**Model Serving Documentation**

A list of model serving tools here : <https://neptune.ai/blog/ml-model-serving-best-tools>

3 commonly used tools:

**Flask API**

* a lightweight web framework that provides basic tools for building RESTful APIs

**BentoML**

* a higher-level framework that provides a unified API for defining, packaging, and deploying machine learning models as API services

**MLflow**

* serve models via a built-in Models server or a Model Registry, which provide flexible options for deploying and serving models in various environments and platforms

**Flask API**

1. Define the model serving task in Airflow: You can **create a new task in Airflow that is responsible for deploying the model**. This task can be triggered by another task that performs the model training.

<https://betterdatascience.com/apache-airflow-rest-api/>

<https://www.youtube.com/watch?v=u7djG5E97tg>

1. Choose a model serving solution: Depending on your use case, you can choose a model serving solution that meets your requirements. For example, if you want to deploy the model as a REST API, you can use a **framework like Flask or FastAPI to build the API endpoints.**

<https://pythonbasics.org/flask-rest-api/>

<https://medium.com/@8B_EC/tutorial-serving-machine-learning-models-with-fastapi-in-python-c1a27319c459>

1. Deploy the model serving solution: Once you have chosen a solution, you can use Airflow to deploy it to your production environment. This may involve **spinning up new servers, containers, or functions** to host the model.

<https://www.analyticsvidhya.com/blog/2022/07/airflow-for-orchestrating-rest-api-applications/>

1. Test the model serving solution: Once the model serving solution is deployed, you can test it to ensure that it is working correctly. This may involve **sending test data to the API endpoints** and verifying that the model is returning accurate predictions.

**BentoML**

<https://www.bentoml.com/>

A picture containing text, screenshot, diagram, parallel

Description automatically generated

A BentoML’s project structure would like this one.

There is a very detailed setup tutorial: <https://docs.bentoml.org/en/latest/tutorial.html>

It can be done with colab/docker/local, just follow this one from dockerA screenshot of a computer

Description automatically generated with low confidence

main steps

1. save your trained models with BentoML API in its model store (a local directory managed by BentoML)
2. define a service.py
3. using models in service
4. Once the service definition is finalized, you can start build a bento
5. Generate docker image

What also could be helpful:

* A bunch of bento MML examples others deployed and served: <https://github.com/bentoml/BentoML/tree/main/examples>, so it is workable for us theoretically

**MLflow**

1. Train the machine learning model using a machine learning library such as Scikit-learn, TensorFlow, or PyTorch, and log the training results and artifacts to an MLflow tracking server.
2. Register the trained model as an MLflow model in the MLflow model registry, specifying the model name, version, and metadata.
3. Serve the registered model using the MLflow Models server, which provides a REST API for serving models via HTTP requests.
4. Wrap the MLflow Models server code into a Python function or module that can be executed as an Airflow task.
5. Define an Airflow DAG that includes the model serving task, along with any other tasks in the data pipeline.

Related links:

<https://towardsdatascience.com/mlflow-model-serving-bcd936d59052>

<https://mlflow.org/docs/latest/model-registry.html>

<https://mlflow.org/docs/latest/models.html#model-api>

<https://www.youtube.com/watch?v=A1NERf_8wwA>

<https://medium.com/@gyani91/serving-a-model-using-mlflow-8ba5db0a26c0>