Big Data and Intelligent Analytics

# Designing a Nowcasting Intelligence System

| Name | NU ID |
| --- | --- |
| Sai Saketh Vootla | 001568984 |
| Shika Shyam | 002194543 |
| Sai Raghavendra Viravalli | 001568587 |

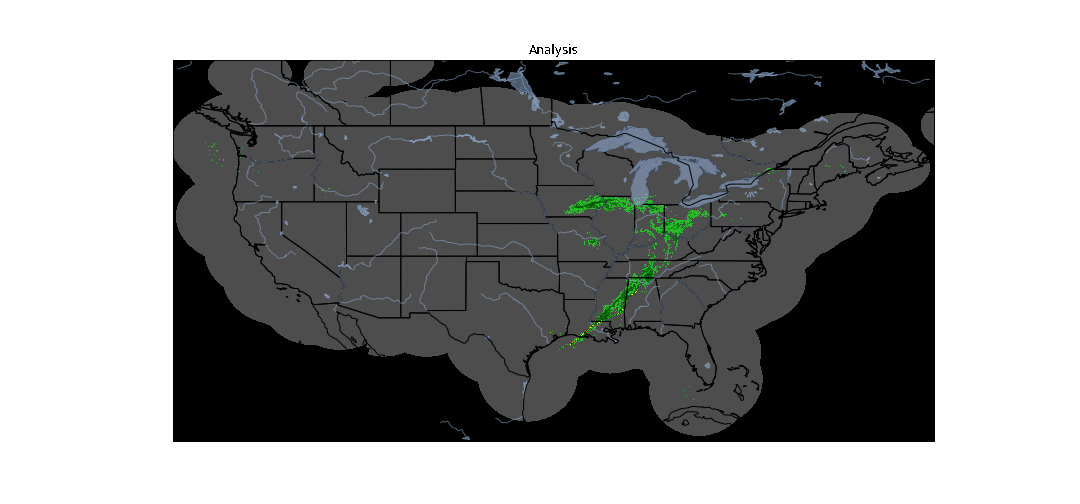
[GITHUB](https://github.com/shikashyam/BigDataSystemsCoursework/tree/main/Assignment2)

# Introduction

This aim of this document is to elaborate in detail about accessing two of the existing models built by the MIT Lincoln lab called Nowcasting and Synrad use cases and assess it’s usability and viability in the domain of NOAA - National Oceanic and Atmospheric Administration.

We aim to provide a Minimum Viable Product to serve two main functionalities of Nowcasting and Backtesting for NOAA which can then be used by NOAA's National Weather Service which is building a Weather-Ready Nation by providing better information for better decisions to save lives and livelihoods. Additionally, NOAA aims to provide data, tools, and information to help people understand and prepare for climate variability and change and our marketing our model to NOAA facilitates that.

# Dataset



The SEVIR Dataset is a Storm Event Imagery Dataset for Deep Learning Applications in Radar and Satellite Meteorology.

The SEVIR dataset consists of

– 107 files in HDF5 format containing imagery

• Files separated by image type and date range for easier access

• File sizes ranging between 1GB and 20 GB

– A catalogue containing metadata of each image, including

• Unique ID assigned to each event

• Times of each image

• Lat / Lon bounds of slice

• Map projection and image extent in projection coordinates for exact georeferencing

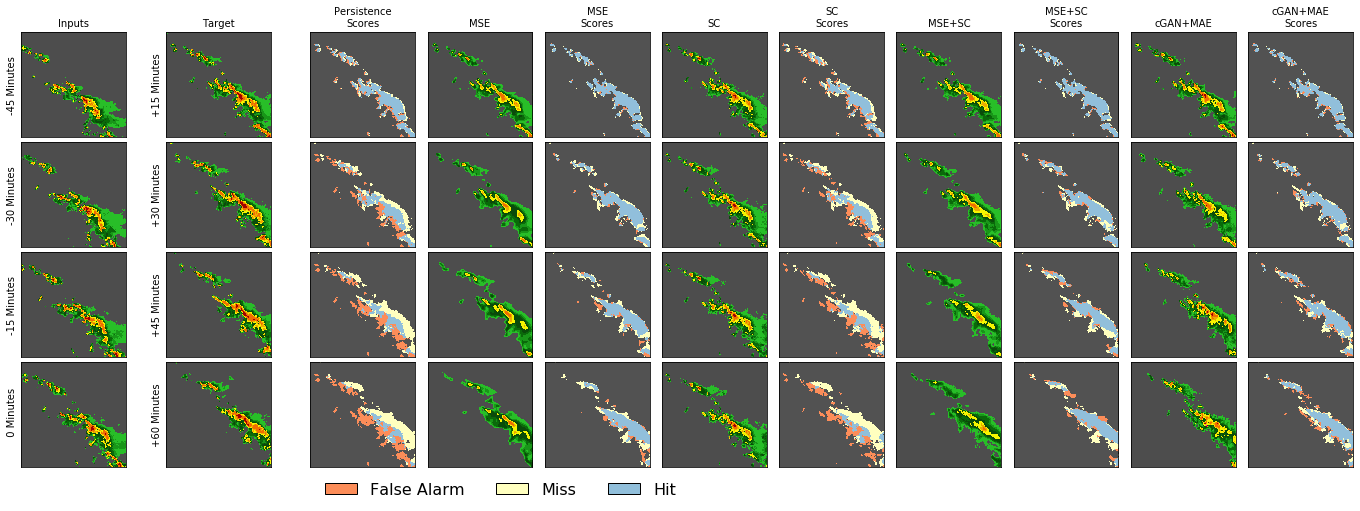
• NCEI Storm Event ID and Episode ID (for non-random cases) SEVIR File Details

Total Size of SEVIR Dataset: ~900 GB

There is bulk data which contains between January 1950 to October 2021 at the NOAA’s National Weather Service(NWS). There are available in CSV format. Using this database we download the appropriate 2019 file to perform further analysis.

# Data Generation for Nowcasting

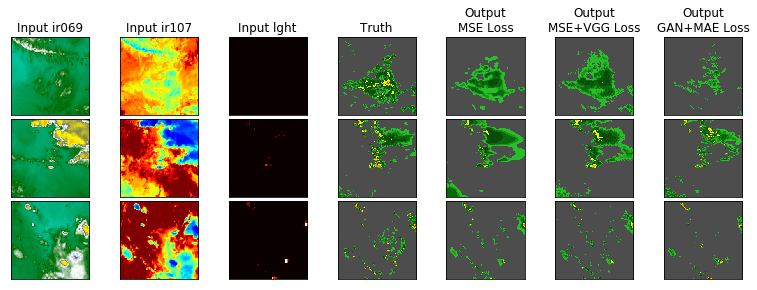
## Nowcast (Radar Nowcasting)



Nowcasts are high resolution, short-term (e.g. up to 2 hours) weather forecasts of radar echos, precipitation, cloud coverage or other meteorological quantities widely used in public safety, air traffic control, and many other areas that require high fidelity and rapidly updating forecasts. Previous work on deep learning for Nowcasting includes convolutional Long Short Term Memory (ConvLSTM) models, recurrent architectures and fully convolutional networks for precipitation nowcasting.

# 

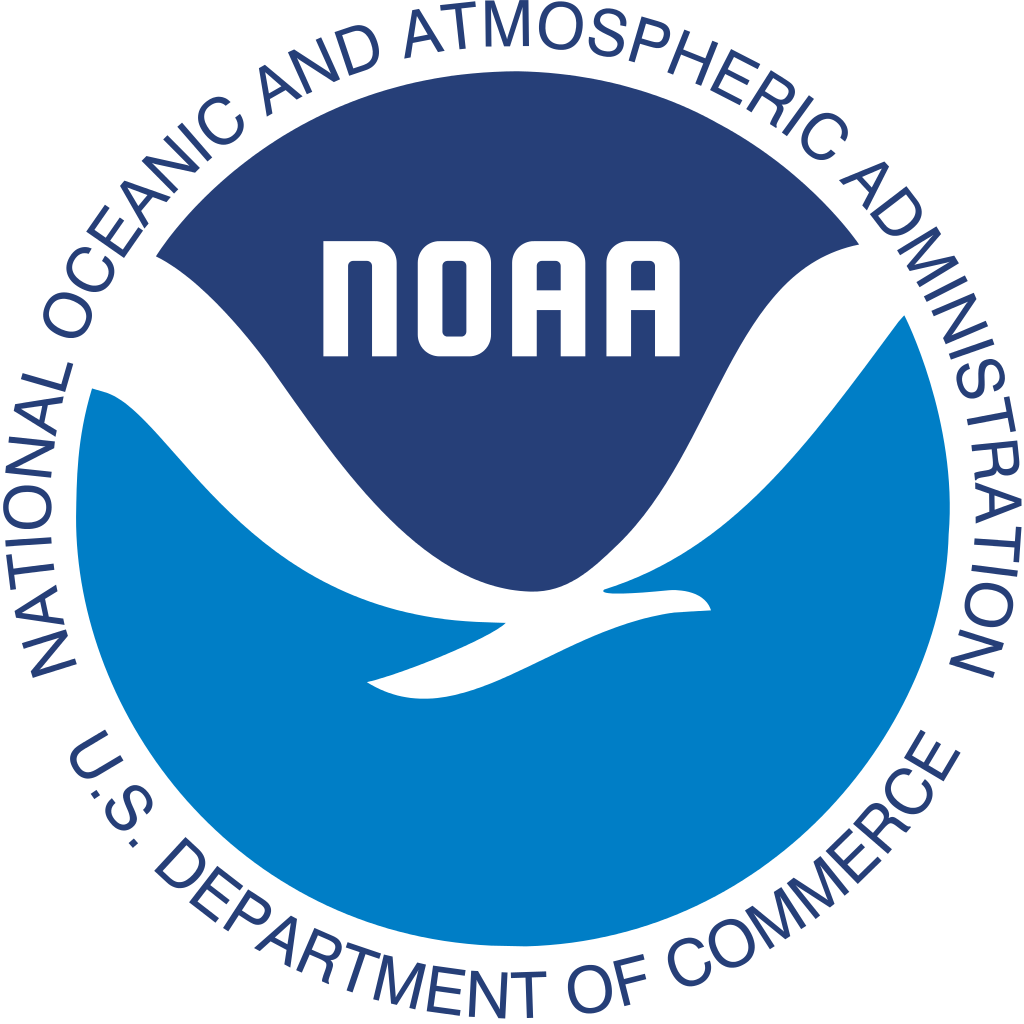
## SYNRAD(Synthetic Weather Radar)



