

Project Report

Enable white label B2B2C deployment on-cloud for low carbon app
Robert Brown, London South Bank University, May 2021

Executive Summary

This document described implementation of Phases 1 and 2 of a collaborative project between Sustainable Innovation at London South Bank University, and a London based SME operating in the smart grid sector. The aim was to make the SME's Low Carbon App white label deployable for any cloud.

The project phases were as follows.

1. Whitelabel deployment development

Phase 1 – Whitelabel deployment development	2
Objectives	2
Description	3
Architecture.....	4
Build Environment	5
Deploy Environment.....	6
Development.....	7
Run the following commands to start the app in dev mode	7
Check the services which came up.....	7
Check the logs	7
Open app in browser	7
Testing.....	8
File upload	8
File Download.....	8
Optimise.....	9
Follow the link given by Location in response header, using GET request	9
Local Deployment.....	11
Production Deployment (manual).....	12
Build the API / Worker and Frontend images locally for remote deployment	12
Login to Dockerhub	12
Push the images to Dockerhub.....	12
Copy docker-compose-deploy.yml to remote server.....	12
Open an ssh session to remote server	12
Edit the whitelabel configuration variables in docker-compose-deploy.yml.....	12
Save the file and close.....	13
Set up Docker.....	13
Pull required images.....	13
Start app	13
Configure firewall.....	13
Test app	13
Production Deployment (automatic).....	14
AWS Console	14
Attach static IP	16
Open firewall ports.....	16
Test the app	16
Summary	18
Conclusions and Recommendations.....	19
Conclusions	19
Recommendations.....	19

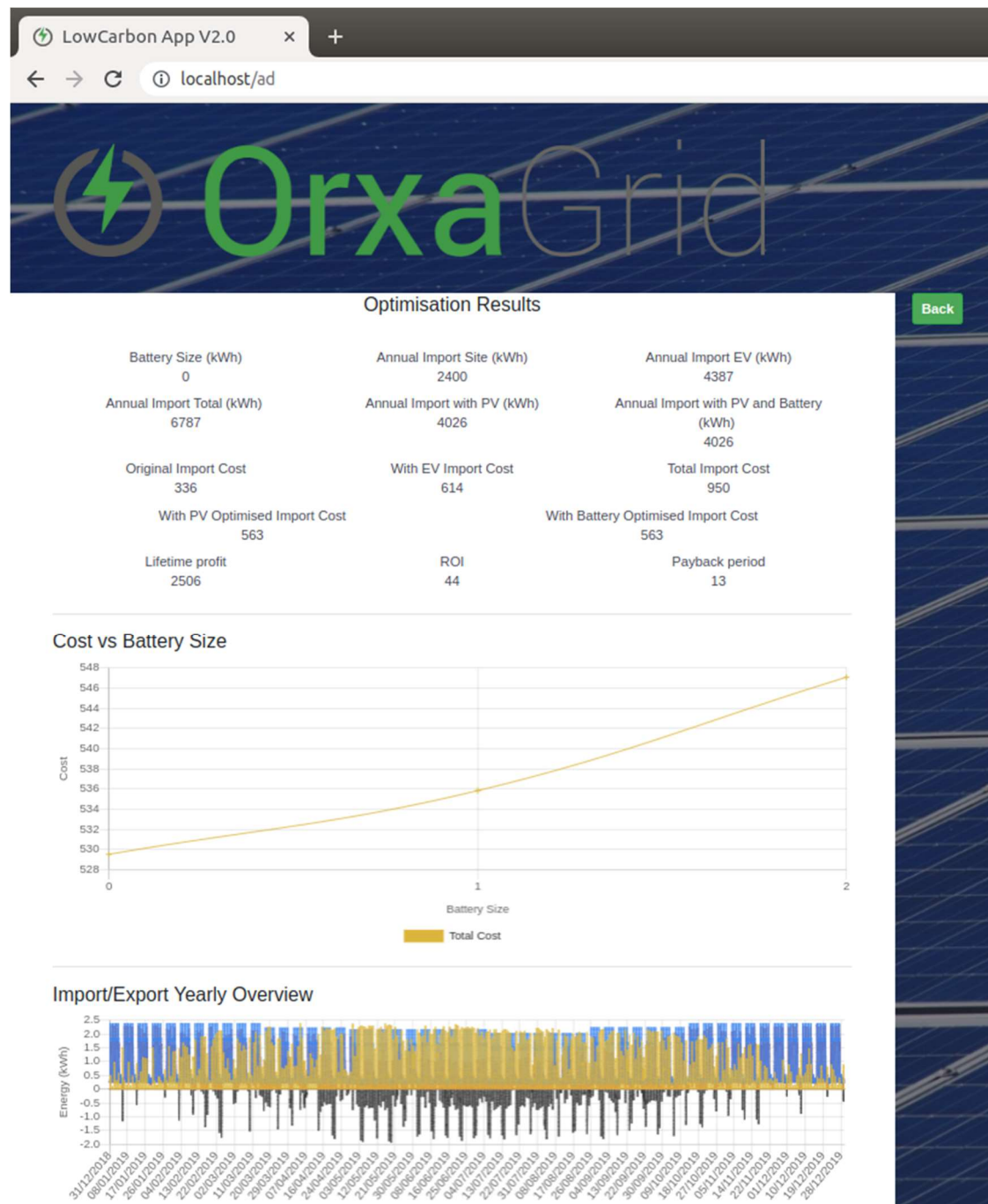
Phase 1 – Whitelabel deployment development

