Cfengine 3 指导手册

Cfengine 公司工作手册

Mark Burgess Cfengine 公司

目录

1 系统自动化

1.1 无缝无形地管理多样化复杂的系统环境

未来从不遥远。在未来，智能化的计算设备无处不在，我们的梦想已经慢慢开始实现。今天，智能化的操作系统已经嵌入计算机和手机，如 Linux 和 Windows 。施乐研究中心的 Mark Weiser 曾写道:

"最有深远意义的技术是那些似乎消失的技术。因为它们已经进入到每天生活中的方方面面，融为一体，直到无法区分出来。"

今天很多人在谈论云计算，并认为它是梦想的另一种显现，那时计算机无处不在，更确切的说计算机覆盖了世界上所有的数据中心，不只是办公室和家里。这是让计算机使用进入我们所需要领域的一个方面。任何技术的基础是需要有能够执行大量准确配置的工具。Cfengine 就是这样的一种工具。

在任何的系统环境中，Cfengine 可以为整个系统的生命周期进行可扩展的配置管理 。几乎所有其他的系统配置都设想有一个稳定可靠的网络并且能够从一台权威的主服务器至上而下的展开更新配置。这些系统在如下的环境中是没有用的：

- 带有局部的或者不可靠的网络连接的移动系统（例如，潜水艇）
- 网速很慢的计算机系统（例如，卫星或航天探测器）
- 计算机电量很低的计算机系统（例如，移动感应器或厨房设备）

Cfengine 不需要可靠的基础设施。它几乎可以在任何环境中运行，并且不耗资源，没有软件依赖性。 因此，它不仅可以在所有的传统计划的情况中运行，并且能够在完全移动的配置中运行，如临时的事故办公室，漂流的潜水艇，卫星或机器人探测车。

有人会反对说：“好吧，但我不需要这样的系统, 因为我的网络很可靠。”然而，你的网络并不是你想的那么可靠，并且网络的移动性是一个越来越重要的话题。即使是一个非常冗余的网络，支持网络的那些设备也可能因为其中任意一台的设备中断而陷入系统瘫痪的状态。在现代普遍的系统环境中，系统能够持有可行性，容错性，并尽可能的独立于外部条件限制，是非常重要和关键的。这就是如何建立可扩展的，可靠的服务。

Cfengine 可以在你所认为的各个地方工作，甚至所有你还没有想过的新地方运行。我们如何能知道？因为它是基于 20 多年的细心研究和实践。

1.2 期望管理—— 一种承诺理论

在系统管理中最困难的事情之一是让每一台计算机明白自己的角色和任务，并接能够依靠其他的计算机执行相同的任务 。信任既省钱又省时间。如果你不信任，你必须去证实，这样的代价是很高的。

为了增进信任我们许下承诺。承诺是旨在以某种方式来行动表现的文件证书。不管它们是机器还是人类，我们需要学习承诺来信任系统。

一个 Cfengine 的用户曾经对我说，Cfengine 基于自主合作的设计曾经最大帮助他配置 Cfengine ."我们的主要问题不是技术问题，而是政治问题，即要让世界上每一个部门的每一个人都统一意见 。" 对于所有的技术而言，这是因为那些做决定的人需要感觉到系统授权于他们，而不是限制他们。

Cfengine 是以简单的承诺理念来工作的。我们可以认为 Cfengine 中的所有一切是一个通过系统不同资源来维持的承诺。

将承诺和形式结合起来描述承诺应该实践的时间和位置，这就是所谓的 Cfengine.

1.3 为什么自动化？

人类擅长于做决定，却不能很好的去执行。而机器不能做决定，但可以很好的去执行。如果人类和机器能够相互取长补短就会很有意义。

在系统管理中的主要问题是缺乏自律。纪律并不是意味着要按照重要的指示来有序执行。它只要求系统的每一部分能够知道它的工作并且能够完整无误的执行。

有技术的工作者易于认为它已经足够的智能化。事实是这是错的：聪明的人易于成为问题解决者并且很开心能够解决相同的问题很多次，这样其实耗时又耗力。此外，人为的干预常常是基于慌张和迷糊，所以每次有人手动登入系统，其实这样有害于每个人对系统的理解。只有稳定章程的自律才能带来可预测性。

零散的更改是不好的，因为：

- 其他人不知道发生了什么。
- 没有记录更改的内容和目的。
- 只留下了更改的痕迹。

人们经常愤怒反对自动化，因为觉的自动化让他们工作失去了人性。事实却相反，强迫人不断重复稳定去做机器做的工作，这才是丧失人性。能够认识到这一坏的习惯并且愿意放弃这一坏习惯就是一大进步。

1.4 升级换代

在过去，让系统升级换代的唯一方式就是让所有系统相同。这不再是正确的方式。

在 1960 年的末期，兼有新闻工作者和未来主义者身份的 Alvin Toffler 描绘了一幅关于西方世界和后工业化未来的美丽惊人的图画。他在 1970 年写的一本书叫《未来冲击》，真实的回应了冷战时期对于在一个共产主义工业阶段大量生产同类产品的恐慌。他的书真实的反驳了那些认为工业化和大量生产就意味着一切都应该是一模一样的人们。我推荐大家去阅读这本书，的确是写的非常好并且有很多今天我们需要学习的经验教训。但是从他冗长的抨击中，我写下一句简单的话来总结我们需要学习的教训：

“当技术越来越先进，潜在的风险代价也越来越小。”

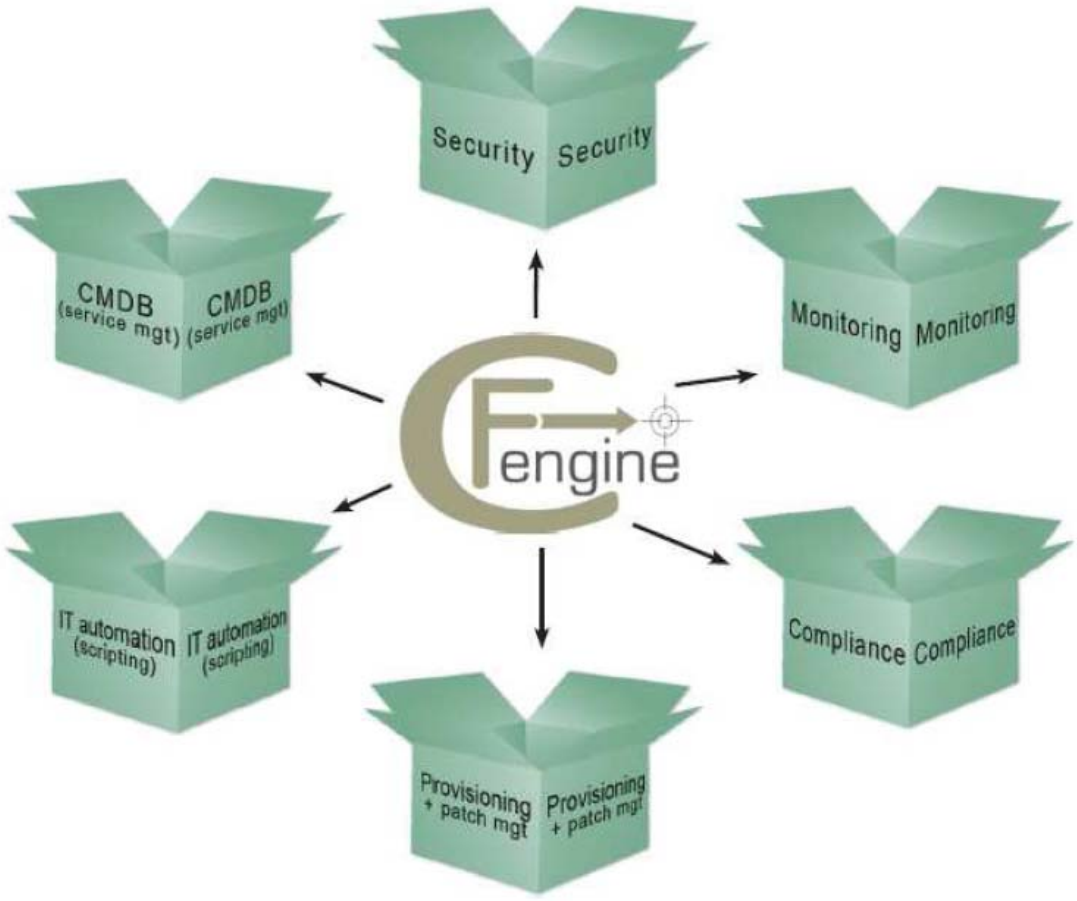
换句话说，用于大量生产的任何中流技术都能帮助我们更加娴熟和灵活，而不是更少。当你可以在机场的 ATM 机上按要求打印出名片，并且可以在眨眼功夫让咖啡杯变的个性化，

在这样一个时代里我们没有理由去坚持一个荒诞的理念：认为大规模的基础设施就要求完全的重复。然而人们仍然这么认为，包括网络工程师，系统管理员。他们甚至说这是可扩展性必不可缺的。

Toffler 的一个重要观点是大量生产化经济与适应性经济并不冲突，并且在 40 年之后，我们仍然在吸取这个教训。

1.5 你如何看待 Cfengine ?

Cfengine 是一个构架。它并没有如此的复杂，但它的确很广泛。经常当试图描述 Cfengine 的时候,似乎有很多要说的并且很难用一个简单方式来表达这个软件能做什么。下面这幅图能够给你一些思路去认识 Cfengine 。



对许多用户来说，Cfengine 是一个简单的配置工具，例如，根据策略来配置和修补系统的软件。策略是通过承诺来描述的，事实上，在 Cfengine3 里的每一个声明都是一个将在某时某地兑现的承诺。除此以外，然而 Cfengine 不像大多数自动化的工具一次性展开一些软件的蓝图并且抱有乐观的希望。你在 Cfengine 里所许下的每一个承诺都是不断被检验和维护。它并不是一次性的操作，而是一个封装的进程，它能够自我修复那些偏离策略的行为。

无疑把 Cfengine 放在一个自动化的王国里，经常会有问题提出：难道它只是另一种脚本语言吗？ 不可否认 Cfengine 确实包含了一种强大的脚本语言，但是它不像其它种语言。Cfengine 不像 Perl ,Python or Ruby 是一种低级语言。它是一种承诺的语言，你可以用它来表达对系统高层次的期望和实现结果所需要的算法机制的内部细节。我们将会在下面再次提到。

对于许多人来说，Cfengine 是一个用来在系统上执行安全稳定的步骤并且在以后进行不断监测的工具。这肯定是一个主要的应用方面。Cfengine 以稳定和安全留下很好的口碑。这是因为它的基本设计是安全的。它不可能从系统外发送有关策略的信息到 Cfengine . 如果被允许进入系统，它只可能发送一些大小有限的简单的协议请求到服务器。这样的设计比大多数防火墙更安全。大多数服务器不能进行安全测试，因为测试有可能会发送数据到服务器。

能够描述一个系统的大多数策略的能力就是意味着我们所建议系统所需要许下的和遵行的承诺。因此 Cfengine 也可以被认为是一个承诺引擎。它可以容易地服从一些框架模型，如 SOX ,'EUROSOX'(第 8 条欧盟资料指引), ITIL 的架构和像 ISO1 7799,ISO 200000 的标准等等。

最后，虽然 Cfengine 不是最初设想用来监测的，但它包含了最灵活和省开销的监测引擎。你可以提取关于系统配置，用法，资源，记录文件的数据，并且将它转化成可读的报告。Cfengine 发现和提取系统信息和报告的能力意味着你可以将系统转化为一个简单的配置管理数据库。在免费版本中，监测是一个还未实行的进程。随着 Cfengine 的商业扩展，对你所作的监测承诺种类几乎没有限制，并且不会像大多数监测系统耗尽全部的资源。

总而言之，Cfengine 目的是促进人类对复杂进程的理解。它的承诺易于提供文档，并且在错误报告中系统会记住并提醒我们。它隐藏了暂时无关的执行细节从而突出承诺背后的目的。这意味着你公司的知识可以编入 Cfengine 的语言中。

为什么知识很重要？其中有两个原因： 第一就是技术性的描述很难被记忆。只有当你在写的时候你才有可能理解你的配置决定，但当几个月后出现一些问题，你很可能已经忘记你当时所想的。这就花费了你的时间和努力来诊断其中的问题。第二个原因就是公司易于依赖那些编写策略的人。一旦他们离开， 剩下的人往往不能能够理解和修复系统。只有完整正确的文档，系统才有可能免于受损。

2 Cfengine 的组件

Cfengine 是由大量的组件构成的。在这章，我们将讨论如何组建它们并解释它们是干什么用的。

2.1 安装

要安装 Cfengine ，你需要一些软件包。如下：

OpenSSL 开源加密的安全套装协议

网址：http://www.openssl.org

BerkeleyDB (3.2 版本或更新的版本)

轻量的平面文件数据库系统

网址：http://www.oracle.com/technology/products/berkeley-db/index.html

其他...

