IBM Rational Build Forge

Build Theory

The Selection Process

Kristofer Duer

February 9, 2011

Note
Before using this information and the product it supports, read the information in “ Notices,” on page 24.

IBM RATIONAL BUILD FORGE.......1

1 INTRODUCTION - BUILD THEORY.......5

1.1 The selection process.......5

2 RATIONAL BUILD FORGE ARTIFACTS USED IN SELECTION PROCESS.......6

2.1 Server definition.......6
2.2 Server authorization.......6
2.3 Manifest.......6
2.4 Collector.......7
2.5 Selector.......7

3 HOW IT ALL FITS TOGETHER.......9

4 SELECTION RULES – HOW A STEP CHOOSES WHICH SELECTOR TO USE.......12

4.1.1Example 1 – No selector on any steps.......13
4.1.2 Example 2 – Selector on specific steps.......14
4.1.3 Example 4 – Inline library without step selectors.......16
4.1.4 Example 5 – Multilevel Inlines with step selectors.......17
4.1.5 Example 6 – Inline project.......18
4.2 Impact of sticky.......19
4.3 Impact of .bset.......20

5 BUILD FORGE INTERNAL DESIGN FOR THE SELECTION PROCESS.......21

5.1 Refresh process.......21
5.2 How a job slot works.......21
5.3 Influencing job slots with BF_RESERVE and BF_EXCLUSIVE.......23
5.3.1 BF_RESERVE.......23
5.3.2 BF_EXCLUSIVE.......23

6 NOTICES.......24

6.1 Trademarks.......25

1 Introduction - Build theory

Build theory is the compilation of using the IBM Rational Build Forge tool within the scope and needs of the automation process to be implemented. It is a generic term that covers how, where, when, and why to use certain features within Rational Build Forge. The theory deals with abstract concepts that can be applied in real-world scenarios by breaking down each component of Rational Build Forge and defining strengths and weaknesses of different approaches.

1.1 The selection process

The selection process refers to the entire scope of physical server management and selection for running steps. All of the concepts covered in this section deal with managing the physical resources available to do the work required of the automation system.

2 Rational Build Forge artifacts used in selection process

2.1 Server definition

A server within Rational Build Forge is defined as a single logical server artifact identified by a logical name. The server artifact has properties that define the host to which the logical server definition points, which server authorization to use, which path to use as the base path for build steps, and so on. The server definition is logical only; multiple Rational Build Forge server definitions can point to the same physical resource. Rational Build Forge treats each server definition as a single instantiation. Some use cases employ the strategy of having different server definitions point to the same server with different server authorizations. In other cases, multiple definitions logically increase the load that the server can take. Rational Build Forge has a Max Jobs property, discussed later, that is per server definition rather than physical host.

2.2 Server authorization

Each server definition within Rational Build Forge requires authentication tokens for the host computer. The server auth artifact contains username=password pairs identified to Rational Build Forge through a logical name for the grouping. This mechanism allows the same username=password combination to be used in multiple server definitions and be declared only once. The true power of this design is seen when the password changes: only one place needs to be updated and all server definitions using the server auth are immediately updated with the new password. It is common to use LDAP or AD users in this scheme as the username=password combination is managed centrally in the external system. You can certainly use local users; however, as the server counts increase, the management of these individual user accounts and making sure they meet corporate password requirements becomes more difficult.

2.3 Manifest

Each manifest artifact has a 1-1 relationship with a logical server. The manifest stores the story about the server. Consider the manifest nothing more than the list of information specific to this particular server definition. The specific information to gather is defined by the collector associated with the server definition. As described in the next section, using the collector, you customize the data a manifest stores for use by the selectors to determine where to run builds. The refresh process places this data in the manifest. The frequency of the refresh process is covered later. For now, you only need to understand that the refresh process updates the values on a cadence specified by certain settings you access through the **Administration > System** menu.

The selector reads the story contained in the manifest. An important characteristic of selectors is that you assign them to Rational Build Forge artifacts, not servers. As a result, a selector reads manifests to determine access to server resources. Selectors do not query a server directly.

There are five manifest variables that are always available for server definitions:
BF_AGENT_VERSION
BF_JOBS
BF_LAST_REFRESH
BF_LAST_UPDATE
BF_LOADRATIO

In addition to these five variables defined for each logical server definition, the internal manifest variable BF_NAME is available. To customize the data contained within the manifest, review the next section on collectors.

2.4 Collector

The manifest of a server definition holds the story. The collector gets the story. Collectors gather information about a server to place in the manifest. The collector gathers certain data about a physical host based on collector variables. Collectors can gather these three types of information from a system:

- Built-in variables
  Information such as operating system version, memory usage, disk space used, and so on
- User-defined variables
  Items such as which building a computer is in, which pool or farm it belongs to, brand names, and so on
- Run-command variables
  Information the computer can determine by running a command, such as java version, perl version, whether a particular compiler is installed

In terms of server and network resources, user-defined variables are the most economical type of collector variable to gather. These variables are in the database; no agent connection is required. For built-in variables, the agent command cmd sysconf gathers all these variables at the same time. Using multiple built-in variables is approximately the same as using one built-in variable. Either way, the agent is used only one time; however, using multiple variables adds more rows to the database, which is still negligible. The run-command variables are the most expensive collector variable to gather. Use run-command variables only if you require specific information about the server that the other variable types cannot provide.

2.5 Selector

The selector exposes the server resources to builds. The selector contains variables that determine a server based on criteria. There are two types of selector variables, required and not required, also known as optional. The required variables scan the server definition manifests and only keep those manifests that meet the required variables. This selection does not use a weighting system; the process is only a Boolean check of whether the manifest meets or does not meet criteria. The optional variables use a weighting system. The BF_JOBS property is used in conjunction with the MAX JOBS property of a server to determine the LOAD RATIO. The load ratio is used internally as a single point for the weighting system. For example a server with 1 build and 10 max jobs has a lower load ratio than a server with 1 build and 3 max jobs. Additional optional variables provide selection data upon which to make a better decision for your environment. The built-in collector variables are often used for this purpose.

For example, using the DISK_FREE optional variable and the greater than (>) operator, you could give more preference to a computer with a certain amount of disk space available. To give preference to a computer with more than 2 GB free, you would use at least one optional selector variable DISK_FREE > 2000000000. This selection gives equal weight to a computer with 2.1 GB free and one with 2 TB free. To give a computer with the most disk free the higher preference, specify multiple entries. For example, consider these three entries:
DISK_FREE > 1000000000
DISK_FREE > 10000000000
DISK_FREE > 100000000000

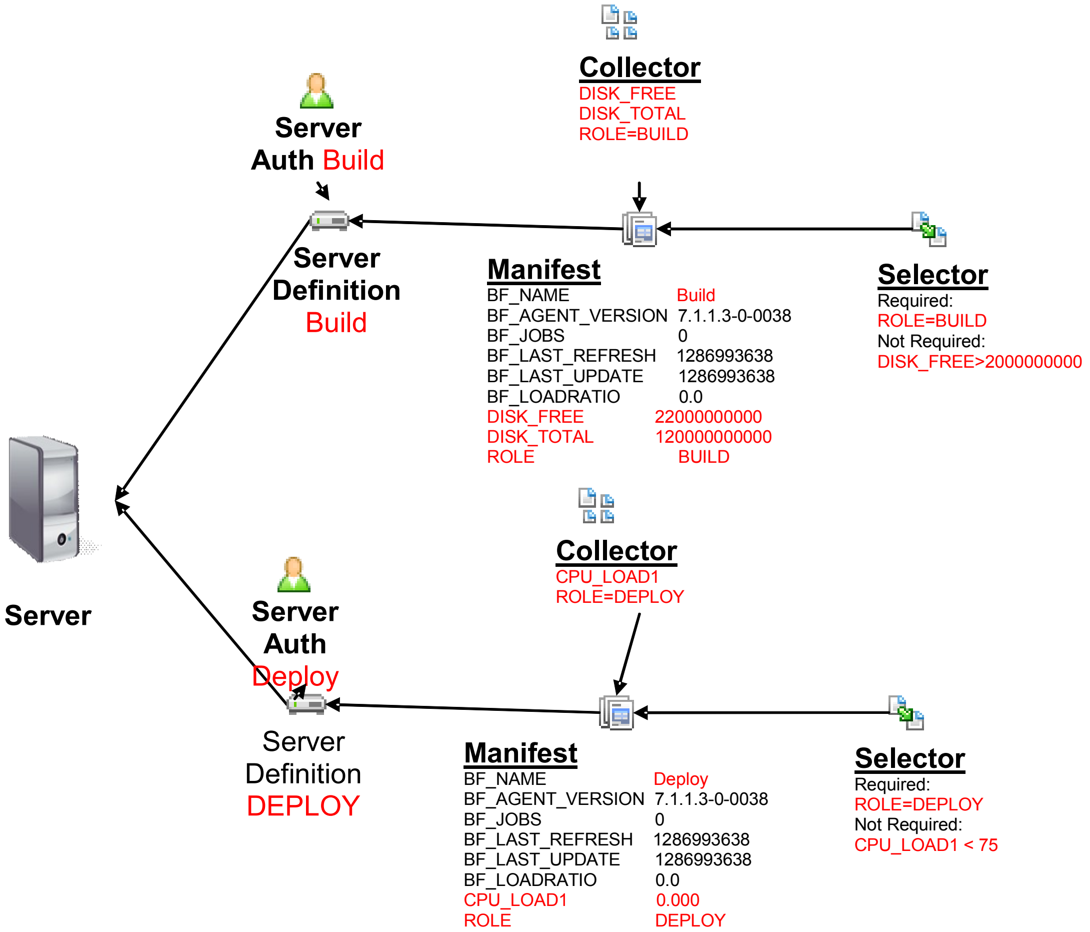
This setup gives a weight point to a computer for each entry it satisfies (1 GB, 10 GB and 100 GB of free disk space available). The more optional entries you use, the closer you get to the computer with the most disk space available. Similarly, you can locate a computer with the least amount of disk space available by flipping the operator to a less than (<) operator. Another example is to use the CPU_LOAD* variables to prefer a computer with more CPU processing free. All of these methods are intended to prefer that criteria or groups of criteria specific to your environment that is most essential: number of CPUs, free CPU cycles, disk space, memory usage, and so on.

The purpose of the selector is to find the "most available" server definition. The most available server definition is the one that passes the Boolean required variable check and has the highest weight from the optional selector variables. As mentioned previously, you can use additional optional variables to influence this most available check through alternative means than just the internal load ratio optional variable.

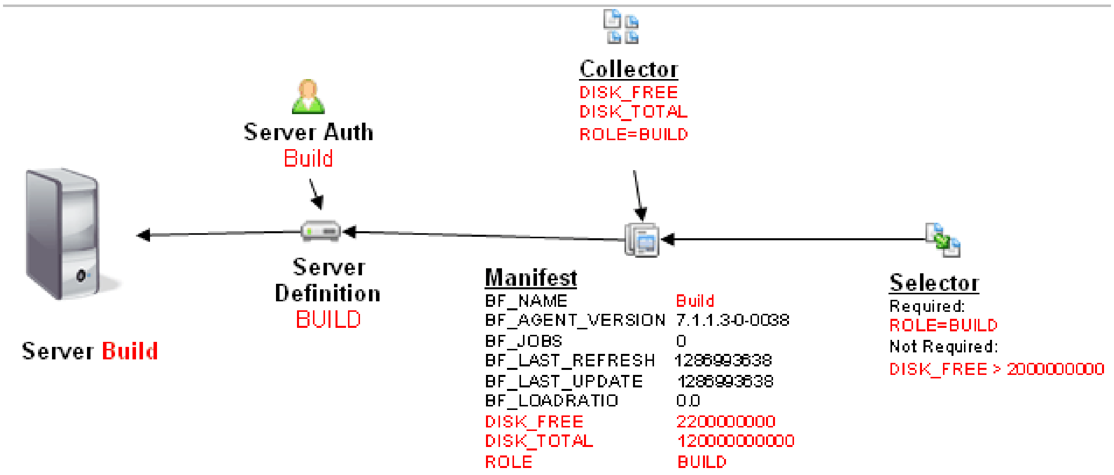
A common type of selector is one without any variables required or optional. In this scenario, the system picks from *any* of the currently available servers and of course evaluates the internal BF_JOBS optional variable. This selector is usually referred to as the ANYMACHINE selector by build engineers. This kind of selector provides incredible flexibility. However, the lack of variables produces another phenomenon: the same server is always picked if no servers are currently running jobs. The same holds true if the same servers are considered equally the "most available" servers each time the selection process occurs. This behavior results from how the Perl engine retrieves the list of servers. It is unpredictable what the order will be, but the order is always the same. So, all things being equal, the "most available" server tends to be the same one if no jobs are running.

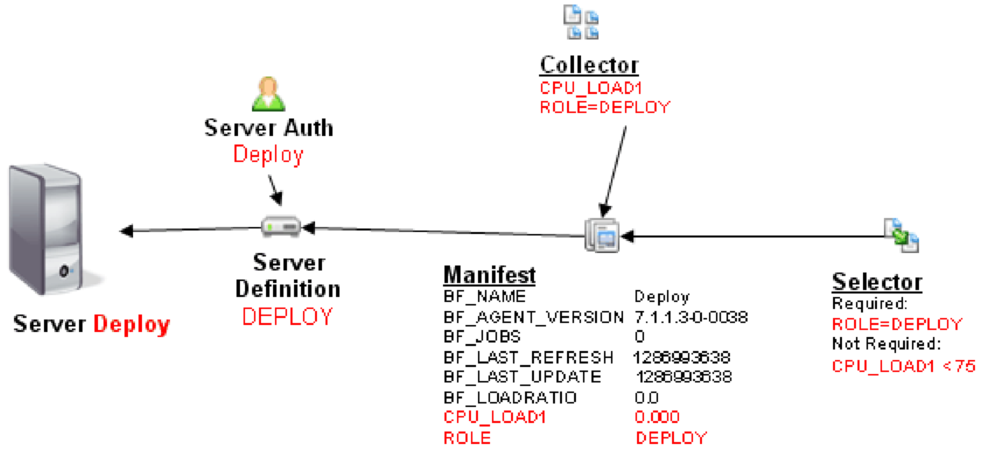
3 How it all fits together

The picture below illustrates the lines of communication. The server is defined as the system with a Rational Build Forge agent installed. The server definition points to that hostname; however, there can be multiple server definitions for each physical host. The following example has two definitions: one for build and one for deploy. Each server definition has its own server auth to hold a different user=password combination for the differing roles on the target system. The collector specifies what story is held in the manifest. In this case, there are two built-in variables, DISK_FREE and DISK_TOTAL, and one user-defined variable, ROLE. The manifest for the Build server definition collects the amount of free disk on the computer and populates the manifest with this dynamic information. In addition, the collector adds the user-defined variable of ROLE and sets it to BUILD in the manifest. The selector is tuned to look for only BUILD role servers and to prefer those with the most available disk from among the server definitions with the BUILD role. The deploy definition operates in a similar manner, only it prefers those hosts with the lowest utilized CPU.

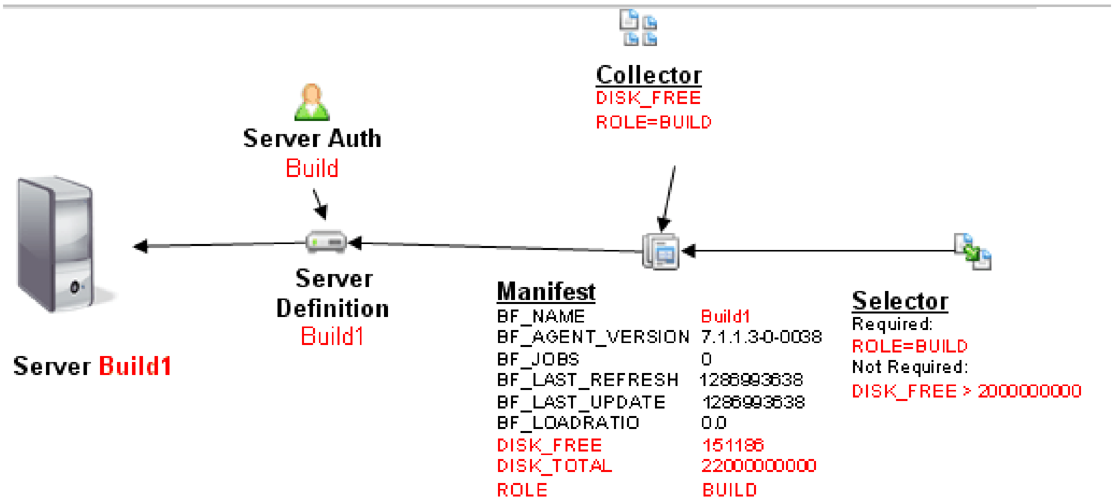


While you can use the same host for multiple different roles within a Rational Build Forge system, it is more common to use two different hosts in a 1-1 relationship with servers and server definitions as the below graphic illustrates.



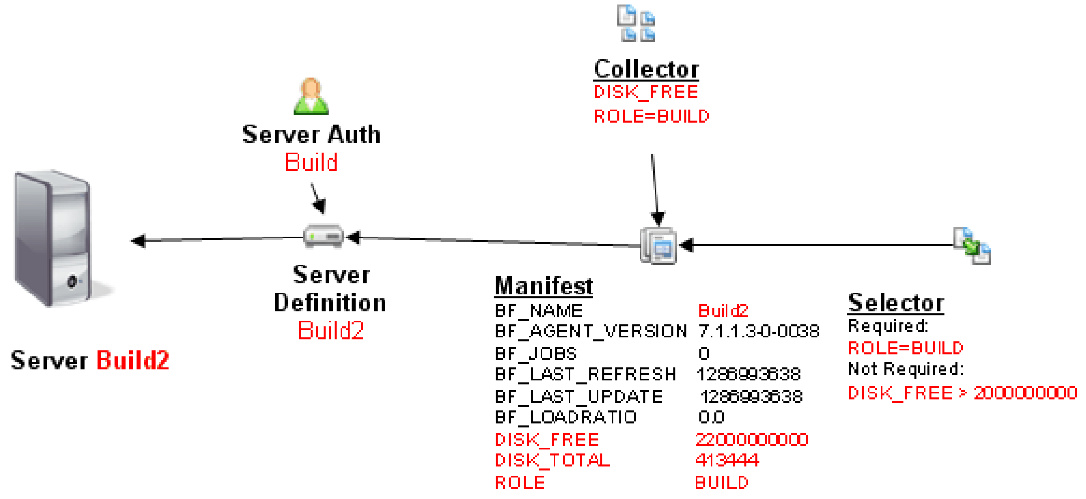


With this design, physical servers are fully pluggable without modifying current builds. For example, to add more build servers a new server definition with the proper collector is all that is required for the selector to access them. If the user=password credentials are the same on both build servers, the same server auth can be used for both. For example in the illustration below, if Build1 must be taken down for maintenance, Build2 can continue accepting new jobs without any Rational Build Forge build knowing Build1 is no longer available. In addition, another server, Build3 perhaps, can be added quickly and easily to the role of build to expand capacity with minimal effort.



Manifest

BF_NAME	Build1
BF_AGENT_VERSION	7.1.1.3-0-0038
BF_JOBS	0
BF_LAST_REFRESH	1286993638
BF_LAST_UPDATE	1286993638
BF_LOADRATIO	0.0
DISK_FREE	151186
DISK_TOTAL	22000000000
ROLE	BUILD



Manifest

BF_NAME	Build2
BF_AGENT_VERSION	7.1.1.3-0-0038
BF_JOBS	0
BF_LAST_REFRESH	1286993638
BF_LAST_UPDATE	1286993638
BF_LOADRATIO	0.0
DISK_FREE	22000000000
DISK_TOTAL	413444
ROLE	BUILD

4 Selection rules – how a step chooses which selector to use

Selectors themselves can be associated with a project, a step, or even a schedule. This flexibility can lead to confusion about how a step picks the selector it uses and by extension which server definitions it uses to execute the commands. The step always looks to itself for an associated selector first, and if none are associated with the step, it looks to the project.

For the following examples, use these selectors and server definitions:
Selectors:
ANYMACHINE
NO VARIABLES – will pick from any computer currently available to Rational Build Forge

Windows
->OS_SYSNAME contains Windows, required

Linux
->OS_SYSNAME=Linux, required

The first selector looks at any server definition currently in the system to find the one that is most available. The second selector limits the search to only those computers with OS_SYSNAME defined in the manifest and containing the characters Windows to find the most available. The third selector operates the same as the second selector, only it looks for Linux instead of Windows in the manifest.

Server Definitions:
Linux_Machine_A
Linux_Machine_B
Win_Machine_A
Win_Machine_B

Collector:
OS_SYSNAME – built in

When looking for a selector, a step completes these checks:
1) Look at step
2) Look at the next scope up

If a step has a selector specified, it uses that selector regardless of what the project has as a selector. The next scope-up check is defined as the following checks:

1) Parent project
    a. If a project is inlined, this check takes precedence over a parent inline calling step's selector. This is illustrated below.
2) Parent inline calling step

So the full selection rules in terms of which selector a step uses are:

1) Step selector
2) Parent project selector
    a. Schedule selector - a selector associated with a schedule replaces the parent project selector of the project being scheduled and acts as the new parent project selector
3) Parent scope selector

Note: The selection rules are only applied to steps that do not have a current server associated with them. Items such as sticky and .bset can associate a server with steps that have not run yet. These items take priority over the selection rules. There is this internal rule 0 applied to each step prior to applying the selection rules.

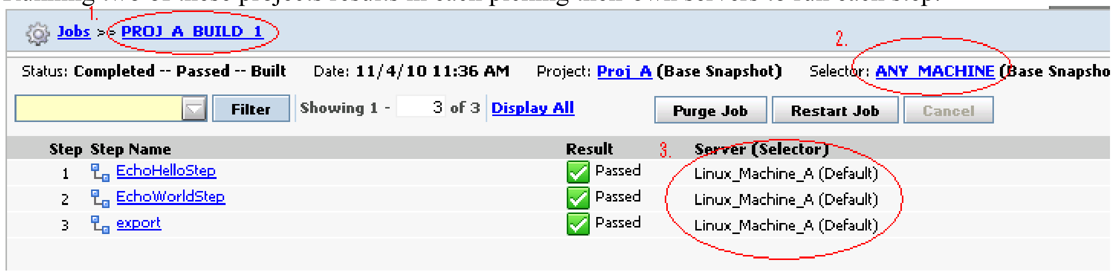
0) Step needs a server
    a. If yes, the selection rules are processed
    b. If no, the selection rules are not needed, and the already assigned server is used
1) Step selector
2) Parent project selector
    a. Schedule selector - a selector associated with a schedule replaces the parent project selector of the project being scheduled and acts as the new parent project selector
3) Parent scope selector

4.1.1 Example 1 – No selector on any steps

Project Definition for the example:

- Proj_A – Selector: ANY_MACHINE
    - EchoHelloStep – Selector: None
    - EchoWorldStep – Selector: None
    - Export Build – Selector: None

Running two of these projects results in each picking their own servers to run each step.

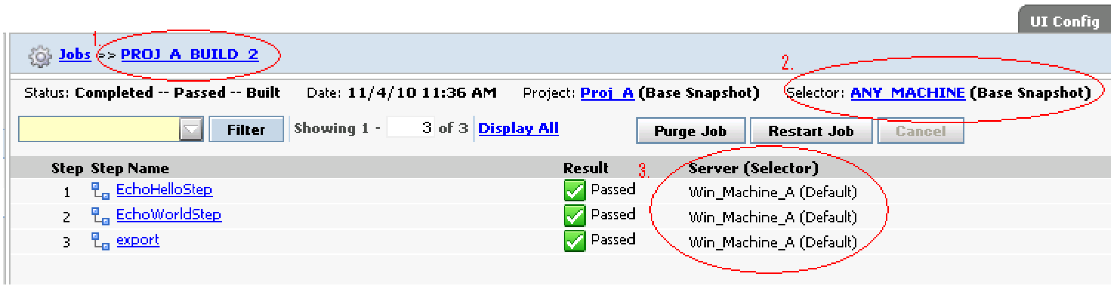


Jobs >> PROJ_A_BUILD_1

Status: Completed -- Passed -- Built Date: 11/4/10 11:36 AM Project: Proj_A (Base Snapshot) Selector: ANY_MACHINE (Base Snapsho

Showing 1 - 3 of 3 Display All

Step	Step Name	Result	Server (Selector)
1	EchoHelloStep	✓ Passed	Linux_Machine_A (Default)
2	EchoWorldStep	✓ Passed	Linux_Machine_A (Default)
3	export	✓ Passed	Linux_Machine_A (Default)



Jobs >> PROJ_A_BUILD_2

Status: Completed -- Passed -- Built Date: 11/4/10 11:36 AM Project: Proj_A (Base Snapshot) Selector: ANY_MACHINE (Base Snapshot)

Showing 1 - 3 of 3 Display All

Step	Step Name	Result	Server (Selector)
1	EchoHelloStep	✓ Passed	Win_Machine_A (Default)
2	EchoWorldStep	✓ Passed	Win_Machine_A (Default)
3	export	✓ Passed	Win_Machine_A (Default)

The build tag is identified in the above example by circle 1. Circle 2 is the selector each project uses. Finally we can see each build used a different server for the steps. The fact that the server is the same for all three steps is a coincidence – there were no built-ins used to influence the behavior and both builds were spawned together. As a result, build 1 found Linux_Machine_A to be the most available for each step using the round-robin mechanism, and build 2 found the same to be true for Win_Machine_A. This result highlights the weakness of not using proper built-ins. The round-robin technique always gets back the same ordered list of server definitions. It then cycles through this order and finds the most available server. If you are not using any optional variables, you will likely find one server definition being used more often. Proper use of built-ins influences this behavior and instead prefers the server definition based on better criteria, whether it is the amount of memory a computer has, amount of available disk space left, or even current CPU consumption.

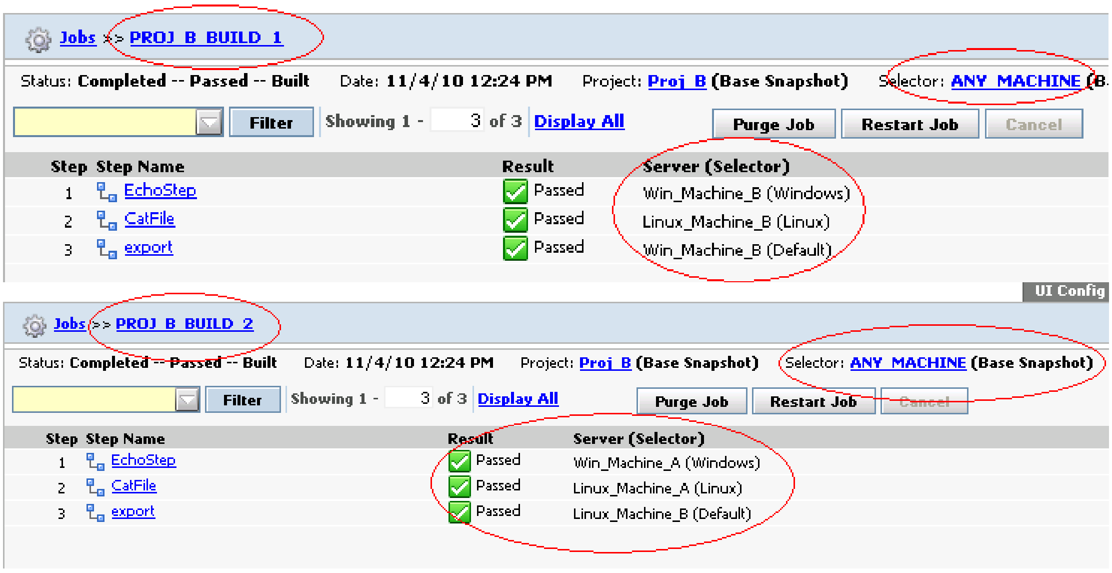
Applying the selection rules:

- Proj_A – Project container, no selector rule needed
  - EchoHelloStep – Rule 2) Parent project selector (Proj_A Selector)
  - EchoWorldStep – Rule 2) Parent project selector (Proj_A Selector)
  - Export Build – Rule 2) Parent project selector (Proj_A Selector)

4.1.2 Example 2 – Selector on specific steps

Project definition for the example:

- Proj_B – Selector: ANY_MACHINE
  - EchoStep – Selector: Windows
  - CatFile – Selector: Linux
  - Export Build – Selector: None



Jobs >> PROJ B BUILD 1

Status: Completed -- Passed -- Built   Date: 11/4/10 12:24 PM   Project: Proj B (Base Snapshot)   Selector: ANY MACHINE (B.

Showing 1 - 3 of 3 Display All

Step	Step Name	Result	Server (Selector)
1	EchoStep	✓ Passed	Win_Machine_B (Windows)
2	CatFile	✓ Passed	Linux_Machine_B (Linux)
3	export	✓ Passed	Win_Machine_B (Default)

Jobs >> PROJ B BUILD 2

Status: Completed -- Passed -- Built   Date: 11/4/10 12:24 PM   Project: Proj B (Base Snapshot)   Selector: ANY MACHINE (Base Snapshot)

Showing 1 - 3 of 3 Display All

Step	Step Name	Result	Server (Selector)
1	EchoStep	✓ Passed	Win_Machine_A (Windows)
2	CatFile	✓ Passed	Linux_Machine_A (Linux)
3	export	✓ Passed	Linux_Machine_B (Default)

This example has a slightly different outcome. The first two steps in Proj_B define their own selectors as these steps require commands specific to those operating systems. The final step uses the default selector, which happens to be the ANY_MACHINE selector. We can see build 1 chose Win_Machine_B for its first step, and build 2 chose Win_Machine_A. This illustrates the ability to specify particular selection criteria for specific steps within a build, while still applying the project-level selector to steps that do not have their own selector.

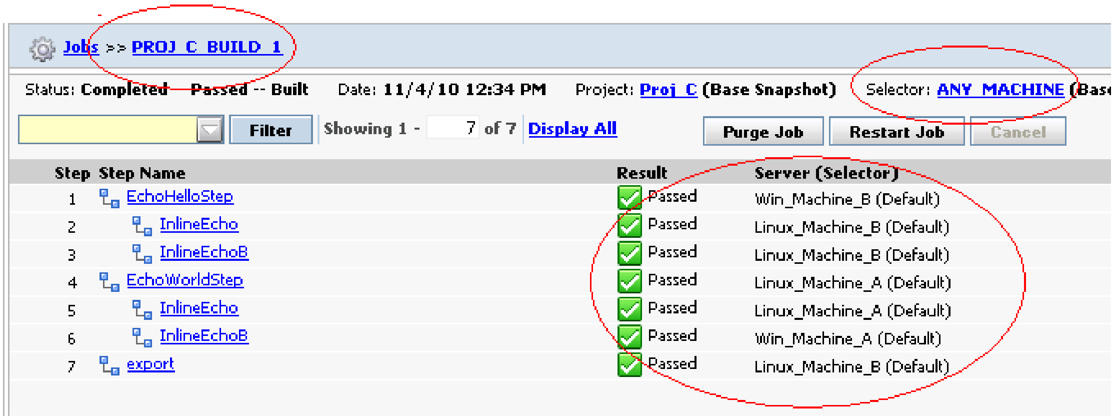
Applying the selection rules:

- Proj_B – Project container, no selector rule needed
  - EchoStep – Rule 1) Step selector (Windows)
  - CatFile – Rule 1) Step selector (Windows)
  - Export Build – Rule 2) Parent project selector (Proj_B Selector)

Example 3 – Inline library no selector on steps

Project Definition for the example:

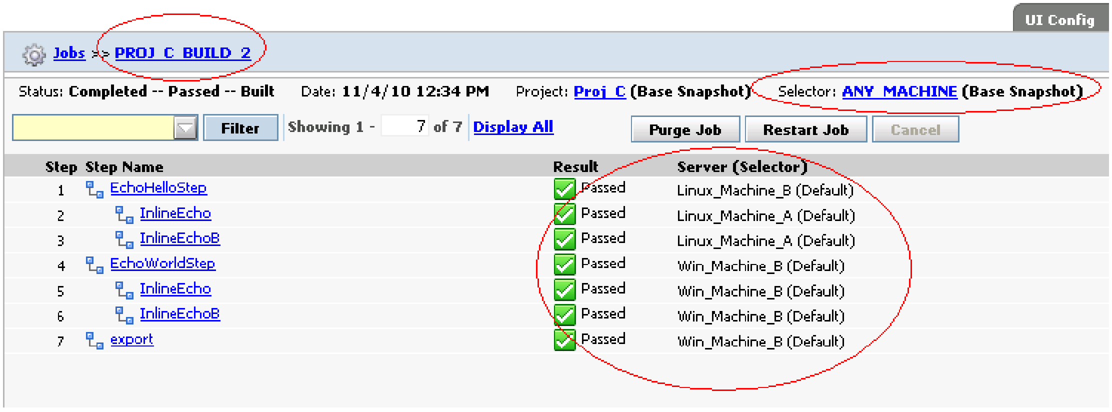
- Proj_C – Selector: ANY_MACHINE
    - EchoHelloStep – Selector: None
    - Inline Inline library – Selector: None
        - ➔ InlineEcho – Selector: None
        - ➔ InlineEchoB – Selector: None
    - EchoWorldStep – Selector: None
    - Inline Inline library – Selector: None
        - ➔ InlineEcho – Selector: None
        - ➔ InlineEchoB – Selector: None
    - Export Build – Selector: None



Jobs >> PROJ_C_BUILD_1

Status: Completed -- Passed -- Built Date: 11/4/10 12:34 PM Project: Proj_C (Base Snapshot) Selector: ANY_MACHINE (Base

Step	Step Name	Result	Server (Selector)
1	EchoHelloStep	✓ Passed	Win_Machine_B (Default)
2	InlineEcho	✓ Passed	Linux_Machine_B (Default)
3	InlineEchoB	✓ Passed	Linux_Machine_B (Default)
4	EchoWorldStep	✓ Passed	Linux_Machine_A (Default)
5	InlineEcho	✓ Passed	Linux_Machine_A (Default)
6	InlineEchoB	✓ Passed	Win_Machine_A (Default)
7	export	✓ Passed	Linux_Machine_B (Default)



Jobs >> PROJ_C_BUILD_2

Status: Completed -- Passed -- Built Date: 11/4/10 12:34 PM Project: Proj_C (Base Snapshot) Selector: ANY_MACHINE (Base Snapshot)

Step	Step Name	Result	Server (Selector)
1	EchoHelloStep	✓ Passed	Linux_Machine_B (Default)
2	InlineEcho	✓ Passed	Linux_Machine_A (Default)
3	InlineEchoB	✓ Passed	Linux_Machine_A (Default)
4	EchoWorldStep	✓ Passed	Win_Machine_B (Default)
5	InlineEcho	✓ Passed	Win_Machine_B (Default)
6	InlineEchoB	✓ Passed	Win_Machine_B (Default)
7	export	✓ Passed	Win_Machine_B (Default)

This example has two builds in which all the steps get the project default selector. The server definition used changes from step to step as the most available server definition changes for each step. Adding an inline to the project does not change how the selection works when compared to Proj_A. The inline does introduce the third rule in the selector use, Rule 3. This rule stipulates that if the parent project of inline steps, the library definition, has no selector, then the next highest scope selector is used. If the next highest scope has no selector, then it looks in the scope preceding that one on up until either a parent inline step has a selector, or the parent scope ends up being the parent project for the entire build.

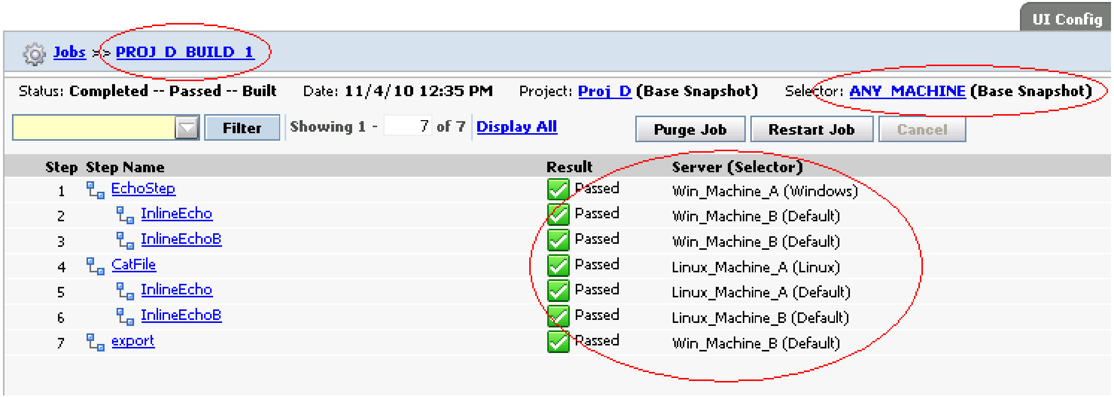
Applying the selection rules:

- Proj_A – container, no selector rule needed
    - EchoHelloStep – Rule 1) Step Selector (Windows)
        - ➔ InlineEcho – Rule 3) Parent Scope Selector (Ends up defaulting to Rule 2)
        - ➔ InlineEchoB – Rule 3) Parent Scope Selector (Ends up defaulting to Rule 2)
    - EchoWorldStep – Rule 1) Step Selector (Linux)
        - ➔ InlineEcho – Rule 3) Parent Scope Selector (Ends up defaulting to Rule 2)
        - ➔ InlineEchoB – Rule 3) Parent Scope Selector (Ends up defaulting to Rule 2)
    - Export Build – Rule 2) Parent Project Selector (Proj_C Selector)

4.1.3 Example 4 – Inline library without step selectors

Project definition for the example:

- Proj_D – Selector: ANY_MACHINE
    - EchoStep – Selector: Windows
    - Inline Inline library – Selector: None
        - ➔ InlineEcho – Selector: None
        - ➔ InlineEchoB – Selector: None
    - CatFile – Selector: Linux
    - Inline Inline library – Selector: None
        - ➔ InlineEcho – Selector: None
        - ➔ InlineEchoB – Selector: None
    - Export Build – Selector: None

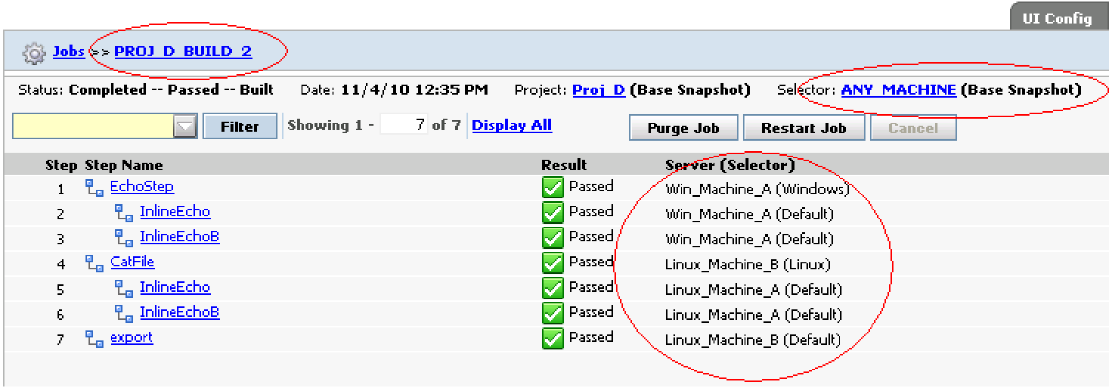


Jobs >> PROJ_D_BUILD_1

Status: Completed -- Passed -- Built Date: 11/4/10 12:35 PM Project: Proj_D (Base Snapshot) Selector: ANY_MACHINE (Base Snapshot)

