



# Data as Code with Dremio Arctic

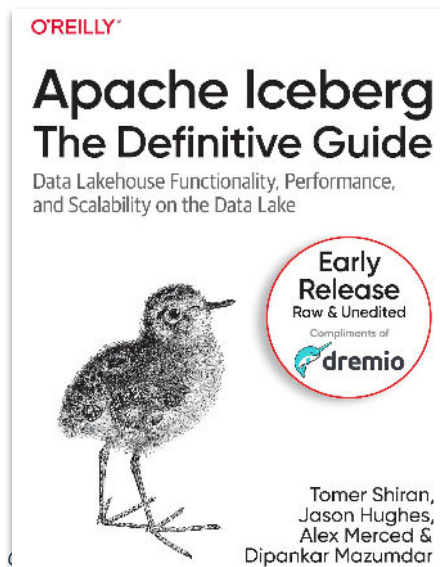


ML experimentation & reproducibility on Lakehouse

# About Me



- **Current:** Developer Advocate @Dremio
- **Open source:** Apache Iceberg, Arrow, Project Nessie
- **Past:** BI, Machine Learning, Data architecture



# Agenda

1 ML Experimentation & Reproducibility

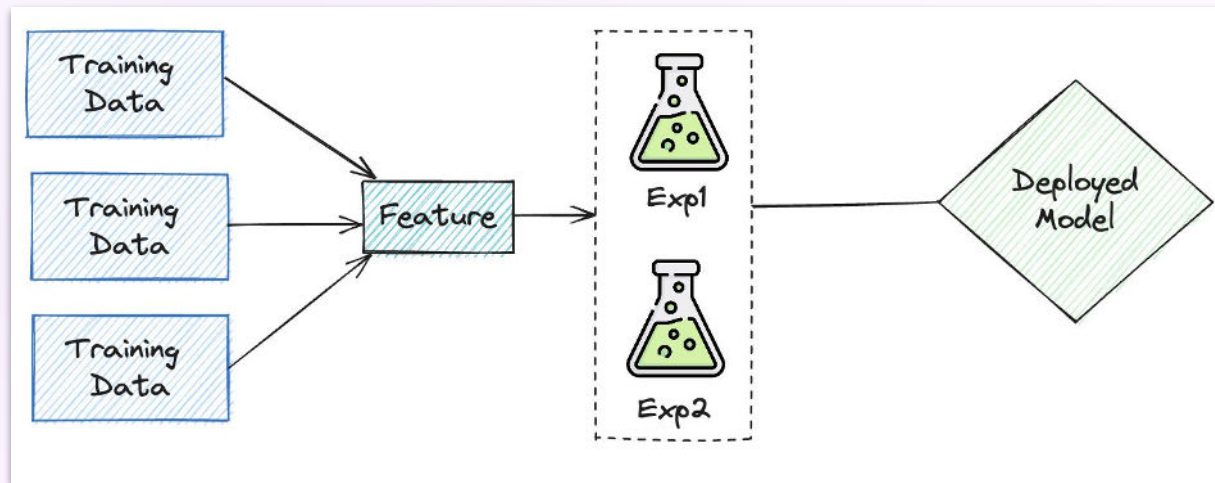
2 Dremio Arctic: data-as-code

3 Creating Branches for ML experiments

4 Using Tags for ML reproducibility

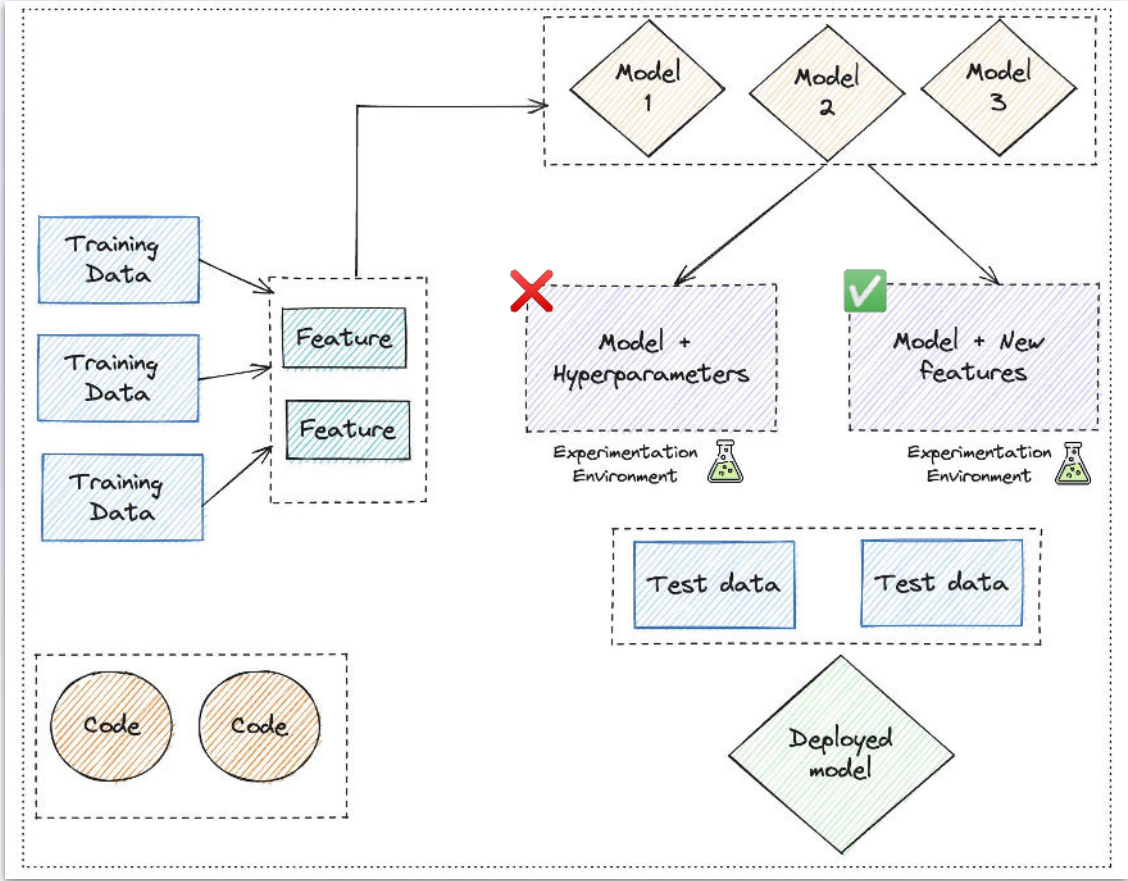
5 Advantages of using data-as-code

