Konfigurations- und Change-Management mit Bcfg2

Robert Gogolok <gogo@cs.uni-sb.de>
Marko Jung <mjung@markojung.net>

01.03.2007

"Daemons loose in system."
(S. Travaglia, B.O.F.H. Excuses)

Konfigurationsverwaltung ist ein Thema, welches eng mit Systemverwaltung und Sicherheit verbunden ist. Mit der stetig wachsenden Zahl der eingesetzten Systeme steigen auch die Anforderungen an die Systemadministratoren, welche die korrekte Konfiguration dieser sicher stellen müssen. Dabei handelt es sich in der Regel nicht nur um eine einmalige Installation, sondern um regelmäßige Software-Updates und -Patches und die damit verbundenen Konfigurationsanpassungen. Ohne diesen ständigen Wartungsaufwand können schnell Sicherheitslücken entstehen oder Systeme nicht verwendbar werden.

Es existiert eine überschaubare Anzahl verschiedener Tools, welche diesen Prozess beschleunigen und weniger fehleranfällig gestalten, weshalb durch deren Einsatz die Zuverlässigkeit und Sicherheit komplexer Computernetzwerke drastisch gesteigert werden kann. Trotz der breiten Anerkennung der Tatsache, dass Systeme zur Konfigurationsverwaltung die Abläufe der Systemadministration bedeutend verbessern können, haben diese bis heute keine weite Verbreitung erlangt.

Bcfg2

Bcfg2 [1] wurde ursprünglich als internes Forschungsprojekt der Mathematik- und Informatikabteilung des Argonne National Laboratory [2] entwickelt und ist ein noch relativ junges Werkzeug dieser Kategorie. Nachdem sich die grundlegenden Design-Prinzipien in verschiedenen Umgebungen als stabil erwiesen hatten, wurde Bcfg2 veröffentlicht.

Bcfg2 ermöglicht einen sanften Migrationspfad zur Einführung einer Konfigurationsverwaltung, da es, auf einem installierten Basissystem aufbauend, die Systemkonfiguration verwaltet. Die Basisinstallation der Systeme geschieht immer unabhängig von Bcfg2. Bei der Einführung des Tools können anfänglich erst Teile der Konfiguration in das Bcfg2-System integriert werden. Allmählich vervollständigt sich die Konfigurationsspezifikation und man erhält ein allumfassendes Konfigurationsmanagement.

Architektur

Wie die meisten Applikationen aus dem Konfigurationsmanagement-Umfeld ist auch Bcfg2 eine Client/Server-Applikation. Der Server erstellt aus der zentralen Konfigurationsspezifikation clientspezifische Konfigurationsbeschreibungen, welche von den Clients abgerufen werden können. Nach einer Katalogisierung des lokalen Systemzustandes führt der Client die notwendigen Konfigurationsänderungen durch, um der Spezifikation zu entsprechen.

Client

Ein Konfigurationsprozess besteht aus drei Phasen: Zuerst verbindet sich der Client mit dem Server und ruft eine Reihe von Tests (sog. *Probes*) ab. Diese Tests werden ausgeführt und die Ergebnisse zurück

an den Server gesendet. Mittels dieser Tests können beispielsweise lokale Änderungen am System, Hardwareinformationen oder lokal gespeicherte Daten abgerufen werden.

In der zweiten Phase lädt der Client eine konkrete Konfigurationsbeschreibung vom Server. Diese besteht aus Konfigurationseinträgen aus der Menge der sechs Basistypen *ConfigFile, Directory, Package, Permissions, Service* und *Symlink*. Als besondere Konfigurationstypen existieren zusätzlich *Action-* und *PostInstall*, welche Befehle nach, im *Action*-Fall auch vor, der Aktualisierung einzelner Konfigurationsabschnitte ausführen können.

Nachdem der Client die Konfigurationsbeschreibung erhalten hat, führt er eine lokale Inventur durch. Dabei werden alle vorhandenen Konfigurationen mit denen der neuen Spezifikation verglichen und alle nicht konformen Einträge zur späteren Korrektur vorgemerkt. Danach werden alle lediglich lokal vorhandenen Einträge überprüft. Diese sogenannte *two-way verfication* erlaubt es, die Konfiguration zu verifizieren und die Konfigurationsspezifikation zu vervollständigen.

Schliesslich beginnt der Client mit der Korrektur der Fehlkonfigurationen, wobei Abhängigkeiten zwischen Konfigurationselementen automatisch aufgelöst werden. Dieser Prozess wird so lange wiederholt, bis kein weiterer Fortschritt mehr gemacht werden kann. Mit Abschluss der Konfigurationsphase ist ein Zustand erreicht, welcher auch bei einem weiteren Aufruf des Clients keine neuen Änderungen bewirken würde.

