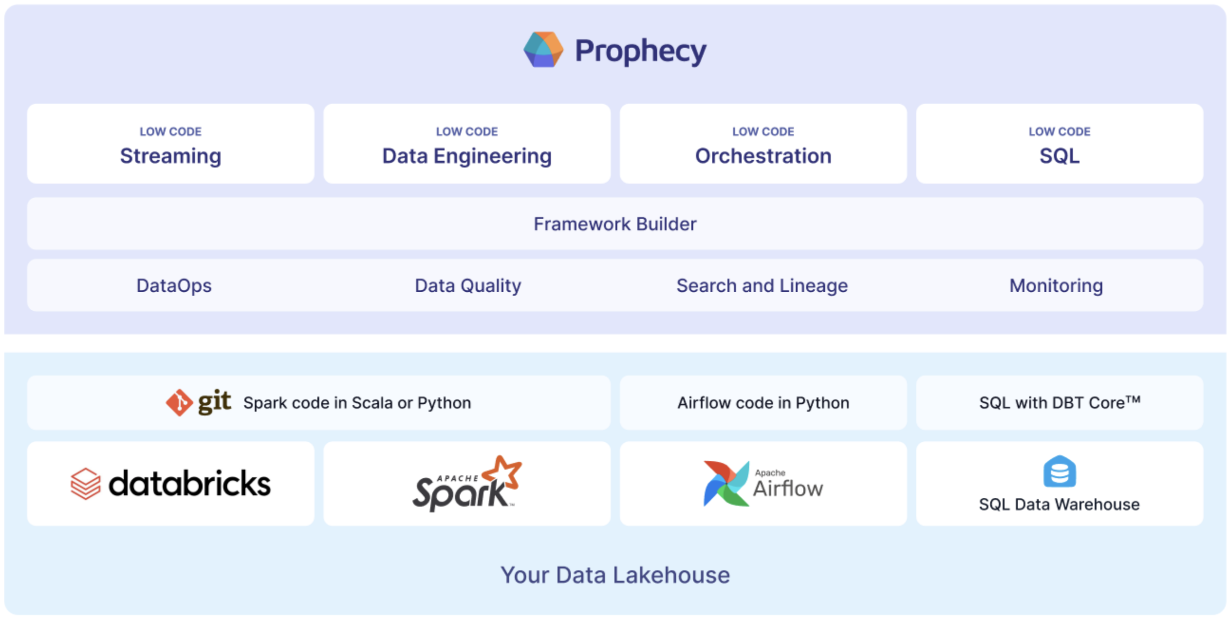
The low-code lakehouse architecture guide

Prophecy | databricks


Prophecy
LOW CODE Streaming
LOW CODE Data Engineering
LOW CODE Orchestration
LOW CODE SQL
Framework Builder
DataOps
Data Quality
Search and Lineage
Monitoring
git Spark code in Scala or Python
Airflow code in Python
SQL with DBT Core™
databricks
Apache Spark
Apache Airflow
SQL Data Warehouse
Your Data Lakehouse

Why Prophecy

The case for a new ETL tool

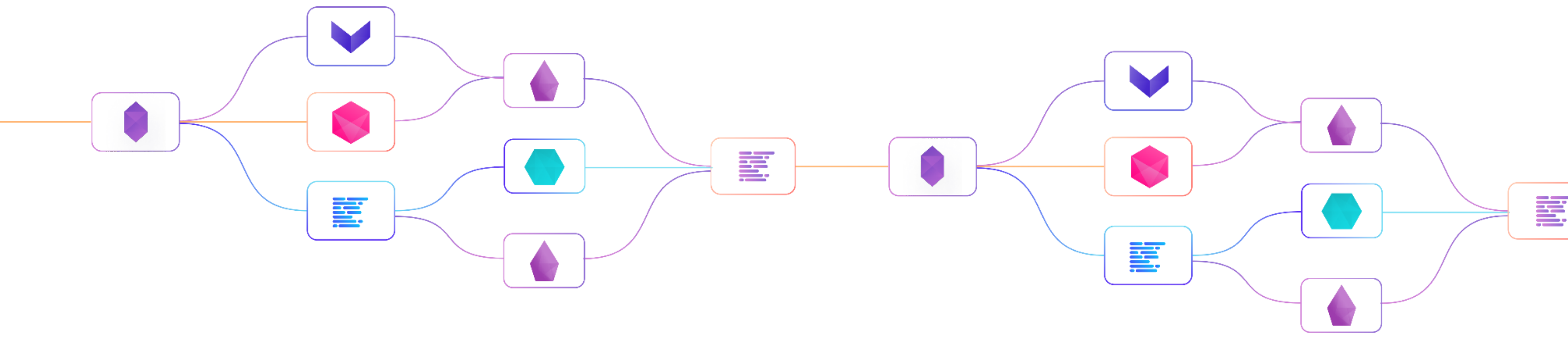
Data engineering is complex. Many enterprises encounter numerous barriers that prevent them from getting value from their data. Data practitioners are struggling to build the next generation of data products. Collecting, storing, processing and analyzing data becomes slow, complicated and hard to maintain. Enterprises have their pick of modern, robust execution engines, but their complexity prevents users from being productive. Significant pains include challenges for individual contributors to learn business and technical complexities, leading to team silos and collaboration challenges. Very few organizations provide a self-service toolkit.

Apache Spark™ is an incredibly powerful execution engine that supports large-scale data processing. It’s reliable, flexible and highly scalable. But, with its power, comes complexity. To onboard, a new data engineer must learn the ins and outs of distributed systems, data storage and data processing in various business verticals. Experts in these broad, highly technical areas are very limited. The knowledge required to build a flexible, robust and performant data stack demands many years of knowledge that can only be obtained by working with large-scale data platforms.

Promoting standardized, high-quality and well-documented code to production is complicated. Many organizations need more collaboration between teams and team members. When team members do not communicate, multiple engineers write the same code, with various standards and quality. Siloed teams find maintenance expensive and grueling because only the handful of engineers that wrote the code can understand it.

Businesses must endure long design and data implementation processes, long onboarding cycles, expensive maintenance, and poor data quality, resulting in inaccurate business intelligence (BI). Complexities and inconsistencies create silos between operations, platforms, infrastructure and analytics teams. Silos prevent teams from gathering accurate business and technical requirements to build functional and optimized data products. Teams are pitted against each other because requirements are thrown "over the wall," delivered without understanding the business need.

Instead, business and data teams need a self-serve paradigm.



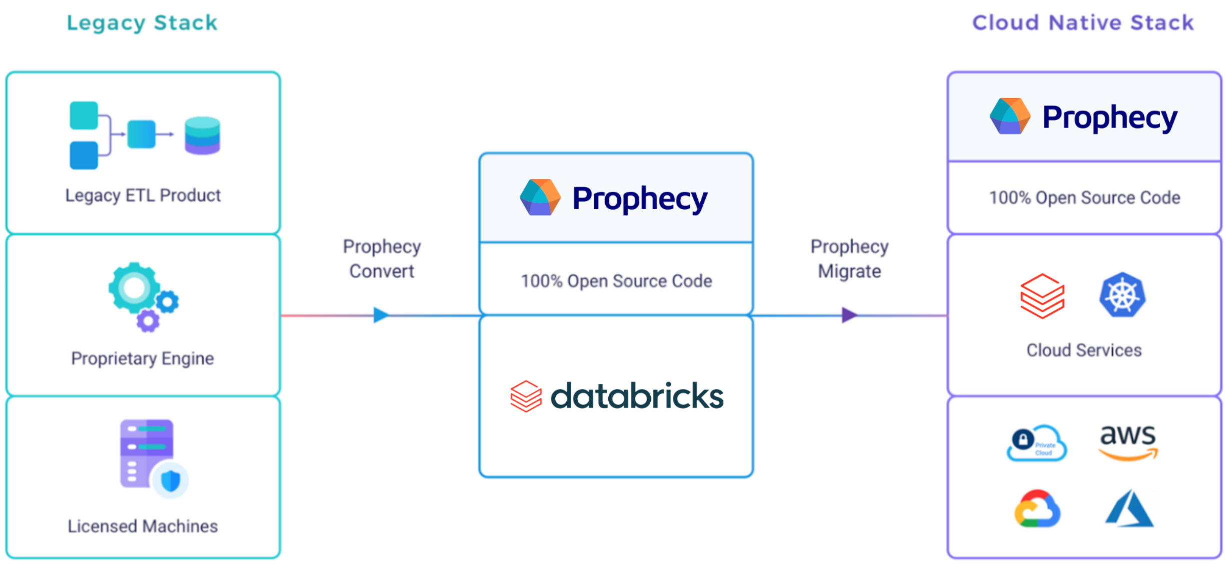
Prophecy as a low-code tool for data practitioners

Prophecy's low-code approach with Apache Spark combines a drag-and-drop visual interface, performance and management of traditional warehouses with the flexibility and scalability of data lakes to deliver a modern, low-code data lakehouse architecture. This architecture allows all data practitioners to build out complex pipelines easily, quickly and at scale.

Many data teams need to onboard new data quickly to meet business needs. Prophecy provides these capabilities using a visual user interface and built-in transformation and enrichment features. Instead of writing thousands of lines of code that few people can understand, a feature-rich UI brings everyone together to build pipelines in minutes. Prophecy uses gems — configurable visual entities representing data sources, transformations, targets and jobs — as building blocks. Organizations find it easy to standardize their development environment using gems. They also enable other teams to reuse these abstractions. Prophecy eliminates duplicated team efforts to reduce development time and resource costs. If a team discovers a pipeline they want to reuse, they can import it into their own project with a few clicks. Code reuse is a central theme in Prophecy, with the ability to reuse individual gems, pipelines or even a portion of a pipeline. In addition to easy building blocks, Prophecy provides popular features such as data lineage, quality checks and governance out-of-the-box.

Traditional data warehouses have scalability and usability concerns. Data teams are moving from classic, centralized data warehouses to a lakehouse architecture. There, data persists in its raw form on low-cost cloud storage in open formats. The architecture also separates compute and storage layers, and supports diverse workloads while providing management and governance capabilities.

Prophecy is used to migrate data from legacy data warehouse environments to a modern data lakehouse architecture powered by Databricks, as described in the picture below.



A powerful architecture isn't valuable if users can't use it. Data engineers can't build data products correctly without understanding the products' use cases. Analysts shouldn't have to wait for data engineers to finish building a pipeline to check on data quality and results. All data practitioners should be capable of transforming from raw data in the lakehouse's bronze layer to usable BI without convoluted processes and extensive coding.

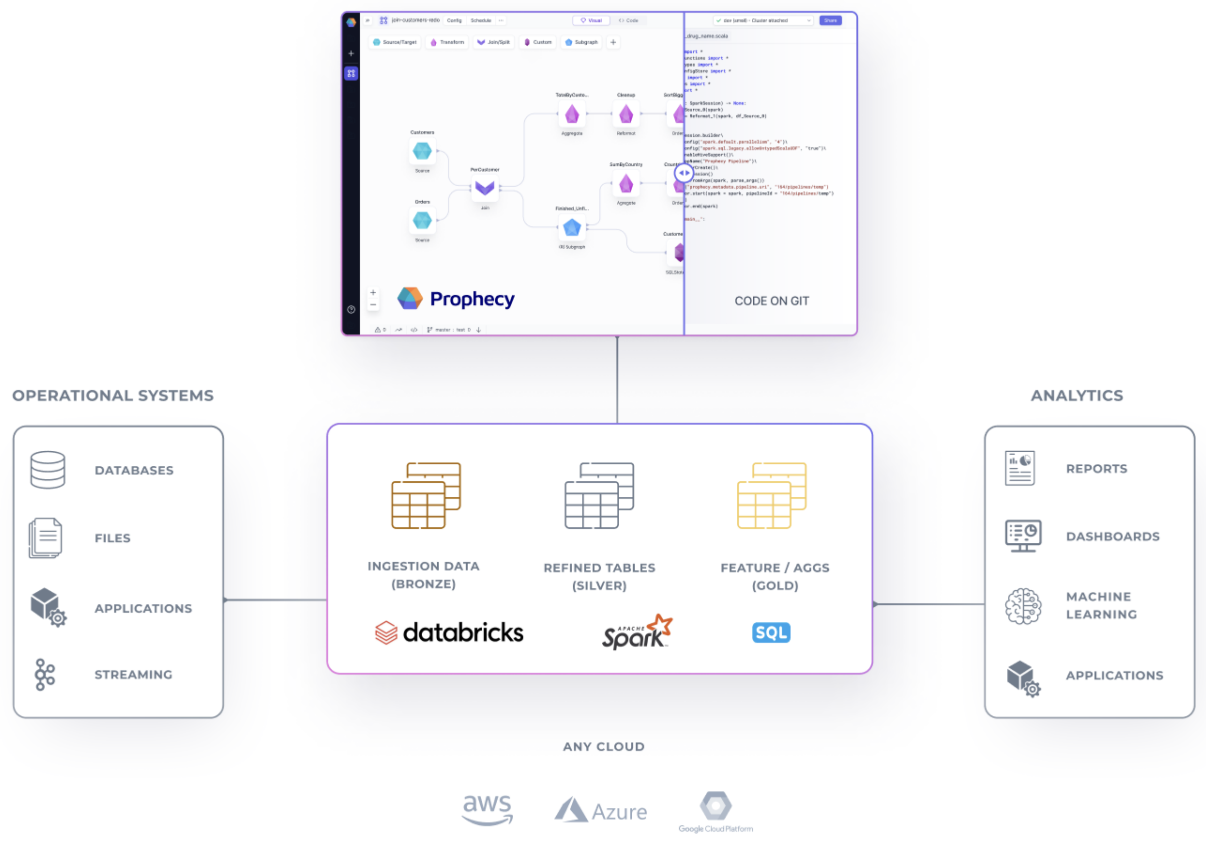
With Prophecy and Databricks, BI users and data scientists can self-serve by using a visual tool and collaborating with engineers to ensure the data meets business and technical requirements. The code generated by Prophecy from each data practitioner is committed to Git and follows software development best practices. This paradigm allows teams to work together instead of in silos, providing data transparency to all data practitioners. The architecture also lays out a single version of truth because data is stored in its raw form in cloud storage. If data processing makes an error, it's easy to replay data from its raw form.

Prophecy enforces data consistency downstream to avoid wrong business decisions caused by poor data quality. Once a few data pipelines are developed, many components can be reused to speed up new data onboarding. Prophecy also makes the iteration process extremely fast because users can drag and drop existing entities into new pipelines.

Understanding that ease of use and fast iterations do not translate to limitations is essential. Prophecy is a visual ETL tool with built-in extensibility and customization. It fits the user's data ecosystem by keeping itself open and extensible. Popular tools such as Airflow, Kafka and Splunk integrate easily into the lakehouse ecosystem for streaming, log analytics and workflow management use cases. Customizable gems and UDFs extend Prophecy further to address specific needs. The Prophecy engine translates visual components into open-source code. Users’ code sits natively on Git and can be shared securely. The users can leverage the CI/CD capabilities to promote their code from development and testing to production in a reliable, standardized environment.

Prophecy was born from the need for a tool that allows data practitioners to collaborate and self-serve in order to deliver data products quickly. The low-code lakehouse architecture brings visual interface, code, BI and machine learning (ML) users together for collaboration. Prophecy users can onboard and deploy a new data product in one sprint (days) instead of multiple sprints (weeks and months).

Prophecy lakehouse architecture tour



A lakehouse is an open architecture that combines the best elements of data lakes and data warehouses. Lakehouses are enabled by a new system design: implementing similar data structures and data management features to those in a data warehouse, directly on top of low-cost cloud storage in open formats.

