

# **Process Scripts**

Cisco IOS XR process scripts are also called daemon scripts. The process scripts are persistent scripts that continue to run as long as you have activated the scripts. An IOS XR process, Application manager (AppMgr or app manager), manages the lifecycle of process scripts. The scripts are registered as an application on the app manager. This application represents the instance of the script that is running on the router.

The app manager is used to:

- Start, stop, monitor, or retrieve the operational status of the script.
- Maintain the startup dependencies between the processes.
- Restart the process if the script terminates unexpectedly based on the configured restart policy.

Process scripts support Python 3.5 programming language. For the list of supported packages, see Cisco IOS XR Python Packages.

This chapter gets you started with provisioning your Python automation scripts on the router.



Note

This chapter does not delve into creating Python scripts, but assumes that you have basic understanding of Python programming language. This section will walk you through the process involved in deploying and using the scripts on the router. A process script refers to code that runs continuously or endlessly.

- Workflow to Run Process Scripts, on page 1
- Managing Actions on Process Script, on page 10
- Example: Check CPU Utilization at Regular Intervals Using Process Script, on page 11

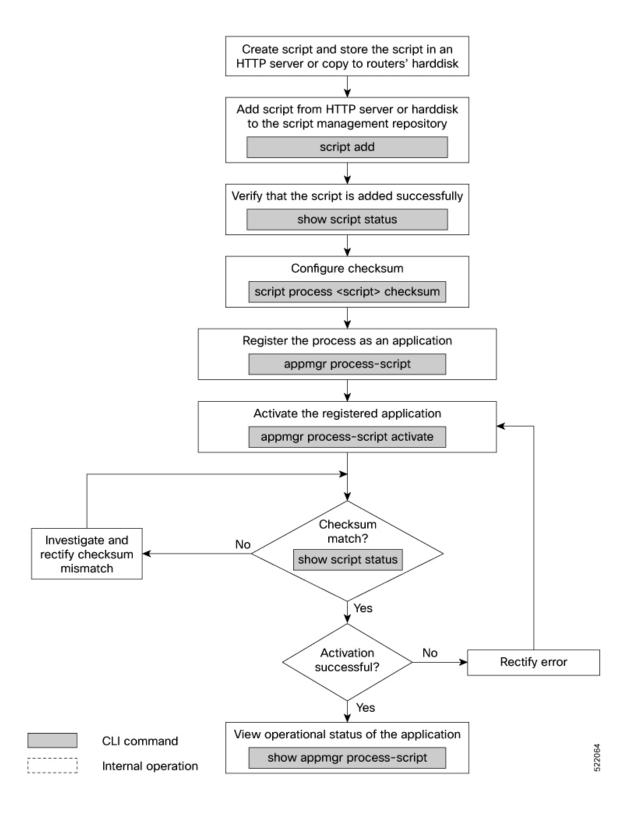
# **Workflow to Run Process Scripts**

Complete the following tasks to provision process scripts:

- Download the script—Store the script on an external server or copy to the harddisk of the router. Add the script from the external server or harddisk to the script management repository on the router using the **script add process** command.
- Configure the checksum—Check script integrity and authenticity using the script process <script.py> checksum command

- Register the script—Register the script as an application in the app manager using **appmgr process-script** command.
- Activate the script—Activate the registered application using **appmgr process-script activate** command.
- View the script execution details—Retrieve the operational data using the **show appmgr process-script** command.

The following image shows the workflow diagram representing the steps that are involved in using a process script:



# **Download the Script to the Router**

To manage the scripts, you must add the scripts to the script management repository on the router. A subdirectory is created for each script type. By default, this repository stores the downloaded scripts in the appropriate subdirectory based on script type.

Script Type Download Location					
config	harddisk:/mirror/script-mgmt/config				
exec	harddisk:/mirror/script-mgmt/exec				
process	harddisk:/mirror/script-mgmt/process				
eem	harddisk:/mirror/script-mgmt/eem				

The scripts are added to the script management repository using two methods:

- Method 1: Add script from a server
- Method 2: Copy script from external repository to harddisk using scp or copy command

In this section, you learn how to add process-script.py script to the script management repository.

#### **Procedure**

#### **Step 1** Add the script to the script management repository on the router using one of the two options:

#### Add Script From a Server

Add the script from any server or the harddisk location in the router.

```
Router#script add process <script-location> <script.py>
```

The following example shows a process script process-script.py downloaded from an external repository http://192.0.2.0/scripts:

```
Router#script add process http://192.0.2.0/scripts process-script.py Fri Aug 20 05:03:40.791 UTC process-script.py has been added to the script repository
```

The script add process supports the HTTP, HTTPS, FTP, TFTP, and SCP protocols for copying a script.

