Continuous Delivery with Containers: *The Good, the Bad, and the Ugly*

Daniel Bryant

@danielbryantuk

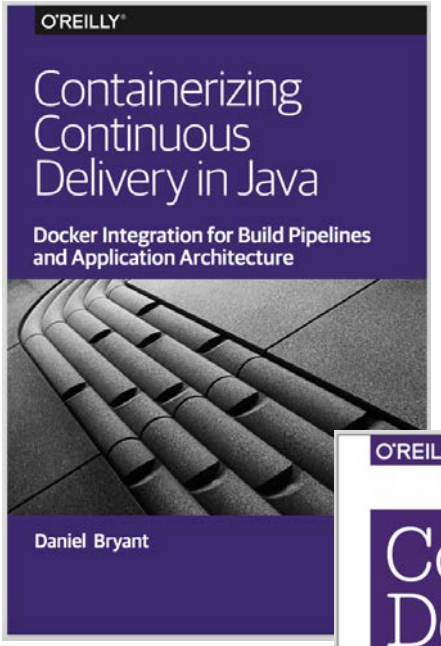
SpectoLabs

Containers: Expectations versus reality

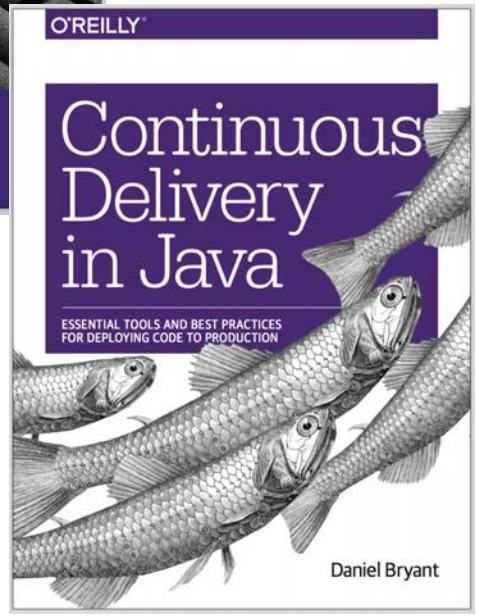
“DevOps”

Setting the scene…

- Continuous delivery is a large topic
  - No business focus today (value stream etc)
  - PaaS and Serverless are super interesting…
  - But I'm assuming you're all-in on containers
- Focusing today on the process and tooling
  - No live coding today
  - Mini-book contains more details (thanks nginx!)



bit.ly/2jWDSF7



TL;DR – Containers and CD

- Container image becomes the build pipeline ‘single binary’
- Adding metadata to containers images is vital, but challenging
- Must validate container constraints (NFRs)
  - Cultivate container ‘mechanical sympathy’

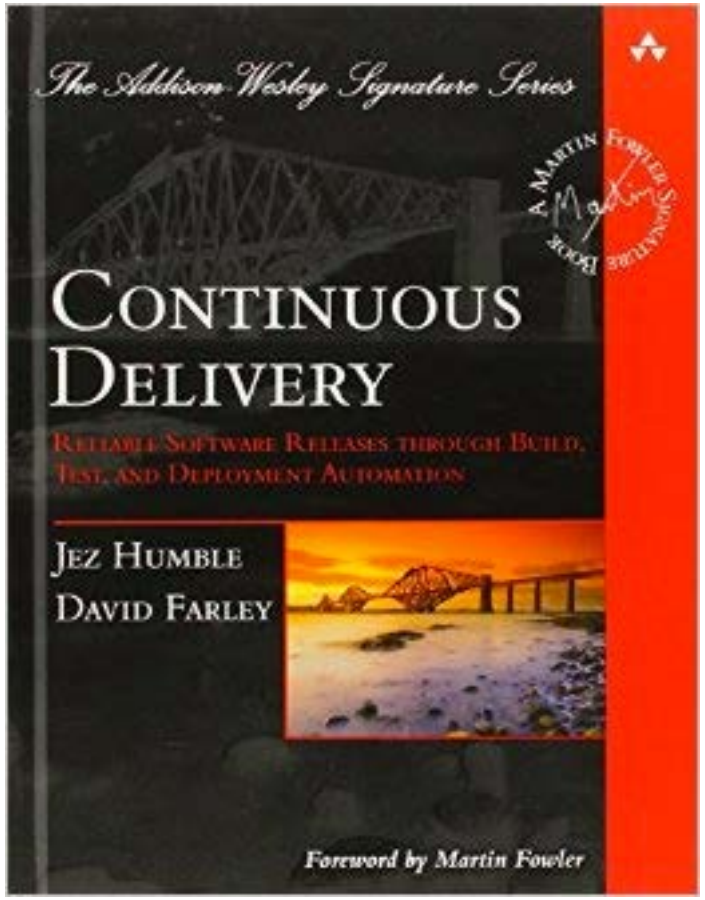
@danielbryantuk

- Independent Technical Consultant, CTO at SpectoLabs
  - Architecture, DevOps, Java, microservices, cloud, containers
  - Continuous Delivery (CI/CD) advocate
  - Leading change through technology and teams