# Software Eng Vs MLE



Unlike Software Engineering, Machine Learning is more of an **empirical** procedure to **test** a hypothesis

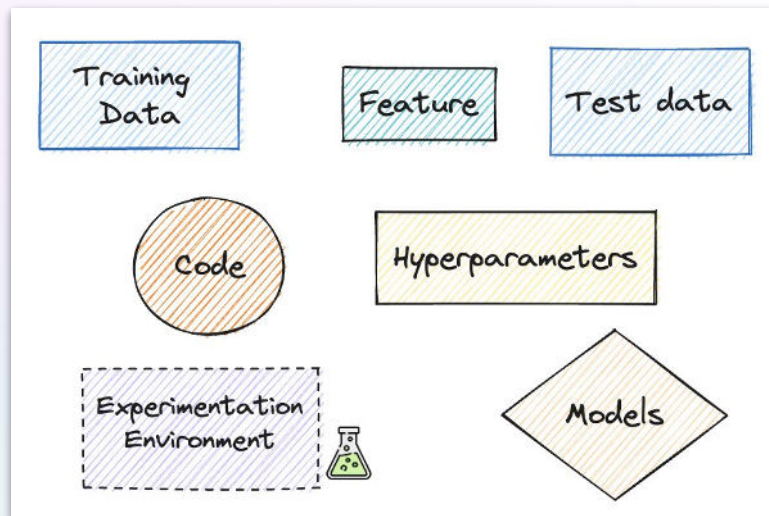
# Experiments in Machine Learning



# Problems

Too many variables:

- Models
- Different segments of train & test sets
- Hyperparameters of the model
- Different Features
- Environment
- Code to train/test models



# Problems

## EXPERIMENTATION

*“We developed a highly accurate model after a series of experiments, but have a tough time figuring out what resulted in that accuracy as we used different datasets with varying features in different environments”*



