# Machine Learning Workflow Management Using

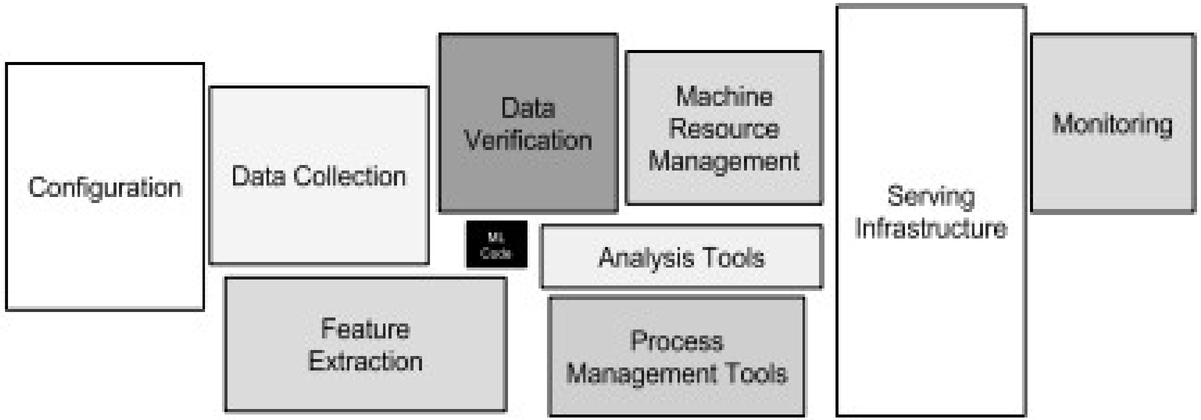
mlflow™

Shouvik Sarkar
Knowledge Associate
CDAC, Bangalore

# What is MLFlow ?

- Open source Machine Learning Operations tool for managing machine learning project lifecycle parts
- Created By DataBricks
  - alpha release in 2018
- Handed over to the Linux Foundation in 2020
- 200+ contributors as of 2020
- Used by companies like Toyota, Accenture, Microsoft and many more *

# Why MLFlow ?




Hidden Technical Debt in Machine Learning Systems, Sculley et al., 2015

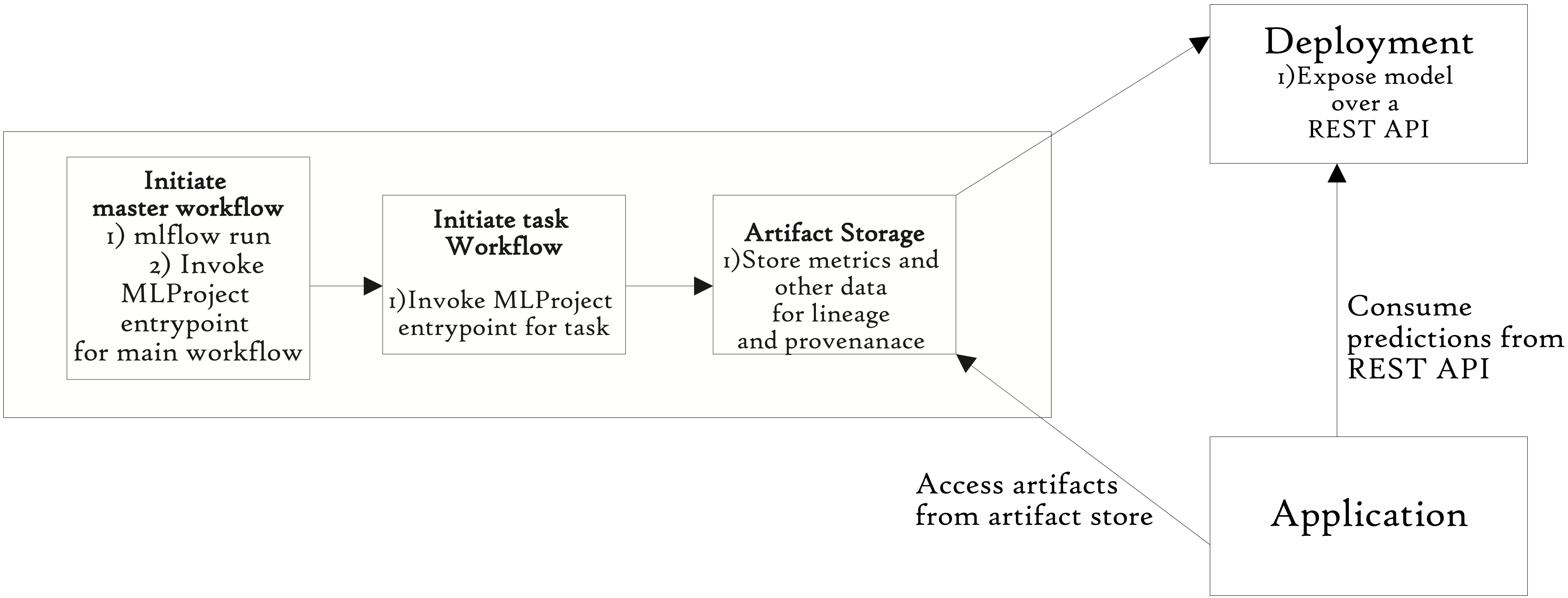
# Why MLFlow ?

- Different tools for different phases of an ML Project lifecycle
  - One stop solution for most
  - 4 components
    - MLFlow Tracking - Logging artifacts and building reproducible workflows
    - MLFlow Projects - Model governance
    - MLFlow Models - Sharing models of different flavours
    - MLFlow Registry - Model versioning and deployment
- Experiment Tracking involves a lot of boilerplate code
  - Provides segregation of experiments with varying parameters
- Reproducibility
  - Provides standard way of building pipelines and logging artifacts and parameters
- Complicated setup for deployment
  - Provides a simple way to deploy models