Showing 1 - 7 of 7 Display All

Step	Step Name	Result	Server (Selector)
1	EchoStep	✓ Passed	Win_Machine_A (Windows)
2	InlineEcho	✓ Passed	Win_Machine_B (Default)
3	InlineEchoB	✓ Passed	Win_Machine_B (Default)
4	CatFile	✓ Passed	Linux_Machine_A (Linux)
5	InlineEcho	✓ Passed	Linux_Machine_A (Default)
6	InlineEchoB	✓ Passed	Linux_Machine_B (Default)
7	export	✓ Passed	Win_Machine_B (Default)



Jobs >> PROJ_D_BUILD_2

Status: Completed -- Passed -- Built Date: 11/4/10 12:35 PM Project: Proj_D (Base Snapshot) Selector: ANY_MACHINE (Base Snapshot)

Showing 1 - 7 of 7 Display All

Step	Step Name	Result	Server (Selector)
1	EchoStep	✓ Passed	Win_Machine_A (Windows)
2	InlineEcho	✓ Passed	Win_Machine_A (Default)
3	InlineEchoB	✓ Passed	Win_Machine_A (Default)
4	CatFile	✓ Passed	Linux_Machine_B (Linux)
5	InlineEcho	✓ Passed	Linux_Machine_A (Default)
6	InlineEchoB	✓ Passed	Linux_Machine_A (Default)
7	export	✓ Passed	Linux_Machine_B (Default)

In this example, we begin to see the power of setting a selector on a step. The selector is used for the entire step scope. The step scope is defined as the current step as well as inline steps as discussed with the rules above. We see the true meaning behind rule 3: using the parent scope selector to allow the same inline library to use different selectors when called from different steps.

Applying the selection rules:

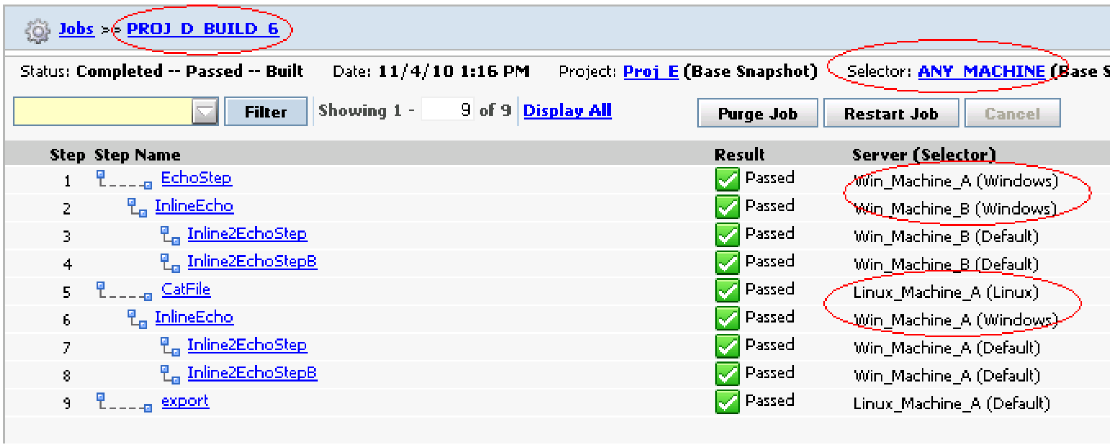
- Proj_D – container, no selector rule needed
  - EchoStep – Rule 1) Step selector (Windows)
    - ➔ InlineEcho – Rule 3) Parent scope selector (EchoStep Selector)
    - ➔ InlineEchoB – Rule 3) Parent scope selector (EchoStep Selector)
  - CatFile – Rule 1) Step Selector (Linux)
    - ➔ InlineEcho – Rule 3) Parent scope selector (CatFile Selector)
    - ➔ InlineEchoB – Rule 3) Parent scope selector (CatFile Selector)
  - Export Build – Rule 2) Parent project selector

4.1.4 Example 5 – Multilevel Inlines with step selectors

Going deeper, the next example sets the selector on the parent inline step and the inline step has a selector and also inlines another build.

Project Definition for the example:

- Proj_A – Selector: ANY_MACHINE
  - EchoHelloStep – Selector: Windows
  - Inline Inline library – Selector: None
    - ➔ InlineEcho – Selector: Windows
    - ➔ InLine2 inline library – Selector: None
      - Inline2EchoStep – Selector: None
      - Inline2EchoStepB – Selector: None
  - EchoWorldStep – Selector: Linux
  - Inline Inline library – Selector: None
    - ➔ InlineEcho – Selector: Windows
    - ➔ InLine2 inline library – Selector: None
      - Inline2EchoStep – Selector: None
      - Inline2EchoStepB – Selector: None
  - Export Build – Selector: None



Jobs > PROJ_D_BUILD_6

Status: **Completed -- Passed -- Built** Date: **11/4/10 1:16 PM** Project: **Proj_E (Base Snapshot)** Selector: **ANY_MACHINE (Base S**

Step	Step Name	Result	Server (Selector)
1	EchoStep	✓ Passed	Win_Machine_A (Windows)
2	InlineEcho	✓ Passed	Win_Machine_B (Windows)
3	Inline2EchoStep	✓ Passed	Win_Machine_B (Default)
4	Inline2EchoStepB	✓ Passed	Win_Machine_B (Default)
5	CatFile	✓ Passed	Linux_Machine_A (Linux)
6	InlineEcho	✓ Passed	Win_Machine_A (Windows)
7	Inline2EchoStep	✓ Passed	Win_Machine_A (Default)
8	Inline2EchoStepB	✓ Passed	Win_Machine_A (Default)
9	export	✓ Passed	Linux_Machine_A (Default)

The selection rules are harder to understand at first, but they are predictable. The first level has an associated selector, as well as the second level. These steps – EchoStep, InlineEcho (both) and CatFile will all follow rule 1, use the step's associated selector. At the third level, additional rules need to be tested. Because the third level is an inline library and not a project, there is no project-level selector. In this case, rule 3 applies: use the parent scope selector. This selector happens to be the selector associated with InlineEcho, the first child in the above example.

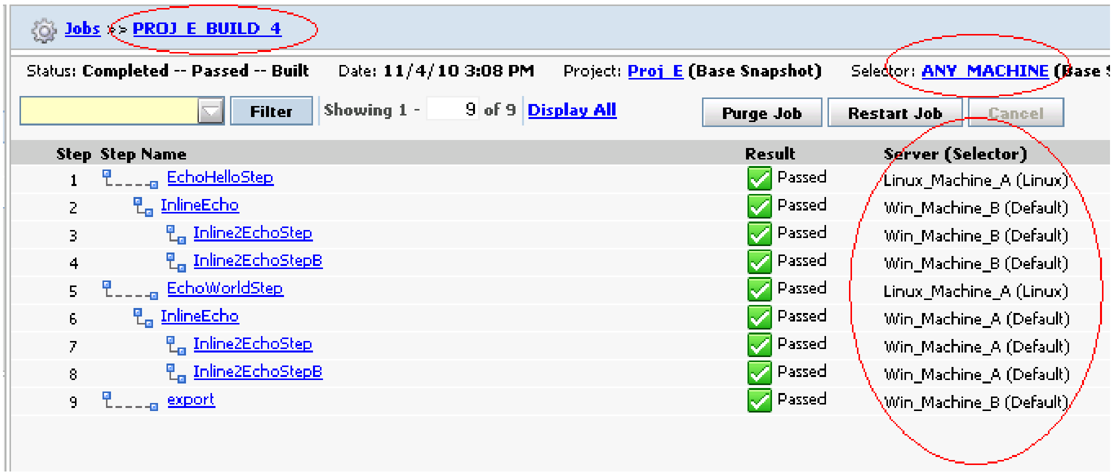
Applying the selection rules:

- Proj_A – Project container, no selector rule needed
  - EchoStep – Selector Use Rule 1) Use step selector
    - ➔ InlineEcho – Rule 1) Use step selector
      - Inline2EchoStep – Rule 3) Use parent scope selector (InlineEcho Selector)
      - Inline2EchoStepB – Rule 3) Use parent scope selector (InlineEcho Selector)
  - CatFile – Selector: Linux
    - ➔ InlineEcho – Rule 1) Use step selector
      - Inline2EchoStep – Rule 3) Use parent scope selector (InlineEcho Selector)
      - Inline2EchoStepB – Rule 3) Use parent scope selector (InlineEcho Selector)
  - Export Build – Rule 2) Use parent project selector

