Item 47
git_comments:

1. Python's logging library doesn't define anything more detailed than DEBUG, but we'd like a finer-grained setting for for highly detailed messages, e.g. logging every single incoming request.
2. 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.
3. TODO: currently we maintain just a simple map from all key info -> value. However, some key fields are far more common to filter on, so we'd want to index by them, e.g. host, client.id, metric name.
4. Don't do any logging here so we get rid of the mostly useless per-request Apache log-style info that spams the debug log
5. The port to listen on on the worker node, which will be forwarded to the port listening on this driver node
6. Convert to tuple of pairs because dicts & lists are unhashable
7. Non-final because these are set via configure()
8. Static package-private so unit tests can mock reading response
9. The set of metrics are updated in init/metricChange/metricRemoval
10. * 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.
11. Static package-private so unit tests can use a mock connection
12. * * MetricsReporter that aggregates metrics data and reports it via HTTP requests to a configurable * webhook endpoint in JSON format. * * This is an internal class used for system tests and does not provide any compatibility guarantees.
13. connection.getResponseCode() implicitly calls getInputStream, so always set to true. On the other hand, leaving this out breaks nothing.
14. For error conditions, we expect them to come with a response body that we can read & log
15. return value not expected to be used
16. * 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.
17. We should be left with the modified version of metric1 and metric3
18. added in init, modified added by change added in init, deleted by removal
19. Expect that a request is made with the given response code
20. Should contain an empty list of metrics, i.e. we report updates even if there are no metrics to report to indicate liveness
21. Should contain client info...
22. public for testing

git_commits:

1. **summary:** MINOR: Add HttpMetricsReporter for system tests
   message: MINOR: Add HttpMetricsReporter for system tests Author: Ewen Cheslack-Postava <me@ewencp.org> Reviewers: Apurva Mehta <apurva@confluent.io>, Ismael Juma <ismael@juma.me.uk> Closes #4072 from ewencp/http-metrics (cherry picked from commit 718dda1144629d824f4bdb8ff73fbd531a22723a) Signed-off-by: Ewen Cheslack-Postava <me@ewencp.org>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** MINOR: Add HttpMetricsReporter for system tests
   body:

github_pulls_comments:

1. @ijuma @apurvam Working on this for some downstream tests, but I'm looking to add an alternative way of collecting metrics that is less painful than using JmxTool. This adds an HttpReporter and ports one service to use it. I think this is a better long term solution, but didn't want to necessarily port everything over immediately.

