



# Machine Learning in Production Infrastructure Quality & MLOps



# Infrastructure and 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: Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)

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).
- Larysa Visengeriyeva. [Machine Learning Operations - A Reading List](#), InnoQ 2020

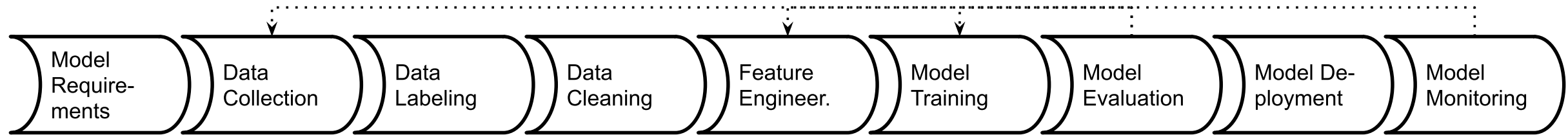
# Learning Goals

- Implement and automate tests for all parts of the ML pipeline
- Understand testing opportunities beyond functional correctness
- Automate test execution with continuous integration
- 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



# Beyond Model and Data Quality

# Possible Mistakes in ML Pipelines



Danger of "silent" mistakes in many phases

**Examples?**



# Possible Mistakes in ML Pipelines

Danger of "silent" mistakes in many phases:

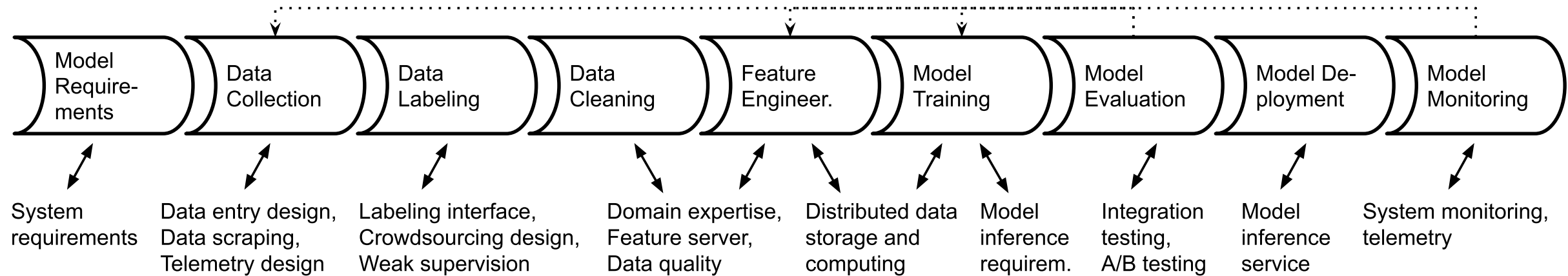
- Dropped data after format changes
- Failure to push updated model into production
- Incorrect feature extraction
- Use of stale dataset, wrong data source
- Data source no longer available (e.g web API)
- Telemetry server overloaded
- Negative feedback (telemtr.) no longer sent from app
- Use of old model learning code, stale hyperparameter
- Data format changes between ML pipeline steps

# Building Robust Pipeline Automation

- Support experimentation and evolution
  - Automate
  - Design for change
  - Design for observability
  - Testing the pipeline for robustness
- Thinking in pipelines, not models
- Integrating the Pipeline with other Components



# Integrating the Pipeline with other Components



# Pipelines are Code

From experimental notebook code to production code

Each stage as a function or module

Well tested in isolation and together

Robust to changes in inputs (automatically adapt or crash, no silent mistakes)

Use good engineering practices (version control, documentation, testing, naming, code review)



# Everything can be tested?



Many qualities can be tested beyond just functional correctness (for a specification). Examples: Performance, model quality, data quality, usability, robustness, ... not all tests are equality easy to automate



# Testing Strategies

- Performance
- Scalability
- Robustness
- Safety
- Security
- Extensibility
- Maintainability
- Usability

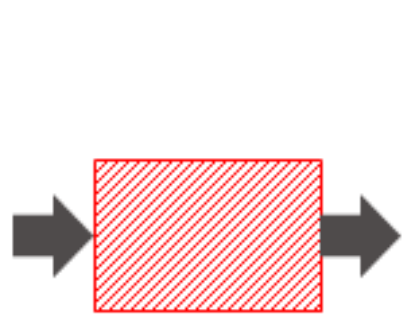
**How to test for these? How automatable?**

# Test Automation

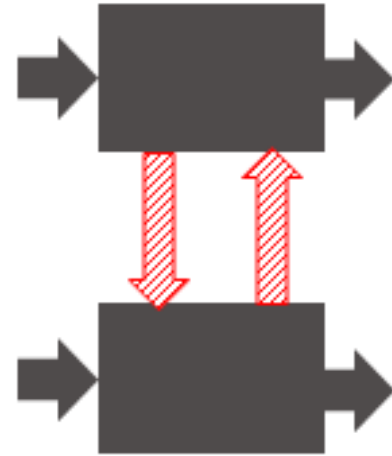
# From Manual Testing to Continuous Integration



# Unit Test, Integration Tests, System Tests



Unit testing



Integration testing



System testing



Acceptance  
testing  
(Demonstration)



Software is developed in units that are later assembled. Accordingly we can distinguish different levels of testing.

Unit Testing - A unit is the "smallest" piece of software that a developer creates. It is typically the work of one programmer and is stored in a single file. Different programming languages have different units: In C++ and Java the unit is the class; in C the unit is the function; in less structured languages like Basic and COBOL the unit may be the entire program.

Integration Testing - In integration we assemble units together into subsystems and finally into systems. It is possible for units to function perfectly in isolation but to fail when integrated. For example because they share an area of the computer memory or because the order of invocation of the different methods is not the one anticipated by the different programmers or because there is a mismatch in the data types. Etc.

System Testing - A system consists of all of the software (and possibly hardware, user manuals, training materials, etc.) that make up the product delivered to the customer. System testing focuses on defects that arise at this highest level of integration. Typically system testing includes many types of testing: functionality, usability, security, internationalization and localization, reliability and availability, capacity, performance, backup and recovery, portability, and many more.

Acceptance Testing - Acceptance testing is defined as that testing, which when completed successfully, will result in the customer accepting the software and giving us their money. From the customer's point of view, they would generally like the most exhaustive acceptance testing possible (equivalent to the level of system testing). From the vendor's point of view, we would generally like the minimum level of testing possible that would result in money changing hands. Typical strategic questions that should be addressed before acceptance testing are: Who defines the level of the acceptance testing? Who creates the test scripts? Who executes the tests? What is the pass/fail criteria for the acceptance test? When and how do we get paid?

# Anatomy of a Unit Test

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class AdjacencyListTest {
    @Test
    public void testSanityTest(){
        // set up
        Graph g1 = new AdjacencyListGraph(10);
        Vertex s1 = new Vertex("A");
        Vertex s2 = new Vertex("B");
        // check expected results (oracle)
```

# Ingredients to a Test

Specification

Controlled environment

Test inputs (calls and parameters)

Expected outputs/behavior (oracle)

# Unit Testing Pitfalls

Working code, failing tests

"Works on my machine"

Tests break frequently

**How to avoid?**



# How to unit test component with dependency on other code?



# How to Test Parts of a System?



```
Model learn() {  
    Stream stream = openKafkaStream(...)  
    DataTable output = getData(testStream,  
                                new DefaultCleaner());  
    return Model.learn(output);  
}
```

# Automating Test Execution



```
DataTable getData(Stream stream, DataCleaner cleaner) { ... }

@Test void test() {
    Stream stream = openKafkaStream(...)
    DataTable output = getData(stream,
                                new DefaultCleaner());
    assert(output.length==10)
}
```

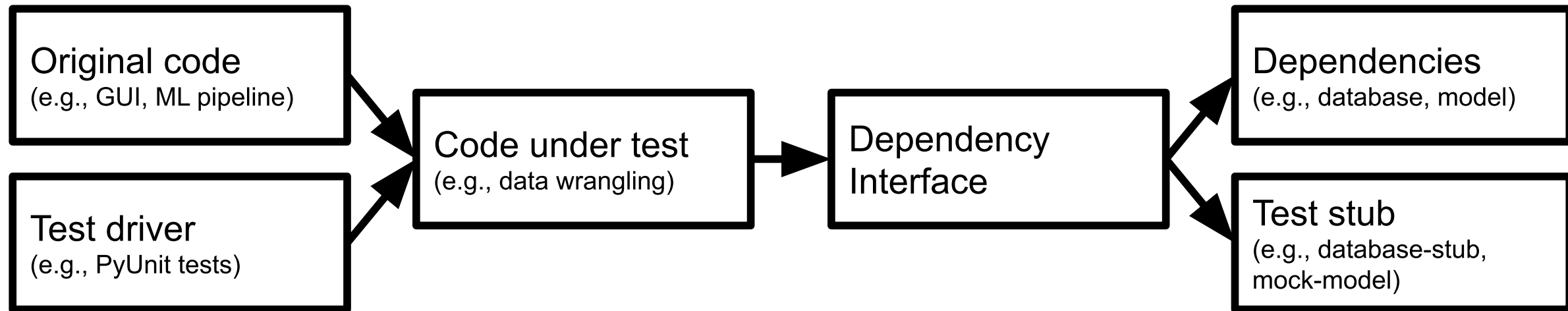
# Decoupling from Dependencies



```
DataTable getData(Stream stream, DataCleaner cleaner) { ... }  
Stream testStream = new Stream() {  
    int idx = 0;  
    // hardcoded or read from test file  
    String[] data = [ ... ]  
    public void connect() { }  
    public String getNext() {  
        return data[++idx];  
    }  
}
```



# General Testing Strategy: Decoupling Code Under Test



# Example: Mocking a DataCleaner Object

```
DataTable getData(KafkaStream stream, DataCleaner cleaner){...  
  
@Test void test() {  
    DataCleaner dummyCleaner = new DataCleaner() {  
        boolean isValid(String row) { return true; }  
        ...  
    }  
    DataTable output = getData(testStream, dummyCleaner);  
    assert(output.length==10)  
}
```

# Example: Mocking a DataCleaner Object

```
DataTable getData(KafkaStream stream, DataCleaner cleaner){...  
  
@Test void test() {  
    DataCleaner dummyCleaner = new DataCleaner() {  
        int counter = 0;  
        boolean isValid(String row) {  
            counter++;  
            return counter!=3;  
        }  
        ...  
    }  
}
```

Mocking frameworks provide infrastructure for expressing such tests compactly.

# Subtle Bugs in Data Wrangling Code

```
df['Join_year'] = df.Joined.dropna().map(  
    lambda x: x.split(',')[1].split(' ')[1])
```

```
df.loc[idx_nan_age, 'Age'].loc[idx_nan_age] =  
    df['Title'].loc[idx_nan_age].map(map_means)
```

```
df["Weight"].astype(str).astype(int)
```

# Subtle Bugs in Data Wrangling Code (continued)

```
df['Reviws'] = df['Reviews'].apply(int)
```

```
df["Release Clause"] =  
    df["Release Clause"].replace(regex=['k'], value='000')
```

```
df["Release Clause"] =  
    df["Release Clause"].astype(str).astype(float)
```



1 attempting to remove na values from column, not table

2 loc[] called twice, resulting in assignment to temporary column only

3 astype() is not an in-place operation

4 typo in column name

5&6 modeling problem (k vs K)

# Tests for Data Wrangling Code?

(data quality checks, data cleaning, feature engineering, ...)



# Modularizing and Testing Data Cleaning

```
def is_valid_row(row):  
    try:  
        datetime.strptime(row['date'], '%b %d %Y')  
        return True  
    except ValueError:  
        return False
```

```
@test  
def test_dates(self):  
    self.assertTrue(is_valid_row(...))  
    self.assertTrue(is_valid_row(...))  
    self.assertFalse(is_valid_row(...))
```

# Modularize and Test Feature Encoding

```
def encode_date(df):  
    df.date_time = pd.to_datetime(df.date_time)  
def encode_day_part(df):  
    def daypart(hour):  
        if hour in [2,3,4,5]:  
            return "dawn"  
        elif hour in [6,7,8,9]:  
            return "morning"  
        elif hour in [10,11,12,13]:  
            return "noon"  
        elif ...
```

```
@test  
def test_day_part(self): ...
```

# Test Error Handling

```
@Test void missingDataErrorTest() {  
    DataTable missingData = new DataTable();  
    try {  
        Model m = learn(missingData);  
        Assert.fail();  
    } catch (NoDataException e) { /* correctly thrown */ }  
}
```

*(This tests expects the learn function to throw an exception. Test fails if no exception thrown.)*

Speaker notes

Code to test that the right exception is thrown





# Testing for Robustness

```
Stream faultyTestStream = new Stream() {  
    ...  
    public String getNext() {  
        if (++idx == 3) throw new IOException();  
        return data[++idx];  
    }  
}  
@Test void retryOnStreamProblemTest() {  
    DataTable output = retry(getData(faultyTestStream, ...));  
    assert(output.length==10)  
}
```

*(manipulating the (controlled) environment: injecting errors into backend to test error handling)*

# Test *Modular* Error Handling

```
Stream faultyTestStream = new Stream() {  
    int idx = 0;  
    public void connect() {  
        if (++idx < 3) throw new IOException(  
            "cannot establish connection")  
        }  
    public String getNext() { ... }  
}  
@Test void integrationTest() {  
    DataLoader loader = new DataLoader(faultyTestStream,  
                                       new DefaultCleaner());
```

*(Modular protection/isolation: Ensure error is handled locally and does not propagate to other modules.)*

Test that errors are correctly handled within a module and do not leak

## Packages

[All](#)

[net.sourceforge.cobertura.ant](#)  
[net.sourceforge.cobertura.check](#)  
[net.sourceforge.cobertura.coveragedata](#)  
[net.sourceforge.cobertura.instrument](#)  
[net.sourceforge.cobertura.merge](#)  
[net.sourceforge.cobertura.reporting](#)  
[net.sourceforge.cobertura.reporting.html](#)  
[net.sourceforge.cobertura.reporting.html.files](#)  
[net.sourceforge.cobertura.reporting.xml](#)  
[net.sourceforge.cobertura.util](#)

## All Packages

## Classes

[AntUtil](#) (88%)  
[Archive](#) (100%)  
[ArchiveUtil](#) (80%)  
[BranchCoverageData](#) (N/A)  
[CheckTask](#) (0%)  
[ClassData](#) (N/A)  
[ClassInstrumenter](#) (94%)  
[ClassPattern](#) (100%)  
[CoberturaFile](#) (73%)  
[CommandLineBuilder](#) (96%)  
[CommonMatchingTask](#) (88%)  
[ComplexityCalculator](#) (100%)  
[ConfigurationUtil](#) (50%)  
[CopyFiles](#) (87%)  
[CoverageData](#) (N/A)  
[CoverageDataContainer](#) (N/A)  
[CoverageDataFileHandler](#) (N/A)  
[CoverageRate](#) (0%)  
[ExcludeClasses](#) (100%)  
[FileFinder](#) (96%)  
[FileLocker](#) (0%)  
[FirstPassMethodInstrumenter](#) (100%)

## Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2179	64%	472/738	2.319
<a href="#">net.sourceforge.cobertura.ant</a>	11	52%	170/330	43%	40/94	1.848
<a href="#">net.sourceforge.cobertura.check</a>	3	0%	0/150	0%	0/76	2.429
<a href="#">net.sourceforge.cobertura.coveragedata</a>	13	N/A	N/A	N/A	N/A	2.277
<a href="#">net.sourceforge.cobertura.instrument</a>	10	90%	460/510	75%	123/164	1.854
<a href="#">net.sourceforge.cobertura.merge</a>	1	86%	30/35	88%	14/16	5.5
<a href="#">net.sourceforge.cobertura.reporting</a>	3	87%	116/134	80%	43/54	2.882
<a href="#">net.sourceforge.cobertura.reporting.html</a>	4	91%	475/523	77%	156/202	4.444
<a href="#">net.sourceforge.cobertura.reporting.html.files</a>	1	87%	39/45	62%	5/8	4.5
<a href="#">net.sourceforge.cobertura.reporting.xml</a>	1	100%	155/155	95%	21/22	1.524
<a href="#">net.sourceforge.cobertura.util</a>	9	60%	175/291	69%	70/102	2.892
<a href="#">someotherpackage</a>	1	83%	5/6	N/A	N/A	1.2

Report generated by [Cobertura](#) 1.9 on 6/9/07 12:37 AM.

# Testable Code

Think about testing when writing code

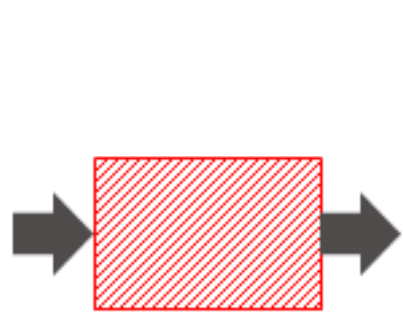
Unit testing encourages you to write testable code

Separate parts of the code to make them independently testable

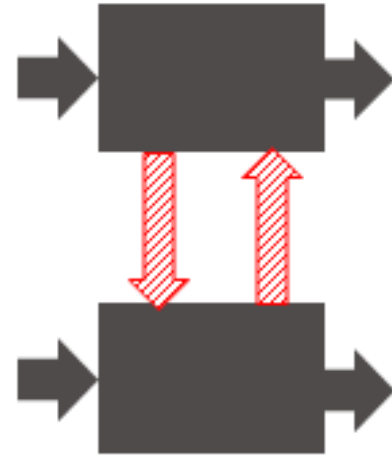
Abstract functionality behind interface, make it replaceable

Bonus: Test-Driven Development is a design and development method in which you *always* write tests *before* writing code

# Integration and system tests



Unit testing



Integration testing



System testing



Acceptance  
testing  
(Demonstration)



# Integration and system tests

Test larger units of behavior

Often based on use cases or user stories -- customer perspective

