A scalable cloud-based platform for laser plasma experimental data collection, sharing, and machine-learning

Archis Joglekar[1,2], C. Sullivan[2], Y. Ma[2], Q. Qian[2], A. Antoine[2], B. Hou[2], A. G. R. Thomas[2]
1 - Ergodic LLC, San Francisco, CA
2 - Center for Ultrafast Optical Science, University of Michigan, Ann Arbor, MI

Laser Plasma Accelerator Online Workshop
January 26, 2022

The primary goal was to build a lightweight system for a small user-facility / research group to be able to organize / store / share data at scale

The system enables 3 workflows

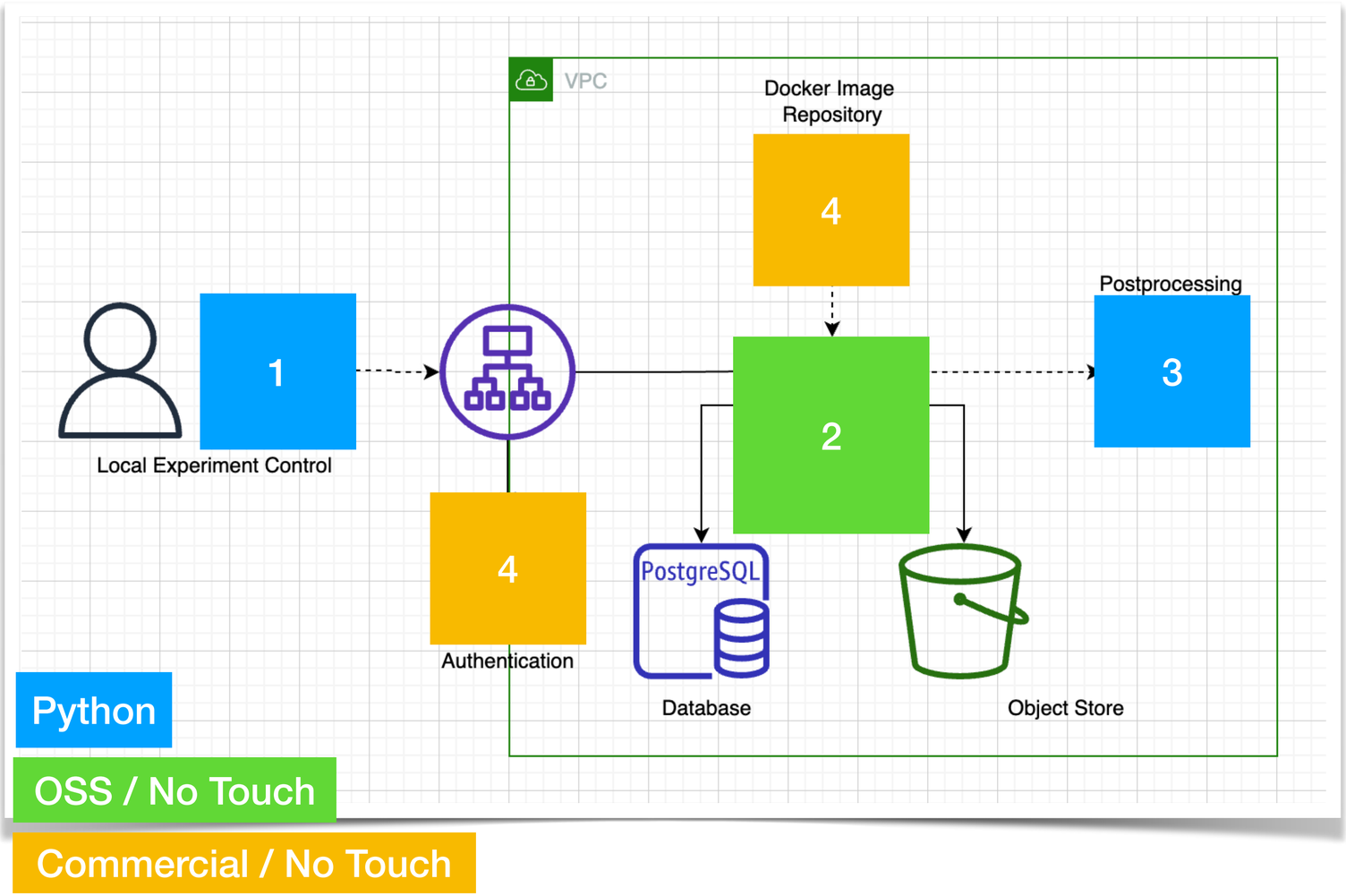
1. Data Entry
2. Data Storage, Access, Download (and some Viz)
3. Downstream use for ML

The primary goal was to build a lightweight system for a small user-facility / research group to be able to organize / store / share data at scale

In the process we learned

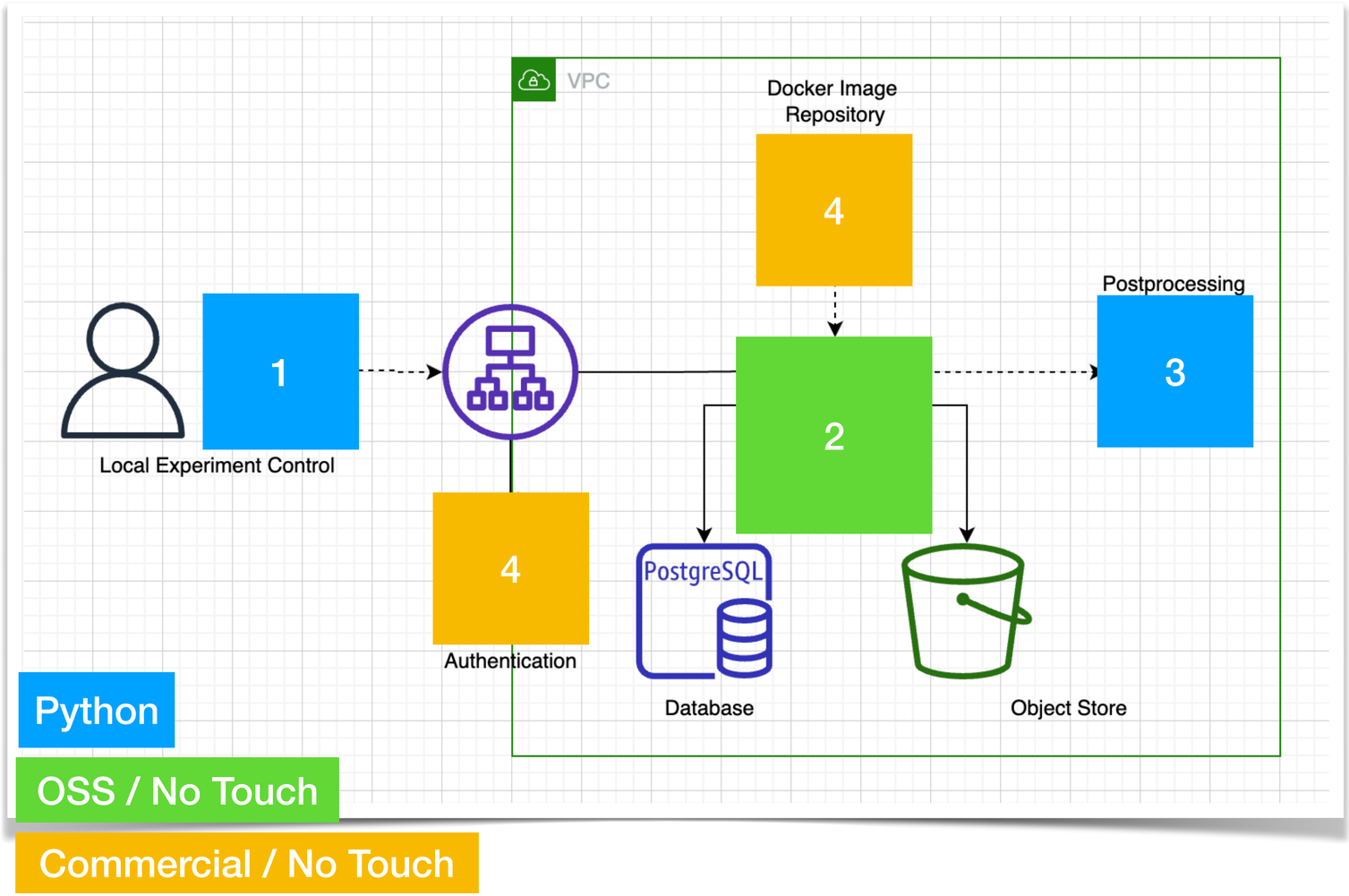
1. What kind of expertise is needed?
2. How can we replicate these efforts?
3. What can be improved?

The 3 workflows rely on Cloud APIs, Open Source Software, and Containers



The 3 workflows rely on Cloud APIs, Open Source Software, and Containers

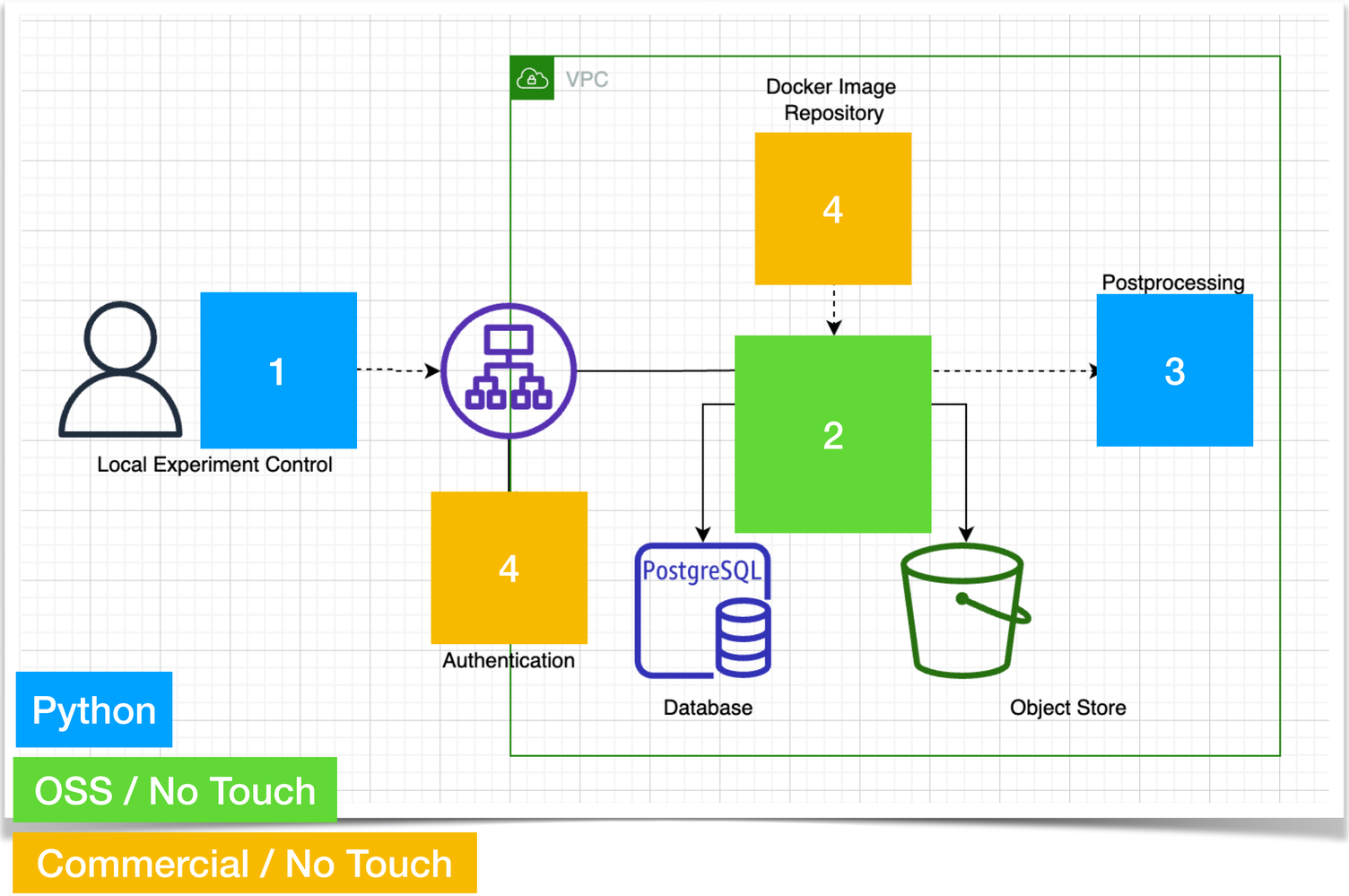
1. **Data Input <> Frontend / Programmatic Access**