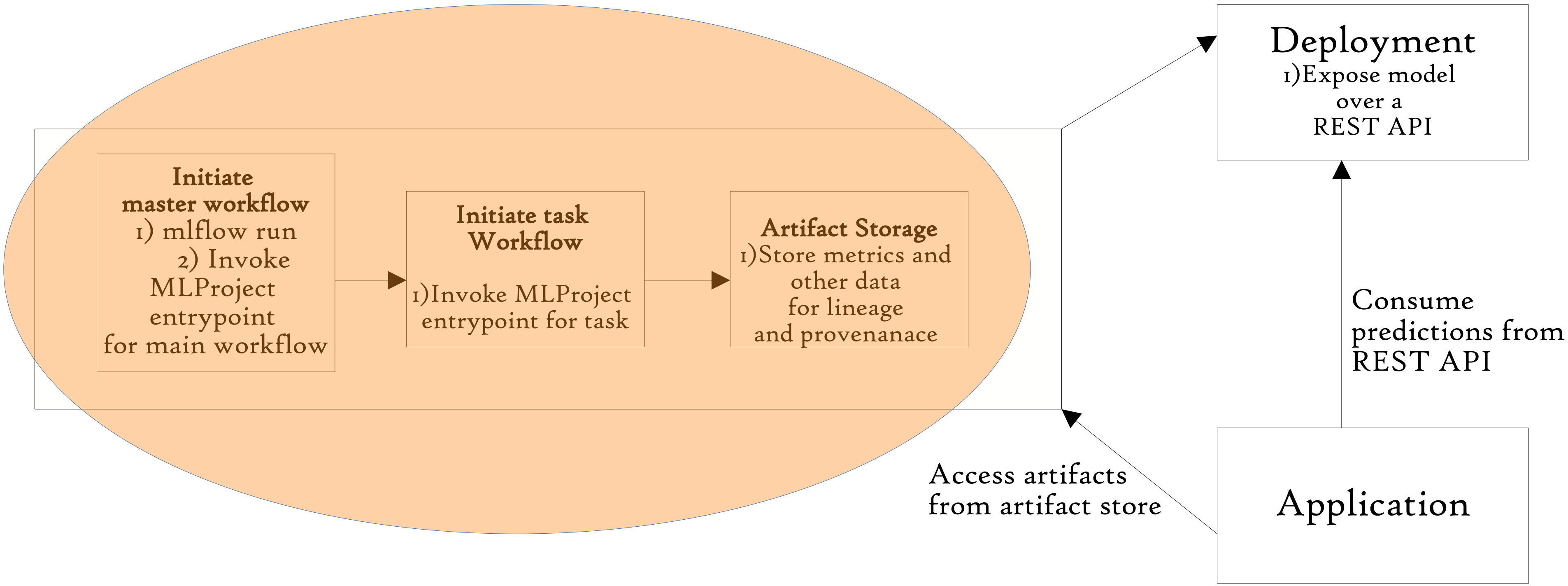
# Features

- Support for Python, Scala, Java and R
- Integrates with most of the popular machine learning libraries
- Runs on major cloud platforms

# Workflow Conceptual Overview



# Workflow Conceptual Overview



# Income Prediction

Dataset comprises
- tabular data about individuals ( age, education, marital status, etc)
- label for each comment
  - 2 labels ( <=50k, >50k )

Type of problem
- Classification
  - Classify given row to one of the given labels
- Supervised Learning Using Random Forest (Using Sklearn)

# Case Study: Cyberbullying Classification

Dataset comprises
- Comments( sentences) in English Language
- label for each comment
  - 4 labels ( Obscene, Hurtful, Insulting, Racist )

Type of problem
- Natural Language Processing
- Text Classification
  - Classify given sentence to one of the given labels
- Supervised Learning
- Deep Learning workflow (Using Tensorflow)

Hyperparameter Tuning
- Finding the optimal set of hyperparameters for the task

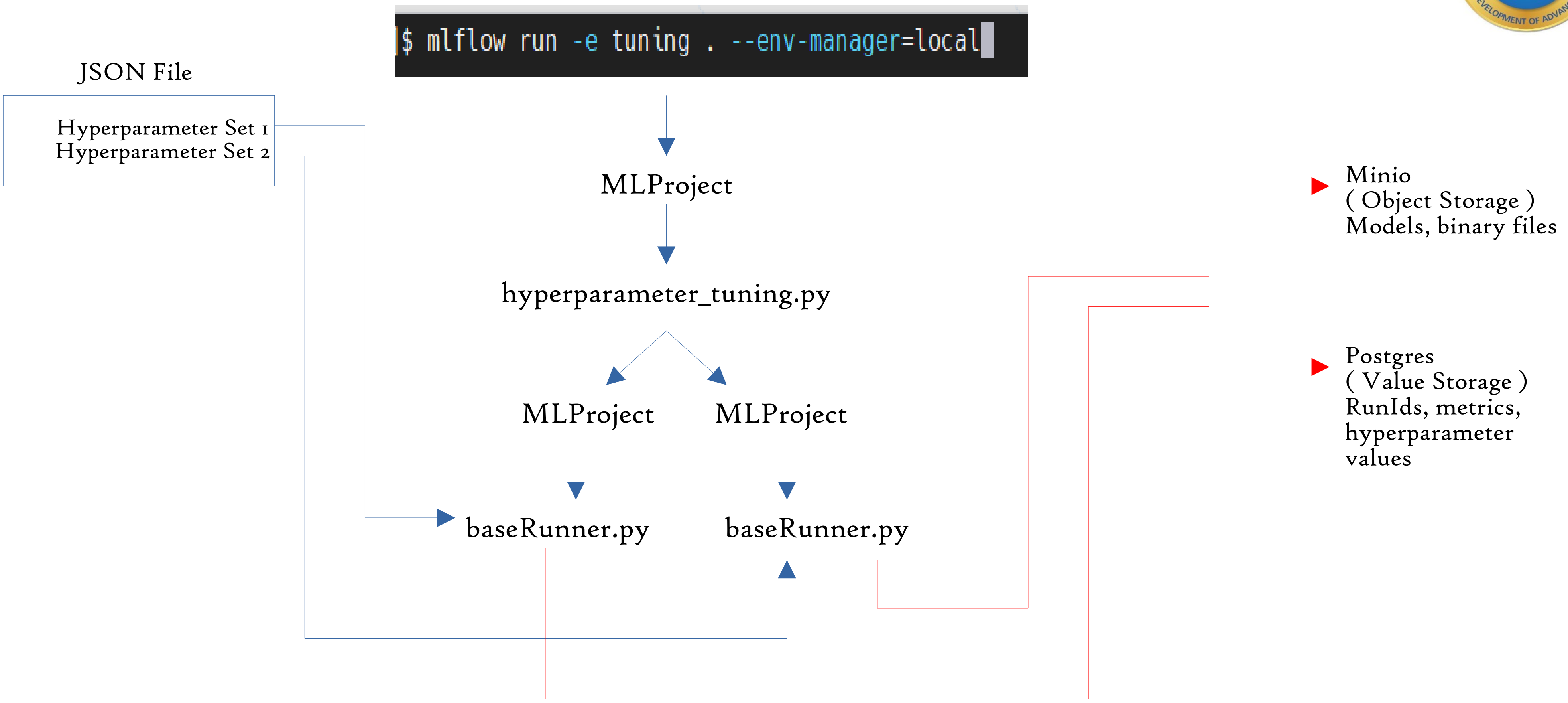
# Prerequisites

1. MLFlow (Can be installed via pip)

2. Postgres ( Relational Database Management System ) and psycopg2 ( Python connector: pip install psycopg2)

3. Minio ( Object Store )

# Model Building workflow



# Initial Setup

## Create “mlflowdb” in database

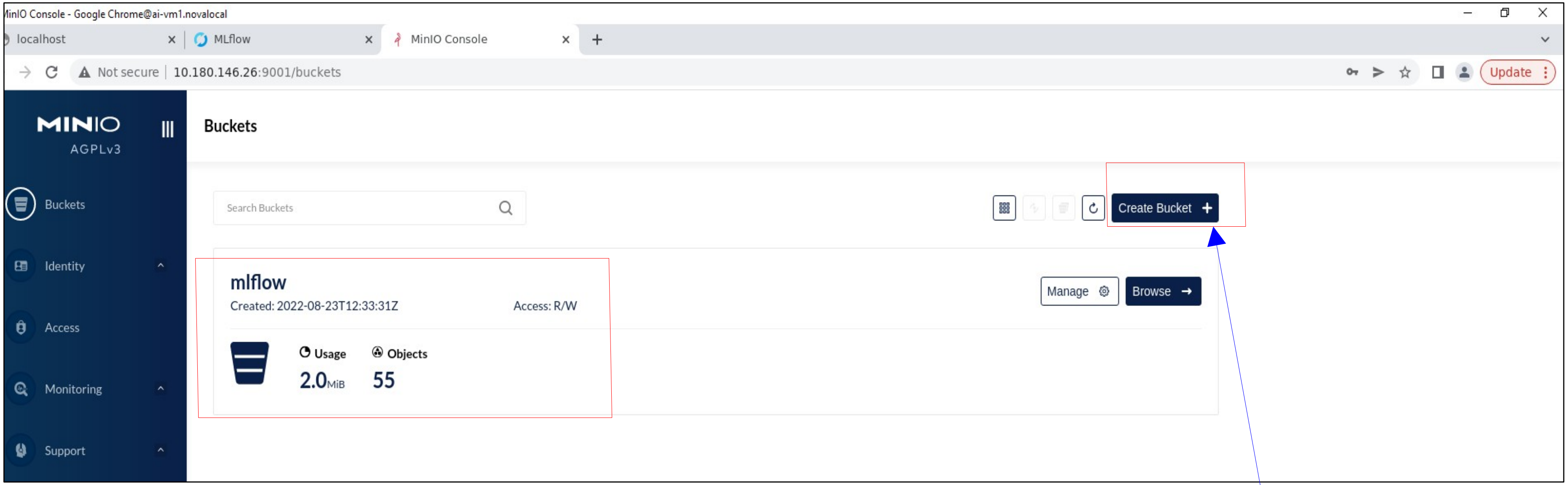
CREATE DATABASE mlflow_db;
GRANT ALL PRIVILEGES ON DATABASE mlflow_db TO postgres;

```
[centos@ai-vm1 register_model]$ psql -U postgres
Password for user postgres:
psql (9.2.24)
Type "help" for help.

postgres=# \c mlflowdb
You are now connected to database "mlflowdb" as user "postgres".
mlflowdb=# \dt
                 List of relations
 Schema |          Name          | Type  |  Owner
--------+------------------------+-------+----------
 public | alembic_version        | table | postgres
 public | experiment_tags        | table | postgres
 public | experiments            | table | postgres
 public | latest_metrics         | table | postgres
 public | metrics                | table | postgres
 public | model_version_tags     | table | postgres
 public | model_versions         | table | postgres
 public | params                 | table | postgres
 public | registered_model_tags  | table | postgres
 public | registered_models      | table | postgres
 public | runs                   | table | postgres
 public | tags                   | table | postgres
(12 rows)

mlflowdb=#
```

# Initial Setup

## Go to the download directory of Minio and start the Minio server

MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=password ./minio server /mnt/data --console-address ":9001"




Create New Bucket

# Initial Setup

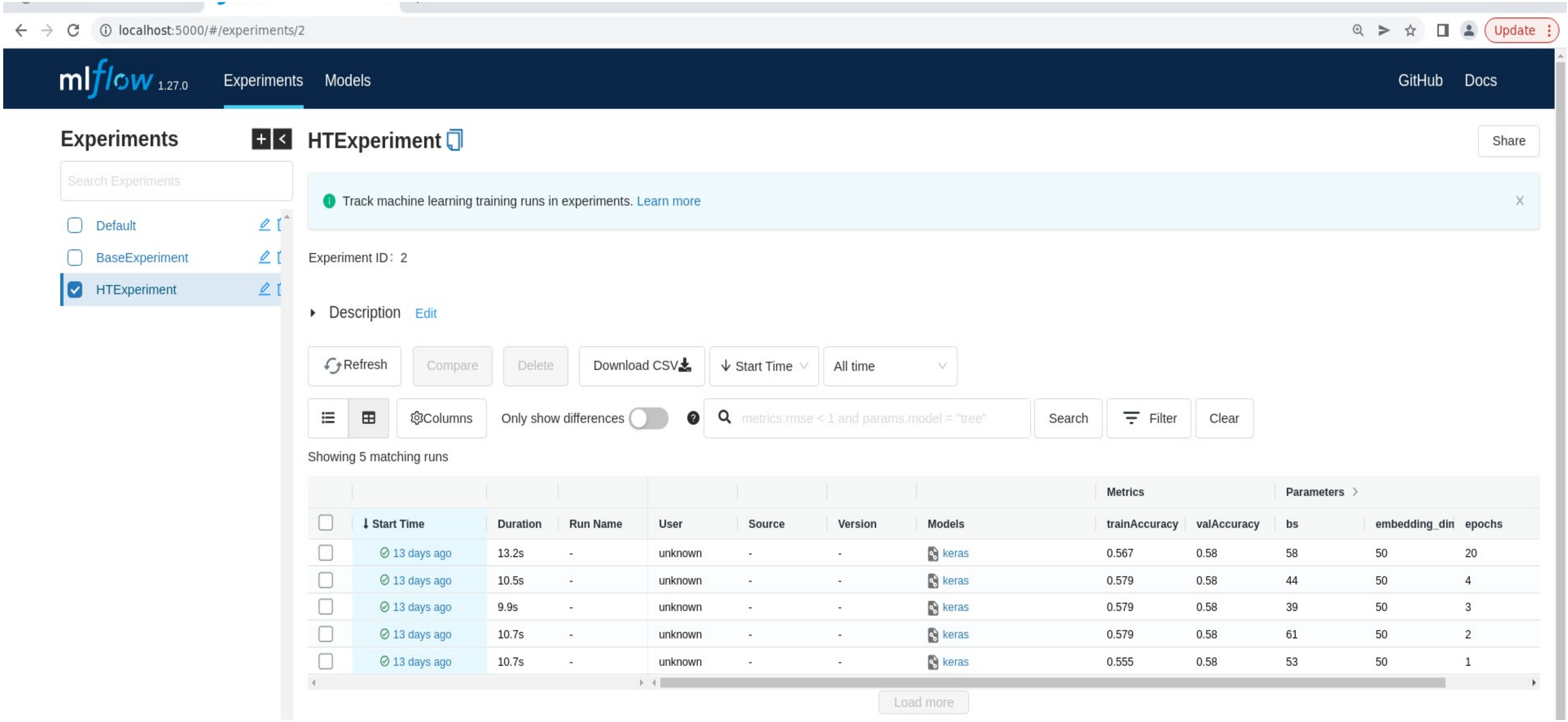
Start the tracking server

```
mlflow server \
--backend-store-uri postgresql+psycopg2://postgres:password@localhost:5432/mlflowdb \
--default-artifact-root s3://mlflow/ --host 127.0.0.1 -p 5000
```

