Getting Started

Frontend Installation

SEMOSS Software Development Kit (SDK)
By using the SEMOSS-SDK you will allow your app to work seamlessly within the CfG.AI environment and make use of the Dockerized backend. To download, use your selected node package manager. In this guide, we will use `npm`.

First, install the sdk using a package manager:

```
npm install @semoss/sdk-react
```

Second, install dependencies using a package manager:

```
npm install @semoss/sdk
```

- InsightProvider

Next, import the InsightProvider. This provider will wrap your components and provide an Insight to all of it's children. Insights are temporal workspaces that allow end users to script and interact with a model, storage engine, or database.
import { InsightProvider } from '@semoss/sdk-react';

```
const App = (props) => {
    const { children } = props;

    return <InsightProvider>{children}</InsightProvider>;
};
```

Once the application is wrapped, You can access the insight through the useInsight hook;

```
import { useInsight } from '@semoss/sdk-react';

const Child = (props) => {
    const { children } = props;

    const {
        /** Track if it is initialized **/
        isInitialized,
        /** Track if the user is authorized **/ isAuthorized,
        /** Any Insight Errors **/
        error,
        /** System information **/
        system,
        /** Actions to update **/
        actions,
    } = useInsight();

    return <InsightProvider>{children}</InsightProvider>;
};
```

Now you are ready to go. You can do things like

Query a LLM and return a result

```
const { actions } = useInsight();

const ask = (question) => {
    const { pixelReturn } = await actions.run(
        `LLM(engine=["${ENGINE}"], command=["<encode>${question}</encode>"]);`,
    );

    // get the message
    const message = pixelReturn[0].output.response;
    console.log(message);
};
```

Run a database query

```
const { actions } = useInsight();

const getMovies = () => {
    const { pixelReturn } = await actions.query(
        `Database(engine=["${ENGINE}"]) | Select(Movie__Title, Movie__Year) | Collect(-1)`,
    );

    // get the data
    const data = pixelReturn[0].output;

    console.log(data);
};
```

Login or Logout

```
const { actions } = useInsight();

const login = (username, password) => {
    const success = await actions.login({
        type: 'native',
        username: username,
        password: password,
    });

    console.log(success);
};
```

```
const logout = (username, password) => {
    const success = await actions.logout();

    console.log(success);
};
```

Environment Variables
Define the required environment/constants variables in the .env file which should be located at the root level of your client directory.

```
    MODULE=http://localhost:9090
    ENDPOINT=/ai/Monolith
```

Define the required environment/constants variables in the .env.local file. The ENDPOINT should point to what your local version of Monolith_Dev is named. This file is only necessary if you're planning on doing local development.

```
MODULE=http://localhost:9090
ENDPOINT=/Monolith_Dev
APP=<<APP_ID>>
```

You can access your APP ID once you have the APP hosted in SEMOSS. Your APP ID will be the numerical at the end of the URL of the app.

For example, for the Report Generation App discussed in the End to End use case, this would be the APP ID:
http://localhost:9090/semoss-ui/packages/client/dist/#/app/75277e50-456e-43f8-8ad7-d03224ebe4da

In this case the app id would be: 75277e50-456e-43f8-8ad7-d03224ebe4da

This can be found by going to the App Library and navigating to the hosted app.

Portals Folder
Within the webpack.config.js file ensure the output creates the portals folder

```
output: {
    path: path.resolve(_dirname, '.../portals'),
    filename: '[name].[contenthash].js',
    clean:true,
}
```

Once all changes have been finalized, bundle your frontend code to the portals folder by running:
`pnpm run build`

Backend Installation

Download Java SE Development Kit (JDK): http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

To view example reactors:
https://repo.semoss.org/semoss-training/backend/-/tree/master/tutorials

Python Installation

The AI Server is currently running python 3.10.6 https://www.python.org/downloads/release/python-3106/. For consistency, we advise your local python environment be at least greater than 3.9.

