Create a git-like experience for Data Lake analytics

October 2021, Itai Admi

lakeFS

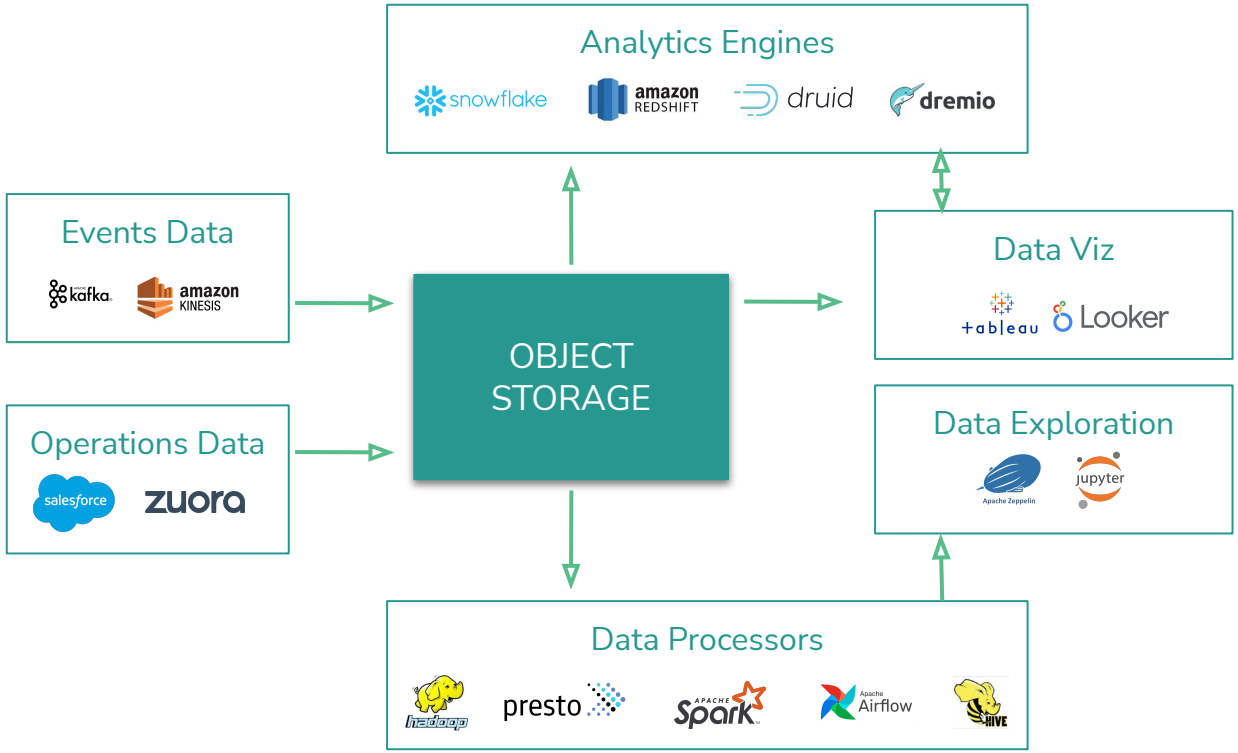
Itai Admi

R&D Team Lead

Treeverse

Tel Aviv, Israel

Data Lake Architecture **Reference**



The Data Lake **Advantage**

1. **Scalability and cost effectiveness**
2. **Accessibility and ease of use**
3. **High throughput**
4. **Rich application ecosystem**

