NASA/TM-20205008494

Power Autonomy Research and Development Environment (PARDE) User's Guide
Version 0.1.2

George Thomas, Kevin Rák, Matthew Muscatello, Marc Carbone, and Jeffrey Csank
Glenn Research Center, Cleveland, Ohio

May 2021

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers, but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., "quick-release" reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 757-864-6500

- Telephone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Program
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM-20205008494

Power Autonomy Research and Development Environment (PARDE) User's Guide
Version 0.1.2

George Thomas, Kevin Rák, Matthew Muscatello, Marc Carbone, and Jeffrey Csank
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

May 2021

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at http://www.sti.nasa.gov/ and http://ntrs.nasa.gov/

Contents

Power Autonomy Research and Development Environment (PARDE) User's Guide
Version 0.1.2

George Thomas, Kevin Rák, Matthew Muscatello, Marc Carbone, and Jeffrey Csank
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

1.0 Introduction

The Power Autonomy Research and Development Environment (PARDE) software package is a version of NASA's Autonomous Power Control (APC) software (Ref. 1) that can be used to evaluate fault management and automatic power system reconfiguration algorithms in a relevant system without having to fully develop all the supporting software. Software items included are a set of C++ class source files representing simplified fault management and reconfiguration logic, a power system simulation representing a notional architecture for NASA's Gateway vehicle, a web-based graphical user interface for running and testing the simulation and APC, a Docker-based automatic setup script for a development environment, and a user's guide.

The software platform for the PARDE package is a Docker-Compose network (environment) that is automatically set up when an end user runs the appropriate "start" script. The network is intended to be interacted with via Visual Studio Code (VS Code), various scripts included in the PARDE zip file, standard Docker tools, or SSH (secure shell). The remainder of this section details the items within PARDE and how they work, in order to provide background knowledge that will help users design and test autonomy algorithms with PARDE.

1.1 Docker Compose Environment

Docker presents the ability to run programs in a repeatable, cross platform manner, using containerized environments (containers) built using images (or baseline builds of an operating system with a minimum set of necessary packages and dependencies) that can be shared via a site called Docker Hub. Docker was chosen as a way to deliver the APC software in PARDE, in order to make the process as repeatable and self-contained as possible.

The Docker-Compose network used for PARDE consists of a main operating system (OS) container that the user interacts with (called *parde_$USER*), a container that runs a RabbitMQ server in the background (*parde_$USER_rabbitmq*), and a container that runs a MySQL server in the background (*parde_$USER_rabbitmq*). The variable $USER represents the username of your user account on your host OS (the computer account you normally log into). The main OS container is based on a baseline image of the CentOS Linux distribution, with APC associated dependencies, as run in a Docker container. The background server containers mentioned above for MySQL and RabbitMQ typically do not require any action from the user, and so once the environment is set up, the user only interacts with the main *parde_$USER* container.

Because Docker can run in a variety of platforms (e.g., Windows, MacOS, Linux), PARDE should be able to run on whatever hardware is desired. Note that PARDE has only been tested on Ubuntu 18.04.4 LTS and Windows 10 build 2004, using the Windows Subsystem for Linux (WSL) version 2 with Ubuntu 20.04 as the backend. It is possible that Mac users may be able to adapt the Linux instructions with

minimal modifications, but this has not been attempted by the developers and so cannot be guaranteed to work. Note that the Docker environment takes fairly significant resources. It is recommended to run it on a capable computer (at least 4 physical CPU cores and 16 GB of RAM).

The Docker environment can be set up by running either ***start*** if running on a Linux host computer, or ***startPARDE.bat*** if on a Windows host computer. Both of these scripts perform similar setup. They load in various environment variables needed to run the Docker environment (***setenv.sh/.bat***), and then start the environment by calling *docker-compose up* on the ***docker-compose.yml***. This Docker-Compose YAML file contains the setup information for the Docker-Compose network. A breakdown of the setup defined in this file is listed in Table 1.

TABLE 1.—SETUP ACTIONS CODED IN THE DOCKER-COMPOSE FILE USED IN THE start/startPARDE.bat SCRIPTS

1. Main OS container setup (*parde_$USER*)
    a. This container is created from an image built with a Dockerfile in the *dev/centos_8* folder. If this image has already been built, this step is skipped and the image is reused
        i. The image build process starts from a CentOS:8 image from Docker Hub
        ii. Then several steps are taken to add necessary dependencies obtained via Yum
        iii. Several other standard dependencies are then obtained as source via GitHub/GitLab. This step can take a while (within an hour or two)
            1. Includes CMake, Ninja build tools, etc.
        iv. Then a software library dependency CMake project is added into the image and built (this is in the zip in *dev/centos_8/deps/src*). Note, this step takes a long time (likely several hours). The dependency libraries include:
            1. SimpleAmqpClient
            2. boost
            3. civetweb
            4. eigen
            5. lapacke
            6. lely-core
            7. rabbitmq-c
            8. soci
    b. Once the image is built, a container is created from this image and several commands are run in the container to prepare it for use, these do the following:
        i. Create user account in *parde_$USER* with username *$USER* and password *dev*
        ii. Set up APC code project in ~/apc/apc (~ is home folder for above user account)
        iii. Symbolically link the library dependencies from the image build step into ~/apc/deps
        iv. Correctly set up ownership of files in user's home folder
2. RabbitMQ server container setup (*parde_$USER_rabbitmq*)
    a. This container is created directly from a Docker Hub image
    b. Configuration and other setup parameter files are included via volumes
3. MySQL server container setup (*parde_$USER_mysql*)
    a. This container is also created directly from a Docker Hub image
    b. Environment variables for this container are set in ***docker-compose.yml***
    c. An initial database is included into this container via a volume
4. Several volumes
    a. The *vscode* volume which is intended to hold VS Code settings for the main OS container, so that they persist between container creation/deletion cycles
    b. The *parde_src* volume which is intended to hold the APC source code project (~/apc/apc), as used in the main OS container, so that this also persists between container cycles

If at any time, the user needs to stop the environment, the ***stop*** or ***stopPARDE.bat*** scripts can be used. Note, these will also destroy the associated containers, ending any currently running sessions in those containers. If the PARDE Docker-Compose network is ever stopped with the stop script, restarting it should be very quick, as the time consuming parts of the setup take place during the main OS image build step (and not the container creation step). Running the ***attach*** and ***attachPARDE.bat*** scripts will start an interactive shell, logged in as your user in the main OS container in your current terminal. This shell can be used to run the APC. The entry point executable to the APC is at ~/apc/apc/bin/web_portal. This web gui and instructions on how to run it and interact with the APC are described in following subsections.

As mentioned in Table 1, item 1.b.ii, the APC code project that you will work on is automatically set up for you when running the Docker-Compose setup script (***start*** or ***startPARDE.bat***). Further, note from item 4.b, that this location in the set up as a Docker volume. By putting the APC source code project into a volume instead of just in files in a container, these files will persist from one startup/shutdown cycle of the PARDE setup to another. This is because volumes persist between cycles, whereas all of the PARDE containers are destroyed when the ***stop*** or ***stopPARDE.bat*** scripts are called.

The APC code project files that are copied over to ~/apc/apc are copied from the apc/apc.zip folder located in the main PARDE zip package. Note that end users do not need to unzip these files, or do anything else with them. The intent is that the apc/apc.zip file is included in PARDE just so that these files can be automatically unzipped and copied into the Docker end user's code project folder by the Docker-Compose setup script during the Docker container creation step. Lastly, note that the Docker-Compose file does this unzip/copy step in a bash script that executes in the *parde_$USER* container, after that container is created. This bash script will only copy these files if they do not exist in the ~/apc/apc volume. This means that normally, these files are only created in that volume the first time the user runs the ***start*** or ***startPARDE.bat*** scripts. However, it also means that if the user wishes to return their APC code project to its initial state, they can delete or prune the ~/apc/apc volume. This can be done using the following calls to the PARDE and Docker API at a Powershell terminal:

```
# stop only needed if the PARDE containers exist (they won't if you have previously run stop)
.\stopPARDE

# this prunes all volumes not currently in use by a container, type 'y' for yes
docker volume prune

# start containers back up, which re-creates the volumes with brand new files
.\startPARDE
```

Or at a bash terminal

```
# stop only needed if the PARDE containers exist (they won't if you have previously run stop)
./stop

# this prunes all volumes not currently in use by a container, type 'y' for yes
docker volume prune

# start containers back up, which re-creates the volumes with brand new files
./start
```

1.2 Autonomous Power Control

NASA's APC software is a prototype demonstration of autonomous control logic for a spacecraft power system and is intended to control the power system simulation described in the following subsection. The APC itself is composed of a web GUI with a variety of functions (diagnostics, power system diagram, fault setter, vehicle manager emulator, etc.). This GUI provides a human interface to the APC logic, which consists of a collection of services, including the fault manager service (***ServiceFaultManager.cpp/.h/.so***) and the reconfiguration service (also known as maintenance, mitigation, and recovery service, or MMR service, which exists in ***ServiceMmr.cpp/.h/.so***). The services are started and coordinated via a service manager executable (***service_manager_gateway_arch***). Note that there are a variety of other services that are not associated with fault management (e.g., communication, management of telemetry data, etc.). For this reason, these are built into the binaries packaged with PARDE and not exposed to the user. Lastly, also included in the APC package is a service manager and collection of services associated with emulation of power system components (known as orbital replacement units or ORUs) and their ability to send telemetry and fault data to the APC. This functionality is similarly distributed in binary executable form only (as ***service_manager_raspberry_pi***), as it is not associated with APC or supervisory level fault management.

The reconfiguration and fault manager services exist as C++ source files and are highly simplified versions of the ones that NASA has produced during the course of its APC research under the Advanced Exploration Systems (AES) Program, Advanced Modular Power Systems (AMPS) Project. These source files exist primarily as templates that end users can rewrite, adding fault management logic they wish to experiment with. More information about these services and their functions are listed in Section 2.0.

The APC is intended to operate as a finite state machine, where any faults that occur in the system are correctly identified (emergency state), mitigated as quickly as possible (restorative state), and the fault is announced to parties that may be able to repair it to bring the vehicle back into full operation (normal state). A diagram of this state machine is shown in Figure 1.

More information about how NASA's baseline APC functions is detailed in Reference 1.

1.3 Simulation (DSG_Main)

A simulation of a notional Gateway spacecraft power system is provided in the form of the executable ***DSG_main*** in PARDE. This version of Gateway is composed of two modules or elements (with a third, unused Lander module sometimes shown). The main functional modules are the Power and Propulsion Element (PPE) and the Habitation and Logistics Outpost (HALO). Screenshots of the web GUI power system diagram, zoomed into the PPE and HALO are shown in Figure 2 and Figure 3 respectively to show the architectures of these modules.