In der abschließenden dritten Phase generiert der Client eine Nachricht, welche neben dem Systemstatus weitere Details, wie beispielsweise die Anzahl der korrekten, modifizierten und fehlgeschlagenen Konfigurationseinträge oder Informationen zu zusätzlichen Konfigurationen auf dem System, enthält. Diese Nachricht wird zum Server gesendet, welcher diese für Berichte (*Reports*) verwenden kann.

Server

Die zentrale Komponente des Bcfg2-Systems ist die Konfigurationsspezifikation. Administratoren beschreiben hiermit die gewünschten Zielkonfigurationen aller verwalteten Systeme. Aufbauend auf diesen Informationen generiert der Bcfg2-Server primär clientspezifische Konfigurationen. Darüber hinaus verwaltet er die Metadaten und konkrete Konfigurationsinformationen und erstellt Berichte über den Status aller Clients in Form von Webseiten, RSS-Feeds und E-Mails.

Beim Einsatz von Bcfg2 können sich die Clients in einem beliebigen Anfangszustand befinden. Durch die Ausführung von Bcfg2 werden Unterschiede zwischen dem Zustand des Clients und der Spezifikation des Servers korrigiert. Alle durchgeführten Anpassungen und die zugehörigen Zielzustände werden in Client-Statistiken beobachtet. Durch die Verwendung der Konfigurationsspezifikation und der Client-Statistiken können die wichtigsten Informationen aller Clients für den Administrator aus rein serverseitig vorhandenen Daten generiert werden: Die spezifizierten Zustände der Systeme, nicht konforme Clients und eine Liste der konkreten Fehlkonfigurationen. Mittels übersichtlicher Berichte erhält der Administrator jederzeit einen guten Überblick des verwalteten Computernetzwerks.

Die Rolle des Servers während der Client-Konfiguration ist, bis auf die Erzeugung der clientspezifischen Konfigurationsbeschreibung, offensichtlich: Er sendet die Tests und Konfigurationsinformation zum Client und empfängt dessen Statusbericht.

Der Prozess der Konfigurationsgenerierung beginnt mit der Suche der Metadaten für den eine Konfiguration anfragenden Client. Diese Daten werden verwendet, um eine abstrakte Konfiguration zu erstellen. Diese ist eine Vorlage, welche alle benötigten Konfigurationseinträge des Zielsystems ohne inhaltliche Informationen beinhaltet. Beispielsweise enthalten *ConfigFile*-Elemente einen Dateinamen, jedoch keinen Dateiinhalt oder Informationen zu Zugriffsrechten und Besitzern.

Nachdem die abstrakte Konfiguration konstruiert wurde, müssen an jeden Eintrag konkrete Informationen gebunden werden. Der Bcfg2-Server verwendet hierzu eine Vielzahl von Generatoren. Diese werden zum Server-Start initialisiert und registrieren ihre Fähigkeit, konkrete Informationen zu bestimmten Konfigurationseinträgen bereitzustellen. Um beim vorherigen Beispiel zu bleiben, könnte ein Generator melden, dass er Informationen für die Datei `/etc/gdm/gdm.conf` bereitstellen kann. Wenn der Server Daten an ein Element binden muss, lokalisiert er den entsprechenden Generator und ruft ihn auf. Dabei kann der Generator verschiedene Verfahren anwenden, um für den Client gültige Werte be-

reitzustellen.

Die Verwendung von Generatoren eröffnet eine Vielzahl interessanter Möglichkeiten. Als wichtigstes Feature sei die Option genannt, beliebige Repräsentationen als Spezifikation zu verwenden. Somit kann man Bcfg2 als ideale Testplattform für Spezifikationssprachen verwenden. Darüber hinaus kann fast beliebig Logik in den Prozess der Konfigurationskonstruktion integriert werden. Dies bedeutet, dass jedes vorstellbare Schema als Grundlage zur Datenkonstruktion verwendet werden kann.