The Data Lake **Challenge**

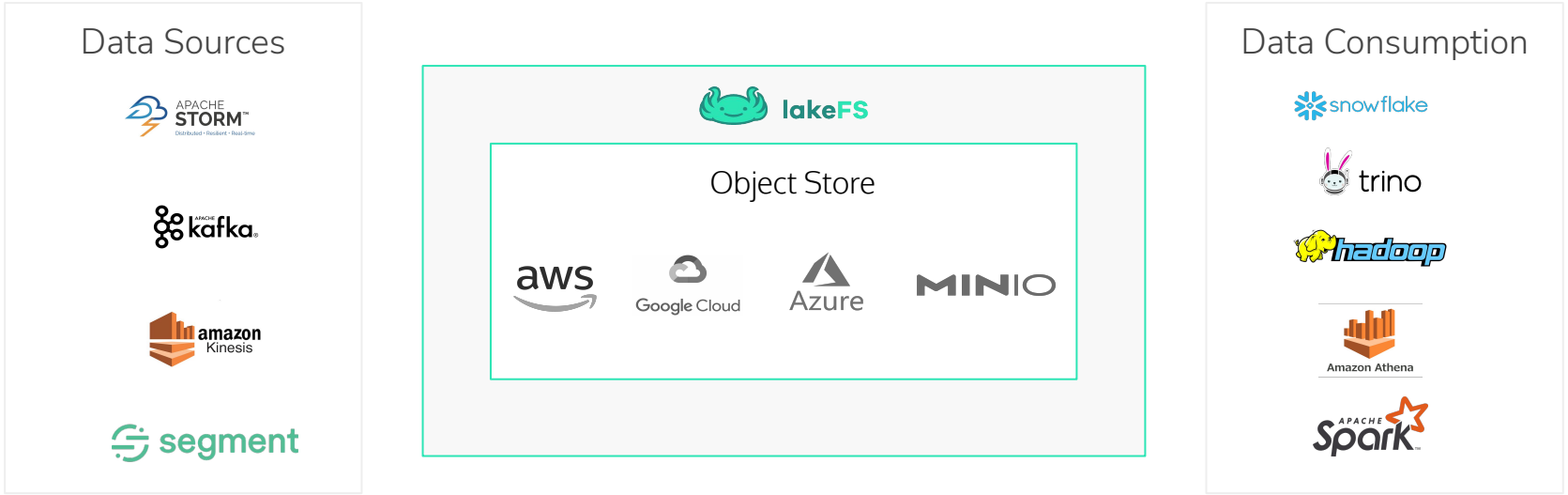
1. **In-ability to experiment, compare and reproduce**
   Example: add new metric for BI
2. **Difficult to enforce data best practices**
   Example: schema, format enforcement
3. **Hard to ensure high quality data**
   Example: validate statistical properties of the data

In a perfect world

we would manage data from dev to production the way we manage code

Atomic versioned data lake on top of object storage

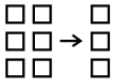
Manageability & Reliability Layer



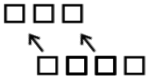
How it works

API Compatibility
with object stores

Versioning Engine
Transactional, atomic, isolated



Git Terminology
Branch, commit, merge



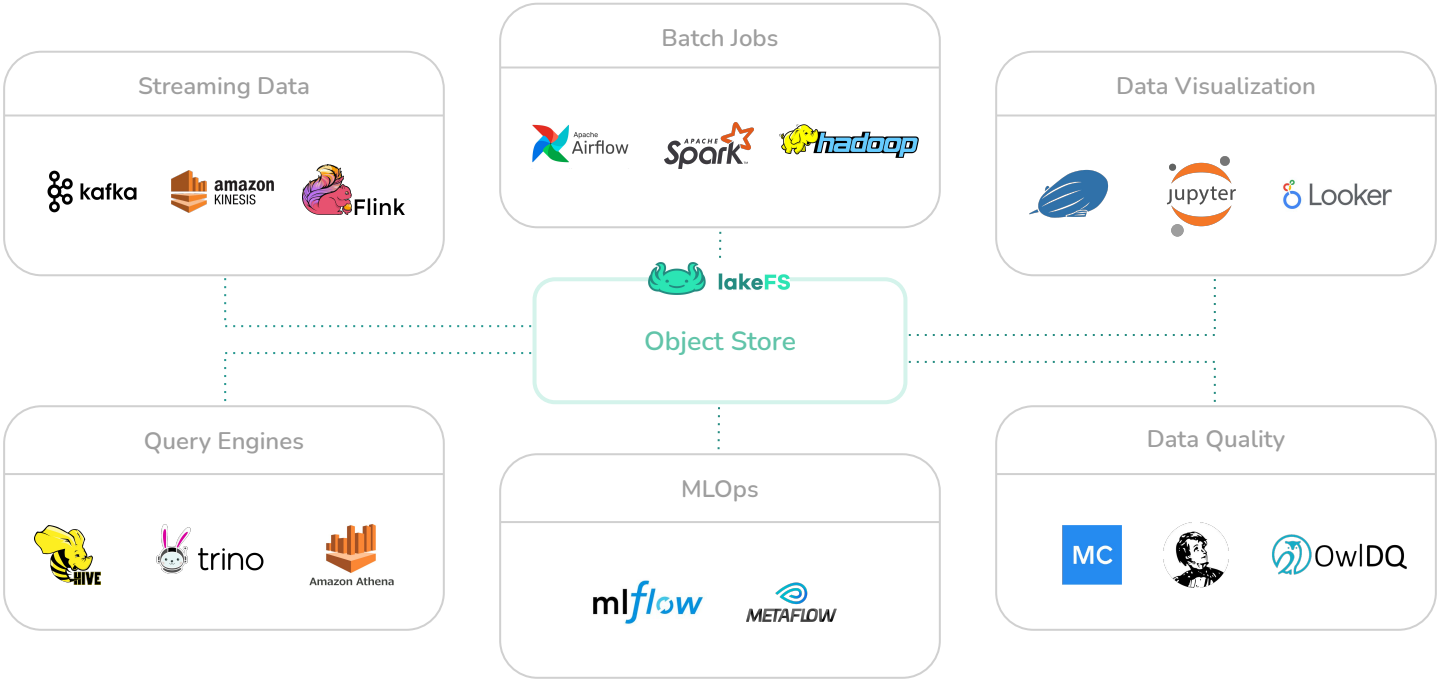
How it works

s3://data-bucket/collections/foo

s3://data-bucket/main/collections/foo

```
# Will create an external table using the same partitions and configurations
# but pointing at paths under the new branch
Lakectl metastore copy \
       --from-table user_events \
       --to-branch dev-experiment-1
```

Integrates with your existing tools

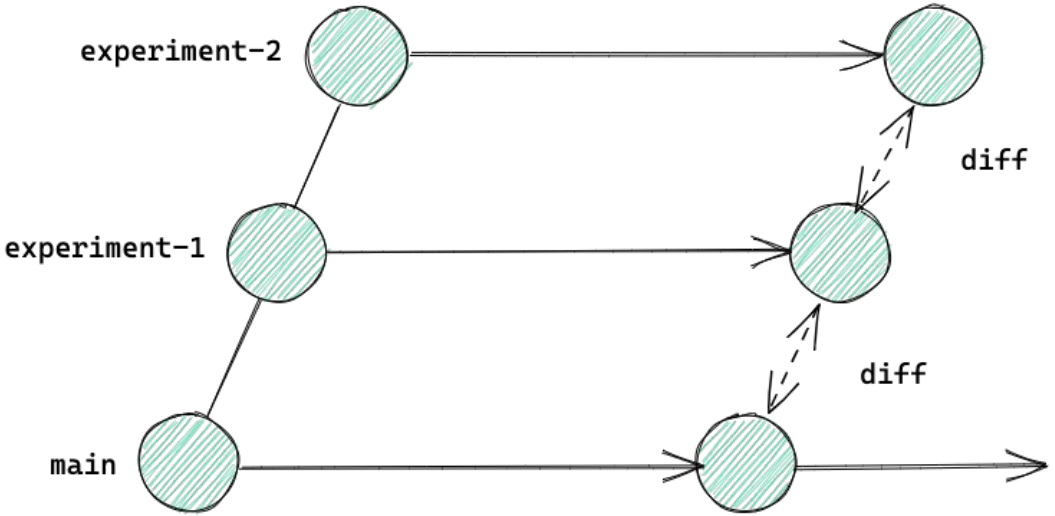


lakeFS can be used in any stage of your data development lifecycle

In Development

- **Experiment -** try new tools, upgrade versions, and evaluate code changes in isolation.
- **Debug -** checkout specific commits in a repository's commit history to materialize consistent, historical versions of your data.
- **Collaborate -** leverage isolated branches managed by metadata (not copies of files) to work in parallel.

