orchestration layer.

### 6.2.3 Short-Term Remediation Interface for Unresolved Stalls

To bridge the gap between incident occurrence and permanent resolution, a short-term remediation interface can be introduced to manage persistent, unresolved stalls. This would allow operational teams to define temporary rules mapping specific stall signatures to corrective actions, such as injecting structured error payloads or triggering predefined scripts. These rules would be logged, version-controlled, and auditable, ensuring traceability and safe rollback. This interface enables continued progress on affected operations without manual intervention, while giving development teams time to deliver permanent fixes.

## 6.3 Existing Non-Functional Service, Resiliency and Recovery Patterns

For the purpose of understanding the scope of R0096, this section details the existing functional and non-functional patterns for service availability, resilience and recovery.

The CSS solution is deployed in two Azure Regions, UK South and UK West with each region able to operate independently in the event of a regional failure. UK South is the write master for storage and queues, with read replicas in UK West that can be promoted to write master with an RPO of 15 mins and RTO of 1 hour.

### 6.3.1 Azure Functions

When the CSS storage queue functions are triggered and a failure occurs, Azure will automatically retry the message up to 5 times.

These retries happen immediately and sequentially and on the failure of the fifth attempt Azure moves the message to a Poison Queue. CSS uses this built in functionality to help recover from transient issues.

**R0096 Design:** Recovery Point Service will trigger a stall state on detection of this condition.

### 6.3.2 Cosmos Database

The CSS is built using the Cosmos Database SDK (Software Development Toolkit) and benefits from robust retry logic, the handling of transient failures and ensuring data consistency and availability.

The SDK retries automatically on:

- TCP connection drops
- Timeouts
- DNS resolution issues
- Connection refused/reset exceptions
- Intermittent Azure routing issues

Exponential Backoff with Jitter:

- Prevents retry storms that could overwhelm Cosmos DB or the network.
- Adds randomness (jitter) to retry timing, improving performance under load.

Handling HTTP 429 (Too Many Requests):

- Cosmos DB enforces RU (Request Unit) limits and returns 429 when throttled.
- The SDK inspects the x-ms-retry-after-ms header and waits the specified duration.
- The CSS uses the default values:
    - Up to 9 retry attempts
    - 30-second wait time per retry

Transparent Error Recovery:

- All retries are performed internally by the SDK

**R0096 Design:** Recovery Point Service will trigger a stall state on detection of strategy exhaustion.

### 6.3.3 Azure Storage (Blob & Queue)

Azure Blob Storage and Queue Storage SDKs are utilised by the CSS and have built-in retry logic using RetryOptions:

- Retries on:
    - Network connectivity errors
    - Timeouts
    - DNS resolution issues
    - Connection reset
    - HTTP 5xx (e.g., 500, 502, 503)
    - Throttling responses (429, 503)
- Current Configured Retry Options:
    - Mode = Exponential (Delay increases exponentially between retries)
    - Delay = 15 seconds
    - MaxDelay = 15 seconds
    - MaxRetries = 6 (Limits the total retry attempts to avoid indefinite retries)
    - NetworkTimeout = 30 seconds (Ensures retry doesn't run forever

**R0096 Design:** Recovery Point Service will trigger a stall state on detection of strategy exhaustion.

### 6.3.4 Address Service

For requests to the external Address Service, transient errors (network hiccups, DNS failures, temporary unavailability) are handled via 'Polly' an open-source .NET resilience and transient fault handling library.

- Configured Retry Policy:
  - Retry 3 times
  - Wait 1 second between attempts
- Configured to retry on:
  - HttpRequestException
  - TaskCanceledException (timeout)
  - HTTP 5xx or 408 responses

### 6.3.5 Event Notification Retry Strategy

When sending messages from the CSS to market participants, the CSS handles the failure to receive those messages in a number of ways. If the HTTP response code from the market participant is one of:

- FailedDependency
- TooManyRequests
- InternalServerError
- BadGateway
- ServiceUnavailable
- GatewayTimeout

Then the CSS will attempt to redeliver the message using the strategies below:

- Short Backoff Retry Strategy:
  - Used for first-tier retry handling.
  - Retries 3 times.
  - Wait 20 seconds between retries.
  - Handles transient failures quickly (e.g., function cold starts, short outages).
- Long Backoff Retry Strategy:
  - Used for messages that failed short-term retries.
  - Retries 12 times.
  - Wait 1 hour between retries.

Any other failure results in the message delivery attempt being logged as a failure. With the release of build R.36 to meet the requirements R0080 Failed to Deliver, CSS will provide a service for market participants to request a re-send of any event notifications that the market participant didn't accept.

### 6.3.6 Message Stage Lifecycle

Operations in the system flow through a defined set of stage gates ensuring each operation is complete before the next stage is processed. However, when errors in these stages occur it is possible for the CSS to recover from the point of failure and continue processing.

#### 6.3.6.1 Get Operation Data

Purpose: Gathers the raw inputs required for the operation.
Source: Pulls from Cosmos DB or directly from the message payload.
Recovery Aspect: If this stage fails, the operation has not yet mutated state — safe to retry.

#### 6.3.6.2 Update Unsigned Payload

Purpose: Enriches the incoming payload by computing additional data not supplied in the original request.
Example: Deriving FuelType from reference data.
Recovery Aspect: Stateless enrichment allows safe retries or reprocessing.

#### 6.3.6.3 Business Rules

Purpose: Applies domain-specific validation to ensure the message or operation complies with current market business rules logic.
Outcome: Pass or fail

If it fails the operation halts, and depending on the operation a validation notification may be sent to the originator of the initial request.

Recovery Aspect: Fully deterministic — same input = same output. At this point all the above stages are idempotent.

#### 6.3.6.4 Perform Operation

Purpose: Executes the required action if business rules have passed.
Example: Create/update documents in the Cosmos DB.

Recovery Aspect: Only the following stages are reperformed on recovery.

- Get Operation Data
- Generate Event Messages
- Queue Outbound Messages
- Generate Scheduled Messages
- Queue Scheduled Messages

#### 6.3.6.5 Generate Event Messages

Purpose: Prepares outbound messages for market participants based on the results of the operation.
Recovery Aspect: Only the following stages are reperformed on recovery.

- Get Operation Data
- Generate Event Messages
- Queue Outbound Messages
- Generate Scheduled Messages
- Queue Scheduled Messages

#### 6.3.6.6 Queue Outbound Messages

Purpose: Places outbound messages onto the appropriate notification queue for delivery to market participants.
Recovery Aspect: Only the following stages are reperformed on recovery.

- Get Operation Data
- Generate Scheduled Messages
- Queue Scheduled Messages

#### 6.3.6.7 Generate Scheduled Messages

Purpose: Creates future-dated events to be executed at a specific time
Example: Intervention window expiry
Recovery Aspect: Only the following stages are reperformed on recovery.

- Get Operation Data
- Generate Scheduled Messages
- Queue Scheduled Messages

#### 6.3.6.8 Queue Scheduled Messages

Purpose: Sends scheduled messages to the scheduled-work queue for time-triggered execution.

Recovery Aspect: Only the following stages are reperformed on recovery.

- Get Operation Data
- Generate Scheduled Messages
- Queue Scheduled Messages

**End of Document**