Link the object store

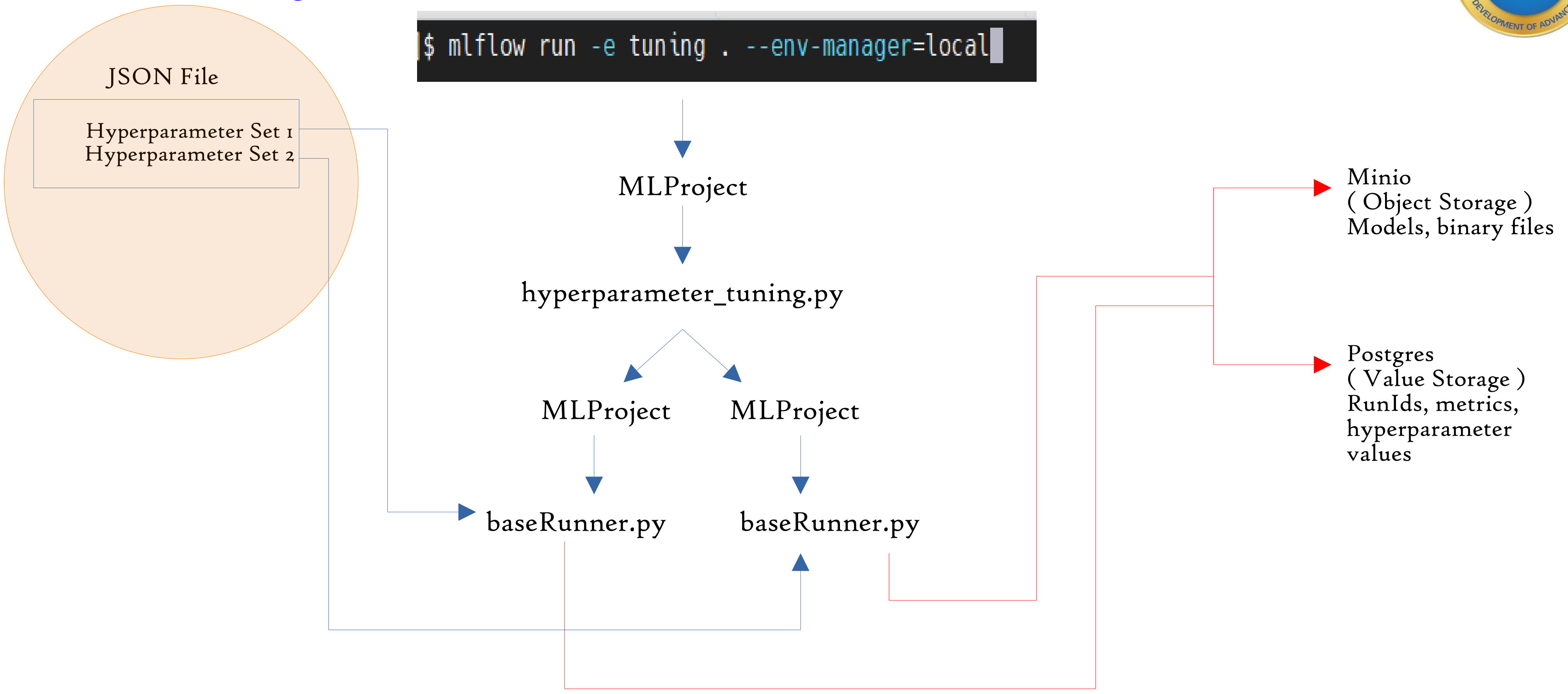
Link Postgres

# MLFlow UI



localhost:5000/#/experiments/2
Update
mlflow 1.27.0
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
BaseExperiment
HTExperiment
HTExperiment
Share
Track machine learning training runs in experiments. Learn more
Experiment ID: 2
Description
Edit
Refresh
Compare
Delete
Download CSV
Start Time
All time
Columns
Only show differences
metrics.rmse < 1 and params.model = "tree"
Search
Filter
Clear
Showing 5 matching runs
Metrics
Parameters
Start Time
Duration
Run Name
User
Source
Version
Models
trainAccuracy
valAccuracy
bs
embedding_din
epochs
13 days ago 13.2s - unknown - - keras 0.567 0.58 58 50 20
13 days ago 10.5s - unknown - - keras 0.579 0.58 44 50 4
13 days ago 9.9s - unknown - - keras 0.579 0.58 39 50 3
13 days ago 10.7s - unknown - - keras 0.579 0.58 61 50 2
13 days ago 10.7s - unknown - - keras 0.555 0.58 53 50 1
Load more

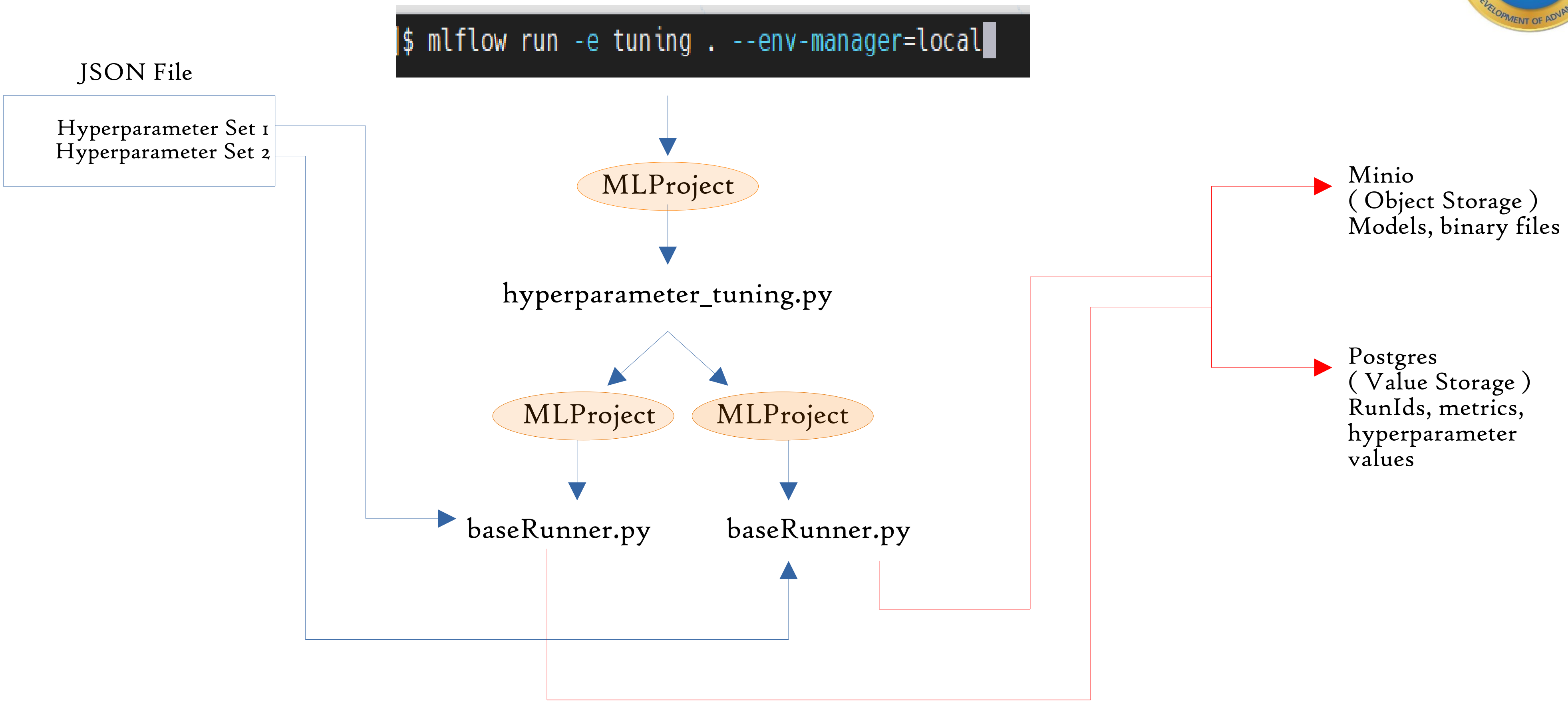
# Model Building workflow



# JSON file


```
[
    {
        “learning_rate”: 0.266,
        “dataPath”: “./”,
        “max_length”: 50,
        “vocab_length”: 100,
        “seq_padding_style”: “post”,
        “seq_truncating_style”: “post”,
        “embedding_dim”: 50,
        “bs”: 53,
        “epochs”: 3
    },
    {
        “learning_rate”: 0.45,
        “dataPath”: “./”,
        “max_length”: 40,
        “vocab_length”: 100,
        “seq_padding_style”: “post”,
        “seq_truncating_style”: “post”,
        “embedding_dim”: 50,
        “bs”: 46,
        “epochs”: 7
    }
]
```

