JUNOS CONFIGURATION GROUPS

By: Juniper Instructor Yasmin Lara

In this instructional article by Sunset Learning Institute, Juniper Specialized Instructor Yasmin Lara provides a definition of Junos Configuration Groups, describes some of the advantages of using configuration groups, and lists important details to keep in mind when using configuration groups. She also provides examples of how to use configuration groups to configure Firewall Filters, Routing Policies, Interfaces MTU, SRX Policies, and Dual Routing Engine. This is a vital resource to have for building configuration groups!

What Are Configuration Groups?

Configuration Groups allow you to create sets of configuration statements that you can later apply to other sections of your configuration. The contents of a group are inherited by the section where you apply the group.

What Are The Benefits Of Using Configuration Groups?

When you use Configuration groups you don't have to configure the same set of configuration statements multiple times, when they are needed in several places. As a result:

- Your configuration file becomes smaller, and more logically constructed.
- You save time configuring the device.
- Your configuration is more consistent.
- You avoid mistakes when configuring the same set of commands in different places.

What Do You Need To Know About Configuration Groups?

- You can name your group whatever you want except for:
  - Anything that starts with the prefix junos-
  - juniper-ais
  - re0/re1
  - node0/node1 (on SRX)
- The same group can be applied to different sections of the configuration.
- In a single configuration group, you can have statements that can be inherited by different sections of your configuration.
  - For example, you can create a single configuration group that is applied to ALL interfaces, but still applies different parameters to ge interfaces, and xe interfaces.
- A wildcard expression can be used to allow configuration statements in the group to be inherited by multiple objects, if they match the expression. For example: "`interfaces <*>`" indicates all interfaces.
- An expression is written between < >, and the most common wildcards you will find are:
  - * Matches any string of characters
  - ? Matches any single character
  - [ ] Matches the characters between the brackets
  - - Indicates a range

Examples:

- `ge-<*>` matches all ge interfaces.
- `ge-<0/0/*>` matches ge interfaces on fpc0/pic0
- `ge-<0/0/[0-3]>` matches ge interfaces 0 through3, on fpc0/pic0
- `ge-<0/0/1?>` matches ge interfaces 10-19, on fpc0/pic0

- Configuration groups are different that other groups required to configure certain features, like for example a BGP group.
- Changes in a configuration group are automatically inherited by the target configuration.
- Inherited values can be overridden in the target configuration, without affecting the group where the values are inherited from, or the inheritance in other places of the configuration.
  - For example, a group can be setting the MTU of ALL ge interfaces to 4500 bytes, but if you specifically configure an MTU 9100 under ge-0/0/0, this value overrides the inherited value. All the other ge interfaces will still use 4500.
- You apply a group using `apply-groups [group-name]`
  - More than group can be applied: `apply-groups [group-name1 group-name2 …]`
  - If you do specify more than one group name, list them in the desired order of inheritance. The configuration data in the first group takes priority over the data in subsequent groups.
- You can exclude a section of your configuration from inheriting configuration statements from a group, by using `apply-groups-except`.
  - For example, if you apply a group named `interface-mtu` to ALL interfaces, and then you configure `apply-group-except interface-mtu` under `ge-0/0/0`, all interfaces but `ge-0/0/0` will inherit the statements from the group.
- To see the results of your configuration group you need to use `| display inheritance` or `| display inheritance | except ##`, as shown in the examples below.

Examples:

1.- Firewall Filter

Description:

In this example, a term called **`reject-rfc1918`** is added to any ipv4 firewall filter, to reject source addresses from the RFC1918 ranges.

Configuration:

```
[edit groups firewall-rfc1918 firewall family inet]
ylara@R1# show
filter <*> {
    term reject-rfc1918 {
        from {
           source-address {
               10.0.0.0/8;
               172.16.0.0/12;
               192.168.0.0/16;
           }
          then reject;
        }
    }
}
```

```
[edit firewall family inet]
ylara@R1# show
apply-groups firewall-rfc1918;        ← will be applied to ALL ipv4 (family inet) filters.
filter filter1 {
    term allow-ospf {
        from protocol ospf;
        then accept;
    }
}
filter filter2 {
    term allow-bgp {
        from {
        protocol tcp;
        from port 179;
        }
        then accept;
    }
}
```

Results:

```
[edit firewall family inet filter filter1]
root@R1# show | display inheritance
term allow-ospf {
        from protocol ospf
        then accept
        }
##
## ' reject-rfc1918 ' was inherited from group 'firewall-rfc1918'
##
term reject-rfc1918 {
    ##
    ## 'from' was inherited from group 'firewall-rfc1918'
    ##
    from {
        source-address {
            ##
            ## '10.0.0.0/8' was inherited from group 'firewall-rfc1918'
            ##
            10.0.0.0/8;
            ##
            ## '172.16.0.0/12' was inherited from group 'firewall-rfc1918'
            ##
            172.16.0.0/12;
            ##
            ## '192.168.0.0/8' was inherited from group 'firewall-rfc1918'
            ##
            192.168.0.0/16;
        }
    }
  ##
  ## 'then' was inherited from group 'firewall-rfc1918'
  ##
  then {
      ##
        ## 'reject' was inherited from group 'firewall-rfc1918'
        ##
        reject;
  }
}
```

```
[edit firewall family inet filter filter1]
root@R1# show | display inheritance | except ##
term allow-ospf {
       from protocol ospf
       then accept
       }
term reject-rfc1918 {
    from {
        source-address {
            10.0.0.0/8;
            172.16.0.0/12;
            192.168.0.0/16;
        }
    }
  then {
       reject;
  }
}

[edit firewall family inet filter filter2]
root@R1# show | display inheritance
term allow-bgp {
  from {
       protocol tcp;
       from port 179;
       }
  then accept;
  }
##
## ' reject-rfc1918 ' was inherited from group 'firewall-rfc1918'
##
term reject-rfc1918 {
    ##
    ## 'from' was inherited from group 'firewall-rfc1918'
    ##
    from {
        source-address {
            ##
            ## '10.0.0.0/8' was inherited from group 'firewall-rfc1918'
            ##
            10.0.0.0/8;
            ##
            ## '172.16.0.0/12' was inherited from group 'firewall-rfc1918'
            ##
            172.16.0.0/12;
            ##
            ## '192.168.0.0/16' was inherited from group 'firewall-rfc1918'
            ##
            192.168.0.0/16;
        }
    }
  ##
  ## 'then' was inherited from group 'firewall-rfc1918'
  ##
  then {
      ##
      ## 'reject' was inherited from group 'firewall-rfc1918'
      ##
      reject;
  }
}
```

```
[edit firewall family inet filter filter2]
root@R1# show | display inheritance | except ##
term allow-bgp {
  from {
       protocol tcp;
       from port 179;
       }
  then accept;
  }
term reject-rfc1918 {
    from {
       source-address {
         10.0.0.0/8;
         172.16.0.0/12;
         192.168.0.0/16;
         }
    }
  then {
       reject;
  }
}
```

2.- Routing Policy

Description:

This example adds a matching condition for RFC1918 prefixes. The group is configured to math on any routing policy, and any term, but then is applied to a specific term called **reject-rfc1918**, within a policy named **to-core**.

Configuration:

```
[edit groups rfc1918 policy-options]
root@R1# show
policy-statement <*> {
    term <*> {
        from {
            route-filter 10.0.0.0/8 orlonger;
            route-filter 172.16.0.0/12 orlonger;
            route-filter 192.168.0.0/16 orlonger;
        }
    }
}

[edit policy-options]
root@R1# show
policy-statement to-core {
    term reject-rfc1918 {
        apply-groups rfc1918; ← only want it on this term.
        then reject;
    }
```

Results:

```
[edit policy-options policy-statement to-core]
root@R1# show term reject-rfc1918 | display inheritance
##
## 'from' was inherited from group 'rfc1918'
##
from {
    ##
    ## 'orlonger' was inherited from group 'rfc1918'
    ##
    route-filter 10.0.0.0/8 orlonger;
    ##
    ## 'orlonger' was inherited from group 'rfc1918'
    ##
    route-filter 172.16.0.0/12 orlonger;
    ##
    ## 'orlonger' was inherited from group 'rfc1918'
    ##
    route-filter 192.168.0.0/16 orlonger;
}
then reject;

[edit policy-options policy-statement to-core]
root@R1# show term reject-rfc1918 | display inheritance | except ##
from {
   route-filter 10.0.0.0/8 orlonger;
   route-filter 172.16.0.0/12 orlonger;
   route-filter 192.168.0.0/16 orlonger;
}
then reject;
```

3.- Interface MTU

Description:

This example configures an MTU of 9100 for IPv4, iso, and MPLS on all ge interfaces (at the logical level), and an MTU of 9192 for all interfaces (at the physical level). Also, interface ge-0/0/0 is excluded from the group inheritance.

Configuration:

```
[edit groups maximum-GigE-MTU interfaces]
root@R1# show
    <ge-*> {
        unit <*> {
            family inet {
                mtu 9100;           ← all ge interfaces get an mtu of 9192 for ipv4 (ifl level)
            }
            family iso {
                mtu 9100;           ← all ge interfaces get an mtu of 9192 for iso (ifl level)
            }
            family mpls {
                mtu 9100;           ← all ge interfaces get an mtu of 9192 for mpls (ifl level)
            }
        }
    }
    <*> {
        mtu 9192;                   ← all interfaces get an MTU of 9192 (physical interface level)
    }
```

```
[edit interfaces]
root@R1# show
apply-groups maximum-GigE-MTU;
ge-0/0/0 {
              apply-groups-except maximum-GigE-MTU; ← exclude ge-0/0/0 from group maximum-GigE-MTU
      gigether-options {
                    802.3ad ae4;
      }
}
ge-0/0/8 {
      unit 0 {
                  family inet;
      }
}
```

Results:

```
[edit interfaces]
root@R1# show ge-0/0/0 | display inheritance
apply-groups-except maximum-GigE-MTU;
gigether-options {
    802.3ad ae4;
}
```

No inheritance.

```
[edit interfaces]
root@R1# show ge-0/0/8 | display inheritance
##
## '9192' was inherited from group 'maximum-GigE-MTU'
##
mtu 9192;
unit 0 {
         ##
         ## 'inet' was inherited from group 'maximum-GigE-MTU'
         ##
     family inet {
         ##
         ## '9100' was inherited from group 'maximum-GigE-MTU'
         ##
         mtu 9100;
     }
         ##
         ## 'iso' was inherited from group 'maximum-GigE-MTU'
         ##
     family iso {
         ##
         ## '9100' was inherited from group 'maximum-GigE-MTU'
         ##
         mtu 9100;
     }
         ##
         ## 'mpls' was inherited from group 'maximum-GigE-MTU'
         ##
     family mpls {
         ##
         ## '9100' was inherited from group 'maximum-GigE-MTU'
         ##
         mtu 9100;
     }
}
```

4.- SRX Security Policy

Description:

The example shows how to add a standard policy to any security policy configured on the firewall.

Configuration:

```
[edit groups STANDARD]
root@SRX1# show | display set
set security policies from-zone <*> to-zone <*> policy STANDARD match source-address any
set security policies from-zone <*> to-zone <*> policy STANDARD match destination-address any
set security policies from-zone <*> to-zone <*> policy STANDARD match application any
set security policies from-zone <*> to-zone <*> policy STANDARD then deny
set security policies from-zone <*> to-zone <*> policy STANDARD then log session-close
set security policies from-zone <*> to-zone <*> policy STANDARD then count

[edit security policies]
root@SRX1# show | display set
   set from-zone trust to-zone trust policy STANDARD description TRUST-TO-TRUST-POLICY
   set apply-groups STANDARD
```

Results:

```
[edit security policy from-zone trust to-zone trust policy STANDARD]
root@SRX1# show | display inheritance
description TRUST-TO-TRUST-POLICY;
##
## 'match' was inherited from group 'STANDARD'
##
match {
      ##
   ## 'any' was inherited from group 'STANDARD'
   ##
   source-address any;
      ##
   ## 'any' was inherited from group 'STANDARD'
   ##
   destination-address any;
      ##
   ## 'any' was inherited from group 'STANDARD'
   ##


        application any;

}
```

```
##
## 'then' was inherited from group 'STANDARD'
##
then {
    ##
    ## 'deny' was inherited from group 'STANDARD'
    ##
    deny;
    ##
    ## 'log' was inherited from group 'STANDARD'
    ##
    log {
    ##
    ## 'session-close' was inherited from group 'STANDARD'
    ##
    session-close;
    }
    ##
    ## 'count' was inherited from group 'STANDARD'
    ##
    count;
}

[edit security policy from-zone trust to-zone trust policy STANDARD]
root@SRX1# show | display inheritance | except ##
description TRUST-TO-TRUST-POLICY;
match {
    source-address any;
    destination-address any;
    application any;

}
then {
    deny;
    log {

            session-close;
    }
    count;
}
```

5.- BGP Import Policy

Description:

This example configures applies an export policy named **import-policy2**, that accepts a default route and rejects everything else, from any BGP neighbor in any BGP groups.

Configuration:

```
{master}[edit groups bgp-import]
root@R1# show | display set relative
set protocols bgp group <*> neighbor <*> export import-policy2

{master}[edit policy-options policy-statement import-policy1]
root@R1# show | display set relative
term 1 {
      from neighbor 1.2.3.4;
      then accept;
}
```

```
{master}[edit policy-options policy-statement import-policy2]
root@R1# show | display set relative
term 1 {
from route-filter 0/0 exact;
      then accept;
}
term 2 {
then reject;
}

{master}[edit]
root@R1# set apply-groups bgp-import

{master}[edit protocols bgp group ebgp-neighbors]
root@R1# show | display set relative
set neighbor 10.11.12.2 description R2                          ← import policy already configured
set neighbor 10.11.12.3 description R3 import import-policy1    ← no import policy
```

Results:

```
{master}[edit protocols bgp group ebgp-neighbors]
root@R1# show | display inheritance
neighbor 10.11.12.2 {
   description R2;
   ##
   ## 'import-policy2' was inherited from group 'bgp-import'        ← import-policy2 was inherited.
   ##
   import import-policy2;
}
neighbor 10.11.12.3 {
   description R3;
   ##
   ## 'import-policy2' was inherited from group 'bgp-import'
   ##
import [import-policy1 import-policy2];            ← import-policy1 explicitly configured,
}                                                    import-policy2 was inherited.

{master}[edit protocols bgp group ebgp-neighbors]
root@R1# show | display inheritance | except ##
neighbor 10.11.12.2 {
   import import-policy1;
}
neighbor 10.11.12.3 {
   description R3;
   import [import-policy1 import-policy2];         ← Import- policy1 will be applied to neighbor
}                                                    10.11.12.3, before import-policy2.
```

Note: When a group applies a BGP import policy to a neighbor with an already configured policy, the neighbor ends up with TWO policies applied. The policy applied by the group is last in the policy chain.

6.- Dual Routing Engine

Description:

For routers with dual Routing Engines, you can create groups re0 and re1. The configuration specified under group re0 is only applied to the RE in slot 0, while group re1 is only applied to the RE in slot 1. As a result, both REs can use the same configuration file, while each uses only the configuration statements that apply.

The example shows how to use this concept to configure a unique hostname and management address for RE0 and RE1.

Configuration:

```
[edit groups re0]
root@R1#
set system host-name R1-RE0
set interfaces fxp0 unit 0 family inet address 10.1.1.1/24

[edit groups re1]
root@R1#
set system host-name R1-RE1
set interfaces fxp0 unit 0 family inet address 10.1.1.2/24

[edit]
root@R1# set apply-groups [ re0 re1 ]

[edit]
root@R1# commit
```

Results:

```
[edit groups re0]
root@R1-RE0# show
      system {
      host-name R1-RE0;
      interfaces {
            fxp0 {
              unit 0 {
                 family inet {
                      address 10.1.1.1/24;
                      }
            }
      }
}

[edit groups re1]
root@R1-RE0# show
      system {
      host-name R1-RE1;
      interfaces {
            fxp0 {
              unit 0 {
                 family inet {
                      address 10.1.1.2/24;
                 }
            }
      }
}
```

```
[edit]
root@R1-RE0# show system | display inheritance | match re0
         ## 'R1-RE0' was inherited from group 're0'
         ## 'fxp0' was inherited from group 're0'
         ## '0' was inherited from group 're0'
         ## 'inet' was inherited from group 're0'
         ## '10.1.1.1/24' was inherited from group 're0'

[edit]
root@R1-RE0# show | display inheritance | match re1
```

Nothing is displayed.

```
root@R1> request routing-engine login re1

root@R1-RE1> edit

[edit]
root@R1-RE0# show system | display inheritance | match
         ## 'R1-RE1' was inherited from group 're1'
         ## 'fxp0' was inherited from group 're1'
         ## '0' was inherited from group 're1'
         ## 'inet' was inherited from group 're1'
         ## '10.1.1.2/24' was inherited from group 're1'
```