INFRASTRUCTURE QUALITY, DEPLOYMENT, AND OPERATIONS

Christian Kaestner

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)

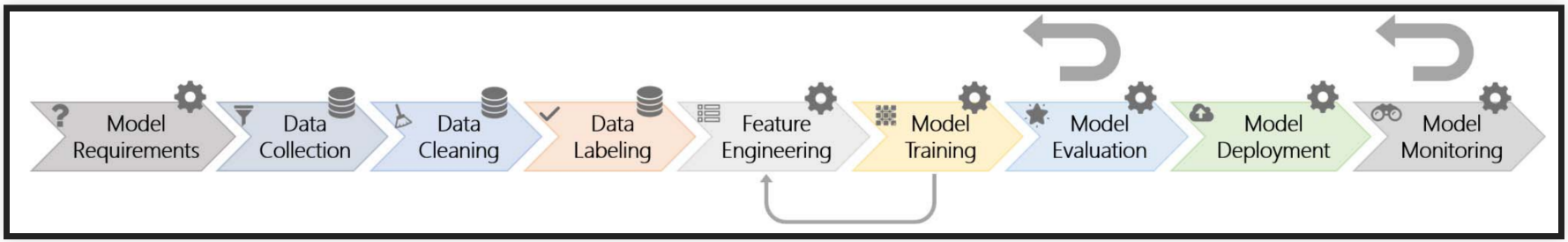
Recommended readings: Larysa Visengeriyeva. Machine Learning Operations - A Reading List, InnoQ 2020

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

BEYOND MODEL AND DATA QUALITY

POSSIBLE MISTAKES IN ML PIPELINES



Danger of "silent" mistakes in many phases

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

EVERYTHING CAN BE TESTED?

Speaker notes

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

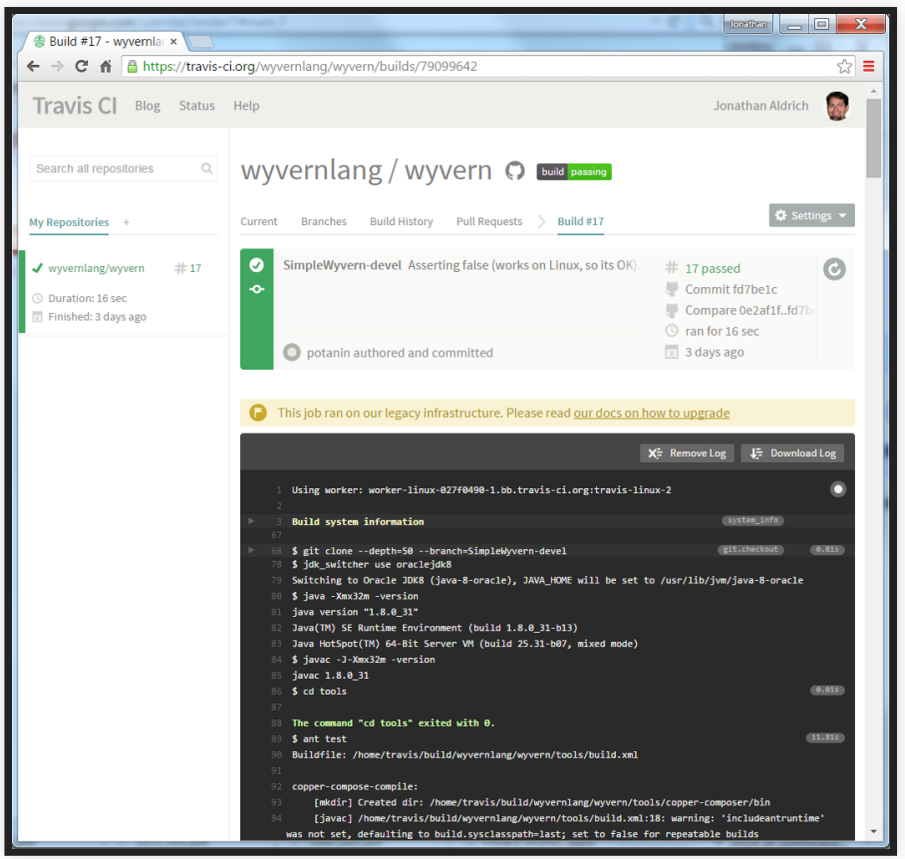
TESTING STRATEGIES

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

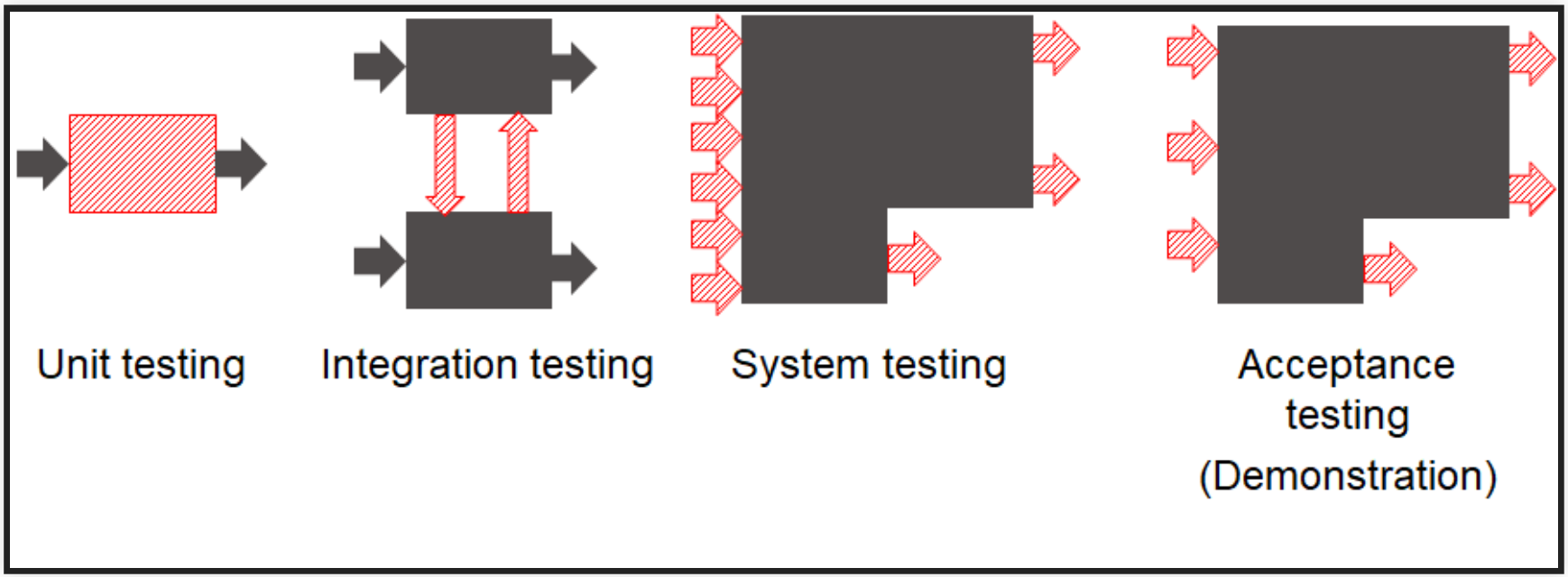
How to test for these? How automatable?

TEST AUTOMATION

FROM MANUAL TESTING TO CONTINUOUS INTEGRATION



UNIT TEST, INTEGRATION TESTS, SYSTEM TESTS



Speaker notes

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)
        assertEquals(true, g1.addVertex(s1));
        assertEquals(true, g1.addVertex(s2));
        assertEquals(true, g1.addEdge(s1, s2));
        assertEquals(s2, g1.getNeighbors(s1)[0]);
    }
```

INGREDIENTS TO A TEST

- Specification

- Controlled environment
- Test inputs (calls and parameters)
- Expected outputs/behavior (oracle)

UNIT TESTING PITFALLS

- Working code, failing tests
- Smoke tests pass
- Works on my (some) machine(s)
- Tests break frequently

How to avoid?

HOW TO UNIT TEST COMPONENT WITH DEPENDENCY ON OTHER CODE?

EXAMPLE: TESTING PARTS OF A SYSTEM


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

EXAMPLE: USING TEST DATA


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

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

EXAMPLE: USING TEST DATA


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

@Test void test() {
    Stream testStream = new Stream() {
        int idx = 0;
        // hardcoded or read from test file
        String[] data = [ ... ]
        public void connect() { }
        public String getNext() {
            return data[++idx];
        }
    }
    DataTable output = getData(testStream, new DefaultCleaner())
    assert(output.length==10)
}
```

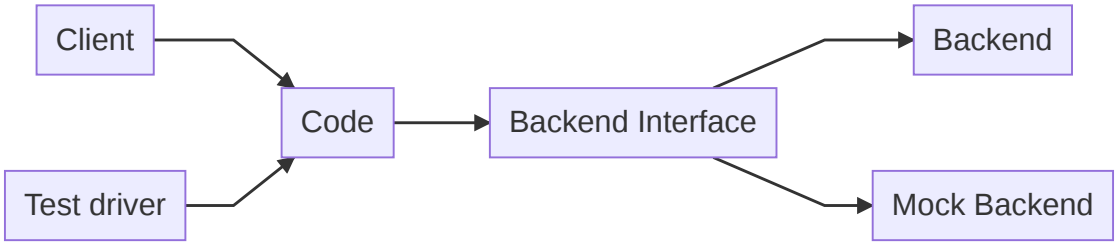
EXAMPLE: MOCKING A DATACLEANER OBJECT

```
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;
        }
        ...
    }
    DataTable output = getData(testStream, dummyCleaner);
    assert(output.length==9)
}
```

Mocking frameworks provide infrastructure for expressing such tests compactly.


Client
Code
Backend Interface
Backend
Test driver
Mock Backend

TEST ERROR HANDLING

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

Speaker notes

Code to test that the right exception is thrown

TESTING FOR ROBUSTNESS

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

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

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

TEST LOCAL ERROR HANDLING (MODULAR PROTECTION)

```
@Test void test() {
    Stream testStream = new Stream() {
        int idx = 0;
        public void connect() {
            if (++idx < 3)
                throw new IOException("cannot establish connecti
        }
        public String getNext() { ... }
    }
    DataLoader loader = new DataLoader(testStream, new DefaultCl
    ModelBuilder model = new ModelBuilder(loader, ...);
    // assume all exceptions are handled correctly internally
    assert(model.accuracy > .91)
}
```

Speaker notes

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.coveragedat
net.sourceforge.cobertura.instrument
net.sourceforge.cobertura.merge
net.sourceforge.cobertura.reporting
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.htr
net.sourceforge.cobertura.reporting.xm
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%)*
HTMLReport *(94%)*
HasBeenInstrumented *(N/A)*
Header *(80%)*
IOUtil *(62%)*
Ignore *(100%)*
IgnoreBranches *(0%)*