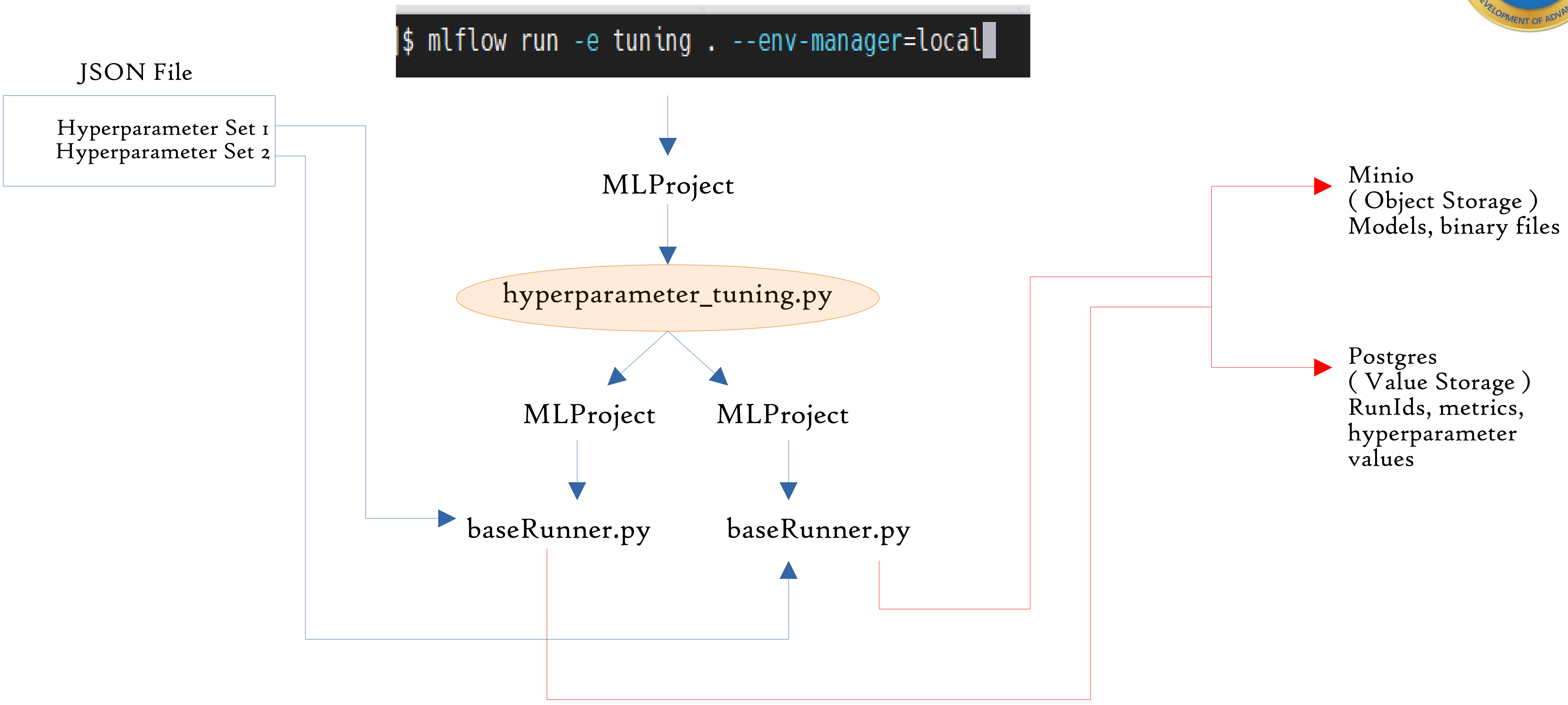
# Model Building workflow



# MLProject file

```
name: tensorflow-example
entry_points:
    tuning:
        command: "python hyperparameter_tuning.py"
    ht:
        parameters:
            learning_rate: {type: float, default: 0.1}
            vocab_length: {type: int, default: 100}
            seq_padding_style: {type: string, default: “post”}
            seq_truncating_style: {type: string, default: “post”}
            embedding_dim: {type: int, deafault:100}
            bs: {type: int, default: 64}
            epochs: {type: int, default: 5}
            max_length: {type: int, default:50}
            run_id: {type: string }

        command: “python baseRunner.py \
            --learning-rate {learning_rate} \
            --vocab-length {vocab_length} \
            --seq-padding_style {seq_padding_style} \
            --seq-truncating-style {seq_truncating_style} \
            --embedding_dim {embedding_dim} \
            --bs {bs} \
            --epochs {epochs} \
            --max-length {max_length} \
            --run-id {run_id}”
```

# Model Building workflow



# Kicking off the main workflow ( hyperparameter_tuning.py)

Load data from JSON file

Create experiment for main workflow (We have multiple runs for the same experiment )

```
mlflow.create_experiment(...)
```

Kick off multiple runs for the experiment

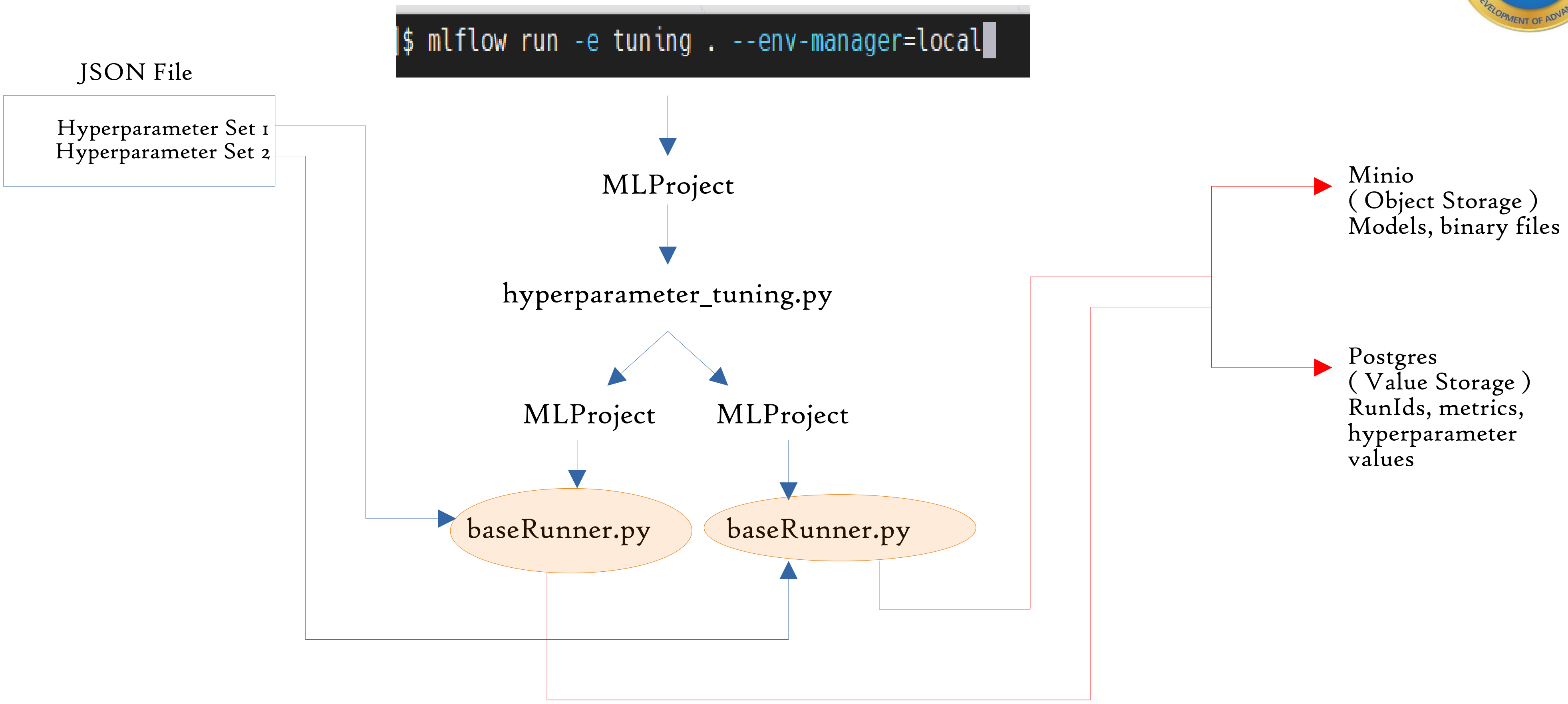
- The runs are created by invoking the “ht” entry point in MLProject

```
client = MlflowClient()
run = client.create_run(exp_id...)
mlflow.run(run_id = run.info.run_id, uri = '.', entry_point = "ht", use_conda = False ...)
```

