

Using script to integrate custom code

Date published: 2022-09-01

Date modified: 2024-07-30



Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Initial setup.....4

 Enabling Python scripting for MiNiFi version 1.23.02 and higher..... 4

 Enabling Python scripting for MiNiFi versions 1.22.08 and 1.22.10.....5

Using the ExecuteScript processor..... 5

Using a Python processor..... 6

Initial setup

Scripting allows you to integrate custom code into MiNiFi C++ agents using Python. You can use either the ExecuteScript processor or custom Python processors.



Note: To use this feature, you need MiNiFi C++ Agent 1.22.08 or higher versions.

Learn how to install and enable the required Python version on all target systems.

Enabling Python scripting for MiNiFi version 1.23.02 and higher

On Linux

- To use the Python processors, copy libminifi-python-script-extension.so located in the nifi-minifi-cpp-...-extra-extensions-centos-bXX.tar.gz archive to the extensions/ folder.
- To use the ExecuteScript processor with Python, copy libminifi-python-script-extension.so and libminifi-script-extension.so located in the nifi-minifi-cpp-...-extra-extensions-centos-bXX.tar.gz archive to the extensions/ folder.

Requirements

Python scripting extension needs the generic Python3 library (libpython3.so) with a minimum version 3.6.

Anaconda

Before starting MiNiFi, set the LD_LIBRARY_PATH environment variable to the lib folder of the installed Python.

```
export LD_LIBRARY_PATH="${CONDA_PREFIX}/lib"
```

PyEnv

Before starting MiNiFi, set the LD_LIBRARY_PATH environment variable to the lib folder of the installed Python.

```
export LD_LIBRARY_PATH="${PYENV_ROOT}/versions/${PY_VERSION}/lib"
```

RHEL/CentOS

```
yum install python3-libs
```

Debian/Ubuntu

```
apt install libpython3-dev
```

Debian/Ubuntu does not provide the generic Python3 library (libpython3.so), but the extension works with the specific libraries as well. To use the extension on a system where the generic libpython3.so is not available, patch the extension to use the specific library.

```
patchelf extensions/libminifi-python-script-extension.so --replace-needed libpython3.so libpython3.9.so
```

On Windows

The Python extension is part of the normal MiNiFi C++ MSI installer, but it is not enabled by default. You need to enable it during installation if you want to use it.

Requirements

Python scripting extension needs the generic Python3 library (python3.dll) with a minimum version 3.6.

Install Python through the GUI installer on <https://www.python.org/downloads/windows>, or through winget.

```
winget install -e --id Python.Python.3.11
```

What to do next

- [Using the ExecuteScript processor](#)
- [Using a Python processor](#)

Enabling Python scripting for MiNiFi versions 1.22.08 and 1.22.10

To use scripting, you need to install the required Python version on all target systems.

On Linux

Python 3.6 is required, which is available on CentOS 7.

If you are downloading MiNiFi C++ for Linux:

1. Find the `nifi-minifi-cpp-...-extra-extensions-centos-bXX.tar.gz` file. This file contains the `libminifi-script-extensions.so` file.
2. Copy the `libminifi-script-extensions.so` file to the `extensions/` directory so that the MiNiFi C++ agent can load it on startup.

There is an additional workaround required to make scripting work if you are using MiNiFi version 1.22.08. You need to patch the MiNiFi binary to link to Python: `patchelf --add-needed libpython3.6m.so MINIFI_HOME/bin/minifi`



Note: This step is unnecessary if you use version 1.22.10 or higher.

On Windows

Python 3.10 and the 64 bit version of the agent are required. The Python extension is already part of the normal MiNiFi C++ MSI installer, but it is not enabled by default. You need to enable it during installation.

Before version 1.23.02, the script-extension was tightly coupled with Python and Lua, so you also need to install the Lua library on all target systems.

Using the ExecuteScript processor

The ExecuteScript processor runs an external stateless script on each processor run, allowing simpler integration. Learn how to use it to integrate custom code into a MiNiFi C++ agent.

When using the ExecuteScript processor, you need to add a Python script on the agents' file systems, and point the ExecuteScript processor to use that script. For more information on how you can send files to agents to be used on the agents, see *Using Asset Push command*. On each execution, the Python script is evaluated, and its `onTrigger` function is called to receive any incoming flow files and to produce the output.

This is an example script that reverses the content of flow files:

```
#!/usr/bin/env python
import codecs
import time

class ReadCallback:
    def process(self, input_stream):
        self.content = codecs.getreader('utf-8')(input_stream).read()
        return len(self.content)

class WriteReverseStringCallback:
    def __init__(self, content):
        self.content = content

    def process(self, output_stream):
        reversed_content = self.content[::-1]
        output_stream.write(reversed_content.encode('utf-8'))
        return len(reversed_content)

def onTrigger(context, session):
    flow_file = session.get()
    if flow_file is not None:
        read_callback = ReadCallback()
        session.read(flow_file, read_callback)
        session.write(flow_file, WriteReverseStringCallback(read_callback.content))
        flow_file.addAttribute('python_timestamp', str(int(time.time())))
        session.transfer(flow_file, REL_SUCCESS)
```

Related Information

[Using Asset Push command](#)

Using a Python processor

Python processors in MiNiFi C++ are loaded from external files, and they keep running a function, while retaining the interpreter state. This makes them well-suited for tasks that require maintaining state between runs or executing initialization logic to be run only once. Learn how to integrate custom code seamlessly into a MiNiFi C++ agent by utilizing custom Python processors. You can unlock the full potential of your MiNiFi C++ agent by enhancing its functionality with the flexibility offered by Python processors.

The workflow of a Python processor:

1. At startup, the MiNiFi C++ agent reads the Python script directory specified in the `minifi.properties` file as the value of the `nifi.python.processor.dir` property. By default, this directory is set to `MINIFI_HOME/minifi-python`.
2. The agent scans the directory for compatible scripts and automatically registers them for use.
3. Python files are evaluated during startup. Their `onSchedule` function is invoked before starting a flow and their `onTrigger` function is called regularly as the processor is scheduled.

For more details, see the following example:

```
#!/usr/bin/env python
def describe(processor):
    processor.setDescription("Adds an attribute to your flow files")
```

```
def onInitialize(processor):  
    processor.setSupportsDynamicProperties()  
  
def onTrigger(context, session):  
    flow_file = session.get()  
    if flow_file is not None:  
        flow_file.addAttribute("Python attribute", "attributevalue")  
        session.transfer(flow_file, REL_SUCCESS)
```

To add or update processors, place the Python files in the designated script directory and restart the agent.