



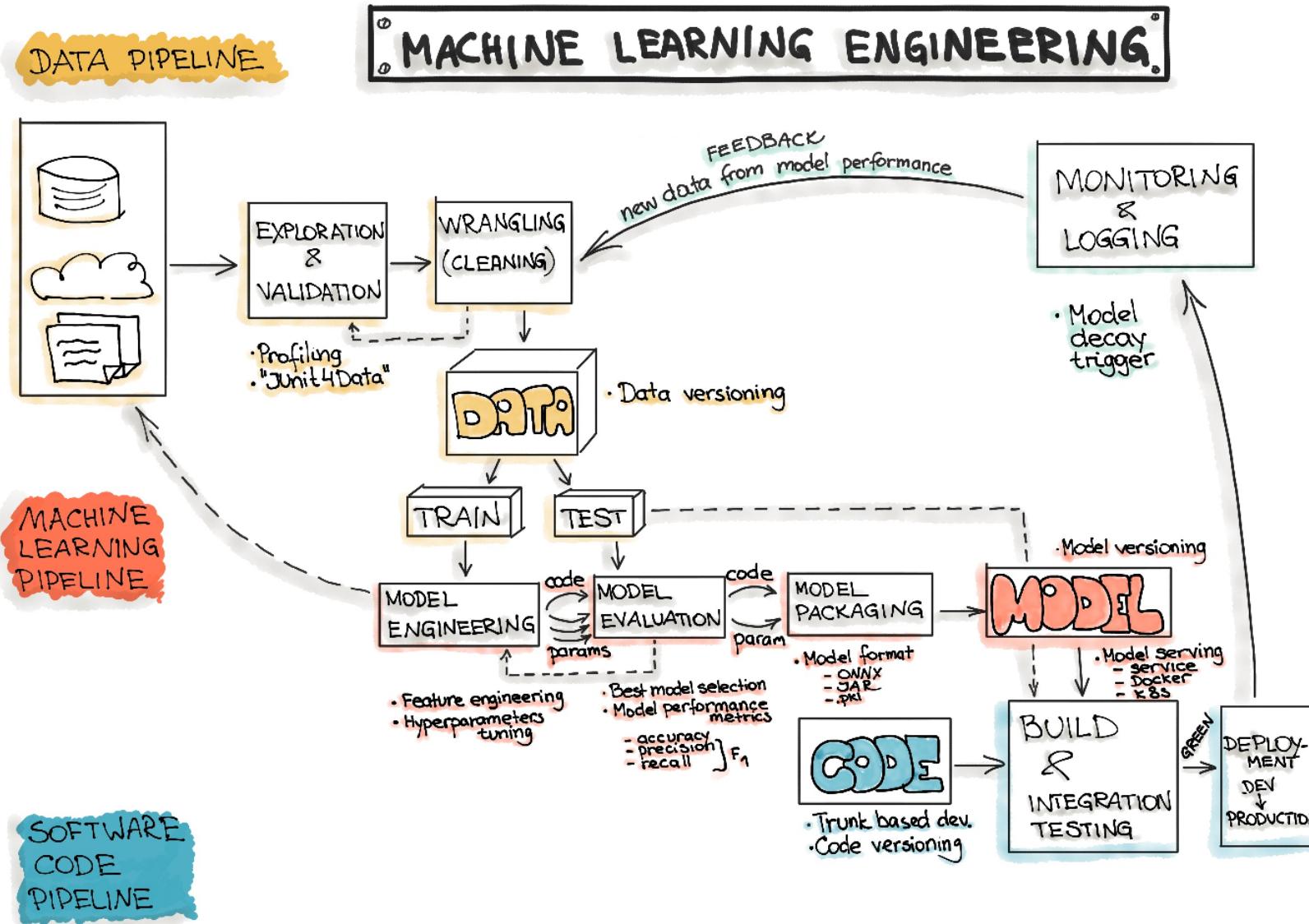
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Model Serving Patterns

