IBM Software Group | Rational software

IBM Rational Build Forge Servers

Abstracting hardware

Rational software

© 2008 IBM Corporation

Converted to video June 29, 2015

This module covers Abstracting Hardware when using Build Forge® Servers for IBM Rational® Build Forge Version 7.0 and above.

This module assumes you are familiar with IBM Rational Build Forge basics. For a primer on Build Forge, exit this module and first review the Introduction to Build Forge module, then continue with this more advanced topic.

Objectives

- To understand what a Build Forge agent is and what it does

- To be able to trace the agent protocol execution

- To have a working knowledge of selectors and collectors

This module explains what the Build Forge agent is and how hardware is used in Build Forge. It also outlines the steps that the agent takes when running a task. Additionally, this module discusses the use of Selector and Collector constructs, and shows how to implement them together to make hardware run a process.

The problem

- There are many varying computers companywide, such as those with Linux® or Windows®, and those with Perl or Java™.

- A user must know:
  - ▶ Where to run programs
  - ▶ How to use common existing hardware
  - ▶ What occurs when something works on one machine and not on another

Abstracting hardware

© 2008 IBM Corporation

There is a great deal of variance between computers, and usage must not be expanded without consideration for how that variance is controlled. Users must be aware of how to properly use resources, and how to appropriately track and redo Builds, if necessary. A prime example of not dealing with this variance occurs when a user declares that something works flawlessly on their computer, despite it not working on any other computers. A user must be able to determine the factor that allows it to work on this one computer.

IBM

The non-ideal solution

- Create a computing lab, and strictly control changes.
    - ▸ All changes are meticulously logged and made manually.
    - ▸ Users tend to avoid the lab unless absolutely necessary to avoid the administrative hassle associated with working in the lab.

4

Abstracting hardware

© 2008 IBM Corporation

One way to resolve the variable environments across computers is to strictly control changes on a set of computers. Those computers then become the "official" Build systems, and any Build submitted must work on those computers. This effectively controls the changes that happen on the computers, and keeps everyone on the same page.

However, this solution has its drawbacks. Users end up avoiding the lab, since it is not likely that the given configuration in the lab fits their needs exactly. If it does not match the required configuration, there is a great deal of paperwork, approvals, and effort needed to make the configuration changes so the lab can test whatever it needs. This results in the lab being used less as users either delay or circumvent using it altogether to avoid the extra hassle associated with it.

Build Forge solution

- Abstract the hardware from the equation.

- No general concern for discrete computers.

- There is a need for a machine:
  - That has Java 1.4 installed
  - With at least 500 MB of Ram available
  - Not already running a Build

- Solution: Collectors and Selectors

The Build Forge solution handles the problem of uncertainty regarding what is installed on the computer, creating varied results when tested on other computers. There is no need to tie a task to a particular computer, as that does not optimize resource usage. It also leaves potential problems exposed with that single fail point. What must be developed is a very specific list of requirements for a particular task to complete. If the machine's environment is known or can be controlled, then the problem will disappear. The two mechanisms Build Forge uses to introduce this feature are known as Collectors and Selectors.

What is a collector?

- Collectors solve the abstraction issue by collecting the "stats" of a machine.

- The collector defines what information is being searched for on a machine.

Abstracting hardware

6

© 2008 IBM Corporation

Collectors are the method used to bring important information about a given machine into Build Forge. Collectors can be set up to gather the available RAM, gather the installed JDKs, and to determine what Build Forge jobs are already running on that machine.

Details on collectors

- **Collector values fall into four categories:**
  - ▸ Set – A static user defined value
  - ▸ Built-in – A list of variables being searched for. For example: processor utilization under 5%
  - ▸ Run – A user-defined custom shell command, such as "java –version." This also might have a regular expression run against it.
  - ▸ .include – Allows nesting for hierarchical collectors

Abstracting hardware

7

© 2008 IBM Corporation

When setting up the Collector, the entered values must fall under one of the four categories listed here:

Set establishes a user-defined name/value pair that does not change. This type of value manually defines a value to a machine in Build Forge that might not physically exist.

Built-In is a list of variable types that are built into Build Forge for values that can be difficult to manually define outside of Build Forge. For example, it is difficult to get available RAM on the machine, so Build Forge provides values integrated into the product that measure various levels of available RAM.

Run is the most flexible of the commands, allowing a user to run a command on the machine and then peruse the results. For the previous example, determining what the installed JDK was on the system, the value can be set to "java –version." The results can be searched with a regular expression to find the version and Build information for the installed Java.

.include allows the abstraction of the Collectors by creating a hierarchy within the Collectors. This establishes a main Collector that every other Collector can inherit from.

Details on collectors

Collectors>>Example Add Collector Variable

Filter | Showing 1 - 17 of 17 Auto Paginate

Variable	Command	Type	Regular Expression
NUM_CPU		Built-in	
WIN_SERVICEPACK		Built-in	
OS_SYSNAME		Built-in	
OS_RELEASE		Built-in	
OS_VERSION		Built-in	
CPU_LOAD1		Built-in	
CPU_LOAD5		Built-in	
CPU_LOAD15		Built-in	
DISK_FREE		Built-in	
PERL_VER	perl -v	Run Command	^This is perl, ...(\d+\.\d+\.\d+)
GCC_VER	gcc -v	Run Command	^gcc version (\d+\.\d+\.\d+)
JVM_VER	java -version	Run Command	^java version "([^"]+)
DISK_TOTAL		Built-in	

(New Variable) Save Delete

Details

Type: Set Value   Variable:
Value:

This is a screen capture of the Collector menu. The columns shown are variable, command, type, and regular expression. Note the different types of Built-In variables that can be drawn upon. Also, notice that the Run command has extra fields for the command to run on the agent and the regular expression to retrieve the version information.

Servers

- To find and gather the required information:
  - ▸ Install the Build Forge agent on a given machine.
  - ▸ Provide the connection information for the machine to the Build Forge console in the server definition.
  - ▸ Set the collector to that server definition. The next refresh gathers the collector information and stores it in the manifest.

With the Collector defined, it must be applied in order to gather the required information. However, the underlying machine must be entered into Build Forge to abstract it. The Build Forge logical representation of hardware is called the Server. To set up the Server, it must be able to connect over the network. Setting up communications requires an installation of the Build Forge agent on the machine that it is connected to. Once the agent is in place, specify the connection information on the Management Console for how to connect to that agent. This completes the server definition in Build Forge, allowing a Collector to be applied. The next time the engine refreshes, the engine creates a connection to that database, and pulls the information specified in the Collector into the manifest. The manifest is the Management Console copy of the important information for the server, which is refreshed at a given time interval.

Server details

- Servers have the following settings:
  - Name – Logical Build Forge server name
  - Path – Agent directory
  - Host – Location of the agent
  - Authentication – Who is connecting to the agent
  - Access – Owner of the server
  - Collector – Stats designated for collection
  - Environment – Details the server specific environment
  - Files – Allows files to be deposited on the server

When setting up the server definition, the following information can be set:

The **Name** value is the logical name within Build Forge for the server definition. This is the name that Build Forge uses to track the server.

Path is the value that indicates where on the Agent machine Build Forge creates its folders and runs the Builds.

Host defines the network address where the agent is located. This can be any value as long as it resolves on the network. For example, this can be a host name or an IP address, as long as it resolves to an address on the Management Console machine.

Authentication is the credentials used when connecting to the Agent.

Access defines the Access Group for the agent. This is further explained in other modules. For this module, a user must know that this defines whom the owner of the Server is in Build Forge.

Collector has previously been discussed in this module. This is where collection information is applied to the server definition.

Environment defines the environment variables associated with this server. PATH or other variables for this server definition can be redefined to ensure that the environments are uniform across various machines. Environments are further explained in other modules. For this module, a user must know that this is where the initial environment on a particular Server definition can be set.

Files determines whether to allow files to be deposited on this server from the Build Forge Management Console.

Server details

Here is where to define the name, path, host, authentication, and access for a server

This is the Server Definition menu in the Management Console. All servers are required to define a name, path, host, access, and authentication.

Manifest

- Every server has a manifest associated with it.

- The manifest holds the "stats" of the associated agent machine.
  - ▸ The collector associated with that server controls the tracked stats.

- The manifest refresh cycle is automatically handled by the refresh process on the console machine.
  - ▸ The interval between refreshes is configurable on the console.

Abstracting hardware

12

© 2008 IBM Corporation

This section of the module focuses on the Manifest. As stated earlier, the Manifest is the compilation of the Collector information gathered in the last refresh. The Manifest is stored on the Management Console database. The information exists locally on the Management Console server, and does not have to be retrieved from the remote Agent every time. Note that this can cause lags between the information in the Manifest and the actual configuration on the Agent. Ensure that the refresh cycle for the Agents is set to work for the particular configuration.

What is a selector?

- Upon abstracting the hardware to a manifest, it must be determined where to run the process.

- The selector searches the manifest and "selects" the best candidate machine for the current needs.

Abstracting hardware

13

© 2008 IBM Corporation

After running Collectors, Build Forge now has the hardware information and the information flagged as important. A user must next decide what hardware to run this information on. The mechanism in Build Forge that decides what hardware to use is the Selector. The Selector takes the Manifest information in the database and searches for criteria that has been defined in the Selector. It then picks the agent that best fits that criteria.

IBM

Selector details

- The following fields are definable in a selector:
  - ▸ Name – The name of the variable
  - ▸ Operator – The logical operation associated with this entry. For example, whether the collector variable value should be greater than or equal to a value
  - ▸ Value – What the name variable is compared to
  - ▸ Required – Determines if this is a required criterion for the Selector, or whether it should be used to rank valid choices

Abstracting hardware

14

© 2008 IBM Corporation

Selectors are set up from one to many criteria. Each criterion needs to define the following values:

Name defines the Collector variable being sought for comparison.

Operator controls how to logically compare this value in the Selector. For example, this is used when determining whether the collector variable value is greater than or equal to a value.

Value defines what the Collection variable is being compared to.

Required states if this criterion is necessary for the selection of the server. If this is set to "No," it is used to rank the available Agent resource choices rather than exclude those choices.

Selector details

This screen capture shows the Selector menu in the Build Forge Management Console, and the list of criteria defined in the Selector. In this case, there is only one criterion defined, the BF_NAME. This must not be equal to "null" to ensure that this Selector will return a pool of agents whose logical name is not "null."

Benefits

- Agent installs are very small and lightweight, thus adding hardware is easy.

- Processes are not tied to a particular physical machine, but rather by requirements.
  - ▸ This allows spread Build work to occur around a pool of Agents defined by their requirements, rather than their physical boxes.

The benefit of the Selector/Collector dynamic is the fact that adding new hardware to the Build Forge agent pool is easy. The agent daemon process has a very small memory and processor footprint, so it is easy to install and leave the agent daemon running on any machine. As long as that machine is connected to the Management Console by way of the network, that Agent can be added.

Since the processes entered into Build Forge are freed from physical machine constraints, there is no longer a concern about where the Build runs. The Agent that it is run on has all the necessary requirements, and will return the same result regardless of what agent was selected from the Agent pool with the Selector.

Agent protocol
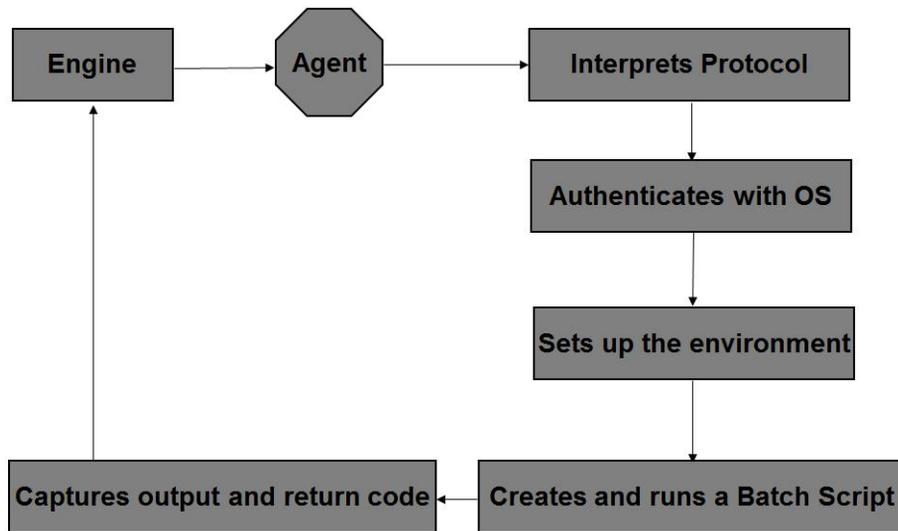
This slide lists the steps during the Agent contact with the Management Console.

First, the engine sends the command request to the agent. It is assumed that the engine has already selected this Agent by way of the Selector as the best option. The agent then receives the command and creates a command shell. The shell reads the protocol request from the engine, and derives the command and environment from the protocol. Afterwards, the shell runs the user credentials from the protocol to switch the shell to the given user. The agent brings in the environment defined in the protocol. Upon its completion, the Agent generates a batch script containing the command sent from the Engine. With the batch script done, the Agent runs it in the generated shell. The output from the batch script execution, in addition to the exit code, is captured by the Agent, and is sent back to the Engine.

Note that it is not required that the engine begins the agent process. The process can be started using a telnet connection to that agent service. When telnetting into the agent on the agent port, a user will receive a "Hello" message from the agent. This allows the manual definition of an Agent protocol request to determine what response it produces.
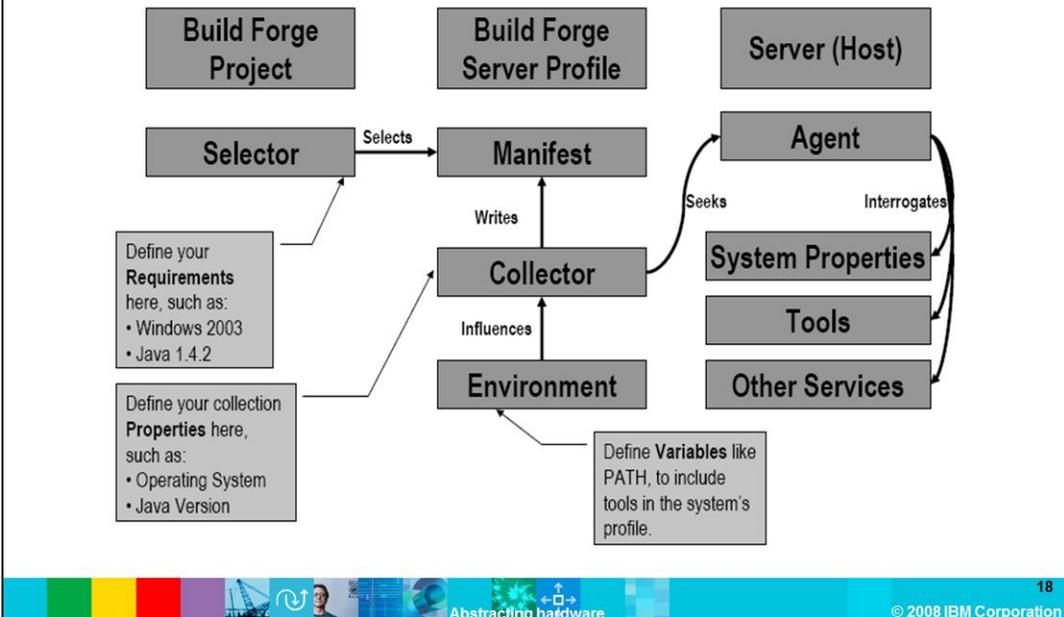
This slide summarizes the interaction between the Agent, Selector, and Collector. The Selector looks at the data in the Manifest generated by applying the Agent environment to an agent, which then looks for the information defined in the Collector. The information from the Collector can be derived from anything in the Agent operating system accessible through the command line.

Summary

- Agent process are accessed by servers.

- Build Forge abstracts hardware into logical units in the console.

- Collectors collect "stats," and selectors select a machine based on that data.

Abstracting hardware

19

© 2008 IBM Corporation

In summary, Server definitions are the logical Build Forge representations of Agents installed on remote computers. A Collector is then applied to that server, and gathers defined information. Once the data is gathered, it is placed in the server manifest. When a process is called in Build Forge, a Selector is used to search the manifests to determine the best candidate to run that particular process.

Trademarks, copyrights, and disclaimers

Abstracting hardware

RBF_Module3_BuildForgeServers.ppt   Page 20 of 20