Appendix: Converting from Archiva to Nexus

Note: This article was originally published in french on Arnaud Heritier's blog in May 2009. It summarizes the process that Arnaud used to move from Archiva to Nexus.

<disclaimer> As a member of the Archiva team (though rarely active, I admit) I will try to defend it throughout this article. However, being a professional consultant first and foremost, I hope to keep my objectivity. I'll let you be the judge ... </disclaimer>

Having recently migrated a significant number of repository servers from Apache Archiva to Sonatype Nexus, I would like to share with you the process I followed, some tips, and point out a few pitfalls I encountered.

Background

A little background to this migration: We had a large enterprise-scale implementation with dozens of JEE projects of all sizes (from a mouse to the ocean liner - and NO! I am not talking about the Titanic!), and at least 200 active developers accessing the system simultaneously during the day. Most projects are using Maven. To help teams to **make** better and faster builds we offered a complete infrastructure for continuous integration: a big server (Dual QuadCore Xeon with 16GB RAM running Red Hat Enterprise Linux 5.1), hosting an Atlassian Bamboo continuous integration server with (approximately): 110 (real time) continuous integration builds, 40 (daily) maven sites builds, 20 (daily) Sonar builds. It also hosted a Maven repository server (Archiva 1.1.1 before migrating), which stored a little less than a dozen local repositories and provided a proxy and cache for twenty external repositories (for almost 400GB of data). The majority of that space was used by the repository of internal snapshots (200GB) and the repository of internal releases (150GB). To understand these sizes you must know that some projects may produce (for good or bad reasons) some EARs up to 100MB in size. By adding the sizes of JARs and WARs that compose these EARs, each deployment quickly grows to take up a lot of space.

Why change?

For several months we were having stability issues with our integration environment. From one build to another we would see execution times fluctuate widely. We also had intermittent errors that were not related to projects, but to the infrastructure :

- 500 or 502 errors on uploads in Archiva without any apparent reason (I tried to track down the issues without success),
- Many periods of unavailability of Archiva or Bamboo after exceeding the maximum number of open file descriptors allowed for the user. We must concede that these tools handle a lot of files (downloads, uploads, and other manipulations on a large number of artifacts). Archiva easily exceeds the default limit of 1024 descriptors which forced us to increase it. But even at 2048 Archiva would still occasionally exceed the limit.
- Inability to download snapshots if the local repository used by Bamboo was emptied just prior to that. This resulted in regular failures of our Maven sites or Sonar builds.

We recently submitted a bug against Archiva (MRM-1136), for which I spent several hours diagnosing the issue (problems with management of meta-data issued by Maven 2.1.0). To resolve the issue would require us to quickly upgrade to Archiva 1.1.4 or 1.2.

All of these problems, even though they were only occasional, strongly discredited our continuous integration service. How could we criticize a project for not using our CI if it kept sending them false negatives? They already have enough work with addressing real errors in their builds. The continuous integration environment must be as reliable as a Swiss watch.

It was time for us to act.

We could update Archiva, which would entail significant validation costs without much improvement. Apart from the incompatibility with Maven 2.1.0, nothing suggested that the new version fixed any of our problems and there were few improvements/new features at that time, due to lack of time by the maintainers.

The other choice was to get another tool with the potential to meet the new requirements.

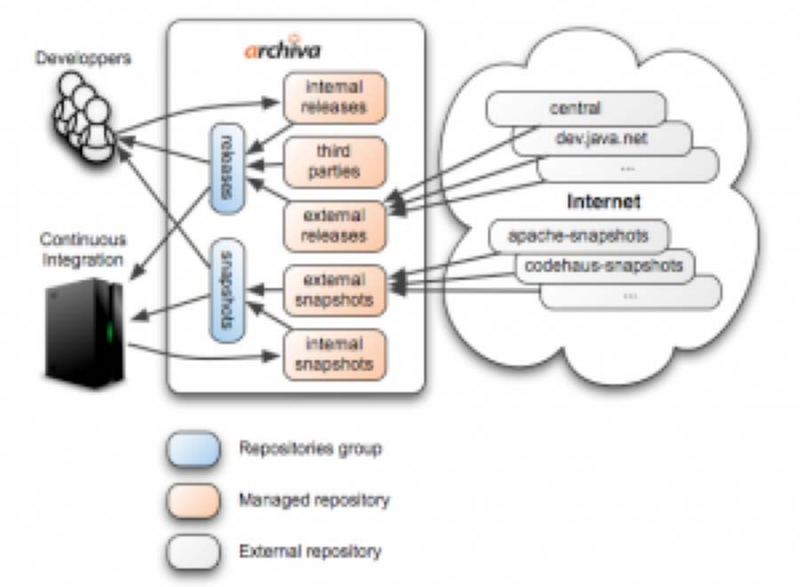
The feedback from the community about Nexus was very encouraging. Moreover, the later pro version opens up new possibilities for the future with its innovative features:

- Staging repositories to put artifacts being validated in a temporary area prior to delivery.
- Opportunity to proxy and aggregate eclipse update sites. Currently we are doing it by hand, which is expensive, complicated (I hate P2), and boring.
- and more...

As the situation seemed to be calling for it, we decided to try our luck with Nexus.

Existing environment

A little overview of our existing environment. Before the migration, our environment was like this:



The environment before the migration

We were hosting our internal repositories in what Archiva calls managed repositories. There is one for releases, one for snapshots and another one for third-party libraries (often those delivered by closed-source vendors). To cache external repositories (configured as “remote repositories”) we are using two more repositories (one for external releases and one for external snapshots).

Our Archiva shows two groups: One to access released artifacts, and another one for snapshots. We never set up a unique group with access to both of them because in our tests we noticed that Archiva often had issues merging descriptors coming from several repositories (maven-metadata.xml), and it can have dramatic consequences if you mix up releases and snapshots.

Groups allow to greatly simplify Maven configuration, avoiding the need to declare too many repositories. Moreover, they enhance your Maven experience, speeding up your builds. Each missing dependency request is sent only once to each group, while the process in the background polls each repository in the group and maintains a cache about missing dependencies to further speed up the build. Sending missing dependency requests to several repositories is an inexcusable waste of time.

Repository users (developers and the continuous integration server) access them using these settings :

```
<settings>
  <profiles>
    <profile>
      <id>default</id>
      <repositories>
        <repository>
          <id>central</id>
          <url>http://serveur.entreprise.fr/archiva/repository/releases/ </url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>false</enabled></snapshots>
        </repository>
        <repository>
          <id>snapshots</id>
          <url>http://serveur.entreprise.fr/archiva/repository/snapshots/ </url>
          <releases><enabled>false</enabled></releases>
```

```
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginrepositories>
        <pluginrepository>
          <id>central</id>
          <url>http://serveur.entreprise.fr/archiva/repository/releases/ </url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>false</enabled></snapshots>
        </pluginrepository>
        <pluginrepository>
          <id>snapshots</id>
          <url>http://serveur.entreprise.fr/archiva/repository/snapshots/ </url>
          <releases><enabled>false</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginrepository>
      </pluginrepositories>
    </profile>
  </profiles>
  <activeprofiles>
    <activeprofile>default</activeprofile>
  </activeprofiles>
</settings>
```

By redefining the central server address (normally located here: http://repo1.maven.org/maven2/) we configured Maven to look only for released artifacts in our group in Archiva. This gives us access to both internal and external releases. In the same way, the repository snapshots allow us to obtain development versions of all artifacts.

Constraints

Migrating a server isn't easy when you have 200 developers using it every day.

First, we had to be sure to only stop the service for a very short time (or we would have to do it outside of working hours... ouch!!!). Stopping the continuous integration server temporarily isn't such a big problem. Asking all teams to use maven in offline mode, and not to use our repositories is more difficult. Usually, we don't have to wait long before someone needs to download a new artifact or a project has to do a release.

Secondly, it was mandatory for us to keep the compatibility with preexisting Maven settings. Due to the large number of projects and developers it would be impossible to change all repositories settings in Maven configuration. It would impact all projects and all developers if we changed the upload URLs (distributionManagement) and download settings (repositories).

The process

To reduce cost, we decided to not perform our tests in a test environment. This would require us to duplicate the entire integration environment we have in production. After some basic tests performed on the tool, the real baptism by fire was the scalability of the server (with an increasing volume of data and number of users). During migration system resources allowed us to easily handle both servers in parallel. We chose the strategy to run Archiva and Nexus in parallel and to switch services to the new server one after another, while keeping the fallback option to revert if necessary.

We had enough space to rebuild caches of external repositories (this required only a few gigabytes), however we could not duplicate our internal repositories, which are too large, and we could have consistency issues. We also did not want to give both products access to internal repositories, because we feared they might access them simultaneously and corrupt our data. Therefore we began our migration leaving our internal repositories on Archiva planning to move them to Nexus at the end of migration.

For URL compatibility we used rewriting rules on the Apache server. It would switch between old and new URLs in a transparent manner.

Installing Nexus

We performed a classic installation of Nexus 1.3.2 (the opensource and standalone bundle) adjusting a few parameters such as the HTTP port, the paths of data and log directories to conform to the organization of our server. All this was quickly done because unlike Archiva we did not have to create a database (which had to be in mysql to follow our standards). Our server was up and ready to be configured.

Configuration of groups and external repositories

We started the Nexus configuration by creating proxy repositories. We also added, for purposes of migration, proxies for our internal repositories hosted on Archiva.

The GUI is ergonomic which makes the registration of 20 external repositories very bearable. Thank you ExtJS!

Unlike Archiva which proposes to store the artifacts of external repositories in the same local repository, Nexus stores the contents of each in a dedicated repository (proxy).

Regarding settings of external repositories, it should be noted that we lose the opportunity provided by Archiva to have white and black lists of artifacts coming from each external repository. Nexus uses the concept of “routes” for this type of filtering. Unfortunately we can apply those routes only at a group level and not on a repository. Since some lists that we have in Archiva are only for performance issues (e.g. do not query a repository in which we know that a certain artifact is not located), we decided to not duplicate this setup in Nexus waiting to see if there were real problems.

In our setting we faced a first disappointment. As we have to proxy some repositories hosted internally (the one delivered by the Sonar server and all of those always on Archiva during the migration) we were forced to declare the company web proxy on each proxy repository. It is impossible to define the web proxy in the global configuration of Nexus and to disable it on a given proxy repository (NEXUS-2317).

While our external repositories configuration was almost complete, we had our first big setback. We discovered that it was impossible to proxy repositories which are in the legacy layout of maven 1 (at Atlassian and dev.java.net for example). To achieve this we need to create virtual repositories which are used by Nexus to convert a repository from one format to another. Tough luck, because two bugs (NEXUS-1909, NEXUS-1910) prevented us from doing it. Fortunately for us, the Nexus team has been very responsive and was able to incorporate the corrections to these bugs in version 1.3.3 which was published the day after our discovery.

Otherwise we have to admit that we probably would not have continued our tests on Nexus. We have updated our server which allowed us to see that the process was very simple since the application, its configuration, and the data were all well separated.

We finalized the configuration of Nexus by creating groups. That is the same concept as in Archiva (which copied it). The configuration of groups is however relatively poorly designed in terms of ergonomics. We had to select each repository we wanted to add to the group and then put it in the correct order in the list. The order is very important, because Nexus will use it to search for artifacts. We must therefore place internal repositories prior to external ones. The problem is that this list displays only the name of each repository (and the name is often truncated because of its length). So you should be careful in naming your repositories to be able to easily sort them when you have several dozen in a group.

We followed Nexus recommendations by creating a single group that exposes all releases and all snapshots. (This will help us avoid having to battle the screen for configuring a group twice ).

From now we were supposed to be able to download all artifacts needed to build our projects with Nexus.