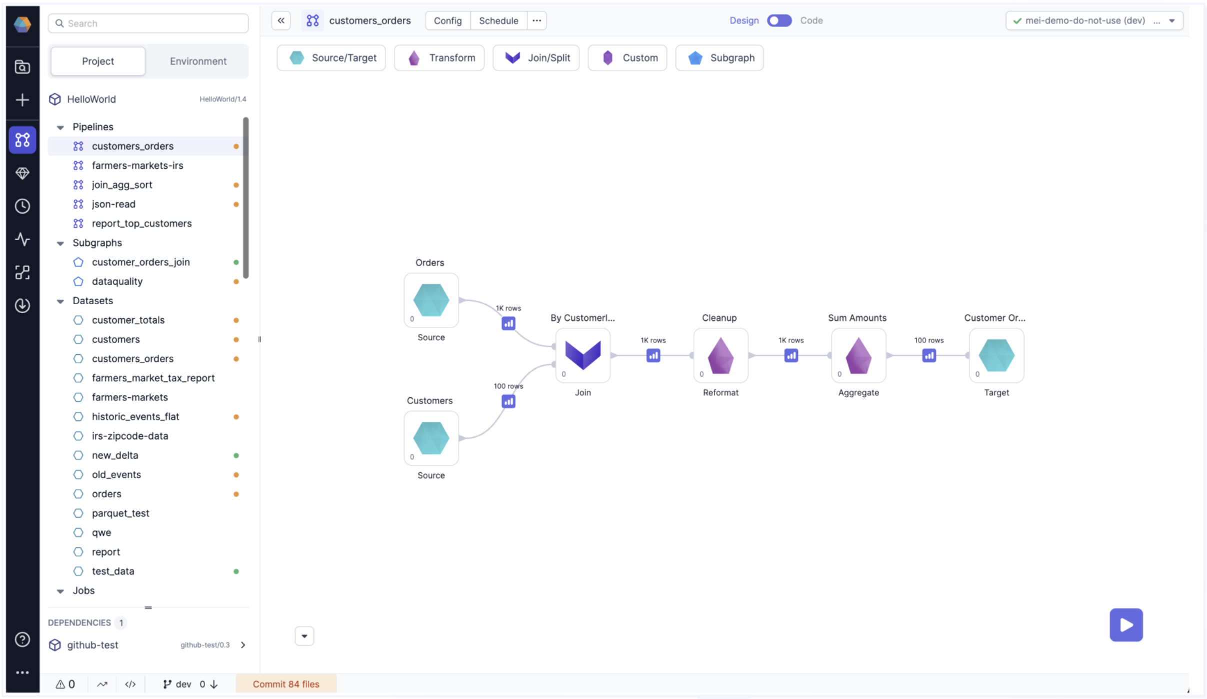
Prophecy is a feature-rich platform on top of the data lakehouse. Data engineers, visual ETL developers, data analysts and data scientists become productive quickly on the Databricks lakehouse using Prophecy's low-code user interface.

Prophecy provides a visual development, deployment and management environment for building data pipelines. As a user drags and drops gems to build out their pipelines, Prophecy simultaneously generates high-quality Apache Spark code stored on Git — code that provides more clarity and optimality than hand-written Spark code.

The lakehouse ecosystem consists of sources supporting both streaming and batch workloads (under operational systems in the diagram). File-based sources are prevalent in batch processing, representing a cost-effective and resilient storage layer that scales independently and is decoupled from compute. Event-based sources usually work with streaming pipelines. A messaging system such as Kafka is ideal for pulling events continuously. Prophecy provides built-in source gems for both batch and streaming pipelines.

The processing layer is responsible for data transformation, aggregation and enrichment. The lakehouse architecture takes raw data consumed from sources (bronze), refines the data, cleanses the data (silver) and transforms it into consumption-ready tables (gold). Prophecy provides the visual layer on top of Spark APIs to build the transformation layer with drag-and-drop components. The UI layer also allows data observability and governance. Different teams can reuse and share entities built in this layer using Prophecy.

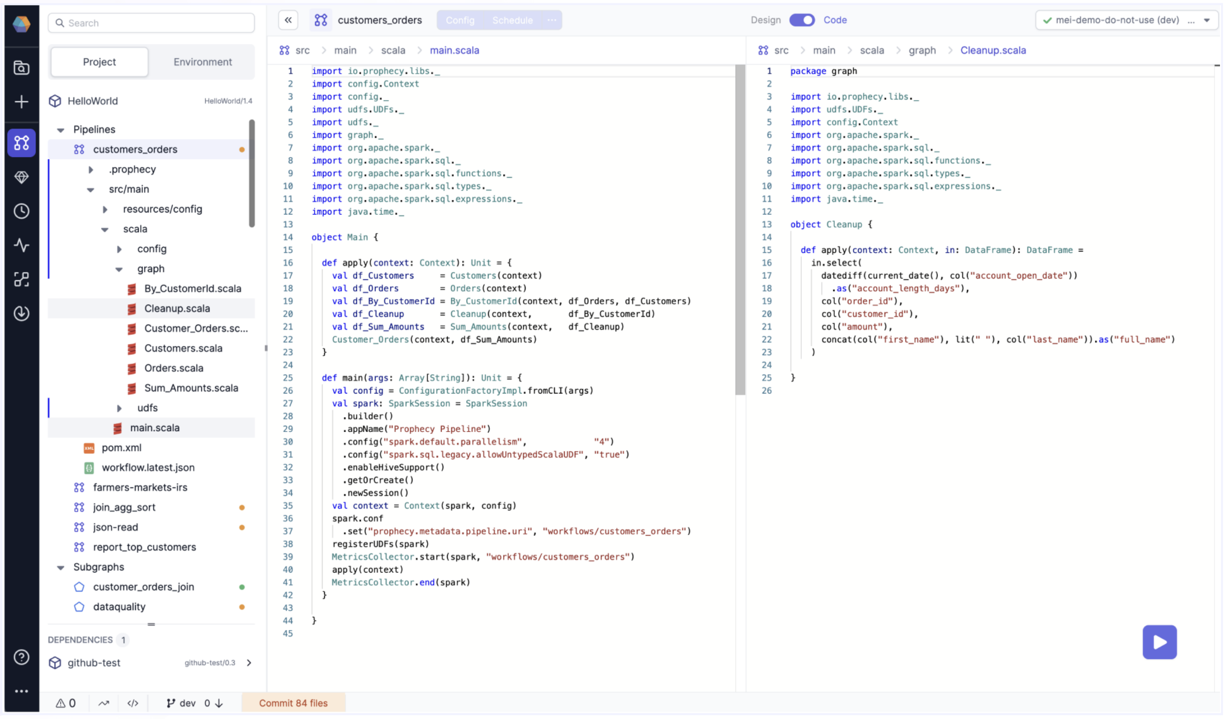
After the data is prepared, analytics tools consume serving-ready data from the gold layer to build BI reports, machine learning models, near real-time dashboards and applications (analytics in the diagram). Prophecy automatically applies all the best practices to your tables (e.g., Z-ordering) and generates the most efficient, Databricks-approved code. Prophecy works well with Delta Lake by reading from and writing to Delta tables. It also supports more complex constructs, such as slowly changing dimensions.



Prophecy users can build and manage entities from the pipeline builder canvas. The UI — which has a project browser located on its left-hand side — makes it seamless to navigate between projects, pipelines, subgraphs, datasets and jobs. Code generated from projects is persisted on Git.

Major Prophecy entities:

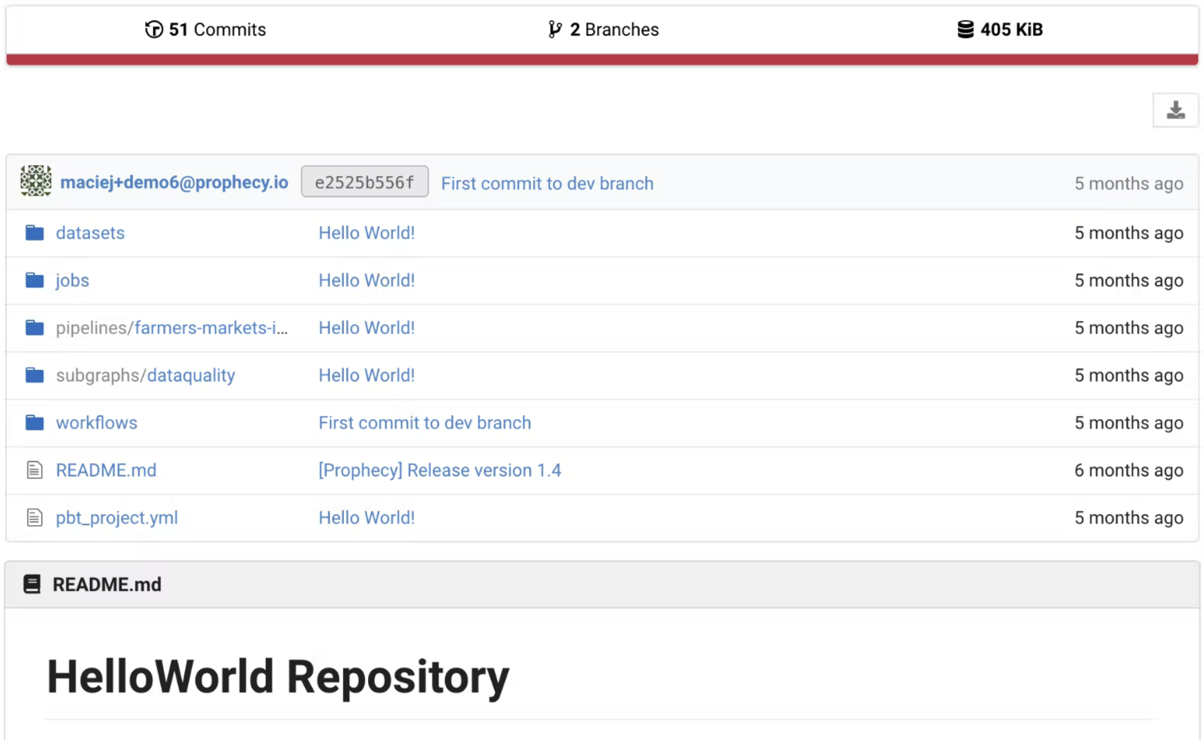
- **Projects —** Git repositories that store all Spark, Airflow and metadata code
- **Data pipelines —** Various ETL/ELT tasks written in PySpark, Scala or SQL
- **Jobs —** The orchestration of data pipelines written in Databricks Jobs or Airflow
- **Teams —** Users grouped in different teams with assigned ACL
- **Fabrics —** Connections to Databricks workspace
- **Datasets —** Representations of where data is stored and the schema of the data



```scala
import io.prophecy.libs._
import config.Context
import config._
import udfs.UDFs._
import udfs._
import graph._
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.sql.expressions._
import java.time._

object Main {

  def apply(context: Context): Unit = {
    val df_Customers     = Customers(context)
    val df_Orders        = Orders(context)
    val df_By_CustomerId = By_CustomerId(context, df_Orders, df_Customers)
    val df_Cleanup       = Cleanup(context,       df_By_CustomerId)
    val df_Sum_Amounts   = Sum_Amounts(context,   df_Cleanup)
    Customer_Orders(context, df_Sum_Amounts)
  }

  def main(args: Array[String]): Unit = {
    val config = ConfigurationFactoryImpl.fromCLI(args)
    val spark: SparkSession = SparkSession
      .builder()
      .appName("Prophecy Pipeline")
      .config("spark.default.parallelism",             "4")
      .config("spark.sql.legacy.allowUntypedScalaUDF", "true")
      .enableHiveSupport()
      .getOrCreate()
      .newSession()
    val context = Context(spark, config)
    spark.conf
      .set("prophecy.metadata.pipeline.uri", "workflows/customers_orders")
    registerUDFs(spark)
    MetricsCollector.start(spark, "workflows/customers_orders")
    apply(context)
    MetricsCollector.end(spark)
  }

}
```

```scala
package graph

import io.prophecy.libs._
import udfs.UDFs._
import config.Context
import org.apache.spark._
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.sql.expressions._
import java.time._

object Cleanup {

  def apply(context: Context, in: DataFrame): DataFrame =
    in.select(
      datediff(current_date(), col("account_open_date"))
        .as("account_length_days"),
      col("order_id"),
      col("customer_id"),
      col("amount"),
      concat(col("first_name"), lit(" "), col("last_name")).as("full_name")
    )

}
```

A Prophecy user can toggle between design and code views. In the code view, for example, users can view a pipeline stored as PySpark, Sclala or SQL in full transparency. (Note, Prophecy does not generate any preparatory code to maximize extensibility and customizations.)


51 Commits | 2 Branches | 405 KiB

maciej+demo6@prophecy.io e2525b556f	First commit to dev branch	5 months ago
datasets	Hello World!	5 months ago
jobs	Hello World!	5 months ago
pipelines/farmers-markets-i...	Hello World!	5 months ago
subgraphs/dataquality	Hello World!	5 months ago
workflows	First commit to dev branch	5 months ago
README.md	[Prophecy] Release version 1.4	6 months ago
pbt_project.yml	Hello World!	5 months ago

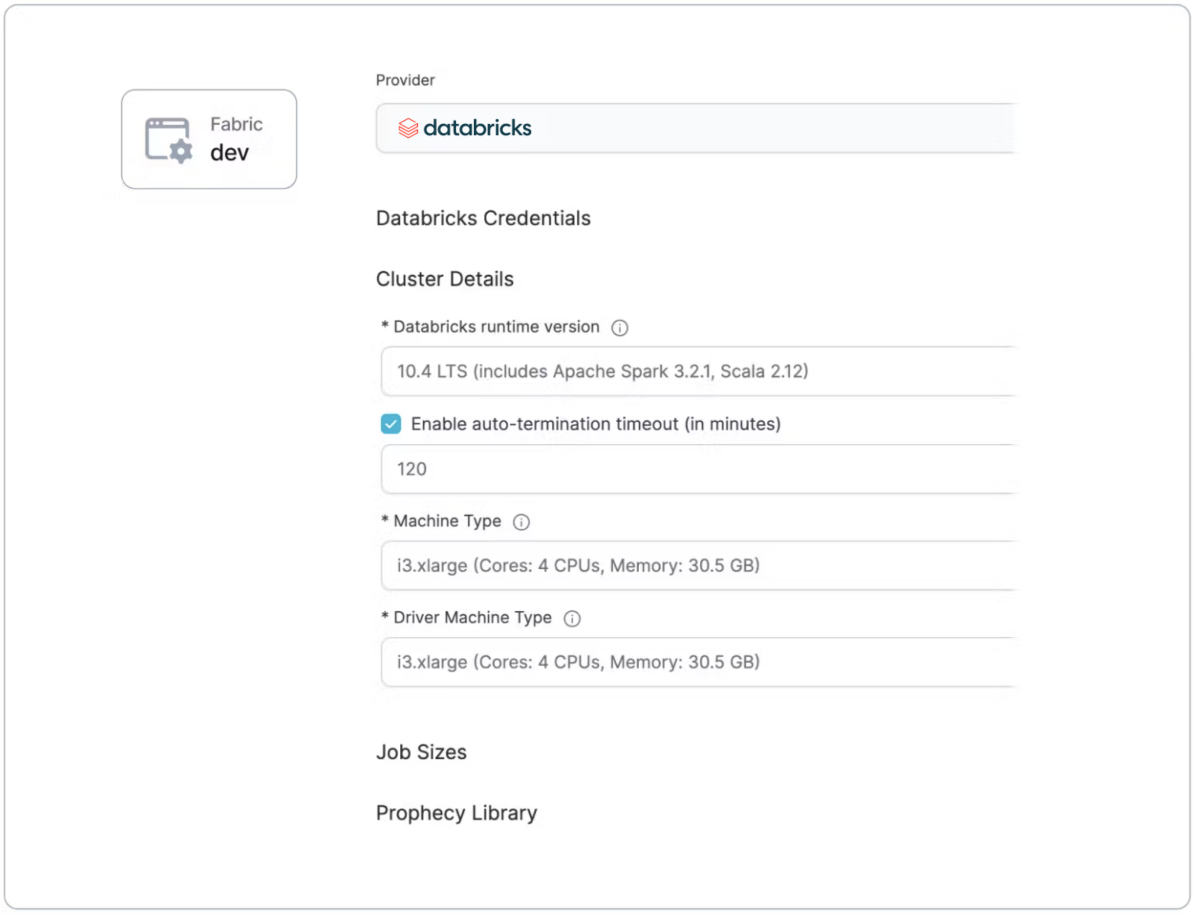
README.md

HelloWorld Repository

Every project in Prophecy is a fully-fledged Git repository that's integrated with Git providers. As mentioned earlier, every change in gems (the visual elements) generates high-quality code in PySpark, Scala or SQL committed to a branch — code Prophecy ensures follows software engineering best practices.

The typical software engineering process involves committing your code changes, resolving any Git conflicts, pushing them to release branches, getting them approved, running any integration tests, building all the artifacts and, finally, deploying them to clusters.

Prophecy data fabrics are connected to Databricks workspaces. The diagram above shows three fabrics connected to three workspaces. Each fabric is a dedicated functional execution environment for development, staging and production, and may have one or more clusters. A pipeline must be attached to a cluster for execution. Bronze, silver and gold tables will be populated when pipelines are executed.



