Script de Coleta e Tratamento de Dados da JIGA-NHR

Fabio Seiti Fukuda

1 de julho de 2024

Introdução

Este documento detalha o funcionamento do script de coleta de dados da JIGA - NHR. Assim, o intuito deste programa é receber como entrada os arquivos de saída gerados pela JIGA-NHR, processá-los, formatatá-los e gerar aquivos para serem subidos no data lake da AWS. Além disso, informações sobre testes também são subidos num banco de dados MySQL.

Arquitetura e Componentes do Sistema

A Figura 1 apresenta um diagrama estrutural do sistema. Observa-se, através dele, que há dois componentes principais : o PC da JIGA e um Raspberry Pi 4.

O computador da JIGA é responsável por executar o software proprietário emPower, o qual comanda a rotina de testes a serem executados pela JIGA. Assim, este software escreve os resultados dos testes em arquivos com extensão .csv, apesar de não serem formatados de acordo com este padrão.

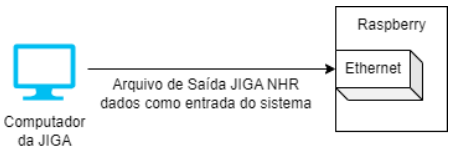

Figura 1: Estrutura do Sistema

Além disso, o computador da JIGA também é responsável por enviar os arquivos contendo os resultados de testes realizados na JIGA para o Raspberry Pi. Sendo assim, no computador da JIGA roda um script responsável por enviar os dados para o Raspberry Pi, o qual salva os arquivos em uma pasta específica.

Por sua vez, o script do Raspberry recebe como entrada os arquivos de saída da JIGA, processa-os e realiza o upload de informações tanto para o data lake AWS, quanto para um banco de dados MySQL, como mostra a Figura 2.

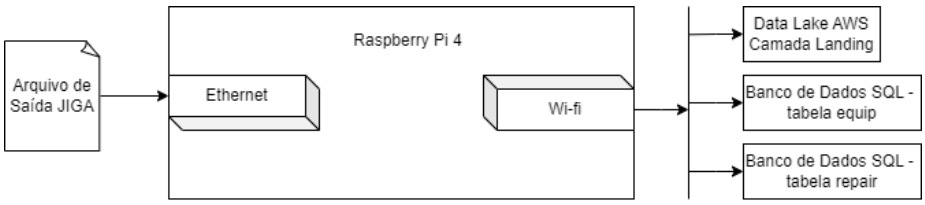

Figura 2: Arquitetura do Programa do Raspberry

Observa-se, com isso, que o Raspberry sobre dados para 3 fontes. A primeira é o data lake AWS, para o qual o script sobe arquivos .csv formatados. O upload é feito para a camada landing. Desa maneira, assim que o arquivo é subido, ele é convertido no data lake automaticamente para um arquivo .parquet, o qual é salvo na camada structured. Esse procedimento é importante para que seja possível realizar consultas SQL sobre os arquivos através do Grafana. As informações que são subidas são os resultados dos testes realizados em cada nobreak. Logo, cada registro contém o número serial do produto, bem como a data/hora do teste.

A segunda fonte para qual o Raspberry sobe informações é para o banco de dados MySQL, mais especificamente para a tabela jiga.equip. São, principalmente, 6 informações que são subidas no banco de dados: data/hora do teste, número serial, rotina de teste aplicada, tensão de saída do nobreak e, por fim, se o nobreak foi aprovado nos testes ou não. O intuito da utilização deste banco de dados é a contrução de uma API que será utilizado pelo ERP da NHS, a qual consulta se um nobreak passou pela JIGA ou não.

Por fim, o Raspberry guarda informações em outra tabela de um banco de dados SQL, a saber, na db_puc.repair. Esse envio de informações é feito para que, quando um nobreak reprove nos testes, o sistema Siri imediatamente faça um registro deste equipamento que apererá na célula de reparo.

Ainda, a JIGA gera arquivos de saída em formato .csv, porém eles não seguem a formatação comum a estes arquivos. Isso porque os resultados dos testes não são organizados em colunas cujos nomes seriam os nomes dos testes. Assim, estes arquivos de saída apresentam algumas peculiaridades.

Primeiramente, os resultados dos testes são organizados em linhas, isto é, cada linha representa uma rotina de teste aplicada em um nobreak. Existem, então, neste registro a data/hora do teste, seguido do número serial do equipamento, e, por fim, os resultados dos testes. Observa-se, assim, que não há o

registro nesses arquivos de saída o nome de cada um dos testes.

