LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)
Data Management

LDM-503-5: (Alert Distribution Validation) Test Plan and Report

Eric Bellm

DMTR-91

Latest Revision: 2019-01-30

Abstract

This is the test plan and report for LDM-503-5: (Alert Distribution Validation), an LSST level 2 milestone pertaining to the Data Management Subsystem.

Change Record

Version	Date	Description	Owner name
	2018-07-16	Initial version.	Bellm, Comoretto
1.0	2019-01-30	DM-16659	Bellm, Comoretto

Document curator: Eric Bellm

Document source location: `https://github.com/lsst-dm/DMTR-91`

Version from source repository: 034820d

Contents

LDM-503-5: (Alert Distribution Validation) Test Plan and Report

1 Introduction

1.1 Objectives

This test activity demonstrates the successful execution of the major components of the Alert Distribution system at the scale of LSST Operations.

It will demonstrate that:

- The Alert Distribution payloads can be made available on systems managed by the LSST Data Facility
- The Alert Distribution payloads can be executed under the control of the relevant LDF services
- All required science data products can be passed through the components of the alert distribution service.

Note that this test plan does not extend to testing integration of the Alert Distribution System with the Alert Generation Science Pipeline or the LSST Science Platform.
We also do not test the Alert Database at this time as its functional requirements are not sufficiently specified to enable testing.

1.2 Scope

The overall test plan for the LSST Data Management system is described in LDM-503.
Success in this test plan is intended to demonstrate completion of the milestone LDM-503-05.
The overall LSST Alert Distribution System test specification is defined in LDM-533.

1.3 System Overview

The LSST Alert Distribution Service is that part of the LSST Data Management System which will be responsible for realtime alert distribution and filtering during LSST operations. The LDM-503-5 milestone exercises only the distribution and filtering aspects; it does not consider generation of alerts by the LSST Science Pipelines or the user interfaces to the simple filtering service through the LSST Science Platform.

1.4 Applicable Documents

LDM-294 Data Management Organization and Management
LDM-503 Data Management Test Plan
LDM-533 LSST Level 1 System Test Specification

1.5 Document Overview

This document was generated from Jira, obtaining the relevant information from the LVV-P1 Jira Test Plan and related Test Cycles (LVV-C3).

Section 1 provides an overview of the test campaign, the system under test (Prompt), the applicable documentation, and explains how this document is organized. Section 2 describes the configuration used for this test. Section 3 describes the necessary roles and lists the individuals assigned to them.

Section 4 provides a summary of the test results, including an overview in Table 1, an overall assessment statement and suggestions for possible improvements. Section 5 provides detailed results for each step in each test case.

The current status of test plan LVV-P1 in Jira is Completed.

1.6 References

[1] **[LDM-533]**, Bellm, E.C., 2017, *Level 1 System Software Test Specification*, LDM-533, URL https://ls.st/LDM-533

[2] **[LDM-294]**, O'Mullane, W., Swinbank, J., Jurić, M., DMLT, 2018, *Data Management Organization and Management*, LDM-294, URL https://ls.st/LDM-294

[3] **[LDM-503]**, O'Mullane, W., Swinbank, J., Jurić, M., Economou, F., 2018, *Data Management Test Plan*, LDM-503, URL https://ls.st/LDM-503

2 Test Configuration

2.1 Data Collection

Observing is not required for this test campaign.

2.2 Verification Environment

This test activity shall be executed on the Kubernetes Commons at the LDF.
As discussed in https://dmtn-028.lsst.io/ and https://dmtn-081.lsst.io/, the test machine should have at least 16 cores, 64 GB of memory and access to at least 1.5 TB of shared storage.

3 Personnel

The following personnel are involved in this test activity:

- Test Plan (LVV-P1) owner: Eric Bellm
- Test Cycles:
  - LVV-C3 owner: Eric Bellm
    - Test case LVV-T216 tester: Eric Bellm
    - Test case LVV-T217 tester: Eric Bellm
    - Test case LVV-T218 tester: Eric Bellm
- Additional Test Personnel involved: None

4 Overview of the Test Results

4.1 Summary

Test Cycle **LVV-C3: LDM-503-5: Alert Distribution Validation**			
test case	**status**	**comment**	**issues**
LVV-T216	Conditional Pass	Missing git-lfs in steps 2 and 3. Used pre-built Docker image instead. Remaining steps pass.	
LVV-T217	Conditional Pass	Missing git-lfs in step 2. Used docker image instead. Remaining steps pass.	
LVV-T218	Conditional Pass	Missing git-lfs in step 2 and 3. Used docker image instead. Last step result deviated sligthly from the expected result for one consumer.	