The 3 workflows rely on Cloud APIs, Open Source Software, and Containers

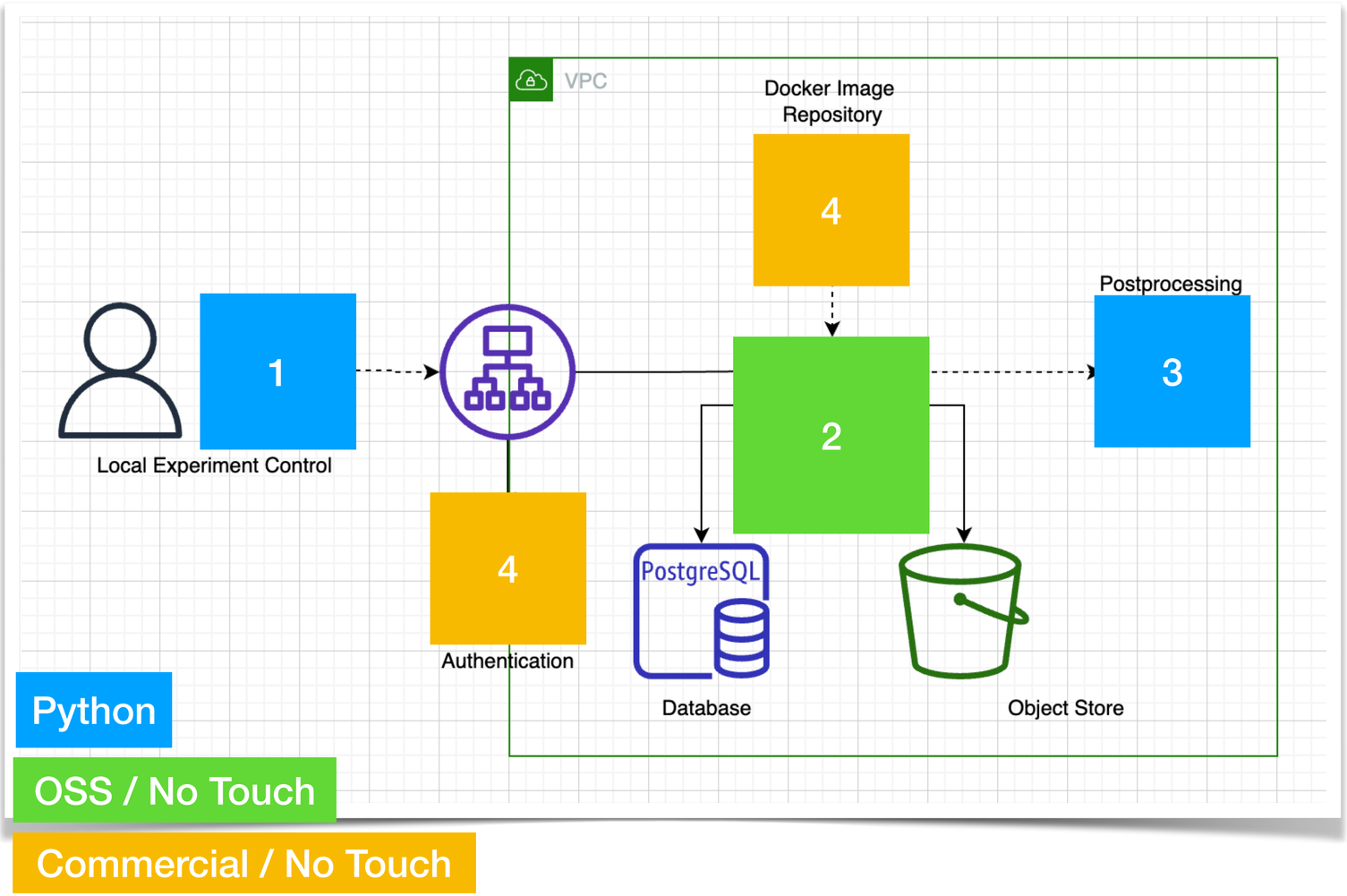
1. **Data Input <> Frontend / Programmatic Access**

2. **Data Storage, Access, Download and Visualization <> Database + Object Storage**



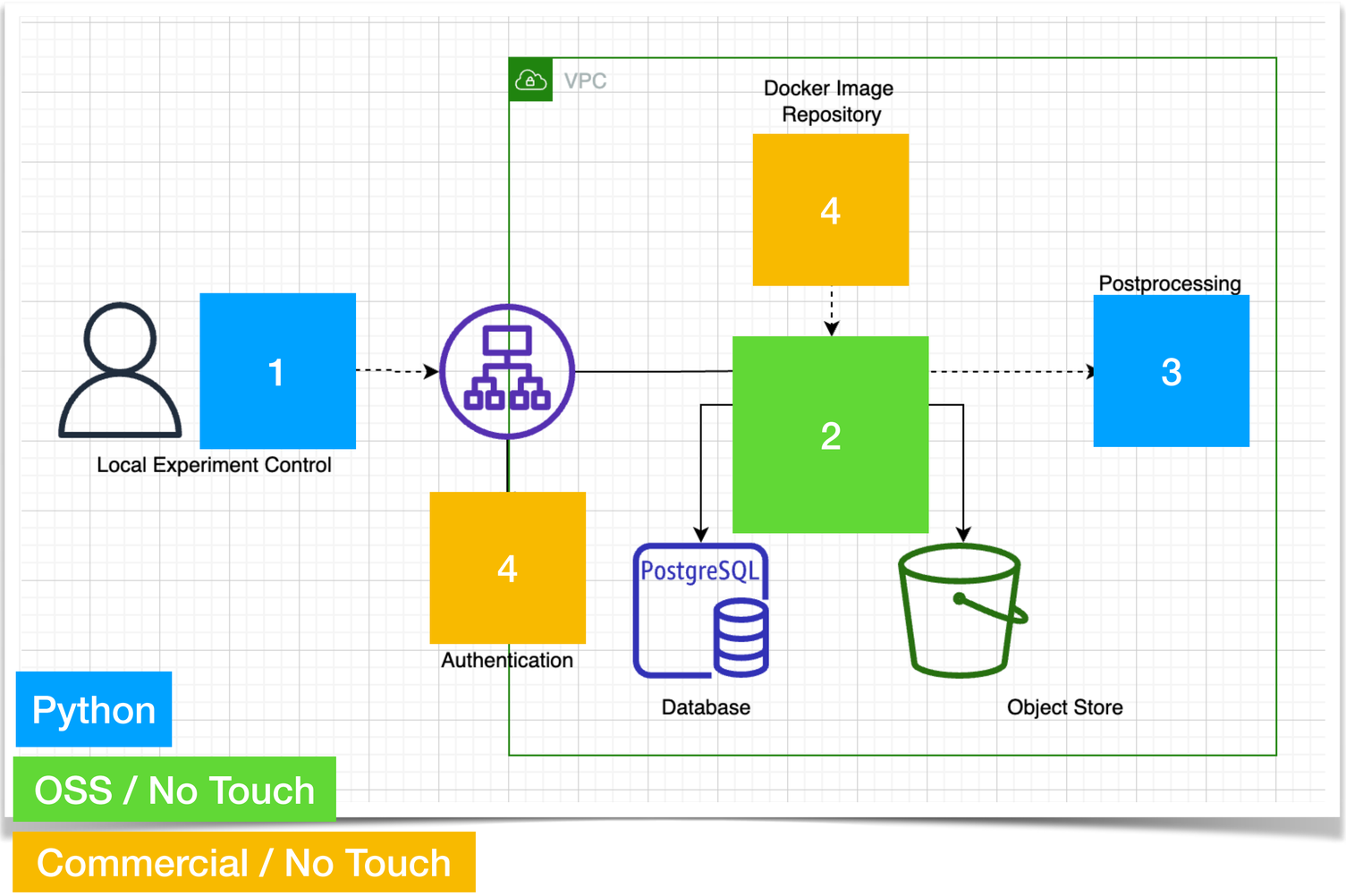
The 3 workflows rely on Cloud APIs, Open Source Software, and Containers

1. Data Input <> Frontend / Programmatic Access
2. Data Storage, Access, Download and Visualization <> Database + Object Storage
3. Downstream use for ML <> Jupyter Notebooks



The 3 workflows rely on Cloud APIs, Open Source Software, and Containers

1. Data Input <> Frontend / Programmatic Access
2. Data Storage, Access, Download and Visualization <> Database + Object Storage
3. Downstream use for ML <> Jupyter Notebooks
4. Other e.g. web hosting, authentication, docker containers



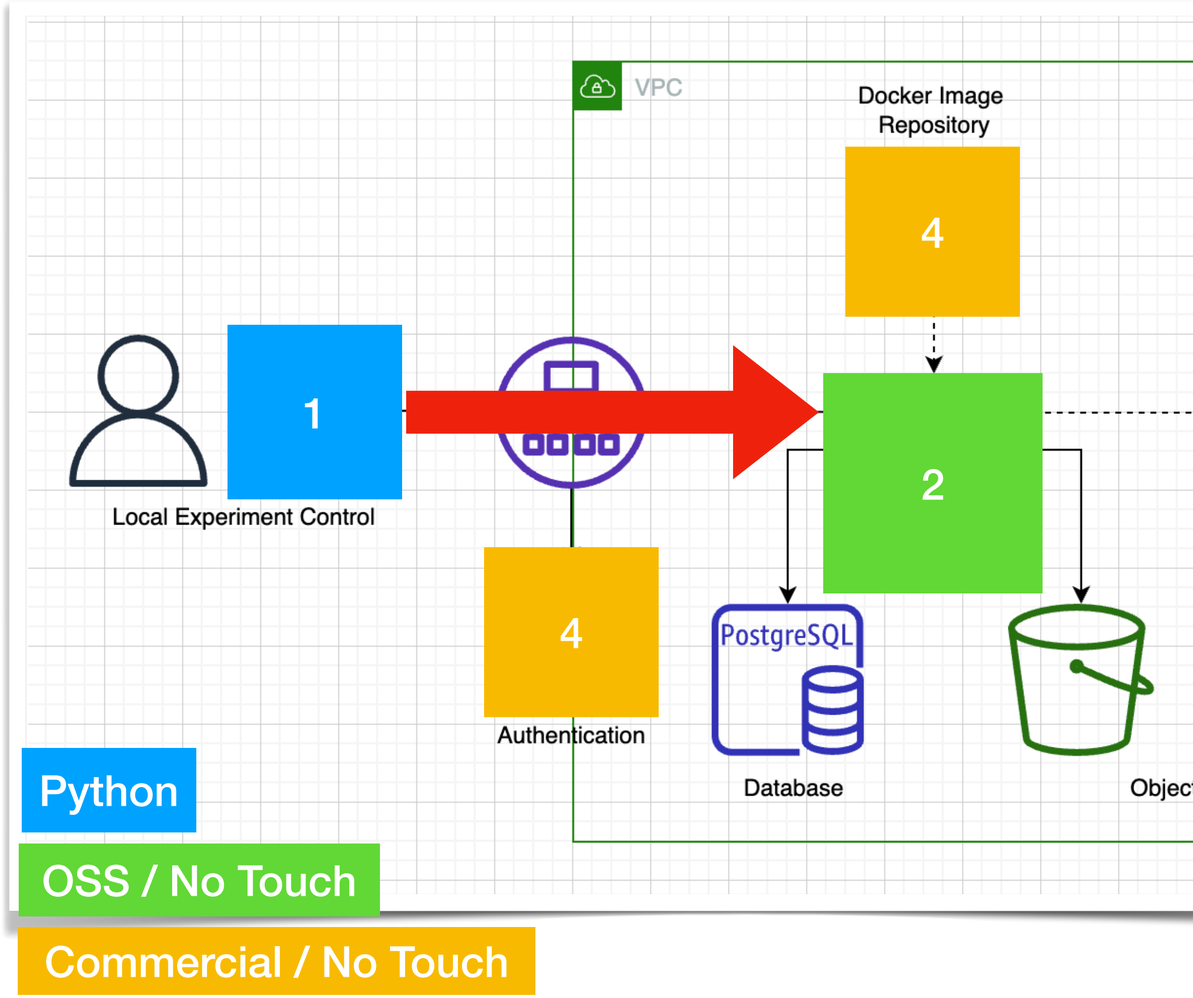
The captured data is logged to an MLFlow Server

- MLFlow describes each “run” (shot) using
  - **Parameters**
    - Can be any fundamental “type” i.e. float, int, string
  - **Metrics**
    - These are typically numeric i.e. float, int
  - **Artifacts**
    - These can be ANY files i.e. .png, .mat, .tiff, .h5 etc.

