komodohealth™

Scalable Data Ingestion Architecture Using Airflow and Spark

April 17, 2019
Data Council
San Francisco, CA

Johannes Leppä
Data Engineer
johannes.leppa@komodohealth.com

Agenda

- ❖ Komodo Health
- ❖ Data Ingestion Challenges
- ❖ Data Ingestion System Architecture
- ❖ Lessons Learned and Future Developments
- ❖ Scaling Processes
- ❖ Conclusions

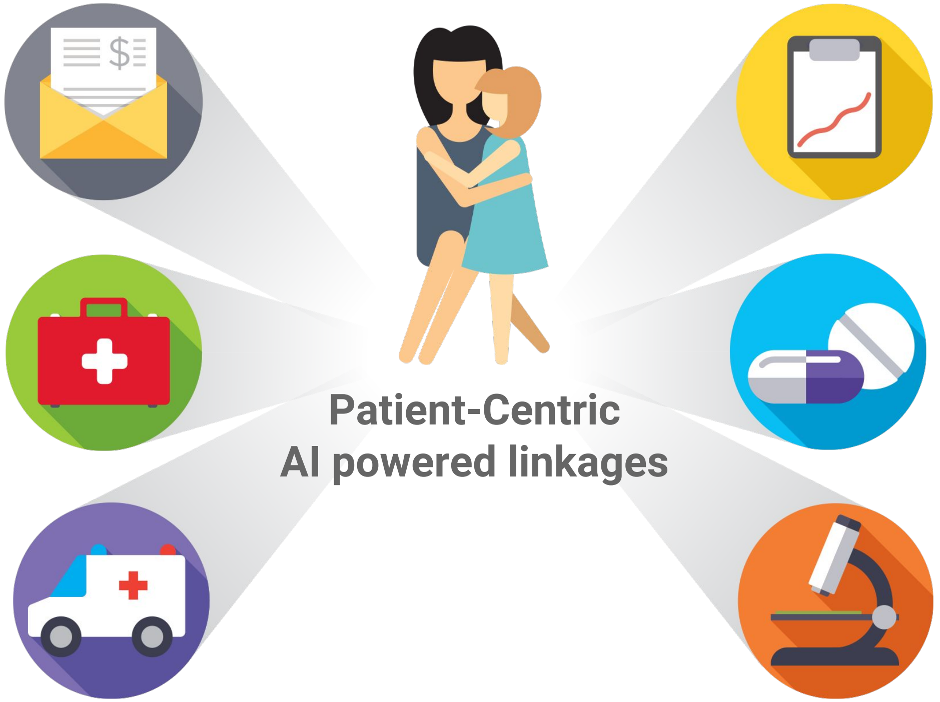
Our Mission

To reduce the global burden of disease through the most actionable healthcare map

komodohealth™

Komodo Health™ Integrity

Our Map Links Activities of the Entire Healthcare System



Payers
- 500+ payers

Providers
- 3.5 M doctors / nurses

Institutions
- 450K hospitals / clinics

Patient-Centric
AI powered linkages

Biopharma
- $20B payments

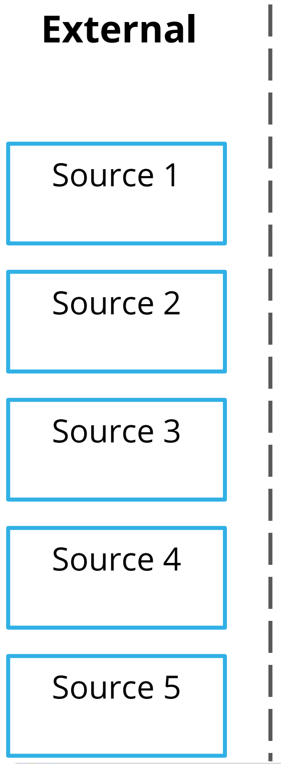
Clinical Trials
- 100k+ Clinical Trials

Scientific Publications
- 20M publications

Agenda

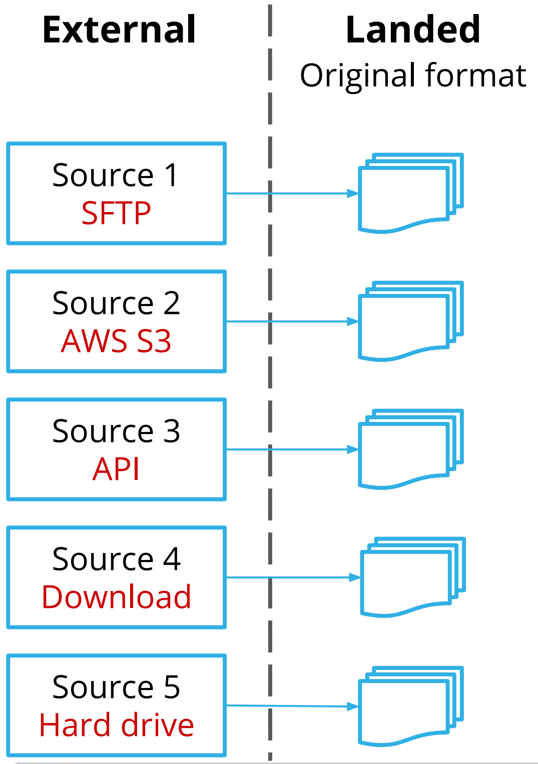
- ❖ Komodo Health
- ❖ Data Ingestion Challenges
- ❖ Data Ingestion System Architecture
- ❖ Lessons Learned and Future Developments
- ❖ Scaling Processes
- ❖ Conclusions

Variation in data size and cadency



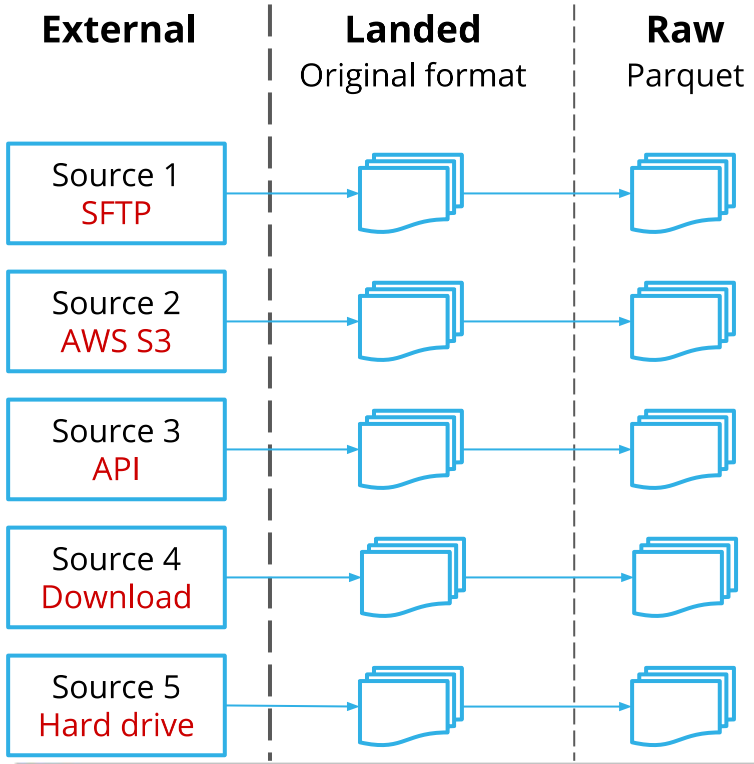
- Public and proprietary sources
- Size of data
  - From MBs to TBs
- Refresh cadencies:
  - Daily
  - Weekly
  - Monthly
  - Quarterly
  - Bi-annual
  - One-off
    - Historical drop followed by incremental additions

