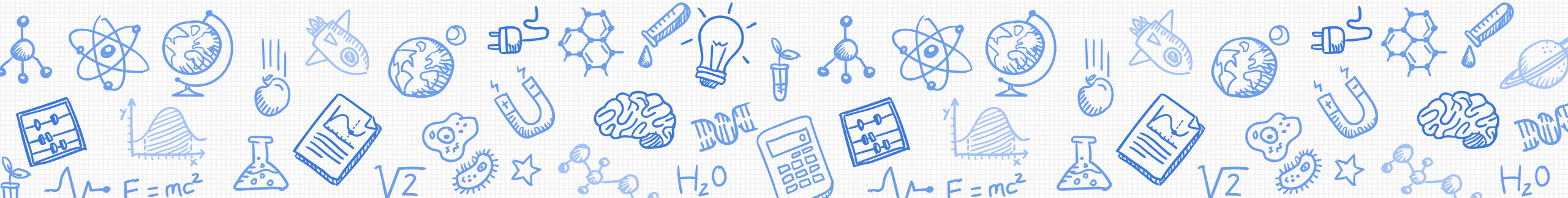
By: Capdev & MLOps Engineering

Introduction to API for Model Deployment

Main Frameworks Discussed:

Flask



Outline of Workshop

Part 1: Overview of APIs

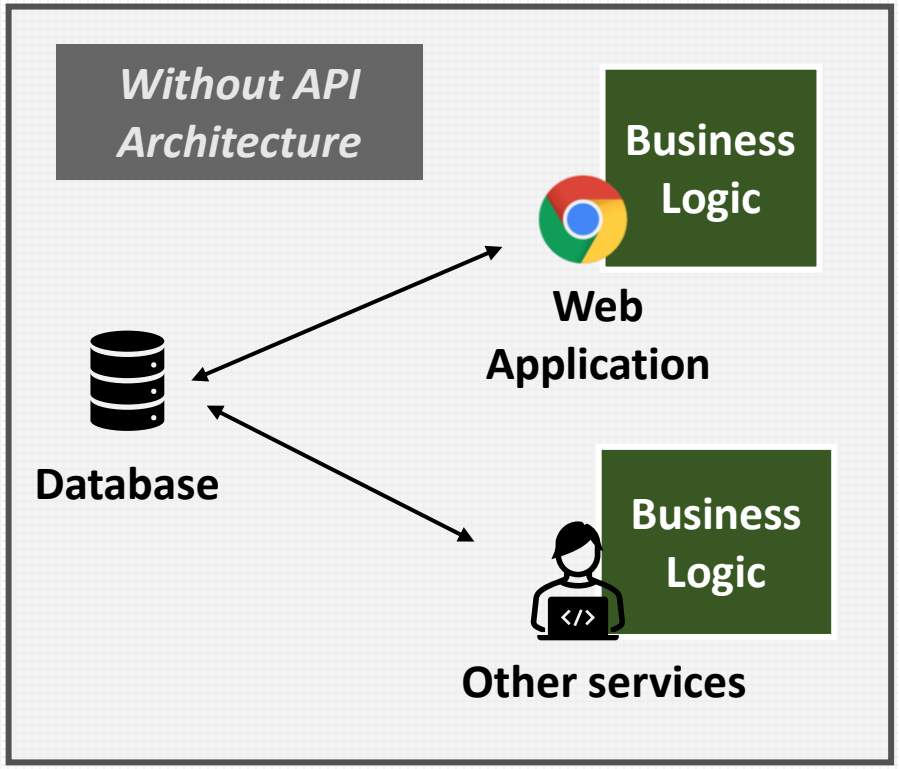
1. *What is API & How does it work?*
2. *REST: Representational State Transfer*
3. *CRUD Framework*
4. *HTTP Status /Response Codes*
5. *Use Case: Text Sentiment Model*
6. *Introduction to Flask & Django*

