

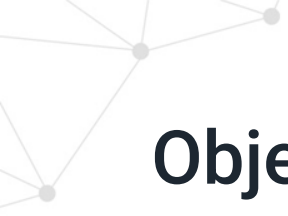


ONLINE TRAINING SERIES

Crosser Academy

EDGE ANALYTICS TRAINING

Fundamentals



Objectives

- Understand what the Crosser Streaming Analytics solution can do
- Basic understanding of stream-based processing
- Familiarize yourself with the Crosser module library
- Hands-on experience of implementing and deploying basic use cases
- How to setup Crosser Nodes for local processing

At the end of the course you should be ready to setup your own edge analytics use cases

Learning by Doing!

Course Setup

- You can use any account to follow this course, including free-trials
- Six sessions with presentations and exercises
 - Video recordings available
 - Presentations as PDFs available for online viewing or download
 - Expected total time to complete the course: 15-20 hours
- Documentation is available in our Help Center (docs.crosser.io):
 - Crosser Control Center documentation
 - Crosser Academy
 - Presentations
 - Videos
- If you have questions, use our support portal (support.crosser.io)
- The exercises are the key to success
 - Work through the exercises in order, they are there for a reason!
- After completing the exercises and the exam you get a diploma

Sessions

- First session
 - Second Session
 - Third Session
 - Fourth Session
 - Fifth Session
 - Sixth Session
- Introduction to Crosser Streaming Analytics solution
 - Create your first flow
 - Streaming data
 - Working with arrays
 - Non-streaming data and local node installation
 - Working with files
 - User management introduction
 - Examination



SESSION 01

Edge Analytics Fundamentals

TRAINING INTRODUCTION

Introduction to Crosser Streaming
Analytics solution



Session 1

Agenda

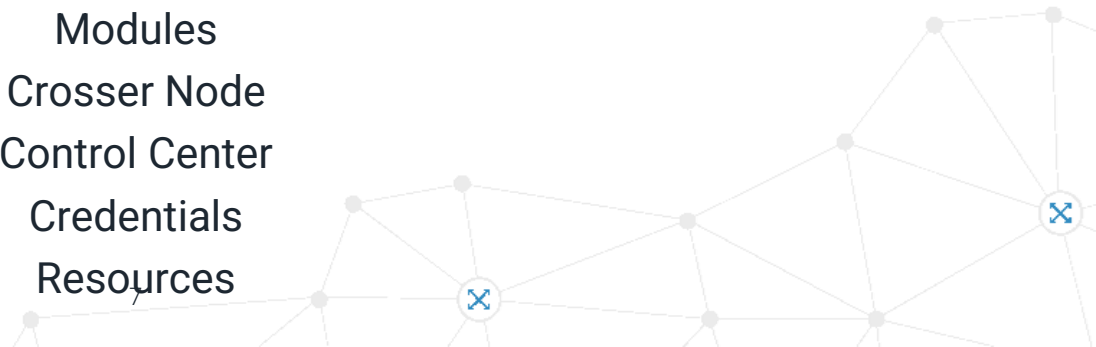
- Crosser Solution Concepts
- Flow-based processing
- Understanding Messages

No exercises in this session



CROSSER SOLUTION CONCEPTS

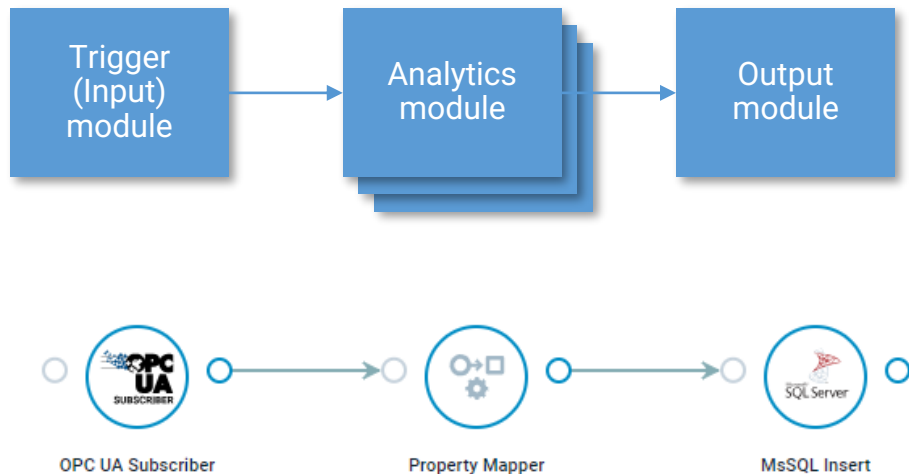
Flows
Messages
Modules
Crosser Node
Control Center
Credentials
Resources



