IOblend

DEVELOPER AND ENTERPRISE ARCHITECTURE

Technical Architecture & Capabilities Guide

From the first working component to production-grade DataOps.

Build. Inspect. Extend. Operate.

A practical view of how IOblend pipelines are designed, tested, deployed and managed across enterprise data environments.



NOVEMBER 2026 / EDITION 0.3

READING MAP

A route through the platform

Start by building one working component. Each later chapter adds the capabilities needed to make that pipeline useful and dependable.

The guide follows the way a developer actually works: start locally, validate each component, connect more inputs and outputs, then promote a proven pipeline into a customer-managed Enterprise environment.

01 **Build and verify** 03-08

Local development, incremental testing, multiple sources and sinks

02 **Connect and transform** 09-13

Enterprise deployment, interfaces, formats and DataFrames

03 **Keep data correct** 14-21

CDC, deduplication, event-time changes, data quality and MDM

04 **Keep processing dependable** 22-24

Checkpoint recovery, independent destinations and security

05 **Extend the pipeline** 25-27

AI inference, external APIs/MCP and cross-Engine data

06 **Operate and evolve** 28-30

Logs, alerting, workload sizing, versioning and scheduling

01 / BUILD AND VERIFY

A data layer that works with what exists

IOblend connects existing systems and processes data in motion without requiring an intermediate warehouse or staging database.

A pipeline begins with sources already in the organisation: databases, files, applications, event brokers or cloud systems. IOblend reads the data, applies configured processing, and delivers the result to the systems that need it.

Designer provides the development experience. JSON playbooks preserve the pipeline definition. Engine runs the workload on the IOblend-supplied Spark runtime inside the chosen environment.

A DIRECT SOURCE-TO-DESTINATION PATH



Data stays in motion

Both batch and streaming pipelines can process and transform data without a mandatory staging platform. Spark still manages execution memory, checkpoints and storage where the configured workload requires them.

The same model scales from a first CSV lab to a continuously running enterprise flow. The complexity grows in the pipeline, not in a new collection of required intermediary platforms.

01 / BUILD AND VERIFY

Start on a developer workstation

The first IOblend experience is a local Developer Edition installation and a small pipeline that can be run and inspected end to end.

Developer Edition includes the Designer and the packaged local processing environment. The developer installs the distribution, prepares the project folders and Run Parameters, then starts with a small source, transformation and destination.

Local installation

Install the supplied Developer Edition package and required supporting dependencies. Follow the Developer Handbook for the Windows or macOS procedure and workstation checks.

First project

Set project paths, a pipeline name, the source and output folders, and the development-time debugging options. Run the Engine locally.

FIRST LEARNING LOOP

Component A
Read a small source

→

Component B
Transform and check

→

Component C
Write and verify

Make the first result correct

A completed run is not the only success criterion. Inspect the source rows, validate the calculation and confirm the destination contains the expected records. The Developer Handbook includes a guided CSV exercise.

01 / BUILD AND VERIFY

Build one component. Test it. Then add the next.

This incremental workflow is central to how IOblend reduces development effort.

Rather than assembling a large pipeline and discovering errors at the end, the developer adds a component, configures it, runs it and examines its output. Only when that stage is correct does the next component get added.

1

Add Component A

Configure the source and verify the data arriving.

2

Add Component B

Apply a transformation and inspect the resulting DataFrame.

3

Add Component C

Connect the sink and check the written results.

4

Extend the pipeline

Add another source, join, transform or sink and repeat the same check.

A working pipeline is assembled from working parts

Each new stage is validated against real data as the design grows. When the pipeline is complete, the developer has already tested its component logic, dependencies and intermediate results rather than deferring the entire debugging effort until the end.

