IBM Rational Build Forge

Build Forge Architecture and

System Capacity Planning Recommendations

September 2007
Leigh Williamson, IBM STSM, Rational BuildForge Architect

Table of Contents

Overview

This paper describes the IBM Rational Build Forge product architecture with an eye to providing readers with enough information to gauge the hardware needed to support the numbers of users and projects they plan to implement. It also includes some suggestions for product configuration, especially in the area of configuring the product components for redundancy and continuous availability.

Build Forge Architecture

The Build Forge product is a system of components arranged in a classic three-tiered architecture. The components of the Build Forge system are:

- **Web Client**: Web browsers are used by administrators and other users to access the Build Forge functionality. The web browser is considered the primary client user interface.

- **Core Components**: These include the primary user interface, as well as the automation engine. There are three primary elements of the Build Forge core components: the Build Forge *engine*, the Build Forge *console* web application (running on an Apache web server and also known as the Build Forge *UI*), and the Build Forge *service interface* (running on an Apache tomcat server).

- **Database**: The database provides a backing store for the core components. The database stores project definitions, system configurations, and user configurations. It is also where the output logs from project executions are stored. The user has a choice of several database vendors to use for the Build Forge database.

- **Agent**: The agent is an installed remote interface on a host machine that lets Build Forge use it as a resource. An agent must be installed on every host that you want to be used as a resource for Build Forge automation. A host system where a Build Agent is installed is known as a *worker machine*.

- **IDE Plug-in Client**: The *plug-ins* are components that are integrated with supported Development Environment products (Eclipse, Visual Studio, Rational Application Developer) to expose a Build Forge user interface to developers in their own tool environment. They provide an alternative user interface to the primary web application console.

The components that make up the Build Forge system can be deployed in a variety of ways, ranging from all components on a single host to a system that uses clustered core components and a large number of distributed agent resources.

The Build Forge core components are the heart of the product. The core includes an Apache web application user interface that allows users to browse, execute, and manage their build and release projects. In addition to this Build Forge console user interface, the core components include the Build Forge engine and the Build Forge service API. The Build Forge engine contains all of the logic for automated process execution and controls the flow and dispatching of the steps in the automated process. The Build Forge service API exposes a non-graphical programmatic interface so that the system can be controlled by other products and programs.

A functional diagram of the Build Forge system is show in the following figure:

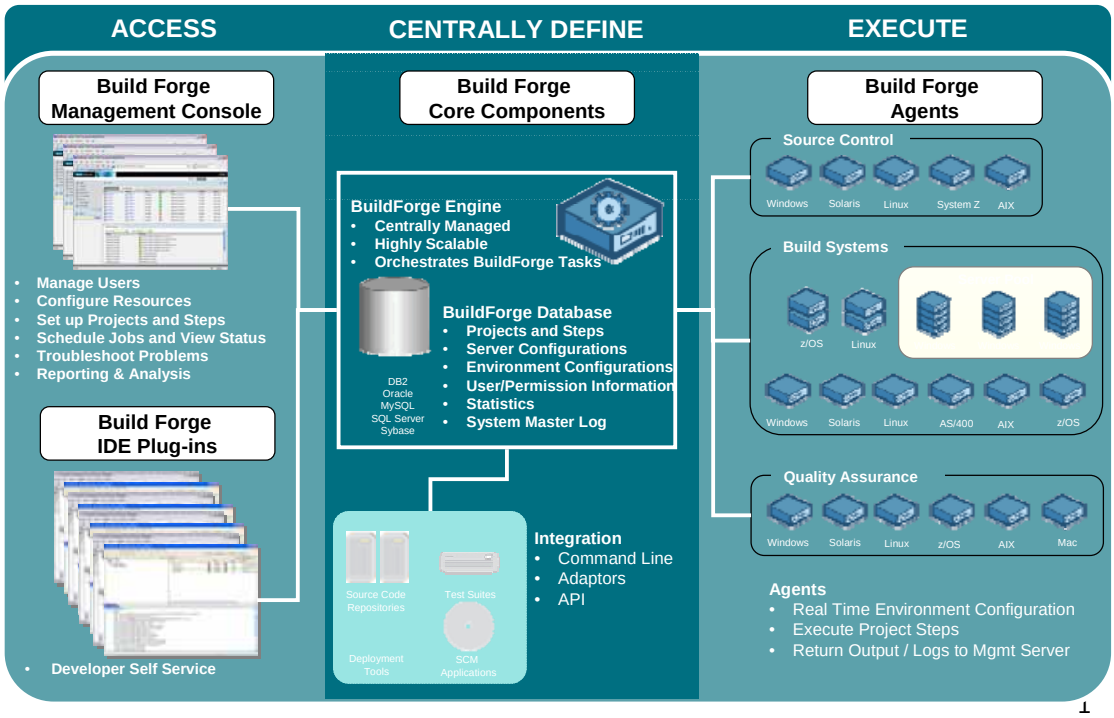


Figure 1 – Build Forge system elements

Typical System Configuration

A typical configuration for the Build Forge system is illustrated in figure 2. The elements of the Build Forge core are typically collocated on the same physical host. In many cases, the database component is also located on this same host machine.

The Build Forge engine reads project configuration data from the database and acts on it. Most typically this means dispatching commands to remote agents executing on the systems where the real work occurs. The dispatching algorithm is sophisticated, supporting the selection of a target system from a pool of machines based on the system characteristics required for the task to be executed. The engine also performs other functions such as sending email notifications when configured tasks are completed, and storing task result information coming back from the agent performing the activity.

The agents are installed on each machine where it is intended that some part of the automated process be executed. Typically there are many agents in use for a Build Forge system, forming a pool of resources from which to select targets for automated process execution. The worker machines contained in a pool of Build Forge resources can be totally different hardware running completely different operating systems.

The Build Forge agent receives instructions from the Build Forge engine and executes those instructions on the system on which it is installed. The agent is very small and simple. It acts as a remote extension of the engine, and does not read or write to the Build Forge database directly. In order for the agent to succeed in executing a particular task, it is typical that environment variables specific to that task must be used. Those environment settings are part of the project configuration stored in the database and transferred to the agent as part of the instructions for each step in the process.

Another element of the product is the set of Build Forge adaptors. An adaptor is a set of reusable instructions for integrating with other products. There are Build Forge adaptors for most source control products as well as many other tools used in the software production process. An adaptor can also be used to supply customized result data back to the engine for storage in the database and subsequent analysis and reports.

The Web Client is a browser typically executed on the end user's machine. The IDE Plug-ins are also executed on the end user's machine, as part of the development tool into which it is integrated.