Depictions of storms obtained from weather radar are extremely important in many areas; however, most areas of the world do not have access to ground based radar. Using SEVIR, we will train a model that creates radar-like imagery of storm depictions using only satellite and lightning as inputs. To do so, we take the set of image types Sinput = {ir069,ir107,lght} as inputs to the model

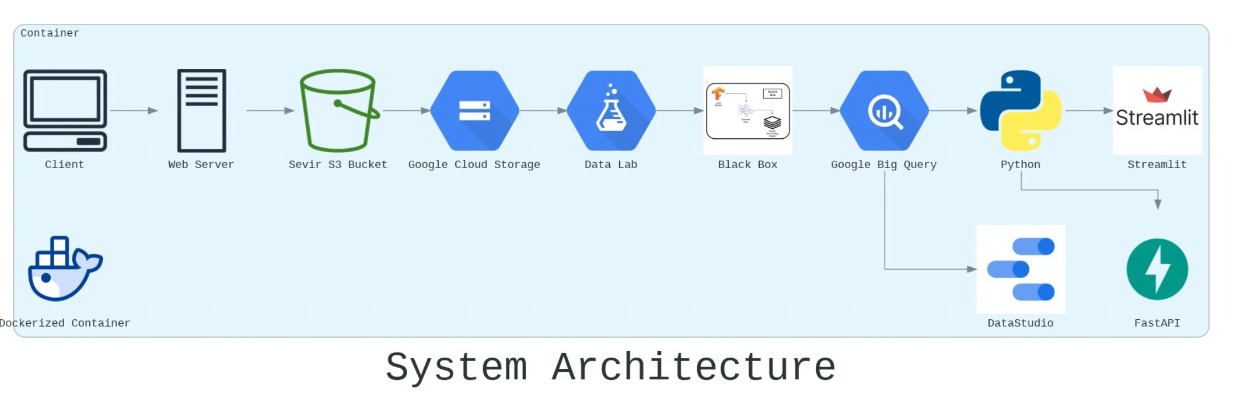
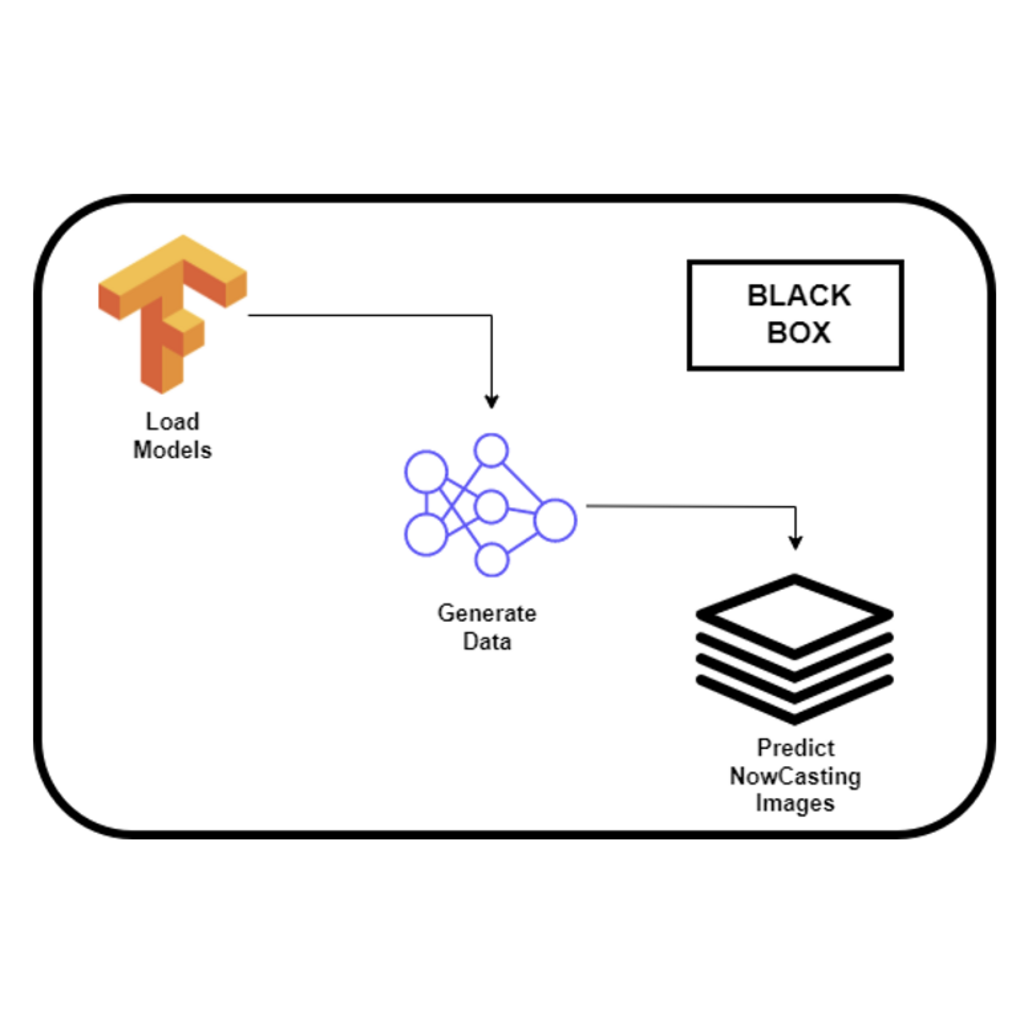
Link to the [Jupyter notebook](https://github.com/shikashyam/BigDataSystemsCoursework/blob/main/Assignment2/notebooks/Analyzing_SEVIR_UseCases.ipynb)

# Applications in NOAA domain



* NOAA lets the citizens use the information needed to check the weather.
* NOAA are the backbone for emergency planning during natural disasters and It will help them in better planning to avert them
* NOAA is the backbone behind the economic vitality

# Data Pipeline Architecture



# 

# 

# Implementing Nowcasting and Backtesting

While marketing this app to NOAA we aim to provide a MVP (Minimum Viable Product) prototype which consists of two major functionalities as below:

## Nowcasting -

As detailed earlier nowcasting is a common term used in weather prediction using satellite images where a current situation of a storm is used to predict the movement of the storm and its subsequent impact for the next window of prediction. In our case this is one hour of predicted images into the future.

NOAA can access these predicted images and assess the impact to the area dwellers and accordingly plan for Adverse weather advisories or Evacuation plans.

