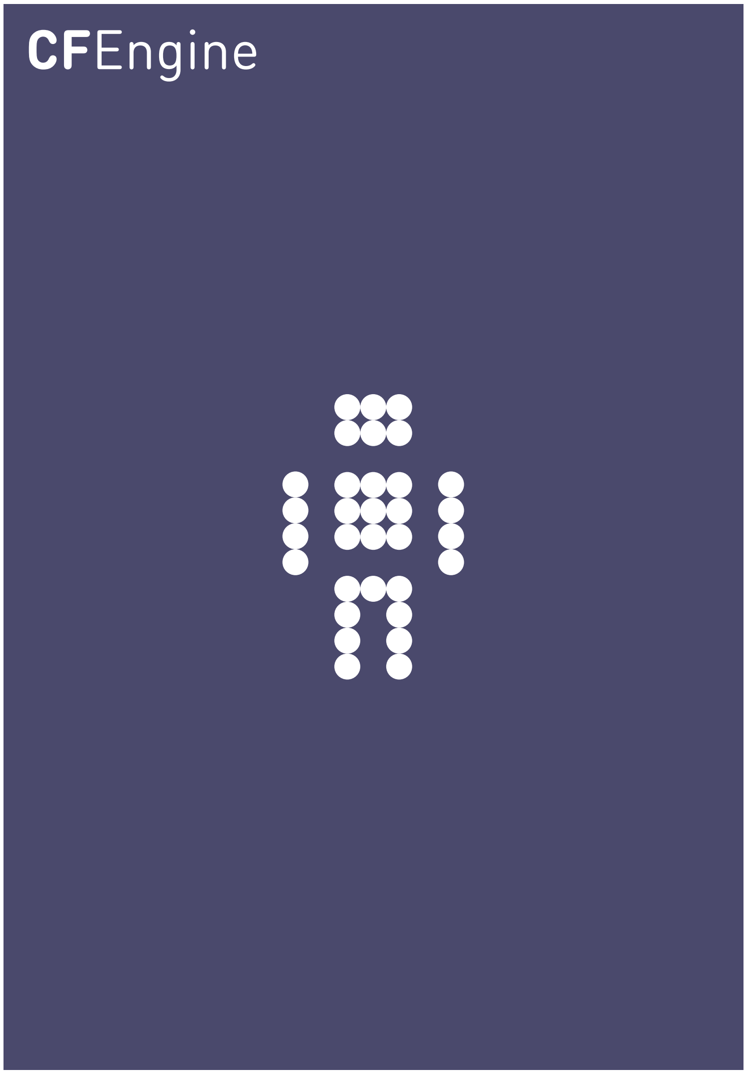


BDMA

A CFEngine Special Topics Handbook

CFEngine AS

Build, Deploy, Manage, Audit is a simple and traditional model of the IT infrastructure lifecycle, based on human workflows and processes. CFEngine's approach to the IT infrastructure is somewhat different, but the four pillars of the lifecycle can still be addressed in the framework of automation. This guide explains how to think about the IT infrastructure lifecycle when using CFEngine.

Table of Contents

What is BDMA?