Coverage Report - All Packages

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

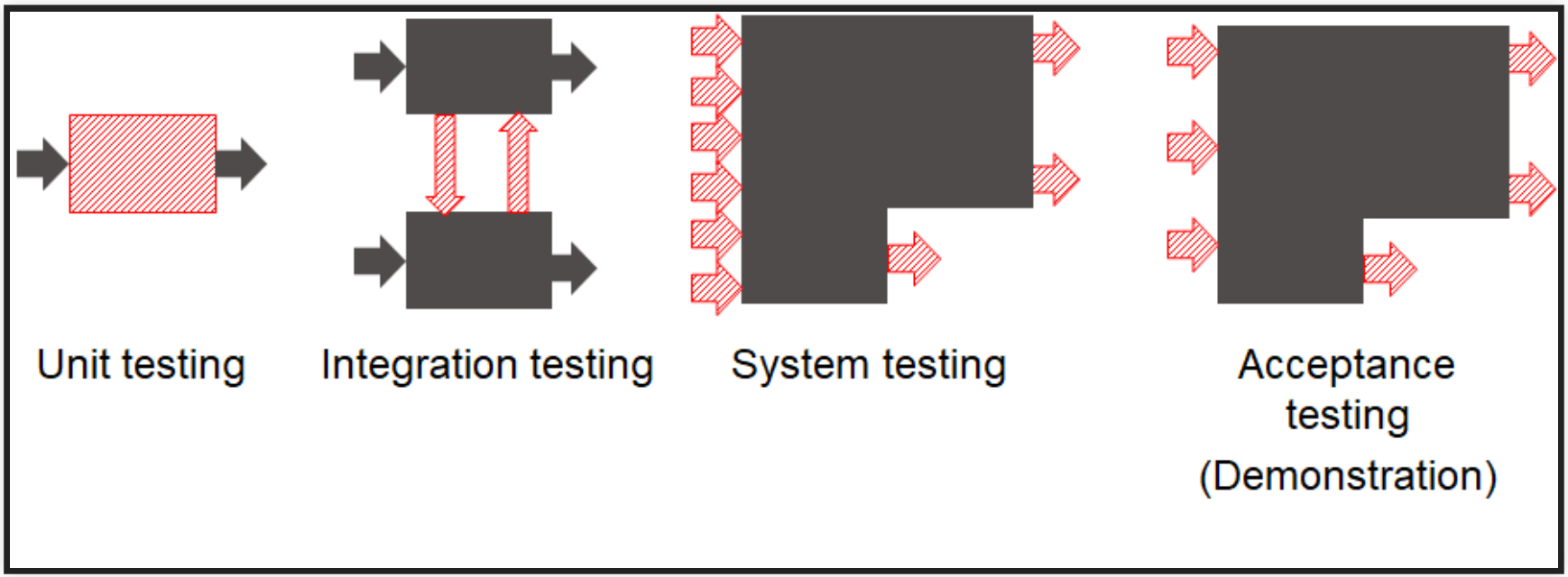
Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

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

- Test-Driven Development: A design and development method in which you write tests before you write the code

INTEGRATION AND SYSTEM TESTS



INTEGRATION AND SYSTEM TESTS

Test larger units of behavior

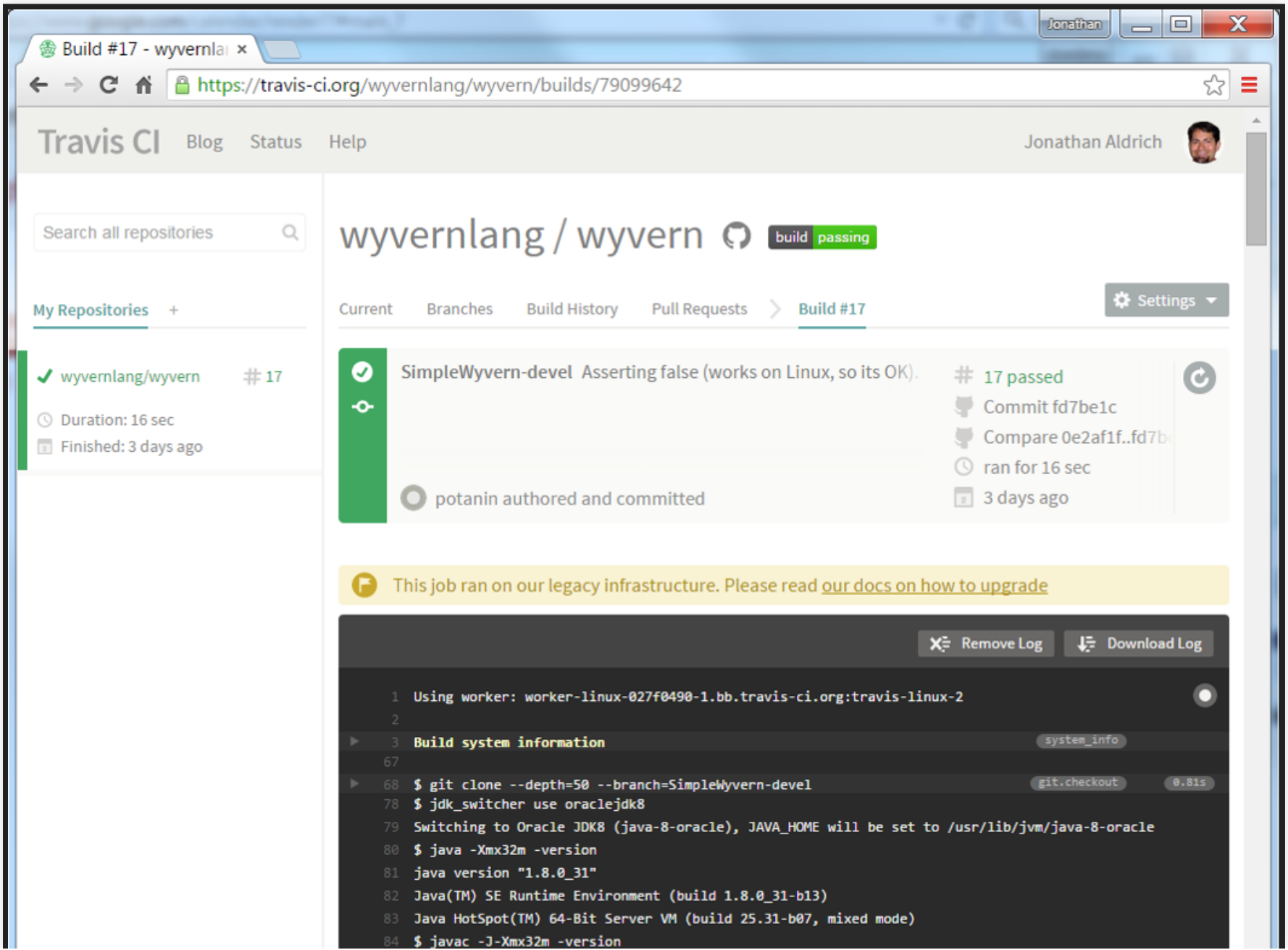
Often based on use cases or user stories -- customer perspective

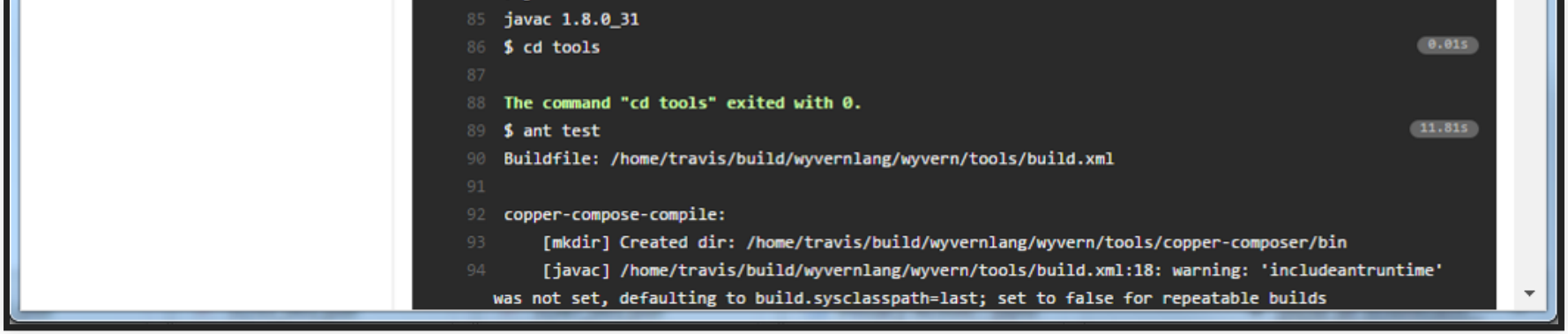
```
@Test void gameTest() {
    Poker game = new Poker();
    Player p = new Player();
    Player q = new Player();
    game.shuffle(seed)
    game.add(p);
    game.add(q);
    game.deal();
    p.bet(100);
    q.bet(100);
    p.call();
    q.fold();
    assert(game.winner() == p);
}
```

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


Build #17 - wyvernla
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).
potanin authored and committed
17 passed
Commit fd7be1c
Compare 0e2af1f..fd7b
ran for 16 sec
3 days ago
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
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
81 java version "1.8.0_31"
82 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
83 Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
84 $ javac -J-Xmx32m -version


85 javac 1.8.0_31
86 $ cd tools
0.01s
87
88 The command "cd tools" exited with 0.
89 $ ant test
11.81s
90 Buildfile: /home/travis/build/wyvernlang/wyvern/tools/build.xml
91
92 copper-compose-compile:
93 [mkdir] Created dir: /home/travis/build/wyvernlang/wyvern/tools/copper-composer/bin
94 [javac] /home/travis/build/wyvernlang/wyvern/tools/build.xml:18: warning: 'includeantruntime'
was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds

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

Jenkins

search

Jenkins › Suisse › Stop-tabac dev

ENABLE AUTO REFRESH

- Back to Dashboard
- **Status**
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Set Next Build Number
- Duplicate Code
- Coverage Report
- SLOCCount
- Git Polling Log

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

Project Stop-tabac dev

CI build

edit description

Disable Project

Coverage Report

Workspace

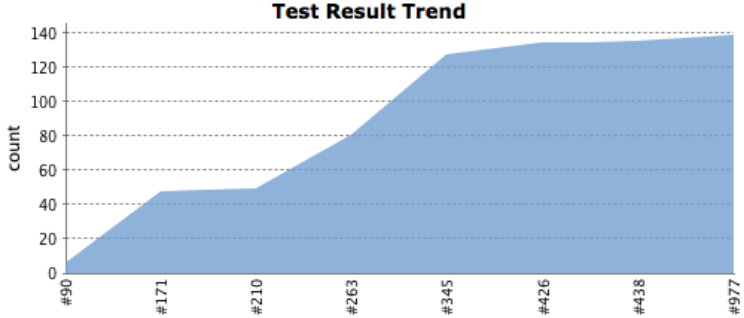
Recent Changes

Latest Test Result (no failures)

