Kafka Installation and Disaster Recovery Documentation pr(and some useful Kafka cli examples)

What is Kafka?

… Kafka is a distributed publish-subscribe messaging system that is designed to be fast, scalable, and durable.

Like many publish-subscribe messaging systems, Kafka maintains feeds of messages in topics. Producers write data to topics and consumers read from topics. Since Kafka is a distributed system, topics are partitioned and replicated across multiple nodes.

Messages are simply byte arrays and the developers can use them to store any object in any format – with String, JSON, and Avro the most common. It is possible to attach a key to each message, in which case the producer guarantees that all messages with the same key will arrive to the same partition. When consuming from a topic, it is possible to configure a consumer group with multiple consumers. Each consumer in a consumer group will read messages from a unique subset of partitions in each topic they subscribe to, so each message is delivered to one consumer in the group, and all messages with the same key arrive at the same consumer.

What makes Kafka unique is that Kafka treats each topic partition as a log (an ordered set of messages). Each message in a partition is assigned a unique offset. Kafka does not attempt to track which messages were read by each consumer and only retain unread messages; rather, Kafka retains all messages for a set amount of time, and consumers are responsible to track their location in each log. Consequently, Kafka can support a large number of consumers and retain large amounts of data with very little overhead.

The above text is from this excellent intro to Kafka:
http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners/

Also:

Kafka was built to serve as a messaging service in "cloud" type environments. In a "cloud" environment, Kafka and Zookeeper make it possible for microservices, big data services, and applications (like Solr) to start up and get config data from a central source (ZooKeeper) and get all the messages they need (Kafka) -- even if the original message event was days ago.

In some cases, Kafka actually serves as THE backup datastore. Similar to just saving all the database logs and replaying them into a new database in order to get back to the correct state for a certain date.

This makes bringing up and taking down these applications more "modular" and straightforward, since a new server with some application on it can be brought up, configured, and then consume all relevant messages from Kafka before going online in Production. This can be automated so that IT no longer has to worry about the nuts and bolts of such work under most circumstances.

Why do we want Kafka?

We will use Kafka as an easy to use “repository” for the processed documents from the STATdx and other application document trees. Documents will be stored as individual messages on a Kafka topic. These messages can be “replayed” ad infinitum to make re-indexing the entire Solr collection much faster. In addition, once in Kafka, the database(s) and other assets in QA or Prod will not experience the load of re-processing the document trees each time a re-index is required.

In addition, since it seems likely that we will have more call from Elsevier for Search functionality, we’re trying to build a “complete” solution that will allow a turnkey approach to later requests for Search functionality.

- <u>SOLR Reindex for Relevancy Testing:</u>

Because Solr cannot re-index itself, documents have to be resubmitted to Solr every time indexing is desired. This is likely to happen frequently during the relevancy testing phase of the project. Running against the database takes much longer than just dumping in all the messages from Kafka.

- <u>Reindex in Production</u>

Similarly, if it ever becomes necessary to re-index Solr in production (to improve search relevancy for example) using stored messages in Kafka will keep load off essential production systems like the database. We would probably use the “upgrade by replacement” strategy here as well as in disaster recovery.

- <u>Disaster Recovery</u>

In the case of a disaster where we need to rebuild the Solr infrastructure, the stored Kafka messages are by far the fastest way to send all documents into newly-built Solr instances.

How will we use Kafka?

We place "finished" XML (or JSON if we want) messages onto a Kafka topic and we can replay them at will. A microservice will consume the messages and submit them to SOLR. In this way we don't need to hit the database and re-build the SOLR data from the STATdx document tree every time we want to re-index or in the case of disaster recovery.

Not Part of Regular Production Search

Kafka will not be part of the "regular" run-time for Search in Production. It will be used when there is a need to re-index, rebuild a SOLR cluster, or for disaster recovery.

<u>Kafka Caveats: (alliteration intended!)</u>

- Currently there is a small issue with getting Kafka to cough up the messages from the beginning when the microservice calls for them. Kafka keeps track of which “group id” has received which messages and the API I used doesn’t allow a reset of that count.

  To solve the problem, I am currently manually editing the “directory structure” in Zookeeper for Kafka and removing the consumer group. This is a hack that we probably don’t want in production.

  I have heard that the Kafka team is coming out with a newer API, but haven’t investigated. There is also an API for Zookeeper that I’m pretty sure will allow me to automate that deletion. Finally, there is a much more complex API for Kafka that theoretically allows the reset of the count. Finally, there is a tool (Exhibitor) that allows a GUI access to Zookeeper’s data structures and this would make the “hack” a lot easier to accomplish at least.

  This problem will need to be solved – or the manual approach will need to be carefully documented. I vote for the Zookeeper API solution, as it seems the most straightforward and simplest. We will want to discuss this at some point although it does not block forward movement at this point.

- I have not texted how (or if) updating a single message id works in Kafka. Logically, it seems reasonable that there would come a time when we want to update a single document in Solr.

