Item 265
git_comments:

git_commits:

1. **summary:** [CXF-6869] Updating SpringBoot version to 1.3.5.RELEASE, patch from Vedran Pavic applied with thanks
   message: [CXF-6869] Updating SpringBoot version to 1.3.5.RELEASE, patch from Vedran Pavic applied with thanks

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Consider adding Spring Boot starter
   description: I've recently authored a PR in Spring Boot to add support for auto-configuration of {{CXFServlet}} and default CXF's configuration: https://github.com/spring-projects/spring-boot/pull/5659 The PR was closed with "won't fix" resolution since Boot team are unwilling to add CXF as a dependency to the project. Instead a 3rd party starter was suggested. The concept of a 3rd party starter is generally encouraged for technologies that don't have first-class support in projects from Spring portfolio. Such 3rd party starters are listed here: https://github.com/spring-projects/spring-boot/blob/master/spring-boot-starters/README.adoc If CXF team is interested, I'm willing to port my PR to CXF. Note that the original PR was focused around JAX-WS support, but can be easily expanded to include JAX-RS support as well.

jira_issues_comments:

1. +1, can you please consider porting your PR ? Supporting both frontends would be great Cheers, Sergey
2. Sergey, I'll be glad port it. Hopefully will be able to fit that to my schedule within the next week or two. Since this will result in a new CXF artifact(s), I need you to point me to place where to put my code. At first look, it looks to be a fit under integration modules. Artifacts should be named with respect to Spring Boot's guidelines for 3rd party starters: http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-custom-starter Assuming the integration assumption from above was correct, the modules would be named {{cxf-integration-spring-boot-autoconfigure}} and {{cxf-integration-spring-boot-starter}}. Would it be possible to omit the {{integration}} part? Regarding sample application, I've seen your already have a space for that.
3. Hi, sure how about integration/spring-boot being a parent directory containing /autoconfigure (cxf-spring-boot-autoconfigure) /starter (cxf-spring-boot-starter) ? By the way I've read in their docs it could be a single module combining the auto-configuration and the starter features. Would it make sense to combine or it is better to keep it separate ?
4. Thanks Sergey, the proposed layout looks good to me! Regarding the separation question - although there are no hard constraints on 3rd party Boot integrations with regard to this concern, it is a Spring Boot practice for starter modules to contain no code. If having two extra modules is not an issue for you, I'd recommend to take that route and be consistent with Boot's approach.
5. Sure, lets follow the best practice Thanks
6. Hi, are you still planning to submit a patch ? Thanks
7. Hey Sergey, sorry for the delay. Yes, of course - I've done some work already and will hopefully submit the PR within the next couple of days. I've also been thinking about the modules, we should have two starters - one for JAX-WS support and the other for JAX-RS. Is that a problem for you? This is so we don't pull in all the dependencies with single starter since some users will be interested in JAX-WS but not JAX-RS, and vice versa.
8. Hi, great, no problems at all, thanks for keeping looking at this issue. It would be nice to get it into 3.1.7, and I guess we have a couple of weeks at least, so no rush :-). You are right, makes sense to avoid bundling JAX-WS and JAX-RS deps together Thanks, Sergey
9. GitHub user vpavic opened a pull request: https://github.com/apache/cxf/pull/133 [CXF-6869] Add Spring Boot support This commit adds Spring Boot support which includes: - Auto-configuration module to provide auto-configuration capabilites for ```CXFServlet``` and ```SpringBus``` - JAX-WS starter module which assembles dependencies required for development of JAX-WS services - JAX-RS starter module which assembles dependencies required for development of JAX-RS services @sberyozkin Please review the PR and let me know if everything is OK from the prespective of CXF project. One question from my side - you mentioned you'd target this at 3.1.7 release, however I've based my PR on the current master, is this an issue? I still plan to test the integration by creating some sample Spring Boot projects. We can update the existing sample projects to make use of this integration in a separate PR if that's OK with you. You can merge this pull request into a Git repository by running: $ git pull https://github.com/vpavic/cxf CXF-6869 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/cxf/pull/133.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #133 ---- commit 606828ba0def77888c0fdbd44ee403547da510c6 Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-05T16:43:58Z [CXF-6869] Add Spring Boot support This commit adds Spring Boot support which includes: - Auto-configuration module to provide auto-configuration capabilites for `CXFServlet` and `SpringBus` - JAX-WS starter module which assembles dependencies required for development of JAX-WS services - JAX-RS starter module which assembles dependencies required for development of JAX-RS services ----
10. **body:** Quick remark: on the spring boot side we tend to avoid creating too much starters for the same library. We often had that request and it was always rejected so far. We prefer that the user adds an additional dependency rather than having to chose between several starters. Looking at the starters, the only difference is one jar. Maybe you should settle for one starter and simply let the user adds the other jar manually if needed?
    label: code-design
11. Thanks, see, CXF users who will prefer to use a JAX-WS starter will unlikely want to use JAX-RS alongside, and it is even more likely that JAX-RS users will not want to have a JAX-WS loaded. In a way, these two starters represent different projects. Some users will be happy to combine, but forcing an either frontend category to have the other one included (JAX-WS -> JAX-RS and vice-versa) will be problematic. I.e., while technically it is a one jar difference, it is bigger for CXF users than just a 1 jar :-) as we may be talking about the major dev preferences (how to write services, etc)
12. The starters may start diverging soon, we will need to review, for example, if having a default JAX-RS starter should ensure Swagger feature is loaded, or Jackson is loaded, or some of security interceptors are up by default. In Karaf features we have for ex some basic feature, with another one extending it with few more modules added, etc. Ex, CXF JAX-RS which is similar to this JAX-RS Spring Boot starter, and then an RS security one, etc. I wonder now if we can emulate that with SpringBoot starters given what Stephane just said.
13. Github user asfgit closed the pull request at: https://github.com/apache/cxf/pull/133
14. Vedran, many thanks for this patch, applied with minor updates (added another pom inside integration/spring-boot) and a minor update to a hamcrest dependency. I'll experiment with updating a SpringBoot JAX-RS demo next, see how having an autoconfigurer and a starter will make the life easier :-), I can already see it will be more optimal, thanks
15. Vedran, I've started experimenting with the new modules, see http://git-wip-us.apache.org/repos/asf/cxf/commit/565319f2 (I can similarly update the JAX-WS demo once I figure few details) I have few questions, but the most immediate one is how to customize a CXFServlet URL pattern. In this demo, before I applied the latest update, after doing "mvn spring-boot:run", one could click at "http://localhost:8080/services/helloservice/sayHello/ApacheCxfUser" and get some data back. Note, the pattern is "/services/helloservice/*". After my last update I see the server responding to "http://localhost:8080/services/sayHello/ApacheCxfUser", i.e, the pattern is now a default "/services". In the demo I created a main/resources/cxf.properties file which contains "path=/services/helloservice", but after rebuilding and starting, it is still ignored. Any idea what am I missing ? Thanks
16. GitHub user vpavic opened a pull request: https://github.com/apache/cxf/pull/134 [CXF-6869] Fixes for Spring Boot integration You can merge this pull request into a Git repository by running: $ git pull https://github.com/vpavic/cxf CXF-6869 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/cxf/pull/134.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #134 ---- commit 2032092e9b550b20109128231ca70df1fa2d232e Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-09T20:48:32Z [CXF-6869] Fix Spring Boot integration dependency management commit c878833740af3eb361674b5ece30e0ef4da392b4 Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-09T20:49:18Z [CXF-6869] Fix CxfAutoConfiguration dependency injection commit

05a58810677c5a9c2a7daa32d7103189327f728e Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-09T20:49:54Z [CXF-6869] Fix configuration properties prefix in CxfAutoConfigurationTests ----

17. Hey Sergey, you suprised me by merging this so quickly :) To answer you question first - by default Spring Boot loads configuration properties from {{application.properties}} file. Or {{application.yml}}, if you prefer YAML for your configuration. So you need to place {{cxf.path}} property in that file to customize the servlet mapping. Having said that, I noticed that {{CxfAutoConfigurationTests}} did not execute during project build. If it did, it would've failed since I forgot to adjust the tests to use {{cxf.}} configuration properties prefix, instead of {{spring.cxf.}}. This is a remnant for my original PR which was opened against Spring Boot project. Another remnant that causes failure is dependency injection in {{CxfAutoConfiguration}} - I've used construction injection in {{@Configuration}} class which is not supported until Spring 4.3. The original PR was targeted at Spring Boot 1.4 (which uses Spring 4.3) - in CXF we have to refer to a stable Spring Boot version, which mean we'll be using Spring 4.2.x. The third omission was in dependency management department - starters did not include {{spring-boot-starter-web}} as a dependency. I've opened a PR to address these 3 issues: https://github.com/apache/cxf/pull/134 Finally, I've put together a JAX-WS sample application to help you with your tests: https://github.com/vpavic-samples/spring-boot-cxf-jax-ws Please let me know what's up with the execution of {{CxfAutoConfigurationTests}}. Thanks.
18. Hi, thanks for another patch, I'm at Apache Con right now, so will try to address it later this week or early next one cheers
19. Thanks for letting me know Sergey!
20. Hi Vedran, I applied your 2nd patch, thanks, the changes look good. However, I'm still unable to customize a servlet URL pattern, adding application.properties, see http://git-wip-us.apache.org/repos/asf/cxf/commit/616ebbc9 makes no difference so far as far as the demo is concerned, i.e. I'm seeing what I described in https://issues.apache.org/jira/browse/CXF-6869?focusedCommentId=15276567&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-15276567 But I feel we are very close to making it work :-). As I said, I will also update a JAXWS demo afterwards cheers
21. Hey Sergey, sorry for late response. I've been swamped with some conference preparations lately and didn't find time to take a closer look at this. Hopefully I'll be able to look into this by the end of this week.
22. Hi Vedran, np at all, thanks for the update, good luck with your talks Sergey
23. **body:** Sergey - I'm back at this, sorry for the delay :) Regarding your sample, I've taken a better look at it, and the problem is that the auto-configuration isn't used at all. This is because your Spring Boot main class ({{SampleScanRestApplication}}) imports {{SpringComponentScanServer}}, which in turn imports {{JaxRsConfig}} which imports {{META-INF/cxf/cxf.xml}}. This registers {{SpringBus}} bean which effectively disabled auto-configuration since it has condition {{@ConditionalOnMissingBean(SpringBus.class)}}. I'm working on improving the auto-configuration to provide better support for JAX-RS use case. Hopefully I'll have this finished very soon, together with my own JAX-RS sample application (similar to JAX-WS one I already provided). Additionally, I saw you removed some imports from {{cxf-spring-boot-autoconfigure}}'s POM in commit {{f8d941e}}, any particular reason for that? I'll need {{cxf-rt-frontend-jaxws}} and {{cxf-rt-frontend-jaxrs}} dependencies back for obvious reasons, and regarding {{spring-boot-configuration-processor}} you can find more info here: http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#configuration-metadata-annotation-processor
    label: code-design
24. Hi Vedran Np, thanks for finding the time. I removed those dependencies because they were 'optional' and the auto configuration module compiled OK without them and jaxws and jaxrs starters do list them, I'm OK with bringing them back, can you explain why it is important ? SpringComponentScanServer helps with auto-discovering JAX-RS specific resources, but it is not the only option, SampleRestApplication (currently disabled) imports JaxRsConfig directly and at the moment I can switch the demo to use SampleRestApplication and remove JaxRsConfig. I can also consider removing JaxRsConfig from the CXF JAX-RS code and document that uses working with SpringComponentScanServer should import it directly, but I'm a bit concerned it might affect some users as I know SpringComponentScanServer is being used. So yes, please check if you can make Autoconfiguration 'prevail' even if some existing code happens to import cxf.xml, I believe I've seen CXF JAX-WS code fragments targeted for SpringBoot where it is imported too. Thanks
25. Regarding dependencies, at the time of writing my comment I had already started refactoring the auto-configuration and the changes now do require frontend modules to compile. Comment on {{spring-boot-configuration-processor}} still checks though. I've separated CXF servlet and JAX-WS/JAX-RS auto-configuration concerns to provide more granular control. I'll submit the PR with these changes at some point today.
26. I'll put the configuration processor dependency back now, thanks for the link. Re JAX-RS vs JAX-WS auto configuration, as I said the fact SpringComponentScanServer indirectly imports cxf.xml is not JAX-RS specific but is rather specific to the way this code has been done in CXF JAX-RS. I can simply remove JaxRsConfig from https://github.com/apache/cxf/blob/master/rt/frontend/jaxrs/src/main/java/org/apache/cxf/jaxrs/spring/AbstractSpringConfigurationFactory.java#L34 if it can help ? FYI, removing it in the demo did help, the auto configuration is effective if no SpringComponentScanServer is imported: http://git-wip-us.apache.org/repos/asf/cxf/commit/9b1b6cb Cheers, Sergey
27. **body:** To expand a bit on my previous comment - the improvements I'm about to commit today will include {{SpringComponentScanServer}} in auto-configuration, since it offers obvious benefits in JAX-RS use case. I'm a long time CXF user but have been using it only for JAX-WS so excuse my lack of knowledge in JAX-RS department, but I'm catching up :) Regarding the removal of {{JaxRsConfig}} import from {{AbstractSpringConfigurationFactory}}, it would certainly help the configuration granularity and I'm all for it. However I don't know if that's a problem for you from backwards compatibility standpoint. IMO ideally you wouldn't want to tie {{SpringBus}} configuration with some particular frontend configuration since, AFAIK, there should be a single {{SpringBus}} instance even in scenarios with multiple frontends.
    label: code-design
28. Hi, thanks for the clarification, and in advance, for your patience :-). So I did experiment with removing @JaxRsConfig from the CXF JAX-RS source - I feel that if it is what blocks the AutoConfigure feature from being operational in JAX-RS cases then it is reasonable to ask existing users to import it directly if needed - minor migration issue. However it did not make any difference as far as loading the auto configuration feature is concerned - it did not work when I updated the demo to start the application which imports SpringComponentScanServer (which this time does not bring @JaxRsConfig any longer). I'm a bit confused right now :-) FYI, I'm not sure having SpringComponentScanServer imported for JAX-RS auto configure by default would work for all the users, there will be cases where they'd prefer to avoid the auto-discovery. I'd like to understand first why importing SpringComponentScanServer has side-effects as far as having the auto-configuration applied. I wonder if it is an ordering issue: importing SpringComponentScanServer starts a JAX-RS endpoint (in its jaxRsServer() bean) - if CXFServlet has not been loaded by that time then an embedded CXF Jetty endpoint will be created. So may be it can be enforced somehow, via a conditional annotation, that SpringComponentScanServer is processed only after the auto-configuration, if any, has been applied... Cheers, Sergey
29. Hi Vedran Re my earlier comment: "FYI, removing it in the demo did help, the auto configuration is effective if no SpringComponentScanServer is imported: http://git-wip-us.apache.org/repos/asf/cxf/commit/9b1b6cb", it was not complete, I've just realized that that demo application was still loading CXFServlet itself, so I removed it: http://git-wip-us.apache.org/repos/asf/cxf/commit/e95c92a3 and it stopped working, i.e, no AutoConfigure is loaded as far as I can tell. In this case neither SpringComponentScanServer nor JaxRsConfig is loaded. I've tried to move this code https://github.com/apache/cxf/blob/master/distribution/src/main/release/samples/jax_rs/jaxrs_spring_boot/src/main/java/sample/rs/service/SampleRestApplication.jav to a separate Configuration bean as in your demo: https://github.com/vpavic-samples/spring-boot-cxf-jax-ws/blob/master/src/main/java/sample/ServiceConfig.java but no luck, CXFServlet is not loaded at all. Which further confuses me - I said in the previous comment about SampleRestApplication2 (which imports SpringComponentScanServer) that it would create a CXF Jetty endpoint if no CXFServlet is loaded (just to try and explain why would the following be the case: https://issues.apache.org/jira/secure/EditComment!default.jspa?id=12959484&commentId=15276567). But the demo project has no http-jetty dependency, so CXFServlet is loaded somewhere :-) Cheers, Sergey
30. No problem Sergey, I'll give my best to explain the current state, and what I'm trying to achieve :) With the current state of auto-configuration, everything is dependent on the presence of {{SpringBus}} bean in the application context ({{@ConditionalOnMissingBean(SpringBus.class)}} in {{CxfAutoConfiguration}}). This means if one either declares that bean manually, or imports {{META-INF/cxf/cxf.xml}} nothing is auto-configured. This is what you've been observing with your JAX-RS demo app. To verify this, try starting your Spring Boot apps with {{--debug}} switch (see http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#howto-troubleshoot-auto-configuration for more info on that). Then search for {{CxfAutoConfiguration}} in the log output. I'd like to improve things not to have such strict criteria for disabling auto-configuration altogether. For example, if one declares {{SpringBus}} bean manually, the auto-configuration for {{CXFServlet}} should still work. I'd also like to make use of {{SpringComponentScanServer}} in auto-configuration, since most Spring Boot users would certainly be interested in benefits it offers, but at the same same have the possibility to easily disable it without dropping other parts of auto-configuration (servlet, bus). Having said that, removing of {{JaxRsConfig}} import from {{AbstractSpringConfigurationFactory}} is highly beneficial since it would allow a clean separation of configuration concerns - servlet/bus/frontend-specific-config. Hope this helps :)