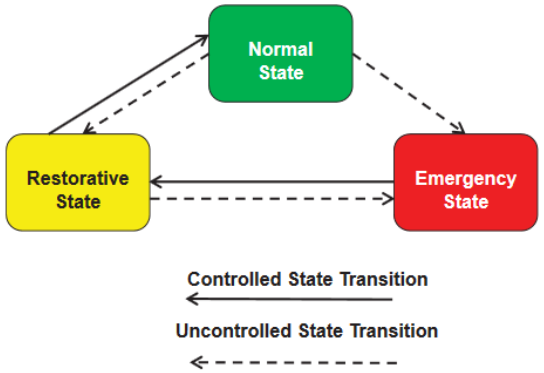

Figure 1.—APC state transition diagram.

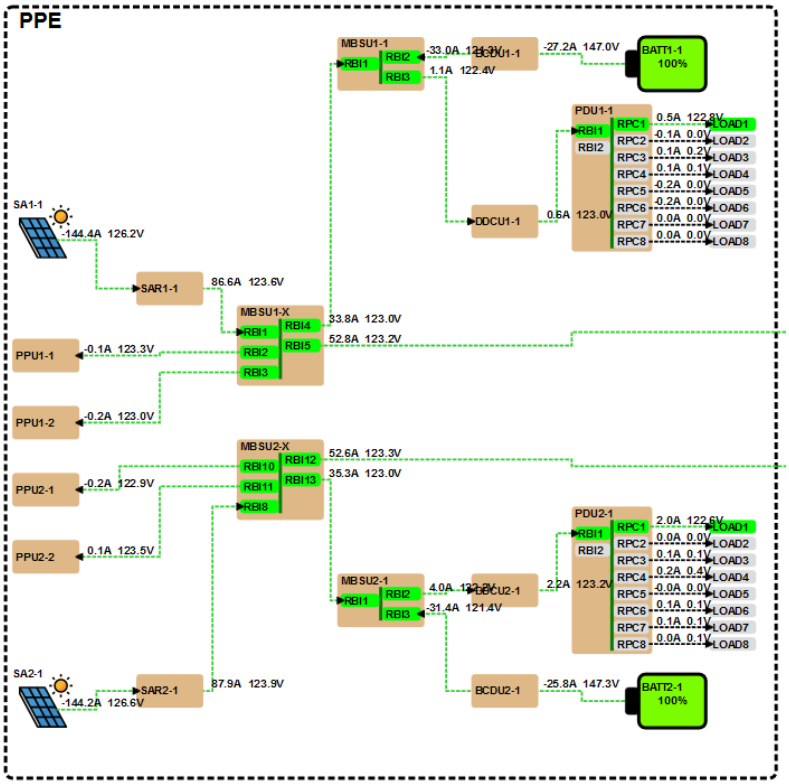

Figure 2.—Power and Propulsion Element (PPE) in the APC Gateway Power System Diagram.

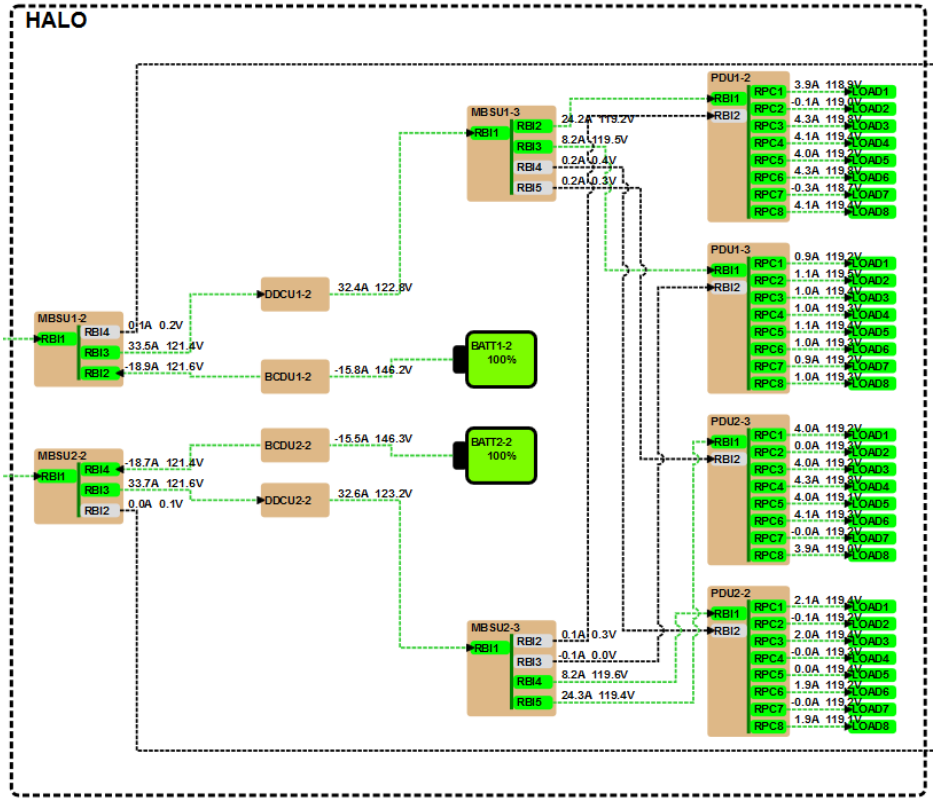

Figure 3.—APC Gateway diagram of the Habitation and Logistics Outpost (HALO).

As seen in these figures, the notional Gateway power system contains sources, including solar arrays (SAs), as well as batteries. These sources are unregulated. In order to regulate that power and interface them to the 120 V DC main power busses, power electronic devices, being solar array regulators (SARs) and battery charge/discharge units (BCDUs), are included. The sources are connected to the main busses via main bus switching units (MBSUs). MBSUs themselves feature switchable bus connections called remote bus isolators (RBIs).

The MBSUs also connect the main busses to downstream busses via DC-DC converter units (DDCUs). Finally, power is delivered to loads via power delivery units (PDUs) that each contain 8 remote power controllers (RPCs) which are switchable load connections with overcurrent trip behavior.

Note that both the HALO and PPE are partitioned into two isolated power domains. That is, the top half of their respective diagrams and bottom half represent two isolated power systems. The one exception to this is that the HALO has crosstie conductors that allow the bottom domain MBSU to power the top domain's PDUs (and vice versa). These crossties are intended to be used in the case of faults or other exigencies to keep all HALO PDUs powered even if an RBI is disconnected.

1.4 RabbitMQ

The APC uses RabbitMQ in order to send data between its constituent services. RabbitMQ is a lightweight and fast messaging system, which uses a server to host queues where clients can publish messages and subscribe to receive them. PARDE sets up a basic RabbitMQ server from a baseline RabbitMQ Docker Hub image using its Docker Compose setup. PARDE automatically sets up various server settings including a RabbitMQ user called *apc* (with a password of *apc*), as well as a virtual host called */SIM*, which is used to locate queues associated with the power system simulation ***DSG_Main***. RabbitMQ can be treated more as a background communication device to end users of PARDE. However, if something goes wrong, users can go to the RabbitMQ control panel in their web browser at localhost:15672, log in as the *apc* user with the password *apc*, and diagnose issues.

1.5 MySQL

The APC also uses MySQL to handle some of the data that the APC exchanges with the included vehicle manager emulator. The MySQL server, and operation of it occur in the background and do not typically need any action from the user.

2.0 Services

This section describes the fault management and reconfiguration service templates that come with PARDE, as well as some suggestions for code that end users can write in their place. In order to first provide context as to the roles that these services play, a bullet list describing the flow of information in the case of a short circuit fault experiment on the PARDE APC is given below

1. A short circuit fault is injected into the system at PDU1-2, RBI1, using the fault setter GUI.
2. Logic behind the GUI sends a signal to the ***DSG_main*** simulation and the RBI is faulted.
3. The RBI will draw a short circuit current and will trip.
4. The short circuit fault is detected at the ORU level in ***service_manager_raspberry_pi*** by polling the trip state of the RBI and seeing a value of 1, this fault is published to the unvetted fault queue
5. A fault detected at the ORU level triggers emergency state.
6. The central layer fault manager logic in *ServiceFaultManager* reads this message from the unvetted fault queue, and publishes it to the vetted fault queue
7. The reconfiguration logic in *ServiceMmr* will read the message from the vetted fault queue, and will determine that the fault is a feeder fault to a PDU. It will then invoke a canned reconfiguration:
   a. Disable MBSU1-3 RBI2 that feeds the faulted RBI in PDU1-2
   b. Activate PDU1-2 (secondary/backup) RBI2
   c. Also activate RBI2 on MBSU2-3 that feeds the secondary RBI on PDU1-2
   d. This corrective action, which restores power to PDU1-2 puts the system in restorative state

2.1 Fault Management

ServiceFaultManager is intended to be logic that parses sensor data and fault detection messages sent by the ORUs, and uses knowledge about the system to determine if and when a fault occurs, and what is the true underlying cause or causes. This fault detection logic is intended to be used by other functions of the APC (especially reconfiguration) in order to mitigate the fault and bring the system to restorative state.

The template version of *ServiceFaultManager* included in PARDE only detects short circuit faults (as described in the bullet list in Section 2.0). Logic in ***ServiceFaultManager.cpp*** reads messages from this queue to look for these faults, and publishes them to the vetted fault queue immediately. Because the trip sequence in the Gateway power system is designed to trip only the one breaker (RBI/RPC) closest to the short, detecting short circuits is relatively easy. This is because the trip state of the breaker that trips identifies where the short is located.

Other faults are more complicated to detect, and some faults may appear as multiple sensor anomalies. NASA's implementation of *ServiceFaultManager* uses a Kalman filter and other logic to detect a wider array of faults in the system (including soft faults and sensor faults) and even multiple concurrent faults. It does this by feeding sensor data and fault states collected from the ORUs into a load flow model of the power system and into the Kalman filter, to determine what faults truly exist in the system and where they are. End users are encouraged to design fault detection and vetting logic that is based on alternative approaches in order to develop and demonstrate new technologies.

2.2 Reconfiguration

The reconfiguration logic in the APC exists in the mitigation, maintenance, and recovery service (*ServiceMmr*). This logic is designed to determine the best configuration of the power system, given the

state of the vehicle. For mitigation and recovery situations, this means reconfiguring the switchgear in the system to remove faults or otherwise put the power system in its most capable state.

The PARDE APC is capable of reconfiguring to isolate HALO feeder faults and battery faults. HALO feeder faults are defined as faults in the RBI's in MBSU1-3 and MBSU2-3 that feed downstream PDUs. These faults are able to be mitigated by switching off the faulted RBI, and powering the PDU that was normally powered by the faulted RBI by a crosstied RBI on another MBSU.

Battery faults are faults at MBSU1-2, RBI2 or MBSU2-2, RBI4, representing a fault of the line between the battery and MBSU or a fault in the battery itself. In this case, the lost battery represents a loss in energy availability in either the top power domain or bottom domain. To handle this, *ServiceMmr* will reconfigure the system so that it powers all HALO PDUs from the undamaged power domain. This reduced energy availability may mean that some loads need to be shed in order to avoid over-discharging the remaining battery or exceeding current limits. Note that the load shedding service will automatically take care of this in such a case.

Both of these fault responses are canned, in that only certain faults are handled, and the responses are predetermined and hardcoded. Others faults are ignored. NASA's solution for reconfiguration involves encoding the system as a graph, and using Dijkstra's algorithm to find the shortest path. End user research teams are encouraged to build logic into their versions of *ServiceMmr* such that they can restore the power system to its most capable state no matter what fault or faults are encountered.

3.0 Quick Start

In order to run PARDE, first obtain Visual Studio Code on the machine you wish to do development on, and obtain Docker on a computer that you wish to host the PARDE Docker environment. Docker can be obtained at https://docs.docker.com/get-docker/, and Visual Studio Code can be obtained at https://code.visualstudio.com/. If desired, both the VS Code IDE and Docker environment can be run on one computer, however, they are designed specifically so that they can run on separate machines. This is accomplished by taking advantage of VS Code's Remote-SSH extension. If the IDE is run on a developer's personal machine and connected to a powerful machine that runs the Docker environment, this will result in a robust development environment that combines the agility of a local IDE and the power of a remote build computer/server. The next subsection gives instructions on how to set up the Docker environment on an Ubuntu Linux machine, and the VS Code on a separate computer. Instructions in another following subsection will describe how to run the setup all on the same machine on Windows (this will also describe some minor differences in terms of how the Docker environment setup works on Windows).

3.1 Setup With Two Computers

Before we begin, Docker must be installed on the associated host machine. Assuming this machine runs Ubuntu, the installation directions can be found here: https://docs.docker.com/engine/install/ubuntu/. Once Docker is obtained, the next step is to establish a user account for the developer on both the development machine and the Ubuntu-based Docker environment host machine. For instance, if the developer is Ryan Smith, rsmith can be used as the user account name on both of these machines. Once the user accounts are set up, the Docker machine administrator must ensure the associated user account on that machine is a member of the docker group. It is recommended *not* to use shared user accounts (for instance, shared by all members of a lab team) for this setup, as this will likely cause problems. For this example, let's say the Ubuntu Docker host computer is called McCarthy.

The lines of bash script below go through the steps the Docker host computer admin will need to take to setup the account and group membership.

```
sudo adduser rsmith
# This will prompt you to pick a password.
# This will also prompt you to enter several pieces of information about the rsmith user.
# The only information you should enter is the `Full Name` as `Ryan Smith`.
# You can leave the other fields such as `Room Number` blank.

sudo usermod -aG docker rsmith
# This adds our new `rsmith` user to the `docker` group which allows us to run Docker containers.
```

Once this is completed, the PARDE zip file must be unzipped to a suitable installation directory. A good location is in the user's home folder (e.g., copy to /home/rsmith/parde). If PARDE is to be installed in the user's home folder, it would be best done while logged in as that user. Assuming this is completed, the following steps can be used to start the Docker environment.

```
# Finally, let's bring our Docker environment online:
cd ~/parde
./start

# Note when you run the `start` script that it tells you a list of port numbers.
# Those numbers are specific to your user account and you'll need to remember them for later.
# See the note at the bottom center of the below image which explains further.
```

Now that the Docker environment is running, we need to establish some SSH tunnels. In the APC lab at NASA GRC, firewall blocks all but port 22 (for SSH) into our lab, so all PARDE traffic must tunnel through this port. This tunneling setup may also be useful for end users to set up the Docker host computer in a firewalled lab at a research institution or university. Figure 4 is an overview of the tunnels that need to be established.

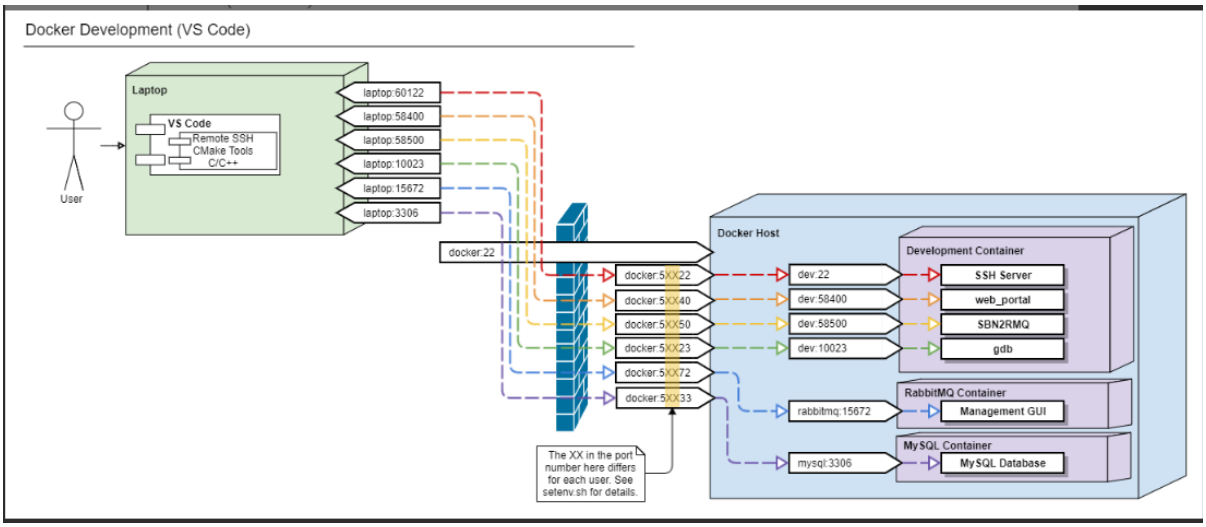

Figure 4.—Overview of the tunnels that need to be established.

Trying to follow all of the port numbers can be a bit confusing, so this diagram attempts to keep the port numbers on the user's development machine matching or analogous to the ports running on the Docker containers. However, the ports on the Docker host itself must be unique to your user since the lab machines might have multiple users working on them, running PARDE environments of their own at the same time. Thankfully, establishing the tunnels is a fairly simple task. You will need to recall the port numbers that were printed out when you ran the start script on the Docker host. For this example, let's assume that the highlighted XX numbers in the above image are 05.

Open Notepad, and open **%USERPROFILE%\.ssh\config** (if the file or folder doesn't already exist then create them). In that file, add the following lines:

```
Host docker_tunnels
  HostName localhost
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
  Port 60122
  User rsmith

Host mccarthy
  HostName aaa.bbb.ccc.53
  User rsmith

Host mccarthy_tunnels
  HostName aaa.bbb.ccc.53
  User rsmith
  LocalForward 15672 localhost:51572
  LocalForward  3306 localhost:51506
  LocalForward 58400 localhost:51540
  LocalForward 58500 localhost:51550
  LocalForward 10023 localhost:51523
  LocalForward 60122 localhost:51522
```

IMPORTANT: In the last few lines of the above file, note that the **50572** port number (and the numbers directly below it) has to match the numbers output from the **./start** script in the previous step. As always, also change the **rsmith** username to match your username.

Now that the SSH config file is created, we can easily connect to the SSH tunnels with a simple command:

```
ssh mccarthy_tunnels
```

With the SSH tunnels established, there is nothing else preventing us from connecting Visual Studio Code to our containers.

Now that our Docker environment has been configured, next time will be much simpler. In fact, if McCarthy were to reboot and completely stop your Docker containers, here's all you would need to do to bring everything back online:

```
# From your laptop's shell
ssh mccarthy_tunnels
cd ~/parde
./start
```

Here is a summary of steps to get started with the Visual Studio Code environment, assuming it is already installed. Run VS Code and click the "Extensions" icon on the left bar of the window (Figure 5).

Figure 5.—Extensions icon on left-hand side of Visual Studio Code window.

Install the Remote - SSH extension.

Select the “Open a Remote Window” icon at the bottom-left corner of the window, see Figure 6.

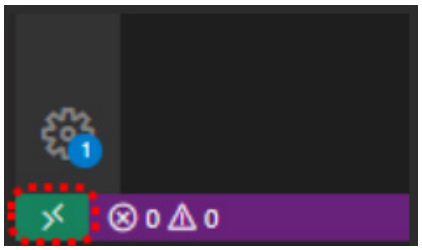

Figure 6.—Open a Remote Window icon.

Select Remote-SSH: Connect to Host (Figure 7).

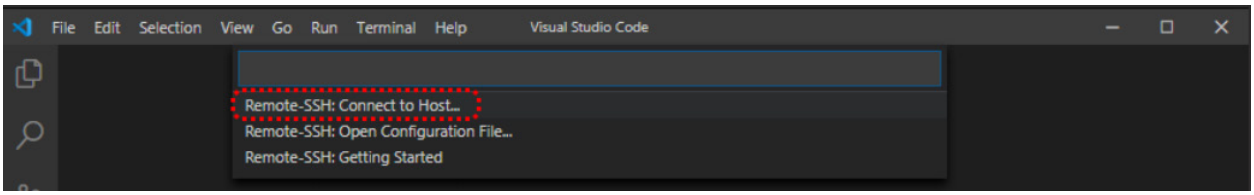

Figure 7.—Remote-SSH: Connect to Host.

Select Add New SSH Host (Figure 8).

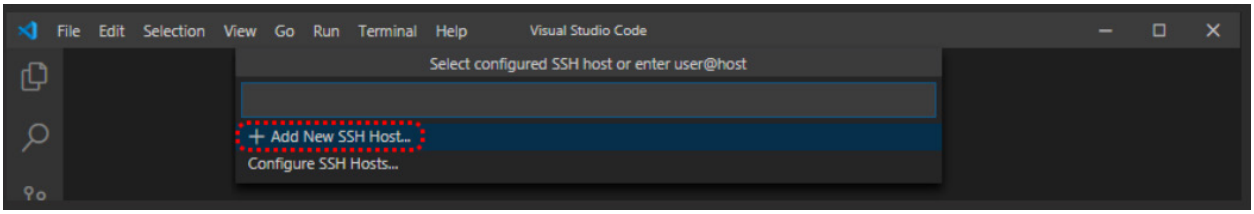

Figure 8.—Add New SSH Host.

Enter the following connection command:

```
-----------------------------------------------------------------------------------------------------------
ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -p 60122 rsmith@localhost
```

It might seem strange that we’re saying to SSH to localhost, but follow the first (red) arrow in the above port diagram. As you can see, your laptop’s port 60122 is connected to the SSH server running inside of your development container (Figure 9).