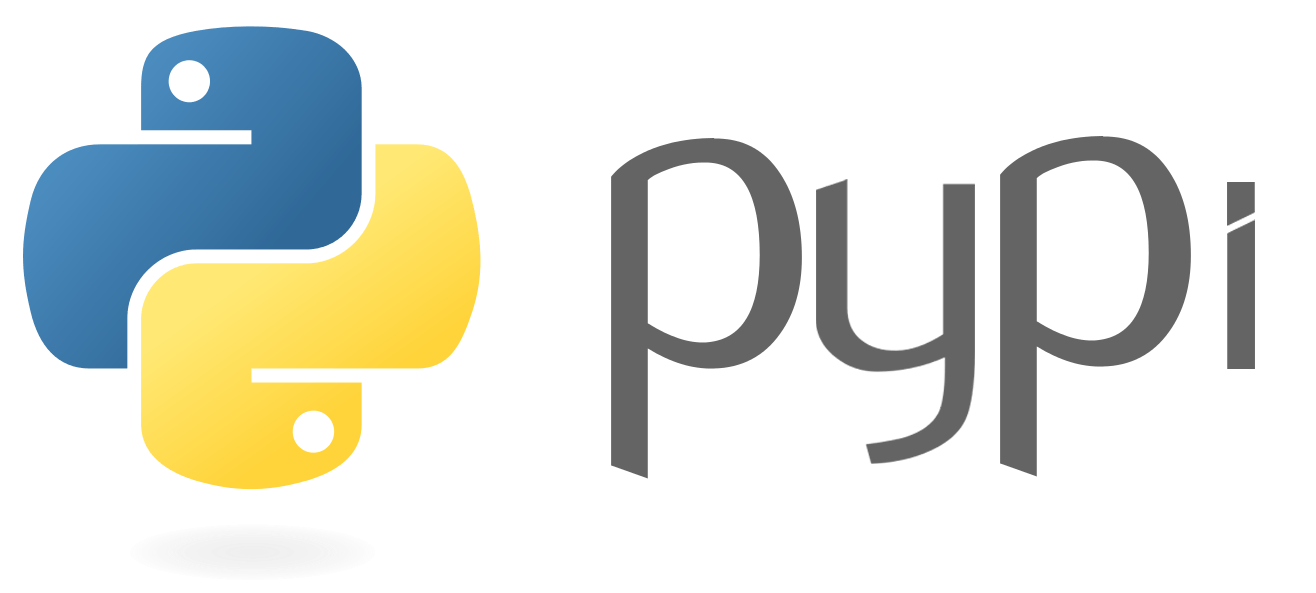
Additionally, since we are providing this MVP to NOAA, they will have the option to use the nowcasting functionality in two ways.

The first is to upload a direct .csv file that has the list of .h5 files that contain the storm or random event data that they want nowcasted. The second option is a much more intuitive one, especially for users in NOAA without direct access to yearly storm data files. These users can enter a Date, Time and Location, and the list of available events will be displayed alongside the nowcasting for those events.

## Backtesting -

The idea of marketing a machine learning product to a user comes with its fair set of challenges. End users do not understand Machine learning metrics such as MAE or GAN loss and because of this we need to have a way to show users how correct or accurate our prediction is. We will be making use of backtesting for this. Since our users are looking at historical data, we will be using the inputs they give to forecast the images for the next hour but also simultaneously display for the users what the ground truth was for that forecasted hour, if there is data available for that hour.

# PyPi



* PyPI is the default software repository for Python developers to store created Python programming language software developers and programmers alike use to publicize and share our software.
* Packages are a way of structuring many packages and modules which helps in a well-organized hierarchy of data set, making the directories and modules easy to access.
* Just like there are different drives and folders in an OS to help us store files, similarly packages help us in storing other sub-packages and modules, so that it can be used by the user when necessary.

# Using PyPi for SEVIR Model

* The first tool you need is pip, which will be used to install other tools required for configuring, building, and publishing Python projects. To install and/or upgrade pip on your computer, run:



* [setuptools](https://setuptools.pypa.io/en/latest/) is the modern and official tool for packaging Python projects. It is a collection of enhancements to the Python [distutils](https://docs.python.org/3/library/distutils.html) package that allows developers to more easily build and distribute Python packages.

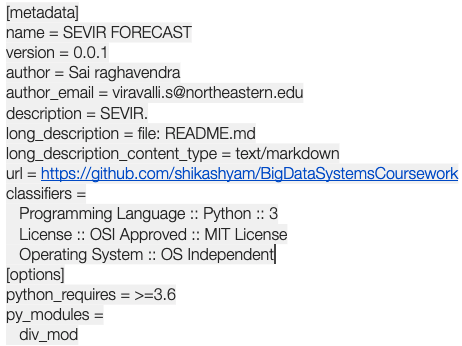


* To configure our project for publishing, we need to create a special file called pyproject.toml, which is the new unified Python project settings file that replaces the traditional setup.py used in many old projects.In pyproject.toml, we need to specify which build tools are required to build your project. In most common cases, pyproject.toml will have the following content:



* We specify that setuptools and wheel will be used to build our project. setuptools will be used to build the source distribution, namely the tar file as we will see soon, andwheel will be used to build a so-called “[wheel](https://pythonwheels.com/)”, which is a pre-built package that can be installed without the requirement to go through the “build” process.
* We should start to use the new style setup.cfg which is more concise and more convenient to read and use. setup.cfg and setup.py have equivalent settings can be very easily translated from one to another.

For this purpose, setup.cfg can be:



* In the metadata category, we need to specify some metadata for the project. The most important metadata is the name of the project. It must be unique in the whole PyPI package index.
* we need to install the build tool:

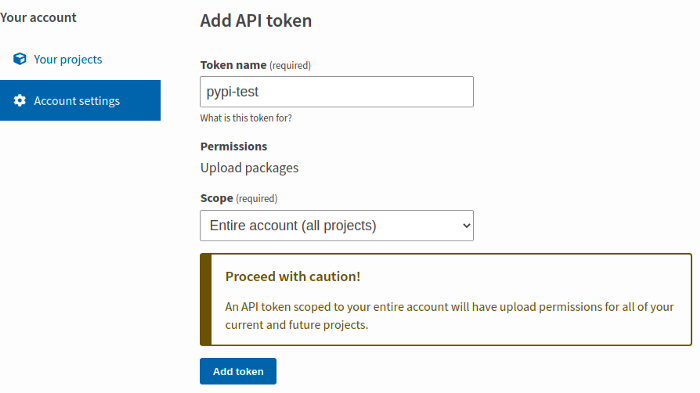




* Make sure you are in the directory where pyproject.toml is located, and then run the build command:



* After the build procedure is successful it displays a message and then shows how the directory is designed.
* The next step would be to go on TestPyPi and create a new project to add it and then assign a necessary API Token.



* Then we need to install twine to upload the archives to PyPI or TestPyPI:



* After twine is installed, we can upload the archives to PyPI or TestPyPI:

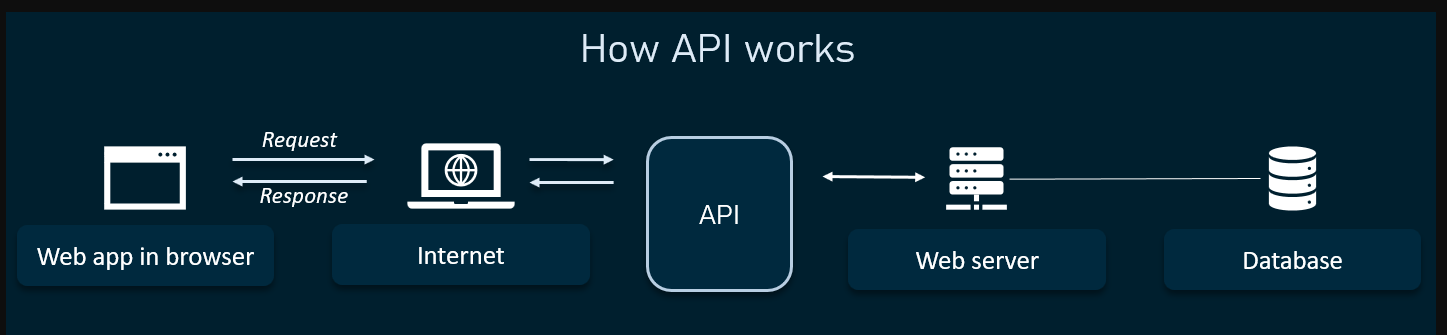
python3 -m twine upload --repository SEVIR dist/\*

* You can see a link like [https://test.pypi.org/project/div-mod-1234/0.0.1/](https://test.pypi.org/project/div-mod-123/0.0.1/) in the console when the upload process is finished. Open this link and you will see the project you just published
* After the project is published to PyPI or TestPyPI, you can install it and verify that it works. For the purpose of package dependency, it’s recommended to create a virtual environment to install and test packages.

# API

**What is an API:**

An API is also known as Application Programming Interface is a set of programming code that enables data transmission between one software product and another. It also contains the terms of this data exchange.

****

APIs are made up of two related elements.

The first is a specification that describes how information is exchanged between programs, done in the form of a request for processing and a return of the necessary data.

The second is a software interface written to that specification and published in some way for use.

The software that wants to access the features and capabilities of the API is said to "call" it, and the software that creates the API is said to "publish" it.

[APIs authorize](https://www.techtarget.com/searchunifiedcommunications/answer/How-do-APIs-work-and-how-can-you-ensure-they-are-secure) and grant access to data that is requested by users and other applications. Access is authenticated to a service or portion of functionality, against predefined roles that govern who or what service can access specific actions or data. APIs also provide an audit trail that details system access: who or what, and when.

Applications that call APIs were traditionally written in specific programming languages. Web APIs can be called through any programming language, but can also be accessed by webpages created in HTML or application generator tools.