31. I've put up a JAX-RS sample that demonstrates how simple it should be to use CXF JAX-RS auto-config: https://github.com/vpavic-samples/spring-boot-cxf-jax-rs Note that this of course won't work until I finish the PR. I'm multi-tasking a lot today but I expect to finish it today :)
32. Hi, yeah, I saw it, thanks. Please take your time :-) However, I do expect https://github.com/apache/cxf/blob/master/distribution/src/main/release/samples/jax_rs/jaxrs_spring_boot/src/main/java/sample/rs/service/SampleRestApplication.jav working with the existing spring-boot/autoconfigure, without any specific JAX-RS auto-configure support, right ? Cheers, Sergey

33. **body:** Hey Sergey, The thing that's causing the issue with your sample is it's unnecessary complexity - you have two Spring Boot main classes in your project, {{SampleRestApplication}} and {{SampleScanRestApplication}}. It does not matter that you explicitly mark one of them as main-class in your {{pom.xml}} - the other is still in the same package tree and it's a valid {{@Configuration}} class (by virtue of {{@SpringBootApplication}} being composite annotation which, among other things, includes {{@Configuration}} annotation) so it's going to get picked up by Spring's application context component scanning mechanisms. In practice, this means that when you designate {{SampleRestApplication}} as your Spring Boot main-class the {{@Import(SpringComponentScanServer.class)}} from {{SampleScanRestApplication}} will still be in effect. This kind of usage is actually not considered the best practice in Spring Boot world. I encourage you to reuse the samples I've prepared as much as possible - they are simple and provide best practices without additional complexity.
    label: code-design
34. Another problem is the {{JaxRsConfig}} class itself - it's annotated with {{@Configuration}} and this makes it not quite Spring Boot friendly since it's being automatically picked up every time even if you don't import it yourself, or import the {{SpringComponentScanServer}} which imports it indirectly.
35. GitHub user vpavic opened a pull request: https://github.com/apache/cxf/pull/139 [CXF-6869] Improve Spring Boot auto-configuration support You can merge this pull request into a Git repository by running: $ git pull https://github.com/vpavic/cxf CXF-6869 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/cxf/pull/139.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #139 ---- commit d5c7548f03f4a4c2c633e1a7bdad573febc710b8 Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-27T23:03:35Z [CXF-6869] Upgrade Spring Boot to 1.3.5.RELEASE commit 89c0f7ce91c70fad8715c2e517eb6856b9173f9a Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-27T23:06:01Z [CXF-6869] Add cxf-rt-rs-service-description to cxf-spring-boot-starter-jax-rs commit 57a57ed68af31e471468cff17bca385e45d7bde7 Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-27T23:07:41Z [CXF-6869] Improve CxfAutoConfiguration commit e36fd658456bfa1221b2365c9f7b30faca19d55d Author: Vedran Pavic <vedran.pavic@gmail.com> Date: 2016-05-27T23:26:33Z [CXF-6869] Polish JAX-RS Spring components ----
36. Sergey, I've created the PR to address the discussed concerns. Changes are probably obvious from the commit messages but I'll explain what I've done once more: - updated Spring Boot dependencies to {{1.3.5.RELEASE}} - added {{cxf-rt-rs-service-description}} to {{cxf-spring-boot-starter-jax-rs}} so that the {{?_wadl}} links from service listing page work out of the box - improved the auto-configuration - removed {{@Configuration}} from {{JaxRsConfig}} and removed {{JaxRsConfig}} import from {{AbstractSpringConfigurationFactory}} - these two were the main offenders from the auto-configuration perspective The auto-configuration support now works like this: - main auto-configuration condition is now {{@ConditionalOnClass(\{ SpringBus.class, CXFServlet.class \})}}, i.e. it is dependent on presence of {{cxf-core}} and {{cxf-rt-transports-http}} modules - it provides auto-configuration capabilities for {{CXFServlet}}, {{SpringBus}} and JAX-RS specific config i.e. {{SpringComponentScanServer}} - {{CXFServlet}} auto-configuration is disabled on presence of bean named {{cxfServletRegistration}} - {{SpringBus}} auto-configuration is disabled on presence of {{SpringBus}} bean - JAX-RS specific configuration is conditional on presence of {{cxf-rt-frontend-jaxrs}} module and is disabled on presence of bean named {{jaxRsServer}} The samples previously linked are working with this implementation: https://github.com/vpavic-samples/spring-boot-cxf-jax-ws https://github.com/vpavic-samples/spring-boot-cxf-jax-rs Looking forward to your feedback.
37. Hi Vedran, that was not a complexity by design, all I wanted is for users to easily switch between the two options, who knew SpringBoot would get confused, it is asked to load a given SpringBoot application, why does it load another one, seems like an obvious bug to me. All this Spring 'simplicity', it evaporates as soon as one starts practically experimenting with it, sorry for being a bit grumpy :-).
38. Hi Vedran, thanks for your patch. FYI, I kind of liked a more neutral AutoConfigure approach, and I'm not keen to introduce a WADL support OOB - some people do like WADL, but quite a few do prefer Swagger these days, we'll need to give it a bit more thinking. Let me look at your patch in more detail a bit later on, given that I'm really keen to understand why SampleRestApplication is still not picking up the auto configuration. FYI, I've removed SampleScanRestApplication - this is now part of a separate demo, jaxrs_sprint_boot_scan - I'll deal with the whole scanning thing later on. In the original jaxrs_spring_boot demo I now only have SampleRestApplication: https://github.com/apache/cxf/blob/master/distribution/src/main/release/samples/jax_rs/jaxrs_spring_boot/src/main/java/sample/rs/service/ It is not importing JaxRsConfig directly or indirectly. I even went ahead and locally only (for the sake of the experiment) removed it from the source completely and recompiled the JAX-RS frontend. Can you spot anything that can explain why it is still not picking up the auto configuration ? FYI, I'm not trying to be difficult :-), and I do appreciate a lot you trying to help Sergey
39. Or may be SampleRestApplication is picking it up, given "http://localhost:8080/services/helloservice/" is returning "No services have been found" which indicates application.properties have been loaded which is where a servlet "/services" context is redefined as "/services/helloworld". But https://github.com/apache/cxf/blob/master/distribution/src/main/release/samples/jax_rs/jaxrs_spring_boot/src/main/java/sample/rs/service/SampleRestApplication.jav is not effective, while ii was effective when SampleRestApplication was loading CXFServlet directly. I'll try to look more into it next week Thanks
40. Ok, I had to get a Bus auto-wired to make sure the one created by AutoConfigure is used. So SampleRestApplication works fine and I'm liking SpringBoot again :-). Will experiment with a demo expecting JAX-RS resources auto-loaded next week
41. **body:** Hey Sergey, Regarding the existing sample complexity - I understand what was your intention, but the bottom line is the implementation introduced hidden complexity that can be confusing, especially for users not so familiar with Spring. You know the old saying - the road to hell is paved with good intentions :) I'd like to stress that Spring Boot didn't get confused here, it just did what it's supposed to do. The key here is to understand the composition of {{@SpringBootApplication}} annotation and the fact that declaring main-class in your build only affects application packaging, i.e. what class should the executable JAR run. The right way to achieve what you were trying to do here is to use Spring {{@Profile}} support. Concerning WADL, all I did is add the {{cxf-rt-rs-service-description}} artifact to the JAX-RS starter. My motivation to add it was the service listing servlet - it generates {{?_wadl}} links by default which don't work without {{cxf-rt-rs-service-description}} artifact, which is not the greatest user experience IMO. From what I can tell, the Swagger support is also located in this module so I don't see any harm including in the starter. It's a single dependency that's very useful in JAX-RS scenario. When debugging the Spring Boot apps, maybe you've missed my previous comment on this, try starting your app with {{--debug}} switch (see http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#howto-troubleshoot-auto-configuration for more info on that). Then search for {{CxfAutoConfiguration}} in the log output. This will tell you which auto-configuration conditions were met and which were not, as well as why. I know you're trying to understand things as much as possible and it's perfectly fine, I'm here to help as much as I can :)
    label: code-design
42. Hi Vedran, thanks, well, if we talk about the users who are not familiar with SpringBoot then it was me who was such a user when I started modifying a demo originally contributed to CXF :-). And I think the principle of the least surprise was not met. Re the service descriptions - an issue exists to move a Swagger related code into a diff module. Adding a WADL filter which will check every request when people would only expect Swagger support and likewise having an extra Swagger endpoint or filter around when only WADL is needed will not be ideal, but I do agree the service description should be coming OOB for RS services too, I'd only like to have a dedicated discussion/solution done later on. Cheers, Sergey
43. Hi Vedran I've started applying some parts of your patch. First I applied your JaxRsConfig changes as part of another issue: http://git-wip-us.apache.org/repos/asf/cxf/commit/8e58f887 I did it because it might be a migration issue for some users and as such I thought about opening a dedicated issue. Next I updated the version: http://git-wip-us.apache.org/repos/asf/cxf/commit/4b0868a8 Finally, the auto-configuration improvements: http://git-wip-us.apache.org/repos/asf/cxf/commit/3615f209 but I haven't applied JAX-RS specific changes. Specifically, SpringScanComponentServer is imported right now into a demo application in jaxrs_spring_boot_scan, and with your JaxRsConfig fixes it all works nice. I know you like the idea of JAX-RS resources being auto-discovered OOB :-) but I have few concerns, specifically, having the auto-discovery enabled by default OOB may cause unexpected side-effects, but also, I wonder, if we really should do the frontend specific configurations in a shared autoconfigure module. For example, I opened a JAX-WS JIRA awhile back for JAX-WS endpoints be auto-created too, so once it is done we'd need to add a JAX-WS specific support, and then, with the frontend specific features likely to be needed provided OOB as well going forward (ex, Swagger vs WADL for RS) we may find it difficult to keep a clean enough code. Explicitly importing SpringScanComponentServer into a demo application is an extra effort compared to it being seamlessly enabled by the virtue of adding a JAX-RS starter dependency, but for now I'm feeling it is a reasonable compromise. I'd like to discuss some options, such as introducing frontend specific auto-configure modules. Or revisiting the idea of having only two modules, two starters, one for JAX-WS and one for JAX-RS, with starters keeping some auto-configuration code ? I appreciate the best practice is to keep the starters code-free, but having a single auto-configure module may become problematic for us going forward ? Overall, I feel we already have a nice start as far as integrating with Spring Boot is concerned and we can keep tuning it :-) Thanks, Sergey
44. Hi Sergey, Thanks for your feedback and merging the latest set of improvements! OK on JAX-RS service descriptions, we can revisit that once your changes in that department unfold. {quote} I wonder, if we really should do the frontend specific configurations in a shared autoconfigure module. {quote} {quote} I'd like to discuss some options, such as introducing frontend specific auto-configure modules. Or revisiting the idea of having only two modules, two starters, one for JAX-WS and one for JAX-RS, with starters keeping some auto-configuration code ? I appreciate the best practice is to keep the starters code-free, but having a single auto-configure module may become problematic for us going forward ? {quote} The whole point is to have a single auto-configuration module which holds all the configuration code, and behaves differently depending on what's found on your classpath (hence the {{@Conditional*}} annotations) and depending on your configuration properties. For example, take a look at Spring Boot's own autoconfigure module: https://github.com/spring-projects/spring-boot/tree/master/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure I don't see the point of splitting the code between the auto-configure and starter modules, since it would IMO defeat the whole purpose of this approach and could potentially confuse Spring Boot users since it would not be consistent approach

with Boot itself. Regarding the {{SpringScanComponentServer}} and JAX-RS specific configuration, I don't know if you noticed this part form one of my previous comments: {quote} JAX-RS specific configuration is conditional on presence of cxf-rt-frontend-jaxrs module and is disabled on presence of bean named jaxRsServer {quote} You need a server bean to register JAX-RS manually anyway, right? Simply name it {{jaxRsServer}} and the auto-configuration of that part will back off, i.e. no JAX-RS resources will be discovered and registered automatically. Alternatively, we could have this conditional on a configuration property, for example {{cxf.jax-rs.component-scan}}. If you prefer not to discover and register JAX-RS resources automatically, the default for that property would be {{false}} and user which like that approach would just have to add {{cxf.jax-rs.component-scan=true}} to their configuration properties. Anyway, I agree that this is a nice start and I'm looking forward to the next release of CXF :) BTW what is the targeted release and do you have an approximate ETA? Thanks, Vedran

45. **body:** Sidenote - I originally named starter modules {{cxf-spring-boot-starter-jax-rs}} and {{cxf-spring-boot-starter-jax-ws}}, but looking at the names of other CXF modules (namely {{cxf-rt-frontend-jaxrs}} and {{cxf-rt-frontend-jaxws}}) I wonder starter should be named {{cxf-spring-boot-starter-jaxrs}} and {{cxf-spring-boot-starter-jaxws}} for consistency?
    label: code-design
46. **body:** Hi Vedran, Good to hear from you, sure, on one hand we have a Spring Boot blueprint recommending a certain module structure. On the other hand we have CXF specifics, mainly the fact that some people do only JAX-WS, some - only JAX-RS, and I feel that taking care of these two different styles in a single auto-configure will sooner or later get us into the corner. The last thing which concerns me is whether our Spring Boot module structure will be deemed to be meeting precisely that blueprint or not. As far as I'm concerned the main criteria is to make CXF work well with SpringBoot. If putting the code into a starter is a really bad style in Spring Boot world then perhaps we can consider introducing front-end specific auto-configure support. We can continue operating with a single one for as long as it proves realistically possible of course, adding one more auto-configure module is not something I'm keen to do but IMHO we should not exclude this option should it prove necessary. Let me try to move our chat into a new direction for a sec :-). So we have CXF endpoints showing up in a Spring Boot container. Now consider this project, https://github.com/spring-cloud-samples/eureka/blob/master/pom.xml Do you reckon it is a matter of adding few dependencies as in that project for CXF endpoints being seen in Spring Cloud registry ? Thanks, Sergey
    label: code-design
47. **body:** Starters have been renamed: http://git-wip-us.apache.org/repos/asf/cxf/commit/4b76cb4d I guess we will need few more weeks to address various CXF 3.1.6 issues, 3.2.0 release is not planned in the short term AFAIK, hopefully we can get initial JAX-RS 2.1. into it or at least will do some more 3.2.0 specific work before it gets released.
    label: code-design
48. Hi Sergey, Sorry for the delay - I don't see any reason to stray away from the approach we have initially taken and what is the recommended way of doing Spring Boot integration. I've deliberately linked the contents of Boot's own auto-configuration module few comments back to demonstrate that there are many different technologies in there, all of which are optional to use/activate. {quote} Alternatively, we could have this conditional on a configuration property, for example {{cxf.jax-rs.component-scan}}. If you prefer not to discover and register JAX-RS resources automatically, the default for that property would be {{false}} and user which like that approach would just have to add {{cxf.jax-rs.component-scan=true}} to their configuration properties. {quote} Do you have any feedback/preference on this? Regarding the Spring Cloud integration, I'm not that much into that project. Maybe we could revisit that after Boot integration is finalized. Thanks for renaming the starter modules and insight into release schedule!
49. Hi Vedran Np, thanks for the comments. Having a property seems marginally more preferable. In the current patch it is conditional on the application not initiating a "jaxRsServer" bean, so if we have a manual setup with multiple CXF endpoints then the users would be somewhat restricted to having at least a single "jaxRsServer" bean as opposed to say "myApplicationEndpoint", etc. Probably a minor issue. But having a property also ensures there will be no unexpected auto-discovery going under the hood. This property can be enabled by default over time if it is what people would prefer in their feedbacks. If we have a property approach, I believe the only thing which would change as far as your last patch is concerned, is that instead we'd have Conditional checking the property is set, right ? Re the recommended approach, yes, as I said I'm fine with the current module structure, and adhering to the blueprint is good, but I'd like to re-iterate that the CXF specifics may need to be taken into the consideration, from the CXF point of view pushing JAX-WS and JAX-RS specifics into a single auto-configure module will not be a clean solution. But for now the only specifics we have is a JAX-RS auto-scan feature so I guess we can indeed enable it via an optional property. What I did not quite get, I tried a jaxws demo with your patch which enables the jaxrs auto discovery and it worked fine without me adding a JAX-RS dep, how does it work if the auto-configure feature has Conditional on JAXRSServerFactoryBean.class ? Yeah, lets chat about Spring cloud later on. I have an action item to check if the fact we have a CXF endpoint up in SpringBoot makes it visible in other specific Spring servers, such as the registry. I'll give it a try later on... Thanks
50. Hey Sergey, {quote} If we have a property approach, I believe the only thing which would change as far as your last patch is concerned, is that instead we'd have Conditional checking the property is set, right ? {quote} Correct, instead of {{@ConditionalOnMissingBean(name = "jaxRsServer")}} you'd put {{@ConditionalOnProperty(prefix = "cxf", name = "jax-rs.component-scan", havingValue = "true")}}. {quote} What I did not quite get, I tried a jaxws demo with your patch which enables the jaxrs auto discovery and it worked fine without me adding a JAX-RS dep, how does it work if the auto-configure feature has Conditional on JAXRSServerFactoryBean.class ? {quote} The JAX-WS demo? Or did you meant to say JAX-RS demo? In any case can you point me to the concrete project we're talking about here? If it's the JAX-RS demo, the dependency comes via {{cxf-spring-boot-starter-jaxrs}}. Otherwise something else must be (transitively) pulling the JAX-RS components.
51. Hi Vedran I did refer to a JAX-WS demo, I updated the one shipped with CXF to use a new starter: https://fisheye6.atlassian.com/changelog/cxf?cs=4837a6cb5ea319f6bf87aaf3bf49c1e4e99e45a7 and it worked fine, and then I wanted to check that having JAX-RS class references in a common auto-configure won't require JAX-WS users add a JAX-RS dep, so I updated the code locally as per your patch, and the demo still worked fine without JAX-RS deps (I can't see in the dependency tree), I've just double checked it again. My guess SpringBoot is ignoring this JAX-RS specific Configuration by catching NoClassDefFoundError. Can you please try on your end too, run your JAX-WS demo with the JAX-RS related patch fragment applied ? Thanks, Sergey
52. Aha, I misunderstood your problem. Yes, that of course works. My samples are tested against the code I submitted in the last PR. Refer to Javadoc of following classes for more details: https://github.com/spring-projects/spring-boot/blob/master/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/condition/ConditionalOnClass.java https://github.com/spring-projects/spring-boot/blob/master/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/condition/OnClassCondition.java Note that inner class annotated with {{@ConditionalOnClass(JAXRSServerFactoryBean.class)}} is actually a {{@Configuration}} class also, meaning it won't get loaded if all its conditions aren't met. Referencing classes in auto-configuration does not mean those classes need to be present on the classpath, that's the power of Spring Boot and the rationale behind having optional dependencies for auto-configuration module. I've tried to stress this earlier by linking to contents of Boot's own auto-configuration module but apparently I failed at that :) https://github.com/spring-projects/spring-boot/tree/master/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure
53. Sure, see http://git-wip-us.apache.org/repos/asf/cxf/commit/32a37d02 (I only removed one dash in a property name, to have 'jaxrs.component-scan', hope you are OK with it). I do like the way the integration is shaping. The auto-scan based enabling of JAX-RS endpoints with a single property touch is impressive. I admit I haven't spent much time with SpringBoot docs but I feel like I've read them nonetheless thanks to your helpful comments on this JIRA issue :-). Thanks for working with us so far, it has been appreciated. Do you think we can resolve this issue ? I'm sure few more things will need to be optimized , things like seeing CXF endpoints showing up in a Spring registry without us having to write a massive amount of the complex integration code, may be detecting Swagger annotations and enabling a Swagger feature automatically, etc, but we can have new JIRAs allocated as needed. Cheers, Sergey
54. Everything looks good to me Sergey, for the initial support I think we managed to squeeze in even more than it was expected. Having used CXF successfully in multiple projects over the years, I'm very happy to have contributed and worked with you :) IMO you can resolve this. I'm looking forward to the 3.1.7 release! Feel free to ping me in case of any issues and further improvements on Spring Boot integration, I'll be glad to take a look and contribute again.
55. **body:** Hi Vedran, nice, thanks for your help. Just to understand, CXF is not going to be listed at https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters ? If not can it be documented somewhere in Spring Boot that CXF starters are available directly in CXF ? Vedran, and Stephane, thanks :-)
    label: documentation
56. Going to resolve it now, but please comment here nonetheless
57. **body:** Hi Sergey, 3rd party starters are listed here: https://github.com/spring-projects/spring-boot/blob/master/spring-boot-starters/README.adoc I'll submit the PR to update this list with CXF support once 3.1.7 is released, it's on my TODO list already :)
    label: requirement
58. **body:** Hi Vedran, brilliant, I thought for it to be listed there the modules need to be located directly in Spring Boot, but referencing it in README.adoc will be OK for now. I guess I'll need to prepare some CXF wiki documentation, will update here once it is ready thanks
    label: documentation
59. Hi Vedran I've added the initial documentation here: http://cxf.apache.org/docs/springboot.html Just added the links to the demos too, will be synced shortly. I guess it will be quite easy to create a similar scanning support for JAX-WS, more or less a copy and paste of a JAX-RS SpringComponentScanServer except that the component scan will check JAXWS specific annotations, and ship it in a 'spring' subpackage of the JAXWS frontend. We can def enhance a JAX-WS auto-configure this way. FYI, I'm off till 20th June Thanks, Sergey

60. Github user vpavic closed the pull request at: https://github.com/apache/cxf/pull/139
61. Hi Vedran I'm planning to update a "cxf.path" property to "cxf.servlet-path" property. It would be more in line with Spring MVC's server.servlet-path, are you OK with it ? Thanks, Sergey
62. Or may be "cxf.servlet.path" so that a 'path' can be aligned in yaml with (cxf.servlet.) "init"
63. **body:** Hi Sergey! I've originally modeled the names of configuration properties to be consistent with [Spring Boot properties|http://docs.spring.io/spring-boot/docs/1.4.0.M3/reference/htmlsingle/#common-application-properties] that register additional servlets, namely with: {code} spring.webservices.path=/services # Path that serves as the base URI for the services. spring.h2.console.path=/h2-console # Path at which the console will be available. {code} You are of course free to adjust these to match your preferences, if the originally chosen names aren't descriptive enough. Among your suggested alternatives, {{cxf.servlet-path}} makes more sense to me.
    label: code-design
64. Hi Vedran Right, I see, I suppose we can keep cxf.path as is. The reason I kind of liked cxf.servlet.path is because it would look neat/well-structured in application.yml, given that we use cxf.servlet.init to pass the init parameters, with both "cxf.servlet.path" and "cxf.servlet.init" related to configuring "CXFServlet": {noformat} cxf: servlet: path: /custom init: services-list-path: /serviceslist {noformat} FYI, I've updated one of the demos to use YAML: https://github.com/apache/cxf/blob/master/distribution/src/main/release/samples/jax_rs/spring_boot_scan/application/src/main/resources/application.yml I'm OK with keeping cxf.path as is but if you happen to warm up a bit to a "cxf.servlet.path" idea, then let me know Thanks, Sergey
65. Hi Vedran, FYI, I've added an option to scan JAX-RS resources without them having to be marked with @Component or returned from @Bean methods: http://git-wip-us.apache.org/repos/asf/cxf/commit/7cd6ca06 It can help with auto-discovering 3rd party providers like Jackson which won't be marked as Components or when modifying the own resources with Component is not really needed. I've added "jaxrs.classes-scan" property. It is mutually exclusive with "jaxrs.component-scan" Thanks
66. **body:** Hi Sergey, {quote} I'm OK with keeping cxf.path as is but if you happen to warm up a bit to a "cxf.servlet.path" idea, then let me know {quote} I'm personally in favor of {{cxf.path}} over {{cxf.servlet.path}}. IMO it's a good thing to be consistent with similar configuration properties in the Spring Boot itself. Regarding the argument for modeling configuration properties this way, consider the {{web.xml}} itself - {{init-param}} and {{load-on-startup}} are attributes of the Servlet whereas the {{servlet-mapping}} is completely external to the Servlet. Good to see you've added some support for more JAX-RS use cases. With the latest change in mind I've got one suggestion for you to make the configuration more concise - you could merge {{jaxrs.classes-scan}} and {{jaxrs.component-scan}} properties into a single {{jaxrs.scan}} which is an enum consisting of {{NONE}} (default), {{CLASSES}} and {{COMPONENTS}}. IMO it makes sense since the options are mutually exclusive.
    label: code-design
67. Hi Vedran I'm not sure "jaxrs.scan = CLASSES" is a better try than "jaxrs.classes-scan = true" Besides, FYI, I'm thinking of relaxing it a bit and enhancing a component-only scan server to optionally check for the classes too (ex to combine the explicit components and 3rd party JAX-RS Providers not marked as Components) Thanks
68. **body:** Hi Sergey, OK, so that actually makes them not truly mutually exclusive, right? In that case enum suggestion is of course not suitable. Also could you give me an example of what qualifies as ??3rd party JAX-RS Provider??? I'd like to get some insight on that use case but I don't have too much CXF JAX-RS experience. Thanks, Vedran
    label: code-design
69. Hi Vedran, jaxrs.component-scan and jaxrs.classes-scan are still exclusive as either of them will initiate the creation of a Server bean, but jaxrs.component-scan is optionally combined with jaxrs.classes-scan-packages in cases non-Component providers need to be added. For example, http://fasterxml.github.io/jackson-jaxrs-providers/javadoc/2.2.0/com/fasterxml/jackson/jaxrs/xml/JacksonXMLProvider.html another example is Swagger2Feature in CXF which is not marked as Component but is annotated with a CXF specific Provider annotation: https://github.com/apache/cxf/blob/master/rt/rs/description-swagger/src/main/java/org/apache/cxf/jaxrs/swagger/Swagger2Feature.java#L57 Thanks, Sergey
70. Hi Sergey, I've noticed yesterday that 3.1.7 was release two weeks ago (hint: neither [CXF homepage|https://cxf.apache.org/] nor [downloads page|https://cxf.apache.org/download.html] have a reference to it yet :)), so I wanted to let you know CXF is now listed among Spring Boot's [3rd party starters|https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters#community-contributions].
71. Hi Vedran, many thanks, I'm going to blog shortly about our joint effort. Yes, CXF pages will be updated in the next couple of days Cheers, Sergey
72. **body:** Hi Vedran The link to the CXF SpringBoot documentation at https://github.com/spring-projects/spring-boot/blob/master/spring-boot-starters/README.adoc is wrong it should be "http://cxf.apache.org/docs/springboot.html" Can you please look into it ? I can easily create a PR myself too Thanks, Sergey
    label: documentation
73. Hi Sergey, The link is actually OK - first column in the table links to the Apache CXF page while second column links to the Github repo.
74. LOL, silly me, I'm glad I did not create a patch request to fix it :-)
75. **body:** No problem. You got me confused for a moment too :)
    label: code-design
76. Hi Vedran, FYI we've updated CXF 3.2.0-SNAPSHOT CxfAutoConfiguration to org.springframework.boot.web.servlet.ServletRegistrationBean which is not available in SpringBoot 1.3.x to make sure CXF starters can be run with SpringBoot 1.5. CXF 3.1.x users would like to run the starters with SpringBoot 1.5 and I think it is reasonable to assume, given that SpringBoot 1.3.x, 1.4.x and 1.5.x now exist that most users are now with 1.4.x. Would it affect your CXF code if we removed the support for SpringBoot 1.3.x in CXF 3.1.11-SNAPSHOT and if yes then would you be OK to migrating to SpringBoot 1.4.x once CXF 3.1.11 is out ? Thanks, Sergey
77. Hi Sergey, Yes, your assumption in right, most Spring Boot users should be on 1.4.x with focus shifting to recently released 1.5.x. Regarding the upgrade of CXF 3.1.x to Boot 1.4.x - I'm all for it :)
78. Hi Vedran, sounds good, I'll then push the changes to 3.1.x to align it with 1.4.x/1.5.x thanks, Sergey