You can add a maximum of 10 scripts simultaneously.

```
Router#script add process <script-location> <script1.py> <script2.py> ... <script10.py>
```

You can also specify the checksum value while downloading the script. This value ensures that the file being copied is genuine. You can fetch the checksum of the script from the server from where you are downloading the script. However, specifying checksum while downloading the script is optional.

Router#script add process http://192.0.2.0/scripts process-script.py checksum SHA256 <checksum-value>

For multiple scripts, use the following syntax to specify the checksum:

Router#script add process http://192.0.2.0/scripts <script1.py> <script1-checksum> <script2-checksum>

```
... <script10.py> <script10-checksum>
```

If you specify the checksum for one script, you must specify the checksum for all the scripts that you download.

#### Note

Only SHA256 checksum is supported.

## · Copy the Script from an External Repository

You can copy the script from the external repository to the routers' harddisk and then add the script to the script management repository.

**a.** Copy the script from a remote location to harddisk using scp or copy command.

```
Router#scp userx@192.0.2.0:/scripts/process-script.py /harddisk:/
```

**b.** Add the script from the harddisk to the script management repository.

```
Router#script add process /harddisk:/ process-script.py
Fri Aug 20 05:03:40.791 UTC
process-script.py has been added to the script repository
```

**Step 2** Verify that the scripts are downloaded to the script management repository on the router.

#### **Example:**

Script process-script.py is copied to harddisk:/mirror/script-mgmt/process directory on the router.

# **Configure Checksum for Process Script**

Every script is associated with a checksum hash value. This value ensures the integrity of the script, and that the script is not tampered. The checksum is a string of numbers and letters that acts as a fingerprint for script. The checksum of the script is compared with the configured checksum. If the values do not match, the script is not run and a warning message is displayed.

It is mandatory to configure the checksum to run the script.



Note

Process scripts support the SHA256 checksum hash.

# Before you begin

Ensure that the script is added to the script management repository. See Download the Script to the Router.

#### **Procedure**

**Step 1** Retrieve the SHA256 checksum hash value for the script from the IOS XR Linux bash shell.

# **Example:**

Router#run

[node0\_RP0\_CPU0:~]\$sha256sum /harddisk:/mirror/script-mgmt/process/process-script.py
94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
/harddisk:/mirror/script-mgmt/process/process-script.py

Make note of the checksum value.

## **Step 2** View the status of the script.

#### Example:

Router#show script status detail

Fri Aug 20 05:04:13.539 UTC

Name | Type | Status | Last Action | Action Time

process-script.py | process | Config Checksum | NEW | Fri Aug 20 05:03:41 2021

Script Name : process-script.py
History:
----
1. Action : NEW
Time : Fri Aug 20 05:03:41 2021
Description : User action IN\_CLOSE\_WRITE

The status shows that the checksum is not configured.

# **Step 3** Configure the checksum.

## Example:

Router#configure

Router(config) #script process process-script.py checksum SHA256 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b

Router(config)#commit
Tue Aug 20 05:10:10.546 UTC
Router(config)#end

## **Step 4** Verify the status of the script.

#### **Example:**

#### Router#show script status detail

Fri Aug 20 05:15:17.296 UTC

The status Ready indicates that the checksum is configured and the script is ready to be run. When the script is run, the checksum value is recalculated to check if it matches with the configured hash value. If the values differ, the script fails. It is mandatory for the checksum values to match for the script to run.

# **Register the Process Script as an Application**

Register the process script with the app manager to enable the script. The registration is mandatory for using process script on the router.

## Before you begin

Ensure that the following prerequisites are met before you register the script:

- Download the Script to the Router
- Configure Checksum for Process Script, on page 5

#### **Procedure**

**Step 1** Register the script with an application (instance) name in the app manager.

#### Example:

```
Router#configure
Fri Aug 20 06:10:19.284 UTC
Router(config)#appmgr process-script my-process-app
Router(config-process)#executable process-script.py
```

Here, my-process-app is the application for the executable process-script.py script.

**Step 2** Provide the arguments for the script.

#### **Example:**

```
Router(config-process) #run-args --host <host-name> --runtime 3 --log script
```

**Step 3** Set a restart policy for the script if there is an error.

#### Example:

```
Router(config-process) #restart on-failure max-retries 3
Router(config-process) #commit
```

Here, the maximum attempts to restart the script is set to 3. After 3 attempts, the script stops.

You can set more options to restart the process:

Keyword	Description
always	Always restart automatically. If the process exits, a scheduler queues the script and restarts the script.
	Note This is the default restart policy.