The most common architectures for APIs are representational state transfer ([REST](https://www.techtarget.com/searchapparchitecture/definition/REST-REpresentational-State-Transfer)) and Simple Object Access Protocol ([SOAP](https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol)), which defines a standard communication protocol specification for XML-based message exchange. SOAP requires less low-level infrastructure-related code than does REST, but REST APIs are easier to scale and redeploy, simpler to implement and integrate with websites and services. The current industry trend is largely to use REST APIs, particularly for web interactions.

**ENDPOINTS:**

Endpoints specify where resources are located and how they can be accessed by third-party software. Usually, they’re accessed via a URI to which HTTP requests are sent and from which the response is expected.

What is a RESTful API? It is an application program interface that uses HTTP requests to GET, PUT, POST, and DELETE data. These are the main methods that indicate what type of operation is to be performed.

GET - Retrieve resources from the server using giving URI

PUT - Send resources to the server

POST - Update existing information with uploaded content

DELETE - Remove all existing resources



**HTTP Status Codes:**

HTTP status codes, sometimes called internet or browser error codes, are standard response codes sent by a web server that identifies whether an operation performed was successful or why pages or resources are not loading properly.

There are 4 categories of responses.

1. 2XX (**Success**)
2. 3XX (**Redirection**)
3. 4XX (**Client Error**)
4. 5XX (**Server Error**)

# FastAPI



**What is FastAPI?**

FastAPI is a modern, fast (high-performance), a web framework for building APIs with Python 3.6+ based on standard Python type hints.

**Requirements**

pip3 install fastapi

pip3 install uvicorn

pip3 install iexfinance

**Getting Started**

Basic usage is available in main.py. Run the server with uvicorn main:app --reload. Go to http://127.0.0.1:8000/docs and you should see the interactive API documentation. This is a simple example that demonstrates the created API that can:

* Receives HTTP requests in the paths / and /items/{item\_id}.
* Both paths take GET operations (also known as HTTP methods).
* The path /items/{item\_id} has a path parameter item\_id that should be an int.
* The path /items/{item\_id} has an optional str query parameter q
* By Clicking on the "Execute" button on the API interface, the user interface will communicate with your API, send the parameters, get the results and show them on the screen

## **FastAPI features**

**FastAPI** gives you the following:

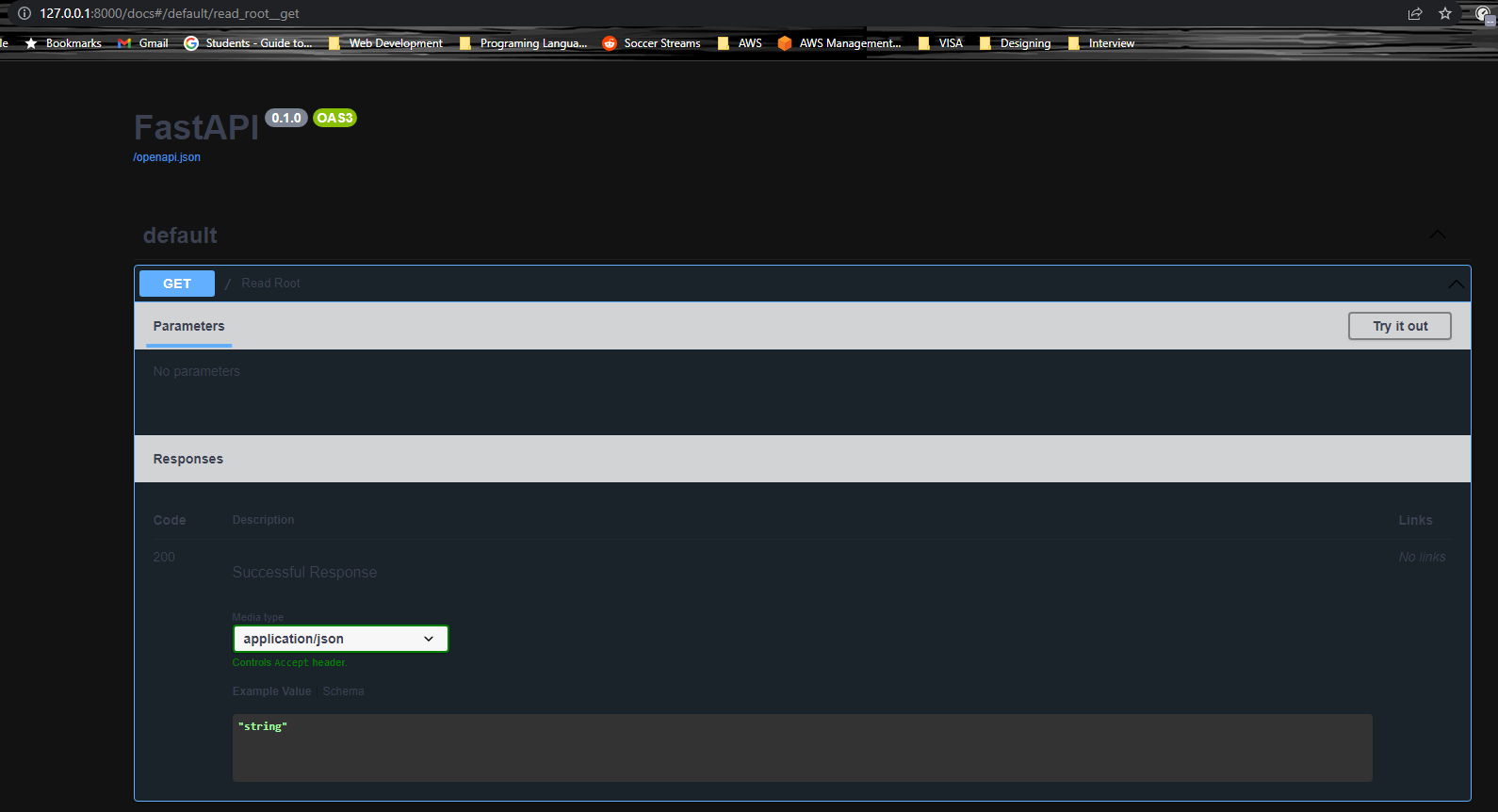
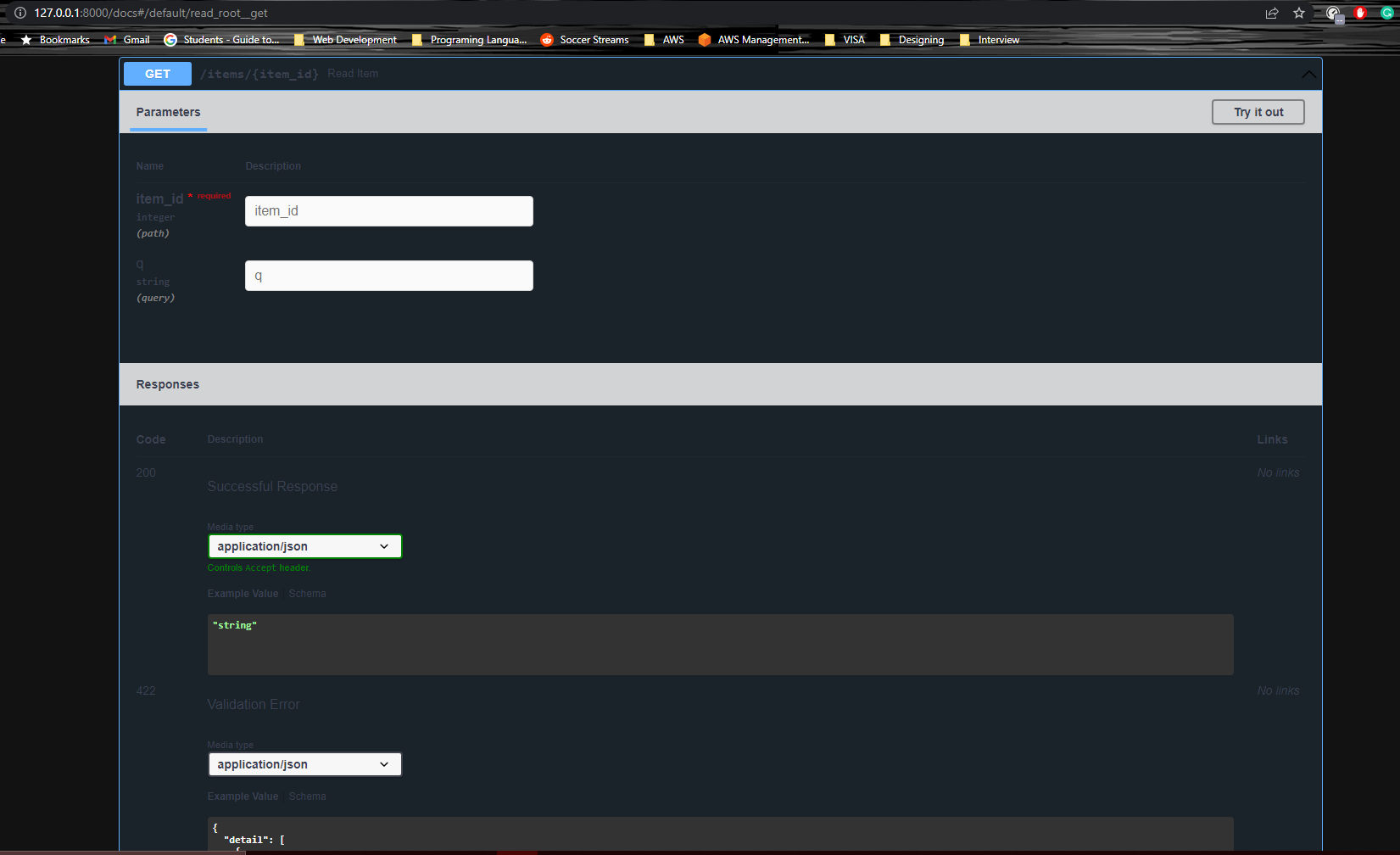
### Based on open standards

