RoboBee
aka
Simple Server Control

User Manual

revision 1

RoboBee aka Simple Server Control

User Manual

Erwin Müller

Revision 1
03/31/2016

Revision 1 03/31/2016

CONTENTS

1 Introduction **1**
1.1 Idea 1
1.2 Expert Knowledge 1
1.3 OSGi Container 1
1.4 Domain Specific Language 1

2 Vision of the Solution **3**
2.1 Vision Statement 3
2.2 Major Features 3
2.3 Dependencies 4
2.4 Related Projects 4

3 Use Cases **5**
3.1 5

4 Script Statements **7**
4.1 Dhclient 7
4.1.1 Dhclient Statements 7
4.1.2 Dhclient Example 7
4.2 Database Script 9
4.2.1 Database Statements 9
4.2.2 Database Example 10

CHAPTER 1 INTRODUCTION

....

1.1 Idea

The idea to create a new server configuration tool was born out of a simple concideration, the configuration of each new server is always the same for the same stack, and the stack is always similar, i.e. web server, database server, email server, DNS server. So, I asked myself, what if I could just tell the computer to install and configure me the stack on the server, without for me to actually have any knowledge how to configure it. The user should be able to tell the computer to just install and setup, for example, a webserver without to have the actual knowledge how to do it. All the knowledge how to install and configure the server should be contained in the application that is installing and configures the server.

....

1.2 Expert Knowledge

RoboBee will contain expert knowledge to install and configure the server with the needed services and applications. The expert knowledge will contain best practices in regards to security and it will updated regularly to fix bugs and to react to new security issues. To ensure a constant flow of updated expert knowledge, the OSGi technology[1] will be used for RoboBee. OSGi container allows for seamlessly updates of the modules that contain the expert knowledge from a central repository.

....

1.3 OSGi Container

There are multiple OSGi container implementations[2] and RoboBee will be able to run on those, but also it will provide its own container, Karaf[3]. Based on the Karaf container, RoboBee can be installed on a GNU/Linux server just as any other application and run as a service in the background. The communication between RoboBee and the user is done via a SSH connection and the OSGi console.

....

1.4 Domain Specific Language

The user is expected to write simple scripts in a domain specific language (DSL) based on Groovy that communicate the wishes of the user to RoboBee. A syntax must be followed that groups the server services into categories, like web server, DNS server, mail server, etc. and also have application categories like Wordpress, Drupal, Postfix, MySQL, etc. The user is assumed to have only very basic knowledge of those services and applications, for example, the user should know that the MySQL database server have an administration user, provides databases and have users that have permissions to create tables in those databases. But this kind of knowledge is not expert knowledge and is expected

[1] `https://www.osgi.org`
[2] Apache Felix, Equinox OSGi, Knopflerfish, etc.
[3] `http://karaf.apache.org`

from the user to have. Expert knowledge in this case would be how to install the MySQL server, configure the server, create databases and users on the server and grant the users permissions to those databases.

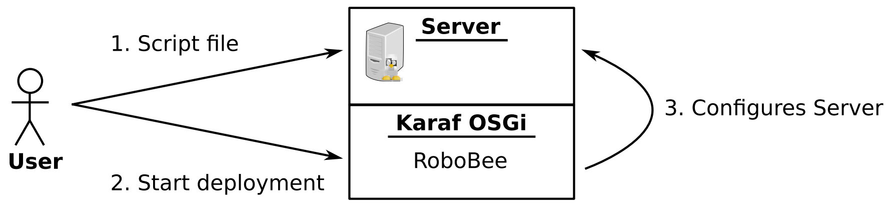

Figure (1.1) RoboBee overview.

CHAPTER 2 VISION OF THE SOLUTION

. . . .

2.1 Vision Statement

The RoboBee (aka Simple Server Control) is about to fully configure a server by defined profiles and script files using a domain specific language (DSL). The goal of the project is to follow high level user specifications to install and configure specific services on the server. RoboBee will follow the "Ask, don't tell"[1], the "Convention over configuration"[2] and the "Principle of least knowledge"[3] philosophies and principles.

"Ask, don't tell", the user is expected to ask the different RoboBee services to install and configure the server. The services will install and configure the server according to the wishes of the user and will not expect from the user to know how to do it.

"Convention over configuration", the user is expected to follow the convention set from RoboBee to reduce the amount of needed configuration. Basically, to follow configuration file name conventions, and to follow protected configuration of the server.

"Principle of least knowledge", the user is not expected to have any expert knowledge of how to install or configure the server, all expoert knowledge is contained in RoboBee.

[1] https://pragprog.com/articles/tell-dont-ask
[2] https://en.wikipedia.org/wiki/Convention_over_configuration
[3] https://en.wikipedia.org/wiki/Law_of_Demeter

. . . .

2.2 Major Features

F1-a. Install Server Services, installs server services on the server. Server services are usually DNS server to resolve DNS names (Bind, MaraDns), Httpd server to serve websites (Apache, Tomcat), proxy server for caching (Nginx, Squid), firewall to block harmful packages, MTA (Postfix, Exim, qmail), MDA (Dovecot, Courier), etc.

F1-b. Configures Server Services, configures server services on the server according to the whishes to the user. The manually edited configuration will be preserved as far as possible.

F2-a. Installs Applications, installs server applications that are run on the server. Applications are usually run on already installed and configured server services like Apache or Tomcat. Those are, for example, Wordpress, Drupal, Squirrelmail, Roundcube, Redmine, JIRA, etc.

F2-b. Installs Applications, configures the applications on the server according to the whishes to the user. The manually edited configuration will be preserved as far as possible.

F3. Profiles, the system dependent configuration is stored in profiles and can be selected without the need to modify the script files. Different systems have different commands to configure the server (SysVInit, upstart, systemd, etc.) and have different paths of the commands and configuration files.

. . . .

2.3 Dependencies

D1. Java, the underlying technology of the project.

D2. OSGi, the underlying technology of the project. RoboBee is seperate the different services in bundles that can be started and updated at run-time.

D3. Groovy, the framework to parse the domain specific language of the script files.

D4. StringTemplate, the framework to create service configuration files.

. . . .

2.4 Related Projects

Puppet, is a configuration management for systems automation and uses Json data structures in manifests files, but also uses a declarative domain specific language (DSL) based on Ruby. It is developed by Puppet Labs.
https://puppetlabs.com

Chef,

> "[is a] configuration management tool written in Ruby and Erlang. It uses a pure-Ruby, domain-specific language (DSL) for writing system configuration "recipes". Chef is used to streamline the task of configuring and maintaining a company's servers, and can integrate with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines. Chef contains solutions for both small and large scale systems, with features and pricing for the respective ranges."

https://en.wikipedia.org/wiki/Chef_(software)

CFEngine,

> "is an open source configuration management system, written by Mark Burgess. Its primary function is to provide automated configuration and maintenance of large-scale computer systems, including the unified management of servers, desktops, consumer and industrial devices, embedded networked devices, mobile smartphones, and tablet computers."

https://en.wikipedia.org/wiki/CFEngine

Salt,

> "SaltStack platform or Salt is a Python-based open source configuration management software and remote execution engine. Supporting the "Infrastructure as Code" approach to deployment and cloud management."

https://en.wikipedia.org/wiki/Salt_(software)

Ansible,

> "a free-software platform for configuring and managing computers, combines multi-node software deployment, ad hoc task execution, and configuration management. It manages nodes (which must have Python 2.4 or later installed on them) over SSH or over PowerShell. Modules work over JSON and standard output and can be written in any programming language. The system uses YAML to express reusable descriptions of systems."

https://en.wikipedia.org/wiki/Ansible_(software)

CHAPTER 3

USE CASES

. . . .

3.1

CHAPTER 4 SCRIPT STATEMENTS

. . . .

4.1 Dhclient

4.1.1 Statements

dhclient { requests prepend }

The dhclient statements must start with “dhclient.with { ... }” and contains the dhclient statements inside the curly braces. The statements take either a list of arguments or a map that contains the arguments as name:=value pairs.

requests

requests requests

Sets the list of DHCP *requests*. Request item that starts with an exclamation mark “!” is removed from the list.

```
dhclient.with {
    requests "!domain-name-servers"
}
```

prepend

prepend option, declaration

Adds the *option* with the *declaration* as a prepend statement to the DHCP configuration.

```
dhclient.with {
    prepend "domain_name_servers", "127.0.0.1"
}
```

4.1.2 Example

```
dhclient.with {
    requests "!domain-name-servers"
    prepend "domain-name-servers", "127.0.0.1"
}
```

Listing (4.1) Removes the DNS-servers from the requests list and prepends the DNS-server with the specified address.

. . . .

4.2 Database Script

The database script is used to configure the database server, to specify the available databases on the server and the users that have access to those databases.

4.2.1 Statements

`database.with { debug bind admin database user }`

The database statements must start with "`database.with { ... }`" and contains the database statements inside the curly braces. The statements take either a list of arguments or a map that contains the arguments as `name:=value` pairs.

debug

`debug name, [level: level] [file: file] [. . . ]`

Sets the debug logging *level* and logging *file* for the specified logger *name*. The *name* identifies the logger of the server and the *level* can be any positive number, zero is normally interpreted as no logging.

```
database.with {
    debug "error", level: 1
}
```

```
debugLogs = []
debugLogs << [name: "error", level: 1]
debugLogs << [name: "slow-queries", level: 1]
debugLogs << [name: "general", level: 1, file: "/
    var/log/mysql/mysql.log"]
debugLogs.each { database.debug it }
```

bind

`bind local|all|host [, port: port]`

Sets the binding *host* and the binding *port*. Set to "`local`" for the local host address `127.0.0.1` or to "all" to listen to all hosts.

```
database.with {
    bind "192.168.0.1", port: 3306
}
```

admin

`admin name, password: password`

Sets the administrator *name* and *password* for the database server. If no password for the administrator account was yet set, this administrator name and password is set. The administrator user is used to to create users, create databases and set permissions.

```
database.with {
    admin 'root', password: 'mysqladminpassword'
}
```

db

`db name [, charset: name] [, collate: name] [, { script }]`

Creates a new database with the specified database *name*, and, optionally, the character set and collate. If the database was already on the server, the character set and collate should be updated to the specified character set and collate. The database server needs to support the specified character set and collate.

```
database.with {
    db "postfixdb", charset: "latin1", collate: "
        latin1_swedish_ci"
}
```

script

`script text|importing: resource [, charset: charset]`

Imports the script given as *text* or imports the script loaded from the specified *resource*. The resource can be a file, URL or URI, and the character set of the resource can be specified as *charset*. A string will be interpreted according to the format, if no scheme is used the string is assumed to be a local file.

```
database.with {
    db "postfixdb" with {
        script """
        SQL-SCRIPT
        """
    }
}
```

```
database.with {
    db "postfixdb" with {
        script importing: "postfixtables.sql",
            charset: "UTF-8"
    }
}
```

user

`user name, password: password [, server: host] [, { access }]`

Creates a new user with the specified *name*, *password* and server *host*. If the user already exists on the server, the password is updated for that user. A user is identified by the user name and the server host. The database that the user have read and write access to will be set. This will not create the database on the server, to create a database use the **`database`** statement.

```
database.with {
    user "drupal6", password: "drupal6password",
        server: "srv2"
}
```

access

`access database: database`

Sets the access rights for the *database* for the user. The user is granted all permissions to the specified database.

```
database.with {
    user "drupal6" with {
        access database: "drupal6db"
    }
}
```

4.2.2 Example

```
/**
 * Database.groovy
 */
database.with {
    bind localhost, port: 3306
    admin user: 'root', password: 'somepass'
    db name: 'phpmyadmindb'
    user 'phpmyadmin', password: '1234' with {
        access database: 'phpmyadmindb'
    }
}
```

Listing (4.2) Example Database script, sets the binding address, the administrator user and creates the phpmyadmin database and user.

It is recommended to put the script variables in a standalone file, for example in "`Globals.groovy`" and load the script variables using the "`evaluate`" statement. Also, if the variables are put into a map, the script statement can be simplified.

```
/**
 * Globals.groovy
 */
def databasemap = [
    bind: [host: '127.0.0.1', port: 3306],
    admin: [user: 'root', password: 'somepass'],
    phpmyadmin: [user: 'phpmyadmin', password: '
        somepass', db: 'phpmyadmindb'],
]
```

Listing (4.3) Predefined database variables.

```
/**
 * Database.groovy
 */
evaluate "./Globals.groovy" as File

database.with {
    bind databasemap.bind
    admin databasemap.admin
    db name: databasemap.phpmyadmin.db
    user databasemap.phpmyadmin with {
        access database: databasemap.phpmyadmin.db
    }
}
```

Listing (4.4) Example Database script that uses simplified statements with arguments from the predifined variables.