Provider

databricks

Databricks Credentials

Cluster Details

* Databricks runtime version

10.4 LTS (includes Apache Spark 3.2.1, Scala 2.12)

✓ Enable auto-termination timeout (in minutes)

120

* Machine Type

i3.xlarge (Cores: 4 CPUs, Memory: 30.5 GB)

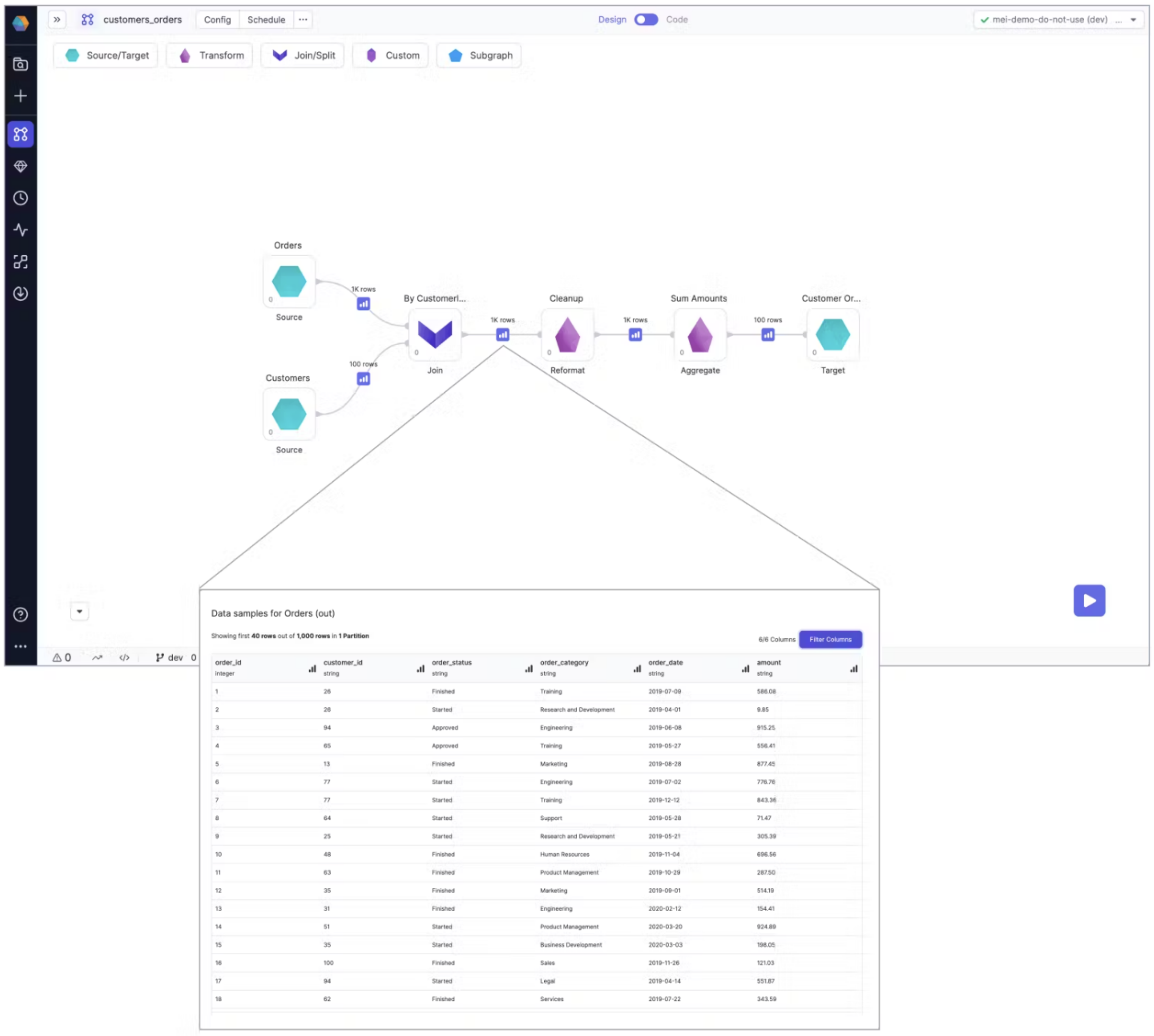
* Driver Machine Type

i3.xlarge (Cores: 4 CPUs, Memory: 30.5 GB)

Job Sizes

Prophecy Library

The administrator of a team can configure multiple fabrics associated with workspaces. The size of clusters can be configured based on the workload (we provide sizing guidelines under the upcoming "Prophecy deployment components" section).



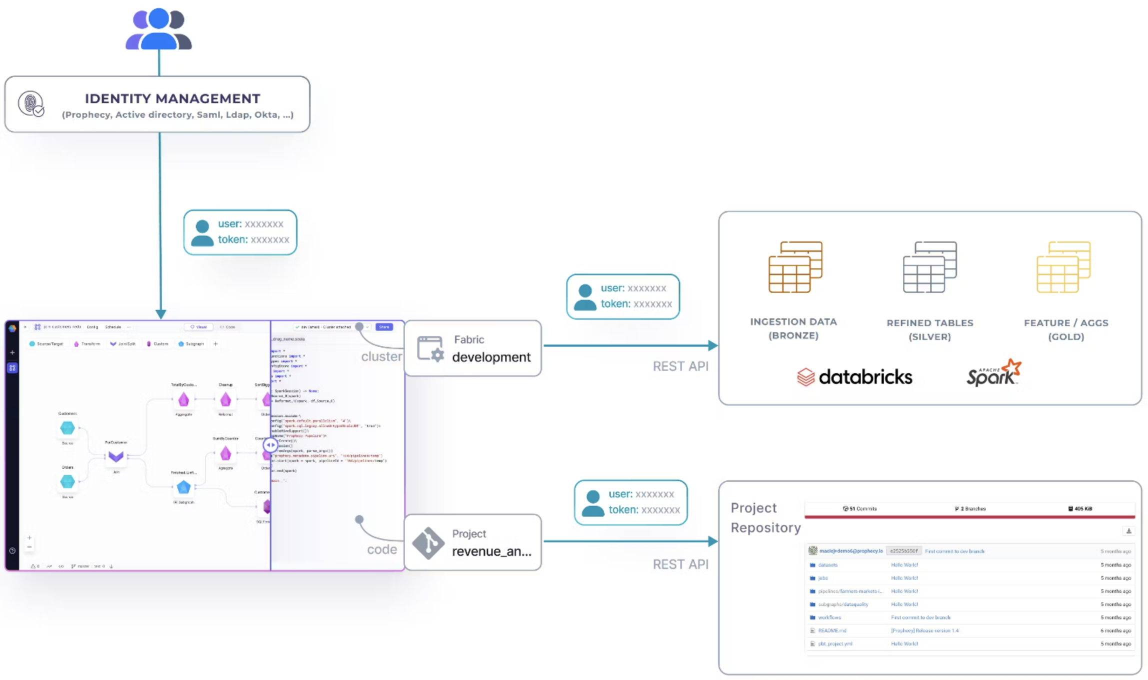
Data samples for Orders (out)

Showing first **40 rows** out of **1,000 rows** in **1 Partition**

6/6 Columns | Filter Columns

order_id integer	customer_id string	order_status string	order_category string	order_date string	amount string
1	26	Finished	Training	2019-07-09	586.08
2	26	Started	Research and Development	2019-04-01	9.85
3	94	Approved	Engineering	2019-06-08	915.25
4	65	Approved	Training	2019-05-27	556.41
5	13	Finished	Marketing	2019-08-28	877.45
6	77	Started	Engineering	2019-07-02	776.76
7	77	Started	Training	2019-12-12	843.36
8	64	Started	Support	2019-05-28	71.47
9	25	Started	Research and Development	2019-05-21	305.39
10	48	Finished	Human Resources	2019-11-04	696.56
11	63	Finished	Product Management	2019-10-29	287.50
12	35	Finished	Marketing	2019-09-01	514.19
13	31	Finished	Engineering	2020-02-12	154.41
14	51	Started	Product Management	2020-03-20	924.89
15	35	Started	Business Development	2020-03-03	198.05
16	100	Finished	Sales	2019-11-26	121.03
17	94	Started	Legal	2019-04-14	551.87
18	62	Finished	Services	2019-07-22	343.59

Pipelines can run interactively by clicking the Play button or scheduled periodically as jobs. Jobs can be orchestrated by Databricks Workflows (recommended), a fully-managed orchestration service that's deeply integrated with the lakehouse platform, or Apache Airflow (advanced). A Prophecy job is a way to run your data pipelines using a data fabric. A job consists of a single pipeline or a large, multi-pipeline workflow with complex dependencies. You can run your pipelines immediately or periodically. Prophecy captures additional metrics and data profiles for observability. You may also set up automated alerts to monitor how the data changes every run.

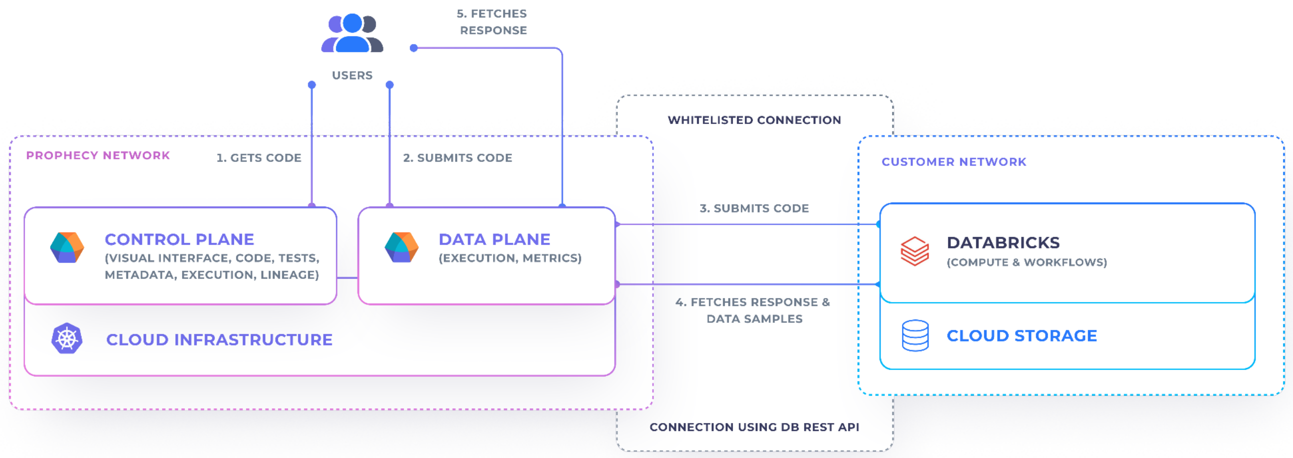


As mentioned earlier, data fabric is a logical execution environment. It includes everything required to run a data pipeline. Teams organize their data domain into multiple development, staging and production environments. Administrators will set up a Prophecy account and create a “dev fabric” for development and “prod fabric” for production. Prophecy connects to Databricks using REST API. Each fabric defined in Prophecy connects to a single Databricks workspace, and each user must provide a personal access token to authenticate. Enterprises that use Databricks with limited network access must add the Prophecy Data Plane IP address (3.133.35.237) to their Databricks-allowed access list.

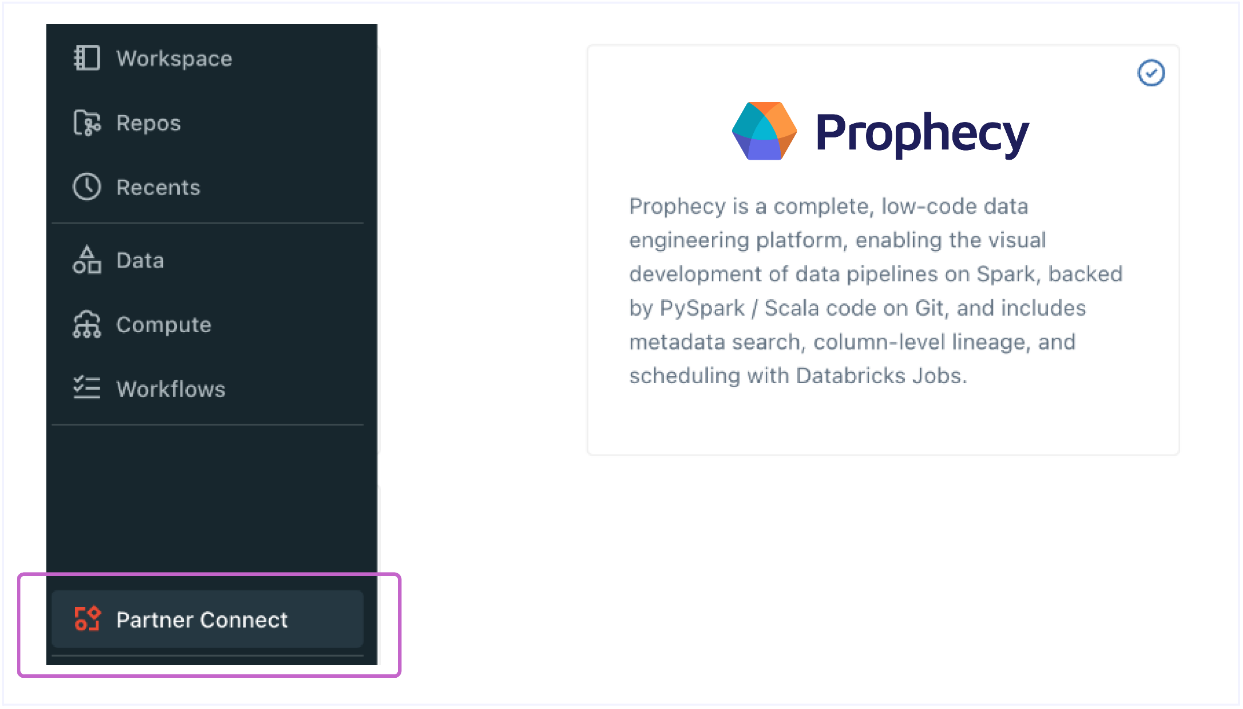
A Prophecy project is the primary unit of development and deployment. It integrates seamlessly with Git providers to manage the code generated by pipelines. Prophecy supports multiple Git providers, including GitHub. It authenticates with GitHub using per-user personal access tokens or OAUTH. Enterprises that use Git providers within private networks behind firewalls must add the Prophecy Control Plane IP address (3.133.35.237) to their private network allow-list or the Git provider-allowed access list.

Integrations

Databricks



Prophecy and Databricks work seamlessly together. Prophecy provides the visualization layer on top of Spark APIs as a low-code visual environment. The integration is accessible using Databricks Partner Connect or via the Prophecy UI by creating a data fabric.



Prophecy is a complete, low-code data engineering platform, enabling the visual development of data pipelines on Spark, backed by PySpark / Scala code on Git, and includes metadata search, column-level lineage, and scheduling with Databricks Jobs.

Prophecy connects to Databricks with data fabrics using REST APIs. A Prophecy data fabric is a logical execution environment. Prophecy users are organized into teams. A team can organize its data fabrics into multiple tiers: development, staging, testing and production. Each data fabric defined in Prophecy connects to a single Databricks workspace. A user is required to provide a personal access token for authentication.

Prophecy uses Databricks for the following functionalities:

- The interactive execution on Databricks is triggered from Prophecy using the Databricks REST APIs. Prophecy allows its users to spin up new clusters or connect to existing clusters. When a cluster connection exists, Prophecy enables the user to run their code in the interactive mode. Interactive code queries are sent to Databricks using the Databricks Command API 1.2.

- Scheduled runs are orchestrated through the Databricks Jobs API 2.1. Prophecy Spark pipeline code and Databricks job definition are deployed through the CI/CD process to Databricks (e.g., through the Prophecy Build Tool or any other build and CI system like Jenkins). Permissions for the CI/CD are managed by the Git provider itself (e.g., GitHub).

