Jenkins Best Practices

Introduction

Continuous Integration (CI) with automated test execution and trends has changed the way companies look at Build Management, Release Management, Deployment Automation, and Test Orchestration. This section describes Jenkins best practices in order to provide executives, business managers, software developers and architects a better sense of what Jenkins can contribute throughout the project lifecycle.

Jenkins Best Practices

> Always secure Jenkins.

This best practice is around authenticating users and enforcing access control on a Jenkins instance
In the default configuration, Jenkins does not perform any security checks. This means any person accessing the website can **execute arbitrary code on the Jenkins master and all connected agents**, including **extracting all your passwords, certificates, and other private data**, besides just configure Jenkins and jobs, and perform builds. This configuration is only acceptable for use in (very small) intranets, and test setups.

> In larger systems, don't build on the master.

If you have a more complex security setup that allows some users to only configure jobs, but not administer Jenkins, you need to prevent them from running builds on the master node, otherwise they have unrestricted access into the JENKINS_HOME directory. You can do this by setting the executor count to zero. Instead, make sure all jobs run on agents. This ensures that the jenkins master can scale to support many more jobs, and it also protects builds from modifying potentially sensitive data on $JENKINS_HOME accidentally/maliciously. If you need some jobs to run on the master (e.g. backups of Jenkins itself), use the Job Restrictions Plugin to limit which jobs can be executed there.

> Backup Jenkins Home regularly.

Between archived builds, build logs that let you determine exactly what happened, and the SCM history information that tells you exactly what was built, Jenkins contains a lot of information you don't want to lose.

> Limit project names to a sane (e.g. alphanumeric) character set

Jenkins uses project names for folders related to the project. Many poorly written tools cannot handle spaces, dollar signs, or similar characters in file paths. So it's easiest to limit yourself to e.g. `[a-zA-Z0-9_-]+` in project names, and use the *Display Name* feature to make them look nice. You can define a pattern for allowed project names in *Configure Jenkins* to enforce this restriction on all your users.

> Use "file fingerprinting" to manage dependencies.

When you have interdependent projects on Jenkins, it often becomes hard to keep track of which version of this is used by which version of that. Jenkins supports "file fingerprinting" to simplify this, so make best use of it.

> The most reliable builds will be **clean builds**, which are built fully from Source Code Control.

To ensure a build can be reproducible, the build must be a **clean build**, which is built fully from Source Code Control. This practice also implies that all code including third-party jars, build scripts, release notes, etc. must be checked into Source Code Control.

> Integrate tightly with your issue tracking system, like JIRA or bugzilla, to reduce the need for maintaining a Change Log

The integration helps to track changes as they are made, including build status, what build has been performed for this requirement or defects, and the link to the actual build results and artifacts.

> Integrate tightly with a repository browsing tool like FishEye if you are using Subversion as source code management tool

Repository browsing provides a quick update on what happens on a Subversion repository. It also provides a graphical diff on what changes have been made from the previous build.

> Always configure your job to generate trend reports and automated testing when running a Java build

Trends helps project managers and developers quickly visualize current project progress status. Moreover, unit testing is often not enough to provide confidence that the delivered software complies to the desired quality. The more you test the software, the better the delivered software complies to the desired quality.

> Set up Jenkins on the partition that has the most free disk-space

Jenkins needs some disk space to perform builds and keep archives. All the settings, build logs, artifact archives are stored under the JENKINS_HOME directory. Simply archive this directory to make a back up. Similarly, restoring the data is just replacing the contents of the JENKINS_HOME directory from a back up.

> Archive unused jobs before removing them.

All unused jobs should be archived so they can be resurrected if the need arises. See Administering Jenkins for ways to do this.

> Setup a different job/project for each maintenance or development branch you create

One of advantages of using CI tools is to detect problems early in the development lifecycle. Setting up a different job/project for each branch you create will help to maximize the benefit of detecting problems early as part of supporting parallel development efforts and reducing risk.

> Prevent resource collisions in jobs that are running in parallel.

Multiple jobs running at the same time often cause collisions if they set up some kind of service, or need exclusive access. If your builds involve use of databases or other networked services, you need to ensure that they don't interfere with each other. Allocate a different port for parallel project builds to avoid build collisions. If that's not possible (e.g. in the case of a persistent resource that needs to be locked) you can prevent builds that use it from running at the same time using e.g. Throttle Concurrent Builds Plugin.

> Avoid scheduling all jobs to start at the same time

Try to avoid scheduling all jobs to start at the same time. If you're using timer triggers or are periodically polling SCM, use the `H` syntax in the cron expression, or predefined tokens such as `@hourly`, to distribute job starting times evenly.

> Set up email notifications mapping to ALL developers in the project, so that everyone on the team has his pulse on the project's current status.

Configure each person on the people list with their correct email address and what role they are currently playing.

> Take steps to ensure failures are reported as soon as possible.

For example, run a limited suite of smoke tests before running time consuming test suites.

> Write jobs for your maintenance tasks, such as cleanup operations to avoid full disk problems.

> Tag, label, or baseline the codebase after the successful build.