Flows

Processing Messages with Modules

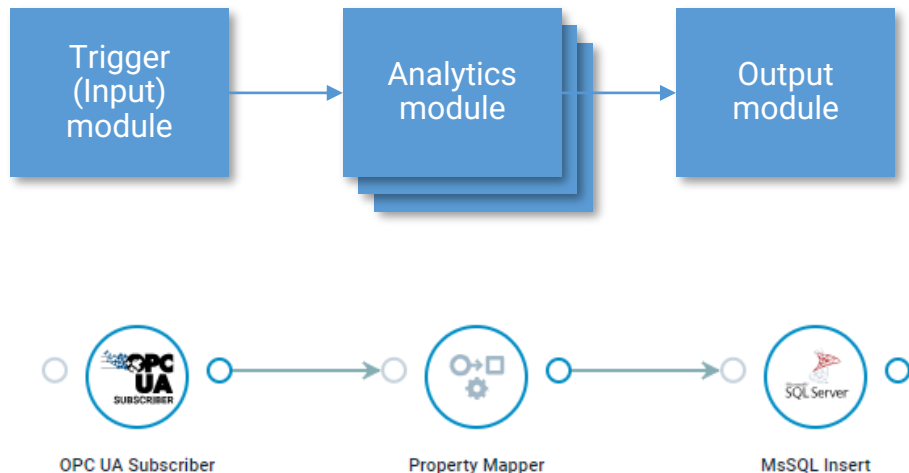
- Crosser uses Flows to implement processing
- A **Flow** consists of **Modules** that process **Messages**
- A **Message** can contain e.g:
 - Sensor values
 - The result of a database query
 - The response from an API call
 - Binary data, such as image, video and audio



Modules

Processing Messages

- A **Module** process messages:
 - **Trigger modules** - starts processing, e.g. at regular intervals or when data arrives
 - **Input modules** - get data from external sources, e.g. PLCs, APIs, MQTT, databases, files
 - **Analytics modules** - do something with the message data they receive and produce modified output messages
 - **Output modules** - send processed messages to external destinations, e.g. PLCs, databases, files, enterprise, cloud services
- The Crosser module library has ~200 modules, can be extended by customers or partners



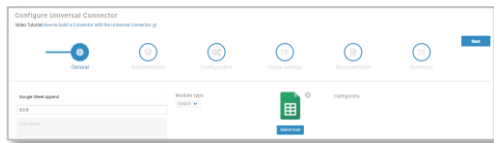
Custom Functionality

Extend the Crosser module library



Code modules

Drop in your own C#, Python or JavaScript code
Standard Python with support for third-party
libraries, including ML



