

# Overview

CFG.AI Server is a dockerized container enabling the ability to develop applications at the client level and deploy within the enterprise. It manages the logging, monitoring, metering, and access management so that developers only have to focus on the business problem.



This documentation describes:

- How to get your environment set up
- How to build your own GenAI app
- Capabilities of an app

## Resources

[Sample Use Case of Gen AI app \(https://github.com/Deloitte-Default/cfgai-docs/blob/master/AppUseCase.md\)](https://github.com/Deloitte-Default/cfgai-docs/blob/master/AppUseCase.md)

[Demo App \(https://github.com/Deloitte-Default/cfgai-apps/tree/master/demo\)](https://github.com/Deloitte-Default/cfgai-apps/tree/master/demo)

[Policy App \(https://github.com/Deloitte-Default/cfgai-apps/tree/master/policy\)](https://github.com/Deloitte-Default/cfgai-apps/tree/master/policy)

## Key Concepts

### What is a GenAI App?

An AI app consists of three key components: an user interface, an integration component, and a data science component. The user interface can be created using modern JavaScript/HTML libraries such as React and Angular. The integration component provides the ability to get data from a storage, put data into a storage, get data from a database, and put data into a database by the way of Python or Java. The last component is the data science portion. Analytics code development needs to be created to connect to the different models.



In practice, the UI is built using a Javascript framework that the end user can interact with. It will make the call to the middleware code, which will implement some business logic to assimilate structured data, storage data, and vector data. Once it finishes, it will pass it onto the data science portion, which will either make the call to the LLM or run an embedded python routine in order to give a response back so that it may be returned and displayed on the UI.

### What is an Insight?

Throughout this documentation we will be using the term Insight, so it behooves us to take a moment and explain what that is. In CFG, an Insight is best described as a temporal space that allows you to create what you want. Put simply, it is a hosted workspace where you are able to upload your app, and access your hosted data.

Storage Catalogue






One of the key components for integrating your data app with the greater CFG environment will be the InsightProvider that comes along with the SEMOSS SDK. The main purpose of this Provider is to give your app access to all of your hosted data, and to create the Insight that is going to host your app.

### What is a Pixel?

A pixel is a CFG specific term that references a backend call. Much like a standardized API call, though it does not require you to specify what type of API call (i.e post/put/get/delete). The pixel call generally specifies a reactor that is being called, and then any variables that the reactor needs in order to obtain the information that it would require. When discussing pixel calls in this documentation we will strive to also include a sample of the pixel string structure required.

### Anatomy of Gen AI Apps

Generally, the anatomy of an app will follow the file structure seen below:

 classes	8/17/2023 2:18 PM	File folder
 client	8/17/2023 2:18 PM	File folder
 java	8/17/2023 2:18 PM	File folder
 portals	8/17/2023 2:18 PM	File folder
 py	8/17/2023 2:18 PM	File folder

1. **client** - houses front end code, which can be a React application or any other desired framework
  - Not supported - StreamLit
2. **java** - a conglomerate of different methods (what we call "reactors") that can directly interact with the CfG.AI Server
  - It is not necessary to have a java folder unless custom backend code is required
3. **classes** - contains the compiled reactors within the java folder
4. **portals** - bundled version of front end code (static html/css/javascript version of your front end code)
5. **py** - series of python scripts/modules that are called into python