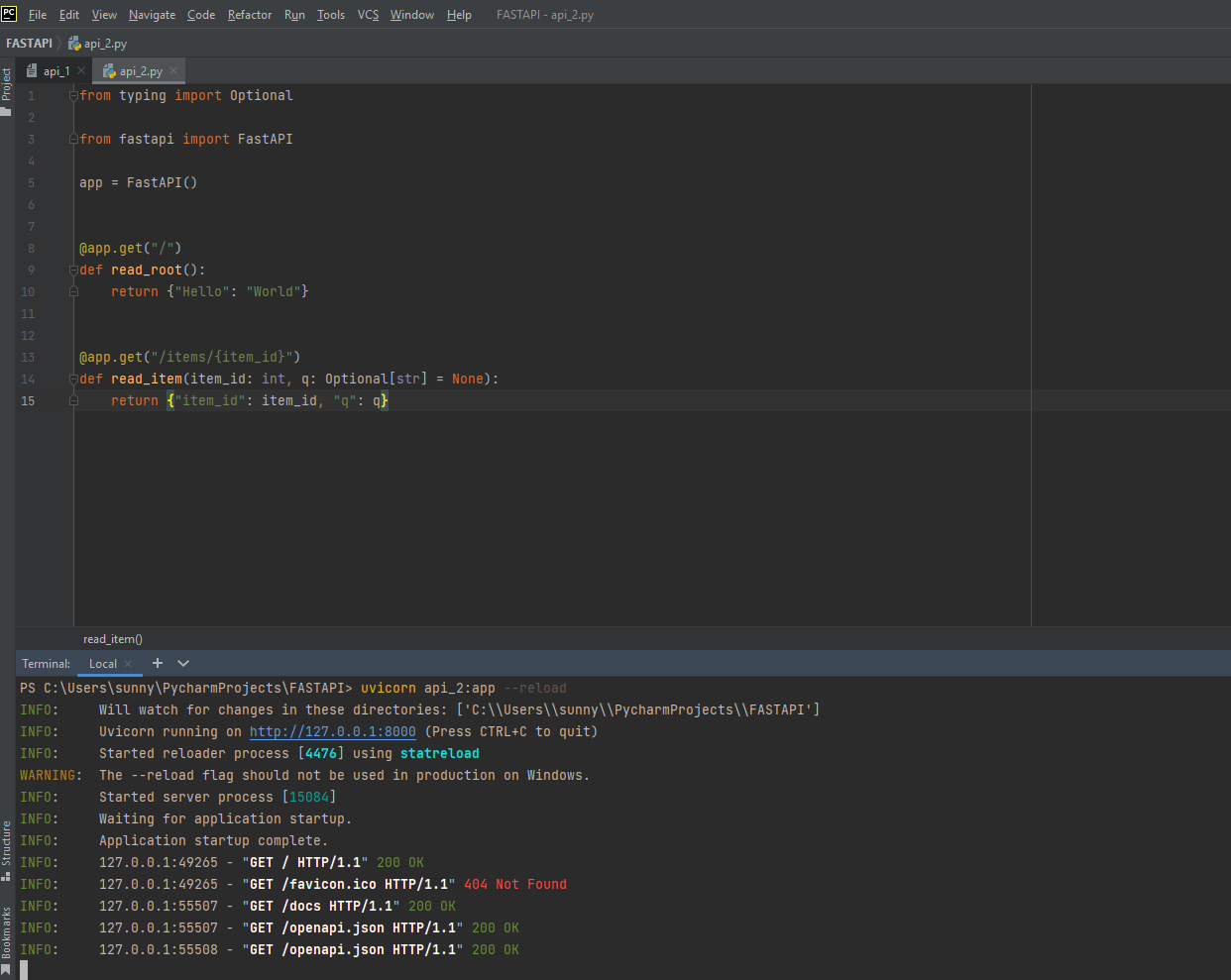
* [**OpenAPI**](https://github.com/OAI/OpenAPI-Specification) for API creation, including declarations of path operations, parameters, body requests, security, etc.
* Automatic data model documentation with [**JSON Schema**](https://json-schema.org/) (as OpenAPI itself is based on JSON Schema).
* Designed around these standards, after a meticulous study. Instead of an afterthought layer on top.
* This also allows using automatic **client code generation** in many languages.

### Automatic docs

### Interactive API documentation and exploration web user interfaces. As the framework is based on OpenAPI, there are multiple options, 2 included by default.

* [**Swagger UI**](https://github.com/swagger-api/swagger-ui), with interactive exploration, call and test your API directly from the browser.





## **Starlette features**

**FastAPI** is fully compatible with (and based on) [**Starlette**](https://www.starlette.io/). So, any additional Starlette code you have will also work.

FastAPI is actually a sub-class of Starlette. So, if you already know or use Starlette, most of the functionality will work the same way.

With **FastAPI** you get all of **Starlette**'s features (as FastAPI is just Starlette on steroids):

* Seriously impressive performance. It is [one of the fastest Python frameworks available, on par with **NodeJS** and **Go**](https://github.com/encode/starlette#performance).
* **WebSocket** support.
* In-process background tasks.
* Startup and shutdown events.
* Test client built on requests.
* **CORS**, GZip, Static Files, Streaming responses.
* **Session and Cookie** support.
* 100% test coverage.
* 100% type annotated codebase.

**Pydantic features**

**FastAPI** is fully compatible with (and based on) [**Pydantic**](https://pydantic-docs.helpmanual.io/). So, any additional Pydantic code you have will also work.

Including external libraries also based on Pydantic, as ORMs, ODMs for databases.

This also means that in many cases you can pass the same object you get from a request **directly to the database**, as everything is validated automatically.

The same applies the other way around, in many cases, you can just pass the object you get from the database **directly to the client**.

With **FastAPI** you get all of **Pydantic**'s features (as FastAPI is based on Pydantic for all the data handling):

* + No new schema definition micro-language to learn.
  + If you know Python types you know how to use Pydantic.
* Plays nicely with your **IDE/linter/brain**:
  + Because pydantic data structures are just instances of classes you define; auto-completion, linting, mypy and your intuition should all work properly with your validated data.
* **Fast**:
  + in benchmarks Pydantic is faster than all other tested libraries.
* Validate **complex structures**:
  + Use of hierarchical Pydantic models, Python typing’s List, and Dict, etc.
  + And validators allow complex data schemas to be clearly and easily defined, checked, and documented as JSON Schema.
  + You can have deeply **nested JSON** objects and have them all validated and annotated.
* **Extendible**:
  + Pydantic allows custom data types to be defined or you can extend validation with methods on a model decorated with the validator decorator.
* 100% test coverage.

**Advertising the API**

Advertising your API is about educating your consumers on its benefits, and advocating for why it can be an integral part of the developer’s toolchain. At the end of the day, you want to make sure your APIs are being discovered and consumed, both internally and externally. In general, marketing is a combination of inbound content, including (includes inbound messaging and documentation) and outbound awareness of the API.

Some ways to Advertise your API would include:

1. Get Listed in API Directory

API directories list and organize APIs, helping developers find new APIs to integrate into their products.

1. Join API marketplaces

API marketplaces take the static API database concept to another level — similar to Unified APIs, they provide a one-stop-registry for integrating with many APIs

1. Distributing SDK through package

Another option to get your API into more projects is to make your APIs available through package managers

1. Partner with Unified API

Unified APIs combine many APIs of the same software type into a single API, creating a more efficient model to build seamless integrations.

**SEVIR**

We can use the API calls to connect it to the model where the user is asked to give input in regard to date/time and location which in turn will generate the output images and that can be viewed in the web application or on cloud storage and a link can be generated to redirect to it.

# What is Streamlit?



Streamlit is a free open-source python-based framework to build interactive dashboards and machine learning or data science web apps Streamlit is a sought-after tool in all modern Data Science solutions because of simplicity, flexibility, robustness and free of cost.

We can instantly develop web apps and deploy them easily using Streamlit with no front-end development experience required. Streamlit allows you to write an app the same way you write a python code. Streamlit makes it seamless to work on the interactive loop of coding and viewing results in the web app.