The following packages are currently available within AI Server:

```
accelerate==0.23.0
aiohttp==3.8.5
aiosignal==1.3.1
annoy==1.15.2
appdirs==1.4.4
asttokens==2.2.1
async-timeout==4.0.2
attrs==23.1.0
backcall==0.2.0
backoff==2.2.1
beautifulsoup4==4.12.2
bleach==6.0.0
blinker==1.6.2
boilerpy3==1.0.6
bs4==0.0.1
canals==0.3.2
cattrs==23.1.2
certifi==2023.7.22
charset-normalizer==3.2.0
click==8.1.6
cloudpickle==2.2.1
cmake==3.27.0
comm==0.1.3
contourpy==1.1.0
cryptography==3.4.8
cycler==0.11.0
dask==2023.7.1
datasets==2.14.2
dbus-python==1.2.18
debugpy==1.6.7
decorator==5.1.1
deepdiff==6.3.1
dill==0.3.7
distro==1.7.0
distro-info===1.1build1
docopt==0.6.2
et-xmlfile==1.1.0
Events==0.5
exceptiongroup==1.1.2
executing==1.2.0
faiss-cpu==1.7.2
farm-haystack==1.19.0
filelock==3.12.2
Flask==2.3.2
fonttools==4.41.1
frozenlist==1.4.0
fsspec==2023.6.0
fuzzywuzzy==0.18.0
greenlet==2.0.2
gunicorn==21.2.0
httplib2==0.20.2
huggingface-hub==0.16.4
idna==3.4
importlib-metadata==6.8.0
inflect==7.0.0
iniconfig==2.0.0
ipykernel==6.25.0
ipython==8.14.0
ipywidgets==8.0.7
itsdangerous==2.1.2
jedi==0.19.0
jeepney==0.7.1
jep==3.9.1
Jinja2==3.1.2
joblib==1.3.1
jsonschema==4.18.4
```

```
jsonschema-specifications==2023.7.1
jupyter_client==8.3.0
jupyter_core==5.3.1
jupyterlab-widgets==3.0.8
keyring==23.5.0
kiwisolver==1.4.4
launchpadlib==1.10.16
lazr.restfulclient==0.14.4
lazr.uri==1.0.6
lazy-imports==0.3.1
Levenshtein==0.21.1
lit==16.0.6
llvmlite==0.40.1
locket==1.0.0
MarkupSafe==2.1.3
matplotlib==3.7.2
matplotlib-inline==0.1.6
monotonic==1.6
more-itertools==8.10.0
mpmath==1.3.0
multidict==6.0.4
multiprocess==0.70.15
nest-asyncio==1.5.7
networkx==3.1
nltk==3.8.1
num2words==0.5.12
numba==0.57.1
numpy==1.24.4
nvidia-cublas-cu11==11.10.3.66
nvidia-cuda-cupti-cu11==11.7.101
nvidia-cuda-nvrtc-cu11==11.7.99
nvidia-cuda-runtime-cu11==11.7.99
nvidia-cudnn-cu11==8.5.0.96
nvidia-cufft-cu11==10.9.0.58
nvidia-curand-cu11==10.2.10.91
nvidia-cusolver-cu11==11.4.0.1
nvidia-cusparse-cu11==11.7.4.91
nvidia-nccl-cu11==2.14.3
nvidia-nvtx-cu11==11.7.91
oauthlib==3.2.0
openai==0.27.8
openai-whisper @ git+https://github.com/openai/whisper.git@0a60fcaa9b86748389a656aa013c416030287d47
openpyxl==3.1.2
ordered-set==4.1.0
packaging==23.1
pandas==2.0.3
pandasql==0.7.3
parso==0.8.3
partd==1.4.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==10.0.0
platformdirs==3.10.0
pluggy==1.2.0
posthog==3.0.1
prompt-toolkit==3.0.39
prompthub-py==4.0.0
protobuf==4.24.3
psutil==5.9.5
psycopg2-binary==2.9.6
ptyprocess==0.7.0
pure-eval==0.2.2
pyarrow==12.0.1
pydantic==1.10.12
Pygments==2.15.1
PyGObject==3.42.1
pyjarowinkler==1.8
PyJWT==2.3.0
pyparsing==2.4.7
pytest==7.4.0
python-apt==2.4.0+ubuntu1
python-dateutil==2.8.2
python-Levenshtein==0.21.1
pytz==2023.3
PyYAML==6.0.1
pyzmq==25.1.0
quantulum3==0.9.0
rank-bm25==0.2.2
rapidfuzz==3.1.2
referencing==0.30.0
regex==2023.6.3
requests==2.31.0
requests-cache==0.9.8
rpds-py==0.9.2
safetensors==0.3.1
scikit-learn==1.3.0
scipy==1.11.1
seaborn==0.12.2
SecretStorage==3.3.1
sentence-transformers==2.2.2
sentencepiece==0.1.99
six==1.16.0
soupsieve==2.4.1
SQLAlchemy==1.4.49
SQLAlchemy-Utils==0.41.1
sseclient-py==1.7.2
stack-data==0.6.2
swifter==1.3.5
sympy==1.12
tenacity==8.2.2
text-generation==0.6.0
threadpoolctl==3.2.0
tiktoken==0.3.3
tokenizers==0.13.3
```

