Machine Learning in Production

Planning for Operations

Operations

Fundamentals of Engineering AI-Enabled Systems

Holistic system view: AI and non-AI components, pipelines, stakeholders, environment interactions, feedback loops

Requirements:
System and model goals
User requirements
Environment assumptions
Quality beyond accuracy
Measurement
Risk analysis
Planning for mistakes

Architecture + design:
Modeling tradeoffs
Deployment architecture
Data science pipelines
Telemetry, monitoring
Anticipating evolution
Big data processing
Human-AI design

Quality assurance:
Model testing
Data quality
QA automation
Testing in production
Infrastructure quality
Debugging

Operations:
Continuous deployment
Contin. experimentation
Configuration mgmt.
Monitoring
Versioning
Big data
DevOps, MLOps

Teams and process: Data science vs software eng. workflows, interdisciplinary teams, collaboration points, technical debt

Responsible AI Engineering

Provenance, versioning, reproducibility

Safety

Security and privacy

Fairness

Interpretability and explainability

Transparency and trust

Ethics, governance, regulation, compliance, organizational culture

Readings

Required reading: Shankar, Shreya, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. "Operationalizing machine learning: An interview study." arXiv preprint arXiv:2209.09125 (2022).

Recommended readings:

- O'Leary, Katie, and Makoto Uchida. "Common problems with Creating Machine Learning Pipelines from Existing Code." Proc. Conference on Machine Learning and Systems (MLSys) (2020).

Learning Goals

- Deploy a service for models using container infrastructure
- Automate common configuration management tasks
- Devise a monitoring strategy and suggest suitable components for implementing it
- Diagnose common operations problems
- Understand the typical concerns and concepts of MLOps

Running Example: Blogging Platform with Spam Filter

"Operations"

Operations

Provision and monitor the system in production, respond to problems

Avoid downtime, scale with users, manage operating costs

Heavy focus on infrastructure

Traditionally sysadmin and hardware skills

Service Level Objectives

Quality requirements in operations, such as

- maximum latency
- minimum system throughput
- targeted availability/error rate
- time to deploy an update
- durability for storage

Each with typical measures

For the system as a whole or individual services

Example Service Level Objectives?

Operators on a Team

Operators cannot work in isolation

Rely on developers for software quality and performance

Negotiate service level agreements and budget (e.g., 99.9% vs 99.99% availability)

Risk management role (not risk avoidance)

Operations and ML

ML has distinct workloads and hardware requirements

Deep learning often pushes scale boundaries

Regular updates or learning in production

Common Themes

Observability is essential

Release management and automated deployments

Infrastructure as code and virtualization

Scaling deployments

Incident response planning

Dev vs. Ops

Common Release Problems?

Common Release Problems?

Common Release Problems (Examples)

- Missing dependencies
- Different compiler versions or library versions
- Different local utilities (e.g. unix grep vs mac grep)
- Database problems
- OS differences
- Too slow in real settings
- Difficult to roll back changes
- Source from many different repositories
- Obscure hardware? Cloud? Enough memory?

Developers

- Coding
- Testing, static analysis, reviews
- Continuous integration
- Bug tracking
- Running local tests and scalability experiments
- ...

Operations

- Allocating hardware resources
- Managing OS updates
- Monitoring performance
- Monitoring crashes
- Managing load spikes, ...
- Tuning database performance
- Running distributed at scale
- Rolling back releases
- ...

QA responsibilities in both roles

Quality Assurance does not stop in Dev

- Ensuring product builds correctly (e.g., reproducible builds)
- Ensuring scalability under real-world loads
- Supporting environment constraints from real systems (hardware, software, OS)
- Efficiency with given infrastructure
- Monitoring (server, database, Dr. Watson, etc)
- Bottlenecks, crash-prone components, … (possibly thousands of crash reports per day/minute)

DevOps

CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

Key ideas and principles

- Better coordinate between developers and operations (collaborative)
- Key goal: Reduce friction bringing changes from development into production
- Considering the *entire tool chain* into production (holistic)
- Documentation and versioning of all dependencies and configurations ("configuration as code")
- Heavy automation, e.g., continuous delivery, monitoring
- Small iterations, incremental and continuous releases

- Buzz word!

CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

Common Practices

All configurations in version control

Test and deploy in containers

Automated testing, testing, testing, ...

Monitoring, orchestration, and automated actions in practice

Microservice architectures

Release frequently

Heavy tooling and automation

Heavy tooling and automation -- Examples

- Infrastructure as code — Ansible, Terraform, Puppet, Chef
- CI/CD — Jenkins, TeamCity, GitLab, Shippable, Bamboo, Azure DevOps
- Test automation — Selenium, Cucumber, Apache JMeter
- Containerization — Docker, Rocket, Unik
- Orchestration — Kubernetes, Swarm, Mesos
- Software deployment — Elastic Beanstalk, Octopus, Vamp
- Measurement — Datadog, DynaTrace, Kibana, NewRelic, ServiceNow

Continuous Delivery

Manual Release Pipelines

Source: https://www.slideshare.net/jmcgarr/continuous-delivery-at-netflix-and-beyond

Continuous Integr.

- Automate tests after commit
- Independent test infrastructure

Continuous Delivery

- Full automation from commit to deployable container
- Heavy focus on testing, reproducibility and rapid feedback, creates transparency

Continuous Deployment

- Full automation from commit to deployment
- Empower developers, quick to production
- Encourage experimentation and fast incremental changes
- Commonly integrated with monitoring and canary releases

Automate Everything

Example: Facebook Tests for Mobile Apps

- Unit tests (white box)
- Static analysis (null pointer warnings, memory leaks, ...)
- Build tests (compilation succeeds)
- Snapshot tests (screenshot comparison, pixel by pixel)
- Integration tests (black box, in simulators)
- Performance tests (resource usage)
- Capacity and conformance tests (custom)

Further readings: Rossi, Chuck, Elisa Shibley, Shi Su, Kent Beck, Tony Savor, and Michael Stumm. Continuous deployment of mobile software at facebook (showcase). In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 12-23. ACM, 2016.

Release Challenges for Mobile Apps

- Large downloads
- Download time at user discretion
- Different versions in production
- Pull support for old releases?

- Server side releases silent and quick, consistent

- -> App as container, most content + layout from server

Real-world pipelines are complex

Containers and Configuration Management

Containers

- Lightweight virtual machine
- Contains entire runnable software, incl. all dependencies and configurations
- Used in development and production
- Sub-second launch time
- Explicit control over shared disks and network connections

Docker Example

```
FROM ubuntu:latest
MAINTAINER ...
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Source: http://containertutorials.com/docker-compose/flask-simple-app.html

Common configuration management questions

What runs where?

How are machines connected?

What (environment) parameters does software X require?

How to update dependency X everywhere?

How to scale service X?

Ansible Examples

- Software provisioning, configuration mgmt., and deployment tool
- Apply scripts to many servers

```
[webservers]
web1.company.org
web2.company.org
web3.company.org

[dbservers]
db1.company.org
db2.company.org

[replication_servers]
...
```

```
# This role deploys the mongod processe
- name: create data directory for mongo
  file: path={{ mongodb_datadir_prefix
  delegate_to: '{{ item }}'
  with_items: groups.replication_serve

- name: create log directory for mongo
  file: path=/var/log/mongo state=direc

- name: Create the mongodb startup file
  template: src=mongod.j2 dest=/etc/ini
```

Puppet Example

Declarative specification, can be applied to many machines

```
$doc_root = "/var/www/example"

exec { 'apt-get update':
 command => '/usr/bin/apt-get update'
}

package { 'apache2':
 ensure  => "installed",
 require => Exec['apt-get update']
}
```

Speaker notes

source: https://www.digitalocean.com/community/tutorials/configuration-management-101-writing-puppet-manifests

Container Orchestration with Kubernetes

Manages which container to deploy to which machine

Launches and kills containers depending on load

Manage updates and routing

Automated restart, replacement, replication, scaling

Kubernetis master controls many nodes

Substantial complexity and learning curve

CC BY-SA 4.0 Khtan66

Monitoring

- Monitor server health
- Monitor service health
- Monitor telemetry (see past lecture)
- Collect and analyze measures or log files
- Dashboards and triggering automated decisions

- Many tools, e.g., Grafana as dashboard, Prometheus for metrics, Loki + ElasticSearch for logs
- Push and pull models

Hawkular APM
localhost:8080/hawkular-ui/apm/components?main-tab=apm
HAWKULAR APPLICATION PERFORMANCE MANAGEMENT
Components
Distributed Tracing
Business Transactions
Filter by
Time Span
10 Minutes
Business Transaction
All
List My Orders
Place Order
Properties
Name
Text
Contains text
Host Name
Enter a host name
Aggregation Interval:
10 Second
Overlap Data
Last Update: 12 Oct 2016 14:35:32
Pause Live Data
Seconds
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
2016-10-12 14:25:30
2016-10-12 14:26:50
2016-10-12 14:28:10
2016-10-12 14:29:30
2016-10-12 14:30:50
2016-10-12 14:32:10
2016-10-12 14:31:10
database 0.000014925
Consumer 0.000111111
Producer 0.000131147
database
Consumer
Producer
Actual (secs)
Elapsed (secs)
Count
Component
URI
Operation
0.000 0.001 1320 consumer /orders POST
0.000 0.001 140 consumer /orders GET
0.000 0.000 1320 consumer GetAccount
0.000 0.000 1102 consumer GetItem

The DevOps Mindset

- Consider the entire process and tool chain holistically
- Automation, automation, automation
- Elastic infrastructure
- Document, test, and version everything
- Iterate and release frequently
- Emphasize observability
- Shared goals and responsibilities

https://ml-ops.org/

On Terminology

- Many vague buzzwords, often not clearly defined
- **MLOps:** Collaboration and communication between data scientists and operators, e.g.,
  - Automate model deployment
  - Model training and versioning infrastructure
  - Model deployment and monitoring
- **AIOps:** Using AI/ML to make operations decision, e.g. in a data center
- **DataOps:** Data analytics, often business setting and reporting
  - Infrastructure to collect data (ETL) and support reporting
  - Combines agile, DevOps, Lean Manufacturing ideas

MLOps Overview

Integrate ML artifacts into software release process, unify process (i.e., DevOps extension)

Automated data and model validation (continuous deployment)

Continuous deployment for ML models: from experimenting in notebooks to quick feedback in production

Versioning of models and datasets (more later)

Monitoring in production (discussed earlier)

Further reading: MLOps principles

Tooling Landscape LF AI

MLOps Goals and Principles

Like DevOps: Automation, testing, holistic, observability, teamwork

Supporting frequent experimentation, rapid prototyping, and constant iteration

3V: Velocity, Validation, Versioning

MLOps Tools -- Examples

- Model registry, versioning and metadata: MLFlow, Neptune, ModelDB, WandB, ...
- Model monitoring: Fiddler, Hydrosphere
- Data pipeline automation and workflows: DVC, Kubeflow, Airflow
- Model packaging and deployment: BentoML, Cortex
- Distributed learning and deployment: Dask, Ray, ...
- Feature store: Feast, Tecton
- Integrated platforms: Sagemaker, Valohai, ...
- Data validation: Cerberus, Great Expectations, ...

Long list: https://github.com/kelvins/awesome-mlops

MLOps Common Goals

Enable experimentation with data and models, small incremental changes; hide complexity from data scientists

Automate (nuanced) model validation (like CI) and integrate with testing in production (monitoring)

Dynamic view of constantly evolving training and test data; invest in data validation

Version data, models; track experiment results

Recall: DevOps Mindset

- Consider the entire process and tool chain holistically
- Automation, automation, automation
- Elastic infrastructure
- Document, test, and version everything
- Iterate and release frequently
- Emphasize observability
- Shared goals and responsibilities

Breakout: MLOps Goals