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

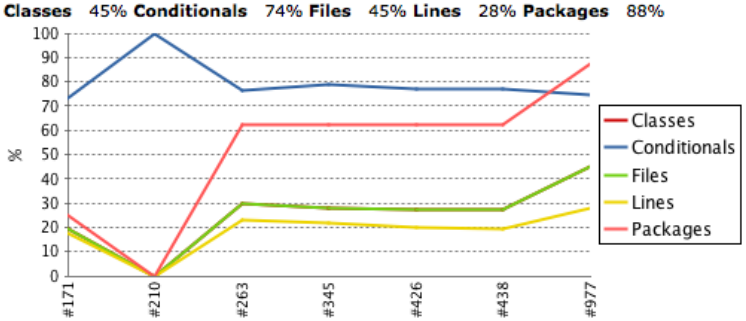
Test Result Trend



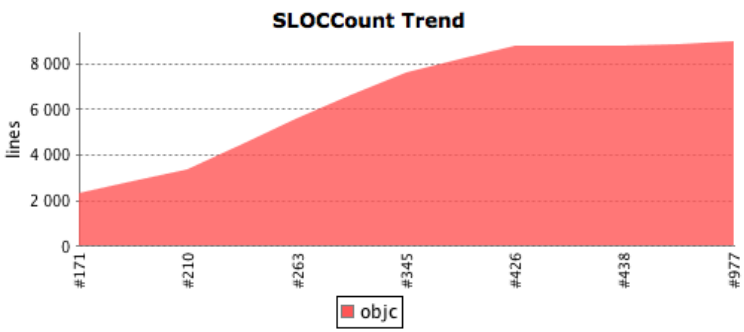
(just show failures) enlarge

Code Coverage

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



SLOCCount Trend



Source: https://blog.octo.com/en/jenkins-quality-dashboard-ios-development/

TEST MONITORING

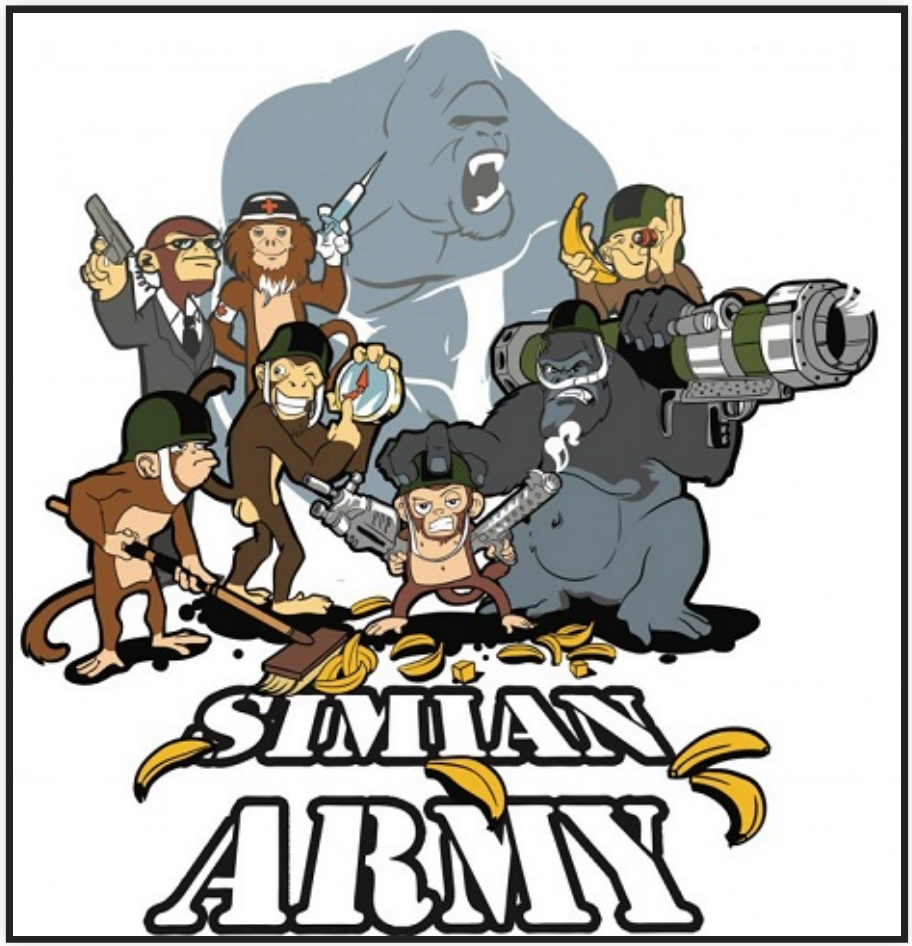
- Inject/simulate faulty behavior
- Mock out notification service used by monitoring
- Assert notification

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

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



http://principlesofchaos.org

Speaker notes

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 on our systems to learn how they will behave in the presence of faults
- Base corrective actions on experimental results because they reflect real risks and actual events

- Experimentation != testing -- Observe behavior rather then 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 other external signs of health to detect unhealthy instances