Variation in access to raw data



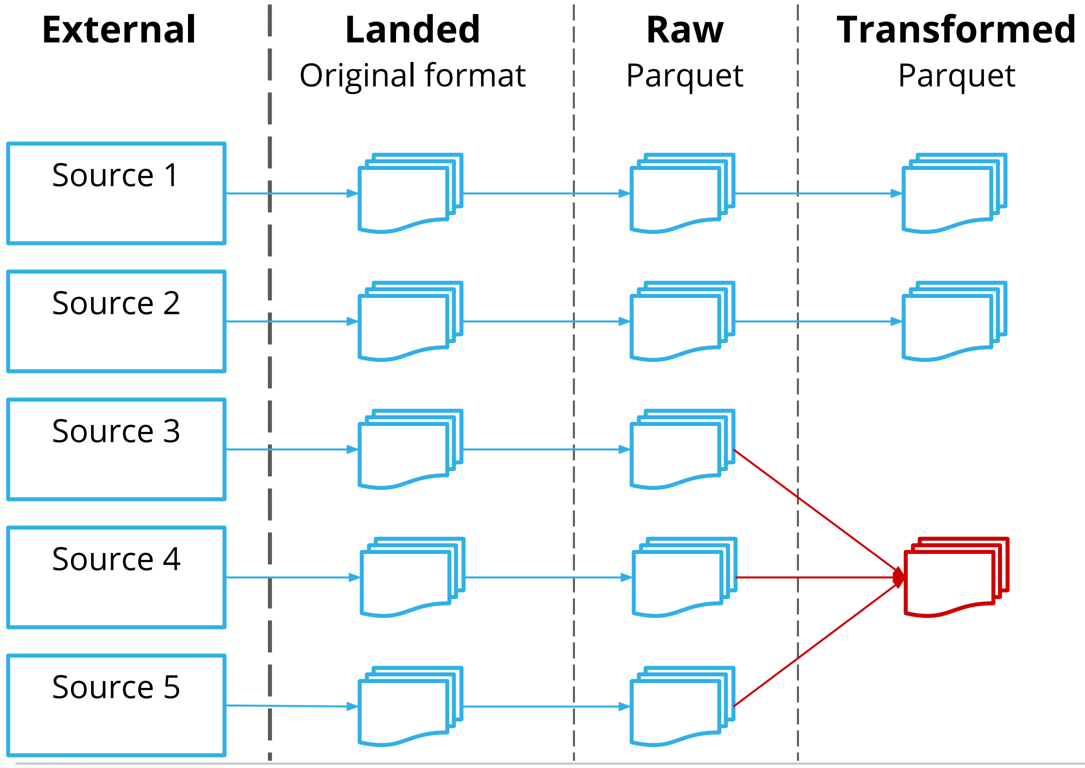
- Public and proprietary sources
- Size of data
  - From MBs to TBs
- Refresh cadencies:
  - Daily
  - Weekly
  - Monthly
  - Quarterly
  - Bi-annual
  - One-off
    - Historical drop followed by incremental additions
- Several interfaces for data extraction

Variation in file formats



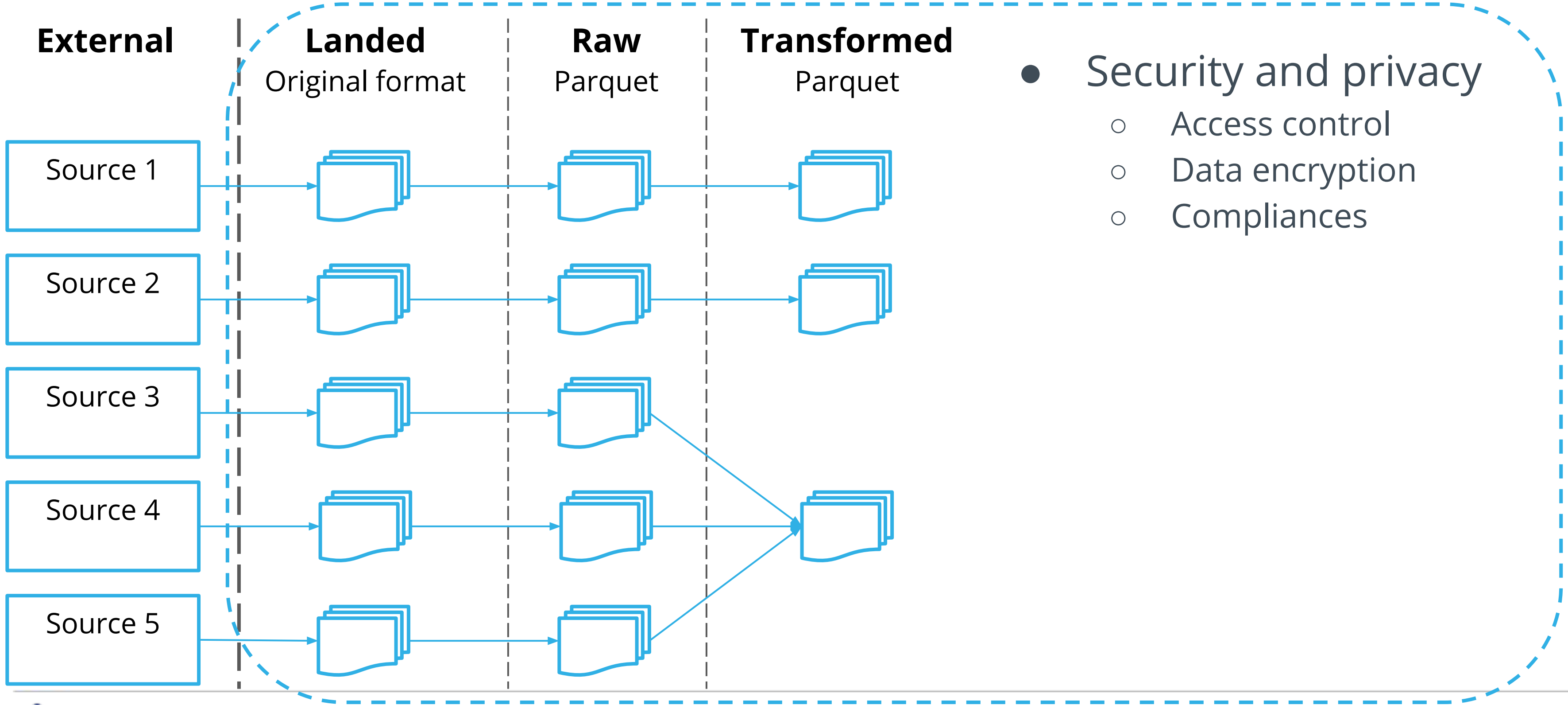
- Original file formats
  - CSV
  - XML
  - SAS
  - Fixed-width
  - Parquet
- Various compression formats
- Encrypted data