Experiment Safely



Experimenting with Spark

```
lakectl branch create \
   lakefs://example-repo@testing-spark-3 \
   --source lakefs://example-repo@main
# output:
# created branch 'testing-spark-3.0', pointing to commit ID: 'd1e9adc71c10a'
```

```
val dfExperiment1 = sc.read.parquet("s3a://example-repo/experiment-1/events/by-date")
val dfExperiment2 = sc.read.parquet("s3a://example-repo/experiment-2/events/by-date")

dfExperiment1.groupBy(" ... ").count()
dfExperiment2.groupBy(" ... ").count() // now we can compare the properties of the data itself
```

Experimenting with Presto

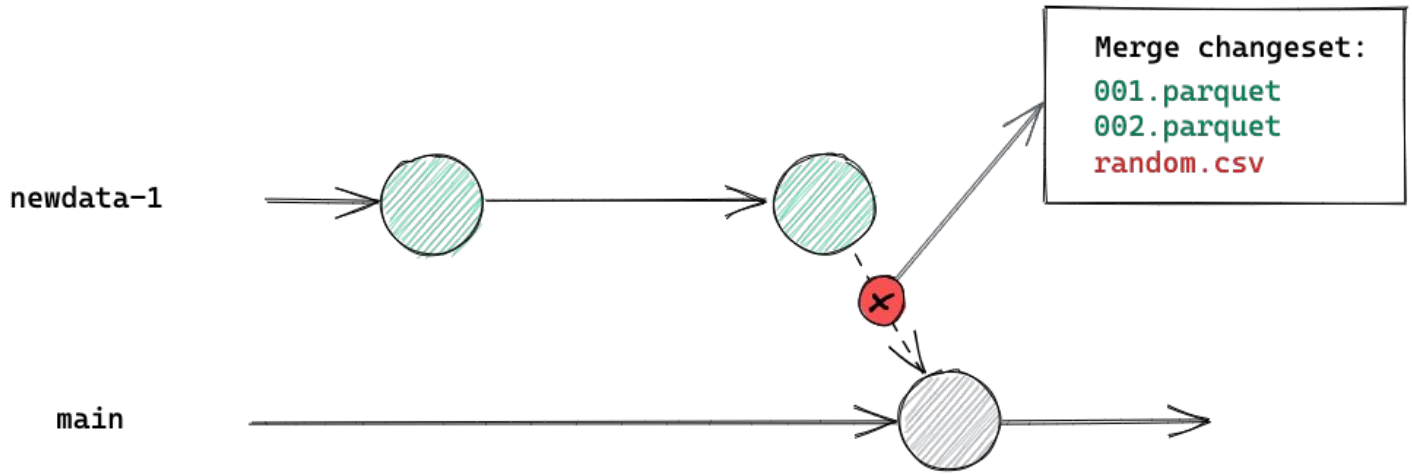
```
lakectl branch create \
   lakefs://example-repo@testing-spark-3 \
   --source lakefs://example-repo@main
# output:
# created branch 'testing-spark-3.0', pointing to commit ID: 'd1e9adc71c10a'
```

```
CREATE TABLE master.request_logs (
  request_time timestamp,
  url varchar,
  ip varchar,
  user_agent varchar
)
WITH (
  format = 'TEXTFILE',
  external_location = 's3a://example/main/data/logs/'
);
```

During Deployment

- **Data update safety -** Ingest new data onto an isolated branch, perform data validations, then add to production through a merge operation.
- **Test -** define pre-merge and pre-commit hooks to run tests that enforce schema and validate properties of the data to catch issues before they reach production.