Janitor Monkey: ensures that our 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 ou
    "description": "If a certificate expires, we should graceful
    "tags": ["tls"],
    "steady-state-hypothesis": {
        "title": "Application responds",
        "probes": [
            {
                "type": "probe",
                "name": "the-astre-service-must-be-running",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "os.path",
                    "func": "exists"
```

http://principlesofchaos.org, https://github.com/chaostoolkit, https://github.com/Netflix/SimianArmy

CHAOS EXPERIMENTS FOR ML INFRASTRUCTURE?

Speaker notes

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

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)

CASE STUDY: SMART PHONE COVID-19 DETECTION

(from midterm; assume cloud or hybrid deployment)

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)

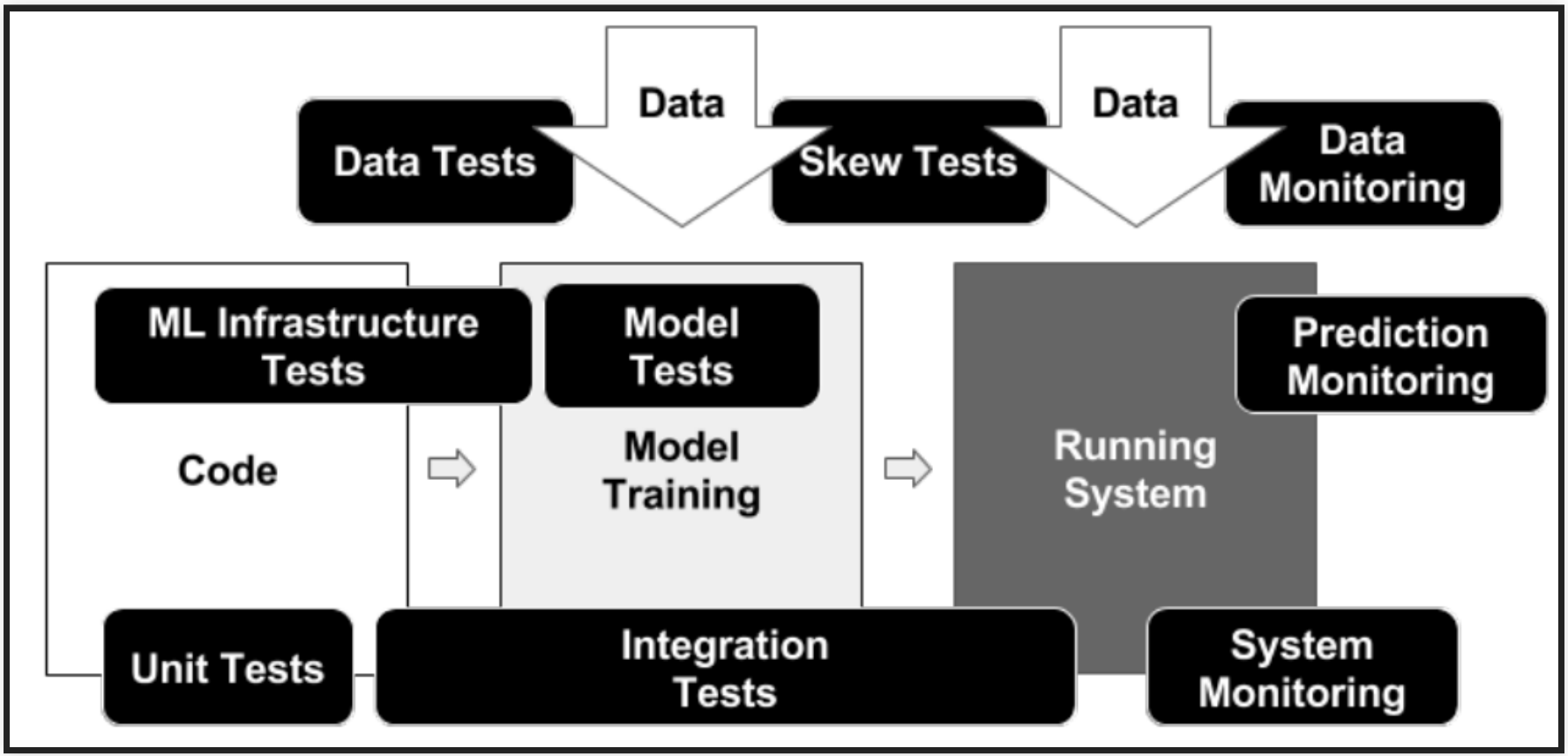
MONITORING TESTS

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)

BREAKOUT GROUPS

- Discuss in groups:
    - Team 1 picks the data tests
    - Team 2 the model dev. tests
    - Team 3 the infrastructure tests
    - Team 4 the monitoring tests
- For 15 min, discuss each listed point in the context of the Covid-detection scenario: what would you do?
- Report back to the class



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 (2017)

ASIDE: LOCAL IMPROVEMENTS VS OVERALL QUALITY

- Ideally unit tests catch bugs locally
- Some bugs emerge from interactions among system components
    - Missed local specifications -> more unit tests
    - Nonlocal effects, interactions -> integration & system tests

Known as **emergent properties** and **feature interactions**

FEATURE INTERACTION EXAMPLES

Speaker notes

Flood control and fire control work independently, but interact on the same resource (water supply), where flood control may deactivate the water supply to the sprinkler system in case of a fire

FEATURE INTERACTION EXAMPLES

Speaker notes

Electronic parking brake and AC are interacting via the engine. Electronic parking brake gets released over a certain engine speed and AC may trigger that engine speed (depending on temperature and AC settings).

FEATURE INTERACTION EXAMPLES

[:weather:]

Plugin

WordPress

:) 8-) ;-) ...

Plugin

Today's weather: [:weather 

Speaker notes

Weather and smiley plugins in WordPress may work on the same tokens in a blog post (overlapping preconditions)

FEATURE INTERACTION EXAMPLES

Speaker notes

Call forwarding and call waiting in a telecom system react to the same event and may result in a race condition. This is typically a distributed system with features implemented by different providers.

FEATURE INTERACTIONS

