1. Configuration and You

Configuration is the medium in which system administrators work. Ranging from the most sweeping decisions to minor troubleshooting of software problems, the act of helping people use computers necessitates a stream of decisions that result in the configuration and software infrastructures that each of us uses on a daily basis. Configuration management is a process whereby administrator interactions with configuration are streamlined and simplified. As the scale of infrastructures grows and the complexity of software systems increases, the cost of system administration rises rapidly. Automation techniques, such as configuration management, can reduce these costs by streamlining common tasks. Automation also implements rigorous processes; automated processes are usually performed more consistently than manual ones.

Our intention is to enable you to understand the basics of configuration management and how to use Bcfg2 to achieve common configuration goals. With this booklet, you will be able both to design appropriate configuration processes and policies for your site and to analyze configuration processes to assess their effectiveness.

1.1 The Configuration Problem

In this booklet, we define configuration as the union of software factors that influence the behavior and performance of computer systems. This is a fairly expansive definition of configuration.

When administrators work with software systems, their activities fall into two major categories. The first uses domain-specific software information. These activities require a high level of knowledge about the capabilities of particular software and the semantics of their configuration files. The tasks are common and may involve the configuration of packages such as the Linux kernel, the Apache Web server, or any of myriad other software packages. This area involves understanding the ways to properly activate and use given software packages.

The second category is what we consider to be configuration. These tasks involve building coherent services from the software components described earlier. Here, high-level requirements become relevant; these may be technical or nontechnical. Operational requirements figure into these systems as well; correctness, security, and robustness are frequent goals of the configuration process.

Activities in both of these categories are required in order to achieve results; the second is clearly dependent on the first. At the same time, we find that the second category poses a much larger and more persistent set of problems for administrators. Hence, these activities are the explicit targets of configuration management, and specifically the architecture of Bcfg2.
1.1.1 Interacting with Configuration

Interacting with configuration is unwieldy for several reasons. Traditional configuration is distributed, potentially existing on every device in a network. For example, firewall configurations on network devices contribute to the operational state of services as much as any configuration located on the server itself. Configuration itself is frequently disorganized and inconsistent, making it hard to troubleshoot systems.

Moreover, interaction with traditional configuration scales poorly along several different axes. As the number of devices grows, so does the sheer quantity of configuration. The amount of configuration diversity present also increases the cost of configuration, as it reduces the amount of commonality between systems. Additionally, administrator count causes scalability problems in a counterintuitive fashion. As more administrators contribute to the operation of systems, more coordination is required. Traditional configuration is poorly suited to aid in this coordination.

1.1.2 The Configuration Process

Configuration itself isn't a monolithic process. Administrators interact with configuration in a number of different ways:

- **Creation:** Administrators create new configuration whenever new services or systems are deployed.
- **Modification:** Administrators modify existing configuration as service requirements change or as new software is released. Security updates frequently provide an urgent example of this category.
- **Analysis:** Administrators need to analyze existing configuration in order to acculturate themselves to unfamiliar surroundings. Also, administrators frequently need to reacquaint themselves with configurations during troubleshooting.
- **Validation:** Administrators need to convince themselves or others that the overall configuration adheres to organizational policies.

Each of these tasks can be greatly eased through new configuration-based functions. For example, configuration creation becomes considerably simpler if existing configuration can be reused. Likewise, modifications can be automated with a machine-consumable definition of configuration state. Modifications can be further streamlined if the specification format is one that can be generated robustly; this allows the construction of programs that generate configuration updates in response to external events.

Analysis poses one of the most pressing issues facing organizations. The combination of staff turnover with a lack of sufficient cross-training and up-to-date documentation requires administrators to work in situations where they need to discover how systems are configured. This requirement leads to a large spin-up time for new employees and a higher time to solution when troubleshooting.

Exacerbating this situation is the need for configuration validation by external entities, be they management, funding sources, or regulators. Anyone who has been through an audit understands the sheer volume of data that must be collected and presented. Traditional configurations are simply difficult to audit.

Together, all of these issues suggest that a different approach is needed.
1.2 Toward a Solution

Manipulation of traditional configuration is so fault-prone that an alternative approach is required. Usually, configuration systems focus on easing the performance of configuration changes, without addressing the other areas of the configuration problem. Much more expansive than those typically held by configuration management systems, the Bcfg2 approach is to build a useful representation of configuration goals that can be correlated with current configuration state. This representation needs to be accurate, compact, verifiable, and centralized. Moreover, these goals must be reconcilable with the current state of systems in a flexible, efficient, and intuitive manner. These requirements are much more expansive than those of most configuration management systems.

Following the requirements that form the basis for Bcfg2’s architecture and approach, we have attempted to streamline the four aspects of configuration described in the previous section.

Bcfg2’s model stores two bodies of information: the configuration state of managed systems and a series of configuration goals, describing the desired configuration state. These goals have three important characteristics: they are verifiable, installable, and discoverable. The definition of goals makes them directly comparable with current system state. Whenever this comparison finds inconsistencies, reconciliation is needed.

Depending on the environment, inconsistencies can imply one of two situations. Either new goals have been specified and should be enforced on clients, or new configuration has been introduced on managed systems and should be reflected in the goals. In many cases, inconsistencies can be resolved by convention (i.e., “No one should make configuration changes directly on servers”), but exceptions frequently occur for legitimate reasons. Therefore, Bcfg2 supports bidirectional flow of configuration data between the manager and managed systems. This is a key, and unique, feature of Bcfg2.

The isolation of goals from state also makes the configuration process transparent. This added information makes it possible to make more effective and informed decisions about management policy.

1.2.1 What Is Bcfg2?

Bcfg2 provides a mechanism for describing system configurations for large, heterogeneous environments of systems. More important, it collects data to streamline the configuration discovery process. Most environments are pre-existing; the introduction of a new configuration tool does not reduce the importance of existing infrastructure. Indeed, it is not always feasible to rebuild all client systems in order to properly manage them.

As a tool, Bcfg2 has been designed to work in a noninvasive manner. Although it can be used as a prescriptive and assertive tool, with complete control over client configurations directed from a central location, it can also be used to selectively manage parts of client configurations and track configuration changes performed on clients. This architecture imposes no deployment strategy on administrators, leaving them free to deploy Bcfg2 in the role that makes sense in a particular environment. In pre-existing environments, this means busy administrators can start by managing their most important configuration elements while working their way toward a more complete configuration state.
Bcfg2 is designed to help administrators get their machines under configuration management quickly. It doesn't require complex modeling and administration paradigm shifts to use effectively, which means it is very accessible to all administrators and environments. Used correctly, it offers administrators benefits that become clear very early in the deployment process.

### 1.3 Deploying Configuration Management

Effective use of software requires understanding the software itself and determining a strategy that is appropriate for a given situation. In this, Bcfg2 is not an exception.

Most system administrators do not have the luxury of deploying a new infrastructure from scratch. The cost of rebuilding all currently existing services and resources is often prohibitively high. Given this limitation, configuration management processes must be layered on top of existing infrastructure, which may be disorganized, pathological, or not well understood.

The deployment of configuration management processes has two main benefits. The first is simple: decreasing the cost of the configuration process. The driving need for this improvement is clear: administrators are expected to manage increasing numbers of systems with fixed or decreasing staffing. The second benefit is a side effect of performing the first: the resulting configuration specification provides a central point of information about systems, their roles, and their required configurations.

Centralized configuration knowledge provides a number of benefits to administrators, particularly in groups. It provides a shared repository of institutional knowledge that administrators can readily use. It functions as a configuration plan of record for systems. Moreover, it injects visibility into the configuration process. Previously, this process was performed in relative darkness. This new information can be used both to assess the performance of the configuration process and to better understand the costs associated with system management.

All of these benefits combine to provide a safety mechanism for system administrators. Configuration management is a potent tool for positive change in system administration environments. However, it has an unparalleled destructive capacity. For this reason, administrators must be provided sufficient information to make informed decisions. Using this information effectively can minimize the danger of having administrators with an incomplete understanding of deployed configurations.

In view of this setting, configuration management processes must first focus on safety, before any other considerations. By safety, we mean that both the configuration management system and processes are designed to ease configuration discovery and build robust, configuration-sensitive control processes. Bcfg2 has been designed to provide these facilities; they are discussed in Section 3.3.
1.4 How to Read This Booklet

This booklet has eight chapters. They are topically split between conceptual overviews and hands-on examples. Chapter 2 provides a high-level architectural overview of Bcfg2. We describe each of the major components of Bcfg2 in turn, noting their roles and interaction points, and highlighting important design features. This chapter provides the big picture; readers who wish to begin by understanding the high-level architecture should start here.

Alternatively, readers wanting to dive in can begin with Chapter 3. This chapter demonstrates how the architecture is used in practice. Here, we provide concrete details showing how Bcfg2 works and how to use it effectively.

Chapter 4 describes how Bcfg2 can be used to accomplish a series of common tasks. Each task highlights several important aspects of working with Bcfg2. Chapter 5 explains the mechanisms that can be used to troubleshoot Bcfg2. Chapter 6 provides an advanced set of examples that use a wide range of Bcfg2 features to achieve sophisticated goals. Chapter 7 offers an overview of how Bcfg2 records system-wide status information and detects any deviation from, or inability to reach, the desired state after the system has been modified. Finally, Chapter 8 provides tips on how to deploy Bcfg2 after configuration policies have been put in place and configuration tasks have been fully automated, along with some additional resources.

Detailed information about low-level formats and processes is provided in two appendices; Appendix A describes the file formats used by the Bcfg2 server, while Appendix B describes the methods used for client identification and authentication.