建议使用 Perl 语言兼容的正则表达式（PCRE）静态库，因为这是对最标准的可移植性操作系统接口静态库的一大改进。这个文档是采取 PCRE 的使用。

在 Windows 系统中，为了保证 Cfengine 正常运行，你需要安装基本小型的 Unix 模拟环境（Cygwin）动态链接库。

网址：http://www.cygwin.com

如果有其他的静态库，如 OpenLDAP, MySQL 的用户机静态库 和 PostgreSQL 等等，另外的功能（其中有一些需要付费使用）也是可用的。没有这些静态库 Cfengine 也能够运行，只是相关的功能将不可用。

除非你已经购买了现成的源代码，或者在正在使用一个发行的包，你需要编译 Cfengine . 如果这样你也同样需要一个已经装有 gcc , flex 和 bison 的系统环境。

推荐的安装方法如下：

```
tar zxf cfengine-x.x.x.tar.gz
cd cfengine-x.x.x
./configure
make
make install
```

这样源代码就安装在`/usr/local/sbin' 目录下。

2.2 工作目录

Cfengine 有一个可以使用的工作空间目录。当以 root 身份运行，它的默认位置在`/var/cfengine' ，而目录~/.cfagent 是留给其他用户使用的。

```
/var/cfengine
```

/var/cfengine/bin
/var/cfengine/inputs
/var/cfengine/outputs

一个可信任的输入文件缓冲区目前必须被放置在`inputs' 子目录下。当 Cfengine 自动被激活，按照预期它就会只读取这个目录下的文件。在每台机器上更新缓冲区是用户的工作（这是由默认配置文件决定的）。

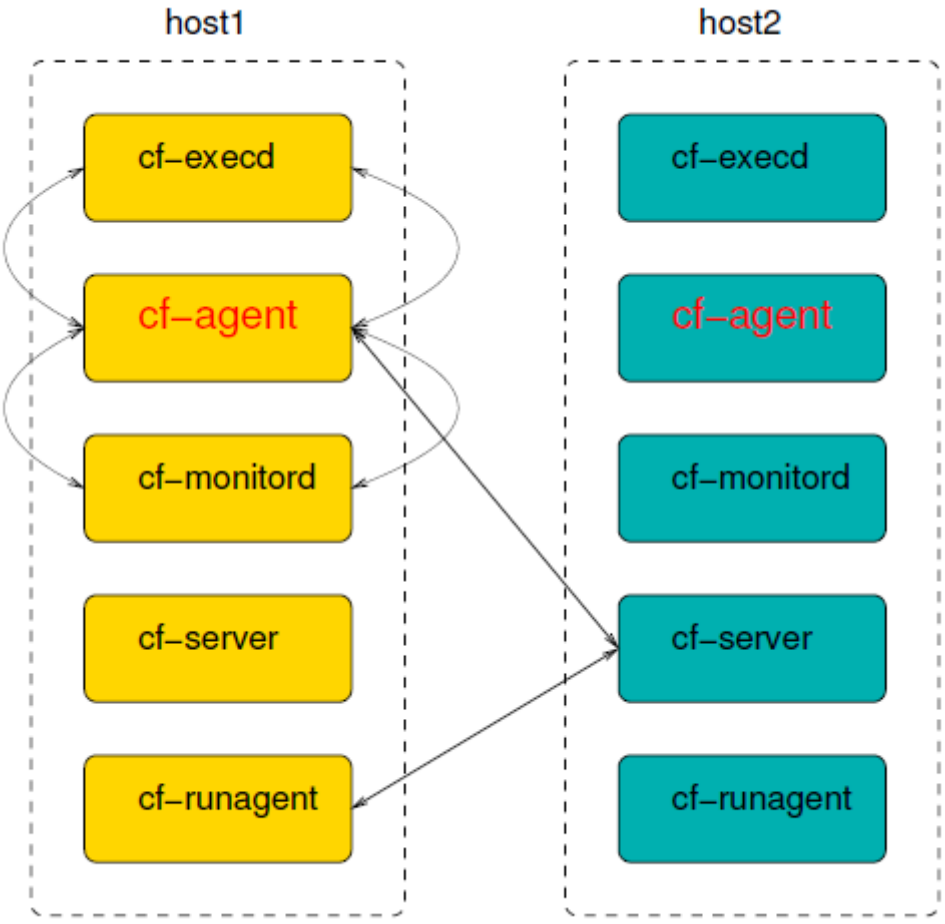
不像 Cfengine 2，Cfengine 3 不能识别 CFINPUTS 环境变量。

现在'outputs' 目录是一系列运行报告的一个记录 。这些往往通过 cf-execd 发邮件给管理员，或者被复制到其他重要的位置并且可被相应的浏览器阅读。

2.3 演奏者

一个 Cfengine 系统有点类似一个管弦乐队。它由任意数量的计算机（演奏者）组成，每一个人都有自己的乐谱并且知道弹奏什么。有无指挥者协调整合每一部分，这是由你自己决定的。

Cfengine 的软件代理在每一台独立的计算机上运行，但在必要情况下可以进行通信，如下图所示。这就意味着你没有必要为了运行你的网络而规范标准以防不安全的登录，如果网络通信出现了问题，在运行中断期间 cfengine 就可以如需修复和保护系统 。



如果网络不工作， cfengin 就会跳过这些部分并且继续它的工作。它具有容错性和多选择性。

cf-promises 承诺的检验者和编译者，在试图执行一套配置承诺前预先检测。

cf-agent 变化的发动者，代理是 Cfengine 操作系统资源的部分。

cf-serverd 服务器能够共享文件并且接受在一台个体计算机上执行当前策略的请求。它不可能从外部发送（推出）新信息给 Cfengine .

cf-execd 这是一个安排日程的后台程序（它能补充或取代 cron）.它也可以作为包装器，执行和收集 cf-agent 的输出，并且发送邮件给有必要的系统用户。

cf-runagent 这是一个帮助者程序，它能够对话 cf-serverd 并要求执行 cf-agent 上的当前策略。如果代理机上的策略包含了检验更新，它因此模拟为一个发动者，对 cfengine 的主机产生变化。

cf-report 它能得出总结和其他报告，以多种格式输出或与其他系统整合。

cf-know 这个代理能够从大量关于系统知识的承诺中产生一个 ISO 标准主题图。它可作为语义网显示文档。

2.4 关于 Cfengine 的架构

这部分解释 cfengine 将如何在你的引导下在网络中自动化的操作。如果你的公司有很大的规模（成千上万的服务器），当这样的规模要求你有更多的基础设施和一个潜在的更复杂的配置，你该花一些时间跟 Cfengine 专家讨论如何将这样的描述调谐到你的系统环境中。任何 cfengine 配置的实质是相同的。

在管理系统中，有四个共同被引用的阶段，总结如下：

- 安装
- 配置
- 管理
- 审核

这些独立的阶段来自于系统管理的一个模式，它是基于事务处理的变化。Cfengine 的管理理念有点不同，因为事物处理的进程对于系统管理来说不是一个好的模式，但我们可以用这个模板去看看 cfengine 如何不同地工作。

安装 一个系统是基于大量的的决定和需要在执行前被安装的资源。建立一个系统的可信任的根基是引导它配置的关键所在。你不需要决定每一个细节，就足以在你的系统中建立信任和良好的可预测性。

在 cfengine 中，你安装的是一个为公司的计算机所推荐的承诺模板，以致如果所有计算机都能做出和保持这些承诺，系统将如期流畅的运行。这就是它如何在人群组织中工作，并且如何为计算机服务。

配置 配置确实是意味着执行已经决定的策略。在事务处理系统中，一台机子试图一个接一个地推出更新，因此需要部署这个决定。在 cfengine 你很简单的发布了

你的策略（在 cfengine 的语法中这些是‘承诺提议’）并且这些计算机看见了新的提议并且能相应的调整。每一台计算机都运行着一个能够随着时间执行策略和维护它们的代理，而无需进一步的协助。

管理 一旦做了一个决定，出乎意料的事将会发生。这种突发事件通常会引起警觉，并且人们会匆匆忙忙的做出新的事务处理来修复它们。在 cfengine 中，自动化的代理能够管理系统，并且你只需要处理那些不能被自动处理的罕见事件。

审核 在传统的配置系统中，在一次通过的事务处理后输出往往不够明确，所以有人来审核系统以便找出问题所在。在 cfengine 中，不只是变化会被发起一次，而且本地的审核和维护也是如此。Cfengine 的设计并不能保证和自动化地维护决定输出，所以主要的担忧是管理冲突的意图。用户可以休息并且检查代理产生的日常编译报告，并且没有必要安排新的备份事务处理。

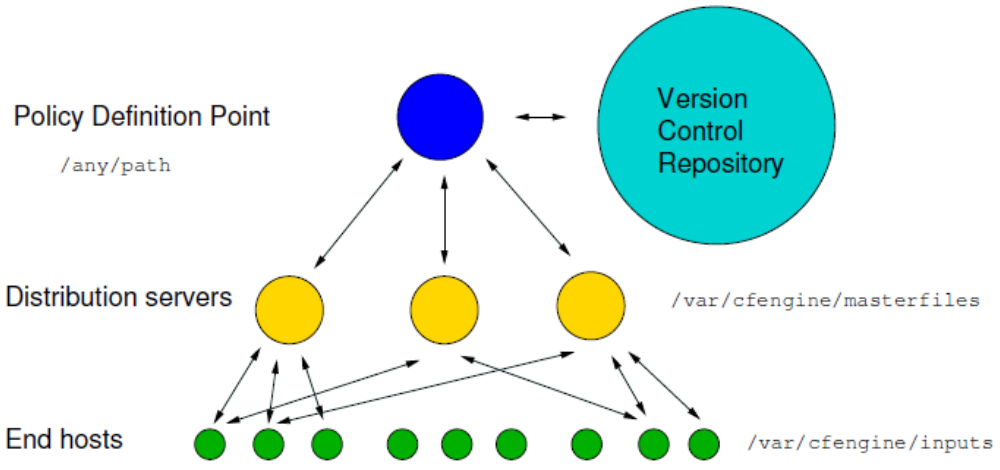
备份和恢复？ 你不应该认为 cfengine 是一个备份系统，例如，有人试图阻止系统绝对的变化并很可能恢复它们以防出错。备份和恢复是在理论上有瑕疵的概念，只是有时在实际中可行。通过 cfengine ,你发布了一系列的策略修改，总是在往前进步（因为不管喜不喜欢，时间只是在流逝）。所有被期望状态的变化是可以通过每一台独立的计算机在本地管理，并且不断的修复以保证进行中的编译与策略一致。

2.5 策略决定流程

Cfengine 没有做出非常多绝对的选择。它的大多数行为是策略的问题，并能够被改变。然而，推荐使用一个结构，见下图。

为了保证操作尽可能简单，cfengine 在每台计算机上保持一个私有工作目录，即在文档中的 WORKDIR，在策略中以变量$(sys.workdir)的形式出现。这个目录的默认位置是在'/var/cfengine'和' c:\var\cfengine' . 它包含了 cfengine 需要运行的一切。

下面的这幅图显示了决定如何在系统之间流动。



- 协调合作的单一点是很有意义的。因此通常单一的本地（策略定义中心）做了决定。你选择一个的版本控制系统（例如，SubVersion ,CVS,等等）可以跟踪决定和变化的历史记录。
- 通过编辑 cfengine 的策略文件 'promises.cf ' （或者它包括的子文件）可以做出决定。这个过程是在非运行状态下执行的。
- 一旦决定被定形并编码了，新的策略会被手动（人为决定）地复制到一个决定分布点，默认是位于所有策略分布服务器上的目录'/var/cfengine/masterfiles '中。
- 每一个客户机联系策略服务器并且下载这些更新。如果客户机的数量很庞大，我们可以复制这个策略服务器，但我们将认为那里还是只有一个策略服务器。

一旦客户机有一个策略的备份，它只从中提出与之相关的承诺提议，并且在没有人为帮助下执行任何的变化。这就是 cfengine 管理变化。

为什么这么做？Cfengine 试图通过分离进程来减少依赖性。通过遵行基于拉的架构。Cfengine 将能容忍网络中断，并很容易从配置错误中恢复。通过把决定的责任重担放在最顶上，然后把执行放在最底下，我们可以避免无需的脆弱，并保持两个独立的质量保证进程分开。

2.6 从免费版本开始

升级到 cfengine 的商业版本是一步到位的操作，这在它们各自的文档中已经描述了。以免费版本开始的最快方法是把安装在`/usr/local/share/cfengine/'的分布的策略文件复制到策略分布点，如下：

1. 选择你的策略服务器。
2. 成为那台服务器的超级用户或者管理员。
3. 创建策略源目录：

```
host# mkdir -p /var/cfengine/masterfiles
host# cp /usr/local/share/cfengine/*.cf /var/cfengine/masterfiles
```