Keyword	Description
never	Never restart automatically. If the process exits, the script is not rerun unless you provide an action command to invoke the process.
on-failure	Restart on failure automatically. If the script exits successfully, the script is not scheduled again.
unless-errored	Restart script automatically unless errored.
unless-stopped	Restart script automatically unless stopped by the user using an action command.

# **Step 4** View the status of the registered script.

# **Example:**

Router#show appmgr process-script-table

Fri Aug 20 06:15:44.244 UTC

Name	Executable	Activated	Status	Restart Policy	Config Pending
my-process-app	process-script.py	No	Not Started	On Failure	No

The script is registered but is not active.

# **Activate the Process Script**

Activate the process script that you registered with the app manager.

# Before you begin

Ensure that the following prerequisites are met before you run the script:

- Download the Script to the Router
- Configure Checksum for Process Script, on page 5
- Register the Process Script as an Application, on page 7

# **Procedure**

# **Step 1** Activate the process script.

# **Example:**

```
Router#appmgr process-script activate name my-process-app Fri Aug 20 06:20:55.006 UTC
```

The instance my-process-app is activated for the process script.

# **Step 2** View the status of the activated script.

# **Example:**

```
Router#show appmgr process-script-table Fri Aug 20 06:22:03.201 UTC
```

Name	Executable	Activated	Status	Restart Policy	Config Pending
my-process-app	process-script.py	Yes	Running	On Failure	No

The process script is activated and running.

#### Note

You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes are pending. The status of the modification is indicated in the Config Pending option. In the example, value No indicates that there are no configuration changes that must be activated.

# **Obtain Operational Data and Logs**

Retrieve the operational data and logs of the script.

# Before you begin

Ensure that the following prerequisites are met before you obtain the operational data:

- Download the Script to the Router
- Configure Checksum for Process Script, on page 5
- Register the Process Script as an Application, on page 7
- Activate the Process Script, on page 8

#### **Procedure**

**Step 1** View the registration information, pending configuration, execution information, and run time of the process script.

# **Example:**

```
Router#show appmgr process-script my-process-app info
Fri Aug 20 06:20:21.947 UTC
Application: my-process-app
  Registration info:
   Executable
                                 : process-script.pv
                                 : --host <host-name> --runtime 3 --log script
   Run arguments
                                 : On Failure
   Restart policy
   Maximum restarts
  Pending Configuration:
                                 : --host <host-name> --runtime 3 --log script
   Run arguments
   Restart policy
                                 : Always
  Execution info and status:
   Activated
                                 : Yes
   Status
                                 : Running
   Executable Checksum
                                 : 94336f3997521d6e1aec0ee6faab0233562d53d4de7b0092e80b53caed58414b
   Last started time
                                : Fri Aug 20 06:20:21.947
   Restarts since last activate : 0/3
```

```
Log location :
/harddisk:/mirror/script-mgmt/logs/process-script.py_process_my-process-app
Last exit code : 1
```

**Step 2** View the logs for the process scripts. App manager shows the logs for errors and output.

## **Example:**

The following example shows the output logs:

```
Router#show appmgr process-script my-process-app logs output

Fri Aug 20 06:25:20.912 UTC

[2021-08-20 06:20:55,609] INFO [sample-process]:: Beginning execution of process..

[2021-08-20 06:20:55,609] INFO [sample-process]:: Connecting to host '<host-name>'

[2021-08-20 06:20:56,610] INFO [sample-process]:: Reading database..

[2021-08-20 06:20:58,609] INFO [sample-process]:: Listening for requests..
```

The following example shows the error logs with errors:

```
Router#show appmgr process-script my-process-app logs errors
Fri Aug 20 06:30:20.912 UTC
-----Run ID:1632914459 Fri Aug 20 06:30:20 2021-----
Traceback (most recent call last):
 File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 121, in <module>
   main(args)
 File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 97, in main
   printer()
 File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 37, in wrapper
   result = func(*args, **kwargs)
 File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 88, in printer
   time.sleep(1)
 File "/harddisk:/mirror/script-mgmt/process/process-script.py", line 30, in handle timeout
   raise TimeoutError(error_message)
 main .TimeoutError: Timer expired
_____Run ID:1632914460 Fri Aug 20 06:31:03 2021-----
```

This example shows the log without errors:

```
Router#show appmgr process-script my-process-app logs errors
```

# **Managing Actions on Process Script**

The process script runs as a daemon continuously. You can, however, perform the following actions on the process script and its application:

Table 1: Feature History Table

Action	Description		
Deactivate	Clears all the resources that the application uses.		
	Router#appmgr process-script deactivate name my-process-app		
	You can modify the script while the script is running. However, for the changes to take effect, you must deactivate and activate the script again. Until then, the configuration changes do not take effect.		
Kill	Terminates the script if the option to stop the script is unresponsive.		
	Router#appmgr process-script kill name my-process-app		
Restart	Restarts the process script.		
	Router#appmgr process-script restart name my-process-app		
Start	Starts an application that is already registered and activated with the app manager.		
	Router#appmgr process-script start name my-process-app		
Stop	Stops an application that is already registered, activated, and is currently running. Only the application is stopped; resources that the application uses is not cleared.		
	Router#appmgr process-script stop name my-process-app		

# Example: Check CPU Utilization at Regular Intervals Using Process Script

In this example, you use the process script to check CPU utilization at regular intervals. The script does the following actions:

- Monitor the CPU threshold value.
- If the threshold value equals or exceeds the value passed as argument to the script, log an error message that the threshold value has exceeded.

# Before you begin

Ensure you have completed the following prerequisites before you register and activate the script:

1. Create a process script cpu-utilization-process.py. Store the script on an external server or copy the script to the harddisk of the router.

```
import time
import os
import xmltodict
import re
import argparse

from cisco.script_mgmt import xrlog
from iosxr.netconf.netconf_lib import NetconfClient
log = xrlog.getScriptLogger('Sample')
```

```
syslog = xrlog.getSysLogger('Sample')
def cpu memory check(threshold):
   Check total routes in router
   filter_string = """
   <system-monitoring xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-wdsysmon-fd-oper">
      <cpu-utilization>
        <node-name>0/RP0/CPU0</node-name>
         <total-cpu-one-minute/>
     </cpu-utilization>
   </system-monitoring>"""
   nc = NetconfClient(debug=True)
   nc.connect()
   do_get(nc, filter=filter string)
   ret_dict = _xml_to_dict(nc.reply, 'system-monitoring')
    total_cpu =
int(ret dict['system-monitoring']['cpu-utilization']['total-cpu-one-minute'])
   if total cpu >= threshold:
       syslog.error("CPU utilization is %s, threshold value is %s"
%(str(total_cpu),str(threshold)))
    nc.close()
def xml to dict(xml output, xml tag=None):
   convert netconf rpc request to dict
   :param xml output:
    :return:
   if xml tag:
       pattern = "<data>\s+(<%s.*</%s>).*</data>" % (xml tag, xml tag)
        pattern = "(<data>.*</data>)"
   xml output = xml output.replace('\n', '')
   xml data match = re.search(pattern, xml output)
    ret_dict = xmltodict.parse(xml_data_match.group(1))
   return ret_dict
def do get(nc, filter=None, path=None):
   try:
        if path is not None:
           nc.rpc.get(file=path)
        elif filter is not None:
           nc.rpc.get(request=filter)
           return False
    except Exception as e:
       return False
    return True
if name == ' main ':
   parser = argparse.ArgumentParser()
   parser.add argument("threshold", help="cpu utilization threshold", type=int)
   args = parser.parse args()
   threshold = args.threshold
   while (1):
        cpu memory check (threshold)
        time.sleep(30)
```

Configure the script with the desired threshold criteria. This default threshold is configured to alert when CPU utilization exceeds this value. The script checks the CPU utilization every 30 seconds.

- **2.** Add the script from the external server or harddisk to the script management repository. See Download the Script to the Router.
- **3.** Configure the checksum to verify the authenticity and integrity of the script. See Configure Checksum for Process Script, on page 5.

## **Procedure**

**Step 1** Register the process script cpu-utilization-process.py with an instance name my-process-app in the app manager.

#### **Example:**

```
Router(config) #appmgr process-script my-process-app
Router(config-process) #executable cpu-utilization-process.py
Router(config-process) #run-args <threshold-value>
```

**Step 2** Activate the registered application.

#### **Example:**

Router(config-process) #appmgr process-script activate name my-process-app

**Step 3** Check the script status.

#### Example:

```
Router#show appmgr process-script-table
```

Thu Sep 30 18:15:03.201 UTC

Name	Executable	Activated	Status	Restart Policy	Config Pending
mv-process-app	cpu-utilization-process.pv	Yes	Running	On Failure	No

**Step 4** View the log.

## Example:

```
Router#show appmgr process-script my-process-app logs errors
RP/0/RP0/CPU0:Sep 30 18:03:54.391 UTC: python3_xr[68378]: %OS-SCRIPT_MGMT-3-ERROR:
Script-test process: CPU utilization is 6, threshold value is 5
```

An error message is displayed that the CPU utilization has exceeded the configured threshold value, and helps you take corrective actions.

**Example: Check CPU Utilization at Regular Intervals Using Process Script**