Pravin Y Pawar

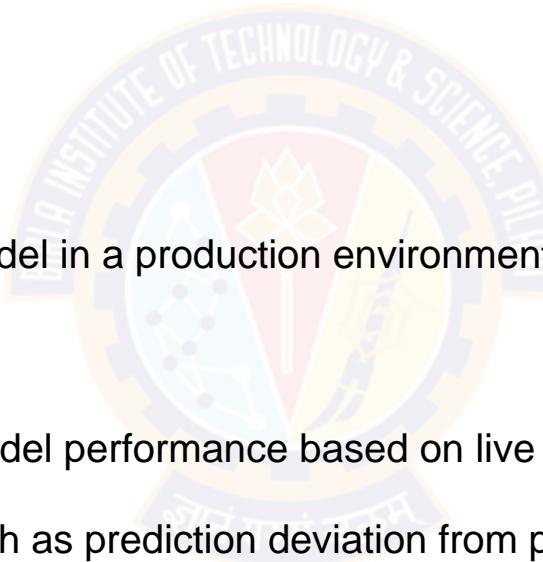
Extracted from
[Three Levels of ML Software](#)

Machine Learning Workflow



Code: Deployment Pipelines

- The final stage of delivering an ML project includes the following three steps:
 - Model Serving
 - Model Performance Monitoring
 - Model Performance Logging
- Model Serving
 - The process of deploying the ML model in a production environment
- Model Performance Monitoring
 - The process of observing the ML model performance based on live and previously unseen data, such as prediction or recommendation
 - interested in ML-specific signals, such as prediction deviation from previous model performance
 - signals might be used as triggers for model re-training
- Model Performance Logging
 - Every inference request results in a log-record.



Model Serving Patterns

- Three components should be considered when ML model is served in a production environment
 - The inference is the process of getting data to be ingested by a model to compute predictions
 - requires a model, an interpreter for the execution, and input data
- Deploying an ML system to a production environment includes two aspects,
 - First deploying the pipeline for automated retraining and ML model deployment
 - Second, providing the API for prediction on unseen data
- Model serving is a way to integrate the ML model in a software system
- Five patterns to put the ML model in production:
 - Model-as-Service
 - Model-as-Dependency
 - Precompute
 - Model-on-Demand
 - and Hybrid-Serving

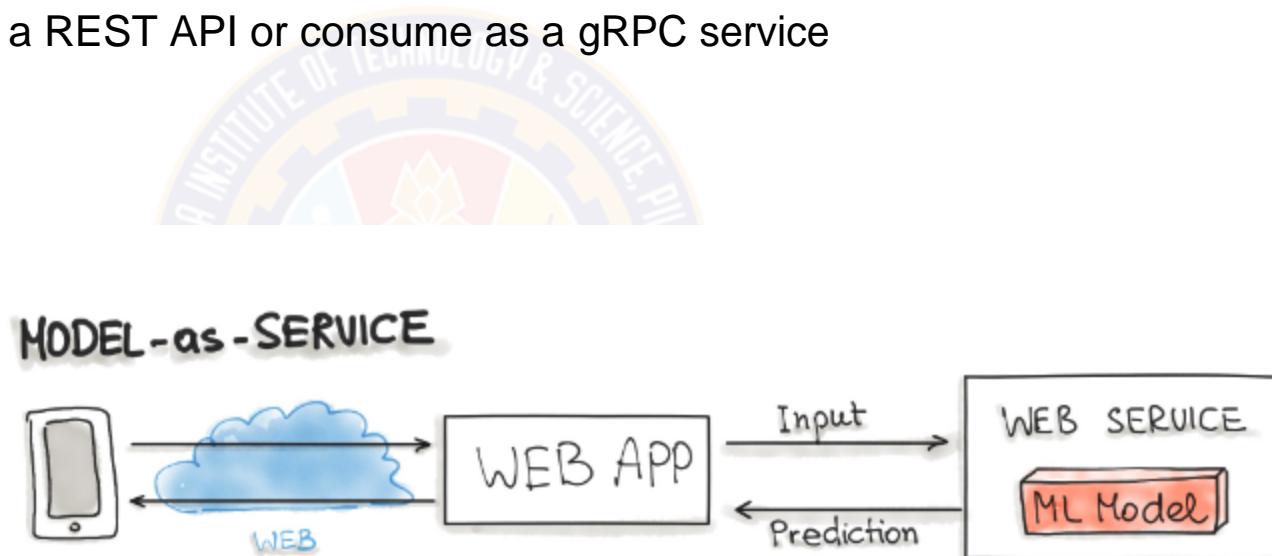
Model Serving Patterns(2)