4. 现在启动系统

```
host# /usr/local/sbin/cf-key
host# cd /var/cfengine/masterfiles
host# /usr/local/sbin/cf-agent --bootstrap
```

现在 Cfengine 在你的服务器上运行，它将做些什么？只是一到两句简短的话语，所以人们知道寻找什么（或知道那是安全的）

```
host# ps waux | grep cf-
```

你应该浏览在目录`/var/cfengine/masterfiles' 的文件，看看它们包含了什么，甚至在启动前做一些修改。在其它事之间，你想要自定义一些参数，就像你站点中‘resolv.conf ’的那些

参数。如果你以前已经使用了 cfengine 的早期版本，这些文件的内容将不会看起来如此的神秘。至于其他，为了有一个概观请继续的关注。

注意：如果你已经为 cfengine 的工作目录手动配置了一个不同的位置，你将需要调整，用你已经配置的路径来取代‘/var/cfengine’。例如，基于包的 Debian 感觉‘/var/lib/cfengine’对于它来说就是正确的位置。

3 如何执行和测试一个cfengine策略

你不需要超级用户的权限来使用 cfengine .以普通用户可以安全地测试大多数的试验。你应该在准备开始配置系统以前花一些时间来试验小的例子。为了这么做你应该以常规无特权的用户登录到你的系统并开始配置：

```
host$ /usr/local/sbin/cf-key
host$ cp /usr/local/sbin/cf-* ~/.cfagent/bin
```

Cfengine 想要看看在工作目录中它的源代码的备份。对于一个常规用户来说，这个通常位于`~/.cfagent' ，而不是在`/var/cfengine'。你应该现在准备好了。

3.1 Hello World

下面是在 cfengine3 中最简单的‘Hello world’程序:

```
# 每个策略必须有一组续发事件
body common control
{
bundlesequence => { "test" };
}
#
bundle agent test
{
reports:                #这个是承诺类型 This is a promise type
cfengine_3::            # 这个是类环境（这个承诺只能在 cfengine3 系统实行）
"Hello world";      # 这是一个简单的承诺(它产生了一个写有"Hello world"的报告)
}
```

在一个文件中输入这个，例如`emacs ~/test.cf' 。然后检查语法如下

```
/usr/local/sbin/cf-promises -f ~/test.cf
```

如果都是正确的，那将没有任何的输出。现在执行如下：

```
/usr/local/sbin/cf-agent -f ~/test.cf
```

你应该看见这个：

```
R: Hello world
```

这个‘R’：告诉你这是个来自于一个报告的输出（和记录文件‘L’或者一些嵌入程序的带引号的输出‘Q’相反）。

这不是一个经典的 cfengine 程序，首先因为 cfengine 并不是通常地意味着输出消息，除了特例情况下。然而作为一个初级使用者，它是便于更好的看到一些输出。

如果你立即重复相同的命令，将什么也不会发生。但如果你等上一分钟，它将会再次的工作。以详细的模式（使用-v 或者--verbose ）运行命令来看看是什么原因：

/usr/local/sbin/cf-agent --verbose -f ~/test.cf

现在你将看到 ：

```
cf3> ========================================================
cf3> reports in bundle hello (1)
cf3> ========================================================
cf3>
cf3> XX Nothing promised here [lock.hello.reports..Hello_worl] (0/1
minutes elapsed)
cf3>
```

这告诉你 cfengine 认为重复指令的时间太短了而不需要再次保持承诺。这个时间是由参数 ifelapsed 决定的，它可以分别为每一个承诺单独的设置。你也可以使用'-K'选项让 cfengine 忽略这些时间的限制。

在'Hello world '语句之前，你看见了类表达`cfengine_3::'。这就是 cfengine 如何做决定。如果条件满足，那么输出信息的承诺将会执行。为了明确这个类是否正确来执行，你可以键入命令来看看详细的输出。你将会看到像这样的一些输出：

```
Defined Classes = ( any verbose_mode Tuesday Hr08 Morning Min48
Min45_50 Q4 Hr08_Q4 Day7 July Yr2009 Lcycle_2 GMT_Hr6 linux atlas
undefined_domain 64_bit linux_2_6_27_23_0_1_default x86_64
linux_x86_64 linux_x86_64_2_6_27_23_0_1_default
linux_x86_64_2_6_27_23_0_1_default__1_SMP_2009_05_26_17_02_05__0400
compiled_on_linux_gnu localhost_localdomain localhost net_iface_lo
net_iface_wlan0 ipv4_192_168_1_100 ipv4_192_168_1 ipv4_192_168
ipv4_192 fe80__21c_bfff_fe6e_70ef cfengine_3_0_2b4 cfengine_3_0
cfengine 3 SuSE lsb_compliant suse suse_n/a suse_11_1 suse_11
agent )
```

例如，一系列目前已定义好的类。任何其中的一个类（或一个组合）本能够被用来标识承诺。这就是 cfengine 所指的哪种场合保持哪种承诺的方法。补充参考见章节 4.4。

最后记住一件事：如果你试图用命令`cf-promises -r' 来处理这个，你将看到这样的一些输出：

```
atlas$ ~/LapTop/Cfengine3/trunk/src/cf-promises -r -f ~/test.cf
Summarizing promises as text to ~/test.cf.txt
Summarizing promises as html to ~/test.cf.html
```

`-r' 选项产生了一个报告。检查所产生的文件：

```
cat ~/test.cf.txt
firefox ~/test.cf.html
```

你将会看到一个关于 cfengine 如何翻译这些 HTML 或 text 文件的总结。在使用这个选项时，所有的 cfengine 组件将产生带有扩展视图的调试文件（例如为了配置文件被命名为 'promise_output_agent.h' ， 它 们 将 创 建 文 件 'promise_output_agent.html' and 'promise_output_agent.txt' ）。

3.2 查看一个文件

键入下面的例子：

```
body common control
{
bundlesequence => { "test" };
}
bundle agent test
{
files:
# 这是一个一次性的评论，下面是具体的承诺
"/tmp/testfile"                    # 许诺者
comment => "This is for keeps...",  # 即时注释
create => "true",                  # 约束条件 1
perms => p("612");                  # 约束条件 2, rw---x-w-
}
# 这是一个普通的填充模板，让承诺主体用参数来表示，更加整洁并可再利用
# 承诺主体 the promise body tidier and re-usable
body perms p(x)
{
mode => "$(x)";
}
```

这个例子显示了其他的属性如何加入承诺的主体。Perms 声明的右边是一个我们已经定义为'p()'的模板，它调用一个参数。模板被定义在要调用它的承诺组下面，显示我们如何创建可再用的参数组合。在这种情况下，例子不重要，但我们仅仅只是开始。当事情变得更娴熟，我们将在这些参数中隐藏大量的细节，因此让主要的承诺更清晰并且目的更明确。

现在执行 cf-agent 上的承诺：

```
host$ /usr/local/sbin/cf-agent -f /tmp/test.cf -I
-> Object /tmp/testfile had permission 600, changed it to 612
host$ ls -l /tmp/testfile
-rw---x-w- 1 mark users 33 2009-06-30 06:06 /tmp/testfile
```

'-I ' 标志告诉 cfengine 只通知我们关于一切的变化。这就提供了可理解的大量输出，并多于默认情况（只报告不能修复的问题或者明确的报告）。我们发现 cfengine 按顺序创建文件，并相应地设置权限。现在试图改变权限：

```
host$ chmod 400 /tmp/testfile
host$ ls -l /tmp/testfile
-r-------- 1 mark users 33 2009-06-30 06:06 /tmp/testfile
host$ /usr/local/sbin/cf-agent -f /tmp/test.cf -I
-> Object /tmp/testfile had permission 400, changed it to 612
```

```
host$ ls -l /tmp/testfile
-rw---x-w- 1 mark users 33 2009-06-30 06:06 /tmp/testfile
```

再来一次，记住前面的例子关于锁定和 ifelapsed 的注释。

注意这个承诺没有一个像 cfengine_3::的类表达。如果没有任何的声明，将采用这个默认的类 any:: ，它意味着任何的时间，任何地点，任何位置。

3.3 改变密码

为了改变系统的超级用户密码，我们需要编辑一个文件。一个文件是一个复杂的实体，一旦打开，那将是一个就它内容许下可能承诺的新世界。Cfengine 拥有一组特别用来编辑承诺。备份 shadow 文件并复制它到'/tmp'以致你能使用它。

```
body common control
{
bundlesequence => { "test" };
}
bundle agent test
{
files:
"/tmp/shadow"
comment => "Set the root password",
edit_line => SetPasswd("root","xyajd673j.ajhfu");
}
```

这就是我们需要在第一次观察中发现去理解已经许下的承诺。

下面的代码属于标准静态库，并可再利用来确保以上承诺。然而，与其他系统不同，你可以用 Cfengine 自己的语言来扩展它. 你不需要自己编写复杂的算法或者拥有一个开发环境。

```
#
# 隐藏的静态库代码
#
bundle edit_line SetPasswd(user,value)
{
field_edits:
"$(user):.*"
# 以用户名开始匹配行
# 设置文件的第二个参数
# 文件拥有格式 root:HASH: 或者 user:HASH:
comment => "Set field 2 of a colon-table",
edit_field => col(":","2","$(value)","set");
```

```
}
########################################
body edit_field col(split,col,newval,method)
{
field_separator => "$(split)";
select_field => "$(col)";
value_separator => ",";
field_value => "$(newval)";
field_operation => "$(method)";
extend_fields => "true";
}
```

3.4 更新组-供应

默认的 cfengine 配置包含了一组承诺，它把 cfengine 的源代码复制到高速缓存目录并且把服务器上的策略文件复制到默认的位置。这个例子是用来说明在本地文件系统上从一个文件复制到另一个文件。后来，当我们建立一个服务器组件，你将能够从远程的主机上复制文件。这是一个能够自动更新的供应系统的简单实例。

```
bundle agent update
{
vars:
# 一个源复制点的标准位置
"master_location" string => "/var/cfengine/masterfiles";
files:
"/var/cfengine/inputs"
comment => "Update the policy files from the master",
perms => my_p("600"),
copy_from => my_cp("$(master_location)","localhost"),
depth_search => recurse("inf");
"/var/cfengine/bin"
comment => "Update the cached binaries from installation",
perms => my_p("700"),
copy_from => my_cp("/usr/local/sbin","localhost"),
depth_search => recurse("2");
}
```

这些承诺在它们的主体中包含了几个我们还没看见的。copy_from 属性告诉 cfengine 如何从服务器复制文件。depth_search 属性告诉它在子目录及其文件中进行递归搜索。

尝试改变源文件和执行代理。

另外那有可再用的静态库模板：

```
body perms my_p(p)
{
mode => "$(p)";
}
#
body copy_from my_cp(from,server)
{
servers => { "$(server)", "failover.example.org" };
source => "$(from)";
compare => "digest";
}
#
body depth_search recurse(d)
{
depth => "$(d)";
exclude_dirs => { "\.X11", ".*kde.*", "logs", "log" };
}
```

这儿有个练习：尝试使用参考手册来查找在这个例子中的元素。看看你能否理解全部。

3.5 报告

Cfengine 包含了一个成为'cf-report '的报告生成器 。它是通过使用控制参数来配置的，这部分将在下章描述。尝试如下：

```
host$ /usr/local/sbin/cf-reports
host$ ls ~/.cfagent/reports
host$ mywebbrowser ~/.cfagent/reports/performance.html
```

大多数的这些报告起初都是空白的，除非你在 cfengine 上已经运行了一些重要的承诺。

3.6 cf-execd

Cfengine 包含了一项服务，它用来运行称作 exec-daemon 的代理，这个代理在目录’WORKDIR/inputs/promises.cf’ 已经有默认配置。如果你直接执行源代码，它将直接进入后台并默认每五分钟执行'cf-agent'上的默认策略。

你可以尝试在前台程序中运行：

```
host$ /usr/local/sbin/cf-execd -F
```

当你像这样运行 cfengine ,来自于 cfengine 的任何输出将被收集并放在在’WORKDIR/outputs’目录中。如果你已经配置了一个邮件地址并且你的主机正在运行 SMTP 服务，这个输出将以邮件形式发送出去。为了如此配置，你要在‘promises.cf’文件中加入一个控制主体。

```
body executor control
{
splaytime => "1";
mailto => "cfengine_mail@example.org";
smtpserver => "localhost";
mailmaxlines => "30";
}
```

这些其他的行改变了执行者固有行为的不同方面。例如，在执行代理前的一个负载平衡的时间延迟，一个邮箱地址，SMTP 邮件服务的别名和 IP 地址，还有被包含在发出的任何邮件中的输出的最多行数。