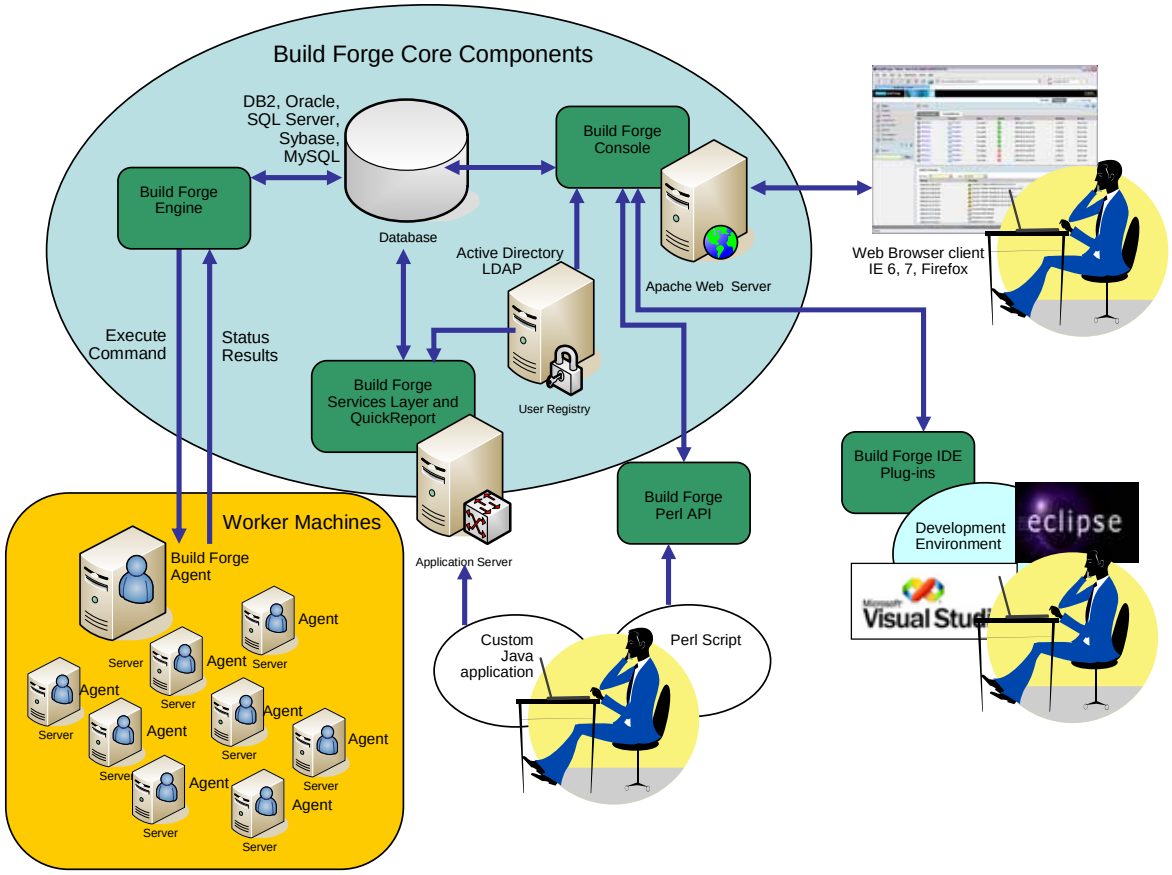


Figure 2 – Build Forge product architecture

The typical configuration, with all Build Forge core components collocated on the same host machine, is quite adequate for small to moderate usage scenarios. As the number of system users increases and the volume of simultaneously executing jobs go up, the core components are typically replicated and placed on different host machines so that the system has the capacity to process the higher volume of work.

The number of projects, steps, project executions, and user sessions that you can have simultaneously running is primarily limited by the computing resources available to the core components. Higher volumes of simultaneous work can be supported by greater numbers of core component instances, as well as larger amounts of memory available to each instance.

Clustered Configuration

In addition to the ability to support greater volumes of simultaneous work, multiple instances of the Build Forge core components provide for system redundancy and more continuous availability of the product functions.

Since Build Forge is built on industry standard technologies such as Apache web server and tomcat server, standard load balancing and clustering techniques can be used in order to spread the work across the multiple instances of those core components.

A typical arrangement for redundancy and improved system availability is to install multiple instances of the Build Forge engine that all work off of the same Build Forge database. These multiple engines will collaborate to pull jobs from the same database and dispatch to the same pool of Build Forge agents. If one engine experiences a crash, the other engines will pick up the load. The instances of the Build Forge engine component can be collocated on the same physical computer or they can be located on different machines from each other (and from the database that they share).

When using the clustered core component configuration for the Build Forge system, it is essential to use one of the databases that are multi-host, network capable. The single, consolidated configuration for Build Forge (where all core components and the database are collocated) can function using a simple non-network supported database product. But the more sophisticated clustered configuration for the Build Forge system cannot.

Geographically Distributed Support

Build Forge components can be distributed to host machines that are widely dispersed geographically. Build Forge supports the ability to partition the Build Forge database and define multiple secondary core component installations for the same system.

When this geographically distributed feature of Build Forge is used, one site is designated the primary site with visibility to all other sites. The other sites are configured as secondary installation sites that can more efficiently process automation steps for the agent machines in their local vicinity. This configuration of the Build Forge system supports centralized oversight while also offering better performance for sites connected to the primary over wide-area networks with higher latency.