Table 1: Test Results Summary

4.2 Overall Assessment

The overall result of the test campaign is: PASS.

Missing elements on the Kubernetes Commons prevented us from building the Docker images in place and using a complete night of unique alerts. However, using pre-built Docker images with a smaller, repeated sample of alerts enabled the test to proceed as expected.

4.3 Recommended Improvements

Re-execution with the missing components on the Kubernetes Commons.

Examination of the deserialized alerts in the logs is cumbersome and does not allow real verification of the alert content. Improved tools for counting alerts, comparing their contents to the sent alerts, and timing throughput would all improve the utility of this test.

5 Detailed Test Results

5.1 Test Cycle LVV-C3

Open test cycle *LDM-503-5: Alert Distribution Validation* in Jira.

LDM-503-5: Alert Distribution Validation
Status: Done

Test run with test cases in draft.

5.1.1 Software Version/Baseline

This test uses only the code in
https://github.com/lsst-dm/alert_stream/ , commit ef3c9e16c42a3265d4f6258ba6983be8b009e7c0

5.1.2 Configuration

Documents

This test report refers to the execution of tests described in the section 1.2 and documented in the LDM-533 test specification issue 2.0.

Hardware

This test activity shall be executed on the Kubernetes Commons at the LDF. As discussed in https://dmtn-028.lsst.io/ and https://dmtn-081.lsst.io/, the test machine should have at least 16 cores, 64 GB of memory and access to at least 1.5 TB of shared storage.

Software

This test uses only the code in

https://github.com/lsst-dm/alert_stream/ , commit ef3c9e16c42a3265d4f6258ba6983be8b009e7c0

Input Data

Input data for all tests was based on simulated alert packets

5.1.3 Test Cases in LVV-C3 Test Cycle

5.1.3.1 Test Case LVV-T216

Open *LVV-T216* test case in Jira.

This test will check:

- That the Alert Distribution payloads are available from documented channels.
- That the Alert Distribution payloads can be installed on LSST Data Facility-managed systems.
- That the Alert Distribution payloads can be executed by LSST Data Facility-managed systems.

Preconditions:

Execution status: **Conditional Pass**

Final comment:
Missing git-lfs in steps 2 and 3. Used pre-built Docker image instead. Remaining steps pass.

Detailed step results:

Step		Description, Results and Status
1	Description	Download Kafka Docker image from https://github.com/lsst-dm/alert_stream.
	Expected Result	Runs without error
	Actual Result	runs without error
	Status	Pass
2	Description	Change to the alert_stream directory and build the docker image. `docker build -t "lsst-kub001:5000/alert_stream" .`
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead.
	Status	Conditional Pass
3	Description	Register it with Kubernetes docker push lsst-kub001:5000/alert_stream
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead
	Status	Conditional Pass

4	Description	From the alert_stream/kubernetes directory, start Kafka and Zookeeper:
		
		`kubectl create -f zookeeper-service.yaml`
		`kubectl create -f zookeeper-deployment.yaml`
		`kubectl create -f kafka-deployment.yaml`
		`kubectl create -f kafka-service.yaml`
		
		(use kubectl get pods/services between each command to check status; wait until each is "Running" before starting the next command)

	Expected Result	Runs without error
	Actual Result	Runs without error
	Status	Pass
5	Description	Confirm Kafka and Zookeeper are listed when running kubectl get pods and kubectl get services

Expected Result	Output should be similar to: kubectl get pods NAME READY STATUS RESTARTS AGE kafka-768ddf5564-xwgvh 1/1 Running 0 31s zookeeper-f798cc548-mgkpn 1/1 Running 0 1m kubectl get services NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE kafka ClusterIP 10.105.19.124 <none> 9092/TCP 6s zookeeper ClusterIP 10.97.110.124 <none> 32181/TCP 2m
Actual Result	as expected
Status	Pass

5.1.3.2 Test Case LVV-T217

Open *LVV-T217* test case in Jira.

This test will check that the full stream of LSST alerts can be distributed to end users.

Specifically, this will demonstrate that:

- Serialized alert packets can be loaded into the alert distribution system at LSST-relevant scales (10,000 alerts every 39 seconds);
- Alert packets can be retrieved from the queue system at LSST-relevant scales.

Preconditions:
Input data: A sample of Avro-formatted alert packets.

