Config scripts (config_script) – expected interface

This document explains how **config scripts** (the value of the config_script key in a node template YML) are called by EVE-NG, which parameters the API passes, and what behaviour is expected. It is intended for anyone writing or adapting a config script.

What is a config script?

- A **config script** is an **executable file** (script or binary) located in:
  - /opt/unetlab/config_scripts/
- The template YML field config_script contains **only the filename** (for example config_ceos.py, config_srlinux.py, config_timos.py).
- The script is used by the API to:
  - **export** a running node configuration to a file (action get);
  - **import/push** a configuration file into a node (action put).

There is **no constraint on the implementation language**:

- You can use **Python, Bash, Go, C, Rust, etc.**
- The only requirements are:
  - the file must be **executable** (chmod 755 /opt/unetlab/config_scripts/* is run by system utilities);
  - it must understand the **CLI interface** described below and follow the expected behaviour.

How the API calls config scripts

EVE-NG calls the script with a fixed set of options. Internally this is done from the PHP/Go API, but from the script's point of view it is simply a process invocation like:

```
/opt/unetlab/config_scripts/<config_script> -a <action> [other options...]
```

There are **two actions**:

- -a get – **export** configuration from a running node to a file.
- -a put – **import** configuration from a file into a running node.

The common options used by the platform are:

Option	Also as long form	Type	Used with	Meaning
-a	--action	string	get / put	Action: get or put.
-f	--file	path	get / put	File path for the configuration on the **EVE-NG host**. For get, the script must **create** this file. For put, the script must **read** this file and push it to the node.
-t	--timeout	integer	get / put	Maximum allowed runtime in **seconds**. The script should honour this as an upper bound and exit if exceeded.

Additional options depend on the node type and transports used:

Port-based scripts (console TCP port)

Most QEMU/IOL/vm-like devices use a console TCP port:

Option	Also as long form	Type	Actions	Meaning
-p	--port	integer	get / put	Console TCP port to connect to (for example, telnet/SSH/serial proxy). The script is responsible for opening the session to this port.

Typical calls:

```
# Export config
/opt/unetlab/config_scripts/config_timos.py -a get -p <port> -f /tmp/export.cfg -t 300

# Import config
/opt/unetlab/config_scripts/config_timos.py -a put -p <port> -f /opt/unetlab/tmp/.../startup-config -t 600
```

Docker-based scripts (container ID / UUID)

For Docker templates, the script may get a **container identifier** instead of a port:

Option	Also as long form	Type	Actions	Meaning
-i	--id	string	usually put (sometimes get)	Container identifier, typically the Docker name or UUID used by EVE-NG. The script uses this to run docker exec, copy files, or open a shell.

Examples from existing scripts:

- config_xrd.py, config_srlinux.py, config_docker.py expect -i <docker_id> for put (and sometimes get).

Remote satellite option (distributed setups)

When a node runs on a **remote satellite**, the same config script is re-used; EVE-NG adds a satellite hint:

Option	Also as long form	Type	Actions	Meaning
-s	--satellite	string	get / put (where supported)	Satellite IP or host (for example 172.29.130.5). The script may use this to talk to Docker or to the device via ssh -H / docker -H ssh://root@<satellite>.

Scripts that support remote satellites treat -s as **optional**: it may be empty or absent when the node runs locally.

Expected behaviour (get vs put)

Action get – export configuration from node to file

When called with -a get, the platform expects the script to:

1. Connect to the node (via console port or container ID, depending on options).
2. Retrieve the **running / startup configuration** in the format appropriate for that device (text, XML, etc.).
3. **Write the configuration to the file path** given by -f on the EVE-NG host.
4. Exit with:
    - **code 0** on success (file exists and is non-empty);
    - **non-zero** on failure (any error condition).

Additional expectations commonly enforced by existing scripts:

- -p must be a valid non-negative port for get if it is used.
- The script must fail if the destination file already exists (to avoid overwriting by accident).
- If the timeout is exceeded, the script should stop its operation and exit with an error.

Action put – import configuration from file to node

When called with -a put, the platform expects the script to:

1. **Read** the configuration from the file path given by -f (EVE-NG created the file, usually startup-config in the node's run directory).
2. Connect to the node (console port or container ID).
3. Push/apply the configuration using the device's native mechanism (CLI, API, copy from file, etc.).
4. Exit with **code 0** when the configuration is successfully applied or queued.

Additional expectations:

- The source file must exist; if not, the script should exit with an error.
- The script should obey the -t timeout and avoid hanging indefinitely.
- For Docker-based scripts using -i, the script should locate the container and perform the necessary docker exec / copy operations.

Execution environment and working directory

Config scripts are executed on the **EVE-NG host**, not inside the guest:

- They run as a normal process on the Linux host (usually as root), so they can:
  - open TCP connections to the node's console port;
  - run docker commands;
  - read/write files under /opt/unetlab/tmp/....
- The **current working directory** is usually set to the node's **running directory** (for Docker export/import) or a temporary directory, depending on the caller. Scripts should **not rely on a specific CWD**; always use absolute paths when possible or derive paths from -f.\n Config scripts can invoke **any other tools** available on the host (telnet, ssh, docker, etc.) as needed.

Language and implementation freedom

There is **no restriction on language**:

- Existing scripts are written in **Python 3**, but this is not mandatory.
- You may implement a config script as:
  - a **shell script** (#!/bin/bash),
  - a **compiled binary** (Go, C, Rust, ...),
  - any other interpreter available on the system.

Requirements:

- The file must be placed in /opt/unetlab/config_scripts/.
- It must be **executable** (chmod 755).
- It must accept the options described above (-a, -f, -t, and optionally -p, -i, -s) and follow the get / put contract.

Minimal example (pseudo-code)

The following pseudo-code shows the minimal logic in a config script:

```
#!/bin/bash

ACTION=\"\"
FILE=\"\"
PORT=\"\"
TIMEOUT=300

while getopts \"a:p:t:f:\" opt; do
  case \"$opt\" in
    a) ACTION=\"$OPTARG\" ;;
    p) PORT=\"$OPTARG\" ;;
    t) TIMEOUT=\"$OPTARG\" ;;
    f) FILE=\"$OPTARG\" ;;
    *) echo \"Usage: $0 -a get|put -p <port> -f <file> -t <timeout>\" >&2; exit 3 ;;
  esac
done

if [ -z \"$ACTION\" ] || [ -z \"$FILE\" ]; then
  echo \"ERROR: missing mandatory parameters\" >&2
  exit 1
fi

case \"$ACTION\" in
  get)
    # 1. Connect to node using $PORT
    # 2. Dump running config into \"$FILE\"
    # 3. Exit 0 on success
    ;;
  put)
    # 1. Read config from \"$FILE\"
    # 2. Push config to node using $PORT
    # 3. Exit 0 on success
    ;;
  *)
    echo \"ERROR: invalid action\" >&2
    exit 1
    ;;
esac
```

You can replace the shell parts by Python, Go, or any other language as long as the observable interface and exit codes remain compatible.

Relation to config_script in template YML

- In a template YML (see NODE_TEMPLATES_YML_SYNTAX.md), the key:

- config_script: config_ceos.py

  means:

  - When the user **exports** configuration from a node of this template, EVE-NG calls:
    - /opt/unetlab/config_scripts/config_ceos.py -a get ...
  - When the user **applies / imports** a startup config, EVE-NG calls:
    - /opt/unetlab/config_scripts/config_ceos.py -a put ...
- Switching to a custom config script is as simple as:
  - placing your executable in /opt/unetlab/config_scripts/, and
  - updating config_script in the YML to its filename.

As long as your script honours the interface defined in this document, the API and UI do not care about the implementation details.