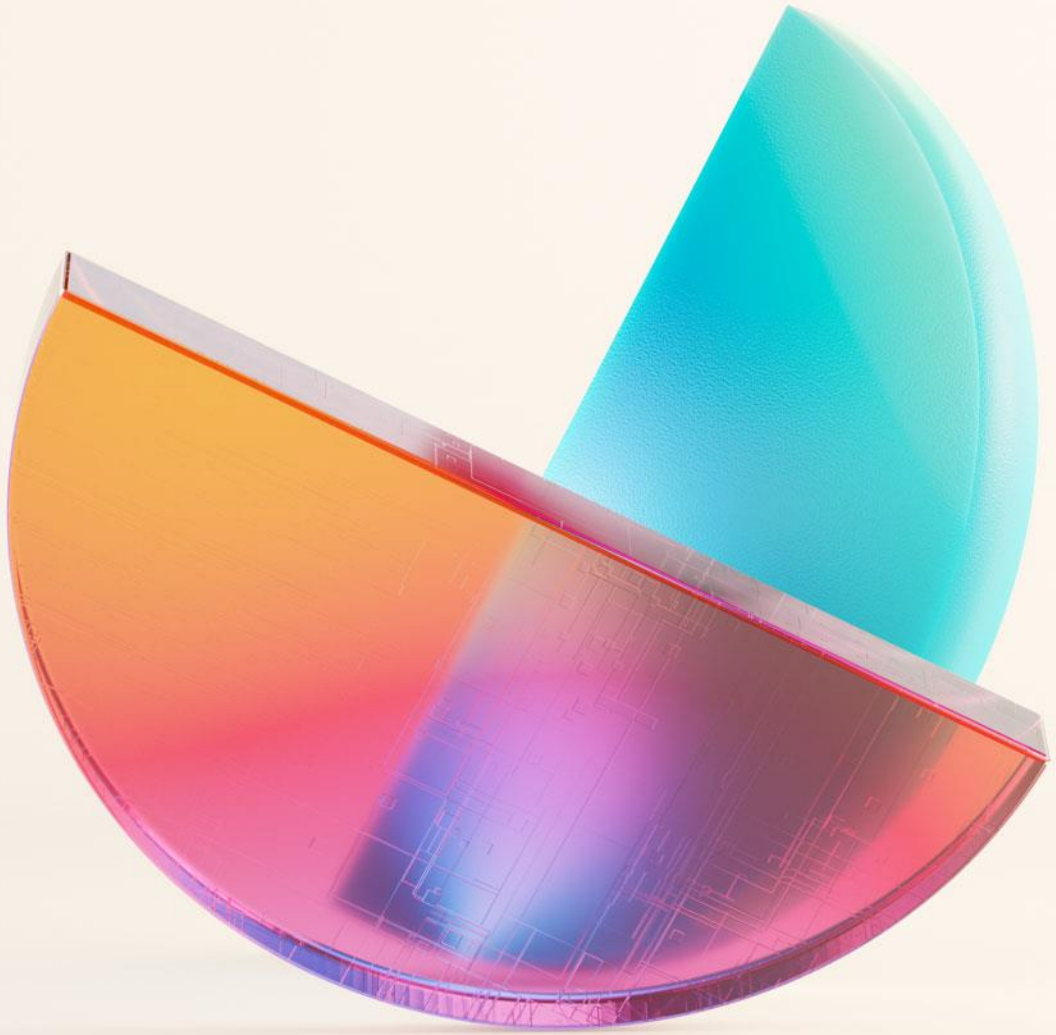
Microsoft

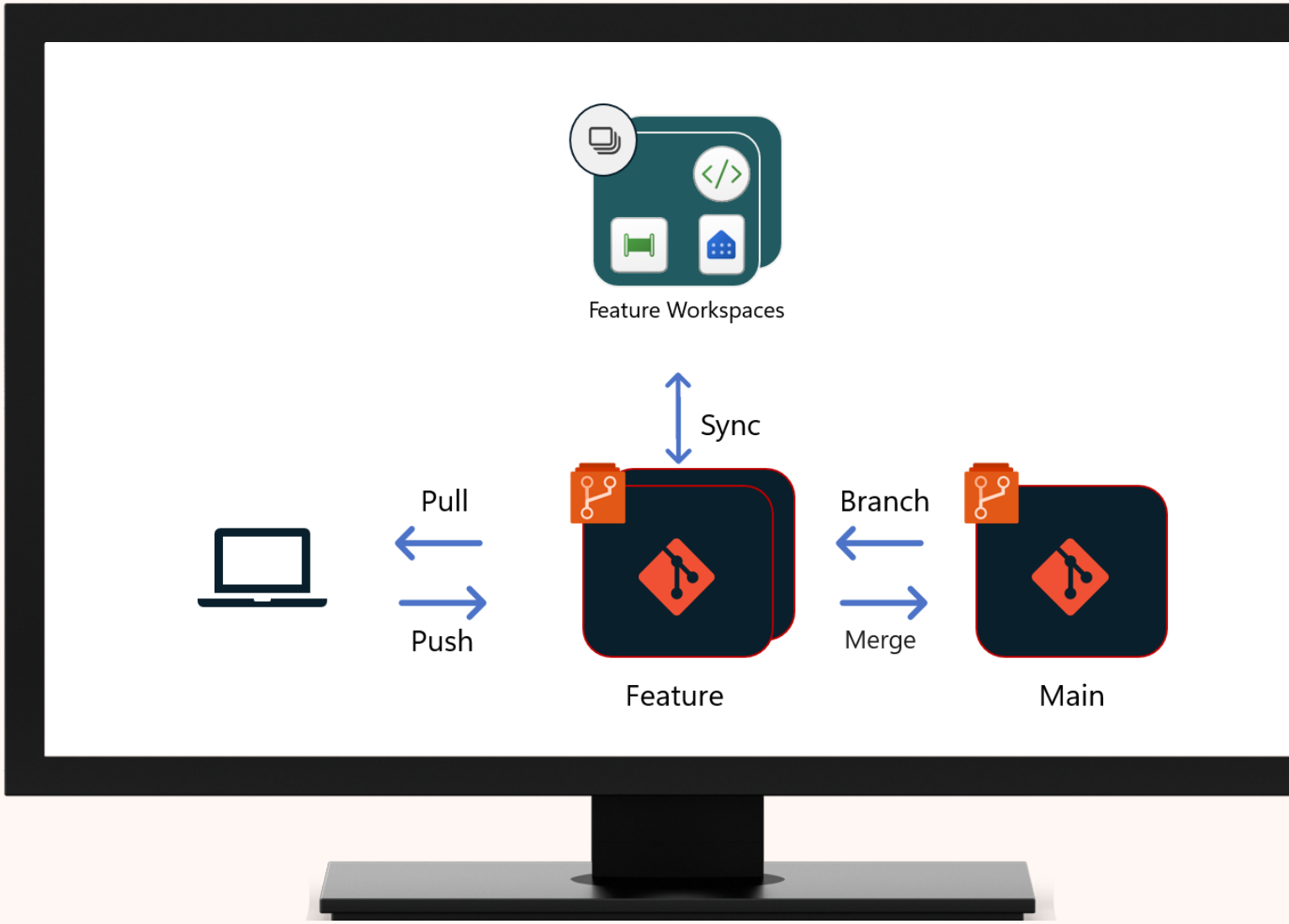
Microsoft Fabric



CI/CD

Lifecycle Management

- Microsoft Fabric's lifecycle management tools provide a standardized system for communication and collaboration between all members of the development team throughout the life of the product.
- You can use 3 methods for lifecycle management:
  - Deployment Pipeline
  - Git Integration
  - API Deployment

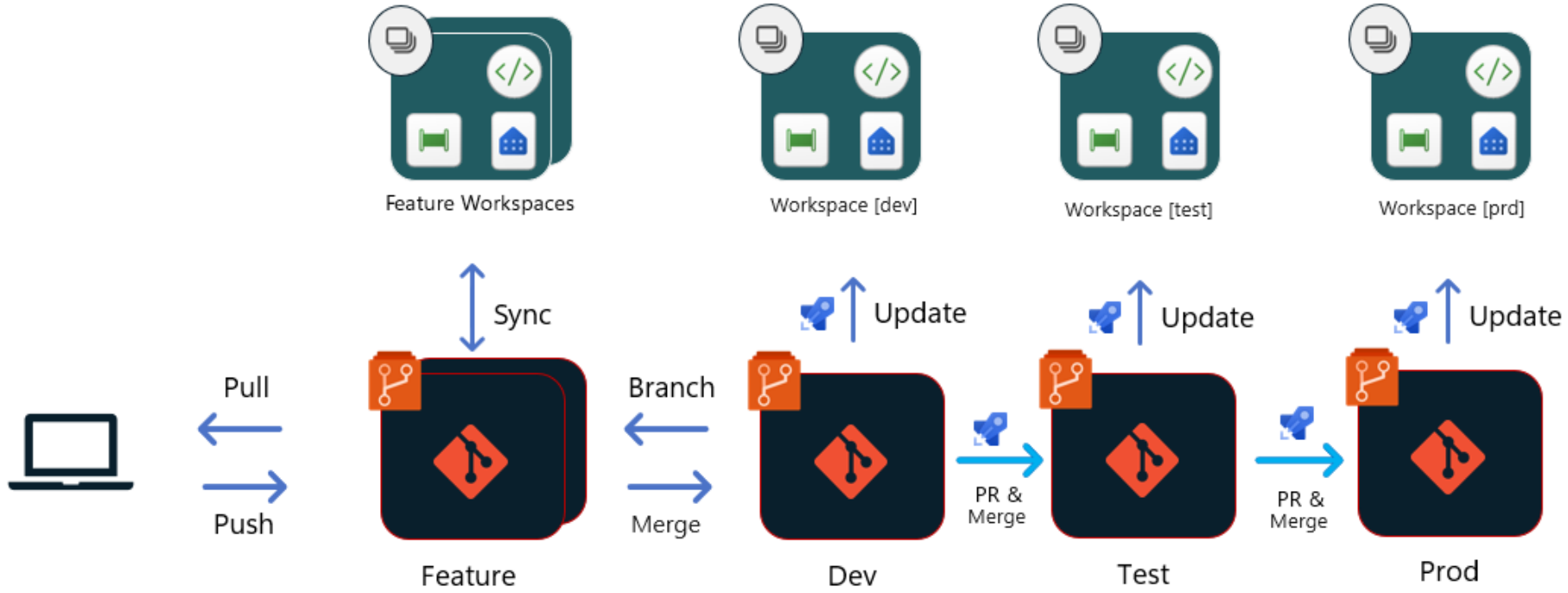


CI/CD Architecture in Fabric Real-Time Intelligence

- Deployment Pipelines:
  - Deployment pipelines orchestrate multi-environment promotion (Dev → Test → Prod) without external tools, using these serialized definitions.
- Git Integration:
  - Workspaces can be connected to Azure DevOps or GitHub repositories to synchronize artifacts bidirectionally.
  - Upon sync, Fabric serializes Real-Time Intelligence objects into a folder-based schema with platform metadata, properties JSON, and/or KQL schema files.
- API-driven deployments:
  - Public REST APIs expose operations for create, update, delete on items
  - Crucially, these APIs accept a "create with definition" payload that encodes object definitions as Base64-encoded scripts.
  - The platform executes these scripts server-side, enabling declarative, idempotent deployments.