Programmatic Data Ingestion

TL; DR -

1. Have pre-existing data
2. Get a local or cloud-based Python process
3. Access cloud server programmatically
4. Log data



Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

```python
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```python
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)


                mlflow.log_artifacts(td)  ## Logging Files Here!
```

Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

- We use a cloud-hosted Jupyter Notebook but any computer that has access to the MLFlow server can do this job

```python
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```python
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)


                mlflow.log_artifacts(td)  ## Logging Files Here!
```

Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

- We use a cloud-hosted Jupyter Notebook but any computer that has access to the MLFlow server can do this job
- The flow is

```python
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```python
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)


                mlflow.log_artifacts(td)  ## Logging Files Here!
```

Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

- We use a cloud-hosted Jupyter Notebook but any computer that has access to the MLFlow server can do this job
- The flow is
  - Loop over all shots

```
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)


                mlflow.log_artifacts(td)  ## Logging Files Here!
```

Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

- We use a cloud-hosted Jupyter Notebook but any computer that has access to the MLFlow server can do this job
- The flow is
  - Loop over all shots
    - Generate Parameters, Metrics, Artifacts for each shot

```
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)


                mlflow.log_artifacts(td)  ## Logging Files Here!
```

Data can be bulk ingested e.g. through a cloud-hosted Jupyter Notebook

- We use a cloud-hosted Jupyter Notebook but any computer that has access to the MLFlow server can do this job
- The flow is
  - Loop over all shots
    - Generate Parameters, Metrics, Artifacts for each shot
    - Log each of the above

```
from matplotlib import pyplot as plt
from tqdm import tqdm
import numpy as np

if __name__ == "__main__":
    mlflow.set_tracking_uri("http://mlfs.scribe:5000")
    mlflow.set_experiment("TA 2")
    data_dir = "./"
    df = pd.read_csv("./TA_2data_info.csv")
    for idx, row in tqdm(df.iterrows()):
        with mlflow.start_run(run_name=str(row["burst number"]) + "-" + str(row["shot name"])):
            with tempfile.TemporaryDirectory() as td:
                params = {
                    label: row[label] for label in ["gas pressure", "cell length", "burst number", "shot name"]
                }

                mlflow.log_params(params) ## Logging Parameters Here!
```

```
                    metrics = {
                        "sum-" + str(interp): np.sum(da.data),
                        "mean-" + str(interp): np.mean(da.data),
                        "max-" + str(interp): np.amax(da.data),
                        "max energy location-" + str(interp): tst["low scale"][unraveled_loc[1]],
                        "max angle location-" + str(interp): tst["arbitrary angle"][unraveled_loc[0]],
                    }
                    mlflow.log_metrics(metrics)  ## Logging Metrics Here!

                    fig = plt.figure(figsize=(10,6))
                    ax=fig.add_subplot(111)
                    da.plot(ax=ax)
                    fig.savefig(os.path.join(plot_dir, interp + ".png"), bbox_inches="tight")

                    da.to_netcdf(os.path.join(files_dir, interp + ".nc"), engine="h5netcdf")
                    da.close()
                    plt.close(fig)

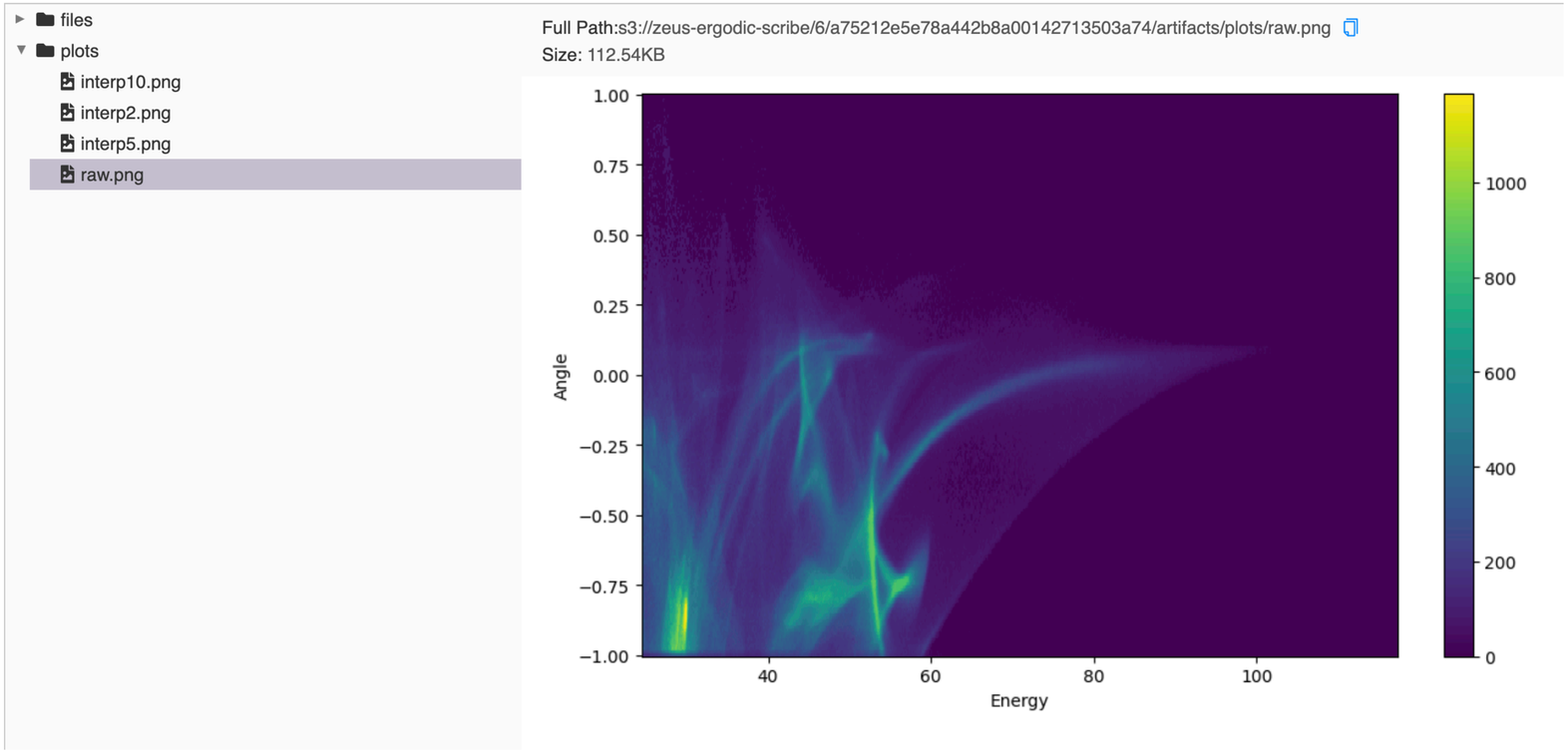

                mlflow.log_artifacts(td)  ## Logging Files Here!
```

This flow is used to collect the experimental data from the TA2 experiment from Streeter et. al

▼ Metrics

Name	Value
max angle location-interp10	-0.856
max angle location-interp2	-0.842
max angle location-interp5	-0.856
max angle location-raw	-0.82
max energy location-interp10	29.95
max energy location-interp2	29.95
max energy location-interp5	29.95
max energy location-raw	30
max-interp10	1186.4
max-interp2	1186.9
max-interp5	1186.4
max-raw	1187.8
mean-interp10	91.03
mean-interp2	89.62
mean-interp5	90.22

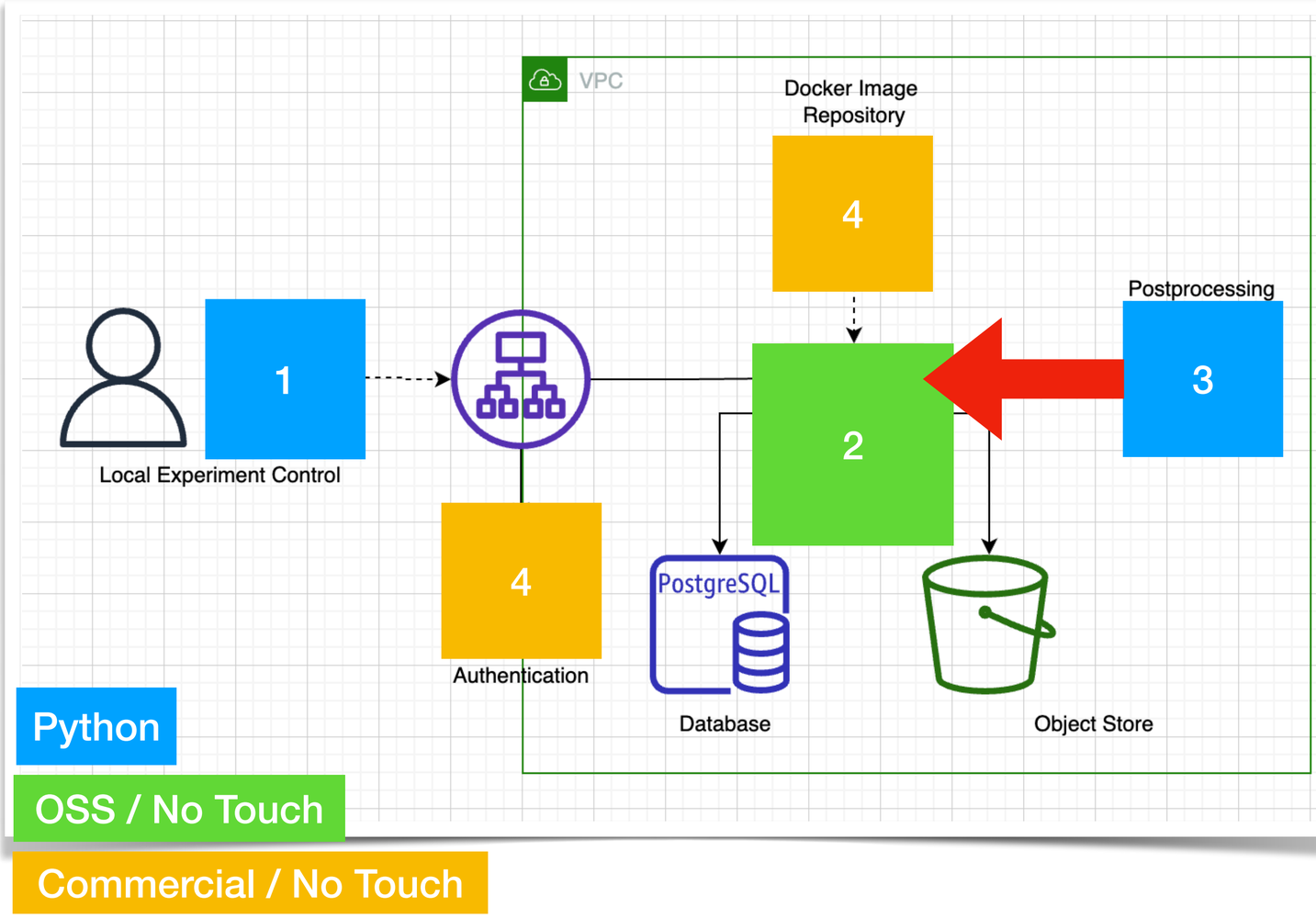
▼ Artifacts



Performing ML on the data

TL;DR -

1. Use a local or cloud machine
2. Access data programmatically
3. Do your ML
4. Log to MLFlow server



Performing ML on the cloud

- Scientists run highly bespoke code for post processing — maybe unified per “experiment”
- Lowest barrier to entry for highly bespoke code is Jupyter Notebooks
- Here we perform a hyper parameter search over a series of generative models
- BONUS: We reuse the MLFlow server, this time to actually track ML experiments!
  - The same principles as for physical data collection apply
    - Log parameters (pre-experiment parameters)
    - Execute experiment / Train Model
    - Log metrics (experiment/training results)
    - Log Artifacts (experiment/training results)

```python
def run_job(run_id=None):
    with mlflow.start_run(run_id=run_id) as run:
        mlflow.set_tags({"job_status": "training"})
        with tempfile.TemporaryDirectory() as td:
            artifact_uri = run.info.artifact_uri

            hparams = download_and_open_file("hparams", artifact_uri, td)

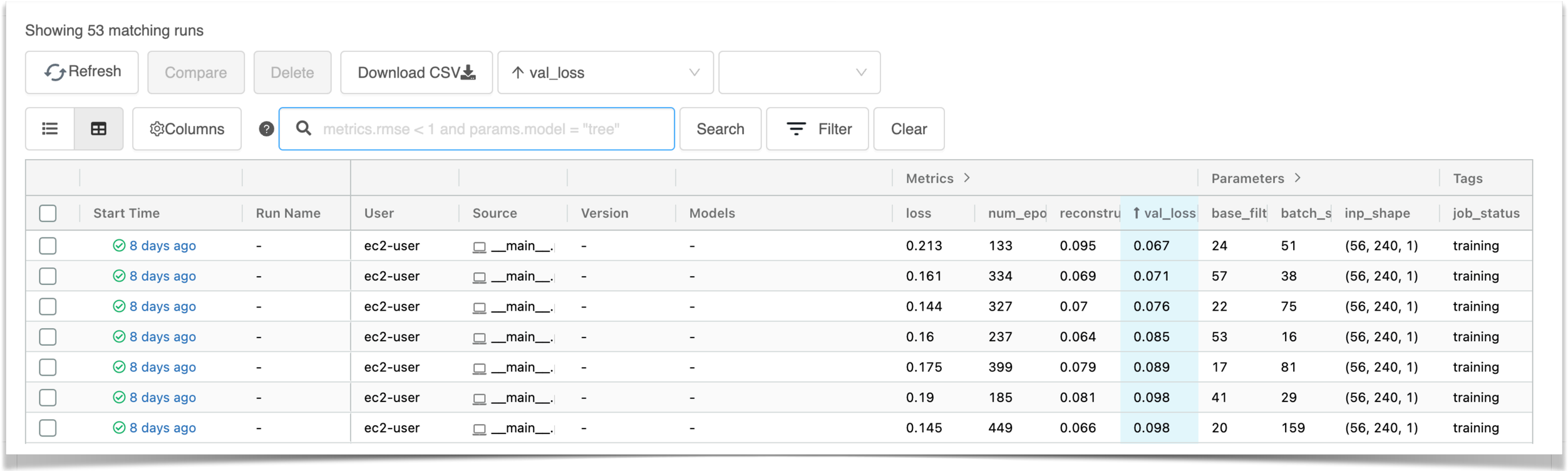
            dirs = {
                "val": os.path.join(td, "validation"),
                "test": os.path.join(td, "test"),
                "data": os.path.join(td, "data"),
                "model": os.path.join(td, "model"),
            }
            _ = [os.makedirs(val) for key, val in dirs.items()]

            data_dict, scaled_data = load_data(
                hparams["interpolation"], dirs["data"], hparams["batch_size"]
            )
            train_ds, val_ds, test_ds = scaled_data

            mlflow.log_params(
                {
                    "batch_size": hparams["batch_size"],
                    "num_embeddings": hparams["num_embeddings"],
                    "latent_dim": hparams["latent_dim"],
                    "base_filter": hparams["base_filter"],
                    "learning_rate": hparams["learning_rate"],
                }
            )
            vqvae.train_and_test(hparams, train_ds, val_ds, test_ds, dirs, data_dict)

            mlflow.log_artifacts(td)
            mlflow.log_params(data_dict)
```