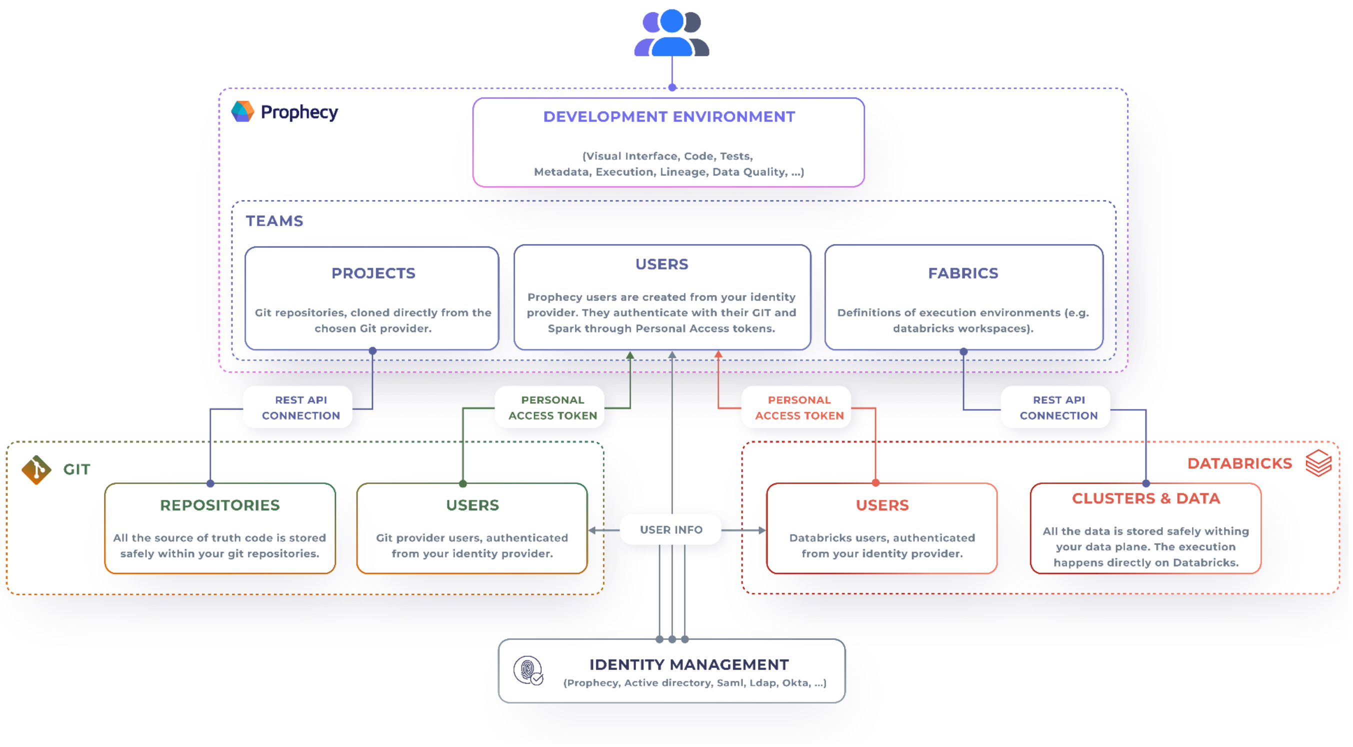
By default, Prophecy does not store any data samples when executing code using Databricks. Data samples (execution metrics) can be optionally stored for observability purposes.

Prophecy takes care of auto-generating and refreshing the Databricks personal access tokens when using Active Directory.

Git

Working in a robust development environment is essential. Every Prophecy project is connected to a Git repository. Prophecy can integrate with the Git provider of the user's choice. Every change in gems (the visual elements) generates optimal PySpark, Scala or SQL code. The code is committed to a specific branch. Additionally, Prophecy complies with all software engineering best practices.

The traditional way of managing ETL code can get complicated, requiring many scripts to be pieced together. Prophecy automates most of those steps, minimizing the number of repeatable, error-prone steps.


Prophecy
DEVELOPMENT ENVIRONMENT
(Visual Interface, Code, Tests, Metadata, Execution, Lineage, Data Quality, ...)
TEAMS
PROJECTS
Git repositories, cloned directly from the chosen Git provider.
USERS
Prophecy users are created from your identity provider. They authenticate with their GIT and Spark through Personal Access tokens.
FABRICS
Definitions of execution environments (e.g. databricks workspaces).
REST API CONNECTION
PERSONAL ACCESS TOKEN
PERSONAL ACCESS TOKEN
REST API CONNECTION
GIT
REPOSITORIES
All the source of truth code is stored safely within your git repositories.
USERS
Git provider users, authenticated from your identity provider.
USER INFO
DATABRICKS
USERS
Databricks users, authenticated from your identity provider.
CLUSTERS & DATA
All the data is stored safely withing your data plane. The execution happens directly on Databricks.
IDENTITY MANAGEMENT
(Prophecy, Active directory, Saml, Ldap, Okta, ...)

Prophecy supports the following Git providers:

- **Prophecy-managed —** Automatically sets up the connectivity between Prophecy and the repositories. Prophecy-managed is based on open-source GitTea.
- **GitHub (including GitHub Enterprise) —** Authenticates using per-user personal access tokens or OAuth.
- **Bitbucket (including Bitbucket self-hosted) —** Authenticates using per-user personal access tokens.
- **GitLab (including GitLab self-hosted) —** Authenticates using per-user personal access token.
- **Azure DevOps —** Authenticates using per-user personal access tokens.

Security-conscious enterprises that use Git providers within private networks behind firewalls have to add the Prophecy Control Plane IP address (3.133.35.237) to their private network allow list or the Git provider allow list.

SQL

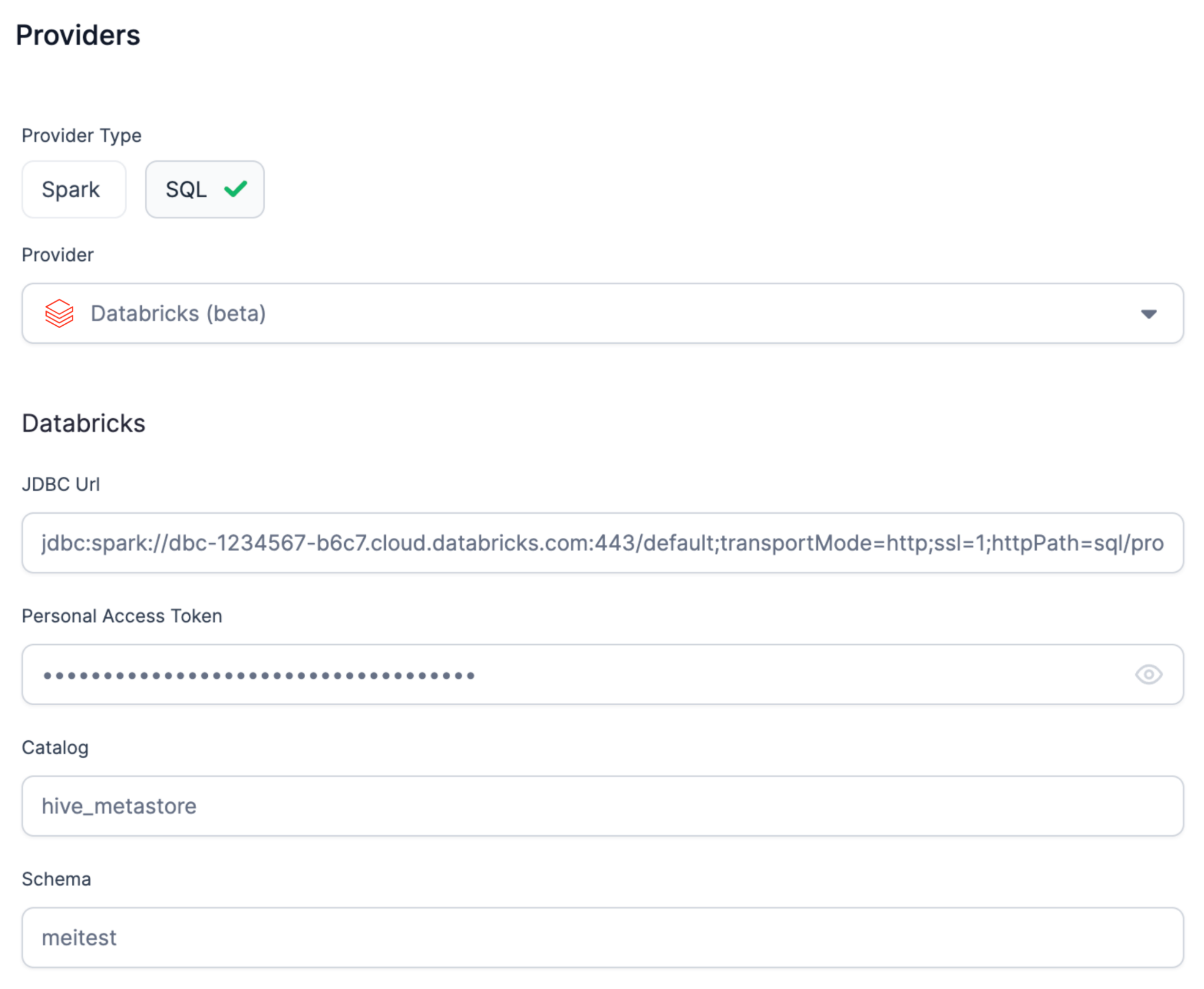
Prophecy 3.0 introduces a feature-rich visual environment that runs natively on Databricks using dbt Core. This functionality expands the existing Prophecy offering, which runs natively with PySpark and Scala.

Users well-acquainted with dbt Core will find familiar entities in Prophecy:

- **Projects:** Prophecy projects are mapped to dbt projects. They inform dbt about the context of your project and how to transform your data (build your datasets). Each project corresponds to a GIT repository. Projects include:
    - **Models:** A set of visual gems to transform data. Prophecy models are mapped to dbt models. Each model lives in a single file and contains logic that either transforms raw data into a dataset ready for analytics or — more often the case — is an intermediate step in such a transformation.
    - **Seeds:** CSV files with static data you can load into your data platform with dbt.
    - **Sources:** A way to name and describe the data loaded into your warehouse by extract-and-load tools.
- **Environments:** A list of entities from the Databricks environment that are associated with the fabric.

Prophecy supports SparkSQL in addition to PySpark and Scala. The execution engine has dedicated compute to process the data. The size of compute clusters can impact performance depending on the number of users running concurrently and the type of workloads running on Spark (please refer to the provider's sizing guide to ensure your clusters are sized appropriately for the workload).

Data fabrics are logical execution environments. The following parameters must be defined to connect with the Databricks cluster via JDBC:



Providers

Provider Type

Spark [✓ SQL]

Provider

Databricks (beta)

Databricks

JDBC Url

jdbc:spark://dbc-1234567-b6c7.cloud.databricks.com:443/default;transportMode=http;ssl=1;httpPath=sql/pro

Personal Access Token

••••••••••••••••••••••••••••••••••••

Catalog

hive_metastore

Schema

meitest

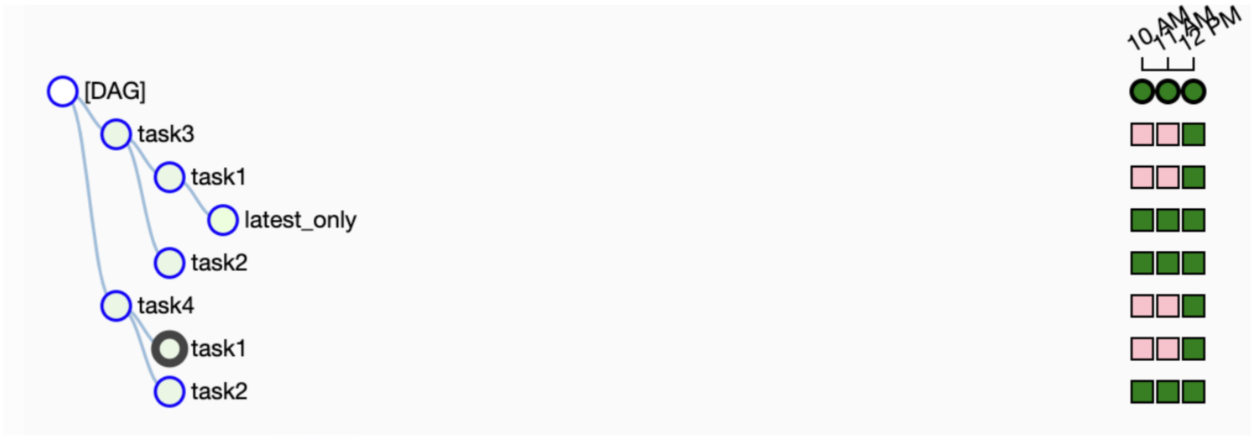
Airflow

Cloud providers offer managed Airflow services. Prophecy currently supports Amazon MWAA and GCP Cloud Composer, with Azure support coming soon.

Prophecy jobs executes pipelines running on Spark. These jobs are triggered on a schedule to keep data up to date. The jobs need an orchestrator to run in a particular time and sequence and to inform administrators of their running status. The Airflow platform is a tool for describing, executing and monitoring workflows. Prophecy enables users to visually orchestrate data pipelines on a specified schedule or condition using Apache Airflow.

Prophecy has the concept of data fabrics. The user can choose Airflow as its provider type. Airflow connection information and configurations (e.g., DAG location) can be defined at the fabric level. A user can create a new job and choose Airflow as the scheduler after creating the fabric. Schedule interval is also defined at the job level. An administrator can manage and submit Jobs directly from Prophecy while monitoring the jobs using Airflow providers.

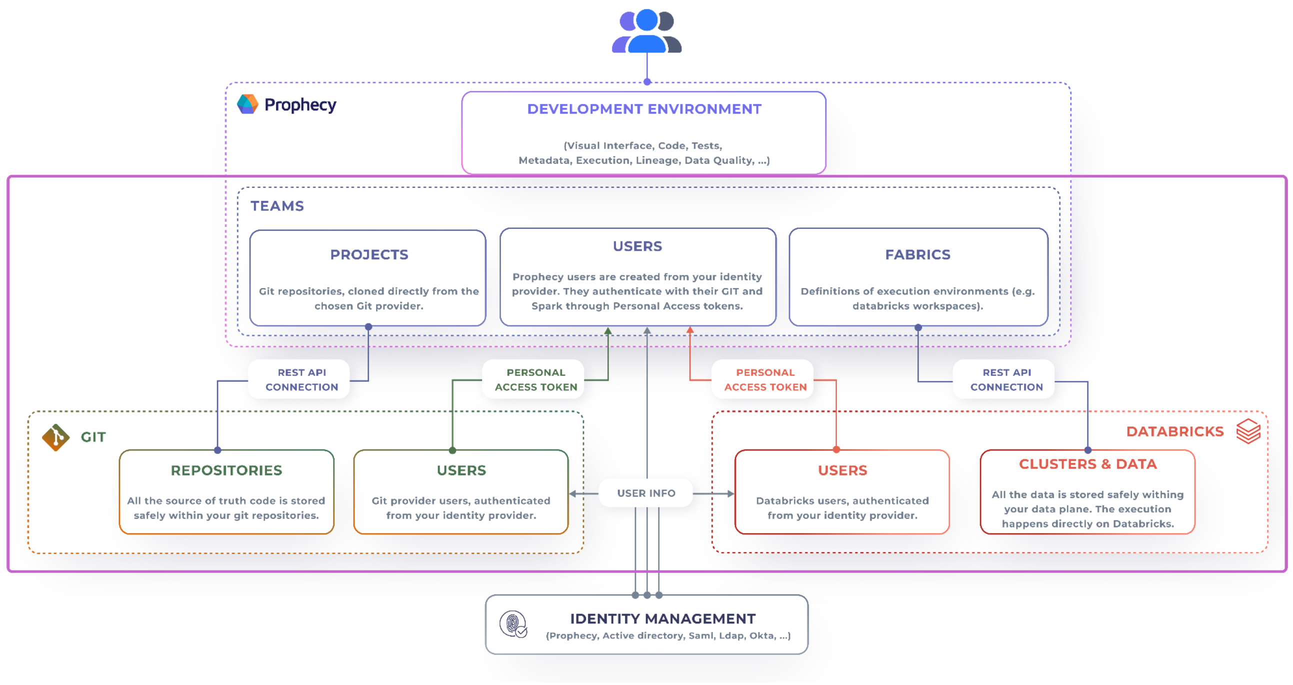
Prophecy pushes the DAG to its location. It waits for Airflow to pick up the new DAG every 10 seconds. If Airflow fails to recognize the DAG within 30 tries, the user will be notified as "DAG not found."



Prophecy converts visual pipeline jobs into Airflow DAG definitions in Python. Users do not need to code their Airflow DAGs manually. They can use the monitoring screen to track statuses and logs.