Metadata/clients.xml:

```
<Clients>
  <Client profile="desktop" name="foo.example.com" pingable="N"/>
  <Client profile="desktop" name="bar.example.com" pingable="Y"/>
  <Client profile="desktop-testing" name="test.example.com" pingable="N"/>
  <!-- .. --!>
</Clients>
```

Metadata/groups.xml:

```
<Groups>
  <Group name='desktop' profile='true'>
    <Bundle name='motd'/>
    <Bundle name='networking'/>
    <Group name='gnome-desktop'/>
    <Group name='ubuntu-stable'/>
  </Group>

  <Group name='desktop-testing' profile='true'>
    <Group name='gnome-desktop-testing'/>
    <!-- ... --!>
    <Group name='ubuntu-testing'/>
  </Group>

  <Group name='gnome-desktop-testing'>
    <Group name='gnome-desktop'/>
    <!-- ... --!>
  </Group>

  <Group name='ubuntu-stable'  toolset='debian'/>
  <Group name='ubuntu-testing' toolset='debian'/>
</Groups>
```

Bundler/motd.xml

```
<Bundle name='motd' version='2.0'>
  <Package name='login' />
  <ConfigFile name='/etc/motd' />
</Bundle>
```

Abbildung 1: Beispiel einer Metadaten-Definition

Metadaten und Bundler

Metadaten sind die Grundlage der Konfigurationsspezifikation. Sie dienen der Zuweisung der Konfigurationsinformation zu allen mittels Bcfg2 verwalteten Systemen.

Jeder Bcfg2-Client muss in der Datei `clients.xml` definiert werden. Hier werden die Clients anhand deren DNS-Namen einer Basisgruppe (*Profile*) zugewiesen. Die Basisgruppen werden in der Datei `groups.xml` näher beschrieben. Jedes Profil fasst verschiedene Konfigurationsgruppen (*Bundles*) zusammen und kann darüber hinaus weitere Gruppen enthalten. Dabei gibt es keine Beschränkung der Rekursionstiefe. Die zusätzlichen Gruppen sind dabei ein probates Mittel, die Konfigurationsspezifikation übersichtlich zu gestalten und Code-Duplikation zu vermeiden.

Die zuvor erwähnten *Bundles* fassen Konfigurationseinträge zu einer logischen Einheit zusammen. Dadurch können beispielsweise alle zu einem Systemdienst gehörenden Konfigurationsdateien und Softwarepakete gruppiert und unter einem Namen verfügbar gemacht werden. Jedes Bundle wird in einer gleichnamigen Datei im Bundler-Verzeichnis abgelegt. Einige gut kommentierte, einführende Beispiele sind im Projekt-Wiki [3] zu finden.

Zusätzlich können globale oder gruppenspezifische Basiskonfigurationen definiert werden. In Abbildung 1 ist ein einfaches Beispiel der Metadaten und Bundle-Definition aufgeführt.

Spezifikation und Generatoren

Die Vielzahl verfügbarer Generatoren ermöglicht es, gleichzeitig verschiedene Sprachen mit diversen Semantiken zur Erstellung der konkreten Konfigurationsinformation zu verwenden. Somit kann für jeden Konfigurationseintrag der am besten geeignete Generator eingesetzt werden. Jeder Generator besteht aus einem ladebaren Python-Modul und in der Regel einem Datenrepository. Im Folgenden werden exemplarisch einige der gängigsten Generatoren vorgestellt.

Cfg

Das *Cfg*-Plugin implementiert ein Dateisystem-Repository zur Produktion der konkreten Konfigurationsinformationen. Für jede durch das Plugin verwaltete Konfigurationsdatei, beispielsweise `/etc/gdm/gdm.conf` wird ein dem Pfad entsprechendes Verzeichnis `./etc/gdm/gdm.conf/.` im Datenrepository erstellt und konkrete Konfigurationsdateien dort abgelegt.

Jede dieser Dateien enthält Konfigurationsinformationen für eine Teilmenge der Clients. Die Zuweisung der Dateien zu Systemen geschieht mittels deren Namen. Alle Namen beginnen mit dem Basisnamen der Konfigurationsdatei. Ohne weiteres Suffix gilt diese als global gültige Konfiguration. Zur Definition spezieller Konfigurationen erstellt man Dateien mit einem Namenssuffix, welches den Metadatentyp (Hostname, Gruppe), den zugehörigen Namen und eine Priorität (zum Auflösen von Konflikten) enthält.

Beispielweise könnte neben der Standardkonfiguration `gdm.conf` auch die Datei `gdm.comf.H_foo.example.com` existieren, welche die hostspezifischen Informationen des Clients *foo.example.com* enthält. Ebenso könnte man eine Gruppe *bar* durch die Datei `gdm.conf.G50_bar` mit abweichender Konfiguration versehen.

Dabei können einfache Basisoperationen, wie beispielsweise Dateikonkatenation oder Ersetzung, zur Kombination globaler und spezifischer Konfigurationen genutzt werden.

TCheetah

Auf der gleichnamigen Template-Sprache *Cheetah* [4] basierend, existiert der TCheetah-Generator. Dieses Plugin ist aufgrund seines großen Funktionsumfangs sehr komplex, bietet jedoch die maximale Flexibilität, da man den vollständigen Instruktionssatz des Cheetah-Systems verwenden kann.

Cheetah ermöglicht es, Anweisungen direkt in die Konfigurationsdateien einzubetten, welche von einfachen String-Operationen über Flusskontrolle bis hin zu vollständigen Python-Anweisungen reicht. Bei Verwendung einer Datenbankverbindung, können somit weitere Datenquellen zur Konfiguration erschlossen werden. Beispielsweise kann man automatisch die DHCP-, DNS- und NIS-Konfiguration aus den gleichen Ursprungsdaten (Datenbank, Verzeichnisdienst, o.ä.) generieren. Abbildung 2 zeigt die Erzeugung der Konfigurationsdatei `/etc/network/interfaces` unter Verwendung einer PostgreSQL-Datenbank.

```
#from Bcfg2.Server.dbconnection import DBPgConnection
#silent result = DBPgConnection().execute("SELECT ip, netmask, broadcast, gateway \
    FROM hosts  WHERE hostname = '%s'" % $self.metadata.hostname)
auto eth0
iface eth0 inet static
    address $result[0]
    netmask $result[1]
    broadcast $result[2]
    gateway $result[3]
```

Abbildung 2: TCheetah-Beispiel zur Konfiguration von `/etc/network/interfaces`

Services

Der Service-Generator stellt eine sehr komfortable und einfache Möglichkeit zur Verfügung, Systemdienste, wie den OpenSSH-Server, zu aktivieren und deaktivieren. Seine Spezifikation besteht aus mehreren XML-Dateien, in welchen, basierend auf den Metadaten, für jede Gruppe Informationen zu allen Services definiert werden können.

Packages

Zur plattformunabhängigen Softwareverwaltung existiert der *Package*-Generator. Dieser verwendet XML-Dateien mit Informationen aller verfügbaren Pakete, welche mit der Paketauswahl eines Installationsservers synchronisiert werden können. Bei Verwendung mehrerer Installationsserver können diesen Prioritäten zugewiesen werden, um zum Beispiel eigene Updates mit einer höheren Relevanz als die Sicherheits-Updates des Herstellers zu behandeln.

Der Bcfg2-Client vergleicht die global spezifizierten Software-Versionen mit den lokal vorhandenen und führt ggf. Up- und Downgrades durch. Das Einfrieren einer speziellen Softwareversion ist durch Definition dieser mit höchster Priorität ebenfalls möglich.

Zusätzlich existieren noch weitere Generatoren zum Erstellen von Verzeichnissen, Symlinks und vielem mehr. Neben den in Bcfg2 selbst vorhandenen Generatoren besteht zusätzlich die Möglichkeit, eigene Plugins zu schreiben, die Python-Funktionen als Generatoren an einzelne Konfigurationselemente binden.

Ausblick

Da Bcfg2 aus einem Forschungsprojekt hervorgegangen ist und durch eine sehr vitale, internationale Community entwickelt wird, kann man viele Innovationen in den kommenden Monaten erwarten. Fast wöchentlich werden neue Konzepte in das System integriert, wie beispielsweise öffentliche Gruppen, welche es dem Client erlauben, sich beim Aufruf an diese zu binden. Vor kurzem hat sich auf der Mailingliste ein neues Team formiert, das sich aktiv um die Verbesserung der Dokumentation bemühen wird. Der Projektleiter Narayan Desai ist sehr aufgeschlossen, freut sich über Anregungen und Feedback und gibt selbst viel Unterstützung beim Einstieg in das System.

Auch wenn Bcfg2 aktuell noch keinen sehr hohen Bekanntheitsgrad erreicht hat, wird das Tool bereits heute in vielen Produktivumgebungen eingesetzt und besticht durch seine extrem hohe Flexibilität und einfache Anpassbarkeit an lokale Bedürfnisse.

Literatur

[1] Bcfg2 Trac
`http://www.bcfg2.org/`

[2] Argonne National Library,
Mathematics and Computer Science Division
`http://www-new.mcs.anl.gov/new/`

[3] Bcfg2 Annotated Examples
`http://www.bcfg2.org/wiki/AnnotatedExamples`

[4] Cheetah Template Engine
`http://www.cheetahtemplate.org/`

Alle Links wurden zuletzt am 10.02.2007 geprüft.