```
@Test void gameTest() {  
    Poker game = new Poker();  
    Player p = new Player();  
    Player q = new Player();  
    game.shuffle(seed)  
    game.add(p);  
    game.add(q);  
    game.deal();  
    p.bet(100);  
    q.bet(100);  
}
```

# Integration and system tests

Test larger units of behavior

Often based on use cases or user stories -- customer perspective

```
@Test void testCleaningWithFeatureEng() {  
    DataFrame d = loadTestData();  
    DataFrame cd = clean(d);  
    DataFrame f = feature3.encode(cd);  
    assert(noMissingValues(f.getColumn("m")));  
    assert(max(f.getColumn("m")) <= 1.0);  
}
```

# Data Pipeline Integration Test

```
@Test void integrationTest() {  
    DataLoader loader = new DataLoader(testStream,  
                                     new DefaultCleaner());  
    ModelBuilder model = new ModelBuilder(loader, ...);  
    // assume all exceptions are handled correctly internally  
    assert(model.accuracy > .91)  
}
```

# Build systems & Continuous Integration

Automate all build, analysis, test, and deployment steps from a command line call

Ensure all dependencies and configurations are defined

Ideally reproducible and incremental

Distribute work for large jobs

Track results

**Key CI benefit: Tests are regularly executed, part of process**

Jonathan

Build #17 - wyvernlang x


← → ↺ 🏠

https://travis-ci.org/wyvernlang/wyvern/builds/79099642

☆ ☰

Travis CI

Blog Status Help

Jonathan Aldrich 


Search all repositories 🔍

My Repositories +

✓ wyvernlang/wyvern #17


🕒 Duration: 16 sec

📅 Finished: 3 days ago

wyvernlang / wyvern  build passing


Current Branches Build History Pull Requests > Build #17 


⚙️ Settings ▾

✓ 

SimpleWyvern-devel Asserting false (works on Linux, so its OK).

# 17 passed

 Commit fd7be1c

 Compare 0e2af1f..fd7be1c

🕒 ran for 16 sec

📅 3 days ago

🔄

🕒 potanin authored and committed

📌 This job ran on our legacy infrastructure. Please read [our docs on how to upgrade](#)

✖ Remove Log

⬇ Download Log

1 Using worker: worker-linux-027f0490-1.bb.travis-ci.org:travis-linux-2

2

3 Build system information 

system\_info

67

68 \$ git clone --depth=50 --branch=SimpleWyvern-devel 

git.checkout

 0.81s

78 \$ jdk\_switcher use oraclejdk8

79 Switching to Oracle JDK8 (java-8-oracle), JAVA\_HOME will be set to /usr/lib/jvm/java-8-oracle

80 \$ java -Xmx32m -version

# Tracking Build Quality

Track quality indicators over time, e.g.,

- Build time
- Test coverage
- Static analysis warnings
- Performance results
- Model quality measures
- Number of TODOs in source code

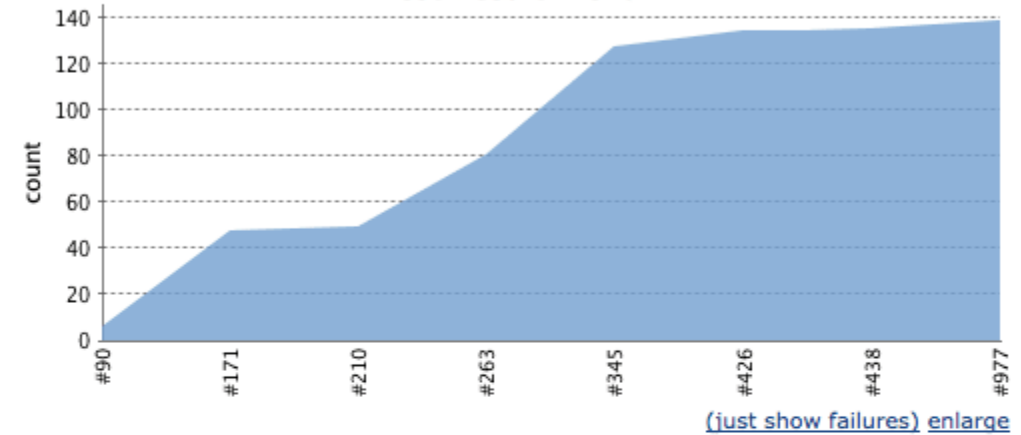
[Back to Dashboard](#)[Status](#)[Changes](#)[Workspace](#)[Build Now](#)[Delete Project](#)[Configure](#)[Set Next Build Number](#)[Duplicate Code](#)[Coverage Report](#)[SLOCCount](#)[Git Polling Log](#)

## Project Stop-tabac dev

CI build

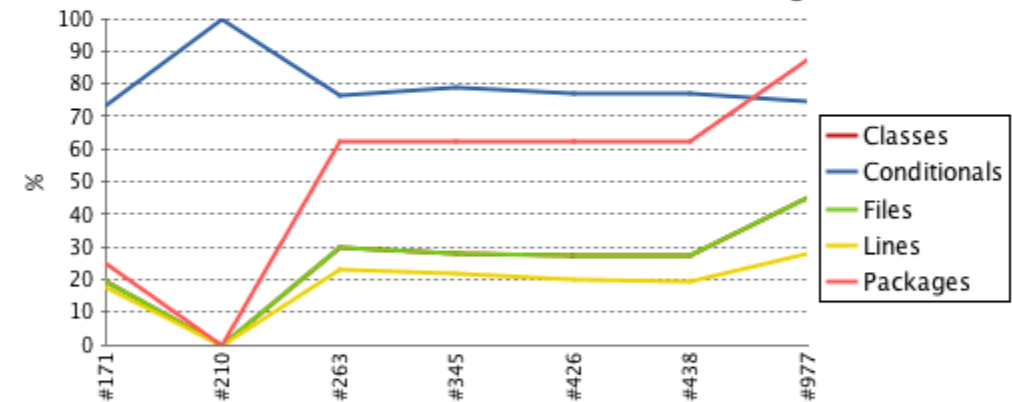
[Coverage Report](#)[Workspace](#)[Recent Changes](#)[Latest Test Result](#) (no failures)[Edit description](#)[Disable Project](#)

### Test Result Trend



### Code Coverage

Classes 45% Conditionals 74% Files 45% Lines 28% Packages 88%



### Permalinks

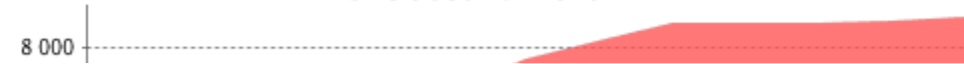
- [Last build \(#977\), 3 min 17 sec ago](#)
- [Last stable build \(#977\), 3 min 17 sec ago](#)
- [Last successful build \(#977\), 3 min 17 sec ago](#)

### Build History (trend)

#977	Aug 27, 2012 4:37:27 PM	
#438	Jun 28, 2012 8:47:42 AM	
#426	Jun 26, 2012 1:39:39 PM	
#345	Jun 19, 2012 9:02:20 AM	
#263	Jun 6, 2012 9:14:42 PM	
#210	May 31, 2012 8:42:29 AM	
#171	May 23, 2012 9:58:18 PM	
#90	May 15, 2012 11:49:41 AM	

[RSS for all](#) [RSS for failures](#)

### SLOCCount Trend





# Tracking Model Qualities

Many tools: MLFlow, ModelDB, Neptune, TensorBoard, Weights & Biases, Comet.ml, ...

# ModelDB Example

```
from verta import Client
client = Client("http://localhost:3000")

proj = client.set_project("My first ModelDB project")
expt = client.set_experiment("Default Experiment")

# log a training run
run = client.set_experiment_run("First Run")
run.log_hyperparameters({"regularization" : 0.5})
model1 = # ... model training code goes here
run.log_metric('accuracy', accuracy(model1, validationData))
```

# Test Monitoring

- Inject/simulate faulty behavior
- Mock out notification service used by monitoring
- Assert notification

