Model Deployment Strategies

Approaches for Deploying Machine Learning Models in Production

Lecturer: Vangelis Oden - Technology Lead (Kera)
Assistant: Natalija Mitic - AI/ML Engineer (Kera)

Agenda

- Why it Matters
- Deployment Options: Batch vs. Real-time Inference
- Serving Models - REST APIs, Tensorflow
- Containerization
- Orchestration
- Edge Deployment and IOT Considerations
- Best Practices for Model Deployment
- Conclusion
- Q&A

Why it matters?

Intuition:

- batch inference : predictions are generated for large sets of data at once. Ideal for non-time-sensitive tasks, such as processing a large number of documents or making daily stock predictions.
- real-time inference : Predictions are generated on the fly for individual or small sets of data.
  - Necessary for applications where responses need to be immediate, such as fraud detection or personalised recommendations.

Deployment Options - Batch vs. Real-time Inference

Intuition:

- batch inference : predictions are generated for large sets of data at once. Ideal for non-time-sensitive tasks, such as processing a large number of documents or making daily stock predictions.
- real-time inference : Predictions are generated on the fly for individual or small sets of data.
    - Necessary for applications where responses need to be immediate, such as fraud detection or personalised recommendations.

Deployment Options - Batch vs. Real-time Inference

python Copy

```python
import pandas as pd
from sklearn.externals import joblib

# Load pre-trained model
model = joblib.load('model.pkl')

# Batch data for inference
data = pd.read_csv('new_data.csv')

# Batch prediction
predictions = model.predict(data)
```

Batch Inferencing

python Copy

```python
from fastapi import FastAPI
import joblib

app = FastAPI()
model = joblib.load('model.pkl')

@app.post("/predict")
def predict(data: dict):
    input_data = [data['feature1'], data['feature2']]
    prediction = model.predict([input_data])
    return {"prediction": prediction[0]}
```

Real-time inferencing

Serving Models - REST APIs

Intuition:

REST APIs are a popular method for serving models in a scalable manner. Models can be deployed on a server, and external systems can query them for predictions using HTTP requests.

- **REST** : **RE**presentational **S**tate **T**ransfer. An architectural style that allows communication between resources over HTTP(S). REST
- **API** : **A**pplication **P**rogramming **I**nterfaces. Set of programming code that enables data transmission between software applications.

Serving Models - TensorFlow Serving

Intuition:

REST APIs are a popular method for serving models in a scalable manner. Models can be deployed on a server, and external systems can query them for predictions using HTTP requests.

- **REST** : **RE**presentational **S**tate **T**ransfer. An architectural style that allows communication between resources over HTTP(S). REST
- **API** : **A**pplication **P**rogramming **I**nterfaces. Set of programming code that enables data transmission between software applications.

Containerization

Intuition:

Encapsulates code and all its dependencies in a lightweight, portable container, ensuring consistency across environments.

- useful for models because most model code environments require high specificity with dependencies.
- docker is the most popular tool.
- ensures consistency across environments

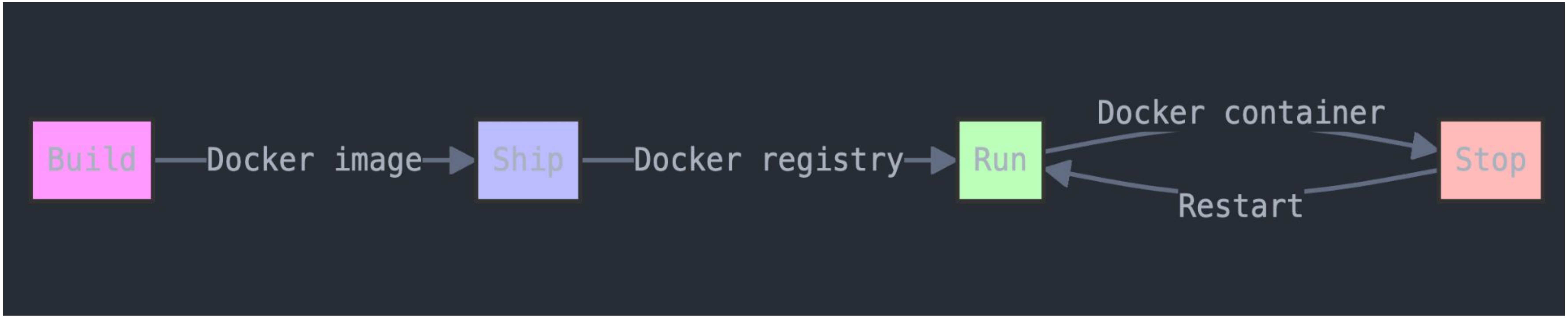
Containerization

Dockerfile

```
# Use a base image
FROM python:3.9

# Set working directory
WORKDIR /app

# Copy requirements and install
COPY requirements.txt .
RUN pip install -r requirements.txt

# Copy model and app files
COPY . .

# Expose port for Flask
EXPOSE 5000

# Run the Flask app
CMD ["python", "app.py"]
```

Containerization - vs VMs

- more lightweight than VMs - uses fewer resources
- support decomposition of applications into microservices
- faster to manage and deploy
- share existing system resources without partitioning



Docker internals

Orchestration

Intuition:

Manages the deployment of containers, ensuring scalability, load balancing and fault tolerance.

