moz://a

Deduplication: Where Beam Fits In

Beam Summit 2021

2021-08-04

Jeff Klukas
Sr. Staff Data Engineer, Firefox Data Platform

Who am I?

- Data engineer at Mozilla primarily working on the Firefox data pipeline
  - Deployed on Cloud Dataflow
  - 20 TB/day, 2 billion records/day
- Occasional poster on Beam mailing lists
- Author of several PRs in the Beam Java SDK, mostly in documentation and BigQueryIO
- Technical writing at https://jeff.klukas.net
- **klukas** on ASF Slack
- Email me: jeff@klukas.net

Objective

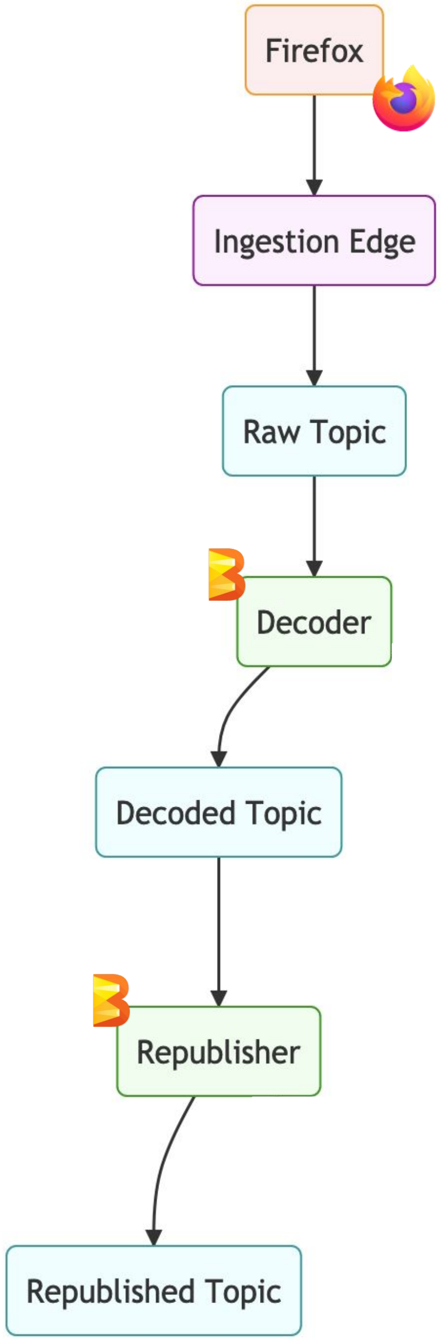
We'll compare the robustness, performance, and operational experience of deduplicating using built-in Beam transforms vs. storing IDs in an external Redis cluster and why Mozilla switched from one approach to the other in our streaming pipelines.

Mozilla's pipeline is open source on GitHub: https://github.com/mozilla/gcp-ingestion

Agenda

1. Sources of Duplicate Messages
2. Beam's Built-In Transforms for Deduplication
3. End-to-End Identifiers
4. Externalizing State
5. Comparison and Questions

Sources of Duplicate Messages



Built-In Transforms

The Java SDK provides two families of transforms relevant to this problem.

```
PCollection<String> words = ...;
PCollection<String> deduplicatedWords =
  words.apply(Deduplicate.<String>values());
```

```
PCollection<String> words = ...;
PCollection<String> uniqueWords =
     words.apply(Distinct.<String>create());
```