Cover several aspects of healthcare system



- Several datasets covering a single aspect of healthcare
  - Different schemas
  - Different conventions
- Need to transform to common schema

Security and privacy



- Security and privacy
  - Access control
  - Data encryption
  - Compliances

Prior to centralized data ingestion system

- Eternal question: What is the priority?
    - Scalability, maintainability, robustness, reliability
    - Rapid development

Prior to centralized data ingestion system

- Eternal question: What is the priority?
  - Scalability, maintainability, robustness, reliability
  - **Rapid development ← startup choice**
    - Provide value to customers and show progress to investors
    - React to changing requirements

Prior to centralized data ingestion system

- Eternal question: What is the priority?
  - Scalability, maintainability, robustness, reliability
  - **Rapid development ← startup choice**
    - Provide value to customers and show progress to investors
    - React to changing requirements
- Consequences:
  - Specialized pipelines
  - Manual operations
  - Variation in technologies and how to use them
  - Less reusable code

Why did we build a centralized ingestion system?

- Previous approach hard to maintain
  - Overhead in onboarding engineers to processes
  - Accumulation of manual tasks
- Project to integrate a few new data sources
  - Daily increments
  - Similar data sources
  - **Opportunity**: build system for these sources and migrate other sources later
- Pros of in-house implementation
  - Flexibility
  - Integrate with our tech stack
    - Leverage previous experience

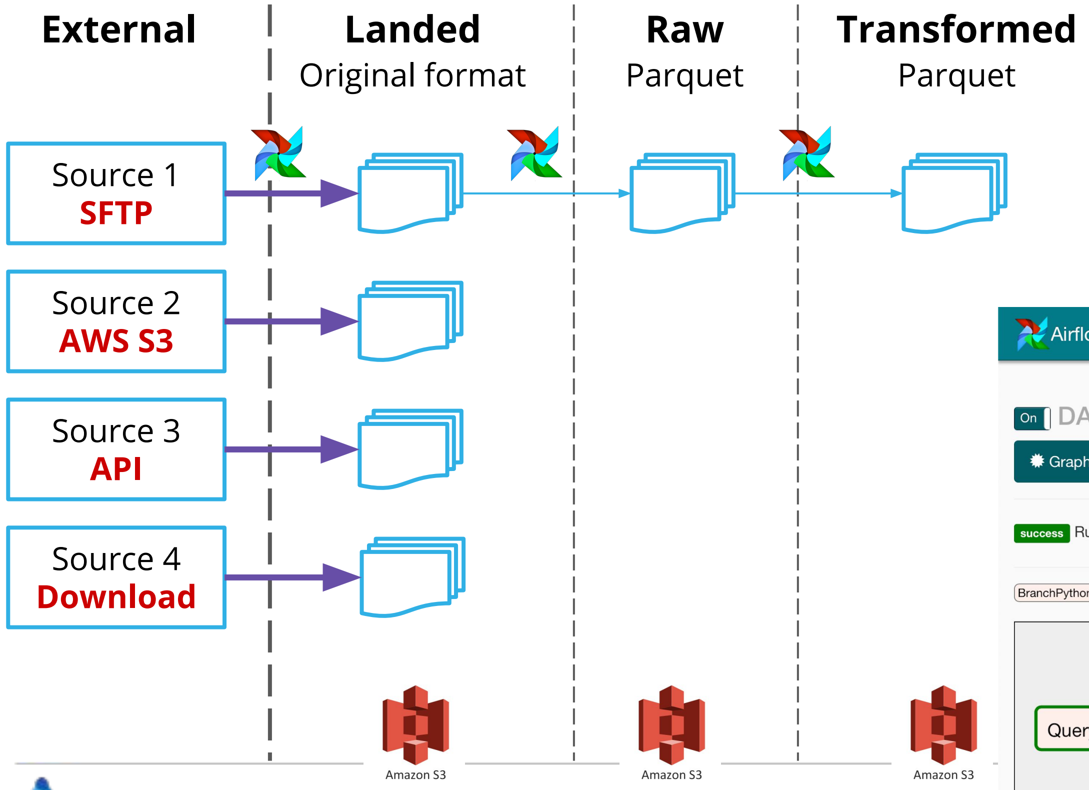
Agenda

- ❖ Komodo Health
- ❖ Data Ingestion Challenges
- ❖ Data Ingestion System Architecture
- ❖ Lessons Learned and Future Developments
- ❖ Scaling Processes
- ❖ Conclusions

Overview of the system infrastructure

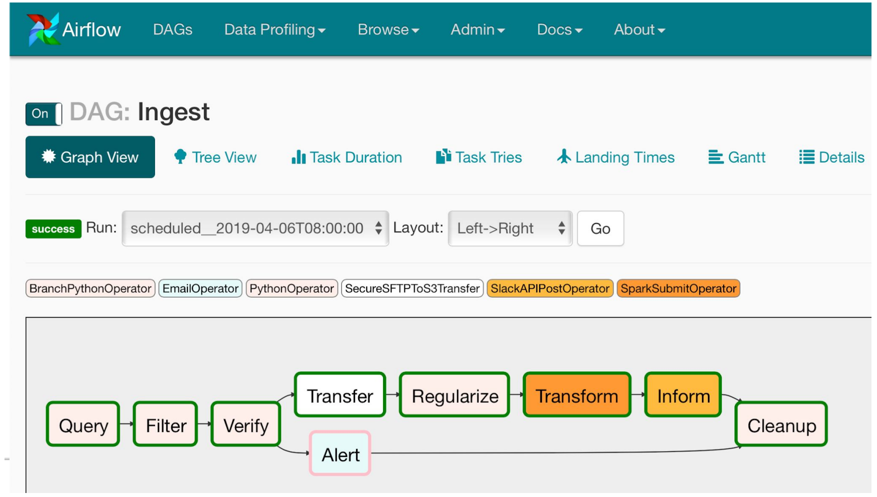
- Airflow
  - Organize workflows
  - Automation
  - Alerting
- Spark
  - Distributed processing
- Kubernetes
  - Container management
- AWS
  - EC2 - servers
  - S3 - store data