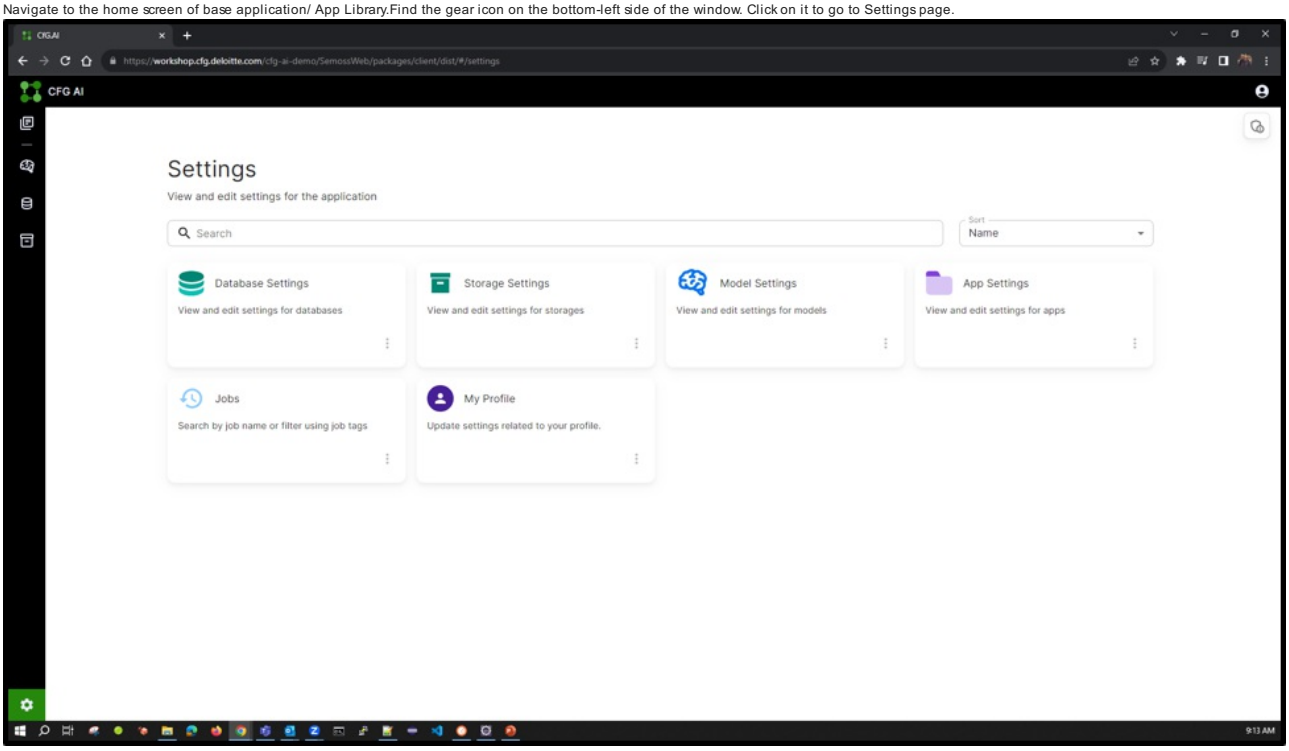
```
tomli==2.0.1
toolz==0.12.0
torch==2.0.1
torchvision==0.15.2
tornado==6.3.2
tqdm==4.65.0
traitlets==5.9.0
transformers==4.31.0
triton==2.0.0
typing_extensions==4.7.1
tzdata==2023.3
unattended-upgrades==0.1
url-normalize==1.4.3
urllib3==2.0.4
wadllib==1.3.6
wcwidth==0.2.6
webencodings==0.5.1
Werkzeug==2.3.6
widgetsnbextension==4.0.8
xlrd==2.0.1
xxhash==3.3.0
yarl==1.9.2
zipp==1.0.0
```

If these packages are not satisfactory for your application, then please reach out to the admin team.