Objectives

- Split existing app into microservices. Strip app down to core functionality for purposes of project. Implement core functionalities using any-cloud services that can be containerised.
- Containerise each service, so that it can run in any cloud, or on a development machine.
- Orchestrate services for deployment. Wrap containers in an orchestration service that can bring all services up with minimal deployment effort.

Description

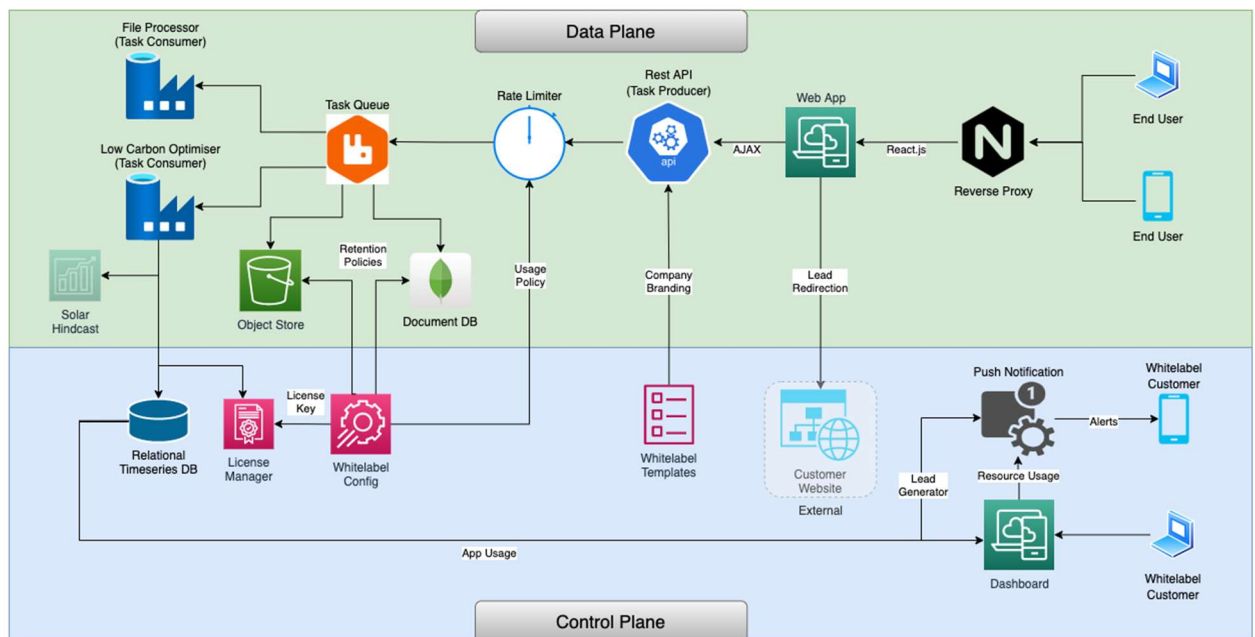
This application is a solar PV and battery system optimiser designed to be deployed by system installers for integration with their company website.



