04. Running the LUTF

Running the LUTF

Video Transcript

Introduction

Hello, this is Amir Shehata with another quick tip for the LUTF.

Now that we have a general understanding of what the LUTF is, how to build it, deploy it and configure it, let's get our hands dirty and run it.

Environment Variables

There are three environment variables we need to set in order to run the LUTF:

```
export LUSTRE=/usr/lib64/lustre
export LD_LIBRARY_PATH=$LUSTRE/tests/lutf:$LUSTRE/tests/lutf/src
export LUTFPATH=$LUSTRE/tests/lutf
```

- LUSTRE: is the base directory where the Lustre tests are
- LD_LIBRARY_PATH: should include the path for the LUTF shared libraries
- LUTFPATH: which is the path to the LUTF binary

With these three environment variables you can really put the LUTF in any directory and it should work.

What I usually do is I add these environment variables in a bash file, say lutf.sh and then source it when I first log in.

Again, I'm explaining the LUTF from the ground up, so you'll realize a bit later on that a method to set these environment variables automatically comes packaged with the LUTF. We'll cover that when we talk about how the LUTF integrates in the existing Lustre test infrastructure.

For now let's just start the LUTF manually and explore it a bit.

Once you have the environment variables set, you'll need to create a YAML configuration file to give to the LUTF on start up. We went over the YAML configuration file in a previous tutorial. For today you can download a sample YAML configuration file for the agent and the master from the Wiki paged linked below.

Running the LUTF

Alright, now we're ready to start the LUTF:

```
source ~/bin/lutf.sh
$LUTFPATH/lutf --config $LUTFPATH/python/config/lustre01.yaml
```

And Voila! Welcome to the LUTF.

You can see since we're running in `interactive` mode we now have an LUTF python shell we can work with.

Right off the bat let's see what we can learn about our test setup.

Suites

First of all the LUTF gives us a simple way to look at all the existing suites. By using `suites.dump()` you get a list of suites.

```
Welcome to the Lustre Unit Test Framework (LUTF)
lutf>>> suites.dump()
suites:
  - dlc
  - dynamic-discovery
  - lnet-health
  - multi-rail
  - samples
```

If you remember our naming convention we talked about last tutorial, here is where it comes into effect. The LUTF gathers all the suites in the python/tests directory.

Say we add a new suite while the LUTF is up, we can reload the suites by:

```
suites.reload()
```

This comes in handy while actively developing test cases.

Scripts

Second, we can see all the scripts under a specific suite. I'm going to use my samples suite for this demonstration:

```
lutf>>> suites['samples'].dump()
scripts:
  - sample_01.py
  - sample_02.py
  - sample_03.py
  - sample_04.py
  - sample_05.py
  - sample_04_1.py
```

Again while we're actively developing test scripts it becomes immensely useful to reload the test scripts as we add more and more test cases. We don't want to keep exiting and restarting the LUTF whenever we make a change to a script or add a new one. Like with the suites we can call the reload() function to reload all the scripts:

```
suites['samples'].reload()
```

The me Variable

We can also display a bunch of information about this node. There is a `me` variable available which gives us some information about the node we're running on.

We can find out the name given to this node by the test setup:

```
lutf>>> me.my_name()
'RCLIENTS'
```

We can find out the hostname of this node:

```
lutf>>> me.my_hostname()
'lustre01'
```

We can find out the type of the LUTF instance:

```
me.my_type()
```

We can find out the listening port for this LUTF instance

```
lutf>>> me.my_listenport()
8282
```

We can find out the telnet port for this LUTF instance

```
lutf>>> me.my_telnetport()
8181
```

We can also find out the interfaces which this node has:

```
lutf>>> me.dump_intfs()
interfaces:
  eth0:
    ip: 192.168.122.100
    netmast: 255.255.255.0
    broadcast: 192.168.122.255
  eth1:
    ip: 192.168.122.101
    netmast: 255.255.255.0
    broadcast: 192.168.122.255
  eth2:
    ip: 192.168.122.102
    netmast: 255.255.255.0
    broadcast: 192.168.122.255
```

Why is all this interesting? Well because a script can just access the "me" variable and get all the basic information it needs about a node in a very simple way. Not only can a script access the local me variable, but it can also access the remote me variable and get all the info it needs about the remote. The script will look exactly the same in both cases. This makes writing test scripts very straight forward as we will see later on.

The `agents` variable

Beside information about the local node, we can find out who's connected to us. You can do that by:

```
lutf>>> agents.dump()
RCLIENTS1:
  id: 0
  ip: 192.168.122.103
  node-type: AGENT
  telnet-port: 8181
```

You can see that it tells you the name of the lutf instance connected and its type. In this case we're the master and one agent is connected to us.

If I run the same command on the agent, I see this:

```
lutf>>> agents.dump()
RCLIENTS:
  id: 0
  ip: 192.168.122.100
  node-type: MASTER
  telnet-port: 8181
```

Conclusion

That concludes our initial active tour of the LUTF. In the next tutorial we'll start looking at how to write and run scripts.