```
class MyNotificationService extends NotificationService {  
    public boolean receivedNotification = false;  
    public void sendNotification(String msg) {  
        receivedNotification = true; }  
}  
@Test void test() {  
    Server s = getServer();  
    MyNotificationService n = new MyNotificationService();  
    Monitor m = new Monitor(s, n);  
    s.stop();  
}
```

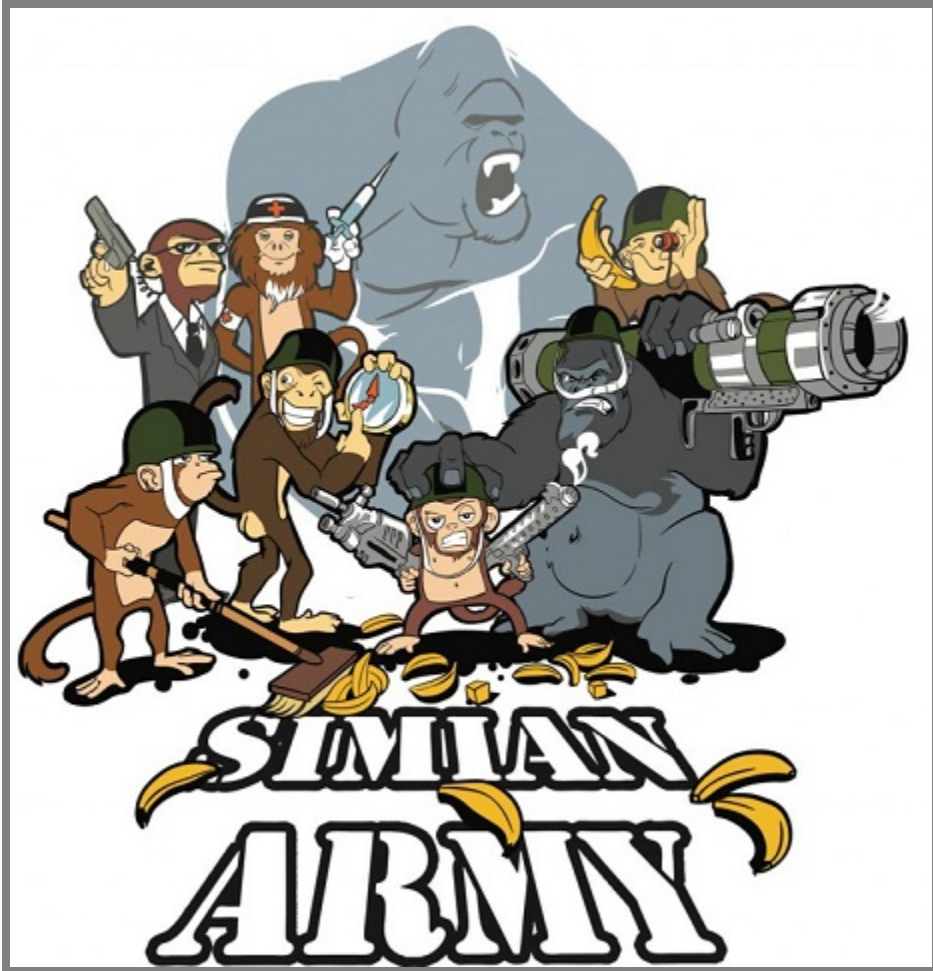
# Test Monitoring in Production

Like fire drills (manual tests may be okay!)

Manual tests in production, repeat regularly

Actually take down service or trigger wrong signal to monitor

# Chaos Testing



Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production. Pioneered at Netflix

# Chaos Testing Argument

- Distributed systems are simply too complex to comprehensively predict
  - experiment to learn how it behaves in the presence of faults
- Base corrective actions on experimental results because they reflect real risks and actual events
- Experimentation != testing -- Observe behavior rather than expect specific results
- Simulate real-world problem in production (e.g., take down server, inject latency)
- *Minimize blast radius*: Contain experiment scope



# Netflix's Simian Army

- Chaos Monkey: randomly disable production instances
- Latency Monkey: induces artificial delays in our RESTful client-server communication layer
- Conformity Monkey: finds instances that don't adhere to best-practices and shuts them down
- Doctor Monkey: monitors external signs of health to detect unhealthy instances
- Janitor Monkey: ensures cloud environment is running free of clutter and waste
- Security Monkey: finds security violations or vulnerabilities, and terminates the offending instances
- 10-18 Monkey: detects problems in instances serving customers in multiple geographic regions
- Chaos Gorilla is similar to Chaos Monkey, but simulates an outage of an entire Amazon availability zone.

# Chaos Toolkit

- Infrastructure for chaos experiments
- Driver for various infrastructure and failure cases
- Domain specific language for experiment definitions

```
{  
  "version": "1.0.0",  
  "title": "What is the impact of an expired certificate on",  
  "description": "If a certificate expires, we should gracefully",  
  "tags": ["tls"],  
  "steady-state-hypothesis": {  
    "title": "Application responds",  
    "probes": [  
      {  
        "type": "probe",
```

# Chaos Experiments for ML Infrastructure?



Fault injection in production for testing in production. Requires monitoring and explicit experiments.



# Code Review and Static Analysis

# Code Review

## Manual inspection of code

- Looking for problems and possible improvements
- Possibly following checklists
- Individually or as group

## Modern code review: Incremental review at checking

- Review individual changes before merging
- Pull requests on GitHub
- Not very effective at finding bugs, but many other benefits:  
knowledge transfer, code improvement, shared code ownership,  
improving testing

This repository

[Explore](#) [Features](#) [Enterprise](#) [Blog](#)

**Sign up**

[Sign in](#)

★ Star 20

Fork 12

**New issue**

Conversation 3

Commits 17

Files changed 97

+1,149 -10,129 ■■■■

ckaestne commented on Jan 29

Owner

Please have a look whether you agree with these refactorings in CRewrite

key changes: Moved ASTNavigation and related classes and turned EnforceTreeHelper into an object

## Labels

None yet

### Milestone

No milestone

Assignee

No one assigned

2 participants



ckaestne commented on Jan 29

Owner

Can one of the admins verify this patch?

# Subtle Bugs in Data Wrangling Code

```
df['Join_year'] = df.Joined.dropna().map(  
    lambda x: x.split(',')[1].split(' ')[1])
```

```
df.loc[idx_nan_age, 'Age'].loc[idx_nan_age] =  
    df['Title'].loc[idx_nan_age].map(map_means)
```

```
df["Weight"].astype(str).astype(int)
```

```
df['Reviws'] = df['Reviews'].apply(int)
```



We did code review earlier together

# Static Analysis, Code Linting

Automatic detection of problematic patterns based on code structure

```
if (user.jobTitle = "manager") {  
    ...  
}
```

```
function fn() {  
    x = 1;  
    return x;  
    x = 3;  
}
```

# Process Integration: Static Analysis Warnings during Code Review

```
package com.google.devtools.staticanalysis;

public class Test {
```

▼ Lint      Missing a Javadoc comment.

Java  
1:02 AM, Aug 21

[Please fix](#) [Not useful](#)

```
    public boolean foo() {
        return getString() == "foo".toString();
    }
```

▼ ErrorProne      String comparison using reference equality instead of value equality  
(see <http://code.google.com/p/error-prone/wiki/StringEquality>)

StringEquality  
1:03 AM, Aug 21

[Please fix](#)

Suggested fix attached: [show](#) [Not useful](#)

