**DataWorks Phase 1**

**Test Principles Document**

The following document outlines the overall design test strategy for DataWorks AWS Migration Phase 1. It must be noted that **this is a working document** and will be updated in accordance with changing dependencies.

**Project Overview**

A new persistent storage layer in AWS will be created in the form of S3 buckets to hold UCFS data. UCFS is currently GDPR non-compliant and can become GDPR compliant by deleting stored data once DataWorks is the persistent data store on completion of the project. The project will include changes to the DataWorks ingest process so that the new process consumes data from UCFS via an event stream instead of a nightly full database export. An encryption design between UCFS and DataWorks will be put in place which will allow DataWorks to decrypt fields which currently cannot be decrypted. It needs to be ensured that DataWorks is able to continue to meet the day -2 SLA for data availability to end users. Lastly, existing services that the DataWorks team already provide UC within the DataWorks AWS environment will be improved.

**Project Key Functional Requirements**

Test design will be based on the following key functional requirements. This is not an exhaustive list and does not include specific test requirements or metrics.

* **Data Retention**
* Persistent and resilient data storage for UCFS data for 25 years post claim closed
* Data is fully resilient to prevent any data loss
* Data security
* **Stability of Data Pipeline**
* The pipeline should operate with no manual intervention
* The number of steps in the pipeline should be kept to a minimum to aid robust design
* Error checking should be built into the code as much as possible
* The pipeline should operate without breaking
* **Monitoring**
* All critical hosting services should be automatically monitored
* High priority alarms should be sent to the DataWorks team for further investigation
* **Scalability**
* Services should be selected and designed to auto scale and be highly available wherever possible
* **Volume**
* The design should be able to handle the full data load of the UC database

**Project Key Non-Functional Requirements**

Test design will be based on the following key non-functional requirements. This is not an exhaustive list and does not include specific test requirements or metrics.

* **Speed**
* The design should reduce the time taken for data to flow through the pipeline to be ready for end users. The SLA is currently day -2 although this should be reduced as much as possible
* Where possible streaming architectures should be used and where not micro-batch should be employed
* **System Availability**
* The system has the same availability requirements as the current DataWorks platform
* Key operational hours are 8am-6pm GMT Mon-Fri
* **Support**
* Support is required 8am-6pm GMT Mon-Fri
* **Minimum Risk Delivery**
* Given the current service in Crown Hosting supports 300 end users and is very fragile (i.e. prone to breaking) the team will minimise risk by working with the minimum number of technical components at any given time. Testing of these components will further minimise risk

**Features to be Tested/Core Epics**

**NB:** Due to this being a working document more information will be added for each feature/epic accordingly.

* AMI Building Service
* Public Key Infrastructure
* UC Kafka messages to be received and stored in DW AWS
* Day 0 data from UC to DataWorks
* Produce a snapshot of data
* Preparation of data for crown
* Security
* Encryption service

**Deliverables**

* **Weekly Test Report**
* Test cases executed and passed in any given week
* Test cases executed and failed in any given week
* Potential blockers
* Bugs/Defect summary
* Upcoming week tasks
* **End of Test Report**
* Scope
* Summary of tests executed
* Summary of tests not executed
* Outstanding bugs/defects
* Recommendations and lessons learned
* Best practices

**Entry/Exit Criteria**

The entry/exit criteria for each component will be included in tests written in Gherkin.

**Defect/Bug Management**

**NB:** The following does not apply to the pre-development phase

Any defects/bugs found during testing will be logged in JIRA and assigned a severity based on the descriptions below:

* **Severity 1**
* The defect affects critical functionality or critical data. It does not have a workaround.
* Blocks any further progress
* Example: CICD pipeline broken or blocked
* **Severity 2**
* The defect affects major functionality or major data. It has a workaround but is not obvious and is difficult to resolve
* Does not completely block QA progress however alternative indirect steps have to be taken
* Example: Kafka consumer goes down; Broker only holds data for 3 days so data ingestion may not occur
* **Severity 3**
* The defect affects minor functionality or non-critical data. It has an easy workaround
* Progress is not blocked
* Example: System notifications to slack and email not working
* **Severity 4**
* The defect does not affect functionality or data. It does not necessarily require a workaround
* There is no impact to productivity or efficiency
* Example: Compaction process slightly slower than normal

**Response Times**

|  |  |
| --- | --- |
| S1 Initial Response | 1 Hour |
| S2 Initial Response | Within 1 Business Day |
| S3 Initial Response | Within 3 Business Days |
| S4 Initial Response | Within 5 Working Days |

The project Delivery Manager will be informed before any defect is assigned a severity. Once the defect has been fixed the functionality will be retested before the test is passed. For any issues found outside of the current sprint it will be determined whether the issue is brought into the sprint according to the severity. Any bug/defect retests will be followed by a regression to ensure no other functionality has been impacted.

**Test Types**

* E2E
* Integration
* Performance
* Regression
* Stress
* Load

**References**

* TDA PowerPoint Presentation