Universal Connectors

Wizard-based tool to create custom
connector modules for REST APIs

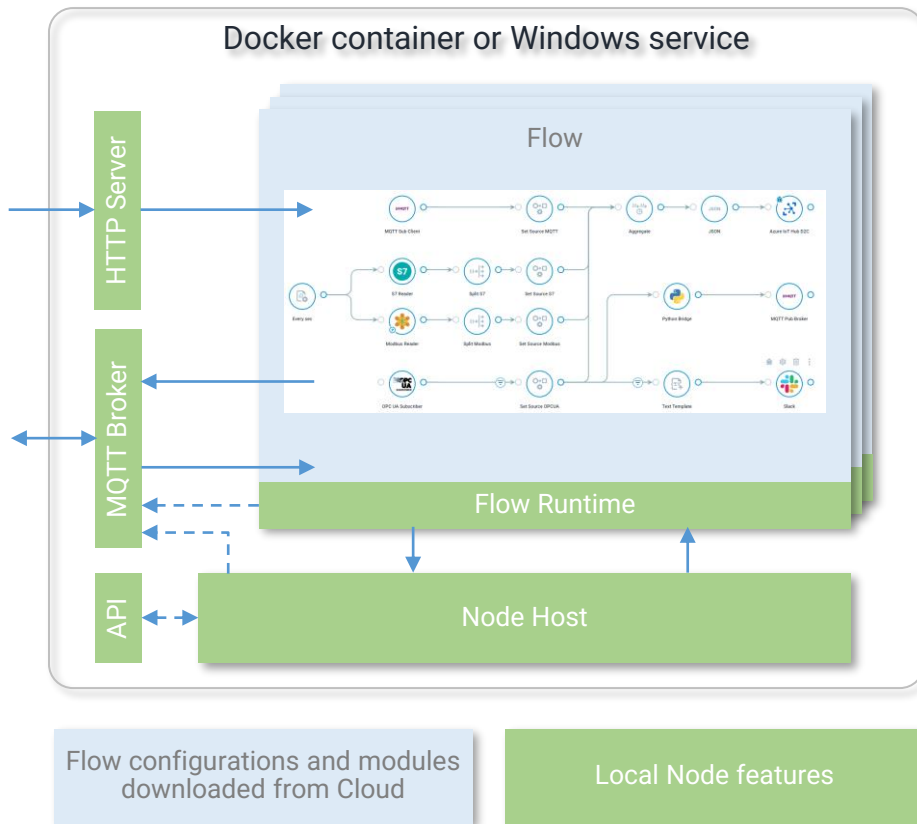


Crosser SDK

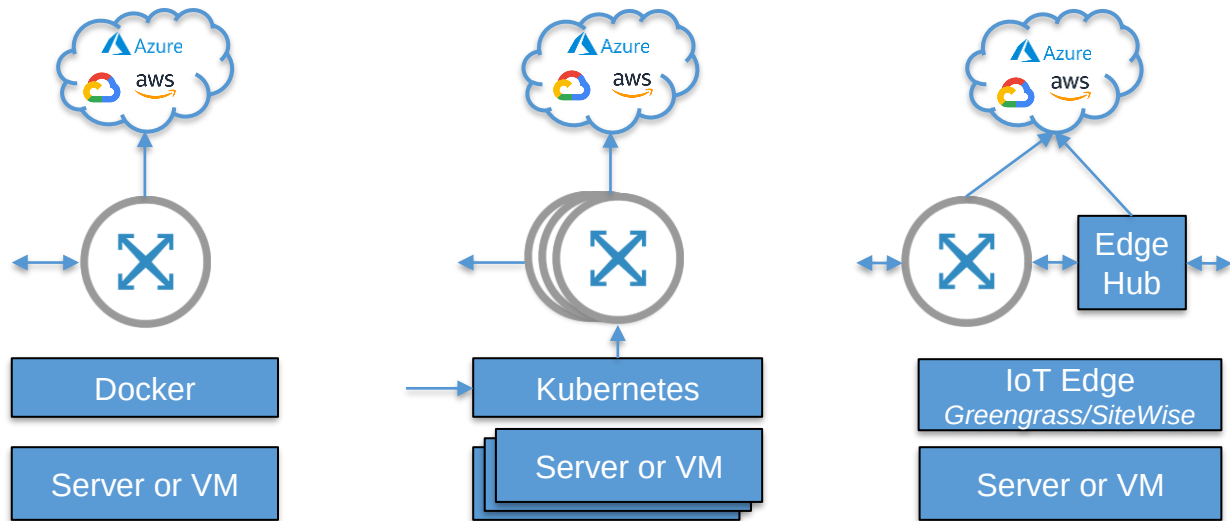
.NET SDK for building fully custom
modules

Flows run on Nodes

- Nodes can be **installed anywhere**: edge, on-prem, cloud
- A Flow is a **configuration** that can be changed any time
- A Node can run **multiple Flows**
- The same Flow can be deployed on **any number of Nodes**
- Flows can be updated or added **without affecting other Flows** on the same Node
- Functionality provided through modules and **Modules are downloaded together with Flows**
- **Local web/API** for monitoring and queue management
- Events published on the **internal MQTT broker**



Node Deployment Options



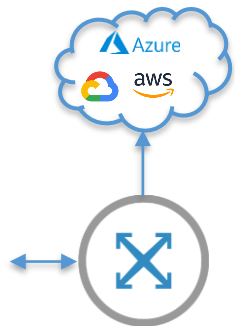
Platform	Docker	Kubernetes/OpenShift	Azure IoT Edge/ AWS Greengrass/SiteWise Edge
Management	Local (Remote)	Remote	Remote
Tools	docker-compose (Podman, Portainer...)	kubectl	Azure/AWS portal, CLI, API
High-availability (multi-server)	No	Yes	No

Node Deployment Options

Windows servers

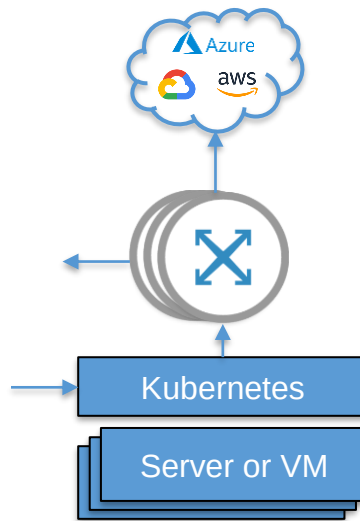
The Node can also be installed as a Windows service.

