



Maintenance and Repair

Cfengine Technology Insight

Mark Burgess, Cfengine AS

Self-repairing systems have long been viewed as Science Fiction, but Cfengine has been researching and building self-healing automation now for 15 years. With an estimated million computers all over the world relying on cfengine for installation and update, hands-free management is a reality. Now Cfengine Enterprise Editions bring professional support and enhancements for the world's most demanding environments.

This technical white paper describes the Cfengine vision of self-healing automation.

DISCLAIMER: Under no circumstances will any part of the Cfengine company be held responsible for errors or omissions in this document.

Copyright © 2009 Cfengine AS

Table of Contents

The afflictions of IT management	1
The myths of IT management.....	1
From re-building to surgical repair	2
How does cfengine work?	4
Least Common Denominator Thinking	5
Future Utility Computing and Clouds	5

The afflictions of IT management

Computer systems fail with astonishing regularity. Despite advertised improvements to each new generation of operating system, an arms race persists between fault and repair, and it never seems to abate. It really doesn't matter whether the source of faults is human error, the rapid pace of change, attack from the network, or simply increased organizational complexity: the result is that datacentre engineers are caught in a poverty trap of fire-fighting, with their hands tied by regulatory compliances and poor tools. As writer Alvin Toffler wrote in his seminal 1970 study *Future Shock*:

"Faced by relatively routine problems, ["Man"] was encouraged to seek routine answers. Unorthodoxy, creativity, venturesomeness were discouraged...performing mindless routine tasks in response to orders from above, ["Man"] must [now] assume decision-making responsibility – and must do so within a kaleidoscopically changing organizational structure built upon highly transient human relationships."

Even back in the 1960s, Toffler saw that the industry of the day was caught in a rut, trapped by a lack of vision brought on by sheer force of habit and fear of new methods of industrial automation. It needed to empower individuals to benefit from their individual capabilities, and adapt to the increasing pace of change and development. Today we see organizations repeating these patterns of behaviour in IT management.

Toffler challenged the popular idea that introducing industrial automation de-humanized people in the workplace, by countering that it was rather forcing humans to repeat senseless tasks that was dehumanizing. At Cfengine, we too challenge industry traditions and advocate automation. We have spent the past 15 years developing the kind of automation methods that bring creativity and agility back to the IT workplace, and which respect human expertise.

The myths of IT management

Many IT management solutions today bind humans needlessly to system monitoring applications, forcing them to spend valuable time watching screenfuls of data to see when errors occur. Trouble tickets are opened for every minor detail and trained engineers are equipped with pagers and cell-phones so they can respond to alarms about even the most trivial of issues.

Many such alarms are avoidable, with proactive maintenance, yet few solutions today fully automate regular system maintenance so that human time is not wasted. Instead we put humans to work for machines, rather than the other way around. The value of separating tasks between humans and machines is being squandered.

The reasons for this are not hard to see. There are two myths that continue to dog the industry:

- *Only the changes we decide to implement ourselves are important ('roll-out' is the method of choice)*

We are basically in control of our environment and we don't need to take the idea of external changes seriously. We don't make changes very often, so automation is non-essential. We decide, then we push out and conquer the datacentre by steam-rolling systems into compliance.



- *Version control can reverse any system errors ('roll-back' is the answer)*

Any faults occur when we make changes and can therefore be handled by reversing bad decisions. We just need tools to help us reverse bad decisions once in a while to keep us safe from ourselves.

Unfortunately both of these ideas are incorrect. In fact all machines are connected to an environment of unpredictable users by terminals and networks. Unless you were able to control every user, you really don't control your system's environment. Systems also have many components created by third parties who provide updates and make changes on the fly, outside of our control. In short, not all changes are a result of planned change. Systems are changing all the time, and automated repair is the only chance we have to keep systems in balance.

If we do make poor change decisions, it is also not necessarily true that those changes can be reversed, because not all of the changes that occur on a system are a direct result of decisions that we make. We don't necessarily know what other changes have occurred. Ultimately one has two choices: live with the mistake and move forward, or destroy a machine and re-build it.

The latter requires human intervention and results in an interruption of service and significant destruction of run-time data. In many companies (search-engines, analysis companies, transaction processors) run-time data are the source of a company's revenue model. Destructive re-building would mean a significant business loss, and a manual interruption.

Organizations have to look for new and improved ways of avoiding the kinds of high-risk mistakes that necessitate rebuilding. The answer is not trying to back out of a train-wreck, it is avoiding the event in the first place. The answer to these problems is pro-active maintenance.

Indeed, the industry has placed the entire burden of fault incidents on humans, causing stress and wasting time. Today, system automation is mature enough to handle most prevention issues and incidents. The answer is (as Toffler pointed out in 1970): when then technology is powerful enough, we can make small individual changes cheaply. This is the Cfengine way.

From re-building to surgical repair

Cfengine is about *predictable* Desired State Management. The Cfengine software operates on computers with surgical precision and in real-time.

Many organizations have no way to automate the implementation minor alterations to systems, nor repair systems without manual intervention. If a system is sick or a change is needed, it is 'put down' and a new one is built with a new design to take its place.