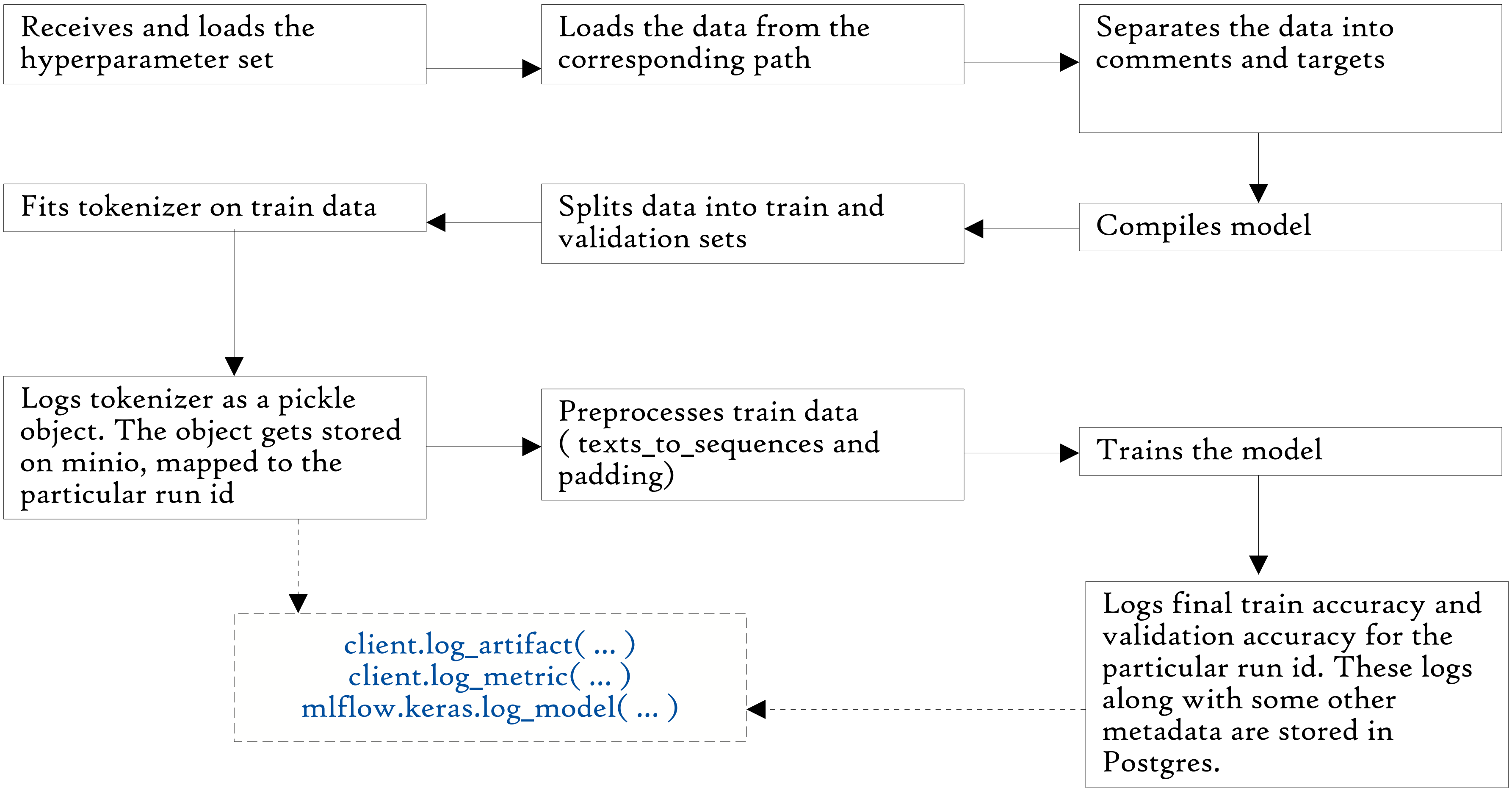
- Each run works on a different set of hyperparameters