Machine Learning Model Serving Taxonomy

Machine Learning Model Serving Taxonomy			
	ML Model		
Service & Versioning	Together with the consuming application	Independent from the consuming application	
Compile/ Runtime Availability	Build & runtime available	Available remotely through REST API/RPC	Available at the runtime scope
Serving Patterns	"Model-as-Dependency"	"Model-as-Service"	"Precompute" and "Model on Demand"
	Hybrid Model Serving (Federated Learning)		

Model-as-Service

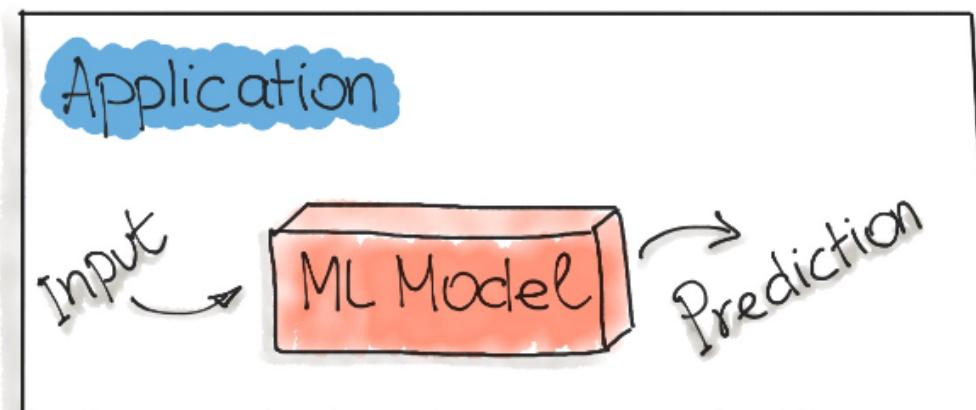
- A common pattern for wrapping an ML model as an independent service
 - can wrap the ML model and the interpreter within a dedicated web service
 - applications can request through a REST API or consume as a gRPC service
- Used for various ML workflows
 - Forecast
 - Web Service
 - Online Learning



Model-as-Dependency

- Probably the most straightforward way to package an ML model
 - mostly used for implementing the Forecast pattern.
- A packaged ML model is considered as a dependency within the software application
 - For example, the application consumes the ML model like a conventional jar dependency by invoking the prediction method and passing the values
 - The return value of such method execution is some prediction that is performed by the previously trained ML model

