configs_generator

Release 0.0.5

ALIASGAR [ALI]

Apr 12, 2022

CONTENTS:

CHAPTER

ONE

CONFIGS_GENERATOR

1.1 configs_generator's documentation

1.1.1 What is configs_generator?

configs_generator is an open python project to help generating text based configuration for various network devices.

Caution: It is solely users responsibility to review the configurations generated by the configs_generator.

Owner of the package or package will not be liable in any manner for any mishap happen then after.

CHAPTER

TWO

INSTALLATION & REQUIREMENTS

2.1 Requirements

1. python >= 3.7

2.2 Installation

Install the configs_generator package:

```
pip install --upgrade configs_generator
```

2.3 Inherited python packages

- nettoolkit
- pandas

CHAPTER

THREE

USER DOCUMENTATION!

3.1 configs_generator User documentation!

3.1.1 What is configs_generator ?

configs_generator is an open-source package, used to generate text configuration of the devices from the excel facts file. It uses configuration template files and Excel data base in order to generate the final configurations.

3.1.2 Who can use ?

Networking geeks can use it to generaete the configurations for their installations or changes.

3.1.3 Where can be used ?

- It is vendor and model agnostic.
- It can be used for generation of config for any vendors such as: Cisco, Juniper, Arista..

3.1.4 Why to use ?

- configuration generation becomes easy, simple
- effortless
- errorless
- reproducive

3.1.5 How can I use it ?

Stay Tuned!

Notice

Make sure to cross-check the generated configurations before using it.

3.2 configuration_generation Usage Guidelines!

3.2.1 Template Preparation

This is very important step in order to generate the configuration accurately.

Through out the template, we will have many variables which requires to be replaced with necessary values. Such static one to one variable v/s value mappings we generally provide thru *var* tab of excel database. There is no constraint on variable naming. Just remember it should match exactly and all match throughout documents will be process and replaced with values.

Apart from static variable/value replacement, we usually have code blocks in the configuration template. Which requires a different kind of treatment.

There are two main types of code blocks.

1. **Conditional Block:** Section which appears in output after successful evaluation of a condition.
2. **Repeat Block:** Section which appears repeatatively in to output after successful evaluation of a condition.

1. Conditional Blocks
    - Such block usually starts with `GOAHEAD FOR <conditions>`, and ends with `GOAHEAD END`.
    - Block start/end string can be modified by providing `condition_starter`, `condition_stopper` arguments while creating `ConfGen` object.
    - In case if condition matches more than one record in database, very first matching row will be considered and used.
    - There must be a conditional-block-end correspond to start, otherwise undesired result may produce.
    - Nested conditional blocks are permitted.
2. REPEAT BLOCK
    - Such block usually starts with `REPEAT EACH <conditions>`, and ends with `REPEAT STOP`.
    - Block start/end string can be modified by providing `repeat_starter`, `repeat_stopper` arguments while creating `ConfGen` object.
    - Each condition matching row from database, will be selected and executed for repetitive config generation.
    - There must be a repeat-block-end correspond to start, otherwise undesired result may produce.
    - Nested repetitive conditional blocks are permitted.

Nesting of *Conditional block* and *Repetitive block* inside each other permitted as well.

Note:

- **<condition>** can be a single match condition or multiple matches (using condition separators either & or |)
- Each condition should be defined within it's own bracket ().
- Each conditions left portion denotes variable to be match in excel fact file, and right portion denotes value to be match with the variable.
- Conditional operators allowed within a condition are *%2==*, *%2!=*, *==*, *!=*, *>=*, *<=*, *>*, *<*
- **%2==**, **%2!=** these are useful for matching odd / even number matching
- String value match should be mentioned with Quote "", : ex: *(LINK_TYPE == "Downlink")*
- Numeric value match should be mentioned without Qoute : ex: *(VLAN == 100)*

3.2.2 Database Preparation

This is second very important step, and should be prepared accurately to get the output configuration correct.

Note: `facts_finder` project can be used to quickly draft the details out of any running device (if already available)

There are two main types of data with which configs_generator interact with. and we need to provide them in two tabs via excel database.

1. **var**: All one to one variables v/s values to be replace in template
2. **tables**: Rest of other matrix details in table format.

