Producer API Toolchain

v0.1.0

Joshua Dotson

Build Date 2020-05-04

Table of Contents

1 Changelog

- **v0.1.0** (josdotso@cisco.com)
  - Initial version.

2 Introduction

The CX Cloud database stored “Device Data” and has tens of database tables. CX Data will expose APIs to read, write, and update these tables.

Manually writing code to implement those APIs would be difficult, time consuming, and error prone. In addition, the database schemas may change slightly through development. It became clear that we needed a means to automatically build the component code instead of writing and supporting that code manually.

We start with human-written database schemas and template files to control the tooling and directly generate gRPC API files and OpenAPI (swagger) files as intermediate outputs. The final outputs are golang client and server code and typescript/javascript code suitable to call the server APIs, as well as command line programs to directly call the server APIs. Using the OpenAPI/swagger files other language bindings can be easily supported as well.

This document describes the tool chain developed to do this code generation.

3 Overview

Three kinds of ingredients are in the Producer API toolchain.

1) Human Inputs - These are files that a person must author.
2) Development Tools - These are tools that take inputs (human or otherwise) and generate some output that we need.
3) Generated Outputs - These are files that the toolchain writes out, based on all Human Inputs, Development Tools (and any configuration parameters – not pictured).

Multiple stages exist in the code-generating pipeline. The diagram below shows three stages of generating code. Only the first stage requires human input files. All remaning stages can be automated with relative ease or hand-configuration.

The only tool in our toolchain that we created (in part) ourself is the db2proto tool, which soon may be renamed as “db2api”.

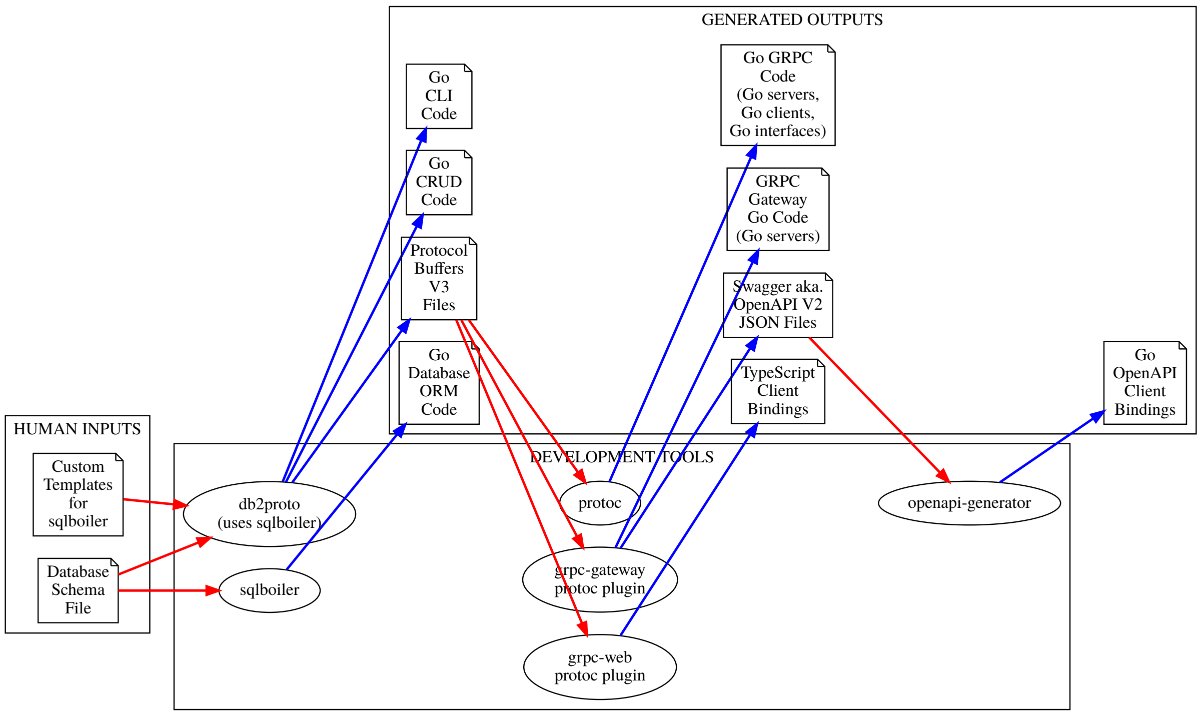
GENERATED OUTPUTS
Go CLI Code
Go CRUD Code
Protocol Buffers V3 Files
Go Database ORM Code
Go GRPC Code (Go servers, Go clients, Go interfaces)
GRPC Gateway Go Code (Go servers)
Swagger aka. OpenAPI V2 JSON Files
TypeScript Client Bindings
Go OpenAPI Client Bindings
HUMAN INPUTS
Custom Templates for sqlboiler
Database Schema File
DEVELOPMENT TOOLS
db2proto (uses sqlboiler)
sqlboiler
protoc
grpc-gateway protoc plugin
grpc-web protoc plugin
openapi-generator

Figure 1: Overview

4 Components

TODO: Fill in the blanks here.

4.1 Human Inputs

4.1.1 Custom Templates for sqlboiler

4.1.2 Database Schema File

4.2 Development Tools

4.2.1 db2proto

(uses sqlboiler)

4.2.2 sqlboiler

4.2.3 protoc

4.2.4 grpc-web protoc plugin

4.2.5 grpc-proxy protoc plugin

4.3 Generated Outputs

4.3.1 Go CLI Code

4.3.2 Go CRUD Code

4.3.3 Go Database ORM Code

4.3.4 Go GRPC Code

(Go servers, Go clients, Go interfaces)

4.3.5 Go OpenAPI Client Bindings

4.3.6 GRPC Gateway Go Code

4.3.7 Protocol Buffers V3 Files

4.3.8 Swagger/OpenAPI V2 JSON Files

4.3.9 TypeScript Client Bindings

5 Appendix A: Discussion Notes 2020-03-09

Attendees: Greg Herlein, Joshua Dotson

5.1 Precedence of semantic fidelity.

- sqlSchema > protoV3Files > openapiV2Files

5.2 Possible Long-term goals

- Get swagger files from meraki.
- Integrate with Meraki APIs.
  - OPTION 1: Map data elements from their swagger.
    - To our database, so that we can use their data.
    - They don't have a schema, but they do have swagger.
    - Most of their fields map somehow to ours.
    - MAY be standard ETL.
  - OPTION 2: Generate Go or Typescript client bindings from Meraki Swagger.
  - OPTION 3: Generate APIs that wrap Meraki but look like our APIs???
- Customer has some Meraki gear and some not, so abstract ALL of their gear to look like one vendor.