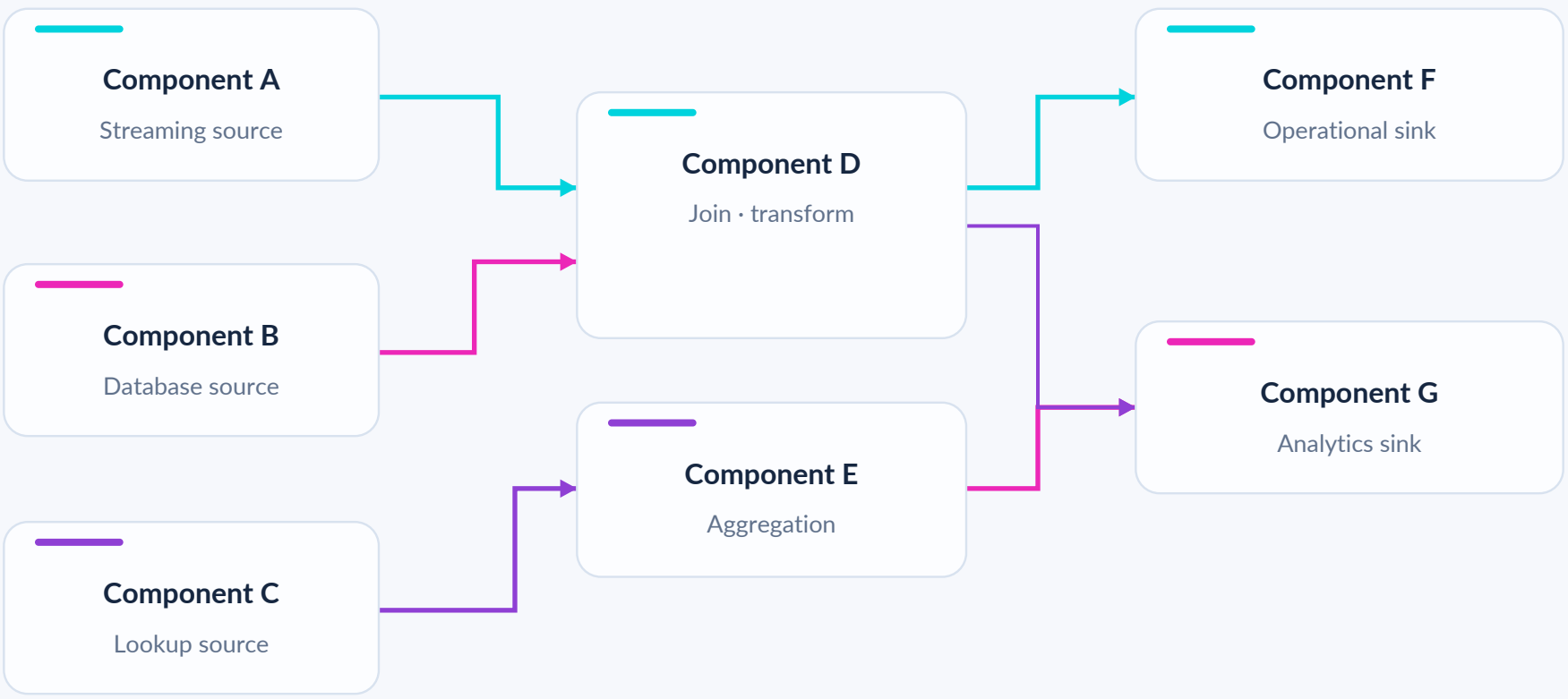
01 / BUILD AND VERIFY

One pipeline can branch in several directions

Multiple Sources, Transformations and Sinks can coexist within the same pipeline, with dependencies controlling how data flows.

A pipeline is not limited to one Source, one Transformation and one Sink. Developers can combine several live or static inputs, reuse intermediate DataFrames, branch processing logic and deliver different results to multiple destinations.

PIPELINE COMPOSITION



Dependencies are part of the design

Designer captures component relationships and run order. In streaming workflows, downstream components can continue as new upstream results become available; they do not need to wait for a continuous pipeline to end.

This composability is also why the one-component-at-a-time development process matters: complex, branching flows remain understandable and testable at every step.

01 / BUILD AND VERIFY

The project preserves the pipeline

Designer saves a reusable collection of component definitions and execution settings rather than a one-off script.

Each component is represented by JSON playbook metadata. Run Parameters identify the project and runtime configuration, while paths for source data, external Python, logs, checkpoints, state and output can be managed independently.

Designer

Build components, configure options, specify dependencies and run order, then inspect and change the logic interactively.

JSON playbooks

Keep the component definitions portable and reviewable, with native project history available for restoring previous versions.

Run Parameters

Provide pipeline paths, execution mode, state and checkpoint locations, and other project-level settings.

Engine runtime

Reads the project definition and executes the configured SQL, Python and Spark processing.

Separate operational artefacts

The customer can keep playbooks, logs, state, checkpoints, output and archived data in different repositories or directories. This makes the project easier to manage across development, production and recovery scenarios.

01 / BUILD AND VERIFY

Inspect every stage before moving forward

The Developer window makes the processing path visible while the pipeline is being built.

During development, interactive execution exposes intermediate DataFrames and component results. A developer can step through a run, compare input and output counts, read a schema, inspect a join or check a calculation directly.

Source inspection

- Confirm input records, schema, paths and source provenance.

Transformation inspection

- Review SQL or Python results, joins, aggregates and unexpected duplicates.

Destination inspection

- Check record counts, written output and generated lineage fields.

Engine logs

- Read processing times, checkpoints, errors and component execution details.

The result is already tested

A pipeline assembled component by component has been executed and checked at each stage. Further validation can be added through the customer's own development and release process.

02 / CONNECT AND TRANSFORM

From a local project to Enterprise execution

Remote Engine deployment and SSH configuration belong to Enterprise Edition, after the developer has established the working pipeline locally.

For production, DevOps provisions the IOblend Engine and packaged runtime in a customer-controlled VM or container. The developer configures the remote Engine location and SSH connection through Run Parameters, then points the project to the appropriate production repositories.

A

Develop locally

Designer and local Engine; test components as they are built.

B

Prepare Enterprise

Provision the Engine and Spark runtime in a VM or container.

C

Connect and validate

Configure remote Engine and paths; run interactively until satisfied.

D

Promote and operate

Complete CI/CD; disable interactive mode for unattended execution.

Production stays in the customer environment

Playbooks may be kept in customer repositories, including Git as part of their own CI/CD process. The Engine runs continuously once started, unless stopped by a configured process, condition or external control.