Re-imaging systems, however, is a destructive process that interrupts service and costs time and resources. It is tantamount to demolition and rebuilding. But if we need to change a light-bulb we do not tear down a building and create a new one – we make a surgical repair.





Would you do this to change a light-bulb?

Why is this method still in use? Remarkably little research has been performed on technologies that can perform pin-point surgery on systems. The research that has been done comes from the world of Unix, where innovation has almost been built into the Unix-derived operating systems from the beginning.

Conventional industry solutions to management only embellished the demolition methods with 'process management' bureaucracy: tear down a system and re-build it – as long as you follow procedure, they think that is okay. Indeed, if maintenance were an extremely rare event and the cost of repairing were large, re-imaging could be the cheapest solution. But the need for maintenance is not rare, and downtime is very expensive. Maintenance is a continual requirement, and the faster the pace of change, the more repairs we need to make.

MAINTENANCE: You decide to paint your house white, but you live next to a freeway and in the desert. You 'roll-out' your white paint-job and don't expect to change the colour for another 5 years. But the sun shines and the cars fly past, and before a few months are up, the surface is covered in soot and sand, and the walls need washing. In a couple of years the sun has reacted with the paint and soot and caused blemishes to form, and you need to repaint. You only planned on the major changes in colour that you wanted to make, but you forgot that the environment you live in dictates the pace of change, and does not always respect your plans.

How does cfengine work?

How do we know that Cfengine can maintain a system's desired state on the fly? This follows from Cfengine's convergent technology for 'computer immunology' that was developed for specifically Cfengine over the past ten years. To understand how it works, look at the figure on the next page.

The two images below represent the approaches used by Cfengine and conventional technology. On the left hand side, the conventional approach to change implementation is shown. Beginning at a well known location (base-camp), and following a set of directions takes you to the desired state (the summit); i.e. you start from a well-understood baseline state, such as 'out of the box', and follow a set of procedures for building until the desired state is reached. If nothing unexpected happens along the way, the system is configured once (and once only) to this condition.



Baseline and recipe

Convergence to end state

The main problem with this approach is that the recipe can only work once. If we deviate from the plan in any way, or if the system wanders off the desired state, the instructions are useless for finding our way back. This is why conventional systems have to start from scratch and rebuild everything to fix the smallest error. Repeating the original recipe would potentially take us somewhere altogether different and actually break the system.

The Cfengine approach turns this upside down (see the right hand side of the figure). We go from a mountain climbing problem to a valley descent. No matter what the starting state of a system (whether 'out of the box', or a long-running legacy system) Cfengine associates target policy with the base of the valley (or fancifully, an irresistible black-hole) and draws the system towards that single place. Every operation that Cfengine performs has this property of 'convergence towards desired state'.

Now, if the system becomes damaged somehow, or if regular maintenance falls behind, so that the actual state is not the desired state, Cfengine doesn't care about the reason, it simply repairs the flaw and draws the system back into to the bulls-eye. This is straightforward as all paths now lead back to the goal. This property is built in at a fundamental level.

How is this possible? The promise language used by cfengine makes it simple to declare policy intentions in a form that clearly define the desired state in simple engineering terms at a low level. From this, Cfengine is able to compute a path to reach the state, no matter what the starting point. In other words, cfengine relinquishes the implementation technology to the

back room, out of sight and leaves only the final goals or intentions in plain sight. By working bottom-up rather than top-down, it is able to avoid obstacles that prevent a repair recipe from working.

Using Cfengine a policy team can craft promises to surgically configure and repair systems matching any convenient pattern, dealing with issues such as: file security attributes and configuration content, running processes, database structure, storage, batch jobs, garbage collection etc, in other words: all of the basic properties of an IT system.

Least Common Denominator Thinking

IT vendors have long perpetuated the myth that the only way to achieve predictable systems is to make them all identical. The reason is clear: the cost of building variations according to the classical demolition and image-build approach is huge. In the management of the information technology itself, we are still hearing about "ways to mass produce 1000 workstations all identical from a common source" – golden master servers that are to be copied by hundreds, perhaps thousands of clones. Convergent, surgical configuration control explodes this myth once and for all.

In his study of industrial production, Alvin Toffler pointed out that automation does not necessitate mass produced identical copies – a world in which one can have any colour as long as it's black. On the contrary, he argued that *"As technology becomes more sophisticated, the cost of introducing variations declines."*

At Cfengine we hear again and again, 'what you claim is impossible – it sounds like science-fiction' and yet we have been doing this for 15 years already. Maintaining systems is not fundamentally difficult, rather there has been no industry incentive to move away from the lucrative alarm systems that hold back progress in the field.

Future Utility Computing and Clouds

Today the industry is talking about 'Cloud Computing', or Utility Computing – how to make the management of IT systems disappear into the walls leaving only a simply control panel that anyone might operate. The future of IT management lies in achieving this level of simplicity. But we do not have to outsource our IT to a provider to achieve this. Powerful and simple 'self-healing' technology can bring this simplicity to all organizations and make the best use of their local expertise and knowledge to accelerate progress and reduce the cost to humans and businesses alike.

Want to understand how cfengine works in more depth? Log on to the the Tech Corner at the cfengine.com web-site or go to the USENIX ;login: magazine and see *Configuration Management, Models and Myths*.



