# Automating Configuration Changes with opConfig


- Introduction
- Concepts
- The Anatomy of a Config Set
    - Storage Format and Location
    - Metadata
    - Candidate Selection
    - Notifications
    - Error Handling
        - Error Detection
        - Reacting to Errors
    - Stages and their Commands
        - The pre-commands Stage (optional)
        - The post-commands and post-rollback-commands Stages (optional)
        - The commands Stage (required)
        - The rollback-commands Stage (optional)
- Config Set Management
- Scheduling of Configuration Changes
    - Time Formats
    - Selection Refinements
    - Scheduling using the CLI Tool
- Limitations in opConfig 3.0.0
- Config Set Example
- Config Set GUI
- Config Set Parameters
        - Example
    - Using a config set with parameters
        - cli Example
        - GUI Example


## Introduction

opConfig 3 introduces the ability to 'push' configuration changes to devices, complete with error detection, support for change rollback and complete audit trails.

This document describes the config push infrastructure.

## Concepts

opConfig has always supported '*command sets*', which consist of (individual) commands that can be sent to devices and whose output is collected and analyzed. Command sets are expected to be independent, safely repeatable and without side effects. Command sets don't support sequencing or error detection, and thus are not suitable (or intended!) for modifying device configurations.

For config push, opConfig requires that the administrator defines one or more suitable '*config sets*'. A config set is a list of commands which are sent to selected devices with the expectation that something on the device is changed by the sequence of commands; to perform such operations safely, a config set can also define error detection patterns and possible reactions to any errors.

## The Anatomy of a Config Set

### Storage Format and Location

The primary storage and exchange format for config sets is JSON; example config sets can be found in `install/config_sets.d/` and it is recommended that you store your actively developed config sets in `conf/config_sets.d/`.

To make a config set active you have to import it into opConfig: the JSON files themselves are *not* consulted.

### Metadata

A config set must have a unique `name` property, and may have a (free-form) `description`.

Every config set is automatically assigned a `revision` by opConfig, which cannot be changed once set. opConfig automatically tracks the history and evolution of your config sets via name and revision: whenever a config set file is (re)imported into opConfig, the config set is checked for differences against the most recent known revision; if there are any, the set is saved with a new revision.

In general, only the most recent revision of a config set will be applied to devices (but this can be overridden).

## Candidate Selection

Configuration changes and the commands to perform them are usually specific to a target platform. To avoid misapplication of config sets opConfig provides a flexible mechanism for selecting candidate devices for each configuration set. This is done by providing rules in the optional `filter` section.

The filter section holds any number of selection clauses, each consisting of a property path and a selection criterion. To pass as a candidate, *all* filters must match.

The property path is given in "dotted notation", compatible with `opnode_admin.pl`'s `act=show` and `act=set` operations.
For example, `os_info.major` would be used to refer to the detected OS major version (which is stored within the `os_info` substructure).

The selection criterion can take one of three forms:

1. an explicit single value
   for example, `"os_info.platform": "x86_64"` would match if the node's OS platform value is exactly equal to "x86_64".
2. a list of alternative values
   for example `"name": [ "nodeA", "nodeB"]` would match either of the two nodes named `nodeA` or `nodeB`. The comparison is again strict equality.
   The filter clause is considered a match if one or more choices amongst the list of alternatives do match.
   a. If you are using the GUI, just introduce a list of node names separated by commas.
3. a regular expression
   for example `"os_info.os": "/IOS/i"` would select nodes whose OS contains the string "IOS" (or "ios", "IoS" etc.) anywhere.
   Note that the regular expression must be given as a string, starting and ending with a "/" and optionally including the "i" modifier for case-insensitive matching.

All known properties for a node are available for filtering; please consult the output of `opnode_admin.pl act=show` or `act=export` for a list of common properties.

## Notifications

opConfig can optionally notify interested parties when a config set was applied to devices.