Failure in compositionality: Components developed and tested independently, but they are not fully independent

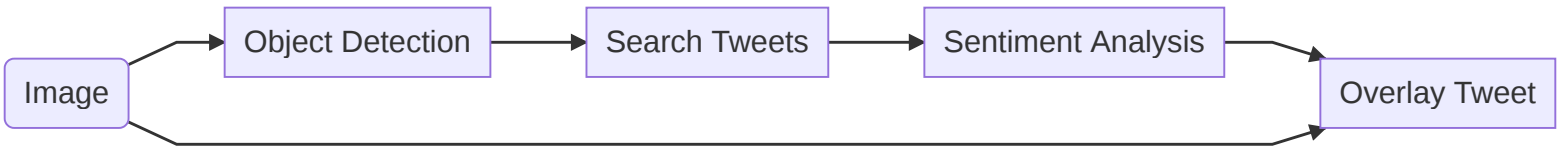
Detection and resolution challenging:

- Analysis of requirements (formal methods or inspection), e.g., overlapping preconditions, shared resources
- Enforcing isolation (often not feasible)
- Testing, testing, testing at the system level

Recommended reading: Nhlabatsi, Armstrong, Robin Laney, and Bashar Nuseibeh. Feature interaction: The security threat from within software systems. Progress in Informatics 5 (2008): 75-89.

MODEL CHAINING

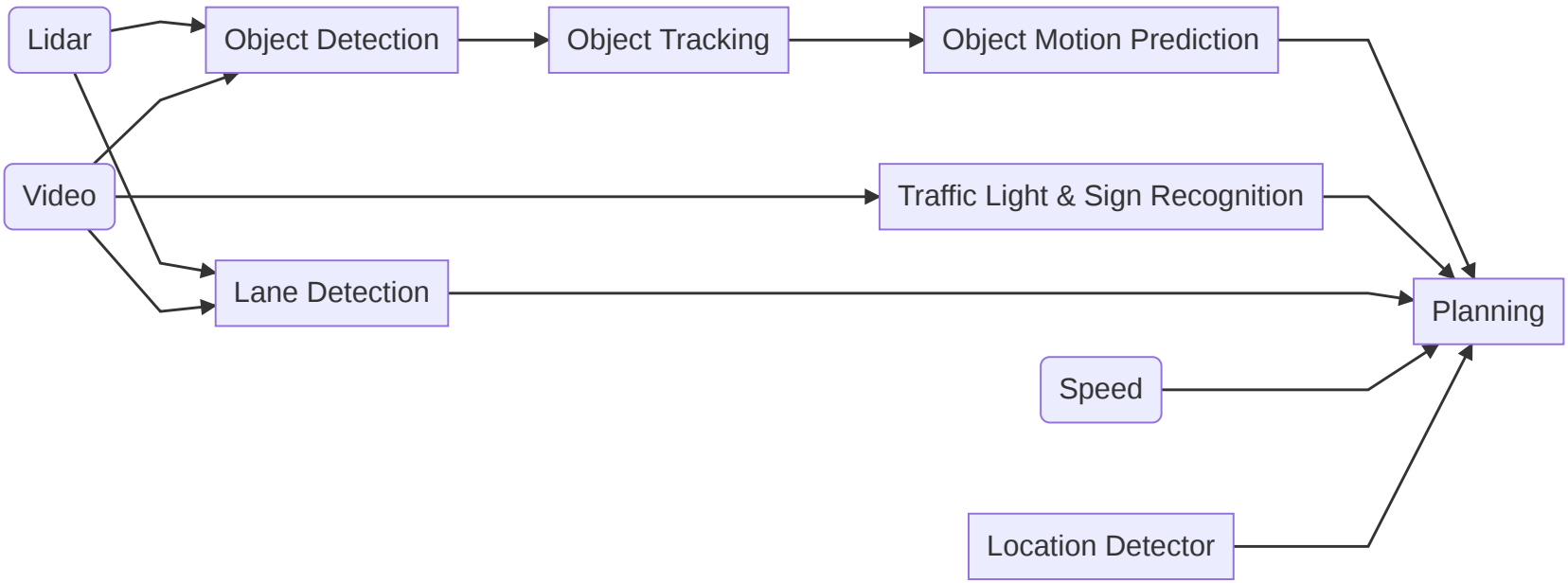
automatic meme generator



Example adapted from Jon Peck. Chaining machine learning models in production with Algorithmia. Algorithmia blog, 2019

ML MODELS FOR FEATURE EXTRACTION

self driving car



Example: Zong, W., Zhang, C., Wang, Z., Zhu, J., & Chen, Q. (2018). Architecture design and implementation of an autonomous vehicle. IEEE access, 6, 21956-21970.

NONLOCAL EFFECTS IN ML SYSTEMS?

Speaker notes

Improvement in prediction quality in one component does not always increase overall system performance. Have both local model quality tests and global system performance measures.

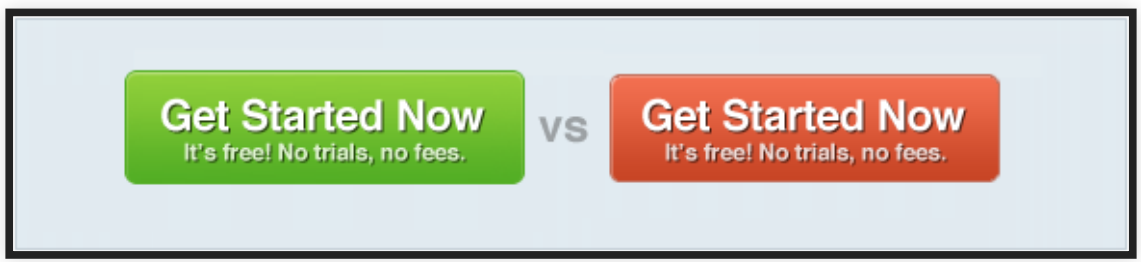
Examples: Slower but more accurate face recognition not improving user experience for unlocking smart phone.

Example: Chaining of models -- second model (language interpretation) trained on output of the first (parts of speech tagging) depends on specific artifacts and biases

Example: More accurate model for common use cases, but more susceptible to gaming of the model (adversarial learning)

RECALL: BETA TESTS AND TESTING IN PRODUCTION

- Test the full system in a realistic setting
- Collect telemetry to identify bugs



RECALL: THE WORLD VS THE MACHINE

- Be explicit about interfaces between world and machine (assumptions, both sensors and actuators)
- No clear specifications between models, limits modular reasoning

RECALL: DETECTING DRIFT

- Monitor data distributions and detect drift
    - Detect data drift between ML components
- Document interfaces in terms of distributions and expectations

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

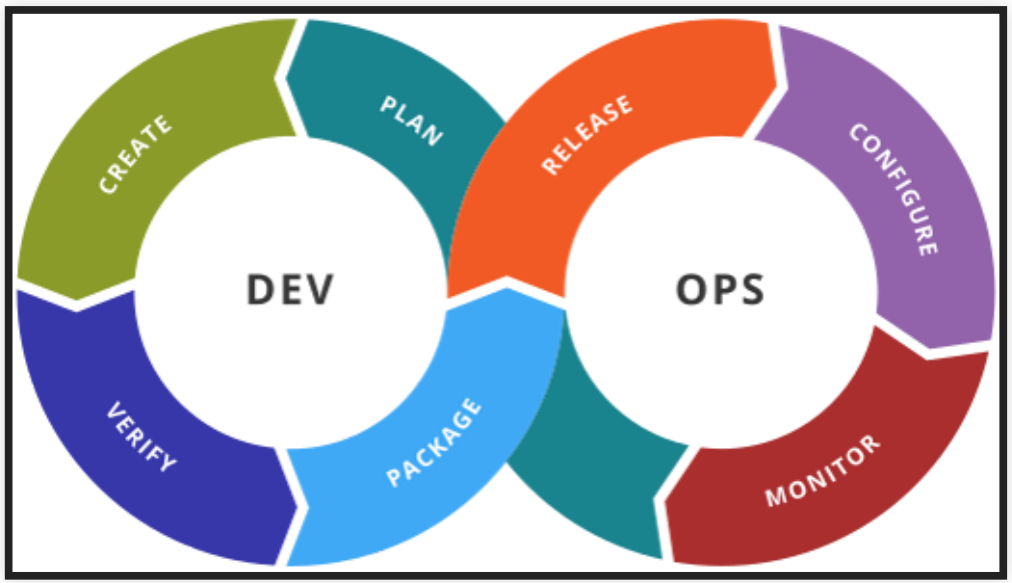
- Allocating hardware resources
- Managing OS updates
- Monitoring performance
- Monitoring crashes
- Managing load spikes, …
- Tuning database performance
- Running distributed at scale
- Rolling back releases
- …

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)

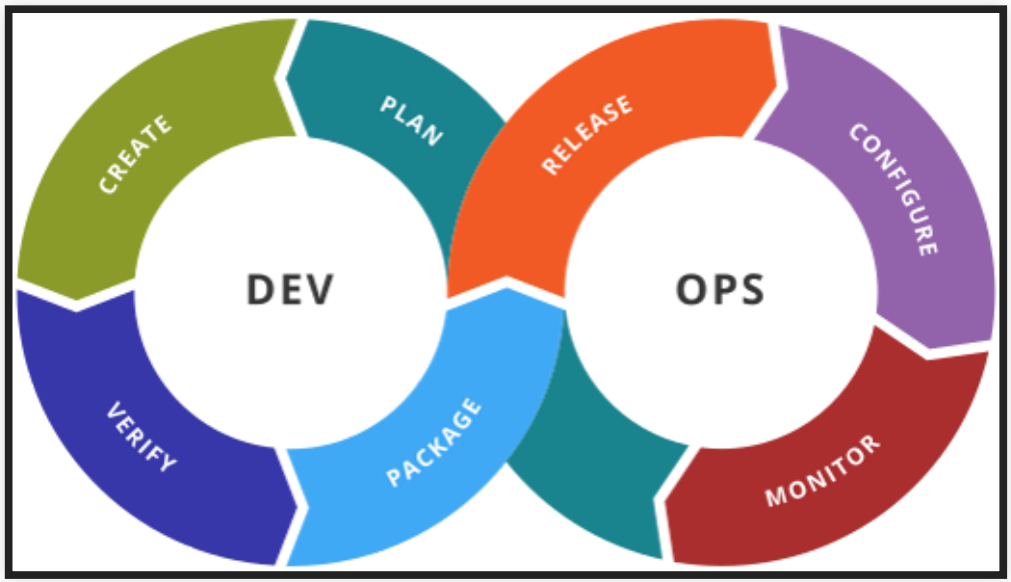
DEVOPS


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

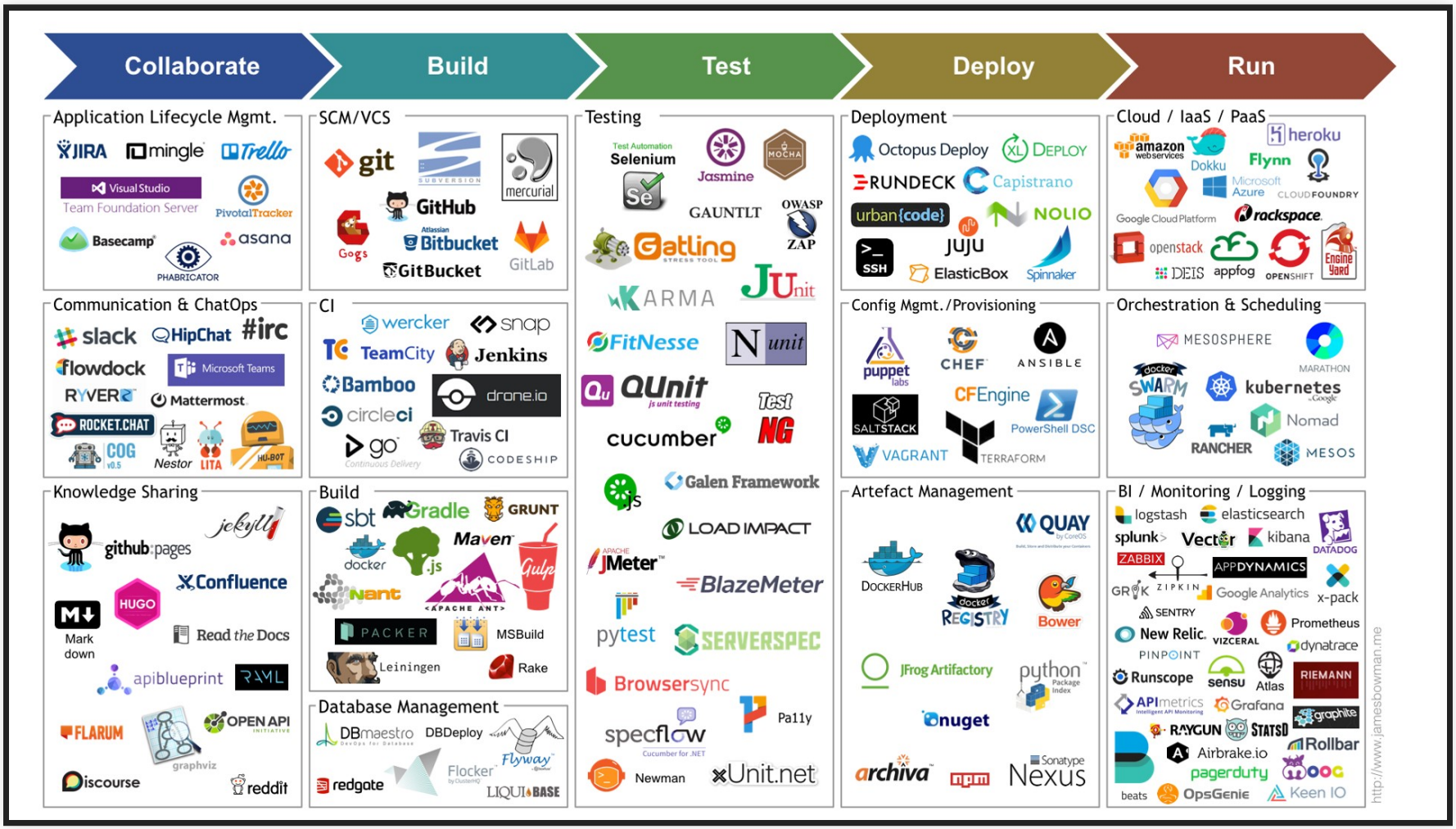
- Buzz word!


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

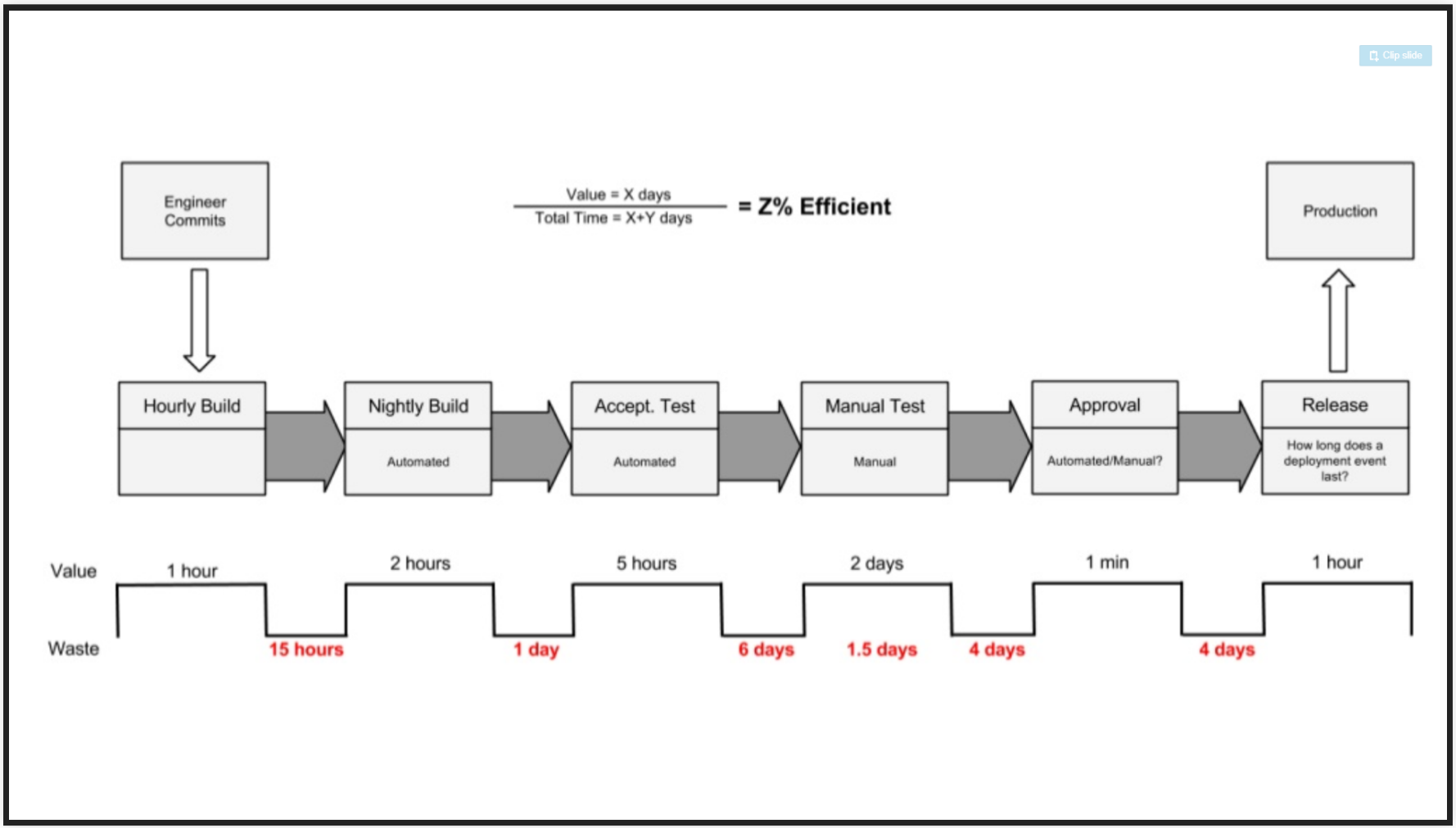


http://www.jamesbowman.me