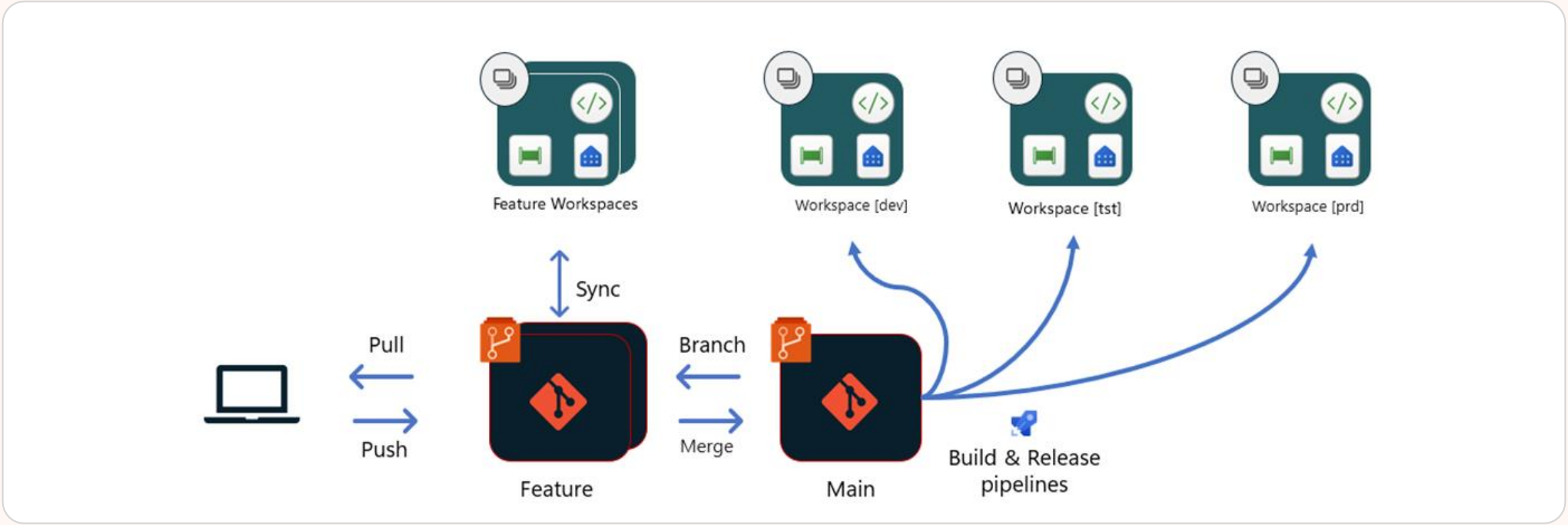
Option 1: Git-Based Deployments with Multiple Branches

In this approach, each environment stage—Development, Test, and Production—corresponds to a dedicated branch in your Git repository.



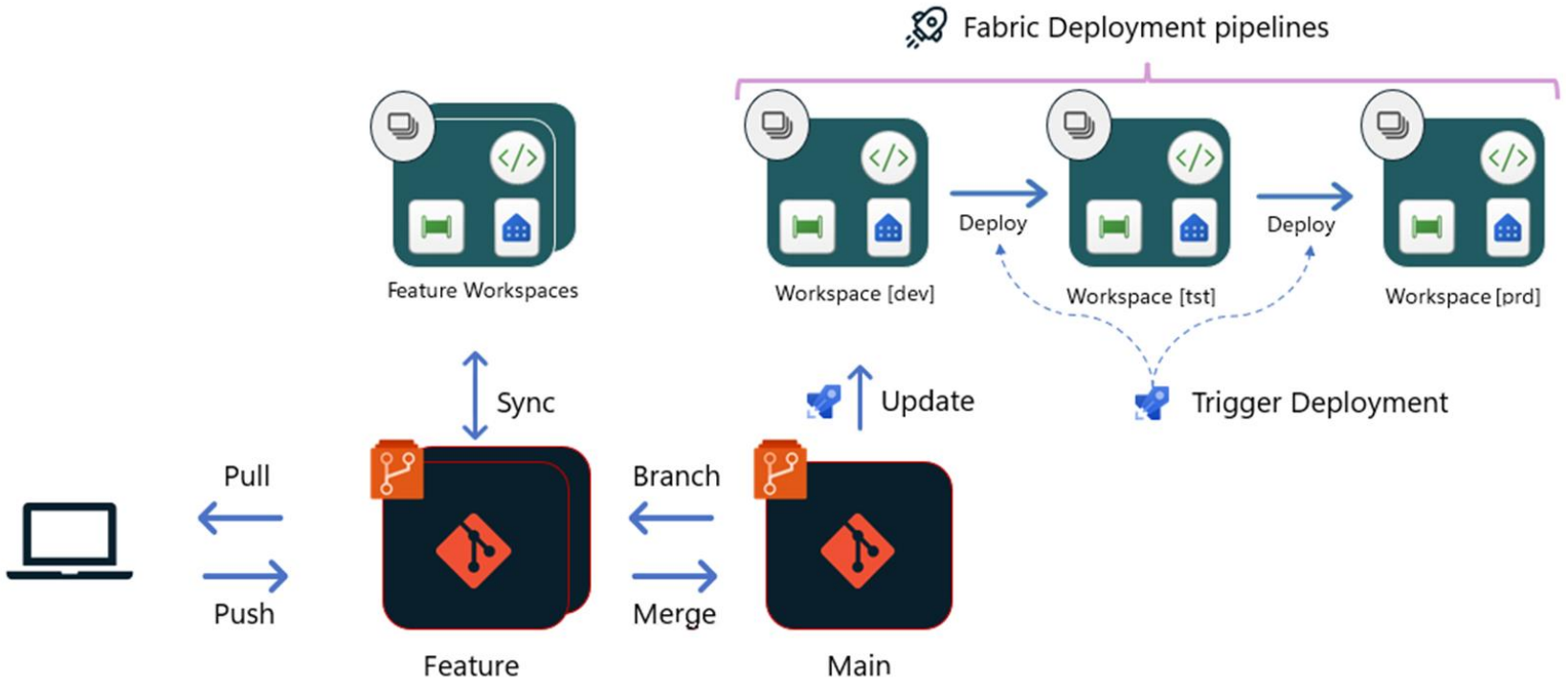
Option 2: Git-Based Deployments Using Build Environments

This method utilizes a single main branch, with build and release pipelines handling environment-specific configurations.



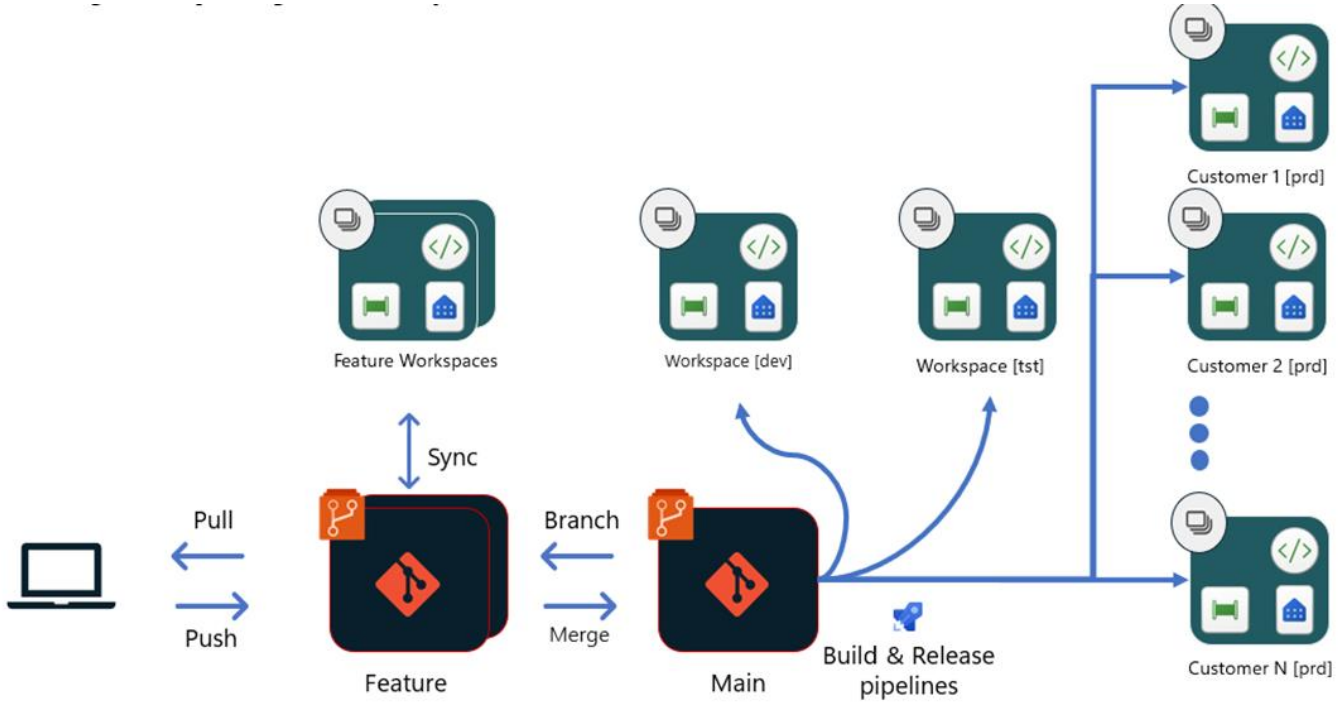
Option 3: Deployments Using Fabric Deployment Pipelines

Fabric's built-in deployment pipelines facilitate direct promotion of artifacts between workspaces without relying solely on Git branches.