Prophecy architecture

Users and teams

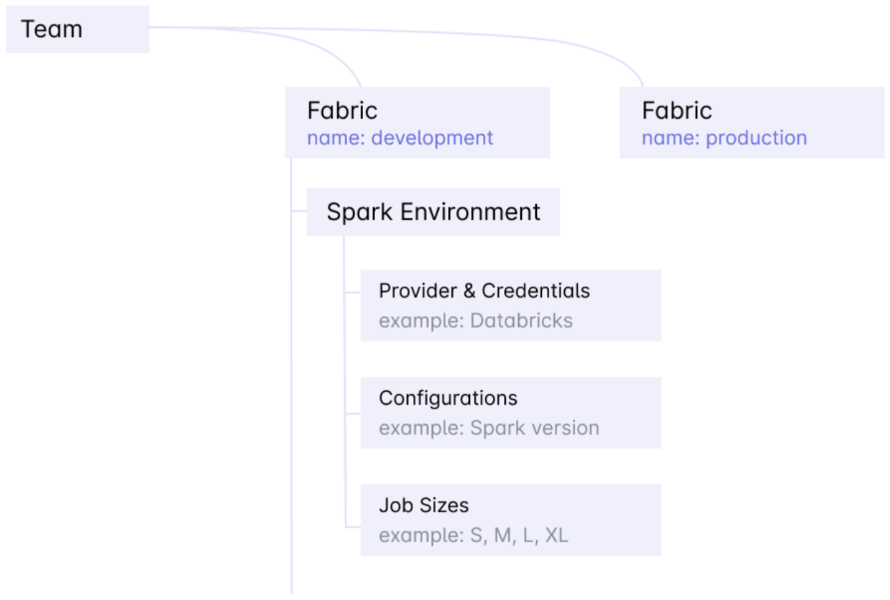

Prophecy
DEVELOPMENT ENVIRONMENT
(Visual Interface, Code, Tests, Metadata, Execution, Lineage, Data Quality, ...)
TEAMS
PROJECTS
Git repositories, cloned directly from the chosen Git provider.
USERS
Prophecy users are created from your identity provider. They authenticate with their GIT and Spark through Personal Access tokens.
FABRICS
Definitions of execution environments (e.g. databricks workspaces).
REST API CONNECTION
PERSONAL ACCESS TOKEN
PERSONAL ACCESS TOKEN
REST API CONNECTION
GIT
REPOSITORIES
All the source of truth code is stored safely within your git repositories.
USERS
Git provider users, authenticated from your identity provider.
USER INFO
DATABRICKS
USERS
Databricks users, authenticated from your identity provider.
CLUSTERS & DATA
All the data is stored safely withing your data plane. The execution happens directly on Databricks.
IDENTITY MANAGEMENT
(Prophecy, Active directory, Saml, Ldap, Okta, ...)

A Prophecy team is a logical grouping of users working together. Prophecy is designed to perform and scale with the number of users and their concurrent workloads. Every Prophecy user belongs to a team. Users are automatically assigned to their default team after they sign up for Prophecy. They may also have access to other existing teams, and they can create new teams. An administrator can invite many users to the team that they manage. An administrator can also assign proper access to their team members — for example, only admins can create or delete execution environments (fabrics).

A team of Prophecy users can access appropriate development and execution environments. Each user in the team must provide their identity to authenticate with the execution environment to which they have access.

The token used to authenticate with the execution environment is managed by Databricks (see more information here). If the user's Databricks token is expired or is invalid, they will be asked to update their Databricks token. The user will be automatically logged out of Prophecy after 48 hours of inactivity.

Data fabrics



A data fabric is the execution environment of data pipelines where code will run and data may be persisted. Not all users within a team can create a data fabric. A team administrator can create data fabrics, and other team members can use the fabric. Each fabric is defined with a functional purpose, either as a development, testing or production environment. Each environment is mapped to a Databricks Workspace and accessed via a token.

The typical flow for a new Prophecy team starts with an administrator creating a data fabric called "dev" for development. They will create a team called "marketing_adhoc" and invite marketing users that typically run ad hoc workloads on their data to the team. Marketing users now have access to the "dev" data fabric, and they can create pipelines that run on "dev." Business users do not have access to the "prod" fabric because they will not deploy their pipelines to production.

As mentioned above, data fabrics are logical execution environments. The size of the execution environment depends on the number of users running concurrently and the type of jobs running on Spark. For example, if pipelines are complex, with many concurrent workloads, an XL Spark cluster of ten i3.xlarge instances, each with 40 CPUs and 300GB of memory, might be a good fit (see Databricks for sizing examples).

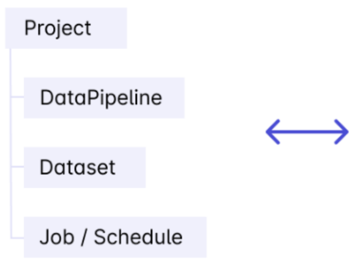
An administrator can configure the following within a data fabric.

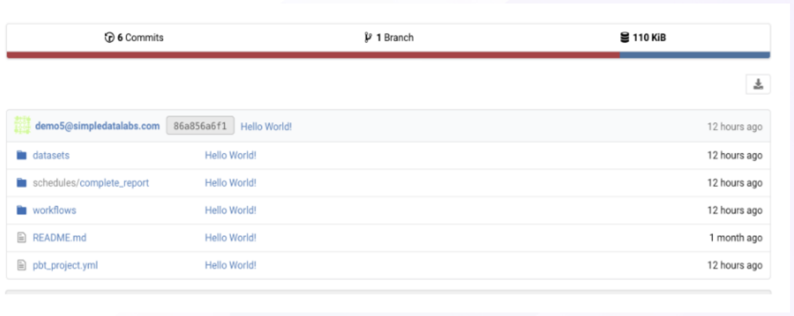
- **Cluster —** The Databricks runtime version, auto-terminate, instance types and the number of instances. This is the cluster that runs your data pipelines.

- **Job —** The default size of the cluster that's going to be created for the job to run. This setting should be the commonly used cluster size.
- **Library —** The Scala and Python libraries written by Prophecy provide additional functionalities on top of Spark. These libraries would get automatically installed in your Spark execution environment when you attach or create a cluster and are also publicly available on Maven Central and PyPI.

Projects

A Prophecy "project" is the primary unit of development and deployment. Prophecy translates visual components into open-source code. It supports multiple project types, including PySpark, Scala and SQL.





demo5@simpledatalabs.com	86a856a6f1	Hello World!	12 hours ago
datasets		Hello World!	12 hours ago
schedules/complete_report		Hello World!	12 hours ago
workflows		Hello World!	12 hours ago
README.md		Hello World!	1 month ago
pbt_project.yml		Hello World!	12 hours ago

The business logic that represents the assets within a project — pipelines, datasets and jobs — is represented as code. A project can be connected to a repository on GitHub or a folder in a repository. A repository may have multiple projects. Each project can be mapped to a folder. These projects are not required to have the same project type — for example, you may have a Python project in one folder and a Scala project in a different folder in the same repository. You cannot mix project types within the same folder.

When a user creates a new project, they will authenticate with their GitHub account. Software development best practices are enforced using Git. The user will develop their pipelines within a project, and code will be committed and merged. CI/CD can be performed using GitHub actions or other third-party tools.

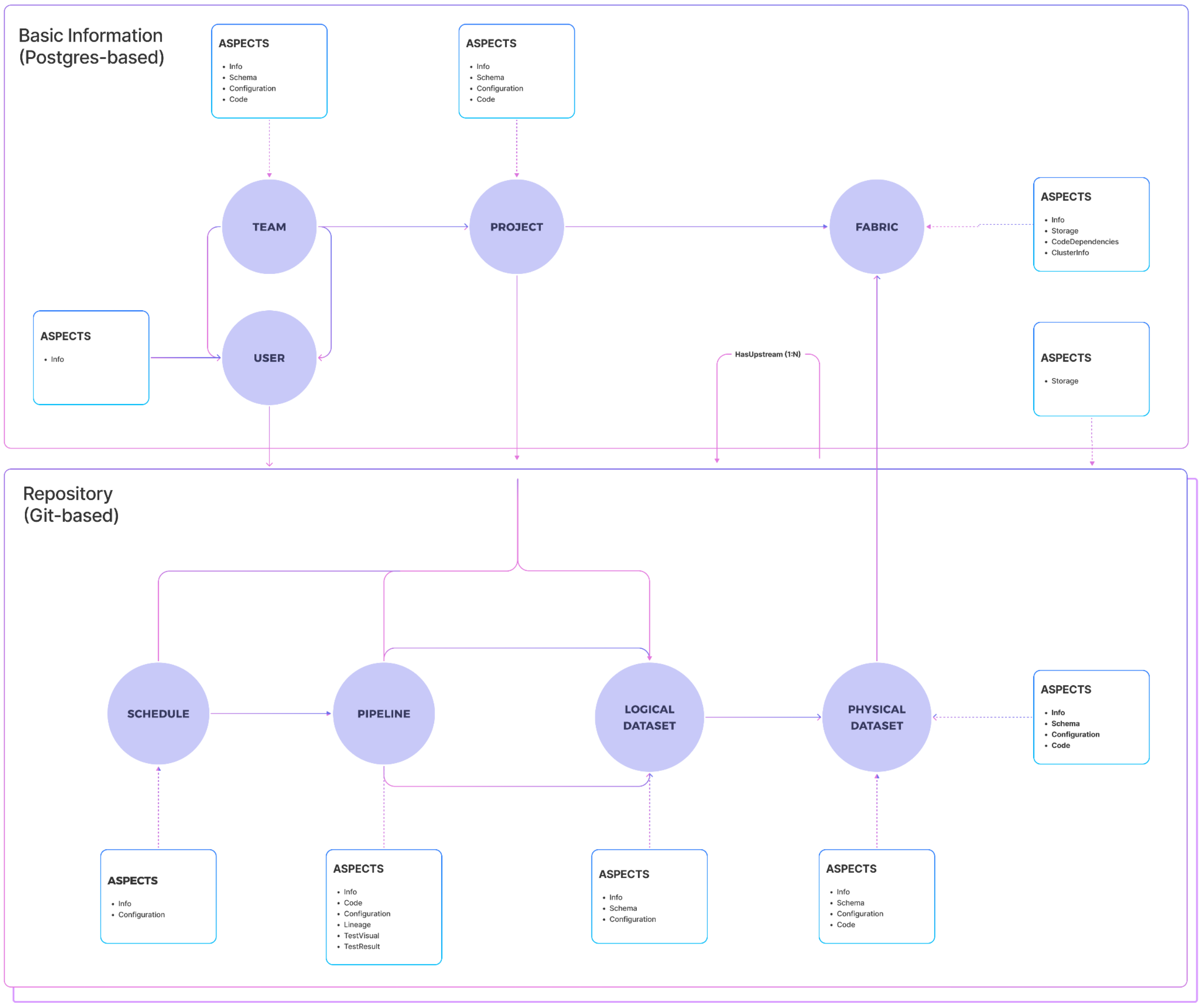
Users can create a new project from scratch or import from an existing project located in a different repository. Importing from another project makes it easy and seamless to share and reuse existing assets.

For sharing projects, the user can create a new project and identify an existing project upon which the new project is dependent. This means the new project can access the pipelines, datasets, subgraphs, custom gems and UDFs from an existing project. All dependencies are stored at the project level. When a dependency is added to a single pipeline, it becomes available by default to all the other pipelines within the same project. Users can enable or disable a given dependency within each pipeline.

Unified metadata

Prophecy produces metadata that describes various aspects of Prophecy entities. An entity represents the primary assets in the metadata system (such as project, pipeline and user, shown in blue in the diagram below). Aspects store details that describe the entities — they contain the content or points to the external systems where content is stored. They can be evolved independently without affecting other aspects. For a pipeline, the info aspect stores information in Postgres, the code aspect stores the code in Git, and the test aspect stores in unit tests. A user can drill down into each entity and discover the components that make up an entity.

Metadata Architecture



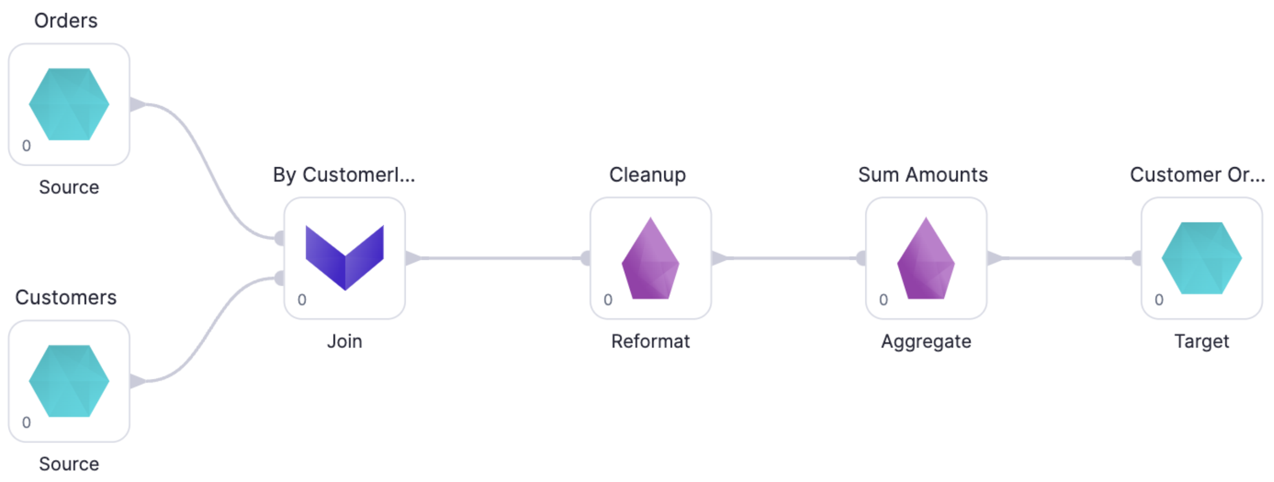
A user can access aspects using REST API and a personal access token. There are two components to the request URL: the base and the path. The base will depend on the endpoint that you use to access Prophecy. Example:

https://test.prophecy.io/api/md/graphql

Data pipelines

What is a pipeline?

A pipeline (formerly known as a "workflow") is a type of entity within Prophecy that represents how data flows. It's similar to a factory production line: you have sources and targets (file-based or event-based) with operations (gems) in between to enrich and transform a data product from raw to finished.



Pipelines are built with UI drag-and-drop features instead of with code. For example, for a pipeline that has multiple data sources from Databricks Filesystem (DBFS) in various formats:

- **Source gems** read the files
- A **join gem** combines the data on the same key(s) and pushes the joined results downstream for reformatting and aggregations
- The **target gem** writes its results to DBFS for analysts and data scientists to build models, reports and dashboards

A user can choose between batch or streaming when creating a pipeline. A unique set of gems are available for building the pipelines for streaming. Prophecy currently supports file-based, event-based and warehouse-based streaming. Users can use Kafka and S3 as sources and target gems. Splunk HEC is available as a target, with more coming soon.

Running a pipeline

Users can run pipelines interactively or by scheduling a job (see jobs). There are two ways to run the pipelines interactively:

1. Run the entire pipeline by clicking the Play button
2. Click the Play button on a specific gem (this option only executes the flow in the pipeline up to and including that gem, and lets users quickly identify issues and debug the relevant gem)

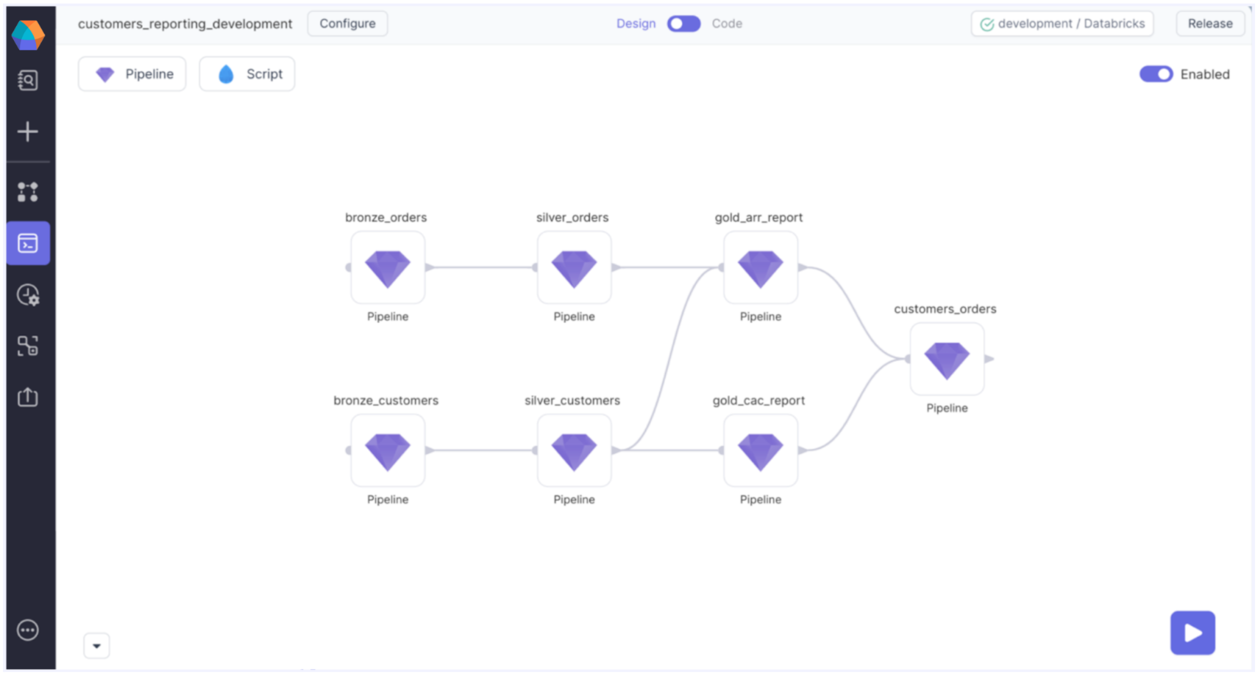
Pipeline interims allow users to troubleshoot and modify their pipeline by checking the output of a gem. Interims cache data samples that appear after executing a pipeline or a gem.