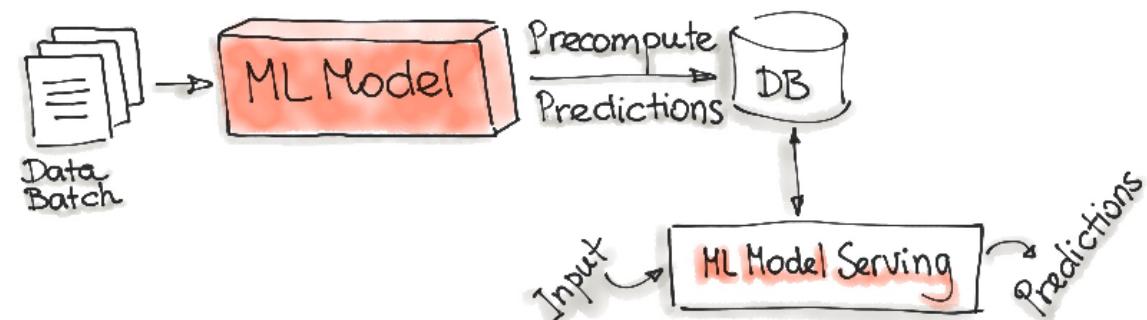
MODEL -as- DEPENDENCY



Precompute Serving Pattern

- Tightly related to the Forecast ML workflow
- Use an already trained ML model and precompute the predictions for the incoming batch of data
 - Resulting predictions are persisted in the database
 - For any input request, query the database to get the prediction result

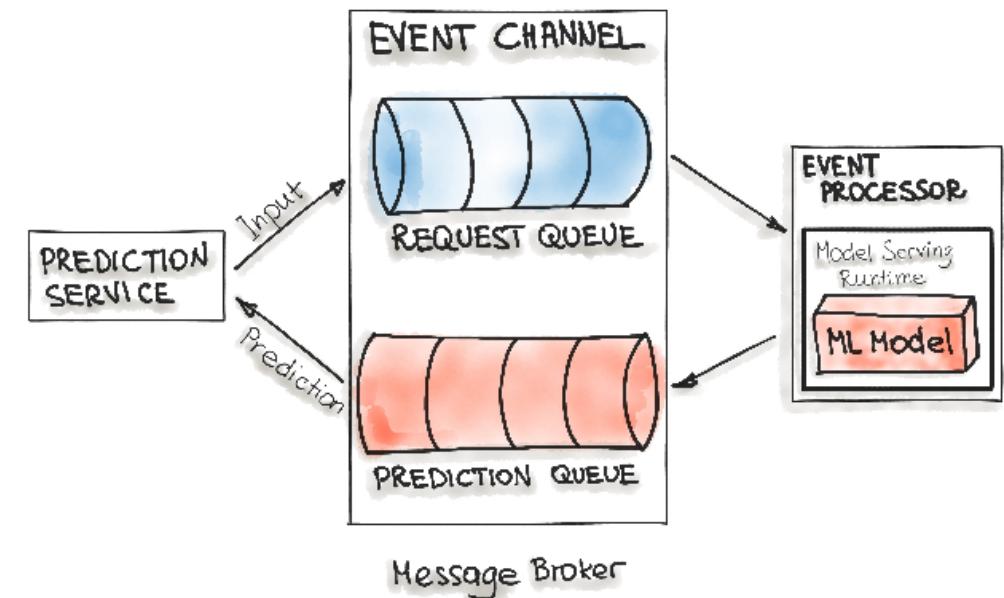
PRECOMPUTE SERVING PATTERN



Model-on-Demand

- Treats the ML model as a dependency that is available at runtime
 - contrary to the Model-as-Dependency pattern, has its own release cycle and is published independently
- Message-broker architecture is typically used for such on-demand model serving
 - contains two main types of architecture components:
 - a broker component
 - an event processor component
 - Broker component is the central part that contains the event channel that are utilised within the event flow
 - The event channels, which are enclosed in the broker component, are message queues
 - Message broker allows one process to write prediction-requests in an input queue
 - Event processor contains the model serving runtime and the ML model
 - connects to the broker, reads these requests in batch from the queue and sends them to the model to make the predictions
- The model serving process runs the prediction generation on the input data
 - writes the resulted predictions to the output queue
 - queued prediction results are pushed to the prediction service that initiated the prediction request