- Each run executes an instance of baseRunner.py

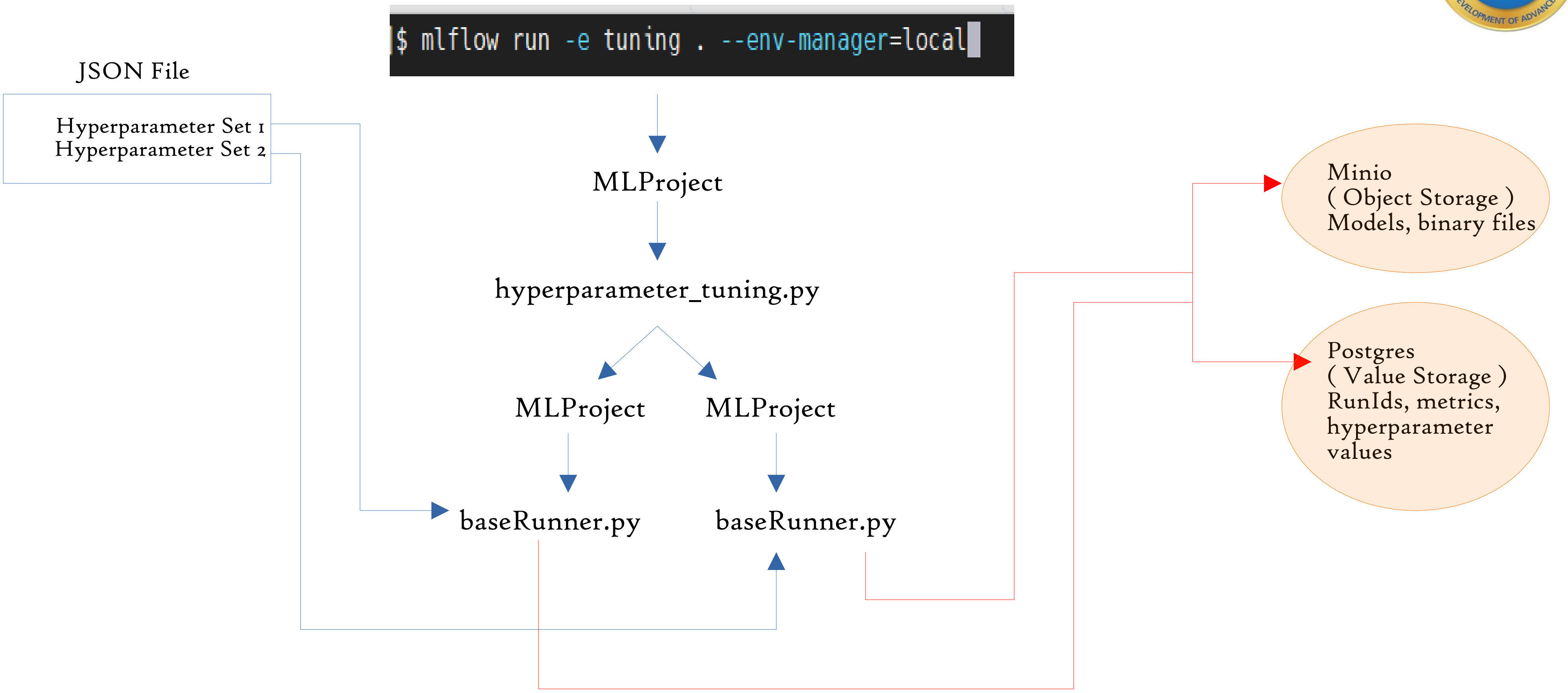
# Model Building workflow



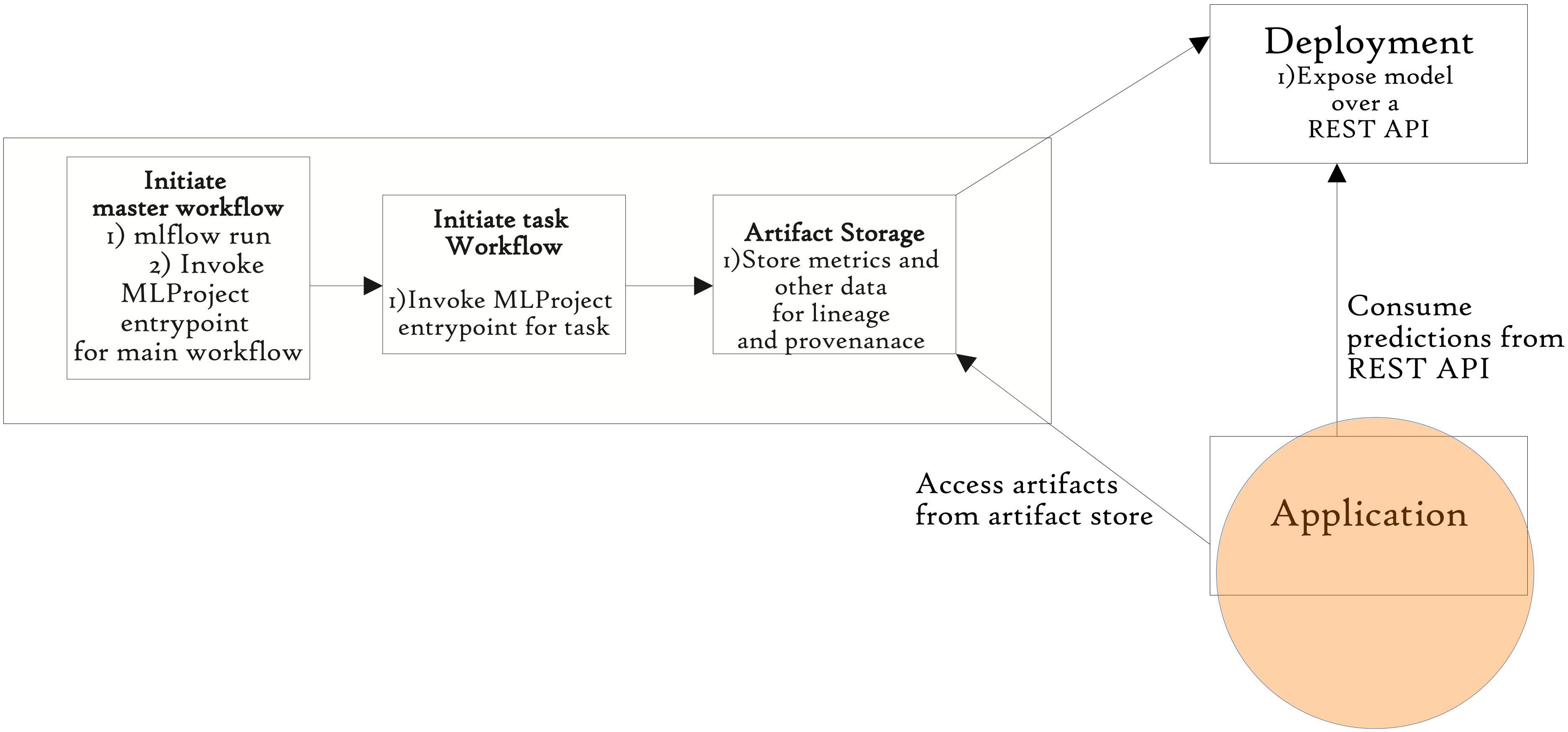
# Text Classification workflow with MLFlow (baseRunner.py)



# Model Building workflow



# Workflow Conceptual Overview



# Inference By Retrieval workflow (infer.py)

1. Iterate over all run ids

exp = client.get_experiment_by_name(...)

runs = mlflow.list_run_infos(exp.experiment_id)

Select the run id with the best metric value ( training accuracy )

( MLFlow provides methods and attributes to access runs and their corresponding artifacts )

Load tokenizer.pkl ( required for preprocessing)

client.download_artifacts( ... )

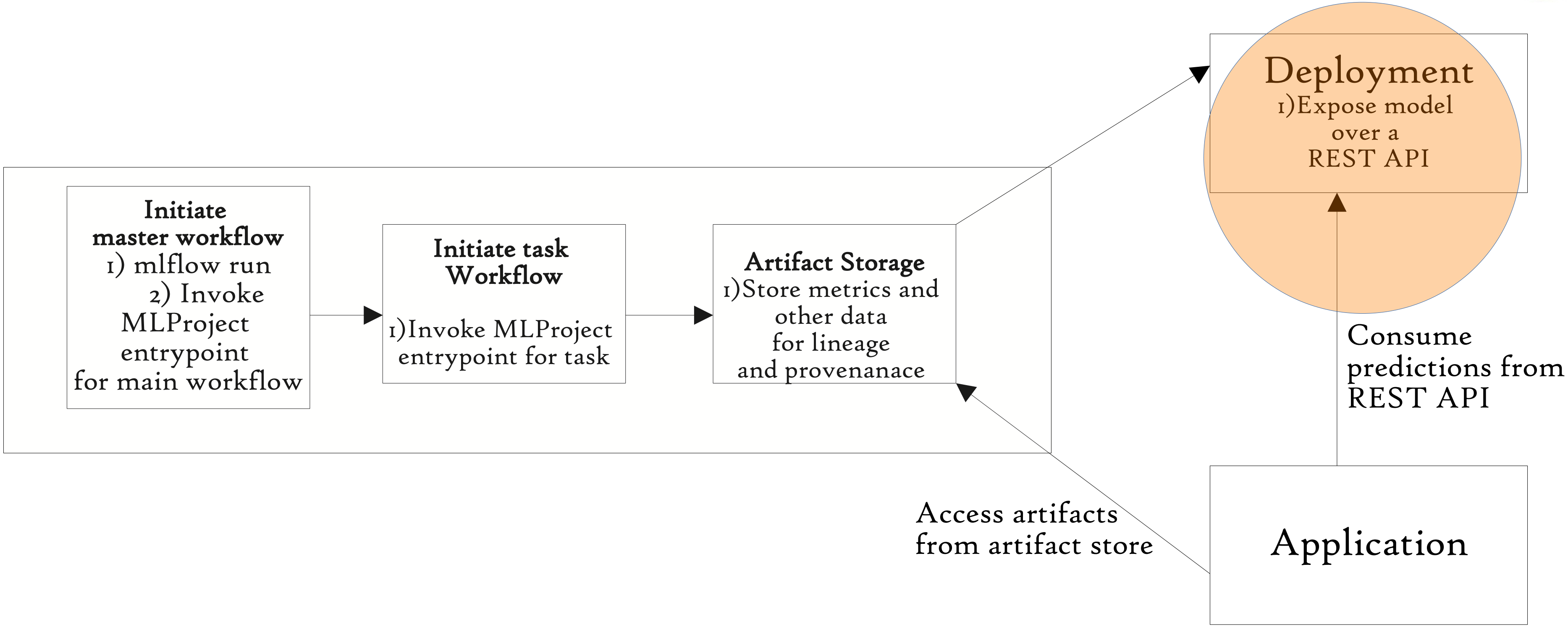
mlflow.keras.load_model(...)

