Using Python in Connectors and SMTs

Partner Labs
laurent@confluent.io

Python Source Connector

```python
def poll(offsets):
    return [{
        "key": 123,
        "value": "Hello, World!"
    }]
```

- The **poll** method is called repeatedly by the Connect framework
- Simply return one or more records as **dicts**
- **Basic + structured** (w/ or w/o schema) **types**
- Offset management to record state between calls

```
{
  "name": "pytools-test-2",
  "config": {
    "connector.class": "io.confluent.pytools.PySourceConnector",

    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",

    "kafka.topic": "test-topic-1",

    "entry.point": "end2end_2.poll",
    "init.method": "init",
    "working.dir": "/tmp/",
    "scripts.dir": "/app/"
  }
}
```

Simple setup:

- Just put your python files into the VM and point to it
- Python executable can be selected
- Creates a **virtual environment**
- Configured like any connector (through properties)

Python SMT

```python
def transform(record):
    # only modify the value
    record['value'] = f"Modified from python --> {record['value']}"
    return record
```

- Access and **modify both key and value**
- Possibility to **filter out** (drop) **messages** by returning None
- **Basic + structured** (w/ or w/o schema) **types**

```
{
  "name": "python-smt",
  "config": {
    .../...
    "transforms": "pyTransform",
    "transforms.pyTransform.type": "io.confluent.pytools.PyConnectSmt",
    "transforms.pyTransform.entry.point": "transform1.transform",
    "transforms.pyTransform.init.method": "init",
    "transforms.pyTransform.working.dir": "/tmp/",
    "transforms.pyTransform.scripts.dir": "/app/"
  }
}
```

Simple setup:

- Just put your python files into the VM and point to it
- Python executable can be selected
- Creates a **virtual environment**
- Configured like any other SMT (as part of the connector properties)

Behind the Scenes

- Confluent PyTools classes use Pemja
- Pemja is developed by Alibaba → https://github.com/alibaba/pemja

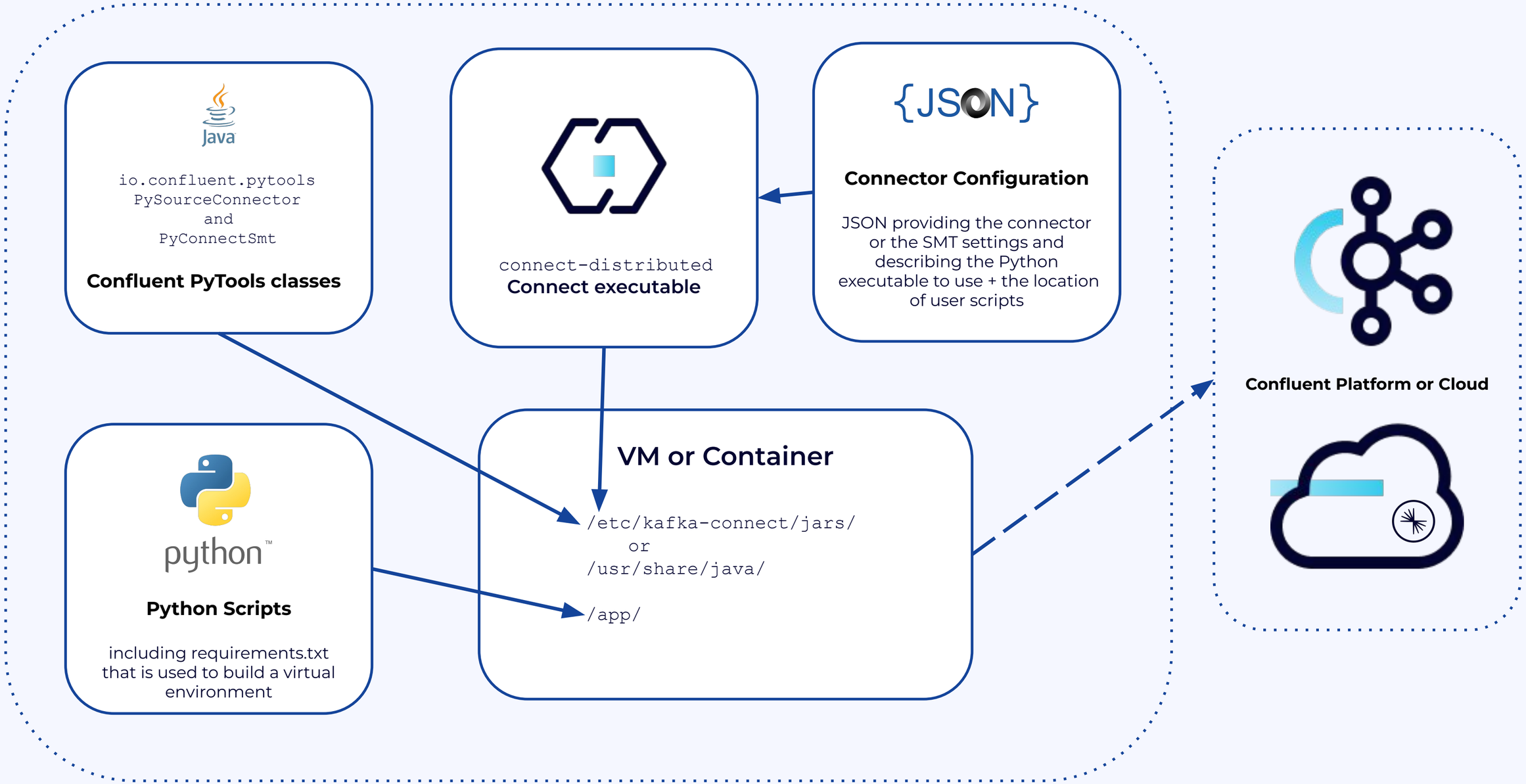
How it works:

- Starts a thread with the main interpreter in the JVM
- Calls python code using Java to C interface (JNI) and a C python module
- Requires a (pemja) library on both sides

Stress tests showed no memory or performance issues.

The virtual environment takes from 6 to 10 seconds to be built. It's done only once at startup.

Typical Deployment




CONFLUENT