Option 4: CI/CD for ISVs Managing Multiple Customers

Independent Software Vendors (ISVs) serving multiple customers can adopt a CI/CD approach that accommodates multi-tenant deployments.



Supported Items

- Git Integration Real-time Intelligence items:
  - Activator *(preview)*
  - Eventhouse
  - EventStream
  - KQL database
  - KQL Queryset
  - Real-time Dashboard
- Deployment Pipelines Real-time Intelligence items:
  - Activator *(preview)*
  - Digital twin builder *(preview)*
  - Eventhouse
  - EventStream
  - KQL database
  - KQL Queryset
  - Real-time Dashboard

Limitation

•**Git Integration** and **Deployment Pipeline** have limited support for cross-workspace scenarios. To avoid issues, make sure all Eventstream destinations within the same workspace. Cross-workspace deployment might not work as expected.If an Eventstream includes an Eventhouse destination using **Direct Ingestion** mode, you'll need to manually reconfigure the connection after importing or deploying it to a new workspace.

•Current limitations: Overview of Fabric Git integration - Microsoft Fabric | Microsoft Learn

Energy_Demo

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts

Project settings

Energy_Demo

Energy_Demo
Energy_Activator.Reflex
.platform
ReflexEntities.json
Energy_Evenstream_WindFarm.Eventstream
.platform
eventstream.json
eventstreamProperties.json
Energy_Eventhouse_WindFarm_queryset.KQLQuer
.platform
RealTimeQueryset.json
Energy_Eventhouse_WindFarm.Eventhouse
.children
.platform
EventhouseProperties.json
Energy_Lakehouse.Lakehouse
Energy_Realtime_Dashboard.KQLDashboard
.platform
RealTimeDashboard.json
luoanaSparkEnvironment.Environment
WindEnergyDataGenerator.Notebook

main / Energy_Demo / Energy_Evenstream_WindFa... / .platform

.platform

Edit

Contents History Compare Blame

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/fabric/gitIntegration/platformProperties/2.0.0/schema.json",
  "metadata": {
    "type": "Eventstream",
    "displayName": "Energy_Evenstream_WindFarm"
  },
  "config": {
    "version": "2.0",
    "logicalId": "384c9184-cd98-a96f-48a5-2286d6279c3c"
  }
}
```

The .platform file contains metadata about each artifact, used by Microsoft Fabric deployment pipelines to define and manage the deployment of a specific artifact

Schema->points to the JSON schema that validates this file, the **type** of object being described, the **display name** is the user friendly name of the artifcact

Config-> the version of the configuration schema being used, **locationId** being the deployment location or environment

Energy_Demo

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts

Project settings

Energy_Demo

- Energy_Demo
  - Energy_Activator.Reflex
    - .platform
    - ReflexEntities.json
  - Energy_Evenstream_WindFarm.Eventstream
    - .platform
    - eventstream.json
    - eventstreamProperties.json
  - Energy_Eventhouse_WindFarm_queryset.KQLQuer
    - .platform
    - RealTimeQueryset.json
  - Energy_Eventhouse_WindFarm.Eventhouse
    - .children
    - .platform
    - EventhouseProperties.json
  - Energy_Lakehouse.Lakehouse
  - Energy_Realtime_Dashboard.KQLDashboard
    - .platform
    - RealTimeDashboard.json
  - luoanaSparkEnvironment.Environment
  - WindEnergyDataGenerator.Notebook

main / Energy_Demo / Energy_Activator.Reflex / ReflexEntities.json

ReflexEntities.json

Edit

Contents History Compare Blame


```
[
  {
    "uniqueIdentifier": "9d3e94b4-4f49-4f51-94da-a37584e41eba",
    "payload": {
      "name": "Energy_Realtime_Dashboard",
      "type": "kqlQueries"
    },
    "type": "container-v1"
  },
  {
    "uniqueIdentifier": "cb2e72c7-6e86-4652-adc7-69bce656c804",
    "payload": {
      "name": "Energy_Eventhouse_WindFarm event",
      "runSettings": {
        "executionIntervalInSeconds": 300
      },
      "query": {
        "queryString": "GoldWindfarm\n| where LastSeen > ago(1h)\n| summarize TotalFaults = sum(FaultCount)\n| where TotalFaults
      },
      "eventhouseItem": {
        "itemId": "a5c2ceeb-2e0e-9571-40c0-63f441e360b5",
        "workspaceId": "00000000-0000-0000-0000-000000000000",
        "itemType": "KustoDatabase"
      },
      "queryParameters": [],
      "metadata": {
        "workspaceId": "489e4685-ae72-4e9c-8202-4cc3b453ff2c",
        "measureName": "TotalFaults",
        "dashboardId": "8afffe2b-43ac-4619-9e7a-73415c7d267e",
        "visualId": "47ef2109-6333-46b1-9997-f7e5900a840f"
      },
      "parentContainer": {
        "targetUniqueIdentifier": "9d3e94b4-4f49-4f51-94da-a37584e41eba"
      }
    },
    "type": "kqlSource-v1"
  },
  {
    "uniqueIdentifier": "6f0e33fc-daa8-4e34-828b-eb8a8869a098",
    "payload": {
      "name": "Energy_Eventhouse_WindFarm event",
      "parentContainer": {
        "targetUniqueIdentifier": "9d3e94b4-4f49-4f51-94da-a37584e41eba"
      },
```


GUID that identifies the entity

Type of entity:
container-v1: dashboard or logical grouping of visuals and rules.
kqlSource-v1: KQL query that pulls data from Eventhouse (e.g., GoldWindfarm) and feeds it into a visual.
timeSeriesView-v1: events, attributes, and transformations used in time series analysis.
Rule: Defines alerting logic (e.g., if FaultCount exceeds a threshold, send an email).

the actual configuration of the entity, including **name**, **query**, **eventhouseItem** links to the Eventhouse & table, **metadata** for the dashboard & visual IDs for binding, **definition** for rules and events, **parentContainer** defines the hierarchical relationship between entities.

Energy_Demo

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts

Project settings

Energy_Demo

- Energy_Demo
  - Energy_Activator.Reflex
    - .platform
    - ReflexEntities.json
  - Energy_Evenstream_WindFarm.Eventstream
    - .platform
    - eventstream.json
    - eventstreamProperties.json
  - Energy_Eventhouse_WindFarm_queryset.KQLQuer
    - .platform
    - RealTimeQueryset.json
  - Energy_Eventhouse_WindFarm.Eventhouse
    - .children
    - .platform
    - EventhouseProperties.json
  - Energy_Lakehouse.Lakehouse
  - Energy_Realtime_Dashboard.KQLDashboard
    - .platform
    - RealTimeDashboard.json
  - luoanaSparkEnvironment.Environment
  - WindEnergyDataGenerator.Notebook

eventstream.json

Edit

Contents | History | Compare | Blame

Where the data originates from, eg: custom endpoint

Where the data is being sent, eg: eventhouse

The data flow between the source & next processing step

Transformations applied to data, Eg: managefields


```
{
  "sources": [
    {
      "id": "2ee0cefb-c63c-4878-a658-af4ee335b6df",
      "name": "WindFarmData",
      "type": "CustomEndpoint",
      "properties": {}
    }
  ],
  "destinations": [
    {
      "id": "0b91dfe5-88a9-4674-b6e7-86153a784e96",
      "name": "Eventhouse",
      "type": "Eventhouse",
      "properties": {
        "dataIngestionMode": "ProcessedIngestion",
        "workspaceId": "00000000-0000-0000-0000-000000000000",
        "itemId": "a5c2ceeb-2e0e-9571-40c0-63f441e360b5",
        "databaseName": "Energy_Eventhouse_WindFarm",
        "tableName": "raw_wind",
        "inputSerialization": {
          "type": "Json",
          "properties": {
            "encoding": "UTF8"
          }
        }
      },
      "inputNodes": [
        {
          "name": "ManageFields"
        }
      ],
      "inputSchemas": [
        {
          "name": "ManageFields",
          "schema": {
            "columns": [
              {
                "name": "TurbineID",
                "type": "Nvarchar(max)",
                "fields": null,
                "items": null
              },
              {
```

```
  "streams": [
    {
      "id": "83b20774-179e-456d-9be4-4ab4e0ac9bb8",
      "name": "Energy_Evenstream_WindFarm-stream",
      "type": "DefaultStream",
      "properties": {},
      "inputNodes": [
        {
          "name": "WindFarmData"
        }
      ]
    }
  ],
  "operators": [
    {
      "name": "ManageFields",
      "type": "ManageFields",
      "inputNodes": [
        {
          "name": "Energy_Evenstream_WindFarm-stream"
        }
      ],
      "properties": {
        "columns": [
          {
            "type": "Rename",
            "properties": {
              "column": {
                "expressionType": "ColumnReference",
                "node": null,
                "columnName": "TurbineID",
                "columnPathSegments": []
              }
            },
            "alias": "TurbineID"
          },
          {
            "type": "Rename",
            "properties": {
              "column": {
                "expressionType": "ColumnReference",
                "node": null,
                "columnName": "Component",
                "columnPathSegments": []
```

Energy_Demo
Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts
Project settings

Energy_Demo
Energy_Demo
Energy_Activator.Reflex
.platform
ReflexEntities.json
Energy_Evenstream_WindFarm.Eventstream
.platform
eventstream.json
eventstreamProperties.json
Energy_Eventhouse_WindFarm_queryset.KQLQuer
.platform
RealTimeQueryset.json
Energy_Eventhouse_WindFarm.Eventhouse
.children
Energy_Eventhouse_WindFarm.KQLDatabase
.platform
DatabaseProperties.json
DatabaseSchema.kql
.platform
EventhouseProperties.json
Energy_Lakehouse.Lakehouse
Energy_Realtime_Dashboard.KQLDashboard

DatabaseProperties.json

Edit

Contents History Compare Blame