Load model

Perform preprocessing using loaded tokenizer.pkl

Perform inference using loaded model

# Workflow Conceptual Overview



# Inference By Request workflow

## Model Registration

Run register_model.py

1. Iterate over all run ids

2. Select the run id with the best metric value ( training accuracy )

3. Push the corresponding model to registry

## Model Staging

Run stagemodel.py

1. Change version of model to Staging

## Model Deployment

1. Start “MLFlow Serving” server

This exposes the specified model over a RESTful Interface

## Inference Request

Run getResults.py

1. Retrieve run id corresponding to registry model
2. Retrieve tokenizer.pkl for preprocessing for the corresponding run id for preprocessing
3.Preprocess and send data to API by invoking a POST request and get back prediction

# Start MLFlow Serving

## Set environment variables for deployment

```
export MLFLOW_TRACKING_URI=postgresql+psycopg2://postgres:password@localhost:5432/braintumor
export MLFLOW_S3_ENDPOINT_URL=http://10.180.146.26:9000
export AWS_ACCESS_KEY_ID=admin
export AWS_SECRET_ACCESS_KEY=password
```

## Start Deployment server

```
mlflow models serve -m "models:/alpha/Staging" -h 127.0.0.1 -p 5004 --env-manager=local
```

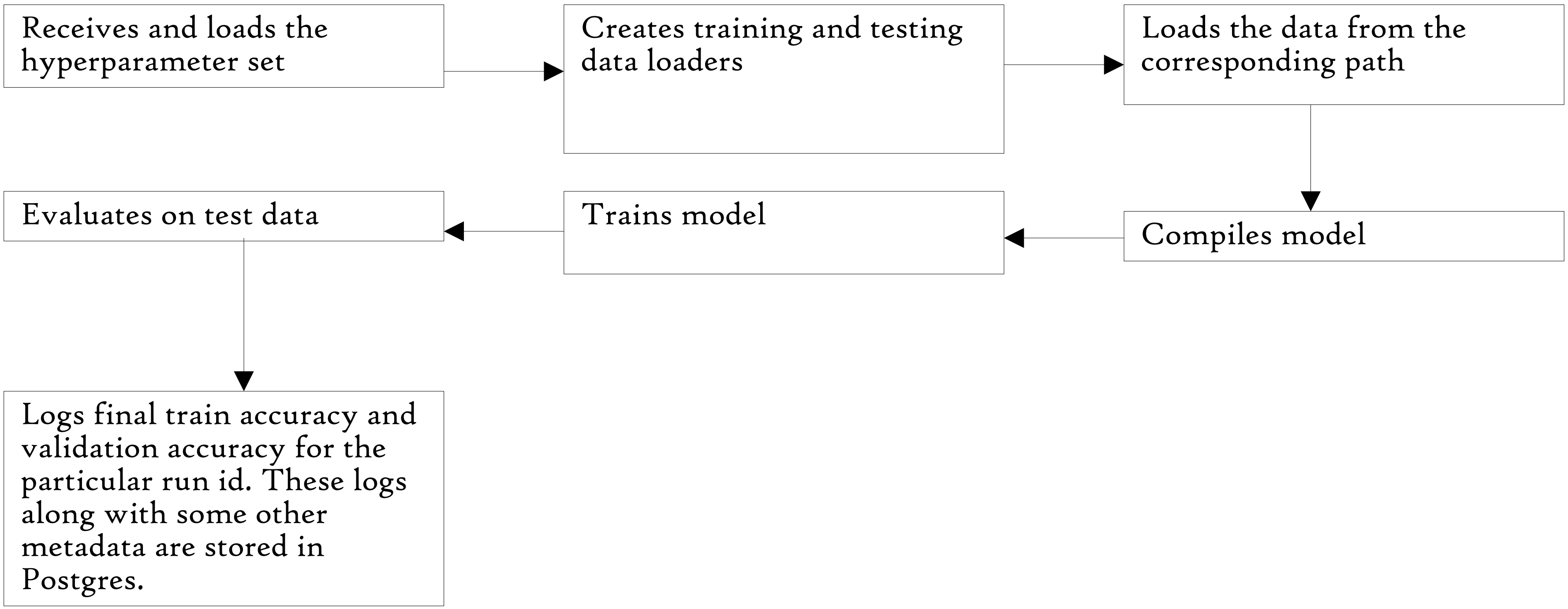
# Case Study: Brain Tumor Classification

Dataset comprises
- Images of Brain CT scans
- label for each comment
    - 2 labels ( Yes, No )

Type of problem
- Computer Vision
- Image Classification
    - Classify given image to one of the given labels
- Supervised Learning
- Deep Learning (Using Tensorflow)

Hyperparameter Tuning
- Finding the optimal set of hyperparameters for the task

# Image Classification workflow with MLFlow (baseRunnerVision.py)



## Start the tracking server

```
mlflow server \
--backend-store-uri postgresql+psycopg2://postgres:password@localhost:5432/braintumor \
--default-artifact-root s3://braintumor/ --host 127.0.0.1 -p 5000
```

## Start MLFlow Serving

### Set environment variables for deployment

```
export MLFLOW_TRACKING_URI=postgresql+psycopg2://postgres:password@localhost:5432/
braintumor
export MLFLOW_S3_ENDPOINT_URL=http://10.180.146.26:9000
export AWS_ACCESS_KEY_ID=admin
export AWS_SECRET_ACCESS_KEY=password
```

### Start Deployment server

```
mlflow models serve -m "models:/alphabt/Staging" -h 127.0.0.1 -p 5004 --env-manager=local
```

# Execution

## 1. Start tracking server and minio server

mlflow server --backend-store-uri postgresql+psycopg2://postgres:password@localhost:5432/mlflowdb --default-artifact-root s3://mlflow/ --host 127.0.0.1 -p 5000

Can access at localhost:5000 in browser

MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=password ./minio server /mnt/data --console-address ":9001"

Can access at localhost:9000 in Browser

## 2. Model Building workflow

mlflow run -e tuning . --env-manager=local

## 3. Inference By Retrieval workflow

python infer.py

## 4. Model Registration

python registermodel.py

## 5. Model Staging

python stagemodel.py

# Execution

## 6. Model Deployment

```
export MLFLOW_TRACKING_URI=postgresql+psycopg2://postgres:password@localhost:5432/mlflowdb
export MLFLOW_S3_ENDPOINT_URL=http://10.180.146.26:9000
export AWS_ACCESS_KEY_ID=admin
export AWS_SECRET_ACCESS_KEY=password
mlflow models serve -m "models:/alpha/Staging" -h 127.0.0.1 -p 5004 --env-manager=local
```

## 7. Inference Request

```
python getResults.py
```

## 8. Delete Postgres data

Create a database “mlflowdb” in Postgres

```
psql -U postgres -h 127.0.0.1 -d mlflowdb -f deletetables.sql
```

# References

https://www.databricks.com/wp-content/uploads/2020/12/LP_2-primary-asset_standardizing-the-ml-lifecycle-ebook-databricks-0626120-v8.pdf

# Thank You