Item 309
git_comments:

1. TODO: we may want a more minimal example here
2. !/bin/bash A script that creates a core by copying config before starting solr. To use this, map this file into your container's docker-entrypoint-initdb.d directory: docker run -d -P -v $PWD/precreate-collection.sh:/docker-entrypoint-initdb.d/precreate-collection.sh solr
3. !/bin/bash This script is mainly an illustration for the docker-entrypoint-initdb.d extension mechanism. Run it with e.g.: docker run -d -P -v $PWD/docs/set-heap.sh:/docker-entrypoint-initdb.d/set-heap.sh solr The SOLR_HEAP configuration technique here is usable for older versions of Solr. From Solr 6.3 setting the SOLR_HEAP can be done more easily with: docker run -d -P -e SOLR_HEAP=800m apache/solr
4. execute command passed in as arguments. The Dockerfile has specified the PATH to include /opt/solr/bin (for Solr) and /opt/docker-solr/scripts (for our scripts like solr-foreground, solr-create, solr-precreate, solr-demo). Note: if you specify "solr", you'll typically want to add -f to run it in the foreground.
5. !/bin/bash docker-entrypoint for Solr docker
6. when invoked with e.g.: docker run solr -help
7. Clear some variables that we don't want runtime
8. !/bin/bash A helper script to initialise an empty $DIR If you use volumes then Docker will copy the $DIR content from the container to the volume. If you use bind mounts, that does not happen, so we do it here.
9. if we're running with `--init` or under tini or similar, follow the upstream behaviour
10. if that hasn't worked, send SIGKILL
11. !/bin/bash
12. Custom oom handler loosely based on https://github.com/apache/lucene-solr/blob/master/solr/bin/oom_solr.sh See solr-forgeground for how to configure OOM behaviour
13. under Docker, when running as pid 1, a SIGKILL is ignored, so use the default SIGTERM
14. !/bin/bash Create a core on disk arguments are: corename configdir
15. !/bin/bash Run the init-solr-home script and source any '.sh' scripts in /docker-entrypoint-initdb.d. This script is sourced by some of the solr-* commands, so that you can run eg: mkdir initdb; echo "echo hi" > initdb/hi.sh docker run -v $PWD/initdb:/docker-entrypoint-initdb.d solr and have your script execute before Solr starts. Note: scripts can modify the environment, which will affect subsequent scripts and ultimately Solr. That allows you to set environment variables from your scripts (though you usually just use "docker run -e"). If this is undesirable in your use-case, have your scripts execute a sub-shell.
16. execute files in /docker-entrypoint-initdb.d before starting solr
17. init script for handling an empty /var/solr
18. solr uses "-c corename". Parse the arguments to determine the core name.
19. See https://github.com/docker-solr/docker-solr/issues/27
20. !/bin/bash This script starts Solr on localhost, creates a core with "solr create", stops Solr, and then starts Solr as normal. Any arguments are passed to the "solr create". To simply create a core: docker run -P -d solr solr-create -c mycore To create a core from mounted config: docker run -P -d -v $PWD/myconfig:/myconfig solr solr-create -c mycore -d /myconfig
21. check the core_dir exists; otherwise the detecting above will fail after stop/start
22. !/bin/bash Configure a Solr demo and then run solr in the foreground
23. check the core_dir exists; otherwise the detecting above will fail after stop/start
24. Presumably we're already running under tini through 'docker --init', in which case we don't need to run it twice. It's also possible that we're run from a wrapper script without exec, in which case running tini would not be ideal either.
25. determine TINI default. If it is already set, assume the user knows what they want
26. Allow easy setting of the OOM behaviour Test with: docker run -p 8983:8983 -it -e OOM=script -e SOLR_JAVA_MEM="-Xms25m -Xmx25m" solr
27. recommended
28. !/bin/bash start solr in the foreground
29. Default to running tini, so we can run with an OOM script and have 'kill -9' work
30. init script for handling an empty /var/solr
31. !/bin/bash Run the initdb, then start solr in the foreground
32. !/bin/bash Create a core on disk and then run solr in the foreground arguments are: corename configdir To simply create a core: docker run -P -d solr solr-precreate mycore To create a core from mounted config: docker run -P -d -v $PWD/myconfig:/myconfig solr solr-precreate mycore /myconfig To create a core in a mounted directory: mkdir myvarsolr; chown 8983:8983 myvarsolr docker run -it --rm -P -v $PWD/myvarsolr://var/solr solr solr-precreate mycore
33. init script for handling an empty /var/solr
34. !/bin/bash configure Solr to run on the local interface, and start it running in the background
35. !/bin/bash stop the background Solr, and restore the normal configuration
36. > 0)); do
37. !/bin/bash A helper script to wait for solr Usage: wait-for-solr.sh [--max-attempts count] [--wait-seconds seconds] [--solr-url url] Deprecated usage: wait-for-solr.sh [max_attempts [wait_seconds]]
38. deprecated invocation, kept for backwards compatibility
39. split on commas, for when a ZK_HOST string like zoo1:2181,zoo2:2181 is used
40. > 0)); do
41. looks like an IPv6 address, eg [2001:DB8::1] or [2001:DB8::1]:2181 getent does not support the bracket notation, but does support IPv6 addresses
42. IPv4, just split on :
43. To test the parsing: bash scripts/wait-for-zookeeper.sh foo bash scripts/wait-for-zookeeper.sh 'ZK_HOST=[2001:DB8::1]:2181,[2001:DB8::1],127.0.0.1:2181,127.0.0.2' ZK_HOST=[2001:DB8::1]:2181,[2001:DB8::1],127.0.0.1:2181,127.0.0.2 bash scripts/wait-for-zookeeper.sh
44. consume for shellcheck
45. !/bin/bash A helper script to wait for ZooKeeper This script waits for a ZooKeeper master to appear. It repeatedly looks up the name passed as argument in the DNS using getent, and then connects to the ZooKeeper admin port and uses the 'srvr' command to obtain the server's status. You can use this in a Kubernetes init container to delay Solr pods starting until the ZooKeeper service has settled down. Or you could explicitly run this in the Solr container before exec'ing Solr. Inspired by https://github.com/helm/charts/blob/9eba7b1c80990233a68dce48f4a8fe0baf9b7fa5/incubator/solr/templates/statefulset.yaml#L60 Usage:

wait-for-zookeeper.sh [--max-attempts count] [--wait-seconds seconds] zookeeper-service-name If no argument is provided, but a Solr-style ZK_HOST is set, that will be used. If neither is provided, the default name is 'solr-zookeeper-headless', to match the helm chart.
46. !/bin/bash
47. !/bin/bash
48. !/bin/bash
49. !/bin/bash Simulate openshift by running with a random uid
50. !/bin/bash
51. !/bin/bash
52. remove the solr-owned files from inside a container
53. !/bin/bash
54. remove the solr-owned files from inside a container
55. !/bin/bash
56. !/bin/bash
57. !/bin/bash
58. when we mount onto /var/solr, that will be owned by "solr"
59. with nocopy, the /var/solr ends up owned by root, so we need to chown it first
60. !/bin/bash
61. !/bin/bash
62. when we mount onto /var/solr, that will be owned by "solr"
63. !/bin/bash
64. when we mount onto /var/solr, it will be owned by "solr", and it will copy the solr-owned directories and files from the container filesystem onto the the container. So from a container running as solr, modify permissions with setfacl to allow our user to write. If you don't have setfacl then run as root and do: chown -R $(id -u):$(id -g) /var/solr
65. check core is created by solr
66. chown it back
67. check test file was created by root
68. !/bin/bash A simple test of gosu. We create a myvarsolr, and chown it
69. TODO: Fix this test on Mac
70. !/bin/bash
71. !/bin/bash
72. !/bin/bash
73. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to You under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
74. !/bin/bash
75. create a core by hand:
76. !/bin/bash
77. create a directory for the core
78. !/bin/bash
79. check that the version of Solr matches the tag
80. The /var/solr mountpoint is owned by solr, so when we bind mount there our directory, owned by the current user in the host, will show as owned by solr, and our attempts to write to it as the user will fail. To deal with that, set the ACL to allow that. If you can't use setfacl (eg on macOS), you'll have to chown the directory to 8983, or apply world write permissions.
81. echo "***** Created varsolr folder $BUILD_DIR / $folder"
82. Create build directory if it hasn't been provided already
83. Shared setup
84. !/bin/bash Shared functions for testing
85. == 0)); then
86. echo "Got status from Solr: $status"

git_commits:

1. **summary:** SOLR-14789: Absorb the docker-solr repo. (#1769)
   message: SOLR-14789: Absorb the docker-solr repo. (#1769)

github_issues:

github_issues_comments:

github_pulls:

1. **title:** SOLR-14789: Absorb the docker-solr repo.
   body: WIP - https://issues.apache.org/jira/browse/SOLR-14789 **This is intended for 9.0 only, and will not be backported to 8.x.** We can continue using the docker-solr repo for all 8.x.y releases. ### Goals The goal of this PR is to move all functionality for how current solr images are built to lucene solr. However there are clearly a lot of other things we want to accomplish too: - Migrate functionality from the `solr/docker/include/scripts` into `solr/bin` or Solr itself - Support the "official" docker releases (https://hub.docker.com/_/solr not just https://hub.docker.com/apache/solr) - Make the solr docker file more "cloud native" Since this docker image will not be officially used until 9.0 at the earliest, this gives us time to iterate and accomplish all of these goals. However merging in this initial functionality that works like the current docker-solr setup first lets us more easily make and test these changes. ### Setup of `/solr/docker` `gradle assemble` will now create a docker image that has been created with contents of `:solr:packaging`. The "Solr image" is broken up into 2 docker files. 1. The `package` dockerfile, which contains the solr release under `/opt/solr-build/solr-${VERSION}`. This image requires no other content at the end. 2. The `runtime` dockerfile, which is given the `package` image name as an input and generates everything currently setup in the Solr docker image, copying the package contents from the `package` image. The `runtime` docker image is the one

that will be pushed to Dockerhub eventually, and is run by users. The idea is that there is also a task to generate the release docker image (which can be found under `/solr/docker/package/Dockerfile.release-package`) that is then passed to the runtime docker image. But this is a future goal and not necessary in order to merge this PR IMO. I basically learned gradle to write this, so please give any advice on how I could make the build process better. I'm sure it is not optimal. ### Testing Once you run `gradle assemble`, you can test the new docker image in two ways: - `gradle :solr:docker:test`, which will run the docker-solr integration tests using the newly assembled solr image. This will eventually fail when it reaches the test that needs root permissions however, since gradle doesn't have root permissions itself. This is a TODO. - `docker run -d -p 8983:8983 -e SOLR_JETTY_HOST="0.0.0.0" apache/solr:9.0.0-SNAPSHOT`, then visit the admin screen at http://localhost:8983 . TODO: - [ ] Add ability to build with "release" package - [ ] Remove ability to push the `package` image. - [ ] Integrate `gradle :solr:docker:tests` into the overall `gradle tests` - [x] Make all of the tests work with gradle.

github_pulls_comments:

1. I should point out that the `tests`, `docs` and `runtime/include/scripts` are almost identical copies from the docker-solr repo. Some small changes have been made to the `scripts` files, but no actual runtime logic has changed. I'd be happy to copy exactly from the repo and make the small changes in a different PR, if that would make y'all more comfortable.
2. > I'd be happy to copy exactly from the repo and make the small changes in a different PR, if that would make y'all more comfortable. Doesn't have to be a different PR but would be very helpful to have the unaltered copy as one commit, and then all of your changes as separate commits on top of that.
3. Also, I imagine @chatman is interested in this.
4. > > I'd be happy to copy exactly from the repo and make the small changes in a different PR, if that would make y'all more comfortable. > > Doesn't have to be a different PR but would be very helpful to have the unaltered copy as one commit, and then all of your changes as separate commits on top of that. I'll try to make this happen.
5. @madrob The first commit has the Dockerfile, tests, scripts and docs copied over from the docker-solr repo. The rest of the commits are small changes to the scripts, splitting the dockerfile logic into two parts, and adding the gradle logic.
6. Looking great @HoustonPutman ! I ran `g assemble` and then launched the Docker image using a compose file: ``` version: '3.2' services: zookeeper: image: zookeeper:3.5.8 ports: - "2181:2181" solr: image: apache/solr:9.0.0-SNAPSHOT depends_on: - zookeeper ports: - "8983:8983" environment: SOLR_JETTY_HOST: "0.0.0.0" command: ["solr-foreground", "-z", "zookeeper:2181", "-c"] ``` Note the need to set SOLR_JETTY_HOST to bind to all interfaces in the container if you want to expose Solr to your local workstation. Still kicking the tires but looking great so far.
7. > Note the need to set SOLR_JETTY_HOST to bind to all interfaces in the container if you want to expose Solr to your local workstation I previously found the need to set `SOLR_HOST=localhost` with docker containers. Is one way preferred over the other? What's the actual difference?
8. > > Note the need to set SOLR_JETTY_HOST to bind to all interfaces in the container if you want to expose Solr to your local workstation > > I previously found the need to set `SOLR_HOST=localhost` with docker containers. Is one way preferred over the other? What's the actual difference? @madrob This commit here (https://github.com/apache/lucene-solr/commit/5377742a62e58c79055f3a2676b77e1ed1d61823#diff-2e431666cd6c6f1e08f79cdefa4988a4) makes it so Jetty only binds to 127.0.0.1 in the Docker container, which messes up Docker's ability to expose port 8983 outside of the container. It looks like that was done for security reasons ... So setting SOLR_JETTY_HOST to 0.0.0.0 allows me to reach Solr from outside Docker using http://locahost:8983
9. I was able to build this and did some very simple validation after launching containers using: ``` docker run --rm -d --name zk-for-solr -p 2181:2181 zookeeper docker run --rm -p 8983:8983 --link zk-for-solr:zk-for-solr -e SOLR_JETTY_HOST="0.0.0.0" apache/solr:9.0.0-SNAPSHOT -c -z zk-for-solr:2181 docker run --rm -p 8984:8984 --link zk-for-solr:zk-for-solr -e SOLR_JETTY_HOST="0.0.0.0" apache/solr:9.0.0-SNAPSHOT -c -z zk-for-solr:2181 -p 8984 ``` Can we make sure that user run instructions are included in the ref guide, or possibly the tutorial?
10. While SOLR_JETTY_HOST is a nice security feature for someone running Solr outside of Docker, isn't it pointless (needless pain) for anyone using Docker? I think so, thus the docker settings could set this so that you don't have to.
11. My first attempt at this gave some bash errors: ``` ./gradlew assemble > Configure project :solr:docker readlink: illegal option -- f usage: readlink [-n] [file ...] /Users/janhoy/git/lucene-solr/solr/docker/tests/cases/create_core_exec/test.sh: line 18: ./../../shared.sh: No such file or directory FAILURE: Build failed with an exception. ``` I recognize this from the docker-solr repo, the test scripts use some commands that do not work on MacOS. So I added the workaround with putting gnu variants of these tools in my path, but then I could still not make the assemble task run: ``` > Configure project :solr:docker Test /Users/janhoy/git/lucene-solr/solr/docker/tests/cases/create_core_exec apache/solr:9.0.0-SNAPSHOT Cleaning up left-over containers from previous runs Running test_apache_solr_9.0.0_SNAPSHOT Unable to find image 'apache/solr:9.0.0-SNAPSHOT' locally docker: Error response from daemon: pull access denied for apache/solr, repository does not exist or may require 'docker login': denied: requested access to the resource is denied. See 'docker run --help'. FAILURE: Build failed with an exception. ``` I had to uncomment the `task test()` from `docker/build.gradle` and then my build succeeded.
12. @janhoy The tests should hopefully work for you now, though I do have the gnu utils installed, so I'm not sure they will. The tests are no longer run by default in the assemble, so that should work for you now regardless. I have done some modification on the tests to simplify out some of the logic. I also changed the permissions of folders created during the tests, so that it doesn't require root permissions anymore to run them. This works for me, but it might not work for others. Not confident on that yet.
13. Ok, will give it another try soon. Btw - should we set `SOLR_JETTY_HOST` as an `ENV` or `ARG` or something?
14. Its currently set as an ENV, the last line of the large ENV section.
15. Thanks for reviewing the docs as well. I plan on migrating a lot of the docs to the ref guide, but I think we should probably leave a bulk of that work to a separate PR. The earlier we get this PR merged, the easier it will be to break up the work we want to do into separate JIRAs that can be attacked independently. (more extensive documentation, reduction of the scripts, make it solr more "cloud native", a plan for release images, etc)
16. CC @dweiss you might want to briefly review this new module for the gradle aspect
17. FYI related conversation on the dev list about higher level integration tests: https://lists.apache.org/thread.html/r27d742357f9e1342cce1c9aaa215a10d94f64b43b2e681f19fa8d150%40%3Cdev.lucene.apache.org%3E
18. Overall looks good, just a pair of minor questions/comments.
19. Thanks everyone for the help! Now we should be able to iterate more easily on the next tasks 😄
20. I'm looking at the docker image this produces closely, and I see it's more than a bit bloated thanks to this line: https://github.com/apache/lucene-solr/blob/3ae0ca23d937bef2865689748ac9e556b40aff38/solr/docker/Dockerfile#L45 COPY --from=solr_package "/opt/solr-$SOLR_VERSION.tgz" "/opt/solr-$SOLR_VERSION.tgz" Which adds an entire layer of waste that is all of Solr 😱 I'm also a confused why we need an entire gradle module `:solr:docker:package` just for providing input to the Dockerfile. I

read the PR opening description which sort of says why it exists... but I think that could be provided all in one module using different tasks for the different steps. I'm also not sure we need to support Dockerfile.release-package does since the image production process is merged with the project; it needn't download a .tgz from anywhere. Right? I don't think direct production of the image prohibits the goals stated above about producing an official image.

21. The extra step exists because there was no consensus around how to do official release images. If we want to decide that the official image should be built the same way as it currently is in the project (via the local build), then we can get rid of the sub-module and the extra step. However if we want to have the official image use the official release binaries, as it does in `docker-solr`, then we will need to keep the submodule. I would have preferred to have all of this done in one module, but the gradle docker plugin only supports building one image per-module. So if we want to build multiple images (which is necessary for supporting the two image types, local and release), we need two modules. I am all for not adding support for official binary release strategy, and consolidating into one docker file. I just don't want to make that decision unanimously.
22. I'm glad your are amenable to changes, and that the complexity & Docker image weight I see will melt away if we only produce an image from the Solr assembly. That is identical to the "official release" except packaging -- plain dir vs tgz of the same. I can appreciate there were unknowns causing you to add this extra baggage because it might be useful but I prefer to follow a KISS/YAGNI philosophy so that we don't pay complexity debt on something not yet needed.
23. +1 to lighter weight. However, our users should somehow be able to verify that a Docker image pulled from Docker Hub (or downloaded from elsewhere) is indeed the officially voted-upon binaries that they find in the release repo. Downloads from mirrors are easy to verify as we provide `.sha512` and `.asc` files for them. Likewise [artifacts from maven](https://repo1.maven.org/maven2/org/apache/solr/solr-core/8.7.0/) also have `.asc` and `.sha1` files for every jar. Current docker-solr Dockerfile can be inspected in that it downloads the official tarball and validates GPG signature. The lightweight Dockerfile performs no such checks and cannot be validated the same way. So here is my proposal. We build the docker image from folder instead of tgz, but also add documentation to our [download page](https://lucene.apache.org/solr/downloads.html) on how to verify the solr binaries inside the image. Could even script it: curl -o verify-docker.sh https://lucene.apache.org/solr/verify-docker.sh docker run --rm -v ./verify-docker.sh:/verify-docker.sh apache/solr:9.0.0 sh /verify-docker.sh
24. I completely agree with everything you've said @janhoy, and really like that solution! I am by no means an expert on this verification stuff, so what do you imagine `verify-docker.sh` would be doing under the hood?
25. Wouldn't it be simpler for the release manager to build the docker image, examine the sha256 hash of the image, and publish that to the download location, making it official? Someone who wants to use the official Solr docker image who is ultra-paranoid can reference the image by hash like so: docker run --rm solr@sha256:02fe5f1ac04c28291fba23a18cd8765dd62c7a98538f07f2f7d8504ba217284d That runs Solr 8.7, the official one. It's compact and can even be broadcasted easily in the release announcement for future Solr releases for people to get _and run_ the latest release immediately, and be assured it's the correct one. I wonder what other major Apache projects do.
26. > what do you imagine `verify-docker.sh` would be doing under the hood It would verify the sha512 sum of each jar with the corresponding checksum published by the project, and also allow the user to verify the PGP signature of each jar to assert it was signed by a committer. > Wouldn't it be simpler for the release manager to build the docker image, examine the sha256 hash of the image, and publish that to the download location, making it official That's a great idea David, that the RM records the image SHA when pushing, and publishes that sha. People can then either pull with SHA or verify later with `docker images --digests solr`. That should be sufficient for most users. For those who want to futher assert that they use the very same binaries signed by the committer, we could still document how to perform PGP checks on the jars.
27. > we could still document how to perform PGP checks on the jars Just confirming... we agree that ./verify-docker.sh is needless, and we need to just _document_ (e.g. in the ref guide) how to verify individual JARs at the CLI.
28. Yea, add the image hash to download page and document how to check each jar file.

github_pulls_reviews:

1. let's remove `mak@trinity10:~$` from the commands
2. Hi, as noted on Slack already. Like for the refguide, don't yet execute docker by default on assemble. I would make it optional for now, until we find a way how to assemble docker without docker or make it print a warning if no docker is enabled.
3. +1 100% to what Uwe said
4. Here and probably some README type places, the assumption is that the image is named "solr". Yet it appears that you're deciding that the name "apache/solr" is the new default name. I think pick one or or the other.
5. Are you making general improvements to the test scripts, or do they relate to necessities of the port to lucene-solr? I suggest suppressing your natural urge to improve code in a porting job to limit to just do the port. Future work items can refactor common code out of bash scripts or whatever else.
6. These changes were necessary for separating out the test output from the source directory from the tests as well as having unique container names for the tests. This was needed in the migration, because the tests assumed that you were running in a particular directory layout before (the layout in docker-solr). Moving to gradle, this needed to be a lot more flexible.
7. But I am happy to leave the testing stuff to a different PR if that will be easier to review.
8. I will pick apache/solr. It will be easier for us to find and replace if we decide to do something else later.
9. I'm also going to go with the assumption that if we go with apache/solr, we will backfill all of the docker-solr releases to apache/solr. (So all of the previous version tags will be available under the new name.)
10. Your justification makes sense. The cause of those changes weren't clear to me before.
11. I think our convention is to declare the plugin version in root `build.gradle` with `apply false`
12. I'm confused why we need this as an intermediate docker image containing just the tgz? Also, why make this configurable and why not `FROM scratch`?
13. Yeah, the idea was to allow for people to build in environments that don't have access to docker hub. Didn't know about scratch, but it's perfect. Thanks for the suggestion.
14. should be `apache/solr`

jira_issues:

jira_issues_comments: