**Using daemontools to manage processes**

**2013-03-18 00:00:00 -0700**

For client process management, we use both [Monit](http://mmonit.com/monit/) and [daemontools](http://cr.yp.to/daemontools.html). When we require log management in addition to process management, daemontools is one of our front pocket techniques. This post will cover the usage and choice of daemontools and a future monit post is in the works.

**When daemontools shines**

* Applications that die with little or no information in logs
* You need both process **and** log management for your application
* Applications that are overly complicated to setup
* Applications that don't write pidfiles easily

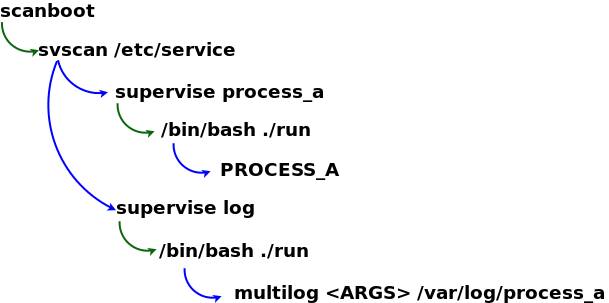
**When monit makes sense**

* Applications that log using syslog or easily rotated files
* Applications that provide api health checks
* You want to limit CPU or memory resource consumption of an application

Daemontools can be installed as package for most Linux distributions.

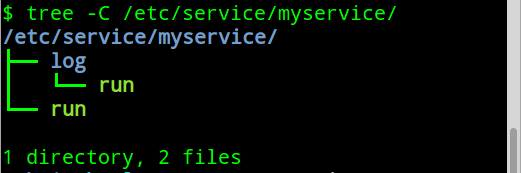
**Processes**

It starts the scanboot process, which in turn starts svscan which looks for service directories. If any of these processes dies they get restarted.



**Directory Structure**

Under the service directory ( /service in CentOs and /etc/service in Ubuntu ) daemontools scans for a service run file ( a simple shell script ) and a multilog log run file ( also a simple shell script ) and ensures that these processes are running and restarts them if they are not. Be sure that both files are executable.



Example run script:

#!/bin/bash

echo "starting logstash"

export PATH=${PATH}:/usr/local/bin:/usr/bin:/usr/sbin

export HOME=/opt/logstash

logger -p local4.err "`hostname -s` logstash[STARTUP]: daemontools started logstash"

cd /opt/logstash

setuidgid logstash /usr/bin/java -jar /opt/logstash/logstash-1.1.9-monolithic.jar agent -f /opt/logstash/config/logstash.conf 2>&1

Tips:

* Use logger to log a startup message to syslog, so you can track the number of application restarts
* Use setuidgid to run the process as a user other than root
* Redirect all output to the console

Example log script:

#!/bin/bash

PATH=/bin:/usr/bin:/usr/local/bin:/sbin:/usr/sbin:/usr/local/sbin:/bin

multilog t n100 s16777215 /var/log/logstash 2>&1

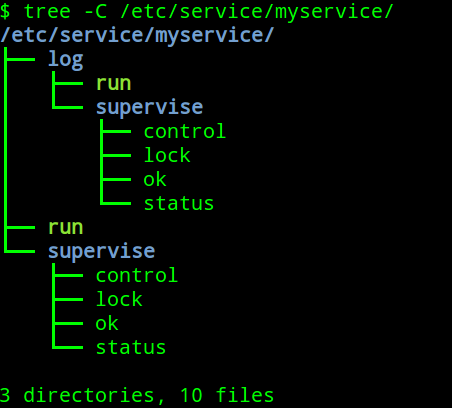
Tips:

* multilog option t adds a timestamp to the beginning of each line
* multilog option n100 keeps a maximum of 100 logfiles rotated automatically
* multilog option s16777215 sets the maximum allowed logfile. This is a limit imposed by multilog

**Pro Tip:**

Make your service tree in /tmp, test it, then copy it into the correct location to insure it runs correctly.

If your process starts sucessfully you should see extra supervise directories in your service directory tree.



**Managing Processes**

Status:

$ sudo svstat /service/logstash

/service/logstash: up (pid 14368) 16763 seconds

Starting:

$ sudo svc -u /service/logstash

Sending a kill which will restart but only if the process responds to signals:

$ sudo svc -k /service/logstash

Or just kill the process and let daemontools restart it.   
  
Stopping a process from restarting then killing it automatically:

$ sudo svc -k /service/logstash

$ sudo killall -u logstash java

**Managing logs**

Multilog will log into the directory you specified and create a link current that will handle logfile rotations.

Multilog uses a special timestamp format, but includes a simple decoder. While this might sound like an extra step, it come in very handy with different timezones in a global organization.

#Box is set to GMT

$tail -f /var/log/logstash/current | tai64nlocal

2013-03-19 02:09:05.347785500 starting logstash

#View same log line in Pacific time

export TZ="right/US/Pacific"

$tail -f /var/log/logstash/current | tai64nlocal

2013-03-18 19:08:40.347785500 starting logstash

You may notice files with names like @400000005147bcd13302dc5c.s, you can also use tai64nlocal to decode these to human readable dates

$ ls \*.s |tai64nlocal

2013-03-19 19:27:44.620628500.s

2013-03-19 19:29:06.620077500.s

2013-03-19 19:30:29.933865500.s

Daemontools and its close cousin [runit](http://smarden.org/runit/) are both excellent tools for keeping your processes running at all times. The initial upfront investment of time is well worth the cost to avoid downtime from crashes and uncaught exceptions, as well as the timestamped log trail to ensure you know exactly what happened and when.