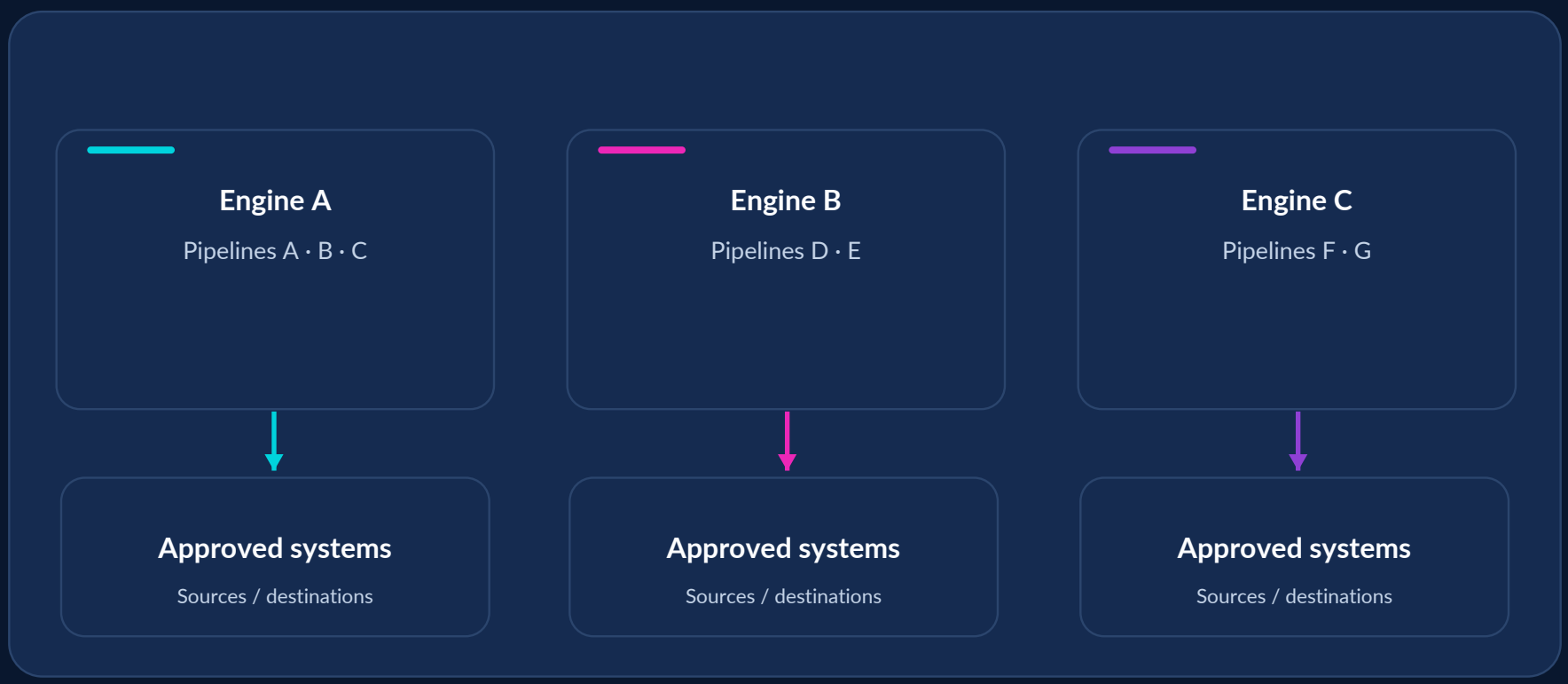
02 / CONNECT AND TRANSFORM

Independent Engines, customer-managed infrastructure

The same runtime can be deployed on-premises or in cloud VMs and containers. Each Engine is a ring-fenced execution environment.

An Engine can run several pipelines concurrently, provided its Spark workload fits the available resources. Organisations can deploy additional Engine instances for isolation or capacity, with their DevOps processes responsible for provisioning and orchestration.

CUSTOMER ENVIRONMENT / VM OR KUBERNETES PODS



Isolation and interoperability

Engines do not automatically share or distribute execution workloads across instances. Developers can nevertheless query DataFrames across Engines when access is configured, without making the Engines a single coordinated cluster.

After an infrastructure failure, DevOps can start a replacement Engine against the existing playbooks and persisted checkpoint/state locations. IOblend resumes processing from its last successful checkpoint.

02 / CONNECT AND TRANSFORM

Connect through standard interfaces

IOblend uses configurable connection methods rather than a catalogue of application-specific, prebuilt connectors.

The developer selects the interface offered by the system, configures credentials and connectivity, and builds the corresponding Source or Sink. The same approach covers modern platforms and legacy applications that expose supported access methods.

Databases

JDBC and ODBC, with the appropriate client-provided drivers, queries and connection permissions.

Applications and services

APIs, ESB interfaces and scripted access to application endpoints or integrations.

Files and object storage

Local and remote files, cloud object locations and supported file-based inputs and outputs.

Events and streams

Streaming data interfaces, including Apache Kafka as an input, alongside other supported event sources.

Connection configuration is part of the project

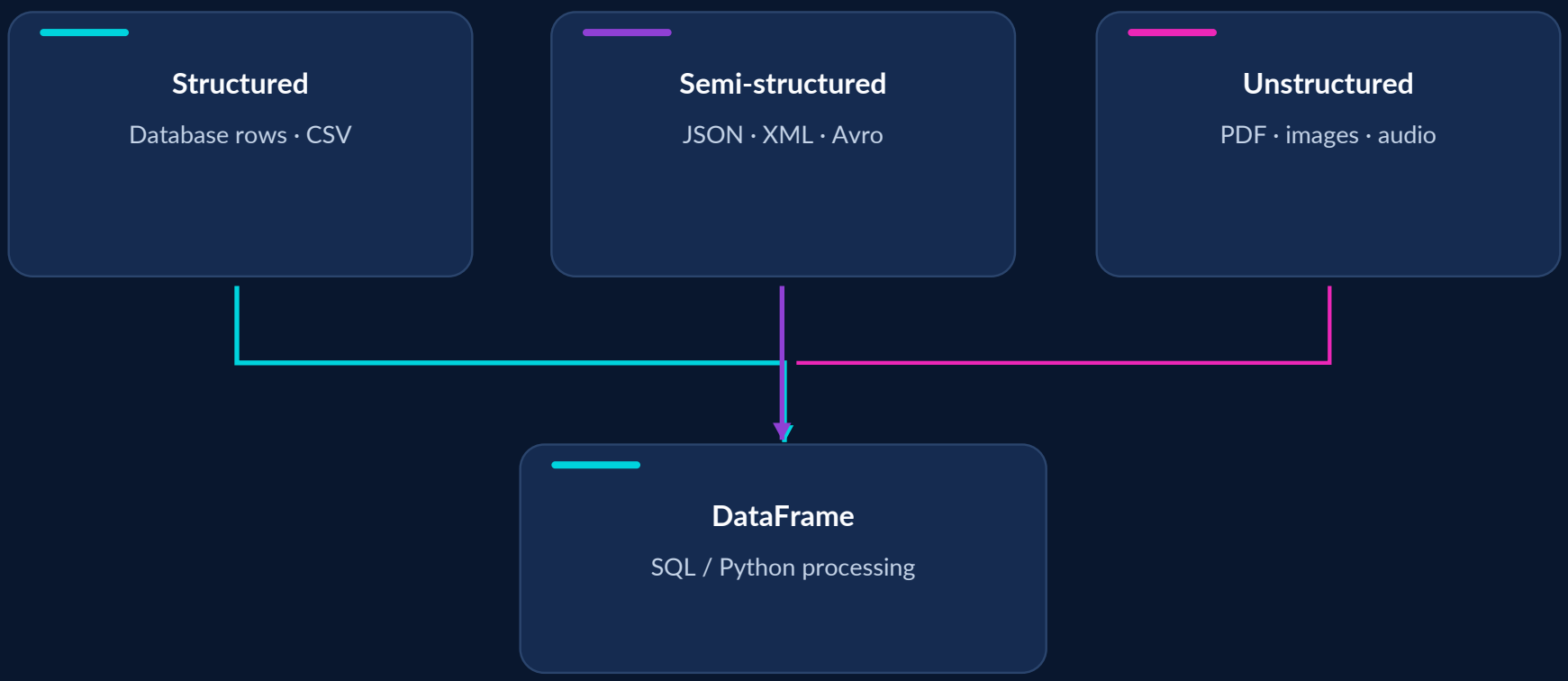
A Source or Sink records how that particular system is accessed. Source and destination credentials are individually authorised under the customer's security policies; IOblend uses the identity provided for each connection.