2. Thanks for the PR. If I understand correctly, Here is the way it's supposed to work: 1. Each service will have to be modified to report metrics using the `HttpMetricsReporter` java class right. 2. The `HttpMetricsReporter` class needs to be configured to submit metrics using `POST` requests to a python http server. 3. Python service classes access the metrics from the `metrics` method of the http service. If so, I expected the `ProducerPerformance` code be updated to write to the `HttpMetricsReporter`. Or am I missing something?
3. I think you got the setup right but am not sure if we're interpreting one of the steps differently. You have the flow of data correct (push from the process containing the metrics -> Python web server). ProducerPerformance now [inherits from the HttpMetricsCollector mixin instead of the JmxMixin](https://github.com/apache/kafka/pull/4072/files#diff-5132132f3a20daf38c398cab25152dc6R27). That mixin runs the http webserver and [stores and exposes the metrics](https://github.com/apache/kafka/pull/4072/files#diff-69c8a16f2e231f17c84bcbf90f21b929R95) so that tests can [query the service for relevant metrics](https://github.com/apache/kafka/pull/4072/files#diff-c0b3be3803017fe10aed86e31fbb91b0R180). The only work the ProducerPerformance service needs to do is make sure the HttpMetricsCollector mixin is properly initialized (to start the http server), that it configures it's own process to point at that http server with the HttpMetricsReporter, and then stop the server when it is done with the service. There are a few differences from the JMX version that might be worth considering: * This opts in for all metrics without any filtering by default. * We store a complete history of all samples. I think JMX was only storing something like min/max/current. I think the main concern here would be for long running tests. For the first issue we can easily add filtering on the storage side in a similar way we had jmx_attributes flags for JMX, though I kind of like minimizing the user effort. For the second one, I am not sure if we really need all of them stored, but another option would be to only save latest by default and log the rest. For debugging purposes, it seems like it would be nice to have them logged. I also considered if a mixin was the right approach here or not. I don't feel that strongly about it, but I'd like to minimize the effort required to get metrics out of a service as much as possible. The main drawback is there is some awkwardness in trying to make sure method names will be properly namespaced/isolated. There are also some utilities that I'm sure we'd eventually want here but I haven't needed for the minimal changes I made so far. In particular, a `wait_until` that works specifically for waiting on metric values could be used elsewhere (the one service I changed just happened not to need it).
4. Thanks for the explanation @ewencp. I was under the impression that this patch would also have to update https://github.com/apache/kafka/blob/trunk/tools/src/main/java/org/apache/kafka/tools/ProducerPerformance.java to populate metrics in the `HttpMetricsReporter.java`, otherwise the metrics would never be exported. Or is this handled implicitly somehow? In particular, we need the java `HttpMetricsReporter` to make the `POST` call to the python http server periodically to export the metrics. So it at least needs the connection details for the http sever, and also periodic updates from the service whose metrics are being reported. As such, my assumption was that the Java service implementations would need to be updated to do this as well.
5. I kicked off a system test run with this branch, it is waiting in the queue.
6. Looks like the throttling test and quota test failed with: ``` [2017-10-18 16:05:28,847] INFO values: client.id = some_id metrics.host = null metrics.period = 1 metrics.url = http://0e12511cc22b:38928 (org.apache.kafka.tools.HttpMetricsReporter$1) [2017-10-18 16:05:28,850] INFO Configured HttpMetricsReporter for http://0e12511cc22b:38928 to report every 1 seconds (org.apache.kafka.tools.HttpMetricsReporter) [2017-10-18 16:05:28,952] WARN The configuration 'sasl.mechanism.inter.broker.protocol' was supplied but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig) [2017-10-18 16:05:28,953] INFO Kafka version : 1.1.0-SNAPSHOT (org.apache.kafka.common.utils.AppInfoParser) [2017-10-18 16:05:28,953] INFO Kafka commitId : 8d76b41e61dce8c4 (org.apache.kafka.common.utils.AppInfoParser) [2017-10-18 16:05:30,004] ERROR Error reporting metrics (org.apache.kafka.tools.HttpMetricsReporter) java.net.UnknownHostException: 0e12511cc22b at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:178) at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392) at java.net.Socket.connect(Socket.java:579) at java.net.Socket.connect(Socket.java:528) at sun.net.NetworkClient.doConnect(NetworkClient.java:180) at sun.net.www.http.HttpClient.openServer(HttpClient.java:432) at sun.net.www.http.HttpClient.openServer(HttpClient.java:527) at sun.net.www.http.HttpClient.<init>(HttpClient.java:211) at sun.net.www.http.HttpClient.New(HttpClient.java:308) at sun.net.www.http.HttpClient.New(HttpClient.java:326) at sun.net.www.protocol.http.HttpURLConnection.getNewHttpClient(HttpURLConnection.java:997) at sun.net.www.protocol.http.HttpURLConnection.plainConnect(HttpURLConnection.java:933) at sun.net.www.protocol.http.HttpURLConnection.connect(HttpURLConnection.java:851) at sun.net.www.protocol.http.HttpURLConnection.getOutputStream(HttpURLConnection.java:1092) at org.apache.kafka.tools.HttpMetricsReporter$HttpReporter.run(HttpMetricsReporter.java:185) at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471) at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:304) at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:178) at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) at java.lang.Thread.run(Thread.java:745) [2017-10-18 16:05:54,962] INFO [Producer clientId=some_id] Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms. (org.apache.kafka.clients.producer.KafkaProducer) (END) ``` It looks like HttpMetricsReporter isn't being configured correctly.
7. Interesting, it looks like it is because the hostname that it gets when running from a docker container is just the docker container's ID. If you run from a normal EC2 instance it works fine. That's actually quite an annoying issue because it means there's no way to contact into the container running for Jenkins unless Jenkins knows to map specific ports. It's a purely Jenkins problem -- running with ducker or from a real ec2 instance works fine. I'll need to think about how to fix that, though it may be about the infrastructure rather than this patch. I could also make the driver pull from the HttpMetricsReporter, but I didn't really want to have to embed an entire webserver in it...