通过 cfengine 的配置，你应该开始明白它的模式。在下一章节，我们将了解一下这些一般的基本内容。

4 一个概念上的简单速成课

4.1 规则就是承诺

Cfengine 3 中的一切都可以被解释为一个承诺。承诺可以被制定关于所有不同种类的主题，从文件的属性到命令的执行及访问控制决定和知识联系。

这个简单但强大的理念保持在 cfengine 语法上的一致性。在编程语言中只有唯一的一个用作声明的语法格式，这是需要你知道的。它看上去一般像这样：

```
type:
classes::
  "promiser" -> { "promisee1", "promisee2", ... }
        attribute_1 => value_1,
        attribute_2 => value_2,
        ...
        attribute_n => value_n;
```

我们谈到承诺者（制定承诺的抽象实体），承诺人是承诺制定的对象实体，并且那有一系列的关系表，我们称其为承诺的主体。承诺的主体和承诺者的类型告诉了我们其中的要领。

> 承诺者总是被承诺影响的对象。

并不是所有的这些元素每次都是必要的。一些承诺包含了很多隐含的行为。在其他情况下我们可能想更加明确些。例如，最简单的报告承诺如下：

```
reports:
"hello world";
```

最简单的命令承诺如下:

```
Commands:
"/bin/echo hello world";
```

承诺有除了承诺者以为的其他一切的默认属性，例如，执行承诺的命令串，一个更复杂的承诺包含了很多属性：

```
# 承诺类型
files:
# 许诺者 ->  承诺者 (如果只有一行，不需要花括号)
"/home/mark/tmp/test_plain" -> "system blue team",
# 属性 => 值
comment => "This comment follows the rule for knowledge integration",
perms => users("@(usernames)"),
create => "true";
```

承诺者的列表被用来提供制作文档，而不被 cfengine 使用，正如命令属性（可以被添加到任何的承诺中）除了在错误跟踪和审查上给用户提供更多的信息外，没有其他实际的功能。

在这个例子中你看到几个不同类型的实体。在 cfengine3 中的所有字符串（例如“true ”）必须加上引号。这提供了绝对的一致性，使类型检查更简单，错误纠正能力更强大。所有类似函数的实体（例如 users("..")）要么是嵌入的特别函数，要么是在右侧带有内容的参数化模板。

4.2 控制承诺

Cfengine 组件制定的某些承诺已经扎根在它们的代码中。例如，通过邮件发送输出到一个正确地址的承诺，或在重复检查一个承诺之前（ifelapsed）的等待时间承诺。虽然这些承诺是固有的，但它们的行为可以被改变。在 cfenigne，行为总是受到承诺主体的约束。因此固有的行为随着每个控制主体的改变而改变。你能在参考手册中找到这些可改动的参数。

Common 组是最重要的组，能被所有 cfengine 的组件读取。它包含了承诺组的列表，这个列表应该被读入并因承诺建议而被检查。在`promises.cf '文件中：

```
body common control
{
bundlesequence => {
"update",
"garbage_collection",
"main",
"cfengine"
};
inputs => {
"update.cf",
"site.cf",
"library.cf"
};
}
#############################################################
body agent control
{
#如果默认运行时间是 5 分钟，为了长的作业我们需要这部分

ifelapsed => "15";
}
#############################################################
body monitor control
```

```
{
forgetrate => "0.7";
histograms => "true";
}
#############################################################
body executor control
{
splaytime => "1";
mailto => "cfengine_mail@example.org";
smtpserver => "localhost";
mailmaxlines => "30";
}
#############################################################
body reporter control
{
reports => { "performance", "last_seen", "monitor_history" };
build_directory => "/tmp/nerves";
report_output => "html";
}
#############################################################
body runagent control
{
hosts => {
"127.0.0.1"
# , "myhost.example.com:5308", ...
};
}
#############################################################
body server control
{
allowconnects => { "127.0.0.1" , "::1" };
allowallconnects => { "127.0.0.1" , "::1" };
trustkeysfrom => { "127.0.0.1" , "::1" };
# 让更新和运行一次性发生
cfruncommand => "$(sys.workdir)/bin/cf-agent -f failsafe.cf &&
$(sys.workdir)/bin/cf-agent";
allowusers => { "root" };
}
```

4.3 变量

变量（或"变量定义"）也是承诺—表示它们值的承诺。我们可以在任何承诺组中写下这些。Cfengine 认识到两个实体类型：标量和列表（列组包括 0 或更多的实体），还有三个数据类型（字符串，整数和实数）。在 cfengine 的输写是灵活的，正如在 Perl 和其他的脚本语言中。因此任何数据类型的变量可能被当作字符串。

4.3.1 标量变量

标量变量拥有一个单一的值。声明如下:

```
bundle <type> name
{
vars:
"my_scalar" string => "String contents...";
"my_int" int => "1234";
"my_real" real => "567.89";
}
```

`<type>' 表示任何类型的组适用于这里。标量变量通过 `$(name)' (or `${name}')被引用，并且它们每一次代表一个单一的值。

- 不带环境的标量 ，例如 `$(myvar)' 只限于当前组
- 如果有人使用了环境来检验它，那么标量在任何处都可用。例如，`$(context.myvar)' 用于访问读取'context'组中的'myvar'变量.

4.3.2 列表变量

列表变量拥有几个值。 声明如下：

```
bundle <type> name
{
vars:
"my_slist" slist => { "list", "of", "strings" };
"my_ilist" ilist => { "1234", "5678" };
"my_rlist" rlist => { "567.89" };
}
```

一个列表会带有符号`@'作为参考，但是在字符串中,这个参考通常没有任何意义。举个例子:

```
reports:
    cfengine_3::
```

"My list is @(my_slist)";

这没有任何意义并且不能展开（它不产生一个错误，但相反插入文字内容@(my slist)到字符串中）；但是如果我们使用标量引用到一个列表变量，cfengine 将在这个列表中重复这个值，有必要将它归进入一个承诺列表。

简而言之：

- 标量引用到 local 的列表变量意味着迭代。例如，假设我们已经有本地列表变量`@(list)', 那么标量`$(list)' 意味着在列表每一个值的一次迭代。
- 列表可在任何环境中全局通用，并且用 `@(list)'表示,

e.g.

vars:

"longlist" slist => { @(shortlist), "plus", "plus" };

"shortlist" slist => { "you", "me" };

声明的顺序不重要—cfenging 将执行分配变量`@(shortlist)'的承诺，然后执行分配变量`@(longlist)' 的承诺。

- 只有本地列表可以直接展开。因此`$(list)' 可以被展开而不是`$(context.list)' 。如果你想用它们来迭代，全局列表引用必须映射到一个本地环境中。为了更多的了解请看参考手册。

4.3.3 关联的数组

关联的数组变量也拥有几个值。声明如下：

```
bundle <type> name
{
vars:
"switches[mellow]" int => "1";
"switches[relaxed]" int => "1";
"off_keys" slist => { "red", "grouchy", "coarse", "febrile" };
"switches[$(off_keys)]" int => "0";
}
```

为了了解关联数组`getindices' 的功能和其他细节部分，请看参考手册。

4.4 决定

Cfengine 的决定是在后台作出的，确定的对或错提议的结果被储存在布尔值中，被称为类。在 cfengine 中没有 if-then-else 声明语句。所有的决定是通过类来作出的。

Cfengine 在每一台电脑上独立地运行，每次当它唤醒潜在的通用代理平台，它能发现运行环境或背景的特性并且进行归类。

类分成硬件型（公开的）和软件型（自定义的）。一个简单的硬件型类可以是几种情况的其中之一：

- 一个操作系统架构的命名 ， 例如 ultrix ,sun4 等等。
- 一个专用主机的未限定名字。如果你的系统返回一个完全限定的主机域名，cfengine将舍去第一部分。注意：www.sales.company.com 和 www.research.company.com 有相同的未限定名字。
- 主机的一个自定义组的名字
- 一星期中一天（以 Monday, Tuesday, Wednesday, .. 格式表示）
- 一天中的一小时，目前时区（以 Hr00, Hr01 ... Hr23 格式表示）
- 一天中的一小时，格林威治时间（以 GMT_Hr00, GMT_Hr01 ... GMT_Hr23 格式表示）。这与世界同步，以防你需要改变合作时保持实际地同步
- 小时中的分钟（以 Min00, Min17 ... Min45 格式表示）
- 小时中的 5 分钟间隔（以 Min00_05, Min05_10 ... Min55_00 格式表示）
- 每一刻钟（以 Q1, Q2, Q3, Q4 格式表示）
- 一月中一天（以 Day1, Day2, ... Day31 格式表示）
- 一个月（以 January, February, ... December 格式表示）
- 一年（以 Yr1997, Yr2004 格式表示）
- 夜晚、早晨、下午和傍晚的一个转换，划分为 6 小时的时间块，以时间 00：00 开始
- 一个生命周期的索引，就是年数以 3 为模 （使用在长期资源的内存中）
- 一个任意的自定义串
- 任何有效接口的 IP 地址八位组（以 ipv4_192_0_0_1,ipv4_192_0_0, ipv4_192_0, ipv4_192 格式表示）

来看看所有定义在专用主机上的类 ，运行

```
host# cf-promises –v
```

作为一个享有特权的用户。注意其中的一些类只有在用 cfenvd 建立信任连接的情况下才能设定。例如，如果都以特权运行，那么文件`/var/cfengine/state/env_data' 是安全的。更多关于类的信息可以参有关 allclasses .

自定义型或软件型类被定义在组中。类型 common 的组可以以类表示，具有全局性，然后在其他组类型中是本地可用的。当评估组时，软件型的类也被评估了。它们可基于测试功能或者来自其他类型的类：

```
bundle agent myclasses
{
classes:
"solinus" expression => "linux||solaris";
# 列出格式有益于所包括的功能
"alt_class" or => { "linux", "solaris", fileexists("/etc/fstab") };
"oth_class" and => { fileexists("/etc/shadow"), fileexists("/etc/
passwd") };
reports:
alt_class::
# 这将在 linux, solaris,或任何系统中只报告 "Boo!"
```

```
# 系统上的文件 /etc/fstab 存在
"Boo!";
}
```

类可以与运算符结合，优先级按从高到低的顺序列出：

`()'　括号组运算符
`!'　非运算符
`.'　逻辑与运算符
`&'　逻辑与运算符 (二选一)
`|'　逻辑或运算符
`||'　逻辑或运算符(二选一).

所以下面的表达是正确的:在 Windows XP 系统中于星期一或星期三的下午 2 点到 2：59 :

(Monday|Wednesday).Hr14.WinXP::

看看下面更复杂的例子。在 common 类型组的承诺具有全局性，而所有其他的承诺只限于他们各自的组中。

```
body common control
{
bundlesequence => { "g","ls_1", "ls_2" };
}
###################################
bundle common g
{
classes:
# "zero" 承诺总是能满足，它不限范围
"zero" expression => "any";
}
###################################
bundle agent ls_1
{
classes:
# "one" 承诺总是能满足，它只限于 ls_1 的范围
"one" expression => "any";
}
###################################
bundle agent ls_2
{
classes:
# "two" 承诺总是能满足，它只限于 ls_2 的范围
"two" expression => "any";
```

```
reports:
zero.!one.two::
# 这个报告@b 将会被生成
"Success";
}
```

在这里我们看到`zero'类具有全局性而`one'类和`two' 类是限制在本地使用的。因为只有`zero' and `two' 在`ls_2'组的范围内。，因此`Success'的报告结果是正确的 result is therefore (并且`ls_2' 组的类表达要求`zero'和`two' 都是正确的，除了`one'之外)。

4.5 循环程序

当你以程序员的思维来思考，如果你在 cfengine 中寻找循环程序，那么我们需要对你进行一部分的洗脑。Cfengine 不是一种意味着给你低级的控制权限的编程语言，它只是一套具体描述过程的声明。这就相当于自行车上的齿轮和传送设备上的自动传送带的区别之处。

在 cfengine 中循环程序是隐含执行的，但没有可见的机制，因为这样会分散了对承诺意图的注意力。表达他们的方式是通过列表。

循环程序确实是一种在一个列表中迭代一个变量的方法。看看下边。

```
body common control
{
bundlesequence => { "example" };
}
#############################################################
bundle agent example
{
vars:
# 这是一个列表
"component" slist => { "cf-monitord", "cf-serverd", "cf-execd" };
#这是一个关联数组
"array[cf-monitord]" string => "The monitor";
"array[cf-serverd]" string => "The server";
"array[cf-execd]" string => "The executor, not executionist";
reports:
cfengine_3::
"$(component) is $(array[$(component)])";
}
```

输出如下所示：

```
/usr/local/sbin/cf-agent -f ./unit_loops.cf -K
R: cf-monitord is The monitor
R: cf-serverd is The server
```

R: cf-execd is The executor, not executionist

你从这里看到，如果我们涉及到一个使用标量引用运算符'$()'的列表变量，cfengine 将进行对此编译，即迭代列表中所有的值。因此为我们已经有一个非常有效的'foreach' 循环，而不需要加入其他的任何的语法。

4.6 主要的承诺类型

以下的承诺类型可以在任何组中使用：

vars	一个表示值的变量的承诺
classes	一个表示系统状态的承诺
reports	一个报告信息的承诺

这些另外的承诺类型只能在代理组中使用：

commands	一个执行命令的承诺
databases	一个配置数据库的承诺
files	一个配置文件的承诺，包括它的存在，属性以及内容
interfaces	一个配置网络接口的承诺
methods	一个承担整个组的承诺
packages	一个安装包的承诺
storage	一个检验附加存储器的承诺

这些承诺类型属于其他的组件：

access	一个授权或拒绝访问在 cf-serverd 中的文件对象的承诺
measurements	一个从系统估量数据或抽样数据承诺，用来在 cf-monitord 监测和报告 (Cfengine 初级版本以及更新的版本).
roles	当在 cf-serverd 上远程的运行 cf-agent ,一个允许特定的用户激活特定类的承诺
topics	一个在 cf-know 将知识与名字和其他可能的主题联系起来的承诺
occurrences	一个在 cf-know 中指向或涉及知识资源的承诺

5. 使用 cfengine 作为前端或者代替 cron

5.1 我需要使用 cron 吗?