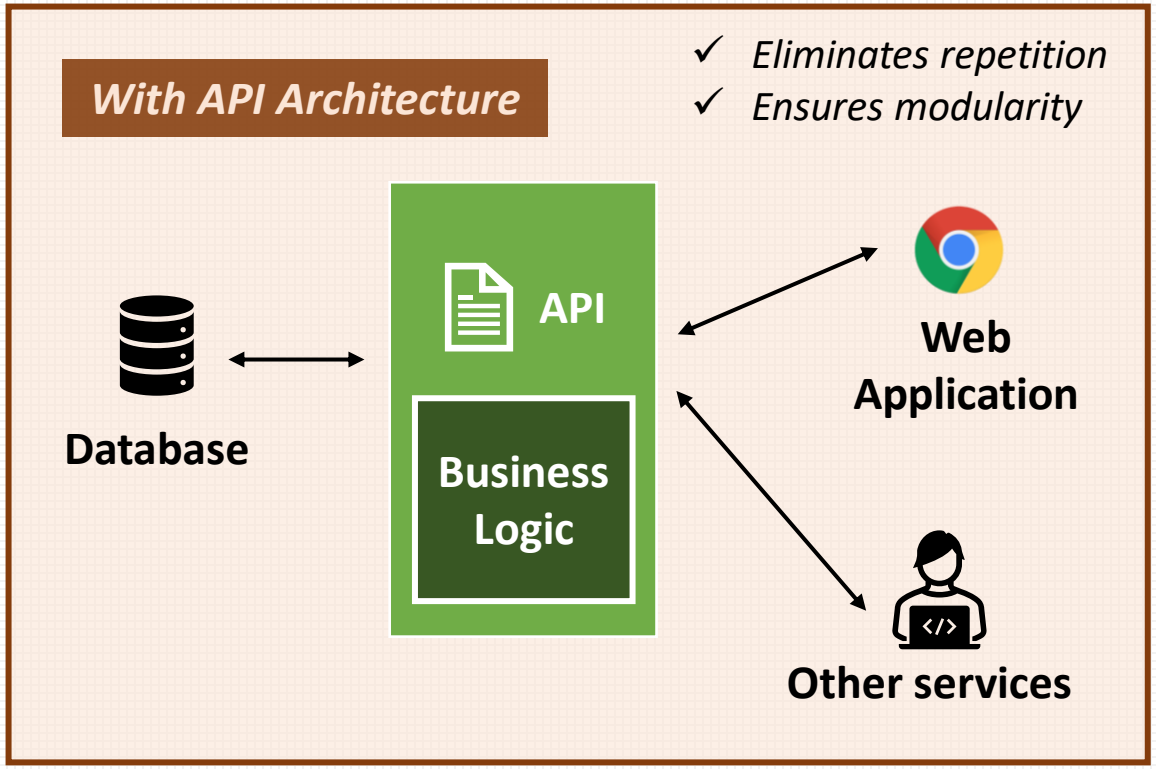
Part 2: Hands On Session

1. *Setting up virtual environment*
2. *Installation of packages*
3. *File Structure of Flask Projects*
4. *Use Case: Text Sentiment Model*
   a) *Creation of API routes for inference*
   b) *Hosting API locally*
   c) *Containerising of API*
   d) *Calling of hosted API for inference*

Application Program Interfaces (APIs)

Connectivity interface that facilitates data transfer, thus allowing different applications to talk to one another





REST: Representational State Transfer

Architectural pattern for creating API that uses HTTP

1 **Uniform Interface**

- *Standardised format (e.g. parameters)*
- *Ensures software engineering principle of generality to the component interface*

2 **Stateless**

- *Each request from client is singular (i.e., server must contain all of the information necessary to understand the request)*
- *Session state is kept entirely on the client*
- *Unable to take advantage of any stored context on the server session*

3 **Cacheable**

- *Able to reuse the response data for equivalent requests in the future*
- *Improves efficiency and scalability*

Extra reading on REST

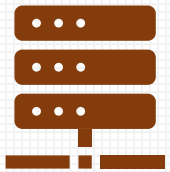
How does HTTP work?

1. User issues URL from browser

HTTP Client
(e.g. web browser)

2. Browser sends HTTP request message

3. Server maps URL to file/program



HTTP Server
(i.e. web server)

