Build and automated builds

www.scmGalaxy.com

Author: Rajesh Kumar
rajesh@scmgalaxy.com

scmGalaxy

Build script

- **A build process is expected to help create a single integrated unit that can be deployed on a target environment**
- **A build process can be a sequence of steps that helps in creating a unit**
  - Compilation - javac
  - Integration – jar
  - Creating one unit – war
  - Deploy – put the war file into a particular directory
- **The need for a build script arises once there is a need to build the unit multiple times and by multiple people**
- **Defining a build script to take care of the sequence of steps helps in faster build process and most importantly the ability to standardize on a build and deployment strategy across the team**

Manual build process

- A manual build process is the one where each individual is expected to go through a set of steps based using his or her memory or a documented steps
- With time parts of steps will get side-stepped
- More often than not parts of the code will be deployed rather than the whole
- A properties file change and the related java class change should be released together. One without the other will not be of any use
- A manual build by a developer or a CM is too risky for the project
- Even addition of new steps (like adding code guards using a script before compilation) cannot be done unless a script is used

Build script

- Each software platform will provide a way to write a script and invoke a sequence of build steps
- Unix shell scripts, make files, windows batch files etc can be used to define a build script
- A framework / tool like ANT, helps in abstracting the script from a platform dependency and use simple XML file to define the build script
- Interestingly ANT itself is only a XML notation, that defines the sequence of steps. The steps themselves rely on the code base, framework binaries, SDK binaries etc.
- In summary using any suitable syntax a build script should be defined. The script itself should automate the sequence of steps that needs to be executed to create a single unit of software

Notes on build script

- Choose a language that can be used in multiple platforms (very important for a java project)
- Ensure that as many steps are automated. Begin with a full clean up and allow full build and deployment to the server
- Automate even server restart to increase developer efficiency
- Each build should ensure that it recompiles all classes and optionally fetches latest code from the repository
- Stale references to others code is the most frequent offense among developers. So use the build script to enforce some discipline too

Notes on build script

- **During peak development, the number of builds and changes to a particular piece of code on a developers machine can be as high as 50**
- **If the developer is made to build and deploy as many times even for small changes like JSP, simple Java class change then the productivity of the developer can be affected**
- **While writing a build script, one needs to take care of small short cuts that can help in a faster build and deployment**
  - Deploy only HTML and JSP
  - Deploy only compiled java classes
  - Do only a restart
  - Deploy only properties files
- **This is not a default option and one should not spend time at this in the beginning. This is a step more relevant to iterative improvement of the build script**

Environment and build

- The unit created by the build is dependent on the environment it will run in
- A build is expected to modify environment related dependencies before it creates a integrated unit
- The build script can also modify the environment before it deploys the unit
- It is important to keep in mind the various environments in which a build will execute the parts of the environment that needs modifications during a build
- Typically a environment consists of
  - A application server
  - A web server
  - A database

Environment and build

- **A build could affects all parts of the environment**
  - App server – deploy the ear file
  - Web server – deploy web assets
  - Database – deploy DB scripts
- **A strategy needs to be evolved to ensure that the automated deployment (or manual) affects all parts of the environment**
- **Recreation of the DB at regular intervals, redeploying web assets (images) with each deployment is an important practice to test the code sanity**
- **Ensure that the build script takes care of tagging, building and deploying all related parts of a project**
- **How often each of them should be refreshed differs from case-2-case, but, the build script should be capable of managing all deployments to all parts of an environment**

Build script and dependencies

- **Once an environment has multiple constituents it is important to understand the inter-dependencies between them**
  - A java code base often is dependent on the table names and column names
  - A UI image path defined in a JSP is dependent on the existence of that asset in the web server
  - The whole java code base depends on the existence of a data source defined in the app server
- **More often than not parts of this will change without the other even being informed of the same**

