Mapping Data to FHIR

Alex Fragotsis
League Inc.
@alexfragotsis
Building the Leading Healthcare CX Platform Company

- Founded in 2014
- The platform used by industry leaders to accelerate transformation
- Reaching over 14 million members across the healthcare ecosystem
Mapping Data to FHIR

(How I tricked my co-workers to use dataflow without even knowing it)
What is FHIR
Fast Healthcare Interoperability Resources

- HTTP RESTful
  - Single / Bundle
- JSON / XML
- Interoperability
  - Modeled and Documented Resources
"resourceType": "Patient",
"id": "", // from Resource: id, meta, implicitRules, and language
"meta": {"resourceType": "Meta"}, // from DomainResource: text, contained, extension, and modifierExtension
"identifier": ["Identifier"], // An identifier for this patient
"active": "true", // Whether this patient's record is in active use
"name": ["HumanName"], // A name associated with the patient
"telecom": ["ContactPoint"], // A contact detail for the individual
"gender": "male", // male | female | other | unknown
"birthDate": "1990-01-01", // The date of birth for the individual
"deceased": "false", // Indicates if the individual is deceased or not. One of these 2:
"deceasedBoolean": "true", // A boolean indicating if the individual is deceased
"deceasedDateTime": "1990-01-01T00:00:00Z", // A date time indicating if the individual is deceased
"address": ["Address"], // An address for the individual
"maritalStatus": ["HumanName"], // Marital (civil) status of a patient
"multipleBirth": "false", // Whether patient is part of a multiple birth. One of these 2:
"multipleBirthBoolean": "true", // A boolean indicating if the patient is part of a multiple birth
"multipleBirthInteger": "1", // A number indicating if the patient is part of a multiple birth
"photo": ["Attachment"], // Image of the patient
"organization": ["Reference(Organization)"], // An organization associated with the patient
"location": ["Reference(Location)"], // A location associated with the patient
"communication": ["Communication"], // A language which may be used to communicate with the patient
"language": ["CodeableConcept"], // The language which can be used to communicate with the patient
"preferred": "true", // Language preference indicator
"generalPractitioner": ["Reference(Organization)"], // A primary care provider
"managingOrganization": ["Reference(Organization)"], // An organization that is the custodian of the patient record
"link": ["ResourceReference"], // A link to a Patient or RelatedPerson resource that concerns the same actual individual
"other": ["Reference(Patient|RelatedPerson)"], // Other patient or related person resource that the link refers to
"type": "Patient" // R! replaced-by | replaces | refer | seealso
The problem
PubSub to CHAPI
(Cloud Healthcare API)

- Data Eng + Product Team
  Understand the data

- Map
  Write a “mapper” dataflow job in Python

- Deploy
  Build CI/CD for test and prod
The solution
PubSub to CHAPI
(Cloud Healthcare API)
PubSub to CHAPI

- Read from Pub/Sub
- Python
- Write to Healthcare API
PubSub to CHAPI

Template Repo

Mapper Repo

- Read from Pub/Sub
- Python
- Write to Healthcare API
- Cloud Storage
def import_code(self, code, name):
    # create blank module
    module = types.ModuleType(name)
    # populate the module with code
    exec(code, module.__dict__)
    return module

def check_inputs(self, code):
    ...

def setup(self):
    if not self.udf_module:
        matches = re.match("gs://([^/]+)/.*.py", self.input_python_udf_gcs_path)
        if not matches:
            raise Exception("Invalid Python UDF path")

        bucket_name, object_name = matches.groups()

        logging.info("Downloading {bucket_name} {object_name}")

        # Download udf from gs
        client = Client(project=self.project)
        bucket = client.get_bucket(bucket_name)
        blob = bucket.get_blob(object_name)
        self.code = blob.download_as_string().decode("utf-8")

        forbidden_lib = self.check_inputs(self.code)
        if forbidden_lib:
            raise Exception(f"UDF uses forbidden import {forbidden_lib}")

        self.udf_module = self.import_code(self, code, 'udf_main')
def process(self, element):
    try:
        result = self.udf_module.udf_main(element, self.settings)
        # In order to support functions that use "return" and "yield"
        if isinstance(result, Generator):
            yield from result
        elif result:  # Check if the function returned anything
            yield result
    except Exception as e:
        logging.info(f"Exception = {e}")
        yield pvalue.TaggedOutput(self.TAG_EXCEPTION, ...)

Processing messages
Write to CHAPI

Group Message...hard key
Running
3 min 15 sec
2 stages

Write to Chapi
Running
6 min 33 sec
1 stage
Write Bundles
Batch Mode

Beam Summit 2022 - Unified Streaming and Batch Pipelines at LinkedIn using Beam
Batch Mode

Beam Summit 2022 - Unified Streaming and Batch Pipelines at LinkedIn using Beam

```python
# 1. Select Input
if options.input_query:
    # Read from BQ for backfilling data
    # SELECT * FROM `some_dataset.fhir_resources.Patient` LIMIT 1000
    messages = {
        pipeline
        | 'Query BQ Table' >> beam.io.ReadFromQuery(query=options.input_query, use_standard_sql=True, use_json_export=True)
    | "convert to PubsubObject" >> beam.Map(lambda elem: PubsubMessage(json.dumps(elem).encode('utf-8'), {}))
    | 'Add Timestamps' >> beam.Map(lambda x: x.window().timestampedValue(x, time.time()))
}

elif options.input_gcs_filepath:
    messages = {
        pipeline
        | 'Read From GCS' >> beam.io.ReadFromText(options.input_gcs_filepath)
        | "convert to PubsubObject" >> beam.Map(lambda elem: PubsubMessage(json.dumps(elem).encode('utf-8'), {}))
        | 'Add Timestamps' >> beam.Map(lambda x: x.window().timestampedValue(x, time.time()))
    }

else:
    # Read from PubSub
    messages = {
        pipeline
        | f'"Read PubSub Messages {options.input_subscription}"' >> beam.io.ReadFromPubSub(subscription=options.input_subscription, with_attributes=True)
    }
```
# udf_main is the main entry-point of every mapper function and needs to exist

def udf_main(message, settings):
    # message is of type PubSubMessage
    # get main body (data) and attributes like this
    data = json.loads(message.data.decode("utf-8"))
    attributes = dict(message.attributes)
    logging.info(f"Transforming {data} with {attributes} ")

    version = settings["version"]
    project = settings["project_id"]
    region = settings["location"]
    dataset = settings["dataset"]
    fhirstore = "core"

    league_user_id = data["league_details"]["league_uid"]
    patient_id = hashlib.sha1(league_user_id.encode("UTF-8")).hexdigest()

    patient_reference_url = f"https://healthcare.googleapis.com/{version}/projects/{project}

    yield get_patient(data, patient_id, league_user_id, patient_reference_url)

    if "services" in data:
        yield from get_services(data, league_user_id, patient_reference_url)

    if "hsa_eligibility" in data:
        yield get_hsa_eligibility(data, league_user_id, patient_reference_url)

    if "active_benefit_plans" in data:
        yield from get_active_benefit_plans(data, league_user_id, patient_reference_url)
FHIR Path

https://build.fhir.org/fhirpath.html
Python

```python
if "services" in data:
    yield from get_services(data, league_user_id, patient_reference_url)

if "hsa_eligibility" in data:
    yield get_hsa_eligibility(data, league_user_id, patient_reference_url)

if "active_benefit_plans" in data:
    yield from get_active_benefit_plans(data, league_user_id, patient_reference_url)
```

FHIR Path

```
"meta":
{
    "id": "template_a",
    ...
    "optional_mapping_query": "services.exists()"
},
```

https://build.fhir.org/fhirpath.html
Multiple Mappings

```
[
  {
    mapping 1
  },
  {
    mapping 2
  },
  {
    mapping 3
  }
]
```

Inheritance

```
"meta":
{
  "id": "template_b",
  "parent": "template_a"
}
```

https://build.fhir.org/fhirpath.html
Architecture
Requirements

- **Set up CI/CD fast**
  - Set up CI/CD for new jobs fast

- **Different Environments**
  - Deploy different version in different environments
    - test / prod

- **Different Projects**
  - Automated deploy in different projects with different configs
Requirements
Template Generation
UDF Deployment

data-dataflow-udfs ~/Documents

├── .github
├── Mapper1
│   ├── main.py
│   └── main_test.py
│       └── test_requirements.txt
├── Mapper2
├── Mapper3
└── Mapper4
UDF Deployment
module "alex_test" {
  source = "./reference-datalake/modules/dataflow/jobs/udf-pubsub-to-chapi"

  project = data.google_project.default.project_id
  project_number = data.google_project.default.number
  metaregion = var.metaregion
  env_type = var.env_type
  udf_name = "alex-test-df-job"

  builder_email = module.dataflow_infra.builder_email
  runner_email = module.dataflow_infra.runner_email

  pubsub_topic = var.fhir_data_topic
  subscription_filter = "attributes.tenant_id = \"${var.tenant_id}\""

  initial_num_workers = 1
  max_num_workers = 1

  chapi_base_url = module.chapi.core_url
  error_bucket_url = module.dataflow_infra.error_bucket_url

  enable_streaming_engine = true
  enable_dataflow_prime = true
  enable_write_raw_to_gcs = false
}
Alerting
Time to Deploy

Days to Deploy

- Old Mapper
- New Mapper
QUESTIONS?