4.1.5 Example 6 – Inline project

Project definition for the example:

- Proj_A – Selector: ANY_MACHINE
  - EchoHelloStep – Selector: Linux
  - Inline Inline project – Selector: Windows
    - ➔ InlineEcho – Selector: None
    - ➔ InLine2 inline library – Selector: None
      - Inline2EchoStep – Selector: None
      - Inline2EchoStepB – Selector: None
  - EchoWorldStep – Selector: Linux
  - Inline Inline library – Selector: Windows
    - ➔ InlineEcho – Selector: None
    - ➔ InLine2 inline library – Selector: None
      - Inline2EchoStep – Selector: None
      - Inline2EchoStepB – Selector: None
  - Export Build – Selector: None



Status: **Completed -- Passed -- Built** Date: **11/4/10 3:08 PM** Project: **Proj_E (Base Snapshot)** Selector: **ANY_MACHINE (Base**

Showing 1 - 9 of 9 Display All

Step	Step Name	Result	Server (Selector)
1	EchoHelloStep	✓ Passed	Linux_Machine_A (Linux)
2	InlineEcho	✓ Passed	Win_Machine_B (Default)
3	Inline2EchoStep	✓ Passed	Win_Machine_B (Default)
4	Inline2EchoStepB	✓ Passed	Win_Machine_B (Default)
5	EchoWorldStep	✓ Passed	Linux_Machine_A (Linux)
6	InlineEcho	✓ Passed	Win_Machine_A (Default)
7	Inline2EchoStep	✓ Passed	Win_Machine_A (Default)
8	Inline2EchoStepB	✓ Passed	Win_Machine_A (Default)
9	export	✓ Passed	Win_Machine_B (Default)

Now the versatility of selection rules really becomes apparent. In the above example, the main calling project has three steps, two of which have their own selectors. The inline is a project, and it too has its own selector. After this, all of the other steps do not have selectors so we can explore the selection rules in detail.

We begin with EchoHelloStep. This step has its own selector and as such follows rule 1, use the step selector. Next, we drop down to the inline project, which has the step InlineEcho. This step follows rule 2, use project-level selector – for the project you are inlining. Because the project-level selector is Windows, the InlineEcho step uses Windows. This step also inlines another library. Because the inline is a library and not a project, rule 2 does not apply to this third level projects step. As such, rule 3 is evaluated for these steps: use parent scope selector, which happens to be Windows from the previous step.

Applying the selection rules:

- Proj_E – Project container, no selector rule needed
    - EchoHelloStep – Rule 1) Use step selector
    - Inline Inline project – Selector: Windows – changes selection rules for the direct steps of this inine to rule 2
        - InlineEcho – Rule 2) Use project level selector
            - Inline2EchoStep – Rule 3) Use parent scope selector (InlineEcho Selector)
            - Inline2EchoStepB – Rule 3) Use parent scope selector (InlineEcho Selector)
    - EchoWorldStep – Rule 1) Use step selector
    - Inline Inline project – Selector: Windows – changes selection rules for the direct steps of this inine to rule 2
        - InlineEcho – Rule 2) Use project level selector
            - Inline2EchoStep – Rule 3) Use parent scope selector (InlineEcho Selector)
            - Inline2EchoStepB – Rule 3) Use parent scope selector (InlineEcho Selector)
    - Export Build – Selector: None

4.2 Impact of sticky

Sticky is a project-level and library-level attribute. With the sticky attribute, you can force all steps in a build *that do not have* a specified selector use only one server. The danger of this approach is the selection process happens only once – when the very first step runs. Only rule 1 still applies for later steps in a build – those steps that have an assigned selector.

The sticky attribute works by assigning the server from the first step to all following steps *within the same scope* that do not have an associated selector. The same scope is defined in this context as all direct steps of the sticky project or library. For example in the following scenario:

- Proj_F – Selector: ANY_MACHINE, sticky
    - EchoHelloStep – Selector: Linux
    - Inline Inline project – Selector: Windows
        - InlineEcho – Selector: None
        - InLine2 inline library – Selector: None
            - Inline2EchoStep – Selector: None
            - Inline2EchoStepB – Selector: None
    - EchoWorldStep – Selector: Linux
    - Inline Inline library – Selector: Windows
        - InlineEcho – Selector: None
        - InLine2 inline library – Selector: None
            - Inline2EchoStep – Selector: None
            - Inline2EchoStepB – Selector: None
    - Export Build – Selector: None

The steps EchoHelloStep, EchoWorldStep, and Export Build are all at the same scope. Sticky only associates the server picked in step EchoHelloStep to the remaining two steps within this scope unless the inlined projects are themselves sticky. If the inlines are sticky, they inherit the server of the calling step, not the selector. If the inlines are not sticky, the inline step inherits the calling step's selector unless the inline is a project. In the case of an inlined project and no sticky attribute, Rule 2 applies before Rule 3 does; the system uses the parent project's selector. This is an important distinction to make: the sticky attribute overrules the base selection rules and deals with the chosen server directly. Finally, there is one rule when projects are involved. A project is defined as a project definition with a selector associated. An inlined project uses its own selector as opposed to the calling step's selector. This is internally following rule 2, use parent project selector. In this case, even though this is an inline the parent project is the inlined project, and if it has a selector the inline steps use this selector. If the project is also sticky, that will be evaluated on the first inline step and stick to that server for the remaining steps within the inline project scope. This occurs *even if* the selector on the inlined project and calling step are the same.

4.3 Impact of .bset

The .bset command has a wide array of options to influence selectors, servers, or buildservers. The syntax is shown below.

```
.bset selector <SelectorName> [<SelectorSnapshotName>]
.bset server <ServerName>
.bset buildserver <ServerName>
```

The Build Forge Online Help illustrates how to use these options. This document mentions them for completeness only.

5 Build Forge internal design for the selection process

5.1 Refresh process

The refresh process uses information found in a collector to fill a server's manifest at regular intervals. Before Rational Build Forge version 7.1.2, the interval is controlled through various throttle settings:

- Active server refresh interval
  - Controls how often built-in collector variable types are refreshed for a server currently running a step.
  - Default: 10 seconds
- Inactive server refresh interval
  - Controls how often built-in collector variable types are refreshed for a server not currently running a step.
  - Default: 3600 seconds, or once every hour
- Default _AGE
  - Controls how often Run Command collector variables are refreshed
  - Default: 86400, or once a day

In Rational Build Forge version 7.1.2 and later, these three settings have been merged into one setting: Server Test Frequency. Both the server test and manifest refresh are joined now to help determine whether a system is still up and able to receive requests, as well as to refresh the manifest. The default value of this new setting is 120 minutes. This setting influences the internal clock mechanisms the refresh process uses. For example, every minute at least one server is refreshed to help smooth out the amount of server tests and refreshes that occur after this time limit is met. The lower the server test frequency is, or higher the count of server definitions, results in more server tests and refreshes in this one-minute window. The most recently, and highest used server definitions receive priority during this internal window. This setting is now more of a dynamic behavior modifier as opposed to the hard-set behaviors in environments before Rational Build Forge version 7.1.2. Now, Rational Build Forge can properly determine which servers actually need to be refreshed because of recent use and leaves the ones that are not used as frequently alone until the server test frequency duration has passed.

