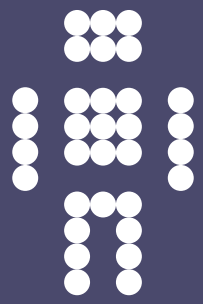
CFEngine



Scheduling and Event Management

A CFEngine Special Topics Handbook

CFEngine AS

CFEngine is able to schedule processes or jobs across all managed nodes, in a platform independent manner. This eliminates the distributed configuration of time-shedulers like cron, and allowed the design of custom calendars trivially with CFEngine's built in logic.

Moreover, CFEngine can respond to the occurrence of any distributed pattern in behaviour or data on the network and bring immediate countermeasures to bear. Thus full location-time scheduling and response is supported.

Table of Contents

What is scheduling?

Scheduling refers to the execution of non-interactive processes or tasks (usually called 'jobs') at designated times and places around a network of computers. On a given computer, jobs might be run sequentially in a queue, one after the other, or they might be run in parallel. Jobs can also be started by triggers when sensors see certain system activity. CFEngine supports a full range of features for customizing hands-free scheduling.

How can CFEngine help?

CFEngine is able to schedule processes or jobs across all managed nodes, in a platform independent manner. This eliminates the distributed configuration of shedulers like cron. The time resolution for this depends on the frequency with with the cf-agent is scheduled to run. A normal recommendation is that cf-agent runs every 5 minutes, which is sufficiently often for most batch scheduling requirements.

Define jobs with basic profile information

Jobs are scheduled using CFEngine as `commands` promises. To determine the conditions under which a job should be promised one uses `classes`.

```
bundle agent batch_jobs
{
commands:

  # Always run job on these three hosts

  host1||host2||host3::

    "/usr/local/bin/my_special_job $(sys.host)"

       comment => "Run the cluster task for this host";
}
```

To limit the job to a special time, we use time-classes:

```
bundle agent batch_jobs
{
commands:

  # Run job on all hosts at 13:05pm

  Hr13.Min00_05::

    "/usr/local/bin/my_special_job $(sys.host)"

       comment => "Run the cluster task for this host";
}
```

To combine, location and time coordinates, simply join the classes:

```
bundle agent batch_jobs
{
commands:

  # Run job on all hosts at 13:05pm

  (host1||host2||host3).Hr13.Min00_05::

    "/usr/local/bin/my_special_job $(sys.host)"

       comment => "Run the cluster task for this host";
}
```

Chaining jobs together

Creating a managed process by chaining jobs together is also done using classes. To chain jobs into a sequence, you simply set a class if when the predecessor completes, and predicate the antecedant on that class:

```
bundle agent order
{
commands:

 # Dummy jobs to illustrate chaining

 Monday.Hr12.Min30_35::

  "/bin/echo Job one" classes => if_else("success","failure");

 success::

   "/bin/echo Next job";

 failure::

  "/bin/echo Error condition?";

}
```

Calendars

You can define classes based on any combination of events in CFEngine, and in this way build up special calendars.

```
bundle agent jobs
{
classes:
```

```
  "holiday" or => {
                  "July.Day4",
                  "May.(Day1|Day2|Day3|Day4|Day5|Day6).Monday",
                  "December.Day25"
                  };

commands:

  !holidays::

    "/usr/local/bin/my_special_job $(sys.host)"

      comment => "Run the cluster task for this host",
      action  => if_elapsed("240");
}
```

Avoiding weekends is a simple matter, as is testing to see if the target system fulfills the requirements for the job:

```
bundle agent batch_jobs
{
classes:

 "weekend" expression => "Saturday|Sunday";

 "have_update_db" expression => fileexists("/usr/bin/updatedb");

commands:

 (host1||host2||host3).!weekend::

   "/usr/local/bin/my_special_job $(sys.host)"

      comment => "Run the cluster task for this host every six hours",
      action  => if_elapsed("240");

 have_locate_db.Hr01::

   "/usr/bin/updatedb"

      comment => "Update the locate db at 1 a.m. each night, if exists",
      action  => if_elapsed("240");
}
```

Here are some other calendar ideas:

```
classes:

"LunchAndTeaBreaks" expression => "!(Saturday|Sunday).(Hr12|Hr10|Hr15)";
```

```
"NightShift"        or => { "Hr22", "Hr23", "Night" };

"ConferenceDays"    or => { "Day26", "Day27", "Day29", "Day30" };

"TimeSlices"        or => { "Min01", "Min02", "Min03", "Min10_15"
                            "Min33", "Min34", "Min35" };

"Exception"         not => "Hr12.Min15_20";
```

Logging execution

There are many ways to log events in CFEngine.

```
bundle agent test
{
commands:

 "/usr/local/myjob"

     action => logme("myjob");
}


body action logme(x)
{
log_repaired => "/tmp/private_$(x)_keptlog.log";
log_string => "$(sys.date) $(x) promised job succeeded";
}
```

This results in a log file '`/tmp/private_myjob_keptlog.log`' which contains data of the form:

```
Sat Aug 22 11:11:01 2009 myjob promised job succeeded
Sat Aug 22 11:11:01 2009 myjob promised job succeeded
```

Scheduling by Sensing Events and Patterns

Any measurable event on a system can trigger a response from cf-agent.

```
bundle agent test
{
commands:

  special_event::

    "/usr/local/open_help_ticket args";
}
```

For example, the monitoring agent `cf-monitord` sets system classes based events that are classified as anomalies on the system, as well as custom defined observations.

Working with Unix `cron`.

One of CFEngine's strengths is its use of classes to identify systems from a single file or set of files. This allows you to have a single, central CFEngine file which contains all the 'cron' jobs on your system without losing any of the fine control which cron affords you.

One way to achieve this is to set up a regular cron job on every system which executes `cf-agent` at frequent intervals. Each time `cf-agent` is started, it evaluates time classes and executes the shell commands defined in its configuration file. CFEngine's time classes are much more powerful than `cron`'s time specification possibilities, and they add control over location too.

> DO I NEED TO USE CRON? No. With CFEngine's `cf-execd` you don't need to use cron at all – CFEngine can schedule itself. Whether you choose to run `cf-execd` in daemon mode, or in wrapper mode is entirely up to you.

Commands promises

CFEngine commands promises have the general form:

```
promise-type:

      time-based classes::

         Promise
```

For example:

```
bundle agent example
{
commands:

# Exec during every first quarter-hour after noon

  Hr12.Q1::

    "/path/myscript -arg1 -arg2";

# Exec during any second quarter-hour

  Q2::

    "/path/otherscript";

# Exec during the intervals 00:10 through 00:15 and 12:45 through 12:55
# (English says ‘‘and’’, but logic says ‘‘if this interval or that is true’’
```

```
  Hr00.Min10_15||Hr12.Min45_55::

    "/path/amongstourscripts";

}
```

If you want to get fancy, you can set parameters for the execution of the script by building a container for it that traps its output and privileges (this applies to root only, since only root has this power to change privilege).

```
bundle agent example
{
commands:

# Exec on the first quarter after noon

  Hr12.Q1::

    "/path/myscript -arg1 -arg2",

          contain => my_custom_jail("nobody","true");
}

# ...

body contain my_custom_jail(owner,devnull)
{
exec_owner => "$(owner)";     # run with this setuid
no_output => "$(devnull)";    # like > /dev/null 2>&1
umask => "77";                # set process umask
}
```

The '`contain`'ment body provides a safe and flexible environment in which to embed scripts.

The time resolution of the classes is limited by how often you execute CFEngine either using cron or `cf-execd`. Five minutes is the recommended scheduling interval.

Splaying host times

In a network of thousands of computers, many agents could start executing and downloading resources from a server at the same time. For instance, if a thousand cf-agents all suddenly wanted to copy a file from a master source simultaneously this would lead to a big load on the server. We can prevent this from happening by introducing a time delay which is unique for each host and not longer than some given interval; `cf-execd` uses a hashing algorithm to generate a number between zero and a maximum value in minutes which you define, like this:

```
body executor control
```

```
{
splaytime => "10"; # Minutes
}
```

Choosing a scheduling interval

How often should you call your global CFEngine configuration? There are several things to think about:

- How much fine control do you need? Running cron jobs once each hour is usually enough for most tasks, but you might need to exercise finer control for a few special tasks.
- Are you going to verify the entire CFEngine configuration file or just selected promises?

CFEngine has an intelligent locking and timeout policy which should be sufficient to handle hanging shell commands from previous crons so that no overlap can take place.

Appendix - Did you know?

Here are some features that can help you with CFEngine's scheduling capabilities:

- Batch jobs can be made to run in parallel by using `background => "true"` in an `action` body.
- You can limit the frequency with which a batch job is executed with or without specifying an actual time of execution using the `ifelapsed` settings. Then a job will only be started if a certain number of minutes have elapsed since it was last started.
- You can make sure that jobs have not crashed or run out of control using the `expireafter` settings. Then a job that seems to have been running for too long will expire after a defined number of minutes and will be killed and restarted.
- The monitor agent `cf-monitord` can watch over special processes and monitor their resource usage, e.g. memory or CPU usage and report on these in CFEngine Nova.
- You can arrange for jobs to run in a 'sandbox' or 'jail', running as a special user, in a special directory without access to system resources. Thus system security can be addressed when running foreign applications.