This is configured by providing the relevant contact details in the optional `notify` section of the config set. When a config set application is scheduled, *ext ra* notification recipients can be specified but the ones given in the config set are always included.

In version 3.0.0 opConfig supports notification by email only.

For example, `"notify": {"email": ["first@mycompany.com","second@othercompany.com" ]}` would have opConfig send emails to both of the given addresses.

Notifications list the candidate nodes and status information for the config set application for each, e.g. complete success, partial success or complete failure.

## Error Handling

A config set can contain a number of error detection and handling directives, contained within an `error_handling` block.

### Error Detection

Error handling requires that one or more `match` properties are given, which are applied to each command's response to determine if the command is considered unsuccessful.
By default no error detection is performed, and all commands are considered successful no matter what their output is.

Similar to the candidate filtering mechanism described above, match properties must either be strings or strings containing a regular expression. Strings are compared exactly. More than one match property is possible.

If the command output meets one or more `match` properties, then the command is considered unsuccessful.

### Reacting to Errors

If error detection is configured, three different reactions are possible:

1. Continue the config set application
   This is the default behaviour. opConfig will keep and report a tally of successful versus failed commands, but will try to apply all of them in sequence.
2. Abort the config set application
   To select this option, the property `break_on_error` must be set to `true`. In this case, the first failed command in a set causes all other commands to be skipped.
3. Apply Rollback commands
   To revert the configuration back to a desired state, you would have to provide a sequence of `rollback-commands` in your error-handling section. Rolling back would generally be combined with `break_on_error`, but opConfig does not strictly require that.

## Stages and their Commands

opConfig distinguishes between five different stages for the application of a config set. You can provide any number of commands for each of the stages; the `command` stage must contain one or more commands.

A stage definition consists of a list of command strings which are sent to the device in sequence, one line at a time, exactly as they're given.

Each command must be complete by itself, and must start and finish at a configured prompt for your device's personality. This means that in general, interaction with a command across multiple lines, responding to confirmation challenges etc. is **not** directly possible.

There are two exceptions:

- opConfig treats the 'pseudo-commands' `__leave__` and `__enter__` specially, and leaves or enters configuration mode.
  Note that there are two "_" on each side of the command. These special commands can be handy for rolling back changes, if your particular rollback command requires to be performed outside of configuration mode.
- Any command that starts with a single "_" is treated as a macro invocation, with optional arguments.
  Macros have to be defined suitably in your device personality's "phrasebook", and they differ from plain individual commands in that macros can define sequences of send-expect-prompt steps.
  This is how *limited* multi-step interaction with a device is possible.
  For example, the default Cisco phrasebook as shipped with opConfig defines the macro `reload_in`, to be used in a command like "`_reload_in 15`". This would configure a Cisco IOS device for a forced reboot in N minutes. This has to be done using a macro because setting up the `reload` operation requires confirmation. There is also a macro called `reload` which, when triggered by the command "`_reload`" would restart your device.

### The `pre-commands` Stage (optional)

Commands listed here are run first, in privileged mode and **outside** of configuration mode. Error detection is not available in this stage.

A common example command could be `reload in 15` for Cisco devices.

### The `post-commands` and `post-rollback-commands` Stages (optional)

These stages are run last, after leaving configuration mode and still in privileged mode. No error handling is available.

The "plain" post-commands stage is chosen in all cases where no rollback was performed (i.e. if error handling was disabled or if all commands succeeded)' on the other hand, if a rollback was performed, then only the post-rollback-commands are applied.

A common task for the `post-commands` stage would be a capture of the newly changed configuration (to double-check that everything has worked), or the cancelling of a future `reload` on Cisco devices.

### The `commands` Stage (required)

Commands in this stage are run in configuration mode (which implies privileged mode). Error handling *is* available, hence the sequence of commands may be aborted early.

### The `rollback-commands` Stage (optional)

If error detection is enabled and if this stage is provided, then its commands will be applied as soon as errors are detected. The rollback commands are sent while in configuration mode, and no error handling is possible.