When running pipelines and jobs, users may be interested in execution metrics. For example, records read, records written, bytes read, bytes written, total time taken and data samples between components. These dataset-, pipeline- and job-related metrics are accumulated and stored on the data plane, where they can be viewed from the Prophecy UI.

Jobs

A pipeline on Databricks can be run interactively or by scheduling for automatic executions. Prophecy provides a low-code layer on top of Databricks jobs and also offers the option for users to use Airflow (See Airflow Integration) for orchestration.

A user can run one or multiple pipelines in a job.



Jobs can be defined with pipeline gems and script gems.

- **Pipeline gems** are defined by various sources, targets, transformations and aggregations to represent data flow
- **Script gems** allow users to define their jobs using Python

You can add gems to the canvas to determine which pipelines will run during the job. To define dependencies between the pipelines within the job, you can simply connect them by dragging and dropping the edges between gems.

Some users may want to have fine-grained control over who has access to view, manage and administer the Databricks jobs created from Prophecy. Prophecy allows the user to define job permission settings, using functionally equivalent to the Databricks interface.

Deploying a job on Databricks requires the user to release the project from Prophecy UI. As soon as the project is released, the job will start appearing on the Databricks Jobs page.

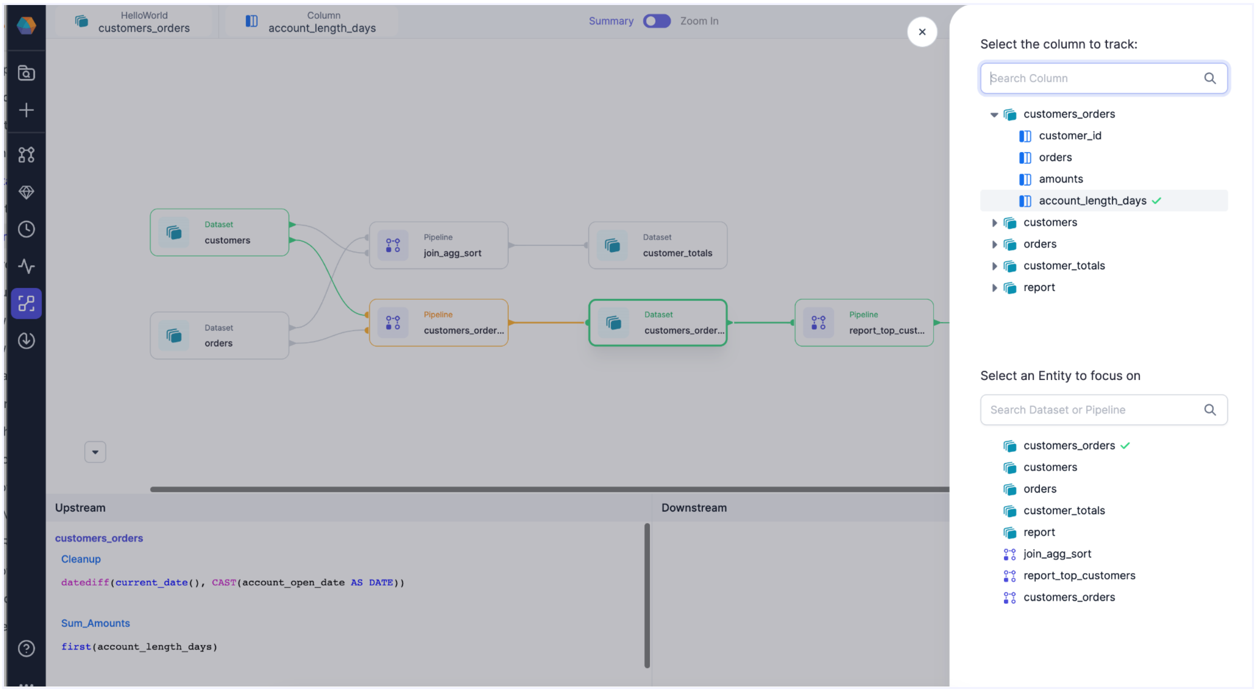
Prophecy supports two different job deployment modes, each of which has various impacts on job cost and parallelism:

- **Multi-job cluster mode —** Each component of job will spawn a separate cluster of its own
- **Single-cluster mode —** Each component of job will run on the same cluster

Prophecy's monitoring page shows the status of all jobs deployed. It also provides the status of historical/current runs. Data quality observability is also available with the release of Prophecy 3.0.

Data lineage

Data lineage tells us about the lifecycle of data. In Prophecy, lineage is the process of understanding, recording and visualizing data as it flows from source to target. The lineage includes all transformations the data undergoes along the way. Knowing the source of a particular data set is not always enough to understand its importance, perform error resolution, understand process changes, and perform system migrations and updates.



customers_orders
Cleanup
```
datediff(current_date(), CAST(account_open_date AS DATE))
```
Sum_Amounts
```
first(account_length_days)
```

It's essential for data practitioners to understand how data is updated and by which transformations. Understanding data lineage improves overall data quality and allows data custodians to protect integrity and confidentiality throughout the data lifecycle.

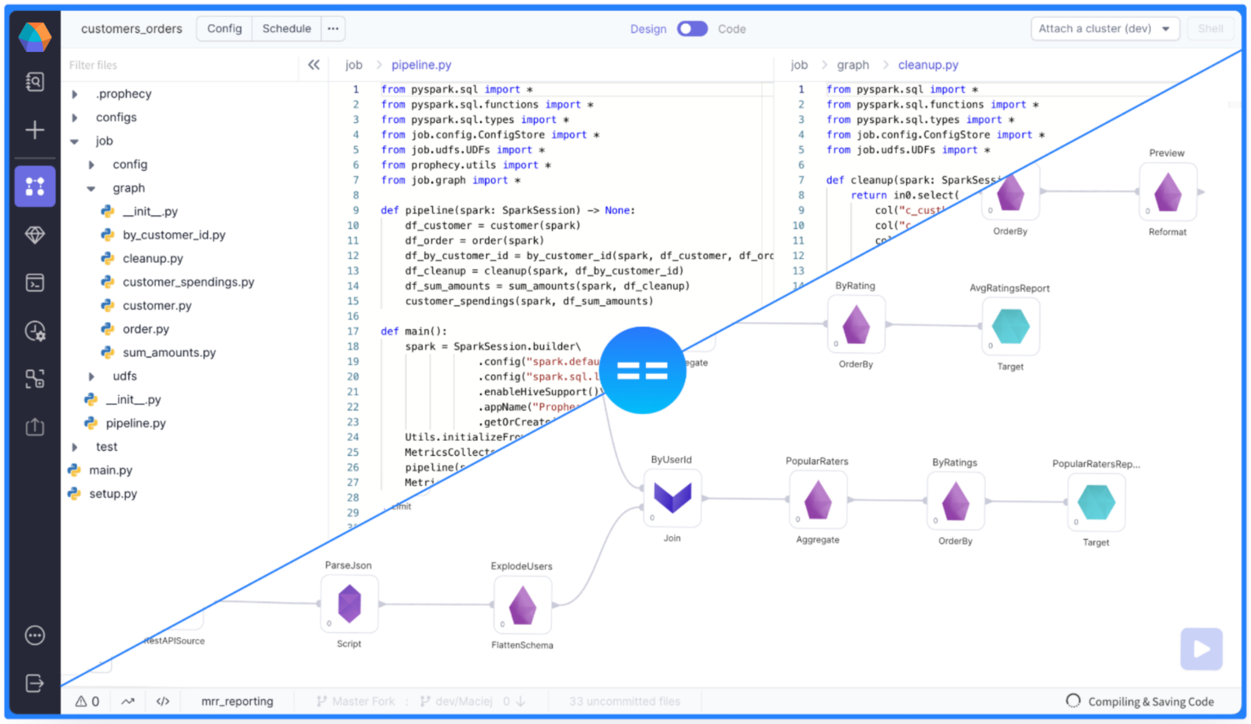
Data lineage enables a user to:

- Track errors in data processes
- Improve overall data quality
- Implement process changes and system migrations with lower risk and more confidence

- Combine data discovery with a comprehensive view of metadata
- Improve overall data governance

Watch a data lineage video here.

Code generation and Git



```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from job.config.ConfigStore import *
from job.udfs.UDFs import *
from prophecy.utils import *
from job.graph import *

def pipeline(spark: SparkSession) -> None:
    df_customer = customer(spark)
    df_order = order(spark)
    df_by_customer_id = by_customer_id(spark, df_customer, df_orc
    df_cleanup = cleanup(spark, df_by_customer_id)
    df_sum_amounts = sum_amounts(spark, df_cleanup)
    customer_spendings(spark, df_sum_amounts)

def main():
    spark = SparkSession.builder\
                .config("spark.defau
                .config("spark.sql.l
                .enableHiveSupport()\
                .appName("Prophe
                .getOrCreate
    Utils.initializeFro
    MetricsCollect
    pipeline(s
    Metri
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from job.config.ConfigStore import *
from job.udfs.UDFs import *

def cleanup(spark: SparkSessi
    return in0.select(
        col("c_cust
        col("c
        co
```

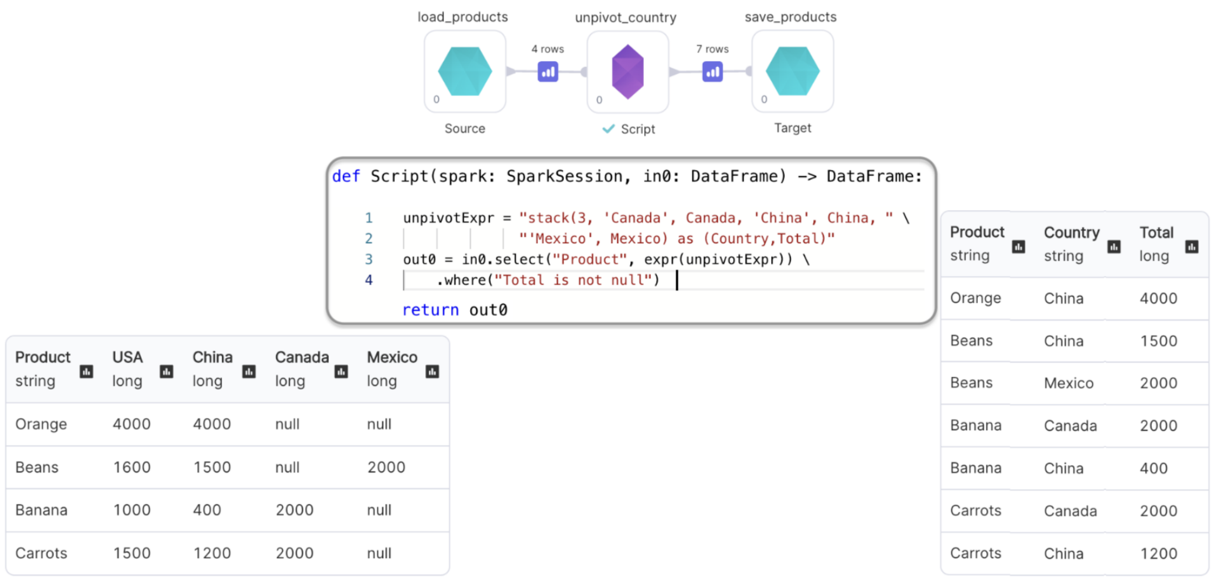
Visual == Code on PySpark

A project in Prophecy is a fully-fledged Git repository that can integrate with the user's favorite Git provider. Every change in gems (the visual elements) generates high-quality code in PySpark, Scala or SQL. The generated code is committed to a specific branch and follows all the best software engineering practices.

Extensibility

Easy usability should not result in a lack of flexibility and customization. Prophecy is designed to be extensible and pluggable. It has pre-built visual components for standard inputs, outputs and transformations. Users can add their gems to read and write to internal systems, and standardize everyday operations (e.g., security ops with encrypt, decrypt and anonymize). Prophecy frameworks are shareable Git projects and used as libraries across teams.

Custom gems



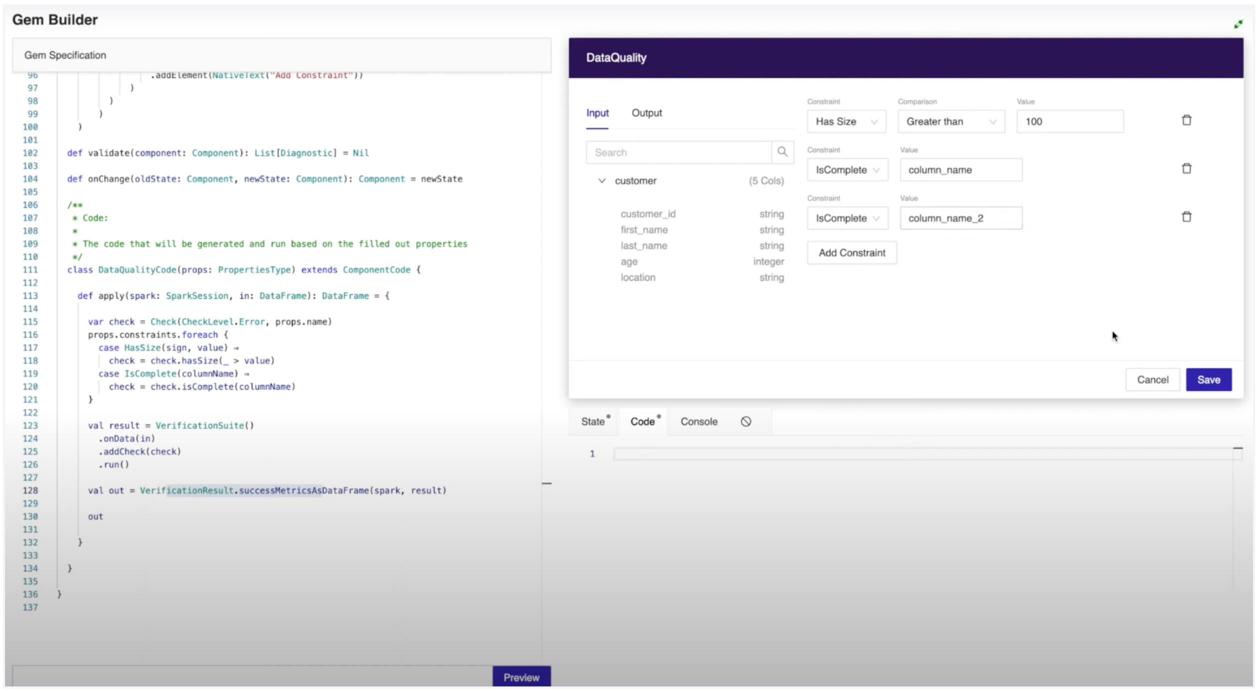
```
def Script(spark: SparkSession, in0: DataFrame) -> DataFrame:

    unpivotExpr = "stack(3, 'Canada', Canada, 'China', China, " \
                  "'Mexico', Mexico) as (Country,Total)"
    out0 = in0.select("Product", expr(unpivotExpr)) \
        .where("Total is not null")

    return out0
```

Product string	USA long	China long	Canada long	Mexico long
Orange	4000	4000	null	null
Beans	1600	1500	null	2000
Banana	1000	400	2000	null
Carrots	1500	1200	2000	null

Product string	Country string	Total long
Orange	China	4000
Beans	China	1500
Beans	Mexico	2000
Banana	Canada	2000
Banana	China	400
Carrots	Canada	2000
Carrots	China	1200

Prophecy offers several custom gems out-of-box. One of the most popular gems is the script gem, which provides a SparkSession and allows users to run custom code. It can be used to write any ad hoc code. Custom gem logic can be shared with other users within the team and organization.

Gem builder



```
                .addElement(NativeText("Add Constraint"))
              )
            )
          )
      )

  def validate(component: Component): List[Diagnostic] = Nil

  def onChange(oldState: Component, newState: Component): Component = newState

  /**
   * Code:
   *
   * The code that will be generated and run based on the filled out properties
   */
  class DataQualityCode(props: PropertiesType) extends ComponentCode {

    def apply(spark: SparkSession, in: DataFrame): DataFrame = {

      var check = Check(CheckLevel.Error, props.name)
      props.constraints.foreach {
        case HasSize(sign, value) ⇒
          check = check.hasSize(_ > value)
        case IsComplete(columnName) ⇒
          check = check.isComplete(columnName)
      }

      val result = VerificationSuite()
        .onData(in)
        .addCheck(check)
        .run()

      val out = VerificationResult.successMetricsAsDataFrame(spark, result)

      out

    }

  }
}
```

customer_id	string
first_name	string
last_name	string
age	integer
location	string

Constraint	Comparison	Value
Has Size	Greater than	100

Constraint	Value
IsComplete	column_name
IsComplete	column_name_2

As mentioned earlier, each Prophecy pipeline is assembled with individual gems that perform transformations on data. While Prophecy offers dozens of gems out-of-the-box, some data practitioners want to extend this idea and create their gems. Our Gem Builder allows users to extend their Prophecy capabilities by adding custom gems. The Gem Builder is an excellent tool for data engineers to include custom code that's easily accessible to other users. Like regular gems, custom gems are available for users to drag and drop into their pipelines.

