

Erlang - Starting a set of cluster nodes

MICKAËL RÉMOND <mickael.remond@erlang-fr.org>

\$Id: \$

Abstract

This tutorial explains how to configure your cluster computers to easily start a set of Erlang nodes on every machine through SSH. It shows how to use the `slave` module to start Erlang nodes that are linked to a main controller.

Please send remarks, questions and evolution request to mickael.remond@erlang-fr.org

1 Configuring SSH servers

SSH server is generally properly installed and configured by Linux distributions, if you ask for SSH server installation¹.

You need to have SSH servers running on all your cluster nodes.

2 Configuring your SSH client: connection without password

2.1 SSH client RSA key authentication

To be able to manage your cluster as a whole, you need to set up your SSH access to the cluster nodes so that you can log into them without being prompt for a password or passphrase. Here are the needed steps to configure your SSH client and server to use RSA key for authentication. You only need to do this procedure once, for each client/server.

1. Generate an SSH RSA key, if you do not already have one:

```
ssh-keygen -t rsa
```

2. Copy the `id_rsa.pub` file to the target machine:

```
scp .ssh/id_rsa.pub userid@ssh2-server:id_rsa.pub
```

¹The SSH server is sometime called *sshd*, standing for *SSH daemon*.

3. Connect through SSH on the server:

```
ssh userid@ssh2-server
```

4. Create a `.ssh` directory in the user home directory (if necessary):

```
mkdir .ssh
```

5. Copy the contents of the `id_rsa.pub` file to the authorization file for protocol 2 connections:

```
cat id_rsa.pub >> $HOME/.ssh/authorized_keys
```

6. Remove the `id_rsa.pub` file:

```
rm $HOME/id_rsa.pub
```

Alternatively, you can use the command `ssh-copy-id ssh2-server`, if it is available on your computer, to replace step 2 to 6. `ssh-copy-id` is for example available on LINUX MANDRAKE and DEBIAN distributions.

2.2 Adding your identity to the SSH-agent software

After the previous step, you will be prompted for the passphrase of your RSA key each time you are initialising a connection. To avoid typing the passphrase many time, you can add your identity to a program called `ssh-agent` that will keep your passphrase for the work session duration. Use of the SSH protocol will thus be simplified:

1. Ensure a program called `ssh-agent` is running. If it is not started², you can create an `ssh-agent` session in the shell with the `screen` program:

```
ssh-agent screen
```

After this command, SSH actions typed into the `screen` console will be handle through the `ssh-agent`.

2. Add your identity to the agent:

```
ssh-add
```

Type your passphrase when prompted.

- You can list the identity that have been added into the running `ssh-agent`:

```
ssh-add -l
```

- You can remove an identity from the `ssh-agent` with:

```
ssh-add -d
```

Please consult `ssh-add` manual for more options (identity lifetime, agent locking, ...)

²Type `ps aux | grep ssh-agent` to check if `ssh-agent` is running under your userid. Type `ps tree` to check that `ssh-agent` is linked to your current window manager session or shell process.

3 Routing to and from the cluster

When setting up clusters, you can often only access the gateway/load balancer front computer. To access the other node, you need to tell the gateway machine to route your requests to the cluster nodes.

To take an example, suppose your gateway to the cluster is 80.65.232.137. The controller machine is a computer outside the cluster. This is computer where the operator is controlling the cluster behaviour. Your cluster internal addresses form the following network: 192.0.0.0. On your client computer, launch the command:

```
route add -net 192.0.0.0 gw 80.65.232.137 netmask 255.255.255.0
```

This will only works if IP forwarding is activated on the gateway computer.

To ensure proper routing, you can maintain an common `/etc/hosts` file with entries for all computers in your cluster. In our example, with a seven-computers cluster, the file `/etc/hosts` could look like:

```
10.9.195.12 controler
80.65.232.137 gateway
192.0.0.11 eddieware
192.0.0.21 yaws1
192.0.0.22 yaws2
192.0.0.31 mnesia1
192.0.0.32 mnesia2
```

You could also add a DNS server, but for relatively small cluster, it is probably easier to manage an `/etc/hosts` file.

4 Starting Erlang nodes and setting up the Erlang cluster

Starting a whole Erlang cluster can be done very easily once you can connect with SSH to all cluster node without being prompt for a password.

4.1 Starting the Erlang master node

Erlang needs to be started with the `-rsh ssh` parameters to use ssh connection to the target nodes within the slave command, instead of rsh connection. It also need to be started with network enable with the `-sname node` parameter.

Here is an example Erlang command to start the Erlang master node:

```
erl -rsh ssh -sname clustmaster
```

Be carefull, your master node short name has to be sufficent to route from the slave nodes in the cluster to your master node. The `slave:start` timeout if it cannot connect back from the slave to your master node.

4.2 Starting the slave nodes (cluster)

The custom function `cluster:slaves/1` is a wrapper to the Erlang slave function. It allows to easily start a set of Erlang node on target hosts with the same cookie.

```
-module(cluster).
-export([slaves/1]).
%% Argument:
%% Hosts: List of hostname (string)
slaves([]) ->
ok;
slaves([Host|Hosts]) ->
  Args = erl_system_args(),
  NodeName = "cluster",
  {ok, Node} = slave:start_link(Host, NodeName, Args),
  io:format("Erlang node started = [~p]~n", [Node]),
  slaves(Hosts).
erl_system_args()->
  Shared = case init:get_argument(shared) of
    error -> " ";
    {ok,[[[]]]} -> " -shared "
  end,
  lists:append(["-rsh ssh -setcookie",
               atom_to_list(erlang:get_cookie()),
               Shared, " +Mea r10b "]).
%% Do not forget to start erlang with a command like:
%% erl -rsh ssh -sname clustmaster
```

Here is a sample session:

```
mremond@controler:~/cvs/cluster$ erl -rsh ssh -sname demo
Erlang (BEAM) emulator version 5.3 [source] [hipe]
Eshell V5.3 (abort with ^G)
(demo@controler)1> cluster:slaves(["gateway", "yaws1", "yaws2", "mnesia1",
Erlang node started = [cluster@frontal]
Erlang node started = ['cluster@f14-1']
Erlang node started = ['cluster@f14-2']
Erlang node started = ['cluster@bigfoot-1']
Erlang node started = ['cluster@bigfoot-2']
Erlang node started = ['cluster@geronimo-1']
ok
```

The order of the nodes in the `cluster:slaves/1` list parameter does not matter.

You can check the list of known nodes:

```
(demo@controler)2> nodes().
[cluster@frontal,
```

```
'cluster@f14-1',  
'cluster@f14-2',  
'cluster@bigfoot-1',  
'cluster@bigfoot-2',  
'cluster@geronimo-1']
```

And you can start executing code on cluster nodes:

```
(demo@controler)3> rpc:multicall(nodes(), io, format, ["Hello world!~n", [  
Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!  
{[ok,ok,ok,ok,ok,ok], []}]
```

Note If you have trouble with slave start, you can uncomment the line:

```
%io:format("Command: ~s~n", [Cmd])
```

before the `open_port` instruction:

```
open_port({spawn, Cmd}, [stream]),
```

in the `slave:wait_for_slave/7` function.

TODO List

History of this document

2004-01-20: Initial document, MICKAËL RÉMOND <mickael.remond@erlang-fr.org>