4. Server returns HTTP response message

5. Browser formats and displays response

Components of an URL

Uniform Resource Locator (URL)

https :// **api.data.gov.sg** : **443** / **v1/environment/air-temperature** **?date=2021-06-08**

protocol
application level protocol (e.g. http or https)

hostname
DNS domain name or IP address

port
tcp port number that the server is listening

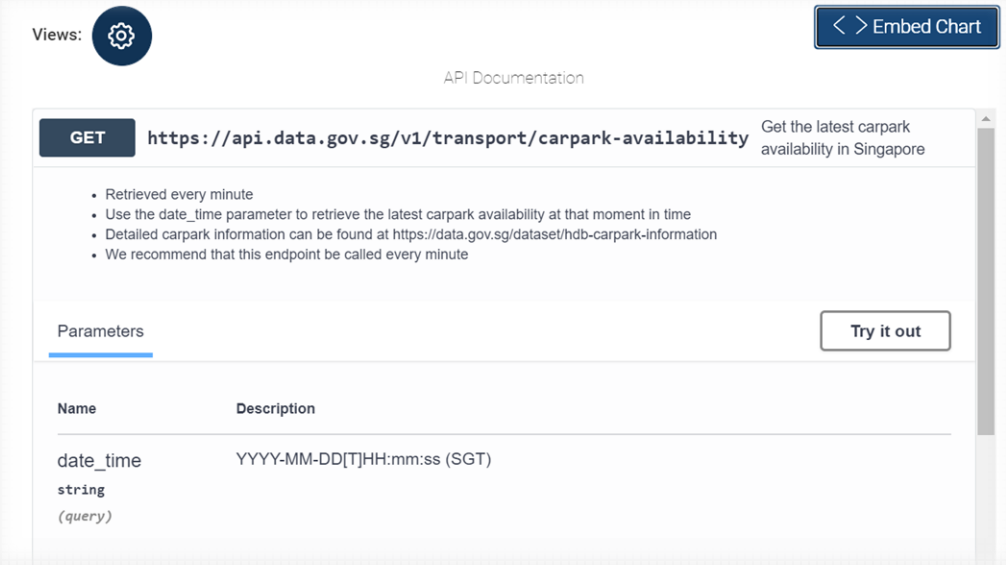
path and filename
name and location of requested resource

Query string parameters
Extra parameters for request (after the ?)

CRUD Framework

Operation		Restful API
Create	→	Post
Read	→	Get
Update	→	Put
Delete	→	Delete

Get request is used to retrieve carpark availability data



- Type of request can be usually seen in API documentation (along with the query parameters)

General HTTP Status Codes

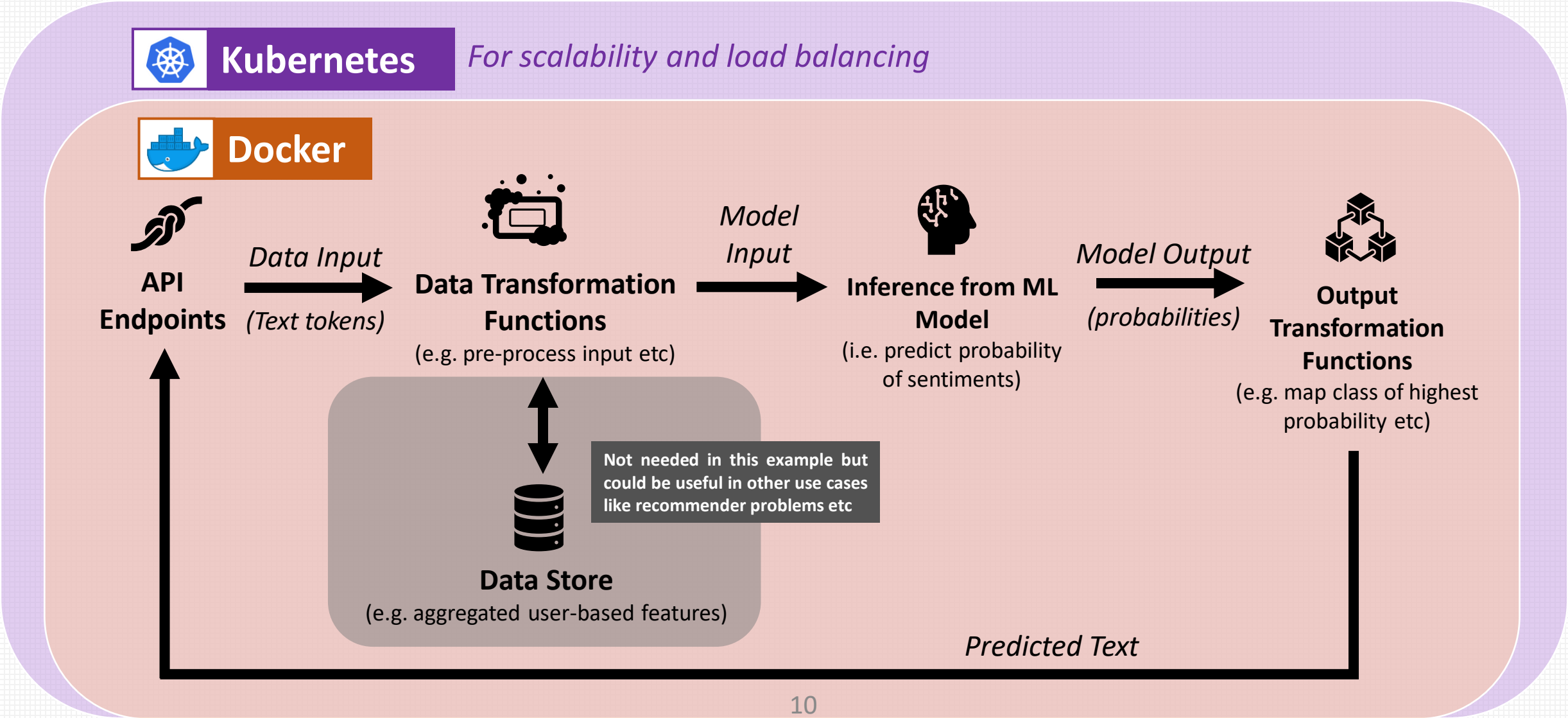
Category	Description
1xx: Informational	Communicates transfer protocol-level information
2xx: Success	Indicates that client's request was accepted successfully
3xx: Redirection	Indicates that the client must take some additional action in order to complete the request
4xx: Client Error	This category of error status codes points the finger at clients
5xx: Server Error	Server takes responsibility for these error status codes

Common HTTP Response Codes

Code	Meaning
200 OK	General **success** status codes
201 Created	Success creation of resource occurred
400 Bad Request	General client-side error status, when no other 4xx error code is appropriate
401 Unauthorised	Client tried to operate on a protected resource without proper authorisation
403 Forbidden	User does not have necessary permissions for resource
404 Not Found	Resource is not found
500 Internal Server Error	General **server error** status code

Use Case of API in Model Deployment

Inference for Text Sentiment Model



Introduction to Flask & Django

Web Application Frameworks for API Implementation

Flask

- **Light-weight framework built for rapid development**

- **URL dispatcher utilises RESTful request format**
(i.e., explicitly states the path endpoints using CRUD framework – get/post/update/delete)

- **To use data models, ORM wrapper packages like SQLAlchemy will be needed to be imported.**

django

- **Full-stack web framework loaded with functionalities suitable for complex systems**

- **URL dispatcher based on controller-regex format**
(i.e., uses regex to match path endpoints for APIs)

- **Provides its own Django ORM (object-relational mapping) and uses data models**
(i.e., able to use any database and perform common DB tasks)

Introduction to Flask & Django

Web Application Frameworks for API Implementation

Flask

- **Light-weight framework built for rapid development**
- **URL dispatcher utilises RESTful request format**

(i.e., explicitly states the path endpoints using CRUD framework – get/post/update/delete)

- **To use data models, ORM wrapper packages like SQLAlchemy will be needed to be imported.**

Main framework used in the Hands-On!

django

- **Full-stack web framework loaded with functionalities suitable for complex systems**
- **URL dispatcher based on controller-regex format**

(i.e., uses regex to match path endpoints for APIs)

- **Provides its own Django ORM (object-relational mapping) and uses data models**

(i.e., able to use any database and perform common DB tasks)

Time to get our hands dirty…

Part 1: Overview of APIs

1. *What is API & How does it work?*
2. *REST: Representational State Transfer*
3. *CRUD Framework*
4. *HTTP Status /Response Codes*
5. *Use Case: Text Sentiment Model*
6. *Introduction to Flask & Django*

Part 2: Hands On Session

1. *Setting up virtual environment*
2. *Installation of packages*
3. *File Structure of Flask Projects*
4. *Use Case: Text Sentiment Model*
   a) *Creation of API routes for inference*
   b) *Hosting API locally*
   c) *Containerising of API*
   d) *Calling of hosted API for inference*

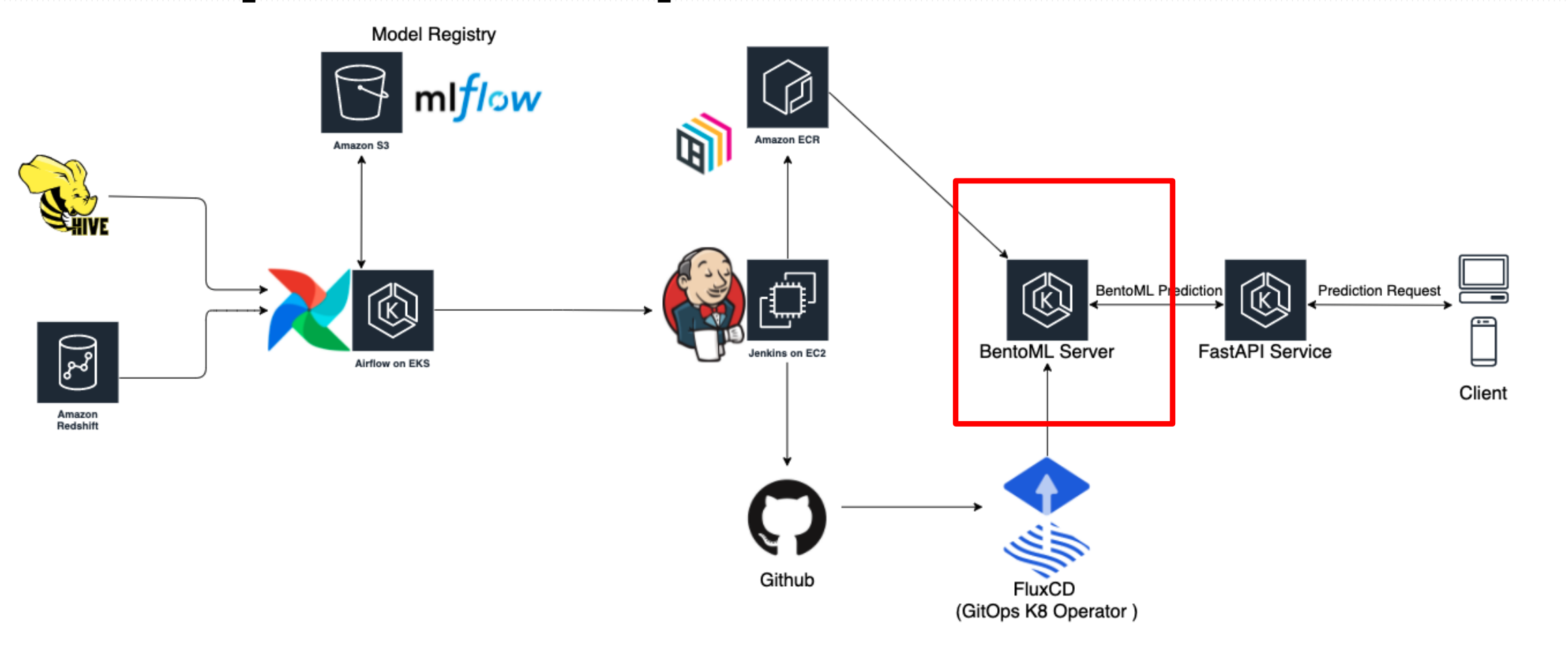
Extra Links for Reading

- **Introduction to MLOps** (Gives a good overview of challenges and components) – Link
- **Article on possibly architecture patterns for model serving – Link**
- **Considerations when productionalising a model (e.g. batch training, real-time serving etc) – Link**
- **Azure architecture for Model Deployment**
    1. Real-time recommendation prediction using Azure – Link
    2. Azure Kubernetes Service for model deployment – Link
    3. Introduction to MLOps with Azure (only with Docker) – Link
- **Docker vs Kubernetes – Link**
- **In-depth discussion on when to use Flask or Django – Link1 / Link2**
    1. Flask Documentation – Link
    2. Django Documentation – Link