MODEL-ON-DEMAND

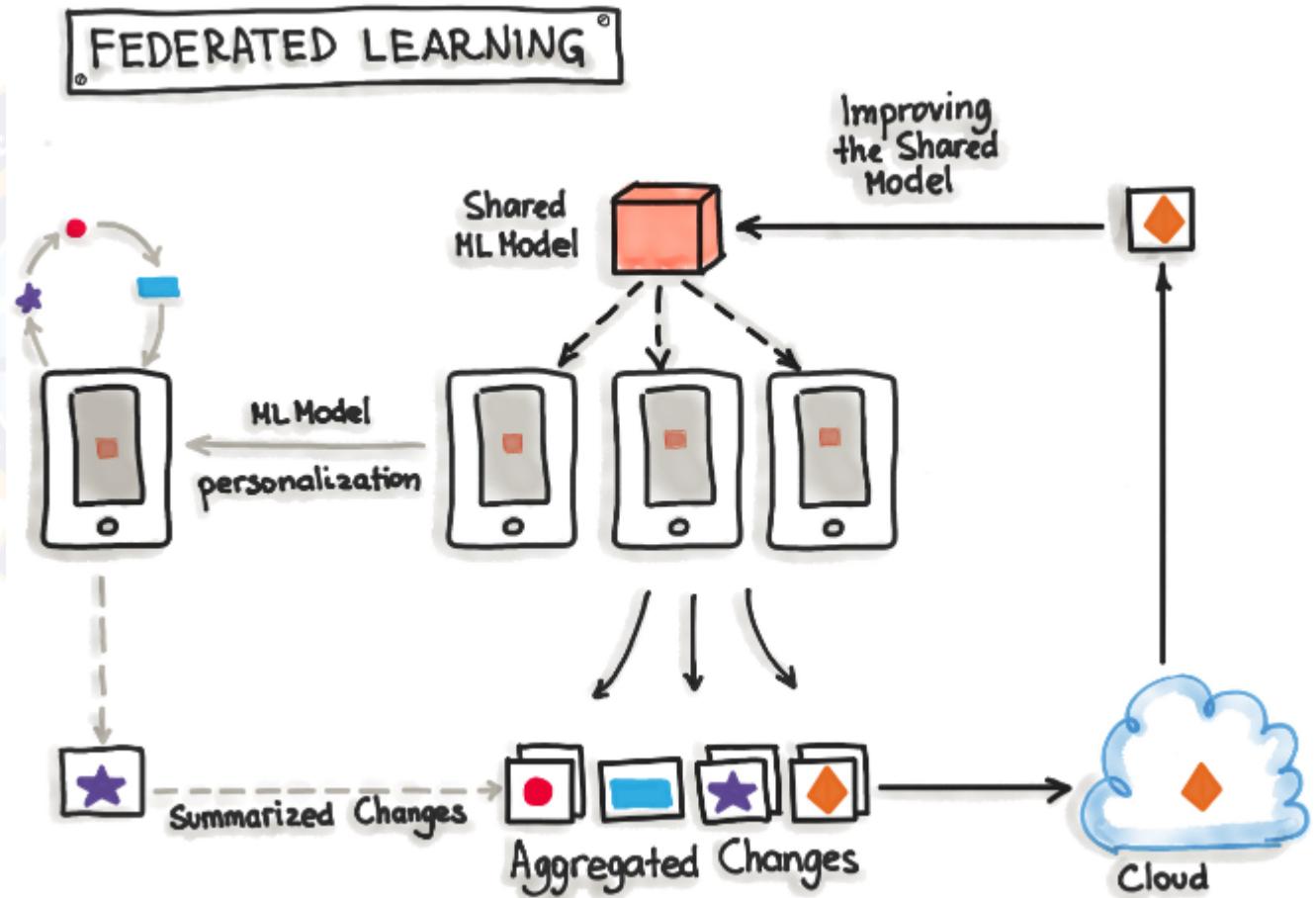


Hybrid-Serving (Federated Learning)

- Unique in the way it does, there is not only one model that predicts the outcome, but there are also lots of it
 - Exactly spoken there are as many models as users exist, in addition to the one that's held on a server
- Start with the unique model, the one on the server
 - Model on the server-side is trained only once with the real-world data
 - sets the initial model for each user
 - a relatively general trained model so it fits for the majority of users
- On the other side, there are the user-side models - really unique models
 - possible for the devices to train their own models due to hardware enhancements
 - devices will train their own highly specialized model for their own user
 - Once in a while, the devices send their already trained model data (not the personal data) to the server
- Then the server model will be adjusted
 - the actual trends of the whole user community will be covered by the model
 - this updated server model is set to be the new initial model that all devices are using