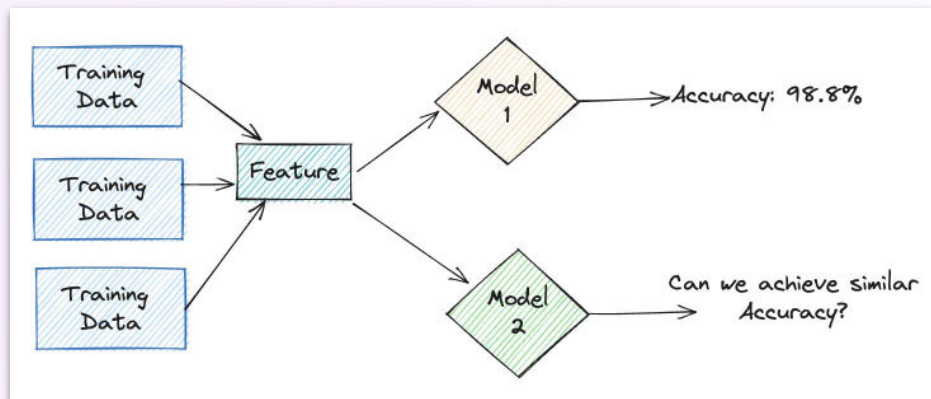
# Solution



Provide flexibility by tracking:  
**Data** and **Model** artifacts



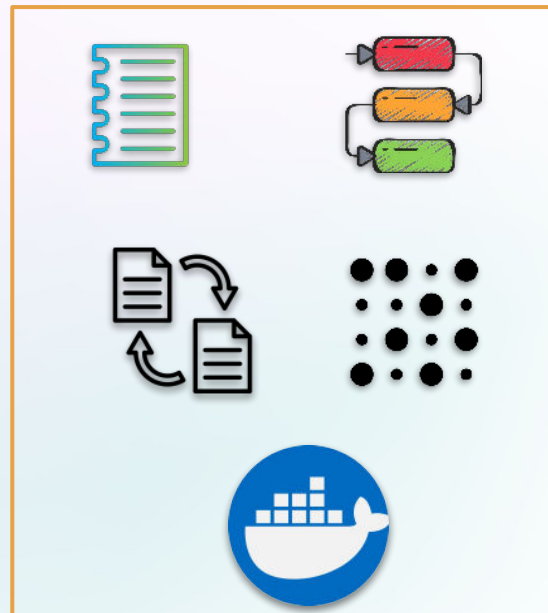
# Machine Learning Reproducibility



- Recreate & achieve consistent results as the original model across different setups
- Involves same training dataset, code, environment
- Important for ML research, building reliable ML systems in industry

# Problems

- Lack of documentation when building models
- Dependency Management - versions, frameworks
- Dataset changes - newly inserted records, etc.
- Randomness
- Environment - Hardware, Platform differences



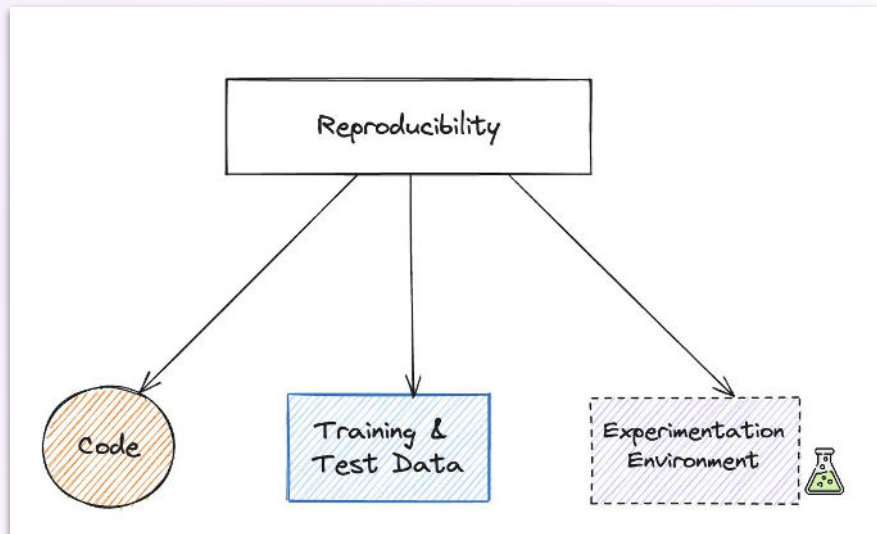
# Problems

## REPRODUCIBILITY

*"We are trying to achieve similar precision & accuracy using the same model & code released by our ML Research team on a similar dataset. We can't seem to replicate"*



# Solution



To reproduce we need to track **Code**, **Data** and **Environment**

# Our focus will be on Data

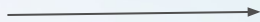
Code



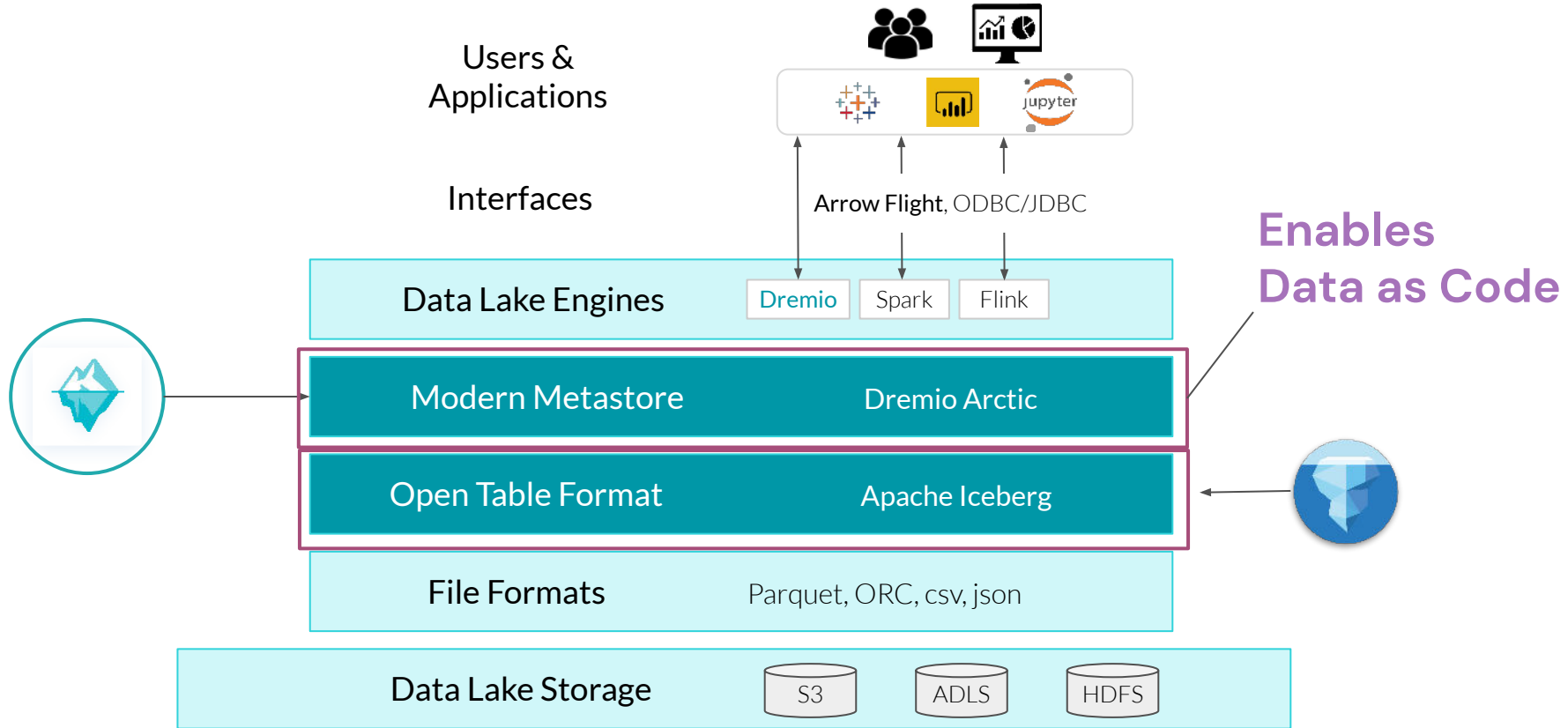
Model Artifacts



Data



# How do we support these in a Data Lakehouse?



# GitHub in Software Engineering

The screenshot displays the GitHub interface for the 'apache/arrow' repository. Key features visible include:

- Repository Overview:** Shows 36 branches, 60 tags, 12,357 commits, 2.5k forks, and 10.4k stars.
- Code Storage:** A 'Switch branches/tags' dropdown menu is open, showing a list of branches like 'master', 'develop', and 'release'.
- Commit History:** A list of recent commits with details like 'MINOR: [Dev] Fix actions/labeler's tap' and 'ARROW-17814: [C++] Remove make\_unique reimplementation'.
- Issues and Pull Requests:** A summary bar indicates 84 Active pull requests, 7 Active issues, 53 Merged pull requests, 31 Open pull requests, 4 Closed issues, and 3 New issues.
- Community Standards:** A sidebar menu includes options for 'Community Standards', 'Commits', 'Code frequency', 'Dependency graph', 'Network', and 'Forks'.
- Releases:** A section for releases with a 'Period: 1 week' filter.
- Activity Graph:** A bar chart showing commit activity over time, with a note: 'Excluding merges, 24 authors have pushed 53 commits to master and 53 commits to all branches. On master, 301 files have changed and there have been 4,682 additions and 2,167 deletions.'