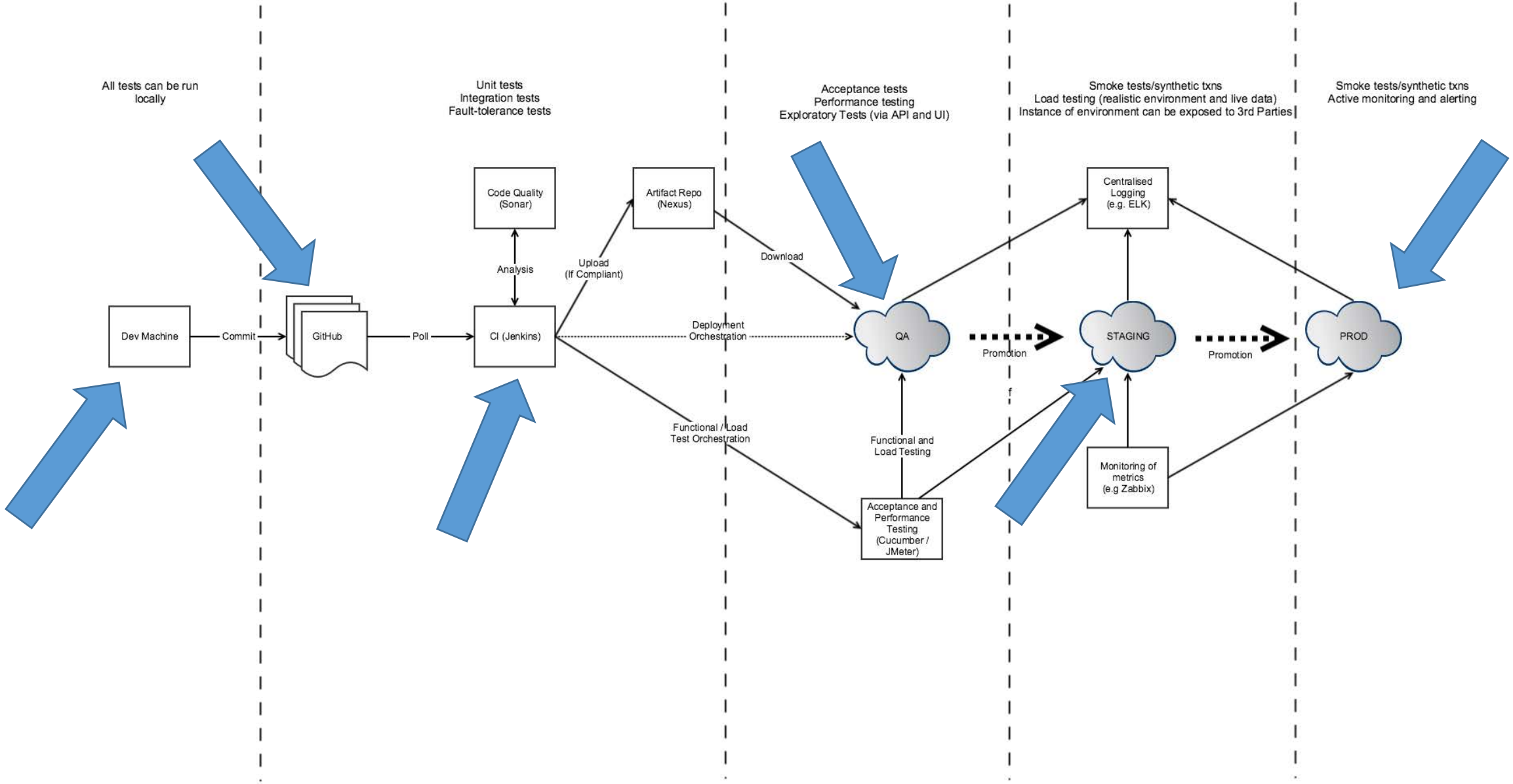
Continuous Delivery

Continuous Delivery

- Produce valuable and robust software in short cycles
- Optimising for feedback and learning
- Not (necessarily) Continuous Deployment