Alternative to Flask/Django

BentoML

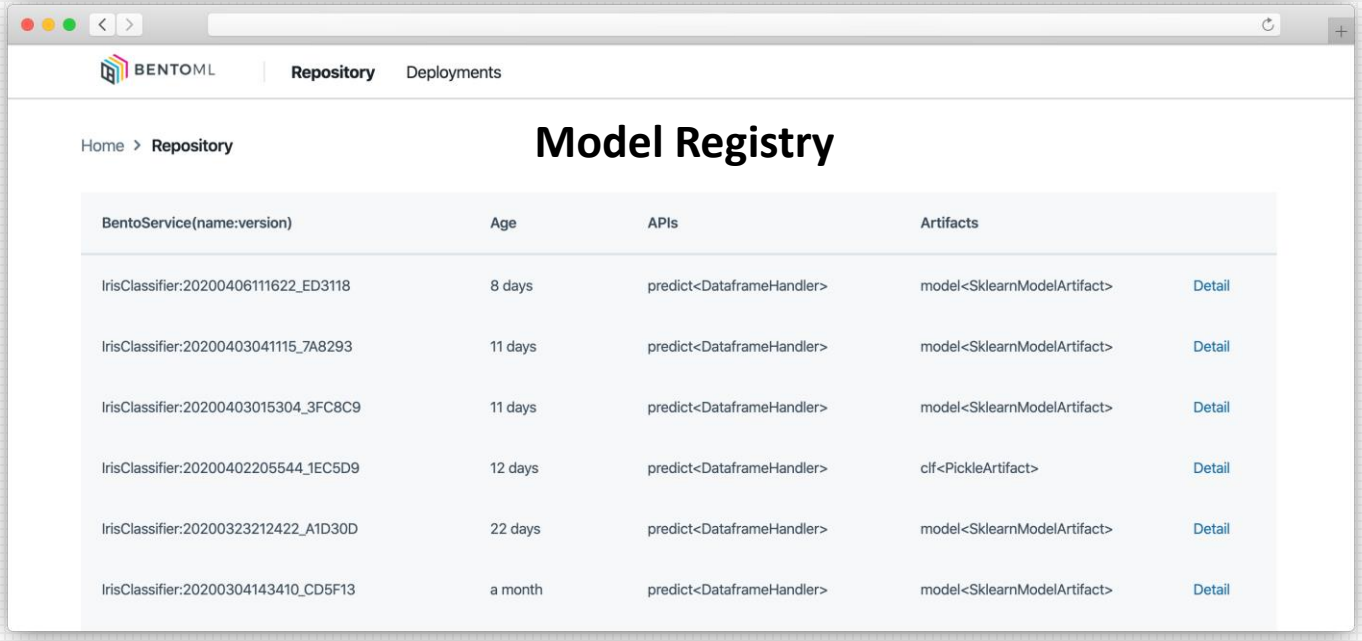
Example of MLOps Flow



BentoML

Link to BentoML example

- Flexible, high-performance framework for serving, managing, and deploying machine learning models.
- Supports multiple ML frameworks, including Tensorflow, PyTorch, Keras, XGBoost and more.
- Cloud native deployment with Docker, Kubernetes, AWS, Azure and more.



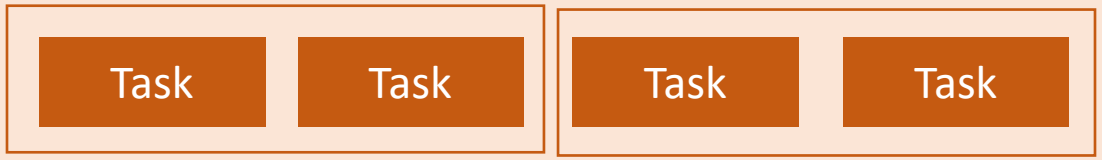
BentoService(name:version)	Age	APIs	Artifacts	
IrisClassifier:20200406111622_ED3118	8 days	predict<DataframeHandler>	model<SklearnModelArtifact>	Detail
IrisClassifier:20200403041115_7A8293	11 days	predict<DataframeHandler>	model<SklearnModelArtifact>	Detail
IrisClassifier:20200403015304_3FC8C9	11 days	predict<DataframeHandler>	model<SklearnModelArtifact>	Detail
IrisClassifier:20200402205544_1EC5D9	12 days	predict<DataframeHandler>	clf<PickleArtifact>	Detail
IrisClassifier:20200323212422_A1D30D	22 days	predict<DataframeHandler>	model<SklearnModelArtifact>	Detail
IrisClassifier:20200304143410_CD5F13	a month	predict<DataframeHandler>	model<SklearnModelArtifact>	Detail

BentoML

There are 3 main types of model serving -

- Online Serving - clients access predictions via API endpoints in near real-time
- Offline Batch Serving - pre-compute predictions and save results in a storage system
- Edge Serving - distribute model and run it on mobile or IoT devices

Adaptive Micro Batching



Incoming prediction requests are grouped into small batches to achieve the performance advantage of batch processing in model inference tasks.