Execution status: **Conditional Pass**

Final comment:

Missing git-lfs in step 2. Used docker image instead. Remaining steps pass.

Detailed step results:

Step		Description, Results and Status
1	Description	Download Kafka Docker image from https://github.com/lsst-dm/alert_stream.
	Expected Result	Runs without error
	Actual Result	
	Status	Pass
2	Description	Change to the alert_stream directory and build the docker image. `docker build -t "lsst-kub001:5000/alert_stream" .`
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead
	Status	Conditional Pass
3	Description	Register it with Kubernetes docker push lsst-kub001:5000/alert_stream
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead
	Status	Conditional Pass

4	Description	From the alert_stream/kubernetes directory, start Kafka and Zookeeper:
		
		`kubectl create -f zookeeper-service.yaml`
		`kubectl create -f zookeeper-deployment.yaml`
		`kubectl create -f kafka-deployment.yaml`
		`kubectl create -f kafka-service.yaml`
		
		(use kubectl get pods/services between each command to check status; wait until each is "Running" before starting the next command)

	Expected Result	Runs without error
	Actual Result	
	Status	Pass
5	Description	Confirm Kafka and Zookeeper are listed when running kubectl get pods and kubectl get services

	Expected Result	Output should be similar to: kubectl get pods NAME READY STATUS RESTARTS AGE kafka-768ddf5564-xwgvh 1/1 Running 0 31s zookeeper-f798cc548-mgkpn 1/1 Running 0 1m kubectl get services NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE kafka ClusterIP 10.105.19.124 <none> 9092/TCP 6s zookeeper ClusterIP 10.97.110.124 <none> 32181/TCP 2m
	Actual Result	
	Status	Pass
6	Description	Start a consumer that monitors the full stream and logs a deserialized version of every Nth packet: `kubectl create -f consumerall-deployment.yaml`
	Expected Result	Runs without error
	Actual Result	
	Status	Pass

7	Description	`Start a producer that reads alert packets from disk and loads them into the Kafka queue:`
		
		`kubectl create -f sender-deployment.yaml`

	Expected Result	Runs without error
	Actual Result	

	Status	Pass
8	Description	Determine the name of the alert sender pod with kubectl get pods Examine output log files. kubectl logs <pod name> Verify that alerts are being sent within 40 seconds by subtracting the timing measurements.
	Expected Result	Similar to kubectl logs sender-7d6f98586f-nhwfj visit: 1570. time: 1530588618.0313473 visits finished: 1 time: 1530588653.5614944 visit: 1571. time: 1530588657.0087624 visits finished: 2 time: 1530588692.506188 visit: 1572. time: 1530588696.0051727 visits finished: 3 time: 1530588731.5900314
	Actual Result	visit: 1570. time: 1530674964.0035944 visits finished: 1 time: 1530675000.5279133 visit: 1571. time: 1530675003.00461 visits finished: 2 time: 1530675039.5390277 visit: 1572. time: 1530675042.005049 visits finished: 3 time: 1530675078.4921534 visit: 1573. time: 1530675081.0037167

	Status	Pass
9	Description	Determine the name of the consumer pod with kubectl get pods Examine output log files. kubectl logs <pod name> The packet log should show deserialized alert packets with contents matching the input packets.
	Expected Result	Similar to {'alertId': 12132024420, 'l1dbId': 71776805594116, 'diaSource': {'diaSourceId': 73499448928374785, 'ccdVisitId': 2020011570, 'diaObjectId': 71776805594116, 'ssObjectId': None, 'parentDiaSourceId': None, 'midPointTai': 59595.37041, 'filterName': 'y', 'ra': 172.24912810036074, 'decl': -80.64214929176521, 'ra_decl_Cov': {'raSigma': 0.0003428002819418907, 'declSigma': 0.00027273103478364646, 'ra_decl_Cov': 0.000628734880592674}, 'x': 2979.08837890625, 'y': 3843.328857421875, 'x_y_Cov': {'xSigma': 0.6135467886924744, 'ySigma': 0.77132648229599, 'x_y_Cov': 0.0007463791407644749}, 'apFlux': None, 'apFluxErr': None, 'snr': 0.36651650071144104, 'psFlux': 7.698232025177276e-07, 'psRa': None, 'psDecl': None, 'ps_Cov': None, 'psLnL': None, 'psChi2': None, 'psNdata': None, 'trailFlux': None, 'trailRa': etc.

Actual Result

topic:full-stream, partition:0, status:end, offset:0, key:None, time:1530674576.589
{'alertId': 12132024420, 'l1dbId': 71776805594116, 'diaSource': {'diaSourceId': 73499448928374785, 'ccdVisitId': 2020011570, 'diaObjectId': 71776805594116, 'ssObjectId': None, 'parentDiaSourceId': None, 'midPointTai': 59595.37041, 'filterName': 'y', 'ra': 172.24912810036074, 'decl': -80.64214929176521, 'ra_decl_Cov': {'raSigma': 0.0003428002819418907, 'declSigma': 0.00027273103478364646, 'ra_decl_Cov': 0.000628734880592674}, 'x': 2979.08837890625, 'y': 3843.328857421875, 'x_y_Cov': {'xSigma': 0.6135467886924744, 'ySigma': 0.77132648229599, 'x_y_Cov': 0.0007463791407644749}, 'apFlux': None, 'apFluxErr': None, 'snr': 0.36651650071144104, 'psFlux': 7.698232025177276e-07, 'psRa': None, 'psDecl': None, 'ps_Cov': None, 'psLnL': None, 'psChi2': None, 'psNdata': None, 'trailFlux': None, 'trailRa': None, 'trailDecl': None, 'trailLength': None, 'trailAngle': None, 'trail_Cov': None, 'trailLnL': None, 'trailChi2': None, 'trailNdata': None, 'dipMeanFlux': None, 'dipFluxDiff': None, 'dipRa': None, 'dipDecl': None, 'dipLength': None, 'dipAngle': None, 'dip_Cov': None, 'dipLnL': None, 'dipChi2': None, 'dipNdata': None, 'totFlux': 7.267262844834477e-06, 'totFluxErr': 1.990927557926625e-06, 'diffFlux': 7.698232025177276e-07, 'diffFluxErr': 2.1003779693273827e-06, 'fpBkgd': None, 'fpBkgdErr': None, 'ixx': None, 'iyy': None, 'ixy': None, 'i_cov': None, 'ixxPSF': None, 'iyyPSF': None, 'ixyPSF': None, 'extendedness': None, 'spuriousness': None, 'flags': 516}, 'prv_diaSources': None, 'diaObject': {'diaObjectId': 71776805594116, 'ra': 172.249128110081, 'decl': -80.64214929171955, 'ra_decl_Cov': {'raSigma': 0.0003098504093941301, 'declSigma': 0.0001309745421167463, 'ra_decl_Cov': 0.0007125047268345952}, 'radecTai': 59595.139501, 'pmRa': -9.0, 'pmDecl': -0.25999999046325684, 'parallax': 1.1651986837387085, 'pm_parallax_Cov': {'pmRaSigma': 0.0, 'pmDeclSigma': 0.0, 'parallaxSigma': 0.0, 'pmRa_pmDecl_Cov': 0.0, 'pmRa_parallax_Cov': 0.0, 'pmDecl_parallax_Cov': 0.0}, 'pmParallaxLnL': 0.290818989276886, 'pmParallaxChi2': 0.399796724319458, 'pmParallaxNdata': 0, 'uPSFluxMean': None, 'uPSFluxMeanErr': None, 'uPSFluxSigma': None, 'uPSFluxChi2': None, 'uPSFluxNdata': None, 'gPSFluxMean': None, 'gPSFluxMeanErr': None, 'gPSFluxSigma': None, 'gPSFluxChi2': None, 'gPSFluxNdata': None, 'rPSFluxMean': None, 'rPSFluxMeanErr': None, 'rPSFluxSigma': None, 'rPSFluxChi2': None, 'rPSFluxNdata': None, 'iPSFluxMean': None, 'iPSFluxMeanErr': None, 'iPSFluxSigma': None, 'iPSFluxChi2': None, 'iPSFluxNdata': None, 'zPSFluxMean': None, 'zPSFluxMeanErr': None, 'zPSFluxSigma': None, 'zPSFluxChi2': None, 'zPSFluxNdata': None, 'yPSFluxMean': None, 'yPSFluxMeanErr': None, 'yPSFluxSigma': None, 'yPSFluxChi2': None, 'yPSFluxNdata': None, 'uFPFluxMean': None, 'uFPFluxMeanErr': None, 'uFPFluxSigma': None, 'gFPFluxMean': None, 'gFPFluxMeanErr': None, 'gFPFluxSigma': None, 'rFPFluxMean': None, 'rFPFluxMeanErr': None, 'rFPFluxSigma': None, 'iFPFluxMean': None, 'iFPFluxMeanErr': None, 'iFPFluxSigma': None, 'zFPFluxMean': None, 'zFPFluxMeanErr': None, 'zFPFluxSigma': None, 'yFPFluxMean': None, 'yFPFluxMeanErr': None, 'yFPFluxSigma': None, 'uLcPeriodic': None, 'gLcPeriodic': None, 'rLcPeriodic': None, 'iLcPeriodic': None, 'zLcPeriodic': None, 'yLcPeriodic': None, 'uLcNonPeriodic': None, 'gLcNonPeriodic': None, 'rLcNonPeriodic': None, 'iLcNonPeriodic': None, 'zLcNonPeriodic': None, 'yLcNonPeriodic': None, 'nearbyObj1': None, 'nearbyObj1Dist': None, 'nearbyObj1LnP': None, 'nearbyObj2': None, 'nearbyObj2Dist': None, 'nearbyObj2LnP': None, 'nearbyObj3': None, 'nearbyObj3Dist': None, 'nearbyObj3LnP': None, 'flags': 873}, 'ssObject': None, 'diaObjectL2': None, 'diaSourcesL2': None}