  One way to do this would be to update the message (or add a new one) in Kafka. Another way is to bypass Kafka and just send it directly to Solr via SolrJ code. I’m writing this down so we don’t forget to think about the possibilities and take the time to get clear on how we want to handle updates to small numbers of documents. For example, we probably can add the document “new” to Kafka and get it into Solr that way, but I have NOT tested this and Kafka does use "keys" which in our case are the UUIDs – if that's the same we may have an issue that needs to be solved. We'll need to discuss as I'm not sure what the frequency is on this and whether a new doc has a new UUID or not.

See these useful URL's

https://objectpartners.com/2014/05/06/setting-up-your-own-apache-kafka-cluster-with-vagrant-step-by-step/

and... at the time of this writing

http://kafka.apache.org/documentation.html

Look here: http://kafka.apache.org/documentation.html#quickstart to see how to use the command line to test that you can create topics and consume messages from the topics.

1. Install Kafka

Download here: https://kafka.apache.org/downloads.html
(a simple tar -zxvf filename will install it on linux)

Put it where you want. Do the OS-specific "thing" or put it in it's own directory. There doesn't seem to be any big deal about needing to do it "right" online anywhere Some folks even say they avoid the OS-specific stuff because they don't like how things get broken up.

I did see in one place that some people had trouble with the download and were building it but I just used the download and it worked fine on Ubuntu Server 14.04.3

2. Configure Kafka after installing

Make the following changes to the server.properties file located here:
install_directory/kafka_kafka_2.10-0.8.2.2/config/server.properties

Assuming you are in the Kafka install directory:

```
cd kafka_2.10-0.8.2.2/config/
nano server.properties
```

a. Suggested: log.retention.hours=1750 #two years

If you care about long-term retention of the data as I do for Solr, I recommend that you set the log.retention for some ridiculously long period of time. Two years may NOT be enough.

It is also highly likely that a copy of the logs folders placed on a new Kafka VM would serve as a successful recovery. I'm aware this has to be tested. I have not done so yet. If we're interested, we should test this.

=========

b. Don't mess with hostname setting
I found that messing with hostname and a few other things that LOOKED like they should be made to match the VM it was on did not end well. For now, leave them alone. These may come in to play if we decide we want more than one Kafka "node."
=========

c. IF you have more than one Kafka instance, change the broker.id - it has to be unique
If you have only one Kafka instance, leave it at zero, but make sure it's not commented out. (broker.id=0)

=========

d. DO NOT touch host.name
DO NOT touch advertised.host.name

These probably have meaning - especially in a Kafka cluster situation, but I have not gone that far afield.

=========

e. This one is QUITE important...!
The initial setting is for localhost, which we do NOT want since we care about Prod.

Note that we reference the "chroot" of Zookeeper in this (IP:Port:/chroot) and it's necessary EVERY time you want to talk to Zookeeper from the command line utilities in the bin folder. Forget and you'll force Zookeeper to create stuff on / instead of /kafka and you'll be unhappy with the results.

```
#zookeeper.connect=localhost:2181
 zookeeper.connect=192.168.56.5:2181/kafka   Or whatever IP for zookeeper.
```

I haven't tested, but this probably takes the z1,z2,z3/chroot construction and that is probably better.

You can also, if you like, use zkCli.sh to verify that there is no "/kafka" on Zookeeper's "filesystem." There should also be nothing you didn't expect either. A call from Kafka as it starts up will add Kafka to Zookeeper, and if you fat-finger the command line and forget the /chroot - you'll have entries in zookeeper you need to delete.

==========

f. Change the logs from /tmp !!!!

I set mine from /tmp/kafka-logs to /var/log/kafka-logs

```
# A comma separated list of directories under which to store log files
#log.dirs=/tmp/kafka-logs
log.dirs=/var/log/kafka-logs
```

g. At this point, I'm unsure of the IT-side considerations about snapshots, how long they should be retained, how often they get created, how big they are, etc. If we're concerned about backup, we'll have to talk about this

```
# the directory where the snapshot is stored.
  dataDir=/tmp/zookeeper
```

Change the above to something intelligent
Maybe /var/zookeeper or similar... Just NOT /tmp

3. **Start Kafka**

```
cd /home/john/cd kafka_2.10-0.8.2.2/bin
  ./kafka-server-start.sh ../config/server.properties

   cd /home/john/cd kafka_2.10-0.8.2.2
   bin/kafka-server-start.sh ../config/server.properties
```

REMEMBER!!!!! The zookeeper IP isn't going in on this command line as it does with SOLR because it's in server.properties. I hope you configured that correctly - WITH the "chroot"!

==

Websites of interest
https://kafka.apache.org/08/ops.html
https://kafka.apache.org/documentation.html
http://blog.cloudera.com/blog/2015/07/deploying-apache-kafka-a-practical-faq/
The above link has a lot of info about Kafka/ZooKeeper in large complex environments.