We can use the MLFlow UI again to observe the results of the ML training runs



Showing 53 matching runs

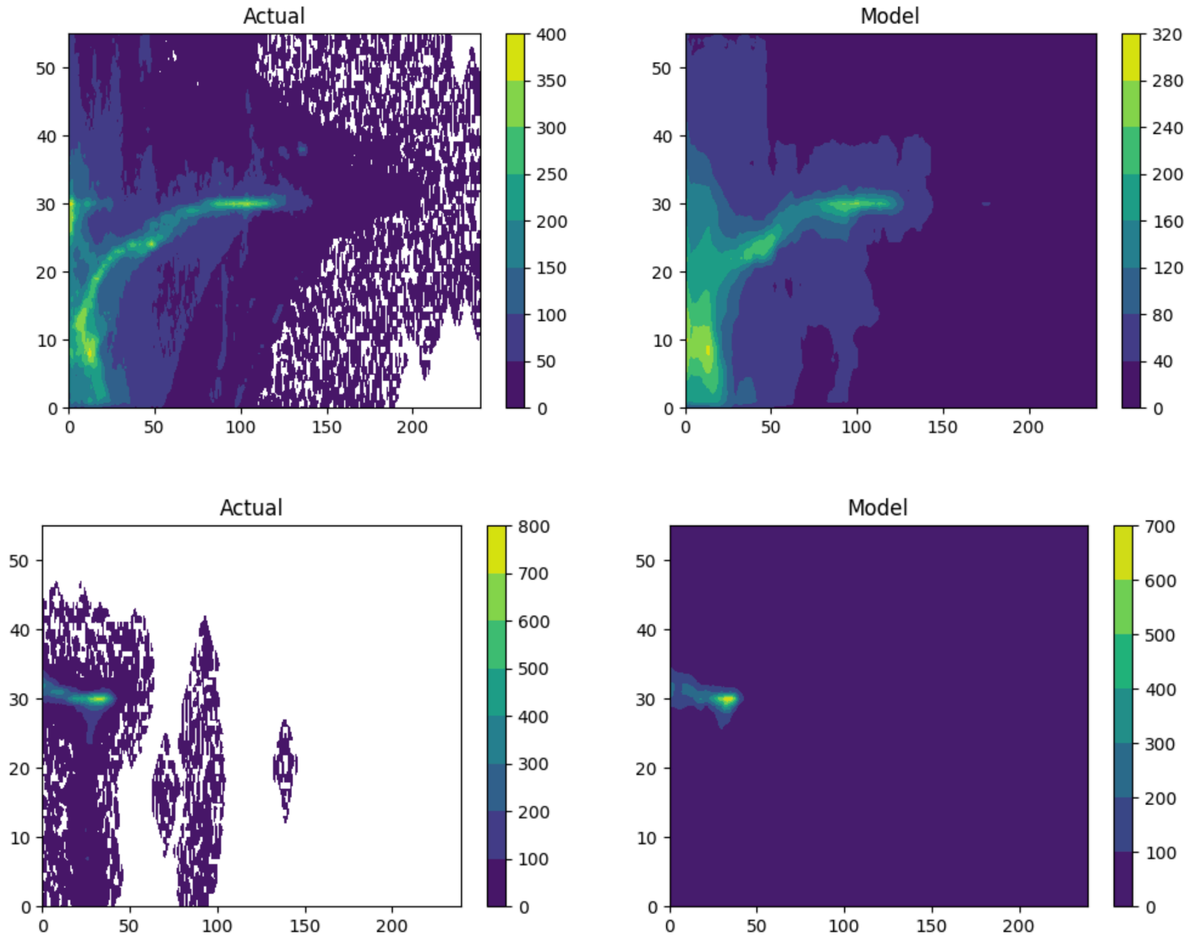
Refresh | Compare | Delete | Download CSV | ↑ val_loss | Columns | metrics.rmse < 1 and params.model = "tree" | Search | Filter | Clear

							Metrics				Parameters			Tags
	Start Time	Run Name	User	Source	Version	Models	loss	num_epo	reconstru	↑ val_loss	base_filt	batch_s	inp_shape	job_status
	8 days ago	-	ec2-user	__main__.	-	-	0.213	133	0.095	0.067	24	51	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.161	334	0.069	0.071	57	38	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.144	327	0.07	0.076	22	75	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.16	237	0.064	0.085	53	16	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.175	399	0.079	0.089	17	81	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.19	185	0.081	0.098	41	29	(56, 240, 1)	training
	8 days ago	-	ec2-user	__main__.	-	-	0.145	449	0.066	0.098	20	159	(56, 240, 1)	training

Example results from the ML model training

- We train ~50 models on Cloud GPUs in order to perform a hyper parameter search over 8 different parameters
- Cloud GPUs use data from existing MLFlow server to do the training

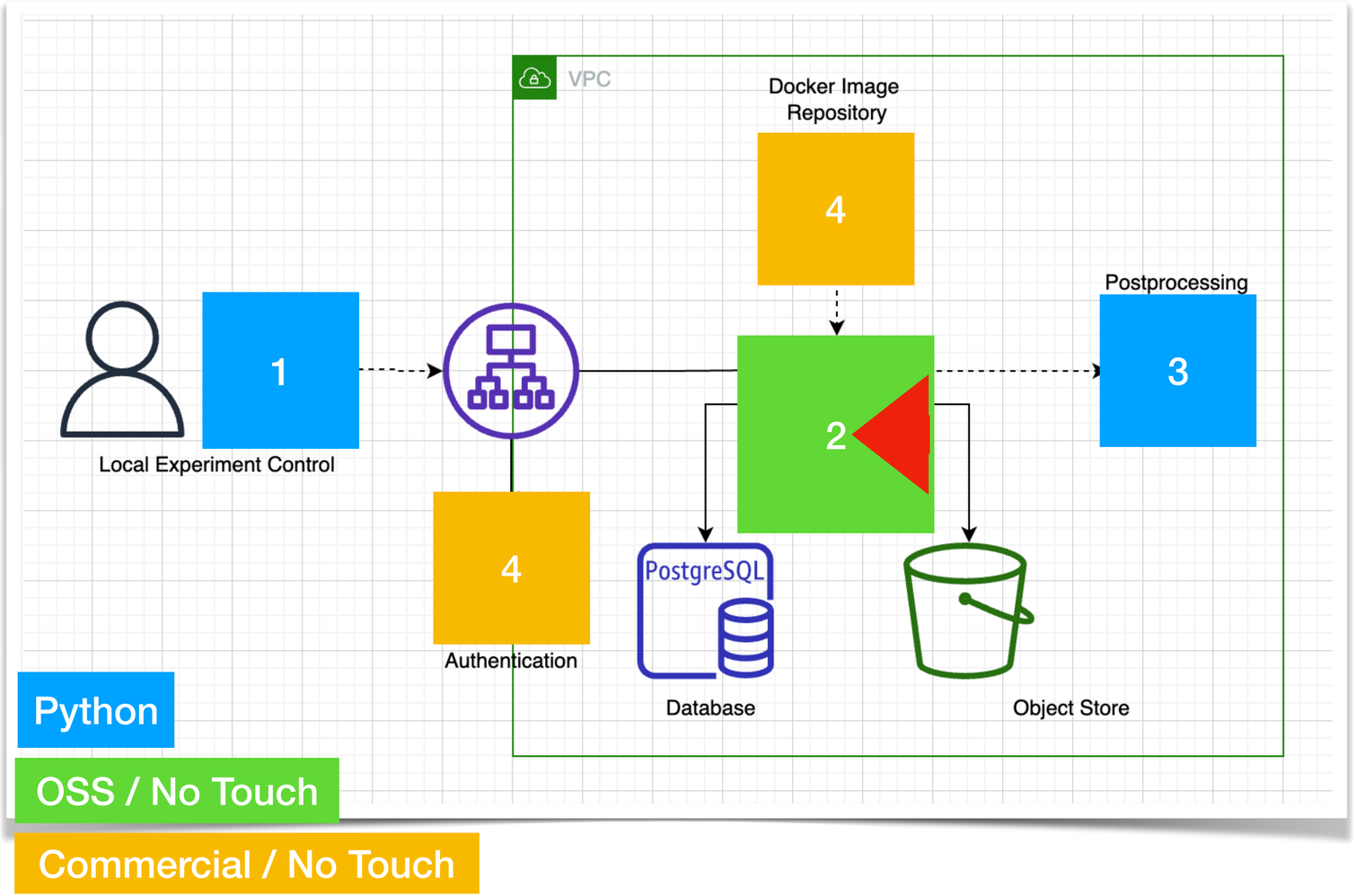
Full Path:s3://zeus-ergodic-scribe/10/3d9d99aadf484d02aba8b670ec039f15/artifacts/validation/6.png
Size: 33.78KB

