Create a Healthcare Data Hub with AWS and Mirth Connect

AWS Big Data Blog

Create a Healthcare Data Hub with AWS and Mirth Connect

by Joseph Fontes | on 12 JAN 2017 | in Amazon EMR, AWS Big Data | Permalink | Comments | Share

As anyone visiting their doctor may have noticed, gone are the days of physicians recording their notes on paper. Physicians are more likely to enter the exam room with a laptop than with paper and pen. This change is the byproduct of efforts to improve patient outcomes, increase efficiency, and drive population health. Pushing for these improvements has created many new data opportunities as well as challenges. Using a combination of AWS services and open source software, we can use these new datasets to work towards these goals and beyond.

When you get a physical examination, your doctor's office has an electronic chart with information about your demographics (name, date of birth, address, etc.), healthcare history, and current visit. When you go to the hospital for an emergency, a whole new record is created that may contain duplicate or conflicting information. A simple example would be that my primary care doctor lists me as *Joe* whereas the hospital lists me as *Joseph*.

Providers record patient information across different software platforms. Each of these platforms can have varying implementations of complex healthcare data standards. Also, each system needs to communicate with a central repository called a health information exchange (HIE) to build a central, complete clinical record for each patient.

In this post, I demonstrate the capability to consume different data types as messages, transform the information within the messages, and then use AWS service to take action depending on the message type.

Overview of Mirth Connect

Using open source technologies on AWS, you can build a system that transforms, stores, and processes this data as needed. The system can scale to meet the ever-increasing demands of modern medicine. The project that ingests and processes this data is called Mirth Connect.

Mirth Connect is an open source, cross-platform, bidirectional, healthcare integration engine. This project is a standalone server that functions as a central point for routing and processing healthcare information.

Running Mirth Connect on AWS provides the necessary scalability and elasticity to meet the current and future needs of healthcare organizations.

Healthcare data hub walkthrough

Healthcare information comes from various sources and can generate large amounts of data:

- Health information exchange (HIE)
- Electronic health records system (EHR)
- Practice management system (PMS)
- Insurance company systems
- Pharmacy systems
- Other source systems that can make data accessible

Messages typically require some form of modification (transformation) to accommodate ingestion and processing in other systems. Using another project, Blue Button, you can dissect large healthcare messages and locate the sections/items of interest. You can also convert those messages into other formats for storage and analysis.

Data types

The examples in this post focus on the following data types representing information made available from a typical healthcare organization:

- HL7 (Health Level Seven) version 2 messages
- CDA (Clinical Data Architecture)/CDD (Continuity of Care Document)
- DICOM (Digital Imaging and Communications in Medicine)

- CSV (comma-separated variable)

HL7 version 2 messages define both a message format and communication protocol for health information. They are broken into different message types depending on the information that they transmit.

There are many message types available, such as ordering labs, prescription dispensing, billing, and more. During a routine doctor visit, numerous messages are created for each patient. This provides a lot of information but also a challenge in storage and processing. For a full list of message types, see Data Definition Tables, section A6. The two types used for this post are:

- ADT A01 (patient admission and visit notification)

View a sample HL7 ADT A01 message

- SIU S12 (new appointment booking)

View a sample SIU S12 message

As you can see, this text is formatted as delimited data, where the delimiters are defined in the top line message called the MSG segment. Mirth Connect can parse these messages and communicate using the standard HL7 network protocol.

hl7-v2-siu-s12.txt Raw

```
MSH|^~\&|SOURCEEHR|WA|MIRTHDST|WA|201611111111||SIU^S12|MSGID10001|P|2.3|
SCH|30000001||||CHECKUP|ROUTINE|NORMAL|20|MIN|201611111400||||JOHN||||(206)555-5309||||ARRIVED
PD1||||123456^DOCTOR^BOB^T^^DR|
PID|1|100001^^^1^MRN1|900001||DOE^JOHN^^^^||19601111|M||WH|111 THAT PL^^HERE^WA^98020^USA||(206)555-5309|||M|NON|999999999|
PV1|1|O||||123456^DOCTOR^BOB^T^^DR|^|||||||||||||||||||||||||||||||||||||||||||||||
AIL|||123456^DOCTOR^BOB^T^^DR|||201611110000|||20|MIN
AIG|||7777|||||201611110000|||20|MIN
NTE|1||ANNUAL HLCK^^|
AIP|1||PBN^LISAPORTER|50|||||||||
```

CCD documents, on the other hand, were defined to give the patient a complete snapshot of their medical record. These documents are formatted in XML and use coding systems for procedures, diagnosis, prescriptions, and lab orders. In addition, they contain patient demographics, information about patient vital signs, and an ever-expanding list of other available patient information.

Health insurance companies can make data available to healthcare organizations through various formats. Many times, the term CSV is used to identify not just comma-delimited data types but also data types with other delimiters in use. This data type is commonly used to send datasets from many different industries, including healthcare.

Finally, DICOM is a standard used for medical imaging. This format is used for many different medical purposes such as x-rays, ultrasound, CT scans, MRI results, and more. These files can contain both image files as well as text results and details about the test. For many medical imaging tests, the image files can be quite large as they must contain very detailed information. The storage of these large files—in addition to the processing of the reporting data contained in the files—requires the ability to scale both storage and compute capacity. You must also use a storage option providing extensive data durability.

Infrastructure design

Before designing and deploying a healthcare application on AWS, make sure that you read through the AWS HIPAA Compliance whitepaper. This covers the information necessary for processing and storing patient health information (PHI).

To meet the needs of scalability and elasticity with the Mirth Connect application, use Amazon EC2 instances with Amazon EBS storage. This allows you to take advantage of AWS features such as Auto Scaling for instances. EBS allows you to deploy encrypted volumes, readily provision multiple block devices with varying sizes and throughput, and create snapshots for backups.