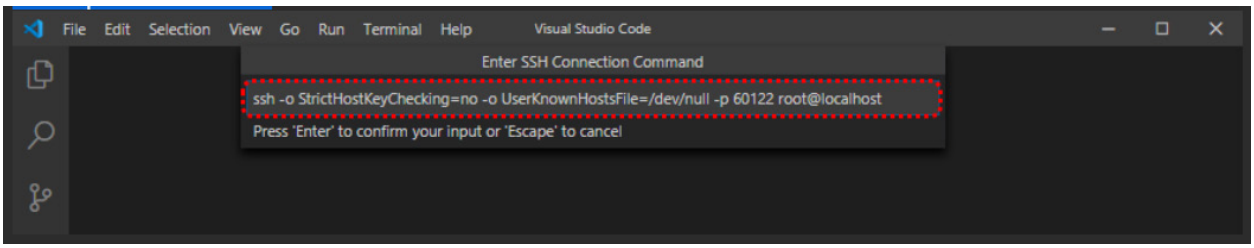

Figure 9.—SSH server command to run inside a development container.

Select the default SSH configuration file (Figure 10).

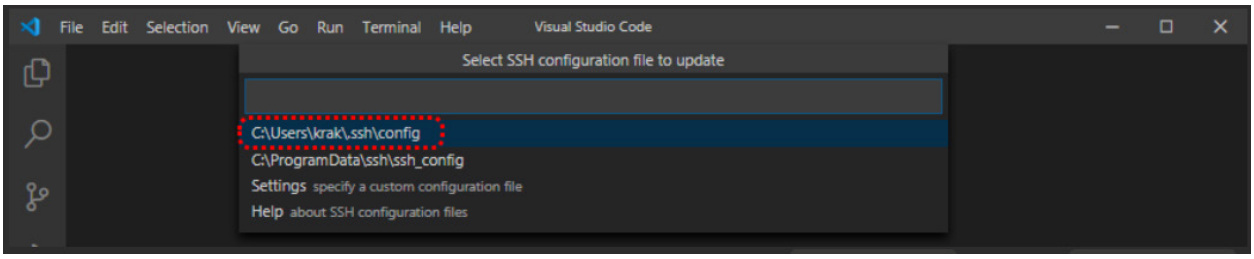

Figure 10.—Default SSH configuration file,

A message will appear in the bottom-right corner of the window titled Host added!. Click the Open Config button (Figure 11).

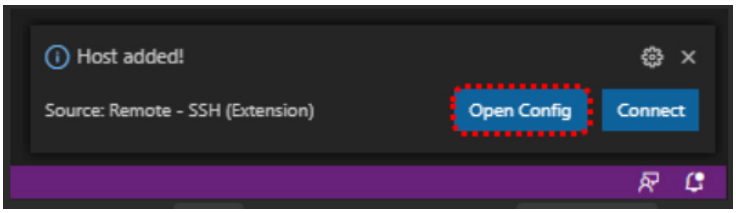

Figure 11.—Open Config button.

By default the **Host** value will be labeled as **localhost** which can be confusing. Change the value of **Host** to **docker_tunnels**. The final ssh config file should appear as follows. Once that is done, save and close the file.

```
Host docker_tunnels
  HostName localhost
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
  Port 60122
  User rsmith
```

Now that we have the IDE configured, let's connect to our containers. Once again, click the "Open a Remote Window" icon at the bottom left corner of the windows (Figure 12).

Figure 12.—Open a Remote Window icon.

Select Remote-SSH: Connect to Host (Figure 13).

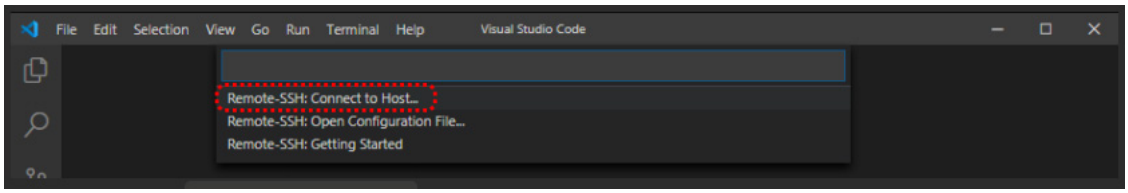

Figure 13.—Remote-SSH: Connect to Host.

Click our new **docker_tunnels** host (Figure 14).

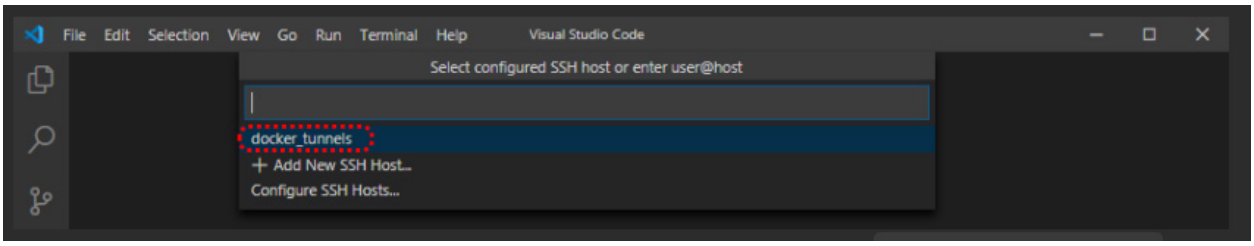

Figure 14.—Selecting docker_tunnels as a host.

After a few seconds, it should prompt you for the password for **rsmith@localhost**.

Recall from Section 1.1 that the Docker-Compose script for this environment will create a user in the main OS container that matches your username and has a password ***dev***. This is what we are logging into with the Remote-SSH extension. So enter ***dev*** as the password (Figure 15).

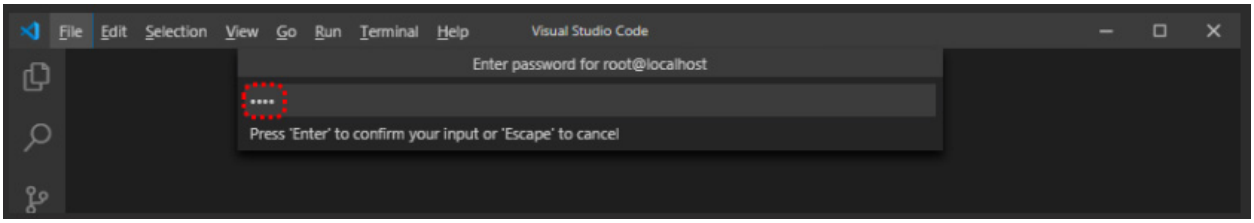

Figure 15.—Enter password.

If all goes well, you'll see the bottom-left corner of the screen show that you are connected via SSH to the **docker_tunnels** host (Figure 16).

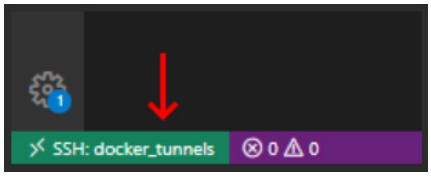

Figure 16.—Shows connection via SSH to the **docker_tunnels** host.

Now, we need to install the VS Code extensions on the container. Click the **Extensions** icon on the left bar (Figure 17).

Figure 17.—Extensions icon in VS Code.

Search for and install the following extensions. Note that the Install button now says Install in SSH: **docker_tunnels**

- C/C++ (ms-vscode.cpptools)
- CMake Tools (ms-vscode.cmake-tools)
- CMake (twxs.cmake)

(Once these extensions are installed. Note that they may say Reload Required)

Click any of the blue Reload Required buttons.

Click Open Folder (Figure 18).

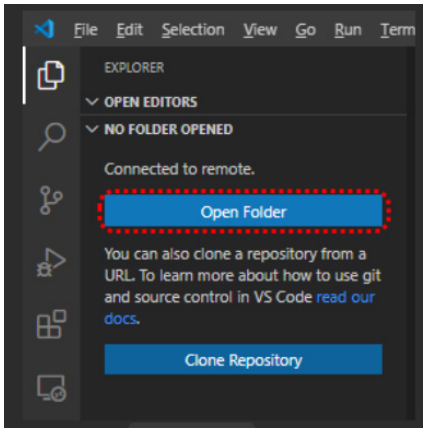

Figure 18.—Open Folder button in VS Code.

Select the **home/rsmith/apc/apc** directory (Figure 19).

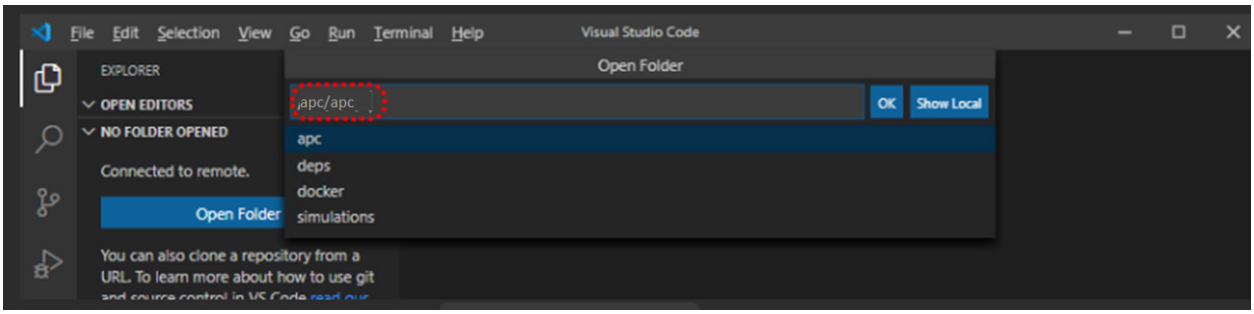

Figure 19.—Selecting apc/apc directory.

You may be prompted for the password again (*dev*).

You may also be prompted to run CMake's configure step on the apc project (Figure 20).

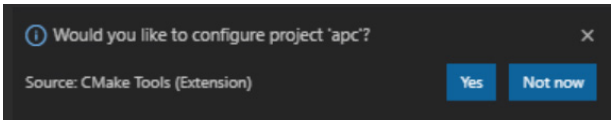

Figure 20.—CMake's Tools configure step as seen in the bottom right corner of VS Code.

You can click **Yes** on the configure prompt. However, if the project has ever been built before, it is recommend to open the VS Code "Command Palette" (View -> Command Palette, or **F1**, or **Ctrl-Shift-P**) and run **CMake: Delete Cache and Reconfigure**.

If you are prompted to select a kit, it is recommended to select whichever one is listed at **/usr/bin/g++** (Figure 21).

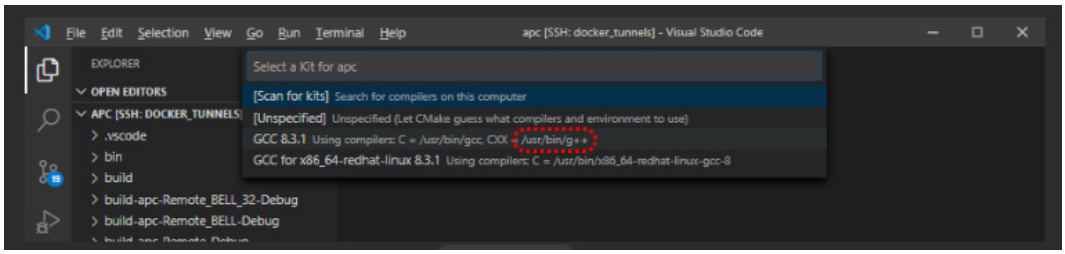

Figure 21.—Selecting the C++ compiler (recommended to select **usr/bin/g++** as shown).

Once that is complete, the Output pane should show **Build files have been written to /amps/apc/build**.

The next step is to build. Open the Command Palette and run **CMake: Build** (Note: This is automatically configured to use all of the CPU cores available.)

If all goes well, the build process will complete with a message **Build finished with exit code 0**.

At this point, you can open a terminal on your laptop, SSH to the Docker host, and then run one of the following two commands:

```
cd ~/parde && ./attach

...or...

docker exec -it parde_${USER} /bin/bash
```

Either of those commands is equivalent. Just pick whichever you prefer.

Now you're on a terminal on the docker container logged in as *rsmith*. You can now navigate to ~/**apc/apc/bin** and then run **./web_portal** to start the APC's main entry point—its web GUI. Once it is running, you can open your development machine's web browser and type **localhost:58400** in the address bar (see the orange arrow in the port diagram above).

3.2 Setup With One Computer

This section details how to set up the Docker environment and VS Code in one computer. The steps are similar to the two computer setup, but with some minor changes. One distinction between this one computer setup and the two computer setup is that we are setting up Docker and the Docker PARDE environment on a Windows machine instead of a Linux one.

In order to run Docker on Windows (and specifically run Linux based containers in Docker on a Windows host), the Windows machine must run a virtual Linux kernel. There are two ways to accomplish this in a modern Docker setup (i.e., version 2.3.7.0, obtained September 2020). The first is to let Docker run its included basic Linux virtual machine (VM). This VM runs in the built-in Windows tool Hyper-V and is the preferred way to run Linux containers on Windows as of September 2020. However, this method will likely become deprecated once the second method is proven by early users. Instructions for setting up Docker on Windows can be found here, https://docs.docker.com/docker-for-windows/install/. At the time of writing, the instructions in the above link reflect this first Docker setup method.

The second method is to use a VM designed for Windows Subsystem for Linux version 2 (WSL2). Several Linux distributions are published for the Windows store to be used with WSL. As of the writing of this document, WSL2 is available in the latest version of Windows 10 (tested with Windows 10 build 2004, Docker version 2.3.6.2). Docker, when used with WSL2, will run its containers with the Linux kernel from a WSL2 operating system installation. If possible, it is recommended to use this second method. As of the writing of this document, instructions for setting up Docker with WSL2 containers can be found here: https://docs.docker.com/docker-for-windows/wsl/.

Once Docker and VS Code are set up on the end user's computer, the process is similar to what is done in the two computer setup. One difference is that it is assumed only one user will use this setup, because it is assumed this Windows machine is owned and used solely by one end-user. The fact that only one user will be logging in also means that port forwarding (tunneling) is not necessarily needed.

Another difference is that the ***startPARDE.bat***, ***stopPARDE.bat***, and ***attachPARDE.bat*** batch scripts are used to interact with the Docker environment instead of the bash scripts used on the Linux computer. Beyond this, since both the PARDE Docker environment and VS code are running on the same computer, the VS Code Remote-SSH feature can connect without SSH tunnels. Per this, the following SSH config entry (**%USERPROFILE%\.ssh\config**) can be used for logging in to the Remote-SSH feature on VS Code:

```
Host docker_local
  HostName localhost
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
  Port 51122
  User rsmith
```

Ensure the *Port* number above matches the one generated by ***setenv.bat*** (it should).

Assuming this entry is set up, the user can open a new Powershell terminal and run:

```
  .\startPARDE.bat
```

This will start the PARDE Docker Compose network. When this step is successfully executed, the user can open VS Code. As with the two computer setup, the user must obtain the Remote-SSH extension via the extensions tab (Figure 22).

Figure 22.—Extension icon in VS Code.

Once this is installed, the user can then use this extension to connect to the *docker_local* SSH host. To do this, click the "Open a Remote Window" icon at the bottom left corner of the window (Figure 23).

Figure 23.—Open a Remote Window icon.

Select Remote-SSH: Connect to Host (Figure 24).

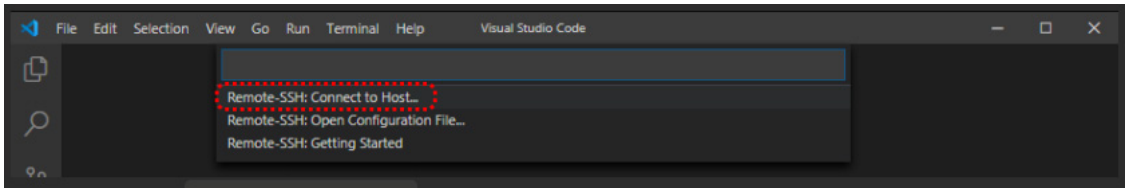

Figure 24.—Remote-SSH: Connect to Host.

Click our new **docker_local** host (Figure 25).

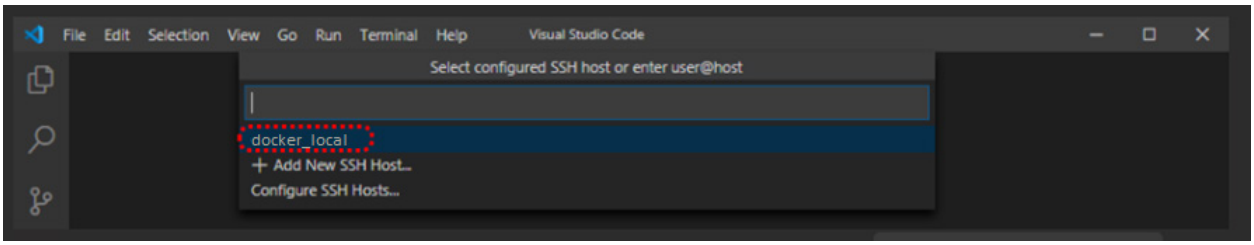

Figure 25.—Select docker_local as SSH host.

After a few seconds, it should prompt you for the password for **rsmith@localhost**.

Recall from Section 1.1 that the Docker-Compose script for this environment will create a user in the main OS container that matches your username and has a password ***dev***. This is what we are logging into with the Remote-SSH extension. So enter ***dev*** as the password

As listed in the previous two-computer section, the *C/C++* and *CMake Tools* extensions must then be installed to this Remote-SSH session of VS Code. Once this is done, it is recommended to clean the CMake cache and rerun as stated in the two-computer section. This can easily be done using the VS Code "Command Palette" (View -> Command Palette, or **F1**, or **Ctrl-Shift-P**) and running **CMake: Delete Cache and Reconfigure**. Once this is done, the APC code project can be built by running **CMake: Build** in the Command Palette.

3.3 Running the APC

To run the APC, first go through either the steps in Section 3.1 or 3.2 to set up and build the APC in either the two or one computer setup. Once these are completed, users can start a SSH session in the main OS container in Windows by changing directory to the location where PARDE was unzipped, and running

```
.\attachPARDE.bat
```

Or in Linux, by changing directory to the PARDE install directory and running

```
./attach
```

Once this step is completed and the user is in a SSH session, logged into the main OS container user account, the web GUI can be run via

```
cd ~/apc/apc/bin
./web_portal
```

Users can then open their browser of choice (Chrome or Firefox recommended) and go to

http://localhost:51140/ (if using the one-computer Windows setup and not using SSH tunneling)

or

http://localhost:58400/ (otherwise)

This will open the APC web portal main page shown in Figure 26.

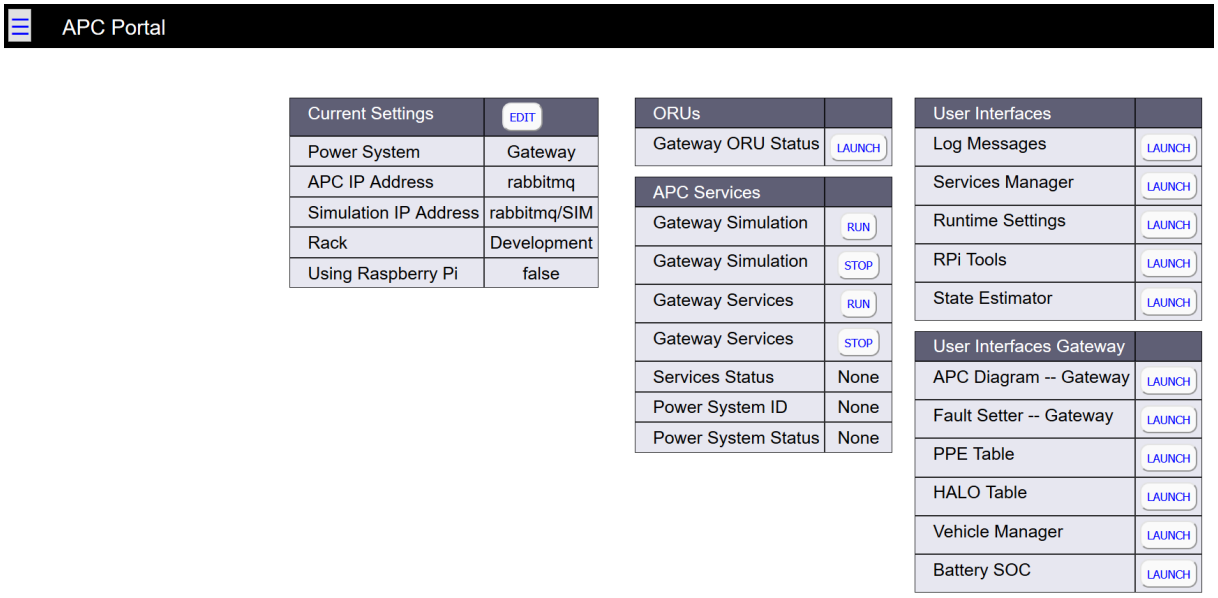

Figure 26.—APC web portal main page.

Once on this page, users can start the service manager by clicking the *run* button next to *Gateway Services*. When this is successful, the user will be able to see message timestamps next to *Services Status* increasing about once per second. The user can then click the *run* button next to *Gateway Simulation* to start the simulation model (***DSG_main***). When the simulation is running, timestamps will be seen next to *Power System Status* in the web GUI as well.

Once both the services and simulation are running, the APC Diagram should be updating with data from the simulation. To observe this data, click the Launch button next to APC Diagram. The diagram shown in Figure 27 will show up in a new tab.

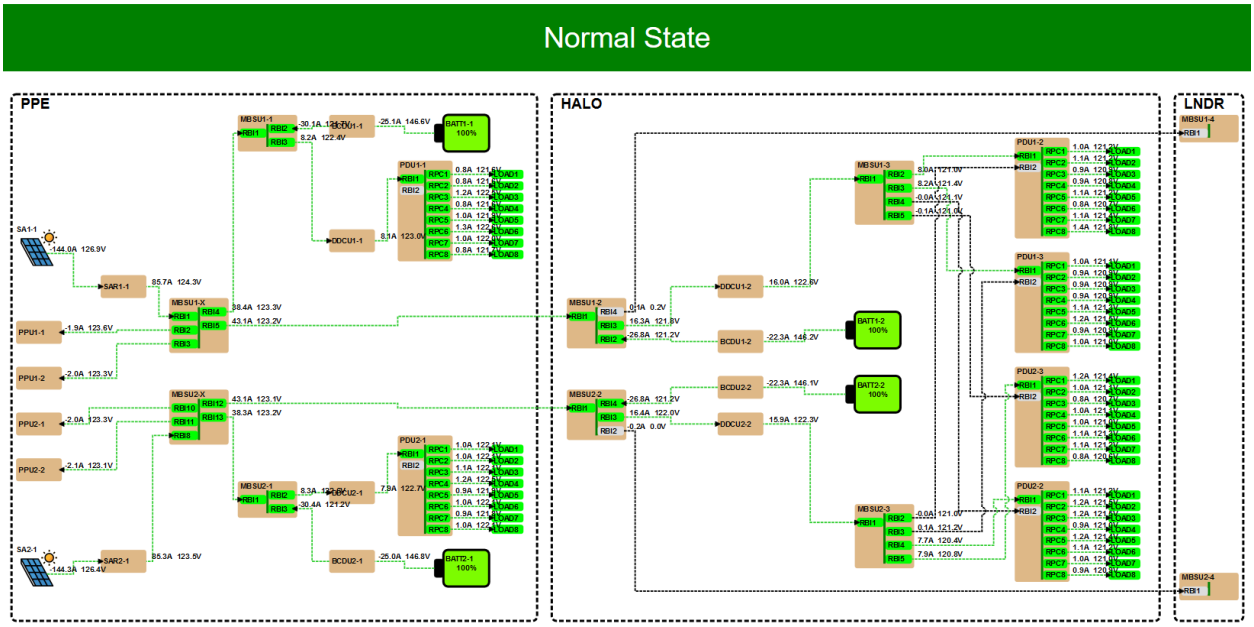

Figure 27.—Gateway power system diagram showing normal state in APC web portal

Users can verify the simulation is running and successfully getting data to the APC via RabbitMQ by verifying that the diagram's voltages and currents are all positive and changing, and that the simulation state at the top of the diagram reads "Normal State." Users may be able to observe other parts of the simulation behavior, such as the vehicle going from insolation (light background, power produced by solar arrays) to eclipse (dark background, and no power produced by the solar arrays).

After verifying the system is running properly, it is recommended to use the GRC Vehicle Manager Emulator to propose a load schedule. The simulation has an outstanding bug where it may run unstable if no load schedule is proposed and the user injects a fault. Setting a load schedule can be done by returning to the main web portal screen and clicking *Launch* next to *Vehicle Manager*. This starts a new tab with the screen shown in Figure 28.

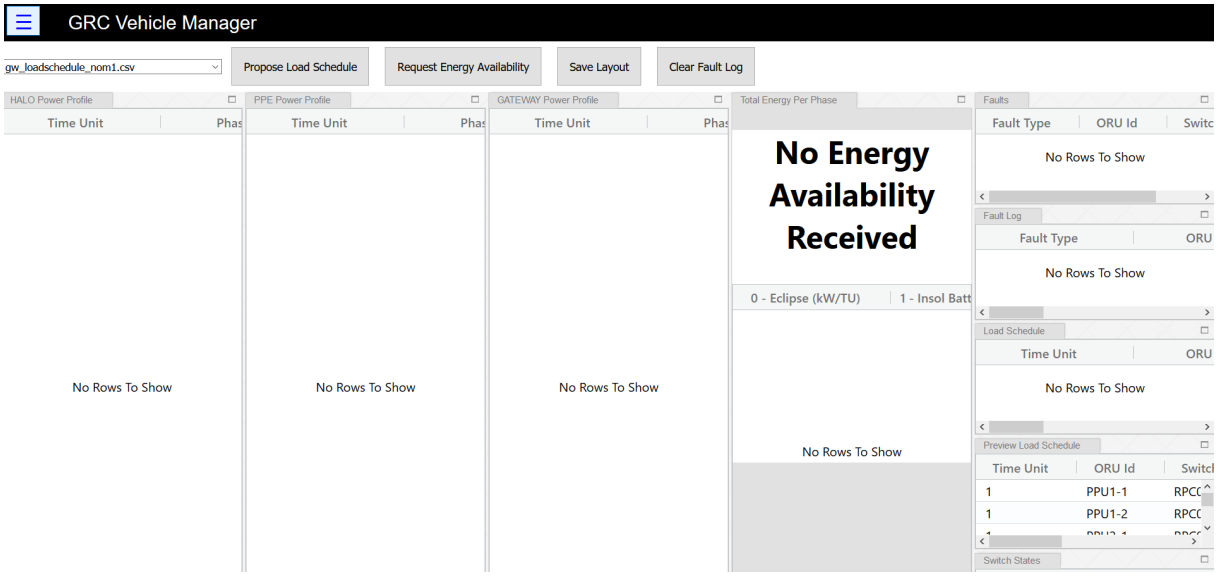

Figure 28.—GRC Vehicle Manager page in APC web portal.

A load schedule can be proposed to the APC by selecting one in the dropdown in the top left of the screen, and clicking *Propose Load Schedule*. The APC will then accept the load schedule and will begin running it to drive the loads. Users may produce their own load schedules, using the existing ones as a reference. Figure 29 is a screenshot of opening one of the load schedule CSV files in VS Code.

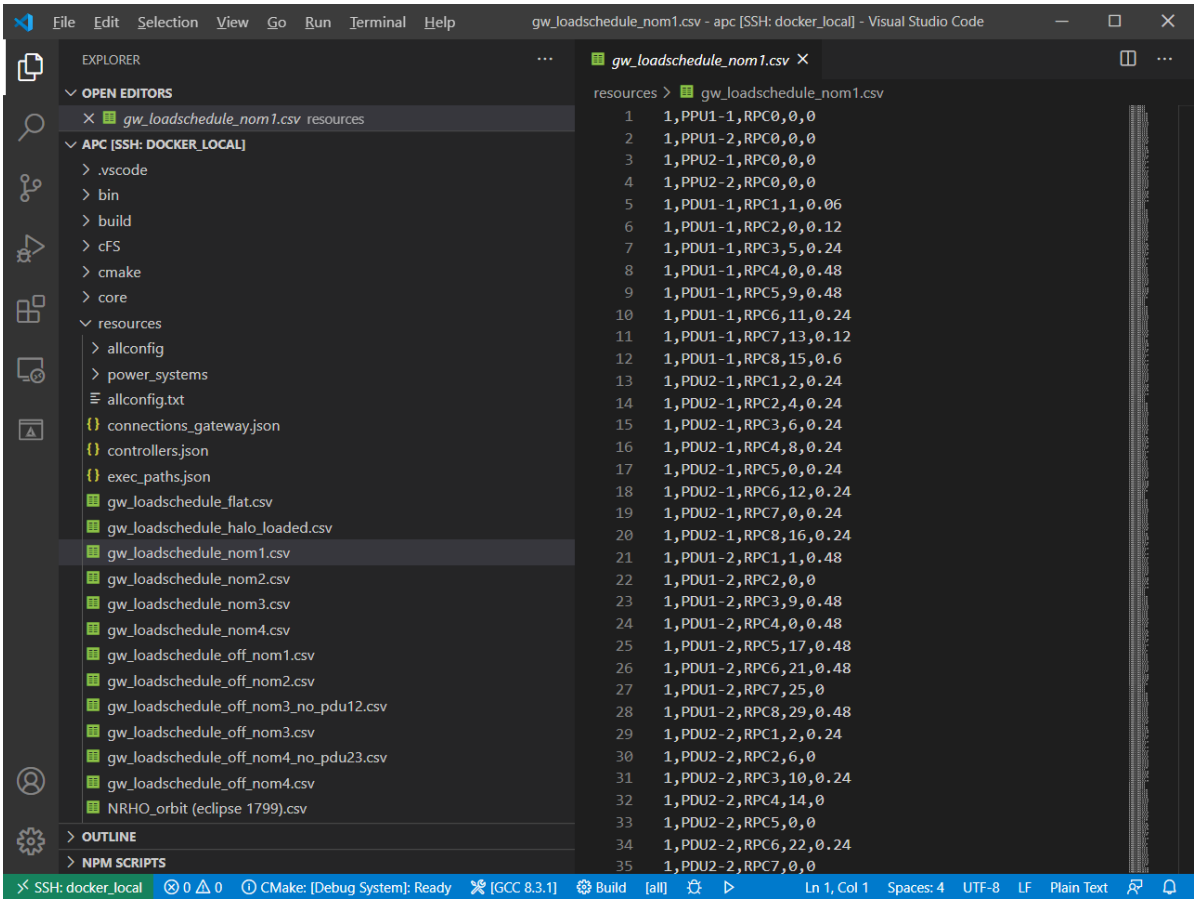

Figure 29.—Raw data from gw_loadshedule_nom1.csv shown in VS Code.

As seen in this screenshot, these are plain text comma separated value data files and can be modified in a text editor, though they may be easier to edit using a spreadsheet editor such as Microsoft Excel. When opening one of these load schedule files in Excel, they will appear as shown in Figure 30.

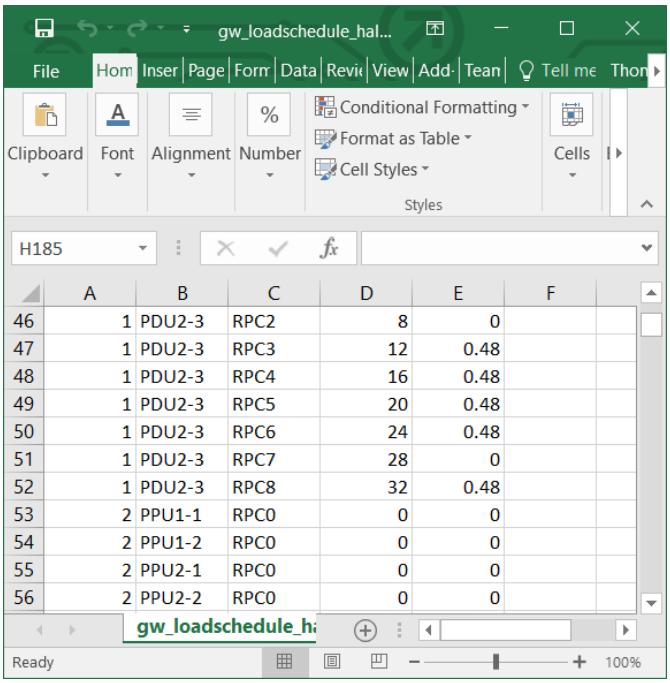

Figure 30.—Example of a load schedule files opened in Excel.

Note that the columns in the load schedule file correspond to (from left to right): Time unit, ORU ID, Switch ID, Priority, and Nominal Power. Note that the time unit specified in this file represents increments of 5 minutes. Further note that the power consumed by the loads at each ORU/Switch in the simulation are set by the nominal power setting in the load schedule file.

In addition to the load schedule files, the simulation also runs a certain orbit. CSV files containing orbit data are included in the model. The orbit simulation is used to determine whether the vehicle is in insolation (where the solar arrays produce power, indicated by a white background in the web GUI), as well as when the vehicle is in eclipse (when the solar arrays do not produce power, indicated with a black background). The orbit files (e.g., NRHO_orbit (VSM).csv) show vehicle, sun, and moon location vs time, as seen in Figure 31.

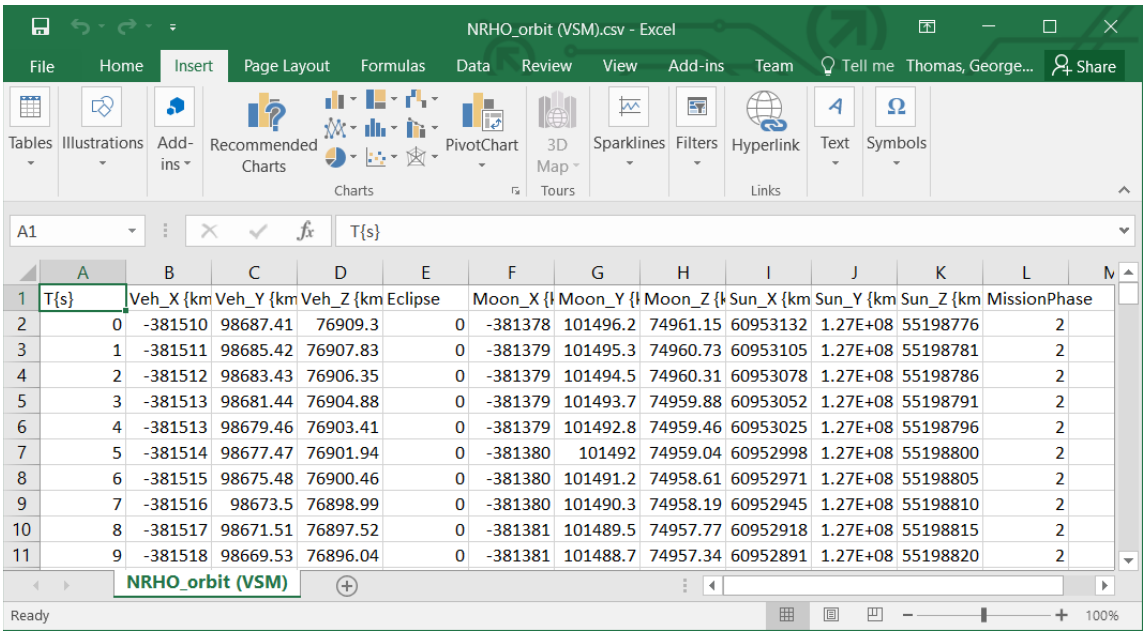

Figure 31.—Example of an orbit file opened in Excel.

The orbit model is driven by these inputs and runs separately from the rest of the simulation model. Timing parameters for the orbit model may be adjusted to speed up or otherwise change the orbit behavior, by adjusting parameters in **apc/resources/allconfig/allconfigGateway.xml**, under the XML entry shown below:

```
<ServiceBroadcastTime id='16' timeUnitPeriodInSeconds='300'
 timeBroadcastPeriodMillis='1000' simTimeSpeedupMultiplier='20'
 timeUnitsInOneOrbit='24'
 name='Time' description='Broadcasts time and oribtal data.'
 orbitData='NRHO_orbit (VSM).csv' />
```

Note that the orbit parameters are intended to be changed before running the Gateway services and simulation model. Make sure that the services and simulation are restarted in the case where any changes are made.

Once a load schedule is selected and assuming the orbit parameters are set as desired, users can inject faults into the system. This can be done by opening the Fault Setter GUI, by returning to the main web portal and clicking *Launch* next to *Fault Setter – Gateway*. This will bring up the screen shown in Figure 32 in a new tab.

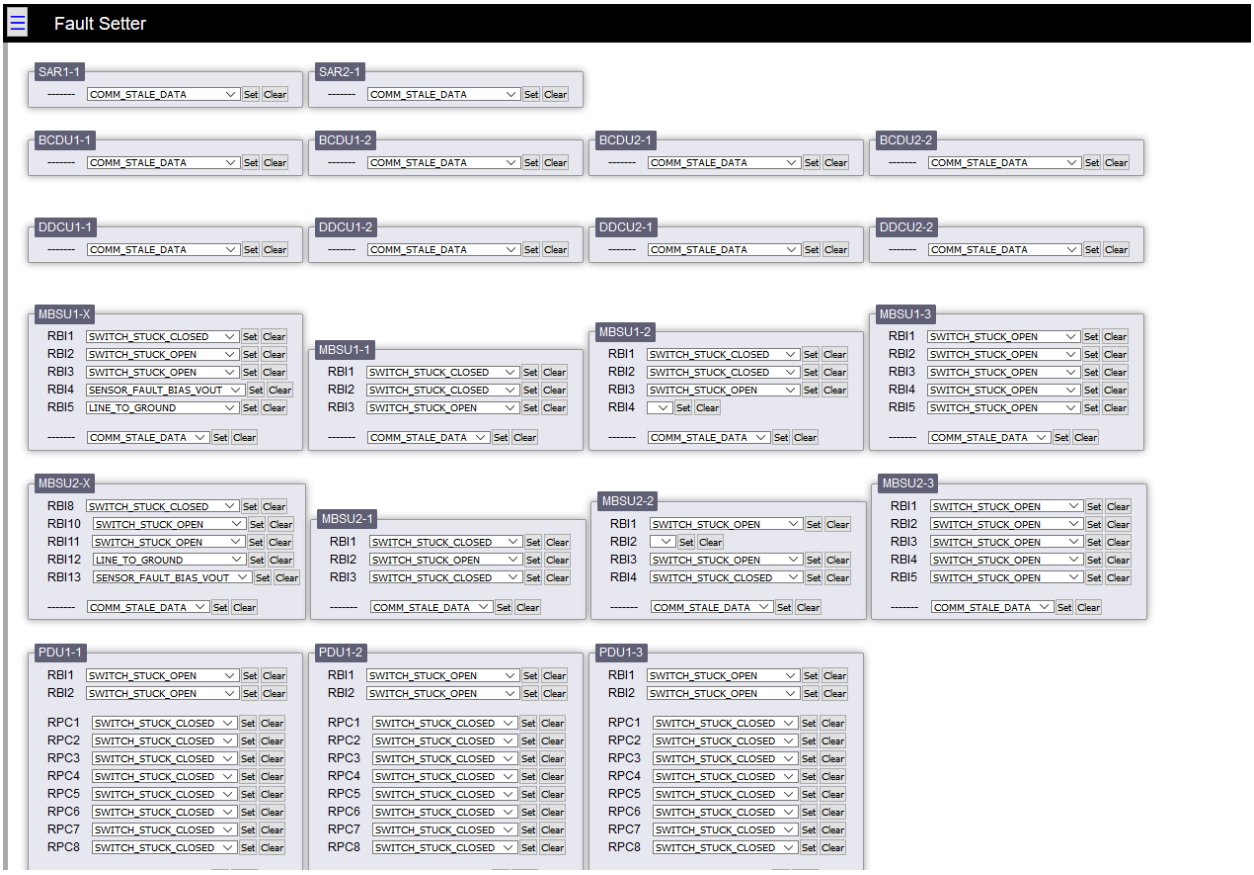

Figure 32.—Fault Setter screen in APC web portal.

This screen lists the ORUs in the system, and if they exist, any RBIs or RPCs in that ORU. These ORUs or their RBIs/RPCs can be faulted using the appropriate dropdown and then clicking *Set*. For demonstration purposes, one can select *Short Circuit Fault* under MBSU1-2, RBI2, and click *Set*. The fault will be received by the simulation, and the APC will detect the fault and reconfigure as shown in the diagram in Figure 33.

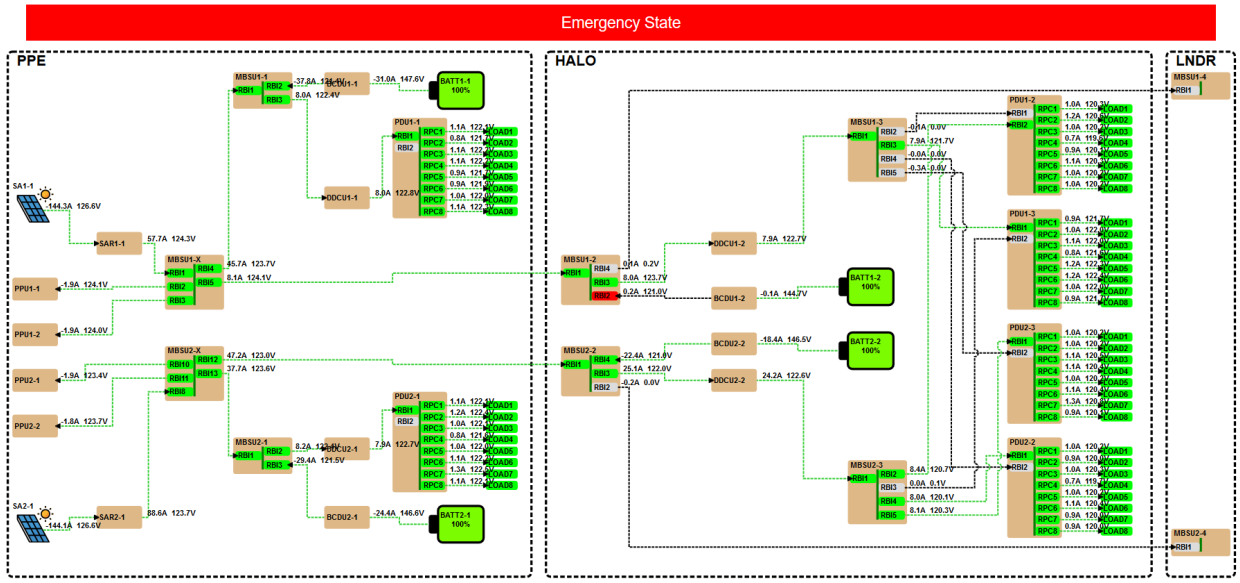

Figure 33.—Gateway power system showing the system in emergency state.

Note that the APC's logic in ***ServiceFaultManager.so*** detects this as a fault, and the faulted RBI is now indicated as red. Also note that the reconfiguration logic in ***ServiceMmr.so*** categorizes this as a battery fault, and responds by setting up 3 out of the 4 PDUs to be powered by the bottom power domain (the one that does not contain a fault). It sets up the remaining 1 PDU to be powered by the top power domain, since it has reduced energy availability due to the battery fault. The fault can be cleared (representing it being physically fixed) by clicking *Clear* next to the fault in the Fault Setter GUI. Alternatively, to represent a scenario where the vehicle manager addresses the fault by adjusting the load schedule, users can set a new load schedule that is more appropriate given the fault. Once the APC gets an updated fault schedule, it will report *Restorative State*. Figure 34 shows an example of proposing a load schedule where the loads in PDU1-2 are disabled, representing a scenario where the Vehicle Manager decides to handle the lower energy availability by reducing the number of powered loads in the schedule.

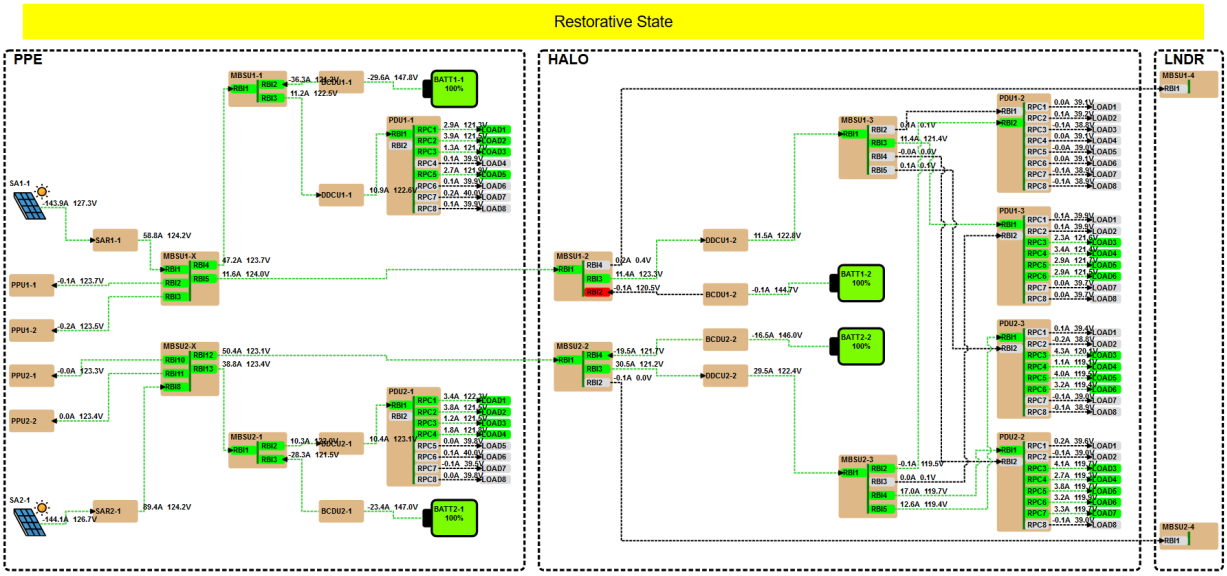

Figure 34.—Gateway power system showing the system in restorative state.

Testing functions in the APC typically consists of actions like these in the web GUI. As users develop their own fault management algorithms in ***ServiceFaultManager.so*** and ***ServiceMmr.so***, they can run whatever fault scenarios they desire using the web GUI and observe the APC's response. Users are encouraged to run fault scenarios to understand how the example fault management and MMR services work with the APC before writing their own.

4.0 API Documentation

4.1 ServiceFaultManager

Functions:

void ListenForFaults();

A method meant to run in a separate thread that runs a loop that checks for any faults added or removed, by polling the RabbitMQ exchanges:
"EXCHANGE_PUBLISH_FAULTED_COMPONENTS" and
"EXCHANGE_PUBLISH_FAULTED_COMPONENTS_NOT_VETTED"

void AddFaultToFaultedComponentMapLatest(Fault::ptr_t ptrFault);

Add faults and Updates the latest FaultedComponetMap.

void RemoveFaultToFaultedComponentMapLatest(Fault::ptr_t ptrFault);

Check if "GetNumberOfFaults()" is larger than "0" and calls "RemovesFault(ptrFault)" to remove that fault.

bool CheckForShortCircuitFaults();

Checks for faults inserted and validates if "Faultype" if it is a "SHORT_CIRCUIT".

bool CheckForShortCircuitFaultsRemoved();

Checks for faults inserted and if any "Faultype" is "SHORT_CIRCUIT" it gets removed.

bool WriteFaultRmq(Fault::ptr_t faultToWrite);

bool RemoveFaultRmq(Fault::ptr_t faultToRemove);

Validates if "IsRemoveFault()" is false. If so set "SetRemoveFault()" true.

4.2 ServiceMMR

Functions:

PowerSystemSwitchStates::ptr_t HandleReconfiguration(FaultedComponentMap::ptr_t ptrFaultedComponentMap);

Calculates the best configuration of the power system based on the availability of the components in the system. This returns a PowerSystemSwitchStates:ptr object of that best configuration.

4.3 Power System Object

Power system data is represented in the APC codebase with a single, dynamically constructed object, called the PowerSystem object (core/PowerSystem/PowerSystem.h). The API for the power system object contains, among other things, methods to read sensor data and switch states, as shown below.

```
// Example: Read Telemetry From MBSU1-1 RBI1 Voltage In
float mbsu11Rbi1VIN;
ptrPowerSystem->GetSensorData("MBSU1-1", "RBI1", "VIN", &mbsu11Rbi1VIN);

// Example: Read Switch State From PDU1-1 RPC1
int pdu11Rpc1SwitchState;
ptrPowerSystem->GetSwitchState("PDU1-1", "RPC1", &pdu11Rpc1SwitchState);
```

Functions:

bool GetSensorData(std::string oruId, std::string switchId, std::string sensorId, float * ptrValue);

Retrieves sensor data from specified ORU, switch ID, and sensed quantity (voltage, current, etc.). The sensor data is passed back to calling function via reference (*ptrValue*)

bool GetSwitchState(std::string oruId, std::string switchId, int * ptrState);

Retrieves current switch state data from specified ORU and switch ID. The switch state data is passed back to calling function via reference (*ptrState*)

5.0 Troubleshooting

- **Issue**: Docker compose fails to build the CentOS/APC image, gives issues saying libgcc or other libs to be installed via yum are incompatible
  - Try updating CentOS base image:

```
docker pull centos:8
```

  - This issue may occur if the centos:8 base image is updated (as may occur from time to time), and you try to rebuild with an old image that is stored on your device. Updating the image will fix the issue.

- **Issue**: APC voltages and currents may become unstable
  - This may occur when a fault is injected or a switch is toggled
  - This can be fixed by turning the gateway simulation off and back on again
  - This issue can be avoided in the first place by going to the Vehicle Manager emulator page on the web GUI and requesting a load schedule before taking any actions in the APC.

- **Issue:** Cannot start Docker containers and see the following message on Docker GUI:

```
Cannot start Docker Compose application. Reason: Command failed: docker-compose --project-
directory "C:\Users\USER\AMPS\apc\repos\apc\docker" up -d The USER variable is not set. Defaulting
to a blank string. Missing mandatory value for "ports" option interpolating
['${PARDE_PORT_RABBITMQ:?Missing PARDE_PORT_RABBITMQ}:15672'] in service "rabbitmq": Missing
PARDE_PORT_RABBITMQ
```

  - Stopping and Re-starting the Docker containers will normally clear this error. This is done with the following commands if running Windows:

```
.\stopPARDE
.\startPARDE
```

  - Or the following commands if running Linux:

```
./stop
./start
```

- **Issue:** Ports are not available when running the **.\startPARDE.bat** command:

```
PS C:\Users\USER\AMPS\apc\repos\apc\docker> .\startPARDE.bat
C:\Users\USER\AMPS\apc\repos\apc\docker>call setenv.bat
Starting parde_USER_rabbitmq ...
Starting parde_USER_mysql    ... error
Starting parde_USER_rabbitmq ... error
ERROR: for parde_USER_mysql  Cannot start service mysql: Ports are not available: listen tcp
0.0.0.0:51106: bind: An attempt was made to access a socket in a way forbidden by its access
permissions.
ERROR: for parde_USER_rabbitmq  Cannot start service rabbitmq: Ports are not available: listen tcp
0.0.0.0:51172: bind: An attempt was made to access a socket in a way forbidden by its access
permissions.
ERROR: for mysql  Cannot start service mysql: Ports are not available: listen tcp 0.0.0.0:51106:
bind: An attempt was made to access a socket in a way forbidden by its access permissions.
ERROR: for rabbitmq  Cannot start service rabbitmq: Ports are not available: listen tcp
0.0.0.0:51172: bind: An attempt was made to access a socket in a way forbidden by its access
permissions.
ERROR: Encountered errors while bringing up the project.
```

  - If this error is seen run the following command from the terminal:

```
netsh interface ipv4 show excludedportrange protocol=tcp
```

  - If it shows that the 511XX ports are excluded, you might need to disable Hyper-V, reserve those ports, and you can re-enable Hyper-V with the following commands in an terminal run as administrator:
    – Disable Hyper-v, this will require one or more restarts:

```
dism.exe /Online /Disable-Feature:Microsoft-Hyper-V
```

    – When completed with all required restarts, reserve the ports you want so Hyper-v doesn't reserve it back:

```
netsh int ipv4 add excludedportrange protocol=tcp startport=51106 numberofports=66
```

    – Re-Enable Hyper-V, this will require one or more restarts:

```
dism.exe /Online /Enable-Feature:Microsoft-Hyper-V /All
```

- **Issue:** If the web portal fails to start when running the **.\web_portal** command:

```
[USER@effd2d5bbeb2 bin]$ ./web_portal
INFO: Found /home/USER/apc/apc/tools/web_portal/www/index.html
INFO: Starting websocket server
INFO: In ../../utilities/Settings/private/RuntimeSettingsMap.cpp:18 Instance -- loaded cascaded
settings
ERROR: In ../../utilities/MessageBroker/private/ChannelManager.cpp:113 CreateChannel --
std::exception while creating AmqpClient::Channel. [a socket error occurred]
        Parameters: localhost
terminate called after throwing an instance of 'boost::wrapexcept<boost::lock_error>'
  what():  boost: mutex lock failed in pthread_mutex_lock: Invalid argument
Aborted
```

  - This may be due to the **runtime_settings.json** file not having the correct settings. Update to the following:

```
{
    "runtimeSettingsMap": {
      "apc_serv": {
        "serviceName": "",
        "settingName": "apc_serv",
        "settingType": "String",
        "settingValue": "rabbitmq"
      },
      "database_servername": {
        "serviceName": "",
        "settingName": "database_servername",
        "settingType": "String",
        "settingValue": "mysql"
      },
      "sim_ip_address": {
        "serviceName": "",
        "settingName": "sim_ip_address",
        "settingType": "String",
        "settingValue": "rabbitmq/SIM"
      }
    }
  }
```

Reference

1. Csank, J.T., et al. “An Autonomous Power Controller for the NASA Human Deep Space Gateway,” AIAA-2018-4634, AIAA Propulsion and Energy Forum, Cincinnati, OH, July 9-11, 2018.