```
{
  "databaseType": "ReadWrite",
  "parentEventhouseItemId": "43eb252f-7982-91b8-4abf-6d436f54c5ea",
  "oneLakeCachingPeriod": "P36500D",
  "oneLakeStandardStoragePeriod": "P36500D"
}
```

Provides the configuration settings for a KQL DB in an Eventhouse, it is a writable db that can ingest & store data, the parentEventhouseId points to the GUID of the parent Eventhouse resource the DB belongs to, the P36500D defines how long the data is cached & retained in OneLake storage.

Energy_Demo

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts

Project settings

Energy_Demo

Energy_Demo
Energy_Activator.Reflex
.platform
ReflexEntities.json
Energy_Evenstream_WindFarm.Eventstream
.platform
eventstream.json
eventstreamProperties.json
Energy_Eventhouse_WindFarm_queryset.KQLQuer
.platform
RealTimeQueryset.json
Energy_Eventhouse_WindFarm.Eventhouse
.children
Energy_Eventhouse_WindFarm.KQLDatabase
.platform
DatabaseProperties.json
DatabaseSchema.kql
.platform
EventhouseProperties.json
Energy_Lakehouse.Lakehouse
Energy_Realtime_Dashboard.KQLDashboard

DatabaseSchema.kql

Edit

Contents History Compare Blame

```
1  // KQL script
2  // Use management commands in this script to configure your database items, such as tables, functions, materialized views, and mc
3
4
5  .create-merge table raw_wind (TurbineID:string, Component:string, Tag:string, Value:dynamic, Timestamp:datetime, EventProcessedUt
6  .create-merge table SilverWindfarm (TurbineID:string, Component:string, Tag:string, Value:real, Timestamp:datetime, FaultType:str
7  .create-or-alter function with (skipvalidation = "true") TransformToSilver() {
8      raw_wind
9      | extend FaultType = case(
10         Tag == "Vibration" and Value > 8.0, "Gearbox Bearing Failure",
11         Tag == "Temperature" and Value > 95, "Generator Insulation Failure",
12         Tag == "Structural Load" and Value > 1200, "Rotor Imbalance",
13         ""
14     )
15     | project TurbineID, Component, Tag, Value = todouble(Value), Timestamp, FaultType
16 }
17 .create-or-alter materialized-view with (Folder = "Gold")  GoldWindfarm on table SilverWindfarm { SilverWindfarm
18 | where isnotempty(FaultType)
19 | summarize FaultCount = count(), LastSeen = max(Timestamp) by TurbineID, FaultType }
20 .alter table SilverWindfarm policy update "[{\"IsEnabled\":true,\"Source\":\"raw_wind\",\"Query\":\"TransformToSilver()\",\"IsTra
21
```

Defines the content (tables, mv, functions, policies) within that DB

Energy_Demo

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts

Project settings

Energy_Demo

- Energy_Demo
  - Energy_Activator.Reflex
    - .platform
    - ReflexEntities.json
  - Energy_Evenstream_WindFarm.Eventstream
    - .platform
    - eventstream.json
    - eventstreamProperties.json
  - Energy_Eventhouse_WindFarm_queryset.KQLQuer
    - .platform
    - RealTimeQueryset.json
  - Energy_Eventhouse_WindFarm.Eventhouse
    - .children
    - .platform
    - EventhouseProperties.json
  - Energy_Lakehouse.Lakehouse
  - Energy_Realtime_Dashboard.KQLDashboard
    - .platform
    - RealTimeDashboard.json
  - luoanaSparkEnvironment.Environment
  - WindEnergyDataGenerator.Notebook
  - README.md

main / Energy_Demo / Energy_Realtime_Dashboar... / RealTimeDashboard.json

RealTimeDashboard.json

Edit

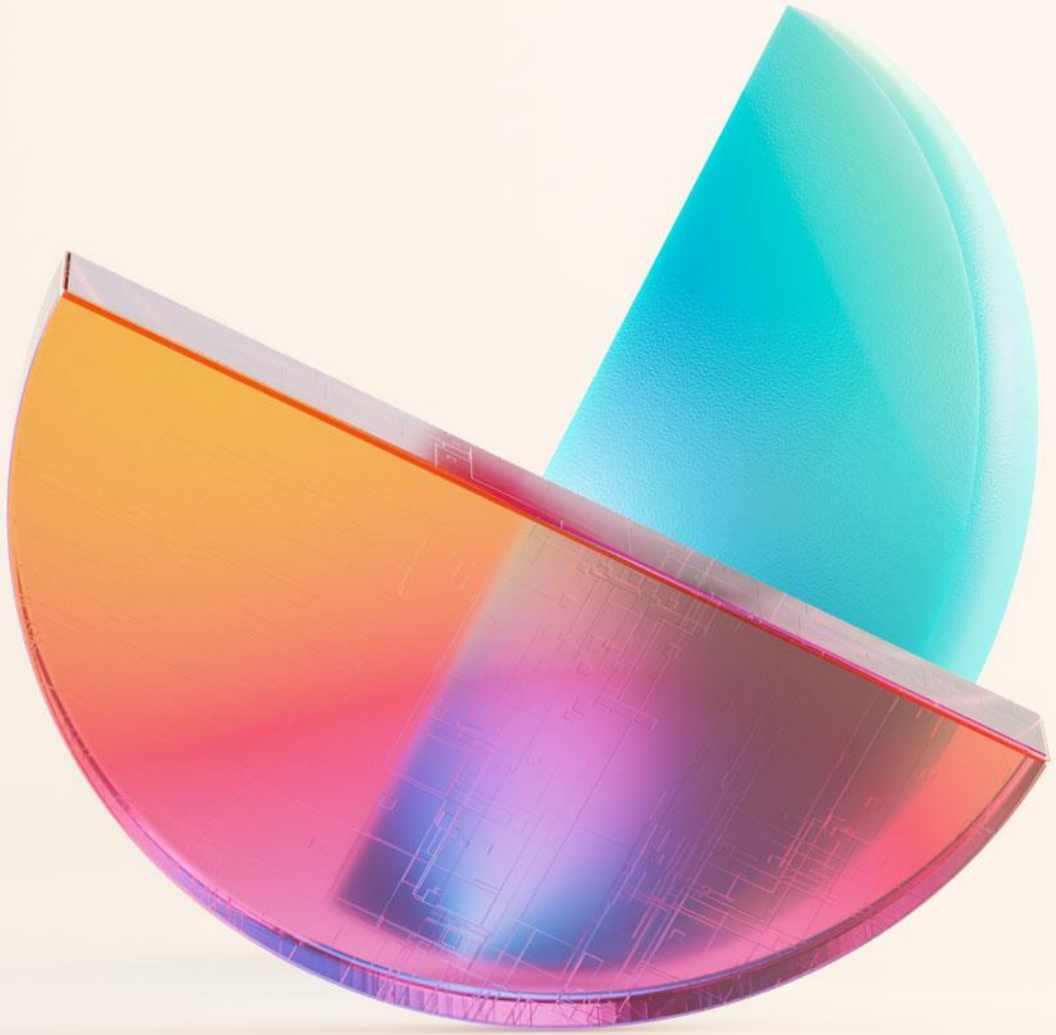
Contents History Compare Blame