Hybrid-Serving (Federated Learning) 2

- Not having any downsides for the users, while the server model gets updated, this happens only when the device is idle, connected to WiFi and charging
 - testing is done on the devices, therefore the newly adopted model from the server is sent to the devices and tested for functionality
- Big benefit
 - data used for training and testing, which is highly personal, never leaves the devices while still capturing all data that is available
 - possible to train highly accurate models while not having to store tons of (probably personal) data in the cloud
- Constraints
 - mobile devices are less powerful
 - the training data is distributed across millions of devices and these are not always available for training
 - Exactly for this TensorFlow Federated (TFF) has been created - lightweight form of TensorFlow



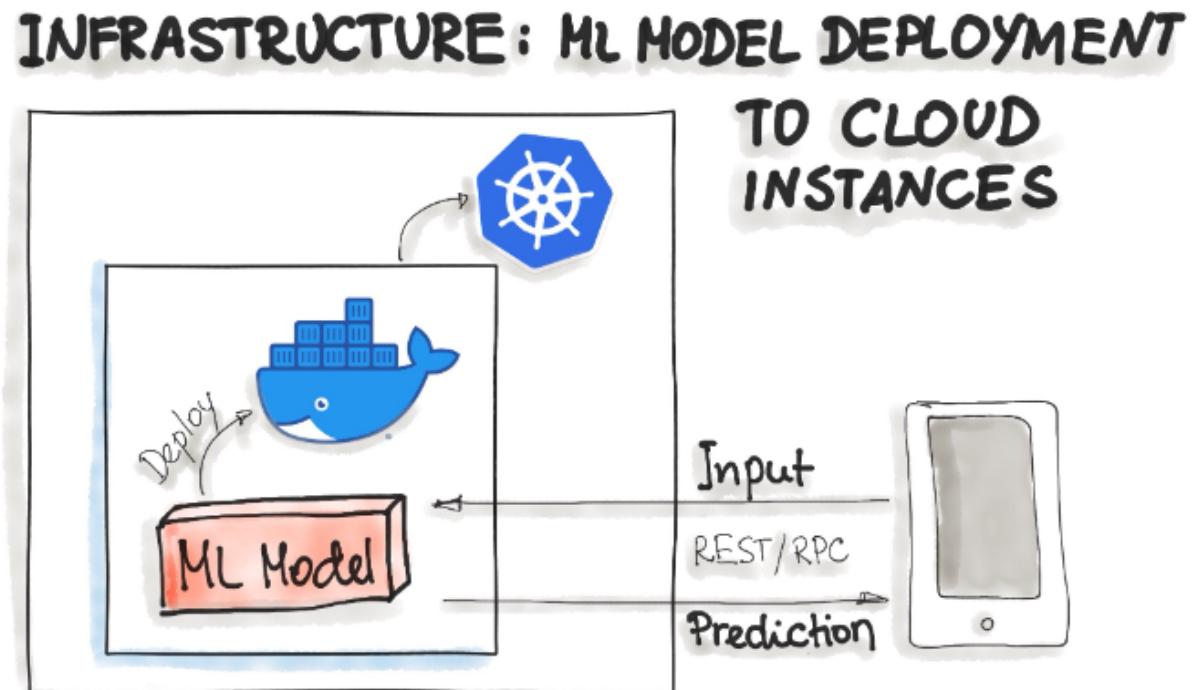
Deployment Strategies

- Common ways for wrapping trained models as deployable services, namely deploying ML models as
 - Docker Containers to Cloud Instances
 - Serverless Functions



