CFEngine 3

Code Freeze 2012

Jan 12, 2012

Brian Repko

http://www.learnthinkcode.com
Agenda

- Introduction to CFEngine
  - What it is
  - How it works
- Examples
- New Machine Setup
- CFEngine vs Chef vs Puppet
IANAS (sysadmin)

- My role has been as a developer / architect working with sysadmins to deploy our applications
- When automated, we have used cfengine
- This presentation will use cfengine configuration from Entagen (triplemap.com) for our examples
- Thanks to Frans Lawaetz!
CFEngine

- Systems Maintenance Tool
  - Configures and automates large-scale IT infrastructure
  - Ensures the availability, security and compliance of systems and applications
- Created by Mark Burgess (Ph.D in Theoretical Physics)
  - CFEngine 1 – released in 1993 (at Oslo University / post-doc)
  - CFEngine 2 – released in 1998 (Computer Immunology)
  - CFEngine 3 – released in 2008 (Promise Theory)
- Two editions – Community (FOSS) and Nova (commercial)
  - Nova has portal, reporting, native MS Mac/Windows mgmt, high-availability, virtualization, KM, commercial support
- Written in C, runs on all types of systems including mobile, tablet and embedded systems
- Tons of low-level functions and action configurations to reuse
Who uses it

- AMD
- Facebook
- Chevron
- AT&T
- TRUSTe
- Cisco
- Bloomberg
- FedEx
- eBay
- Shell
- PayPal
- Ernst & Young
- IBM
Components

- cf-execd
  - Cron-like lightweight daemon that runs cf-agent (default is every 5 minutes)
  - More sophisticated than cron (randomness, fail-safe operation, time exclusions)
- cf-agent
  - The main component that does the system automation and maintenance (maintaining promises)
  - Runs cf-promises (another executable) to validate bundles/promise syntax
- cf-serverd
  - Distributes policy and data files (cf-agent pulls from cf-serverd)
- Hubs (Nova only), Servers and Clients
  - Can run standalone as well
  - All systems run the same configuration policies (Hubs/Servers are Clients too!)
  - Top-level system(s) get their files from version control (please say yes)
- Promises (statements)
  - Describes the desired state of some aspect of a system (file, process, package...)
- Bundles (collections of promises, aka policies)
  - promises.cf as bootstrap policy – says what other bundles to execute
- Variables and Classes (CFEngine defined and User defined)
  - Classes are booleans that allow control over which promises are executed and when
Bundles

- Bundles are typically designed to target a particular set of systems and/or a particular objective.
- Bundles have a type (which executable) and name:
  - “common” is for all executables
  - Each bundle type has a control (configuration) promise body
- Each bundle is run three times in order to get each promise fulfilled (convergence).
- Bundles have sections (“promise type”) whose order is fixed based on the executable.
- promises.cf, failsafe.cf and update.cf are “standard”
- cfengine_stdlib.cf has lots of the low level functionality.
- The “common control body” specifies the files to process and the bundle sequence – specified in promises.cf (or standalone files).
Bundles

• Bundle promise type order (for agent)
  • vars – define variables
  • classes – define classes
  • outputs – (Nova) data collection
  • interfaces – network interfaces
  • files – files and file contents
  • packages – packages / libraries to install / uninstall
  • environments – (Nova) virtualization
  • methods – process other bundles
  • processes – processes that should / should not be running
  • services – collections of processes
  • commands – commands to execute
  • storage – disks and filesystems
  • databases – (Nova) LDAP, SQL and Windows Registry
  • reports – reporting on cfengine state
Variables and Classes

- Variables
  - Special Variables (CFEngine defined)
  - Scalar types – string, int, real
  - List types – slist, ilist, rlist
  - Scalar expression ${var}$, List expression @${var}$
  - Scalar expression of a list is a loop / for–each

- Classes
  - Booleans that guard promises
  - CFEngine–defined (aka hard) Classes
  - User–defined (aka soft) Classes
  - Often set by using built–in functions
  - Can also be defined by promise repair or failure
  - Combined with not ('!'), and ('.' or '&'), or ('|' or '||')
Promises