Mirth Connect also requires a database backend that must be secure, highly available, and scalable. To meet these needs with a HIPAA-eligible AWS service, use Amazon RDS with MySQL.

Next, implement JSON as the standard format for the message data storage to facilitate easier document query and retrieval. With the core functionality of Mirth Connect as well as other open source software offerings, you can convert each message type to JSON. The storage and retrieval of these converted messages requires the use of a database. If you were to use a relational database, you would either need to store blobs of text for items/fields not yet defined or create each column that may be needed for a message.

With the ever-updating standards and increases in data volume and complexity, it can be advantageous to use a database engine that can automatically expand the item schema. For this post, use Amazon DynamoDB. DynamoDB is a fully managed, fast, and flexible NoSQL database service providing scalable and reliable low latency data access.

If you use DynamoDB with JSON, you don't have to define each data element field of a database message before insert. Because DynamoDB is a managed service, it eliminates the administrative overhead of deploying and maintaining a distributed, scalable NoSQL database cluster. For more information about the addition of the JSON data type to DynamoDB, see the Amazon DynamoDB Update – JSON, Expanded Free Tier, Flexible Scaling, Larger Items AWS Blog post.

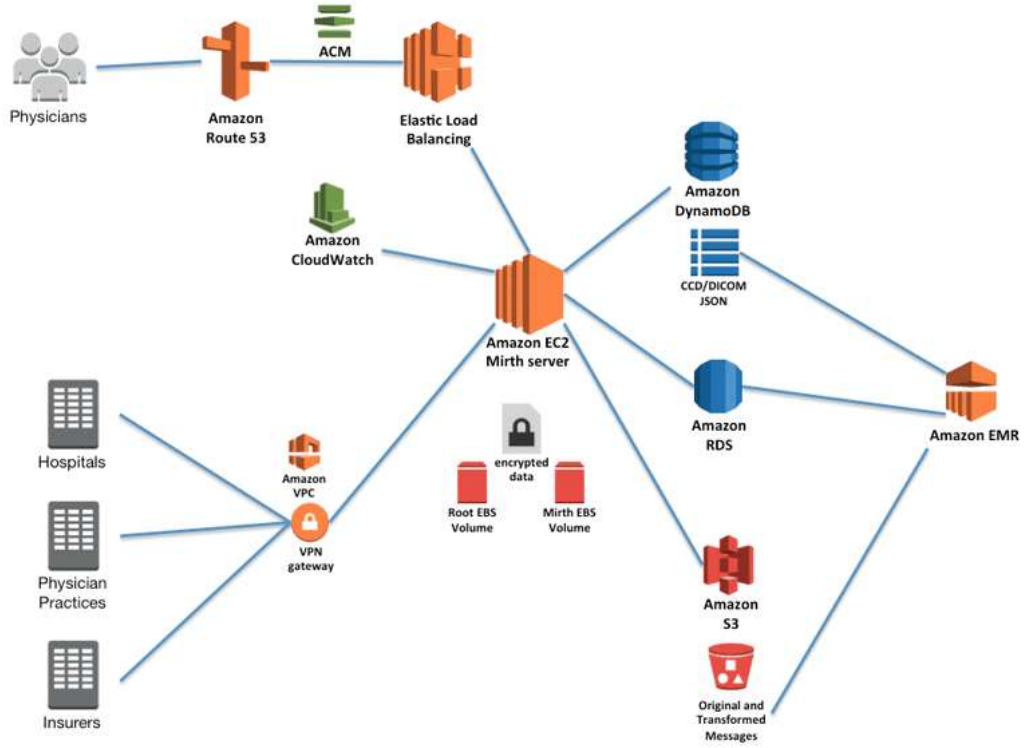
The file storage requirements for running a healthcare integration engine can include the ability to store large numbers of small files, large numbers of large files, and everything in between. To ensure that the requirements for available storage space and data durability are met, use Amazon S3 to store the original files as well as processed files and images.

Finally, use some additional AWS services to further facilitate automation. One of these examples is Amazon CloudWatch. By sending Mirth Connect metrics to CloudWatch, we can enable AWS alerts and actions that ensure optimal application availability, efficiency, and responsiveness. An example Mirth Connect implementation design is available in the following diagram:

1. HL7传输协议(161)
2. 第壹課-Install：Mirth Conn 安装步骤(99)
3. 开篇:Mirth Connect系统集成 5)
4. HL7标准的版本(75)
5. 第三課：信道学习Source C Destinations File Writer(60)

评论排行榜

1. 第八課-Channel Study For R Lib(1)



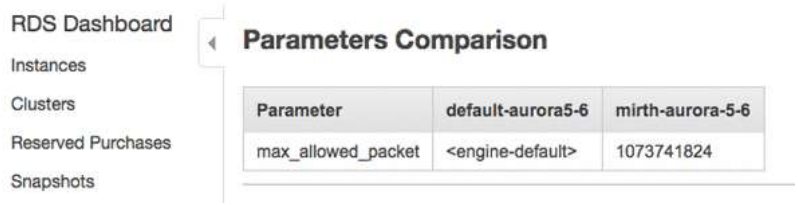
AWS service configuration

When you launch the EC2 instance, make sure to encrypt volumes that contain patient information. Also, it would be a good practice to also encrypt the boot partition. For more information, see the New – Encrypted EBS Boot Volumes AWS Blog post. I am storing the protected information on a separate encrypted volume to make for easier snapshots, mounted as */mirth/*. For more information, see Amazon EBS Encryption.

Also, launch an Amazon RDS instance to store the Mirth Connect configuration information, message information, and logs. For more information, see Encrypting Amazon RDS Resources.