Unix 的 cron 命令是有用的，但它使用起来又往往非常笨拙。 Cfengine 的一大优点就是它能使用类并通过一个或者多个文件识别不同的系统。许多管理员认为如果 cron 进程也能以这种方式工作，那该多好。 一种从全局配置的角度设置 cron 的可行方法就是使用 cfengine 的文本编辑功能来分别编辑每一个 cron 文件。但不管怎样这种方法还是错过了一个很明显的机会。

一个更好的方法是使用 cfengine 的时间类让 cron 以用户接口的方式工作。 这种方法可以让你拥有一个单独的包含了系统所有 cron 任务的 cfengine 的主要文件，并且不会失去 cron 所提供给你的终端控制。这种方法具备了所有普遍的优点:

当你把所有的东西都放在一处时， 它会使得保持跟踪你的系统正在运行什么样的 cron 任务变的更容易。

你可以使用所有你精心设计的组和用户定义的类来识别哪一个主机得要有运行哪些程序。

这个机制的核心思想是在每一个系统上设置一个规律的 cron 任务,这个任务会按一定的频率间隔执行 cfagent。 每次 cfagent 启动后， 它会评估一下时间类并且执行配置文件中定义的 shell 命令。通过这种方式 我们使用 cfagent 封装 cron 的脚本,从而可以使用 cfengine 的类来控制多个主机的任务。Cfengine 的时间类至少与 cron 的时间说明书的功能差不多强大，它们还增加区域控制的功能。这一点完全不会限制你，见章节 5.5 [创建灵活的时间类]，第 32 页。做这些的唯一的代价就是需要解析 cfengin 中无关紧要的文件。

我是否需要使用 CRON? 不需要。有了 cfengine 的 cf-execd, 你就不需要使用 cron 了，cfengine 能够自动安排好一切。关于是选择以后台程序的模式（daemon mode）还是封装的模式(wrapped mode)来运行 cf-execd，这完全由你来决定。在 cfengine 的商业版本中，可执行的后台程序(exec daemon)具备了更加精细的可靠性。在开源的版本中，你也能够非常舒适的使用 cfengine 独立地监视系统，尤其是遇到源代码更新的过程中出现的运行程序由于出错而死机的情况。

5.2 单一 cron 任务的实现

为了能更明确的说明，假设我们在你的网络的每一台主机上都安装了以下的 ‘crontab’ 文件：

```
#
# Cron 全局文件
#
*/5 * * * * /usr/local/sbin/cf-execd -F
```

5.3 结构化命令承诺

一个承诺组的结构需要能反映你的系统上正在运行的任务的策略。你需要根据每天的时间打开相关的任务并且关闭那些不想要的任务。这个工作可以通过把不同的动作放在相应的时间类下来实现，这些时间类能够限制在什么时间来执行这些任务。

```
promise-type:
        time-based classes::
          Promise
```

例如：

```
bundle agent example
{
commands:
# 在午后的第一个 15 分钟内执行
    Hr12.Q1::
        "/path/myscript -arg1 -arg2";
# 在任何时间的 15 分钟内执行
    Q2::
        "/path/otherscript";
# 在时间段 00:10 到 00:15 和 12:45 到 12:55 执行
# 在英语语言中表示为"和"， 但在逻辑语言中表示为"只要满足其中一个时间段"
  Hr00.Min10_15||Hr12.Min45_55::
        "/path/amongstourscripts";
}
```

如果你想要更花哨一点，你可以同创建一个层次结构来放置执行脚本的参数，这个层次结构能够限制输出结果和优先级（这个功能只有 root 用户可以使用，因为只有 root 用户有权利改变优先级）。

```
bundle agent example
{
commands:
#在午后的第一个 15 分钟内执行
    Hr12.Q1::
      "/path/myscript -arg1 -arg2",
      contain => jail("nobody","true");
}
# ...
body contain jail(owner,devnull)
{
    exec_owner => "$(owner)"; # 使用 setuid 运行
    no_output => "$(devnull)"; # 类似 > /dev/null 2>&1
    umask => "77"; # 设置文件权限掩码
```

}

`contain'方法体提供了一个安全灵活的环境来嵌入脚本。

类的时间分辨能力受限于你多久运行一次 cfengine 或者 cron 或者 cf-execd。通常 5 分钟是比较推荐的时间间隔。

5.4 展开主机时间

在一个成千台计算机的网络中，许多代理（agents）可以同时开始执行并从服务器上下载资源。例如，如果有一千台 cf-agents 都突然想从一个服务源（master source）同时拷贝一个文件，这将会导致服务器承载太大的负载。我们可以通过引进时间延时来避免这个问题的产生，时间延时对于每一个主机都是唯一的并且不会超过给定的时间间隔；cf-execd 使用散列算法（hashing algorithm）生成一个数值，范围在 0 到你所规定的分钟数的最大值之间，例如：

```
body executor control
{
   splaytime => "10"; # Minutes
}
```

如果这个数值非零，cf-execd 会在解析了配置文件并读了时钟后进入休眠状态。每一台机器的 cf-execd 都会在不同的时间长度后进入休眠状态，这个时间长度不会超过规定的时间。

散列算法（hashing algorithm），基于完全合格的主机名，被用来为每台主机计算不同的时间。间隔时间越短，主机所属的组越多。间隔时间越长，你服务器所承担的负载越轻。运行时间的‘展开’能够减轻服务器的负担，即使它们来自于你所不能控制的域，但只要有类似的 cron 策略就可以。

5.5 创建灵活的时间类

每次 cfengine 运行的时候，它会读取系统时钟并基于时间和日期来定义类（见参考手册）。

时间类是基于 cfagent 启动时的具体分钟，但它不会直接被用在策略中（除非 cf-execd 中已经规定了）。许多事情可能恰巧会延迟 cfengent 启动的具体时间。能够监测具体启动时间的主要目的是为了能够定义关于任意时间间隔的组合类。为了实现这个功能，我们使用类组的行为来创建一个组的时间值的别名。接下来是一些很有创造性的例子：

```
classes: # synonym groups:
"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";
```

在前三个例子中，左边的附值对于右边的 ORed 结果非常有效。因此如果花括号里的任何类

被定义了，那左边的那个类就会成为已定义的。这为规定一个程序的时间间隔提供了灵活的可读的方式，而不再需要到处使用‘|’ 和‘.’运算符。

5.6 选择一个可安排的时间间隔

你得多久一次调用你的全局 cfengine 配置呢？有如下几件事情你得考虑：

- 你需要多好的控制？通常每小时运行一次 cron 任务对于大多数任务来说已经足够了，但你可能需要为一些特别的任务做特别的更好的控制。
- 你是要验证整个的 cfengine 配置文件还是只是有选择的承诺？

cfengine 拥有智能的锁策略(locking)和超出时间策略（timeout），这些策略足够将 shell 命令挂载到之前的 cron， 从而不会产生隔阂。

6. 网络服务

这一章讲述怎样创建 cfengine 的网络服务来处理远程文件分布以及远程执行 cfengine， 而不需要使你的主机可能面临使用 rsh 协议的黑客攻击。

6.1 cfengine 网络服务

通过启动 cf-serverd 进程，你可以为主机之间创建通信线路，允许它们通过网络交换文件或者在另一个系统上远程地执行 cfengine。 Cfengine 网络服务是基于以下组件创建的：

Cf-agent 引擎的配置与网络的唯一联系方式就是通过远程拷贝请求。它不会也不能应答何来自网络的访问系统请求。它只能够从服务器的组件来请求访问文件。

Cf-serverd 是一个既可以作为文件服务器也可以作为一个远程的 cf-agent 执行器的后台程序（daemon）。 这个后台程序可以认证来自网络的请求并且根据服务器的控制体（control body）和包（bundles）中包含的访问承诺（access promises）所定义的规则来处理这些请求。

Cf-runagent 这是一个简单的初始化程序，可以被用来在多个远程主机上运行 cf-agent。它不能被用来告诉 cf-agent 做什么，它只能让远程主机上的 cf-serverd 根据现有的配置来运行 cf-agent。通过授权给用户来为部分现有的策略提供一种基于角色的访问控制（RBAC）。

有了这些组件，你就有了所有你需要的东西对系统的资源进行有效地分布。

6.2 服务怎样工作

6.2.1 远程文件分布

这个部分讲述了怎样把 cf-serverd 创建成一个远程的文件服务器，从而使文件以一种安全可靠的方式分布到客户主机上。

Cfengine 与其它系统的一个重要的不同在于文件分布的方式。Cfengine 使用了一种‘拉动’（pull）而非‘推进’（push）的模型来分布网络文件。而大多数的系统，可能通过强迫服务器上的一个镜像文件分布到所有的客户机上。这种方式在黑客攻击的时候常常发生—接收者确实常常被要求打开各种端口并且接受他们所接收到的所有信息。原则上讲，Cfengine 是不会支持这种技术的。

使用‘推动’的方法，文件会在分布者希望它改变的时候改变，而客户却没有选择的余地，只能接受这样的后果。Cfengine，从另一个方面来说，通过自愿协作（voluntary cooperation）

的方式来工作。主机也被允许保持它们的防护控制，从而按照他们自己的意愿来保护自己免受黑客攻击和‘推动’。

事实上，(在设计上) cfengine 不能把自己的意志强加给其他主机，它自身也不会被强迫。为了能以最好的信号分布到所有机器上，并且如果这些机器愿意才向它们搜集文件。换句话说，cfengine 模拟了一种‘推进’模型，它通过民意调查了解每个客户以及其正在运行的本地的 cfengine 配置脚本，从而让主机有机会能够从远程的服务器‘拉动’任何已更新的文件，但仍然由客户机来决定是否要进行更新。

还有，对照一些程序，如 rdist，它被用来分布文件到许多主机上，而 cfengine 不需要使用‘.rhosts’文件或者‘/etc/hosts.equiv’文件来进行任何的 root 访问系统。以 root 身份运行后台程序已经足够了。但你不能通过把它加到系统的‘/etc/inetd.conf’文件中来运行它。这种对后台程序功能的限制能够保护你的系统，避免使用 rsh 作为 root 运行通常的命令。

要想远程地访问服务器上的文件，你可以在一个‘files’承诺中使用 copy_from attribute 的属性：

```
bundle agent example
{
files:
"/var/cfengine/inputs"
   perms => my_p("600"),
   copy_from => rcp("$(master_location)","localhost"),
   depth_search => recurse("inf"),
   action => immediate;
}
# 库模板
body copy_from rcp(file,server)
{
servers => { "$(server)", "failover.example.org"};
source => "$(file)";
}
```

假设 cf-serverd 后台进程正运行在 server-host 上，cf-agent 将会与这个后台进程联系并试图获得文件的信息。在这个过程中，cfengine 验证了这两台逐级上的系统时钟是同步的。如果没有同步，它不会允许远程拷贝除非在服务器的控制体（control body）中 denybadclocks 是 false。

如果 cf-agent 决定一个文件需要从远程服务器上被更新，它就开始在一个相同的文件系统上拷贝远程的文件到一个新的文件来作为目的文件。这个文件的后缀名是‘.cfnew’。

只有当文件被成功地搜集了，cf-agent 才会拷贝旧的文件，(见参考手册的存储(repository))，并且给新文件改名字。设计这种行为是为了避免网络连接中的竞争状态(race-condition)，确实任何操作都会占用时间。如果文件能够很简单地被直接拷贝到它想要的新的目的地，而一个网络连接的错误可能会中断传输从而只留下一个中断了的文件在那儿。Cf-agent 给网络连接设置了几秒钟的中止时间来避免悬挂的进程。

通常后台进程休眠，等待网络连接。这种连接可能会通过另一台主机上的一个运行的 cf-agent 程序请求远程文件进行初始化，或者通过 cf-runagent 简单地要求主机后台运行的进程来运行本地的 cf-agent 或 cf-execd 程序进行初始化。

