Start Lab 01:30:00

Streaming HL7 to FHIR Data with Dataflow and the Healthcare API

Lab 1 hour No cost Intermediate

GSP894

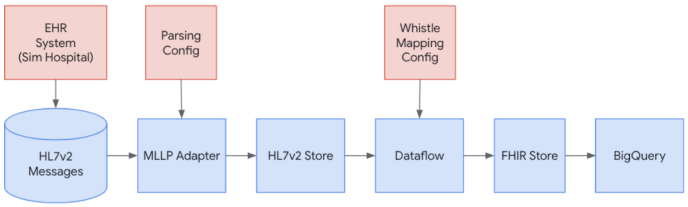
Google Cloud Self-Paced Labs

Overview

In the healthcare industry today, data such as Electronic Healthcare Records (EHR) is in disparate formats and silos and often non-interoperable, meaning the silos cannot be shared with each other. Additionally, the global healthcare community has adopted FHIR as the main format for web applications. Getting the disparate formats into FHIR has been a challenge for many healthcare organizations.

In this lab, you will explore some of the features of the Cloud Healthcare API (HCAPI) and the Healthcare Data Harmonization tool to stream HL7v2 messages into HCAPI datastores and convert HL7v2 to Fast Healthcare Interoperability Resources (FHIR), then import the FHIR data into BigQuery for analytical use.

HL7v2 to FHIR streaming pipeline



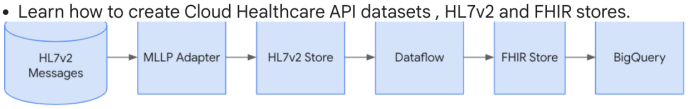
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



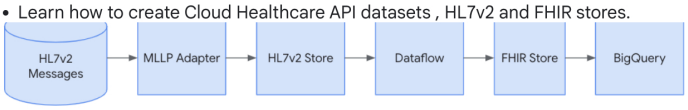
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



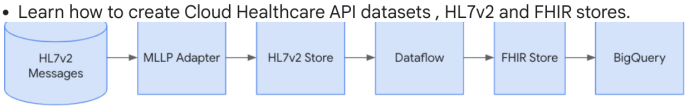
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



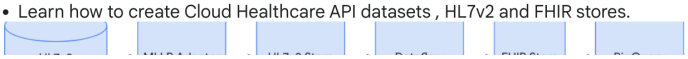
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



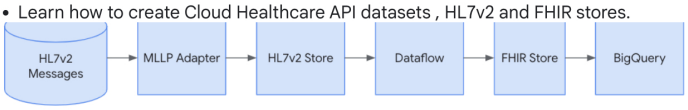
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



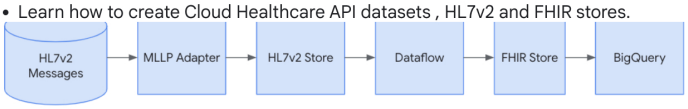
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.



To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended) and a good Internet connection.
- The gcloud and bq utilities installed on the system on which you will run the lab. These utilities are already installed in Qwiklabs and Cloud Shell. If you are completing this lab in another environment, install Google Cloud SDK, which includes these utilities. See Installing Google Cloud SDK for steps.
- Time to complete the lab.

What you learn

In this lab, you will:

- Gain a general understanding of the Cloud Healthcare API and its role in managing healthcare data.
- Learn how to create Cloud Healthcare API datasets , HL7v2 and FHIR stores.

3. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: "ACCOUNT"

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

4. (Optional) You can list the project ID with this command:

```
gcloud config list project
```

Output:

```
[core]
project = "PROJECT_ID"
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Healthcare API introduction

Cloud Healthcare API provides a managed solution for storing and accessing healthcare data in Google Cloud, providing a critical bridge between existing care systems and applications hosted on Google Cloud. Using the API, you can unlock significant new capabilities for data analysis, machine learning and application development, and use these capabilities to build the next generation of healthcare solutions.

The API is comprised of three modality-specific interfaces that implement key industry-wide standards for healthcare data:

- FHIR, an emerging standard for health data interchange
- HL7v2, the most widely adopted method for health systems integration
- DICOM, the dominant standard for radiology and imaging-related disciplines

Each interface is backed by a standards-compliant data store that provides read, write, search, and other operations on the data.

The Cloud Healthcare API provides a number of key features that are critical to bridging current technologies to the next generation of healthcare systems and applications:

- **Standards conformance** - Google supports the use of standards-based interoperability through its participation in a number of healthcare standards bodies. In the Cloud Healthcare API each modality-specific data store and its associated API is substantially conformant with its relevant standard. For example, FHIR stores implement STU3, the current version of the FHIR specification, and DICOM stores implement DICOMweb, a web-based standard for exchanging medical images. In future updates, we expect to support additional versions of these specifications as well as the ability to request a resource in a different version than its canonical representation.
- **Compliance with privacy regulations** - Google Cloud provides detailed guidance regarding how it supports compliance with HIPAA in the US, the PIPEDA in Canada, and other global privacy standards at cloud.google.com/security/compliance.
- **Data location control** - The Cloud Healthcare API treats data location as a core component of the API. You have the option to select the storage location for each dataset from a list of currently available locations which correspond to distinct geographic areas aligned with Google Cloud's regional structure. Future Google Cloud regions will allow for the distribution of storage across wider geographic areas.
- **Security** - The Cloud Healthcare API security model is based on Google's proven Identity and Access Management (IAM) system. IAM's fine-grained permissions give you complete control over access to your healthcare data. In addition, we've created

open-source proxies for our powerful Apigee API Management system, which provides comprehensive threat detection and traffic management capabilities that allow you to securely expose sensitive ePHI with patient and provider applications.
- **Bulk import and export** - The Cloud Healthcare API's DICOM and FHIR modalities support bulk import and export of data, making it easier to transfer data via the Cloud Storage system.
- **De-identification** - De-identification support for DICOM is available, making it much easier to redact patient information from studies for research and other purposes. The de-identification process operates on a data store basis.
- **Auditability** - Both administrative and data access requests to the Cloud Healthcare API can be audited. Logs are available through Google Cloud's Stackdriver hybrid monitoring system.
- **High availability** - Availability for mission-critical scenarios is made possible through Google Cloud's robust and highly redundant infrastructure.

For many applications, the Cloud Healthcare API can provide a modern alternative to legacy stacks implementing DICOM, HL7v2 or FHIR STU3 standards, simplifying data integration with existing systems and enabling the application developers to focus on their differentiating features such as UX and intelligence.

Healthcare API concepts

To get the most out of the Cloud Healthcare API, there are a few key concepts you'll want to understand. The information below should give you a good sense of Cloud Healthcare API capabilities, but you can find more details in the Cloud Healthcare API documentation.

General structure of the Cloud Healthcare API

The Cloud Healthcare API exposes interfaces that enable you to perform different types of functions:

- **Administrative functions**, such as creating or listing datasets and stores that will contain your data.
- **Data access functions** that allow you to create, update, delete and search the data stored in Cloud Healthcare API, or to perform bulk import and export operations.
- **Security functions** that allow you to impose access controls on data stored in Cloud Healthcare API.
- **De-identification functions** that allow you to replace ePHI with anonymized data, or to obfuscate ePHI so that it cannot be used.
- **Metadata functions**, such as retrieval of a FHIR capabilities statement for the FHIR API.

These functions may vary slightly depending on the modality of data (FHIR, HL7 v2 or DICOM) being operated on. For example, data retrieval operations against an FHIR data store use an API that conforms to the FHIR standard, but data retrieval operations against an HL7 v2 store use operations better suited to operating on HL7v2-structured data.

Datasets and stores

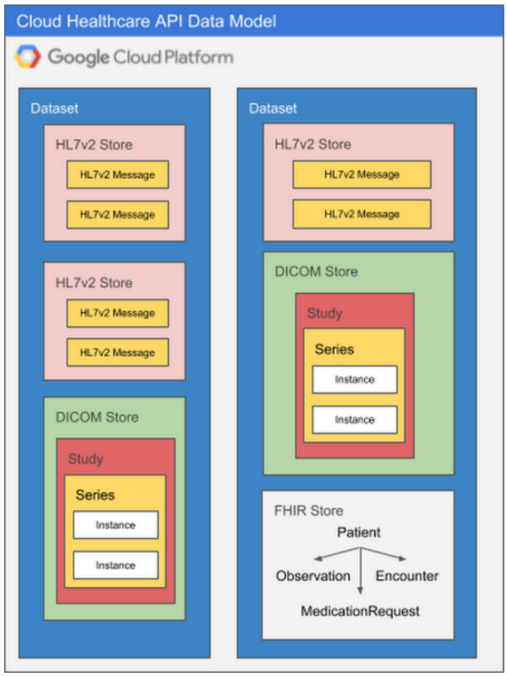
All Cloud Healthcare API usage occurs within the context of a Google Cloud project. Projects form the basis for creating, enabling, and using all Google Cloud services including managing APIs, enabling billing, adding and removing collaborators, and managing permissions for Google Cloud resources. Cloud Healthcare API can be used in one or many Google Cloud projects, as appropriate; this flexibility allows you to separate production from non-production usage, for example, or to segregate applications and resources in order to better manage access or accommodate different development lifecycles.

Within a project, data ingested through Cloud Healthcare API is stored in a **dataset**, which resides in a geographic location corresponding to a specific Google Cloud region

which resides in a geographic location corresponding to a specific Google Cloud region. You use the Cloud Healthcare API's administrative functions to create a dataset in a particular location; doing so facilitates implementation of data location requirements for the countries in which your applications provide services. For example, you can choose to create a dataset in Google Cloud's "us-central1" region for US-based applications, or in an EU or UK region for applications serving those customers. This level of location control is also available in other Google Cloud products, which can be combined with Cloud Healthcare API to create a complete application architecture. A list of generally available Google Cloud products and the regions in which they are implemented can be found on Google Cloud, Cloud locations.

Because each healthcare data modality has different structural and processing characteristics, datasets are split into modality-specific **stores**. A single dataset can contain one or many stores, and those stores can all service the same modality or different modalities as application needs dictate. Using multiple stores in the same dataset might be appropriate if a given application processes different types of data, for example, or if you'd like to be able to separate data according to its source hospital, clinic, department, etc. An application can access as many datasets or stores as its requirements dictate with no performance penalty, so it's important to design your overall dataset and store architecture to meet the organization's broad goals for locality, partitioning, access control, and so on.

The diagram below illustrates two datasets in a Google Cloud project, each of which contains multiple stores.



There are many ways to structure datasets and stores. As you design systems that use the Cloud Healthcare API, you may want to take the following into consideration:

- **Security and access control:** Rules can be defined at both a dataset and store level, but you may choose to group all data for a particular application into the same dataset, and set access control rules such that only that application can access the dataset.
- **Application requirements:** An application processing different types of data may have all of its data for all modalities in a single dataset.
- **Source systems:** Often, the structure of healthcare data can vary according to the source system and modality. Separating data for different source systems into their own datasets may facilitate processing.
- **Intended use:** Data from different systems can have different intended uses, such as research, analytics, or machine learning predictions. Grouping data by intended use may facilitate ingestion into the target system.
- **Separating ePHI from de-identified data:** Cloud Healthcare API data de-identification functions read from a source dataset and write the output into a new dataset that you specify. If you are preparing data to be used by researchers, for example, this approach to de-identifying data may be a consideration in how you use datasets to segregate data.

MLLP Adapter

The minimal lower layer protocol (MLLP) is the standard used for transmitting HL7v2 messages over TCP/IP connections within a network, such as a hospital.

MLLP does not offer an exact mapping to the Cloud Healthcare API HL7v2 REST API], which uses HTTP. Therefore, an MLLP adapter must be used to convert messages transmitted over MLLP into a format that an HTTP/REST API can accept. To transmit messages over MLLP and then to the Cloud Healthcare API, use the Google Cloud MLLP adapter.

Task 1. Set SDK defaults

- Set the region and zone:

```
gcloud config set compute/region "REGION"
gcloud config set compute/zone "ZONE"
```

Task 2. Enable APIs needed

- Enable the required Google Cloud APIs needed for the lab by running the following command:

```
gcloud services enable compute.googleapis.com
container.googleapis.com dataflow.googleapis.com
bigquery.googleapis.com pubsub.googleapis.com
healthcare.googleapis.com
```

Task 3. Set environment variables

- Export the variables needed for the lab by running the following commands:

```
export PROJECT_ID=$(gcloud config list --format
'value(core.project)')
export PROJECT_NUMBER=$(gcloud projects list --
filter=${PROJECT_ID} --format='value(PROJECT_NUMBER)')
export COMPUTE_SA=${PROJECT_NUMBER}-compute
export LOCATION=$(gcloud config get-value compute/region)
export DATASET_ID=datastore
export FHIR_STORE_ID=fhirstore
export FHIR_TOPIC=fhirtopic
export HL7_TOPIC=hl7topic
export HL7_SUB=hl7subscription
export HL7_STORE_ID=hl7v2store
export BQ_FHIR=fhirdata
export PSSAN=pubsub-svc
export FILENAME=svca-key
export DEFAULT_ZONE=$(gcloud config get-value compute/zone)
export ERROR_BUCKET=${PROJECT_ID}-df-pipeline/error/
```

```
export MAPPING_BUCKET=${PROJECT_ID}-df-pipeline/mapping
```

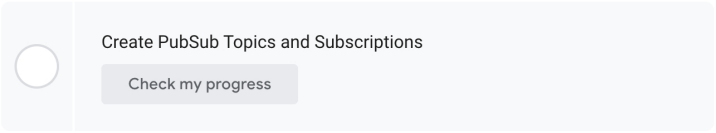
Task 4. Create PubSub topics and subscriptions

This lab uses PubSub as a message bus for HL7v2/FHIR messages that are created from the Simulated Hospital. Once the topic is created, you'll also create a Subscription to this topic. This will provide your Dataflow Job, which you will create in later steps, to take these messages and convert them to FHIR.

- The commands below will create the HL7v2/FHIR topic and the HL7v2 subscription:

```
gcloud pubsub topics create ${HL7_TOPIC}
gcloud pubsub subscriptions create ${HL7_SUB} --
topic=${HL7_TOPIC}
gcloud pubsub topics create ${FHIR_TOPIC}
```

Click *Check my progress* to verify the objective.



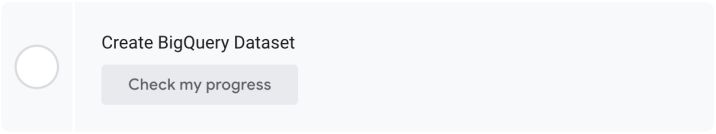
Task 5. Create BigQuery dataset

BigQuery will be used as the end destination for the FHIR data. This allows you to perform analytics on the data in near real-time.

- Create the BigQuery dataset for your FHIR messages:

```
bq mk --dataset --location=${LOCATION} --
project_id=${PROJECT_ID} --description HCAPI-FHIR-dataset
${PROJECT_ID}:${BQ_FHIR}
```

Click *Check my progress* to verify the objective.



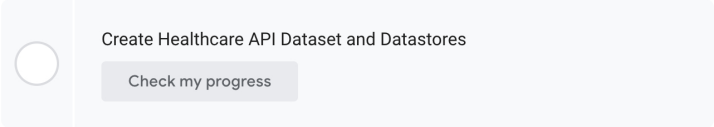
Task 6. Create Healthcare API dataset and datastores

The Healthcare API Dataset is where the HL7v2 and FHIR data will be stored and managed.

- Create the datasets where the FHIR and HL7v2 stores will reside:

```
gcloud healthcare datasets create ${DATASET_ID} --
location=${LOCATION}
gcloud healthcare hl7v2-stores create ${HL7_STORE_ID} \
--dataset=${DATASET_ID} \
--location=${LOCATION} \
--notification-config=pubsub-
topic=projects/${PROJECT_ID}/topics/${HL7_TOPIC}
gcloud healthcare fhir-stores create ${FHIR_STORE_ID} \
--dataset=${DATASET_ID} \
--location=${LOCATION} \
--version=R4 \
--pubsub-topic=projects/${PROJECT_ID}/topics/${FHIR_TOPIC} \
--disable-referential-integrity \
--enable-update-create
```

Click *Check my progress* to verify the objective.



Task 7. Create service account and add IAM binding

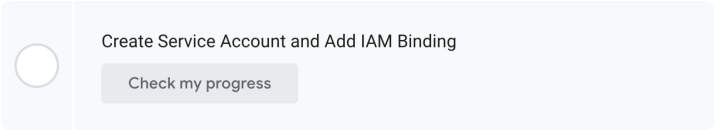
The service account will be used to manage PubSub and HL7v2 ingest. You will also set up the appropriate permissions to enable exporting data from your FHIR store to BigQuery. This involves granting the Healthcare API service account (created per project) to access BigQuery, HC API Datastores, and Google Cloud buckets where you will store the sample data.

- Finally, the Compute Engine service account is also granted the roles allowing access to HC API, Cloud Storage, and PubSub for the Dataflow job:

```
gcloud iam service-accounts create ${PSSAN}
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${PSSAN}@${PROJECT_ID}.iam.gserviceaccount.cc
\
--role=roles/pubsub.subscriber
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${PSSAN}@${PROJECT_ID}.iam.gserviceaccount.cc
\
--role=roles/healthcare.hl7V2Ingest
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${PSSAN}@${PROJECT_ID}.iam.gserviceaccount.cc
\
--role=roles/monitoring.metricWriter
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-
healthcare.iam.gserviceaccount.com \
--role=roles/bigquery.dataEditor
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-
healthcare.iam.gserviceaccount.com \
--role=roles/bigquery.jobUser
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-
healthcare.iam.gserviceaccount.com \
--role=roles/storage.objectAdmin
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
```

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-
healthcare.iam.gserviceaccount.com \
--role=roles/healthcare.datasetAdmin
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-
healthcare.iam.gserviceaccount.com \
--role=roles/healthcare.dicomStoreAdmin
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${COMPUTE_SA}@developer.gserviceaccount.com
\
--role=roles/pubsub.subscriber
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${COMPUTE_SA}@developer.gserviceaccount.com
\
--role=roles/editor
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${COMPUTE_SA}@developer.gserviceaccount.com
\
--role=roles/healthcare.hl7V2Consumer
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--
member=serviceAccount:${COMPUTE_SA}@developer.gserviceaccount.com
\
--role=roles/healthcare.fhirResourceEditor
```

Click *Check my progress* to verify the objective.



Task 8. Create Storage buckets for mapping configs

When you create the dataflow job in later steps, it will make use of mapping files to convert HL7v2 over to FHIR (R4).

- Run this code to create the Cloud Storage bucket, clones the git repo, modifies the path to the mapping, and then uploads the modified files into the bucket:

```
gsutil mb gs://${PROJECT_ID}-df-pipeline
git clone https://github.com/GoogleCloudPlatform/healthcare-
data-harmonization
sed -i 's|\$MAPPING_ENGINE_HOME|gs://'"${MAPPING_BUCKET}"'|g'
healthcare-data-
harmonization/wstl1/mapping_configs/hl7v2_fhir_r4/configurations/ma
sed -i 's|local_path|gcs_location|g' healthcare-data-
harmonization/wstl1/mapping_configs/hl7v2_fhir_r4/configurations/ma
gsutil -m cp -r healthcare-data-
harmonization/wstl1/mapping_configs gs://${PROJECT_ID}-df-
pipeline/mapping/mapping_configs
touch empty.txt
gsutil cp empty.txt gs://${PROJECT_ID}-df-
pipeline/error/empty.txt
gsutil cp empty.txt gs://${PROJECT_ID}-df-
pipeline/temp/empty.txt
```

Click *Check my progress* to verify the objective.

Create Cloud Storage Buckets for Mapping Configs

Check my progress

Task 9. Updating configurations of the HC API datastores

Execute the following `curl` commands on the Cloud Shell command line. This will patch the respective datastores to modify their behaviors.

1. Patch HL7v2 store to parse HL7v2 to JSON:

```
curl -X PATCH \
   -H "Authorization: Bearer $(gcloud auth application-default
print-access-token)" \
   -H "Content-Type: application/json; charset=utf-8" \
   --data "{
     'parserConfig': {
       'schema': {
         'schematizedParsingType': 'HARD_FAIL',
         'ignoreMinOccurs' :true,
         'unexpectedSegmentHandling' : 'PARSE'
       }
     }
   }"
"https://healthcare.googleapis.com/v1/projects/${PROJECT_ID}/locati
updateMask=parser_config.schema"
```

2. Enable Streaming from FHIR Store to BigQuery:

```
curl -X PATCH \
  -H "Authorization: Bearer $(gcloud auth application-default
print-access-token)" \
  -H "Content-Type: application/json; charset=utf-8" \
  --data "{
 'streamConfigs': [
   { 'bigqueryDestination': {
     'datasetUri': 'bq://${PROJECT_ID}.${BQ_FHIR}',
'schemaConfig': {
       'schemaType': 'ANALYTICS'
       }
     }
   }
 ]
}"
"https://healthcare.googleapis.com/v1/projects/${PROJECT_ID}/locati
updateMask=streamConfigs"
```

Task 10. Create GKE cluster

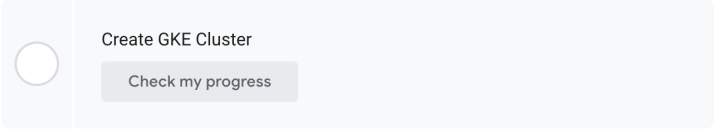
A GKE (K8S) cluster is needed to run the SimHospital, Dataflow Job Creator, and the MLLP Adapter containers. For the purposes of lab, you have been granted full API access to this cluster. Create a Kubernetes Cluster by executing this code block in the Cloud Shell Command Line. This command will take at least five minutes to execute. You can follow along with the creation of this K8S cluster by navigating from the *_Main Menu* > **Kubernetes Engine** > **Clusters**

- In a production environment, please follow least-priviledges principles to support good security practices:

```
gcloud container clusters create "hl7-fhir-gke" \
--project ${PROJECT_ID} \
--zone ${DEFAULT_ZONE} \
--no-enable-basic-auth \
--cluster-version "$(gcloud container get-server-config --
flatten="channels" --filter="channels.channel=STABLE" --
format="yaml(channels.channel,channels.defaultVersion)" --
zone=${DEFAULT_ZONE} --verbosity=none | grep default | awk
'{print $2}')" \
--release-channel "stable" \
--machine-type "e2-standard-4" \
--image-type "COS_CONTAINERD" \
--disk-type "pd-standard" \
--disk-size "100" \
--metadata disable-legacy-endpoints=true \
--scopes "https://www.googleapis.com/auth/cloud-platform" \
--max-pods-per-node "110" \
--enable-autoscaling \
--num-nodes "3" \
--min-nodes "2" \
--max-nodes "3" \
--logging=SYSTEM,WORKLOAD \
--monitoring=SYSTEM \
--enable-ip-alias \
--network "projects/${PROJECT_ID}/global/networks/default" \
--subnetwork
"projects/${PROJECT_ID}/regions/${LOCATION}/subnetworks/default"
\
--no-enable-intra-node-visibility \
--default-max-pods-per-node "110" \
--no-enable-master-authorized-networks \
--addons
HorizontalPodAutoscaling,HttpLoadBalancing,GcePersistentDiskCsiDriv
\
--enable-autoupgrade --enable-autorepair --max-surge-upgrade 1 -
-max-unavailable-upgrade 0 --enable-shielded-nodes \
--node-locations ${DEFAULT_ZONE}
```

This step may take about 7 minutes to complete. You can check to see that the Kubernetes Cluster has been created by navigating to **Navigation Menu** > **Kubernetes Engine** > **Clusters** and noting that the cluster is up and the status is ready.

Click *Check my progress* to verify the objective.



Task 11. Setup auth for kubectl

- Once the cluster is created, you need to set up your authentication to be able to use `kubectl` to manage the K8S cluster from the command line. Copy and paste this code block into the Cloud Shell command line:

```
gcloud container clusters get-credentials hl7-fhir-gke
```

Task 12. Create deployment & service for MLLP adapter

HL7v2 pre-dates the regular use of the TCP/IP protocol. Because of this, the lab needs to use a MLLP adapter or gateway to move the data from the HL7v2 storage to the Dataflow Pipeline before it can be go to both the Healthcare FHIR Datastore, and the BigQuery Dataset. Because of this, once the command is given the service must start, and then the load balancer endpoint created. This will take around five minutes to accomplish all of these tasks. If you move to the next step before this is completed, the rest of the lab will not work properly.

1. Copy and paste this code block into the Cloud Shell command line to run the following, which will deploy a service for the MLLP Adapter on the GKE cluster:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
 name: mllp-adapter-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: mllp-adapter
 template:
   metadata:
     labels:
       app: mllp-adapter
   spec:
     containers:
       - name: mllp-adapter
         imagePullPolicy: Always
         image: gcr.io/cloud-healthcare-containers/mllp-adapter
         ports:
           - containerPort: 2575
             protocol: TCP
             name: "port"
         command:
           - "/usr/mllp_adapter/mllp_adapter"
           - "--port=2575"
           - "--hl7_v2_project_id=${PROJECT_ID}"
           - "--hl7_v2_location_id=${LOCATION}"
           - "--hl7_v2_dataset_id=${DATASET_ID}"
           - "--hl7_v2_store_id=${HL7_STORE_ID}"
           - "--logtostderr"
           - "--receiver_ip=0.0.0.0"
EOF
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
 name: mllp-adapter-service
 annotations:
   cloud.google.com/load-balancer-type: "Internal"
spec:
 type: LoadBalancer
 ports:
 - name: port
   port: 2575
   targetPort: 2575
   protocol: TCP
 selector:
   app: mllp-adapter
EOF
```

2. Validate that the MLLP Adaptor Service is Running by navigating to **Navigation Menu** > **Kubernetes Engine** > **Services and Ingress** and noting that the MLLP Adapter has been assigned an external IP address. This indicates that the endpoint for the load balancer has been created and is running. Check that there is an external IP address assigned to this service before moving on to the next

is an external IP address assigned to this service before moving on to the next step:

```
kubectl get pods
```

Output:

```
NAME                                    READY   STATUS      RESTARTS
AGE
mllp-adapter-deployment-6db77874fc-txzps   1/1     Running     0
5m
```

Click *Check my progress* to verify the objective.

Create Deployment & Service for MLLP Adapter

Check my progress

Task 13. Create SimHospital deployment

1. The SimHospital application that will generate simulated HL7v2 data that will be used to test the Dataflow pipeline. In the Cloud Shell command, copy the following code block and paste it on the command line:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
 name: simhospital-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: simhospital
 template:
   metadata:
     labels:
       app: simhospital
   spec:
     containers:
       - name: simhospital
         imagePullPolicy: Always
         image: us-docker.pkg.dev/qwiklabs-resources/healthcare-
qwiklabs-resources/simhospital:latest
         ports:
           - containerPort: 8000
             protocol: TCP
             name: "port"
         command: ["/health/simulator"]
         args: ["-output=mllp", "-mllp_destination=$(kubectl get
svc | grep mllp-adapter | awk {print'$4'}):2575"]
EOF
```

The following Cloud Shell code block will show the status of the SimHospital Service in your GKE cluster.

```
kubectl get pods
```

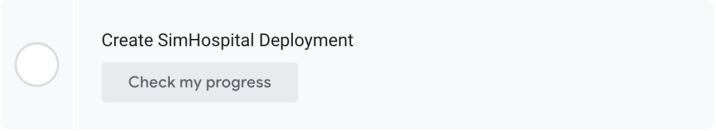
2. Validate that the SimHospital pod is Running before moving on to the next step. You can observe the SimHospital Service is running by navigating from **Main**

Menu > **Kubernetes Engine** > **Workloads** and seeing that both the MLLP and SimHospital services are running. Th SimHospital service usually gets started in about than 3 minutes.

Output:

```
NAME                                    READY   STATUS      RESTARTS
AGE
simhospital-deployment-9fff9c8c6-mqhwx  1/1     Running     0
1m
```

Click *Check my progress* to verify the objective.



Task 14. Create Dataflow job

1. Now that the container is ready, you can run the Dataflow job against the k8s cluster:

```
cat <<EOF | kubectl replace --force -f -
apiVersion: v1
kind: Pod
metadata:
 labels:
   run: dataflow-pipeline
 name: dataflow-pipeline
spec:
 containers:
 - command:
   - /usr/local/openjdk-11/bin/java
   - -jar
   - /root/converter-0.1.0-all.jar
   - --
pubSubSubscription=projects/${PROJECT_ID}/subscriptions/${HL7_SUB}
   - --readErrorPath=gs://${ERROR_BUCKET}read/read_error.txt
   - --writeErrorPath=gs://${ERROR_BUCKET}write/write_error.txt
   - --
mappingErrorPath=gs://${ERROR_BUCKET}mapping/mapping_error.txt
   - --
mappingPath=gs://${MAPPING_BUCKET}/mapping_configs/hl7v2_fhir_r4/cc
   - --
fhirStore=projects/${PROJECT_ID}/locations/${LOCATION}/datasets/${D
   - --tempLocation=gs://${PROJECT_ID}-df-pipeline/temp
   - --runner=DataflowRunner
   - --project=${PROJECT_ID}
   - --region=${LOCATION}
   image: us-docker.pkg.dev/qwiklabs-resources/healthcare-
qwiklabs-resources/dataflow-pipeline:v0.02
   imagePullPolicy: Always
   name: dataflow-pipeline
 restartPolicy: Never
EOF
```

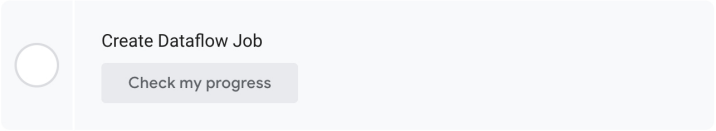
2. Validate that the Dataflow pod is **Running** and not **Completed** or **Succeeded** by navigating to **Main Menu** > **Kubernetes Engine** > **Workloads** before moving on to the next step. It might take a few seconds after you run the job for it to appear in the list. If the Dataflow job failed to start to stream the data the status will be **Succeeded** or **Completed**. Try to restart the job either by cloning it by navigating to **Main Menu** > **Dataflow** > **Jobs** and selecting **Clone**, the job can also be restarted by pasting the code block into the Cloud Shell again.

```
gcloud dataflow jobs list --region ${LOCATION}
```

Output:

```
JOB_ID                                    NAME
TYPE       CREATION_TIME        STATE    REGION
2021-05-03_13_51_57-14284187373112084237  hl7v2tofhirstreamingrunner-
root-0503205154-138576fd  Streaming  2021-05-03 20:51:59  Running  us-
east1
```

Click *Check my progress* to verify the objective.



Task 15. Testing

Validate HL7 data creation

- Now that the SimHospital application has been deployed, make sure that HL7v2 data is stored in the HL7v2 datastore:

```
curl -X GET \
     -H "Authorization: Bearer "$(gcloud auth print-access-
token) \
     -H "Content-Type: application/json; charset=utf-8" \

"https://healthcare.googleapis.com/v1/projects/${PROJECT_ID}/locati
```

Validate FHIR data

1. With confirmation that the HL7v2 data exists in the HL7v2 datastore, make sure that the data has also been ingested into the FHIR datastore. A positive validation should be a JSON block returned with no error messages.

Note: It can take 5 to 7 minutes for messages to start showing in the FHIR view.

2. Using the **Navigation menu**, navigate to **Healthcare** > **FHIR viewer**.
3. In the FHIR Store drop down select **datastore** > **fhirstore**.
4. In the Filter search type **Patient** for Resource type.
5. Click on the **Patient** under Resource Type, then you will see the data.

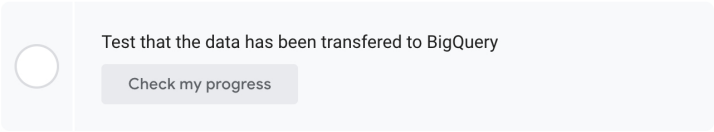
Explore the data in BigQuery

The data should also be ingested into BigQuery so that you can access it using SQL commands.

1. Using the **Navigation menu**, navigate to **BigQuery**.
2. Under resources find your **project ID** and then expand the drop-down

2. Under resources find your **project ID** and then expand the drop-down.
3. Under the drop-down find **fhirdata** and then expand the drop-down.
4. Select the **Patient** table under the drop down and then navigate to the **Preview** section, where it will display the recent exported data. You might have to scroll right to see the data in the cells.

Click *Check my progress* to verify the objective.



Lab review

Cloud Healthcare API provides a comprehensive facility for ingesting, storing, managing, and securely exposing healthcare data in FHIR, DICOM, and HL7 v2 formats. Using Cloud Healthcare API, you can ingest and store data from electronic health records systems (EHRs), radiological information systems (RISs), and custom healthcare applications. You can then immediately make that data available to applications for analysis, machine learning prediction and inference, and consumer access.

Cloud Healthcare API enables application access to healthcare data via widely-accepted, standards-based interfaces such as FHIR STU3 and DICOMweb. These APIs allow data ingestion into modality-specific stores, which support data retrieval, update, search and other functions using familiar standards-based interfaces.

Further, the API integrates with other capabilities in Google Cloud through two primary mechanisms:

- Cloud Pub/Sub, which provides near-real-time updates when data is ingested into a Cloud Healthcare API data store, and
- Import/export APIs, which allow you to integrate Cloud Healthcare API into both Google Cloud Storage and Google BigQuery.

Using Cloud Pub/Sub with Google Cloud Functions enables you to invoke machine learning models on healthcare data, storing the resulting predictions back in Cloud Healthcare API data store. A similar integration with Cloud Dataflow supports transformation and cleansing of healthcare data prior to use by applications.

To support healthcare research, Cloud Healthcare API offers de-identification capabilities for FHIR and DICOM. This feature allows customers to share data with researchers working on new cutting-edge diagnostics and medicines.

Congratulations!

In this lab, you:

- Gained a general understanding of Cloud Healthcare API and its role in managing healthcare data.
- Learned how to create datasets and FHIR stores.
- Imported and exported FHIR data.

Finish your quest

This self-paced lab is part of the Cloud Healthcare API quest. A quest is a series of

This self-paced lab is part of the Cloud Healthcare API quest. A quest is a series of related labs that form a learning path. Completing this quest earns you a badge to recognize your achievement. You can make your badge or badges public and link to them in your online resume or social media account. Enroll in this Quest or any quest that contains this lab and get immediate completion credit. See the Google Cloud Skills Boost catalog to see all available quests.

Take your next lab

Continue your quest with Ingesting DICOM Data with the Healthcare API or try one of these suggestions:

- Ingesting FHIR Data with the Healthcare API
- De-identifying DICOM Data with the Healthcare API

End your lab

When you have completed your lab, click **End Lab**. Your account and the resources you've used are removed from the lab platform.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.

Manual Last Updated February 1, 2024

Lab Last Tested February 1, 2024