8. @ijuma @apurvam A few updates: * Minor modifications to the reporter and I've added unit tests that provide decent line % coverage * Updated the vagrant setup to arrange for the driver hostname -> ip mapping to be setup automatically. I'm open to other options, but this seems like the best option to me given the various test configurations we see today (completely local Vagrant setup, docker driver + Vagrant ec2 VMs, ducker) * Moved shutdown for the metrics listener to the `stop()` method for services. This is the right place for it, but does also assume subclasses call super(Class, self).stop() properly * Not directly related, but the ducker commands were assuming `releaseTarGz` target should be executed, but only the `systemTestLibs` target is actually necessary. I adjusted this in the scripts, but we can refactor this to a separate PR if we want to keep the changes separated. I kicked off a previous round of tests w/ an earlier version of patches https://jenkins.confluent.io/job/system-test-kafka-branch-builder/1113/console and they seem to be good. Plan to do the same with these updates once I'm back on the VPN.
9. Thanks @ewencp. With regards to ducker, you need to update the readme as well. cc @cmccabe
10. @ijuma ack, updated the README as well as the `.travis.yml` that was doing more than it needs to.
11. Thanks for the clarifications, @ewencp . My comments were just that: I am happy with things as they are. The patch LGTM!
12. Cool, was looking for a clean build but Jenkins seems to be in a screwy state. Also, need a +1 from another committer -- @ijuma or @hachikuji perhaps?
13. SUCCESS 8063 tests run, 5 skipped, 0 failed. --none--
14. SUCCESS 8063 tests run, 5 skipped, 0 failed. --none--
15. SUCCESS 8063 tests run, 5 skipped, 0 failed. --none--
16. Thanks for the reviews. Merging this to trunk and 1.0. I'd also like this on 0.11 but it doesn't cherry-pick cleanly. I will follow up with a PR for that just so I can verify clean unit and system test runs for the cherry-picked and conflict-resolved version.

github_pulls_reviews:

1. It seems a bit asymmetric to have to call `stop` in the subclass when we never had to call start. It might be missed in future subclasses, causing the thread to leak.
2. It would be good to have unit tests for this class. Also, do we want to make it clear from the name that this pushes data to an endpoint? One could also imagine a HTTP reporter that returns metrics via a `GET`.
3. Perhaps it would be worth mentioning that that passing these properties directly into `producer.properties` or `consumer.properties` would mean that those clients will automatically use the the `PushHttpMetricsReporter` configured for pushing their metrics to this server. I had missed this link in the first pass since I missed the `%s(metrics-props)s` argument added to the `--producer-props` of the performance producer. So it seemed like it was working by magic.
4. To clarify for my understanding, the worker which runs the service uses HTTP on the reverse tunnel to push metrics to the server running on the test driver?
5. The `ProducerPerformanceService` would have to call `super.stop()` for this to be invoked right? I don't think it does it presently. Please correct me if I am wrong.
6. Sure thing, added a note.
7. It would need to if it overrode `stop()`. However, the method resolution order for multiple inheritance will make this run first (because the mixin is listed in the super classes of `ProducerPerformanceService` first), then this `super` call will pass it along. Here's a quick example showing the equivalent class hierarchy: ``` >>> class Base(object): ... pass ... >>> class Mixin(object): ... pass ... >>> class Concrete(Mixin, Base): ... pass ... >>> Concrete.mro() [<class '__main__.Concrete'>, <class '__main__.Mixin'>, <class '__main__.Base'>, <type 'object'>] ``` When python dispatches the initial call and any `super` calls, it works in the order of that list. Putting mixins first usually makes life easier since the mixins have to take care to properly call `super` but classes using the mixin often don't, or only need to for methods they would have to call it for under normal circumstances in order to let the super class do its work. This is also why the constructor is written the way it is with the flexible `**kwargs` -- it allows the mixin to work with pretty much any other constructor parameters from the class using the mixin.
8. Btw, the call order being confusing is a downside of mixins in python, I was also debating whether we should just use composition and have the classes using this do everything explicitly. The nice thing with this approach is that as long as this class implements start + clean up properly, there isn't any burden on the class using it.
9. That's correct.
10. Thanks for that note ewen. I learned something!
11. Is there a reason to start the server thread here in stead of in the `start_node` method?
12. Oh my, that is quite some mock code!
13. In this an the next test, I am not sure what exactly is being tested.
14. main reason was that we only need to start one server thread but there could be N nodes reporting to it (so `start_node` would be called N times). I can move it to the `start` method if preferred. that's really the right way to handle this, though I'm not sure there's much effect to moving it. only thing i can think of is if done here we may be unnecessarily binding a bunch of ports early based on when the test instances are instantiated, but I honestly can't remember when exactly instantiation occurs in ducktape (during scanning to build out the list of tests or at the time of the test run).
15. It's just testing that when there is an error, we read the response. Mainly we want to verify that we are getting and logging the response when there is an error, but testing for the logging step is harder. Even the actual `readResponse` is mocked out because mocking the calls that `Scanner` makes seemed way messier, error prone, and possibly even JVM-specific.
16. I did consider a non-strict mock here since there are so many calls. Some are useful to verify (e.g. content type), others seem less so (e.g. `setUseCaches`. I'm happy to switch to a non-strict mock to cut down on test noise if that's preferred.
17. Interesting, so `releaseTarGz` was never needed? I had assumed that there was an actual reason why this was different when the system tests were run via docker.
18. What's our compatibility guarantees for this class? If we offer none, it might be worth mentioning it in the class javadoc.
19. Nit: can we move all non-final fields after the final ones? Also, we can maybe add a comment above the non-final ones stating that they are set via `configure`. It might also be worth mentioning that `metrics` gets updated via

`init`/`metricChange`/`metricRemoval`. That makes it easier for someone to understand the lifecycle without having to look at the whole class.
20. Can we use try with resources?
21. Are we using `PowerMock` just because of the static methods that we're mocking? Do they have to be static methods? I've had the pleasure of debugging `PowerMock` issues when upgrading to Java 9 and I think it's best to restrict usage to those cases where it's really beneficial.
22. We've had the `systemTestLibs` target for quite a long time for this. There definitely shouldn't be a need for different targets between regular and Docker versions of the tests, so I think this just slipped in accidentally. If we do diverge, we're better off just getting them consistent than having different requirements.
23. None, it's not public API or anything so the guarantees are just as lacking as any other internal class :) I've added a note to the javadoc about this.
24. Ack
25. good point, fixed
26. Hmm, interesting. Are you suggesting a partial mock instead? I usually avoid them because the setup is way messier than mocking statics.
27. It could be a separate class/interface for the the methods that need to be mocked. We don't have to do it here (I approved the PR), but I just wanted to raise that there's a cost to PowerMock's magic and that it may be better to use it in exceptional cases instead of by default.

jira_issues:

jira_issues_comments: