# INSANE DCP Reliability WG Master Document



## Last updated

2019-11-01

## Status

A lot for 2019Q4 has been implemented - waiting for Parth to finish things up and announce the new reliability tracking. Once the metrics are surfaced, decisions can be made about whether they need to be improved and how. Prototype dashboard.

## Contributors

- Justin Clark-Casey
- Parth Shah

# Why this document?

To coordinate work for the 2019Q3 reliability task and beyond.

# Links

2019-09-18 DCP Reliability Working Group - Reliability standup and Parth's presentation of DCP-wide reliability information collection and tracking
2019-09-25 Devsecops - Discussion of reliability scope for 2019Q4
DCP can I rely on you? 2019-07-31 - Presentation on reliability problems and how to consistently surface them
#dcp-reliability-wg Slack channel

# Overall Goal

Reduce dataset processing failures from 1 in 3 to 1 in 20 (SLOs) and report 100% of inter-component handover errors.

This addresses the INSANE (INgest, Store, AnalyzE) part of the data cycle only. It does not address consumer component reliability (Azul, Matrix, Query, etc.). It does, however, include azul's indexing and processing of bundle notifications from the dss.

This goal isn't set in stone and can be softened if necessary for this quarter. Follow on work can become a candidate for scheduling in subsequent quarters.

# High-level actions

There are many potential actions. It is likely they couldn't all be done in Q3. The most important ones may include creating and automatically tracking metrics so we can easily track how well we are doing on reliability in all subsequent quarters.

1) **Agree** on metrics for measuring reliability. The failure numbers above come from dataset re-ingestion. An ingestion is considered a failure if it required some manual intervention to complete. In most cases there were multiple failures spread across many components. This metric is crude but simple - is it suitable?
2) Get to a point where 100% of inter-component failures are **reported** in a manner that they can be automatically recorded (see below). As a side effect, this may make it easier to show them to an operator instead of requiring them to comb through component-specific logs.. Any instances of this must raise a ticket for the component which appears to have the issue and this must be addressed at the priority set for reliability work (which currently is high as per its voting number from PLs).
3) **Record** INSANE reliability metrics so that we can see if we're getting better or worse. This should be an output from the scale test and also a statistic made available in the tracker for production ingestion.

4) Where necessary, improve **the DCP Data tracker** to a) tell users the status of their submission and b) make it easier to identify INSANE failures that don't produce external error reports.
5) Scope and perform **per-component reliability** tickets. For example, ingest has a lot.
6) Make sure that the existing daily integration tests **exercise** the most INSANE functionality. This may already be the case but should probably be extended to updates (Ingest has a ticket for this though that's scoped to ingest parts only).
7) **Increase the success** rate of the daily integration test in staging and production environments to the goal SLO or above.
    a) This is proving very challenging right now, especially with multiple analysis pipelines. Is it possible to lower the bar for success (e.g. just require SS2 to pass)?
8) **Reinstate the scale tests** to run at a bi-weekly cadence on weeks 2 and 4 of the DCP sprint (this to give the components opportunity to incorporate reliability work items before the next sprint starts [week 4] and after they've been able to do some work on other items [week 2]).
9) Make sure the scale tests reflect **real dataset structures** in form if not in scale. This is where Matt Green's existing work becomes relevant (need link) https://github.com/HumanCellAtlas/ingest-graph-validator.
    a) Do we want to push scale tests beyond the equivalent of our largest/most complex current datasets? I (justincc) would say not at this stage. At the very least for ingest this is a complex goal which I think needs its own theme in a subsequent quarter.
10) If scale test performance falls below the SLO have an agreed upon SLO for **fixing** this.
11) If **real dataset ingestion** fails during this period investigate and incorporate lessons.
12) The devops team will **maintain** tests and monitoring, as per a tech-arch meeting discussion. They need the remit to ask components to fix issues, backed up by the PM team. This will require agreement in tech-arch about the level of people time and a negotiation with PM. Initially, this will probably have to come from existing teams, which can be justified by the high priority of this item. Eventually new dedicated people time may be required.
    a) Include in devsecops or data operations?

Ideally we would look to re-ingest real datasets that have previously failed and compare reliability over time. However, these tend to be large. The larger ones will probably be better at revealing problems but I (justincc) doubt that test re-ingestion will fit into Q3. In fact, there are a lot of items above so spillover from Q3 and into 2020 will likely be inevitable but this simply illustrates the complexity of becoming more reliable, not that we shouldn't do the work.

Next steps? [Parth] At a minimum, next steps include 1) reinstate scale testing and discussions on making scale tests more representative 2) Identify existing reliability tickets in each component and prioritize 3) Set unit of work SLO and measure this in both scale tests and production ingestions

# Phase 2019Q3

By common agreement, the above constitutes too much work to perform in a single quarter. For the 2019Q3 I (justincc) propose that we:

1) **Agree** on 2 metrics. Why only 2? This is to stop us getting bogged down by trying to agree everything up front. It's an arbitrary number that can be changed. More metrics can be agreed in subsequent quarters or ad hoc. This will be determined by the *reliability wg*. It could even just be 1 metric to do an initial end-to-end.
2) **Determine** the mechanism for automatically gather metrics. This will be determined by the *reliability wg*.
3) All components **report** information that is relevant to those metrics such that it can be automatically gathered. This will be performed by *component teams*.
4) Find or create an automated way to **gather** metrics. The *devops* team will be responsible for this.
5) Find or create an automated way to **track** metrics. This may be part of the DCP data tracker. The *devops* team will be responsible for this.

The two metrics we will gather are *how often successful* and *how long*. What do these mean?

# Metrics

## How successful?

In this phase, I (justincc) propose this is a simple "Did everything succeed first time?". It would take no account of size/complexity of the submission or the number of failures - these refinements would be left to a later iteration.

### Detection points

- How do we know when a new submission has started? When a new submission envelope appears in the Ingest API.
- How do we know when ingest has encountered an error? The number of submission errors in an envelope > 0.
- How do we know when the upload service encountered an error? Reported via ingest submission envelope errors.
- How do we know when the datastore encountered an error? Reported via ingest submission envelope errors.
- How do we know when analysis encountered an error? Through Cromwell.
- How do we know when the submission has completed? Short answer - not sure. We would need accurate information as to what analysis should have been triggered. This will depend on active analysis pipelines and how they respond to notifications. One would need to scan the bundles in the datastore and determine how many we expect to trigger a workflow.

Is there value in splitting this right now into 2 metrics - failures in primary submission and total failures including analysis?

### Breakdown

- How successful (%)
    - Dcp wide integration tests (separated by pipeline and environment)
    - Scale tests (per bundle)
    - Project prod ingestion (with and without analysis)

## How long?

Again, I (justincc) propose that this is extremely crude at first. But this is arguably not meaningful at all if it doesn't account for at least the size of the submission. Perhaps should be time per bundle?

As above, we need to know when the submission starts (relatively easy) and when it finishes (hard). Knowing when primary submission finishes is easier.

### Breakdown

- How long (mins)
    - Scale test (per project, avg size of project by bundle count and gb, and time per gb)
    - Prod ingestion (with and without analysis, per project and per gb)

# Mechanism

1) cloudwatch metrics emitted from `dcp` repo to grab the metrics for integration and scale tests
2) cloudwatch metrics emitted from tracker/`data-monitoring-dashboard` repo for production ingestion
3) grafana dashboard for live view of metrics
4) api endpoint in tracker to generate monthly report json that will be incorporated into the monthly dcp data and ops newsletter

# Status

Parth has done considerable work on this and we're looking forward to seeing this public soon. This effectively (?) fulfills Q3 requirements, though the question is whether this will be fully ready within Q3.

Definition of done is a public dashboard. We'll move this to Q4. A lot has been done but it needs a little bit more time.

# Phase 2019Q4

Dashboard showing reliability information now available needs to be completed. Historical information needs to be captured so we can tell how reliability is trending over time and how it changes in response to infrastructure investment. Planning on completing this within Q4. It doesn't make sense to look for more information until we have feedback on what we are gathering.

## Discussion

Once we have actionable information (e.g. failure rate is 33%) 2 questions arise

Question 1: Who is responsible for setting reliability targets? This should come from product, not from the technical team. For example, from MattW: CZI has funded data contributors at the rate of 3 datasets a month, so the system at least has to be reliable enough to ingest that data.

Question 2: What would be the process for determining action items from the metrics? Who is responsible for driving this work? How would we come up with them and how would we assign them? What priority would they have?

This is a problem larger than just reliability work. For example, Matt has done a lot of work to put information in Grafana but nobody pays attention.

Justin - these are critical issues that will need to be addressed. I believe that once we have a dashboard there will be energy for doing something about them.

DaveR - could take a look DCP wide and enumerate interfaces in terms of is it queued, how does it handles messages, etc (there was a third thing, sorry I forgot it [justincc])? These things could get rid of a lot of inter-component problems.

# Phase 2020H1

More work will be required both to record metrics and reduce failures to provision target levels. On an **extremely** rough estimate (without yet discussion with other tech leads) this will take at least the first half of 2020, if not beyond.

Essentially, what I (justincc) think we need to do is

a) get, record and track the statistics

b) get a process for using these to drive reliability improvements

c) get actionable targets from product/data ops. These have to account for user requirements and internal requirements (e.g. reduce errors to stop developers spending time on manual remediation)

d) identify the areas that need to improve to hit these targets

e) do the work
f) repeat

# Beyond 2020H1

Beyond this

a) Other reliability goals may be set as required by users.
b) We will need to reserve time for ongoing maintenance of codes and tests to continue measuring success and respond to issues that cause us to fall below target reliability levels.