1. **var**
    - There should be a tab named `var` in Excel database.
    - There should be two columns named **FIND** and **REPLACE**, for mapping each variable v/s value
    - Column Header can be changed instead of FIND/REPLACE to any other name, but it needs to be declared via `find_column_name` and `replace_column_name` arguments while creating ConfGen object.
    - All variables defined here under *find_column_name* will get replaced with value mentioned in *replace_column_name* while generating configuration.
2. **tables**
    - There should be a tab named `tables` in Excel database
    - There can be any number of columns with any custom column names
    - Data defined here in this tab gets evaluated while executing the condition check during Conditional/Repeat block from template
    - Column header mentioned here in should refer to left portion of condition

3.2.3 Executions

There are three kinds of exection possible.

1. Run thru script.
2. Passing necessary arguments via command line.
3. Interactive command line mode.

Note:

- Template file and database file are mandatory fields, rest of others are optional
- So pre-requisite is: database and configuration template must be ready.

3.2.3.1 Execute via script : Step by step guide

Here is how to do via script

Step 1. Import necessary packages:

```
import configs_generator as cg
```

Step 2. Define Inputs:

```
db = "data.xlsx"
template = "template.txt"
output = "output.txt"
```

Step 3. Create ConfGen object:

```
cfg = cg.ConfGen(
        # ~~~~~~~~~~ Mandatory Arguments ~~~~~~~~~~
        template_file=template,                 # template
        db=db,                                  # database

        # ~~~~~~~~~~ Optional Arguments ~~~~~~~~~~
        output_file=output,                     # output filename ( default: output.txt)
        confGen_minimal=False,                  # execution of var sheet replacement␣
↪only.

        find_column_name="FIND",                # FIND/REPLACE column headers from 'var'␣
↪tab
        replace_column_name="REPLACE",

        condition_starter="GOAHEAD FOR",        # conditional block identifiers
        condition_stopper="GOAHEAD END",
        repeat_starter   ="REPEAT EACH",        # repeat block identifiers
        repeat_stopper   ="REPEAT NEXT",

        nested_section_var_identifier= "PARENT" # nested section variable identifier␣
↪string
)
```

Step 4. Generate configuration using created object:

```
cfg.generate()
```

script will evaluate the template v/s database for the conditions defined in template and generates new configuarion file.

3.2.3.2 Execute via argument parsing

Here is how to do via arguments parsing

To Execute via passing arguments, all inputs should be passed along CLI at once.

Keys available are:

```
python -m configs_generator -h
------------------------------
usage: configs_generator [-h] [-i] [-t TEMPLATE_FILE] [-d DB] [-o OUTPUT_FILE]
                  [-m] [-f FIND_COLUMN_NAME] [-r REPLACE_COLUMN_NAME]
                  [-cs CONDITION_STARTER] [-ce CONDITION_STOPPER]
                  [-rs REPEAT_STARTER] [-re REPEAT_STOPPER]
                  [-nv NESTED_SECTION_VAR_IDENTIFIER]

optional arguments:
-h, --help            show this help message and exit
-i, -interactive      run command interactive mode (default: False)
-t TEMPLATE_FILE, -template TEMPLATE_FILE
                      Template File (text file) (default: None)
-d DB, -database DB   Database File (Excel file) (default: None)
-o OUTPUT_FILE, -output OUTPUT_FILE
                      output File (text file) (default: None)
-m, -minimal          execution of var sheet replacement only.( default:
                      False) (default: False)
-f FIND_COLUMN_NAME, -find FIND_COLUMN_NAME
                      FIND column headers from "var" tab: (default: FIND)
-r REPLACE_COLUMN_NAME, -replace REPLACE_COLUMN_NAME
                      REPLACe column headers from "var" tab: (default:
                      REPLACE)
-cs CONDITION_STARTER, -condition_start CONDITION_STARTER
                      conditional block start identifier (default: GOAHEAD
                      FOR)
-ce CONDITION_STOPPER, -condition_end CONDITION_STOPPER
                      conditional block end identifier (default: GOAHEAD
                      END)
-rs REPEAT_STARTER, -repeat_start REPEAT_STARTER
                      repeat block start identifier (default: REPEAT EACH)
-re REPEAT_STOPPER, -repeat_end REPEAT_STOPPER
                      repeat block end identifier (default: REPEAT STOP)
-nv NESTED_SECTION_VAR_IDENTIFIER, -nested_var NESTED_SECTION_VAR_IDENTIFIER
                      nested section variable identifier string (default:
                      PARENT)
```

Mandatory Inputs:

- **-t TEMPLATE_FILE**: Configuration template text file must be passed along with `-t` key
- **-d DB**: Excel Database file must be passed along with `-d` key

Optional Inputs:

- Other optional arguments are required if there is any deviation from standard.

3.2.3.3 Execute via Interactive CLI

Here is how to do via interactive cli mode

To Execute via Interactive CLI, all inputs should be passed along on CLI while asked.

To start the Interactive mode - pass `-i` key while running the package.

```
C:\..>python -m configs_generator -i
Enter Template File (text file): template.txt
Enter Database File (Excel file): data.xlsx
Enter output File (text file): output.txt
Do you want to execution of var sheet replacement only [yes/no] .(default: no)
change FIND column headers on database "var" tab [default: FIND]:
change REPLACE column headers on database "var" tab [default: REPLACE]:
change conditional block start identifier in tempalte [default: GOAHEAD FOR]:
change conditional block end identifier in tempalte [default: GOAHEAD END]:
change repeat block start identifier in tempalte [default: REPEAT EACH]:
change repeat block end identifier in tempalte [default: REPEAT STOP]:
change nested section variable identifier string  in tempalte [default: PARENT]:
Executing, please wait...
```

3.3 some advance usage of configs_generator

3.3.1 Advance Template Variables

Apart from usual variable/value swap and two types of code blocks explained earliar, we could have separate type of code block such as child block depends on parent block selection. Such block is to be treated slight differently.

See below example config template:

```
# ------------------------------------------------
#  repeat below block for all available vrf's
# ------------------------------------------------
routing-instance [vrf_name] {
    # ------------------------------------------
    #  Repeat below for all interface for given vrf
    # ------------------------------------------
    interface [int_name] ;
    #
}
#
```

Here `[int_name]` variable selection depends on parent `[vrf_name]` selection. so normal **REPEAT_FOR <condition>** block will not work here b/c it requires a static value in it's condition.

Here is solution to that:

use `nested_section_var_identifier` in order to select parent variable and child variable comparision.

So for our above example Template requires to be edited as below with conditions. where `[int_vrf]` is another column which should contain the vrf name for the particular interface.

Modified Template will look like:

```
REPEAT EACH ([vrf_name] != "")
routing-instance [vrf_name] {
        REPEAT EACH ([int_vrf] == PARENT.[vrf_name])
        interface [int_name] ;
        REPEAT STOP
        #
}
REPEAT STOP
#
```

our database looks like this on *tables* tab.

	A	B	C	D	E
1	**detail_type**	**[int_name]**	**[int_vrf]**	**[vrf_name]**	
2	vrf			cust_a	
3	vrf			cust_b	
4	vrf			cust_c	
5	interface	Gig0/2	cust_a		
6	interface	Gig0/3	cust_b		
7	interface	Gig0/4	cust_c		
8	interface	Gig0/5	cust_c		
9	interface	Gig0/6	cust_a		
10	interface	Gig0/7	cust_b		
11	interface	vlan501	cust_a		
12	interface	vlan502	cust_b		
13					
14					
15					

```
# Execution will show these
Template Verified
configuration generation is in progress, please wait...
Input Read.. OK
ConfGen.. OK
Write Config to File.. output.txt OK
[Finished in 1.1s]
```

Here is what output generated there after:

```
routing-instance cust_a {
        interface Gig0/2 ;
        interface Gig0/6 ;
        interface vlan501 ;
        #
}
routing-instance cust_b {
        interface Gig0/3 ;
        interface Gig0/7 ;
```

(continues on next page)

(continued from previous page)

```
        interface vlan502 ;
        #
}
routing-instance cust_c {
        interface Gig0/4 ;
        interface Gig0/5 ;
        #
}
#
```

3.3.2 Advance condition Matching

Now consider below database, from where we want to get the delta out only for odd vlan numbers.

	A	B	C	D	E
1	**detail_type**	**[int_name]**	**[int_address]**	**[vlan_name]**	
2	irb	501	1.1.1.1/24		
3	irb	502	1.1.2.1/24		
4	irb	503	1.1.3.1/24		
5	irb	504	1.1.4.1/24		
6	irb	505	1.1.5.1/24		
7	irb	506	1.1.6.1/24		
8	irb	507	1.1.7.1/24		
9	irb	508	1.1.8.1/24		
10	vlan	501	n/a	customer_a	
11					
12					
13					
14					
15					

Given Original Template:

```
interfaces {
  irb {
    # ------------------------------------------------
    # Repeat below segment for each odd vlan numbers
    # ------------------------------------------------
    unit [int_name] {
      inet {
        address [int_address];
      }
    }
    #
  }
}
```

Template after adding conditions:

```
interfaces {
  irb {
    REPEAT EACH ((detail_type == "irb") & ([int_name] %2== 1))
    unit [int_name] {
      inet {
        address [int_address];
      }
    }
    #
    REPEAT STOP
  }
}
```

Notice the use of **%2== 1** which matches all odd numbers

Here is the output generated after execution:

```
interfaces {
  irb {
    unit 501 {
      inet {
        address 1.1.1.1/24;
      }
    }
    #
    unit 503 {
      inet {
        address 1.1.3.1/24;
      }
    }
    #
    unit 505 {
      inet {
        address 1.1.5.1/24;
      }
    }
    #
    unit 507 {
      inet {
        address 1.1.7.1/24;
      }
    }
    #
  }
}
```

CHAPTER

FOUR

TECHNICAL DOCUMENTATION!

4.1 configuration generation module

class configs_generator.confGen.**ConfGen**(*db=None*, *template_file=None*, *output_file='output.txt'*, *xls_db=None*, *xls_db_sheet='tables'*, *var_db=None*, *var_db_sheet='var'*, *confGen_minimal=False*, *find_column_name='FIND'*, *replace_column_name='REPLACE'*, *repeat_starter='REPEAT EACH'*, *repeat_stopper='REPEAT STOP'*, *condition_starter='GOAHEAD FOR'*, *condition_stopper='GOAHEAD END'*, *nested_section_var_identifier='PARENT'*)

Bases: object

create an object by providing necessary details to generate configuration Mandatory inputs are template_file, db

depricationwarning()

generate()

Check Template consistency and start execution to generate config

inputcheck()

check for mandatory inputs

istemplatevariried()

sequence of checks to validate the conditions consistency in template. Errored Template can cause infinite loop in execution. raises exception if any error or returns True

templateconditionscountcheck()

template check step 2. count of conditions and condition closures matching

templatefiletypecheck()

template check step 1. File type verification (.txt)

templatenestingconditioncheck()

template check step 3. nested conditions closure check to avoid infinite loop

class configs_generator.confGen.**Read**(*section_starter*, *section_stopper*, *var_db_columns*, *xls_db_sheet*)

Bases: object

database(*xls_db*)

dataframes(*xls_db*)

read Excel sheets and store it in Dictionary

dataframes_depricated(*dataframes*)

Depricated way of reading sheets. To be Remove in next version

read_temporary_template(*template_file*)

Reads provided template file, generate sections in the template

template(*template_file*)

class configs_generator.confGen.**Replicate**(*section_dict*, *dataframes*, *confGen_minimal=False*)

Bases: object

Replicate section configs using database provided in dataframes.

get_section_dataframe(*df_condition*)

returns filtered dataframe for given condition

get_table_line_vars(*RowData*)

data dictionary for a single row

go_thru_section_list()

Entry point for sections

initial_Go()

start parent section/i.e. file

logic_line_to_df_condition(*logic_line*)

convert condition to pandas dataframe condition

nested_section_var_identifier = 'PARENT.'

property output

replicate_for_data()

Go thru each line of data (data frame), and update configs in that section

start()

Executes the replication

update_condition_for_cf_var(*logic_line*)

update condition line with carried forwarded Nested Section Variable value. –>updated condition (str)

update_config_section(*row*)

Add config with updates for given one row

updated_line(*line*, *RowData*, *header*)

class configs_generator.confGen.**Section**(*template_name*, *section_starter*, *section_stopper*)

Bases: object

get(*f*, *logic_line*)

initiates the of section

read_lines(*f*)

lines inside of a section

configs_generator.confGen.**random_text_file**(*source_file*, *temporary_prefix*)
create a duplicate of source text file with random name using given prefix.

configs_generator.confGen.**replace_var_candidates**(*temporary_template*, *var_db_df*, *var_db_columns*)
update find/replace pairs in temporary template file

configs_generator.confGen.**section_type**(*line*, *section_starter*)
checks section type in line and returns tuple with details if section is conditional/repeatative

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

U