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



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

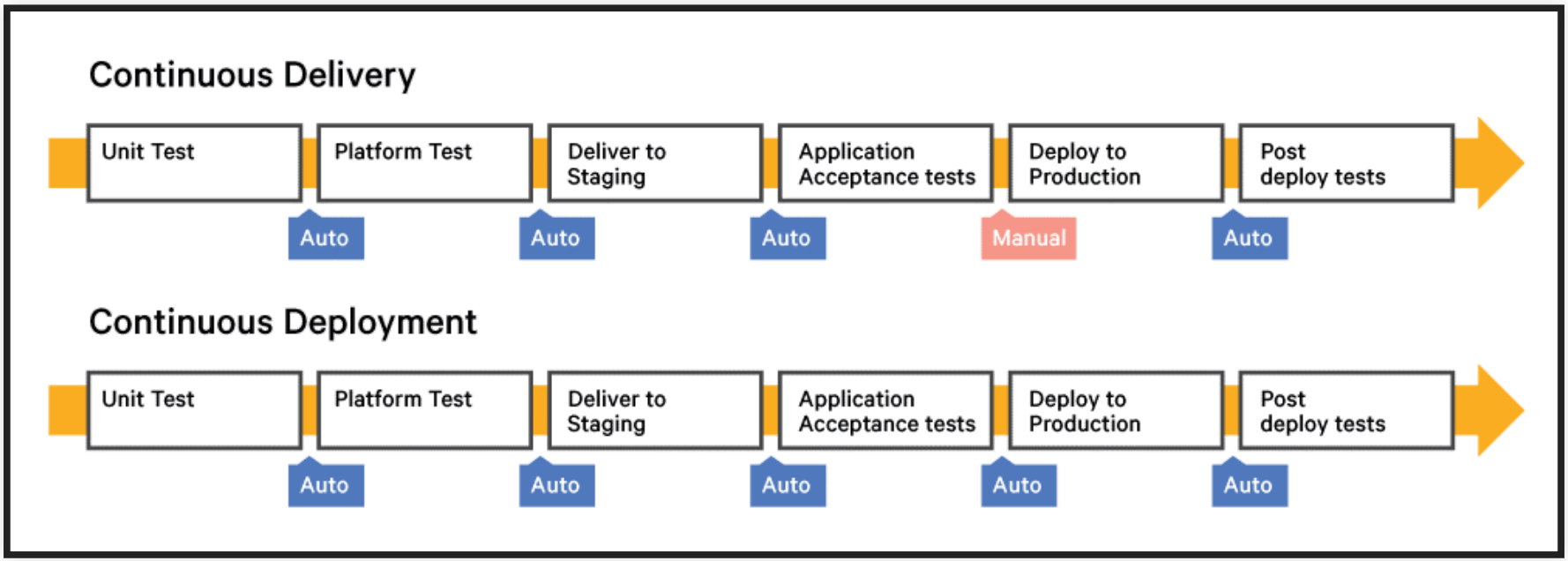
TYPICAL MANUAL STEPS IN DEPLOYMENT?

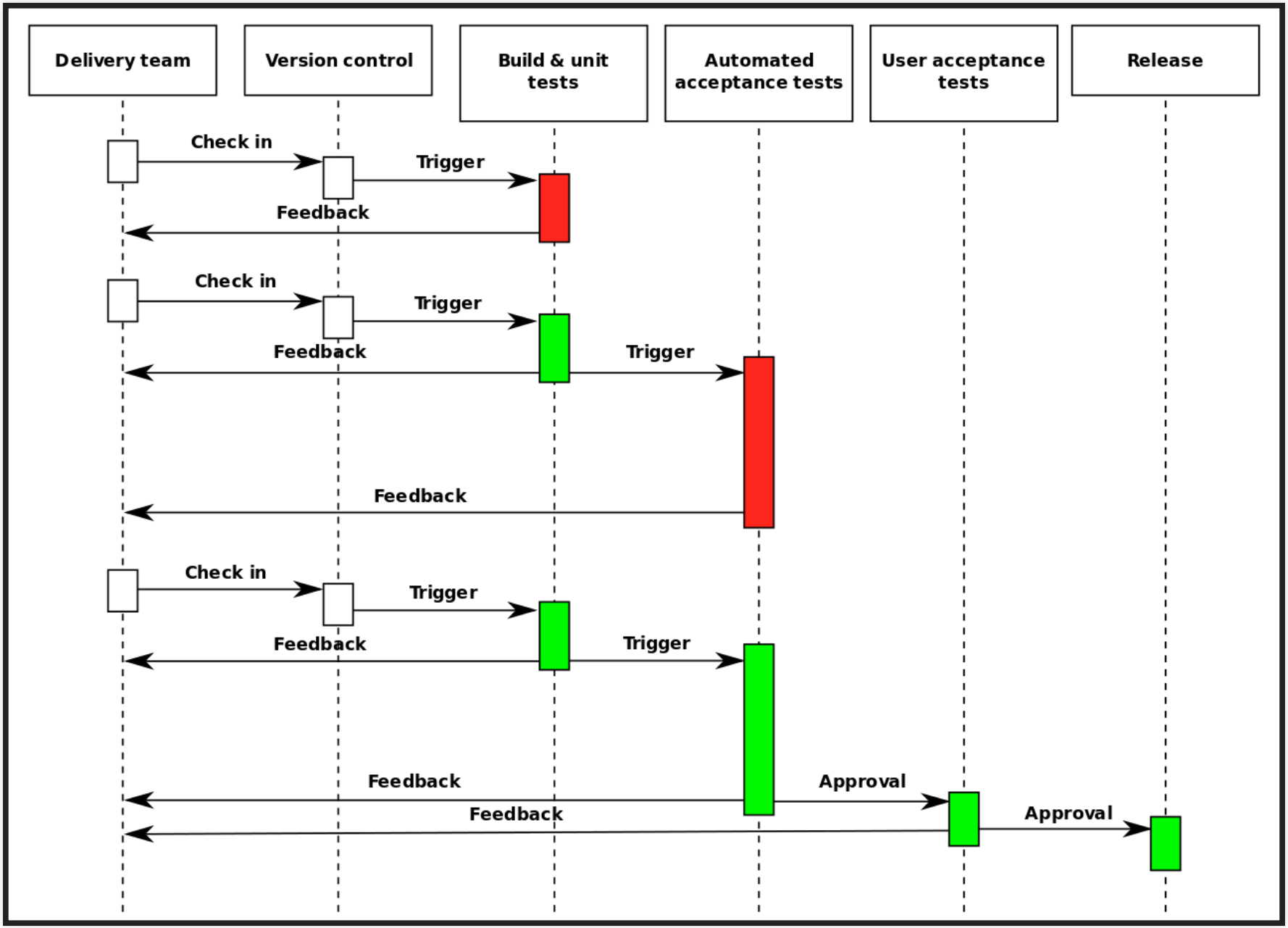
CONTINUOUS DELIVERY

- Full automation from commit to deployable container
- Heavy focus on testing, reproducibility and rapid feedback
- Deployment step itself is manual
- Makes process transparent to all developers and operators

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


Continuous Delivery
Unit Test
Platform Test
Deliver to Staging
Application Acceptance tests
Deploy to Production
Post deploy tests
Auto
Auto
Auto
Manual
Auto
Continuous Deployment
Unit Test
Platform Test
Deliver to Staging
Application Acceptance tests
Deploy to Production
Post deploy tests
Auto
Auto
Auto
Auto
Auto


Delivery team
Version control
Build & unit tests
Automated acceptance tests
User acceptance tests
Release
Check in
Trigger
Feedback
Check in
Trigger
Feedback
Trigger
Feedback
Check in
Trigger
Feedback
Trigger
Feedback
Approval
Feedback
Approval

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.

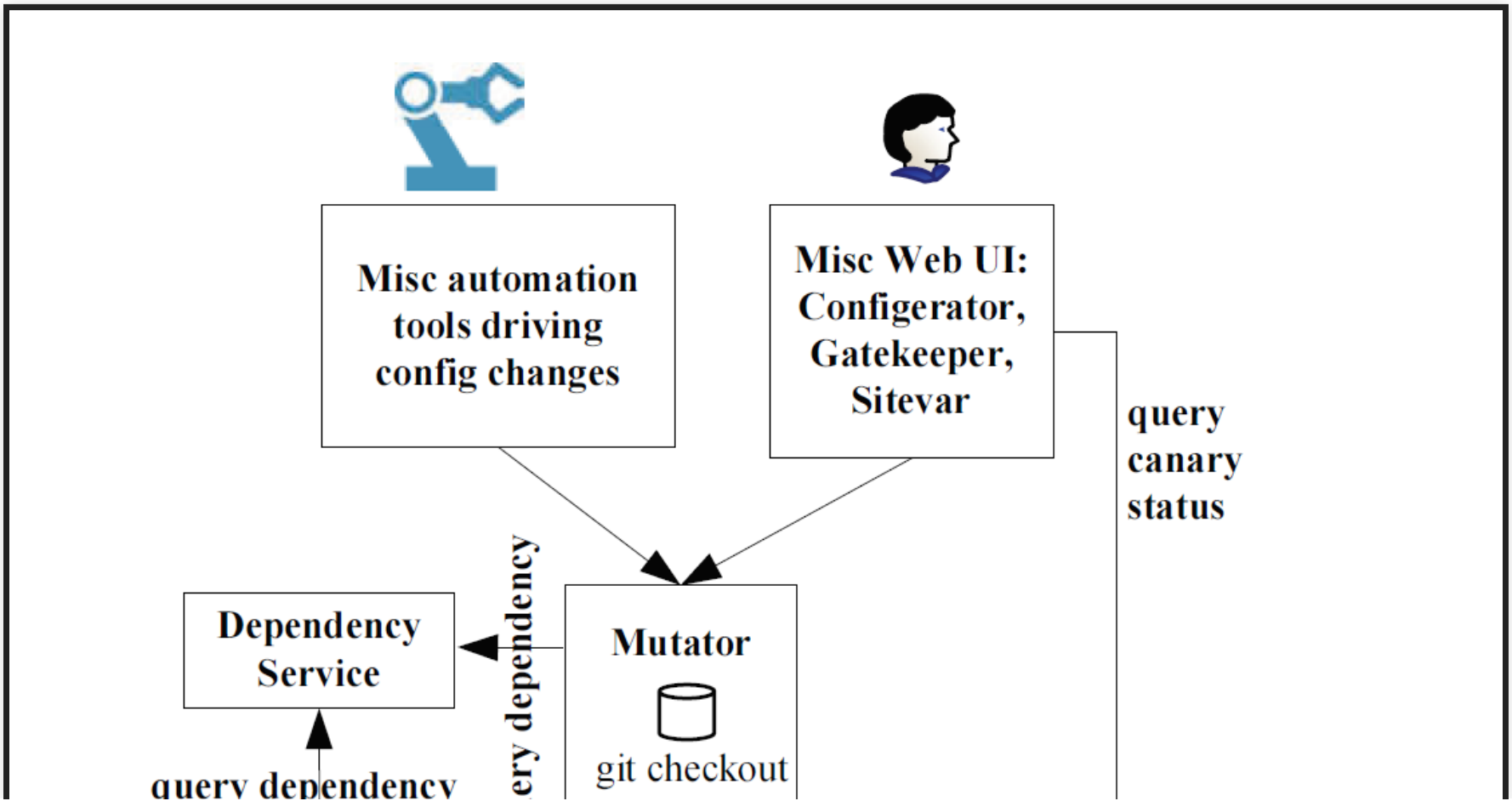
RELEASE CHALLENGES FOR MOBILE APPS

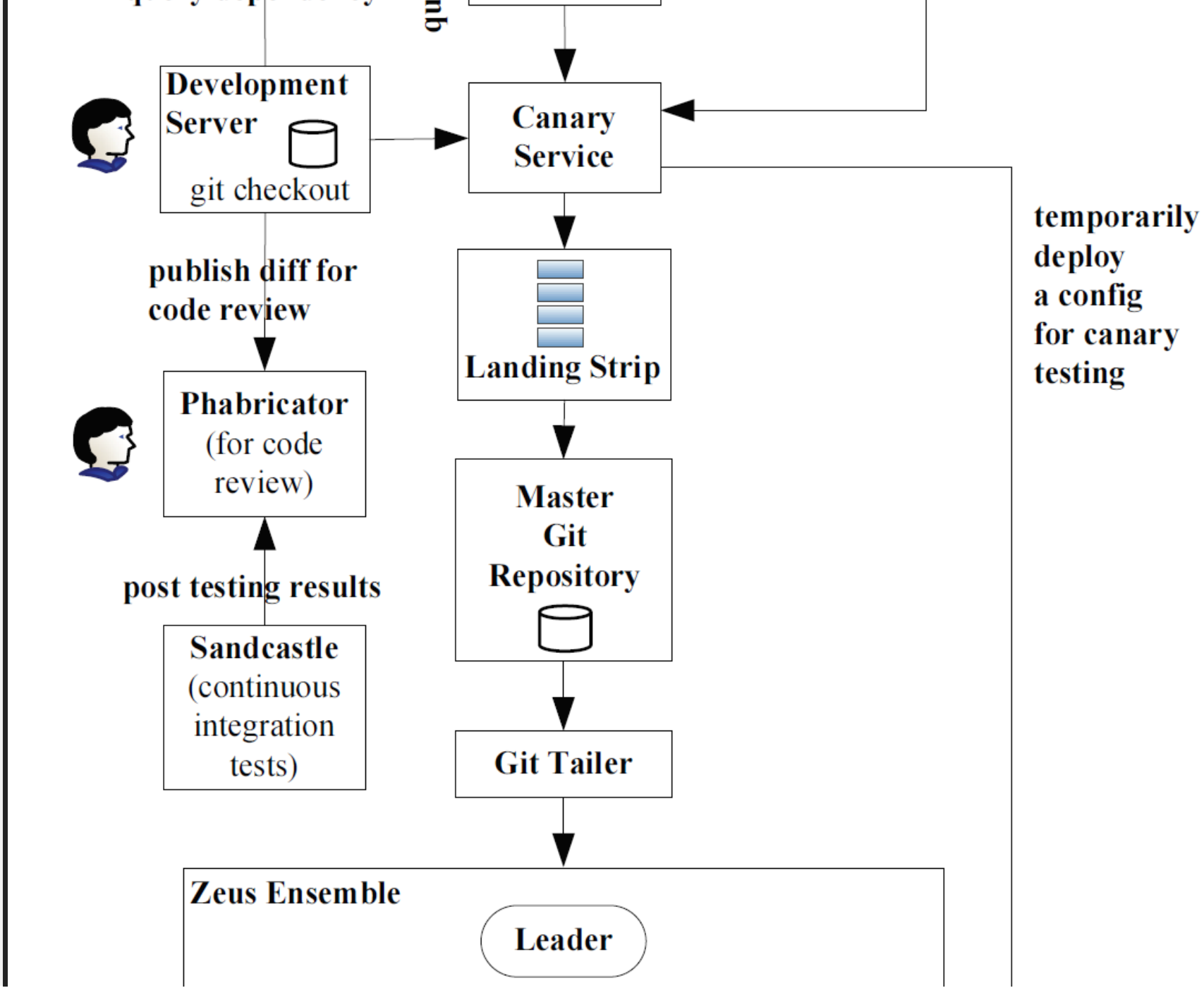
- Large downloads
- Download time at user discretion
- Different versions in production
- Pull support for old releases?

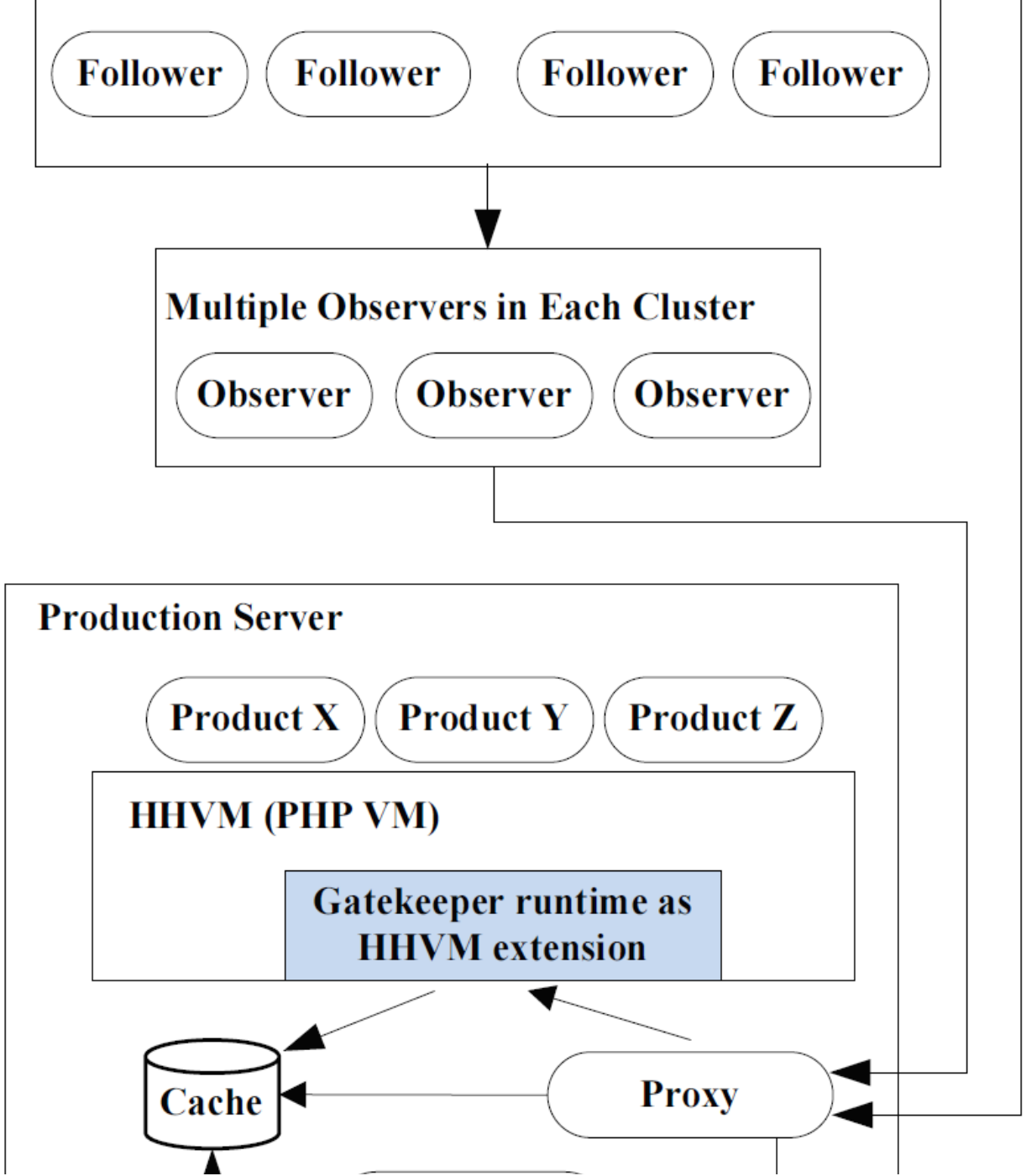
- Server side releases silent and quick, consistent

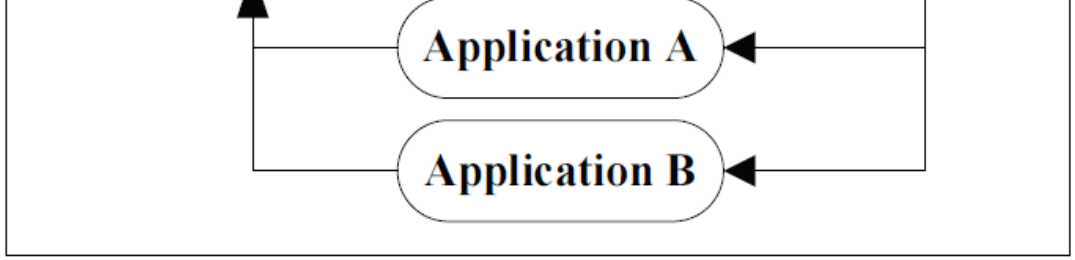
- -> App as container, most content + layout from server

REAL-WORLD PIPELINES ARE COMPLEX




Development Server
git checkout
Canary Service
publish diff for code review
Landing Strip
temporarily deploy a config for canary testing
Phabricator (for code review)
Master Git Repository
post testing results
Sandcastle (continuous integration tests)
Git Tailer
Zeus Ensemble
Leader

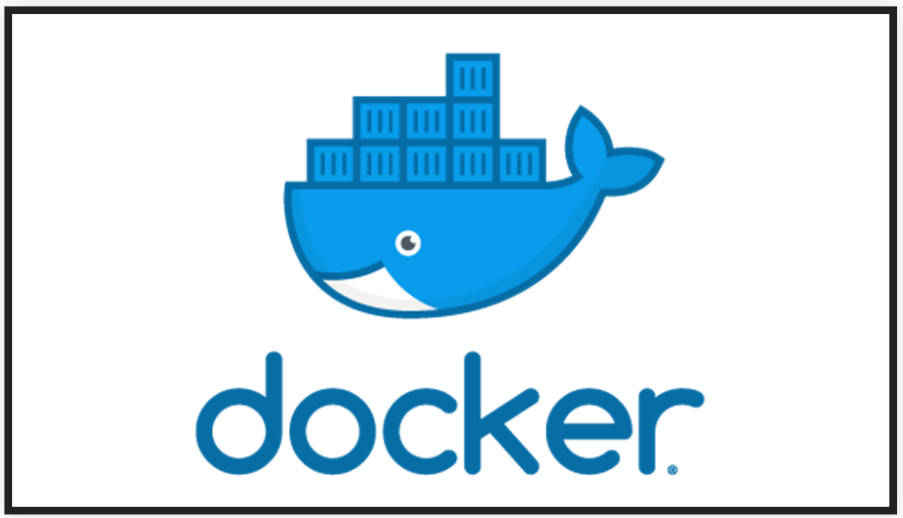

Follower
Follower
Follower
Follower
Multiple Observers in Each Cluster
Observer
Observer
Observer
Production Server
Product X
Product Y
Product Z
HHVM (PHP VM)
Gatekeeper runtime as HHVM extension
Cache
Proxy


Application A
Application B

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 management, and application-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_server
...
```

```
# This role deploys the mongod processes and
- name: create data directory for mongodb
  file: path={{ mongodb_datadir_prefix }}/mon
  delegate_to: '{{ item }}'
  with_items: groups.replication_servers

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

- name: Create the mongodb startup file
  template: src=mongod.j2 dest=/etc/init.d/mo
  delegate_to: '{{ item }}'
  with_items: groups.replication_servers


- name: Create the mongodb configuration file
```