Airflow: Schedule workflows

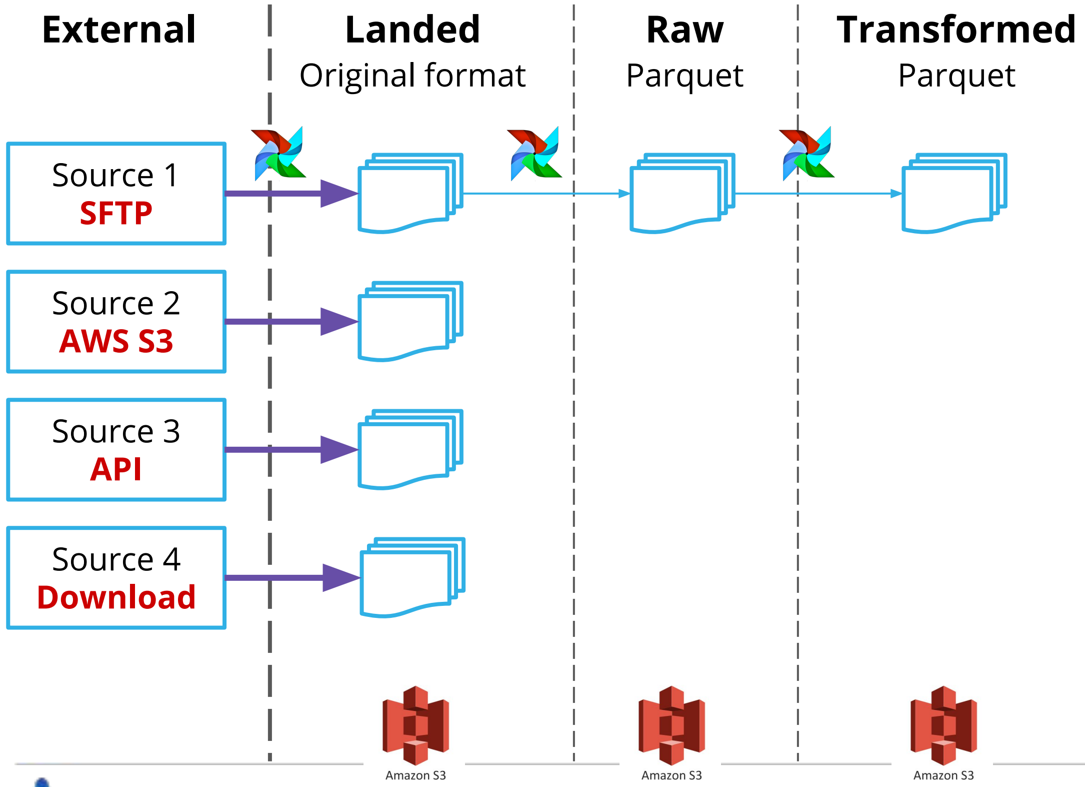


Pros:

- DAGs written in Python
- **Hooks** to integrate with sources
- **Operators** for common tasks
- Alert on success/failure
- Monitoring
- Parallelize DAGs and tasks



Airflow: Schedule workflows



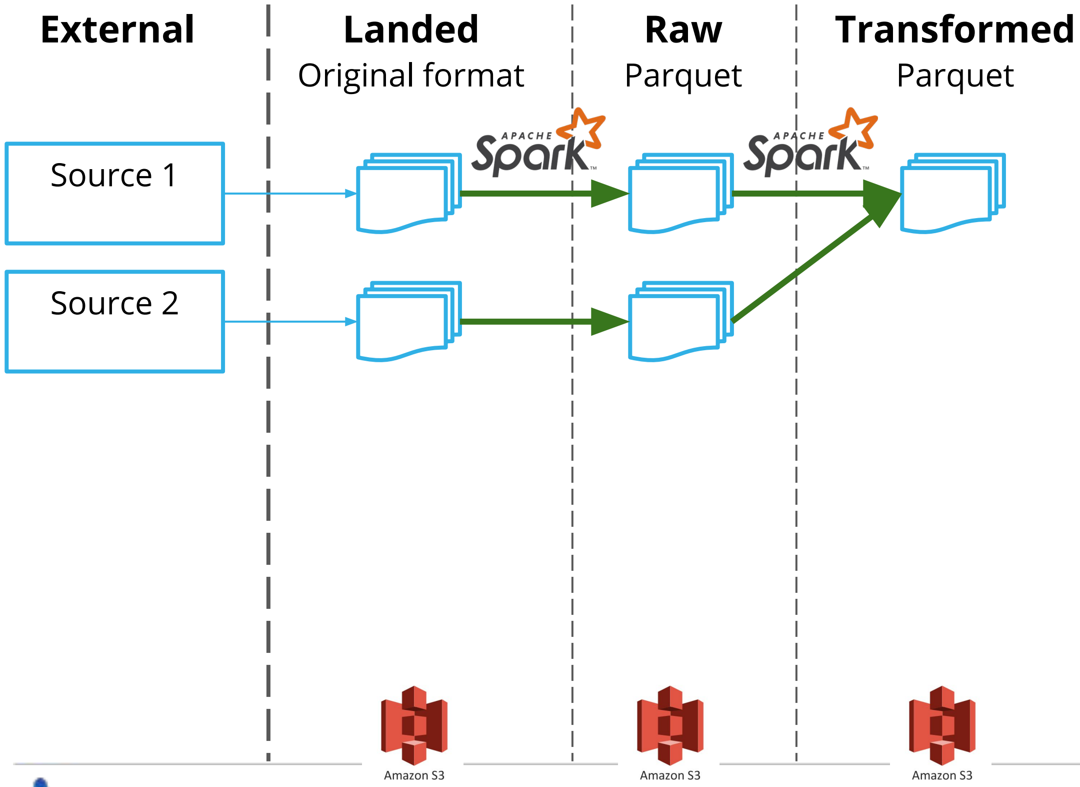
Pros:

- DAGs written in Python
- **Hooks** to integrate with sources
- **Operators** for common tasks
- Alert on success/failure
- Monitoring
- Parallelize DAGs and tasks

Cons:

- Had to customize hooks and operators
  - Handling credentials
  - Needing additional S3 metadata