## Config Set Management

As mentioned above, config set documents must be imported into opConfig to become active. Configuration sets cannot be deleted (to ensure a valid audit trail), but they can be superseded and optionally disabled and hidden from the GUI.

opConfig's `opconfig-cli.pl` provides full config set management, using the functions `import_configset`, `list_configset`, `export_configset` and `enable_` and `disable_configset`; As of version 3.0.0, the opConfig GUI only allows to display the newest revision for a config set.

A new revision of a config set automatically starts in "enabled" state, and scheduled operations normally use the highest enabled config set revision. To switch to an earlier revision instead you may disable the unwanted but higher numbered revision; in which case the GUI will no longer show the disabled revision, nor will scheduling of config changes pick this disabled revision.

## Scheduling of Configuration Changes

Scheduling of config set applications can be performed both from the GUI as well as using `opconfig-cli.pl`. When you schedule a change, you have to provide at least an execution time and a config set (name); optionally you may also provide refinements for device selections, or additional notification recipients. A schedule is immutable once set and can only be removed entirely.

### Time Formats

opConfig accepts all common time formats as described on the Supported Time Formats page. These include both absolute time and date formats like the ISO 8601 type "2016-05-20T14:40", as well as very handy relative formats like "`now + 45 minutes`" or "`tomorrow midnight`".

To enter relative formats when scheduling config changes from the GUI you'll have to use the Target Date/Time input field as the Date and Time picker only produces absolute outputs.

If you schedule a change for a time in the past, then the opConfig daemon will execute the operation immediately.

### Selection Refinements

The device filters and notification recipients given in a config set cannot be overruled when scheduling a config change; they can only be refined and extended.

- If you use the Refine Node Selection options in the scheduling GUI, then all candidate devices will have to match **both** filters from the config set **and** your refinement filters in order to be targetted.
  If your refinements or the config set filters are too strict and don't meet at least a single existing node, the change will not be scheduled and you will see an error message in the scheduling GUI.
- If you add Email Notification recipients when scheduling, then these recipients will be added to recipients from the config set.

### Scheduling using the CLI Tool

To perform a configuration change operation from the command line you would be using `opconfig-cli.pl act=push_configset name=<cset name>`. Refinements can be given using an explicit list of desired nodes with `nodes=nodeA,nodeB...`; extra email recipients can be given with the optional `email=a@b.c,e@f.g...` argument.

If you don't provide a target time (with `at=<timespec>`), then opconfig-cli will execute the configuration change immediately. With an `at` argument in the future, the change will be scheduled for the given date and time.

You may check the list of scheduled jobs using the command `opconfig-cli.pl act=list_queue`, and to delete a scheduled (but not yet active) job use the argument `act=remove_queued` with the job's id (shown by `act=list_queue with_ids=true`).

## Limitations in opConfig 3.0.0

As of version 3.0.0, only one-off scheduling is supported; recurring schedules are planned for a future release.

The shipped phrasebooks are not fully primed for configuration changing for device types other than Cisco IOS; more specifically most other phrasebooks do not have the necessary macros and prompts for handling configuration mode yet.

Within the opConfig GUI, config sets can only be viewed, not edited or imported/exported; A full config set management GUI is planned for the next release.

## Config Set Example

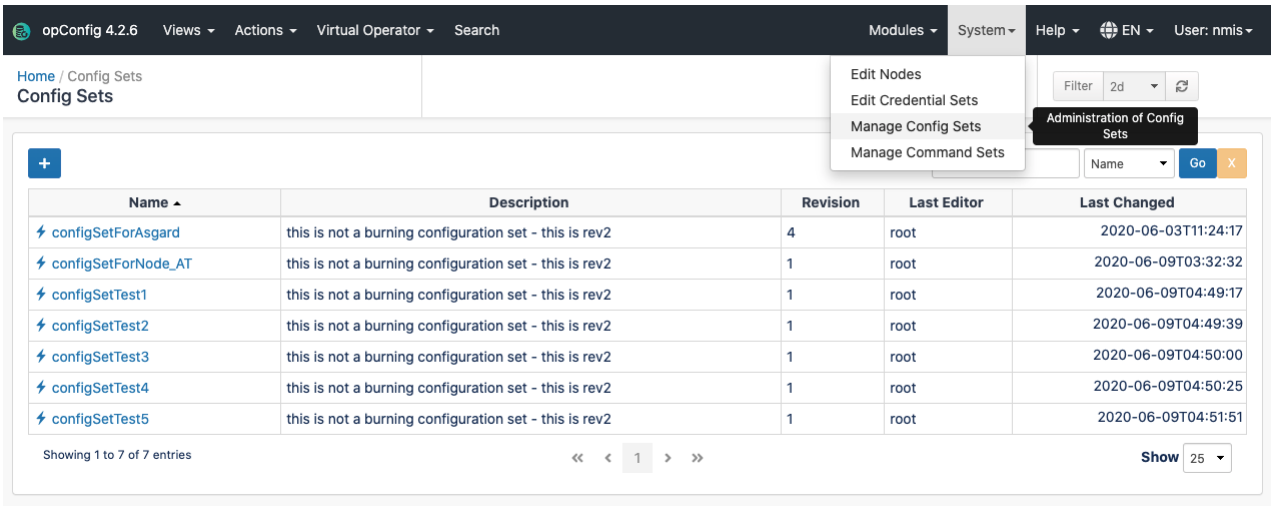
This fully functional example config set ships with opConfig 3.0.0 as `install/config_sets.d/IOS-Configuration-Best-Practices.json`.

```
{
    "name": "IOS - Configuration Best Practices",
    "description": "A configuration set to configure the IOS device, by enabling and disabling various services
and features.",
    "filter":
    {
        "os_info.os": "IOS"
    },
    "notify": { "email": "you@company.com" },
    "error_handling":
    {
        "match": [ "/Invalid input detected/" ],
        "break_on_error": false
    },
    "pre-commands":
    [
        "_reload_in 5"
    ],
    "commands":
    [
    "no ip http server",
    "no ip http secure-server",
    "no ip finger",
    "no service finger",
    "no service udp-small-servers",
    "no service tcp-small-servers",
    "no boot network",
    "no service config",
    "service password-encryption",
    "service timestamps debug datetime msec",
    "service timestamps log datetime msec",
    "service sequence-numbers",
    "service tcp-keepalives-in",
    "service tcp-keepalives-out",
    "no ip source-route",
    "line con 0 ",
    "exec-timeout 10 0",
    "exit",
    "line vty 0 4",
    "exec-timeout 10 0",
    "exit"
    ],
    "post-commands":
    [
        "reload cancel"
    ]
}
```