6.2.2 远程执行 cf-agent

偶尔你想要立刻运行 cf-agent 在一个或多个主机上尽快地实现配置的改变。如果通过手动登录每一台主机来做这件事情会非常不方便。

如果你安排的时间间隔足够的话，这就没有必要了，因为 cfengine 将在你试图登录的时候已经运行了– 这种并行方式表明在数分钟之内整个网络能够被改变而没有由于等待中心控制而产生延迟。

但也许你想要发送一些特别的信号，例如，运行策略含有一个只能被一些机器激活的特殊类。那么一个更好的方式就是运行一个简单的命令来与远程主机联系并且运行具有基于角色访问控制（RBAC）的 cf-agent，在你自己的屏幕上能够立即显示输出的结果：

```
host$ cf-runagent remote-host -v
output....
```

- 为了重新配置远程的主机你不需要登录远程的主机。
- 用户而非 root 可以运行 cf-agent 来解决系统的任何问题，他们有权限访问一些个体和类。

任何此类系统的一个潜在缺陷是一些恶意的用户能够在远程主机上运行 cf-agent。事实上非 root 用户能够运行 cf-agent 本身并不成问题，毕竟他们能够做的大部分的恶意事件都能够被系统配置检测出来并且修复。没有人能告诉 cf-agent 使用 cfrun 程序能够做什么，它只能够运行一个现有的配置。但一个更严重的问题是有些恶意的用户可能试图多次运行 cf-agent（所谓的‘Denial of Service’攻击），以致系统由于连续运行 cf-agent 而负载过重。为了避免这种情况，服务器使用同样的 ifelapsed 锁来辅助访问控制。

6.3 远程访问解释

6.3.1 服务器连接

为了能够连接到 cfengine 服务器，你需要

一对公钥私钥

想要创建一对密钥，运行 cf-key

一个 IPv4 或者 IPv6 地址

你必须在线并拥有一个配置好的网络地址。

一个客户端程序

Cf-agent 和 cf-runagent 都是能连接到服务器的客户端。

允许连接到服务器，并且

服务器控制体（control body）必须允许对你计算机和基于名字或者 ip 地址的公钥的访

问，通过在一个列表中显示出来（如下）。

你的公钥必须被服务器信任，并且你必须信任服务器的公钥

通过相互信任彼此的密钥，客户端与服务器统一使用该密钥作为计算机的一个足够的标示符。

允许访问一些东西

你的主机名或者 IP 地址必须在一个你想要访问的文件所产生的服务器包(server bundle)的访问承诺（access promise） 中被提及。

如果以上的所有标准都达到了，连接将会被建立并且数据会在客户端和服务器之间被传输。根据 cfengine 的协议，客户端只能发送简短的请求。服务器能够以多种形式返回数据，通常是文件形式，但有时会是终端输出形式。

6.3.2 远程访问的疑难解答

当创建 cf-serverd， 你可能会看到错误信息’Unspecified server refusal’。

这表明 cf-serverd 不能或者不愿意认证来自你的客户端机器的连接。这个信息是一般性存在的：它故意不是专门的，因而任何人想要攻击或者利用这个服务将不会得到可能对他们有用的信息。这儿有一个简单的检查列表来解决这个问题：

1. 确保在被客户端和服务器读取的配置文件中设置域名变量；或者使用 skipidentify 和 skipverify 来减弱 DNS 的认证。
2. 确保你在服务器实体(server body)中授权访问你的客户端

```
body server control
    {
    allowconnects => { "127.0.0.1" , "::1" ...etc };
    allowallconnects => { "127.0.0.1" , "::1" ...etc };
    trustkeysfrom => { "127.0.0.1" , "::1" ...etc };
    }
```

3. 确保你已经使用了 cf-key 为主机创建了有效的密钥。
4. 如果你在使用安全拷贝，确保你创建了一个密钥文件并且你已经在所有参与的主机群组中分布并安装了这个密钥文件。

一直要记得你既可以以冗余模式(verbose mode)运行 cfengine 也可以以调试模式(debugging mode)运行 cfengine 来看认证过程是怎样发生的：

```
Cf-agent   -v
Cf-serverd –v
```

不管是什么错误属性，cf-agent 都会报告被拒绝的访问，从而避免泄露信息，给黑客可乘之机。想要找到被拒绝的真正原因，使用冗余模式‘-v’或者调试模式‘-d2’。

6.3.3 交换密钥

Cfengine 使用的密钥交换模式是基于 OpenSSH 的密钥模式。它是一个点对点（peer to peer）的交换模式，而不是一个中心化的认证授权模式。这意味着它没有可扩展性的方面的瓶颈（至少设计上讲，你可以介绍你自己的想法如果你想要一个过度中心化的结构）。

密钥分布的问题是每一个公钥结构的难题。Cfengine 能自动实行交换密钥，并且你所要做的只是决定该相信那个密钥。

当公钥被提供给一个服务器，他们可能会自动被接受为可以信任的，因为没有人可以决定它们。这会导致关于第一个提交密钥申明的竞争。

即使有 DNS 来检验相关名字或者 IP 地址的正确性（关闭 skiperify），它仍然可能提交给服务器一个错误的密钥。

服务器 cf-serverd 默认能够阻止接受未知的密钥。为了能够接受一个新的密钥，假定的客户端的 IP 地址必须被列在 trustkeysfrom 中。一旦一个密钥被接受了，它将永远不会被一个新的密钥所替换，因此不再需要提供更多的信任。

一旦你已经安排连接到服务器，你必须决定哪些主机将要访问哪些文件。这将由访问规则（access rules）来决定。

```
bundle server access_rules()
{
access:
"/path/file"
admit => { "127.0.0.1", "127.0.0.2", "127.0.0.3" },
deny => { "192.*" };
}
```

在客户端，例如，cf-runagent 和 cf-agent, 有三个问题需要考虑：

1. 选择连接哪个服务器
2. 信任任何先前未知服务器的身份，例如，信任服务器的公钥就是该服务器的而不是别人的(这点对于服务器也是一样)。
3. 选择数据传输是否需要加密（带密码）。

因为有两个客户端连接到 cf-serverd(cf-agent 和 cf-runagent)，所以客户端还是有两种方法来管理服务器密钥的信任。一个是自动选项(automated option)，在一个 copy_from 中设置 trustkey 选项，例如，

```
body copy_from example
  {
   # .. other settings ..
   trustkey => "true";
  }
```

另一种方法是以交互的模式运行 cf-runagent。当你运行 cf-runagent 时，未知的服务器密钥会互动地提供给你（例如通过 ssh），让你能够手动地接受或者拒绝：

WARNING - You do not have a public key from host ubik.iu.hio.no =
128.39.74.25
Do you want to accept one on trust? (yes/no)
-->

6.3.4 时间窗口（竞争）

一旦公钥已经被从客户端到服务器和从服务器到客户端交换，根据公钥认证机制，信任的问题就已经解决了。你只需要担心那些从一端到另一端从未见过的连接是否值得信任。

通常你会有一个中心的服务器和许多客户端卫星。而传输所有密钥的的最好方式是在服务器端和客户端上设置 trustkey 标识，并与你知道 cf-agent 将运行的时间保持一致，从而欺骗者就不可能进行干涉。

这是一个一次性的任务，黑客能够伪装成密钥传输者的几率很小。它需要技巧以及关于交换过程的内部信息，而这易于暗示信任模型已经破裂。

另一个方法是所有的在一组的主机都从中心服务器运行 cf-runagent 并且逐个地手动接受密钥，尽管这种方法所获很少。

信任一个主机来进行密钥交换是不可避免的。没有更聪明的方法来避免它。即使使用磁盘来传输文件，并检查你所拥有的每台计算机的序列号，主机也必须得信任你给它的信息。这都是基于假设。你几乎不能够使得密钥造假或者被攻击，但你又不能使它绝对不会发生。安全是关于如何合理的管理风险程度，而不是魔术。

所有的安全都是基于某个时刻及时信任某个点。密码学的密钥方法知识去除了反复做信任抉择的需要。在第一次交换后，信任不再需要，因为他们的密钥允许身份被实际验证。

即使你开放信任的选项，你并不是盲目地信任你知道的主机。唯一的潜在的不安全性在于那些你所不知道的新的密钥。如果你在信任规则中使用通配符或者 IP 前缀，那其他的主机可能就会有一些欺骗行为，因为你给他们留了些敞开的漏洞。这就是为什么推荐，在每次传输完密钥后，可通过注释信任选项将系统回归到零状态。

我们可以通过一些有可能但费力的方式将密钥通过不同的频道信号进行传输，通过拷贝‘/var/cfengine/ppkeys/localhost.pub’到‘/var/cfengine/ppkeys/user-aaa.bbb.ccc.mmm’(假设使用 IPv4) 在另一台主机上。例如，

localhost.pub -> root-128.39.74.71.pub

这可能是一种比较愚蠢的方法在你自己控制的邻近主机之间传输密钥，但如果传输到远距离，远程的主机上，它可能是一种管理信任的简单方法。

6.3.5 除了 root 以外的用户

Cfengine 通常以‘root’用户运行（除在 Windows 系统上通常没有 root 用户），例如， 一个高级权限的管理员。如果其他用户也被授权可以访问这个系统，他们必须也生成一个密钥并且经过相同的过程。除此之外，用户还必须被加到服务器的配置文件中。

6.3.6 加密

Cfengine 为传输过程保持文件内容的隐私性提供了加密。它假设用户能够明智地使用。通过加密传输的公用文件，就不能被窃取任何东西了–过度使用加密只会造成全球变暖，耗费不必要的 CPU 处理时间而不能提供任何安全。

配置管理中加密的主要作用就是认证。Cfengine 总是使用加密来进行认证，因而加密设置不会影响认证安全。

7 知识管理

Cfengine 具有一个非同寻常的能力就是它能把知识管理统一到自动化过程之中，并将它的配置技术作为一个‘语义上的’文件引擎。

知识管理是对我们时间的挑战。组织机构浪费了难以想像的大量的精力来重新学习旧的课程因为这些课程没有被做成文件留给后一代。现在你可以通过一些简单的规则来缓解这个问题并且甚至能建立成熟的文件索引数据库。

7.1 承诺与知识

这些年来，配置管理系统的学习曲线已经成为了经常批评观摩的动力。用户被期望能够以一个较细节的水平来面对信息的复杂性，或者能够放弃完好统一控制的想法。这导致的信息过载或者过于简单化。因此这种处理信息复杂性的能力成为了 IT 管理的基础。

Cfengine 为配置个引入了承诺模型，是为了使这个学习曲线更平坦。这能够简化使用的过程，因为许多需要思考的工作已经完成并且被封装成了模型。它的一个特殊属性是它既是一个系统行为的模型也是一个知识展现的模型（这正是声明语言所寻求的）。更特别的是，它把一个 ISO 的标准子集合并成了‘话题图形’（Topic Maps），这是一个用于信息资源的语义索引的开放技术。通过把这两个技术融合在一起（这两个技术是高度兼容的），最终，我们能够将前端准确无误地吻合在一起并可以浏览系统信息。

知识管理本身就是一个研究领域，它涉及了人与科技大量的相关问题。大部分人都会认可知识是事实和关系的组合，因此为了合理地诠释知识，既需要清晰的定义，又需要语义内容；但是我们怎样才不会模棱两可地定义原信息？

知识与配置有很多的共同之处：到底什么是知识而不是我们思想中的一种理念配置，或者在一些表现媒介上（论文，硅等）。它是一种被编码过的形式，通常更倾向于一种我们能够与其他人达成一致意见并且共享的形式。知识和配置管理都是关于描述样式。一个简单的知识模型能够被用来表示一种策略或者配置；相反地，一个简单的策略配置模型能够产生一种知识结构，就如同它能产生一个文件系统或者一系列的服务。

7.2 知识的基础

知识真正开始于我们记录下事情的时候：

- 将某些东西用书写来表达能带来一种想法的自律，而不仅只是使一个想法更清晰。
- 在你试图将一个想法记录成语言之前，你是不会完全地正视这个想法的。
- 任何书写记录都能使得其他人阅读它并流传知识。

麻烦的是，人们通常不喜欢写东西，而且也很少有人是非常擅长写东西的。对于一个工程师而言，通常会觉得停下正在做的事情而去记录所做的事情，感觉是在浪费时间，尤其是

当这天比较繁忙的时候。而且，写东西需要一种创新性想法的绽放，相对于写些流畅的看似冗长的语言形式，工程师们通常更喜欢使用技术性的模式和标记。

Cfengine 试图通过简化文档和部分技术配置来弥补这个差异。Cfengine 的知识代理使用 AI 和网络科学算法(network science algorithms)，基于技术注释来创建可读的文档。它之所以能这样做是因为许多想法早已经融入了承诺模型的想法之中。