Status	Pass

5.1.3.3 Test Case LVV-T218

Open *LVV-T218* test case in Jira.

This test will demonstrate the LSST Alert Filtering Service that returns a subset of alerts from the full stream identified by user-provided filters.

Specifically, this will demonstrate that:

- The filtering service can retrieve alerts from the full alert stream and filter them according to their contents;
- The filtered subset can be delivered to science users.

Preconditions:
Input data: A sample of Avro-formatted alert packets derived from LSST simulations corresponding to one night of simulated LSST observing.

Execution status: **Conditional Pass**

Final comment:
Missing git-lfs in step 2 and 3. Used docker image instead. Last step result deviated sligthly from the expected result for one consumer.

Detailed step results:

Step		Description, Results and Status
1	Description	Download Kafka Docker image from https://github.com/lsst-dm/alert_stream.
	Expected Result	Runs without error

	Actual Result	
	Status	Pass
2	Description	Change to the alert_stream directory and build the docker image. `docker build -t "lsst-kub001:5000/alert_stream"`
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead
	Status	Conditional Pass
3	Description	Register it with Kubernetes docker push lsst-kub001:5000/alert_stream
	Expected Result	Runs without error
	Actual Result	Did not execute because of missing git-lfs; used pre-built Docker image instead
	Status	Conditional Pass

4	Description	From the alert_stream/kubernetes directory, start Kafka and Zookeeper:
		
		`kubectl create -f zookeeper-service.yaml`
		`kubectl create -f zookeeper-deployment.yaml`
		`kubectl create -f kafka-deployment.yaml`
		`kubectl create -f kafka-service.yaml`
		
		(use kubectl get pods/services between each command to check status; wait until each is "Running" before starting the next command)

	Expected Result	Runs without error
	Actual Result	
	Status	Pass
5	Description	Confirm Kafka and Zookeeper are listed when running kubectl get pods and kubectl get services

	Expected Result	Output should be similar to: kubectl get pods NAME READY STATUS RESTARTS AGE kafka-768ddf5564-xwgvh 1/1 Running 0 31s zookeeper-f798cc548-mgkpn 1/1 Running 0 1m kubectl get services NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE kafka ClusterIP 10.105.19.124 <none> 9092/TCP 6s zookeeper ClusterIP 10.97.110.124 <none> 32181/TCP 2m
	Actual Result	
	Status	Pass

6	Description	Start 100 consumers that consume the filtered streams and logs a deserialized version of every Nth packet:
		
		`kubectl create -f consumer1-deployment.yaml`
		`kubectl create -f consumer2-deployment.yaml`
		`kubectl create -f consumer3-deployment.yaml`
		`kubectl create -f consumer4-deployment.yaml`
		`kubectl create -f consumer5-deployment.yaml`
		`kubectl create -f consumer6-deployment.yaml`
		`kubectl create -f consumer7-deployment.yaml`
		`kubectl create -f consumer8-deployment.yaml`
		`kubectl create -f consumer9-deployment.yaml`
		`kubectl create -f consumer10-deployment.yaml`

	Expected Result	Runs without error
	Actual Result	

	Status	Pass

7	Description	Start 5 filter groups:
		
		`kubectl create -f filterer1-deployment.yaml`
		`kubectl create -f filterer2-deployment.yaml`
		`kubectl create -f filterer3-deployment.yaml`
		`kubectl create -f filterer4-deployment.yaml`
		`kubectl create -f filterer5-deployment.yaml`