report

Architecture

- Reverse Proxy routes requests to the relevant service (todo)
- Web App serves HTML/CSS/JS content
- REST API interchanges data between the Web App and all other backend services
- Rate limiter prevents a single user over using the api (todo)
- Task queue routes work between API and data factories
- File processor is a data factory for cleaning uploaded files
- Low Carbon Optimiser is a data factory for sizing battery and solar PV systems
- Solar Hindcast provides solar generation data for a given location (todo)
- Object Store persists uploaded and cleaned files
- Document DB persists optimisation results
- Relational DB stores app usage statistics
- License Manager restricts operation of the optimiser to licensed installations (todo)
- Whitelabel Config enables custom branding of the Web App
- Whitelabel Templates enables custom branding of the optimisation reports (todo)
- Push Notifications generates sales leads and sends to system installers (todo)
- Dashboard displays business intelligence about how the app is being used (todo)



Architecture Diagram

Build Environment

This application should be built using Linux Docker on a Mac, Linux or Windows host with x64 architecture.

Deploy Environment

This application has been tested on a Linux Ubuntu host with x64 architecture, but should be deployable on any modern Linux host with Docker installed.

Development

- Install Docker (<https://docs.docker.com/get-docker/>)
- Install Docker Compose (<https://docs.docker.com/compose/install/>)

Run the following commands to start the app in dev mode

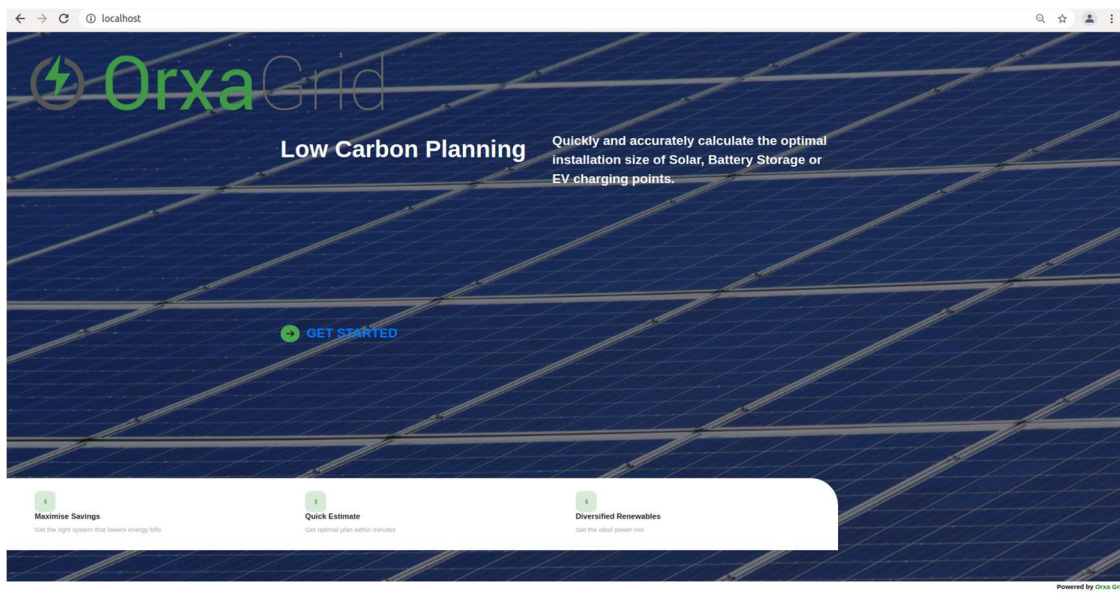
Frontend and API services will hot-reload

```
docker-compose pull
docker-compose build .
docker-compose up -d
```

Check the services which came up
`docker stats`

Check the logs
`docker-compose logs -f`

Open app in browser
`http://localhost`



homepage

Testing

- Install Postman (<https://www.postman.com/downloads/>)
- Import API endpoint tests into Postman (orxagrid_si.postman_collection.json) (included in root of this repo)
- Run the following endpoints

File upload

Choose *'sample_load.csv'* as *'file'* (included in root of this repo)