```
    public String getString() {
        return new String("foo");
    }
}
```

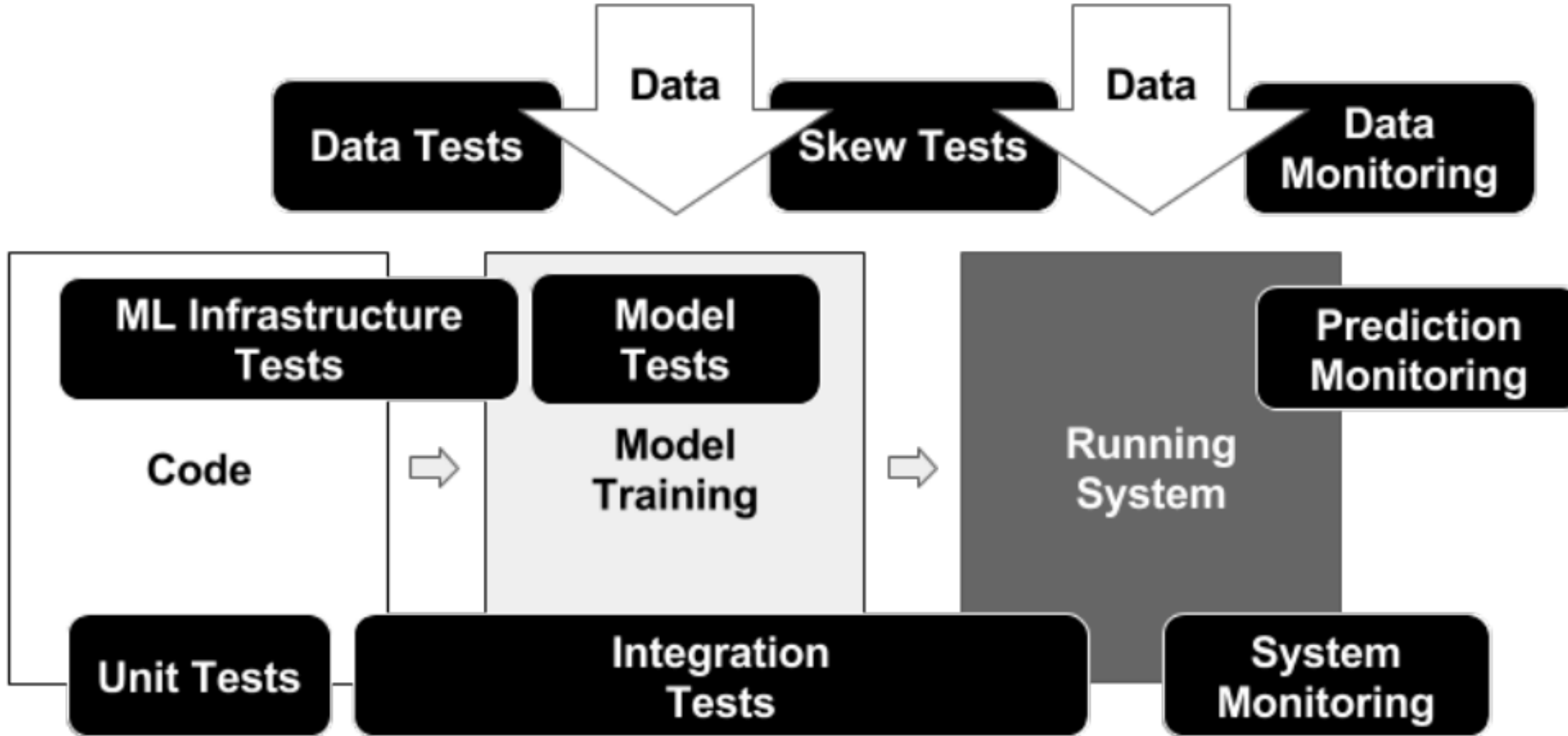
Social engineering to force developers to pay attention. Also possible with integration in pull requests on GitHub.



# Infrastructure Testing



Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)



Source: Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data

# Data Tests

1. Feature expectations are captured in a schema.
2. All features are beneficial.
3. No feature's cost is too much.
4. Features adhere to meta-level requirements.
5. The data pipeline has appropriate privacy controls.
6. New features can be added quickly.
7. All input feature code is tested.

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)

# Tests for Model Development

1. Model specs are reviewed and submitted.
2. Offline and online metrics correlate.
3. All hyperparameters have been tuned.
4. The impact of model staleness is known.
5. A simpler model is not better.
6. Model quality is sufficient on important data slices.
7. The model is tested for considerations of inclusion.

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)



# ML Infrastructure Tests

1. Training is reproducible.
2. Model specs are unit tested.
3. The ML pipeline is Integration tested.
4. Model quality is validated before serving.
5. The model is debuggable.
6. Models are canaried before serving.
7. Serving models can be rolled back.

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)

# Monitoring Tests

1. Dependency changes result in notification.
2. Data invariants hold for inputs.
3. Training and serving are not skewed.
4. Models are not too stale.
5. Models are numerically stable.
6. Computing performance has not regressed.
7. Prediction quality has not regressed.

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. [The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction](#). Proceedings of IEEE Big Data (2017)