	Expected Result	Runs without error
	Actual Result	
	Status	Pass
8	Description	Start a producer that reads alert packets from disk and loads them into the Kafka queue: `kubectl create -f sender-deployment.yaml`
	Expected Result	Runs without error
	Actual Result	
	Status	Pass

9	Description	Determine the name of the alert sender pod with kubectl get pods Examine output log files. kubectl logs <pod name> Verify that alerts are being sent within 40 seconds by subtracting the timing measurements.
	Expected Result	Similar to kubectl logs sender-7d6f98586f-nhwfj visit: 1570. time: 1530588618.0313473 visits finished: 1 time: 1530588653.5614944 visit: 1571. time: 1530588657.0087624 visits finished: 2 time: 1530588692.506188 visit: 1572. time: 1530588696.0051727 visits finished: 3 time: 1530588731.5900314
	Actual Result	visit: 1570. time: 1530675627.0249922 visits finished: 1 time: 1530675664.2054636 visit: 1571. time: 1530675666.00571 visits finished: 2 time: 1530675702.7181122 visit: 1572. time: 1530675705.0034182 visits finished: 3 time: 1530675741.6658716 visit: 1573. time: 1530675744.0031254 visits finished: 4 time: 1530675780.6261215 visit: 1574. time: 1530675783.0040474

	Status	Pass
10	Description	Determine the name of the consumer pods with kubectl get pods Examine output log files. kubectl logs <pod name> The packet log should show deserialized alert packets with contents matching the input packets.
	Expected Result	Similar to {'alertId': 12132024420, 'l1dbId': 71776805594116, 'diaSource': {'diaSourceId': 73499448928374785, 'ccdVisitId': 2020011570, 'diaObjectId': 71776805594116, 'ssObjectId': None, 'parentDiaSourceId': None, 'midPointTai': 59595.37041, 'filterName': 'y', 'ra': 172.24912810036074, 'decl': -80.64214929176521, 'ra_decl_Cov': {'raSigma': 0.0003428002819418907, 'declSigma': 0.00027273103478364646, 'ra_decl_Cov': 0.000628734880592674}, 'x': 2979.08837890625, 'y': 3843.328857421875, 'x_y_Cov': {'xSigma': 0.6135467886924744, 'ySigma': 0.77132648229599, 'x_y_Cov': 0.000746379140764 4749}, 'apFlux': None, 'apFluxErr': None, 'snr': 0.36651650071144104, 'psFlux': 7.698232025177276e-07, 'psRa': None, 'psDecl': None, 'ps_Cov': None, 'psLnL': None, 'psChi2': None, 'psNdata': None, 'trailFlux': None, 'trailRa': etc.

Actual Result	kubectl logs consumer1-6cdf9b57f4-swrpk | head -n 2 topic:Filter001, partition:0, status:end, offset:0, key:None, time:1530675384.799 {'alertId': 12132024334, 'l1dbId': 71776752985092, 'diaSource': {'diaSourceId': 73499395056734209, 'ccdVisitId': 1020011570, 'diaObjectId': 71776752985092, 'ssObjectId':... consumer 2: topic:Filter011, partition:0, status:end, offset:0, key:None, time:1530675474.542 {'alertId': 12132024334, 'l1dbId': 71776752985092, 'diaSource': {'diaSourceId': 73499395056734209, 'ccdVisitId': 1020011570, 'diaObjectId': 71776752985092, 'ssObjectId': None, ... (similar results for consumers 3-5, 7-10) consumer 6: %3|1530675391.040|FAIL|rdkafka#consumer-1| [thrd:kafka.alerts-lsst.svc.cluster.local:9092/bootstrap]: kafka.alerts-lsst.svc.cluster.local:9092/bootstrap: Failed to resolve 'kafka.alerts-lsst.svc.cluster.local:9092': Temporary failure in name resolution %3|1530675391.040|ERROR|rdkafka#consumer-1| [thrd:kafka.alerts-lsst.svc.cluster.local:9092/bootstrap]: kafka.alerts-lsst.svc.cluster.local:9092/bootstrap: Failed to resolve 'kafka.alerts-lsst.svc.cluster.local:9092': Temporary failure in name resolution %3|1530675391.040|ERROR|rdkafka#consumer-1| [thrd:kafka.alerts-lsst.svc.cluster.local:9092/bootstrap]: 1/1 brokers are down
Status	Conditional Pass