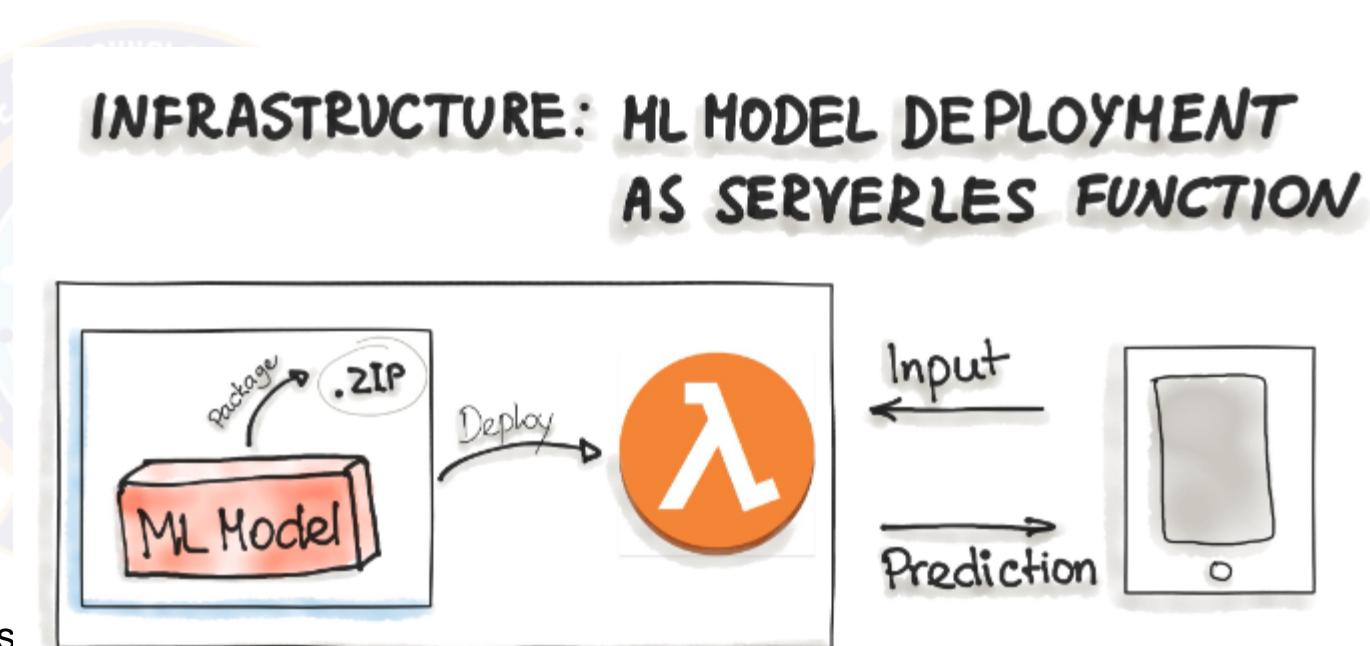
Deploying ML Models as Docker Containers

- No standard, open solution to ML model deployment!
- Containerization becomes the de-facto standard for delivery
 - as ML model inference being considered stateless, lightweight, and idempotent
 - means deploy a container that wraps an ML model inference code
- Docker is considered to be de-facto standard containerization technology
 - for on-premise, cloud, or hybrid deployments
- One ubiquitous way is to package the whole ML tech stack (dependencies) and the code for ML model prediction into a Docker container
 - Then Kubernetes or an alternative (e.g. AWS Fargate) does the orchestration
 - The ML model functionality, such as prediction, is then available through a REST API (e.g. implemented as Flask application)



Deploying ML Models as Serverless Functions

- Various cloud vendors already provide machine-learning platforms - can deploy model with their services
 - Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning
 - Commercial cloud services also provide containerization of ML models such as AWS Lambda and Google App Engine servlet host
- In order to deploy an ML model as a serverless function,
 - the application code and dependencies are packaged into .zip files, with a single entry point function
 - could be managed by major cloud providers such as Azure Functions, AWS Lambda, or Google Cloud Functions
 - However, attention should be paid to possible constraints of the deployed artifacts such as the size of the artifacts





Thank You!

In our next session: