System Overview

The ADARA Processing and Cataloging System (APCS) is implemented as separate services performing the needed work. The major services are:

1. workflow manager
2. permissions
3. cataloging
4. automated reduction
5. reduced data cataloging

Each service communicates with an ActiveMQ Broker as shown in Figure 1.

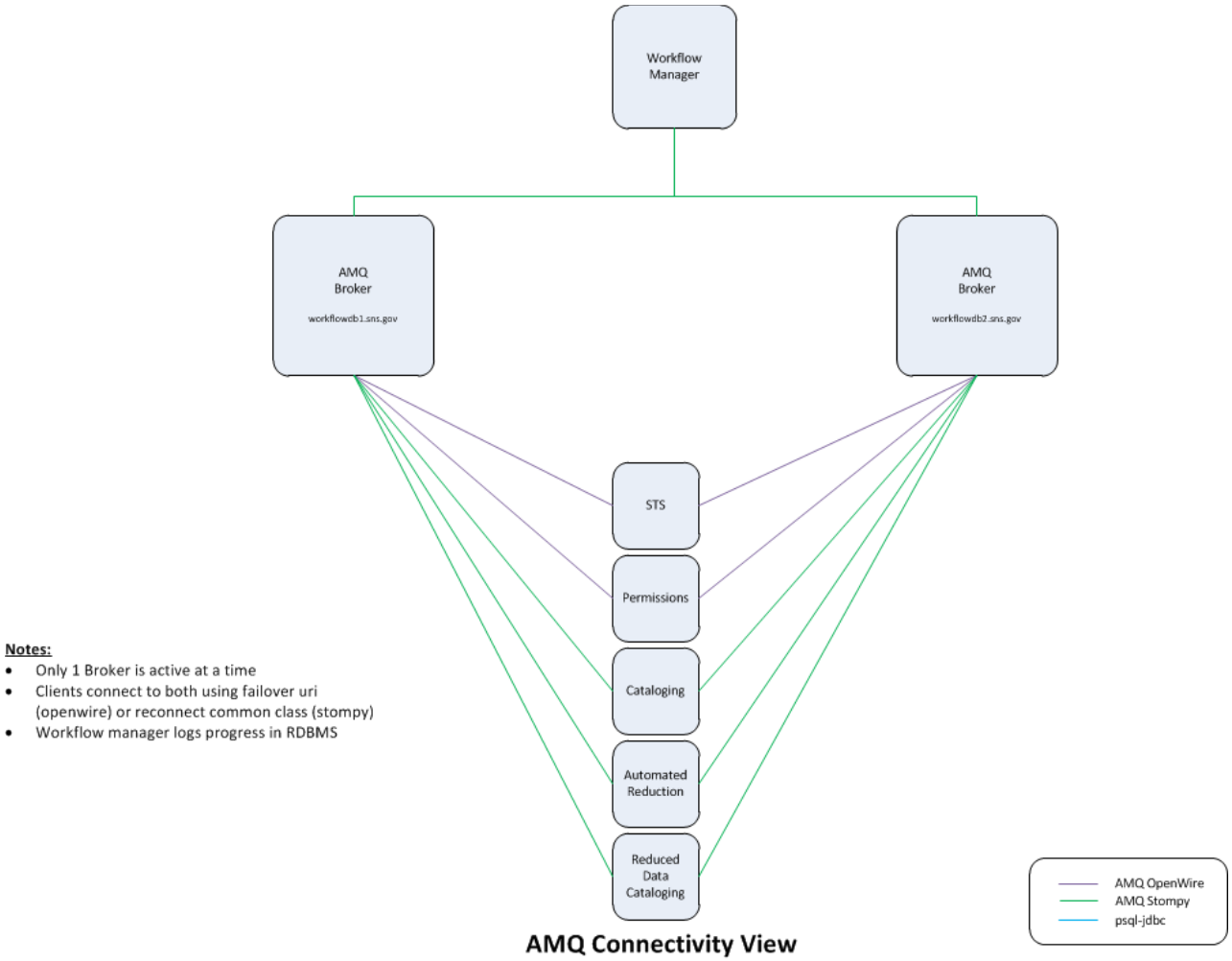
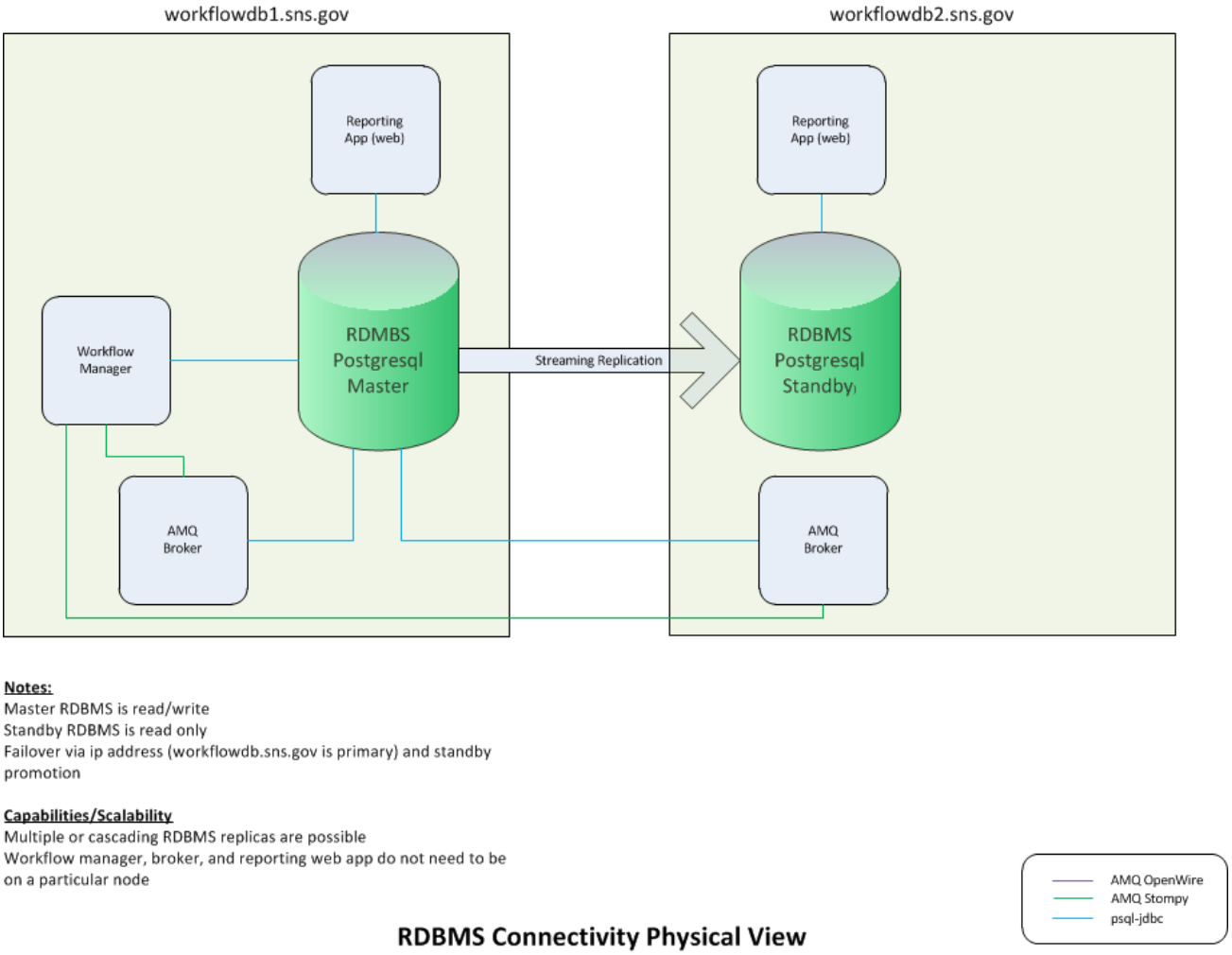


Figure 1

The ActiveMQ brokers utilize the Postgresql RDBMS for persistent message storage. In the APCS setup, the RDBMS is configured in a Master/Standby configuration with the Streaming Replication to the Standby server.

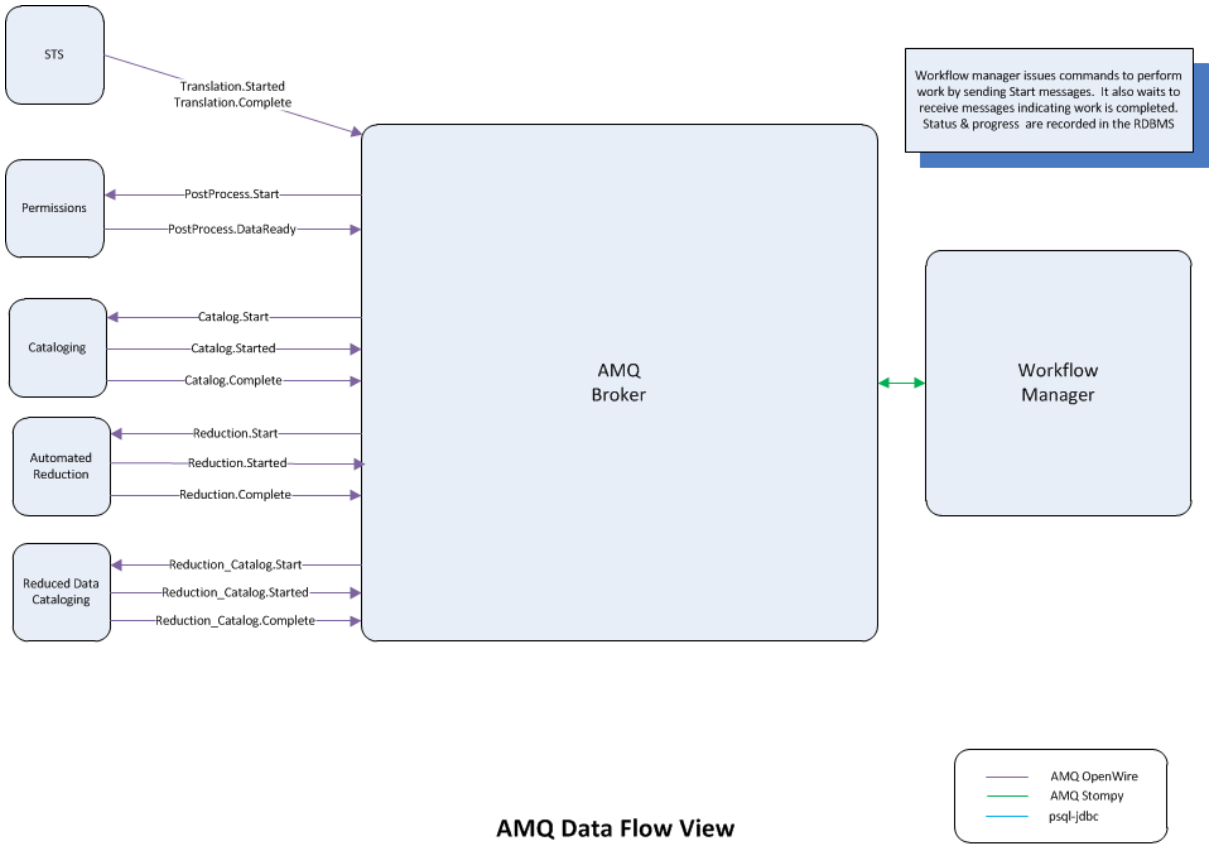


RDBMS Connectivity Physical View

Figure 2

The Workflow manager also uses a database in the Postgresql RDBMS for storing workflow progress and status. The reporting application in-turn reads data from the RDBMS to report a run's processing/cataloging progress and status. The RDBMS setup is illustrated in Figure 2.

The communication messages and workflow sequence are also defined in the RDBMS. This allows for customization and tailoring per instrument, and it provides the ability to define different workflow types for a particular instrument. The messages from a typical workflow are illustrated in Figure 3.



AMQ Data Flow View

Figure 3

Failover

Quigley – primary workflowdb.sns.gov, ip change, standby database can be promoted with signal (pg_ctl promote). For recovery the formerly standby Postgresql server should remain the master and a new standby setup from the new master.

Miscellaneous Notes

Quigley has newer notes with proper users & passwords
Streaming Replication

1. Setup the master database
   postgresql.conf (not complete, just some highlighted ones)
   a. activate_archive_mode = on
   b. wal_keep_segments = 1000

2. Start the master
3. Copy the master to the slave data dir using pg_basebackup
4. Setup recovery.conf
5. Keep recovery.conf as recovery.done on the master

Command to get backup to create standby from

pg_basebackup –D ./ -F tar –x –z –P –v –U <username> -h <host>

Setting up the master and standby

http://www.postgresql.org/docs/devel/static/warm-standby.html

- postgresql.conf (not complete, just some highlighted ones)
  - activate_archive_mode = on
  - wal_keep_segments = 1000

To change so commits block until the replica is written to disk:
Set synchronous_standby_names to a non-null value. And synchronous_write to 'on'

Tested and this works. If the standby goes down the db stops until it returns. You can have more than one and only one has to confirm the write. But if sync replication is enabled you cannot continue with a single master server. Async replication may be sufficient for our use.
http://archives.postgresql.org/pgsql-hackers/2012-07/msg00409.php

DB Failover
http://www.postgresql.org/docs/devel/static/warm-standby-failover.html

Using Postgresql as persistent amq store

To create a postgres database for activemq
Ref: http://trenaman.blogspot.com/2008/09/setting-up-postgresql-database-for.html

Setup PostgreSQL database for activemq

1. become user postgres
   su - postgres
2. Create a database called activemq-db
   $ created activemq-db
3. login to the db as the admin user (i.e. postgres)
   psql activemq-db
   Note: security needs to be discussed with Quigley so the following should be reviewed and altered as needed
4. set password for user postgres
   activemq-db=> alter user postgresql with password 'postgres';
5. create activemq user
   activemq-db=> create user activemq with password 'activemq;
6. Exit psql
   activemq-db=> \q
7. Edit pg_hba.conf file to allow activemq user acess to the activemq-db
   Example

```
# for streaming replication
host  replication  replication 172.16.162.1/32         trust
host  all          postgres    172.16.162.1/32         md5
# for activemq
host  activemq-db   activemq   172.16.162.1/32         md5
host  activemq-db   activemq   172.16.162.177/32       md5
local activemq-db   all                                trust
# for reporting_db
host  reporting_db  icat       172.16.162.177/32       md5
```

8. Tell server to reload config
   $ pg_ctl reload
9. Test login
   $ pgsql –U activemq activemq

```
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

 activemq-db=>
```

10. The database tables will be created by the broker when it connects

Configure the ActiveMQ brokers to use the postgresql

1. Determine path to reporting app (i.e. /opt/ornl/sns/workflow, or /var/www/workflow)
2. /etc/httpd/conf/httpd.conf – setting to get the django config from workflow reporting/apache/apache_django_wsgi.conf
3. Add PYTHONPATH to django conf
   i.e. WSGIPythonPath /var/www/workflow/app
4. Configure activemq config to use postgres persistent store
5. Installation/configuration of workflow manager, tests, etc.

Ref: http://activemq.apache.org/jdbc-master-slave.html

1. Add bean for postgres in activemq.xml

```
<!-- Postgres DataSource Sample Setup -->
<bean id="postgres-ds" class="org.postgresql.ds.PGPoolingDataSource">
  <property name="serverName" value="heidelberg"/>
  <property name="databaseName" value="activemq-db"/>
  <property name="portNumber" value="0"/>
  <property name="user" value="activemq"/>
  <property name="password" value="activemq"/>
  <property name="maxConnections" value="10"/>
</bean>
```

2. Activemq needs to use postgresql-jdbc.jar
   Copy jar file (postgresql-jdbc.jar) into activemq/lib/optional/
   (There maybe other ways to set the search path, but this works.)

3. Turn off query timeout (since jdbc class does not implement it)

```
<broker useJmx="false" brokerName="jdbcBroker" xmlns="http://activemq.apache.org/schema/core">
<persistenceAdapter>
 <jdbcPersistenceAdapter dataDirectory="${activemq.data}" dataSource="#postgres-ds">
   <databaseLocker>
     <database-locker queryTimeout="0" />
   </databaseLocker>
  </jdbcPersistenceAdapter>
 </persistenceAdapter>
```

The above is for master/slave brokers. You can have multiple slaves. The way it works is the first broker gets a database lock and becomes the master. The remaining brokers block waiting to get the lock. If the master exits or dies, one of the waiting brokers will take over as the master.

> For Clustered JDBC database set useDatabaseLock to false. See 20120919 and 20120913 notes. This setting is not recommended due to unreliable failover/handoff.

```
<jdbcPersistenceAdapter dataDirectory="${activemq.data}"
dataSource="#postgres-ds" useDatabaseLock="false">
```

Ref:
http://fusesource.com/docs/esb/4.3/amq_persistence/FuseMBPersistJDBCNoJournal.html

The recommended configuration is master/slave brokers using JDBC Persistent store. Ref:
http://fusesource.com/docs/broker/5.5/clustering/Failover-MasterSlave-JDBC.html

WARNING / ISSUES / NOTES

20120919-1: Tested with C++ client using openwire failover. The client blocks and switches over fine. However, there are issues with dual (or probably more) brokers when not using the useDatabaseLock. There were attempts in the brokers to insert duplicate primary keys in the database after switching over. Had to restart activemq server(s) to remedy. Note: when using master/slave (useDatabaseLock="true") the c++ client always switch fine and there were no broker problems detected.

20120913-1. Using cluster db store had three undelivered messages in table. Had to restart a broker for them to be sent. Not sure why or how it got in that condition, but there was a lot of thrashing, heavly load stop & start testing before that. (Update: this may have been with multiple brokers - not master/slave, see 20120919 note)

20120911-1. Had scenario where there were zombie (maybe) connections to the activemq tables and the new activemq could not get the master lock and just stopped there (never created 616xx port openings), and it was not operational. CONCLUSION: This is how the master/slave broker works. One blocks trying to get the lock until the master releases the lock (or dies). Normal mechanism