```
{
 "schema_version": "60",
 "tiles": [
   {
     "id": "bba31a2c-bd54-4553-b951-a542fa63181c",
     "title": "Top Fault Types",
     "visualType": "heatmap",
     "pageId": "2a2289cd-0639-40c5-a018-135b228dec35",
     "layout": {
       "x": 0,
       "y": 7,
       "width": 9,
       "height": 7
     },
     "queryRef": {
       "kind": "query",
       "queryId": "9b4eb9da-319f-410e-bf2b-3db493182e07"
     },
     "visualOptions": {
       "xColumn": null,
       "xColumnTitle": "",
       "yColumnTitle": "",
       "heatMap__dataColumn": null,
       "heatMap__colorPaletteKey": "blue",
       "yColumns": null
     }
   },
   {
     "id": "0bab4829-23ad-4e58-82e6-708876076191",
     "title": "Trend Over Time by Fault Type",
     "visualType": "area",
     "pageId": "2a2289cd-0639-40c5-a018-135b228dec35",
     "layout": {
       "x": 9,
       "y": 7,
       "width": 9,
       "height": 7
     },
     "queryRef": {
       "kind": "query",
       "queryId": "7984a819-af95-40f5-a456-5817d05b6c9b"
     },
     "visualOptions": {
       "multipleYAxes": {
```


Defines the layout, visuals, queries & data sources

Microsoft

Microsoft Fabric



Activator

What you will learn in this module

- Architect stateful, rule-based detection logic on live data streams.
- Build responsive pipelines triggered by real-time events and patterns.
- Implement best practices to avoid alert fatigue, manage state transitions, and orchestrate Activator with the wider Fabric Real-Time Intelligence suite.
- Understand the pricing, monitoring, and schema evolution considerations essential for stable production use.

What is Activator?

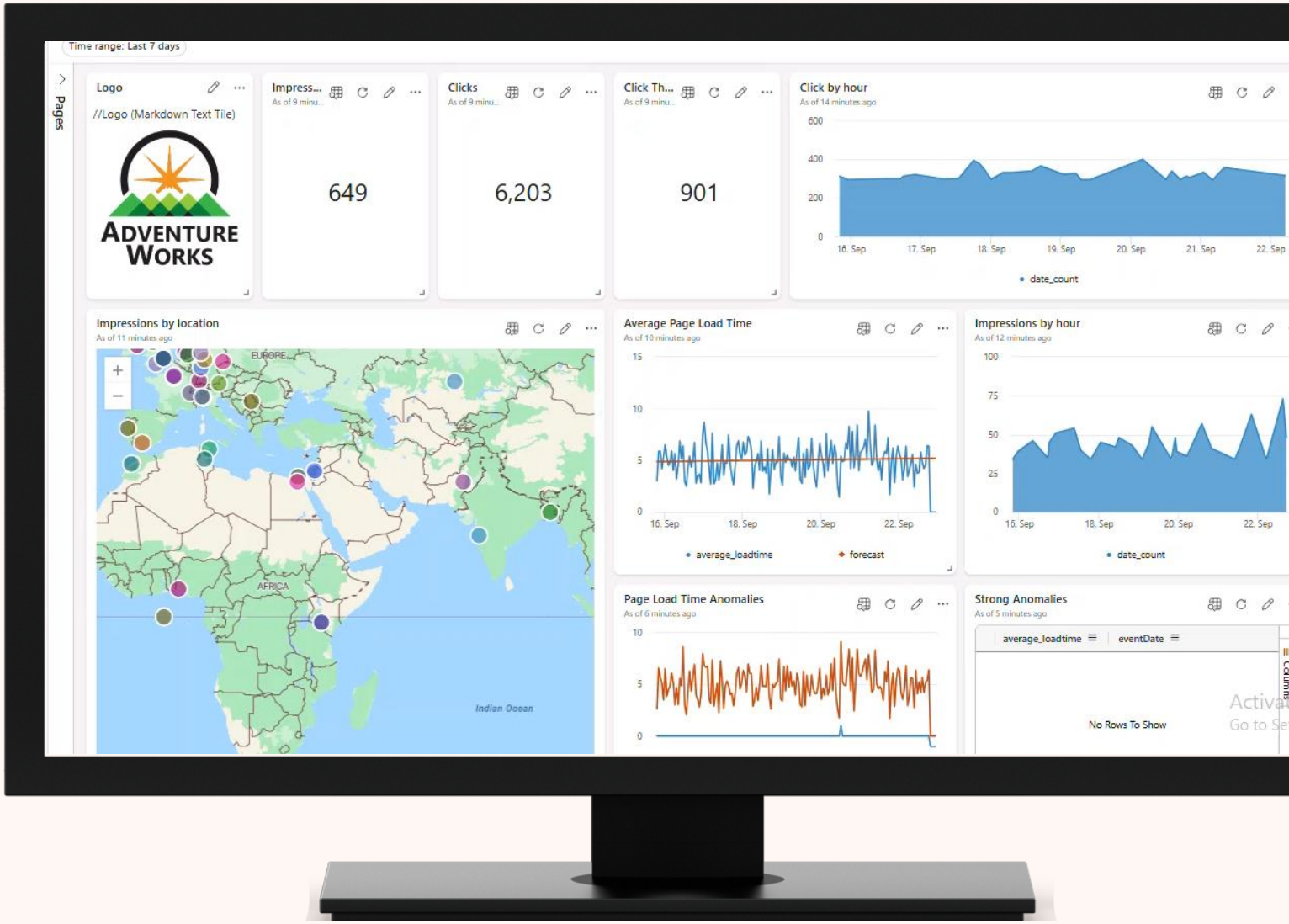
- For Business Users and Developers
- No-code & intuitive UI
- Track business object status over the time, handles complex rules in near real-time

Sources it supports

- Event streams
- Fabric Events [Workspace, JOB, Azure events]
- Power BI
- Realtime Dashboards & KQL Queryset

Actions it can trigger

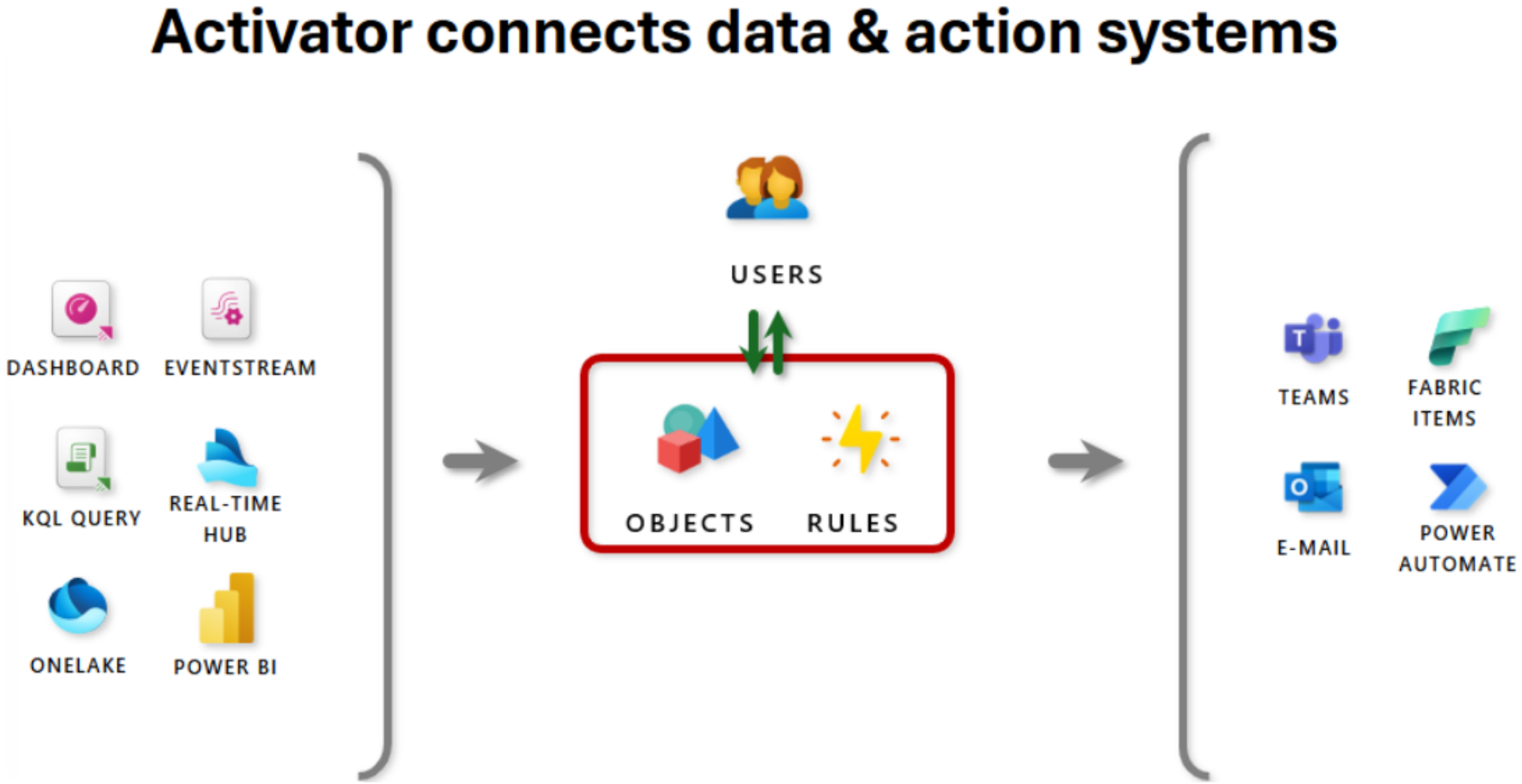
- Run Pipelines, Notebooks
- Teams Messages & E-Mails
- Power Automate Flows



Business Value of Activator

- Enhance business operations in near real-time with data driven actions.
- Activator helps reduce time to action

Architectural Overview



Core Architectural Elements

Event Sources (Eventstream)

Direct connection to eventstream

Source of events

Subscribe to one or more eventstreams

Fabric events

Activator on Power BI Report or Real-Time Dashboard

Events and Objects

Events

Individual records, rules are evaluated per event

Objects

Events are grouped into objects based on shared identifier [Key].

Object states are monitored over time

Rules and Conditions

Activator has one or more rules; evaluated continously

Actions

Can trigger: Data Pipelines, Power Automate Flows, Notebooks. Notify Emals & Teams

Alert Management and Rules Testing

You can see what might happen based on your current data steam so you can prevent too many actions that are triggered

Monitoring and Cost Control

Pricing only applies when rules are executed

Deployment Model

Activator deployed per workspace

Bound to specific sources

Activator can have rules belong to different sources

Multiple Activators can monitor the same stream → parallel rule evaluations

Pricing only applies when rules are actively running and data being ingested → cost efficiency

Event Ingestion and Object Binding

Events ingested into Activator originate from:

- Eventstream, which supports multiple upstream sources (e.g., Azure Event Hubs, IoT Hub)
- Fabric Events
- Azure Events [Blob Storage triggers)]

Each event contains:

- A timestamp
- A payload (structured or semi-structured data)
- One or more attributes used for object identification (e.g., device_id, bikepoint_id)

Object creation is implicit:

- Activator groups events using a designated object key.
- Rules are scoped to objects
- All evaluation logic is object-aware and independent across instances.

Rule Evaluation Engine

- Rules in Activator can be stateless or stateful:
  - **Stateless rules** evaluate each event in isolation (e.g., value < 50)
  - **Stateful rules** maintain memory across events per object (e.g., value DECREASES, BECOMES, EXIT RANGE)
- Stateful evaluation relies on:
  - **Delta detection**: Tracks changes between prior and current event values
  - **Temporal sequencing**: Evaluates time-based conditions like absence of events (heartbeat detection)
  - **State transitions**: Rules only fire on entry into a new state, preventing repeated firings in unchanged conditions
- Each rule condition is compiled into an execution graph that is
  - evaluated continuously,
  - in-memory,
  - and near-instantly.
  - optimized for sub-second decisioning latency after event arrival.

Event deduplication and Alert Suppression

To prevent alert flooding:

Activator maintains a state cache for each object-rule combination

Firing conditions are edge-triggered (not level-triggered)

"Suppression windows" (implicit via state model) reduce redundant actions

This design ensures that a rule like value BECOMES critical only fires when the condition first becomes true—not every time an event with the same value arrives.

Actions and Fabric item invocation and PA flows

When a rule fires:

- Activator emits a trigger message with the current object state and rule metadata
- Payloads are parameterized automatically using event properties (e.g., filename, device status)
- Supported targets:
  - Fabric Pipelines (for data movement, enrichment)
  - Fabric Notebooks (for ML scoring, diagnostics)
  - Power Automate Flows (for business process integration)
  - Teams Notifications (using template-based messaging)
- Actions are non-blocking, and Activator does not wait for downstream execution to complete—enabling scalable asynchronous flows.

Resilience and Error Handling

Timeout behavior:

- If no events arrive for an object after ~5 minutes, rule engine doesn't evaluate it, this is configurable

Monitoring and diagnostics:

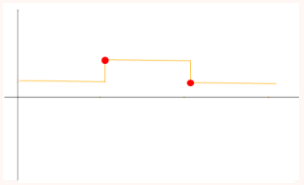
- Trigger logs and preview tools assist in testing and runtime validation

Fault tolerance:

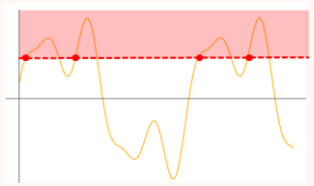
Activator persists minimal metadata; evaluation state is maintained in volatile memory and reconstructed from Eventstream if needed

Typical Implementation Workflow

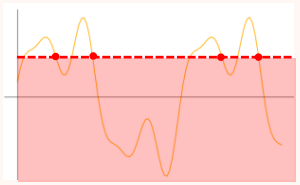
- Ingest data using Eventstream Connect structured or semi-structured sources like Azure Event Hub, IoT Hub, or Blob Storage. Eventstream standardizes input and enables schema projection.
- Define objects and conditions in Activator Choose a key to group events into objects (e.g., device_id, sensor_location). This becomes the unit of rule evaluation. Design rules using natural language operators like:



Changes

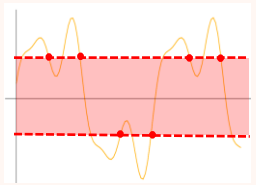


Increases

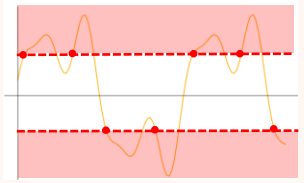


Decreases

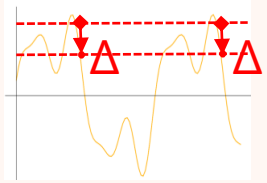
Heartbeat



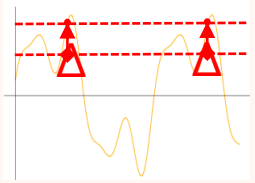
Enter range



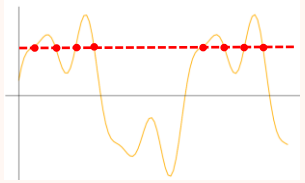
Exit range



Decreases by



Increases by



Becomes

- Bind to downstream actions

Trigger

- Data pipelines to transform or ingest data downstream
- Notebooks for scoring models or enrichment
- Power Automate flows for notifications, tickets, or system updates
- Microsoft Teams for alerting or collaboration (via Flow Builder or future Web API)
- Preview and test Validate the expected behavior before activation. Activator allows rule preview and event replay, showing how often a rule would have fired on recent data.
- Activate and monitor Deploy rules and monitor trigger frequency. Logs are available through Eventstream or tied to triggered actions. Use this to tune rule logic and cost footprint.

Key Design Considerations

- Stateful logic vs. stateless filtering Stateless filters (e.g., value < 5) may be too noisy. Prefer transitional logic like DECREASES or BECOMES to reduce false positives and spam.
- Object key cardinality Each unique object key (e.g., device ID) incurs memory and compute tracking. High-cardinality implementations (>10,000 unique objects) should be carefully profiled to maintain performance.
- Combining rules Activator supports AND to mix condition and filtering. Use them to build complex detection trees (e.g., temperature decreases AND status becomes “critical”).
- Alert fatigue management Design rules to fire on edge transitions only. Activator automatically suppresses repeat alerts unless a new state is entered, but good design further reduces noise.

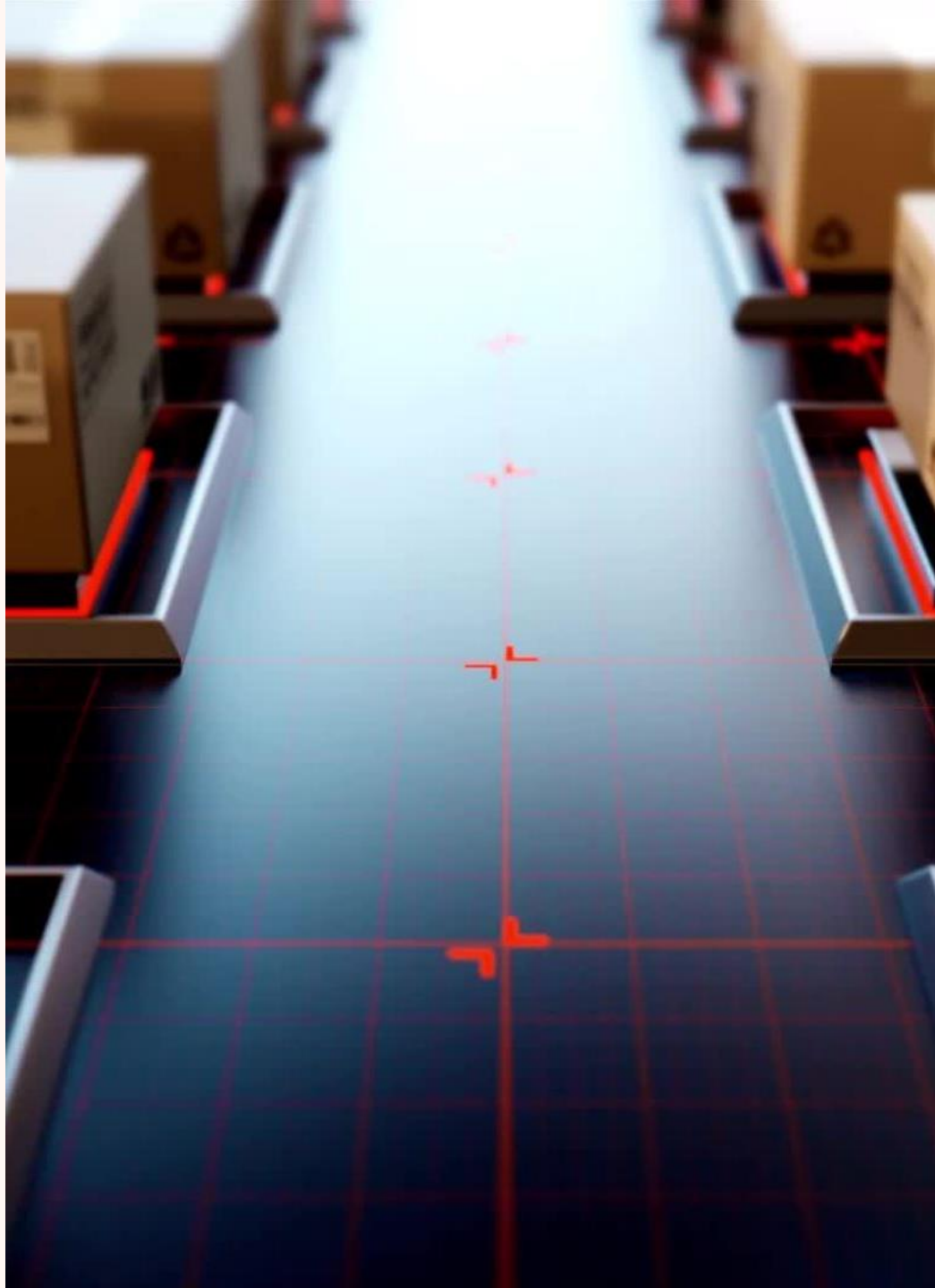
Practical Use Cases

- IoT & Asset Monitoring Trigger pipelines when vibration or temperature decreases across thresholds, or when a device stops sending data (heartbeat failure).
- File-based Workflows In Blob Storage scenarios, use Eventstream to detect file drops (e.g., CSVs) and route metadata (file name, path) into pipelines automatically.
- Public Transportation / Urban Infrastructure As seen in the bikepoint example, Activator can be used to track real-time bike availability at docking stations and send alerts when inventory falls or rises unexpectaedly.
- Security & Compliance Future support for sensitivity labels (e.g., from Reflex) will allow context-aware triggers, such as alerting on files labeled as “Confidential” being moved or accessed.

Reusability and Maintainability

Activator supports multiple rules per object and multiple Activators per stream, enabling modular designs (e.g., separating operational and compliance triggers).

Rules are managed independently, but shared context (object key, schema) enables a common detection fabric across business domains.



Troubleshooting

While Activator abstracts away much of the complexity behind real-time event processing, implementations at scale may encounter data, configuration, or orchestration-related issues that require systematic troubleshooting. This section provides a deep dive into how to identify, analyze, and resolve common operational problems in Activator.

Common issues and symptoms

Symptom	Possible Cause	Remediation
Rules not firing	Eventstream not pushing data, incorrect object keys, rule conditions never met	Validate stream connectivity, use rule preview, check object schema mapping
Alert spam or repeated triggers	Use of stateless operators (e.g., less than) instead of stateful ones (e.g., decreases)	Redesign rules to use transitional operators like DECREASES, BECOMES, or enable debounce logic
No actions triggered	Misconfigured pipeline/notebook/action target, or rule not satisfied	Check action bindings, ensure trigger payloads match pipeline expectations. Test and make sure action works
Unexpected latency (over 30s)	Cold start on Activator, high object cardinality, or Eventstream bottlenecks	Warm up Activator with test events, review object cardinality, check Eventstream diagnostics
“No data” errors after 10 min	Eventstream source is idle or disconnected	Monitor Eventstream source health; implement heartbeat detection rules to proactively catch these outages
Rules firing too frequently in test mode	Using synthetic or replayed test data with fast-changing states	Use controlled, deduplicated test data; apply filters to focus on realistic objects/events

Diagnosing Issues with Built-in Tools

- **Rule Preview Before activating**, use the preview feature to see how often a rule would have fired on historical events. This is critical for:
  - Identifying misconfigured filters
  - Detecting high-frequency triggers
  - Ensuring correct object grouping
- **Eventstream Monitoring Use Eventstream's monitoring panel to:**
  - Confirm event flow from upstream sources
  - Check timestamps and payload structure
  - Identify dropouts or format mismatches
- **Triggered Actions Logs Actions triggered by Activator** (pipelines, flows) log execution metadata:
  - Timestamp, payload, rule that triggered
  - Execution success/failure (via pipeline run history or Power Automate logs)
- **Schema Mismatch** Detection Activator may silently fail to bind rules to data if:
  - The field name is misspelled
