Item 103
git_comments:

1. ~ Druid - a distributed column store. ~ Copyright 2012 - 2015 Metamarkets Group Inc. ~ ~ Licensed 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.
2. Tests
3. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
4. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
5. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
6. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
7. * * {@inheritDoc}
8. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
9. * * {@inheritDoc}
10. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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

11. * * When used in MapBasedRow, field in GenericRecord will be interpret as follows: * <ul> * <li> avro schema type -> druid dimension:</li> * <ul> * <li>null, boolean, int, long, float, double, string, Records, Enums, Maps, Fixed -> String, using String.valueOf</li> * <li>bytes -> Arrays.toString() </li> * <li>Arrays -> List&lt;String&gt;, using Lists.transform(&lt;List&gt;dimValue, TO_STRING_INCLUDING_NULL)</li> * </ul> * <li> avro schema type -> druid metric:</li> * <ul> * <li>null -> 0F/0L</li> * <li>int, long, float, double -> Float/Long, using Number.floatValue()/Number.longValue()</li> * <li>string -> Float/Long, using Float.valueOf()/Long.valueOf()</li> * <li>boolean, bytes, Arrays, Records, Enums, Maps, Fixed -> ParseException</li> * </ul> * </ul>
12.
13.
14.
15. * * This implementation using injected Kafka topic name as subject name, and an integer as schema id. Before sending avro * message to Kafka broker, you need to register the schema to an schema repository, get the schema id, serialized it to * 4 bytes and then insert them to the head of the payload. In the reading end, you extract 4 bytes from raw messages, * deserialize and return it with the topic name, with which you can lookup the avro schema. * * @see SubjectAndIdConverter
16.
17. * * Schema Repository is a registry service, you can register a string schema which gives back an schema id for it, * or lookup the schema with the schema id. * <p> * In order to get the "latest" schema or handle compatibility enforcement on changes there has to be some way to group * a set of schemas together and reason about the ordering of changes over these. <i>Subject</i> is introduced as * the formal notion of <i>group</i>, defined as an ordered collection of mutually compatible schemas, according to <a href="https://issues.apache.org/jira/browse/AVRO-1124?focusedCommentId=13503967&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-13503967"> * Scott Carey on AVRO-1124</a>. * <p> * So you can register an string schema to a specific subject, get an schema id, and then query the schema using the * subject and schema id pair. Working with Kafka and Avro, it's intuitive that using Kafka topic as subject name and an *

incrementing integer as schema id, serialize and attach them to the message payload, or extract and deserialize from * message payload, which is implemented as {@link Avro1124SubjectAndIdConverter}. * <p> * You can implement your own SubjectAndIdConverter based on your scenario, such as using canonical name of avro schema * as subject name and incrementing short integer which serialized using varint.
18. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
19. 2. write new avro file using AvroStorage
20. 0. write avro object into temp file.
21. 3. read avro object from AvroStorage
22. 1. read avro files into Pig
23. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
24. serde test
25. encode data
26. towards Map avro field as druid dimension, need to convert its toString() back to HashMap to check equality
27. test dimensions
28. write avro datum to bytes
29. test metrics
30. * Licensed to Metamarkets Group Inc. (Metamarkets) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. Metamarkets 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.
31. encode schema id
32. prepare data

git_commits:

1. **summary:** Merge pull request #1858 from zhaown/avro-module
   message: Merge pull request #1858 from zhaown/avro-module Support avro ingestion for realtime & hadoop batch indexing

github_issues:

1. **title:** Avro support
   body: Should probably be an extension. For realtime we need a ByteBufferInputRowParser (something similar to the ProtoBufInputRowParser, but for Avro). For batch we need a recommended Avro-aware InputFormat and an InputRowParser that can read whatever type is returned by that InputFormat. I haven't used Avro before so I'm not sure what the right choice of InputFormat is. `AvroKeyInputFormat` from https://avro.apache.org/docs/1.7.0/api/java/org/apache/avro/mapreduce/AvroKeyInputFormat.html seems like a possible candidate.
   label: code-design

2. **title:** Avro support
   body: Should probably be an extension. For realtime we need a ByteBufferInputRowParser (something similar to the ProtoBufInputRowParser, but for Avro). For batch we need a recommended Avro-aware InputFormat and an InputRowParser that can read whatever type is returned by that InputFormat. I haven't used Avro before so I'm not sure what the right choice of InputFormat is. `AvroKeyInputFormat` from https://avro.apache.org/docs/1.7.0/api/java/org/apache/avro/mapreduce/AvroKeyInputFormat.html seems like a possible candidate.
3. **title:** Avro support
   body: Should probably be an extension. For realtime we need a ByteBufferInputRowParser (something similar to the ProtoBufInputRowParser, but for Avro). For batch we need a recommended Avro-aware InputFormat and an InputRowParser that can read whatever type is returned by that InputFormat. I haven't used Avro before so I'm not sure what the right choice of InputFormat is. `AvroKeyInputFormat` from https://avro.apache.org/docs/1.7.0/api/java/org/apache/avro/mapreduce/AvroKeyInputFormat.html seems like a possible candidate.
4. **title:** Avro support
   body: Should probably be an extension. For realtime we need a ByteBufferInputRowParser (something similar to the ProtoBufInputRowParser, but for Avro). For batch we need a recommended Avro-aware InputFormat and an InputRowParser that can read whatever type is returned by that InputFormat. I haven't used Avro before so I'm not sure what the right choice of InputFormat is. `AvroKeyInputFormat` from https://avro.apache.org/docs/1.7.0/api/java/org/apache/avro/mapreduce/AvroKeyInputFormat.html seems like a possible candidate.
5. **title:** Avro support
   body: Should probably be an extension. For realtime we need a ByteBufferInputRowParser (something similar to the ProtoBufInputRowParser, but for Avro). For batch we need a recommended Avro-aware InputFormat and an InputRowParser that can read whatever type is returned by that InputFormat. I haven't used Avro before so I'm not sure what the right choice of InputFormat is. `AvroKeyInputFormat` from https://avro.apache.org/docs/1.7.0/api/java/org/apache/avro/mapreduce/AvroKeyInputFormat.html seems like a possible candidate.

github_issues_comments:

1. **body:** We have had avro working for a while, but code is not generic enough and very specific to our schemas. In fact, it will not be possible to take a general avro schema and convert it to druid row because avro has support for very many complex types, so we will have to compromise anyway. Also, it was written pre druid-0.8.0 era where it wasn't possible to have InputFormats that could return anything but Text records. With druid-0.8.2, the limitation regarding Text records is gone. In my org, some people are working on next gen druid avro integration.
   label: code-design
2. I'm using avro with druid for production, for batch indexing, it's not complicated based on @himanshug 's https://github.com/druid-io/druid/pull/1472, and I'm using my `AvroValueInputFormat` which is the mirror of `AvroKeyInputFormat`. But for realtime indexing, it's a bit more cumbersome because you need an schema to deserialize avro object from binary stream, and you don't want to send schema with every serialized record to kafka. Then you need an schema registry, currently we are using [schemarepo](https://github.com/schema-repo/schema-repo) and `camus schema registry client`, the latter is not in the maven central... I'll try to clean my code and try to submit an PR for this this weekend if I got some time.
3. Please check https://github.com/druid-io/druid/pull/1858

github_pulls:

1. **title:** Support avro ingestion for realtime & hadoop batch indexing
   body: [Apache Avro™](https://avro.apache.org/) is a schematic data serialization system, well integrated with hadoop ecosystem. Object schema is saved with encoded objects into a [Object Container File](http://avro.apache.org/docs/1.7.7/spec.html#Object+Container+Files), so decoding is easy and natively supported by hadoop. Adding avro ingestion support to druid hadoop indexing is not complicate. Sending encoded avro objects into streams like kafka, you don't want to send schema with every object, the overhead is usually too big, but you still need schema to decode it, so people use [schema repository](https://issues.apache.org/jira/browse/AVRO-1124). In producer end you register schema to the repository before sending, get an schema `id`, send the `id` with encoded data to stream. In consumer end you extract the schema `id` from stream message, lookup the schema from repository, use it to decode the

data. This PR treat avro object as `Map<String, Object>`, actually in [early versions](http://avro.apache.org/docs/1.1.0/api/java/org/apache/avro/generic/GenericRecord.html) avro `GenericRecord` does extend `Map<String, Object>`. In hadoop batch ingestion, it uses custom `AvroValueInputFormat`, `AvroHadoopInputRowParser` and `AvroParseSpec`. If your need a reader schema that differs with writer schema, you can set it in the `tuningConfig#jobProperties` with name `avro.schema.input.value` or `avro.schema.path.input.value`, the former is JSON text while the later is the file path in HDFS of the reader schema, if both are setted, the former is used. NOTE: **The reader schema is for all input files.** The spec file will look likes: ``` json { "type": "index_hadoop", "spec": { "dataSchema": { "parser": { "type": "avro_hadoop", "parsSpec": { "format": "timeAndDims", "timestampSpec": {}, "dimensionsSpec": {} } } }, "ioConfig": { "type": "hadoop", "inputSpec": { "type": "", "inputFormat": "io.druid.data.input.avro.AvroValueInputFormat" } }, "tuningConfig": { "jobProperties": { "avro.schema.path.input.value": "/path/to/my/schema.avsc", "avro.schema.input.value": "my_schema_JSON_text" } } } ``` Towards realtime ingestion, it needs an schema repository, using [shcema-repo](https://github.com/schema-repo/schema-repo). I'm aware there are two schema repository implementaions, the other is [schema-registry](https://github.com/confluentinc/schema-registry) from [Conflunt](https://github.com/confluentinc), probably written by same team. The latter is easier to use, but not in maven central, so I choosed the formmer. This extension should be easy to extend to communate with some other certain kind of schema registry servers. The spec file will look likes: ``` json { "dataSchema": { "parser": { "type": "avro_stream", "avroBytesDecoder": { "type": "schema_repo", "subjectAndIdConverter": { "type": "avro_1124", "topic": "${YOUR_TOPIC}" }, "schemaRepository": { "type": "avro_1124_rest_client", "url":"${YOUR_SCHEMA_REPO_END_POINT}", } }, "parsSpec": { "format": "timeAndDims", "timestampSpec": {}, "dimensionsSpec": {} } } } ``` Fixes #1844

github_pulls_comments:

1. Towards wrapping avro object as `Map<String, Object>`, I'm thinking maybe druid could support nested dimensions/metrics. It could be expressed like`level_0_name.level_1_name`, in `InputRow#getDimension()` and `InputRow#getLongMetric()`, it can check if the full name in the keySet, if not then check if the `level_0_name` in the keySet && `map.get("level_0_name")` is a map, and do it recursively. So if there is a top level field named "level-0-name.level-1-name", then druid gets its value as dimensions/metrics, if there isn't the top level but a nested field `{"level_0_name": {"level_1_name": "nested_value:}}`, then druid gets the nested_value. The cons is the ambiguity, but I think in some scenario it's acceptable?
2. At least for batch indexing, it would be ideal if an avro reader schema could be passed in as part of the parse spec (I would prefer passing a filename rather than embedding the entire schema directly in the parse space). That way, if the schema of the avro files changes over time, the files can still be ingested together and provide a consistent view. If no reader schema is specified, the schema embedded in the avro file should be used. Something like: ``` "parser": { "type": "avro-hadoop", "parsSpec": { "format": "avro", "readerSchema": "/path/to/my/schema.avsc" "timestampSpec": {}, "dimensionsSpec": {} } } } ```
3. @will-lauer Yes, you are right, reader schema should be supported. But it should not be one schema per `parserSpec` nor `parser`, but per `pathSpec`, because you could need one parser to parse all avro datas in different dirs with different schema, like pv/impr/click. I currently cannot find a decent way to do this. Adding custom properties into `PathSpec` seems a bit overdo?
4. @drcrallen the CI build is failed in jdk8 while successed in jdk7, and the failed module(druid-server) is irrelevant to this PR, need it a rebuild?
5. Test in failure: `AnnouncerTest.testSanity:99 » test timed out after 60000 milliseconds` Some of the tests which use zookeeper have unpredictable problems in Travis CI. Please close then open the PR to restart the test.
6. @zhaown FYI, a few of these files will need newlines at the end of the files.
7. @himanshug @codingwhatever Please check if the commit is ok, there is something to be done in unit tests, I'll make them right later, have to go now... The main mess is that testing serde with avro-1124-relevant classes, if you init an `Avro1124RESTRepositoryClient` with an endpoint url, it will actually communicate with the endpoint... PS: I think the `io.druid.TestObjectMapper` is not packaged with the druid-api.jar, so I have to use and configure my own `ObjectMapper`.
8. **body:** can you please add documentation for this module under docs/content/development/avro-extensions.md and link that as an experimental feature in docs/content/toc.textile ?
   label: documentation
9. @zhaown The schema reading logic and the union/array logic for hadoop ingestion look correct now.

10. @zhaown if we can get some docs, I'm :+1:
11. @zhaown can we finish this one off?
12. **body:** @fjy and all, really sorry for the disappearance, I've been busy on another project, and I think it needs some field tests because the code is changed a bit from what I've tested initially. I'll try to add docs and do the tests this week.
    label: test
13. **body:** @fjy Added some docs, please check if them are OK.
    label: documentation
14. I'm :+1: . @himanshug ?
15. @zhaown I'm assuming nothing changed other than addition of documentation. :+1: besides the minor comments. pls squash your commits. thanks for the contrib, this will be very useful to many.
16. I can merge this once teh documentation changes are addressed.
17. @zhaown For information about squashing your commits, please see: https://github.com/druid-io/druid/blob/master/CONTRIBUTING.md
18. @fjy @himanshug Done some docs refinement, pls check if the grammer and else is OK, if it's OK I'll squash my commits.
19. :+1: Please squash your commits @himanshug any more comments?
20. :+1: ready to merge after squashing
21. Commit squashed, thanks you guys.

github_pulls_reviews:

1. **body:** Is this always a hard failure? meaning is it possible to skip bad values?
   label: code-design
2. **body:** Can you add a comment here as to why there are so many unsupported operation exceptions? as in, they were simply not needed for the initial use case, but may be added later? or simply can never be added?
   label: documentation
3. The IOException comes from the InputStream#read(), usually because corrupt data or wrong schema/data match, which make the whole record is unreadable and the partially read data meaningless. So I think it's impossible to skip the bad field in one row(avro record). And for the wrong schema/data match, there is a better way than skipping the bad field: set proper reader schema against the writer schema.
4. Mostly because the methods are unused for druid. Some of them can be added while some of them cannot, like `size()` can be added while "remove(Object value)" cannot, the `remove()` cannot be added because avro is schematic and some filed is must-have. I'll add comments in next commit.
5. can you please use https://github.com/druid-io/druid/blob/master/extensions/namespace-lookup/src/test/java/io/druid/query/extraction/namespace/NamespacedExtractorTest.java#L1 for the license header?
6. since is only intended to work with one type of value, can you implement InputRowParser &lt;AvroValue&gt; instead?
7. this looks same as https://github.com/druid-io/druid-api/blob/master/src/main/java/io/druid/data/input/impl/TimeAndDimsParseSpec.java ? It seems this one is not needed.
8. **body:** can you use `com.metamx.common.logger.Logger` everywhere for logging?
   label: code-design
9. **body:** nit: in general druid files always end with a newline, can you have same in all the files?
   label: code-design
10. there are multiple files which need that. I just had a comment in the main thread of the PR
11. **body:** nit: newline here
    label: code-design
12. **body:** [get/put]SubjectAndId(..) are probably more appropriate names for these 2. Also can you add some javadocs on this interface e.g. in what situation would someone need to add another implementation?
    label: documentation
13. **body:** can you add some more descriptive text to the comment?
    label: documentation
14. **body:** can we call it "AvroExtensionsModule" just to be consistent with name of this extension
    label: code-design
15. I agree with @will-lauer that the schema should be specified as a path to a file on the hadoop FS.

16. GenericRecordAsMap is fine for primitives, but we should handle ARRAY for multivalued dimensions, UNION in case schema was something like ["null","int"] instead of just "int" (pig AvroStorage would generate schema like that). you could write a map converter something like below (we've been using something similar in our production for a while) ``` protected static Map<String, Object> genericRecordToMap(GenericRecord genericData) throws IOException { Map<String, Object> data = new HashMap<String, Object>(); List<Schema.Field> fields = genericData.getSchema().getFields(); for(Schema.Field field : fields) { String name = field.name(); Object value = genericData.get(name); data.put(name, value); if(value != null) { Schema.Type type = field.schema().getType(); if(type == Schema.Type.UNION) { //handling primitive types defined in schema like ["null","int"] Schema fieldSchema = field.schema(); int index = GenericData.get().resolveUnion(fieldSchema, value); type = fieldSchema.getTypes().get(index).getType(); } if(type == Schema.Type.STRING) { data.put(name, value.toString()); } if(type == Schema.Type.ARRAY) { //handling multivalued dimensions List<String> dimValues = new ArrayList<String>(); for(Object dimValue : (Collection)value) { Object subvalue = dimValue; if(dimValue instanceof GenericRecord) { /* This addresses the way pig AvroStorage outputs an array/ * bag of values. */ subvalue = subvalue = ((GenericRecord)dimValue).get(name); if (subvalue == null) { throw new IOException("Could not process bag " + name + ", it's elements should be named " + name); } } dimValues.add(subvalue.toString()); } data.put(name, dimValues); } } } return data; } ```
17. just FYI, above code is taken from @codingwhatever 's work.
18. @zhaown I think it is fine to assume that if user specified explicitly then same reader schema can be used to read all the pathSpecs using AvroValueInputFormat .
19. I think getting it from jobProperties would be a good place, what say?
20. **body:** should this be ParseException instead of RuntimeException ?
    label: code-design
21. @will-lauer @codingwhatever @himanshug Added reader schema setting in jobProperties. I think although in this way user cannot set one schema per pathSpecs, but it's better than cannot set reader schema at all.
22. @himanshug I think ARRAY is already handled because ARRAY is expressed as `org.apache.avro.generic.GenericData#Array` which extends `AbstractList`. For UNION, can we just check the actual type? For instance field type is ["null","int","array"], if the actual field value is int, then return `int`, if array, then return `Array`? I'm not using pig AvroStorage, maybe I didn't understand the problem, correct me if I'm wrong please.
23. @zhaown You are correct about the handling of unions. The problem with array is that pig AvroStorage does not support the array type for primitives. The closest data structure is a bag of tuples; so the array ends up looking like [{"name": "val1"}, {"name": "val2"}, ...]. Unfortunately the AvroStorage types are not specifically mapped to pig types. See the type table in https://cwiki.apache.org/confluence/display/PIG/AvroStorage
24. @codingwhatever After checking out the AvroStorage wiki I got it, I just added a switch to let user can tell the hadoop parser to treat AvroStorage output array properly, please check if that is ok. It's odd that in my tring to generate an AvroStorage output of avro array, the name of elements is "name_0" instead of "name", eg. input array: `{"someArray": [4, 9, 1]}`, AvroStorage output array: `{"someArray": [{"someArray_0": 4}, {"someArray_0": 9}, {"someArray_0": 1}]}`, didn't find why. I did't follow the `genericRecordToMap` way, because I have several avro data whose schema is about 30 fields including nested maps, while my druid data source is only about 5 each for dimensions and metrics, so convert the whole data into map is a bit wasted(I didn't use an avro reader schema because there are 5 different not-well-designed schema in one druid data source, writting a common reader schema is kind of not easy). I'm wondering if there is someone else in the situation like me that want to save the slightly overhead of creating one `Map` for each avro record.
25. **body:** com.metamx.common.Logger.info has the formatter builtin so String.format isn't needed
    label: code-design
26. is it possible to use GenericRecordAsMap instead of building another just to extract timestamp?
27. **body:** not sure what is the use of parameterization here, It appears that this could be simplified to ``` public class AvroValueRecordReader extends AvroRecordReaderBase<NullWritable, GenericRecord, GenericRecord> ``` and that simplifies things in many places where AvroValue&lt;T&gt; has been used.
    label: code-design
28. **body:** nit: can you change the key to `avro.schema.input.value.path` , that is more conventional with key used by avro itself for the schema in configuration?
    label: code-design
29. can you add a serde test for AvroHadoopInputRowParser?
30. can you add a serde test for AvroStreamInputRowParser?

31. add new line
32. **body:** nit: I would put AvroExtensionsModule inside io.druid.data.input.avro instead of creating a package just for that class.
    label: code-design
33. I am not very familiar with schemarepo, but from a quick read of https://github.com/schema-repo/schema-repo it appears that this is a specific way of managing per record schema via the REST service. this class is forcing every customer to use same. haven't thought through completely, but does it make sense to abstract away `ByteBuffer -> GenericRecord` instead and one implementation could be Avro1124 like? also, are you using this input row parser in production with schema repo?
34. pls see https://github.com/druid-io/druid-api/blob/master/src/test/java/io/druid/data/input/impl/InputRowParserSerdeTest.java#L42 for example.
35. Just copied it from org.apache.avro.mapreduce.AvroKeyRecordReader, did't think it over. For this perticular use it's no different that remove the parameterization of `AvroValueRecordReader`, um... I think there will be nobody using this class for other use...
36. I'm using the slightly different version of this row parser in production with schema repo(2 REST service behind nginx, 3-clusters-ZK backend) in production for about 10 months, no problem, our thoughput is about 300 kafka producers, 30 kafka consumers, 30 topics, rare schema revolution. By saying "_this class is forcing every customer to use same._" you mean druid customer? I'm not guru with avro things, but after googling with "avro schema registry", I think there is not very much options, and this implemention is suitable for all(both) of search result in first page(schemarepo's and Confluent's)... Of course if you guys want support every possible deserialization from bytes to avro, I think abstracting it into `ByteBuffer -> GenericRecord` is the only way?
37. Sure, I'll catch up later, have to go now...
38. yes, I meant druid customers. I have nothing against schema-repo lib but many people (including me) have our own ways of managing schema registry and it is useful to use those existing setups. Or, some ppl might be happy enough with just distributing files (containing id->schema mapping) manually across the nodes and have code that simply works via those files instead of setting up and calling a web service. In general, I think this is something that is left to individual people , and we can give a default implementation based on schema-lib. For example, we could have an interface and impl ``` interface AvroBytesParser { GenericRecord parse(ByteBuffer bytes); } class SchemaLibBasedAvroBytesParser implements AvroBytesParser { .. } ```
39. I don't think you can mark two constructors as `@JsonCreator` , in the one you can use Boolean instead of boolean to for fromPigAvroStorage and choose false for null .
40. pls add some tests for AvroStorage based avro data parsing as well.
41. **body:** can you call it "schemaRepo" ? that would be consistent with other names used across druid.
    label: code-design
42. it will be nice if you could use camelCase for various names instead of using '_' as separator.
43. @zhaown can you include a full spec that works for avro ingestion?
44. can you say that default is TextInputFormat?
45. can you say that default is TextInputFormat?
46. **body:** nit: it will be nice to have actual json example one for the hadoop parser and one for the streaming parser.
    label: code-design

jira_issues:

1. **summary:** RESTful service for holding schemas
   description: Motivation: It is nice to be able to pass around data in serialized form but still know the exact schema that was used to serialize it. The overhead of storing the schema with each record is too high unless the individual records are very large. There are workarounds for some common cases: in the case of files a schema can be stored once with a file of many records amortizing the per-record cost, and in the case of RPC the schema can be negotiated ahead of time and used for many requests. For other uses, though it is nice to be able to pass a reference to a given schema using a small id and allow this to be looked up. Since only a small number of schemas are likely to be active for a given data source, these can easily be cached, so the number of remote lookups is very small (one per active schema version). Basically this would consist of two things: 1. A simple REST service that stores and retrieves schemas 2. Some helper java code for fetching and caching schemas for people using the registry We have used something like this at LinkedIn for a few years now, and it would be nice to standardize this facility to be able to build up common tooling around it. This proposal will be based on what we have, but we can

change it as ideas come up. The facilities this provides are super simple, basically you can register a schema which gives back a unique id for it or you can query for a schema. There is almost no code, and nothing very complex. The contract is that before emitting/storing a record you must first publish its schema to the registry or know that it has already been published (by checking your cache of published schemas). When reading you check your cache and if you don't find the id/schema pair there you query the registry to look it up. I will explain some of the nuances in more detail below. An added benefit of such a repository is that it makes a few other things possible: 1. A graphical browser of the various data types that are currently used and all their previous forms. 2. Automatic enforcement of compatibility rules. Data is always compatible in the sense that the reader will always deserialize it (since they are using the same schema as the writer) but this does not mean it is compatible with the expectations of the reader. For example if an int field is changed to a string that will almost certainly break anyone relying on that field. This definition of compatibility can differ for different use cases and should likely be pluggable. Here is a description of one of our uses of this facility at LinkedIn. We use this to retain a schema with "log" data end-to-end from the producing app to various real-time consumers as well as a set of resulting AvroFile in Hadoop. This schema metadata can then be used to auto-create hive tables (or add new fields to existing tables), or inferring pig fields, all without manual intervention. One important definition of compatibility that is nice to enforce is compatibility with historical data for a given "table". Log data is usually loaded in an append-only manner, so if someone changes an int field in a particular data set to be a string, tools like pig or hive that expect static columns will be unusable. Even using plain-vanilla map/reduce processing data where columns and types change willy nilly is painful. However the person emitting this kind of data may not know all the details of compatible schema evolution. We use the schema repository to validate that any change made to a schema don't violate the compatibility model, and reject the update if it does. We do this check both at run time, and also as part of the ant task that generates specific record code (as an early warning). Some details to consider: Deployment This can just be programmed against the servlet API and deploy as a standard war. You have lots of instances and load balance traffic over them. Persistence The storage needs are not very heavy. The clients are expected to cache the id=>schema mapping, and the server can cache as well. Even after several years of heavy use we have <50k schemas, each of which is pretty small. I think this part can be made pluggable and we can provide a jdbc- and file-based implementation as these don't require outlandish dependencies. People can easily plug in their favorite key-value store thingy if they like by implementing the right plugin interface. Actual reads will virtually always be cached in memory so this is not too important. Group In order to get the "latest" schema or handle compatibility enforcement on changes there has to be some way to group a set of schemas together and reason about the ordering of changes over these. I am going to call the grouping the "group". In our usage it is always the table or topic to which the schema is associated. For most of our usage the group name also happens to be the Record name as all of our schemas are records and our default is to have these match. There are use cases, though, where a single schema is used for multiple topics, each which is modeled independently. The proposal is not to enforce a particular convention but just to expose the group designator in the API. It would be possible to make the concept of group optional, but I can't come up with an example where that would be useful. Compatibility There are really different requirements for different use cases on what is considered an allowable change. Likewise it is useful to be able to extend this to have other kinds of checks (for example, in retrospect, I really wish we had required doc fields to be present so we could require documentation of fields as well as naming conventions). There can be some kind of general pluggable interface for this like SchemaChangeValidator.isValidChange(currentLatest, proposedNew) A reasonable implementation can be provided that does checks based on the rules in http://avro.apache.org/docs/current/spec.html#Schema+Resolution. Be default no checks need to be done. Ideally you should be able to have more than one policy (say one treatment for database schemas, one for logging event schemas, and one which does no checks at all). I can't imagine a need for more than a handful of these which would be statically configured (db_policy=com.mycompany.DBSchemaChangePolicy, noop=org.apache.avro.NoOpPolicy,...). Each group can configure the policy it wants to be used going forward with the default being none. Security and Authentication There isn't any of this. The assumption is that this service is not publicly available and those accessing it are honest (though perhaps accident prone). These are just schemas, after all. Ids There are a couple of questions about ids how we make ids to represent the schemas: 1. Are they sequential (1,2,3..) or hash based? If hash based, what is sufficient collision probability? 2. Are they global or per-group? That is, if I know the id do I also need to know the group to look up the schema? 3. What kind of change triggers a new id? E.g. if I update a doc field does that give a new id? If not then that doc field will not be stored. For the id generation there are various options: - A sequential integer - AVRO-1006 creates a schema-specific 64-bit hash. - Our current implementation at LinkedIn uses the MD5 of the schema as the id. Our current implementation at LinkedIn uses the MD5 of the schema text after removing whitespace. The additional attributes like doc fields (and a few we made up) are actually important to us and we want them maintained (we add metadata fields of our own). This does mean we have some updates that generate a new schema id but don't cause a very meaningful semantic change to the schema (say because someone tweaked their doc string), but this doesn't hurt anything and it is nice to have the exact schema text represented. An example of uses these metadata fields is using the schema doc fields as the hive column doc fields. The id is actually just a unique identifier, and the id generation algorithm can be made pluggable if there is a real trade-off. In retrospect I don't think using the md5 is good because it is 16 bytes, which for a small message is bulkier than needed. Since the id is retained with each message, size is a concern. The AVRO-1006 fingerprint is super cool, but I have a couple concerns (possibly just due to misunderstanding): 1. Seems to produce a 64-bit id. For a large number of schemas, 64 bits makes collisions unlikely but not unthinkable. Whether or not this matters depends on whether schemas are versioned per group or globally. If they are per group it may be okay, since most groups should only have a few hundred schema versions at most. If they are global I think it will be a problem. Probabilities for collision are given here under the assumption of perfect uniformity of the hash (it may be worse, but can't be better) http://en.wikipedia.org/wiki/Birthday_attack. If we did have a collision we would be dead in the water, since our data would be unreadable. If this becomes a standard mechanism for storing schemas people will run into this problem. 2. Even 64-bits is a bit bulky. Since this id needs to be stored with every row size is a concern, though a minor one. 3. The notion of equivalence seems to throw away many things in the schema (doc, attributes, etc). This is unfortunate. One nice thing about avro is you can add your own made-up attributes to the schema since it is just JSON. This acts as a kind of poor-mans metadata repository. It would be nice to have these maintained rather than discarded. It is possible that I am misunderstanding the fingerprint scheme, though, so please correct me. My personal preference would be to use a sequential id per group. The main reason I like this is because the id doubles as the version number, i.e. my_schema/4 is the 4th version of the my_schema record/group. Persisted data then only needs to store the varint encoding of the version number, which is generally going to be 1 byte for a few hundred schema updates. The string my_schema/4 acts as a global id for this. This does allow per-group sharding for id generation, but sharding seems unlikely to be needed here. A 50GB database would store 52 million schemas. 52 million schemas "should be enough for anyone". :-) Probably the easiest thing would be to just make the id generation scheme pluggable. That would kind of satisfy everyone, and, as a side-benefit give us at linkedin a gradual migration path off our md5-based ids. In this case ids would basically be opaque url-safe strings from the point of view of the repository and users could munge this id and encode it as they like. APIs Here are the proposed APIs. This tacitly assumes ids are per-group, but the change if pretty minor if not: Get a schema by id GET /schemas/<group>/<id> If the schema exists the response code will be 200 and the response body will be the schema text. If it doesn't exist the response will be 404. GET /schemas Produces a list of group names, one per line. GET /schemas/group Produces a list of versions for the given group, one per line. GET /schemas/group/latest If the group exists the response code will be 200 and the response body will be the schema text of the last registered schema. If the group doesn't exist the response code will be 404. Register a schema POST /schemas/groups/<group_name> Parameters: schema=<text of schema> compatibility_model=XYZ force_override=(true|false) There are a few cases: If the group exists and the change is incompatible with the current latest, the server response code will be 403 (forbidden) UNLESS the force_override flag is set in which case not check will be made. If the server doesn't have an implementation corresponding to the given compatibility model key it will give a response code 400 If the group does not exist it will be created with the given schema (and compatibility model) If the group exists and this schema has already been registered the server returns response code 200 and the id already assigned to that schema If the group exists, but this schema hasn't been registered, and the compatibility checks pass, then the response code will be 200 and it will store the schema and return the id of the schema The force_override flag allows registering an incompatible schema. We have found that sometimes you know "for sure" that your change is okay and just want to damn the torpedoes and charge ahead. This would be intended for manual rather than programmatic usage. Intended Usage Let's assume we are implementing a put and get API as a database would have using this registry, there is no substantial difference for a messaging style api. Here are the details of how this works: Say you have two methods void put(table, key, record) Record get(table, key) Put is expected to do the following under the covers: 1. Check the record's schema against a local cache of schema=>id to get the schema id 3. If it is not found then register it with the schema registry and get back a schema id and add this pair to the cache 4. Store the serialized record bytes and schema id Get is expected to do the following: 1. Retrieve the serialized record bytes and schema id from the store 2. Check a local cache to see if this schema is known for this schema id 3. If not, fetch the schema by id from the schema registry 4. Deserialize the record using the schema and return it Code Layout Where to put this code? Contrib package? Elsewhere? Someone should tell me...

jira_issues_comments:

1. **body:** This looks great! Re: Fingerprints These are not ideal identifiers for a schema registry for the reasons you cite. These serve a variety of purposes however, one of which is that they can be used to make equivalence checks significantly faster by comparing the (cached) fingerprint before all of the schema elements. I agree that an incrementing integer is an ideal identifier. Even with 1M schemas, that would take only 3 bytes as an Avro encoded integer. If per group, the first 63 versions take one byte and up to 8191 take 2 bytes. It makes sense to allow this id generation to be pluggable and support other types for those that want to use UUIDs, hashes, or strings. Re: code layout -- I propose we make set of new maven modules for this (e.g. avro-registry, avro-registry-persist-jdbc, avro-registry-persist-zookeeper, etc). If you have code, I can do the maven part quickly. Adding modules is easy as long as the code already has clean separation of packages and dependencies.
   label: code-design
2. This sounds great. Getting something committed to Apache SVN sooner might make it easier for folks to collaborate on extension points, layout, etc. We might commit the first version of this to a development branch named AVRO-1124. Then we can collaboratively make incompatible changes to APIs, packaging, etc., and merge it to trunk once we think things are stable enough to include in the next release.
3. Cool, I will try to get a patch against trunk ready. We can apply that on a branch or wherever makes sense. I should have something ready by early next week.
4. Is the branch necessary? This service should not require changing existing avro modules and will only depend on them. As long as it is in its own directory (e.g. lang/java/registry) and was not referenced by the aggregator lang/java/pom.xml then it will not be built, packaged, or published by the main build. While under development it can be built by executing its build scripts within that location with no side effects outside of that. I look forward to seeing it!
5. It's too soon to tell whether a branch would be useful. I wanted to mention it as an option, to remove obstacles to committing something early.
6. Is this still in progress? I am willing to contribute resources to help push this through. If it is partially complete, I would be happy to see a patch of the work in progress as myself or others may have time to take the next steps.
7. Yeah, got distracted for a few weeks, I will come back to it early next week.
8. Any status update? I will begin building the core of this next week as I need this ASAP. I would start with the core REST service and schema management with a stubbed out API for pluggable storage, id generation, and schema compatability checks. My assessment is that with the tools I know well for a standalone REST service (Jersey + Jetty + Guice + Maven), I can get this done fairly rapidly. A separate maven module in Avro can have a Java Client API, and later others can contribute implementations of the pluggable parts -- perhaps a Zookeeper id generator and an HBase storage back-end.
9. Here is a patch that has basic functionality: - Storage in mysql (probably good up to your first 20 million schemas) - Just a basic servlet for the service - Simple LRU caching that works on both the client and server side - Schemas are just strings which makes testing easy. The avro validation can be done with the pluggable validator. - An integration/stress test that registers a bunch of schemas and checks the answer Current patch works with jetty from command line. Theoretically it should be possible to package for any servlet container (added a context listener for that), but I haven't tried it.
10. Hey Scott, I uploaded what I have, take a look and let's discuss.
11. Looks like a good start, thanks! A couple thoughts so far: I'll split this up into a family of maven modules: || path from lang/java/ || description || | schema-repo/common | common libraries and abstractions. Ideally doesn't even depend on avro, only strings, etc. | | schema-repo/client | for the client, so that a client app can have minimal dependencies | | schema-repo/server | standalone server instance, can have dependencies that clients would not want | | schema-repo/mysql | perhaps jdbc, but there might not be much to share with a postgres implementation I'd make. | h3. ID Generation I think the API should be closer to {code} String getNextId(String source); {code} The SchemaIdGenerator should be able to look up the latest id for a source if required. I can think of a few cases that could generate a series of ids -- Zookeeper, with an atomic int kept per source, Postgres, with a sequence kept per source, and HBase atomic increment, and all of those store the latest in order to generate the next. For an md5 generator it does not need the last id, but may need to validate that there is no collision per source. I am also thinking about how to make the ID optionally be a long instead of a string, so that we can use Avro to encode it before the datum in places like HBase or Kafka. h3. Schema change validator For this interface, I also

believe that it may be simpler to simply provide the source and the new schema, that is: {code}void validate(String source, String newSchema);{code} instead of {code}void validate(String oldSchema, String newSchema){code} Assuming the validator has access to all schemas in the source. Some validators may only be interested in the delta between the new one and the latest, others may need to check against every schema. Thoughts?

12. Yeah it definitely makes sense to split the packages up to help with dependencies, your proposal sounds good. For the id generation, I think you can use a long even if it is stored as a string. I think in this case String is just a stand-in for "some bytes". The id generator i gave uses sequential integers. It just isn't captured in the type signature. WRT to the API question I think you are proposing adding the source to the API, which is definitely needed in both interfaces for the reasons you give. I would actually argue for giving the current schema as well, so that it doesn't need to be looked up twice: void validate(String source, String oldSchema, String newSchema); void nextId(String source, String schema, String currentId); Alternately we could substitute in a reference to the SchemaRepository from which you could lookup the latest schema/id, the only downside is you would need to do it twice (once in the validator and once in the id generator if you needed both). Of course if the user provides a custom implementation, all of these can be done in inline in any custom way as part of the register() call.
13. Rather than supplying a SchemaRepository in the API for a generator or validator to use, the repository can be provided when the generator is constructed if it needs it. This however creates a circular dependency -- a repository requires a generator which requires a repository. To decouple this, we can build it up by layers. For example, a ValidatingRepository is created by combining a Validator and Respository. A CachingRepository is created by combining a Cache and a Repository, etc. This further decouples the system, allowing for any combination of features to be configuration driven and each layer to be tested in isolation. For the ID generator, neither a postgres id generator, HBase id generator, zookeeper generator, nor hash generator needs to know the prior id. MySQL might (does it not have sequences? Its been a while since I used it). Anything that does can look it up -- it is likely cached or very fast to retrieve, and id generation will be infrequent. For the validator, do we assume that we are only validating the new versus the current? What if a validator needs to compare the new schema to all previous to ensure forwards compatibility? The simplest API would appear to be: validate(String source, String newSchema); All other context, such as checking against the latest version, can be encapsulated in the validator instance and passed in its constructor. We could have an abstract validator that implements the above, gets the latest, and then has an abstract validate(String source, Sring newSchema, String latestSchema); resulting in the more limited API you list above. Doing the reverse however, would create a wasted lookup for validators that do not require it.
14. I am nearing completion on the first patch attempt for this ticket. The implementation differs from Jay's first pass in several ways, but has the same core concept. * The "group" or "source" concept has been renamed "Subject" -- this is a collection of mutually compatible schemas, evolved over time according to a specific notion of compatibility. I am happy to pick another name if that makes sense. "Group" is too broad, "Topic" is taken by pub-sub systems and does not map 1:1 to that (though often does), and our engineers found "source" confusing. * The client API attains a _Subject_ from a _Repository_ and does Schema/Id lookups on this _Subject_ object, not the repository. Since many common use cases of the repository map to one _Subject_ (e.g. a collection of Avro files, a column in HBase, or a Topic in Kafka all map to one Subject), it was cleaner as a client to bind the Subject to the object in the client in code (perhaps as a final member variable) than to hold on to the Repository object and _always_ pass in the same constant subject name. * The client API is the same as the server implementation API. To implement a JDBC or HBase persistent store, implement Repository and Subject. This allows composition of layers and proxy implementations. For example, a a persistent Repository implementation can be wrapped in a CachingRepository, much like an InputStream can be wrapped in a BufferedInputStream. One of our critical use cases leverages this -- a repository proxy is constructed as a REST server that uses the REST client as a backing store, with a cache layer in the middle. * Validation is not yet implemented (working on it, some parts are stubbed out). There are at least 5 basic notions of 'compatibility' that I believe should be built-in, see the Javadoc. Validation and compatibility need to be configurable on a per-Subject basis. * Caches are pluggable and composible with storage implementations. * I did not implement a MySQL or JDBC storage layer (we use Postgres, if we build that we will contribute). Instead I wrote a simple file based repository as one of two reference implementations. The other reference implementation is an in memory repository. * I modified the REST semantics a bit, which required a "register_if_latest" variant of register to avoid race conditions when two registrations occur at the same time and validation would fail if it is order dependent. * The eventual implementation needs to support both integer and string IDs. The current implementation leaves this up to the persistence layer, but we need to expose on a per-subject basis whether a key can be interpreted as an

Integer or not. Open questions: * What should the restrictions be on Subject names? I propose that subject names are required to be valid Avro identifiers: http://avro.apache.org/docs/current/spec.html#Names (e.g. com.richrelevance.Stuff_Number-2 is valid, but 34;'xyz is not) this should mean that the names are valid file/directory names but that some escaping may be needed to map to a table or column name in an RDBMS due to '.' * Is there a cleaner way to deal with validation race conditions in the API? * This model has been proven to work and tested at RichRelevance internally when paired with Kafka and/or collections of Avro files. I have not deeply considered HBase/Cassandra/Hive/Pig etc yet. I believe the concept works broadly for any case where you want to encode a short ID prefix instead of full schema before a record. * Rather than have named parameters in the API (REST and Java) specifically for the type of validation, I think it is wiser to have an arbitrary Map<String, String> for each subject for extensible configuration. We can reserve the "avro." prefix namespace for internal use. Other tidbits: * The entire repository core does not depend on Avro at all -- it is a system for mapping a namespace of (subject_name + id) to a schema string. The only requirement is that the string not be empty. Avro (or other interpretations of the schema) is a separate concern handled in a different layer. * I have added JSR-330 annotations for some constructor Injection, and use Guice in the server component to wire it all up and launch flexibly -- the persistence implementation and cache can be configured in a properties file and a server launched. Guice does _not_ leak to any client libraries, it is only for the stand-alone server. Clients can optionally include a jsr-330 jar for use with Spring/Guice or ignore them. * Client libraries for reading/writing (id:avro_bytes) pairs with Avro and caching the avro machinery appropriately are additional things I would like to later contribute. We have started using the implementation internally and will migrate to the eventual version submitted to Apache Avro. I am targeting end of year to wrap this up. I will post the work in progress to this ticket.
15. To be clear, I am targeting today or tomorrow for the first patch and work in progress, and later this year to finish review and get community consensus for commit with all open questions addressed and no missing features.
16. **body:** Work in progress for Schema Repo attached as AVRO-1124.patch
    label: requirement
17. I have a question, mainly for the LinkedIn folks, but anyone's welcome to give their opinion of course :) How do you deal with nested schemas? In particular, how do you deal with nested schemas that are defined externally and imported into the containing schema? http://avro.apache.org/docs/current/idl.html#imports Do you version them somehow? If so, how? Do you tag them with a custom metadata field containing the version/ID of the nested schema? Do you instead always flatten the nested schemas into the containing schema instead of importing them from the outside? If there is no current support for versioning externally defined imported schemas, then is it something that's worth supporting? or should it instead be enforced to only have flattened schemas and not use imported ones? (By flattening, I do not mean to remove the nesting, but rather to include the definition of the nested schema directly in the containing schema's definition, rather than importing it.) Also, if a metadata field is defined which contains the version of a nested externally defined schema, then should the value of this metadata field be included in the schema fingerprinting algorithm? Thanks :)
18. Our setup works as follows: 1. We have a giant directory of version controlled schemas for the whole company. We have another directory called "includes" which includes any shared record type that is included in multiple schemas. 2. We always fully expand referenced types in the schemas. So if you have a type Header defined in your record and it is not found in that same file, we look in the includes directory and try to get it from there. 3. We don't use the idl since our setup predates that. 4. We send each message with a schema id which is the checksum of the fully expanded schema. 5. This means that any change to either the record or any includes effectively changes the version, but this is fine, since version changes are automatically handle when the md5 of the fully expanded schema changes. This approach has worked really well for us. The way to think about includes is just as a concise notation for the fully expanded schema.
19. Here at RichRelevance we do something extremely similar to LinkedIn, but use incrementing integer IDs rather than md5s. The number of total schemas is not likely to be so high that holding references to nested versioned bits is worth the added complexity. It has been claimed that Google's internal schema repository for protobuf is on the order of 10K schemas including all versions.
20. **body:** With HAvroBase I started out using SHAs but ultimately switched to a monotonically increasing number and look them up with their text value. As long as you have a central repository either one will work with similar performance characteristics but having a single integer is just simpler.
    label: code-design
21. Have there been any more talks on a potential release for the schema repository? We are definitely interested in this feature and are considering implementing a JDBC based repository. Before doing so we

wanted to check on the status as I see that Postgres and MySQL implementations have already been mentioned.

22. After some time off working on other priorities, I have returned to work on this. We have been running a version of this internally for ~4 months (not too different than the November patch). It is missing a few key features I am adding now to make subject configuration flexible and allow for configuration of custom or built-in validation rules. I will be updating this ticket with my latest work this week, and plan to push it to completion/contribution this month.
23. Wow that's service! Thanks for the very welcomed contribution!
24. @Scott, I wanted to follow up on your patch. I am still new to this process, will it be applied under AVRO-1124 or another one? I don't mean to rush you at all. I just want to make sure I am looking in the right place.
25. You are looking in the right place. I'll post the patch here and it will be against trunk at the time.
26. Current progress -- I have added a generic way to set / store configurations per subject, and reserved configuration keys starting with "repo.". I'm in progress of wiring up the process of identifying validators based on the configuration.
27. That's great! Thanks for the update. Will this patch include a Postgres Repository implementation as well?
28. No, we are currently using a boring (and not horizontally scalable) file based repository that will be included since it depends on nothing outside of the Java 6 JDK. An HBase one may follow that. I am also including generic unit testing tools for testing any repository implementation in a unit test maven artifact. Database implementations should be easy. I suspect that Postgres and MySQL will probably end up with differing implementations sharing some common code, due to differing feature sets underneath.
29. **body:** Most recent patch for the schema repository. This is very close, and supports configurable validators, repository implementations, and caching. What is missing: * One more maven module for Avro specific components and validators. None of the core code here depends on Avro specifically. * We need to decide what is allowed in a schema, currently a schema containing a newline will break the schema listing. * Another pass through the javadoc. I have not reviewed changes to the documentation needed due to my recent changes. * Wiki or other external documentation.
    label: documentation
30. Some changes in the last patch: There is now a SubjectConfig class paired with a Builder. This is serialized as a Map<String,String> in the format of java.util.Properties. Keys that start with "repo." are reserved. "repo.validators" stores a comma separated list of validator names. Validator names can be anything that does not contain a comma or have leading or trailing whitespace. A repository REST server can be started up with a configuration property file that contains the HTTP configuration, implementation of the repository to use, the cache to use, and validator mappings. All properties in the property file are bound using Google Guice, so additional mappings or configurations can be used by any other repository implementation or cache implementation. For example, the FileRepository and RESTRepositoryClient repository implementations so far use Guice to inject the repository instance and pass configuration information to their constructor. Therefore, it should be possible for alternate repository implementations to back the RESTRespository server provided by setting the implementation classes and their configuration needs in the property file, and placing the required classes in the classpath. Validator mappings are configured in the property file by prefixing the validator name with "validator." in the property key. For example, a line in the property file: "validator.avro.disallowUnions=org.apache.avro.repo.validators.DisallowUnionValidator" would bind the name "avro.disallowUnions" to the implementation class "org.apache.avro.repo.validators.DisallowUnionValidator", and any Subjects configured with that validator name would use that class to validate schemas. Subject configuration is currently set once at subject creation time, and if there is a race condition when creating subjects, the first one wins. I have not added the ability to set validation configuration via the Repository/Subject API yet, or built the REST API for that. At this time, if one wants to modify the subject configuration they will need to do so through a back-door such as modifying the files on disk for the file repo, or a database record in another store. Lastly, I added a method to a Subject: "boolean integralKeys()" to indicate whether the keys generated by this subject can be expected to parse as an integer. This delegates all the way through to the backing store and is not configurable through the Repository/Subject API, since implementations of the backing store are what determines how keys are generated; the contract otherwise is merely that they are Strings and unique per subject.
31. To help disambiguate where Properties are used more clearly: There is a set of properties used by o.a.a.repo.server.ConfigModule that is application scope, and required for the o.a.a.repo.server.RepoServer.main() to run. This contains the Jetty, repo, cache, and validator

configuration information that is related to the Guice-based application initialization and configuration. Properties is also used as a mechanism to serialize data in a SubjectConfig in the FileRepository implementation, but conceptually SubjectConfig contains a Map<String,String> that implementations of persistent stores must maintain per Subject.

32. A little wrinkle. I've built Avro validators for the before mentioned backwards and forwards compatibility modes. However, there is no facility in core Avro to ask "can I read schema X with schema Y"? I thought that ResolvingGrammarGenerator.generate(writer, reader) would do so, but it is lazy and only breaks when one attempts to read data (at least for missing fields without defaults. I've written a method on Symbol that seems to work to check the result of ResolvingGrammarGenerator.generate(writer, reader) to see if it contains errors, and will include this shortly, but it is poorly tested at the moment.
33. An in-progress patch of the avro validators and some tagged writing facilities.
34. My current approach to the validators is in the most recent patch (plus in progress work on writing avro messages tagged with an id prefix to a byte[]). On the validators, I am relativelly happy with them, but I have modified other code to do the "Can I read Schema X with Schema Y" work, and that is not well tested and based on an assumption of how the ResolvingGrammarGenerator works. We may want to move that portion of the work into another ticket and commit it first. I'll attach one more patch with the work I did to support this.
35. **body:** This patch adds {code} public static boolean hasErrors(Symbol symbol) {code} to Symbol.java. This seems to work but this may not be the best place for this code. It re-uses the ResolvingGrammarGenerator but perhaps instead we could make a version of that that fails fast rather than inserting error symbols. Lazy errors are nice in case your data doesn't trigger them, but in this case we want to know if there are any errors possible.
    label: code-design
36. Hey Scott, This patch is very promising. I believe I can fairly quickly provide an HBase storage, based on what we already have in Kiji. I have a few questions below. - It seems to me that validators should not be tied to the repository server. I may want to use validators in a different context. Could they be introduced independently in a separate change? - Can I add metadata to the schema entries? I'd need this to add a custom ID for a schema (I assume the existing schema entry ID is opaque and not to be interpreted). I might want to keep track of when the schema was added and by who, etc. - A Subject is currently a monotonically increasing ordered list of schemas. In some circumstances, I may want to remove a schema from a subject. - Can I change the validator of a subject (eg. to relax/tighten the constraints on a subject)?
37. * Validators outside of the repository. I agree, I'll look at how we can separate them out. There are a few parts to them -- the pure string validators that are tied to the repository and the Avro logic for representing various forms of validation. I'll think about how to separate the avro logic to be more consumable outside the repository. * Schema entry metadata -- I had not thought of this. It poses some challenges and questions. Is it immutable? Is the metadata essentially another Map<String, String> or more complicated? Immutable data can be cached, but must be set when the schema is registered. That opens up other issues related to race conditions on registration or modification. * Removing a Schema -- this introduces major issues with caching. How would one signal to all clients that a schema has been removed? Or is it ok if clients have an old mapping? If these are expected to be very rare, I suggest that they are handled not by the Repository API but by an individual implementation (e.g. Postgres, HBase, File) and require a restart of clients. If it is OK that clients cache old entries, then it may work out OK in the API as long as the back-end repositories 'tombstone' the ID to prevent it from getting reassigned to a new schema. * Changing the validator of a subject is something we should support. Only the back-end 'physical' implementations of the repository actually validate, the others delegate. As long as there is not an attempt to apply validation at an intermediate level this should be safe. One question related to this: do we want to attempt to run the new validator on the existing schemas (in order) or only apply it to future attempts to add schemas to a subject?
38. * schema entry metadata: what you suggest seems appropriate, I believe an immutable Map<String,String> is good enough; mutable data associated with a schema should probably be handled externally. * removing a schema: subjects are mutable, as monotonically growing lists of schemas. Does the caching layer already handle such subject updates? * updating the validator: it feels to me that the validator should always successfully run on the existing list of schemas in a subject. If the validator is "last 3 schemas must be backward and forward compatible", as a user of the subject, I may assume the currently existing last 3 schemas are backward and forward compatible.
39. * schema metadata: There is one race condition to consider -- currently subject.register(foo) is idempotent and also never fails unless there is a schema validation failure. Two users simultaneously registering the same schema end up with the same schema/id pair -- both fail or both succeed and get the same result. If we tag metadata along with it, then two concurrent registrations with the same schema but different

metadata might occur. The actions are still idempotent and the two users get the same result, but only one will have the metadata expected set. I will still have register() never fail outside of validation, but the schema metadata is not guaranteed to be what the user requested when there is a race condition -- the same thing happens with subject creation now. If metadata is immutable, it can be cached and part of the SchemaEntry. If it is not, it will need to be uncached or have a TTL, the latter I would like to avoid due to complexity. * In a subject, schema/id pairs are only added. The caching layer is free to assume that once an id/schema relation exists, it will forever, there is no propagation of updates. This is the sane thing to do -- once a datum has been written with an id, the schema tied to that key should be kept forever. If a schema could be removed, we would need to check the repository for every record or have a TTL in the cache. It would be easier to support 'deactivating' a schema/id pair so that it is not returned when scanning all the active schemas in a subject, or with validation, but can still be found by looking it up. Can you describe the use case for deleting a schema? Under what conditions would you want to do so? * I have opened https://issues.apache.org/jira/browse/AVRO-1315 to cover the avro schema validation components that live outside of the repo projects. Please provide feedback, Thanks!

40. * schema metadata: I believe the race condition is avoidable if I use registerIfLatest(). Am I correct here? * deleting a schema from a subject: the use-case is, I want to enforce some level of compatibility on all the schemas in a subject; at some point I realize one schema in the subject prevents me from moving forward in a sane way; I don't want to relax the compatibility constraint, but I am willing to actively deprecated this one schema (eg. by rewriting the records according to another schema from the subject), so that I can exclude this schema from the validation. The schema does not necessarily need to disappear from the subject, it may stay there, but I want to exclude it from the validation check. Does this make sense? * AVRO-1315: awesome, I'll check this tonight or tomorrow!
41. Is anyone still pushing on this one? One question for those using incremental IDs. Do you ever move serialized data from one environment to another, and if so do those environments each have their own schema registry or is there a shared one? It seems if each environment has its own schema registry then data may not be portable if the assigned ID sequence was not identical in each environment.
42. Yes. I have quite a bit of work outstanding on this to finish and submit for review. But I'll be on vacation for 2 weeks. Re: Incremental ids: The ids don't have to be incremental, that is an option that is up to the repository implementation. They can also be an arbitrary string. Your repositories probably will not map directly to environments, but to the data. If you are sharing data across environments, you will share repositories (or clone them) with the data.
43. **body:** Thanks for the update, Scott. I'm excited to see this work get in, it looks great. I realize the IDs are pluggable, but I would love to avoid 8 byte+ overhead, especially for small records, so I was considering the incrementing IDs. However, this portability issue occurred to me so I was curious to hear if those already using an incremental ID had dealt with it. We envision having a schema registry running in each environment alongside the other services we provide there. I.e. one in production, one in staging, one in each development or integration environment. Then the applications could talk to the registry in their environment. I'd be nervous having a shared service that production, staging, integration and development environments are all using. {quote} If you are sharing data across environments, you will share repositories (or clone them) with the data.{quote} That sounds reasonable, unless you're adding data to an environment that has already mapped the same IDs to different versions of the schema. What about another possibility of having a schema ID directly as a metadata entry in the schema? Then perhaps the build system could be increment it whenever the schema is revved.
    label: code-design
44. Schema id generation pluggable, so there are many options. The _only_ requirement is that within a subject ids are unique and correspond to unique schemas. We also have staging/prod/qa/dev environments. There is a repo for each, but when qa gets its data snapshot from prod, we also clone the repo. For dev/staging, we have a kafka mirror that is exactly the production data. Both of these environments access the prod repo read-only. In fact, even in production, most subjects are read-only to all applications. Operations has to add a new schemas for a release. This is akin to operations executing SQL scripts to do DDL prior to a code push. Having applicaitons 'automagically' update sql schemas or push avro schemas can lead to accidents, unless the security model is implemented properly.
45. Hi Scott, A little bump for this ticket :) I've been able to apply the April 1st 2013 patch on top of the 1.7.5 avro release after doing some minor tweaks. I've then been able to get the shaded jar located at bundle/target/avro-repo-bundle-1.7.5-withdeps.jar running. I'm wondering if I should also apply the two April 5th patches, and possibly the AVRO-1315 patch as well, on top of all that? It'd be nice to get this committed into trunk (or into a development branch) sooner than later, like Doug initially suggested, so we don't have to mess around with several patch files which end up breaking on the tip of the trunk as avro releases move forward. I'll be looking into integrating the usage of the schema repo into our various

Kafka publishers and consumers over the next little while. I'd be curious to know how stable you think the current API is? Thanks :) !

46. [~felixgv] Mind sharing your tweaks or updates? I'm also looking to get this available, even if it doesn't have the automatic validation yet.
47. **body:** All: I apologize for the long delay. What we have used in production for about a year is very close to what has been in this ticket the whole time. I have never considered it complete for a few reasons. I have been "close" to done with this for some time now but swamped by other responsibilities and what is currently in use has been good enough for now, but it won't be for long. The latest changes however, would significantly impact some of the API with respect to how the schema repo manages validation and compatibility. This would be significantly more flexible for interfacing with other systems. It boils down to the following observation: It appears that all notions of schema compatibility share a common form. The previously discussed "forwards compatible" or "N + 1" compatibility are all flavors of the same set of constraints. In any set of schemas you wish to consider for compatibility (a "Subject" here), at any given time you have a subset of these schemas that you wish to be able to read with, a subset you must be able to read from. You may have some that you neither wish to read from or write to but must keep the mapping of the id. The way to represent this is to have a "read" state and a "write" state per schema in the subject. The read state has two possible values, naming help needed: "reader", "not_readable" The write state has three possible values, naming help needed:, "writer", "written", "not_writable" The constraint of the system is that all "reader" schemas can read all "writer" and "written" schemas, per subject. A schema can transition either state, one at a time, leading to pair-wise testing of "schema X can read Y": * A schema transition from "not_readable" to "reader" succeeds only if it can read all schemas that are currently "writer" and "written". * A schema transition from "reader" to "not_readable" requires no pairwise schema validation, but some other pluggable validation may apply. * A schema transition from "not_writable" to "writer" or "written" requires pairwise validation that that schema can be ready by all current "reader" schemas. * A all other schema write state transitions do not require pair-wise schema validation, but other pluggable validation may apply. The write state has three possibilities because it is important to differentiate between the case where you allow new records of this type to be written ("writer") from one where you wish no new records to be written, but the data store still has values with the schema present ("written"). Every compatibility scheme can fit in the above. Single-reader, multiple writer. Single writer, multiple reader. N+-1 compatibility. Full cross-compatibility. The above is significantly more flexible than the early proposals on this topic, but will require changes to the REST interface. Loading data from the old, into the new, will be fairly simple however -- some curl commands and bash scripts will do it. , and enforce the constraint that all schemas that have their "can read with" state set must be able to read all schemas that have their write state in "can write with" or "must be able to read"
label: code-design
48. Maybe I am missing something, but stated in the original ticket it read that the RepositoryServer should be load balanced. The thing that appears to be an issue for me, but maybe I have a misunderstanding, is that each Producer must register the schemas it is using with the SchemaRepository. If this is true, and the RepositoryServer is load balanced, it means schema IDs will slowly diverge between RepositoryServers if an increment count is used for IDs. This issue is solved with MD5s, which is what you stated as being done at LinkedIn. Another issue I foresee is if load balanced, and that particular schema only got registered with one instance of the RepositoryServer but not with another instance, and the Consumer asks the server it hasn't been registered with, then it will fail. How do you guys manage or mitigate these risks? Do you first run something to register schemas with all RepositoryServers? Does the job just get processed again and *hope* that it doesn't hit the same RepositoryServer again?
49. Ah, I missed the part about the pluggable key value store, so the onus is put on another technology to handle the consistency and the Repository just needs to handle the distribution.
50. Getting back into this after getting pulled into other projects for a while... [~davelatham], I imagine you probably figured this out on your own by now (sorry for not replying earlier), but in case it can help you or anyone else, here's a GitHub repo containing the release of Avro 1.7.5 with the 01/Apr/13 patch applied onto it and small tweaks so that the build works: https://github.com/mate1/avro/tree/release-1.7.5-with-AVRO-1124 To everyone else, I'd like to get your thoughts on something. There are certain avro schemas that we use across many Kafka topics (1:M relationship between schema and topics). I would like to benefit from the facilitated evolution capabilities of the schema repo, but I'm not 100% sure of the best way to proceed. I would like to avoid: 1. Having to register the same schema (and each further schema evolutions) into many subjects. 2. Having to externally manage a mapping of "Kafka topic" => "subject registered into the repo". I have considered prefixing each of my Kafka message payloads with not just the writer's schema ID, but also a subject ID, but I have though of many caveats to this approach,

which are probably not worth enumerating here, for the sake of brevity. Another possibility would be to introduce the concept of a SubjectAlias. The way it would work is that you would register a SubjectAlias with an aliasName and a targetName. If the aliasName already exists, or if the targetName does not exist, the operation would fail. Afterwards, any lookup for the aliasName would return a DelegatingSubject containing the Subject referenced by the targetName of the alias that was looked up. This change seems clean and not too intrusive, and also wouldn't require encoding both subject ID and schema ID in my message payloads. But perhaps there are problems to this approach that I haven't thought of. Do you think this approach makes sense? And would it be worth contributing back into the main schema repo code? Is this (multiple topics sharing a single set of compatible schemas) something that other organizations have had to deal with? Hopefully I've properly explained what I'm trying to achieve. All feedback would be appreciated! Thanks (:

51. Hi everyone, Integrating this schema repo in our stack has been a back-burner project of mine for a while now, so I haven't been able to make progress as fast as I wanted. Anyway, I finally found some time to get back into it again. I have a question: Why is this project completely serialization-agnostic? I understand that there is value in having a general solution, but it seems like it would be useful to have a standard Avro-specific implementation, at least on the client-side. Basically, something like what Camus is doing here: https://github.com/linkedin/camus/blob/master/camus-schema-registry-avro/src/main/java/com/linkedin/camus/schemaregistry/AvroRestSchemaRegistry.java https://github.com/linkedin/camus/blob/master/camus-api/src/main/java/com/linkedin/camus/schemaregistry/SchemaRegistry.java?source=c It's very simple, but it seems like the type of stuff that every project would need to re-implement on their own. Of course, you could use the schema repo code provided here as is if you're okay with re-parsing the Avro Schema on every invocation of the repository, but that seems pretty wasteful, especially in high-throughput environments... Would there be any desire to integrate this kind of Avro-specific wrapper for the caching client implementation?
52. Hi guys! If I want to use the repository, what patches are useful? 1--> AVRO-1124.patch 01/Apr/13 18:52 2--> AVRO-1124.patch 26/Nov/12 20:32 3--> AVRO-1124-can-read-with.patch 05/Apr/13 14:26 4--> AVRO-1124-draft.patch 31/Aug/12 20:00 5--> AVRO-1124-validators-preliminary.patch 05/Apr/13 14:18 From what I understand, 1 is the newest version of 2, so 2 is useless/deprecated? 3 is also in AVRO-1315 so useless too since I'm using AVRO-1315 4 is a draft(?) so deprecated too? 5 is some validators so that one is useful? So I should need 1, 5 and AVRO-1315?? Thank for your help!
53. Hi Francois, I noticed you applied these patches, and the code tweaks from Felix in mate1/release-1.7.5-with-AVRO-1124. I was able to pull and build it perfectly. Thanks a lot! For others, here's the branch from trunk with patches for both AVRO-1315 and AVRO-1124 applied: https://github.com/mate1/avro/tree/from-98ec5f2a172391cb5dfa7b4d85f39065bae22754-with-AVRO-1315-and-AVRO-1124 BTW, are we expecting AVRO-1124 to end up in an 'official' AVRO release or will this constantly be an open issue with a set of patches? I don't necessarily mind this way; we have figured it out and gotten it working. I'm just curious what the end plan is for it. I'd be glad to help out in whatever way I can. Thanks again everyone. Thunder
54. Hi Felix, was just reading through comments, and saw this from you. We are going through the exact same thing right now as well. {quote} There are certain avro schemas that we use across many Kafka topics (1:M relationship between schema and topics). I would like to benefit from the facilitated evolution capabilities of the schema repo, but I'm not 100% sure of the best way to proceed. I would like to avoid: 1. Having to register the same schema (and each further schema evolutions) into many subjects. 2. Having to externally manage a mapping of "Kafka topic" => "subject registered into the repo". {quote} We also are trying to avoid this. We are currently planning to take a topic naming convention approach where we combine the subject name (avro class FQN) with a "topic suffix" when naming our topic. So a topic would be named: '<subject_name>--<topic_suffix>' where we don't use the -- delimeter in either subject name or suffix. I think this avoids both of the above issues, as well as not requiring each message to have a subject ID. It does however add complexity to all consumers on how to "parse" topic names. {quote} Another possibility would be to introduce the concept of a SubjectAlias. The way it would work is that you would register a SubjectAlias with an aliasName and a targetName. If the aliasName already exists, or if the targetName does not exist, the operation would fail. Afterwards, any lookup for the aliasName would return a DelegatingSubject containing the Subject referenced by the targetName of the alias that was looked up. This change seems clean and not too intrusive, and also wouldn't require encoding both subject ID and schema ID in my message payloads. But perhaps there are problems to this approach that I haven't thought of. Do you think this approach makes sense? And would it be worth contributing back into the main schema repo code? {quote} I think this would be a fine approach, and that would simplify our kafka consumers to not need to "understand" the convention we came up with. It

would also free you to use any convenient topic name for any subject schema without having to adhere to that convention on naming. Have you had any progress or other thoughts on this issue since January? I realize I'm a little late to the party :) Cheers, Thunder

55. Hi Thunder, Disclaimer: I don't work for Mate1 anymore, so what I'm going to say might be out of date by now. The Mate1 guys will need to chime in for the latest state of their work on this... I did not end up coding SubjectAliases into Avro proper because there didn't seem to be any interest from the OSS community and I had limited time to build this completely generically. Mate1 had a strategy that is kind of similar to the one you're describing. We had arbitrary topic names, which could be dynamically appended with certain suffixes (like "__PROCESSING_FAILURE", "__DECODING_FAILURE" or whatever). Those dynamically created topics could then be re-processed later on, as convenient, and would contain the same schema as the topic they derived from. In our Camus decoders, we hard-coded some topic alias resolution right there in the code (since the amount of suffixes was limited in our case). There were talks of porting that topic alias resolution logic to our schema-repo-client implementation (https://github.com/mate1/schema-repo-client/), so that it is more conveniently available to all Kafka consumers (not just Camus), but that didn't end up happening before I left. So essentially, we went for option 2, above. For an organization that has more numerous/diverse suffixes, that strategy would probably not be ideal, but for small amounts of suffixes (or prefixes), it was deemed acceptable. Hopefully, that sheds some light (: -F
56. **body:** > BTW, are we expecting AVRO-1124 to end up in an 'official' AVRO release or will this constantly be an open issue with a set of patches? I'd love to see this in a release sooner rather than later. Since it's new functionality there should be no compatibility issues. All it takes is someone to declare a particular patch ready to be committed, and for one or more other folks to endorse that. We also need to have some confidence that, even if it's incomplete, the public APIs it exposes can be supported compatibly as functionality is improved.
    label: code-design
57. The patch from April 1st (I think its the latest) seems based on Avro 1.7.5, it will need a rebase on Trunk. I think its just a matter of merging pom.xml.
58. Watch out. The April 1st 2013 patch has a bug in the file-based repo where multi-line schemas get all lines dropped beyond the first one. This bug is fixed (and unit-tested) in the Mate1 GitHub repo posted above, so I'd advise creating the next patch off of that. The pom files probably need to be modified either way.
59. Felix, thanks for the headsup. I'm trying to pluck a patch out of the Mate1 repo, but it looks like there's no clean commit of just the Repo patch, its mixed with other stuff? If I'm wrong, please tell me which Mate1 commit to use for the diffing. Otherwise, I'll rebase the April 1 patch and try to selectively apply later Mate1 fixes on top.
60. **body:** Felix and others interested. More questions: 1. There was a lot of discussion on the implementation of "schema includes" (i.e. imports) within the repo. It looks like it was never implemented. That is, there's nothing here that will support compilation of schemas where I refer to a repo+subject when importing another schema. Correct? am I missing anything? 2. I'd like to rebase and nag everyone for commit, but without the additional validator patch. There was a lot of discussion around validators, it doesn't look like it ever got resolved, and I don't see it as mandatory going forward. Does that make sense? Was validation issue actually resolved somewhere and I missed that? Is validation super essential? If we decide to go forward with no validation, I'll remove the promise of validation from the registration API. We can always add "registerWithValidation" later for those who like safety. 3. Few Jiras will probably need to be opened immediately when committing. The first is to add a user-guide :)
    label: requirement
61. Hi Gwen, I don't think that there is a single commit in the Mate1 GH repo that atomically contains everything we need (if that is even what you're looking for... not 100% sure I understood your request correctly). If you want to rebuild the patched Avro repo yourself, then this is the (atomic) commit that contains the multi-line fix I was talking about: https://github.com/mate1/avro/commit/a3960590177e08b3586584cf288b1d4ba9ca1572 Alternatively, I've put together a branch that contains the very latest trunk (as it was on the GH mirror, a few minutes ago), with the M1 fixes on top: https://github.com/FelixGV/avro/tree/AVRO-1124 From that branch, I can create a patch against trunk using `git diff trunk > AVRO-1124.patch` (which results in this: https://gist.github.com/FelixGV/7f0978dfa90fe4430c24), but that's not the precise format Apache wants (Apache wants the svn diff...). I don't have time to fiddle with this right now, but maybe you know how to generate the patch properly? On to your other points: 1. Imports are not supported. The recommended way of using the repo is to flatten schemas so that nested schemas are completely contained in their parent schemas. 2. I think adding validators later on does not break the contract of the current register

APIs. Validators should be pluggable and the register API should be allowed to fail for arbitrary reasons (i.e.: validators could be tied to custom in-house policies for schemas, not just backward/forward compatibility). Validation is important, but I think there is still value in releasing the repo without any built-in validators in the first release. 3. Agreed.

62. Took Felix's gist, applied on clean checkout of Trunk (no conflicts). Created an SVN patch and attaching it here. I haven't tested functionality yet, except making sure the project still builds and unit tests pass. However, considering that its: 1) All new code, therefore no risk of regression 2) There's wide agreement that the API is reasonable (evidenced by the fact that multiple organizations currently using it) 3) After 2 years of discussing and using the code we haven't found a single reason to make non-compatible API change I recommend committing it, and continuing the improvements in separate issues. Felix, please double check that I didn't drop anything on the floor by mistake and do another +1 :)
63. Hi Gwen, Your latest patch looks good to me. I suspect I might not be entitled to say +1, but if I am, then +1. -F
64. From a brief look, patch looks fine. Should we get rid of: {code} ./server/src/main/java/org/apache/avro/repo/server/RepositoryServer.java:88: sleep(3000); //TODO DELETE!!! FOR DEBUG ONLY {code} Regarding new validators : From what I understand, the interfaces in this patch provide a model where one can validate a sequence (timeline) of Avro schemas. Do you think this could later be extended to support a model where a subject manages a set of reader schemas validated against a set of writer schemas instead (with no notion of backward/forward)?
65. LOL, good catch [~kryzthov]. There's a lot of discussion about different notions of compatibility, but the interface simply requires implementing: void validate(String schemaToValidate, Iterable<SchemaEntry> schemasInOrder) throws SchemaValidationException; This will allow you to validate a single schema against a set. Validating a set against another set seems like a fairly trivial extension. Note that the repository doesn't have a notion of reader schema vs. writer schema. Is this something you need for your use-case? Note for [~felixgv] and others: The original issue (from Jay Kreps) discusses not just a repository, but supporting the use of repository identifier inside Avro serialization, instead of embedding the entire schema. The goal, as I understood it, was to support smaller objects with less overhead for streaming applications. This patch only provides the repository, but each Avro object will continue to contain the entire schema. We'll probably need another follow up issue for that.
66. Compatibility between readers and writers is something we would indeed benefit from. A {{Subject}} is currently defined as an ordered collection of mutually compatible Schemas, which is fairly restrictive and does not allow for deprecation/removal of schemas. Supporting the reader/writer use-case requires dropping the "mutually" : some schemas may only be used for reading purposes, while others are or were used for writing purposes. The reader/writer use-case would also allow the removal of a schema from a Subject. By the way, I uploaded the patch on https://reviews.apache.org/r/23293/ if folks want to look at the diff and comment more easily.
67. [~gwenshap], this patch is not the whole story. It is only the repo server and a basic client to interact with it. What's missing (at least, in open source, as I suspect everyone has their own custom internal implementation of this) is the versioned Avro encoders and decoders that make use of the schema repo under the hood, and provide easy to use APIs for dealing with it properly. I believe this functionality definitely deserves to be open sourced as well, but it could be done as later tickets... [~kryzthov], I think you're right that the use case you describe doesn't seem well supported at the moment in the current APIs. There are arbitrary key/value pairs that can attached to Subjects via their SubjectConfig (but not to individual SchemaEntries). This MIGHT be usable to maintain a list of deprecated/read-only IDs for a given Subject, in which case the validate function might need to be passed a reference to its Subject/Config. Or perhaps we should add explicit support for it, by adding new (specific or arbitrary) fields in the SchemaEntry class... Do you have a proposal of how the API should be extended to support this use case? One thing to keep in mind is that whatever data we choose to add needs to be supported by all Repo implementations (right now there is only file-based and in-memory, but still, it would be good to keep the data structures somewhat simple, so that we can easily add new storage implementations).
68. > Was validation issue actually resolved somewhere and I missed that? Yes, it was committed in AVRO-1315. It was split out since it is more general than the repo service. The patch for this issue should be refactored to use those interfaces (should just be a package change). Also, I noticed that Gwen's latest patch changes Snappy Java 1.0.5 to 1.0.5-M3, which looks like a mistake (remnant from an earlier patch?).
69. Part of the API implemented in Mate1 patch is "get ID by subject + schema". Reverse lookup of sorts. I want to remove support for this API. Reasons: * Current implementation does not work * I can't figure out why we'll need it We can add it later if turns out its actually important. Thoughts?
70. Never mind my previous comment. I thought it doesn't work - didn't notice its a POST rather than GET.

71. [~tomwhite] : the milestone snappy version does seem like a mistake. [~gwenshap] : that API definitely does work, and it's essential when using auto-incremented IDs, rather than schema hashes. You need it to know what ID to encode along with your payload on the publishing side of your pipeline. Schema hashes can instead be inferred, but have other tradeoffs (i.e.: size, collision, obfuscated ordering semantics).
72. Question for [~felixgv] or [~scott_carey]: One of the APIs implemented is GET {subject}/integralKeys Can you explain what are the integralKeys? I see the setters and getters, but no documentations on how we expect to use them.
73. Attaching patch with correction to issues found by [~kryzthov] (sleep statement left from debugging) and [~tomwhite] (wrong snappy dependency).
74. **body:** [~gwenshap], Regarding integralKeys, there is an earlier comment from [~scott_carey] in this thread which explains it: "Lastly, I added a method to a Subject: "boolean integralKeys()" to indicate whether the keys generated by this subject can be expected to parse as an integer. This delegates all the way through to the backing store and is not configurable through the Repository/Subject API, since implementations of the backing store are what determines how keys are generated; the contract otherwise is merely that they are Strings and unique per subject." It would be good to include this in the javadoc of the Subject abstract class, because there currently seems to be no way of figuring this out from the code. For now, all provided Repo implementations are hard-coded to always return true on that function, so it is a bit pointless at the moment, but I think the API makes sense to have (especially in the context of supporting hash schemas for IDs, not just auto-incremented integers). What do you think? Should it be documented and left there, or removed for now?
    label: code-design
75. [~gwenshap], there's an unused static import of java.lang.Thread.sleep in schema-repo/server/src/main/java/org/apache/avro/repo/server/RepositoryServer.java that causes the audit check to fail.
76. I noticed that all of the Named injection points related to repository implementations are prefixed with "avro." except for REPO_CLASS and REPO_CACHE in server/ConfigModule, was this intentional? The example configuration properties file (bundle/config/config.properties) also refers to them as "avro.repo.class" and "avro.repo.cache", which causes it to not work.
77. Has anyone though about using a maven repo as a avro schema repository? It would be nice if a java project that is using a particular schema can simply add it as a <dependency> Schema dependencies could be maintained nicely as well. (Schemas would be developed in a schema maven project, taking advantage of versioning, dependencies...) What about exposing this Schema Service also as a Maven Repo transparently?
78. **body:** Hello, Little update on this. [~gwenshap]: I added the javadoc for integralKeys. Also fixed some more javadoc that was outdated or missing, but I haven't checked everything, so it might need some more javadoc improvements still. [~busbey]: I removed the unused import and fixed the inconsistent config names. Thanks (: Besides that, I've also added a (very terse) README and run script for launching the schema repo server. Attached is a new patch based off of 1.8.0-SNAPSHOT. Do you guys think this is good enough for committing into 1.8.0 ? If anyone is interested in an already patched git branch, I got these: https://github.com/FelixGV/avro/tree/release-1.7.7-with-AVRO-1124 https://github.com/FelixGV/avro/tree/1.8.0-SNAPSHOT-with-AVRO-1124 Thanks, -F
    label: documentation
79. [~zolyfarkas], I don't think anyone suggested that yet. Are you saying every Avro record would be a different Maven artifact? In any medium- or large-size project, that would result in a huge number of artifacts, which would then result in huge dependency/build files. While it could work, I'm not convinced that it's a sensible use of Maven. Let me know if that's not what you meant or if I'm missing something. -F
80. Hi Felix. I agree, that versioning every avro record separately would not be practical. (although doable) Let me give an example of how I envision this with a example for a rest service: /demo-service /demo-service-schema (versioned and released independently, will contain all avro schemas for this sample rest service) /demo-service-server (depends on schema) .... /other-app (depends on schema) All schemas will reside in the demo-service-schema project and will be built and versioned as a standard maven project. (together) schemas in demo-service-schema can depend on other schemas in other "schema projects" and this will be described with the maven dependency mechanism. Part of this "schema build" the java, c# record classes are generated, and the schema files are packaged and everything uploaded to maven repository. I managed to get this running without creating any special maven plugin... But I can see handy having an specialized plugin to verify schema compatibilities, enforce doc fields to be present... I also wrote a schema retrieval client that will retrieve any version of schema from a maven repository. (with local persistent caching on the file system) Some advantages to this approach: 1) dependencies are clear,

and easy to trace. (schemas are in one place, no more emailing around) 2) classes are generated once and reused by all projects. 3) version names are really whatever the dev wants, but major.minor works well 4) no custom service needed, just a plain apache server will do. 5) For in development schemas the maven SNAPSHOT fits pretty well let me know what you think.

81. Hi [~zolyfarkas], I believe what you describe is a good solution for build management. In fact, I think lots of companies/projects already operate like this (minus the automated schema compatibility check, which could be handy to have as a maven task, indeed). I'm not sure how related this is to the effort described in the current ticket, however. It may be orthogonal. I think the main goal of the schema repo is to provide easy access to sets of ID=>schema pairs grouped per subject and to act as a gatekeeper for the creation of these pairs. Lumping all schemas in one versioned artifact makes sense from a build perspective but seems inconvenient from a programmatic standpoint when fine-grained look-ups might be desired. Say, if project A wants to use v1 of topic_x and v2 of topic_y, whereas project B wants the opposite (v2 of topic_x and v1 of topic_y), how does that translate to the single big artifact Maven model? Can things easily evolve independently like this? That being said, I may not be grasping the value right now, but nothing prevents someone from taking this code and exposing a Maven-like HTTP interface form it, and/or using a real Maven repo as its backing storage implementation.
82. Hi Felix, it relates to this ticket by addressing the same motivation: "It is nice to be able to pass around data in serialized form but still know the exact schema that was used to serialize it." What we implemented is to send the version of the schema(maven version) along with the avro messages. (HTTP header) at deserialization time this version is used to retrieve the writer schema from maven repo, without the need of a custom service.
83. You need a "custom service" because the client may not have the schema. Sending the version is great if your client happens to have copies of all of them. Without a service to vend those schemas though you can't send a message to another service that doesn't have the latest ones yet. On Thu, Oct 16, 2014 at 7:41 AM, Zoltan Farkas (JIRA) <jira@apache.org>
84. Hi Sam, Yes you need a service, but that is what the maven repository is. (you can use nexus, artifactory or plain old apache HTTP) as I said: "at deserialization time this version is used to retrieve the writer schema from maven repo" The client we implemented caches the schemas locally(persistent+ram) , but when a message is received with a version that it does not have, it will go to the maven repo and retrieve that schema, cache it and use it.
85. I would never use the maven repo from production. YMMV. ---Sent from Boxer | http://getboxer.com [https://issues.apache.org/jira/browse/AVRO-1124?page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel&focusedCommentId=14173861#comment-14173861] Zoltan Farkas commented on AVRO-1124: ----------------------------------- Hi Sam, Yes you need a service, but that is what the maven repository is. (you can use nexus, artifactory or plain old apache HTTP) as I said: "at deserialization time this version is used to retrieve the writer schema from maven repo" The client we implemented caches the schemas locally(persistent+ram) , but when a message is received with a version that it does not have, it will go to the maven repo and retrieve that schema, cache it and use it. -- This message was sent by Atlassian JIRA (v6.3.4#6332)
86. Sam, a maven repo is nothing special, and we have a separate instance for this purpose... I see no reason for not using it from production... Instance is a plain apache server which I would argue is faster and more reliable than a java REST service... By caching schemas and persisting locally, our services can weather outages on this instance pretty well....
87. Ok, so, if what you are suggesting is that your solution is that URLs map to schemas based on the version, I think we are all in agreement. Let's specify what those URLs look like and a way to add new schemas (PUT?) and we have defined a REST service that does what we are looking for in this issue. I suggest not using the SC version though and instead canonically hashing the schema and using that hash in the URL. On Thu, Oct 16, 2014 at 10:22 AM, Zoltan Farkas (JIRA) <jira@apache.org>
88. Hello, I talked with [~gwenshap] last week during Hadoop World and she made me see another point of view. I now think this project belongs outside of Avro. It has been built as a serialization-agnostic repository, so it is not tied to Avro at all. Furthermore, there's only a filesystem based storage backend at the moment, but I'm working on a Zookeeper backend (for HA) and other people have expressed interest in building RDBMS and HBase backends. It seems wrong for the Avro project to start depending on all these extra things just because of the schema repo. Perhaps the schema repo belongs in another project like HCatalog. That being said, I think being accepted as part of a bigger project should not be an impediment to developing the schema repo. Developing as part of a patch in a JIRA ticket or even as a fork of another project on GitHub is not a viable solution long-term. I'm going to pull the schema repo out of Avro and make it a standalone project so that it can be iterated on more quickly. If any bigger project

wants to pull it into itself, they can do so whenever they want, as it's all Apache licensed (: I'll keep you posted. Please let me know if you have any comments or concerns. -F

89. Hi [~felixgv] and others, I've been watching this thread for some time as over in Apache Gora (0) we are very interested in schema evolution... this ticket tickled my fancy and I really wish I had discovered it earlier. bq. Furthermore, there's only a filesystem based storage backend at the moment, but I'm working on a Zookeeper backend (for HA) and other people have expressed interest in building RDBMS and HBase backends. If anyone is interested in implementing a backend agnostic (as far as datastore support goes in Gora) implementation of this service, then please get in touch with me directly or alternatively over in dev@gora. I would be very willing to put time into this effort. Thank you (0) http://gora.apache.org
90. Hi [~lewismc], The code in this ticket is already backend agnostic. You configure it with a file and it dynamically injects the correct backend implementation at runtime. For now, the only provided implementation uses the local filesystem, but other backends should be easy to add. -F
91. Hi [~felixgv] thanks for clarification.. the patch is a beast so I missed the key value config file {code}lang/java/schema-repo/bundle/config/config.properties{code} Thanks for the corrections here, I must say however that my suggestion still stands :) if anyone is interested in implementing a Gora backend for this which would provide access to a number of underlying storage options then please get in touch. There could potentially be a much less investment in actual code writing if this were done in Gora instead of writing an HBase one, then a ZooKeeper one, etc. Thank you very much for the context [~felixgv]
92. **body:** [~felixgv] I, too, implemented and deployed a version of this patch backed by ZooKeeper / Curator (The zookeeper protocol is not for mere mortals, at least not for me :-)). I also massaged the actual transport layer into an AVRO-IPC implementation (vs pure Jersey REST), making it easier to generate low-level polyglot clients to bind to the avro-schema registry/service. Granted, it is arguable how much Avro-IPC vs raw http will get me. Time will tell. I look forward to seeing this work be promoted to RESOLVED state whether it is here in AVRO or some other project where it might more closely align.
    label: code-design
93. [~felixgv] If you're pulling things out of Avro, consider moving the service implementation over to [Kite|http://kitesdk.org/]
94. Hi everyone, The standalone repo is here: https://github.com/schema-repo/schema-repo There is a stable 0.1.0 release here: https://github.com/schema-repo/schema-repo/tree/0.1.0 There is a mailing list here: https://groups.google.com/forum/#!forum/schema-repo Having it separate should make it easy to build and integrate into whatever other project needs it, be it Kite, HCat or something else. [~lewismc], if you want to take a stab at creating a Gora backend, please go ahead! [~neoword], if you want to contribute your ZK/Curator backend, please go ahead! [~busbey], if you think Kite can benefit from the repo, please don't hesitate to integrate it. I'll handle pull requests for now, but it would be good if others helped with that in the future. I'll try to setup a GitHub-based maven repo to distribute the artifacts more easily ASAP. I suggest moving further discussions to the GG mailing list, since the single thread of this ticket is starting to be pretty chaotic with the half dozen discussions going on in parallel. Thanks! -F
95. **body:** Fantastic, Github repos, starred, forked and I will begin working on this once I get through my TODO list of TODO's ;)
    label: requirement
96. Is there any update on this?
97. I for one, redact my previous note on contribution of a ZK/Curator backend, since [Confluent|http://confluent.io] now provides a perfectly good solution to an Avro-Based Schema Registry. If you are looking for a good "off-the-shelf", "out of the box" avro schema registry solution, check out [Confluent's Schema Registry|http://docs.confluent.io/current/schema-registry/docs/index.html] -- it "just works". And the backend, well, its kafka itself! Full disclosure. I do not work for Confluent. Just a big fan of tech that "works". :-) Given, that, I wonder what the life-cycle of this JIRA ticket is in 2016+ given that [Confluent's Schema Registry|https://github.com/confluentinc/schema-registry] exists.
98. I have recently published an example of what I described in earlier comment (using a maven repo as a schema repo REST service). With this approach, you get more than a schema repo, you get a validation/versioning/release system... Example is at: https://github.com/zolyfarkas/avro-schema-examples any comments welcome.