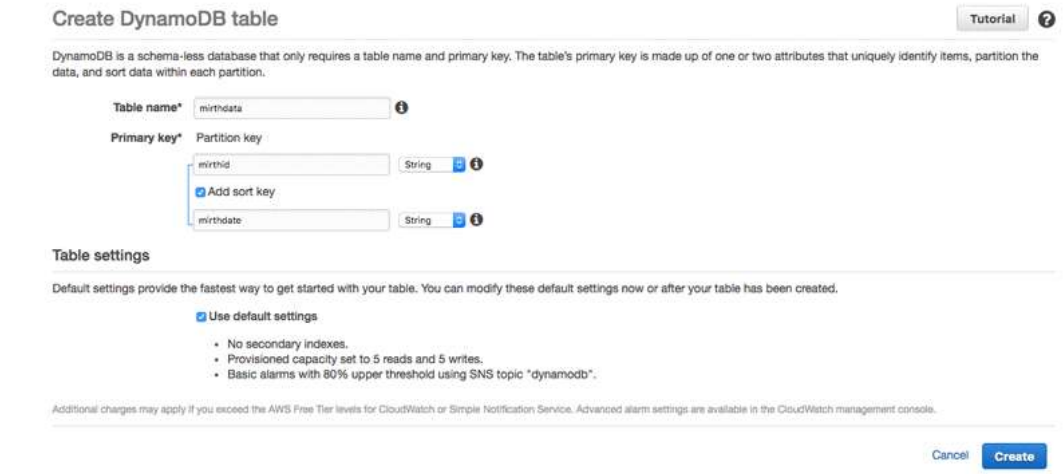
During the configuration and deployment of your RDS instance, update the *max_allowed_packet* value to accommodate the larger size of DICOM images. Change this setting by creating a parameter group for Amazon RDS MySQL/MariaDB/Aurora. For more information, see Working with DB Parameter Groups. I have set the value to 1073741824.

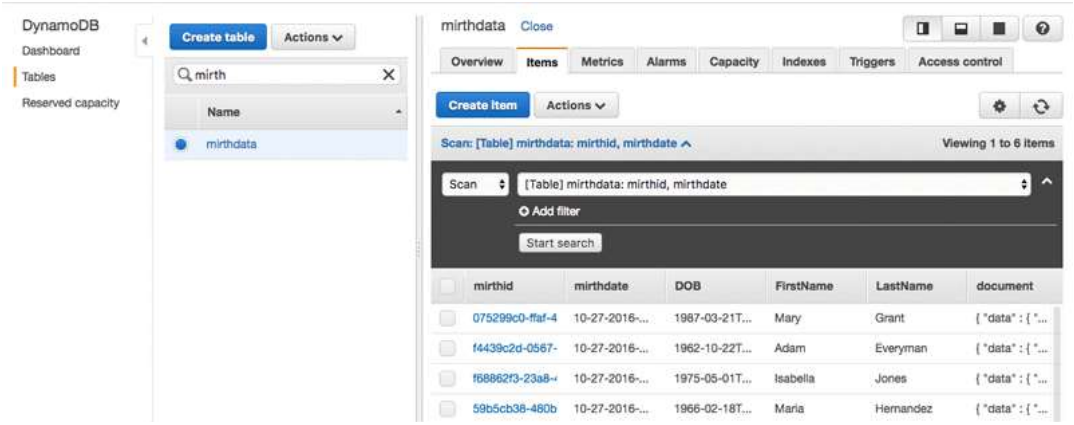
After the change is complete, you can compare database parameter groups to ensure that the variable has been set properly. In the Amazon RDS console, choose Parameter groups and check the box next to the default parameter group for your database and the parameter group that you created. Choose Compare Parameters.



Use an Amazon DynamoDB table to store JSON-formatted message data. For more information, see Creating Tables and Loading Sample Data.

For your table name, use *mirthdata*. In the following examples, I used *mirthid* as the primary key as well as a sort key named *mirthdate.* The sort key contains the date and time that each message is processed.





Set up an Amazon S3 bucket to store your messages, images, and other information. Make your bucket name globally unique. For this post, I used the bucket named mirth-blog-data; you can create a name for your bucket and update the Mirth Connect example configuration to reflect that value. For more information, see Protecting Data Using Encryption.

Mirth Connect deployment

You can find installation packaging options for Mirth Connect at Downloads. Along with the Mirth Connect installation, also download and install the Mirth Connect command line interface. For information about installation and initial configuration steps, see the Mirth Connect Getting Started Guide.

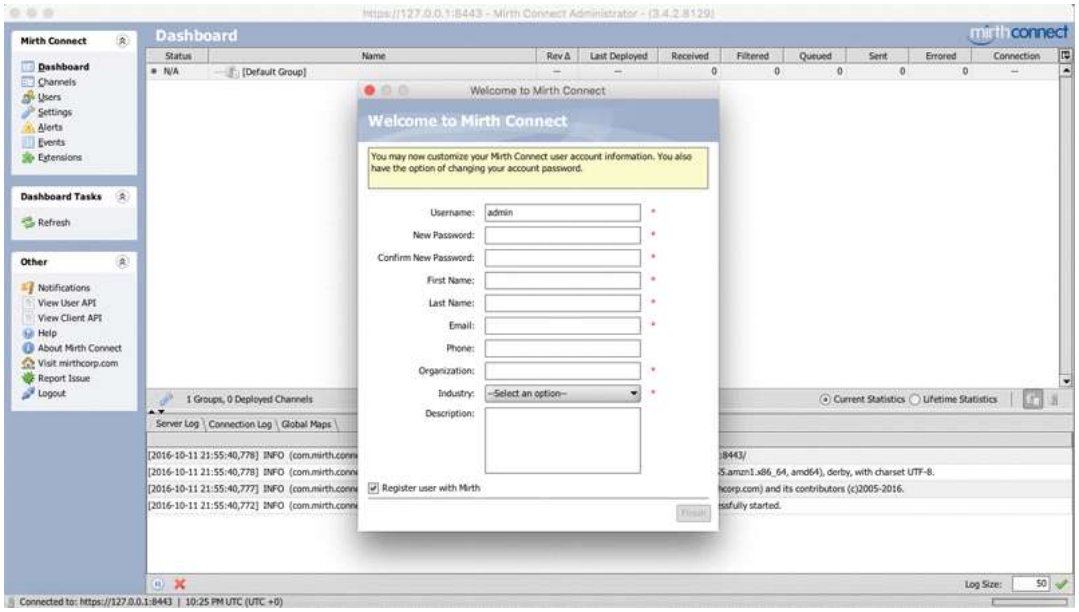
After it's installed, ensure that the Mirth Connect service starts up automatically on reboot, and then start the service.

```
chkconfig --add mcservice
/etc/init.d/mcservice start
```

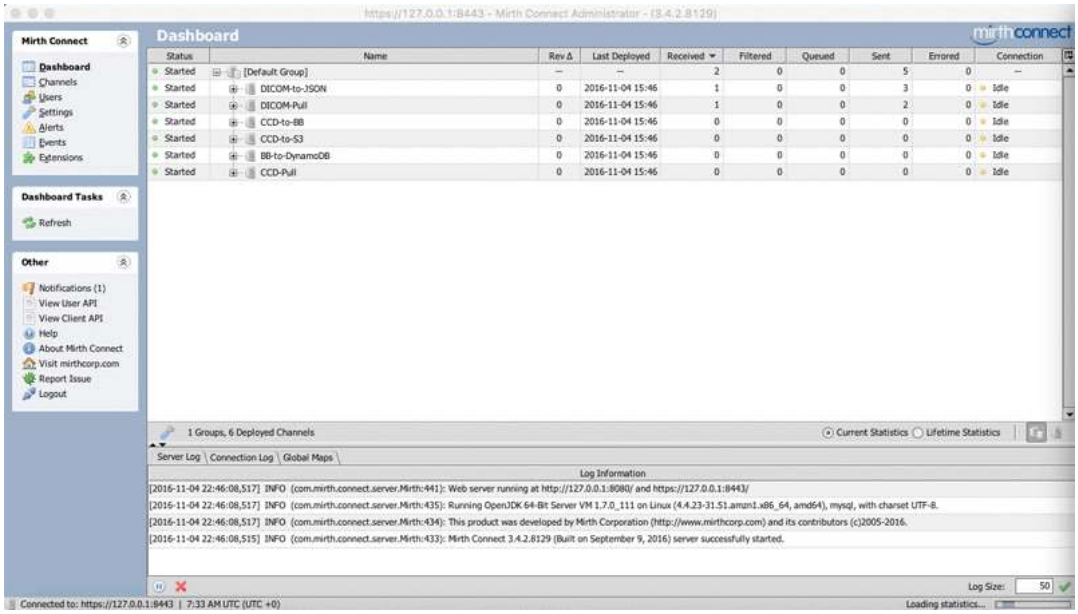
Bash

```
root@mirth-demo-blog (/var/tars/mirth-cli)chkconfig --add mcservice
root@mirth-demo-blog (/var/tars/mirth-cli)chkconfig --list | grep mcservice
mcservice       0:off   1:off   2:on    3:on    4:on    5:on    6:off
root@mirth-demo-blog (/var/tars/mirth-cli)/etc/init.d/mcservice start
Starting mcservice
root@mirth-demo-blog (/var/tars/mirth-cli)
```

The *Mirth Connect Getting Started Guide* covers the initial configuration steps and installation. On first login, you are asked to configure the admin user and to set a password.



You are now presented with the Mirth Connect administrative console. You can view an example console with a few deployed channels in the following screenshot.



By default, Mirth Connect uses the Derby database on the local instance for configuration and message storage. Your next agenda item is to switch to a MySQL-compatible database with Amazon RDS. For information about changing the Mirth Connect database type, see Moving Mirth Connect configuration from one database to another on the Mirth website.

Using the Mirth Connect command line interface (CLI), you can perform various functions for CloudWatch integration. Run the *channel list* command from an SSH console on your Mirth Connect EC2 instance.

In the example below, replace *USER* with the user name to use for connection (typically, *admin* if you have not created other users). Replace PASSWORD with the password that you set for the associated user. After you are logged in, run the command *channel list* to ensure that the connection is working properly:

```
mccommand -a https://127.0.0.1:8443 -u USER -p PASSWORD
channel list
quit
```

Bash

```
root@mirth-demo-blog (/opt/mirthconnect)/opt/mirthconnect/mccommand -a https://127.0.0.1:8443 -u admin -p $MIRTH_PASSWORD
Connected to Mirth Connect server @ https://127.0.0.1:8443 (3.4.2.8129)
$channel list
ID                                   Enabled         Name
c05e7751-5e94-433c-9b5e-db5bab30aa9f YES             BB-to-DynamoDB
0d69b092-33d1-4213-8684-dbf3a7e265b4 YES             CCD-Pull
a53219c7-8fb4-48aa-8c58-2031fdc4c862 YES             CCD-to-BB
9b1eec10-3b79-40a6-9a46-8bc06943d06c YES             CCD-to-S3
a2f09b6e-4eab-4b3c-a74f-b9bc76be351e YES             DICOM-Pull
49f9ef42-307c-40fe-b32b-58776a102088 YES             DICOM-to-JSON
$quit
Disconnected from server.
root@mirth-demo-blog (/opt/mirthconnect)
```

Using the command line interface, you can create bash scripts using the CloudWatch put-metric-data function. This allows you to push Mirth Connect metrics directly into CloudWatch. From there, you could build alerts and actions based on this information.

```
echo "dump stats \"/tmp/stats.txt\"" > m-cli-dump.txt
mccommand -a https://127.0.0.1:8443 -u admin -p $MIRTH_PASS -s m-cli-dump.txt
cat /tmp/stats.txt
```

Bash

```
root@mirth-demo-blog (/opt/mirthconnect)echo "dump stats \"/tmp/stats.txt\"" > m-cli-dump.txt
root@mirth-demo-blog (/opt/mirthconnect)mccommand -a https://127.0.0.1:8443 -u admin -p $MIRTH_PASS -s m-cli-dump.txt
Connected to Mirth Connect server @ https://127.0.0.1:8443 (3.4.2.8129)
Executing statement: dump stats "/tmp/stats.txt"
Stats written to /tmp/stats.txt
Disconnected from server.
root@mirth-demo-blog (/opt/mirthconnect)cat /tmp/stats.txt
Mirth Channel Statistics Dump: Thu Nov 10 22:51:00 UTC 2016
Name, Received, Filtered, Queued, Sent, Errored
CCD-to-BB, 0, 0, 0, 0, 0
CCD-to-S3, 0, 0, 0, 0, 0
BB-to-DynamoDB, 0, 0, 0, 0, 0
CCD-Pull, 0, 0, 0, 0, 0
DICOM-Pull, 1, 0, 0, 1, 0
DICOM-to-JSON, 1, 0, 0, 3, 0
root@mirth-demo-blog (/opt/mirthconnect)
```

More information for the administration and configuration of your new Mirth Connect implementation can be found on both the Mirth Connect Wiki as well as the PDF user guide.

Example configuration

Mirth Connect is designed around the concept of channels. Each channel has a source and one or more destinations. In addition, messages accepted from the source can be modified (transformed) when they are ingested at the source, or while they are being sent to their destination. The guides highlighted above as well as the Mirth Connect Wiki discuss these concepts and their configuration options.

The example configuration has been made available as independent channels available for download. You are encouraged to import and review the channel details to understand the design of the message flow and transformation. For a full list of example channels, see mirth-channels in the Big Data Blog GitHub repository.

Channels:

- CCD-Pull – Processes the files in the directory
- CCD-to-BB – Uses Blue Button to convert CCD into JSON
- CCD-to-S3 – Sends files to S3
- BB-to-DynamoDB – Inserts Blue Button JSON into DynamoDB tables
- DICOM-Pull – Processes the DICOM files from the directory
- DICOM-Image – Process DICOM embedded images and stores data in Amazon S3 and Amazon RDS
- DICOM-to-JSON – Uses DCM4CHE to convert XML into JSON. Also inserts JSON into DynamoDB tables and saves the DICOM image attachment and original file in S3
- HL7-to-DynamoDB – Inserts JSON data into Amazon DynamoDB
- HL7-Inbound – Pulls HL7 messages into engine and uses NodeJS web service to convert data into JSON

For these examples, you are using file readers and file writers as opposed to the standard HL7 and DICOM network protocols. This allows you to test message flow without setting up HL7 and DICOM network brokers. To use the example configurations, create the necessary local file directories:

```
mkdir -p /mirth/app/{s3,ccd,bb,dicom,hl7}/{input,output,processed,raw}
```

Blue Button library

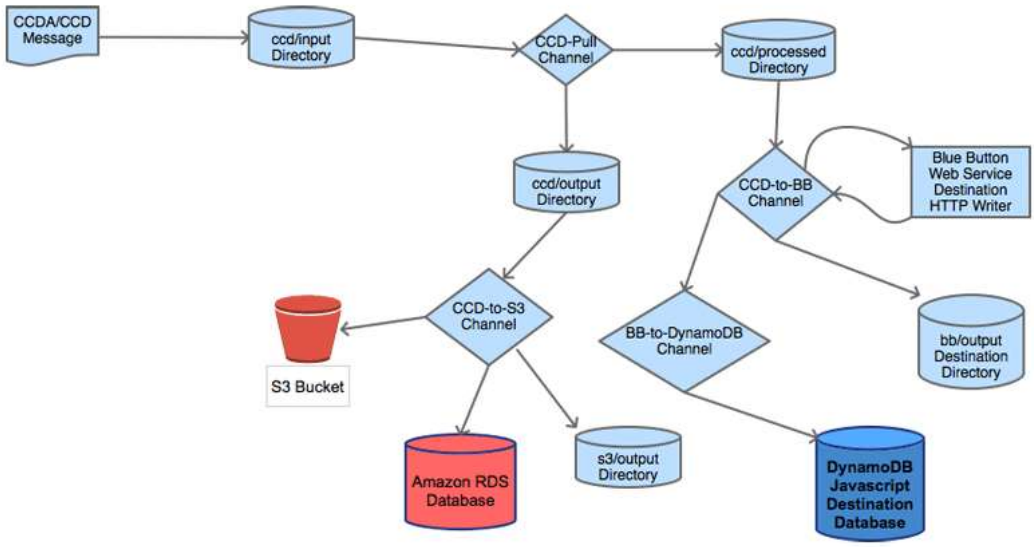
Blue Button is an initiative to make patient health records available and readable to patients. This initiative has driven the creation of a JavaScript library that you use to parse the CCD (XML) documents into JSON. This project can be used to both parse and create CCD documents.

For this post, run this library as a service using Node.js. This library is available for installation as *blue-button* via the Node.js package manager. For details about the package, see the npmjs website. Instructions for the installation of Node.js via a package manager can be found on nodejs.org.

For a simple example of running a Blue Button web server on the local Amazon EC2 instance with the test data, see the bb-conversion-listener.js script in the Big Data Blog repository.

Processing clinical messages

The flow diagram below illustrates the flow of CCDA/CCD messages as they are processed through Mirth Connect.



Individual message files are placed in the "/mirth/app/ccd/input/" directory. From there, the channel, "CCD-Pull" reads the file and then transfers to another directory for processing. The message data is then sent to the Mirth Connect channel "CCD-to-S3", which then saves a copy of the message to S3 while also creating an entry for each item in Amazon RDS. This allows you to create a more efficient file-naming pattern for S3 while still maintaining a link between the original file name and the new S3 key.

Next, the Mirth Connect channel "CCD-to-BB" passes the message data via an HTTP POST to a web service running on your Mirth Connect instance using the Blue Button open source healthcare project. The HTTP response contains the JSON-formatted translation of the document you sent. This JSON data can now be inserted into a DynamoDB table. When you are building a production environment, this web service could be separated onto its own instance and set up to run behind an Application Load Balancer with Auto Scaling.

You can find sample files online for processing across different locations. Try the following for CCD and DICOM examples:

- sample_ccdas GitHub repository
- OsiriX DICOM Image Library

AWS service and Mirth integration

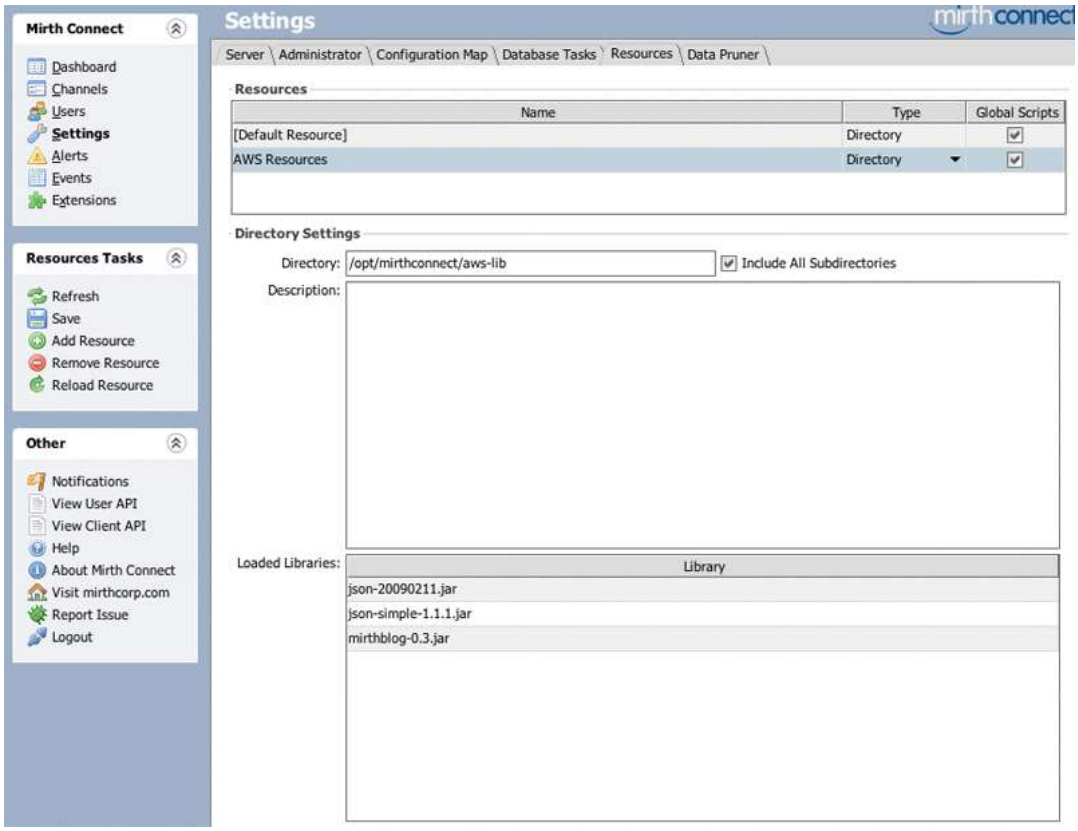
To integrate with various AWS services, Mirth Connect can use the AWS SDK for Java by invoking custom Java code. For more information, see How to create and invoke custom Java code in Mirth Connect on the Mirth website.

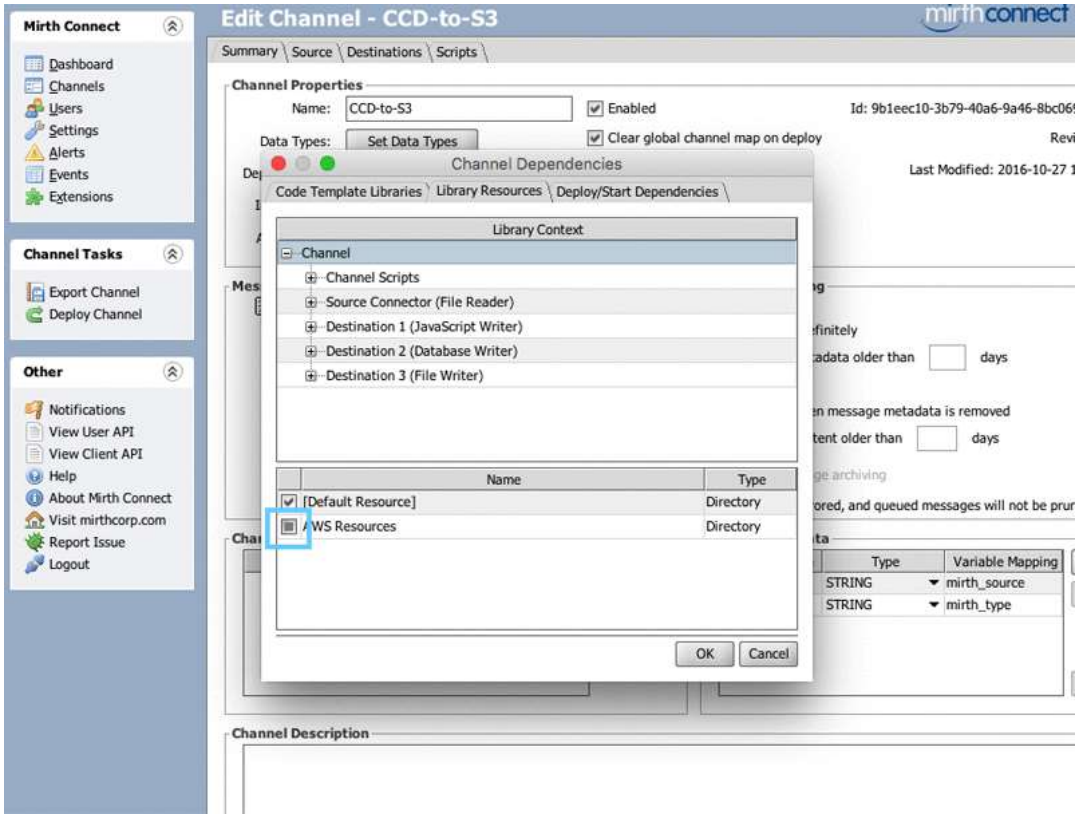
For more information about the deployment and use of the AWS SDK for Java, see Getting Started with the AWS SDK for Java.