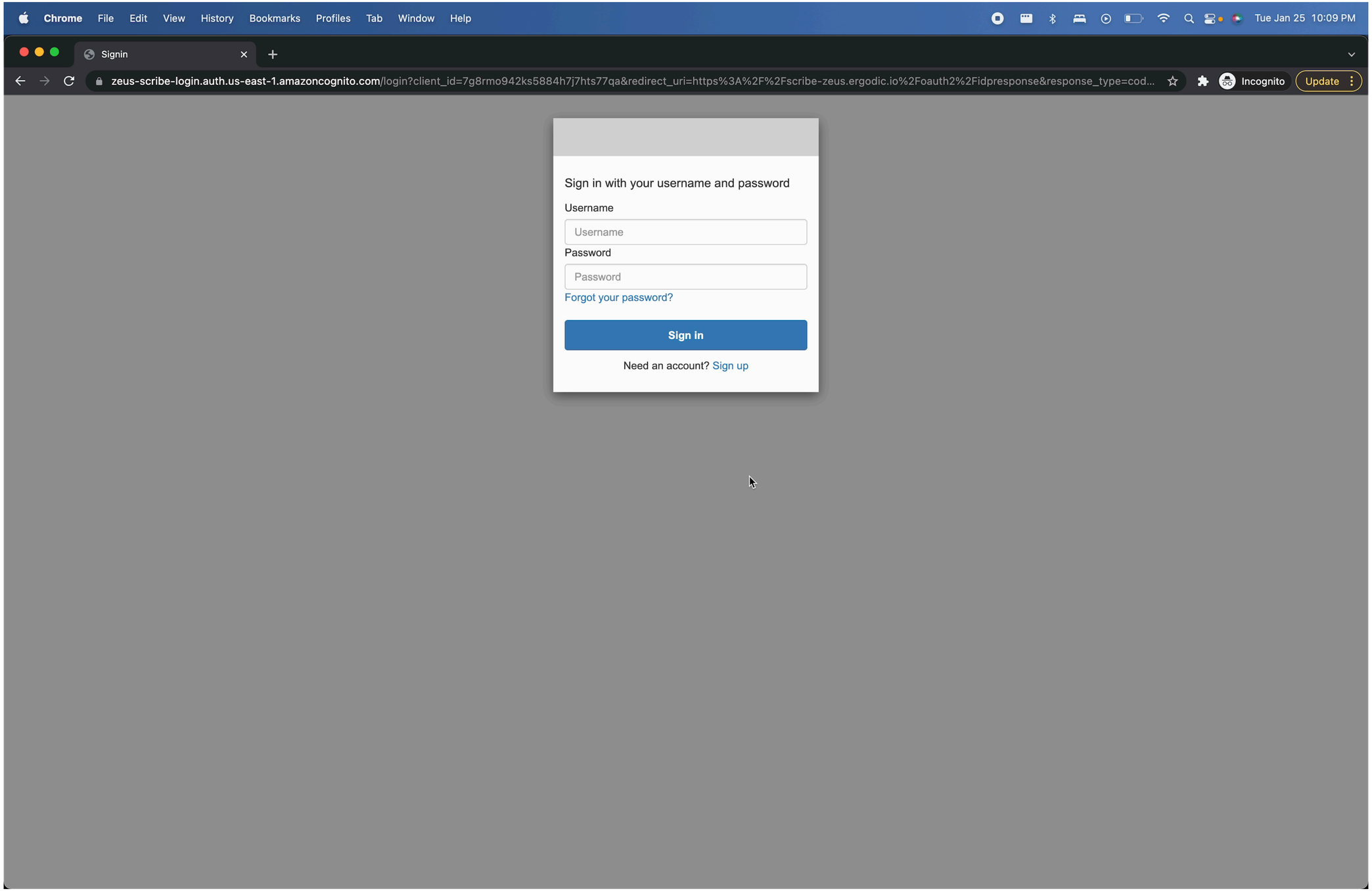


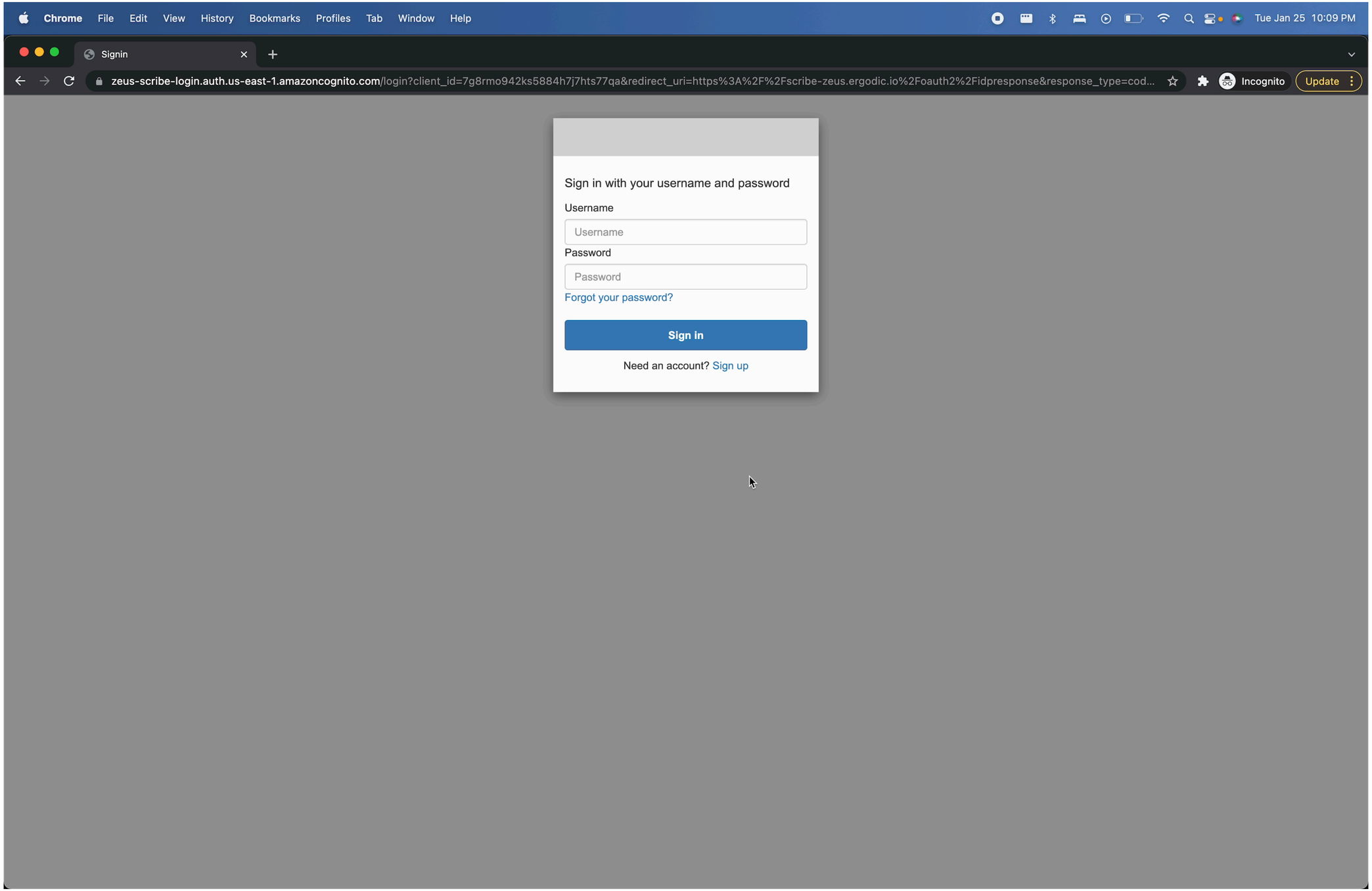
Accessing Data using MLFlow

TL; DR -

1. Browse MLFlow Server through Web Browser
2. Visualize data
3. Sort data
4. Download data




Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Jan 25 10:09 PM
Signin
zeus-scribe-login.auth.us-east-1.amazoncognito.com/login?client_id=7g8rmo942ks5884h7j7hts77qa&redirect_uri=https%3A%2F%2Fscribe-zeus.ergodic.io%2Foauth2%2Fidpresponse&response_type=cod...
Incognito
Update
Sign in with your username and password
Username
Username
Password
Password
Forgot your password?
Sign in
Need an account? Sign up


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue Jan 25 10:09 PM
Signin
zeus-scribe-login.auth.us-east-1.amazoncognito.com/login?client_id=7g8rmo942ks5884h7j7hts77qa&redirect_uri=https%3A%2F%2Fscribe-zeus.ergodic.io%2Foauth2%2Fidpresponse&response_type=cod...
Incognito
Update
Sign in with your username and password
Username
Username
Password
Password
Forgot your password?
Sign in
Need an account? Sign up

This software enables the following workflows

This software enables the following workflows

1. Data Entry

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure
2. How can we easily replicate these efforts?

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure
2. How can we easily replicate these efforts?
    1. Infrastructure-as-code

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure
2. How can we easily replicate these efforts?
    1. Infrastructure-as-code
3. What can be improved (if needed)?

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure
2. How can we easily replicate these efforts?
    1. Infrastructure-as-code
3. What can be improved (if needed)?
    1. Customizable DB UI

This software enables the following workflows

1. Data Entry
    1. Web UI - Streamlit (Python) ~ 200 lines
    2. Programmatic Access (Python) ~ O(100) lines
2. Data Storage, Access, Download (and some Viz)
    1. Web UI - MLFlow (Python)
3. Downstream use for ML
    1. Managed Jupyter Environment (Python)

In the process we learned

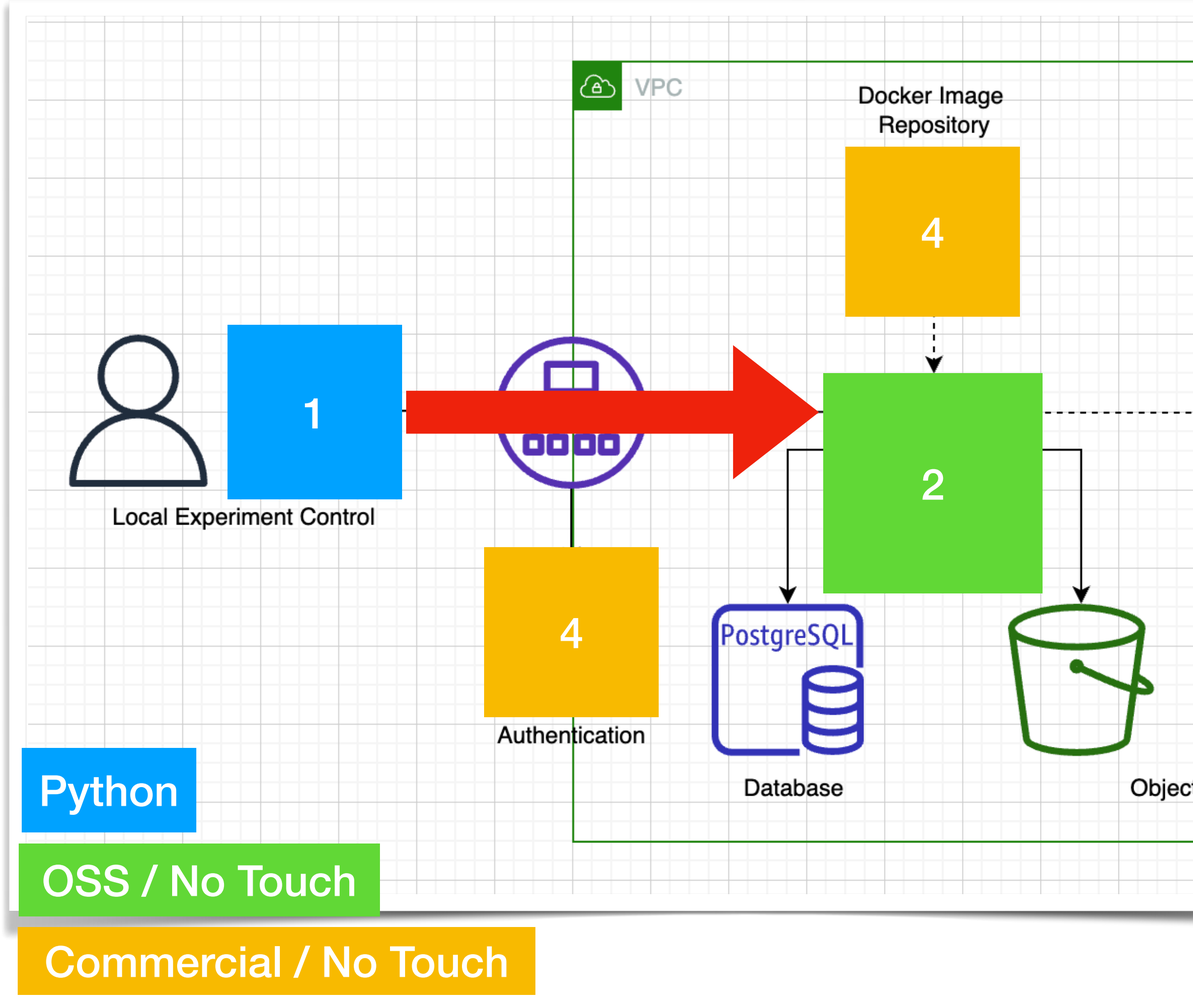
1. What kind of expertise is needed?
    1. Python for scientists
    2. 0.1(?) FTE for Infrastructure
2. How can we easily replicate these efforts?
    1. Infrastructure-as-code
3. What can be improved (if needed)?
    1. Customizable DB UI
    2. Bigger / Better infrastructure

Extras

Data Entry Web App

TL;DR -

1. Customizable Python App for Data Entry UI
2. Log Python Dictionary with an API call to MLFlow Server
3. Log Files with an API call to MLFlow Server