Requires manual installation of Python



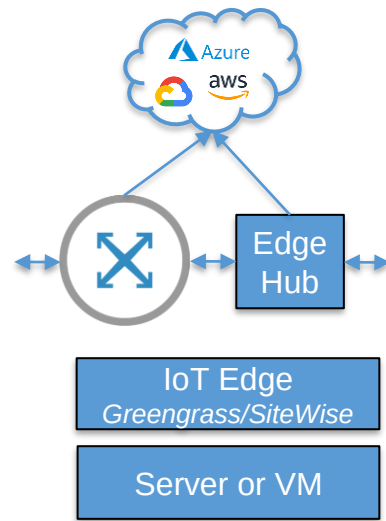
Docker

Server or VM



Kubernetes

Server or VM



IoT Edge

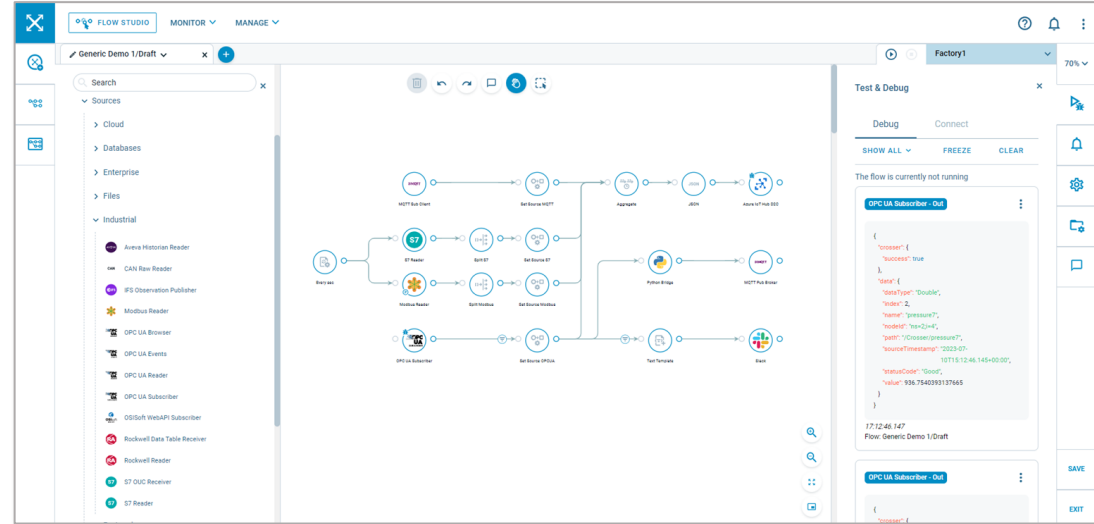
Greengrass/SiteWise

Server or VM

Platform	Docker	Kubernetes/OpenShift	Azure IoT Edge/ AWS Greengrass/SiteWise Edge
Management	Local (Remote)	Remote	Remote
Tools	docker-compose (Podman, Portainer...)	kubectl	Azure/AWS portal, CLI, API
High-availability (multi-server)	No	Yes	No

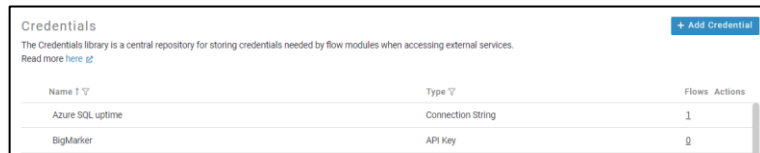
Cro
Cent

- Hosted by Crosser (Azure) or by Customers or Partners (white-label)
- Flow Design
 - Flow Studio Visual design tool
- Flow Management
 - Flow versioning
 - Flow deployment
- Node Management
 - Registration
- Monitoring
 - Flows and Nodes
- User Management
 - Access control
 - Internal or external authentication (OpenID connect)



Credentials

- The Credentials library is used to store credentials that will be used for access to external services
- Credentials could for example be:
 - Username/password
 - API keys
 - Connection strings
- Credentials are used in modules settings
- All credentials are stored encrypted both in the Crosser Control Center and the Node and the data is never exposed to users



The screenshot shows the 'Credentials' page in the Crosser interface. At the top, there's a header 'Credentials' with a sub-header 'The Credentials library is a central repository for storing credentials needed by flow modules when accessing external services.' and a link 'Read more [here](#)'. A blue button '+ Add Credential' is in the top right corner. Below this is a table with columns 'Name', 'Type', and 'Flows'. The table contains two entries: 'Azure SQL uptime' with type 'Connection String' and 1 flow, and 'BigMarker' with type 'API Key' and 0 flows. There are also 'Actions' icons for each entry.

Name	Type	Flows	Actions
Azure SQL uptime	Connection String	1	
BigMarker	API Key	0	

*Add credentials on the
Credentials page or from
module settings*

Resources

- Resources are additional data needed by your flows, for example:
 - PLC tag lists
 - Python or C# code
 - Any file, e.g. ML models, CSV files...
- Resources are downloaded into local storage when the flow is deployed and can be accessed from within a module when running on the edge node.
- Added on the **Resources** panel in the *Flow Studio* or on the *Resources page*, by uploading a file or by entering information straight into the UI.

```
1 {
2   "name": "compressor_01",
3   "unitId": "1",
4   "tags": [
5     {
6       "id": "Nm3",
7       "name": "Nm3",
8       "modbusDataType": "Float",
9       "modbusFunction": "ReadHoldingRegisters",
10      "address": "0088"
11     },
12     {
13       "id": "bar",
14       "name": "bar",
15       "modbusDataType": "Float",
16       "modbusFunction": "ReadHoldingRegisters",
17       "address": "0028"
18     }
19   ]
20 }
```

Name ↑ ▾	Resource type ▾	Created at	Status ▾	Flows	Actions
cloudstatus.txt	File	2020-11-09 11:59	🟢	0	
OLT Modbus Demo	File	2020-12-11 14:59	🟢	1	
OLT Training CSV data	File	2020-12-11 15:00	🟢	0	
OLT Work order XML template	File	2020-12-11 15:00	🟢	0	
OPC UA tags	OPC	2021-12-07 15:25	🟢	0	

From Idea to Reality

Low Code Design | Central Orchestration | Local Execution

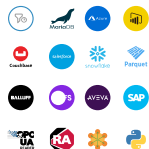


Create Flows



Deploy Flows

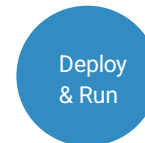
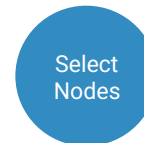
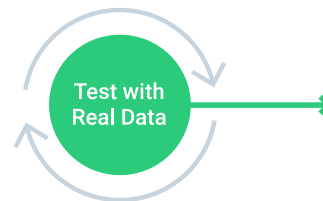
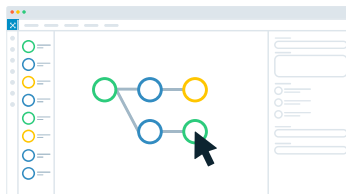
Module Library



New Flow



Existing Flow



Start from scratch or use a template or an existing flow.

Combine pre-built modules into processing flows with drag&drop editor. Add custom algorithms and/or Machine Learning.

Connect to real data sources and test your data flow.

Select one or many nodes and deploy your data flow.



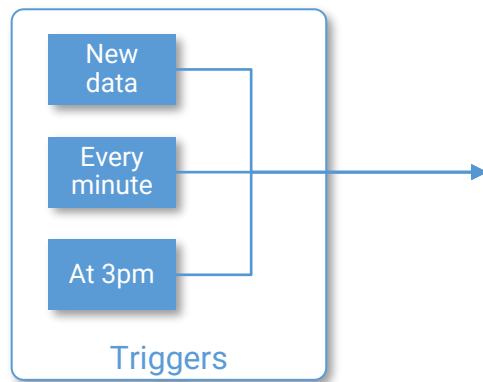
FLOW BASED PROCESSING

Basic structure of Flows

How to implement logic with Flows

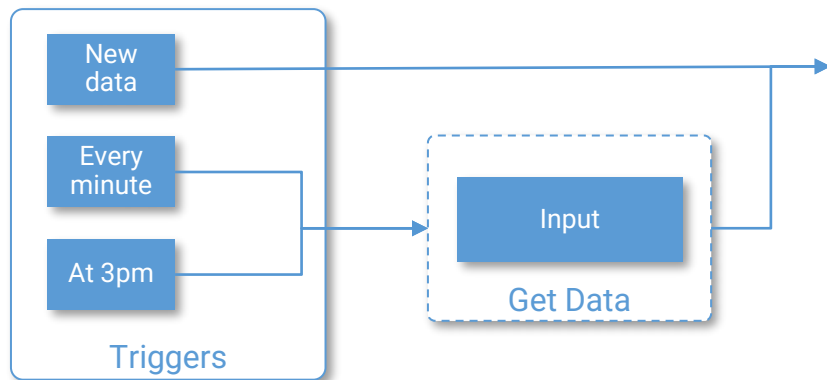
Flow processing

Start with a Trigger



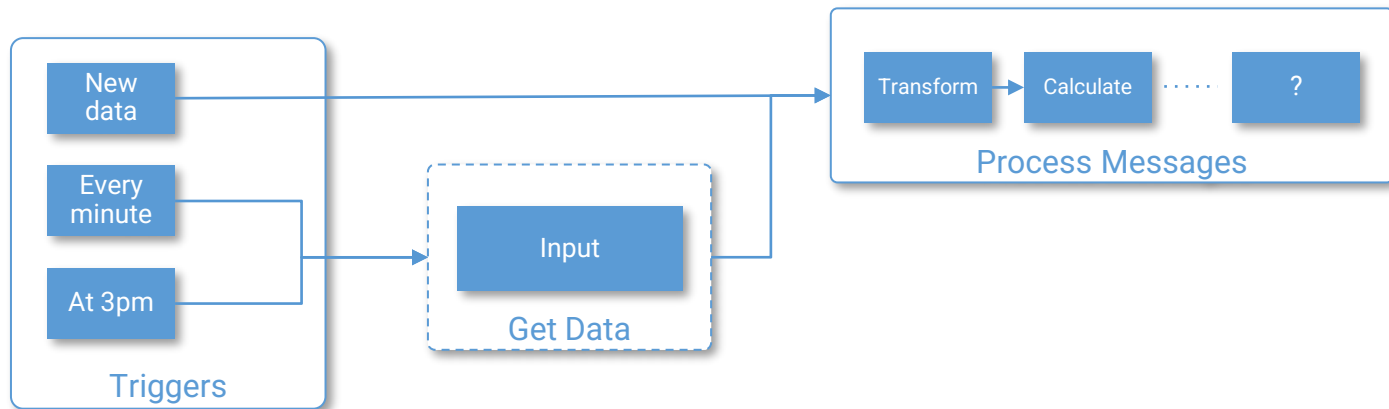
Flow processing

Get Data (pull)