Além disso, a JIGA gera um arquivo de saída por dia. Assim, todos os resultados dos testes de uma determinada data são salvos no mesmo arquivo. Isso implica em, por exemplo, registros de equipamentos diferentes (com rotinas de testes diferetens) serem salvos no mesmo arquivo.

Então, para saber qual rotina de testes foi aplicada em determinado equipamento, há nos arquivos de saída o nome da rotina de teste aplicada em uma série de equipamentos.

Bibliotecas em Python

Todo o script de coleta de dados é feita utilizando a linguagem Python. Assim, algumas bibliotecas de apoio são utilizadas na solução, as quais estão listadas abaixo.

Watchdogs

A biblioteca watchdog é utilizada para fazer monitoramento de eventos em diretórios ou arquivos. Assim, o módulo watchdogs.observer cria uma thread e monitora eventos como criação ou modificação de arquivos. Algumas funções deste módulo são:

- **schedule (event_handler, path, recursive=False)** : Configura um monitoramento de um local dado o seu caminho, e chama métodos para gerenciar os eventos que acontecem nele.
- **start()** : Inicializa o monitoramento.

Ainda, o módulo watchdog.events implementa a classe FileSystemEventHandler, a qual é um gerenciador de eventos de modificação de arquivo/pasta. Assim, alguns de seus métodos podem ser sobrescritos, como on_modified e on_created, as quais, respectivamente, são chamadas quando há modificação e criação de arquivos.

Pandas

A biblioteca Pandas é utilizada para análise e manipulação de dados. Assim, é possível utilizá-la para estruturar dados para criar arquivos .csv.

Para tanto, a biblioteca implementa uma estrutura de dados chamada DataFrames, a qual representa uma tabela. É possível, então, ler um arquivo .csv e criar um DataFrame utilizando o método pd.read_csv. Podemos, com isso, fazer algumas manipulações na tabela utilizando métodos como:

- **pandas.concat**: concatena objetos pandas ao longo de algum eixo.
- **DataFrame.reindex**: reorganiza os índices de acordo com alguma lógica.
- **DataFrame.drop_duplicates**: remove linhas duplicadas.
- **DataFrame.to_csv**: Salva o DataFrame em arquivo no formato .csv.

Design da Solução

A Figura 3 apresenta um fluxograma do processamento de um arquivo de saída da JIGA. Observa-se que com base no arquivo dado como entrada são criados outros arquivos, os quais serão subidos no data lake e excluídos da memória do Raspberry. Já os arquivos de entrada permanecem armazenados em memória.

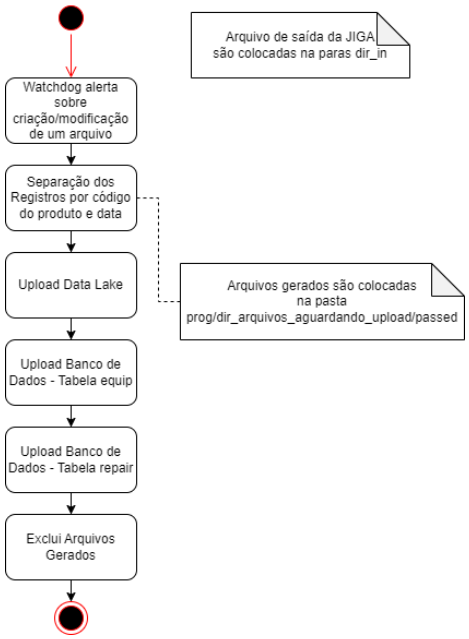

Figura 3: Fluxograma do Processamento de um Arquivo de Saída da JIGA

Já a Figura 4 apresenta o diagrama UML do software implementado.