This configuration of Build Forge is illustrated in the following picture:

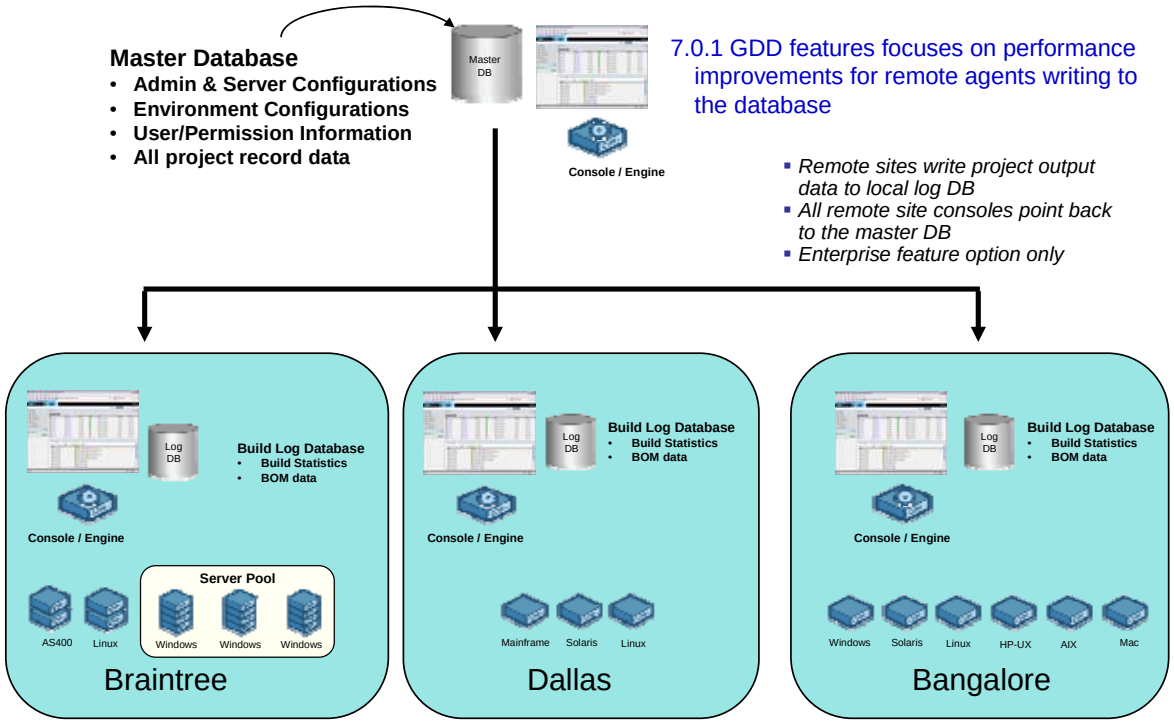


Figure 3 – Build Forge Geographically Distributed Architecture

Build Forge Configuration Elements

The Build Forge system does have performance constraints. The gating factor for scalability is the number of simultaneous jobs and user sessions that can be executed by the core components. In order to understand this, a discussion of some of the elements of the Build Forge configuration is necessary.

Any process that is to be automated is defined in Build Forge as a *project* that contains one or more *steps*. Steps define commands that are to be executed in order to perform a portion of the overall process. Each step has an *environment* associated with it that contains all of the settings for the agent command shell that are necessary in order for the command to succeed. The environment for a step may be explicitly defined for the step or it may inherit a default project environment definition.

Steps

Steps are individual units of work performed as part of a project. Each step contains one or more command line statements to be executed on a selected agent machine. A step can run a script, invoke another application, move or copy files, export a project for backup, and more.

Step

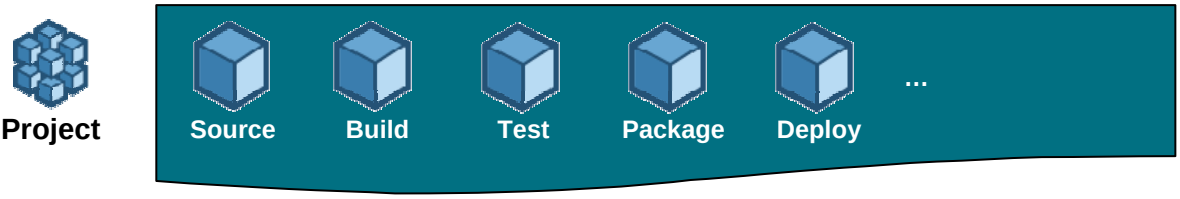
```
1a. cleartool mkview -snapshot -tag $BF_TAG -vws
      $VIEW_STG\$BF_TAG.vws $BF_SERVER_ROOT\$BF_TAG
1b. cleartool setcs -tag $BF_TAG config.spec
```

```
2a. gcc main.c -o main.o
2b. gcc ui.c -o ui.o
2c. make Release
```

```
3a. testscript.sh –run –r $RELEASE –module HelloWorld.exe
```