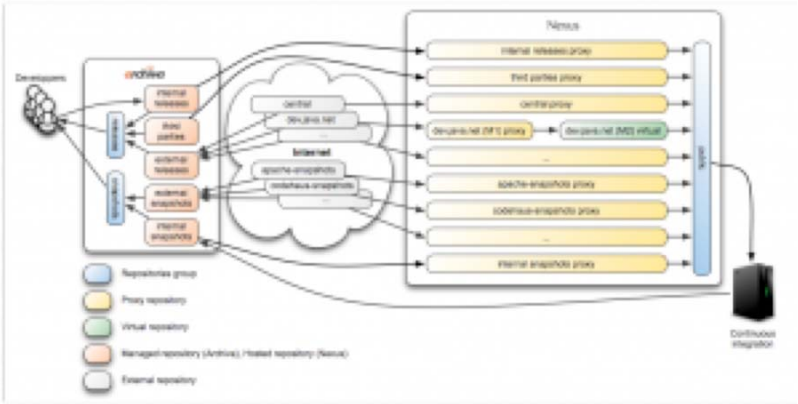
After several successful tests on our desktop we tested the whole system with our continuous integration server. We deleted its local repository and updated its configuration:

```
<settings>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorof>*</mirrorof>
      <url>http://serveur.entreprise.fr/nexus/content/groups/public/ </url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>default</id>
      <!--Enable snapshots for the built in central repo to direct -->
```

```
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central </url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginrepositories>
        <pluginrepository>
          <id>central</id>
          <url>http://central </url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginrepository>
      </pluginrepositories>
    </profile>
  </profiles>
  <activeprofiles>
    <!--make the profile active all the time -->
    <activeprofile>default</activeprofile>
  </activeprofiles>
</settings>
```

Please note: I am not at all a fan of using the mirror * which requires us to put releases and snapshots in the same group. However, this is the only solution, as there is no way to tell Maven to find releases on one mirror location and snapshots on another one. I find this dangerous because e.g. when you call a plugin without defining its version (a plugin in command line such as eclipse:eclipse or archetype:generate) you may retrieve a snapshot version of it. Thus you have to take care to follow the recommendation of Maven to define all the versions of plugins that are used in your project descriptor (directly or by inheritance) to avoid surprises. In our context we won't have this issue since the recommendation is followed by all projects using a parent POM that is setting all versions for them.

The environment with Nexus and Archiva running in parallel was now ready to go :



The environment while migrating

Nothing changed for developers who still use Archiva while the continuous integration server retrieves all artifacts from Nexus.

Tuning

After testing it a few days we could see that Nexus sometimes had difficulty quickly delivering artifacts. The time has come to look at a few more advanced settings.

Routes allow to restrict the list of repositories that Nexus has to verify when we request an artifact to a group. This can be done via permissions or prohibitions.

The first route we create is the one which tells to Nexus to search for our artifacts (*./com.mycompany/.*) only in proxy repositories giving access to our internal artifacts on Archiva. *When the switch to Nexus will be completely finalized, this route will point only to our internal repositories hosted by Nexus.*

We also add rules for *./org/apache/.* and *./org/codehaus/.* which are heavily used by Maven, so that artifacts are retrieved only from the central repository and snapshots repositories of each community.

This gave a pretty good boost to our Nexus implementation.

After several days of use by the continuous integration server, Nexus was running fine. Therefore we got ready for the second part of our migration: hosting of our internal repositories on Nexus.

Setting internal repositories and security

Before we could move internal repositories from Archiva to Nexus we had to create hosted repositories. We created directories for our repositories in a different location from Archiva, as to not let both products access the same data. We reused repository identifiers used in Archiva to easily create rewriting rules on Apache HTTP server.

We spent some time learning the security mechanism in Nexus. Settings are so fine that even the most basic use cases require convoluted settings.

We began by creating privileges (read, write, update) on each hosted repository. We created roles that grouped some privileges on hosted repositories together. In our case, one role to deploy released artifacts on repositories (for projects teams) and another role to deploy snapshots on repositories (for continuous integration server).

We finished by creating users with roles defined above without forgetting the role to download from any repository.

Internal repositories were ready, all was left to do was to move data to Nexus and to stop Archiva.

The final steps

To keep the compatibility between Archiva and Nexus we had to do two things. For each internal repository on which we'll have to upload artifacts we explicitly wrote a rule to transform the URL from Archiva to Nexus. Apache configuration example:

```
RewriteRule ^/archiva/repository/internal-releases/(.*) http://
localhost/nexus/content/repositories/internal-releases/$1 [P]
RewriteRule ^/archiva/repository/internal-snapshots/(.*) http://
localhost/nexus/content/repositories/internal-snapshots/$1 [P]
```

```
RewriteRule ^/archiva/repository/third-parties/(.*) http://localhost/nexus/content/repositories/third-parties/$1 [P]
```

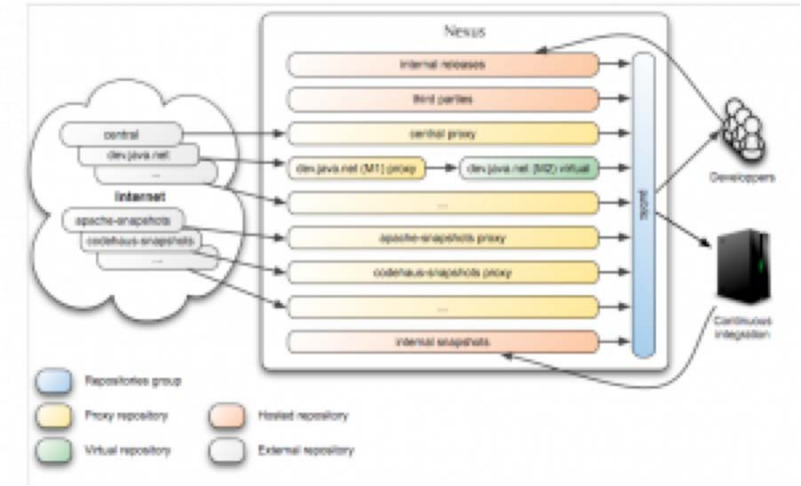
For all other types of read access we just created a rule which transfered them to our unique group in Nexus. Apache configuration example:

```
RewriteRule ^/archiva/repository/[a-z\-]+/(.*) http://localhost/nexus/content/groups/public/$1 [P]
```

Now we were ready to migrate:

1. Stop Nexus.
2. Stop Archiva.
3. Install rewriting rules on Apache.
4. Move content of internal repositories from Archiva to Nexus.
5. Restart Nexus.

On D-day we made the switch in less than 10 minutes. Nexus took a few hours to index the hundreds of GB of data, but without taking much of a performance hit.



The final environment

Despite some pitfalls we encountered, the migration wasn't technically really complex. We did it in 2 weeks and we spent less than half that time actually working on it.

Now we are studying how to configure scheduled services. Sonatype made a major effort on product documentation. It allows us to quickly discover all its features. However, we are disappointed when we want to go further than the reference guide. Some best practices about the usage of the tool in a corporate environment are still missing:

- What scheduled tasks should be used?
- Under what circumstances?
- When?
- Why should we repair metadata?
- Why should we delete caches?
- How are we supposed to use the trash?

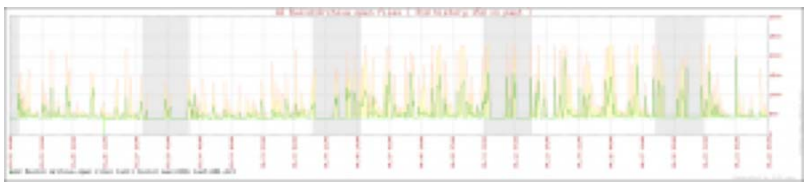
Results

Two weeks after the migration we drew a first conclusion of our Nexus experience.

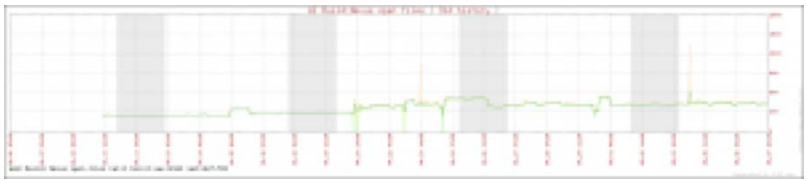
Let's begin by changes on issues we previously had with Archiva and which led us to attempt the Nexus adventure.

The number of file descriptors simultaneously opened

It's a little bit early to declare victory, however after running for two weeks we can see that our problem on the number of file descriptors opened simultaneously disappeared. **Whereas Archiva easily exceeded the 1024 descriptors opened, Nexus uses only 400 which enormously reduces the load on the server.**



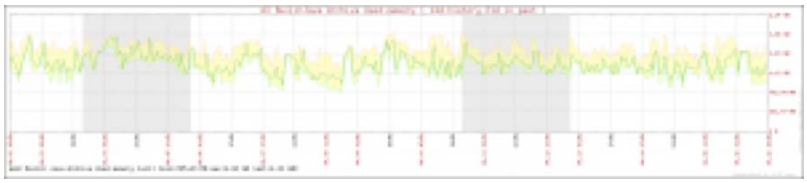
Number of file descriptors simultaneously opened by Archiva



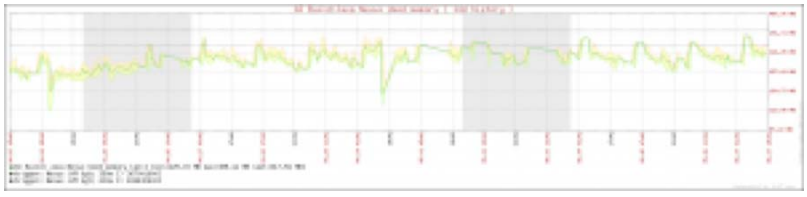
Number of file descriptors simultaneously opened by Nexus

Memory consumption

Even if we had enough RAM, Archiva is a big consumer of memory. **Archiva took an average of 1.3 GB to run. Nexus requires many less and works with 400MB (Even my Eclipse requires more). It saves almost 1GB that we can now allocate to our builds.**



Memory consumption by Archiva



Memory consumption by Nexus

Not found snapshots

Sadly this problems always persists. Because migration to Nexus hasn't fixed it, we investigated a little bit more and discovered two causes for it :

- The first one is a bug in Maven (MNG-4142) : It's a subtle problem halfway between the use of artifacts with classifiers and a corruption of metadata in the local repository.
- The second one is a set of bugs in Nexus (NEXUS-1333, NEXUS-2036): The scheduled service used to repair metadata deletes them if it cannot read it. The problem is that it doesn't succeed to read them if you are using some deprecated properties like {parent.*} (replaced by ${project.parent.*}) or ${pom.*} (replaced by ${project.*}). *Note that NEXUS-2036 is marked as fixed on version 1.3.4.*

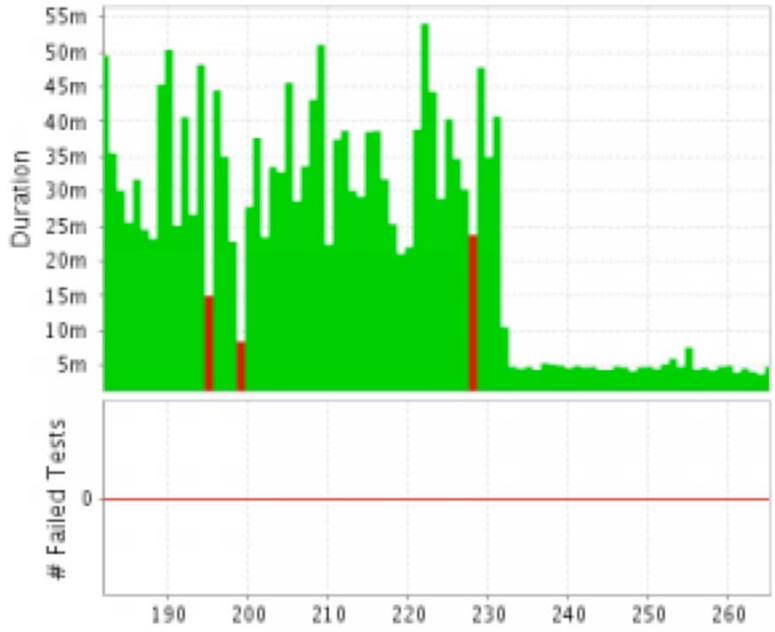
Nothing is perfect

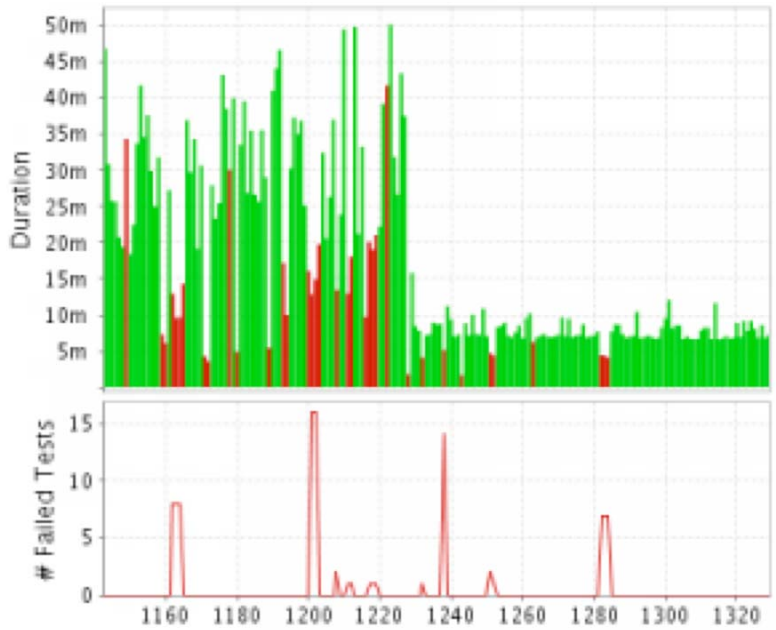
A bug we didn't have before appeared with the usage of Nexus. Because of our usage of rewriting rules in apache, Maven now complains that cookies are rejected when we upload artifacts on an internal repository (NEXUS-1967). This is a minor issue which doesn't alter our system, but is annoying because it pollutes our Maven logs on the integration server.

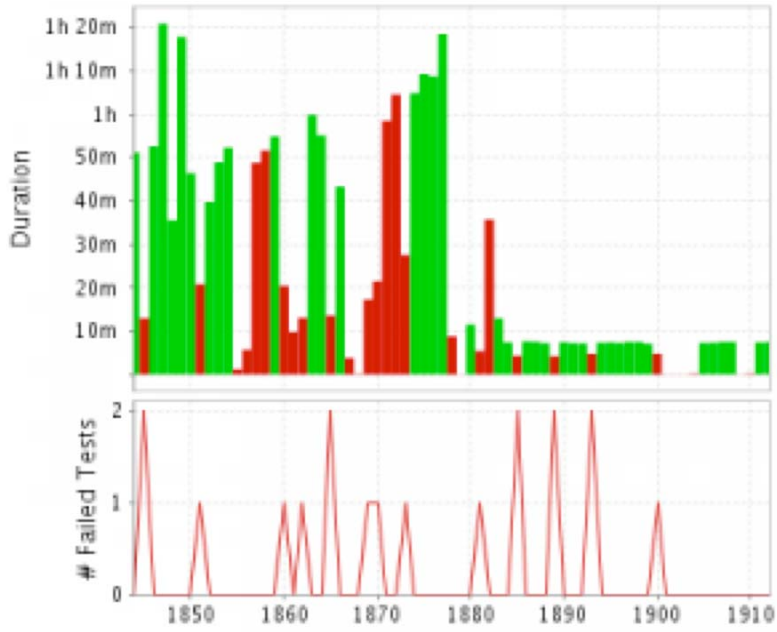
But we had a good surprise

Resource savings on the server (IO, RAM) and the quality of Nexus on the speed to upload artifacts (whatever their size) lead to an important improvement of quality of service and stability of the continuous integration server. As you can see below on some graphs taken from different builds in Bamboo there's a big difference between before and after Nexus. **Build times decreased and are consistent like never before.**


Duration
7m
6m
5m
4m
3m
2m
1m
Failed Tests
50
40
30
20
10
0
40
60
80
100
120
140
160


Duration
55m
50m
45m
40m
35m
30m
25m
20m
15m
10m
5m
Failed Tests
0
190
200
210
220
230
240
250
260





Conclusion

Did we find the silver bullet ? Of course, no. I have already listed some constraints and problems we had and there are probably many others to discover (the team already has a good backlog). Any product can be improved. Despite all that, how could I say something other than... migrate! It's unfortunate for me to have to admit this as a member of the Archiva team, but my own research proves it. Nexus is a product of great quality, which delivers the service you need for your development projects very well. Furthermore, developed by full-time

employees (thank you Sonatype), it enjoys an easily accessible and responsive support (I recommend the channel #nexus on #irc.codehaus.org). Archiva itself is governed by the laws of open source, suffers from the lack of time of its maintainers (even if they are doing as much as they can) which doesn't help it to stay in the competition.

So, enjoy and benefit from service and quality by adopting Nexus!