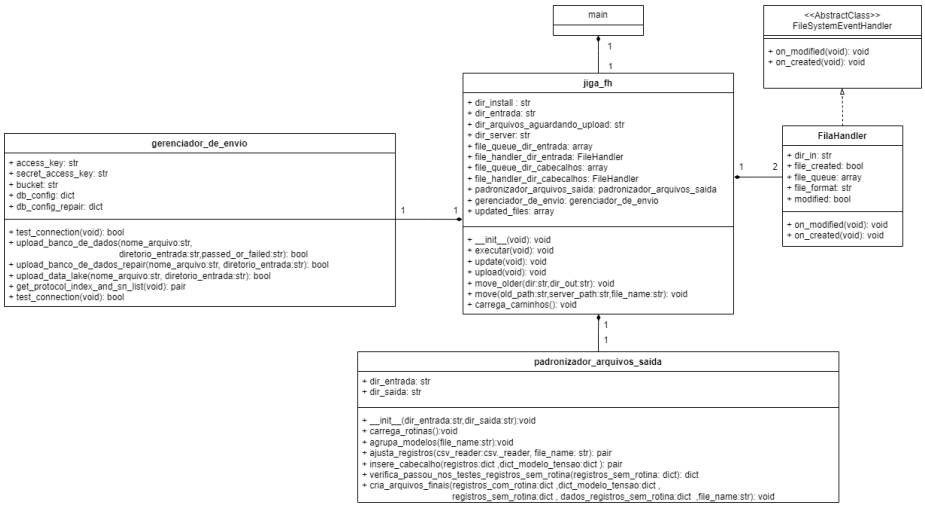

Figura 4: Diagrama de Classes do Software

Assim, traçando uma relação entre o fluxograma e o diagrama de classes, pode-se afirmar que a classe padronizador_arquivos_saida implementa a separação dos registros por código do produto e data. Além disso, os uploads são implementados pela classe gerenciador_de_envio.

Classes do Software

Nesta seção, cada uma das classes implementadas serão detalhadas.

jiga_fh

Função principal do sofware, pois é responsável por fazer chamadas aos métodos das classes responsáveis por processar e subir os arquivos.

Métodos:

- executar(void): executa o loop principal do programa. A Figura 5 apresenta o fluxograma de execução dessa função.

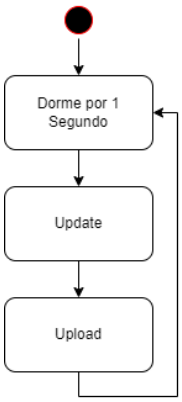

Figura 5: Fluxo da função executar

- update(void): Função que faz chamada ao método agrupa_modelos(file_name:str) da classe padronizador_arquivos_saida.
- upload(void): Função que faz chamadas aos métodos da classe gerenciador_de_envio para upload dos dados processados. Como são 3 locais para os quais são subidos dados, é necessário assegurar que, caso haja algum erro no upload para um dos locais, o algoritmo tente subir este arquivo novamente.

  Assim, são utilizadas 3 flags para sinalizar o sucesso ou não do upload. Se, ao final das 3 tentativas, as 3 flags estiverem setadas, o arquivo com os dados processados é deletado da memória do Raspberry.

- move_older(void): Move os arquivos dados de entrada antigos para a pasta apontada pelo atributo dir_server.

- move_older(void): Move os arquivos dados de entrada antigos para a pasta apontada pelo atributo dir_server.

- move(old_path,server_path,file_name): Move um arquivo de nome file_name de old_path para server_path.

gerenciador_de_envio

Classe responsável por subir os dados tanto no data lake quanto no banco de dados.

Métodos:

- test_connection(void): Realiza um teste de conexão com o banco de dados.

- upload_banco_de_dados(nome_arquivo:str,diretorio_entrada:str,passed_or_failed:str): Função que sobe informações do arquivo *nome_arquivo* do diretório *diretorio_entrada*. Se o valor de *passed_or_failed* for igual a 'passed', são subidas informações na tabela *equip*, o que significa que o arquivo de saída da JIGA foi processado corretamente. Porém caso *passed_or_failed* for igual a 'failed', isso significa que o arquivo de saída da JIGA não foi processado corretamente, pois não foi encontrado a rotina de testes na memória do Raspberry, ou a rotina não está condizente com os testes.

- upload_data_lake(void): Função que faz upload dos arquivos no data lake da AWS.

- upload_banco_de_dados_repair(nome_arquivo:str,diretorio_entrada:str): Realiza upload de informações para o sistema SIRI sobre nobreaks que reprovaram nos testes.

padronizador_arquivos_saida

Esta classe é responsável por formatar as informações do arquivo de saída da JIGA em um formato .csv para upload no data lake. Além disso, as informações subidas no banco de dados são retirados desses arquivos .csv de saída gerados nessa classe.

- carrega_rotinas(void): Carrega as rotinas de testes de cada equipamento, as quais estão registradas em um arquivo .xlsx.

- agrupa_modelos(file_name:str): Função principal da classe, que faz chamadas aos demais métodos para o processamento do arquivo de saída da JIGA.

- ajusta_registros(csv_reader:csvreader,file_name:str): separa os registros de acordo com a rotina de teste aplicada. Assim, os registros são armazenados em um dicionário, no qual a chave é o nome da rotina de teste, e o valor é uma lista contendo os resultados dos testes.

- insere_cabecalho(registros,dict_modelo_tensao): função que compara os registros com o cabeçalho. Com base nisso, classifica se o registro se refere a um equipamento que foi aprovado nos testes, ou não.

- verifica_passou_nos_testes_registros_sem_rotina(registros_sem_rotina:dict): Função responsável por tratar registros cujas rotinas de testes não foram encontradas na memória do Raspberry. Assim, existem alguns números seriais para os quais é sabido o tamanho do registro esperado, pois essa informação é armazenada em um outro arquivo .xlsx . Com isso, é possível verificar se

o nobreak passou nos testes, pois se o tamanho do registro for menor, isso significa que o equipamento foi reprovado.

- cria_arquivos_finais(registros_com_rotina:dict,dict_modelo_tensao:dict, registros_sem_rotina:dict,dados_registros_sem_rotina:dict,file_name:str): responsável por gerar os arquivos finais referentes aos registros de entrada.

Considerações

Nome dos arquivos de saída .csv

Para o upload dos arquivos .csv para o datalake, deve-se seguir uma padronização de nomes. Assim, o arquivo deve começar com 'td' (*testing data*), seguido de ou 'jhnr120' ou 'jhnr220', isto é, caso os registros deste arquivo se refiram a nobreaks de saída 120, utiliza-se jhnr120, caso contrário, jhnr220. Após isso, vem o código de produto, seguido do ano, mês e dia. Todas as informações são separadas por hífen.

Assim, um exemplo de nome de arquivo é td-jnhr120-90A1006100-2024-01-01.csv. Esse registro se refere ao equipamento de código 90A1006100, com saída 120, cujo teste foi realizado em 01/01/2024. Aliás, esta data é obtida pelo nome do arquivo de saída da JIGA.

Nome das Colunas dos Arquivos .csv

Além disso, outra restrição para os arquivos .csv a serem subidos no datalake são as colunas. Assim, existem duas padronizações para o conjunto de colunas: um para equipamentos de saída 120, e outra para 220.

Tomando como exemplo os equipamentos de saída 120, todos os arquivos .csv possuem colunas referentes a todos os testes realizados em equipamentos 120, mesmo que o teste não seja aplicável ao equipamento em um nobreak em específico. Por exemplo, supondo que esteja sendo escrito os resultados de teste de um nobreak mono. Mesmo que não seja aplicável o teste "sobre-tensão 220", teu arquivo de saída .csv terá uma coluna referente a este teste, evidentemente com valor nulo.

Além disso, para cada teste foi criado um nome apropriado (isto é, sem caracteres especiais, espaços...) para ser subido no datalake. A Tabela 1 apresenta este mapeamento.

Nome do teste	Nomenclatura
Teste ALIMENTACAO	sup_120v_v
Sub Tensao REDE 120V	sub_v_grd_120v_v
Sub Tensao INVERSOR 120V	sub_v_inv_120v_v
Retorno Rede 120V	grd_ret_120v_v
Retorno Rede	grd_ret_v
Sobre Tensao REDE 120V	srg_grd_120v_v
Sobre Tensao INVERSOR 120V	srg_inv_120v_v
Alimentacao 220V	sup_220v_v
Sub Tensao REDE 220V	sub_v_grd_220v_v
Sub Tensao INVERSOR 220V	sub_v_inv_220v_v
Retorno Rede 220V	grd_ret_220v_v
Sobre Tensao REDE 220V	srg_grd_220v_v
Sobre Tensao INVERSOR 220V	srg_inv_220v_v
Potencia excessiva Rede 110% carga (W)	ext_pow_grd_110perc_v
Alerta (Potencia excessiva)	ext_pow_wng
COMUTACAO Rede/Inversor - 130%	com_grd_inv_130perc
COMUTACAO Rede/Inversor – 130% Carga – 120Vin	com_grd_inv_130perc_vin_120
COMUTACAO Rede/Inversor – 130% Carga – 220Vin	com_grd_inv_130perc_vin_220
RETIRA CARGA	dsc
Tensao de saida INVERSOR 120V - 100%	vo_inv_120v_100perc_v
Tensao de saida INVERSOR 120V	vo_inv_120v_v
Consumo em Vazio Rede 120V WATTS	emp_csm_grd_120v_w
Consumo em Vazio Rede 120V VA	emp_csm_grd_120v_va
Consumo em Vazio Rede 220V WATTS	emp_csm_grd_220v_w
Consumo em Vazio Rede 220V VA	emp_csm_grd_220v_va
Corrente de Pico – Curto Circuito - 120Vout	pc_sc_vo_120v_a
Corrente de Pico – Curto Circuito – 220Vout	pc_sc_vo_220v_v
Tensao de saida INVERSOR 220V - 100%	vo_inv_220v_100perc_v
Tensao de saida INVERSOR 220V	vo_inv_220v_v
Eficiencia 120Vin – 220Vout (%)	efc_vin_120v_vo_220v_perc
Eficiencia 120Vin – 220Vout Input (W)	efc_vin_120v_vo_220v_input_w
Eficiencia 120Vin – 220Vout Output (W)	efc_vin_120v_vo_220v_output_w
Eficiencia 220Vin – 120Vout (%)	efc_vin_220v_vo_120v_perc
Eficiencia 220Vin – 120Vout Input (W)	efc_vin_220v_vo_120v_input_w
Eficiencia 220Vin – 120Vout Output (W)	efc_vin_220v_vo_120v_output_w

Tabela 1: Nomenclatura dos testes

Assim, as colunas dos arquivos .csv estão aprensetadas na Tabela 2.

120	220
ts	ts
sn	sn
lt	lt
id_operator	id_operator
cmt	cmt
sup_120v_v	sup_120v_v
sub_v_grd_120v_v	sub_v_grd_120v_v
sub_v_inv_120v_v	sub_v_inv_120v_v
grd_ret_120v_v	grd_ret_120v_v
srg_grd_120v_v	srg_grd_120v_v
srg_inv_120v_v	srg_inv_120v_v
sup_220v_v	sup_220v_v
sub_v_grd_220v_v	sub_v_grd_220v_v
sub_v_inv_220v_v	sub_v_inv_220v_v
grd_ret_220v_v	grd_ret_220v_v
srg_grd_220v_v	srg_grd_220v_v
srg_inv_220v_v	srg_inv_220v_v
ext_pow_grd_110perc_v	ext_pow_grd_110perc_v
ext_pow_wng	ext_pow_wng
vo_inv_120v_100perc_v	vo_inv_220v_100perc_v
com_grd_inv_130perc	com_grd_inv_130perc_vin_120
dsc	dsc
vo_inv_120v_v	vo_inv_220v_v
emp_csm_grd_220v_w	emp_csm_grd_120v_w
emp_csm_grd_220v_va	emp_csm_grd_120v_va
efc_vin_220v_vo_120v_perc	efc_vin_120v_vo_220v_perc
efc_vin_220v_vo_120v_input_w	efc_vin_120v_vo_220v_input_w
efc_vin_220v_vo_120v_output_w	efc_vin_120v_vo_220v_output_w
test_ok	test_ok
pc_sc_vo_120v_a	com_grd_inv_130perc_vin_220
pc_sc_vo_220v_v	emp_csm_grd_220v_w
com_grd_inv_130perc_vin_120	emp_csm_grd_220v_va
com_grd_inv_130perc_vin_220	efc_vin_220v_vo_120v_perc
emp_csm_grd_120v_w	efc_vin_220v_vo_120v_input_w
emp_csm_grd_120v_va	efc_vin_220v_vo_120v_output_w
grd_ret_v	vo_inv_120v_100perc_v
efc_vin_120v_vo_220v_perc	vo_inv_120v_v
efc_vin_120v_vo_220v_input_w	pc_sc_vo_120v_a
efc_vin_120v_vo_220v_output_w	pc_sc_vo_220v_v
vo_inv_220v_100perc_v	test_failed
test_failed	test_routine
test_routine	output_voltage
output_voltage	

Tabela 2: Colunas dos arquivos .csv de saída

Tabelas Banco de Dados

São utilizadas 3 tabelas para o upload de informações, as quais são descritas abaixo.

Tabela equip

Utilizada para o upload de informações de testes de cada nobreak. Assim, são 3 informações principais subidas: data do teste (coluna ts), código serial do teste (coluna sn) e se o nobreak passou ou reprovou nos testes (coluna test_ok).

Tabela repair

Tabela utilizada pelo sistema Siri. Assim, toda vez que um equipamento reprova nos testes da JIGA, é criado um registro nessa tabela com as seguintes colunas.

- protocol (string): Código para identificar de onde veio o equipamento, no caso, da JIGA. Assim, este campo inicia com "JIG"seguido de um número inteiro da vez, isto é, pega-se o último inteiro do ultimo protocolo do tipo "JIG", e soma-se 1.
- sn (string): Número serial do nobreak.
- ts (timestamp): Horário do upload do dado (ou seja, não é o horário que foi realizado o teste).
- user_id (inteiro): Utilizado para identificar que fez o upload do dado. No caso da JIGA, utiliza-se o identificador 10971.
- repair_state_id (inteiro): Código para identificar o estado do reparo. Assim, o registro deve, inicialmente, iniciar no estado 1.
- repair_type_id (inteiro): Código para identificar o tipo de reparo. Assim, o registro deve ser cadastrado como tipo 1.