To facilitate the use of the example channels, a sample Java class implementation for use with Mirth Connect can be found at https://github.com/awslabs/aws-big-data-blog/tree/master/aws-blog-mirth-healthcare-hub.

For more information about building the code example using Maven, see Using the SDK with Apache Maven.

You can view the addition of these libraries to each channel in the following screenshots.





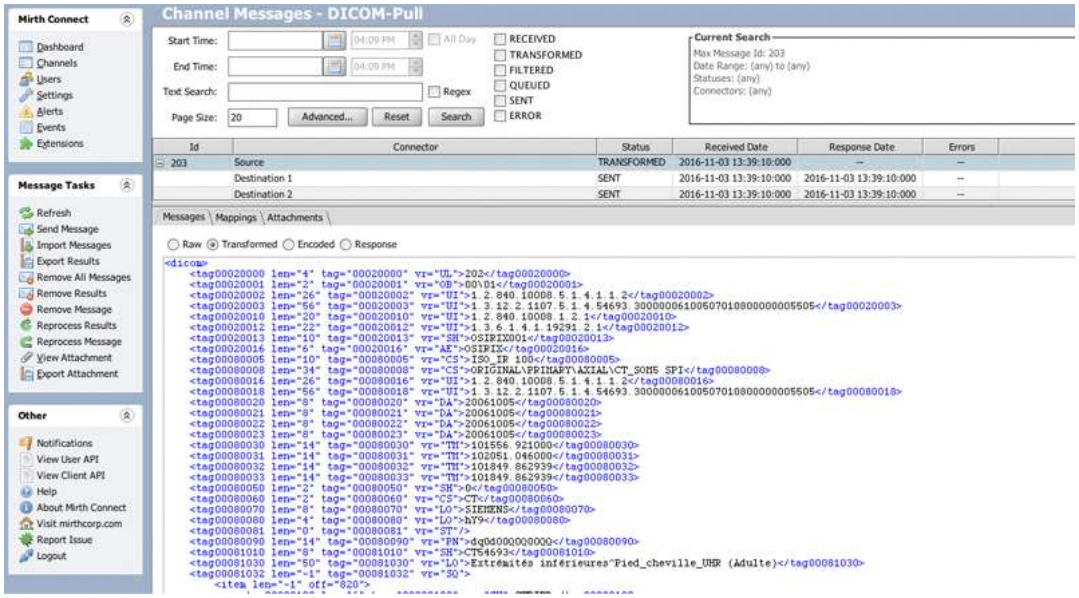
After test messages are processed, you can view the contents in DynamoDB.