✓ Code storage

✓ Atomic changes

✓ Documentation

✓ Access old versions

✓ Recover from mistakes

✓ Trace history

✓ Isolated development

✓ CI/CD

✓ Collaboration

# Don't we already do that in software development though?

Data teams gain the same capabilities that took application development market by storm...

- 1 Consistency
- 2 Isolation
- 3 Experimentation
- 4 Pre-prod verification

... through the same functionality GitHub enabled for developers

- **Commits** – Atomic transactions affecting multiple tables/views at once
- **Branches** – Free to create (virtual) copies of entire data lake for development, verification or experimentation
- **Tags** – Immutable reference used to permanently record a specific state of the Data Lake
- **Merges** – Migrate changes from development, ETL or experimentation from one branch to another.

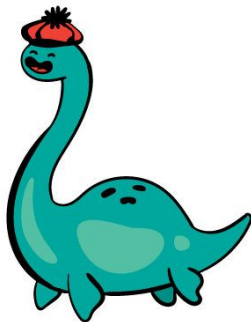


# Nessie: A Git-like Experience

## Data Branching

- Multi-table consistency/transactions
- Experimentation (isolated dev/test)
- Pre-prod verification (stage->prod)

```
CREATE BRANCH etl
[Kafka] Data Ingestion
[Spark] Transformation 1
[Spark] Transformation 2
[Dremio] Reflection Refresh 1
[Dremio] Reflection Refresh 2
USE BRANCH main
MERGE BRANCH etl
```

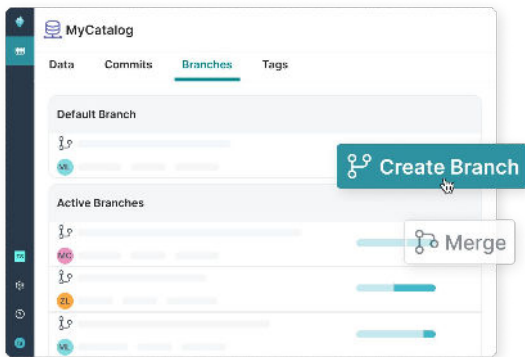


## Data Version Control

- Reproducibility
- Compliance
- Historical comparisons

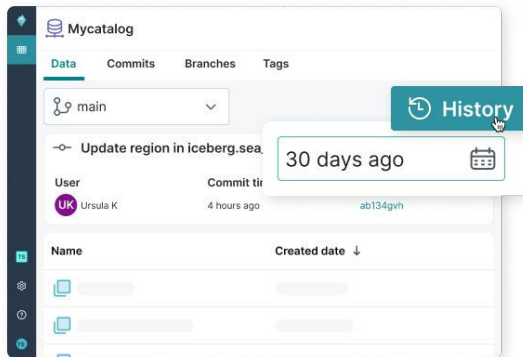
```
USE BRANCH 'main'
SELECT * FROM t1 // main implicit
SELECT * FROM t1@etl
SELECT * FROM t1 AT '2020-10-26'
SELECT * FROM t1@etl AT '2020-10-26'
```

# Dremio Arctic enables Data as Code (DaC)



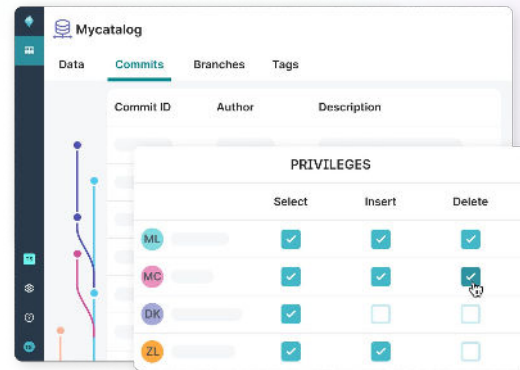
## ISOLATION

- Experiment with data without impacting other users
- Ingest, transform and test data before exposing it to other users in an atomic merge



## VERSION CONTROL

- Reproduce models and dashboards from historical data based on time or tags
- Recover from any mistake by instantly undoing accidental data or metadata changes

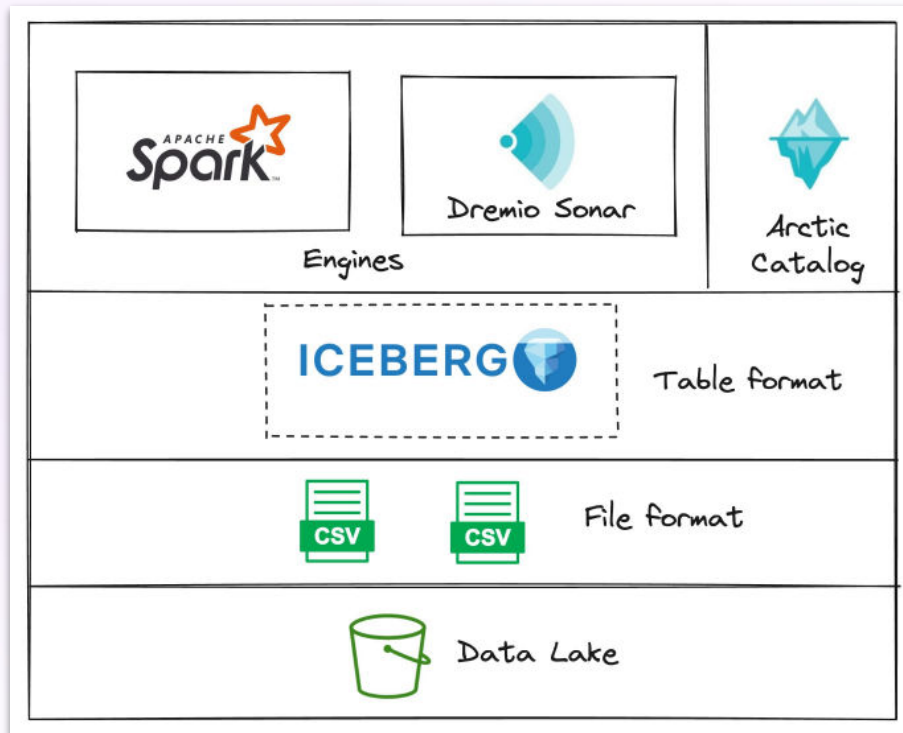


## GOVERNANCE

- All changes to the data and metadata are tracked: who accessed what data and when
- Fine-grained privileges to control access to the data at the table, column and row level

# Live Tour & Demo!

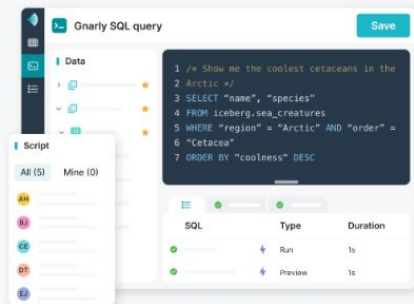
# Setup for Demo



Lakehouse Architecture for  
ML Experiments & Reproducibility

# Sign-up for Dremio Cloud

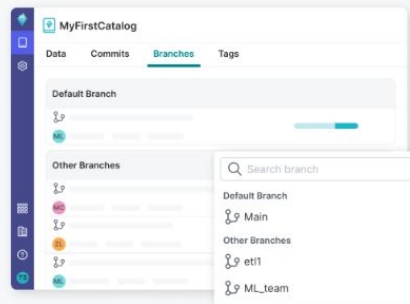
## Services



**Sonar**

A lakehouse query engine that allows analysts to explore data with sub-second query response times. [Learn more.](#)

[+ Add Sonar Project](#)



**Arctic (Preview)**

A catalog for Iceberg tables that enables data to be managed like code with Git-like capabilities. [Learn more.](#)

[+ Add Arctic Catalog](#)

# Connect to Data Lake



dremio.demo / Sonar / My First Project / Datasets

**My First Project** Dipankar.dm-iceberg.datasets

dipankar.mazumdar@dremio.com

Spaces (1) +

- Marketing 0

Sources +

Data-as-code (3)

- arctic main >
- arctic2 main >
- myarctic main >

Metastores (2)

- gluecatalog 11
- gluecatalog2 11

Object Storage (4) **Data Lake S3**

- awss3 3
- Dipankar** 2

**Datasets**

Name ↑

- articles.csv
- churn-bigml-20\_allfeat\_Oct\_train\_data.csv**
- churn-bigml-20\_allfeat.csv**
- customers.csv
- HandM.parquet
- NYC\_taxi.csv
- transactions\_train.csv

# Create Iceberg tables using Dremio

The screenshot displays the Dremio interface for a project named "My First Project". On the left, a navigation sidebar shows the user profile "dipankar.mazumdar@dremio.com", a "Spaces (1)" section with a "Marketing" space containing 0 items, and a "Sources" section with "Data-as-code (3)" items. The "arctic" source is selected, and the "main" workspace is active. On the right, a table listing shows several tables under the heading "Name ↑". The tables listed are "churn", "churn\_allfeat", "churn\_nov\_data", and "churn\_oct\_data". The "churn" and "churn\_oct\_data" tables are highlighted with orange rounded rectangles.

My First Project		arctic.telco <a href="#">main &gt;</a>	
dipankar.mazumdar@dremio.com		Name ↑	
▼ Spaces (1)		churn	
Marketing 0		churn_allfeat	
Sources		churn_nov_data	
▼ Data-as-code (3)		churn_oct_data	
arctic <a href="#">main &gt;</a>			

# Spark + Arctic + Iceberg Setup

```
conf.set(
    "spark.jars.packages",
    f"org.apache.iceberg:iceberg-spark-runtime-3.3_2.12:1.2.0,org.projectnessie:nessie-spark-
    extensions-3.3_2.12:0.54.0,software.amazon.awssdk:bundle:2.17.178,software.amazon.awssdk:url-
    connection-client:2.17.178",
)

# create catalog named arctic as an iceberg catalog
conf.set("spark.sql.catalog.arctic", "org.apache.iceberg.spark.SparkCatalog")

# tell the catalog that its a Nessie catalog
conf.set("spark.sql.catalog.arctic.catalog-impl", "org.apache.iceberg.nessie.NessieCatalog")

# set the location for the catalog to store data. Spark writes to this directory
conf.set("spark.sql.catalog.arctic.warehouse", "s3://myrepo")
```





# Spark + Arctic + Iceberg Setup

```

# set the location of the Nessie/Arctic server.
conf.set("spark.sql.catalog.arctic.uri", "https://nessie.dremio.cloud/v1/repositories/xyz")

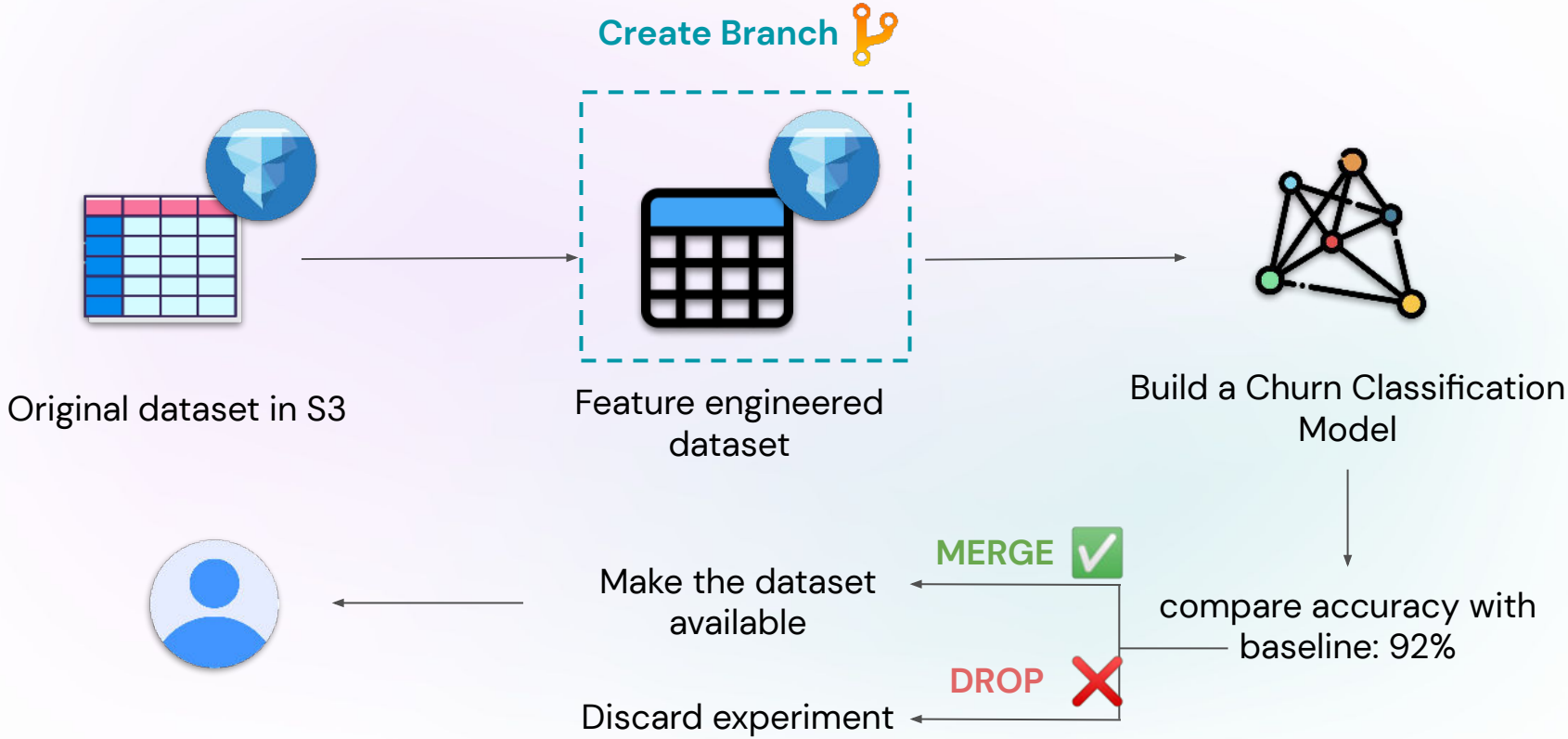
# Authentication mechanism. Here, we use AWS with BEARER
conf.set("spark.sql.catalog.arctic.authentication.type", "BEARER")
conf.set("spark.sql.catalog.arctic.authentication.token", token)

# enable the extensions for both Nessie and Iceberg
conf.set(
    "spark.sql.extensions",
    "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,org.projectnessie.spark.extensions.NessieSparkSessionExtensions",
)

```



# Use Case 1: ML Experimentation

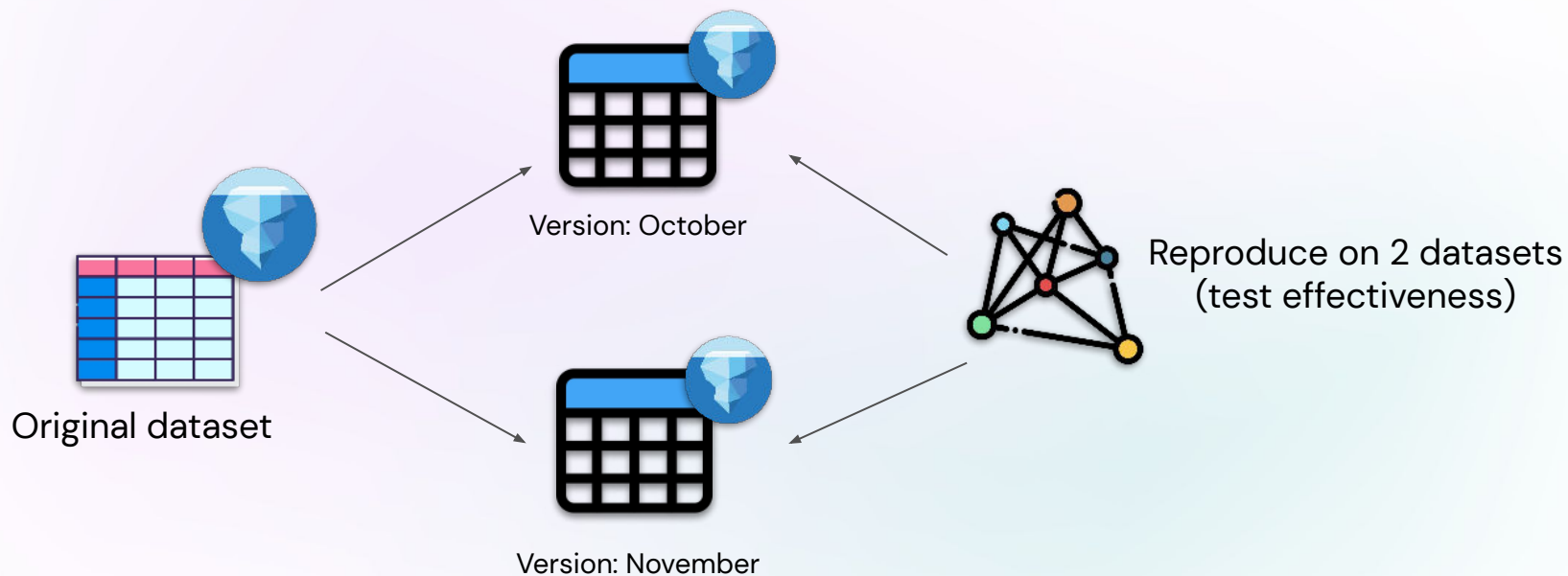


# Values Gained



- Greater flexibility in experimentation with the **isolated** branches feature
- Ability to work with data without needing extra **copies** for each and every experiment.
- Establish the viability of the dataset to ensure the data is **good** enough for the model

# Use Case 2: ML Reproducibility



# Values Gained



- Understand the **effectiveness** of our model for further deployment in other environments.
- Arctic provides an easy way to create **tagged** datasets & keep track of data **versions**

Thank you!