7.3 注释承诺

知识的开始是能够注释技术文档。记住一个承诺的重点是要表达一种意图。在写承诺的时候，要养成一个习惯，就是能够给每个承诺一个评论来解释它的意图。并且，期望能够一些特殊的承诺句柄(handles)，或者有用的标签，使得在其他承诺的描述中能参考这些特殊的承诺。一个句柄(handle)可以是一些任意的词像‘xyz’，但是你得尽量使用一些更有意义的标签从而使得参考更加清晰。

```
files:
"/var/cfengine/inputs"
handle => "update_policy",
comment => "Update the cfengine input files from the policy server",
perms => system("600"),
copy_from => rcp("$(master_location)","$(policy_server)"),
depth_search => recurse("inf"),
file_select => input_files,
action => immediate;
```

如果一个承诺在某方面影响另一个承诺，你可以使那个被影响的承诺成为其中之一的承诺者(the promisees)， 像这样：

```
access:
"/master/cfengine/inputs" -> { "update_policy", "other_promisee" },
handle => "serve_updates",
admit => { "217.77.34.*" };
```

相反地，如果一个承诺在某些方面依赖于另一个承诺（甚至是间接地），也把它记录下来。

```
files:
"/var/cfengine/inputs"
handle => "update_policy",
comment => "Update the cfengine input files from the policy
server",
depends_on => { "serve_updates" },
perms => system("600"),
copy_from => rcp("$(master_location)","$(policy_server)"),
```

```
depth_search => recurse("inf"),
file_select => input_files,
action => immediate;
```

注释的使用是 cfenging 中文档的第一个层次。这些注释会在 cfengine 内部用于提供有意义的错误信息内容并且能计算出相关过程链存在的依赖关系。这些可以被转换成一个话题图形（topic map）用于浏览策略关系是一个网页浏览器，使用 cf-know。

为了建立这个知识库，你将需要一台运行 Apache 网页服务器的计算机，并在该计算机上安装 PHP 组件。这个知识基地也可以与另一台网页服务器一起运行，只要 Apache 是正在运行的。要生成图解表示法，你需要 GraphViz 包。更多细节见最后的 annex。

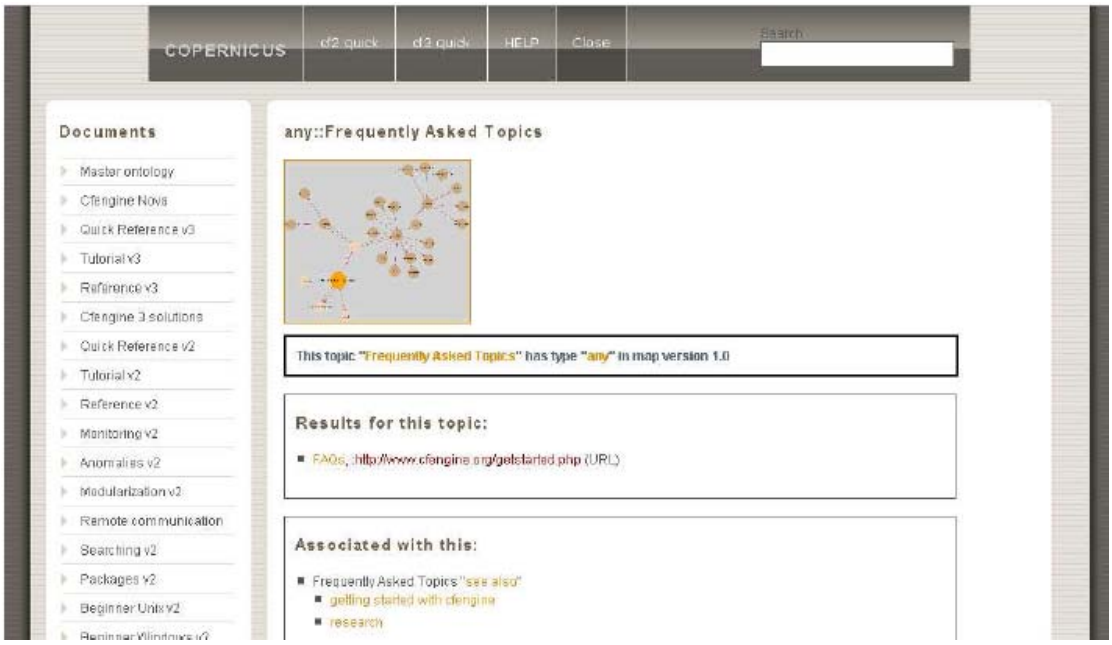
7.4 话题图形（topic maps）提供些什么

商业版本的 cfengine 能够自动提供策略文档，通过使用上面所提供的基本注释作为一个知识图形。它们基本不需要用户做什么。如果你正在使用开源版本的 Cfengine，所有的技术都可以获得使用，但你得手动来完成这个工作。不管哪种情况，一旦你熟悉了话题图形(Topic Maps) 的使用，你就可以手动地拓展你的知识，并与以下一些事情结合起来：

- 本地策略文档（高层次的）
- 相关数据库，例如 CMDBs

所以让我们花点时间来看看怎样使用 cf-know 将知识嵌入到话题图形中。

你所期望的一些结果在下图中显示。 这些示例图显示了由知识代理 cf-know 生成的典型的页面。第一张图显示我们如何在 cfengine 商业版本支持入口'Copernicus'中使用技术，从而使网页知识基础更强大。

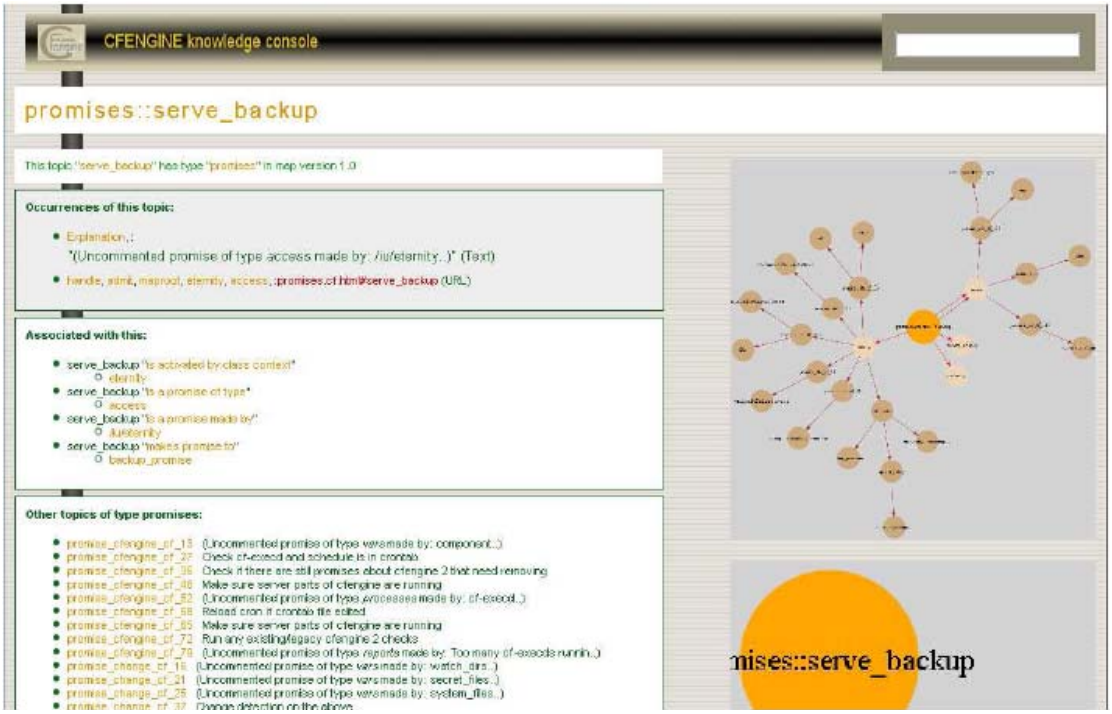


在使用过程中，所有的数据都是基于 cfengine 软件的文档，并且所有的关系都是通过手动输入的。

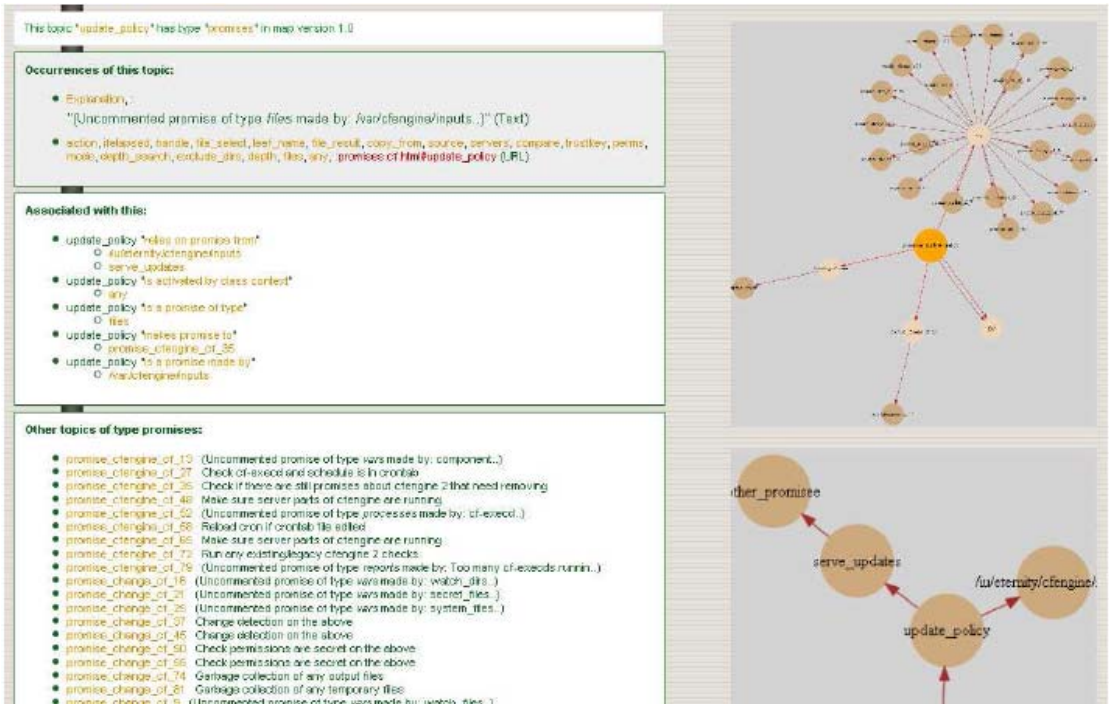
在第二个例子中，考虑了 cfengine 是怎样生成关于其配置的知识图形分析（自我分析）。下面图形中的数据描述了 cfengine 配置承诺。一个这样的页一旦生成，例如，对于每一个策略承诺，会从不同的计算机生成页面用于报告。你还可以为你所拥有的本地（企业）信息创建你自己的‘话题页面’。

在这个例子中，一个承诺被给予了一个承诺句柄 update_policy，并且这种关联和图形表明了这个承诺通过存档的依赖关系与其他的承诺相关联(这些依赖关系来自于被承诺者的存档，并且依赖于其他承诺的属性)。

示例页面中显示了两幅图形，一个在另一个的上面。在上面的图像表明了 30 个与此相关的最接近的话题（任何形式）。在此这种关系没有被确定。这个图表能够表示一些意料之外的相关信息的路径，并且说明它们之间的关系，从而拓宽了我们对于当前承诺在整个图表中所处位置的理解。



尽管图形的说明只是更充分地表现了文本中的语义关系，但他们在相互关联的网络中还可以虚拟化几个层次的深度。这在集体讨论或者推理时可能会非常惊人的有用。尤其是，如果我们对当前的承诺做了一些改变，其他的一些承诺也可能会受到影响。这样的影响分析对于改变计划和策略发布管理可能是非常重要的。



一个知识库是对于一个 ISO 标准技术的话题图形（Topic Map）的实现。一个话题图形会像一个索引那样工作，它能够指向许多不同类型的外部资源，并且可以包含内部的简单文本和图片。因此你可以使用它来绑定任何类型的文档。一个 cfengine 的知识库并不是一个新的文档格式，它是一个将想法和资源结合起来的叠加图形，并且可以显示关系。

7.5 循序渐进

你可以使用 cf-know 以文本（用于命令行）或者 HTML 的形式来表示一个话题图形（用于网页显示）。我们先来说明一下文本显示，因为它不需要太多的结构。你只需要一个数据库。