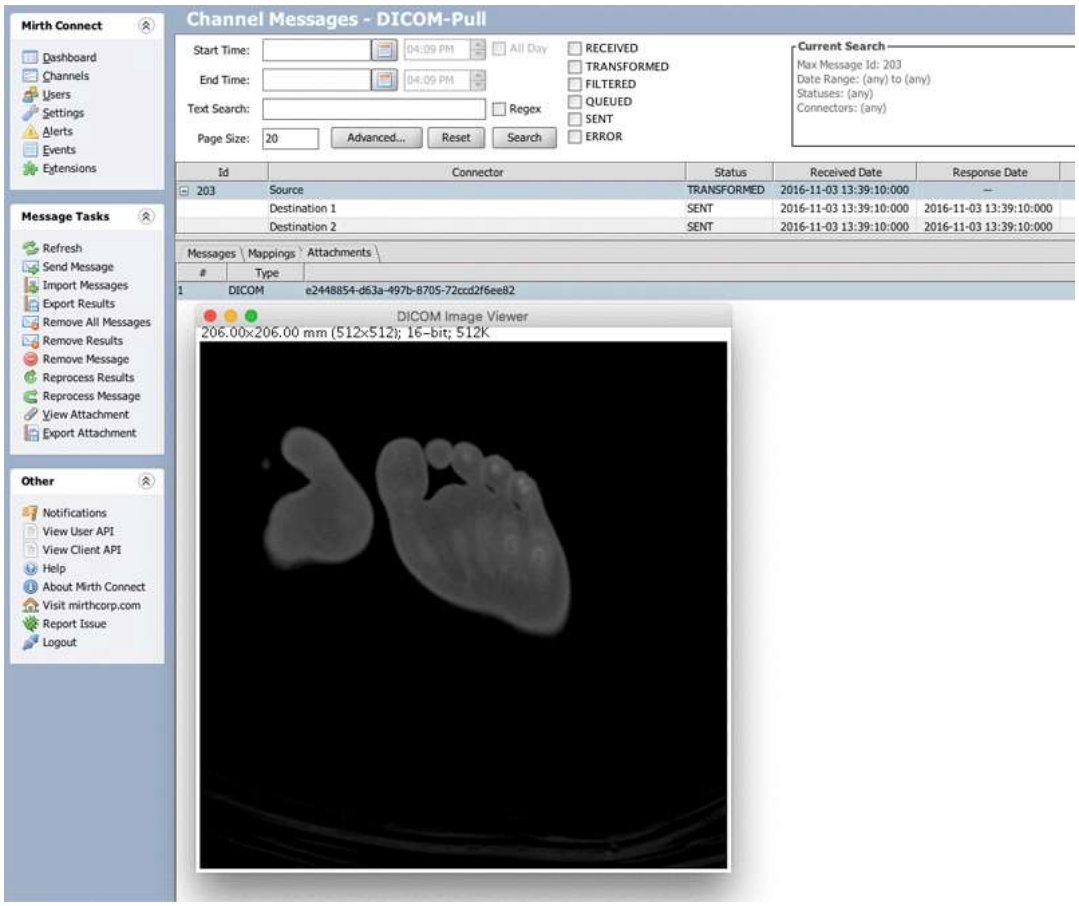
DICOM messages

The dcm4che project is a collection of open source tools supporting the DICOM standard. When a DICOM message is passed through Mirth Connect, the text content is transformed to XML with the image included as an attachment. You convert the DICOM XML into JSON for storage in DynamoDB and store both the original DICOM message as well as the separate attachment in S3, using the AWS SDK for Java and the dcm4che project. The source code for dcm4che can be found in the dcm4che GitHub repository.

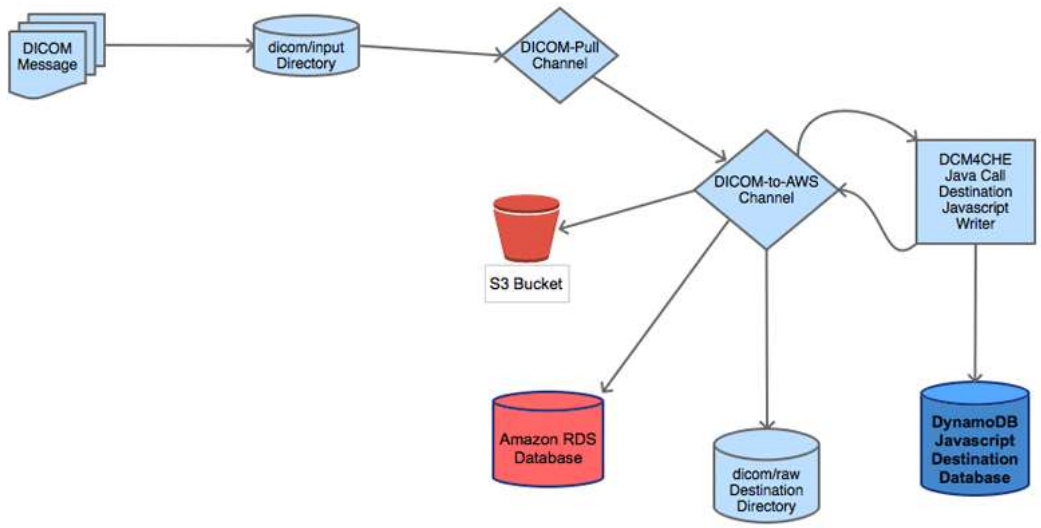
Readers can find the example dcm4che functions used in this example in the mirth-aws-dicom-app in the AWS Big Data GitHub repository.

The following screenshots are examples of the DICOM message data converted to XML in Mirth Connect, along with the view of the attached medical image. Also, you can view an example of the data stored in DynamoDB.





The DICOM process flow can be seen in the following diagram.



Conclusion

With the combination of channels, you now have your original messages saved and available for archive via Amazon Glacier. You also have each message stored in DynamoDB as discrete data elements, with references to each message stored in Amazon RDS for use with message correlation. Finally, you have your transformed messages, processed messages, and image attachments saved in S3, making them available for direct use or archiving.

Later, you can implement analysis of the data with Amazon EMR. This would allow the processing of information directly from S3 as well as the data in Amazon RDS and DynamoDB.

With an open source health information hub, you have a centralized repository for information traveling throughout healthcare organizations, with community support and development constantly improving functionality and capabilities. Using AWS, health organizations are able to focus on building better healthcare solutions rather than building data centers.

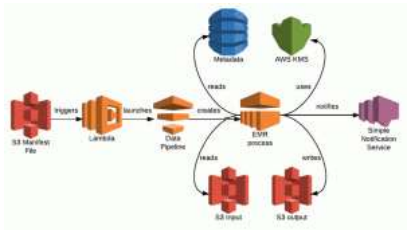
If you have questions or suggestions, please comment below.

About the Author

Joseph Fontes is a Solutions Architect at AWS with the Emerging Partner program. He works with AWS Partners of all sizes, industries, and technologies to facilitate technical adoption of the AWS platform with emphasis on efficiency, performance, reliability, and scalability. Joe has a passion for technology and open source which is shared with his loving wife and two sons.

Related

How Eliza Corporation Moved Healthcare Data to the Cloud



TAGS: Amazon EMR, Healthcare, Mirth Connect

https://github.com/amazon-archives/aws-big-data-blog/tree/master/aws-blog-mirth-healthcare-hub
https://github.com/chb/sample_ccdas/tree/master/Cerner%20Samples

潤沁網路大學

分类: Mirth

标签: Mirth

潤沁網路大學
关注 - 0
粉丝 - 2

posted @ 2021-03-08 17:33 潤沁網路大學 阅读(0) 评论(0) 编辑 收藏

潤沁網路大學