Spark: Distributed processing



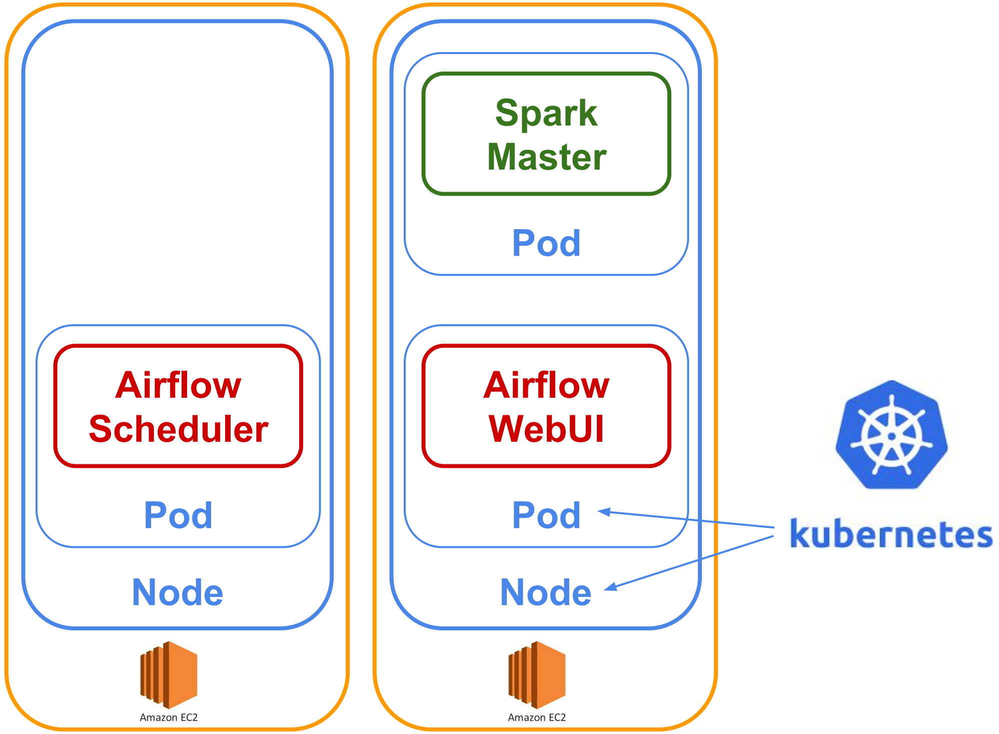
Pros:

- Reliable
- Python and Scala APIs

Cons:

- Performance tuning can be tricky

Kubernetes: Container management



Pros:

- Environments isolated to namespaces
- Node selectors for resource allocation
  - Nodes labeled based on the Auto Scaling Groups instances are tied to
- **Self-healing of pods!**

Cons:

- Occasional stability issues
  - Networking issues
- Difficult to troubleshoot

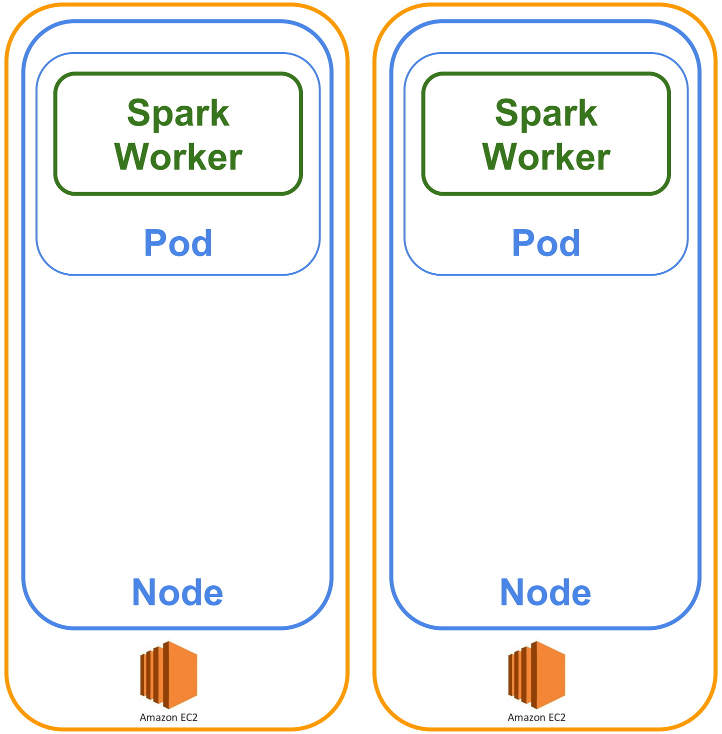
So far so good

- ✓ Scheduled execution
- ✓ Parallelized tasks
- ✓ Scalable resources
- ✓ Alerting
- ✓ Monitoring
- ✓ Resilient infrastructure
- ✓ Isolated environments

Agenda

- ❖ Komodo Health
- ❖ Data Ingestion Challenges
- ❖ Data Ingestion System Architecture
- ❖ **Lessons Learned and Future Developments**
- ❖ Scaling Processes
- ❖ Conclusions

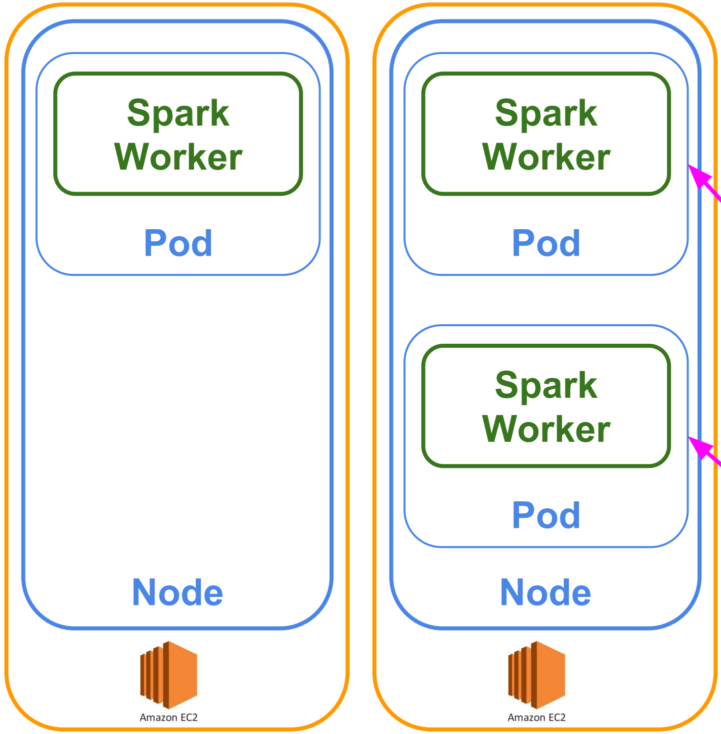
Infra limitation: Spark scaled manually



Big spikes in resource usage

- Wasteful to keep scaled up
- Scaling down is tricky
- Currently run big workloads on separate cluster
    - Manual operation :(