The documentations accompanying Streamlit is detailed and comprehensive and can guide any beginner to create and deploy web apps within minutes. There are also various ready to use templates and examples available on the Streamlit website at<https://streamlit.io/gallery>

For our requirements we will be making use of Streamlit to provide an interactive UI to the end user and the option to provide inputs in the form of various features that our model uses and then view the results in real time on the web app. The user can also tweak the inputs and view how the model behavior changes.

Advantages of Streamlit

· Streamlit is an open-source tool. Unlike many of the alternative visualization apps such as Tableau which require a dedicated paid membership to create and view visualizations, Streamlit comes with a free of cost and scalable alternative for visualization heavy use cases like ours.

· Streamlit requires little to no preset installations. It can be installed in the Python environment using a pip install Streamlit and can be included in our existing python script with a simple import statement.

· Streamlit does not require front-end development experience and the entire Web Application can be coded in pure python. This comes out to be very useful in small scale projects that span only Data Science skillsets.

· Streamlit is an ever-growing tool with new functionalities added every day. Their most recent addition to the tool is the webcam input feature which can take in inputs from the webcam of our computer as an input to the underlying ML model. This is very useful in CV applications.

· Streamlit community has a lot of example projects which are open-source and can be reused and additionally the community is very active on all social media platforms and ready to help fellow developers.

Starting with Streamlit

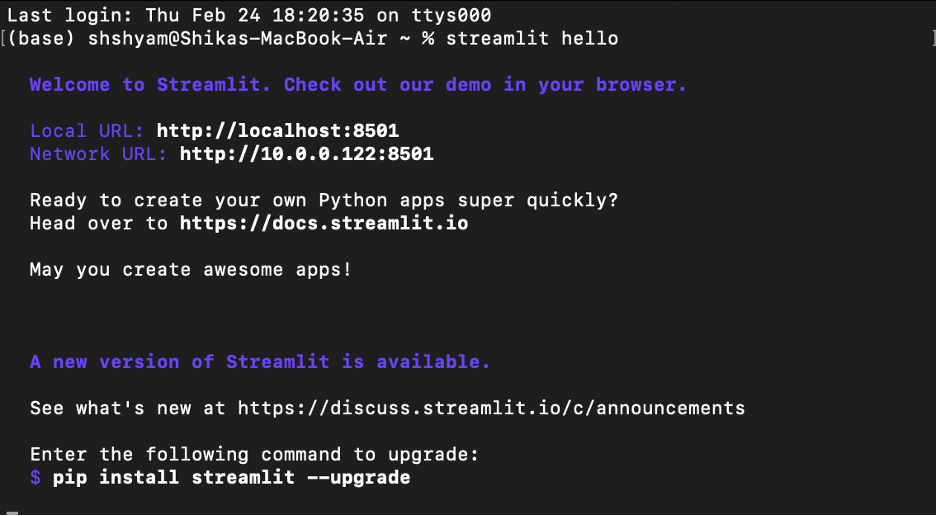
The first step to working with Streamlit is to install it in your python environment. This can be done using a pip install or conda install as shown below:



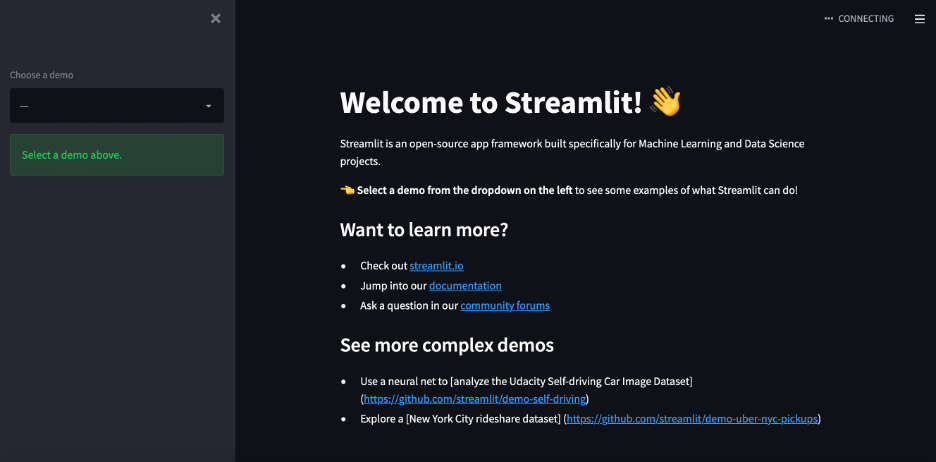
For in-depth understanding of the Streamlit library, check out the Streamlit GitHub repo:

<https://github.com/streamlit/streamlit>

We can check if Streamlit install is successful by running the below command and receiving the output as shown:



It also opens up a sample webpage on your localhost browser as shown below:



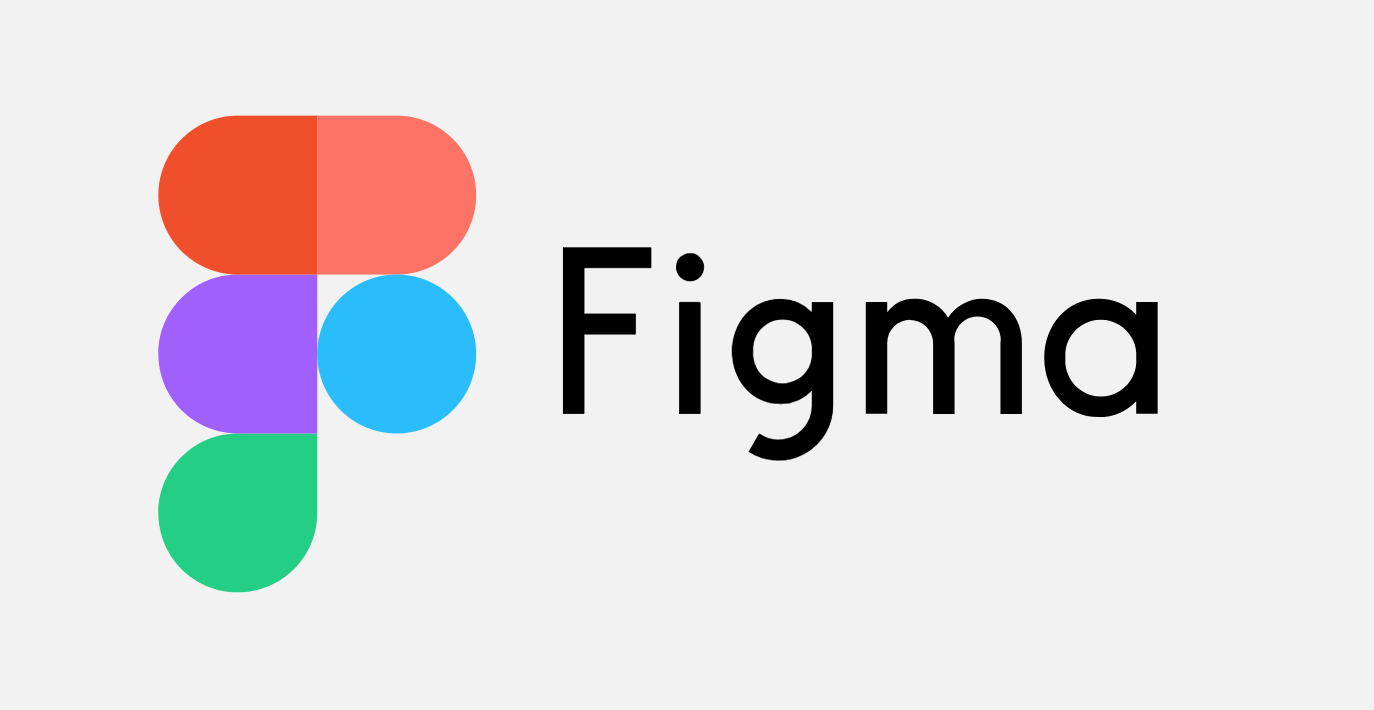
Streamlit can now be used within our python code by bringing it in via an import statement like below:



Finally, once the code is completed as detailed in the next section, the Streamlit app can be run by using the below command:



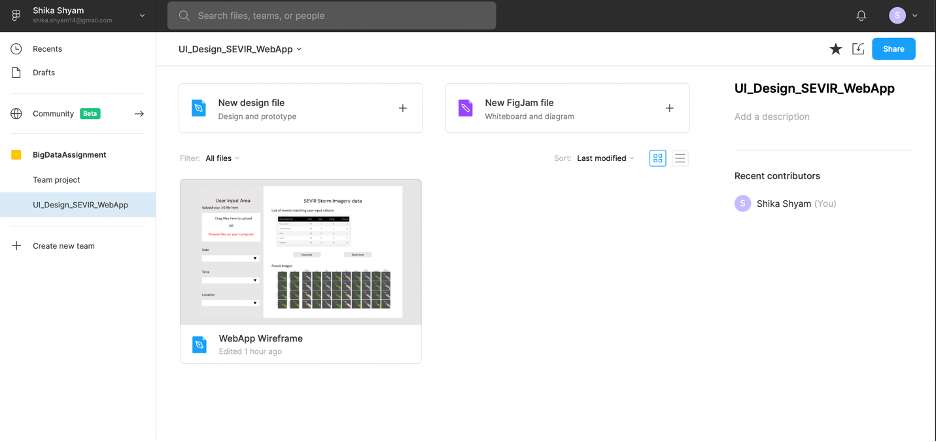
# Working with Figma



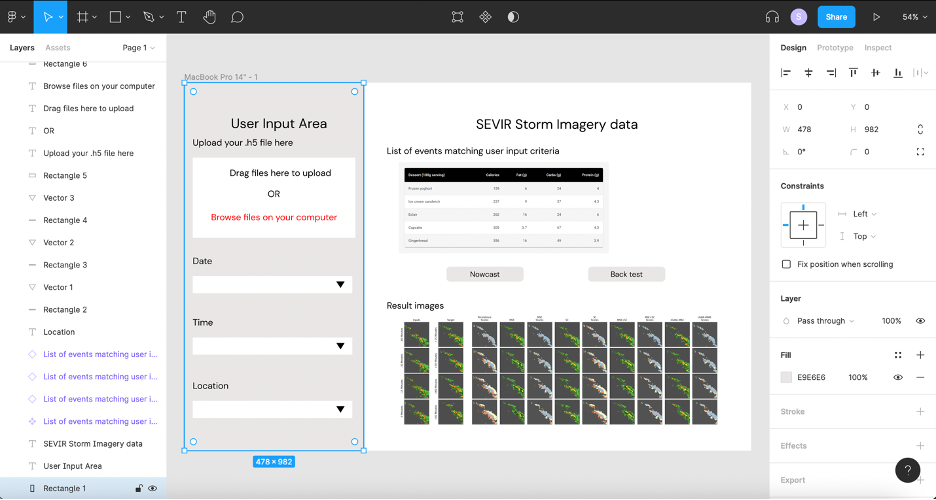
Figma is a free, online UI tool to create, collaborate, prototype wireframes and user interfaces. It is a very useful tool in terms of collaboration and enables users to collaborate and work together seamlessly on the online platform. Figma although priced, has a Education pricing section which makes it free for students to use.

For starting with Figma, we have made an account, and accessed Education pricing by verifying University email of the account holder. Secondly, we add collaborators on to the project in order to be able to design effectively.

Below is a view of the Figma home page with the project name and the names of the collaborators listed:

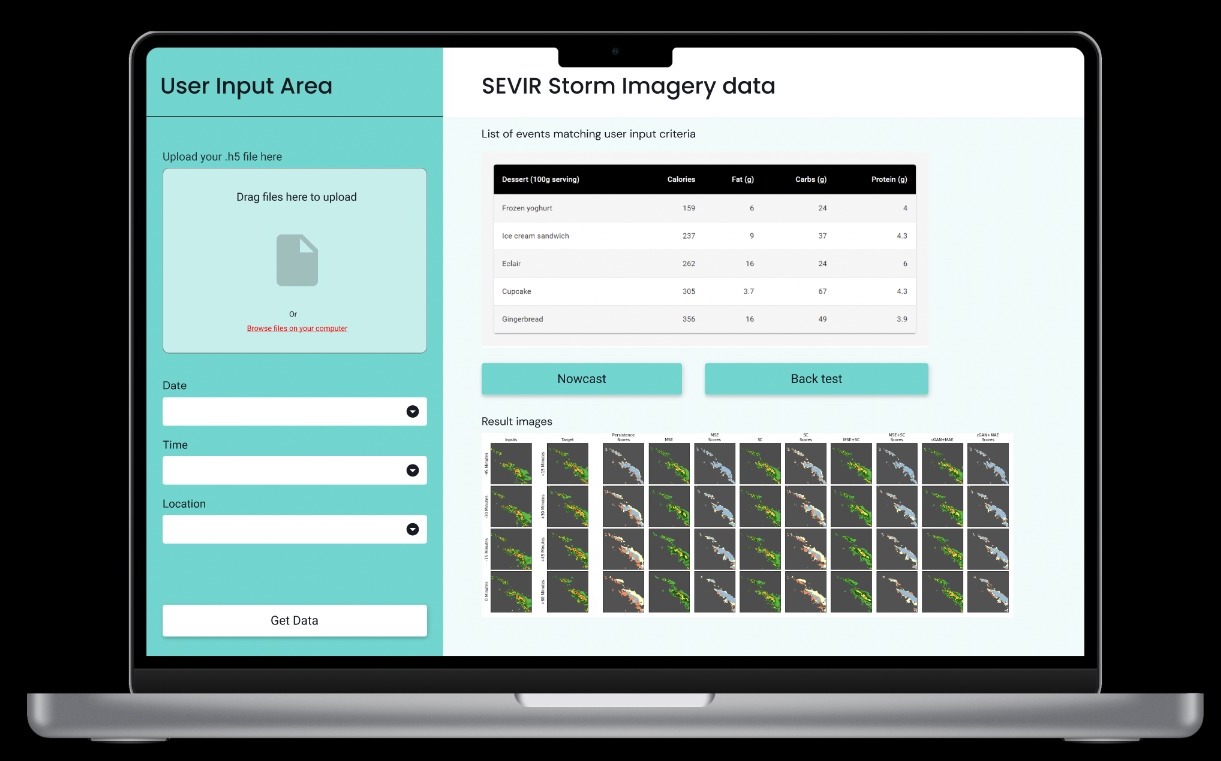


Below is the intuitive workspace which can be used to create the wireframe. The top panel specifies the various kinds of elements that can be added to the wireframe starting with the frames. Additionally various elements can be added on top of the frame in layers and these layers can been visualized on the left-hand side panel. The right-side panel specifies the properties and design elements of the object that is selected and can be tweaked as per the vision of the design.



Below is a screenshot of the final wireframe. It can also be viewed at

<https://www.figma.com/proto/MFT2nS3H0pGPIeSXrgZQGz/WebApp-Wireframe?node-id=5%3A3&scaling=scale-down&page-id=0%3A1>

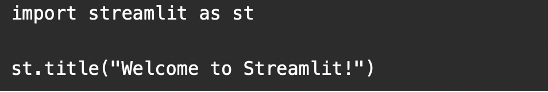


# Creating the Streamlit Web Application

For our use case as described in the previous sections, the Streamlit web app will act as an interactive User Interface for our users to be able to give inputs to the model and visualize the results that are returned by the model. The key idea is to encapsulate the model and data as a black-box to which information is passed ( user input) and retrieved (model output).

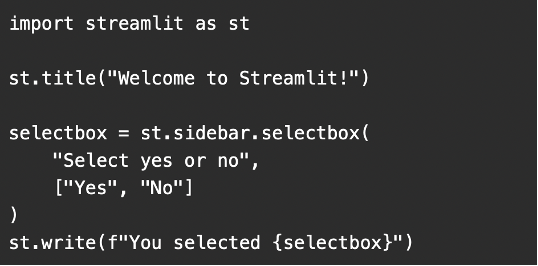
We will be attempting to recreate the wireframe created in the previous section and we will use various features available within the Streamlit library to create these design elements.

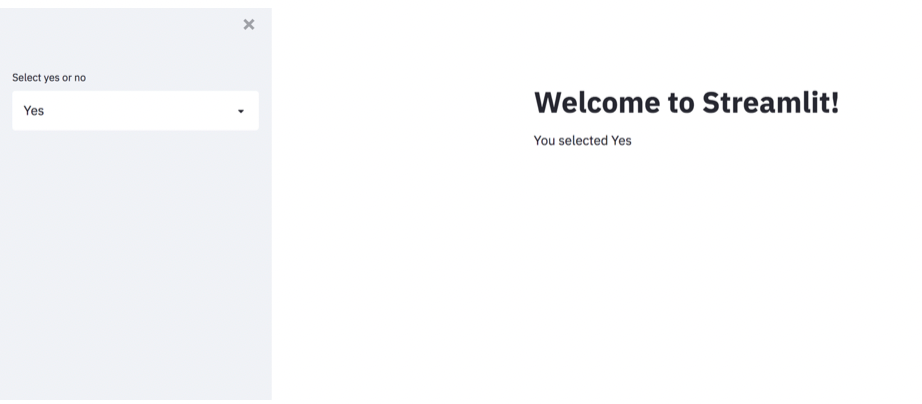
The first aspect is a title for our Streamlit app. It can be implemented using



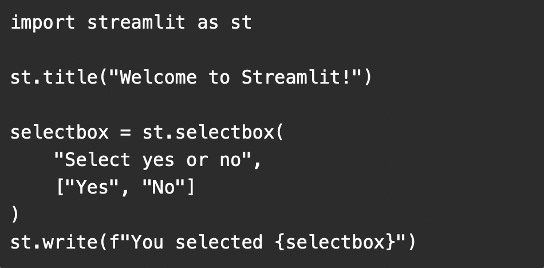


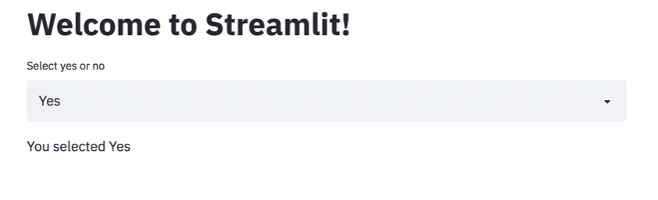
Now as per our wireframe design we will be splitting the layout into two panes.



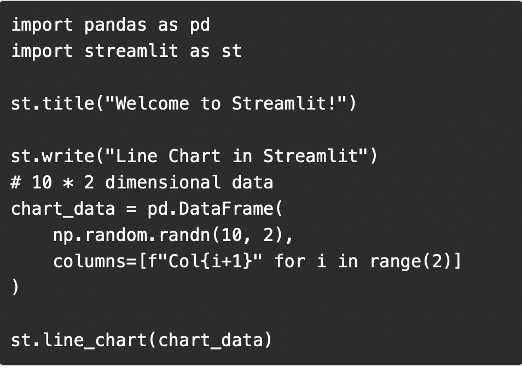


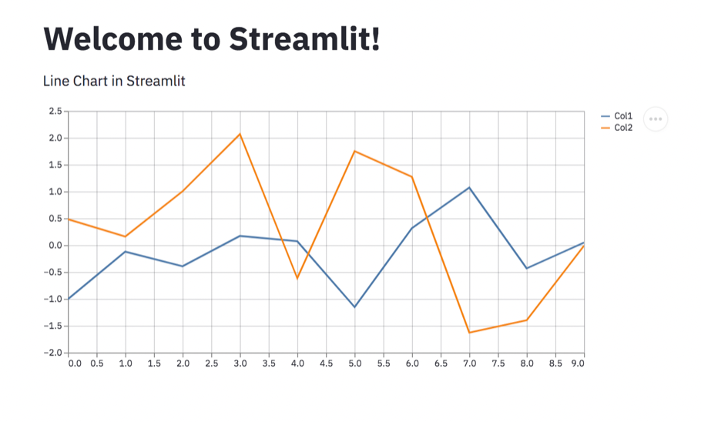
We will include select boxes in the sidebar using the snippet below:





We can also use Streamlit for visualizations and other additional functionalities such as file upload and table display.

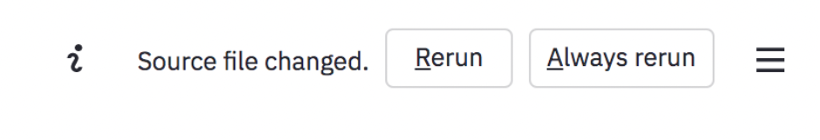




Finally as discussed in the previous sections we can deploy our Streamlit app using the command

>> streamlit run <file\_name.py>

One additional thing to notice is as we make changes to the py file, we can refresh the Streamlit app using the widget on the top right corner as shown below



# Docker



Docker is an open source [containerization](https://www.ibm.com/ae-en/cloud/learn/containerization) platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid [multicloud](https://www.ibm.com/cloud/learn/multicloud) environments.

## Uses of Docker

[Containers](https://www.ibm.com/ae-en/cloud/learn/containers) are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities - such as *control groups* (Cgroups) for allocating resources among processes, and *namespaces* for restricting a processes access or visibility into other resources or areas of the system - enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple [virtual machines (VMs)](https://www.ibm.com/ae-en/cloud/learn/virtual-machines) to share the CPU, memory and other resources of a single hardware server.

As a result, container technology offers all the functionality and benefits of VMs - including application isolation, cost-effective scalability, and disposability - plus important additional advantages:

* Lighter weight: Unlike VMs, containers don’t carry the payload of an entire OS instance and hypervisor; they include only the OS processes and dependencies necessary to execute the code. Container sizes are measured in megabytes (vs. gigabytes for some VMs), make better use of hardware capacity, and have faster startup times.
* Greater resource efficiency: With containers, you can run several times as many copies of an application on the same hardware as you can using VMs. This can reduce your cloud spending.
* Improved developer productivity: Compared to VMs, containers are faster and easier to deploy, provision and restart. This makes them ideal for use in [continuous integration](https://www.ibm.com/ae-en/cloud/learn/continuous-integration) and [continuous delivery](https://www.ibm.com/cloud/learn/continuous-delivery) (CI/CD) pipelines and a better fit for development teams adopting Agile and [DevOps](https://www.ibm.com/ae-en/cloud/learn/devops-a-complete-guide) practices.

## **Create the requirements.txt File**

The Jupyter Docker core images contain the most common libraries, but it is possible to need to install some extra.

## **Create the Dockerfile**

So, we start our image with the Nowcast and Synrad Notebook then we copy the required files from our local computer to the image. Note that we could have used paths and directories. Finally, we install the required libraries in the requirements.txt file.

## **Build the Dockerfile**

Since we have created the Dockerfile, we are ready to build it. The command is the following.

$ docker build -t bigdata\_sevir .

$ docker images

## **Run the Image**

If we want to make sure that the image is running as expected we run:

$ docker run -it -p 8888:8888 bigdata\_sevir

## **Push your Image to Docker Hub**

Once you make sure that the image works as expected, you can push it to Docker Hub so that everyone will be able to pull it. The first thing that you need to do, is to tag your image.

$ docker tag

$ docker push

## **Pull the Image from Docker Hub**

# References

* <https://discuss.streamlit.io/t/upload-keras-models-or-pickled-files/2246/2>
* <https://github.com/MIT-AI-Accelerator/eie-sevir>
* <https://github.com/MIT-AI-Accelerator/neurips-2020-sevir>
* <https://github.com/MIT-AI-Accelerator/sevir_challenges>
* <https://sevir.mit.edu>
* <https://streamlit.io>
* <https://streamlit.io/gallery>
* <https://docs.streamlit.io/library/get-started>
* <https://www.youtube.com/watch?v=JwSS70SZdyM>
* <https://www.youtube.com/watch?v=FTFaQWZBqQ8>
* <https://www.figma.com/resources/learn-design/>
* <https://www.noaa.gov/>
* <https://en.wikipedia.org/wiki/National_Oceanic_and_Atmospheric_Administration>
* <https://pypi.org/>
* <https://the-hitchhikers-guide-to-packaging.readthedocs.io/en/latest/quickstart.html>
* <https://carpentries-incubator.github.io/python-packaging-publishing/03-packaging-installing/index.html>
* <https://realpython.com/pypi-publish-python-package/>
* <https://towardsdatascience.com/how-to-package-your-python-code-df5a7739ab2e>
* <https://fastapi.tiangolo.com/>
* <https://github.com/tiangolo/fastapi>
* <https://realpython.com/fastapi-python-web-apis/>
* <https://www.toptal.com/python/build-high-performing-apps-with-the-python-fastapi-framework>

# Appendix

The following code snippet has been used to generate the system architecture

from diagrams import Diagram, Cluster

from urllib.request import urlretrieve

from diagrams.k8s.controlplane import API

from diagrams.aws.network import APIGateway

from diagrams.aws.general import TraditionalServer

from diagrams.aws.general import GenericDatabase

from diagrams.aws.storage import SimpleStorageServiceS3Bucket

from diagrams.gcp.storage import \_GCP, Filestore, PersistentDisk, Storage

from diagrams.aws.general import Client

from diagrams.gcp.analytics import BigQuery, Dataflow, Datalab

from diagrams.gcp.database import Datastore, Bigtable, SQL

from diagrams.programming.framework import FastAPI

from diagrams.programming.language import Python

from diagrams.onprem.analytics import Tableau

from diagrams.digitalocean.compute import Docker

from diagrams.custom import Custom

graph\_attr = {

"fontsize": "45",

"bgcolor": "transparent"

}

with Diagram("System Architecture", show=False, graph\_attr=graph\_attr, direction='LR') as diag:

with Cluster("Container"):

TraditionalServer = TraditionalServer("Web Server")

Client = Client("Client")

Docker=Docker("Dockerized Container")

BCQ = BigQuery("Google Big Query")

Datalab = Datalab("Data Lab")

Python = Python("Python")

Datastudio = Custom("DataStudio", "./DataStudio.png")

API2 = FastAPI("FastAPI")

CloudStorage = Storage("Google Cloud Storage")

SimpleStorageServiceS3Bucket = SimpleStorageServiceS3Bucket("Sevir S3 Bucket")

BB = Custom("Black Box", "./Black\_Box.png")

Streamlit = Custom("Streamlit", "./Streamlit\_logo.png")

Client >> TraditionalServer

TraditionalServer >> SimpleStorageServiceS3Bucket

SimpleStorageServiceS3Bucket >> CloudStorage

CloudStorage >> Datalab

Datalab >> BB

BB >> BCQ

BCQ >> Datastudio

BCQ >> Python

Python >> API2

Python >> Streamlit

diag # This will illustrate the diagram