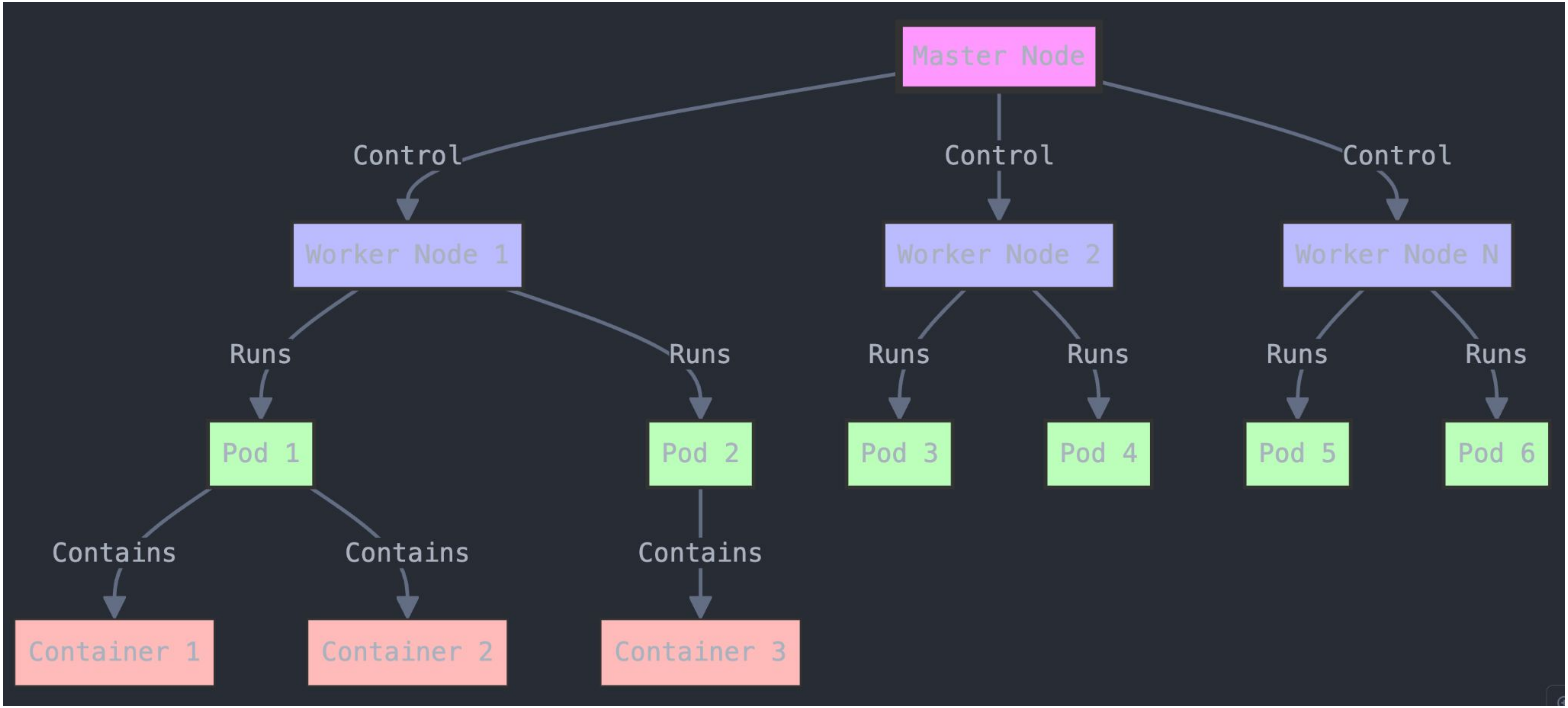
- useful for ensuring containers are always available to serve requests.
- deploy multiple services as containers in a single process
- ensures services can speak with themselves with low latency
- kubernetes is the most popular tool

Orchestration

yaml Copy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-model-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ml-model
  template:
    metadata:
      labels:
        app: ml-model
    spec:
      containers:
      - name: ml-container
        image: ml-model-image:latest
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: ml-model-service
spec:
  selector:
    app: ml-model
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