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

☒ file

Send the request, and note the returned file handles

POST localhost:5000/upload

file upload

POST localhost:5000/upload

Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

file sample_load.csv

lat 51.50635249404038

lon -0.10415590573792667

Key Value Description

Body Cookies Headers (6) Test Results

Status: 200 OK Time: 3.04 s Size: 280 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "handle": "a772e045-b4e9-4382-aa2d-7c4691a14cb5",
3   "raw_upload": "54d13d9c-1ef9-45d6-8eaa-0690bf3ca730"
4 }
```

File Download

Replace the file handle with that noted in previous step

GET localhost:5000/download/a772e045-b4e9-4382-aa2d-7c4691a14cb5

Send Save

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (6) Test Results

Status: 200 OK Time: 312 ms Size: 54.86 KB Save Response

Pretty Raw Preview Visualize Text

```
1 kWh
2 2.6
3 0.4
4 2.1
5 2.1999999999999997
6 1.0
7 0.5
8 0.7
```

Optimise

GET optimise

localhost:5000/task_optimise

Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Status: 202 ACCEPTED Time: 64 ms Size: 333 B Save Response

Header	Value
Content-Type	application/json
Content-Length	66
Location	http://localhost:5000/task_callback/654e1989-da20-42b0-8c6c-2c21652740bc
Access-Control-Allow-Origin	*

Bootcamp

optimise

Follow the link given by Location in response header, using GET request

Content-Length	66
Location	http://localhost:5000/task_callback/654e1989-da20-42b0-8c6c-2c21652740bc
Access-Control-Allow-Origin	*

location

Response status code should be 202 until the results are ready

GET optimise

GET http://localhost:5000/task_callback/4c0f2591-fb3b-4592-b38e-952d711207c8

Send Save

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results

Status: 202 ACCEPTED Time: 84 ms Size: 185 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {}
```

Keep calling the endpoint

Response status code should be 200 when results are ready

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, URL: `http://localhost:5000/task_callback/4c0f2591-fb3b-4592-b38e-952d711207c8`, Environment: No Environment.
- Params Tab:** Query Params table with columns KEY, VALUE, and DESCRIPTION.
- Body Tab:** Shows the response body in JSON format, with status 200 OK, time 100 ms, and size 2.27 MB.

Query Params Table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

Response Body (JSON):

```
1 {
2   "charts": {
3     "buildings": [
4       {
5         "name": "building 1",
6         "pv_cost_curve": {
7           "Profit_PA": [
8             0.0,
9             84.3422506897578,
10            85.33807868256034,
```

Local Deployment

Locally built images, with production web servers, no hot-reloading of code

```
docker-compose -f docker-compose-prod.yml build
docker-compose -f docker-compose-prod.yml up -d
```

```
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$ docker-compose -f docker-compose-prod.yml up -d
Creating network "orxagridlcapp2_default" with the default driver
Creating object-store-prod ...
Creating api-prod ...
Creating nosql-db-prod ...
Creating worker-prod ...
Creating task-queue-prod ...
Creating frontend-prod ...
Creating object-store-prod
Creating api-prod
Creating nosql-db-prod
Creating frontend-prod
Creating worker-prod
Creating object-store-prod ... done
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$
```

local deploy

docker stats

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
5888f043474b	worker-prod	0.00%	297.7MiB / 7.748GiB	3.75%	36kB / 23.9kB	1.03MB / 0B	9
8af98d8d7413	task-queue-prod	0.28%	106.5MiB / 7.748GiB	1.34%	35.4kB / 22.6kB	3.14MB / 713kB	37
12895b0e6a4f	api-prod	0.03%	279.3MiB / 7.748GiB	3.52%	21.5kB / 9.95kB	365kB / 0B	42
a3649972dfb8	frontend-prod	0.00%	9.52MiB / 7.748GiB	0.12%	11.1kB / 0B	1.22MB / 0B	9
40d30525e249	nosql-db-prod	1.23%	158.9MiB / 7.748GiB	2.00%	13.4kB / 0B	4.22MB / 246kB	32
3d74bf5f1d9	object-store-prod	0.00%	44.48MiB / 7.748GiB	0.56%	26.1kB / 11.9kB	3.93MB / 0B	11

stats

Open app in browser

http://localhost

Production Deployment (manual)

Build the API / Worker and Frontend images locally for remote deployment

```
docker build -t rabwent11/lcapp2:api-v1 -f ./api/Dockerfile.prod ./api
```

```
docker build -t rabwent11/lcapp2:frontend-v1 -f ./frontend/Dockerfile.prod ./frontend
```

Login to Dockerhub

```
docker login -u rabwent11 -p <password_or_token>
```

```
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$ docker login -u rabwent11 -p e4bc27c7-4bbb-4beb-983a-14087d76abad
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$
```

docker login

Push the images to Dockerhub

```
docker push rabwent11/lcapp2:api-v1
```

```
docker push rabwent11/lcapp2:frontend-v1
```

```
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$ docker push rabwent11/lcapp2:api-v1
The push refers to repository [docker.io/rabwent11/lcapp2]
eaece621f1b0: Pushed
cd2b679981fe: Pushed
cc1cdedd76de: Pushed
638816a1721d: Pushed
50e90dc0dfaf: Pushed
490c2f015c3e: Layer already exists
88fb2db345cd: Layer already exists
747aa001f428: Layer already exists
f9ef7f1bcb19: Layer already exists
02c055ef67f5: Layer already exists
api-v1: digest: sha256:842e1965faf90298b36b8e8901d46594c463dddf818e27760c8a1ce0878648a1 size: 2417
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$
```

push

Copy docker-compose-deploy.yml to remote server

- e.g. using scp

```
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$ scp -i ../cohort_1.pem docker-compose-deploy.yml ec2-user@lowcarbon.tk:~/
docker-compose-deploy.yml
(base) robert@ubuntu:~/si/orxagrid_lc_app_2$
```

scp

Open an ssh session to remote server

- see (<https://www.ssh.com/academy/ssh/command>) for instructions
- following steps are carried out on remote server over ssh

Edit the whitelabel configuration variables in docker-compose-deploy.yml

```
nano docker-compose-deploy.yml
```

- GENERIC_API_URL_FROM_ENV - this should be the URL or IP of remote server, port 5000
- COMPANY_LOGO_URL - url for the logo of the company purchasing / hosting the app
- COMPANY_WEBSITE_URL - url for the website of the company purchasing / hosting the app

```
GNU nano 2.9.8 docker-compose-deploy.yml
Version: '3'

services:

  frontend:
    container_name:
      frontend-prod
    image: rabwent11/lcapp2:frontend-v1
    environment:
      - "GENERIC_API_URL_FROM_ENV=http://127.0.0.1:5000/"
      - "COMPANY_LOGO_URL=https://www.orxagrid.com/images/logo-03-02-2-250x72.png"
      - "COMPANY_WEBSITE_URL=https://www.orxagrid.com"
    restart: always
    ports:
      - '80:80'

  api:
    container_name:
      api-prod
    image: rabwent11/lcapp2:api-v1
    restart: always
```

nano

Save the file and close

<CTRL-O>

<CTRL-X>

Set up Docker

- Install Docker (<https://docs.docker.com/get-docker/>)
- Install Docker Compose (<https://docs.docker.com/compose/install/>)
- Login to Dockerhub
`docker login -u rabwent11 -p`

Pull required images

`docker-compose -f docker-compose-deploy.yml pull`

Start app

`docker-compose -f docker-compose-deploy.yml up -d`

Configure firewall

- Ensure ports 80 and 5000 are open

Test app

- Open remote server url in a browser on local machine

Production Deployment (automatic)

AWS Console

- Sign up / sign in to (<http://console.aws.amazon.com>)
- Select All Services >> Lightsail
- Select Networking >> Create Static IP >> Create

Static IP addresses can only be attached to instances in the same region.



Identify your static IP

Your Lightsail resources must have unique names.

Static IP addresses are free only while attached to an instance.
You can manage five at no additional cost.

Create

ip

- Note the Static IP address



lcapp_demo

Static IP, Not attached
London, all zones (eu-west-2)

Static IP: **18.169.92.18**

[Details](#) [Delete](#)

 **This static IP is not attached to an instance.**
You will be charged until you reattach this static IP to an instance.
[How much do static IPs cost?](#)

Public static IP address

This static IP is available for public connection worldwide.

18.169.92.18

Attach to an instance

Attaching a static IP replaces that instance's dynamic IP address.

