Secure Computer Cluster Administration with SSH

Jens Getreu

Revision History

Revision 1.2	19.1.2016	JG

Table of Contents

This article compares the SSH-cluster-tools `parallel-ssh`, `clusterssh` and `clustershell`. SSH-cluster-tools are handy to execute shell-commands on a group of computers in parallel.

[1]

[1] For other solutions see: What is a good modern parallel SSH tool? [http://serverfault.com/questions/2533/linux-running-the-same-command-on-many-machines-at-once] and Linux - Running The Same Command on Many Machines at Once [http://serverfault.com/questions/2533/linux-running-the-same-command-on-many-machines-at-once]

1. parallel-ssh

`parallel-ssh` is a program for executing ssh in parallel on a number of hosts. It provides features such as sending input to all of the processes, passing a password to ssh, saving output to files, and timing out.

Packet name: `pssh`

1.1. Install parallel-ssh

```
$ apt-get install pssh
```

1.2. Enter passwords interactively

`parallel-ssh` can run ssh using the mode referred to as “keyboard interactive” password authentication.

```
$ parallel-ssh -x "-o StrictHostKeyChecking=no" -i  -l root -A -H
 localhost hostname
```

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
Password:
[1] 21:16:33 [SUCCESS] localhost
matou2
Stderr: Warning: Permanently added 'localhost' (ECDSA) to the list of
 known hosts.
```

1.3. Define node groups

With the option `-h` `parallel-ssh` reads host domain names or IPs from the file “hostlist”. The **lines** in this file are of the form `[user@]host[:port]` and can include blank lines and comments (lines beginning with “#”). If multiple host files are given (the `-h` option is used more than once), then parallel-ssh behaves as though these files were concatenated together. If a host is specified specified multiple times, then parallel-ssh will connect the given number of times.

Filecopy example

```
$ cat hostlist
sysadm.local
192.168.122.57

$ parallel-scp -h hostlist -l root /etc/hosts /tmp/hosts
```

1.4. parallel-ssh without entering passwords using sshpass

`sshpass` is a utility designed for running ssh using the mode referred to as "keyboard interactive" password authentication, but in non-interactive mode.

We assume you have multiple machines to manage but don't have SSH-keys on them. It's still possible to use `parallel-ssh` and `sudo` without having to enter any passwords. [2]

1. Install parallel-ssh and sshpass

   ```
   $ apt-get install sshpass pssh
   ```

2. Usage example with `sudo`.

   ```
   $ sshpass -f remotePasswordFile parallel-ssh -I -A -h hostlist "sudo -
   S apt-get dist-upgrade" < remoteRootPasswordFile
   ```

   "remotePasswordFile" contains your password to log in to the remote computer. Concerning `parallel-ssh`, `-I` reads from stdin and `-A` asks for a password. For `sudo`, `-S` reads the password from stdin here "remoteRootPasswordFile".

3. You might also want to disable `StrictHostKeyChecking` to avoid the question: "Do you want to add this host to the list of known hosts?" This will automatically answer "yes".

[2] https://nyxi.eu/blog/2013/08/26/parallel-ssh-and-sudo/, Aug 26th, 2013.

```
$ sshpass -f remotePasswordFile parallel-ssh -x "-o
 StrictHostKeyChecking=no" -I -A -h hostlist "sudo -S apt-get dist-
upgrade" < remoteRootPasswordFile
```

From the second invocation on `-x "-o StrictHostKeyChecking=no"` can be omitted.

1.5. Distribute SSH-key to a cluster with parallel-ssh

Task

Provide secure access without passwords to a big number of computers via ssh.

Precondition

All nodes have a `sshd`-server running and they all have the same root password.

Solution

Automatic distribution of the local `id_rsa.pub` key to all nodes by using `parallel-ssh` and `sshpass`.

1. Store remote computers password in a file.

```
$ echo "secret" > remotePasswordFile
```

All passwords must be the same on all remote computers.

2. Store remote computers domain names or IP's in list.

```
$ cat >>hostlist <<EOF
sysadm.local
EOF
```

In this example the list contains only one host. More then one host domain names or IP's are separated by newline.

3. Contact remote hosts in order to populate the local `known_hosts` file.

```
$ sshpass -f remotePasswordFile parallel-ssh -x "-o
 StrictHostKeyChecking=no" -i -A -l root -h hostlist "hostname"
```

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
[1] 21:31:13 [SUCCESS] sysadm.local
sysadminclass
Stderr: Warning: Permanently added 'sysadm.local' (ECDSA) to the list
 of known hosts.
```

4. Generate key pairs on remote computers.

```
$ sshpass -f remotePasswordFile parallel-ssh -i -A -l root -h hostlist
 "ssh-keygen -t rsa -N '' -f /root/.ssh/id_rsa"
```

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
[1] 20:36:00 [SUCCESS] sysadm.local
Generating public/private rsa key pair.
Created directory '/root/.ssh'.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
8f:4f:57:97:cd:81:cc:2c:51:a0:f3:76:fb:a5:52:42 root@sysadminclass
The key's randomart image is:
+--[ RSA 2048]----+
|           oo.   |
|          . = .  |
|         o . = . |
|          o E  .+|
|        S  + ...+|
|         o. o.o. |
|        . o .+  .|
|         o .. ...|
|          .  ... |
+-----------------+
```

5. Append the local public key to remote `authorized_keys` files.

```
$ sshpass -f remotePasswordFile parallel-ssh -I -A -l root -h
 hostlist "cat - >> /root/.ssh/authorized_keys;chmod 600 /root/.ssh/
authorized_keys" < ~/.ssh/id_rsa.pub
```

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
[1] 20:36:27 [SUCCESS] sysadm.local
```

1. Check access without providing passwords.

```
$ parallel-ssh -i  -l root -h hostlist "hostname"
```

```
 [1] 20:37:12 [SUCCESS] sysadm.local
sysadminclass
```

2. clusterssh

The command `clusterssh` opens an administration console and an xterm to all specified hosts. Any text typed into the administration console is replicated to all windows. All windows may also be typed into directly.

Packetname: `clusterssh`

Writen in Perl.

Preparation

- Make sure to be able to connect to all node without password (`/root/.ssh/authorized_keys`). This can be realized with the method explained in Section 1.5, “Distribute SSH-key to a cluster with parallel-ssh”.

  

  If you omit this step you will have to enter a password in every opened window manually.

- Resolve hostnames locally if you do not want to trust your nameserver.

  Append the following to `/etc/hosts`

```
192.168.1.80 matou5.local
192.168.1.88 matou2.local
192.168.1.77 matou1.local
```

Execution

```
# cssh localhost,matou1.local,matou5.local$ cat /root/.ssh/id_rsa.pub |
 clush -O ssh_path='sshpass -p <OtherHostsPassword> ssh' \
                                    -O ssh_options='-oBatchMode=no -
oStrictHostKeyChecking=no' \
                                    -l root -w <OtherHosts> -b 'cat - >>/
root.ssh/authorized_keys; chmod 600 /root.ssh/authorized_keys'
```

Node group definition

```
# cat /etc/clustershell/groups
```

```
home: localhost matou1.local matou5.local
```

> The `clustershell` `/etc/clustershell/groups` configuration file (see below) and the `clusterssh` configuration file have similar syntax. This is why the configuration file of the other program is reused here.

Execution with node groups

```
$ cssh -l root -c /etc/clustershell/groups home
```

opens the node group “home”.

3. clustershell

This tool is intended for (but not limited to) cluster administration where the same configuration or commands must be run on each node within the cluster.

Packet name: `clustershell`

3.1. Command overview

Most of the following examples are taken from Bastian Ballmann's cluster-shell github-page[3].

- Define node groups

  Edit `/etc/clustershell/groups`

```
node_all: cluster-node[001-999].somewhere.in-the.net
```

- Run a command on all nodes

```
clush -w @node_all "my_command with_params"
```

- Iterate over nodes

```
for NODE in $(nodeset -e @node_all); do scp some_file root@$NODE:~/;
done
```

- Diff results

```
clush -w @node_all --diff "dmidecode -s bios-version"
```

- Combine results

```
clush -w @node_all -b "uname -a"
```

- Copy file

```
clush -v -w @node_all --copy a_file
```

[3] https://raw.githubusercontent.com/balle/balu-wiki/master/cluster/clustershell.rst

- Retrieve a file

  Will create files like `id_rsa.node001`

```
clush -v -w @node_all --rcopy /root/.ssh/id_rsa
```

- Append a local file to remote files

```
cat ~/.ssh/id_rsa.pub | clush -w matou5.local -b 'cat - >>/root/.ssh/
authorized_keys'
```

- Limit number of connections / command timeouts

```
fanout: 256
connect_timeout: 15
command_timeout: 0
```

- Scripting in Python

```
from ClusterShell.Task import task_self, NodeSet

task = task_self()
task.run("/bin/uname -r", nodes="mynode[001-123]")

for output, nodes in task.iter_buffers():
    print NodeSet.fromlist(nodes), output
```

More examples can be found in ClusterShell Documentation[4] and in the manual page `man clush`.

3.2. Distribute SSH-key to a cluster with clustershell

> The following works with `clustershell` version >= 1.7 only!
> More information can be found here[5].

1. Install `sshpass`
2. Test remote access.

[4] https://media.readthedocs.org/pdf/clustershell/latest/clustershell.pdf
[5] https://github.com/cea-hpc/clustershell/pull/248#issuecomment-104192680

```
> clush -O ssh_path='sshpass -p <OtherHostsPassword> ssh' \
        -O ssh_optons='-oBatchMode=no -oStrictHostKeyChecking=no' \
        -l root \
        -w <OtherHosts> echo hello
```

Every remote host should answer with "hello".

3. Generate key pairs on remote hosts.

```
$ clush -O ssh_path='sshpass -p <OtherHostsPassword> ssh' \
        -O ssh_optons='-oBatchMode=no -oStrictHostKeyChecking=no' \
        -l root \
        -w <OtherHosts> \
        -b  "ssh-keygen -t rsa -N '' -f /root.ssh/id_rsa"
```

4. Distribute and deploy a public ssh-key to remote hosts.

> Please take into account the section “SECURITY CONSIDERATIONS” of the manual page `man sshpass` concerning the use of the `-p` option!

```
$ cat /root/.ssh/id_rsa.pub | \
    clush -O ssh_path='sshpass -p <OtherHostsPassword> ssh' \
     -O ssh_options='-oBatchMode=no -oStrictHostKeyChecking=no' \
       -l root \
       -w <OtherHosts> \
       -b 'cat - >>/root.ssh/authorized_keys; chmod 600 /root.ssh/
authorized_keys'
```

5. Test remote access without password.

```
> clush -l root -w <OtherHosts> echo hallo
```

Every remote host should answer with "hallo".

Now you should be able to `slogin` as `root` to every node without password.

3.3. Define node groups

/etc/clustershell/groups.conf

```
# cat /etc/clustershell/groups.conf
```

```
[Main]
default: local

[local]
map: sed -n 's/^$GROUP:\(.*\)/\1/p' /etc/clustershell/groups
all: sed -n 's/^all:\(.*\)/\1/p' /etc/clustershell/groups
list: sed -n 's/^\([0-9A-Za-z_-]*\):.*/\1/p' /etc/clustershell/groups
```

/etc/clustershell/groups

```
 # cat /etc/clustershell/groups
```

```
home: localhost matou1.local matou5.local
```

Optional: If you do not want to trust your nameserver you can resolve host-names locally by appending the following to `/etc/hosts`

/etc/hosts

```
192.168.1.77 matou1.local
192.168.1.88 matou2.local
192.168.1.80 matou5.local
```

3.4. Non-interactive mode

Example 1

```
 $ clush -l root -w @home -b cat /proc/version
```

```
---------------
matou[1,5].local,localhost (3)
---------------
Linux version 3.16.0-0.bpo.4-amd64
```

Example 2

```
 $ clush -l root -w @home hostname
```

```
localhost: matou2
matou1.local: matou1
matou5.local: ssh: connect to host matou5.local port 22:
Connection timed out
```

Execute and aggregate output

```
 $ clush -l root -w @home -bL date
```

```
matou1.local,localhost: Sun Mar  1 18:54:36 EET 2015
matou5.local: Sun Mar  1 16:55:07 GMT 2015
```

3.5. Interactive mode

Single-character interactive commands

`clush` also recognizes special single-character prefixes that allows the user to see and modify the current nodeset (the nodes where the commands are executed). These single-character interactive commands are detailed below:

Interactive special commands	**Comment**
`clush> ?`	show current nodeset
`clush> +<NODESET>`	add nodes to current nodeset
`clush> -<NODESET>`	remove nodes from current nodeset
`clush> !<COMMAND>`	execute <COMMAND> on the local system
`clush> =`	toggle the ouput format (gathered or standard mode)

```
 $ clush -l root -w @home -b
```

```
Enter 'quit' to leave this interactive mode
Working with nodes: matou[1,5].local,localhost
clush> hostname
---------------
matou1.local
---------------
matou1
```

```
---------------
matou5.local
---------------
matou5
---------------
localhost
---------------
matou2
clush> quit
```

3.6. Further reading

- ClusterShell, a scalable execution framework for parallel tasks[6]
- ClusterShell * v1.6, User and Programming Guidei (obsolete)[7]
- ClusterShell Documentation[8]
- Bastian Ballmann's clustershell github-page[9].
- Blog about clustershell[10]
- Clush and Cluster Auditing[11]

[6] http://landley.net/kdocs/ols/2012/ols2012-thiell.pdf
[7] https://cloud.github.com/downloads/cea-hpc/clustershell/ClusterShell_User-Guide_EN_1.6.pdf
[8] https://media.readthedocs.org/pdf/clustershell/latest/clustershell.pdf
[9] https://raw.githubusercontent.com/balle/balu-wiki/master/cluster/clustershell.rst
[10] http://www.nexrol.com/clustershell/
[11] https://www.mapr.com/developercentral/code/clush-and-cluster-auditing#.VP1wFnS-VhE