Generating Access and Secret Keys

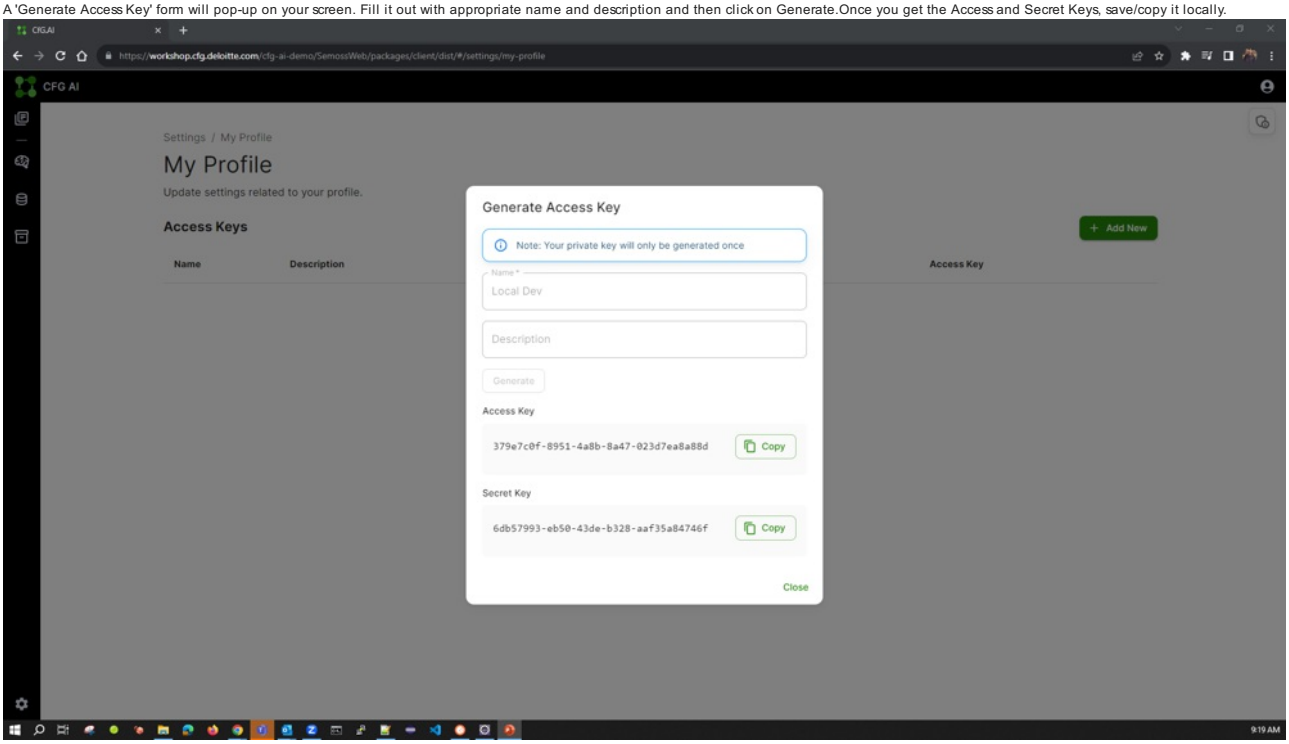
To access the Cfg.AI server from your computers, you need to create a user Access and Secret Key through the base application. The steps are as follows:

Navigate to the home screen of base application/ App Library.Find the gear icon on the bottom-left side of the window. Click on it to go to Settings page.



Go to 'My Profile' to access your user profile and click on 'Add New' to create new Access / Secret Key.

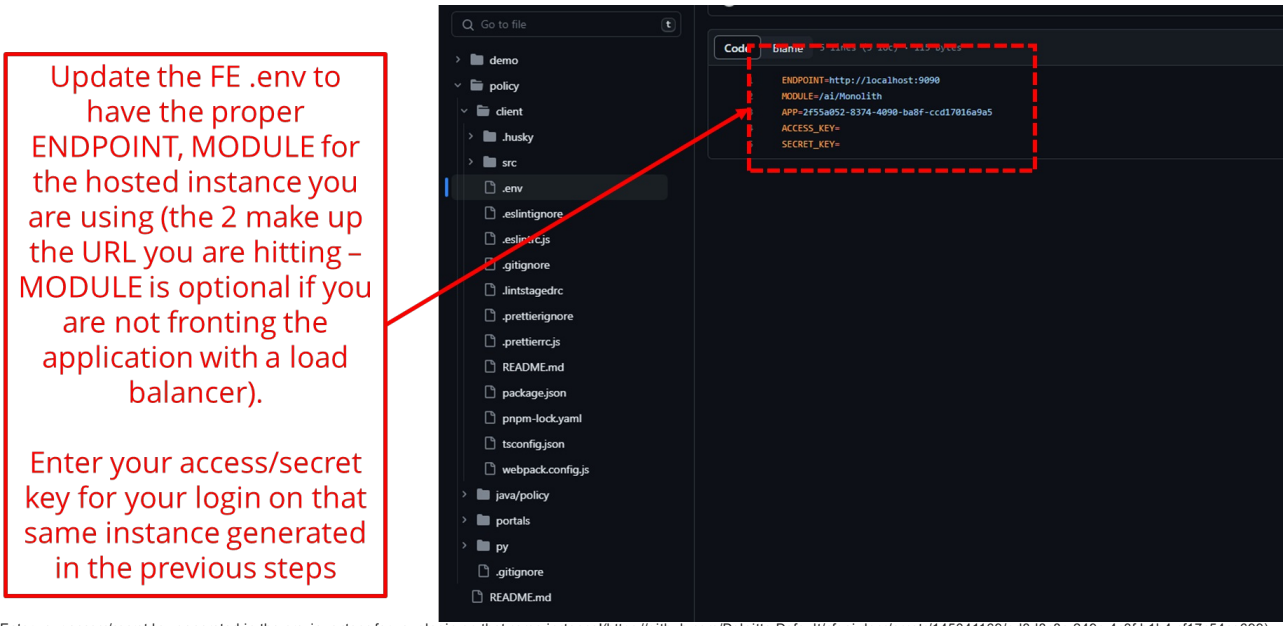
A 'Generate Access Key' form will pop-up on your screen. Fill it out with appropriate name and description and then click on Generate.Once you get the Access and Secret Keys, save/copy it locally.



[**IMPORTANT NOTE:** Please do not share your credentials with anyone]

Update the Front End Environment to have the proper ENDPOINT, MODULE for the hosted instance you are using. MODULE is optional if you are not fronting the application with a load balancer.

```
ENDPOINT=http://localhost:9090 MODULE=/ai/Monolith APP= ACCESS_KEY= SECRET_KEY=
```



Enter your access/secret key generated in the previous steps for your login on that same instance!(https://github.com/Deloitte-Default/cfgai-docs/assets/145041169/ed3d8c8e-249a-4c3f-b1b4-cf17c54ce699)

Tips and Tricks

Here are a few tips and tricks that can help streamline the development process.

Development Environment

Note: We recommend manually setting the environment only in development mode.

You can setup a development environment and use access keys to authenticate with the app server. Generate the keys on the server and then update the Env module. See:

```
// import the module
import { Env } from '@semoss/sdk';

// update the environment
Env.update({
    /**
     * Url pointing to the app server
     **/
    MODULE: '',
    /**
     * Access key generated by the app server
     **/
    ACCESS_KEY: '',
    /**
     * Secret key generated by the app server
     **/
    SECRET_KEY: '',
    /**
     * Optional field. This will load app specific reactors into the insight. Your app has to be hosted and running on the app server.
     **/
    APP: '',
});
```

Note: Please do not commit your keys. Instead externalize your keys to a .env and load them in as environment variables during development

Python
The app server allows you to write custom python to power your app. You can initialize your python environment by:

Loading via a file

The sdk will load python via an external file.

```
# ./hello.py
def sayHello(name):
    print(f'Hello {name}')
Set the option on initialize:
// import the module
import { Env } from '@semoss/sdk';

// update the environment
insight.initialize({
    python: {
        /**
         *  Load the python via an external file
         **/
        type: 'file',
        /**
         *  Path to the file
         **/
        path: './hello.py',
        /**
         *  Alias for the file
         **/
        alias: 'smss',
    },
});
```

Loading via js

The sdk will load python via an external file.

```
# ./hello.py

Set the option on initialize:
// import the module
import { Env } from '@semoss/sdk';

// define it int he js
const py = `
def sayHello(name):
    print(f'Hello {name}')
`;

// update the environment
insight.initialize({
    python: {
        /**
         *  Load the python via js
         **/
        type: 'script',
        /**
         *  Path to the file
         **/
        script: py,
        /**
         *  Alias for the file
         **/
        alias: 'smss',
    },
});
```

Next you can the preloaded python methods by calling the runPy action. See

```
const hello = (name) => {
    const { pixelReturn } = await insight.actions.runPy(
        `smss.sayHello(${name})`,
    );

    // get the data
    const data = pixelReturn[0].output;

    console.log(data);
};
```