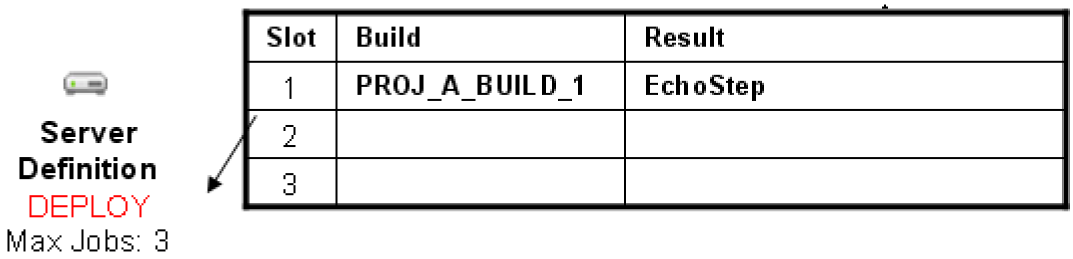
Understanding how the refresh process works is vital to unleashing the power of collector variables. Some built-in variables, such as MEM_FREE, lose their efficacy if they are not updated frequently. Run commands conversely usually only need to be run every week or even just monthly because the value is typically not likely to change. The Server Test Frequency setting helps update all these variable types on a reasonable schedule. Lowering the value refreshes the manifest more often and gathers values for built-in variables more often. However, the lower value also gathers values for the run command collector variables more often, which might not be needed and can consume resources unnecessarily. You must decide what is best in your environment based on the collector variables in use.

5.2 How a job slot works

Each server definition has three job slots by default. Despite the name, steps, rather than builds or jobs, use job slots. With three job slots, a server definition can run three steps at the same time, regardless of whether the steps are from the same build. The Max Jobs server definition attribute controls how many job slots a server definition has.

A step must enter the running state for it to consume a job slot. The step completes the following actions to find and use a server:

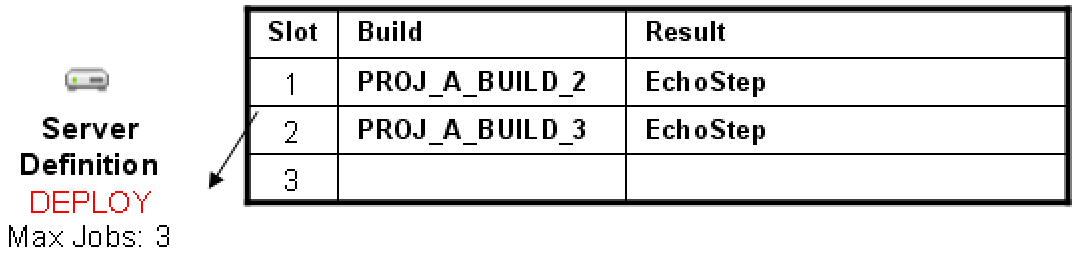
1. Retrieves selector information
2. Uses required variables in a selector to find appropriate agents for consideration
3. Uses optional variables in a selector to pick the most available agent
4. Checks the server's current job slot usage against the Max Jobs attribute for the server
    a. If the current job slots used = Max Jobs, then the server is removed from consideration and the next most available server definition is checked
    b. If no servers have available job slots, the process starts over on the next build loop cycle
5. Adds the job slot to the selected server definition
6. Passes the server details to the step processing logic
7. Step runs



Server Definition
DEPLOY
Max Jobs: 3

Slot	Build	Result
1	PROJ_A_BUILD_1	EchoStep
2		
3		

Run this same project two times together and when both steps start, the job slot consumption appears similar to the following graphic internally:

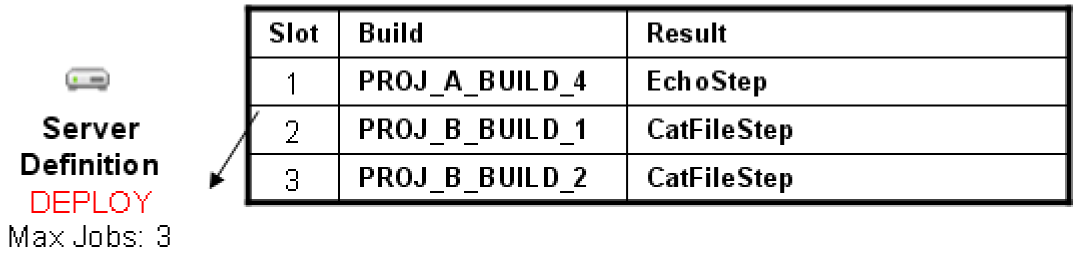


Server Definition
DEPLOY
Max Jobs: 3

Slot	Build	Result
1	PROJ_A_BUILD_2	EchoStep
2	PROJ_A_BUILD_3	EchoStep
3		

One final example to illustrate multiple projects hitting the same server – consider the following two projects:
Proj_A – build tag PROJ_A_BUILD_$B
  ->EchoStep
Proj_B – build tag PROJ_B_BUILD_$B
  ->CatFileStep

Run Proj_A once and Proj_B twice and internally the job slots will appear as:

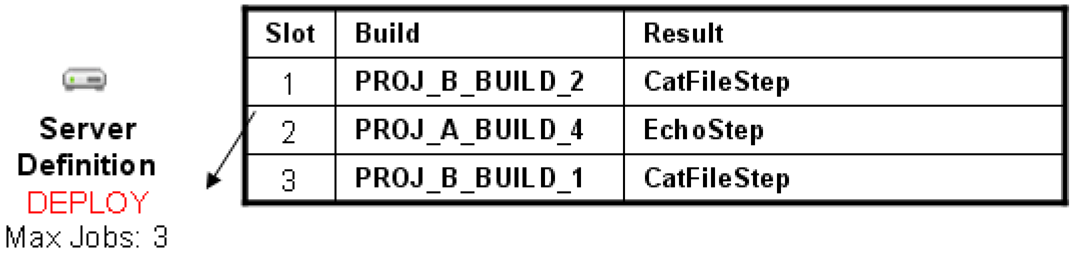


Server Definition
DEPLOY
Max Jobs: 3

Slot	Build	Result
1	PROJ_A_BUILD_4	EchoStep
2	PROJ_B_BUILD_1	CatFileStep
3	PROJ_B_BUILD_2	CatFileStep

Now this server is fully utilized and will no longer accept new steps until one of the currently running steps completes.

The order in the illustrations is arbitrary – it is only meant as a visual example of how internally Build Forge fills job slots on a server. For example the previous picture could just as easily appear like the following:



Slot	Build	Result
1	PROJ_B_BUILD_2	CatFileStep
2	PROJ_A_BUILD_4	EchoStep
3	PROJ_B_BUILD_1	CatFileStep

5.3 Influencing job slots with BF_RESERVE and BF_EXCLUSIVE

You might want to reserve job slots for concurrency or to ensure a build always has a slot available to it. You can control a single slot using the BF_RESERVE built-in variable. With the BF_EXCLUSIVE built-in variable, you can control all job slots on a target server. Be cautious when reserving slots. Reserve slots only when needed.

5.3.1 BF_RESERVE

The BF_RESERVE selector variable reserves a single job slot for a build. Use BF_RESERVE with builds that have only serial steps. Using threaded steps with BF_RESERVE can result in exceeding the Max Jobs value for the number of slots assigned to the server.

You can see the reserved job slots in the manifest for a server through by finding the BF_RESERVE:*step_number* text, as shown in this example:

```
BF_AGENT_VERSION   7.1.2.0-0-0016
BF_JOBS           6
BF_LAST_REFRESH      1288632757
BF_LAST_UPDATE       1288632757
BF_LOADRATIO         0.8
BF_RESERVE:1 f7ddfeec0c4e1000a230133d51fa51fa
BF_RESERVE:2 f7de03220c4e1000ae47133d4b684b68
BF_RESERVE:3 f7de08530c4e1000bc58133d53105310
BF_RESERVE:4 f7de12260c4e10009cac133d50e450e4
BF_RESERVE:5 f7de17270c4e1000ae53133d4b684b68
```

This manifest shows five job slots in the server definition. Five builds are currently running. Each build reserves a slot through the server's manifest and increments the BF_JOBS internal variable.

The BF_RESERVE variable reserves a slot throughout the life of the build, including pass chains with pass wait set in steps.

5.3.2 BF_EXCLUSIVE

The BF_EXCLUSIVE selector variable reserves all job slots for a single build. BF_EXCLUSIVE supports only a single build having access to all the job slots on a server. The steps in the build can be threaded.

In the manifest, BF_EXCLUSIVE appears with the build ID as the argument.

The BF_EXCLUSIVE variable reserves a server for the life of the build.

6 Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

6.1 Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.