02 / CONNECT AND TRANSFORM

Different formats, one processing model

Once the data arrives, IOblend makes it available as DataFrames for consistent SQL and Python processing.

Structured sources, semi-structured messages and unstructured files can all participate in a pipeline. Major file formats are supported, including CSV, JSON, XML, Parquet and Avro, as well as table formats such as Apache Iceberg. Custom Python handles specialised formats or parsing needs.



Unstructured processing

Documents, scans, images and audio may be ingested directly. Extraction, OCR, transcription and AI interpretation are not built-in services: developers implement them through Python libraries or external models.

Original documents and source data can be archived when required. Ownership, source paths, keys and timestamps can travel with the processed data for later traceability.

02 / CONNECT AND TRANSFORM

A common language for processing

SQL and Python transformations work with the DataFrames produced by source and upstream components.

SQL covers filtering, joins, aggregations, windowing, derivations and business rules. Python adds custom logic, specialised libraries, machine learning and external-service calls. Multiple transformations can be chained and reused across dependent pipelines.

SQL transformations

Shape and query DataFrames; calculate metrics; join reference data; implement business validation and multi-stage processing.

Python transformations

Call libraries and APIs; process specialised formats; run inference; implement additional validation and control logic.



Processing without mandatory staging

Intermediate results can be queried directly as DataFrames by other components and pipelines. Persistence remains optional when a workflow needs results to survive beyond the live execution context.

03 / KEEP DATA CORRECT

Change Data Capture without one fixed method

IOblend can process INSERT, UPDATE and DELETE events using the approach that fits the source system.

Where database logs are accessible, the pipeline can read change information directly. Trigger-based CDC is available for database changes, and query-based CDC can compare snapshots to identify physical deletions.

Method	How the source is observed	Changes
Log-based	Database change log reading	Insert / update / delete
Trigger-based	Configured database triggers	Insert / update / delete
Query-based	Queries and snapshot comparisons	Insert / update / delete

CDC remains part of normal processing

Once a change is detected, IOblend routes it through the pipeline's transformation logic and updates the relevant destinations. Corrected source records can follow the same pattern during reprocessing.

The choice of CDC method depends on the interfaces, permissions and change information available from the source database. Developers configure the appropriate method for each integration.

03 / KEEP DATA CORRECT

Continuous deduplication and live windows

Native deduplication operates across streaming processing cycles and maintains its own persistent history.

A new record is not checked only against the current batch. IOblend can identify duplicates across prior processing cycles, independently of whether the original raw data has been archived.

Persistent deduplication

Maintained record history allows duplicate detection as the stream continues, including when a matching record arrived much earlier.

Configurable live window

Developers choose the incoming data window for the use case. Processing can consult storage when the available memory is insufficient.

Raw archive is a separate decision

The incoming raw data may be retained for any customer-defined period or not archived at all. A pipeline can deliver only fully deduplicated records while preserving the state needed for continued duplicate detection.

Retention policy, state sizing and the chosen window are design choices that should reflect the workload and available storage, rather than an imposed fixed retention period.

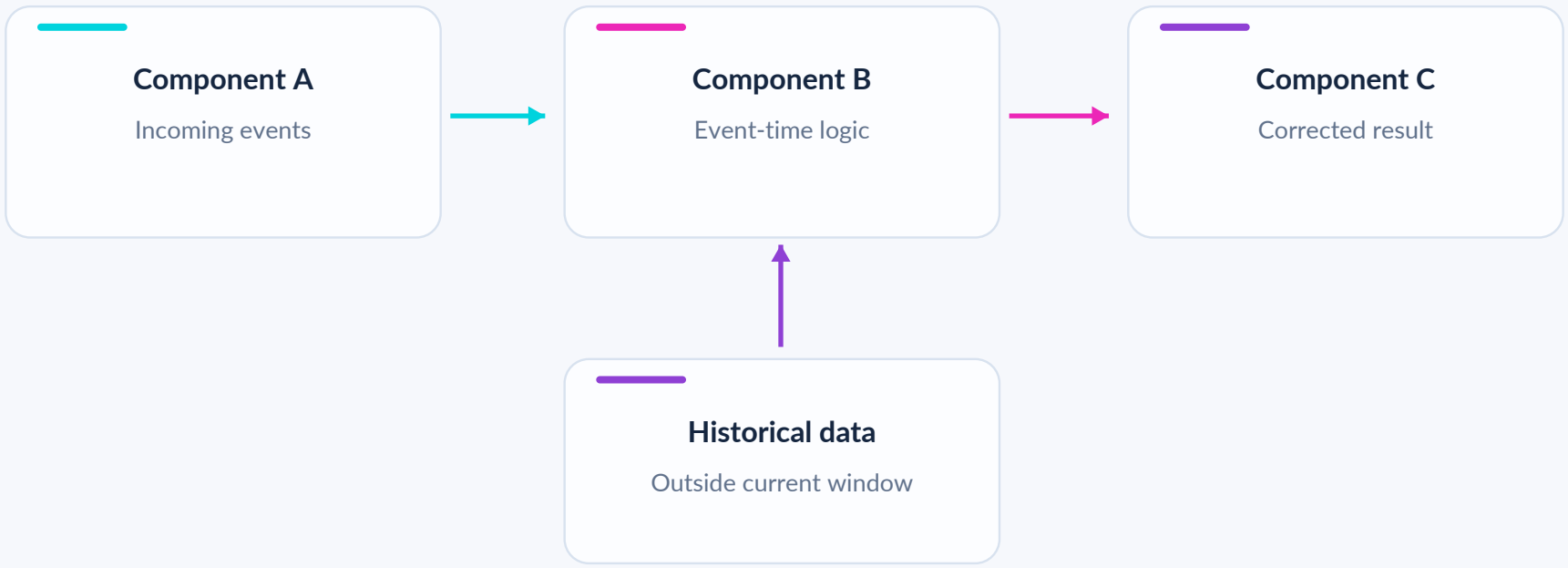
03 / KEEP DATA CORRECT

Late events need not stay late forever

IOblend can revisit historical processing when data arrives after its associated event-time window has already been calculated.

A delayed record may belong to a period closed hours, days or months earlier. IOblend can be configured to look beyond the live window, compare the incoming record against historical data, and recalculate the results affected by its arrival.

EVENT-TIME CORRECTION



Historical regression

The affected event periods are reconsidered, results are recalculated, and corrections are carried forward through the configured dependencies. The window length and history examined are controlled by the developer.

This is valuable when source systems deliver records out of sequence, when network outages delay events, or when earlier operational measures must be revised.

03 / KEEP DATA CORRECT

One correction can update a whole chain

Dependent aggregations are recalculated when an upstream historical result changes, preserving consistent downstream results.

Consider a late record that changes one grouped measure. IOblend can propagate the correction through subsequent aggregations rather than forcing the developer to rerun each stage manually.



Before and after are available

IOblend can preserve prior and corrected processing states in separate materialised tables. Developers choose whether to materialise only affected records or a complete dataset, according to the application's needs.

This creates a useful history of how calculations change when new facts arrive. The same selective or full materialisation choice can be applied across affected aggregation stages.

03 / KEEP DATA CORRECT

Schema changes are detected as data arrives

IOblend adapts to additive changes while flagging changes that could disrupt a working pipeline.

When a source introduces a new column, the column is detected and added automatically, and the pipeline continues. Columns that disappear are not silently deleted. Type changes are identified and highlighted for review.

Incoming change	Behaviour	Next step
New column	Added automatically; pipeline keeps running	Use the new field when needed
Column removed	Not automatically deleted	Review dependencies and schema
Data type changed	Detected and flagged	Resolve or implement custom handling

Custom handling when appropriate

A developer can implement explicit schema-specific behaviour using a Python transformation. This is useful when the application can safely accommodate a change that would otherwise interrupt processing.

This avoids treating all changes alike: harmless additions can flow through while potentially incompatible changes remain visible to the developer.

03 / KEEP DATA CORRECT

Validate the good. Quarantine the bad.

Data quality rules are defined in SQL or Python and can split a stream into accepted and rejected records.

A validation component may check required fields, schemas, business thresholds or complex rules. Invalid rows can be delivered to a quarantine destination while the valid records continue normally.



Reprocessing is another data flow

Developers can add a new Source and Transformation component and wire corrected records back into processing. The corrected records follow a CDC pattern, as updated source records do.

03 / KEEP DATA CORRECT

Master data governance stays with the customer

IOblend reconciles against the client's authoritative MDM table, then synchronises connected systems in either direction.

IOblend does not decide which master record is correct or run the approval workflow. The customer maintains the MDM repository and its stewardship process; IOblend follows the approved result when reconciling source and destination systems.

MDM reconciliation

Use the externally governed MDM table as the authority. Reconcile connected applications and synchronise approved values, including write-back when configured.

Native SCD Type I / II

Use configurable destination options to overwrite changed attributes for Type I, or preserve earlier versions for Type II.

GOVERNED SYNCHRONISATION



Rules and authority remain distinct

SQL and Python can implement reconciliation logic, but the authorised master-data workflow determines the accepted values. IOblend performs the corresponding data movement and synchronisation.

03 / KEEP DATA CORRECT

Trace data from source to outcome

Record-level provenance follows data through sources, joins, transformations, AI steps and destinations.

IOblend preserves information such as ownership, source path, record identifiers or keys and processing timestamps. During development, the engineer can step through intermediate DataFrames; in unattended execution, lineage metadata and logs continue to be produced.

Traceable information	How a developer uses it
Source and ownership	Identify the original document, dataset or system and responsible owner.
Keys and timestamps	Follow the relevant records and when they entered or changed in processing.
Component path	Inspect joins, transformations and downstream results in Designer or logs.
AI provenance	Retain processing context, including model identity, version and relevant configuration.

Lineage in production

The Engine continues recording logs and provenance when interactive mode is off. Logs are written continuously to the repository chosen by the customer, and lineage columns can be included in sink outputs.

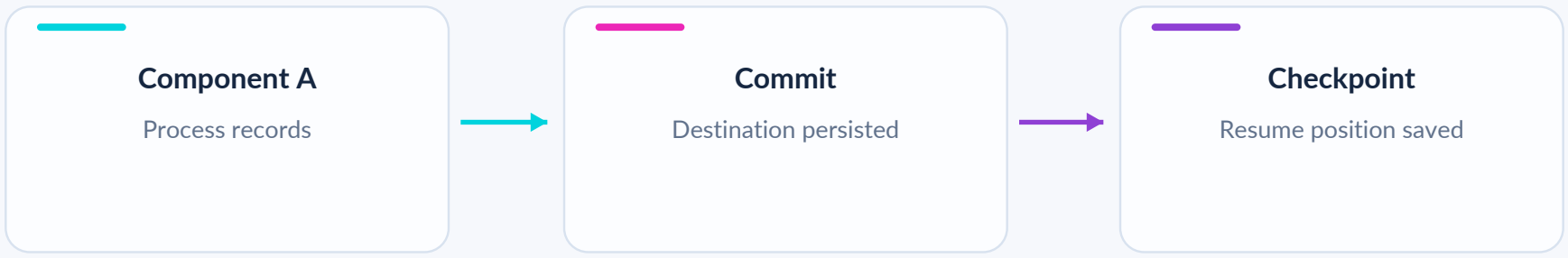
04 / KEEP PROCESSING DEPENDABLE

Checkpoint the commit, then move forward

Commit-aware checkpoints support restart from the last successfully persisted record instead of replaying the whole workload.

When the destination commit succeeds, the processing checkpoint is recorded. After interruption, the Engine resumes from that saved position, with the normal deduplication checks remaining active. A clean restart is also available when a workflow requires it.

NORMAL PROCESSING AND RESTART



A reproducible recovery point

The same checkpoint-based recovery pattern applies across supported destinations. Developers may choose to replay available historical data, including selected date ranges, when they deliberately change checkpoint behaviour.

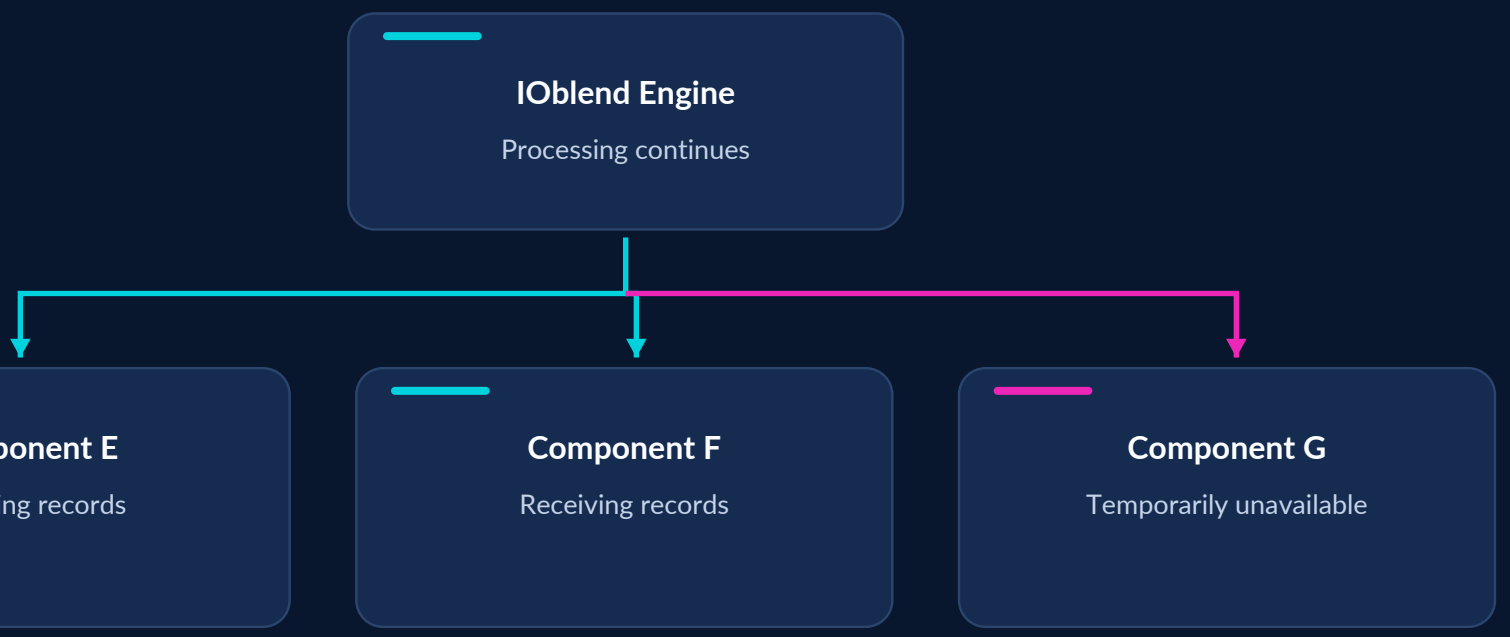
The exact replay scope follows the available source history and the configured transformations; it is not necessary to reconstruct the pipeline itself.

04 / KEEP PROCESSING DEPENDABLE

One unavailable sink need not stop the others

Each destination has its own tracked delivery progress, allowing healthy outputs to continue during a partial failure.

If Component G is unavailable while Components E and F accept their records, IOblend can continue processing for the healthy destinations. When G returns, the missing records are reconciled from its outstanding delivery history.



Recover after an outage or Engine restart

Outstanding data can be held for a developer-defined period and reconciled when the destination becomes available, even if the Engine was stopped in between. Explicit pipeline dependencies may instead require processing to stop.

If the host itself fails, customer DevOps provisions a replacement VM or container and reconnects the playbooks, checkpoints and state. IOblend resumes from the last successful checkpoint.

04 / KEEP PROCESSING DEPENDABLE

Run inside the customer’s security boundary

Data processing and operational information stay within the chosen customer infrastructure unless the customer chooses to share them.

Engine deployment, user access and permissions are governed by customer infrastructure policies. Every data Source and Sink requires its own authorised identity or credentials; IOblend uses what the customer provides.

Customer-managed access

Network isolation, IAM and user permissions are enforced by the customer environment. IOblend does not provide its own separate role-based permission system.

Data movement security

IOblend provides encryption in transit. Customers can add their own network and encryption controls under their policies.

Persisted information

Checkpoint, state, logs and materialised data reside in customer-controlled storage; at-rest protection follows the customer’s infrastructure controls.

Private operation

IOblend can operate entirely inside the customer network without sending information to IOblend infrastructure.

Security follows the data path

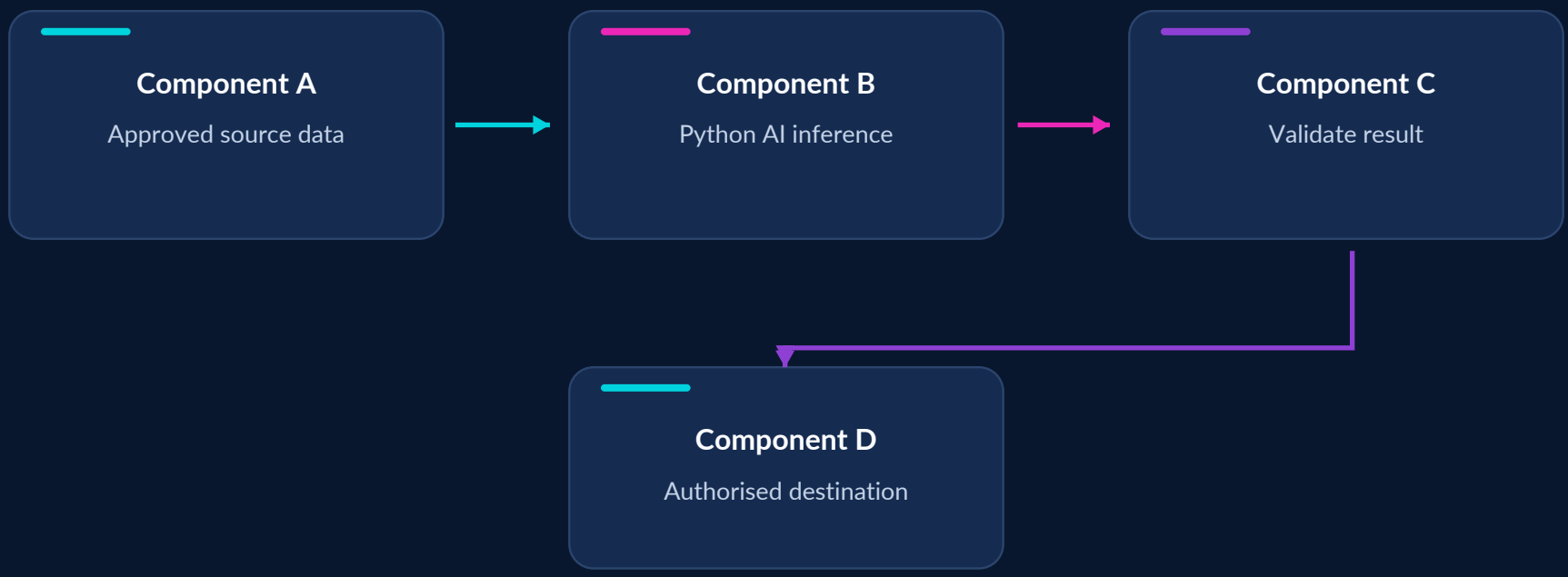
Approved access is required separately for source systems, destination systems, archived data and any external model/API services. The organisation retains its existing ownership of identity, governance and data residency.

05 / EXTEND THE PIPELINE

AI becomes another transformation stage

Custom Python can call a local model or an external AI service, then feed the result into the same DataFrame processing flow.

The pipeline prepares the data, invokes the selected model, checks its output and delivers the results. SQL or Python handles the business-specific validations before data is exposed to AI and before any proposed update is written back.



Same operating model

AI-enabled components participate in the usual dependencies, execution logs, error handling and checkpoint/recovery arrangements. Model hosting remains with the selected local or external service.

The customer defines what data may be supplied, which AI results are acceptable, and what downstream actions are permitted.

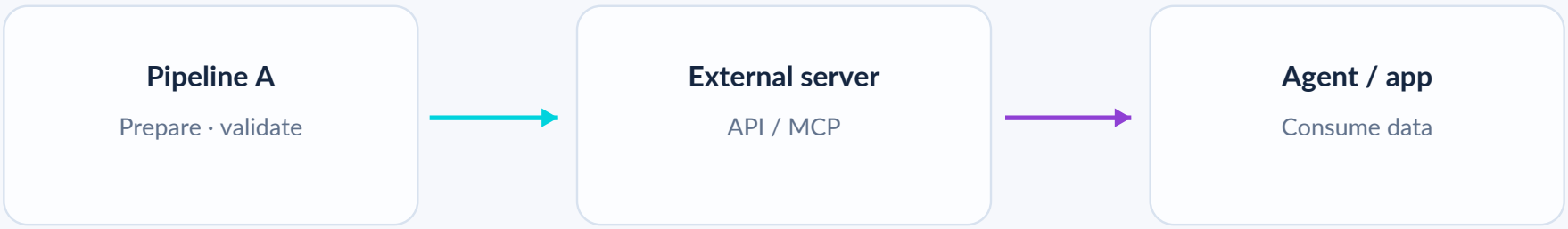
05 / EXTEND THE PIPELINE

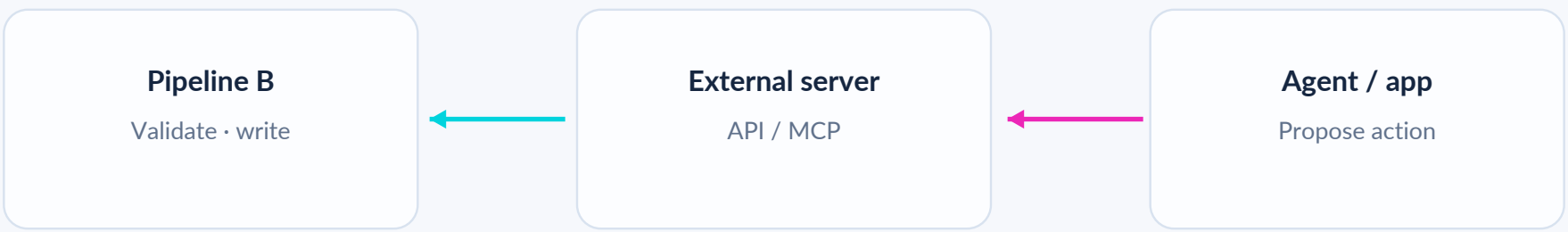
External APIs and MCP, connected in both directions

IOblend integrates with externally hosted services; each interface is configured through templated, scripted transformations.

The API or MCP server runs outside IOblend, under customer or provider management. IOblend can prepare approved data for an external application and accept requests or results back into another validated pipeline.

TWO COMPLEMENTARY PATHS





Bidirectional by configuration

Developers may use one configured integration or split the two directions into cooperating pipelines. Each pipeline can use information produced by the other when determining its actions.