The four mission phases are sometimes referred to as

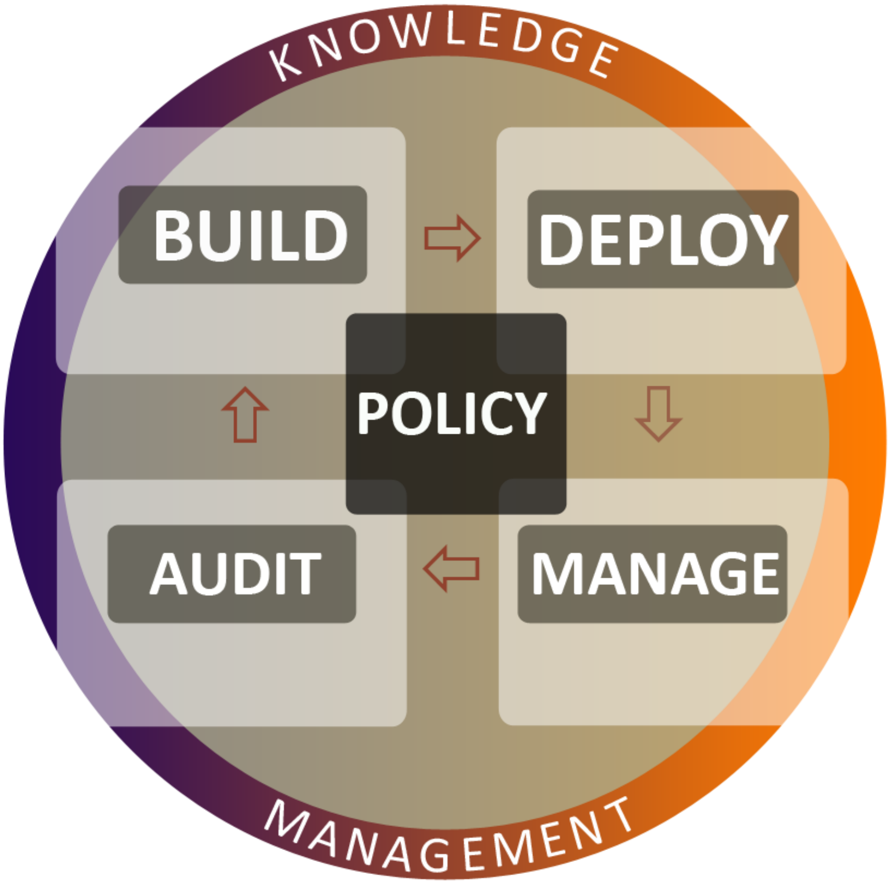
Build A mission is based on decisions and resources that need to be put assembled or 'built' before they can be applied. This is the planning phase.

In CFEngine, what you build is a template of proposed promises for the machines in an organization such that, if the machines all make and keep these promises, the system will function seamlessly as planned. This is how it works in a human organization, and this is how is works for computers too.

Deploy Deploying really means launching the policy into production. In CFEngine you simply publish your policy (in CFEngine parlance these are 'promise proposals') and the machines see the new proposals and can adjust accordingly. Each machine runs an agent that is capable of keeping the system on course and maintaining it over time without further assistance.

Manage Once a decision is made, unplanned events will occur. Such incidents traditionally set off alarms and humans rush to make new transactions to repair them. Under CFEngine guidance, the autonomous agent manages the system, and humans only manage knowledge and have to deal with rare events that cannot be dealt with automatically.

Audit CFEngine performs continuous analysis and correction, and commercial editions generate explicit reports on mission status. Users can sit back and examine these reports to check mission progress, or examine the current state in relation to the knowledge map for the mission.



Stem cell hosts

At CFEngine we talk about stem cell hosts. A stem cell host is a generic foundation of software that is the *necessary and sufficient* basis for any future purpose. To make a finished system from this stem cell host, you only have to 'differentiate' the system from this generic basis by running CFEngine.

Differentiation of hosts involves adding or subtracting software packages, and/or configuring the basic system. This strategy is cost effective, as you do not have to maintain more than one base-line 'image' for each operating system; rather, you use CFEngine to implement and maintain the morphology of the differences. Stem cell hosts are normally built using PXE services by booting and installing automatically from the network.

Recommendations for Build

There are many approaches to building complete systems. When you use CFEngine, you should try to progress from thinking only about putting bytes on disks, to planning a long term set of promises to keep.

- What services do you want to support?
- What promises do you want to keep concerning these services?
- Are these promises sustainable and convergently implementable?

- Formulate proposed intentions in the form of CFEngine promises.
- Discuss the impact of these in your team of CFEngine Mission Specialists (more than one pair of eyes).

It is worth spending extra time in the build planning to simplify your system as much as possible. A clear formulation here will save time both in maintenance and training later, as employees come and go. The better you understand your intentions, the simpler the system will be.

We cannot emphasize enough the value of the promise discipline. If you can formulate your requirements as promises to be kept, you have identified not only what, where, when and how, but also who is reponsible and affected by every promise.

Building systems is resource intensive. CFEngine works well with **rPath**, allowing optimized build that can shave off many minutes from the build time for machines. CFEngine can then take over where rPath leaves off, performing surgically precise customization.

Recommendations for Deploy

Deploying a policy is a potentially dangerous operation, as it will lead to change, with associated risk. Side-effects are common, and often result from incomplete planning. (See the CFEngine Special Topics Guide on *Change Management*).