PUPPET EXAMPLE

Declarative specification, can be applied to many machines

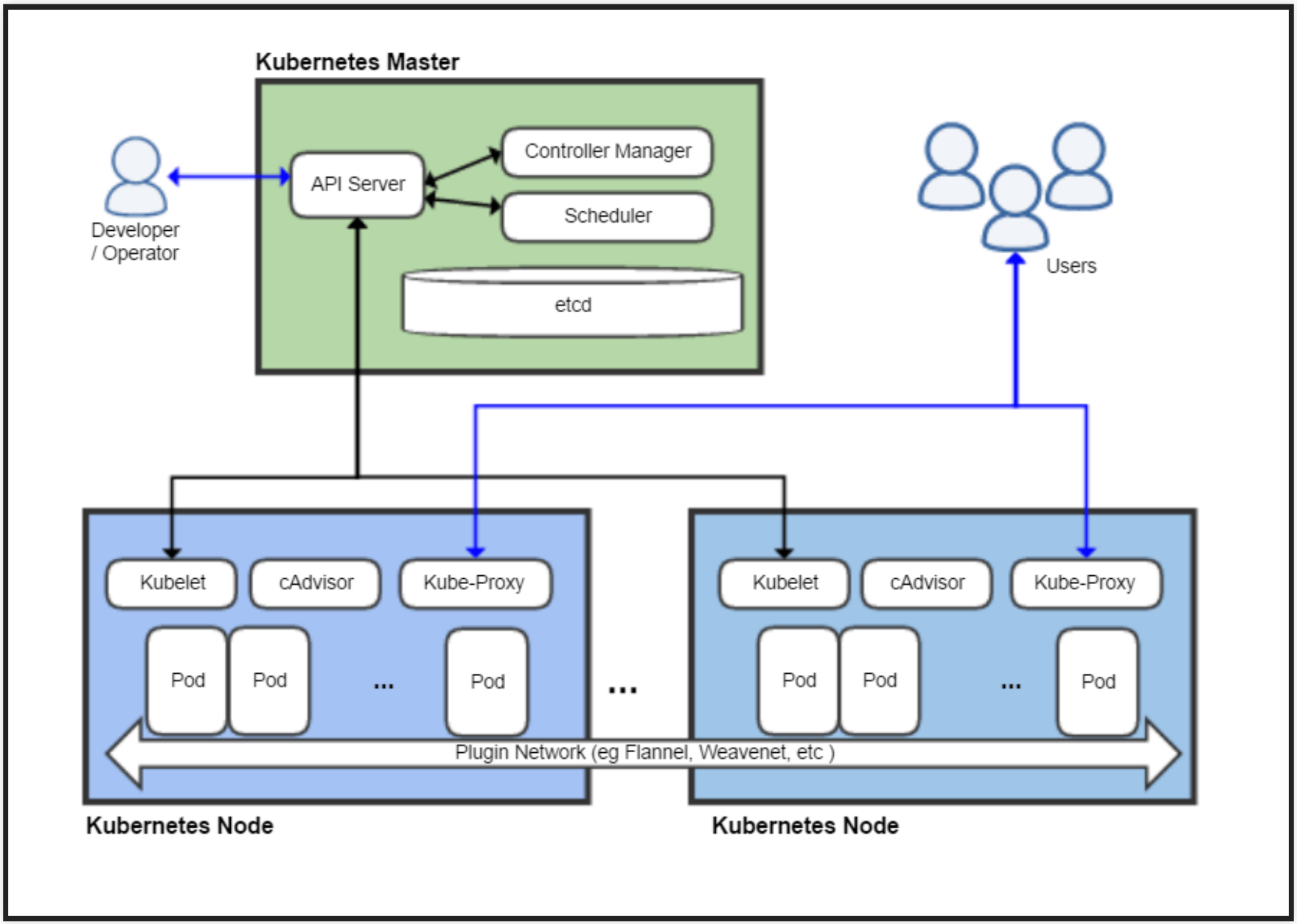
```
$doc_root = "/var/www/example"

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

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

file { $doc_root:
 ensure => "directory",
 owner => "www-data",
 group => "www-data",
 mode => 644
```

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


Kubernetes Master
API Server
Controller Manager
Scheduler
etcd
Developer
/ Operator
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

MONITORING

- Monitor server health
- Monitor service health
- 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