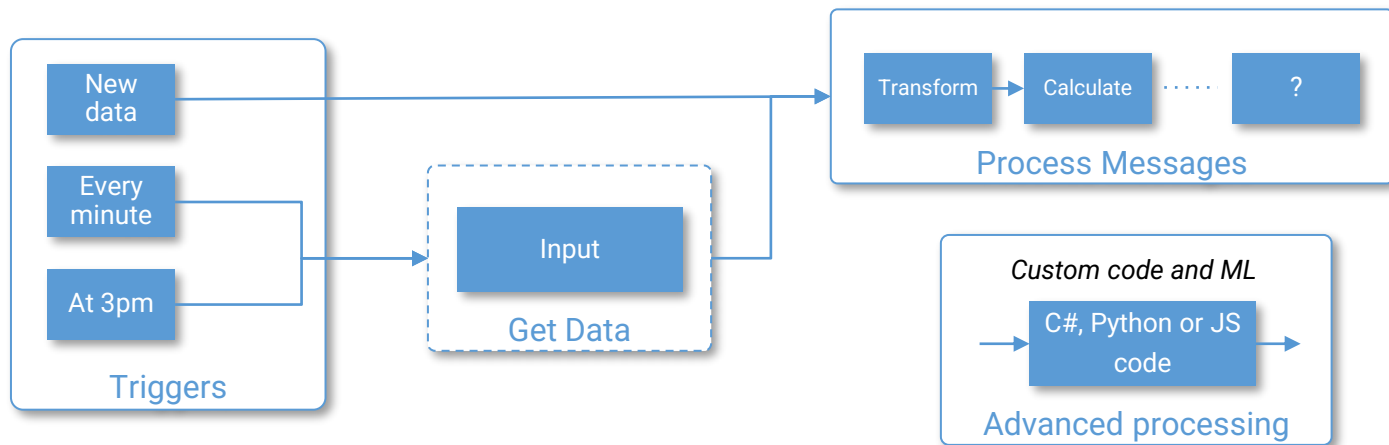
Flow processing

Do something with the Data



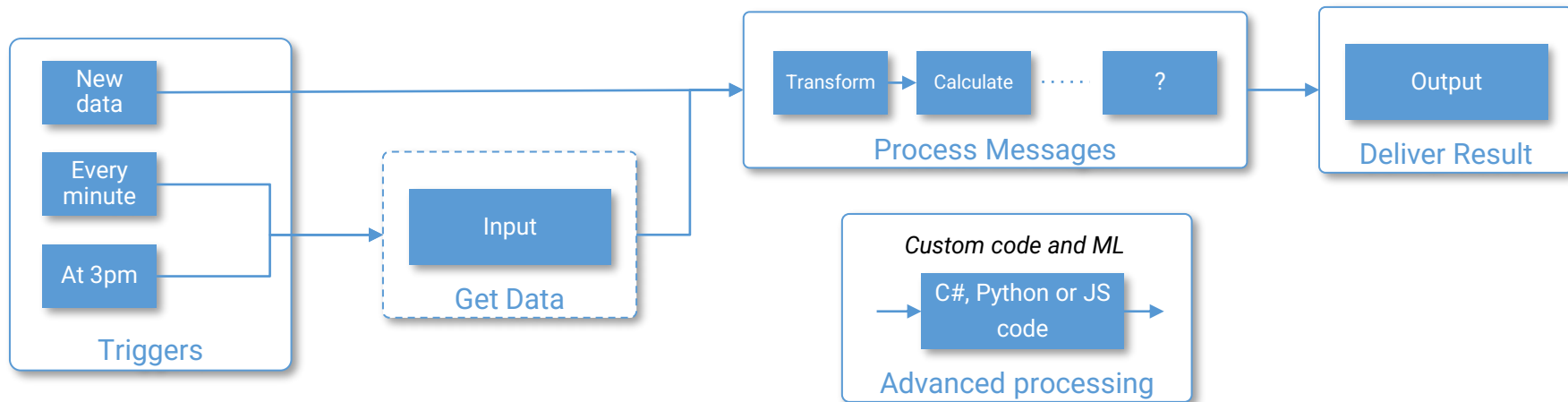
Flow processing

Do something with the Data



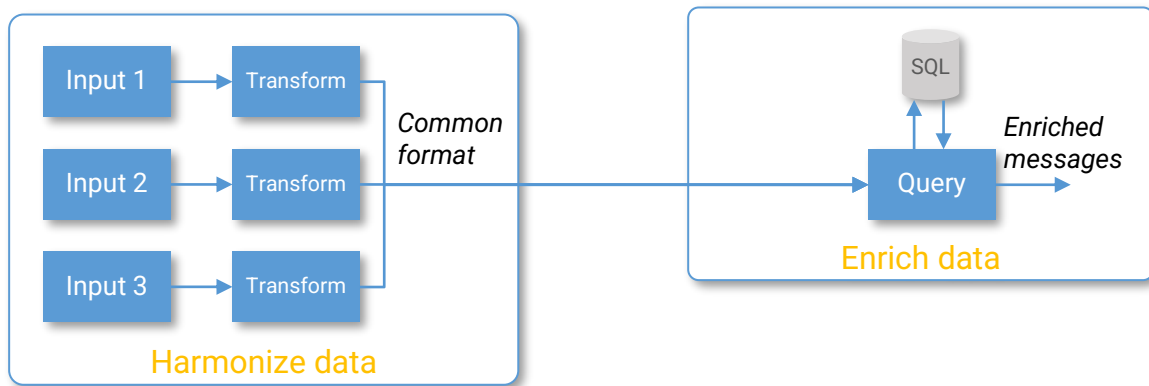
Flow processing

Deliver the Result



Flow processing

Using multiple Input sources

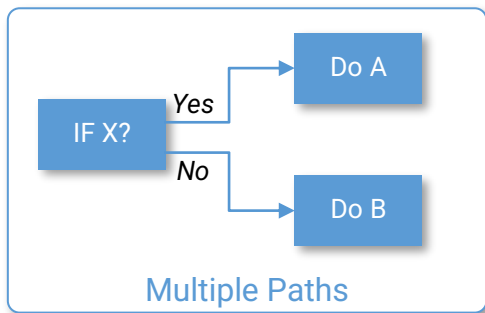


Harmonize data from different sources to simplify downstream processing

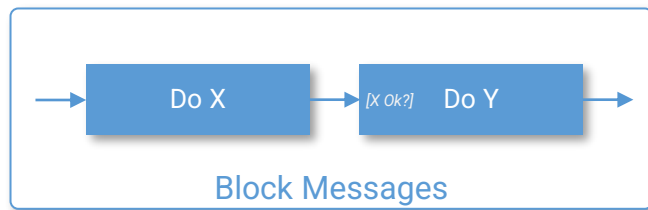
Use data from one source to request additional data from another source