The following sequence forms a checklist for deploying successful policy change:

1. Discuss the impact of changes in the team.
2. Commit the changes to promises in version control, e.g. subversion.
3. Make a change in the CFEngine input files.
4. Run the configuration through '`cf-promises --inform`' to check for problems.
5. Move the policy to a test system.
6. Try running the configuration in dry-run model: '`cf-agent --dry-run`'
7. Try running the policy once on a single system, being observant of unexpected behaviour.
8. Try running the policy on a small number of systems.
9. Construct a test environment and examine the effect of these promises in practice.
10. Move the policy to the production environment.
11. If possible, test on one or a few machines before releasing for general use.

CFEngine recommends a process of many small incremental changes, rather than large high-risk deployments.

CFEngine allows you to apply changes at a much finer level of granularity than any package based management system, thus it complements basic package management with its deployment and real time repair (see next section).

Recommendations for Manage

Managing systems is an almost trivial task with CFEngine. Once a model for desired state has been created, you just sit back and watch. You should be ready for 'hands free' operation. No one should make changes to the system by hand. All changes should follow the deployment strategy above.

All that remains to do is wait for email alerts from CFEngine and to browse reports about the system state. In CFEngine Nova, these reports are generated automatically and integrated into the system knowledge base.

Most email alerts from CFEngine are information only. It is possible (but not recommended) to make CFEngine very verbose about its operations. It is common to look for confirmation early in the phase of adopting CFEngine, as trust in the software is building. Eventually users turn off the verbosity and the default is for CFEngine to send as little email or output as possible.

Consider a single line E-mail, in confirmation of a change, arriving from 1000 computers in a single day. Learning to trust the software saves unnecessary communication and needless human involvement. The Nova Mission Portal makes notification and alerting largely unnecessary.

Recommendations for Audit

Auditing systems is a continuous process when using CFEngine Nova. Report data are collected on a continuous and distributed basis. These data are then collected from each distributed location according to a schedule of your choosing to collate and integrate the reports from all systems.

The reports CFEngine provides are meant to offer simple summaries of the kind of information administrators need about their environment, avoiding unnecessary detail[1].

Available patches report
Patches already installed on system if available.

Classes report
User defined classes observed on the system – inventory data.

Compliance report
Total summary of host compliance, all promises aggregated over time.

File_changes report
Latest observed changes to system files with time discovered.

File_diffs report
Latest observed differences to system files, in a simple diff format.

Hashes report
File hash values measured (change detection).

Installed patches report
Patches not yet installed, but published by vendor if available.

[1] Reporting intelligence is a major area of development in CFEngine, and the next commercial edition CFEngine Constellation will include a generic queryable semantic web of the integrated system, with correlation analyses and more. The capabilities for reporting grow exponentially as one progresses from Community Edition to Nova to Constellation.

Installed software report
Software already installed on system if available.

Lastseen report
Time and frequency of communications with peers, host reliability.

Micro-audit report
Generated by CFEngine self-auditing. This report is not aggregated.

Monitor summary report
Pseudo-real-time measurement of time series data.

Performance report
Time cost of verifying system promises.

Promise report
Per-promise average compliance report over time.

Promises not kept report
Promises that were recently un-kept.

Promises repaired report
Promises that were recently kept by repairing system state.

Setuid report
Known setuid programs found on system.

Variables report
Current variable values expanded on different hosts.

Summary BDMA workflow

1. Define a stem cell host template
2. Set up PXE network booting and kickstart / jumpstart OS tools with CFEngine integrated
3. Get CFEngine running and updating on all hosts, but making no system changes.
4. Define a service catalogue.
5. Discuss and formulate a policy increment, thinking convergence at all times
6. Publish (deploy) the policy.
7. Follow emails and reports in the CFEngine Knowledge Map (Manage).
8. Adjust policy if necessary, following procedures for change management (Manage)
9. View reports (or enjoy the silence) to audit system state.

CFEngine works well with package based management software. Users of rPath, for example, can achieve substantially improved efficiency in the build phase. CFEngine takes over where package based systems leave off, providing an unprecedented level of control 'hands free'.

CFEngine®