Infra limitation: Spark scaled manually

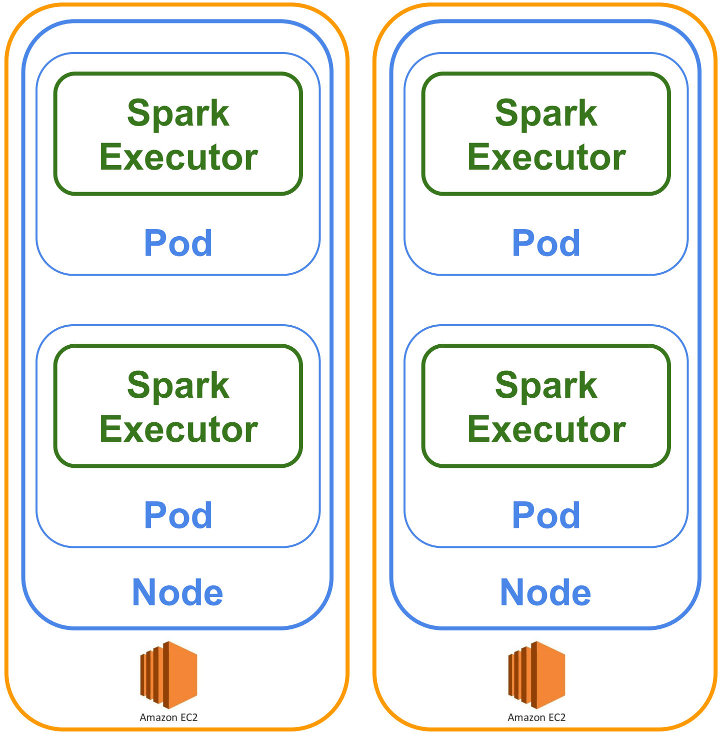


Big spikes in resource usage

- Wasteful to keep scaled up
- Scaling down is tricky
- Currently run big workloads on separate cluster
    - Manual operation :(

Two Spark workers on the same node resulted in double counting Spark resources

Automatic scaling under development



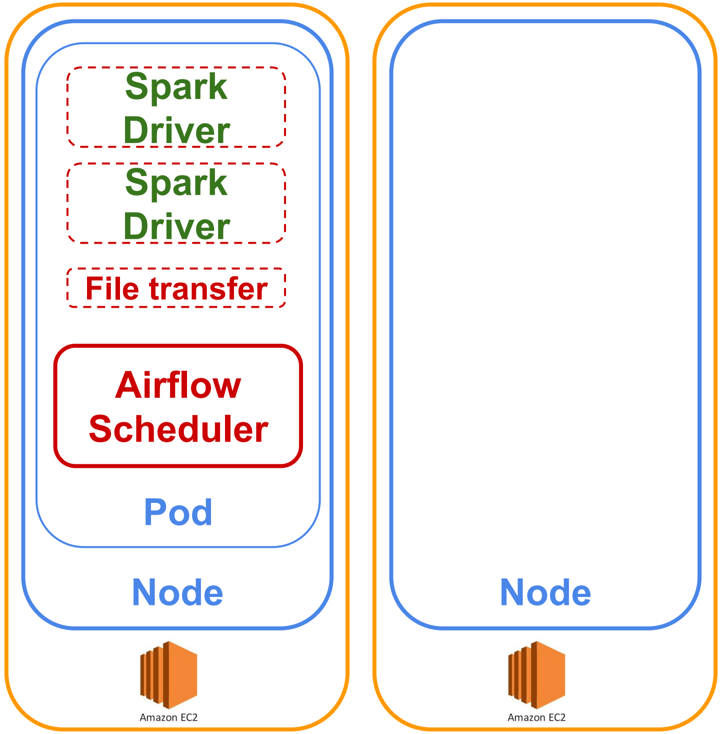
Big spikes in resource usage

- Wasteful to keep scaled up
- Scaling down is tricky
- Currently run big workloads on separate cluster
    - Manual operation :(

Future solution:

- Run Spark directly on Kubernetes
    - Introduced in Spark 2.4.0 for client mode
- K8s autoscaler to scale nodes

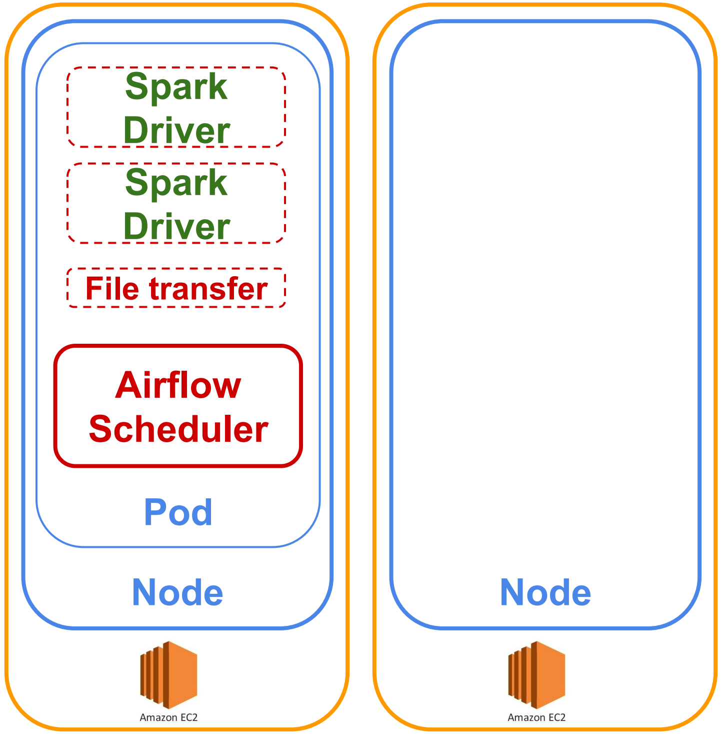
Infra limitation: Scheduler a single point of failure



Using local executor

- Tasks executed as subprocesses of scheduler
- Scale resources vertically
- Self-healing on failures? It depends...

Infra limitation: Scheduler a single point of failure



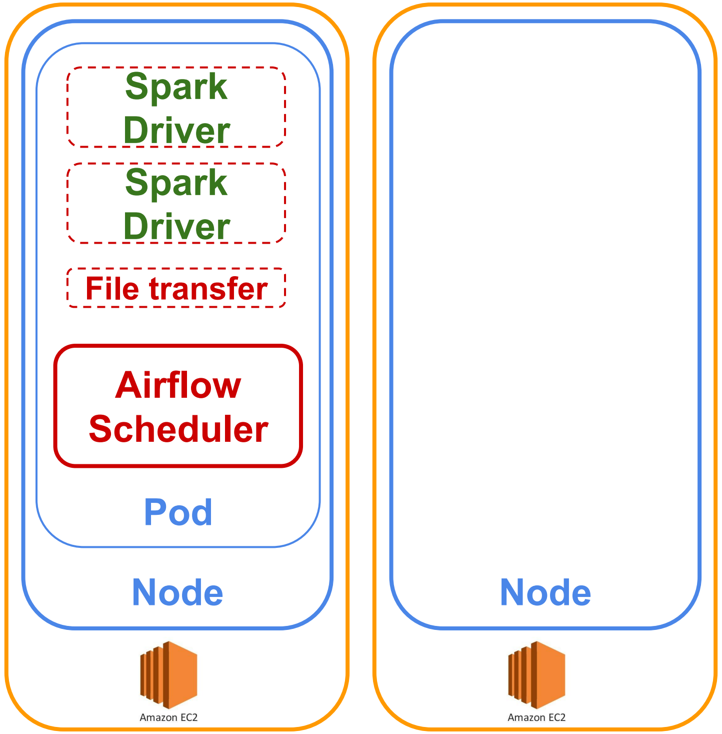
Using local executor

- Tasks executed as subprocesses of scheduler
- Scale resources vertically
- Self-healing on failures? It depends...

Issues in self-healing:

- Inconsistency in Airflow database
- Dependency on lost local file
- Pod evicted due to disk pressure

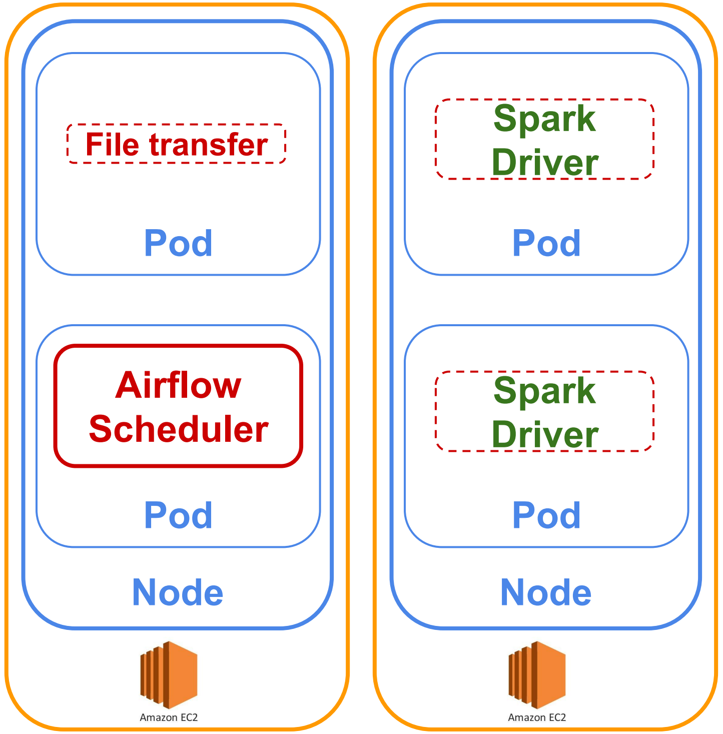
Why are you using local executor?



It has served us well, so far

- It was enough when we started
- Did not want to add complexity

Automatic scaling under development, again



It has served us well, so far

- It was enough when we started
- Did not want to add complexity

Future solution:

- Kubernetes executor
  - Introduced in Airflow 1.10.0
- K8s autoscaler to scale nodes

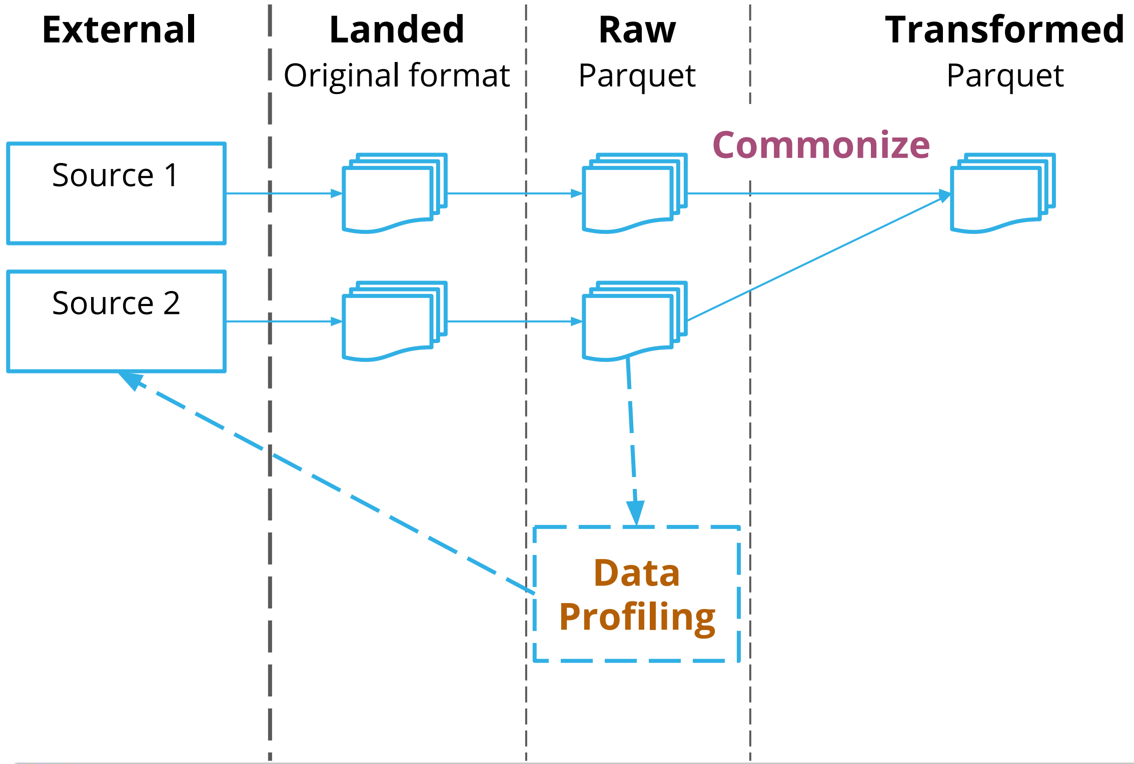
Agenda

- Komodo Health
- Data Ingestion Challenges
- Data Ingestion System Architecture
- Lessons Learned and Future Developments
- **Scaling Processes**
- Conclusions

Beyond infra - Scaling the ingestion processes

- Our data ingestion priorities:
  - Speed of data delivery
  - Data quality
  - Security and privacy
- Bottleneck is engineering time spent on integrating new data sources
  - Tools to simplify processes