Flow processing

Conditional Processing



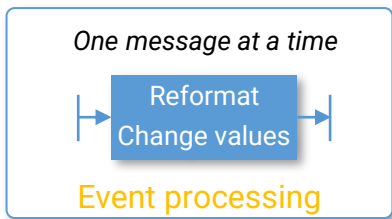
Create separate paths
based on conditions
(Split module)



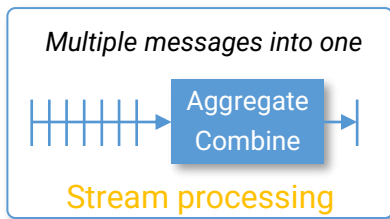
Only process messages
that meet some condition
(Message Filters)

Flow processing

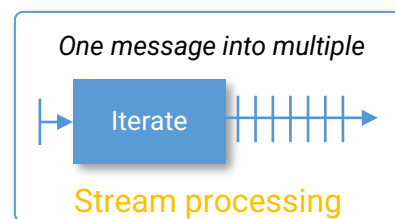
Events and Streams



Each message is processed independently



A sequence of messages is used by the module



An array is broken up into individual messages



UNDERSTANDING MESSAGES

Message structure (Properties and Values)

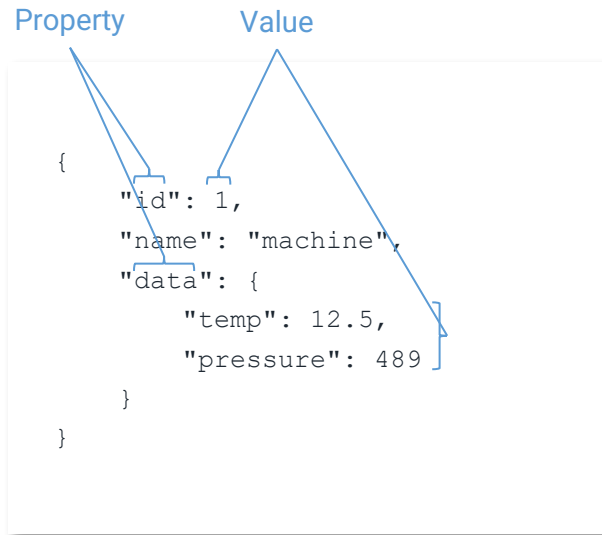
Accessing data in Modules

Building up Messages

Message Structure

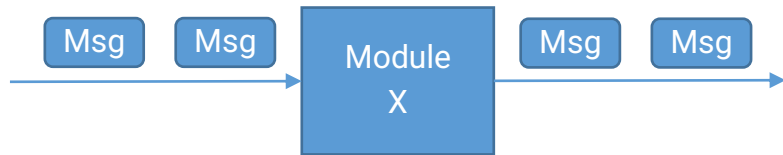
- Messages are objects with *properties* and *values*
- The value of a property can be another object, thereby creating hierarchies
- The value of a property can be an array, where each element is either a simple value or an object
- Hierarchies are traversed using "." notation, e.g. *data.temp*
- Values have a type, eg int, float, string, object...
- Messages use a custom .NET type: *FlowMessage**

**) When presented in the FlowStudio it may look like JSON, but it's NOT*



Messages in Modules

- Modules act on messages, they will only do something when a message is received
- A Module uses selected parts of incoming messages, based on settings, and can create modified or new messages on the output
- A module can:
 - Produce an output message for each input message, e.g. *Property Mapper*
 - Create one message per multiple input messages, e.g. *Aggregate*
 - Create multiple output messages for each input message, e.g. *Array Split*
- Modules can only use data with the correct type



Accessing Message Data in Modules

Input message

```
{
  "name": "machine-3",
  "data": {
    "temp": 12.5,
    "pressure": 489
  }
}
```

Module settings

Source Property	name
Property to read the name from	
Value Property	data.temp
Property to read the value from	
Target Property	aggregate
Property to write the result into, empty is allowed.	