试着输入下面的知识承诺：

```
body common control
{
bundlesequence => { "tm" };
}
body knowledge control
{
query_output => "text";
query_engine => "none";
sql_database => "test_map";
sql_owner => "mark";
sql_type => "mysql";
sql_passwd => ""; # No passwd for localhost
}
#######################################################
bundle knowledge tm
{
topics:
any::
#我们需要从某处开始
"Processes" comment => "Programs running on a computer";
"Computers" comment => "Generic boxes",
association => a("run","Services","are run on");
Computers::
"server" comment => "Common name for a computer in a desktop";
"desktop" comment => "Common name for a computer for end users";
Programs::
"httpd" comment => "A web service process";
"named" comment => "A name service process";
Services::
```

```
"WWW" comment => "World Wide Web service",
association => a("is implemented by","httpd","implements");
"WWW" association => a("looks up addresses with","named","serves
addresses to");
#
occurrences:
httpd::
"http://www.apache.org"
represents => { "website" };
}
#########################################################
body association a(f,name,b)
{
forward_relationship => "$(f)";
backward_relationship => "$(b)";
associates => { $(name) };
}

atlas$ mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.67 SUSE MySQL RPM
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database test_map;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE TABLE topics
-> (
-> topic_name varchar(256),
-> topic_comment varchar(1024),
-> topic_id varchar(256),
-> topic_type varchar(256)
-> );
```

一旦你已经创建了数据库，你可以通过输入下面的命令来填充它：

```
host$ /usr/local/sbin/cf-know -f ./unit_knowledge_txt.cf -s
```

你可以验证数据已经插入：

```
mysql> use test_map;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> select * from topics;
+------------+----------------------------------------+-----------
+------------+
```

```
| topic_name | topic_comment | topic_id |
topic_type |
+------------+----------------------------------------+-----------
+------------+
| WWW | World Wide Web service | WWW |
Services |
| named | A name service process | named |
Programs |
| httpd | A web service process | httpd |
Programs |
| desktop | Common name for a computer for end users| desktop |
Computers |
| server | Common name for a computer in a datacent| server |
Computers |
| Computers | Generic boxes | Computers |
any |
| Processes | Programs running on a computer | Processes |
any |
+------------+----------------------------------------+-----------
+------------+
```

此后，你就不需要解析整个的数据来使用这个话题图形。你可以使用一个轻省的的‘驱动脚本’，它足够可以查询数据库的关系。

```
body common control
{
bundlesequence => { "tm" };
}
body knowledge control
{
query_output => "text";
query_engine => "none";
sql_database => "test_map";
sql_owner => "mark";
sql_type => "mysql";
sql_passwd => ""; # No passwd for localhost
}
#########################################################
bundle knowledge tm
{
topics:
any::
"Nothing needed here -- we get everything from the db cache";
```

```
}
```

7.6 查询话题图形

现在你可以直接从数据库查询缓存的话题图形。‘-t’或者‘--topic’选项可以用来输入一个话题的名字。还可以使用‘-r’或者‘--regex’选项来输入任意规则的表示来匹配这些话题。

```
_ _ atlas$ ~/LapTop/Cfengine3/trunk/src/cf-know -f ./unit_knowledge_txt.cf
-t Computers
Topic "Computers" found in the context of "any"
Results:
Explanation: "Generic boxes" (Text)
Topics of the type Computers:
desktop
server
Associations:
Computers "run"
- Computers
Computers "are run on"
- any::Computers
Other topics of the same type (any):
Processes - Programs running on a computer
```

现在如果我们按照下面的步骤：

```
atlas$ ~/LapTop/Cfengine3/trunk/src/cf-know -f ./unit_knowledge_txt.cf
-t httpd
Topic "httpd" found in the context of "Programs"
Results:
Explanation: "A web service process" (Text)
website: http://www.apache.org (URL)
Topics of the type httpd:
(none)
Associations:
httpd "implements"
- Services::WWW
Other topics of the same type (Programs):
named - A name service process
```

在这个例子中，注意这两个结果，一个是 URL，一个是文字文本字符串。还有一个与 WWW 服务的关联。如果我们按照下面的步骤：

```
atlas$ ~/LapTop/Cfengine3/trunk/src/cf-know -f ./unit_knowledge_txt.cf
-t WWW
Topic "WWW" found in the context of "Services"
Results:
Explanation: "World Wide Web service" (Text)
Topics of the type WWW:
(none)
Associations:
WWW "is implemented by"
- httpd
WWW "looks up addresses with"
- named
Other topics of the same type (Services):
（none）
```

为了能以网页格式显示这个，我们将查询的输出变成‘html’；cf-know 将会显示 html 页面。可以通过一个简单的 PHP 脚本创建一个简单的封装脚本来使它成为一个网页，例如：

```
<?php
$arg1 = $_GET['next'];
$cfknow = "/usr/local/sbin/cf-know";
$file = "/path/to/portal/overview.cf";
if ($arg1)
{
system("$cfknow -t $arg1 -f $file");
}
else
{
system("$cfknow -t some_start_topic -f $file");
}
?>
```

在这里，不足的话题也能产生一些图形。。但一个话题必须至少有两个关联才能确保产生一个图表。

7.7 话题图形的具体细节

7.7.1 话题图形的定义

话题图形其实是电子索引，但它们的形成和工作方式类似于网页。一个话题是代表了一个技术的‘主题’，例如，任何你可能想要讨论的事情，摘要的或是物质的，例如，一个‘知识摘要’项目，它可能有许多具体的范例。它可能是一个人，一台机器，一种质量等。

话题可以被分类成不同的类型，称为 topic-types，因而相关的事情能够被整理并且不相关联的事情可以被分开，例如，话题类型能够让我们区分 UNIX 命令的 rmdir 和 Unix 系统调用的 rmdir。

每一种被分类的话题能够进一步地指向大量的参考或者范例，称为事件（occurrences）。例如，一个话题‘computer’的事件可能包括书，网页文档，数据库条目，物质的显现，或者一些其他的信息。一个事件是一个能够举例证明摘要话题的参考。事件参考（Occurrence references）类似于索引中的页码。

一本书的索引通常有‘参见’(‘see also’)的参考，它可以从一个话题指向另一个话题。话题图形允许我们定义任何类型的话题之间的关联形式。与一个普通的索引不同的是，一个话题图形有各种丰富的（可能是无限的）交错的参考类型。例如，

topic_1 ``is a kind of'' topic_2
topic_1 ``is improved by'' topic 2
topic_1 ``solves the problem of'' topic_2

因而这个话题图形的模型有三个层次的容器：

类型 我们将一个话题划分成不同的类型从而消除了具有相同名字的不同话题之间产生的歧义。

话题 一个主题的表现（一个索引词）

事件类型 用于解释一个真正的文档事件是怎样与话题相联系的，该术语用于声明一些东西，例如（讲义，手册，或者示例，定义，照片相册等）

事件 专门的信息资源：这些是指向我们想读的那些实际文档的指针（类似于索引中的页码）。

我们可以很容易地将话题加入到 cfengine 的类中。话题图形也能很容易的加入到承诺者中。事件也能够映射到不同类型的承诺者中。这三个标签区分了不同意义的间隔尺度层次。类型代表了一系列的话题，这些话题还包含一系列的事件。其中，主要的话题来自于他们通过关联性来形成网络的能力。

信息模型化的经典方法是建立非叠加的对象的层次化分解结构。数据被强制地分在非叠加的容器中，这些容器通常是具有过度的限制性的。话题图形使得我们能够避免各种类型的错误，例如导致类似带有成千上万严格的非叠加类型分类的共同信息模型（CIM）的畸形。

每个话题使得我们能够有效地‘表现’信息事件从而突出关于该话题的恰当的概念。

7.7.2 cf-know

Cfengine 的知识代理器 cf-know 使得你能够对于知识和它们的内部关系作出承诺。它不是专门地一个通用的话题图形语言：而是它能够为管理一个知识库提供一种强大的配置语言，从而把它编译成一种话题图形。

一个标准的 ISO 话题图形模型由于太丰富而不能成为一个系统知识管理的有用工具。然而，这就是强大的配置管理能够帮助简化这个过程的所在之处：为一个话题图形编码是配置中一个复杂的问题，而 cfengine 恰好能解决这个问题。Cfengine 的话题图形承诺有下面的形式：

```
bundle knowledge example
{
topics:
topic_type_context:: #标准的容器
"Topic name" #简短的话题名称
comment => "Use this for a longer description",
association => a("forward assoc to","Other topic","backward assoc");
"Other topic";
occurrences:
Topic_name:: # Topic
"http://www.example.org/document.xyz" # URI to instance
represents => { "Definition", "Tutorial"}; # sub-types
}
```

关联体的模板如下：：

```
body association a(f,name,b)
{
forward_relationship => "$(f)";
backward_relationship => "$(b)";
associates => { $(name) };
}
```

承诺理论给话题图形本体增加了一个清晰的构架，这是非常有用的，经验表明脆弱的概念模型会导致很差的知识图形。

7.8 作为话题图形的模型配置承诺

在 cfengine 中我们能够将话题图形塑造成承诺模型；剩下的问题就是关于怎样使用话题图形形成模型配置从而 cfengine 的使用者能够使用网页浏览器查找浏览存档的承诺，并且能够看清原本孤立和被分割的规则之间的所有关系。这个将为 cfengine 的软件形成基本的语义配置管理数据库（sCMDB）。实现这个目标的关键是将话题图形的配置看成话题抽象空间生成的大量承诺，并且能够将每个承诺变成一个承诺元组（meta-promise），从而将配置模型成一个

带有各种关联的话题。看以下的 Cfengine 的承诺。

```
bundle agent update
{
files:
any::
``/var/cfengine/inputs" -> { ``policy_team", "dependent" },
comment => ``Check policy updates from source",
perms => true,
mode => 600,
copy_from => true,
copy_source => /policy/masterfiles,
compare => digest,
depth_search => true,
depth => inf,
ifelapsed => 1;
}
```

Cfengine 可以将系统配置承诺映射到大量的其他承诺提议，用于 cf-know 的代理器。隐藏一些细节，我们有：

```
type_files::
"/var/cfengine/inputs"
association => a("promise made in bundle","update","bundle
contains promise");
"/var/cfengine/inputs"
association => a("specifies body type","perms","is specified in");
"/var/cfengine/inputs"
association => a("specifies body type","mode","is specified in");
"/var/cfengine/inputs"
association => a("specifies body type","copy_from","is specified
in");
# etc ...
occurrences:
_var_cfengine_inputs::
"promise_output_common.html#promise__var_cfengine_inputs_update_cf_13"
represents => { "promise definition" };
```

在该映射中要注意，实际的承诺（被看作一个现实世界的实体）是一个话题‘承诺’事件；同时每个承诺作为一个不同的话题讨论，使得实体关系模型形成现实世界数据的模型元组。相反的，话题本身成了该承诺模型中的配置项目或者‘承诺者’。作用是创建一个浏览策略的导航语义网页；这可以使用一个小的标准概念的实体来存档策略的结构和目的，并且能被人类域名专家们无限地进行拓展。

7.9 附件：技术的前提要求

7.9.1 知识基本要求

你需要一台电脑来运行一个 apache 的网页服务器，并具有一些激活的服务器页面技术作为 cfengine 的封装。我们示例假设这个封装是一个使用 PHP 的网页服务器。你还需要一个后端的一个 SQL 数据库（用于 cfengine 而不是 PHP）。Cfengine 当前支持 MySQL 和 PostreSQL。一个 PHP 激活的索引页面包含了一个封装，能够运行 cf-know 组件，并且将这个页面提给网页浏览器。

为了能够工作，你创建 cfengine 进行编译的时候需要数据库的支持。MySQL 和 PostgreSQL 数据库目前是支持的。你在安装 php 的时候不需要这些数据库的模块，因为 cfengine 能够直接与数据库对话，但 cfengine 在编译的时候需要数据库的支持。

你可以创建一个 cf-know 的控制体来指向一个数据库：

```
body knowledge control
{
# Decide the name of a local database
sql_database => "cf_knowledge_map";
sql_owner => "root";
sql_passwd => "";
sql_type => "mysql";
sql_server => "localhost";
}
```

在开源版本中，你得为 cf-know 创建 SQL 数据库进行手动写入，然后记得为上面的使用者授权，否则 cf-know 将不能加载数据。这个数据库必须有下面的表格。

```
CREATE TABLE topics
(
topic_name varchar(256),
topic_comment varchar(1024),
topic_id varchar(256),
topic_type varchar(256)
);
CREATE TABLE associations
(
from_name varchar(256),
from_type varchar(256),
from_assoc varchar(256),
to_assoc varchar(256),
```

```
to_type varchar(256),
to_name varchar(256)
);
CREATE TABLE occurrences
(
topic_name varchar(256),
locator varchar(1024),
locator_type varchar(256),
subtype varchar(256)
);
```

7.9.2 知识库问题解决

Cfengien Nova 将试图自动地创建数据库和所有的表格。如果这没有发生的话，可能的解释是数据库服务器没有权限来做这件事情。

连接到数据库并给它授权，例如：

```
mysql
USE mysql_mananage_point
GRANT ALL on * to root;
```

or

```
psql postgres_manage_point
GRANT ALL on * to root;
```

8. 更多...

你可以寻求广泛的帮助，示例和文档，这是Cfengine 商业版本中功能的一部分。登录网站 www.cfengine.com 查看更多细节。