# Config Set GUI

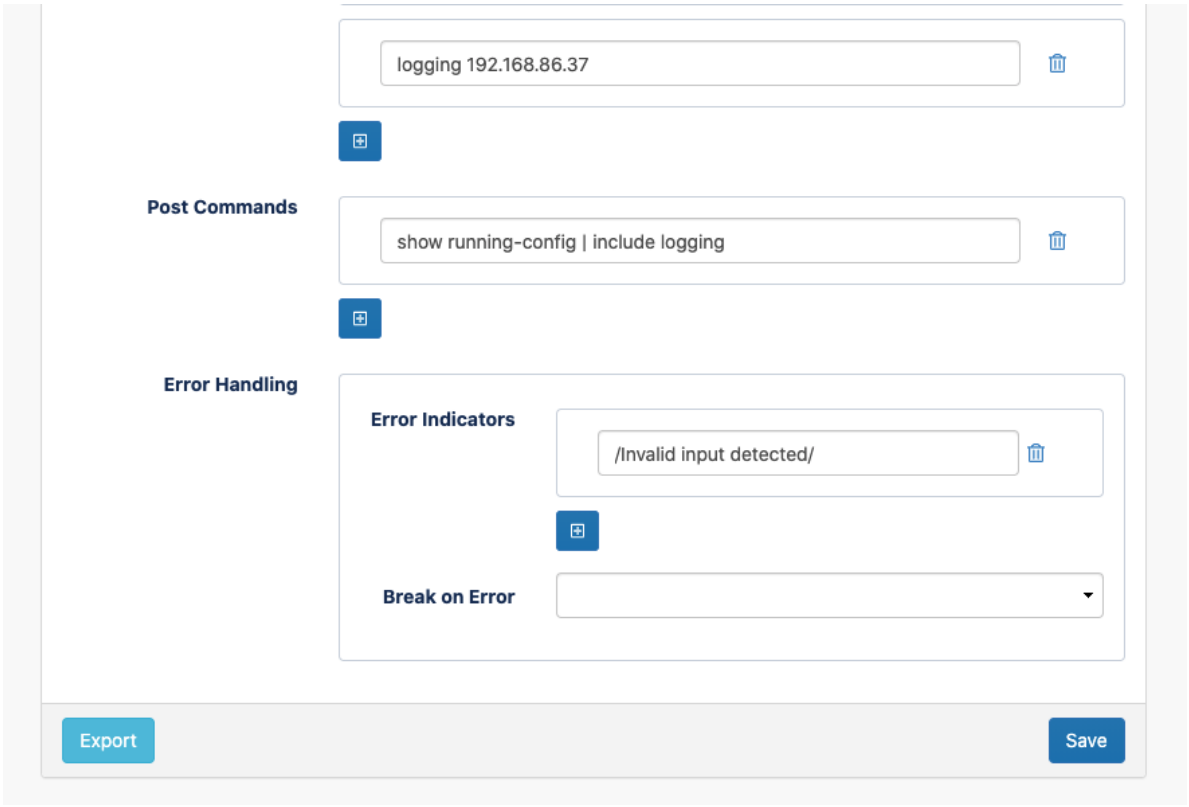
From opConfig 4.2.6 it is possible to edit the config sets in the GUI.

In the menu *System* > *Manage Config Sets* you can access the editor:




Please note, once a config set is saved, a new version will be created in the database.

It is not possible to delete a config set, but you can export one in json format:

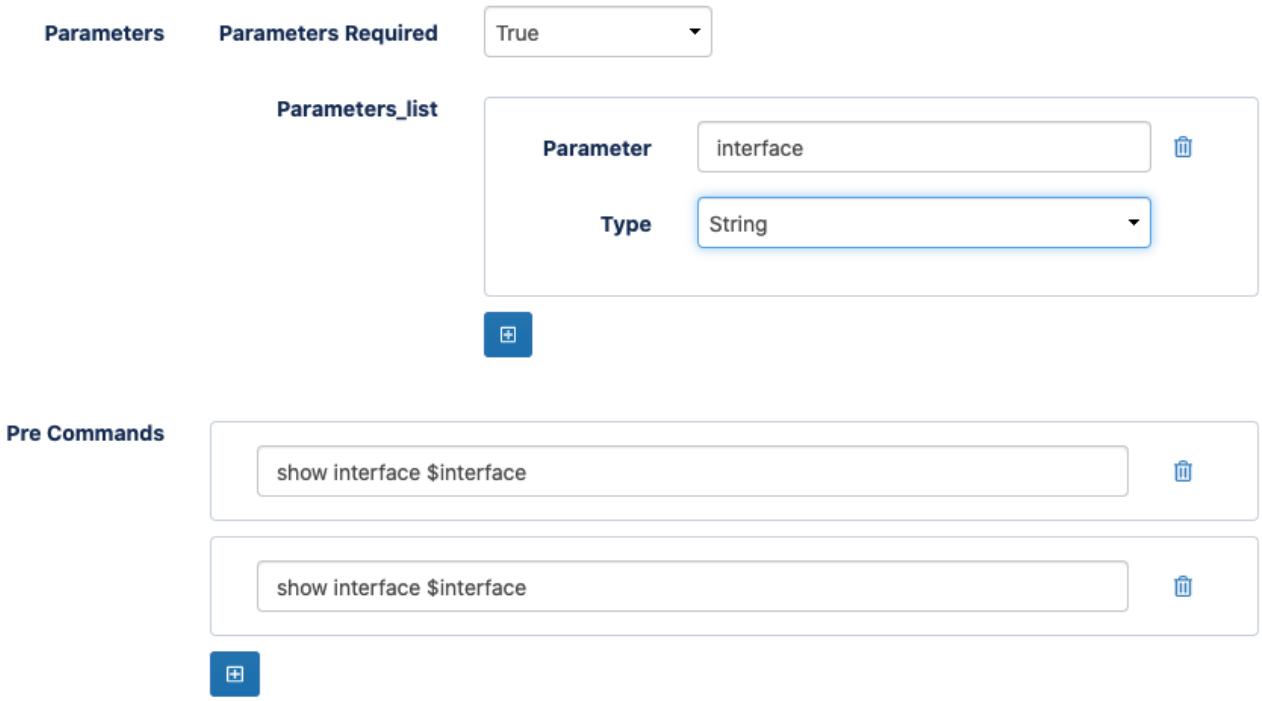



# Config Set Parameters

From opConfig 4.2.6 it is possible to use parameters in a config set.

The parameters are global to commands, pre commands or post commands, and are specified with the parameter name preceded by $

### Example




## Using a config set with parameters

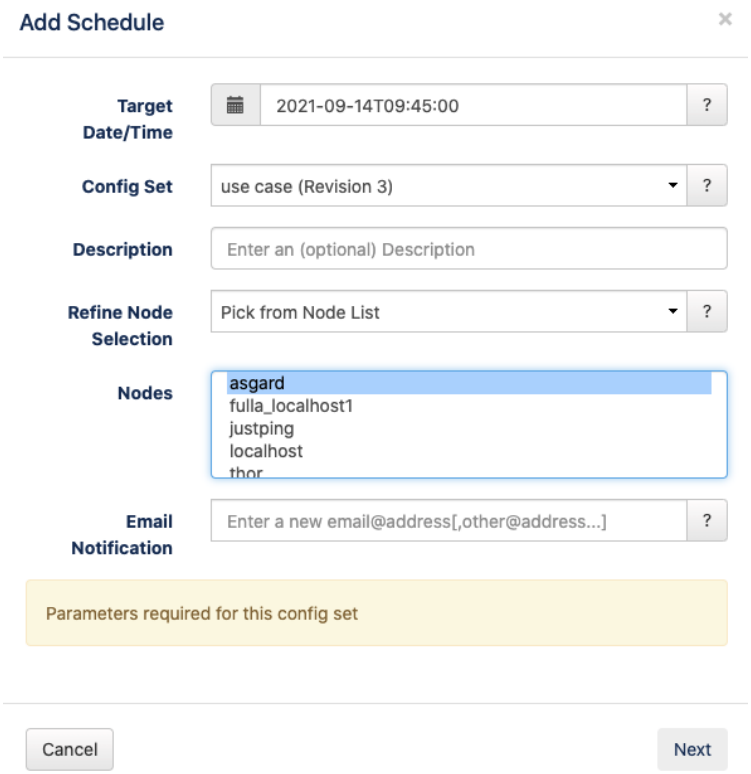
### cli Example

Using the cli tool opconfig-cli, you can specify parameters to push a config set using parameter.parametername, like:

```
./opconfig-cli.pl act=push_configset name="my configset" parameter.interface=Tunnel100 parameter.description="
Configuration change example" debug=2
```

### GUI Example

Using the GUI, if a config set has parameters, you will have a second modal window to specify all the needed parameters:




Config set parameters will be introduced on a second screen:

