Alchemy Cookbook

Open Source Text Analytics System Setup and Usage

Table of Contents

The Alchemy Process

Historically, Alchemy is the attempt to turn lead into gold. In this context, we strive to turn large quantities of vague unstructured data into explicit structured tables that can be used to gain insight into a set question.

Our process consists of five steps:

- Gathering
- Filtering
- Annotating
- Restructuring
- Visualizing

Gathering

Crawling the web is our basic method of gathering unstructured data. We start with a list of URLs, then scrape the pages, parse them into an index-able format, and store them off for later use.

In this step, we also collect any already structured data that we may find useful, such as geolocations or historical weather for an area. This data is stored away until the Restructuring step.

Filtering

Because our crawling is unspecific and grabs everything it sees on the pages we give it, we end up with a lot of noise in our initial dataset. We whittle the dataset down a little by creating Mindmaps or PESTLE (Political, Economical, Social, Technical, Legal, Environmental) trees that we can use to query for specific topics within our collected webpages.

Annotating

Annotating is a second run of culling our data to gain insight into our question. We can use regular expression matching to pull out a word from a sentence or a sentence from a paragraph, to find names, dates, or money (specific values or in general), or any number of other types of information that are relevant to the question being asked.

Restructuring

After annotating the data, we export it into a structured format (usually csv or json) to be stored in a SQL database. Typically, each type of annotation we have created is given its own table, along with the tables of pre-structured data we gathered in the first step.

Visualizing

Where the magic happens. Now that we have all of our data in an easily-managed format, we can analyze the relationships between our annotations and structured data, and create a Truth Table with adjustable weighted values to craft an answer to our question that's backed by the data.

Components

Our stack is comprised of several Apache open source projects and a few in-house applications built specifically for Alchemy. We strive to keep our process as simple as possible, while maintaining the necessary functionality. Each project and their purpose in the Alchemy process is described below.

Nutch

Apache Nutch is an open source web crawler, known for its extensibility and scalability. Plug-ins are easy to configure, and it's able to use Cassandra as a storage backend, as well as Solr as its indexer.

Ant

Apache Ant is a Java build tool that allows easy configuration of large Java projects and their dependencies with a simple xml file. We use it to build both Nutch and Solr from source.

Cassandra

Apache Cassandra is an open source database system that supports high availability and easy scalability. Performance-wise, Cassandra ranks higher than most NoSQL databases, and is decentralized and fault-tolerant. We chose Cassandra over other Nutch backends, such as Apache HBase or straight SQL, to avoid installing the entire Hadoop system.

UIMA

Apache UIMA (Unstructured Information Management Applications) is an open source project for analyzing data and pulling out relevant entities as a basic Natural Language Processing toolkit.

Solr

Apache Solr is an open source search platform, wrapped around Lucene. Nutch is able to use Solr to index webpage crawl results, and users can query Solr to see results without digging through Cassandra's database.

Mortar

Mortar is an in-house Django web application built to create and manage PESTLE trees and Mindmaps. Mortar performs the first round of filtering on our dataset by allowing users to create custom queries based on their trees.

Portrait

Portrait is an in-house Django web application that leverages Django SQL Explorer to analyze and combine data into useful insights and visualizations.

System Requirements

In order to install the Alchemy components, you will need at least two network-connected machines, each possessing at minimum:

- 4GB RAM
- 2 CPU cores
- 40GB HDD

In addition, we strongly recommend that the machine (or machines) backing the database stores contain at least a 500GB mount point for storage.

The following instructions are written for CentOS 7.2, however, most packages are platform independent and therefore the manual may be easily adapted for other operating systems.

Installation and Configuration

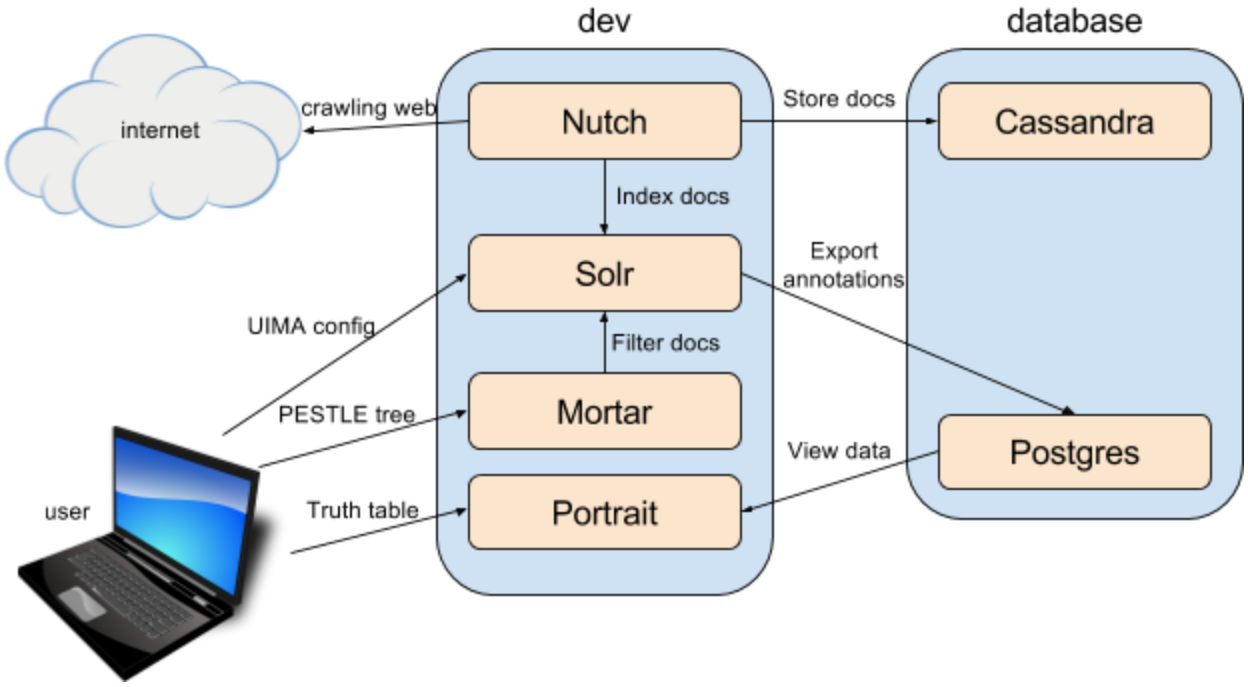
For the purposes of this manual, we will assume that we're installing Alchemy in the minimal configuration, with two Virtual Machines at our disposal: alchemy-dev.oscar.priv and alchemy-db.oscar.priv

At the end of the manual, alchemy-dev.oscar.priv will be configured with (shown in Figure 1):

- Nutch
- Solr
- Mortar
- Portrait

and alchemy-db.oscar.priv will be configured with:

- Cassandra
- PostgreSQL



System Packages

First, we need to set up both machines with Java 1.7 and wget

```
[user@localhost]$ ssh user@alchemy-dev.oscar.priv
[user@alchemy-dev]$ sudo yum install -y wget java-1.7.0-openjdk
java-1.7.0-openjdk-devel
[user@alchemy-dev]$ exit
```

```
[user@localhost]$ ssh user@alchemy-db.oscar.priv
[user@alchemy-db]$ sudo yum install -y wget java-1.7.0-openjdk
```

Apache Ant 1.9.6

1. From the Apache Archives, copy the link for the latest version of Ant (**apache-ant-1.9.6-bin.tar.gz** at the time of writing this document) and untar it on the nutch machine.

```
[user@localhost]$ ssh user@alchemy-dev.oscar.priv
[user@alchemy-dev]$ sudo su
[root@alchemy-dev]# cd /opt
[root@alchemy-dev]# wget http://link_to_ant.tgz
[root@alchemy-dev]# tar zxvf apache-ant-1.9.6-bin.tar.gz
[root@alchemy-dev]# ln -s apache-ant-1.9.6/ ant
```

2. Set **JAVA_HOME** to the correct place for your machine. For nutch-test, which has several versions of Java installed, we link it specifically to 1.7, the version Nutch 2.2.1 prefers. If you copy these lines directly, replace the quotation marks within the console before running the **echo** command. When you **cat** the file, there should be no quotation marks.

```
[root@alchemy-dev]# cd /etc/profile.d/
[root@alchemy-dev]# echo "export
JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk" > java.sh
[root@alchemy-dev]# cat java.sh
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk
```

   You will have to leave your session on the machine and log back in for this change to take affect.

3. To verify that ant is installed and can find java correctly before attempting to compile nutch:

```
[root@alchemy-dev]# /opt/ant/bin/ant -h
/opt/ant/bin/ant [script options] [options] [target [target2 [target3]
..]]
Script Options:
       ...
```

Apache Nutch 2.2.1

After Ant is installed, install and compile Nutch onto the nutch-test machine.

1. From the Apache Archives, copy the link for **apache-nutch-2.2.1-src.tar.gz** and untar it on the nutch machine.

```
[root@alchemy-dev]# cd /opt
[root@alchemy-dev]# wget http://link_to_nutch.tgz
[root@alchemy-dev]# tar zxvf apache-nutch-2.2.1-src.tar.gz
```

```
[root@alchemy-dev]# ln -s apache-nutch-2.2.1/ nutch
```

2. Configure nutch to use Cassandra as its backend storage. In **/opt/nutch/conf/gora.properties** change the following lines to ensure gora uses Cassandra and is listening to one of cassandra's seed servers.

   line 16:
   ```
   gora.datastore.default=org.apache.gora.cassandra.store.CassandraStore
   ```
   line 63:
   ```
   gora.cassandrastore.servers=10.36.10.20:9160
   ```

   Next, open **/opt/nutch/conf/gora-cassandra-mapping.xml** and replace any lines with localhost to the same server mentioned in gora.properties.

   line 20:
   ```
   <keyspace name="webpage" cluster="Test Cluster" host="10.36.10.20">
   ```
   line 60:
   ```
   <keyspace name="host" cluster="Test Cluster" host="10.36.10.20">
   ```

   Finally, update nutch's ivy settings to pull in the required gora-cassandra libraries by uncommenting **line 122** in **/opt/nutch/ivy/ivy.xml**

3. Update **/opt/nutch/conf/nutch-site.xml** with any settings you need. Minimally, the file should contain at least these lines:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
    <name>http.agent.name</name>
    <value>ITng-Test</value>
  </property>
  <property>
    <name>storage.data.store.class</name>
    <value>org.apache.gora.cassandra.store.CassandraStore</value>
  </property>
  <property>
    <name>http.content.limit</name>
    <value>100000</value>
  </property>
  <property>
    <name>plugin.includes</name>
    <value>protocol-httpclient|urlfilter-regex|parse-(html|tika)|
index-(basic|anchor)|scoring-opic|indexer-solr|
urlnormalizer-(pass|regex|basic)</value>
```

```
  </property>
</configuration>
```

4. Compile nutch with ant. The first time this is run, it will take a few minutes, but all subsequent compilations should be closer to a few seconds.

```
[root@alchemy-dev]# cd /opt/nutch
[root@alchemy-dev]# /opt/ant/bin/ant runtime
...
BUILD SUCCESSFUL
Total time: 2 minutes 15 seconds
[root@alchemy-dev]# cd /opt/nutch/runtime/local
[root@alchemy-dev]# bin/nutch -h
Usage: java [-options] class [args...]
```

Apache Cassandra 2.2.8

1. Copy the link to the tarball for **apache-cassandra-2.2.8-bin.tar.gz** from the Apache Archives. Assuming **wget** is installed on the VM (and running **sudo yum install -y wget** if it isn't), grab the tarball and untar it in /opt on the cassandra machine. Then, optionally, create a soft link between the directory and the name cassandra, for ease of use.

```
[user@localhost]$ ssh user@alchemy-db.oscar.priv
[user@alchemy-db]$ sudo su
[root@alchemy-db]# cd /opt
[root@alchemy-db]# wget http://link_to_tar.gz
[root@alchemy-db]# tar zxvf apache-cassandra-2.2.8-bin.tar.gz
[root@alchemy-db]# ln -s apache-cassandra-2.2.8/ cassandra
```

2. Next, configure Cassandra to run on the private network interface (10.36.0.0/16) instead of localhost. Open **/opt/cassandra/conf/cassandra.yaml** in a text editor and change the following lines to the ip addresses needed. Other settings may be changed in this step, but we opted to leave the defaults for most options for now.

```
line 302:
      - seeds: "10.36.1.20"
line 416:
      listen_address: 10.36.1.20
line 462:
      start_rpc: true
line 482:
      rpc_address: 10.36.1.20
```

3. Start Cassandra binary, should automatically drop to background mode.

```
[root@alchemy-db]# cd /opt/cassandra
```

```
[root@alchemy-db]# bin/cassandra
...
INFO  14:38:04 Initializing system_auth.resource_role_permissons_index
INFO  14:38:04 Waiting for gossip to settle before accepting client
requests...
INFO  14:38:12 No gossip backlog; proceeding
```

Apache Solr 6.3.0

1. Copy the link to the source tarball for **solr-6.3.0.tgz** from github. Assuming **wget** is installed on the VM (and running **sudo yum install -y wget** if it isn't), grab the tarball and untar it in /opt on the nutch machine. Then, optionally, create a soft link between the directory and the name solr, for ease of use.

```
[user@localhost]$ ssh user@alchemy-dev.oscar.priv
[user@alchemy-dev]$ sudo su
[root@alchemy-dev]# cd /opt
[root@alchemy-dev]# wget http://link_to_solr.tgz
[root@alchemy-dev]# tar solr-6.3.0.tgz
[root@alchemy-dev]# ln -s solr-lucene-6.3.0/ solr-lucene
```

2. Create a new Solr user, since Solr 6 began restricting processes running as root.

```
[root@alchemy-dev]# useradd solr
[root@alchemy-dev]# chown -R solr:solr /opt/solr-lucene
```

3. Install Java 1.8, then compile lucene and solr using ant.

```
[root@aclhemy-dev]# sudo yum install java-1.8.0-openjdk
java-1.8.0-openjdk-devel
[root@alchemy-dev]# sudo -iu solr
[solr@alchemy-dev]$ cd /opt/solr-lucene
[solr@alchemy-dev]$ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
[solr@alchemy-dev]$ /opt/ant/bin/ant ivy-bootstrap
[solr@alchemy-dev]$ /opt/ant/bin/ant compile
[solr@alchemy-dev]$ /opt/ant/bin/ant jar
[solr@alchemy-dev]$ cd solr
[solr@alchemy-dev]$ /opt/ant/bin/ant dist
[solr@alchemy-dev]$ /opt/ant/bin/ant process-webpages
```

4. Link the necessary lucene jars needed for solr-uima

```
[solr@alchemy-dev]$ mkdir /opt/solr-lucene/solr/contrib/uima/lucene-libs
[solr@alchemy-dev]$ cd /opt/solr-lucene/solr/contrib/uima/lucene-libs
```

```
[solr@alchemy-dev]$ ln -s
/opt/solr-lucene/lucene/build/analysis/common/lucene-analyzers-common-6.
3.0-SNAPSHOT.jar lucene-analyzers-common.jar
[solr@alchemy-dev]$ ln -s
/opt/solr-lucene/lucene/build/analysis/uima/lucene-analyzers-uima-6.3.0-
SNAPSHOT.jar lucene-analyzers-uima.jar
```

5. Start Solr and create at least the nutch core, and optionally a uima one for annotations.

```
[solr@alchemy-dev]$ ./bin/solr start
[solr@alchemy-dev]$ ./bin/solr create -c nutch_core
[solr@alchemy-dev]$ ./bin/solr create -c mortar_core
[solr@alchemy-dev]$ ./bin/solr create -c uima_core
```

6. Replace the default schema.xml with one configured for nutch. Move the default schema.xml and solrconfig.xml to a backup file.

```
[solr@alchemy-dev]$ cd /opt/solr/server/solr/nutch_core/conf
[solr@alchemy-dev]$ mv managed-schema managed-schema.old
[solr@alchemy-dev]$ mv solrconfig.xml solrconfig.xml.old
```

   Download solr-configs.zip from the shared drive folder.

   Place each schema.xml and solrconfig.xml in the respective core's configuration folder (e.g. /opt/solr/server/solr/nutch_core/conf) making sure to rename each to the generic solrconfig.xml and schema.xml

   Note that IP addresses will need to be updated for your configuration within mortar_solrconfig and uima_solrconfig's DataImportHandlers. Uima_schema will also need to be updated for your specific annotator, described more here.

7. Restart Solr to apply configuration changes

```
[solr@alchemy-dev]$ cd /opt/solr
[solr@alchemy-dev]$ ./bin/solr restart
```

PostgreSQL 9.2.1

1. Install postgres server package on alchemy-db

```
[user@localhost]$ ssh user@alchemy-db.oscar.priv
[user@alchemy-db]$ sudo su
[root@alchemy-db]# yum install -y postgresql-server
[root@alchemy-db]# service postgresql initdb
[root@alchemy-db]# service postgresql start
```

2. Configure access settings to allow remote connections

```
[root@alchemy-db]# vim /var/lib/pgsql/data/postgresql.conf
line 59:
      listen_addresses = '*'
```

3. Restart postgres server to apply configuration
4. Install csvsql for easier importing of the csv files we will generate from Solr

```
[root@alchemy-db]# pip install csvkit
```

Mortar

1. Grab a tarball of the latest release code and untar in alchemy-dev

```
[root@alchemy-dev]# cd /opt
[root@alchemy-dev]# wget
https://github.com/ITNG/mortar/archive/master.zip
[root@alchemy-dev]# yum install -y unzip
[root@alchemy-dev]# unzip mortar-master.zip
```

2. Create Mortar user

```
[root@alchemy-dev]# useradd mortar
[root@alchemy-dev]# chown -R mortar:mortar /opt/mortar-master
```

3. Create a virtual environment and install pip requirements

```
[root@alchemy-dev]# sudo -iu mortar
[mortar@alchemy-dev]$ cd /opt/mortar-master
[mortar@alchemy-dev]$ virtualenv .env
[mortar@alchemy-dev]$ source .env/bin/activate
[mortar@alchemy-dev]$ pip install -r requirements.txt
```

4. Create database and user on alchemy-db

postgres=# GRANT ALL ON DATABASE mortar TO mortar;

5. Edit django settings
6. Set up supervisor with gunicorn
7. Set up nginx
8. (optional) set up proxy passthrough

Apache UIMA 2.9.0 and Eclipse UIMA Workbench

This portion is written for a personal machine, rather than a server, where annotators will be developed.

Follow the instructions here for installation and setup of Eclipse and UIMA, including the instructions for viewing example code (this provides the CAS Visual Debugger, which will come in handy when testing annotators)

Example Usage

Crawling the Web

1. Create the list of urls to crawl. Make sure that any quotation marks copied over have been replaced manually in the command line. When you **`cat`** seed.txt, there should be no quotation marks.

```
[root@alchemy-dev]# export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk
[root@alchemy-dev]# export NUTCH_OPTS="-Xmx2048m"
[root@nutch-test]# cd /opt/nutch/runtime/local/
[root@nutch-test]# mkdir urls
[root@nutch-test]# echo "http://nutch.apache.org" > urls/seed.txt
[root@nutch-test]# cat urls/seed.txt
http://nutch.apache.org
```

2. Inject the urls from seed.txt into the Cassandra database. Make sure that the InjectorJob is using CassandraStore, and the total number of urls injected is equal to the total number of urls in urls/seed.txt

```
[root@nutch-test]# bin/nutch inject urls
InjectorJob: starting at 2015-12-23 12:28:23
InjectorJob: Injecting urlDir: urls
InjectorJob: Using class org.apache.gora.cassandra.store.CassandraStore
as the Gora storage class.
InjectorJob: total number of urls rejected by filters: 0
InjectorJob: total number of urls injected after normalization and
filtering: 1
Injector: finished at 2015-12-23 12:28:25, elapsed: 00:00:02
```

3. Generate a batch of urls from the injected urls.

```
[root@nutch-test]# bin/nutch generate -topN 500
GeneratorJob: starting at 2015-12-23 12:31:14
GeneratorJob: Selecting best-scoring urls due for fetch.
GeneratorJob: starting
GeneratorJob: filtering: true
GeneratorJob: normalizing: true
GeneratorJob: topN: 500
GeneratorJob: finished at 2015-12-23 12:31:16, time elapsed: 00:00:02
```

```
GeneratorJob: generated batch id: 1450891874-843461018
```

4. Copy the batch id from GeneratorJob, and fetch the created batch of urls. There's a lot of output for this step, only a little is shown below.

```
[root@nutch-test]# bin/nutch fetch 1450891874-843461018 -resume
FetcherJob: starting
FetcherJob: batchId: 1450891874-843461018
...
Fetcher: throughput threshold sequence: 5
-finishing thread FetcherThread0, activeThreads=0
0/0 spinwaiting/active, 1 pages, 0 errors, 0.2 0 pages/s, 102 102 kb/s,
0 URLs in 0 queues
-activeThreads=0
FetcherJob: done
```

5. Parse through the fetched batch of urls, make sure every url that listed in urls/seed.txt has a line that says "Parsing url"

```
[root@nutch-test]# bin/nutch parse 1450891874-843461018 -force
ParserJob: starting
ParserJob: resuming:    false
ParserJob: forced reparse:      false
ParserJob: batchId:     1450891874-843461018
Parsing http://www.google.com/
ParserJob: success
```

6. Update the Cassandra database with the results of the crawl

```
[root@nutch-test]# bin/nutch updatedb
DbUpdaterJob: starting
DbUpdaterJob: done
```

7. Finally, index the results with Solr

```
[root@nutch-test local]# bin/nutch solrindex
http://localhost:8983/solr/nutch_core -all
SolrIndexerJob: starting
Adding 1 documents
SolrIndexerJob: done.
```

Developing Annotators

If you're unfamiliar with the concept of Annotators or developing them, a good first example is located here. This tutorial is also recommended if you're used to IBM's Content Analytics Studio.

The basic steps of creating a new Annotator are as follows:

1. Create a new Java Project
2. Add UIMA Nature to project
3. Configure Build Path to include UIMA_HOME and Extend uima-core.jar
4. Create a new Analysis Engine file in the desc/ folder (New -> Other -> Analysis Engine Descriptor) and allow it to open up the default CAS View
5. Set the Annotator class to org.apache.uima.yourproject.YourProjectAnnotator and save (this will error until we create the Annotator)
6. Under “Type System” in the Analysis Engine, add a new Annotation of type org.uima.tcas.Annotation with the same classpath scheme as your Annotator class above
7. Save the Analysis Engine, and generate the annotation class files by hitting JCasGen
8. Create a new Annotator class file in src/ with the same package and name as given in the Analysis Descriptor that inherits from org.apache.uima.analysis_component.JCasAnnotator_ImplBase
9. Write the regex matching functions for the annotation you want
10. Ensure your annotator works as planned by loading the AE in the CAS Visual Debugger (right click on example projects -> run -> CAS)
11. Export your annotator as a jar file and scp or sftp it to the machine running Solr.

Filtering with Mortar

1. Log in to the Mortar interface
2. Choose which Project you’d like to query from, or create a new Project from the admin panel
3. Choose a Tree from within the Project, or create one from the /projects page
4. Upload any changes to your Tree from .csv or mindmap
5. Query Solr by selecting the ‘Query Solr’ button from the Tree view and checking the boxes you want to query solr with (and filtered)
6. Check the Solr UI to make sure your documents have been reindexed in mortar_core

Indexing and Annotating Documents

Installing Your Annotator in Solr

1. Move your annotator jar (that was scp or sftp’d up to our solr machine earlier) to the lucene-libs directory

```
[root@nutch-test]# cp annotator.jar
/opt/solr-lucene/solr/contrib/uima/lucene-libs
```

2. Update the solrconfig.xml for uima_core by placing the following lines in the file under the “config” tag, changing the lines in red to match the annotations you created.

```
<lib dir="${solr.install.dir:../../../..}/contrib/uima/lucene-libs" />
<lib dir="${solr.install.dir:../../../..}/contrib/uima/lib" />
<lib dir="${solr.install.dir:../../../..}/dist/"
regex="solr-uima-\d.*\.jar" />

<updateRequestProcessorChain name="uima" default="true">
      <processor
class="org.apache.solr.update.processor.IgnoreFieldUpdateProcessorFactor
y">
        <str name="fieldRegex">_version_</str>
      </processor>
      <processor
class="org.apache.solr.uima.processor.UIMAUpdateRequestProcessorFactory"
>
        <lst name="uimaConfig">
          <lst name="runtimeParameters">
          </lst>
          <str name="analysisEngine">/desc/SolrTestAE.xml</str>
          <bool name="ignoreErrors">true</bool>
          <lst name="analyzeFields">
            <bool name="merge">false</bool>
            <arr name="fields">
              <str>content</str>
            </arr>
          </lst>
          <lst name="fieldMappings">
            <lst name="type">
              <str name="name">org.apache.uima.solrtest.Money</str>
              <lst name="mapping">
                <str name="feature">coveredText</str>
                <str name="field">currency</str>
              </lst>
            </lst>
          </lst>
        </lst>
      </processor>
      <processor class="solr.LogUpdateProcessorFactory" />
      <processor class="solr.RunUpdateProcessorFactory" />
  </updateRequestProcessorChain>

<requestHandler name="/update" class="solr.UpdateRequestHandler">
    <lst name="defaults">
      <str name="update.processor">uima</str>
    </lst>
</requestHandler>
```

3. Move uima_core's managed_schema to schema.xml, then make sure the following fields are included (for nutch), adding the fields specific to your annotator at the same time
4. Restart solr to update the configurations

Reindexing from Mortar/Nutch to UIMA

1. Update URL in solrconfig.xml to pull from either nutch_core or mortar_core
2. Restart solr to update the configurations
3. Import data through Solr's UI (CLI instructions to be written)

Importing Unstructured Data to PostGreSQL

1. Exporting data from Solr as a CSV: from alchemy-db run a query on Solr's UIMA core with wt=csv as an argument and store it in a csv file

```
[user@localhost]$ ssh alchemy-db.oscar.priv
[user@alchemy-db]$ sudo -iu postgres
[postgres@alchemy-db]$ curl
http://alchemy-dev.oscar.priv:8983/solr/uima_core/select?indent=on&q=cur
rency:*&wt=csv > currency.csv
```

2. Importing CSV to PostgreSQL

```
[postgres@alchemy-db]$ csvsql --db
postgresql://dbuser:dbpass@localhost:5432/dbname --insert --encoding
utf-8 --delimiter \, --quotechar \" --no-constraints --no-inference
```

Resources

http://blog.aplikate.eu/2015/01/03/import-csv-data-into-postgresql-the-comfortable-way/