# Case Study: Covid-19 Detection



(from S20 midterm; assume cloud or hybrid deployment)

# Breakout Groups

- In the Smartphone Covid Detection scenario
- Discuss in groups:
  - Back left: data tests
  - Back right: model dev. tests
  - Front right: infrastructure tests
  - Front left: monitoring tests
- For 8 min, discuss some of the listed point in the context of the Covid-detection scenario: what would you do?
- In #lecture, tagging group members, suggest what tests to implement

# Dev vs. Ops



# 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
- ...

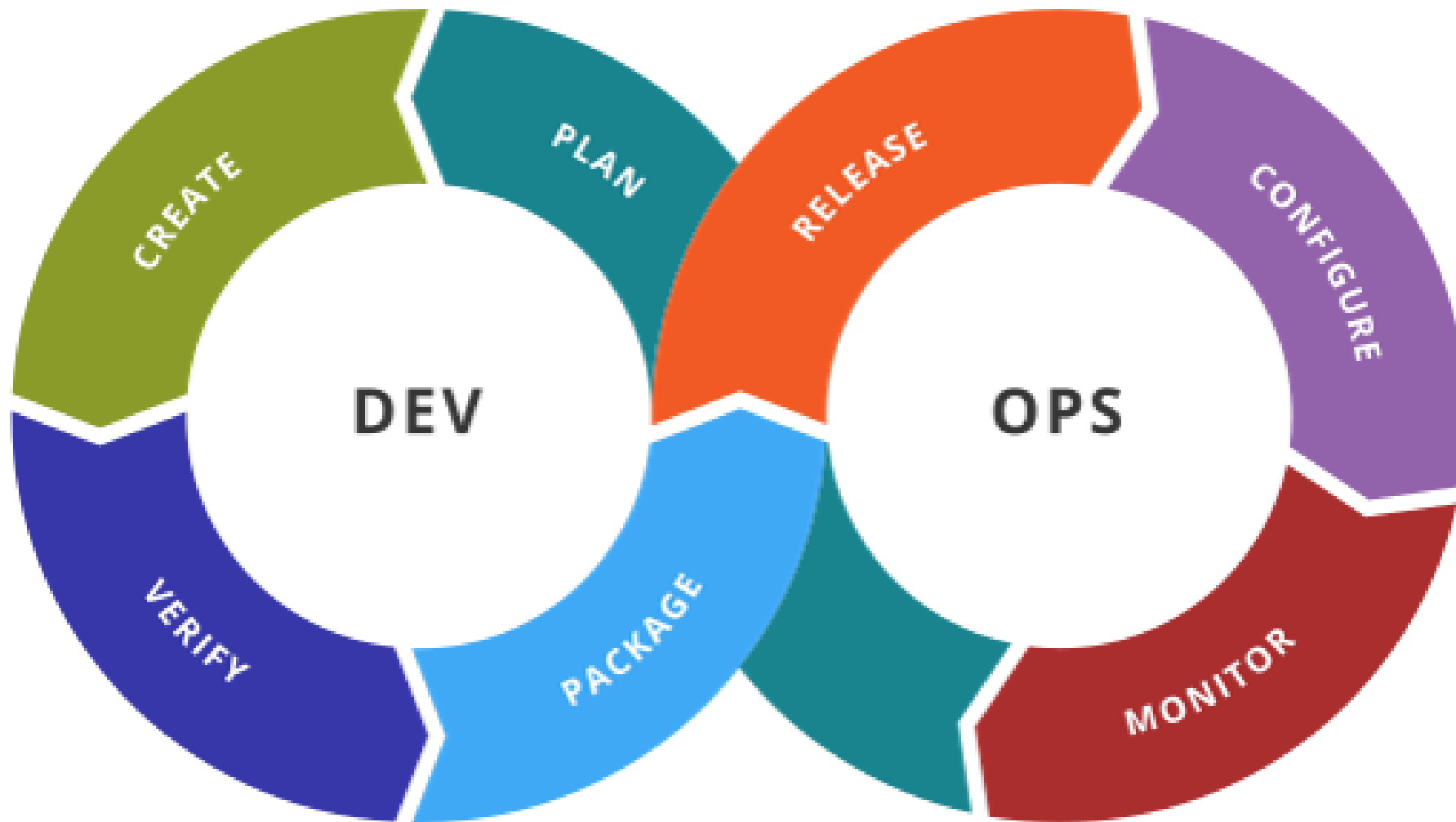
QA responsibilities in both roles



# Quality Assurance does not stop in Dev

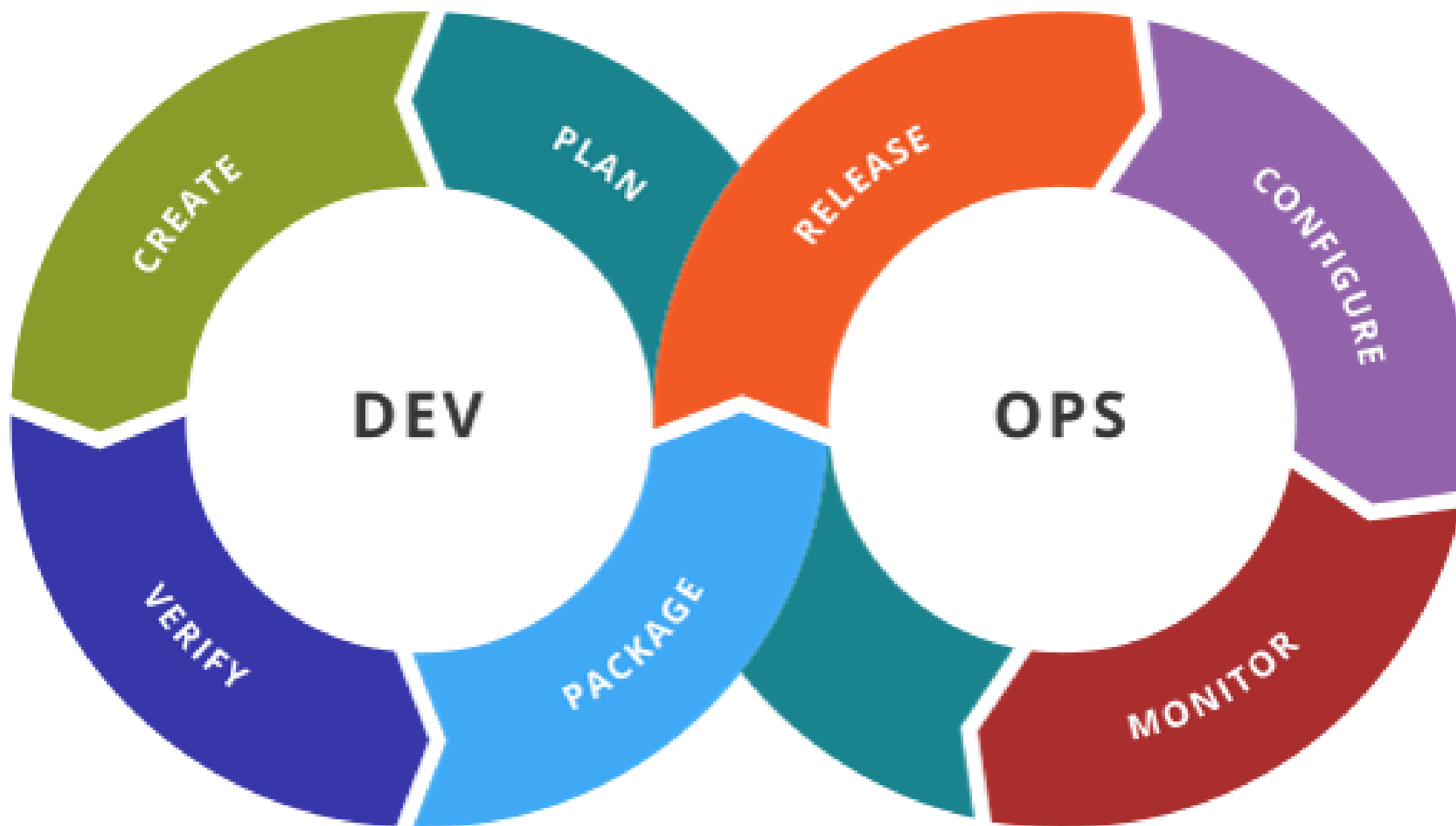
- Ensuring product builds correctly (e.g., reproducible builds)
- Ensuring scalability under real-world loads
- Supporting environment constraints from real systems (hardware, software, OS)
- Efficiency with given infrastructure
- Monitoring (server, database, Dr. Watson, etc)
- Bottlenecks, crash-prone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

# DevOps



# 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!



# 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

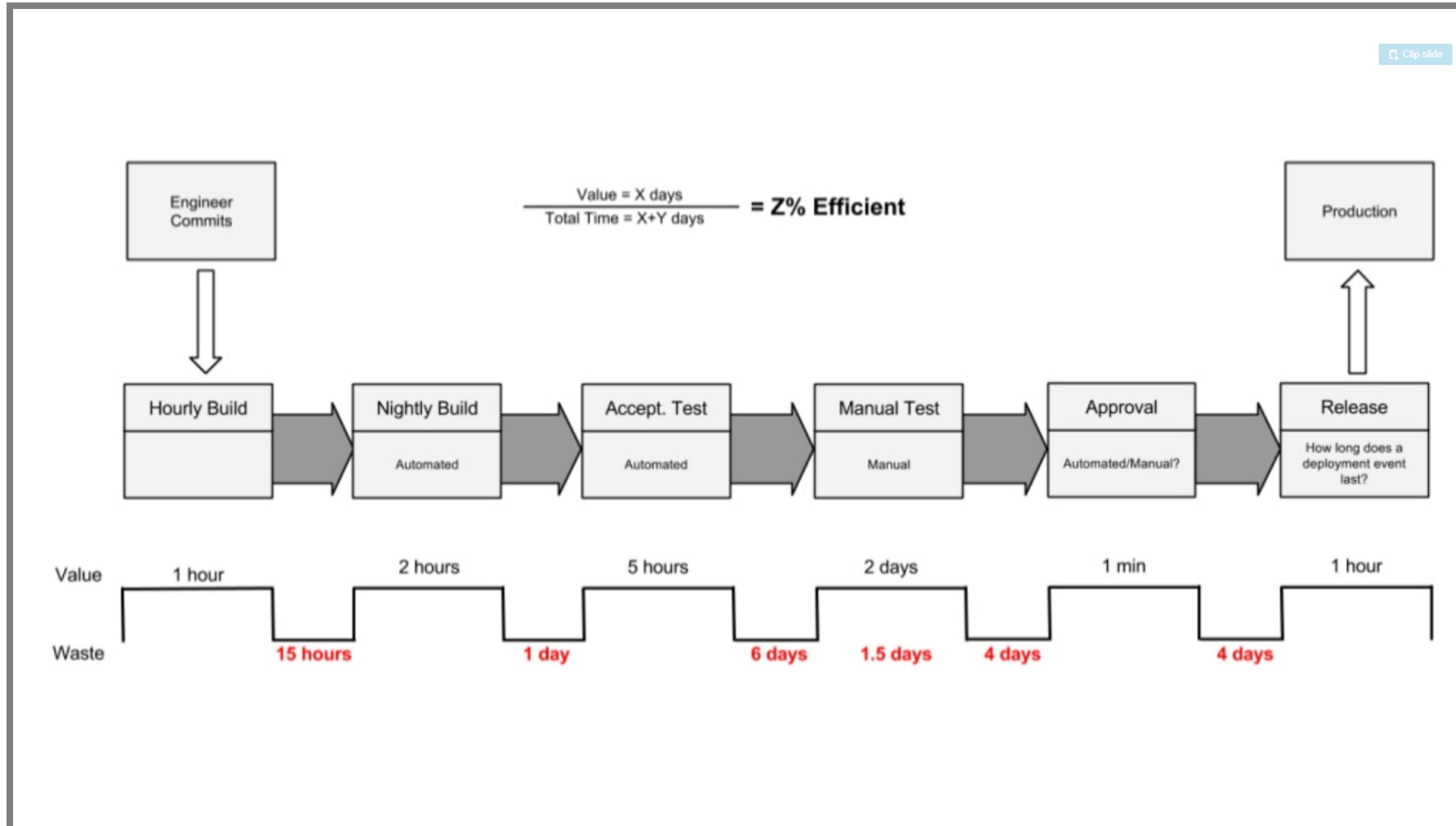
# Heavy tooling and automation -- Examples

- 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

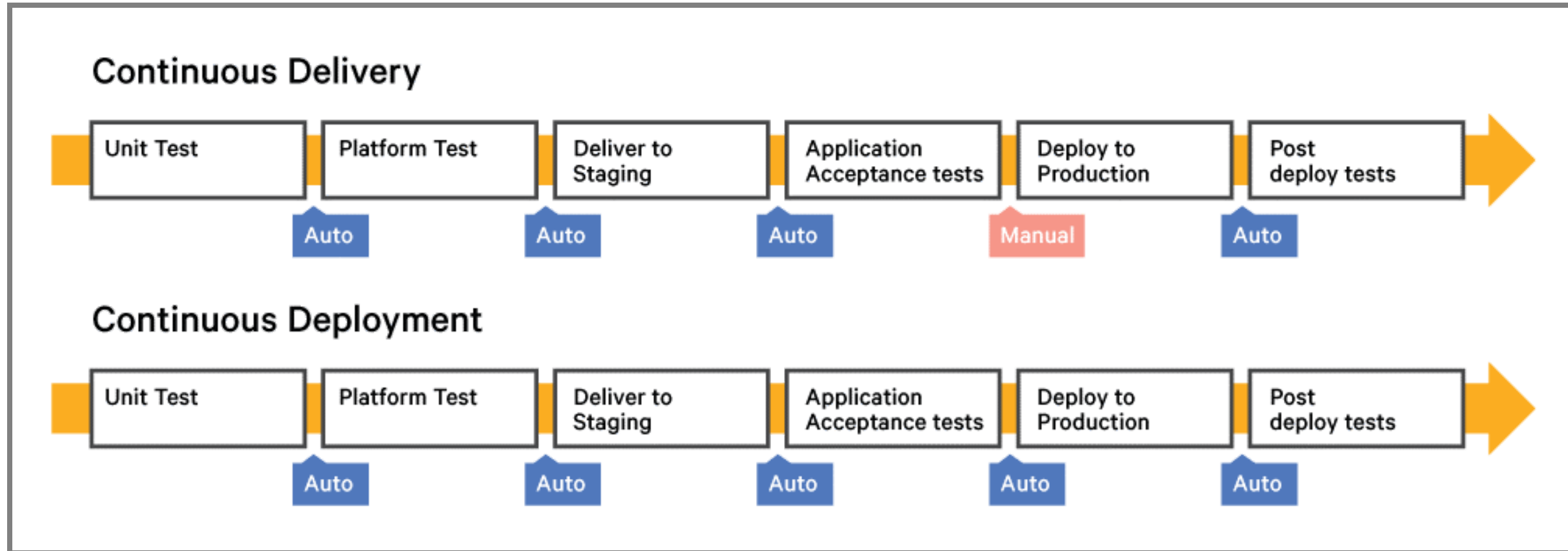
# Continuous Delivery

- 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

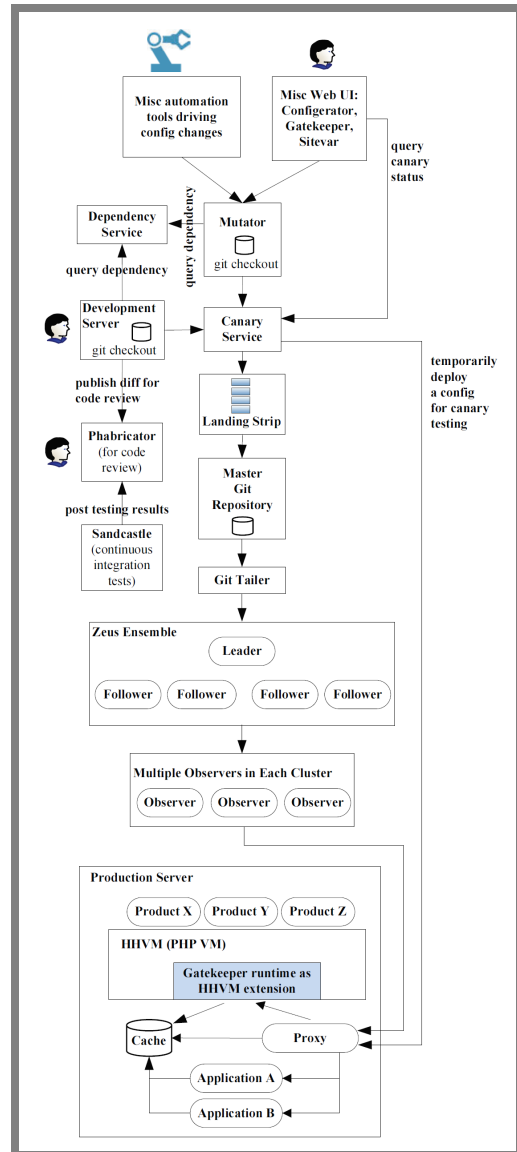
- Unit tests (white box)
- Static analysis (null pointer warnings, memory leaks, ...)
- Build tests (compilation succeeds)
- Snapshot tests (screenshot comparison, pixel by pixel)
- Integration tests (black box, in simulators)
- Performance tests (resource usage)
- Capacity and conformance tests (custom)

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

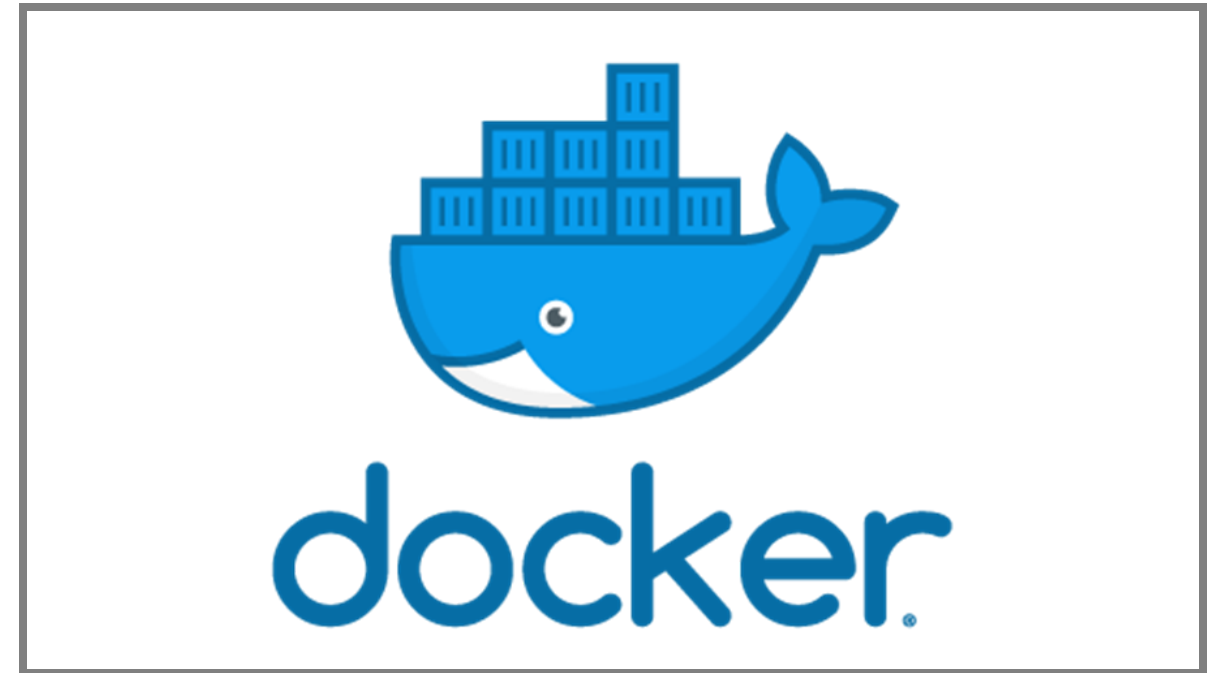
# Real-world pipelines are complex



# 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"]
```

# 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 processes
- name: create data directory for mongod
  file: path={{ mongodb_datadir_prefix }}
  delegate_to: '{{ item }}'
  with_items: groups.replication_servers

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

- name: Create the mongod startup file
  template: src=mongod.i2 dest=/etc/init.d
```

# 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']
}
```

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

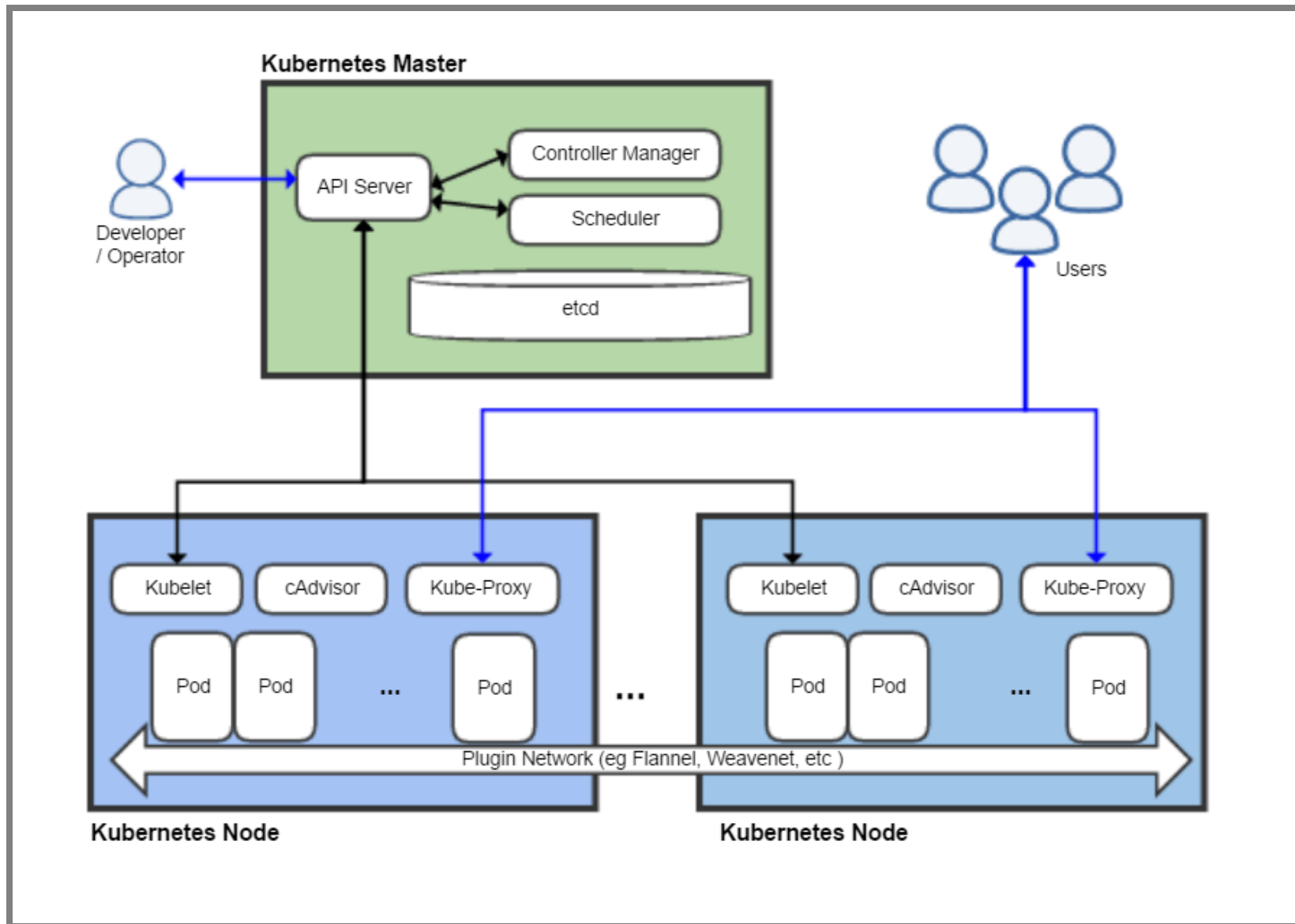
Launches and kills containers depending on load