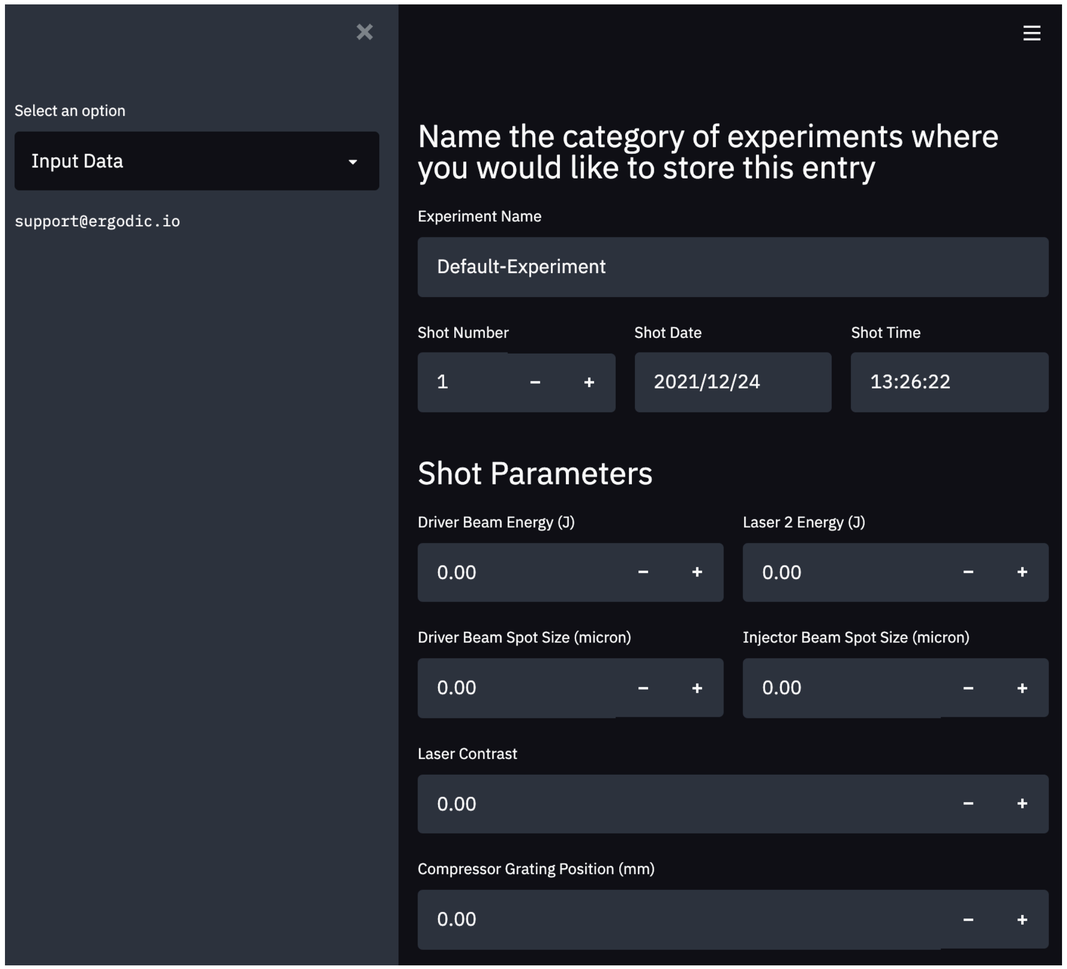
Streamlit is OSS for writing Simple Data Apps in Python

```python
def _input_parameters_(shot_num):

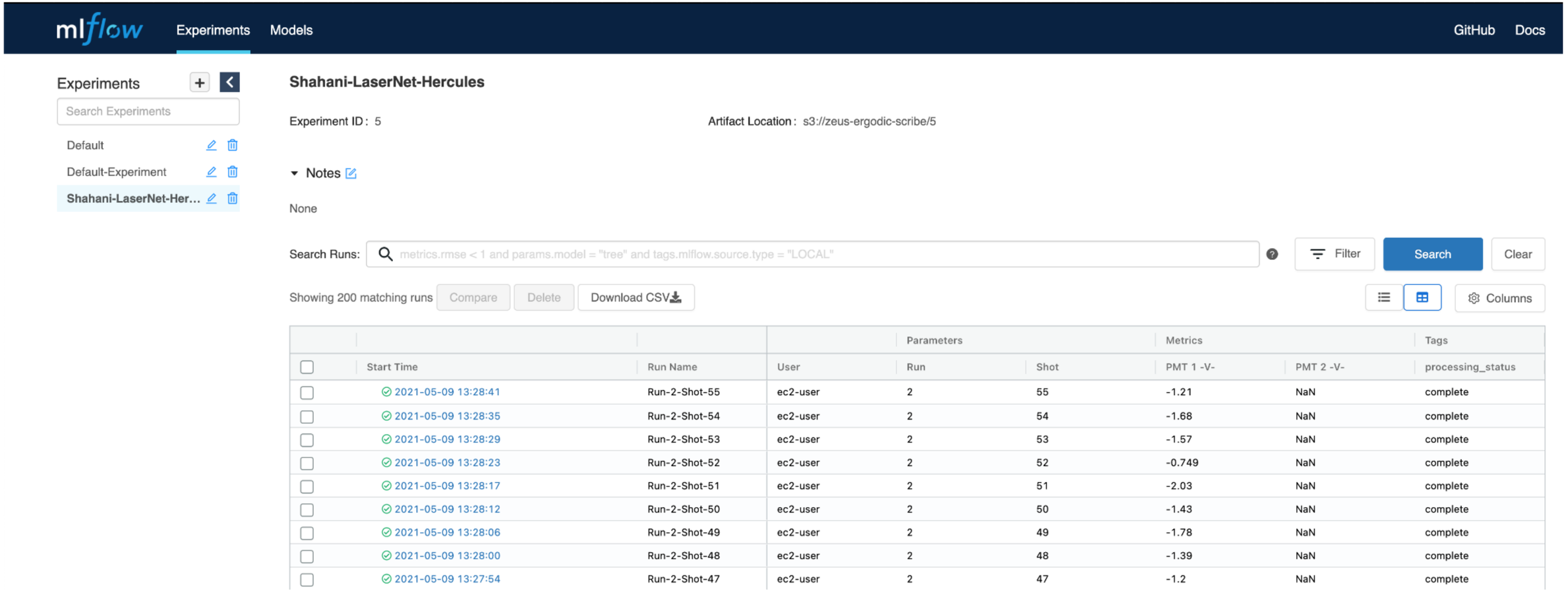
    shot_id_col_1, shot_id_col_2, shot_id_col_3 = st.beta_columns(3)

    tz = timezone("US/Eastern")
    est_tz = datetime.now(tz=tz)

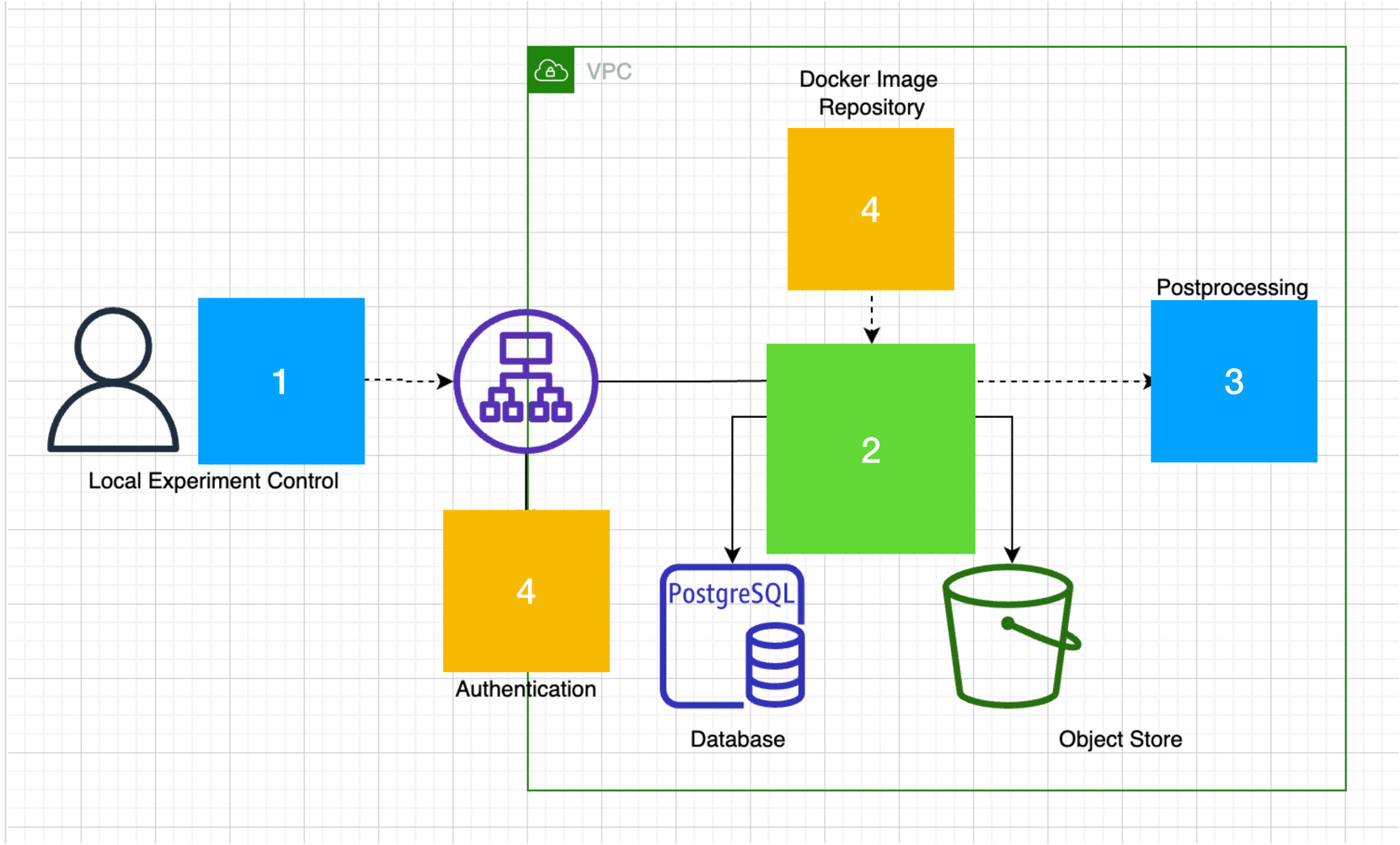
    with shot_id_col_1:
        shot_number = st.number_input("Shot Number", value=shot_num)
    with shot_id_col_2:
        shot_date = st.text_input("Shot Date", value=est_tz.strftime("%Y/%m/%d"))
    with shot_id_col_3:
        shot_time = st.text_input(
            "Shot Time", value=est_tz.strftime("%H:%M:%S")  # .%f")[:-3]
        )
```



The logged data can be accessed and visualized from a browser — This is all OSS, UI comes for free, all we do is host



How is this provisioned?



Infrastructure-as-code enables easy replication

- Easily replicated to new Cloud environments
- Enables version-control and incremental development
- 900 lines at last count
- Most specialized/unlikely skill-set in scientists <> Most unlikely to need updates

```python
def setup_fe_service(
    stack,
    vpc,
    ecr_repo,
    s3_dict,
    mlflow_tracking_uri,
    app_dns_name,
    user_pool_id,
    mlflow_public_path="",
    app_prefix="",
):

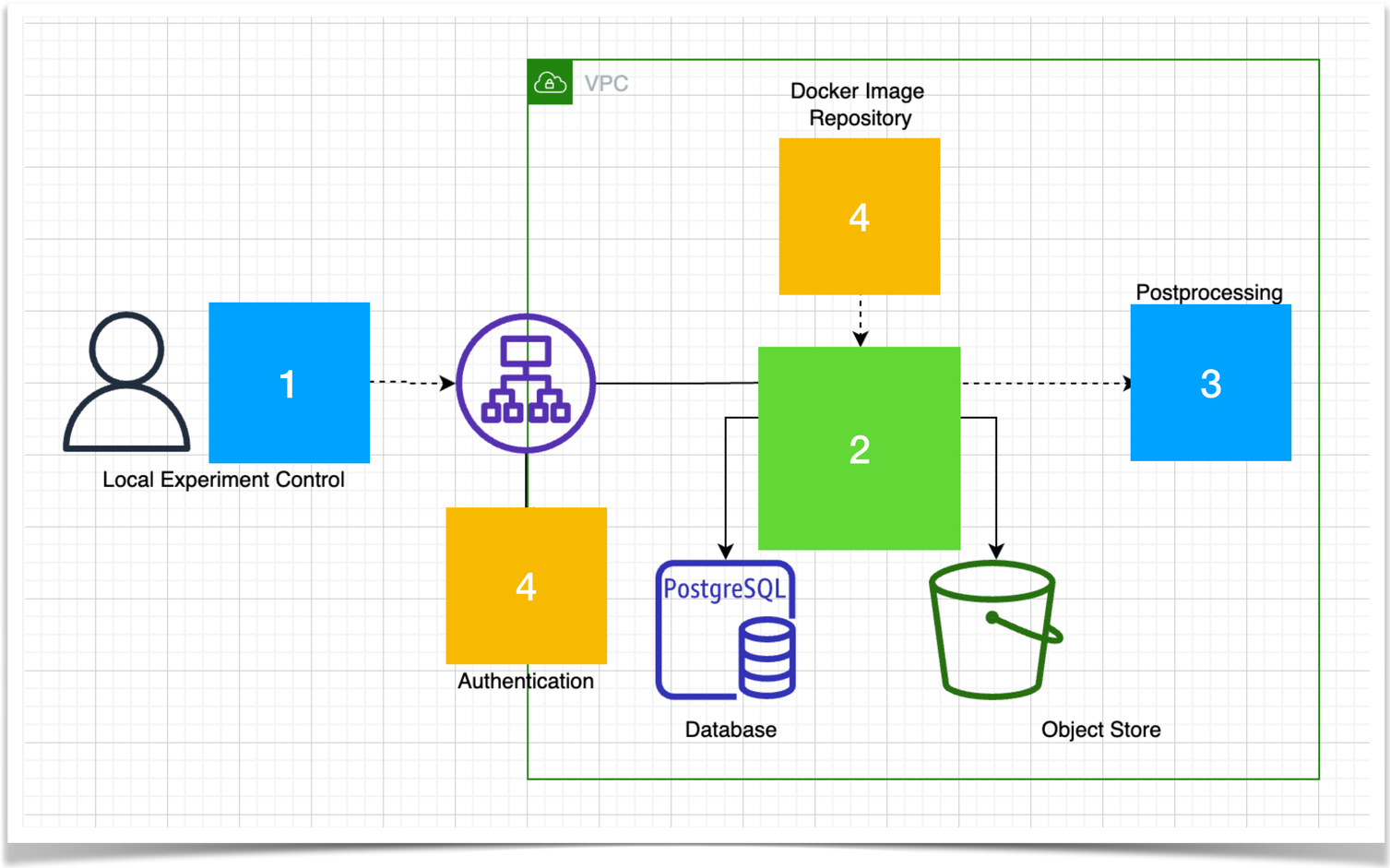
    fecluster = ecs.Cluster(stack, app_prefix + "-FECluster", vpc=vpc)
    fe_task_def = ecs.FargateTaskDefinition(
        stack, app_prefix + "-FETaskDef", memory_limit_mib=1024
    )
    fe_task_def.add_to_task_role_policy(statement=s3_dict["policy_statement"])

    fe_container = fe_task_def.add_container(
        app_prefix + "-FEContainer",
        image=ecs.ContainerImage.from_ecr_repository(
            repository=ecr_repo, tag="latest-fe"
        ),
        logging=ecs.LogDriver.aws_logs(stream_prefix=app_prefix + "-fe_log"),
        environment={
            "MLFLOW_PUBLIC_PATH": "https://" + app_dns_name + mlflow_public_path + "/",
            "MLFLOW_TRACKING_URI": mlflow_tracking_uri,
            "USER_POOL_ID": user_pool_id,
        },
    )
    fe_container.add_port_mappings(
        ecs.PortMapping(protocol=ecs.Protocol.TCP, container_port=8501)
    )

    fe_service = ecs.FargateService(
        stack,
        app_prefix + "-FEService",
        cluster=fecluster,
        vpc_subnets=ec2.SubnetSelection(subnet_type=ec2.SubnetType.PRIVATE),
        assign_public_ip=False,
        task_definition=fe_task_def,
        desired_count=1,
    )

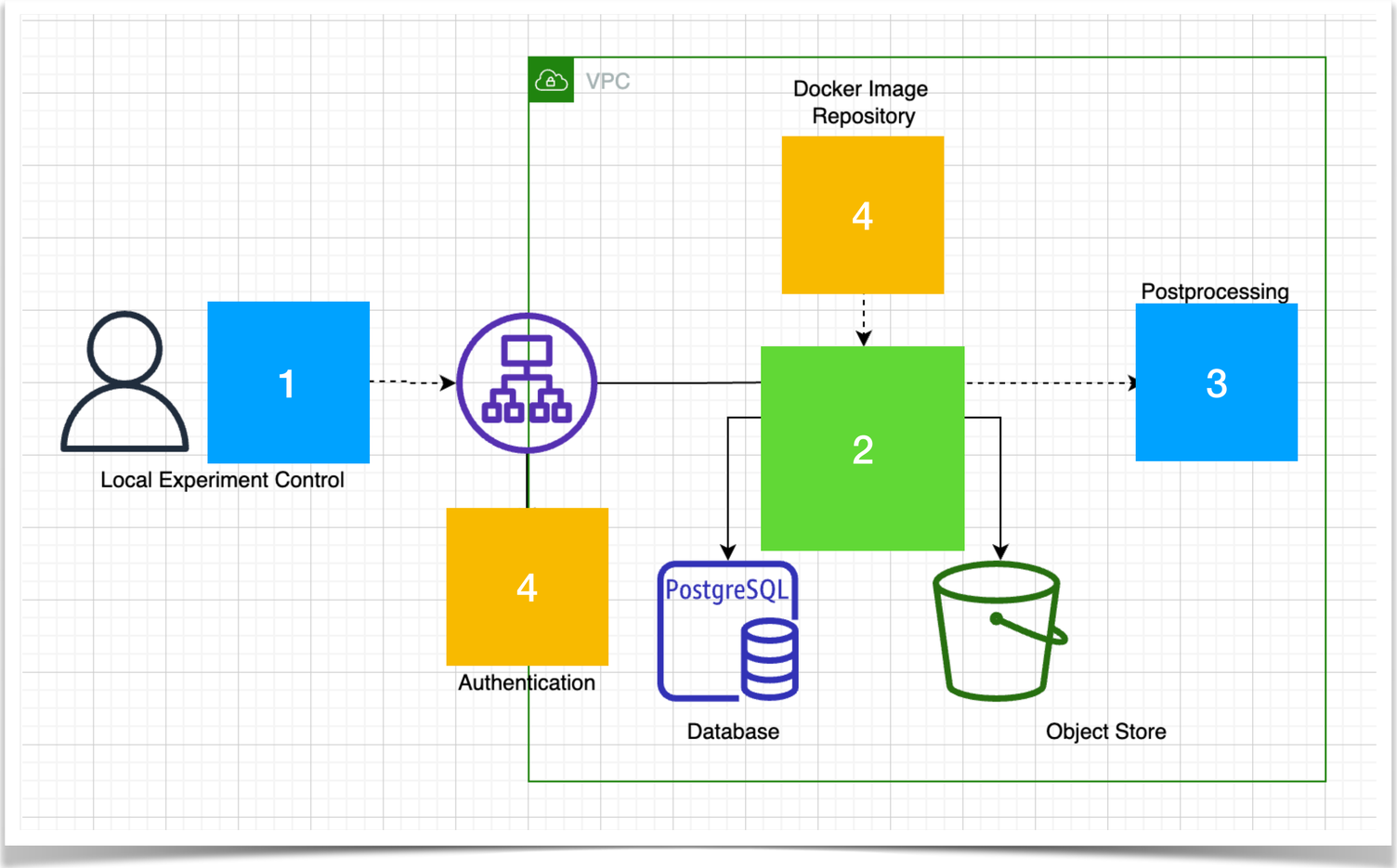
    return {"service": fe_service, "container": fe_container}
```

The workflows rely on Cloud APIs, Open Source Software, and Containers



The workflows rely on Cloud APIs, Open Source Software, and Containers

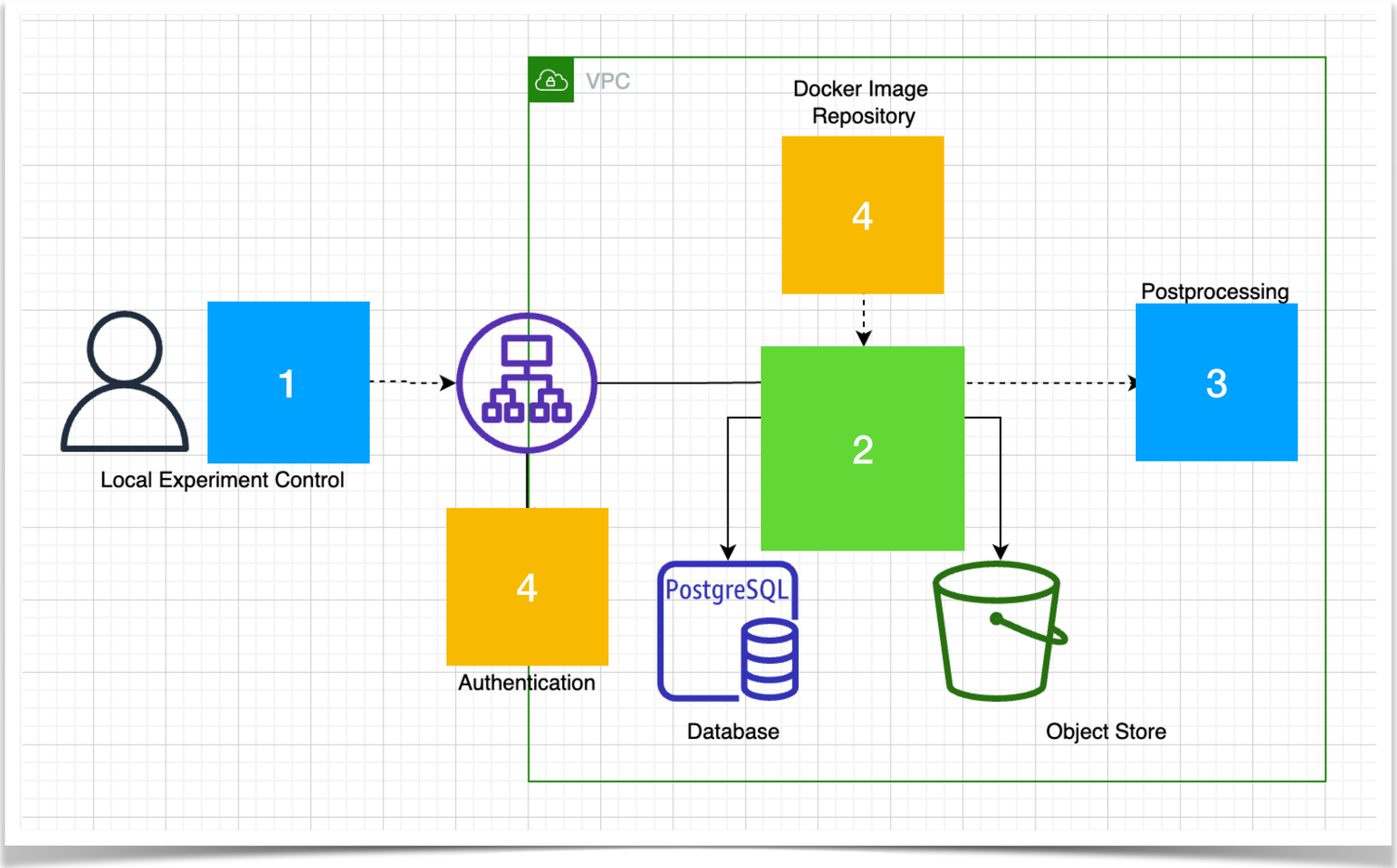
1. **Streamlit (OSS) for Frontend**



The workflows rely on Cloud APIs, Open Source Software, and Containers

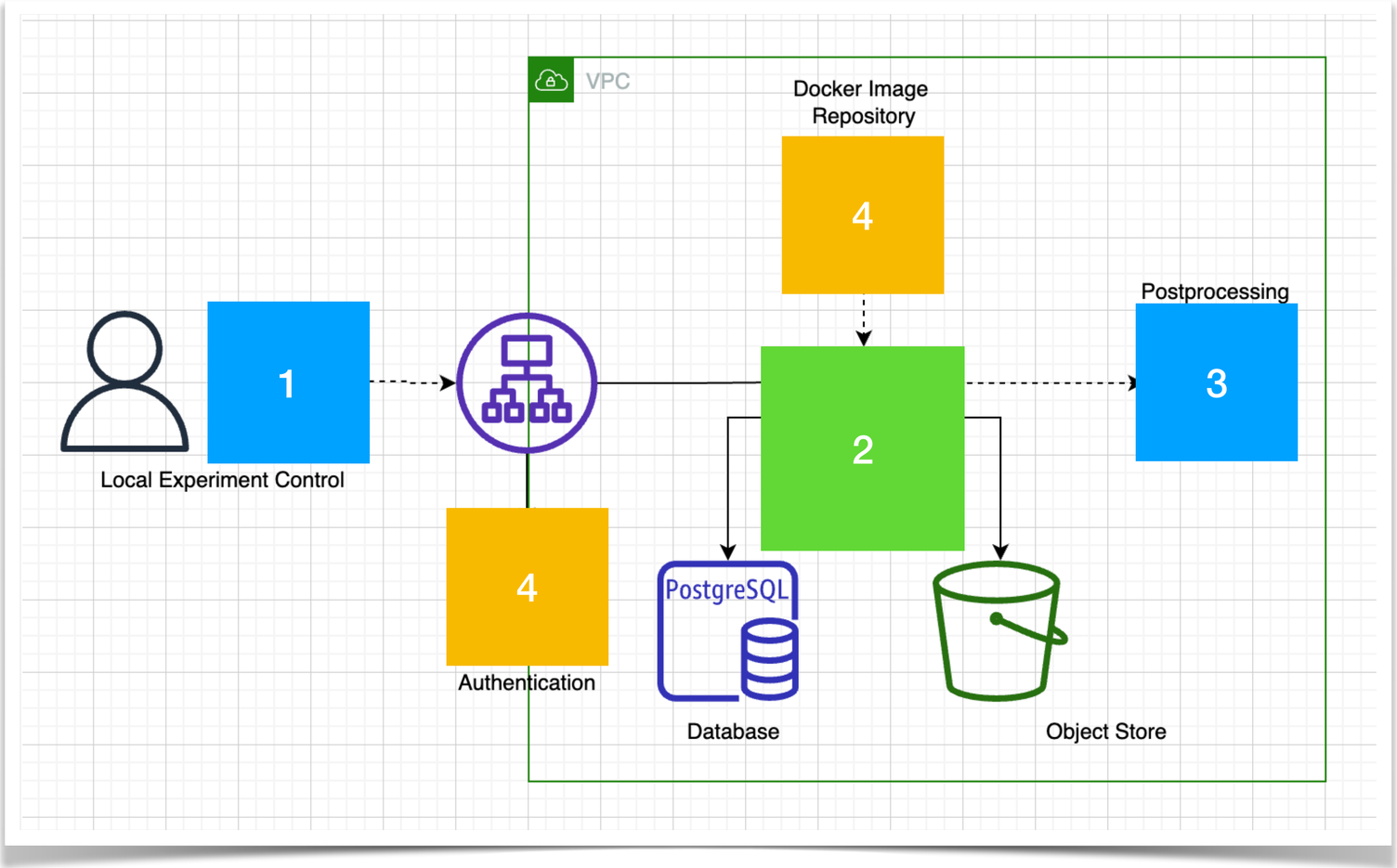
1. **Streamlit (OSS) for Frontend**

    1. **GitHub Actions can create and push containers that get consumed by AWS**



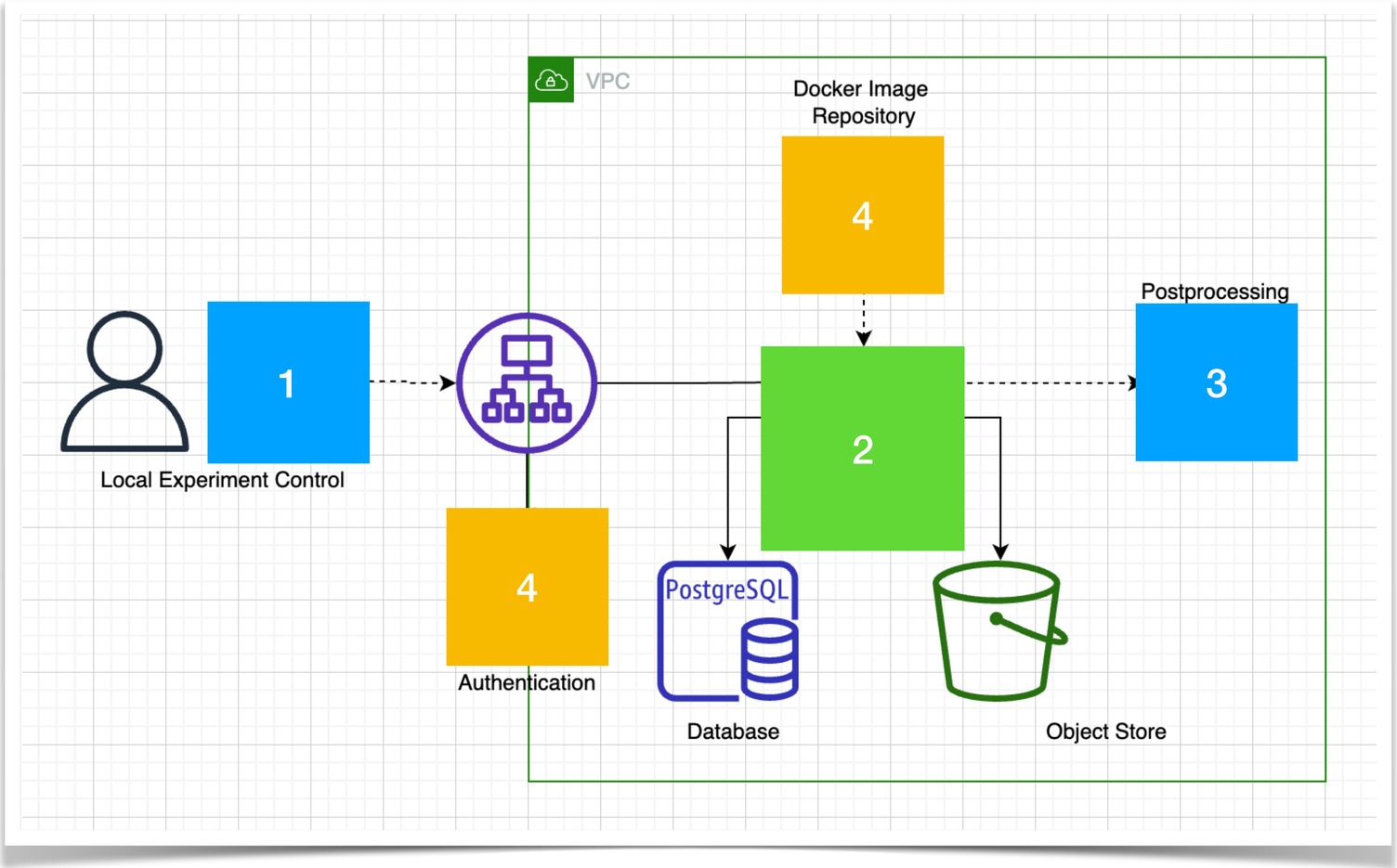
The workflows rely on Cloud APIs, Open Source Software, and Containers

1. Streamlit (OSS) for Frontend
    1. GitHub Actions can create and push containers that get consumed by AWS
2. MLFlow Server (OSS) for Data Management



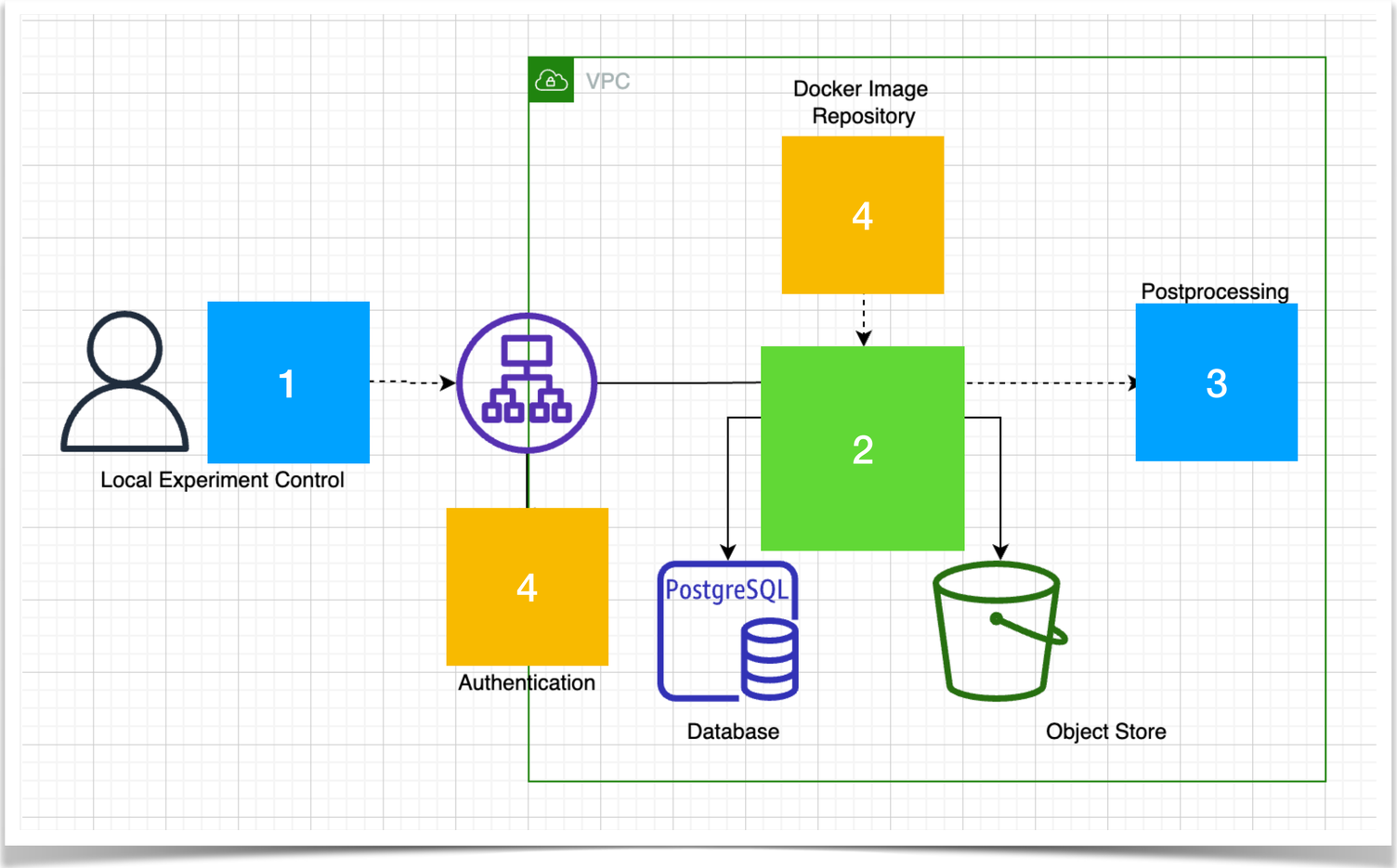
The workflows rely on Cloud APIs, Open Source Software, and Containers

1. Streamlit (OSS) for Frontend
    1. GitHub Actions can create and push containers that get consumed by AWS
2. MLFlow Server (OSS) for Data Management
3. Postprocessing code (Python)



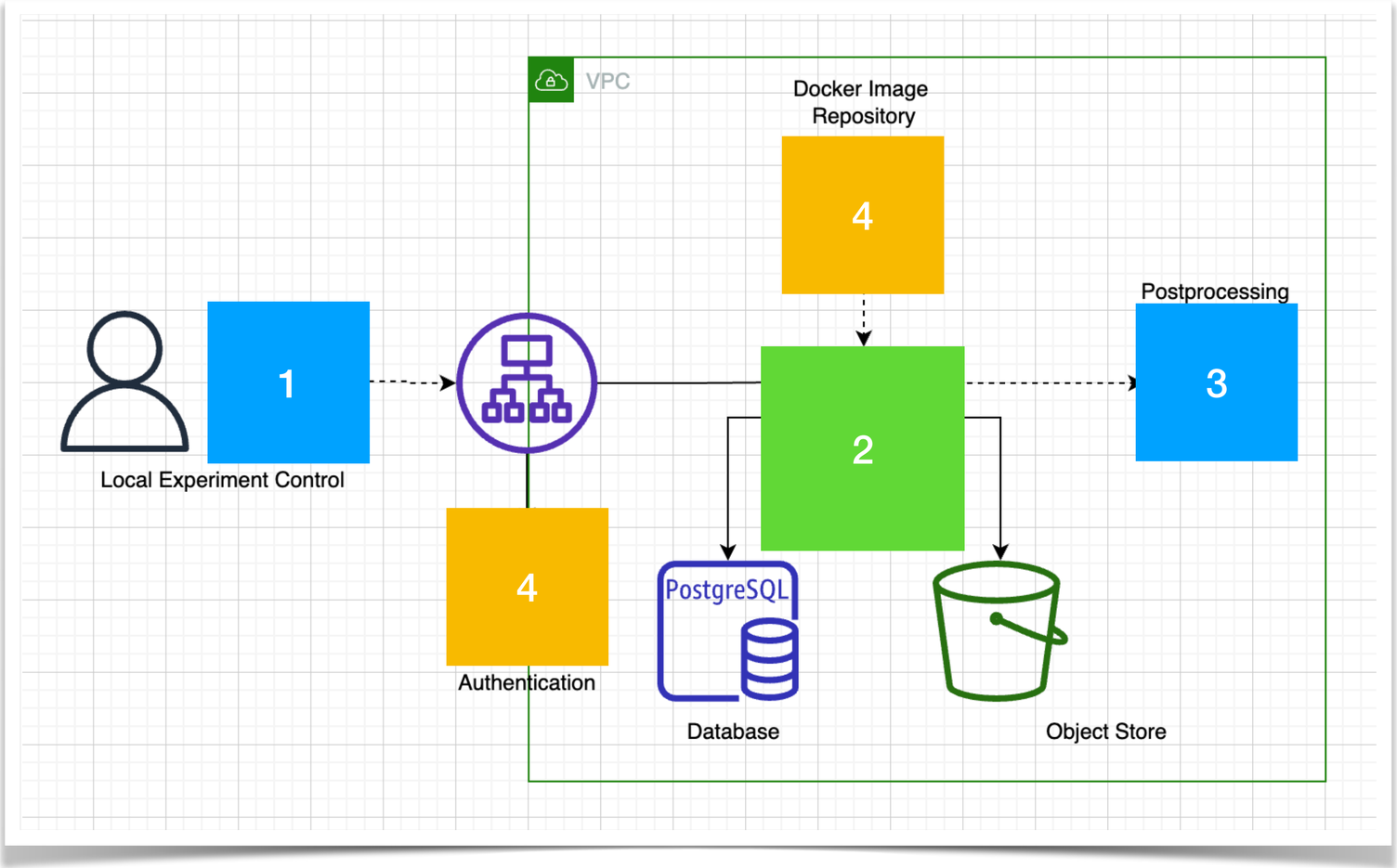
The workflows rely on Cloud APIs, Open Source Software, and Containers

1. Streamlit (OSS) for Frontend
    1. GitHub Actions can create and push containers that get consumed by AWS
2. MLFlow Server (OSS) for Data Management
3. Postprocessing code (Python)
    1. Cloud Jupyter Notebook (more secure)



The workflows rely on Cloud APIs, Open Source Software, and Containers

1. Streamlit (OSS) for Frontend
    1. GitHub Actions can create and push containers that get consumed by AWS
2. MLFlow Server (OSS) for Data Management
3. Postprocessing code (Python)
    1. Cloud Jupyter Notebook (more secure)
    2. Local machine (less secure)



The workflows rely on Cloud APIs, Open Source Software, and Containers

1. Streamlit (OSS) for Frontend
    1. GitHub Actions can create and push containers that get consumed by AWS
2. MLFlow Server (OSS) for Data Management
3. Postprocessing code (Python)
    1. Cloud Jupyter Notebook (more secure)
    2. Local machine (less secure)
4. OSS for provisioning infrastructure — database, object store, container hosts