Interval

Seconds

Aggregation time before calculation

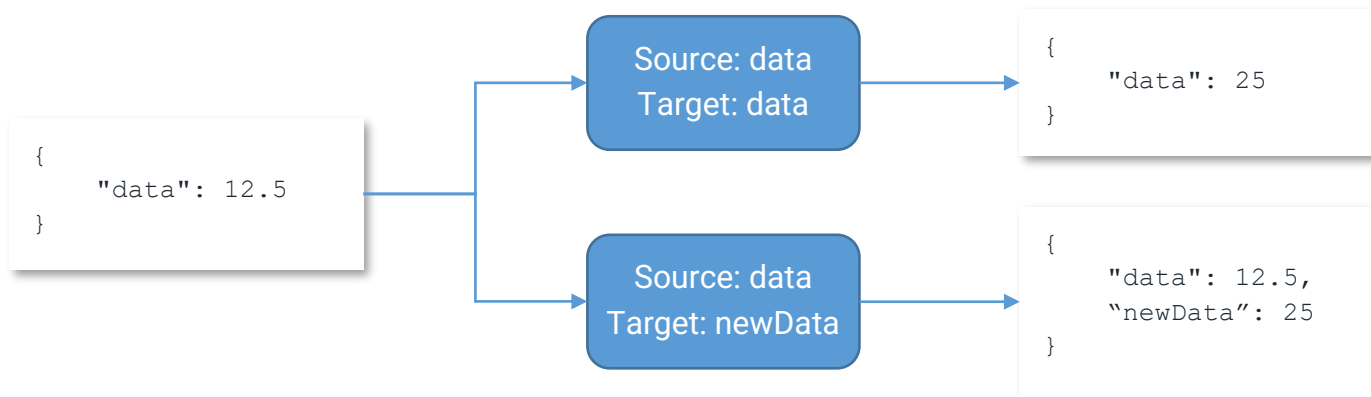
Output message

```
{
  "name": "machine-3",
  "aggregate": {
    "average": 62,
    "name": "machine-3",
    "min": 13,
    "max": 87,
    "count": 3,
    "timestamp": "2020-02-21..."
  }
}
```

- *Property* settings reference message data
- Other settings control the operation of the module

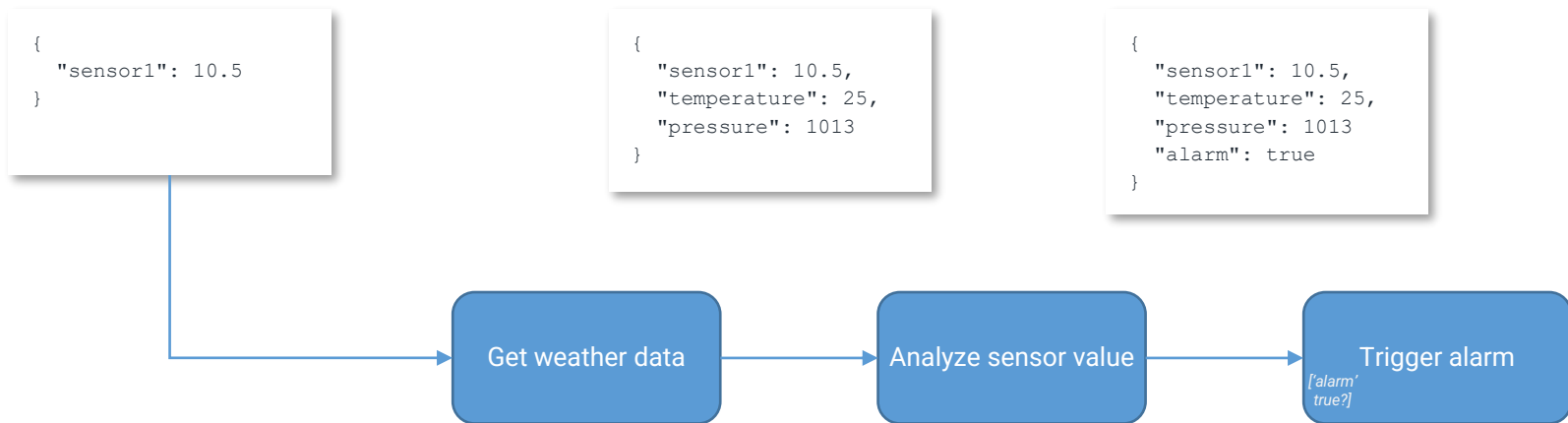
Source and Target properties

Overwrite or Add data



Modules can **add** new or **overwrite** existing properties (Target)
Properties remain in messages until explicitly removed (Property Mapper)

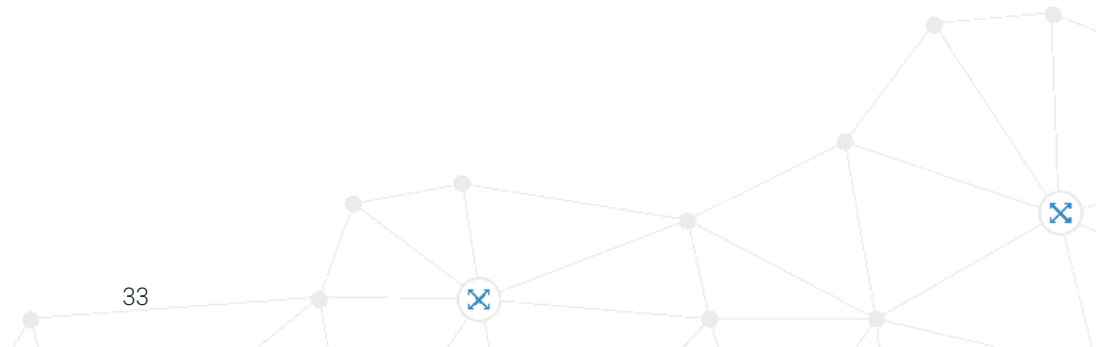
Building up Messages



Most modules can only reference data in a single message!
Keep data together in a single message by adding properties through each module



SESSION – 01 END





SESSION 02

Edge Analytics Fundamentals

CREATE YOUR FIRST FLOW

Work with modules and flows