Non-related changes that break the build

- **Case 1**
  - DBA makes changes to the DB model based on the requests from the module leads
  - But the code in the module will not need this change for another week
- **Case 2**
  - DBA modifies a column name of an existing column to resolve a review comment
  - The existing java code needs to reflect this change
- **Case 3**
  - The Driver for a DB connectivity has changed
  - The app server configuration needs to change

Non-related changes that break the build

- **In all these case, a part of the environment has changed and the rest of the application get affected**
- **The key question is how can the build and release process be synchronized to** reduce **impact**
- **The answer is in bringing about a convention and a team level practice that when followed can reduce the impact**
  - First understand the individual parts and their dependency
  - Measure risk of stale data
  - Synchronize the deployments

Understand dependency

- **First understand the dependency, the risk and the level of impact**
  - A web asset if not refreshed only results in an old image being shown – relatively low impact in a dev phase
  - A DB column change can affect the way in which the java code accesses the DB and can result in an exception – high impact
  - A new DB driver will result in a change in the data source configuration. The whole application needs to be tested – high risk

Synchronize the deployments

- **The answer to all problems is in synchronizing the deployments**
  - Strategy 1 - Full deployments each time
  - Strategy 2 - Full application deployment and partial deployments of dependent systems
  - Strategy 3 - Full application deployments and dependent systems deployed at regular intervals

Synchronize deployments - Strategy 1

- **Full deployments each time**
- **Here the DB or other dependent systems are rebuilt with each code deployment**
- **Pros**
  - The dependent systems and the code base are refreshed each time and errors can be caught very early
  - Reference to stale column names, images etc can be easily discovered
  - Stale data is also refreshed each time
  - Complete automation can be achieved
- **Cons**
  - Needs thorough testing and automation of functional testing to ensure that the breaks can be discovered early
  - Frequent builds can lead to high code breaks because the code cannot change at the same pace as a DB
  - Team discipline needs to be very high with respect to code modifications and modifying the dependent assets
  - Build and configuration management team has too many tasks to work with

Synchronize deployments - Strategy 2

- **Full application deployment and partial deployments of dependent systems**
- **The partial deployments to the dependent systems can be controlled and be executed on a need basis**
- **The team decides which script needs to be deployed and only those scripts are deployed with each build**
- **Pros**
  - The partial deployment helps in better scheduling of code modifications
  - Testing can be targeted at a particular area where the highest amount of change has been delivered
- **Cons**
  - Automation cannot be achieved due to human decision intervention
  - Can lead to erroneous code being used till such time the new DB changes are introduced
  - Bugs and errors are caught late in the cycle
  - Coordination between the team could be a weak point

Synchronize deployments - Strategy 3

- **Full application deployments and complete deployment of dependent systems at regular intervals**
- **In this strategy the dependent systems is refreshed at regular intervals (every fortnight, ad-hoc but well announced dates)**
- **Pros**
  - A blended approach and helps teams schedule code deployments better
  - The fact that the dependent systems are rebuilt regularly ensures that the bugs will surface with each rebuild
  - Automation can be achieved.
- **Cons**
  - Code breaks will be discovered late but it will be discovered
  - Needs automation of functional test cases

Frequency of builds

- **A build essentially brings an integrated unit of code and deploys the unit into an environment**
- **It is the first point of integration where compilation errors will automatically be discovered**
- **With a higher amount of integration and integrated builds the risk of integration level bugs will get lower**
- **Typically an integrated build can expect to find the following types of bugs**
  - Method signature incompatibility
  - Incorrect reference to column / table names
  - Some amount of testing can help in discovering flow continuity bugs

Frequency of builds

- The frequency of builds is dependent on the phase of the project
- During the initial phases, code incompatibility is definitely expected, so the builds can be less frequent and sometimes not even required
- Gradually the team begins to use each others code and the inter-dependency increases between individuals and modules
- Initiate a weekly build and then increase it to 3ce a week in a short period of time
- A daily build and a continuous integration build can be a goal once the build process stabilizes and lesser breaks occur

Environments in a project

- How many environments should a project have?