Building a custom gem is simple:

1. Navigate to the gem listing to review Prophecy-defined and user-defined gems
2. Add a new gem or modify an existing gem — specify the gem name, preferred language and gem category
3. Paste/Write your code specification at the prompt
4. Click "Preview" to review the UX
5. Fill in some values and click "Save" to check the Python or Scala code generated
6. When the gem is ready, hit "Publish"! The new custom gem is available to use in pipelines!

Dependencies

Using dependencies is one of many ways to extend Prophecy. Prophecy offers users the capability to define third-party or custom code libraries. This extension allows users to use third-party or custom code in data pipelines and Jobs. They can be written in Java, Scala or PySpark and connected to data pipelines by pointing to Maven or PyPi coordinates.

UDFs

User-defined functions (UDFs) are another way to extend Prophecy. A UDF is a function defined by a user that allows custom logic to be reused in the user environment. Prophecy supports many types of UDFs to allow for distributing extensible logic. The UDFs can be utilized anywhere in the pipeline.

Using Prophecy to construct data pipelines

Gems

As mentioned above, Prophecy uses gems to build pipelines and jobs. A gem is a unit of functionality ranging from reading, transforming, writing and various other ad hoc operations on data. A gem instance has its configurations and produces output code. Each gem instance can be seen as a Spark DataFrame. Prophecy provides gems out of the box. They're visual components that a user can drag and drop without writing code.

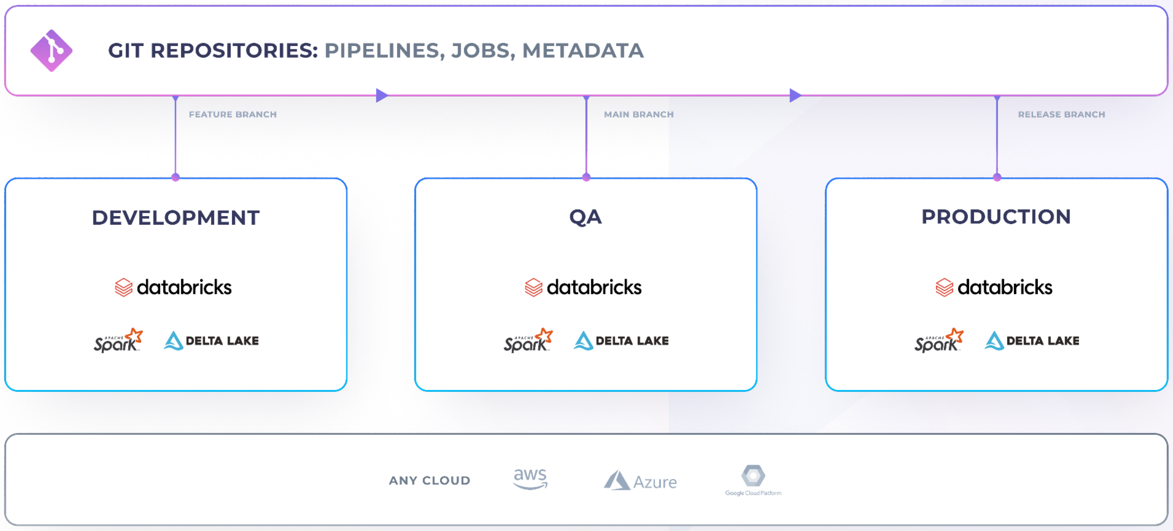
The Gem Drawer, located in Prophecy's pipeline editor, organizes gems into several categories. See Appendix for more details.

gem	Category	Description
⬢	Source and Target	Gems that conduct loading and writing data
◆	Transform	Gems that help transform data
⌄	Join and Split	Gems that merge or split data
⬢	Custom	Custom-built for a particular use case using the Gem Builder then made available to the broader user group.
⬟	Subgraph	A gem that can group multiple gems for reusability.

Deploying pipelines (CI/CD)

Continuous integration and continuous deployment (collectively known as CI/CD) is a set of methodologies used to continuously deliver new features. CI/CD is used to solve the problems development and operations teams often encounter when integrating new code.

But implementing CI/CD can be challenging. That's why Prophecy integrates seamlessly with Git to provide a centralized environment to persist code for data pipelines and jobs. Users can utilize Git to apply their DevOps best practices.



Prophecy works best with physical environments connected to different stages of development. A common example is a setup with three fabrics — development, QA and production — where each environment has its independent data, metastore, clusters and permissions.

Steps for deploying pipelines

The following steps work within two environments: development and production.

- The **development environment** is accessible to the entire organization (developers, analysts, support) and is connected to Databricks development workspace with mock data
- The **production environment** is only accessible to the production support team and is connected to our Databricks production workspace with real data

To set up entities:

1. Create two teams:
    - **developers —** A superset of all the teams, which includes developers and members of the prod_support team
    - **prod_support —** A team composed of members who have privileged production access permissions
2. Create two fabrics:
    - **development —** Owned by the developers' team
    - **production —** Owned by the prod_support team
3. Set up projects: Create projects, which should be owned by the developers' team
4. Set up jobs: Create two jobs for every single set of pipelines in a project ready for scheduling:
    - **Job_development —** Jobs built by the developers for integration and testing purposes
    - **job_production —** Jobs built by the prod_support team, based on the development Jobs, which will run in the production environment
5. Test the entire data flow on the development environment
6. Deploy tested data flow to the production environment
    - Log in as a production support engineer (only a production support engineer can perform the deployment)
    - Duplicate the job on the production fabric by calling the job and passing the parameter for Fabric ID using PBT
    - Set appropriate pipeline configurations then enable the job
7. Commit any remaining changes and release the pipeline.

Prophecy automatically takes care of the release process by building the pipelines, running unit tests and deploying the pipeline JARs/wheels alongside the job definition directly to Databricks (or AirFlow).

Unit testing

Writing good unit tests is one of the key stages of the CI/CD process. It ensures the changes developers make to projects will be verified, and all the functionality will work correctly after deployment.

Prophecy makes the process of writing unit cases easier by providing an interactive environment via which unit test cases can be configured across each component.

Two types of unit test cases can be configured through the Prophecy UI:

1. Output rows equality
2. Output predicates

Output rows equality

This type of testing is the most basic way to check the functionality of a gem's code by providing a sample input and a sample output. If the function works correctly, the input should always be computed to generate the output. It's a great way to verify code keeps working the same way as the codebase evolves and new versions are released.

Output predicates

You can use this type of testing for more advanced logic, using Spark expressions as predicates. Consider requiring multiple rules for the test to pass.

Prophecy also supports generating sample input data automatically from the source DataFrame; this option can be enabled while creating unit tests.

Working with other tools

Users may want to deploy pipelines from their Git, which enables users to work with a secure production environment that doesn't have to connect directly to Prophecy.

As mentioned earlier, Prophecy publishes all entities (e.g., pipelines, jobs, metadata, etc.) directly on Git. That means you can easily deploy that code to any chosen Databricks environment.

The Prophecy-built Tool (PBT) CLI can be used to build, test and deploy projects created by Prophecy that are present in your local filesystem. It can be used to integrate with your own CI/CD tools such as GitHub Actions. It allows you to utilize build systems such as Jenkins. Databricks Workflows or Airflow can be used for orchestration.

Prophecy also allows opening pull requests on a user's external Git provider in order to merge development branches to the base branch of a project's remote repository. During the PR creation process, Prophecy will redirect users to their external Git provider based on the template defined in the Advanced > Pull Request Template tab in a project's settings.

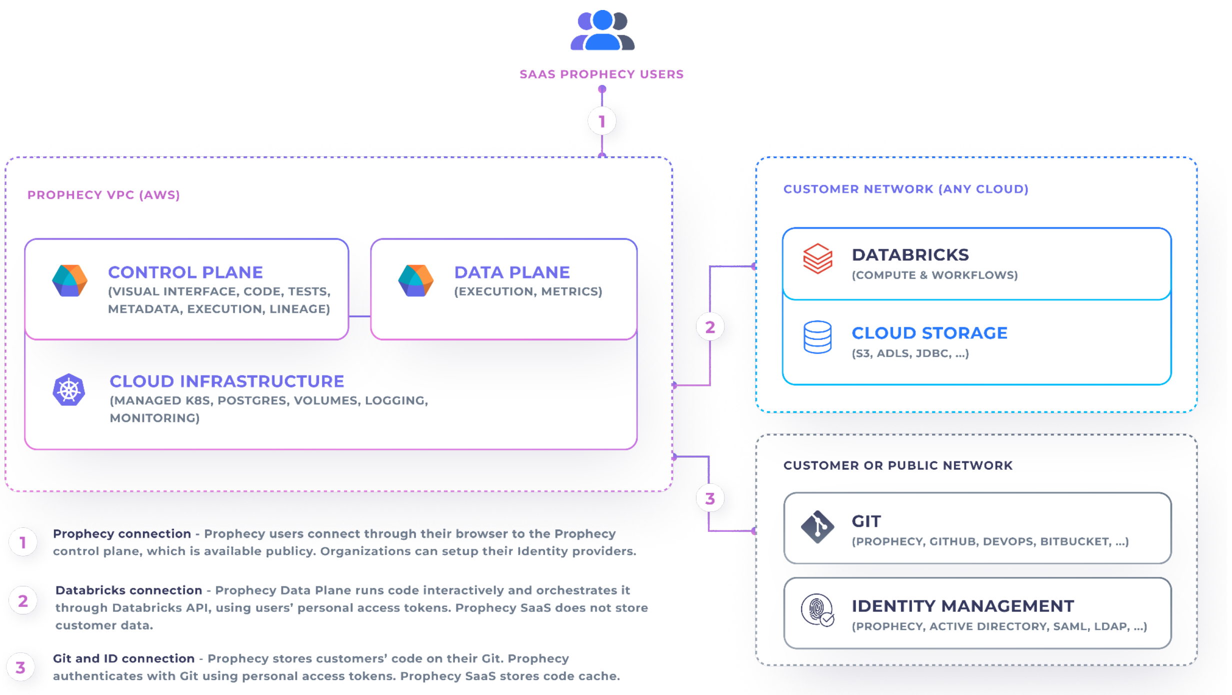
Prophecy installation and deployment

Choosing the right deployment model is critical for the success and scalability of the Prophecy platform. We want to provide the right configurations for each use case to ensure the system reliably provides high-performance pipelines and workflows.

Cloud-managed and on-premises infrastructure

Prophecy supports deployments on Amazon AWS, Microsoft Azure or on premises (on prem). It utilizes Kubernetes to manage its services and deployments. This paper will use Amazon AWS components to explain Prophecy's deployment model. The architecture is similar on Microsoft Azure and on prem with different terminologies. The easiest way to install and deploy Prophecy is through AWS Marketplace and Azure Marketplace.

Deployment architecture



SAAS PROPHECY USERS
1
PROPHECY VPC (AWS)
CONTROL PLANE
(VISUAL INTERFACE, CODE, TESTS, METADATA, EXECUTION, LINEAGE)
DATA PLANE
(EXECUTION, METRICS)
CLOUD INFRASTRUCTURE
(MANAGED K8S, POSTGRES, VOLUMES, LOGGING, MONITORING)
2
3
CUSTOMER NETWORK (ANY CLOUD)
DATABRICKS
(COMPUTE & WORKFLOWS)
CLOUD STORAGE
(S3, ADLS, JDBC, ...)
CUSTOMER OR PUBLIC NETWORK
GIT
(PROPHECY, GITHUB, DEVOPS, BITBUCKET, ...)
IDENTITY MANAGEMENT
(PROPHECY, ACTIVE DIRECTORY, SAML, LDAP, ...)

1 **Prophecy connection** - Prophecy users connect through their browser to the Prophecy control plane, which is available publicy. Organizations can setup their Identity providers.

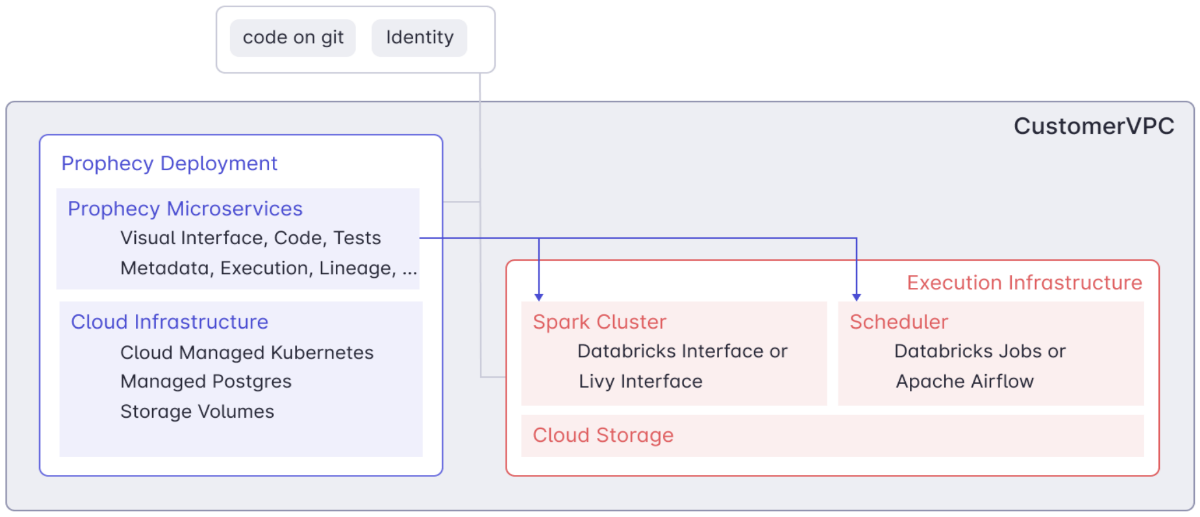
2 **Databricks connection** - Prophecy Data Plane runs code interactively and orchestrates it through Databricks API, using users' personal access tokens. Prophecy SaaS does not store customer data.

3 **Git and ID connection** - Prophecy stores customers' code on their Git. Prophecy authenticates with Git using personal access tokens. Prophecy SaaS stores code cache.

Public SaaS Architecture

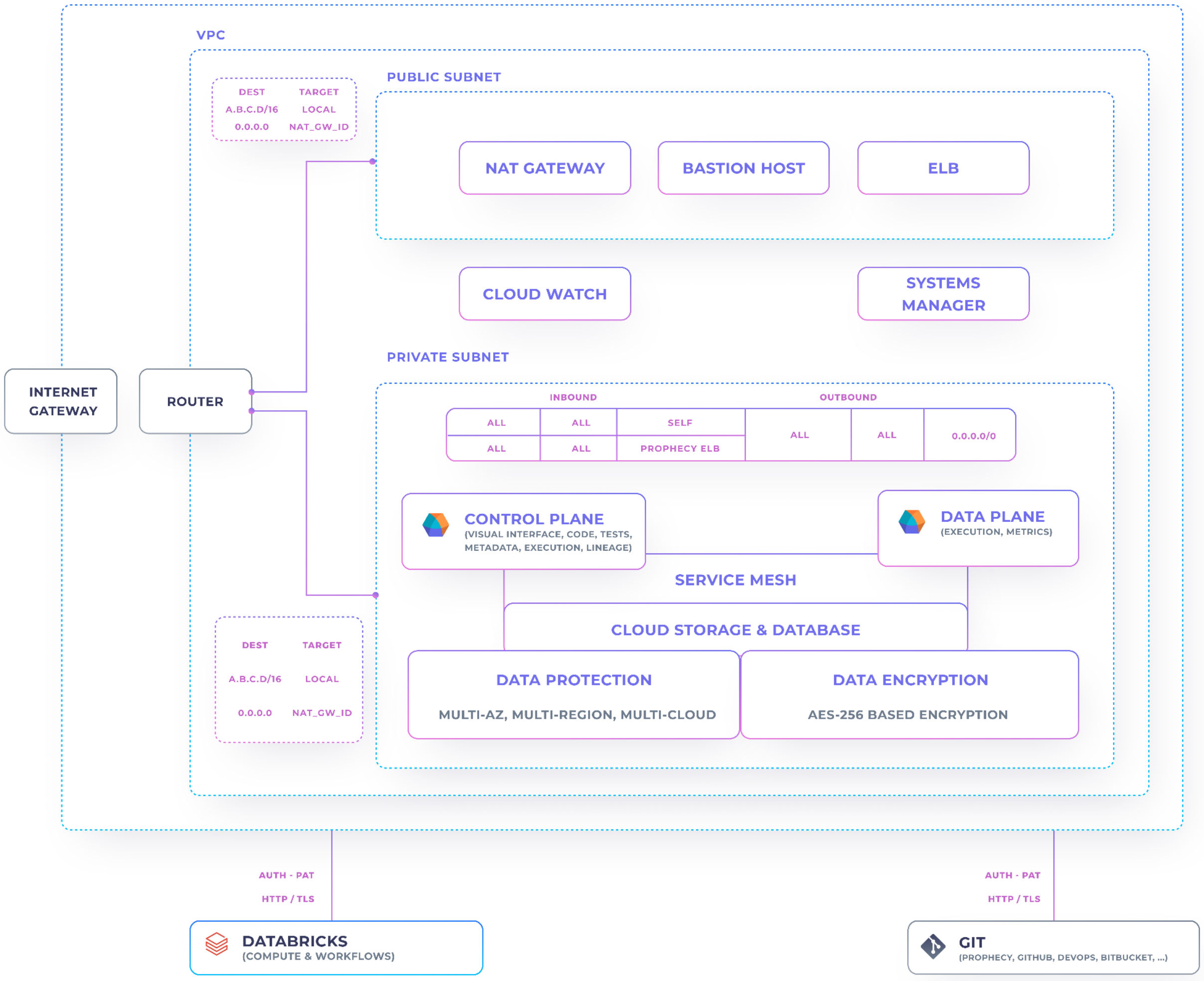
The cloud-managed (SaaS) deployment hosts Prophecy in its managed VPC. A public subnet and a private subnet reside in a managed VPC.