Projects

A Build Forge *project* is a container that holds a series of steps necessary to execute an entire process. A project could contain a series of automated tests, contain the entire build process for a product, or a Web site update.



Environment

…

A Build Forge *environment* is a collection of environment variables which can be maintained separate from projects and steps, then assigned to projects and steps as needed.

Environment

```
1. RELEASE=Release_1.1
2. JAVA_HOME=C:\Program Files\Java\jdk1.5.0_06
3. PATH=C:\Program Files\Java\jdk1.5.0_06\bin
4. …
```

Operational Architecture

Once a project is defined, it is either scheduled or manually run on an agent machine (or pool of machines) with its own environment variable group. An operational diagram is shown below.

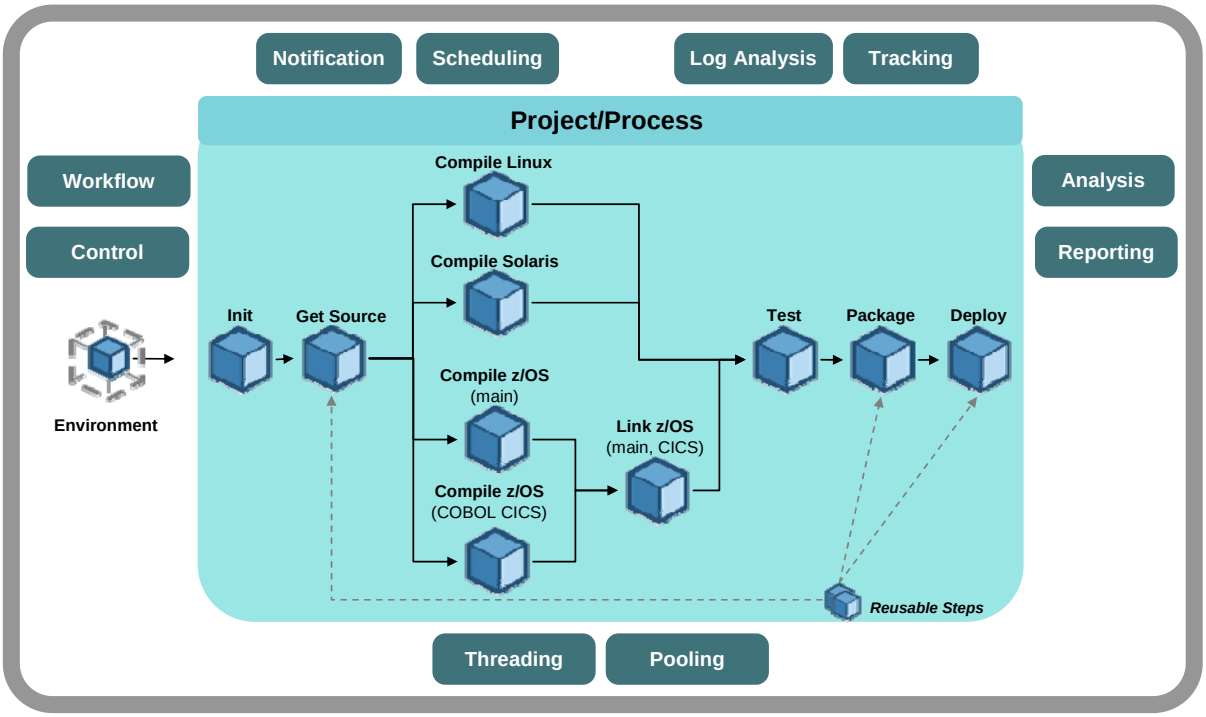


Figure 4 – Build Forge Project Flow

Note that the project can have steps that execute in parallel. For each step in a Build Forge project, a new thread is spawned within the Build Forge engine to communicate with the agent and capture any output from running that particular step.

