**🧩 QuantumStream Plugin System – Developer Integration Guide**

**📦 Feature:** Modular Plugin Architecture\ **🛠️ Available Since:** v2.4.0\ **📅 Last Updated:** July 2025\ **🎯 Objective:** *"Extend, customize, and scale QuantumStream with modular plugins."*

**🧠 Overview**

The **QuantumStream Plugin System** enables developers to extend the core functionality of the QuantumStream platform through a modular, interface-driven architecture. Plugins can be used to introduce custom data processors, connectors, validators, metrics exporters, and more—without modifying the core codebase.

This system promotes flexibility, reusability, and clean separation of concerns, making it ideal for teams building specialized streaming solutions or integrating with third-party systems.

**🧱 Plugin Architecture**

* **Interface:** All plugins must implement the IQuantumPlugin interface.
* **Types of Plugins Supported:**
  + Data transformers
  + Source/sink connectors
  + Stream validators
  + Custom metrics reporters
  + Event hooks (e.g., onStart, onError, onShutdown)
* **Plugin Lifecycle Hooks:**
  + initialize(config)
  + start()
  + process(record)
  + shutdown()

**⚙️ Plugin Registration**

To register a plugin, update your project’s qs.config.yaml file:

plugins:

- name: "CustomTransformer"

path: "./plugins/custom\_transformer.py"

class: "CustomTransformer"

enabled: true

Each plugin entry must include:

* name: A unique identifier
* path: Relative or absolute path to the plugin file
* class: The class name implementing IQuantumPlugin
* enabled: Boolean flag to activate or deactivate the plugin

**🧪 Example Plugin Implementation**

from quantumstream.interfaces import IQuantumPlugin

class CustomTransformer(IQuantumPlugin):

def initialize(self, config):

self.prefix = config.get("prefix", "LOG")

def process(self, record):

record["message"] = f"{self.prefix}: {record['message']}"

return record

def shutdown(self):

print("CustomTransformer shutting down.")

**🧰 Development Tools & Best Practices**

* Use the qs validate command to check plugin compatibility and syntax.
* Log plugin activity using the built-in qs.logger module.
* Isolate plugin dependencies using virtual environments or containers.
* Write unit tests for each plugin method to ensure reliability.

**📊 Monitoring Plugin Activity**

Plugins can emit custom metrics using the qs.metrics API:

qs.metrics.increment("custom\_transformer.records\_processed")

These metrics will appear in the qs monitor dashboard under the “Plugins” tab.

**🧭 FAQs**

**Q: Can I load multiple plugins at once?**\ A: Yes. List multiple entries under the plugins section in your config file.

**Q: Are plugins hot-reloadable?**\ A: Not currently. Changes require a restart of the stream process.

**Q: Can plugins be written in other languages?**\ A: No. Plugins must be written in Python and conform to the IQuantumPlugin interface.

**Q: Is there a plugin marketplace?**\ A: A community plugin registry is in development and will be available in Q1 2026.

The QuantumStream Plugin System empowers you to tailor your streaming architecture to your exact needs—securely, modularly, and at scale.