- The **public subnet** hosts NAT Gateway, Bastion Host and ELB to manage connectivity and external requests.
- The **private subnet** hosts Prophecy deployments and services using Kubernetes. Network traffic management, data storage, protection and encryption also reside in the private subnet.



Private deployments have a similar architecture, except they're deployed within the user's VPC or private network. It's essential to understand that Prophecy does not persist user data.

Prophecy deployment components



PROPHECY PUBLIC SAAS SECURITY & GOVERNANCE AT A GLANCE

Prophecy can handle management and execution requests at scale. A Kubernetes cluster is deployed in the private subnet. Cluster is deployed using Terraform or CloudFormation. The Control Plane manages the global decisions whenever a user makes a request (e.g., creating pipelines or jobs, or checking metadata). Every user action is managed using the Control Plane.

Prophecy users send requests to the Control Plane. The Control Plane receives requests and schedules them. The Data Plane receives decisions from the Control Plane and executes the requests. For example, if a Control Plane receives a request to create a pipeline, the creation of the pipeline code happens on the Data Plane. The execution services on the Data Plane connect to a Spark environment and run Spark jobs on a Spark cluster. In other words, the Data Plane translates Prophecy pipelines into Spark code and tells the Spark cluster to run them by making API calls. The execution of jobs can be immediate or scheduled for later.

Installation requirements include Kubernetes Version: 1.14+. For private deployments, images must be accessible from a secure container registry available at gcr.io. You can configure the location to use an internal container registry.

Kubernetes is a highly scalable platform for managing containerized workloads and services. It's important to size the platform correctly — the following table provides general guidelines:

For a small cluster up to 25 users.

Namespace	Description	# Cores	# RAM	Block/File Storage
Control Plane	Main services (front-end, code editor, metadata, lineage, etc...)	34 Cores	68GB	150GB
Data Plane	Services serving as a bridge between Spark and Prophecy UI	6 Cores	10GB	10GB
Platform	Backup (2x/day, configurable), monitoring, logging services (optional)	4 Cores	8GB	200GB

Prophecy requires block or file storage to be available for the Kubernetes cluster. A Kubernetes cluster can be configured in the multi-availability zone or single availability zone mode. For multi-availability zone mode, the block or file storage requires volume binding mode specified as "waitforfirstconsumer". The storage is dynamically provisioned, and Prophecy supports any storage class out of the box.

ClusterRole permissions are optional for Custom Resource installation. Users can create the required CRDs by deploying a single Helm Chart (shared upon request) if the role is unavailable.

For networking, Prophecy supports a Prophecy-managed or user-managed NGINX Ingress Controller. Both Kubernetes NGINX and official NGINX Controllers are supported. Prophecy supports Istio-based ingress gateways as well.

Wildcard certificates are used for path-based routing. Certificates can be managed by Prophecy or the user's cert manager. Certificates are automatically generated for every ingress resource created. To successfully resolve the service host names, Prophecy supports a Prophecy-managed or user-managed external DNS (or equivalent).

The requirements outlined earlier can support up to 25 Prophecy users actively working on pipelines at the same time. Prophecy supports vertical and horizontal scaling for additional concurrent users.

Prophecy utilizes an autoscaler to scale a cluster whenever needed. If the number of users increases by 10x, the Prophecy deployment will automatically scale out appropriately and require a proportionate amount of resources.

Approximate resource by the number of users:

Number of users	25	50	150
CPUs	44 vCPUs	80 vCPUs	230 vCPUs
Memory	86 GB	160 GB	460 GB
Disk space (with backups)	360 GB	720 GB	1440 TB

The numbers above are an estimation — the actual recommended resources may vary based on the complexity of a user's workload.

Security

Authentication

Users can authenticate with Prophecy using an external identity provider. Prophecy currently supports the following identity and access management systems:

- Azure Active Directory
- Okta (SAML)
- Active Directory (LDAP)

Prophecy supports SSO-based authentication to Prophecy's SaaS. Passwords are managed by the user's SSO system. Personal access tokens will be stored on encrypted storage with AES-256-based encryption and always transmitted from encrypted storage to microservices over TLS1.3 in a secure, segregated environment.

Authorization

When users access the underlying Spark execution infrastructure, the identity is passed through by Prophecy. The existing authorization mechanisms are respected. The minimum scope for the token for interactive execution is "Can Attach To" permissions. For cluster creation, "Can Manage permissions" are required. Any user with "Can Manage" permission for a cluster is allowed to configure whether a user can attach to, restart, resize and manage that cluster.

Encryption

Prophecy supports both encryption-at-rest and encryption-in-transit for sensitive data.

Encryption-in-transit is supported inherently. Prophecy supports working with encryption-at-rest solutions. If the public cloud has standard storage classes, Prophecy can provision storage with encryption-at-rest. If it's a private cloud, Prophecy requires a provisioned encrypted storage system and corresponding storage class.

Upgrades

Prophecy manages upgrades and patches automatically for fully managed (SaaS) accounts. If the user chooses a private deployment, Prophecy can manage upgrades automatically after "controlcenter.prophecy.io" on port 443 is whitelisted. Prophecy will provide a list of the latest containers, and users are required to manually deploy the upgrades if the environment is completely locked down, with no inbound or outbound traffic.

Monitoring

Prophecy deploys Loki, a log database that contains an index to efficiently access logs and provides storage to persist the logs, for log management. Prophecy deploys Loki using a log volume for storing logs and BoltDB to store indices. If Prophecy private SaaS is deployed on AWS, Loki can also leverage S3 to store the logs and DynamoDB to store the indices.

Prophecy runs a log client to pull logs from the services and push them to Loki. It utilizes Promtail as a log client. Promtail is configured to push logs at regular intervals to Loki. The default batch size for the intervals is 1048576 bytes. The maximum amount of time to wait before sending a batch, even if that batch isn't full, is one second (these are configurable values).

Promtail runs as a sidecar for aggregated log volume for Prophecy private SaaS deployment. In the fully managed setup, Prophecy deploys an RWM volume and mounts it to all service pods. The services write the logs to the RWM volume in different directories. Promtail runs in a pod with the same volume mounted. Promtail accesses these volumes and ships the logs to Loki.

Users can whitelist "controlcenter.prophecy.io" on port 443 to receive critical alerts from clusters. This will help Prophecy provide proactive support and enable Prophecy to push any new release availability information using the same channel. User clusters will be automatically updated with all the latest critical patches and new features.

For clusters with table ACL enabled, users may have limited access to catalogs, schemas and tables. It's recommended users set up the execution metrics tables beforehand. Data is stored in the workspace storage. The tables can be selected from the team view. The following table options are available at the time of team creation:

- **Pipeline metrics —** Contains metrics and code for pipeline runs
- **Component (dataset) metrics —** Contains metrics for individual component runs
- **Interim —** Contains samples of data, depending on the interim mode selected

Conclusion

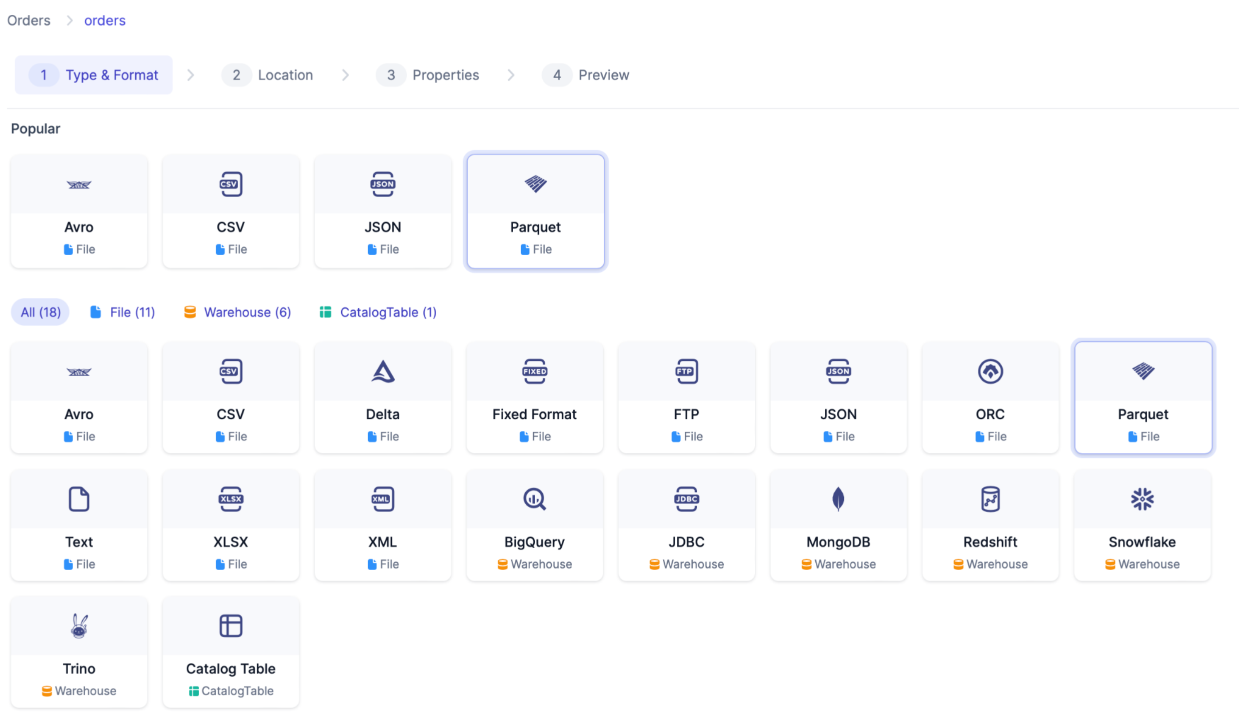
This guide is intended to highlight a few of our best practices for low-code Spark architecture. Each real use case and workload will be slightly different, and the best architectures are tailored to the specific requirements of the organization.

When designing an architecture, it's also critical to consider things like pipeline characteristics, data usage patterns and SLAs, which are too specific to cover here. For help choosing the right architectural strategy for your particular use cases, we recommend you engage directly with Prophecy and Databricks data engineers for an architecture and operational review.

Appendix

Sources and targets Gems

Sources and targets are a set of gems that load data from various sources and write to different targets. These gems support batch or streaming pipelines (the user can define a pipeline as a batch or streaming during the creation process). Prophecy supports many files, data warehouses and data catalogs as sources. Users can read their data and automatically infer the schema. Advanced options are available for parsing source data.



Special sources are also available, including streaming and lookup gems:

- **Streaming gems —** Support file-based (e.g. object store), event-based (e.g. Kafka) and more
- **Lookup gems —** Allow users to designate a particular dataframe as a broadcast dataframe

Spark ensures this data is available on every computation node. These lookups can be done without shuffling data. This is useful for looking up values in tables.

gem Name	Category	Type	Description
Source	File	Parquet, JSON, CSV, Avro, Kafka, FTP, Text, Delta, ORC, Fixed Format, XLSX	Reads files directly from storage (e.g. DBFS).
Source	Warehouse	Snowflake, Teradata, Redshift, MongoDB, Oracle, DB2, Salesforce, JDBC	Reads data from traditional data warehouses.
Source	Catalog Table	Hive / Delta	Reads data from tables connected to Metastores.
Lookup	Lookup	User-defined	Defines the columns to perform lookups.
Target	File	Parquet, JSON, CSV, Avro, Kafka, Text, Delta, ORC, Fixed Format, DataCreatorFormat	Writes transformed data to storage (e.g. DBFS)
Target	Warehouse	Snowflake, Redshift, Salesforce, JDBC	Writes transformed data to data warehouses.
Target	Catalog Table	Hive / Delta	Writes data to tables connected to metastores.

Transform Gems

Transformation gems are a set of gems that transform or aggregate data from upstream gems. These built-in gems allow users to perform simple and complex data transformations, enrichments and aggregations by configuring using a visual interface. The gems convert to code that runs on Spark clusters. Each gem represents a function that defines a Spark DataFrame.

gem Name	Description
Reformat	Formats a column's name or value
Filter	Filters DataFrames based on the provided filter condition
OrderBy	Sorts a DataFrame on one or more columns in ascending or descending order
Aggregate	Groups the data; applies aggregation methods and pivot operations
Deduplicate	Removes rows with duplicate values of specified columns
FlattenSchema	Flattens complex data types (e.g., structs and arrays)
Limit	Limits the number of rows in the output
SchemaMapper	Allows multiple columns to be renamed all at once
SchemaTransform	Is similar to the reformat gem, (e.g. for transforming columns), except it uses the withColumn syntax rather than the SELECT column AS syntax
SetOperation	Adds or subtracts rows from DataFrames with identical schemas and difference in data
WindowFunction	Defines a WindowSpec; applies Window functions on a DataFrame

Join and Split Gems

Join and split gems are used to merge or split DataFrames to create new DataFrames. Since Spark is a parallel processing engine, it's important to partition data appropriately to achieve desirable performance. Under-partitioning, over-partitioning and incorrect partitioning can significantly degrade performance.

gem Name	Description
Join	Joins two or more DataFrames based on the given condition
Repartition	Repartitions or coalesces the input DataFrame
RowDistributor	Creates multiple DataFrames based on provided filter conditions from an input DataFrame

Custom Gems

Custom gems are ones that cannot be categorized at the moment.

gem Name	Description
SQLStatement	Runs defined SQL queries against one or more input DataFrames to create new DataFrames
Script	Provides a SparkSession and allows users to run custom code
FileOperation	Runs file operations (e.g., copy and move) on different file systems where data is located
DeltaTableOperations	Conducts table operations (e.g., register table in a catalog, vacuum table, optimize table, etc.)
RestAPIEnrich	Enriches the DataFrame by adding columns with content from REST API output based on defined parameters

Subgraph Gems

Subgraphs allow users to wrap multiple, different gems under a single reusable parent gem. They're useful for decomposing complex logic into reusable components and simplifying the data engineering process.

For example, take a large financial company that computes the monthly recurring revenue (MRR) with a calculation that applies to different data sources (e.g. Stripe, Salesforce, etc.). Without a subgraph, when the business needs to compute MRR, every developer would have to create a set of transformations: reformat gem to clean the data, aggregate gem to sum the amounts by month and OrderBy gem to sort the months from the latest. Instead, with a subgraph, one person can define the business logic for the transformation and wrap all of them up in a subgraph, which can be easily shared with the rest of the team.

Prophecy

Prophecy is a low-code data transformation platform that offers an easy-to-use visual interface to build, deploy, and manage data pipelines with software engineering best practices. Prophecy is trusted by enterprises including multiple companies in the Fortune 50 where hundreds of engineers run thousands of ETL workloads every day. Prophecy is backed by some of the top VCs including Insight Partners and SignalFire. Learn how Prophecy can help your data engineering in the cloud at www.prophecy.io.

databricks

With origins in academia and the open source community, Databricks was founded in 2013 by the original creators of Apache Spark™, Delta Lake and MLflow. As the world’s first and only lakehouse platform in the cloud, Databricks combines the best of data warehouses and data lakes to offer an open and unified platform for data and AI.