All of the projects that will be running simultaneously on the Build Forge engine must be taken into consideration when planning the resources to be allocated for that core component. While each step does not require a lot of computing power in the engine, it does utilize memory resources to log the return data. Therefore the practical limit to the number of simultaneous projects that can be run on a single Build Forge engine is the amount of memory available to that engine.

Memory Requirements

This section describes how the Build Forge system uses memory. The goal is to facilitate decisions about the amount of memory needed for computers that host the Build Forge components. The information provided here does not address disk I/O, network bandwidth, and CPU speed and so it is not meant as a performance-tuning guide.

Executables Used by the System

Executables relevant to executing projects from the Build Forge engine include the following:

Process	Description	Memory Usage on Windows	Memory Usage non-Windows
Bfproject.exe	One instance for each project executed on the console	16MB	8MB

These processes each take approximately the same amount of memory on the system hosting the core components, estimated at 16MB of RAM on Windows platforms and 8MB of RAM on other platforms where the core components are supported. The main difference attributed to the memory footprint of Build Forge processes executed on Windows versus non-Windows platforms is the use of statically linked executables versus dynamically linked executables that share library information at run time.

Relevant Build Forge Configuration Settings

Configuration settings that control the number of processes executing on the system hosting the core components include the following:

Parameter	Description
Max Console Procs	Sets the maximum number of processes the core component installation runs at one time. Use as a general throttle on project activity. The system manages Max Console Procs processes by storing an ID for each process in the database, and checking the total before launching a new external process. Make sure this value is greater than your Run Queue Size setting by at least 5; otherwise the system cannot run enough processes to support the run queue.
Max Threads	The maximum number of parallel processes the project is allowed to launch. Use this field to keep a project from using too many system resources. Each thread-enabled step and any inline projects (which themselves might launch thread-enabled steps) can result in parallel processes, but all of those processes are counted against the maximum for the parent project. The system stops launching new parallel processes when it reaches the Max Threads value, and waits until the number of parallel processes for the project drops below the Max Threads value before continuing.
Run Queue Size	This value limits the number of jobs the system attempts to run at once. When the number of runs in the queue equals or exceeds this number, the system stops moving runs from the Wait queue to the Run queue until the number of jobs drops below this value. If you change your Run Queue Size, check the Max Console Pros setting, which should be greater than the Run Queue Size by at least 5.
Max simultaneous server tests	Specifies how many server tests can be run at once. Depending on your system resources, running too many server tests at one time can severely slow or lock up the core components.

Estimating Your Build Forge Server Requirements

As your use of the Build Forge product expands across the enterprise there are several strategies that can be employed to handle the load. The most simple and straightforward is to begin to add more Build Forge core component installations to support individual business units or geographical locations. The two key factors to consider are the number of simultaneous projects to be run, and the number of concurrent user sessions that need to be served.

The following numbers are estimates and are highly dependent on the environment and project variables. These recommendations are provided as a guide and are subject to specifics of your environment.

In terms of Simultaneous Running Jobs (Projects)

Build Forge Core Components

50M RAM overhead + 16M for each project (8M for non-Windows)
1 GHz CPU for every 10 concurrent steps

In terms of Concurrent User Sessions

Anticipated User Load - 100-150 Concurrent User Sessions

Typical Operating system: Windows
Hardware: Server-quality, multiprocessor (dual) recommended
Number of Core Component Installations: 1
CPU: 2 GHz minimum
RAM: 1 GB minimum, 2 GB recommended
HDD: 200 GB recommended

Anticipated User Load – 150-450 Concurrent User Sessions

Typical Operating system: Red Hat Linux
Number of Core Component Installations: 2-3 (150 users per installation)
Dual CPU: 2 GHz minimum
RAM: 2 GB recommended
HDD: 200 GB recommended

Note: The same hardware can support more load when using the Linux operating system. This is due to the more efficient threading and resource utilization of Linux.