Deployment Automation & Compliance

AI Masters Capstone Project - Presentation 4

Jonathan Agustin

2024

Modern MLOps Architecture

- CI/CD with automated testing, bias checks, reproducible deployments
- Infrastructure Automation (Docker, Kubernetes, IaC) for scalability
- Production Ops: monitoring, rollback, performance optimization

CI/CD Pipeline

- Triggered by code changes (push or pull request)
- Runs unit/integration tests, model validation scripts
- Checks bias metrics, security scans
- Deploys to staging if all checks pass

CI/CD Pipeline Code (Part 1)

```yaml
name: ML Model Deploy
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test-and-validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'
```

CI/CD Pipeline Code (Part 2)

```yaml
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
    pip install -r requirements-test.txt

- name: Run tests and validation
  run: |
    pytest tests/
    python scripts/validate_model.py
    python scripts/check_bias.py
```

CI/CD Pipeline Code (Part 3)

```yaml
- name: Security scan
  uses: anchore/scan-action@v3
  with:
    image: "model-service:${{ github.sha }}"
    fail-build: true
    severity-cutoff: high

- name: Deploy to staging
  if: github.ref == 'refs/heads/main'
  env:
    KUBE_CONFIG: ${{ secrets.KUBE_CONFIG }}
  run: |
    echo "$KUBE_CONFIG" > kubeconfig.yaml
    kubectl --kubeconfig=kubeconfig.yaml apply -f k8s/staging/
    python scripts/validate_deployment.py --environment staging
```

Model Validation

- Checks accuracy, F1, ROC AUC against thresholds
- Evaluates fairness: max disparity among groups
- Validates latency (p95) not exceeding set limits
- Logs results for audit and compliance

Model Validation Code (Part 1)

```python
from typing import Dict, List
import numpy as np
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

class ModelValidator:
    def __init__(self, performance_thresholds: Dict[str,float],
                 fairness_thresholds: Dict[str,float],
                 latency_threshold: float):
        self.performance_thresholds = performance_thresholds
        self.fairness_thresholds = fairness_thresholds
        self.latency_threshold = latency_threshold
        self.validation_results = {}

    def validate_performance(self, y_true: np.ndarray,
                            y_pred: np.ndarray, y_prob: np.ndarray) -> bool:
        metrics = {
            'accuracy': accuracy_score(y_true, y_pred),
            'roc_auc': roc_auc_score(y_true, y_prob),
            'f1': f1_score(y_true, y_pred)
        }
        self.validation_results['performance'] = metrics
        return all(metrics[k]>=v for k,v in self.performance_thresholds.items())
```

Model Validation Code (Part 2)

```python
def validate_fairness(self, y_true: np.ndarray, y_pred: np.ndarray,
                      protected_groups: Dict[str, np.ndarray]) -> bool:
    fairness_metrics={}
    for gname,mask in protected_groups.items():
        g_err = np.mean(y_pred[mask]!=y_true[mask])
        fairness_metrics[gname]=g_err
    max_disp = max(fairness_metrics.values()) - min(fairness_metrics.values())
    self.validation_results['fairness']={'max_disparity': max_disp}
    return max_disp <= self.fairness_thresholds.get('max_disparity',0.2)

def validate_latency(self, latency_samples: List[float]) -> bool:
    p95_latency = np.percentile(latency_samples,95)
    self.validation_results['latency']={'p95': p95_latency}
    return p95_latency <= self.latency_threshold

def summarize(self):
    return self.validation_results
```

Docker Optimization

- Multi-stage builds for lean images
- Distroless base for security
- Health checks, resource env vars
- Model artifacts separated from code

Dockerfile Code (Part 1)

```dockerfile
# Build stage
FROM python:3.10-slim as builder
WORKDIR /build
COPY requirements.txt .
RUN pip install --user -r requirements.txt

# Runtime stage
FROM gcr.io/distroless/python3
WORKDIR /app
COPY --from=builder /root/.local /root/.local
COPY ./src /app/src
COPY ./models /app/models

ENV PYTHONPATH=/app
ENV PATH=/root/.local/bin:$PATH

HEALTHCHECK --interval=30s --timeout=30s \
  --start-period=5s --retries=3 \
  CMD ["python","src/health_check.py"]
```

Dockerfile Code (Part 2)

```dockerfile
ENV MALLOC_ARENA_MAX=2
ENV PYTHONUNBUFFERED=1
ENV OMP_NUM_THREADS=1
ENV MKL_NUM_THREADS=1

EXPOSE 8080
CMD ["python", "src/main.py"]
```

```python
# health_check.py
import requests,time
def check_model_health():
    start=time.time()
    resp=requests.get("http://localhost:8080/health",timeout=5)
    lat=time.time()-start
    return resp.status_code==200 and lat<=0.5
```

Kubernetes Config

- Rolling updates, max surge/unavailable
- Resource requests/limits for stable performance
- Prometheus annotations for observability
- PVC for model storage

Kubernetes Config (Part 1)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-service
  labels:
    app: model-service
    environment: production
spec:
  replicas: 3
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: model-service
```

Kubernetes Config (Part 2)

```yaml
template:
  metadata:
    labels:
      app: model-service
    annotations:
      prometheus.io/scrape: "true"
      prometheus.io/port: "8080"
  spec:
    securityContext:
      runAsNonRoot: true
      runAsUser: 1000
    containers:
    - name: model-server
      image: model-service:latest
      ports:
      - containerPort: 8080
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
        initialDelaySeconds: 30
        periodSeconds: 30
```

Kubernetes Config (Part 3)

```yaml
  readinessProbe:
    httpGet:
      path: /ready
      port: 8080
    initialDelaySeconds: 15
    periodSeconds: 15
  env:
  - name: MODEL_PATH
    value: "/models/current"
  - name: MONITORING_PORT
    value: "8080"
  volumeMounts:
  - name: model-store
    mountPath: "/models"
    readOnly: true
volumes:
- name: model-store
  persistentVolumeClaim:
    claimName: model-store-pvc
```

Monitoring System

- Prometheus metrics for latency, errors, drift
- Accuracy & fairness gauges for ethical alignment
- Rapid alerting for anomalies

Monitoring Code (Part 1)

```python
import numpy as np
import prometheus_client as prom
from typing import Dict

class MLMetrics:
    def __init__(self):
        self.prediction_latency=prom.Histogram(
            'prediction_latency_seconds','Time for inference',
            buckets=np.logspace(-3,2,20)
        )
        self.prediction_errors=prom.Counter(
            'prediction_errors_total','Total prediction errors',['error_type']
        )
        self.feature_drift=prom.Gauge(
            'feature_drift_score','Feature drift',['feature_name']
        )
        self.model_accuracy=prom.Gauge('model_accuracy','Current model accuracy')
        self.fairness_score=prom.Gauge(
            'fairness_score','Model fairness',['protected_group']
        )
```

Monitoring Code (Part 2)

```python
class ModelMonitor:
    def __init__(self, metrics: MLMetrics):
        self.metrics = metrics
        self.baseline_distributions = {}

    def record_prediction(self, features: Dict, prediction: float,
                          latency: float, error: str=None):
        self.metrics.prediction_latency.observe(latency)
        if error:
            self.metrics.prediction_errors.labels(error_type=error).inc()
        for fname,val in features.items():
            drift=self._compute_drift(fname,val)
            self.metrics.feature_drift.labels(feature_name=fname).set(drift)

    def update_fairness_metrics(self, predictions, protected_attrs: Dict[str,np.ndarray]):
        for group,mask in protected_attrs.items():
            score = self._calculate_fairness(predictions, mask)
            self.metrics.fairness_score.labels(protected_group=group).set(score)

    def _compute_drift(self, fname, val):
        return 0.0 # placeholder

    def _calculate_fairness(self, predictions, mask):
        return 0.95 # placeholder
```

Automated Rollback

- Checks health metrics against thresholds
- If violated, initiates Kubernetes rollback
- Ensures continuous quality under changing conditions

Rollback Code (Part 1)

```python
import kubernetes as k8s
import logging

class RollbackConfig:
    def __init__(self,error_threshold,latency_threshold,fairness_threshold):
        self.error_threshold=error_threshold
        self.latency_threshold=latency_threshold
        self.fairness_threshold=fairness_threshold

class RollbackManager:
    def __init__(self, config: RollbackConfig, namespace='default'):
        self.config = config
        self.k8s_client = k8s.client.AppsV1Api()
        self.namespace = namespace

    def check_health(self, metrics):
        if metrics['error_rate']>self.config.error_threshold:
            return False,'High error rate'
        if metrics['p95_latency']>self.config.latency_threshold:
            return False,'High latency'
        if metrics['fairness_score']<self.config.fairness_threshold:
            return False,'Fairness violation'
        return True,None
```

Rollback Code (Part 2)

```python
def initiate_rollback(self, deployment_name: str, reason: str):
    try:
        deployment=self.k8s_client.read_namespaced_deployment(
            deployment_name,self.namespace)
        revisions=self.k8s_client.list_namespaced_replica_set(
            self.namespace,label_selector=f"app={deployment_name}")
        last_good=self._find_last_good_revision(revisions.items)
        if not last_good:
            logging.error("No good revision found")
            return False
        body={
          "kind": "DeploymentRollback",
          "apiVersion": "apps/v1",
          "name": deployment_name,
          "rollbackTo": {"revision":last_good.metadata.annotations['revision']}
        }
        self.k8s_client.create_namespaced_deployment_rollback(
            deployment_name,self.namespace,body)
        logging.info(f"Rollback initiated: {reason}")
        return True
    except k8s.client.rest.ApiException as e:
        logging.error(f"Rollback failed: {str(e)}")
        return False

def _find_last_good_revision(self,rs_list):
    return rs_list[-1] if rs_list else None
```

Model Serving API

- FastAPI for performance & doc generation
- Request validation ensures data quality
- Integrated monitoring and latency tracking
- Handles single & batch predictions

Serving API Code (Part 1)

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, validator
from typing import List, Dict
import numpy as np
import time

app = FastAPI()
metrics = MLMetrics() # from previous code
model = load_model()  # placeholder

class PredictionRequest(BaseModel):
    features: Dict[str,float]
    request_id: str

    @validator('features')
    def validate_features(cls,v):
        required={'feature1','feature2','feature3'}
        if missing:=required-set(v):
            raise ValueError(f"Missing: {missing}")
        return v
```

Serving API Code (Part 2)

```python
def record_latency(func):
    def wrapper(*args,**kwargs):
        start=time.time()
        resp=func(*args,**kwargs)
        duration=time.time()-start
        metrics.prediction_latency.observe(duration)
        return resp
    return wrapper

@app.post("/predict")
@record_latency
async def predict(request: PredictionRequest):
    try:
        features=np.array([request.features[f] for f in sorted(request.features)])
        pred=model.predict(features.reshape(1,-1))
        return {"prediction":float(pred),"request_id":request.request_id}
    except Exception as e:
        metrics.prediction_errors.labels(error_type="prediction").inc()
        raise HTTPException(status_code=500,detail=str(e))
```

Serving API Code (Part 3)

```python
@app.post("/batch_predict")
@record_latency
async def batch_predict(requests:List[PredictionRequest]):
    if len(requests)>100:
        raise HTTPException(400,"Batch size too large")

    feats=np.array([[r.features[f] for f in sorted(r.features)] for r in requests])
    try:
        preds=model.predict(feats)
        return [
          {"prediction":float(p),"request_id":r.request_id}
          for p,r in zip(preds,requests)
        ]
    except Exception as e:
        metrics.prediction_errors.labels(error_type="batch").inc()
        raise HTTPException(500,detail=str(e))
```

Production MLOps Best Practices

- System Design: Graceful degradation, circuit breakers, zero-downtime
- Monitoring: Business & tech metrics, robust logging, model performance
- Security & Compliance: Regular audits, automated checks, incident response
- Operational Excellence: Automate routine tasks, document processes, test disaster recovery

Next Steps

- Advanced Deployment: A/B testing, canary analysis, shadow deployments
- Enhanced Monitoring: Custom dashboards, predictive maintenance, performance reports
- More Automation: Self-healing capabilities, automated documentation, routine ops automation