Manage updates and routing

Automated restart, replacement, replication, scaling

Kubernetes master controls many nodes

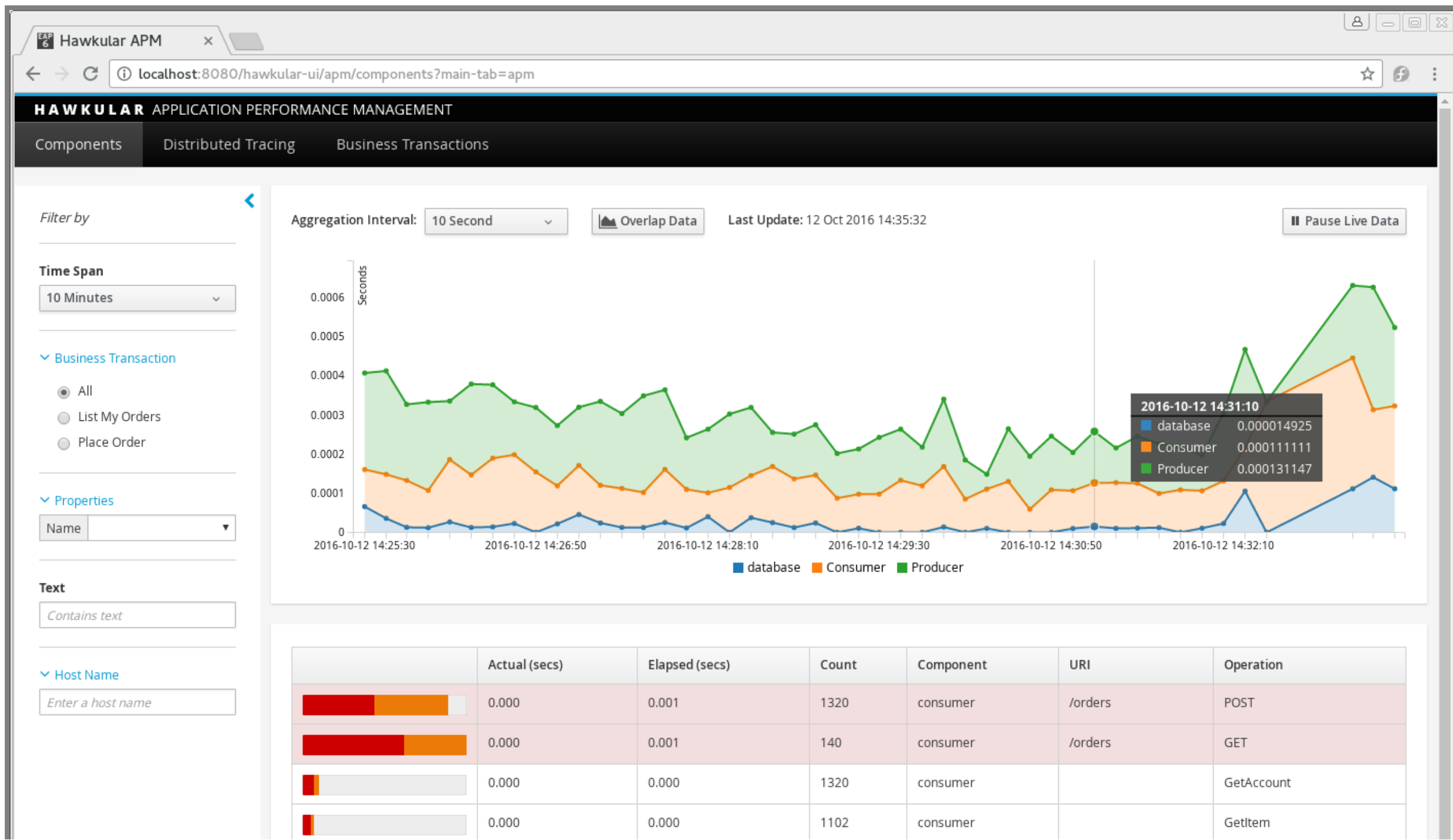
*Substantial complexity and learning curve*



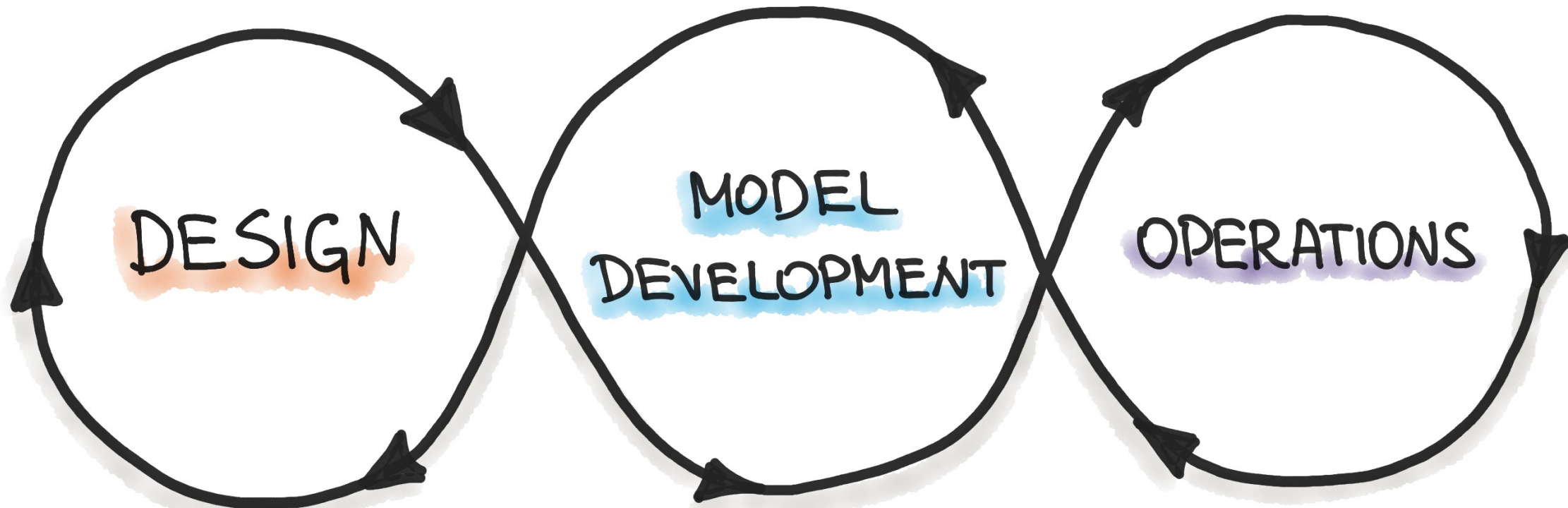
# Monitoring

- 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





MLOps



# 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)

# Tooling Landscape LF AI

# MLOps Tools -- Examples



- Model 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>

# Summary

- Beyond model and data quality: Quality of the infrastructure matters, danger of silent mistakes
- Automate pipelines to foster evolution and experimentation
- Many SE techniques for test automation, testing robustness, test adequacy, testing in production useful for infrastructure quality
- DevOps: Development vs Operations challenges
  - Automate everything: deployment, configuration, testing
  - Telemetry and monitoring are key
- MLOps: Automation around ML pipelines, incl. training, evaluation, versioning, and deployment

# Further Readings

- O'Leary, Katie, and Makoto Uchida. "[Common problems with Creating Machine Learning Pipelines from Existing Code](#)." Proc. Third Conference on Machine Learning and Systems (MLSys) (2020).
- Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. Proceedings of IEEE Big Data (2017)
-  Zinkevich, Martin. [Rules of Machine Learning: Best Practices for ML Engineering](#). Google Blog Post, 2017
- Serban, Alex, Koen van der Blom, Holger Hoos, and Joost Visser. "[Adoption and Effects of Software Engineering Best Practices in Machine Learning](#)." In Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (2020).
-  Larysa Visengeriyeva. [Machine Learning Operations - A Reading List](#), InnoQ 2020