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

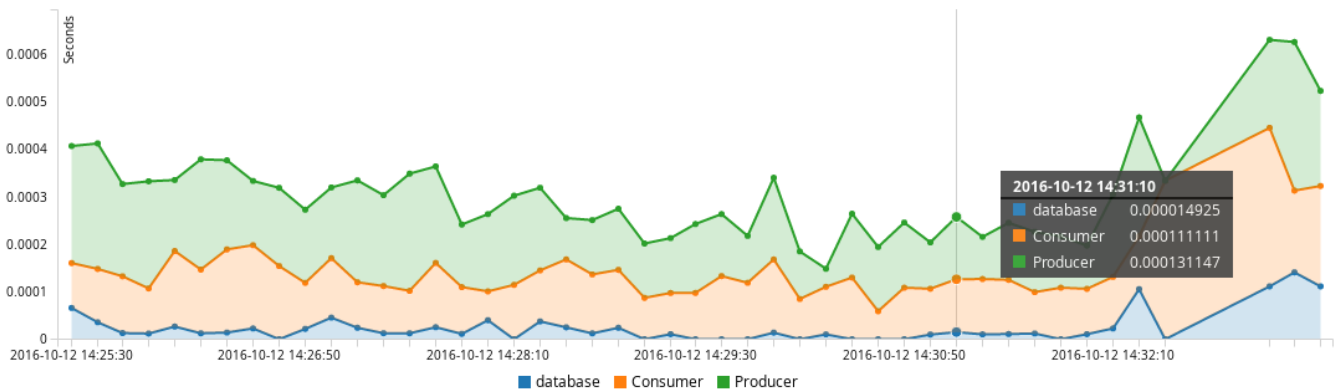
Text

Contains text

Host Name

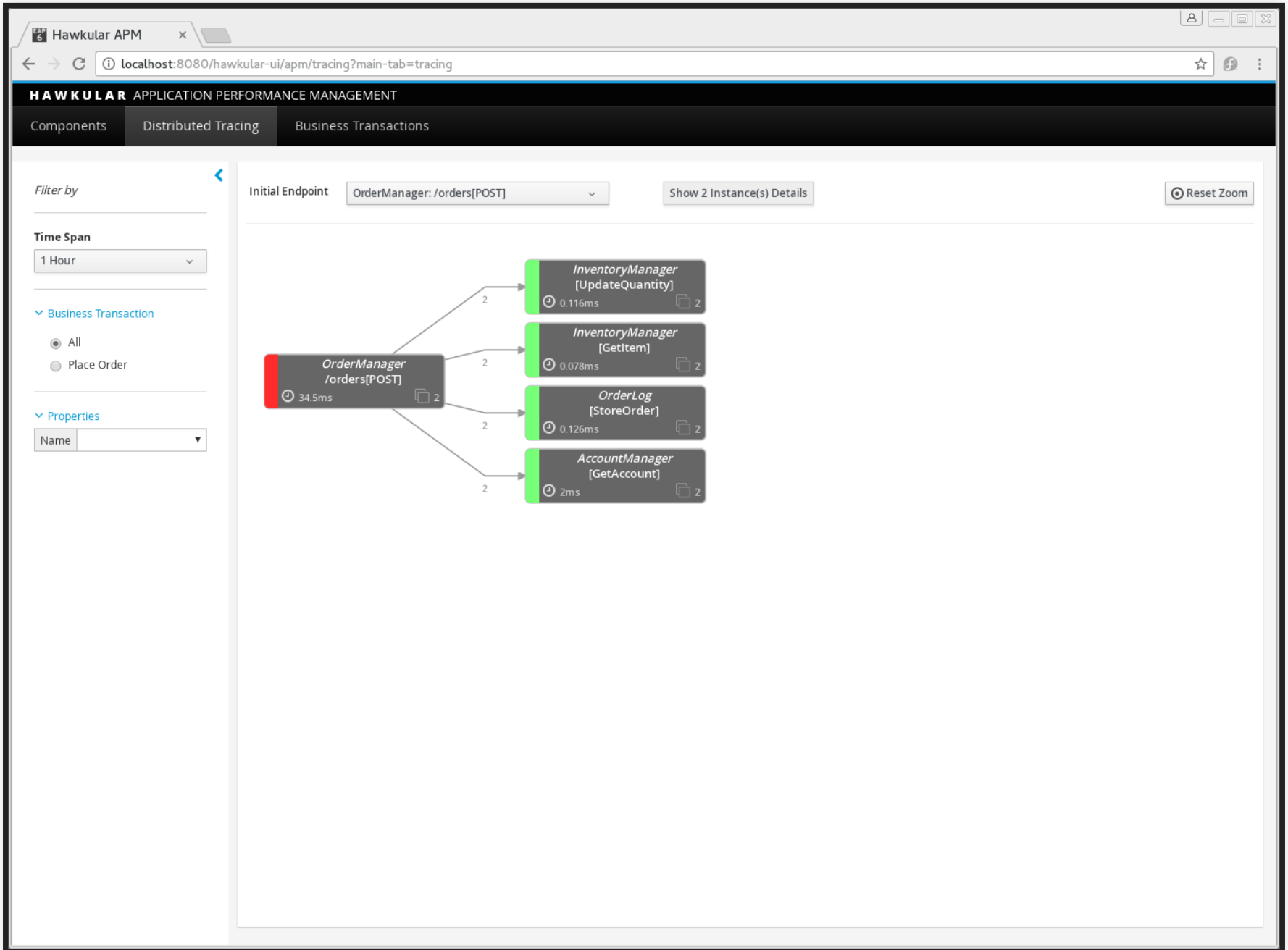
Enter a host name

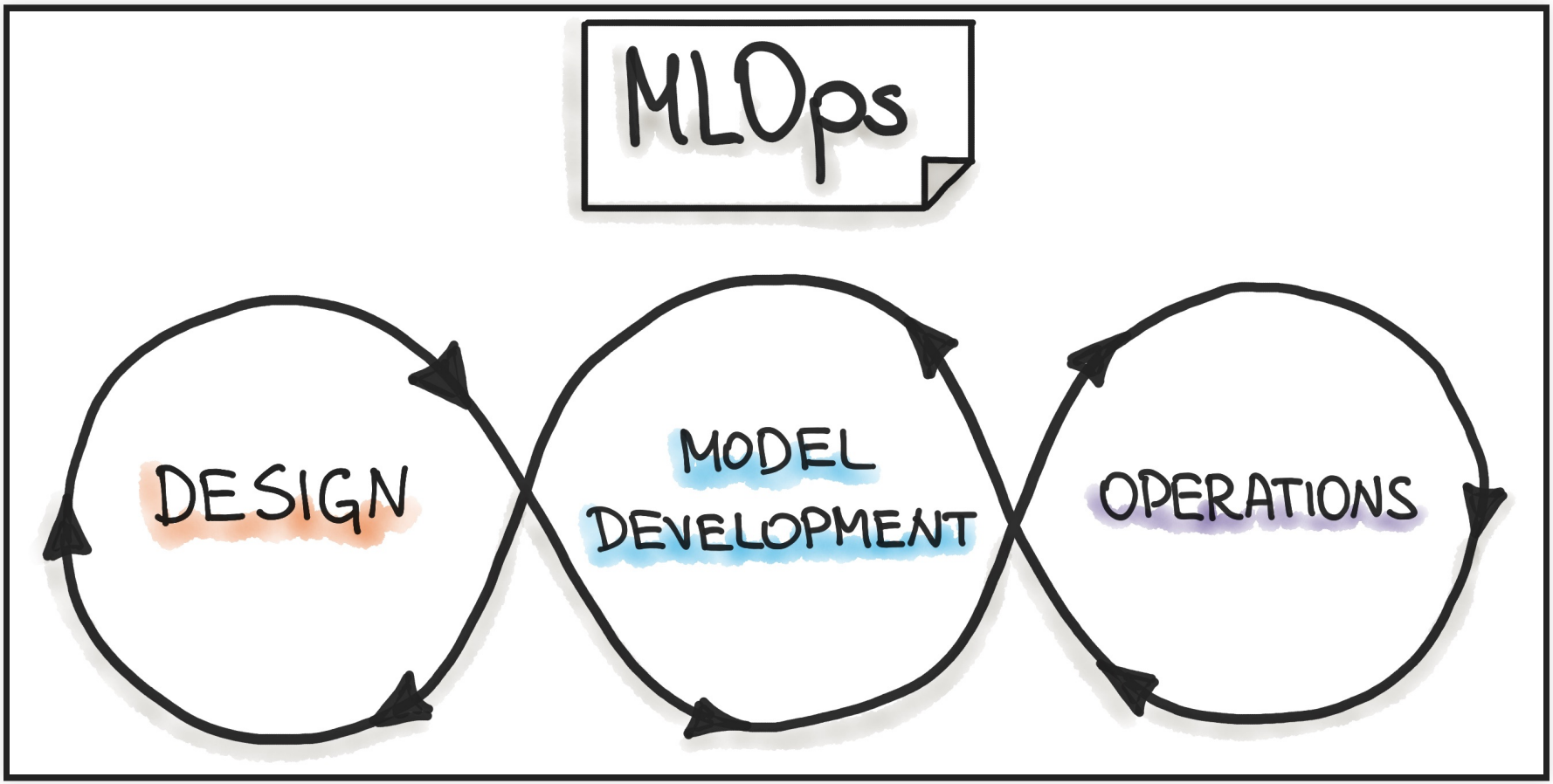
Aggregation Interval: 10 Second | Overlap Data | **Last Update:** 12 Oct 2016 14:35:32 | Pause Live Data



	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
	0.000	0.000	535	consumer		StoreOrder
	0.000	0.000	535	consumer		UpdateQuantity
	0.000	0.000	140	consumer		GetOrders
	0.000	0.000	1102	database	InventoryDB	QueryInventory
	0.000	0.000	535	database	InventoryDB	WriteInventory
	0.000	0.000	1320	database	AccountsDB	RetrieveAccount

HAWKULAR


Hawkular APM
localhost:8080/hawkular-ui/apm/tracing?main-tab=tracing
HAWKULAR APPLICATION PERFORMANCE MANAGEMENT
Components
Distributed Tracing
Business Transactions
Filter by
Time Span
1 Hour
Business Transaction
All
Place Order
Properties
Name
Initial Endpoint
OrderManager: /orders[POST]
Show 2 Instance(s) Details
Reset Zoom
OrderManager
/orders[POST]
34.5ms
2
InventoryManager
[UpdateQuantity]
0.116ms
2
InventoryManager
[GetItem]
0.078ms
2
OrderLog
[StoreOrder]
0.126ms
2
AccountManager
[GetAccount]
2ms
2
2
2
2
2



https://ml-ops.org/

ON TERMINOLOGY

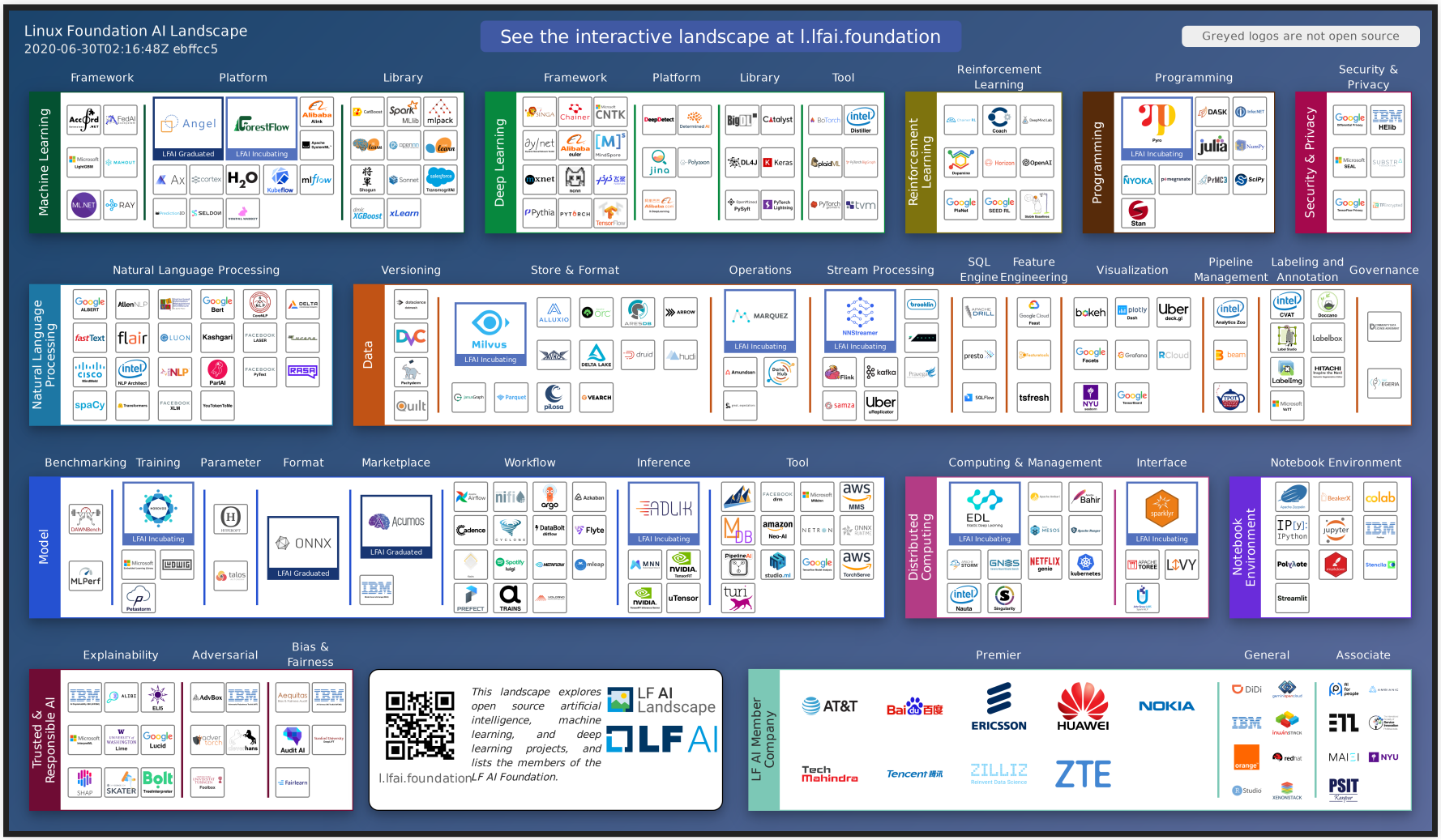
- Many vague buzzwords, often not clearly defined
- *MLOps*: Collaboration and communication between data scientists and operators, e.g.,
    - Automate model deployment
    - Model training and versioning infrastructure
    - Model deployment and monitoring
- *AIOps*: Using AI/ML to make operations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

MLOPS OVERVIEW

- Integrate ML artifacts into software release process, unify process
- Automated data and model validation (continuous deployment)
- Data engineering, data programming
- Continuous deployment for ML models
    - From experimenting in notebooks to quick feedback in production
- Versioning of models and datasets
- Monitoring in production

Further reading: MLOps principles

TOOLING LANDSCAPE LF AI



Linux Foundation AI Initiative

SUMMARY

- Beyond model and data quality: Quality of the infrastructure matters, danger of silent mistakes
- Many SE techniques for test automation, testing robustness, test adequacy, testing in production useful for infrastructure quality
- Lack of modularity: local improvements may not lead to global improvements
- DevOps: Development vs Operations challenges
    - Automated configuration
    - Telemetry and monitoring are key
    - Many, many tools