## Presia Agricultural API Documentation:

In addition to providing some videos with examples of the API being run, I have enclosed some written documentation of how to run the API.

The API itself is designed using AWS’s API Gateway, and uses a lambda to invoke object storage into S3. This allows the S3 bucket to remain private, while the API to securely transmit the information required.

*Authentication:*

The API does not require AWS IAM credentials to run, but does require an API key to run.

Because I designed the API/S3 bucket under my personal AWS account, and I do not want to incur massive fees, I added a relatively conservative usage plan:

The API has throttling enabled, with a ‘burst limit’ of 3 (allowing only 3 requests to run in parallel), and 500 requests per day. While I would not use these in a final production API, these are small measures enacted to ensure there is a ceiling placed on personal costs incurred due to accidental misuse.

I will include the API Key in an e-mail sent to you once I ‘submit’ this project.

URL: the invocation URL for the API is ‘<https://vw4mpw6lx8.execute-api.us-east-2.amazonaws.com/develop/>’

There is only endpoint, and invoking this specific address is a post command for the geojson file.

My justification for using /develop/ as the path name (rather than /POST or something similar) was to assume that this is a two-tiered environment API, and that the submission I am providing for the API represents a ‘develop’ environmental endpoint that is being submitted to approve for promotion to production deployment.

Note that this API operates using REST Protocol. This ensures that the API conforms to all the requirements of an HTTP POST command, while also providing additional features which improve the developer experience for the API itself.

*Architecture:*

The API is designed using an AWS environment. It uses API Gateway to parse the input of the request, then has API Gateway invoke a lambda function to validate the geojson file, either to submit or reject the file for S3 ingestion, and pass back an HTTP Status Code. This status code is passed back to API Gateway, and appears in the response.

The Lambda makes a connection to S3 using the python library boto3, and takes on a pre-configured custom IAM role in AWS to ensure that it has the security credentials to access the otherwise private S3 bucket. This ensures the bucket remains secure, even from the person submitting data to the API.

The lambda uses the geojson python library to parse data using a lambda layer. It was downloaded from pypi, and is the same version as was frozen in the requirements.txt

The unit tests performed on the application code used in AWS lambda were done through a local environment with Pytest. A video of the tests running was included in the zip sent to you.

The code used on the lambda is submitted in the repo to lambda-code.py

The python version running on Lambda and for the tests was 3.9

*Inputs:*

The API accepts two header arguments:

“x-api-key”: this is where the AWS API key I provided will go.

“Content-Type”: this will always be “application/json”.

The API accepts a json body. Once submitted through API gateway and to the lambda, the application also determines whether it meets the standards required for a geojson file. While the API itself will allow you to submit non-geojson json files, the API will only accept a json file for ingestion to S3 if it is also conforms to the standards of a geojson file.

No query string is required.

*Validation:*

The particular tests conducted to ensure the data validity are located in test\_lambda-code.py. I made a point to comment with docstrings, but they include all required tests mentioned in the instructions as well as a few others.

The tests evaluate the failure of several geojson files which should not be accepted for ingestion on account of missing fields or structure.

As mentioned in exchanges surrounding our discussions: In addition to being a valid geojson, it is generally assumed that the data conforms to a similar schema as good\_fields\_nopoints.geojson.

*Additional requirements:*

Data cannot have nulls in any property field

The Date fields ‘pdate’, ‘gsstart’, and ‘gsend’ must be present, and must be in YYYY-MM-DD format.

Lastly, within the crs property of the geojson, the geojson file must have a name. This name is used to name the file in S3. This name cannot be empty, and needs to have at least 1 alphanumeric character (as it is otherwise stripped of punctuation, and cannot be an empty string).

*Submission naming convention:*

Note that while the time that a file enters the system in S3 is recorded, files with the same name will presently over-write. Should this be an issue, I can easily refactor the API to submit files with a timestamp, allowing several files with the same name but different times to be submitted. There are a couple of other alternative ways to handle that, should it be an issue, though as no specification was made it was considered out of scope.

*Set-up & Running the API:*

Since this API is not self-hosted, and is hosted on AWS, there is minimal set-up required. You must use a valid means of querying an HTTP API. In the video, I show two means of doing so: Using python’s requests module, and using a GUI tool called HTTPie. Note that while I used HTTPie as a standalone desktop application (and it is available for Windows, MacOS and most flavours of Linux), it is also available as a browser-based webapp as well.

If you wish to run the API programmatically, you are welcome to do so through your language of choice. I have attached a small python script which serves as an example of how to run the API using the requests module.

Should you wish to use this (optional) python file. you can either install the requests library from the requirements.txt file in the repository (using ‘pip install requirements.txt’ from the command line on the file), or you can bring your own python environment.

Please note that to run this code, you will require the aws-key within the same directory as api-post-sample.py, and that you will need to save it as ‘aws-key.json’, with the following text within:

{“key”: “[API KEY GOES HERE]”}

*Responses:*

General HTTP error responses on AWS are viewable here:

<https://docs.aws.amazon.com/apigateway/latest/developerguide/supported-gateway-response-types.html>

In addition to these, 400 codes are used for general semantic errors in the data input, resulting in a failure for the data to load into S3.

500 is used for issues connecting to S3 service, result in a failure to load the data into the S3 bucket.

200 means a successful operation, and data has loaded into S3.

Additional details about the nature of the errors relating to the response codes can be viewed specifically within the code provided- and errors are named explicitly to ensure that the cause of the problem is easily diagnosable.

Finally, please note that I am a resource that can be tapped to better under this API, and the development work that went into it.