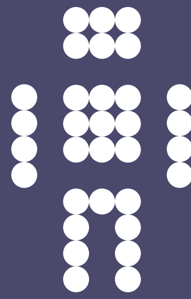
CFEngine



The Vision

A CFEngine Special Topics Handbook

CFEngine AS

This is an internal note that intends to provide a quick summary of CFEngine concepts.

Copyright © 2010 CFEngine AS

What is the vision?

The CFEngine vision is a set of dreams, goals and principles that guide us in the development of the software. A separate vision statement exists for the company.

Slogan: 'Assured infrastructure, connecting the dots between Business and IT.'

From goals to promises

Everything we do at CFEngine follows the promise model. Promises offer a consistent language for expressing desired state, compliance, agreements, services, and practically anything else. It is a language that focuses on knowledge and assurance. Promises also provide a measuring stick against which to measure or assess systems. Promises-kept is a simple nmeasure of compliance.

> CFEngine's attention to knowledge and assurance aims to connect the dots between your business goals and a set of verifiable promises.

Note: the term 'knowledge' should not be overused when talking to clients, as they find the term scary. We should look for better alternatives.

The Primacy of Knowledge

CFEngine began as a tool for *assuring* a system state or configuration. Assurance is about wanting to *know*. Users want to know that state of the system is okay, that the system has certain properties that we have decided comply with policy.

For the past 15 years, we have been concerned with the mechanics of how to ensure this desired state, but now CFEngine has solved this problem for most important cases, and we are turning to the issue of *knowledge retention* and *system comprehension*.

> Scalability is limited not only by machine resources but by our comprehension. If we can't understand a system, then it is not under control.

Use of Patterns

Patterns are the essence of *information compression* and *comprehension*.

- We typically think we understand something when we see how it falls into a pattern.
- If we have a pattern, we don't need a list of instances. Replacing instances with a shorter pattern is what data-compression is about. Patterns are therefore 'cheap'.
- Every pattern is based on a model of the possible instances, so it is about 'model-based' or 'policy-based' management.

The complexities of a real environments throw up many instances. Sometimes it can seem that there are more exceptions than general cases. Many vendors try to simply by over-simplifying – discouraging specialization. Alvin Toffler wrote about this the latre 1960s, in connection with industrialization and automation. He opposed the view that automation would lead to mass production of identical instances (like the people in Chinese Communism):

> 'As technology becomes more sophisticated, the cost of introducing variations declines' (Alvin Toffler, Future Shock)

i.e. Automation implies more variety, not less. This is because we can deal with special cases more cheaply by exploiting the generalities first. Most of the information can be made general, and special cases just require us to model *context*.

CFEngine models context using 'classes' (not Object Oriented classes, but 'classifications' of the environment in the manner of an ontology/taxonomy of state). This is knowledge management. Most vendors only know how to do this in a procedural, imperative of 'scripted' approach. They re-image or 'baseline' systems so that they have a known context, and the rebuild delta-changes from that. CFEngine does not need to do this, because it starts with a model of the end-state, not the start-state.

An example that we have used to good effect is the simplest pattern: a list. Most system administrators a familar with ACLs (Access Control Lists). We can turn many configuration issues into two lists:

- What we want.
- What we don't want.
- Everything else we don't care about (tolerated variation)

Virtualization

Virtualization is a way of using patterns to replace many cases with a single interface. It is about hiding ugly reality behind a more congenial user experience, so it is a recognizable 'pattern'.

The IT industry will soon close the loop on computing. IBM began with its mainframe computers, designed to scale for business needs by providing extra capacity on demand. It's virtualization allowed multiple customers to run in segregated environments. The past 40 years have been an effort to escape from a mainframe architecture to a commodity version that can scale in the same way, with the same assurances.

Tomorrow's operating system will be more seamless, a virtual interface to an underlying diversity of devices.

CFEngine has been about virtualization from the beginning – hiding the differences between underlying operating systems

Scalability

Scalability refers to the capacity for a system to grow in size without losing functional efficiency. Specifically it describes the ability of a system to deal with increasing volume of input, and the efficiency with which it produces output. The implication is that it should be cheap to increase the volume of processing, but design issues such as bottlenecks usually intervene. There are two approaches for scaling:

- Increases rate of processing (throughput) of each part of the system (especially its weakest links).
- Increase the parallelism in the system (non redundant processing).

System administrators tend to see limitations only in technology. However, humans are often the weakest link. Scalability requires:

- The ability to comprehend the system as it grows (human):
  - Promised properties and behaviours (Functions)
  - The ability to perceive the actual properties and behaviours (visualize)
    - The ability to grow system processing (automation).

  Most technologies have no strategy for helping humans to comprehend. Menu-based systems try to simplify by taking away control from the user, but this only leads to distrust and frustration for experts.

  CFEngine's model does not take away control, but manages the amount of information that an end user has to deal with, by using design-patterns and automated analysis to reduce the cost for the end user.

    - Policy editor with syntax aware interface (for extensible language input)
    - Content-driven policies (fixed spreadsheet approach to input)
    - Knowledge Map (for browsing policy and documentation)
    - Monitoring interface and reporting engine (for browsing system state)