  - The value types (e.g., string vs. numeric) are mismatched Use Eventstream's live sample view to confirm schema correctness before rule creation.

Best Practices to Prevent Issues

- Always use rule preview before going live, especially on high-volume streams
- Prefer stateful operators to avoid repeated triggering
- Implement heartbeat rules to catch silent stream failures
- Periodically audit object key cardinality and streamline schemas
- Design rules with monitoring and fallback in mind (e.g., alert if rule hasn't fired in X time)

Orchestration Patterns in Fabric Real-Time Intelligence

Pattern	Flow Description
Ingestion → Detection → Transformation	Events flow from Eventstream into Activator, which triggers a Data Pipeline to enrich or move the data.
Ingestion → Detection → Notification	Activator triggers Power Automate to send alerts or push status into Teams, Outlook, or ServiceNow.
Ingestion → Detection → Model Scoring	Activator triggers a Notebook to score an ML model or perform advanced analytics based on real-time anomalies.
Feedback Loop with Reflex (planned)	Reflex-generated insights (e.g., sensitivity labels) are fed into Activator rules, enabling semantically enriched automation.

Optimization Guidelines

Rule Logic Design

- Use transitional operators (DECREASES, BECOMES, EXIT RANGE) to reduce rule churn.
- Avoid overly broad or nested rules that track too many properties—split rules per logical condition or concern.
- Combine AND/OR conditions carefully; complex chains can slow evaluation on large object sets.

Object Key Management

- Minimize high-cardinality object keys unless necessary. Each unique object incurs memory and compute cost.
- Example: If you're tracking device_id, avoid using a compound key like device_id + timestamp unless temporal uniqueness is essential.

Activation Scope

- Use multiple Activators to separate functional domains (e.g., operations vs. compliance), keeping rule sets manageable.
- Use tagging or naming conventions to group Activators logically by stream, domain, or business function.

Optimization Guidelines

Cold Start & Latency

- Activator exhibits minor cold start delays for first time(~15–30s). Mitigate by:
- Test the rule before go-live.

Action Optimization

- For pipelines: Use parameterized triggers with scoped datasets (e.g., pass in only the bikepoint_id and filter downstream).
- For Power Automate: Use prebuilt templates and message formatting best practices to reduce flow processing overhead.

Scheduling, Dependency, and Coordination

- While Activator is fundamentally event-driven, its outputs often trigger components that require sequential or dependent execution. For example:
- A triggered pipeline may depend on recent output from a Reflex process.
- A notebook scoring function may require cached feature lookups before scoring.
- In such cases:
  - Use pipeline orchestration features (e.g., dependency nodes, activity-level retries) to coordinate downstream processes.
- If conditional branching is needed, use Dataflows Gen2 or Pipelines as orchestrators downstream of Activator—not within Activator itself.

Cost Optimization

Activator is capacity-bound and charged only when running. To reduce cost:

- Deactivate unused or low-value rules during idle periods.
- Use rule preview to assess firing frequency and prune noisy logic.
- Favor centralized Activators when multiple rules share the same eventstream.
- When ingesting data from Eventstream be watchful on events/sec, filter out as needed

Schema Handling in Activator

Activator inherits event schemas directly from Eventstream, which defines the shape of incoming data via structured ingestion sources (e.g., Azure Event Hubs, Blob Storage, IoT Hub).

Key characteristics:

- Schema is inferred dynamically Activator uses the live schema presented in Eventstream samples to generate rule conditions and object mappings. There is no static schema binding—changes are applied as events evolve.

Field names and data types are critical

- Rules are tightly coupled to exact field names (case-sensitive).
- Type mismatches (e.g., using a string field in a numeric comparison) cause silent failures—rules will not evaluate properly, and no error is shown.

Best Practice: Always verify schema field names and types in the Eventstream sample pane before creating rules in Activator.

Missing or optional fields

- If a field is missing in an event but referenced in a rule, the condition may evaluate as false or not at all.
- Design rules to be resilient to partial records where appropriate (e.g., via fallback conditions or default values downstream in the pipeline).

Structured vs. Semi-Structured Input

- Activator is best suited for structured or semi-structured data (e.g., JSON, CSV with consistent keys).
- For unstructured payloads (e.g., raw logs, text blobs), preprocessing must be handled in upstream like Eventstream or Real=Time Dashboards

Object Granularity and Evaluation Impact

- Throughput is heavily influenced by:
  - The number of distinct objects (i.e., unique keys like device_id, sensor_name)
  - The frequency of events per object
  - The complexity of rule logic per object
- Each object incurs:
  - State tracking in volatile memory (if stateful rules are used)
  - Per-object rule execution on each new event

Throughput and Latency Benchmarks

Factor	Observed Behavior
Cold Start	Time 15–30 seconds from activation to first rule evaluation
Evaluation Latency	Typically <5s from event arrival to rule firing
Event Ingestion Rate	Scales with Eventstream limits; Activator is optimized for high-volume ingestion
Rule Evaluation Rate	Parallelized across objects; bottlenecks occur with deep nesting or excessive rule chains

Optimization Techniques

Limit schema width
Avoid pushing entire event payloads if only 2–3 fields are used in rules.

Normalize event shape upstream
Ensure consistent schemas to prevent schema drift or incompatible types from breaking rules.

Control object key cardinality
Design Activators with carefully scoped object keys (e.g., avoid time-based IDs or highly unique keys that offer no reuse).