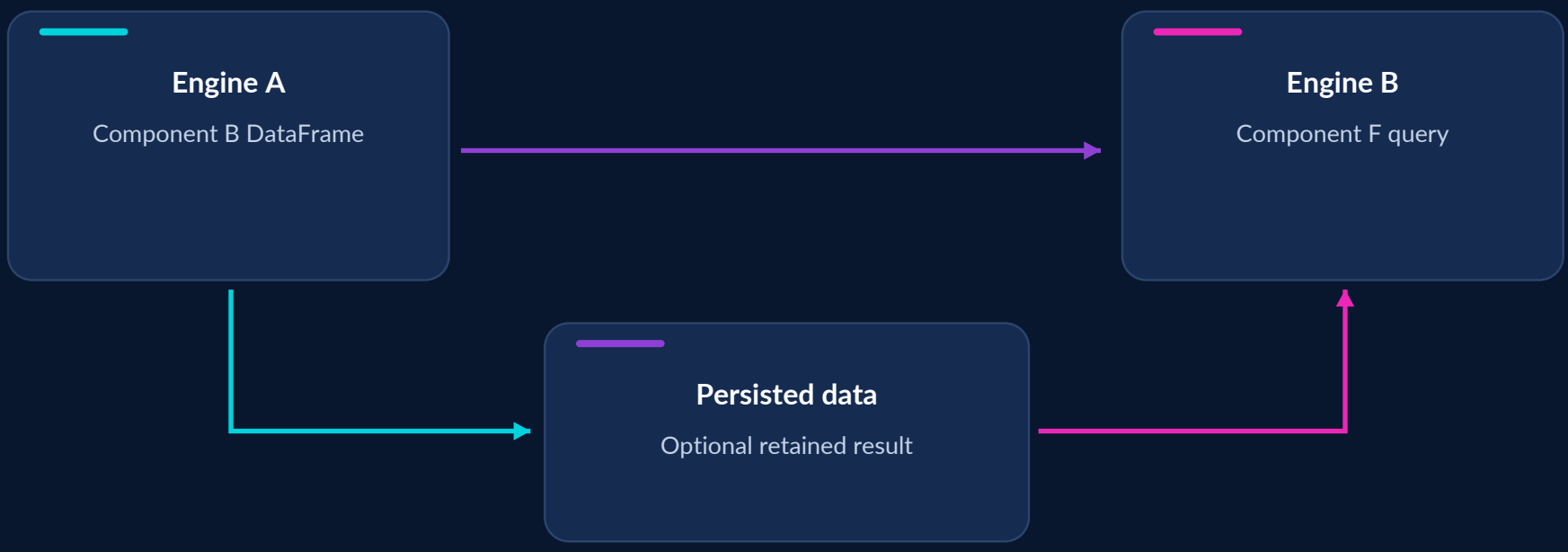
The integration is flexible, but the server remains external, and business validation is implemented through the configured SQL and Python logic.

05 / EXTEND THE PIPELINE

DataFrames can be reused across pipelines and Engines

A DataFrame has a unique identifier, allowing other components or independently deployed Engines to query its data.

Within a running Engine, pipelines can consume upstream DataFrames directly without placing every intermediate result into a separate database. Cross-Engine queries are also possible through configured access to the identified data.



Persistence controls availability

If a producing Engine stops, another Engine can still query the DataFrame only when its data has been persisted. A unique identifier alone does not preserve unpersisted data.

Developers can configure persistence and retention through Python transformations. Retention follows the customer's policy and storage capacity, with no IOblend-imposed maximum period.

06 / OPERATE AND EVOLVE

Every production run leaves an operational trail

Logs, state and lineage remain available when interactive development is switched off.

The Engine records execution activity and component-level statistics in customer-specified log/state locations. Developers and operators can inspect processing times, record counts, failures, checkpoint activity and the path taken through a pipeline.

Execution diagnostics

Component-level timings, record counts and Engine messages support troubleshooting and performance work.

Operational alerts

IOblend can send its own alerts or integrate with the customer's existing monitoring and notification mechanisms.

Business monitoring

Developers can implement alert conditions in SQL/Python: missing data, thresholds, exceptions or application-specific events.

Production traceability

Lineage and logs continue while unattended pipelines run, using the customer's preferred storage and retention policies.

Use the same evidence when debugging

What the developer saw during local step-through remains understandable later through stored logs and metadata. Spark diagnostics add infrastructure-level detail where deeper profiling is required.

06 / OPERATE AND EVOLVE

Size for the workload, not a universal benchmark

One Engine can run multiple concurrent pipelines within the limits of its allocated Spark resources.

IOblend supplies its own enhanced, tested Spark distribution with the Engine. Customers use that packaged runtime rather than substituting an independent Spark installation. Standard Spark settings are available for resource allocation and managing uneven streaming workloads.

What shapes performance	What to measure
Compute and memory	CPU, RAM, container/VM limits, concurrent pipeline demand.
Data and transformations	Record rate, batch size, joins, windowing, Python/AI operations.
Connected systems	Source read speed, network latency, destination write behaviour.
Operational recovery	Checkpoint restart, backlog and destination reconciliation time.

Benchmark representative workloads

Use real data volumes and transformation complexity to size the target infrastructure. Performance benchmarks are published on the IOblend website; actual outcomes depend on each deployment's infrastructure and workload.

All pipelines in an Engine share installed dependency versions. Additional Python, Java and JDBC libraries can be installed while processing continues, but a restart is needed before newly installed dependencies become active.

06 / OPERATE AND EVOLVE

A repeatable path from change to production

IOblend project history, external orchestration and the customer's release process complete the lifecycle.

Changes begin in Designer, one component at a time. Developers test their modifications, review the resulting DataFrames and outputs, then promote approved playbooks using the organisation's CI/CD controls.

Version and restore

The playbook project retains change history. Previous definitions can be restored, with checkpoint behaviour chosen for normal resumption or historical replay.

Upgrade and roll back

Designer and Engine may be upgraded independently across supported versions, and earlier Engine releases can be restored when required.

Schedule externally

Time-based scheduling belongs to tools such as Apache Airflow. They launch an IOblend pipeline by executing the appropriate command.

Run with defined dependencies

Designer controls component and pipeline dependencies and run order, including downstream processing as new streaming results become available.

The development advantage

By the time a completed pipeline is promoted, its Sources, Transformations, Sinks and dependencies have already been built and tested stage by stage. That shortens debugging cycles and makes the resulting production pipeline easier to understand, maintain and extend.

NEXT STEP

The next step is a working pipeline.

Start locally with IOblend Developer Edition. Build and test one component at a time, then extend the pipeline as each stage works.

GET FROM READING TO RUNNING

01

Download

Install the free Developer Edition on your Windows or macOS workstation.

02

Build & test

Start with a Source, test it, then add a Transform and Sink in turn.

03

Inspect & extend

Check your results and logs, then expand to more components and data flows.

START WITH DEVELOPER EDITION

Download IOblend Developer Edition

Free for development | Windows and macOS

KEEP EXPLORING

Documentation & tutorials

ioblend.com/documentation

Discuss Enterprise deployment

ioblend.com/contact

Build on your own machine. Prove it with your own data. Take it further when you are ready.