Orchestration - Kubernetes



Containerization Orchestration

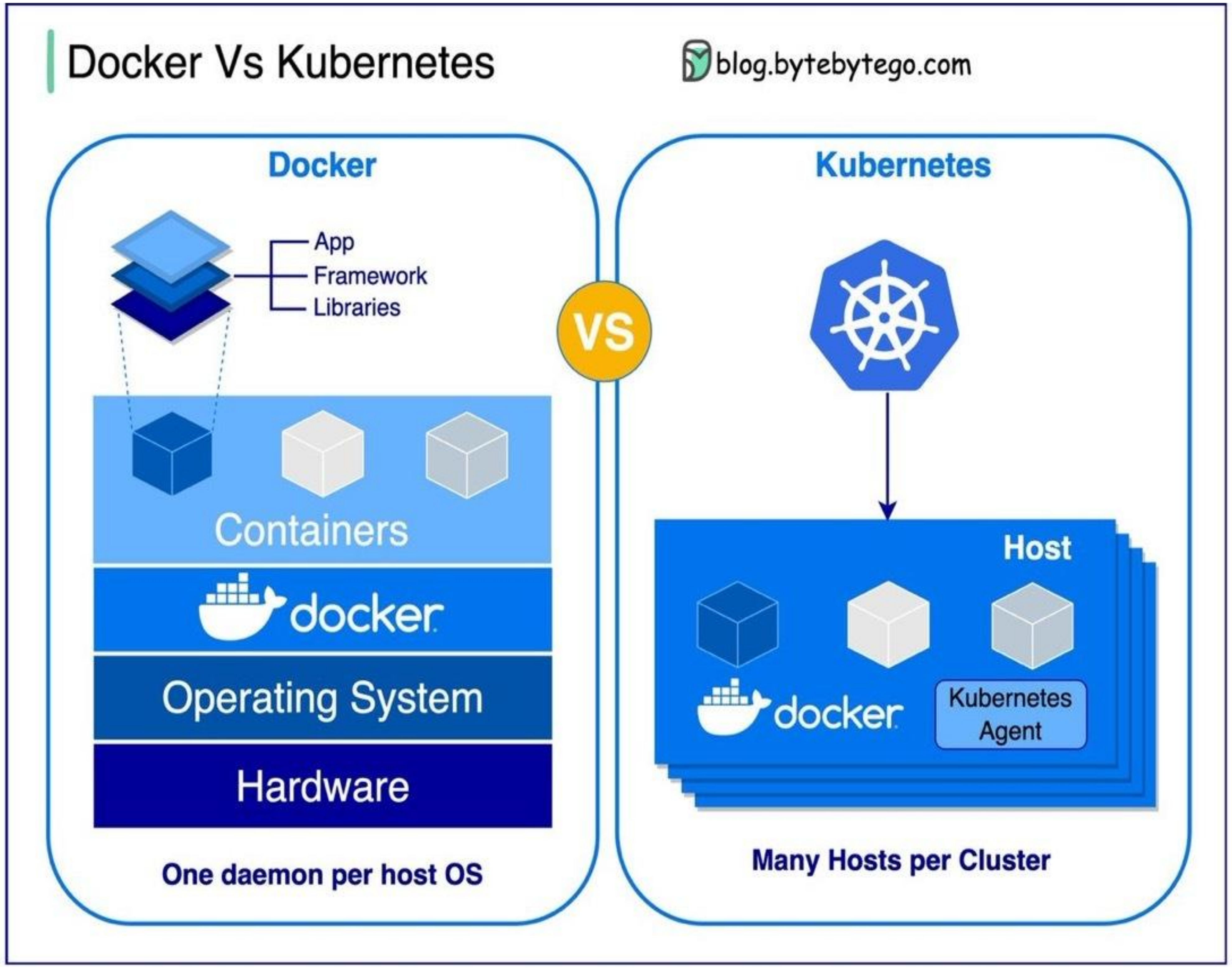
Why it matters?

- Docker allows easy packaging and portability, while Kubernetes ensures your model scales efficiently across multiple instances.

NOTE:

- You can decide to use only Docker on a VM
- You can decide to use Docker with Kubernetes
- They don’t necessarily have to go together, but work nicely together.

Containerization Orchestration



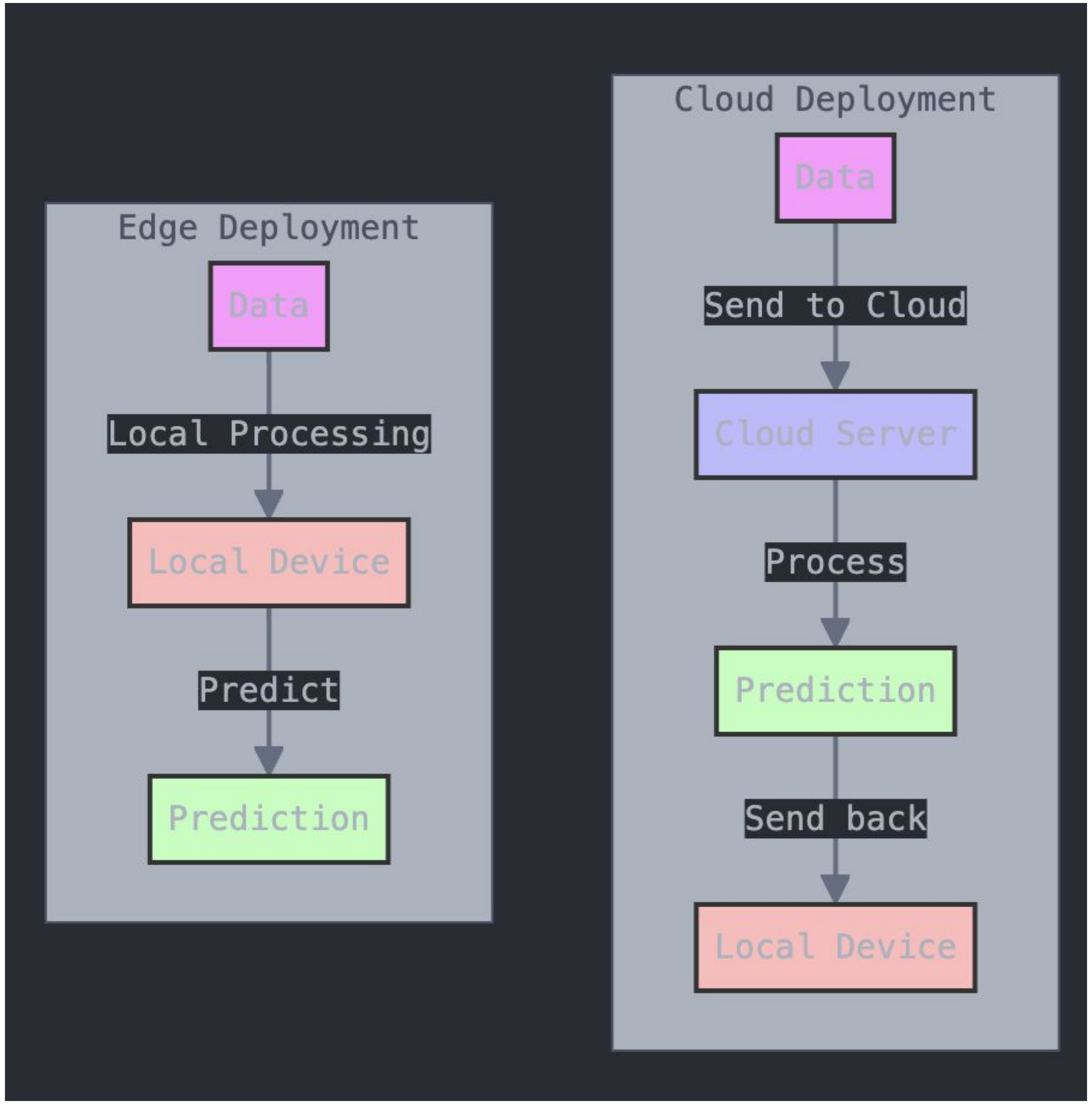
Edge Deployment

Intuition

Involves running machine learning models on local devices (e.g., smartphones, IoT devices) instead of sending data to a central server. This is critical for the following:

- real-time decision making
- low-latency
- privacy and localisation preservation.
- crucial in sectors like healthcare, autonomous driving, etc.

Edge Deployment - vs Cloud



Edge Deployment

Cons

- limited processing power of edge devices
- managing and maintaining models in multiple devices

NOTE:

- use Tensorflow Lite, Pytorch Mobile or Onnyx to create edge compatible models for deployment on edge devices.

Best Practices for Model Deployment

- **monitor models in production** : set up monitoring for performance drift and accuracy.
- **automate deployment** : use CI/CD pipelines for automating model updates.
- **ensure security** : secure APIs, containers, and edge devices to prevent breaches.

Conclusion

- Batch vs Real-time: Understand the trade-offs.
- REST APIs: Standard for scalable model serving.
- Docker & Kubernetes: Essential tools for packaging and scaling models.
- Edge Deployment: Key for IoT applications and low-latency needs.

Q&A

Thank You!