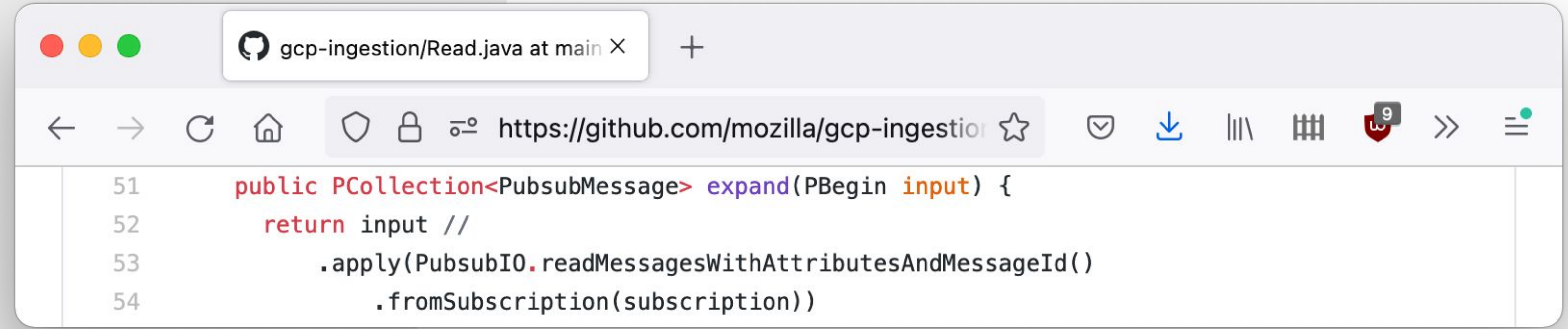
“`Distinct` guarantees uniqueness of values within a PCollection but may support a narrower set of windowing strategies or may delay when output is produced” compared to `Deduplicate`.

Readers and PubsubIO

Beam's I/O machinery includes hooks for deduplication. For example, `PubsubIO.Read` calls `Deduplicate` under the hood to ensure each message is read only once.

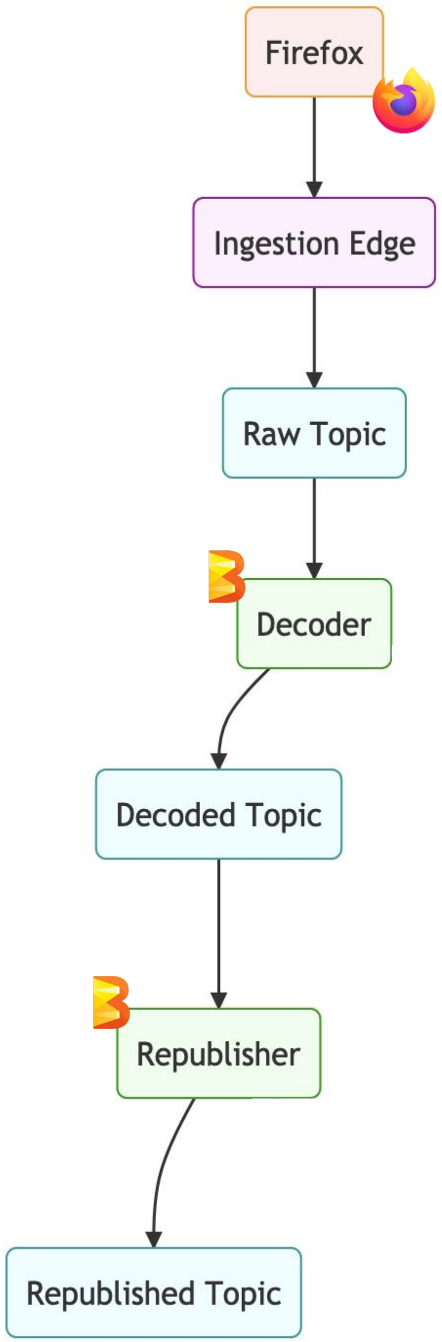
The Dataflow runner even pushes PubsubIO to a separate service, so deduplication state does not consume worker resources.

This code snippet implicitly includes deduplication:

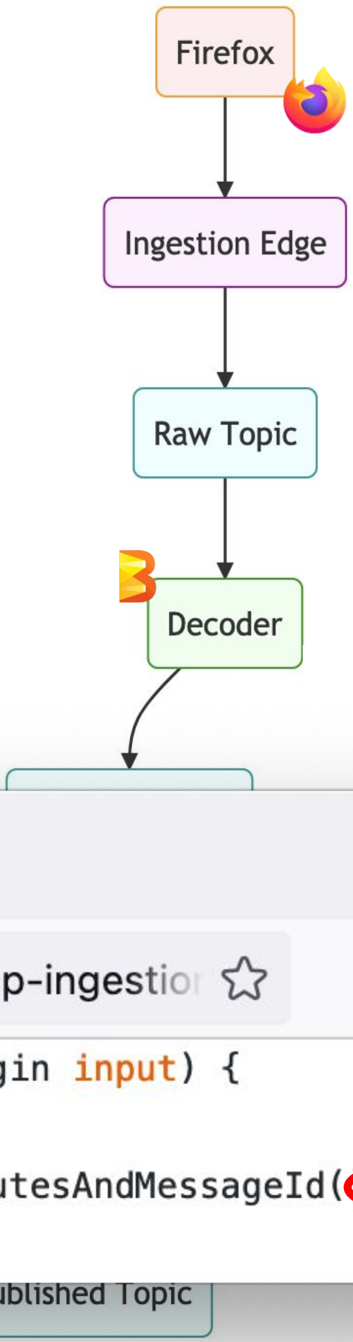

gcp-ingestion/Read.java at main
https://github.com/mozilla/gcp-ingestio

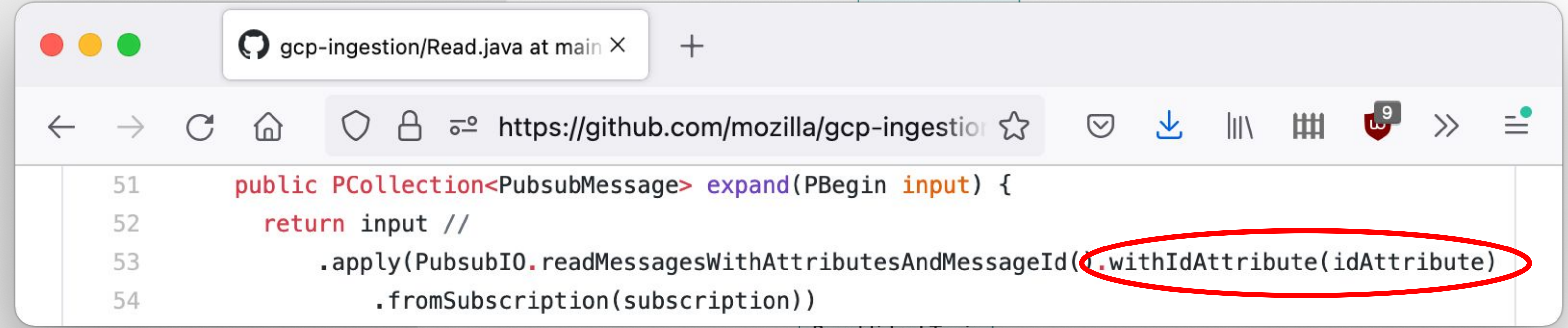
```java
  public PCollection<PubsubMessage> expand(PBegin input) {
    return input //
        .apply(PubsubIO.readMessagesWithAttributesAndMessageId()
            .fromSubscription(subscription))
```

Readers and PubsubIO



Readers and PubsubIO




gcp-ingestion/Read.java at main
https://github.com/mozilla/gcp-ingestio

```
    public PCollection<PubsubMessage> expand(PBegin input) {
      return input //
          .apply(PubsubIO.readMessagesWithAttributesAndMessageId().withIdAttribute(idAttribute)
              .fromSubscription(subscription))
```

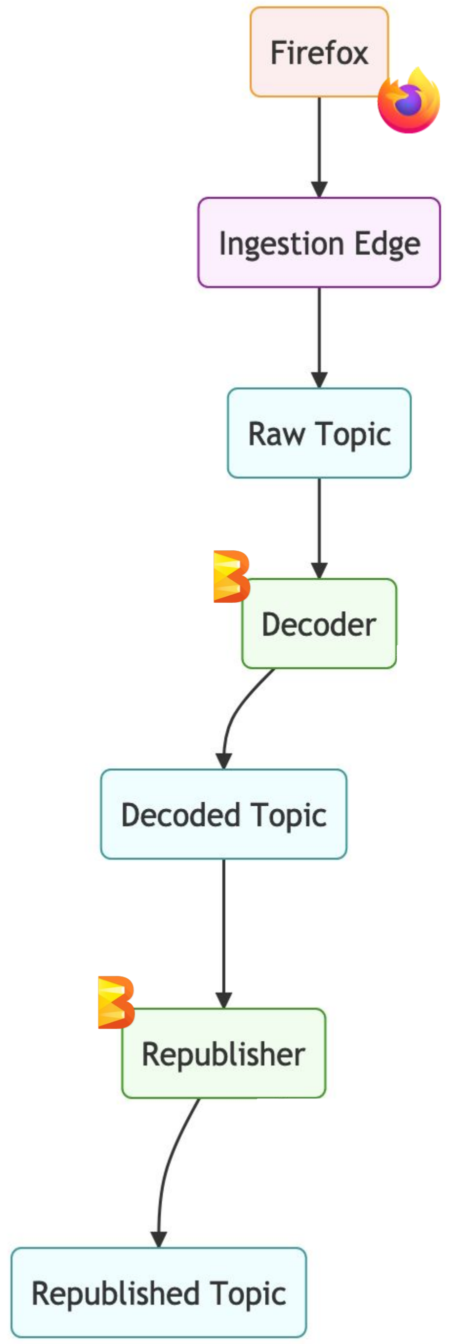
End-to-End Identifier

The Firefox telemetry API requires that the client include a randomly generated UUID as part of the URL for each document sent.

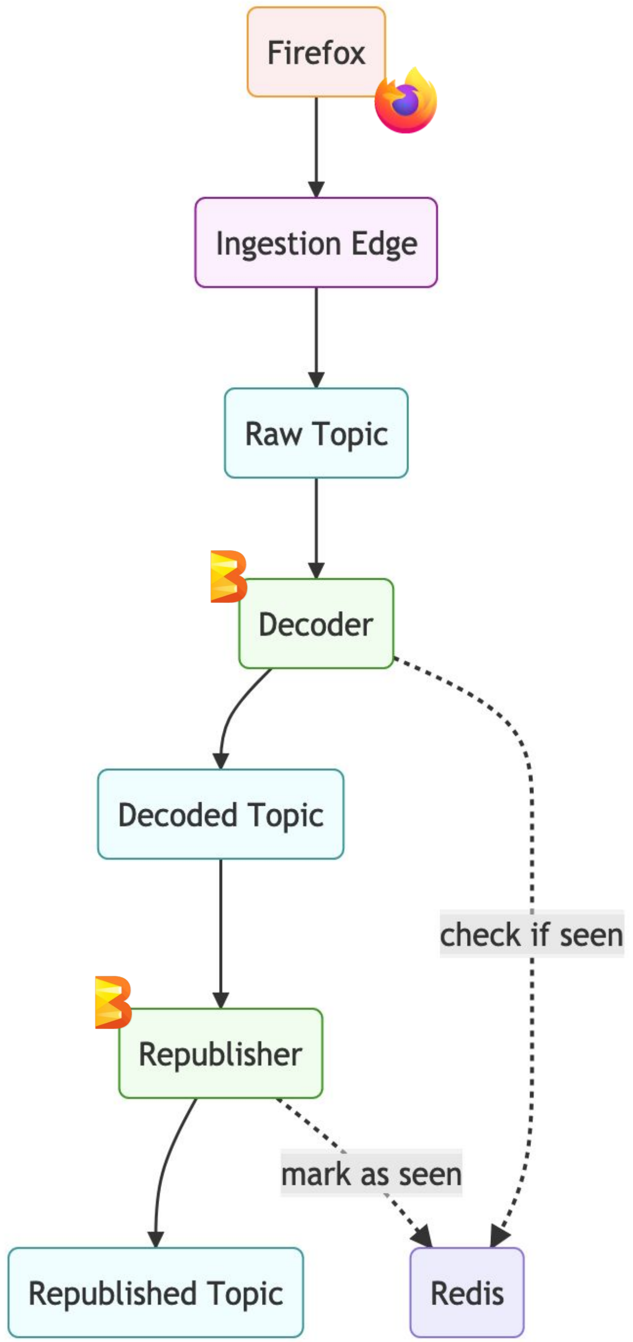
We call this *document_id* and it serves as an *end-to-end identifier* for the document.

```
/submit/<namespace>/<doctype>/<version>/<document_id>
/submit/eng-workflow/hgpush/1/2c3a0767-d84a-4d02-8a92-fa54a3376049
```

End-to-End Identifier



Externalizing State



See Redis-based deduplication code in the gcp-ingestion repo

Batch Deduplication

Our “stable” tables for historical analysis in BigQuery are populated once per day,

guaranteeing that each *document_id* is unique per table partition.

See full copy_deduplicate code in the mozilla/bigquery-etl repo

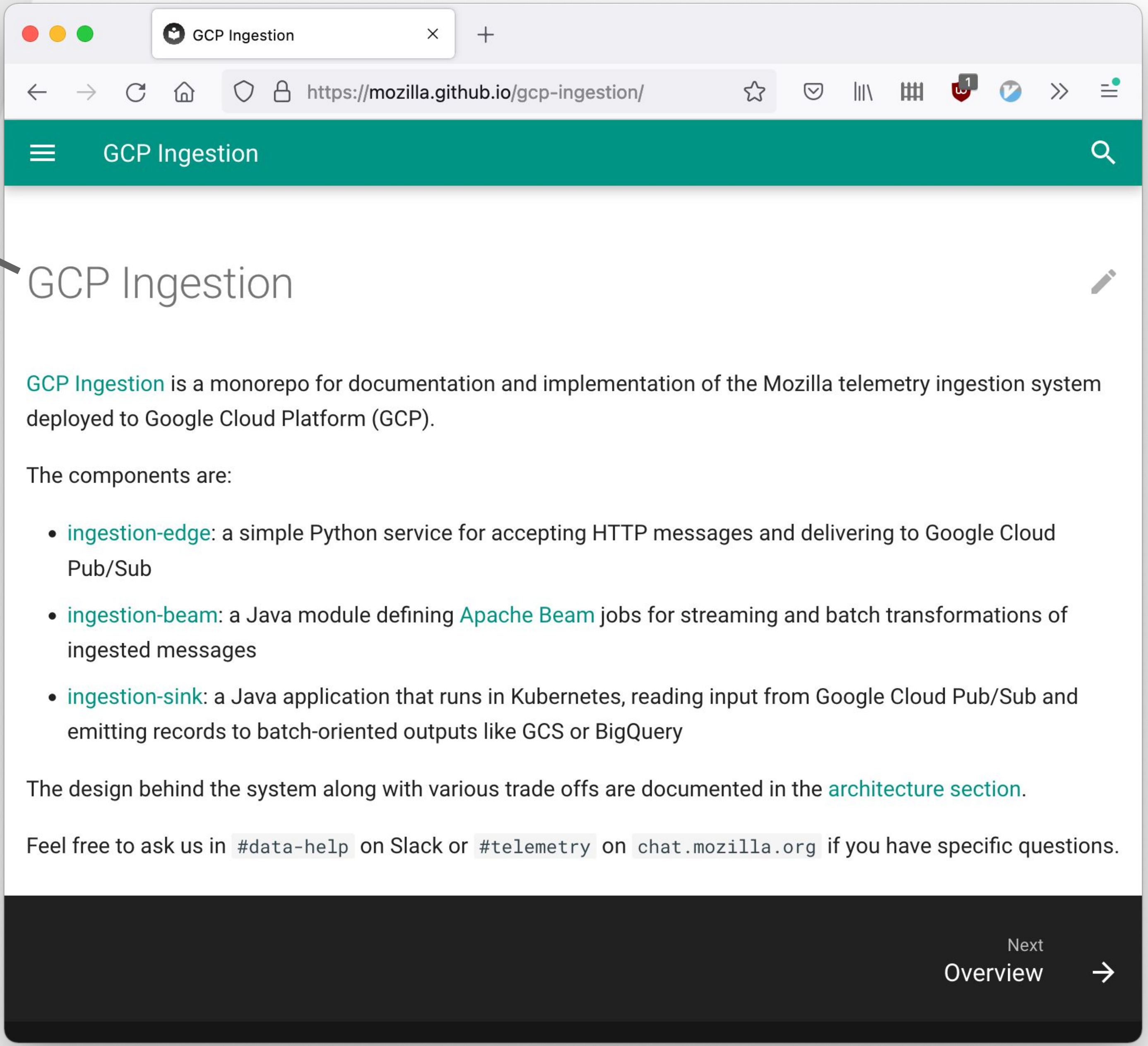
```
WITH
 numbered_duplicates AS (
 SELECT
   *,
   ROW_NUMBER() OVER (
     PARTITION BY document_id
     ORDER BY submission_timestamp) AS _n
 FROM
   live_table
 WHERE
   DATE(submission_timestamp) = @submission_date)

SELECT
 * EXCEPT(_n)
FROM
 numbered_duplicates
WHERE
 _n = 1
```

	Distinct.java	Deduplicate.java	Dataflow PubsubIO	External State	Batch
Time domain	Event	Processing	Processing	Processing	Event
Duration	Minutes	Minutes	10 min	Hours or days	Hours or days
Built-in	✅	✅	✅	❌	✅
No worker resource consumption	❌	❌	✅	✅	❓
Allows monitoring of duplicate rate	❌	❌	❌	✅	✅

https://github.com/mozilla/gcp-ingestion

Explore!



GCP Ingestion

GCP Ingestion is a monorepo for documentation and implementation of the Mozilla telemetry ingestion system deployed to Google Cloud Platform (GCP).

The components are:

- ingestion-edge: a simple Python service for accepting HTTP messages and delivering to Google Cloud Pub/Sub
- ingestion-beam: a Java module defining Apache Beam jobs for streaming and batch transformations of ingested messages
- ingestion-sink: a Java application that runs in Kubernetes, reading input from Google Cloud Pub/Sub and emitting records to batch-oriented outputs like GCS or BigQuery

The design behind the system along with various trade offs are documented in the architecture section.

Feel free to ask us in `#data-help` on Slack or `#telemetry` on `chat.mozilla.org` if you have specific questions.

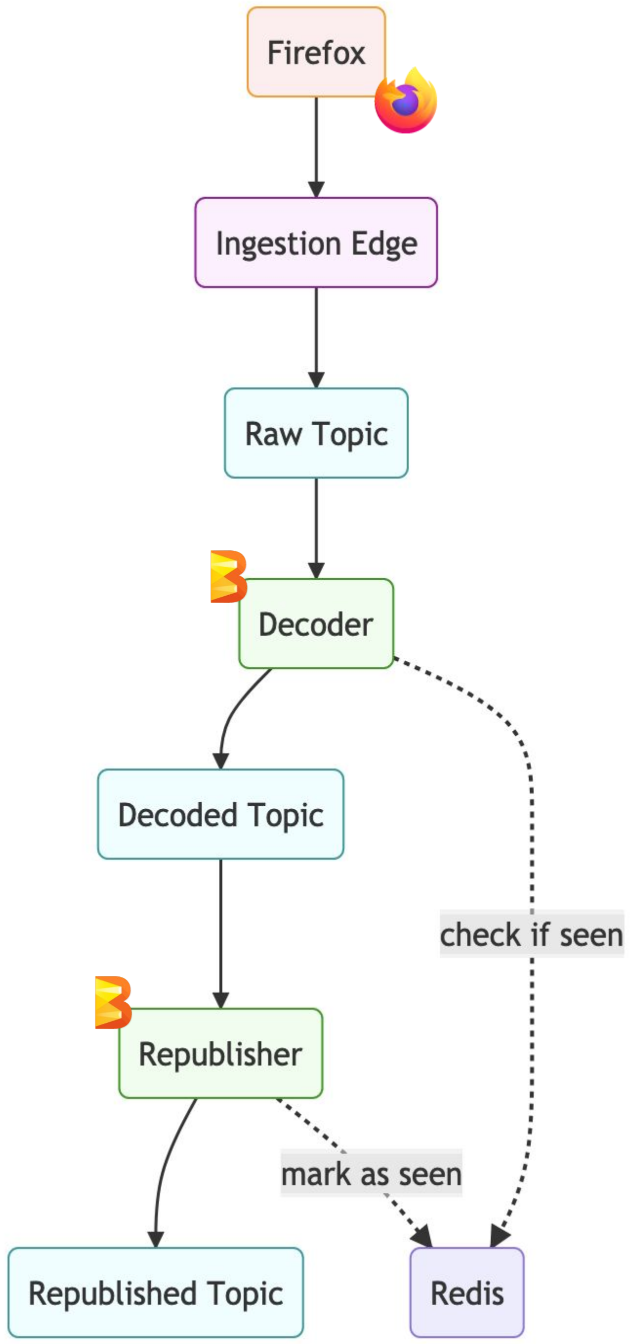
https://github.com/mozilla/gcp-ingestion

Thank you!

https://jeff.klukas.net

klukas on ASF Slack

jeff@klukas.net


Firefox
Ingestion Edge
Raw Topic
Decoder
Decoded Topic
check if seen
Republisher
mark as seen
Republished Topic
Redis