Environments in a project

- **No correct answer. But it is a function of the various types of audience that the project needs to cater to**
- **For a project that is being developed and it is in the dev phase, the various environments could be**
  - Client
  - QA environment Internal testers
  - Integration unit for developers
  - Developer
- **For a project that is on production**
  - Production
  - Production-like (can be dormant)
  - QA for production
  - A QA environment per release being worked upon (usually 1 or 2)
  - Integration environment
  - Developer
- **Understand the need for a new environment and how to decide on the final count**

Environments in a project

- Each environment should have a separate app and web server. It needs to have a separate DB instance
- The integration environment should be machine on which the build itself is executed
- It helps in ensuring that any target platform specific changes can also be done to the code base
- The propagation between some environments should also be thought through.
- Ideally a build that reached QA and once certified by the QA team as release candidate, should automatically be promoted to client and production environments
- Should a rebuild be done on the same tag or should the same code be propagated is a project specific question?
- As a CM understand the needs of each environment and the frequency of builds to estimate your own schedule

Automated build

- **A build script itself is a good start point for automation**
- **It automates the sequence of steps to be executed per build**
- **However, with increasing need for integration and maintaining multiple environments, one needs to worry about the availability of bandwidth to perform such tasks**
- **Automating builds helps answer the following questions**
  - How often should the build be started?
  - What event should trigger it off?
  - On build breaks how should the feedback be routed to the team / individuals?
  - How can an auto deploy be configured and managed?

Automated build

- Automated build can be configured to leverage existing build scripts
- Tools such as cruise control, bamboo etc. provide us the ability to define an automated build and monitor them using web consoles
- The data reported from such tools can be used to provide effective feedback on build breaks and code quality
- Automating builds is the next step after defining the build script
- Multiple builds can be defined to ensure a full build and a full application build are automated and kicked off based on different events
- Automated build helps in reducing the manual work of starting off a build

Note on continuous integration

- **Continuous integration is a term used to describe the process of triggering a build for each incremental code modification**
- **The integration is a continuous process and not a big bang approach**
- **It advices the following rules**
  - single repository
  - Event triggered build
  - Commit code continuously (every day at least)
  - Use a production like machine and deploy the integrated build
  - Execute a fast build (take only modified code)
  - Use these builds to tag and monitor the best version of the day
  - Send feedback of breakages, test coverage reports , code reviews etc

Benefits of continuous integration

- The cost of fixing a bug early in the project is less than doing the same at a later stage in the project
- The key is to identify bugs early.
- CI helps in doing this on an incremental basis
- Ability to use automated builds and monitor the progress of build breaks, code coverage on continuous basis helps in making some decisions easier
- Project delivery risk is reduced

Tool integration with build

- **A typical software project faces the following issues**
  - Lack of adherence to coding standards
  - Lack of good programming practices, especially with novice programmers
  - Lack of white box testing in many cases
- **Using simple command line tools one can integrate a build script with 3rd party tools to obtain simple reports about the code quality**
- **If the build script is written using ant, then most tools provide a simple way to integrate with the ant tasks**
- **It is first important to understand the 3rd party tool and the type of report it generates.**
- **Most importantly, how often the tool should be run and the intent of the report**

Tools integration with build

- PMD – automated code review, standards adherence, dead code, duplicate code etc.
- JUnit – Unit testing and regression test cases
- EMMA – unit test code coverage
- Code guards – adding conditional execution of log statements
- AndroMDA – model driven code generator
- ..
- http://www.javapowertools.com/cooltools

- In summary, each of these tools can be iteratively added to the build script in order to obtain a few reports about the code.
- The feedback on the quality of the code should be given to the team for continuous improvement

Reports and data analysis - demo

- PMD – code review
- JUnit and EMMA – code coverage

Thank You !

www.scmGalaxy.com

Author: Rajesh Kumar
rajeshkumar.raj06@gmail.com