Static IP addresses can only be attached to instances in the same region.

ip addr

- Navigate back to Lightsail Dashboard
- Create Instance >> Linux (OS Only) >> Ubuntu 20
- Select a machine with at least 2 CPUs and 4GB RAM

- Click 'Add Launch Script'
- Paste in contents of (install.sh) from root of this repo

Ubuntu 20.04 LTS

Ubuntu 20.04 LTS - Focal. Lean, fast and powerful, Ubuntu Server delivers services reliably, predictably and economically. It is the perfect base on which to build your instances. Ubuntu is free and will always be, and you have the option to get support and Landscape from Canonical.

Learn more about Ubuntu on the [AWS Marketplace](#).

By using this image, you agree to the provider's [End User License Agreement](#).

OPTIONAL

Launch script [?](#)

You can enter user data to configure the instance type you've chosen.

```
# EDIT THE VALUES BELOW AS APPROPRIATE

DEPLOY_KEY=<your_deploy_key_provided_by_orxagrid>
SERVER_URL=http://<your_server_url_or_ip>
COMPANY_LOGO_URL=<url_for_your_company_logo>
COMPANY_WEBSITE_URL=<url_for_your_company_website>

# DO NOT EDIT BELOW THIS LINE
```

You are using the **default** SSH key pair for connecting to your instance.

[Change SSH key pair](#)

Automatic snapshots create a backup image of your instance and attached disks on a daily schedule.

launch script

- Edit the whitelabel constants at top of launch script
- DEPLOY_KEY - Dockerhub deploy key provided by OrxaGrid
- SERVER_URL - Static IP noted above
- COMPANY_LOGO_URL - url for the logo of the company purchasing / hosting the app
- COMPANY_WEBSITE_URL - url for the website of the company purchasing / hosting the app

OPTIONAL

Launch script [?](#)

You can enter user data to configure the instance type you've chosen.

```
# EDIT THE VALUES BELOW AS APPROPRIATE

DEPLOY_KEY=4dafbc96-d463-4232-9803-4c66d84879cd
SERVER_URL=http://18.169.92.18
COMPANY_LOGO_URL=https://storage.googleapis.com/gd-wagtail-prod-
assets/original_images/evolving_google_identity_2x1.jpg
COMPANY_WEBSITE_URL=https://google.com

# DO NOT EDIT BELOW THIS LINE
```

whitelabel

- Click 'Create Instance'

Attach static IP



- Click on the instance, then select 'Networking'
- Attach the static IP created in previous steps

address is accessible only to other resources in your Lightsail account

PUBLIC IP

lcapp_demo

+ Create static IP

Cancel  Attach 

PRIVATE

17

What

attach

Open firewall ports

- Ensure port 80 is open
- Add a new rule to open port 5000

+ Add rule

Specify a port and protocol to open. Specify a port range using a dash, such as 0 - 65535.

Application

Protocol

Port or range

☐ Restrict to IP address






Custom

TCP

5000

Cancel  Create 

Duplicate rule for IPv6 ☒

Application	Protocol	Port or range / Code	Restricted to	
SSH	TCP	22	Any IPv4 address Lightsail browser SSH/RDP 	 
HTTP	TCP	80	Any IPv4 address	 

firewall

Test the app

- Wait 5 minutes for the app to come up after starting the instance with launch script
- Navigate browser to Static IP address of instance
- Web app logo and it's hyperlink should be as specified at top of launch script



deployed

Summary

This app has been converted from B2B / B2C to B2B2C with whitelabel deployment. All generated intellectual property will be shared with the SME upon project sign-off by means of access to the project GitHub repository.

Conclusions and Recommendations

Conclusions

- The SME's Low Carbon app to optimally size grid connected battery and solar PV systems across multiple buildings with EV chargers was converted from a B2C / B2B product to a B2B2C solution.
- The app can now be sold directly to renewables technology installers
- The installers can customise the branding on the app and integrate it into their company website

Recommendations

- Acceptance testing for the B2B2C app by product owner
- Verification of the readme documentation by in-house development team
- Get early customer feedback
- Seek funding to implement Solar Hindcast service and add most requested customer features
- Advertise and sell the product as a Self-Hosted or SaaS application
- Investigate application of containerisation and orchestration technologies to the SME's other products, enabling rapid customer deployments.