Enforce Best Practices



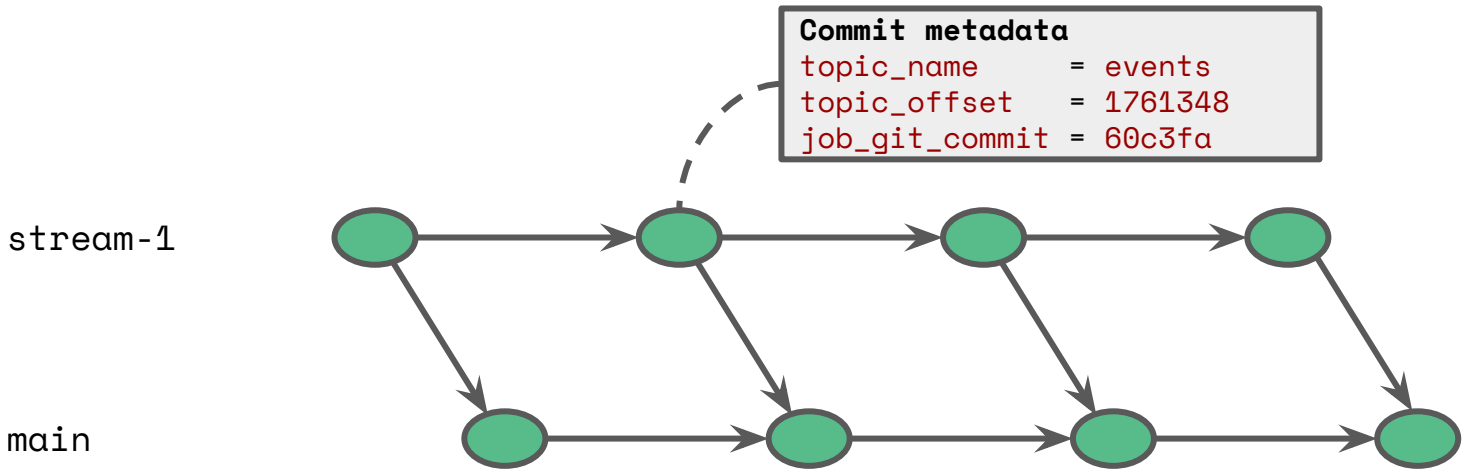
CI with PyArrow

```
import lakefs
import pyarrow.parquet as pq

# Setup a PyArrow FileSystem that we can use to query data in the source ref
fs = lakefs.get_filesystem(repo, source_ref)

for change in lakefs.diff(repo, source_ref, target_branch, prefix='public/'):
    if not change.path.endswith('.parquet'):
        continue
    # Read Parquet column metadata
    schema = pq.read_schema(fs.open_input_file(change.path))
    if filter(lambda column: column.name == 'user_id', schema):
        raise ValidationError('user_id column not allowed')
```

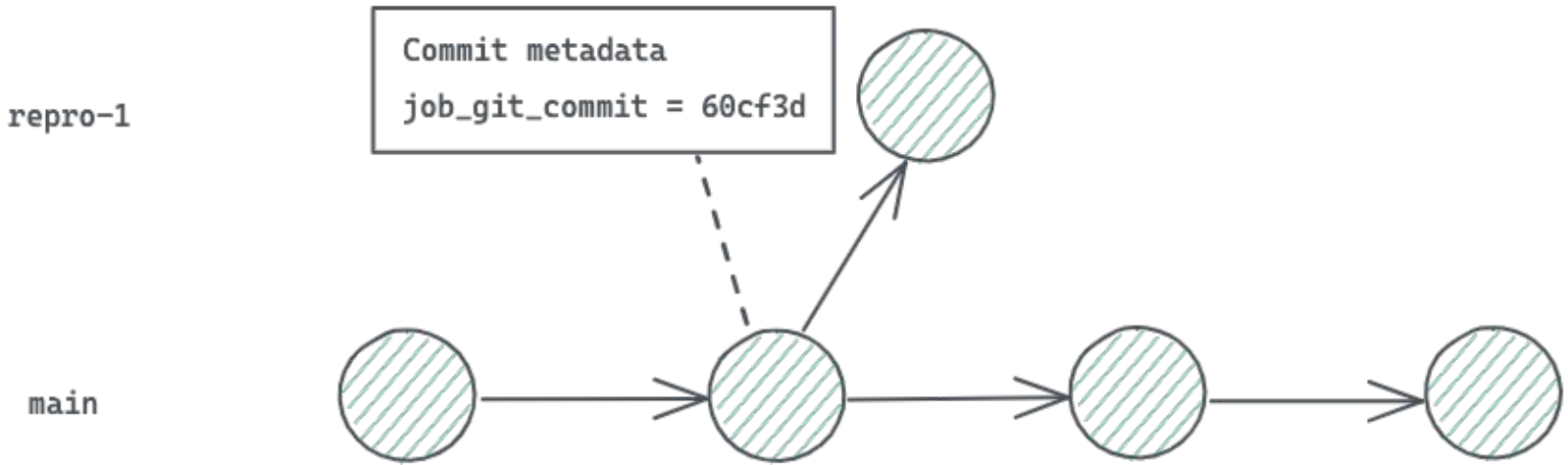
Streaming Data with Kafka Connect



In Production

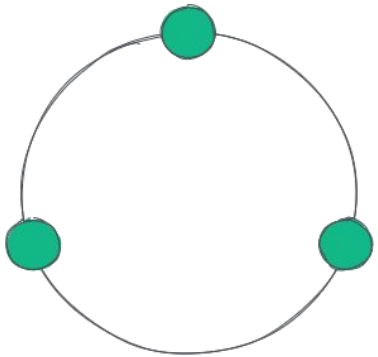
- **Roll Back -** recover from errors by instantly reverting data to a former, consistent snapshot of the data lake.
- **Troubleshoot -** investigate production errors by starting with a snapshot of the inputs to the failed process.
- **Cross-collection Consistency -** provide consumers multiple synchronized collections of data in one atomic, revertable action.

Troubleshoot - Reproduce a bug in production



Simplify workflows at each step of the data lifecycle

Development
test new tools, upgrade versions, and evaluate in isolation



Deployment
enforce best practices to catch changes before they reach production

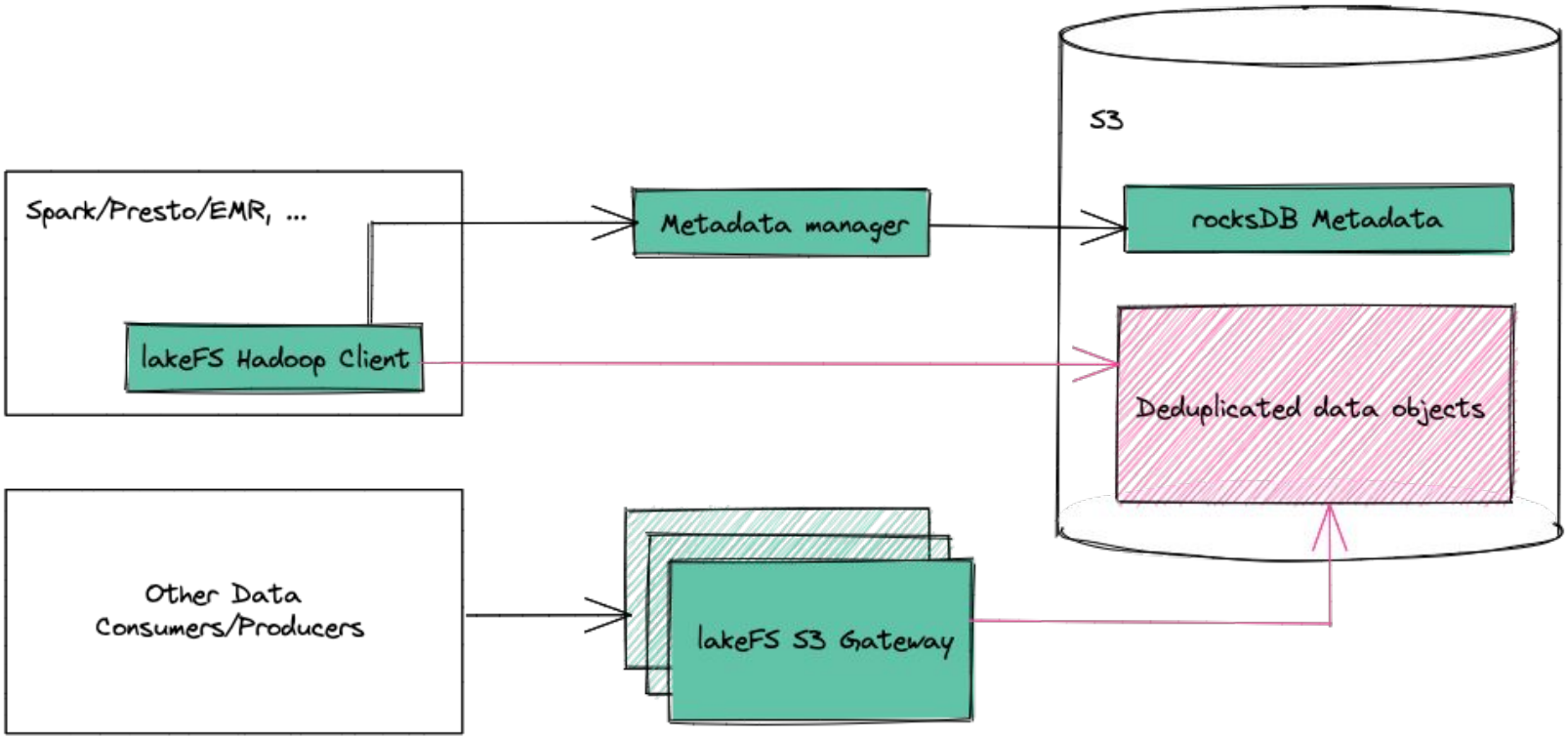
Production
instantly recover from errors by reverting data

lakeFS Architecture

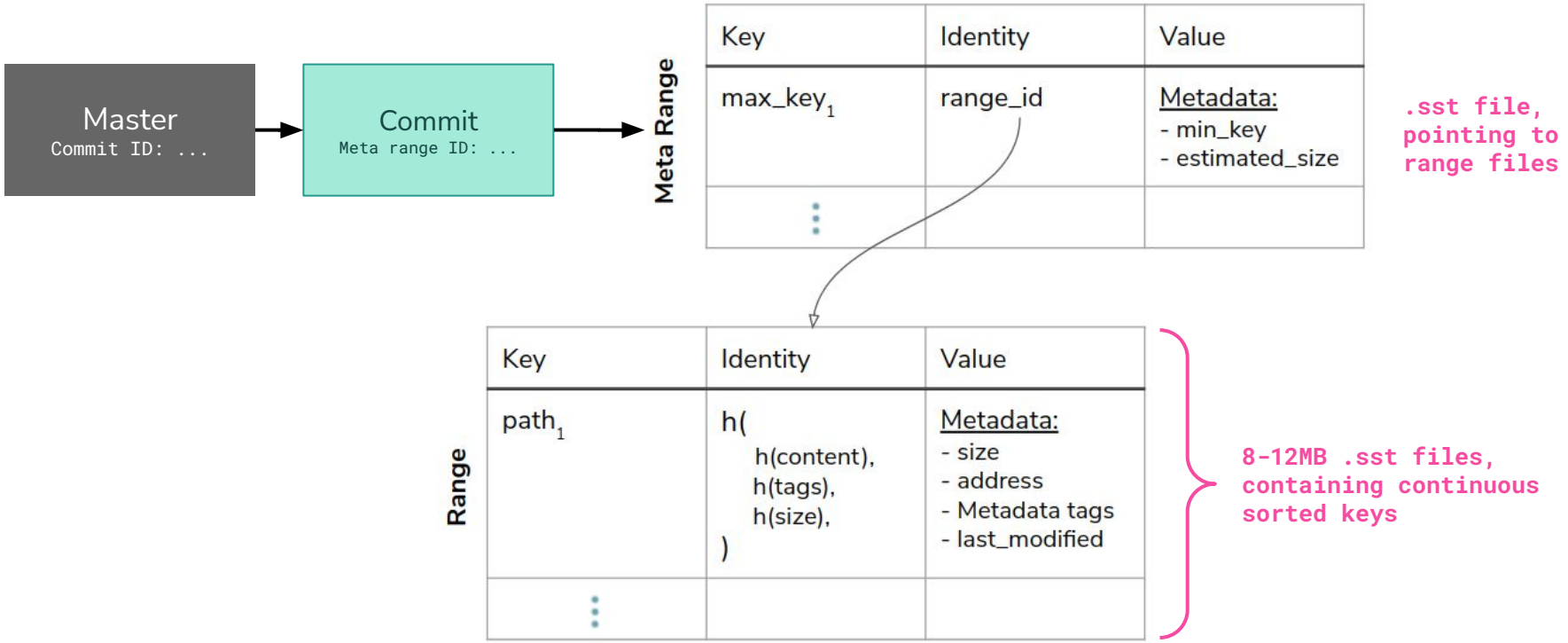
Data != code

	Code	Data Lake
Amount of files tracked	Thousands-3.5 million	Millions-billions
File format	Mutable text files	Immutable binary data files, unstructured data
Change velocity	Hundreds of changes/day	Potentially millions/day
Change resolution	Lines in files	files
Mode of operation	Clone locally, push changes	Unless you're rich, you work remotely

Architecture



Data Format



Further Reading

Demo

Additional Resources

Getting started

Check out the docs

Join the lakeFS Slack Channel

Contribute and star the repo

Thanks!

lakeFS

Join the community

GitHub

einat.orr@treeverse.io