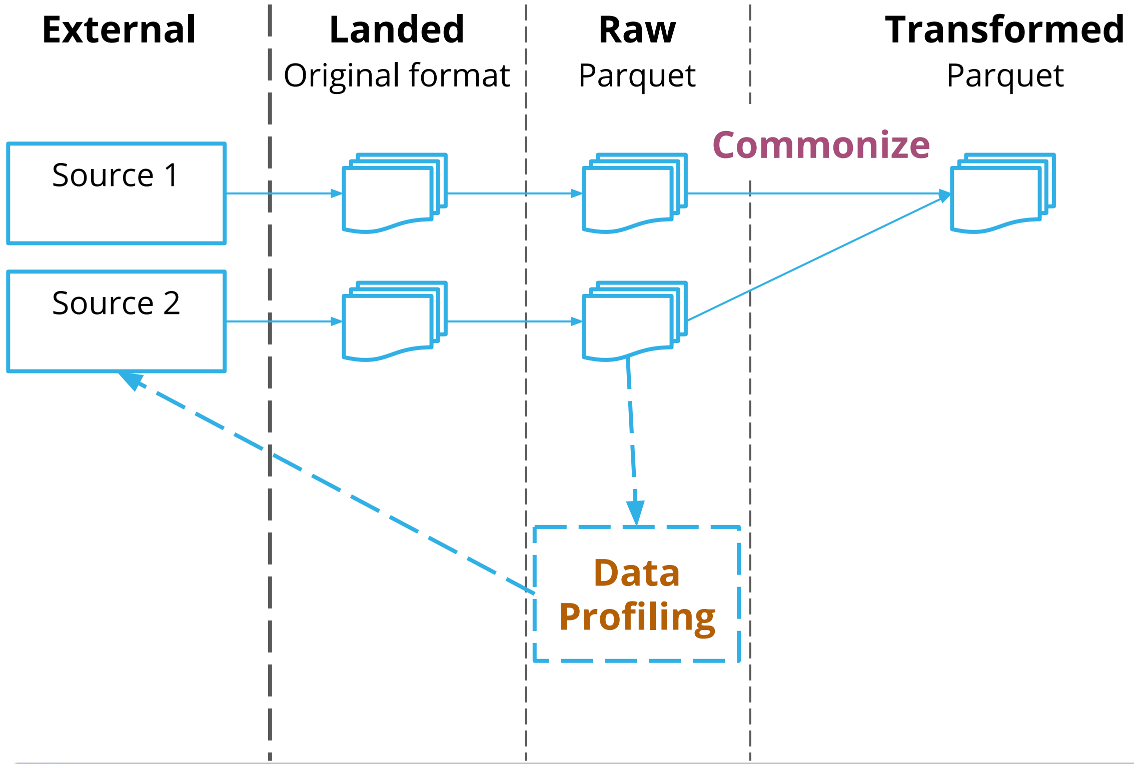
Early and fast iterations



Data profiling tool:

- Recognize columns
  - Simplifies commonization
- Validate raw data
  - Communicate issues with source
  - Compliance risks

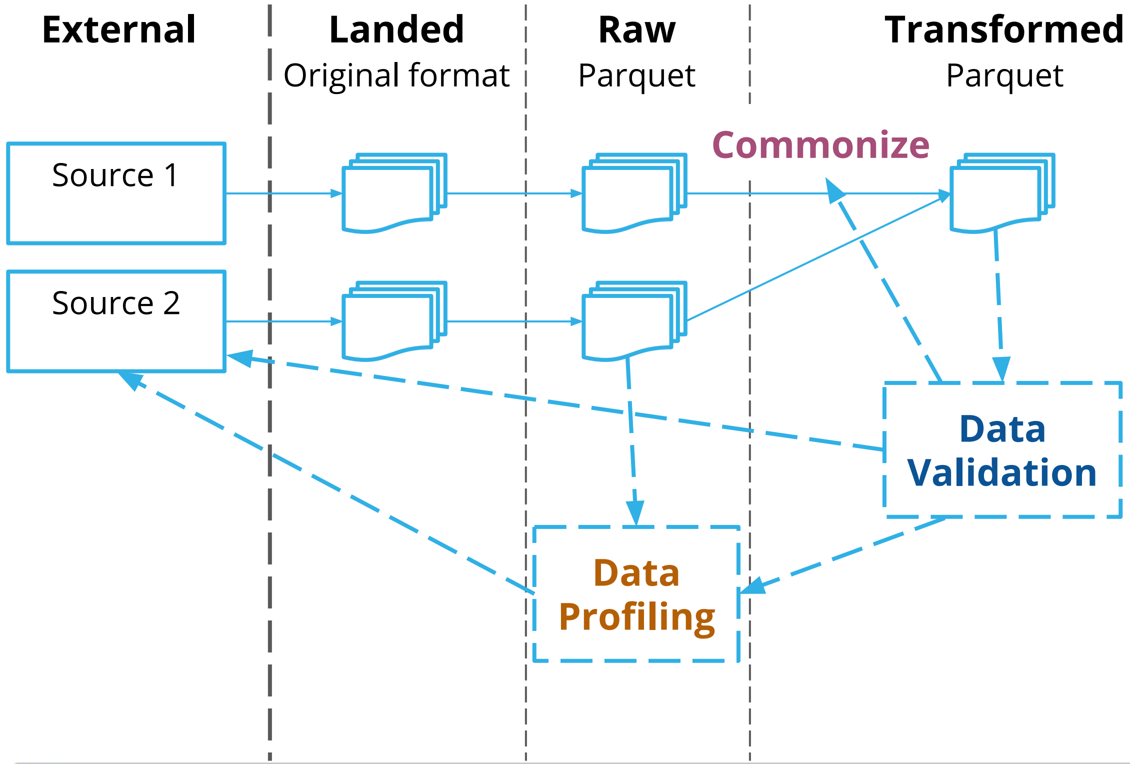
Avoid repeated work



Commonization tool:

- Similar data to common schema
- Based on configuration file
  - Very little code needed

Emphasis on data quality



Data validation tool:

- Validate against data standard
  - Catch bugs in commonization
  - Improve data profiling
  - Communicate issues with source

Conclusions

- ❖ Architecture with Airflow, Spark and Kubernetes very flexible for complex data ingestion
- ❖ Lots of nuances with these technologies and their interactions
- ❖ These technologies are constantly improving
- ❖ Not just infra that needs to scale, but also the processes
- ❖ Make sure you know your specific priorities

Thank you for your attention!

- Architecture with Airflow, Spark and Kubernetes very flexible for complex data ingestion
- Lots of nuances with these technologies and their interactions
- These technologies are constantly improving
- Not just infra that needs to scale, but also the processes
- Make sure you know your specific priorities