For the blog spam filter scenario, consider DevOps and MLOps infrastructure (CI, CD, containers, config. mgmt, monitoring, model registry, pipeline automation, feature store, data validation, ...)

As a group, tagging group members, post to `#lecture`:

- *Which DevOps or MLOps goals to prioritize?*
- *Which tools to try?*

Incident Response Planning

Mistakes will Happen. Be Prepared

Even with careful anticipation and mitigation, mistakes will happen

Anticipated or not

ML as unreliable component raises risks

Design mitigations help avoid anticipated mistakes

Incident response plan prepares for unanticipated or unmitigated mistakes

Incident Response Plan

- Provide contact channel for problem reports
- Have expert on call
- Design process for anticipated problems, e.g., rollback, reboot, takedown
- Prepare for recovery
- Proactively collect telemetry
- Investigate incidents
- Plan public communication (responsibilities)

Incident Resp. Plan for Blog's Spam Filter?

Excursion: Organizational Culture

Organizational Culture

“this is how we always did things”

Implicit and explicit assumptions and rules guiding behavior

Often grounded in history, very difficult to change

Examples:

- Move fast and break things
- Privacy first
- Development opportunities for all employees

Source: Bonkers World

Organizational Culture

Levels of Organizational Culture

Artifacts -- What we see

- Behaviors, systems, processes, policies

Espoused Values -- What we say

- Ideals, goals, values, aspirations

Basic assumptions -- What we believe

- Underlying assumptions, "old ways of doing things", unconsciously taken for granted

Iceberg models: Only artifacts and espoused values visible, but practices driven by invisible basic assumptions

Culture Change

Changing organizational culture is very difficult

Top down: espoused values, management buy in, incentives

Bottom up: activism, show value, spread

Examples of success of failure stories?

MLOps Culture

Dev with Ops instead of Dev vs Ops

A culture of collaboration, joint goals, joint responsibilities

Artifacts: Joint tools, processes

Underlying assumptions: Devs provide production-ready code; Ops focus on value, automation is good, observability is important, ...

Resistance to DevOps Culture?

From "us vs them" to blameless culture -- How?

Introduction of new tools and processes -- Disruptive? Costly? Competing with current tasks? Who wants to write tests?

Future benefits from rapid feedback and telemetry -- Unrealistic?

Automation and shifting responsibilities -- Hiring freeze and layoffs?

Past experience with poor adoption -- All costs, no benefits? Compliance only?

Successful DevOps Adoption

Need supportive management; typically driven by advocacy of individuals, convincing colleagues and management

Education to generate buy-in

Experts and consultants can help with initial costly transition

Demonstrate benefits on small project, promote afterward

Focus on key bottlenecks, over perfect adoption (e.g., prioritize experimentation, test automation, rapid feedback with telemetry)

Luz, Welder Pinheiro, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study." Journal of Systems and Software 157 (2019): 110384.

Summary

- Plan for change, plan for operations
- Operations requirements: service level objectives
- DevOps integrations development and operations tasks with joint goals and tools
  - Heavy automation
  - Continuous integration and continuous delivery
  - Containers and configuration management
  - Monitoring
- MLOps extends this to operating pipelines and deploying models
- Organizational culture is slow and difficult to change

Further Reading

- Shankar, Shreya, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. "Operationalizing machine learning: An interview study." arXiv preprint arXiv:2209.09125 (2022).
- https://ml-ops.org/
- Beyer, Betsy, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. Site reliability engineering: How Google runs production systems. O'Reilly, 2016.
- Kim, Gene, Jez Humble, Patrick Debois, John Willis, and Nicole Forsgren. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution, 2nd ed, 2021.
- Treveil, Mark, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. Introducing MLOps: How to Scale Machine Learning in the Enterprise. O'Reilly, 2020.
- Luz, Welder Pinheiro, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study." Journal of Systems and Software 157 (2019): 110384.
- Schein, Edgar H. *Organizational culture and leadership*. 5th ed. John Wiley & Sons, 2016.