**Knitro License API flow Diagram:**

opf python code generates 2 files the

model\_feeder.nl file and model\_feeder.nl.symbol\_map.pickle file

through write\_nl() function

****

drsdevdb

server

Python code compresses the .nl file to model\_feeder.nl.gz file to reduce network bandwidth

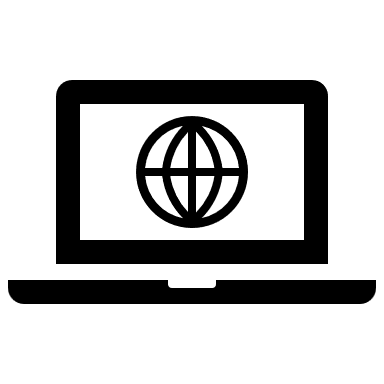
Note: From multiple opf python code running on multiple machines can do this api call simultaneously.

Post method takes one .nl.gz file and a payload

{ “algorithm”: 0, “presolve”: 1, “wantsol”: 1,

“par\_numthreads”: 1 }

Note: this can be extended by changing the web api c# code and rebuilding the api

<http://solvertestapp:3000/api/knitro/solve>

Hosted on a dotenet Kestrel standalone Exe

Can be added as a windows service to make hosting api endpoint permanent.

The webapi first unzips the received .nl.gz file, Stores it in a temporary folder, runs the command

“knitroampl.exe” model.nl algorithm=1 wantsol=1 [and other options as provided in the payload]

It generates the model.sol after knitro solves the model, it can be feasible or infesible.

And the api returns the sol file content as the response

Then we generate all charts and the doe csv exports and the hourly opendss export csv files

Instance.solution.load\_from(problem)

Using this function we updates the instance with the results received from the knitro solution file

The Read\_sol() function takes the output.sol file and model\_feeder.nl.symbol\_map.pickle file and returns the Problem, and we check if it returns Optimal

the response is received back to the opf python code and if the status code is 200 OK

It stores the sol file locally in a model.sol file

{ “message”: solver processing successful, “result”: command output of knitro,

“contenttype”: , “filename”: , “filesize”: , “data”: sol file content in byte format }