- Within its promise type
  class-expression::

    “promiser” [→ {“promisee1”,...}]
    attr1 => value1,
    attr2 => value2, ...
    attrN => valueN;

- Promisee is only used by Nova
- Promise body is all the attributes
- Promise body may reference body-parts
- Many common attributes, others based on type
Promises

- File promise – edit_line bundle
  - Ability to handle file edits
  - Templates or line changes
- Promise Body Parts
  - Reusable setup of promise attributes
  - Can take parameters / variables
  - Can use classes internally
- Functions
  - Lots of built-in functions
  - Many functions return a class (boolean)
Example 1 – Java Setup

• Review tm-java.cf (a portion of tm.cf)
• Goal is to have Oracle Java (JDK) as java
• No RHEL repo for it
Example 2 – DB, Tomcat, HTTP

- Review tm-server.cf
- All install as packages
- All run as services
- Configuration and setup
  - Httpd configuration as template
  - Restart httpd if configuration changed
  - Tomcat to run as non-root user
  - Tomcat context copied in
  - MySQL db creation as idempotent(?)
Example 3 – Web App Deploy

- Review tm-webapps.cf
- Web applications are built with Jenkins
- Jenkins jobs configure which hosts get which builds
  - Save war with .<host> appended to name
- CFEngine needs to know when to pick up the wars (and not catch them in the middle)
  - Trigger file with <host> name
- Some applications require special files
- And then unzip everything and restart Tomcat
New Machine Setup

- Review Entagen Triplemap Demo script
  - Quick script to bootstrap Triplemap Demos
  - Builds <x>.triplemap.com
  - Machine setup with KVM
- Some VM support in Nova (I've not seen it)
- Have seen knife (Chef) used to bring up CFEngine policy servers and clients
  - And then CFEngine to setup Puppet
  - And then Puppet to setup Chef
- http://blog.afistfulofservers.net/
Challenges to CFEngine

- Learning Curve
  - Syntax and flow in general
  - No “standard style”
  - Global vs Local on variables / classes
  - Reserved names
  - Timing / Locks

- Large Ecosystem
  - Tons of variables, classes, functions
  - Start small! Collaborate!
  - Code review files in VCS!

- Be careful – you are root everywhere at once!
  - cf-promises will verify syntax – it will not prevent stupid – test your files!
  - Use test.cf and limit its execution
  - Run on a virtual machine
  - Run your tests at least twice on a given machine
Goofus and Gallant

"Watch me make this fast cf3 change!!"

"Friends, I have an idea for a new bundle, let's discuss it and then test it."

"Nobody's around, I'll just take this GE promise out and see what happens."

"Jean, Eric, what do you think about these proposed updates to the GE promises?"
CFEngine vs Chef vs Puppet

- Chef and Puppet are both Ruby-based
- CFEngine is C
  - Runs on anything (Nova) and is fast and lightweight (memory footprint)
  - CFEngine scales very well (AMD running 10k clients per server)
- Commercial support available for all
  - Is “only in Nova” driving people away from CFEngine?
- Execution repairs itself (failsafe/update) – don't know for Chef/Puppet
- CFEngine language is “odd” – learning curve for all
- CFEngine file editing capabilities seem strong (lots of body parts in stdlib)
- CFEngine will be more verbose
  - but everything is available and easily tweakable (vs “Puppet defines the how”)
  - to an extent...promise types are written in C
- CFEngine based on heuristics / Promise Theory vs Graph / Dependencies / Model
- Lots of sharing of Chef recipes and Puppet scripts – not so much with CFEngine
  - Community Open Promise Body Library but that's it...
- Puppet a reaction to CFEngine 2 (Luke Kanies) – CFEngine 3 is a rewrite
- Chef a reaction to Puppet
References

• CFEngine
  • http://cfengine.com/getting-started

• Vertical SysAdmin (Aleksey)
  • http://www.verticalsysadmin.com/blog

• Sample WordPress setup
Q/A – Thank You!