Test with representative loads
Use live sample previews in Eventstream and dry-run Activator rules to model throughput impact before going live.

Monitoring Activator in Production

Rule Preview and Impact Estimation

- Before activating any rule, use Preview Mode to simulate how often it would have fired against recent event data. This helps assess:
  - Trigger frequency
  - Data noise level
  - Risk of alert flooding

Ideal for validating high-volume rules or new schema inputs.

Live Monitoring via Eventstream

Activator receives its data from Eventstream, which exposes:

- Source health and connection status
- Incoming event rates
- Schema snapshots

Use this to diagnose upstream data delays, gaps, or schema changes.

Monitoring Activator in Production

Triggered Action Logs

- Every time a rule fires and triggers an action (pipeline, notebook, Power Automate flow), metadata is logged:
  - Timestamp
  - Triggered rule ID
  - Action result (success/failure)

For pipelines and notebooks, use Fabric monitoring tools to inspect run status, duration, and errors. For Power Automate, use the Flow Run History for each trigger.

Heartbeat and Inactivity Detection

- Activator supports rules that detect absence of data (e.g., if no event is received for a defined time period).
- These are critical for monitoring stream health and data source availability.

Alert Effectiveness Audits

- Review how many alerts resulted in downstream impact (e.g., notebook executed, incident resolved).
- Helps tune false positive rates and eliminate redundant rules.

Pricing Model

Activator follows a capacity-based pricing model within Microsoft Fabric:

Pricing Principle	Description
Per capacity, not per rule	You pay for Activator runtime only while rules are actively running in your Fabric capacity and data being observed by Activator.
Usage-based billing	Cost is driven by duration of activation and number of running rules- not volume of events or number of rules.
Deactivation halts billing	If rules is paused & ingestion stopped, it stops incurring cost.

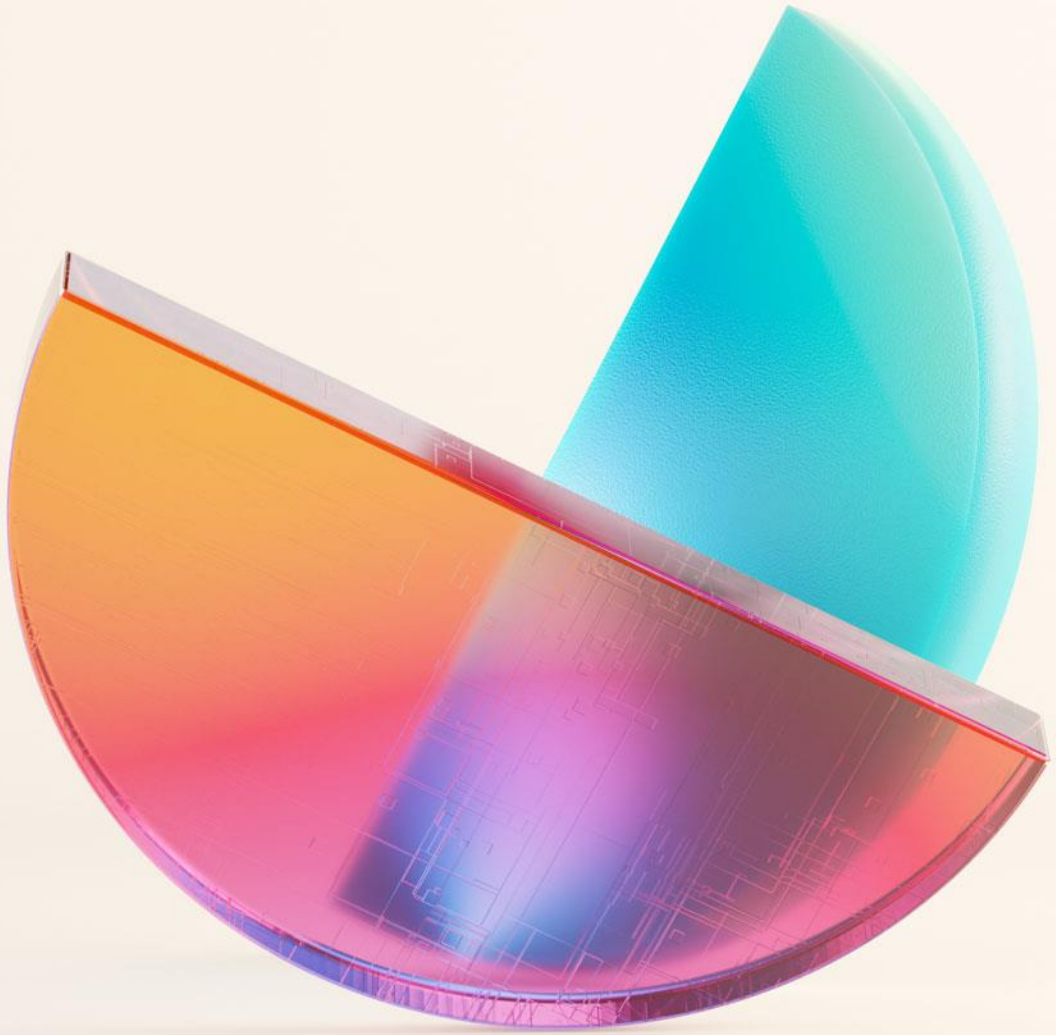
Best Practice
Deactivate inactive or experimental rules when not in use to avoid unnecessary runtime charges.

Cost Optimization Strategies

- Use Rule Preview to estimate volume Don’t activate rules blind. Preview them first to avoid overly aggressive triggers.
- Group related rules logically Reduce the number of separate Activators by using multiple rules within a single instance (when applicable). This lowers activation cost and simplifies monitoring.
- Control object cardinality Large numbers of distinct objects (e.g., millions of device IDs) can increase processing overhead—even if the rule volume is stable.
- Batch actions downstream Where appropriate, trigger pipelines or flows that can handle multiple events in one execution to reduce downstream workload costs.

Microsoft

Microsoft Fabric



RTI Cost Estimation

What’s behind an Eventstream?

Sources

- Custom Endpoint
- Connected Sources
- Sample Data

Stream

- Under the hood Eventhub

Transformational

(Optional)

- Aggregate
- Expand
- Filter
- Manage Fields
- Union
- Group By
- Join

Under the hood stream analytics jobs

Destinations

- Custom Endpoint
- Derived Stream
- Eventhouse
  - Direct (w/o transformation)
  - Event processing before ingestion
- Lakehouse
- Activator

Eventstream Pricing Model

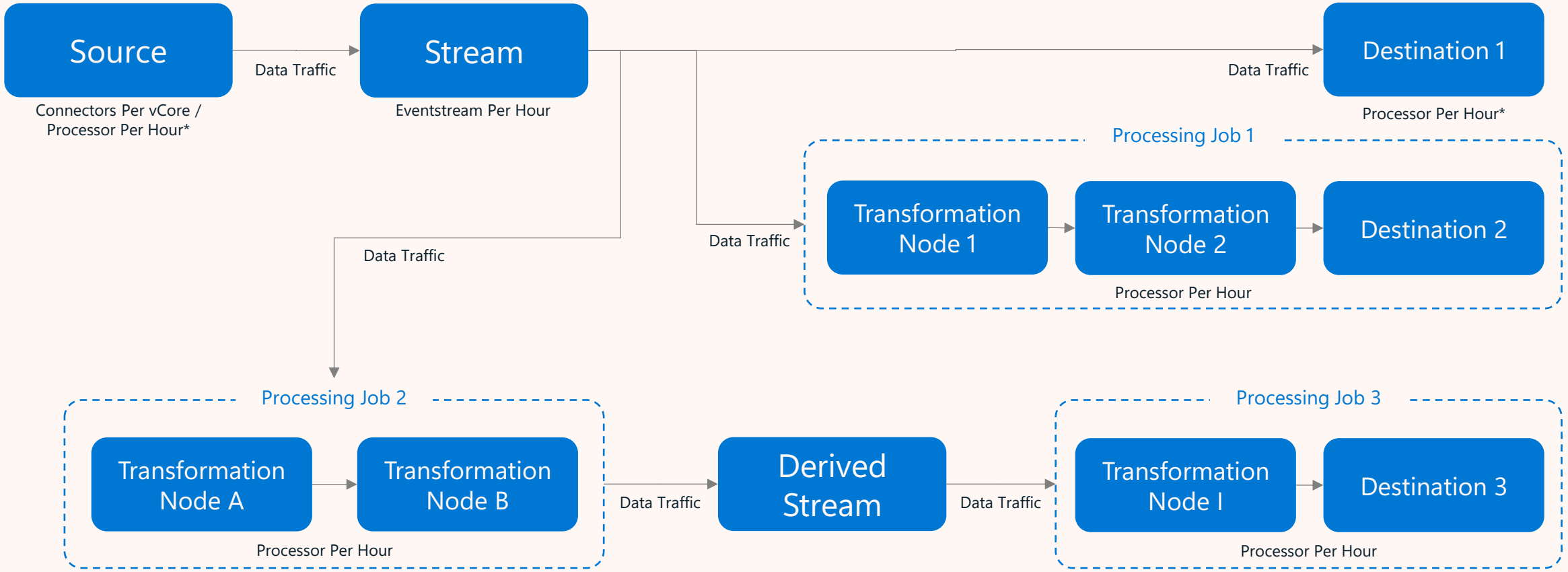
Operation types

Operation	Description
Eventstream Per Hour	A flat charge while the Eventstream is active (event´s been flowing into the stream). If there's no traffic flowing in or out for the past two hours, no charges apply.
Eventstream Data Traffic per GB	Data ingress & egress volume in default and derived streams (Includes 24-hour retention)
Eventstream Processor Per Hour*	Computing resources consumed by the ASA Jobs needed for processing data.
Eventstream Connectors Per vCore Hour**	Computing resources consumed by the connectors.

*CU consumption of the Eventstram Processor Per Hour is designed to correlate with throughput, complexity of processing logic, and partition count of input data. The process autoscale accordingly.
**CU consumption of the Eventstream Connectors Per vCore Hour is designed to correlate with throughput. The process autoscale accordingly.

Microsoft Fabric event streams capacity consumption - Microsoft Fabric | Microsoft Learn

How does Eventstream treat CU with multiple routing and transformation paths?



*Custom Endpoints and Eventhouse Direct Ingest don´t contribute to CU consumption

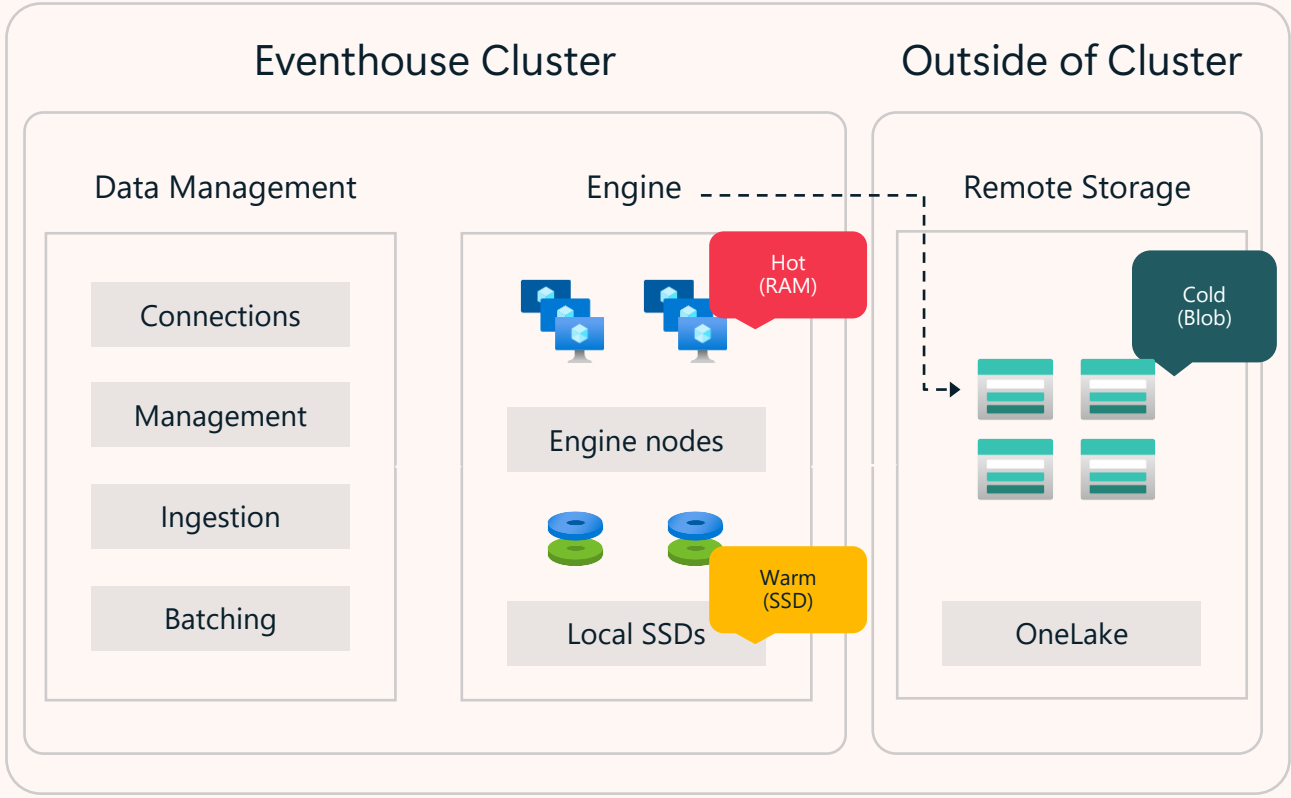
Eventstream Pricing Model

Storage billing

Pricing Principle	Description
OneLake Standard Storage	Standard storage that's used to persist and store all data. When you set the retention setting for more than 1 day (that is, 24 hours), you're charged as per OneLake Standard storage

Microsoft Fabric event streams capacity consumption - Microsoft Fabric | Microsoft Learn

What´s behind an Eventhouse?



Eventhouse Pricing Model

Eventhouse uses a cluster model for the consumption of CU

Pricing	Description
Eventhouse UpTime	The number of seconds that your eventhouse is active in relation to the number of virtual cores used by your eventhouse. An autoscale mechanism is used to determine the size of your eventhouse. This mechanism ensures cost and performance optimization based on your usage pattern. An eventhouse with multiple KQL databases attached to it only shows Eventhouse UpTime for the eventhouse item.
OneLake Cache Storage*	Premium storage that is utilized to provide the fastest query response times. When you set the cache policy, you affect this storage tier.*
OneLake Standard Storage	Standard storage that is used to persist and store all queryable data. When you set the retention policy, you affect this storage tier.

*Enabling minimum consumption means that you aren't charged for OneLake Cache Storage. When minimum capacity is set, the eventhouse is always active resulting in 100% Eventhouse UpTime.

Eventhouse and KQL Database consumption - Microsoft Fabric | Microsoft Learn