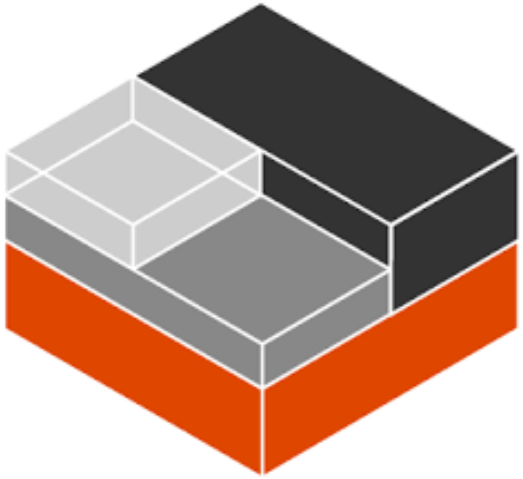


Creation of a build pipeline is mandatory for continuous delivery

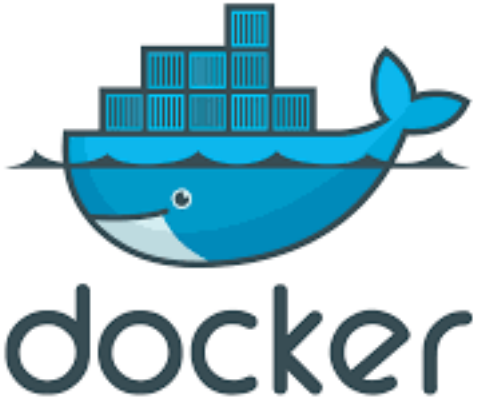

All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests
Performance testing
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
CI (Jenkins)
Code Quality (Sonar)
Analysis
Upload (If Compliant)
Artifact Repo (Nexus)
Download
Deployment Orchestration
Functional / Load Test Orchestration
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)

The Impact of containers on CD

Container technology (and CD)



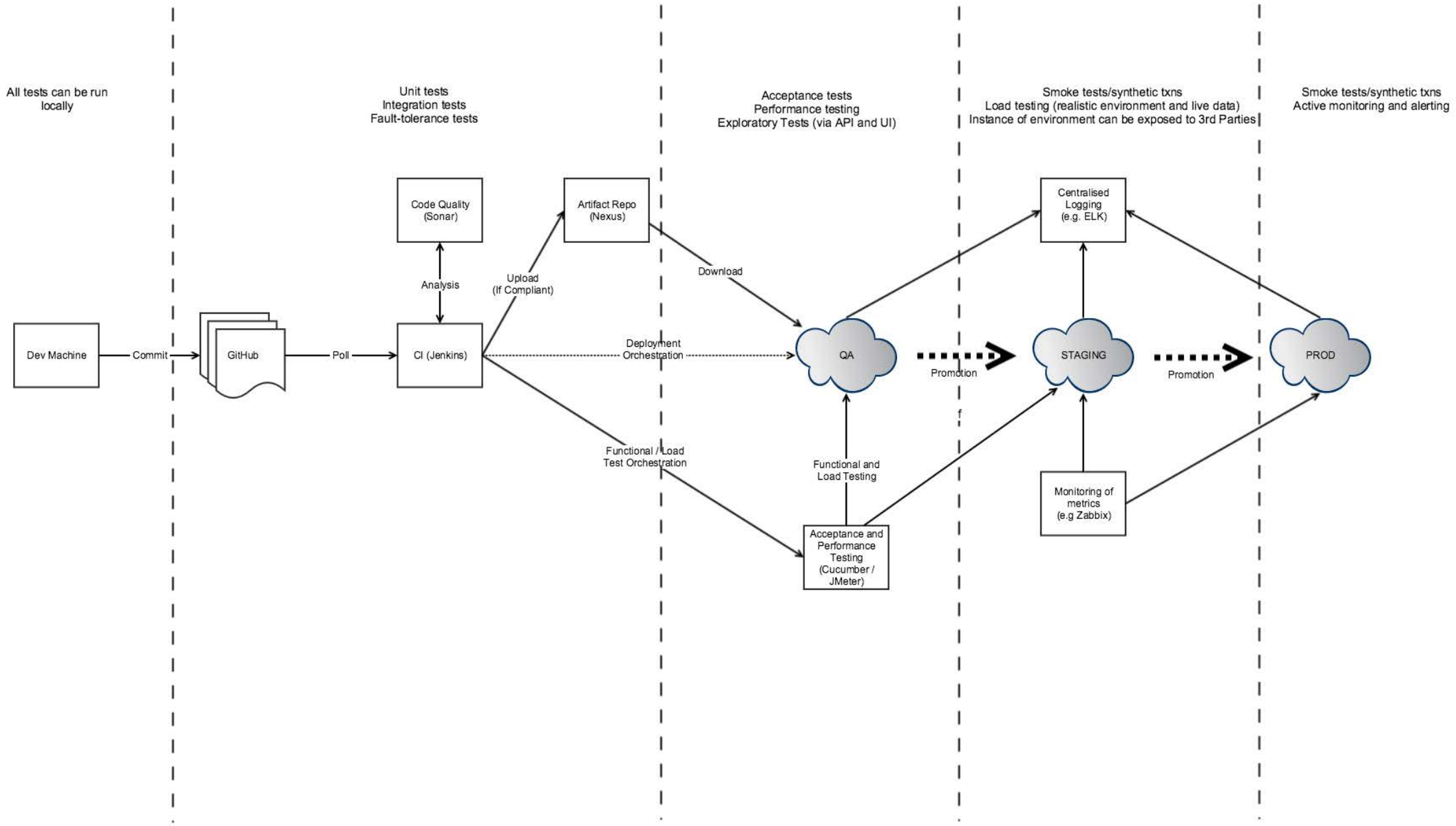
- OS-level virtualisation
  - cgroups, namespaces, rootfs

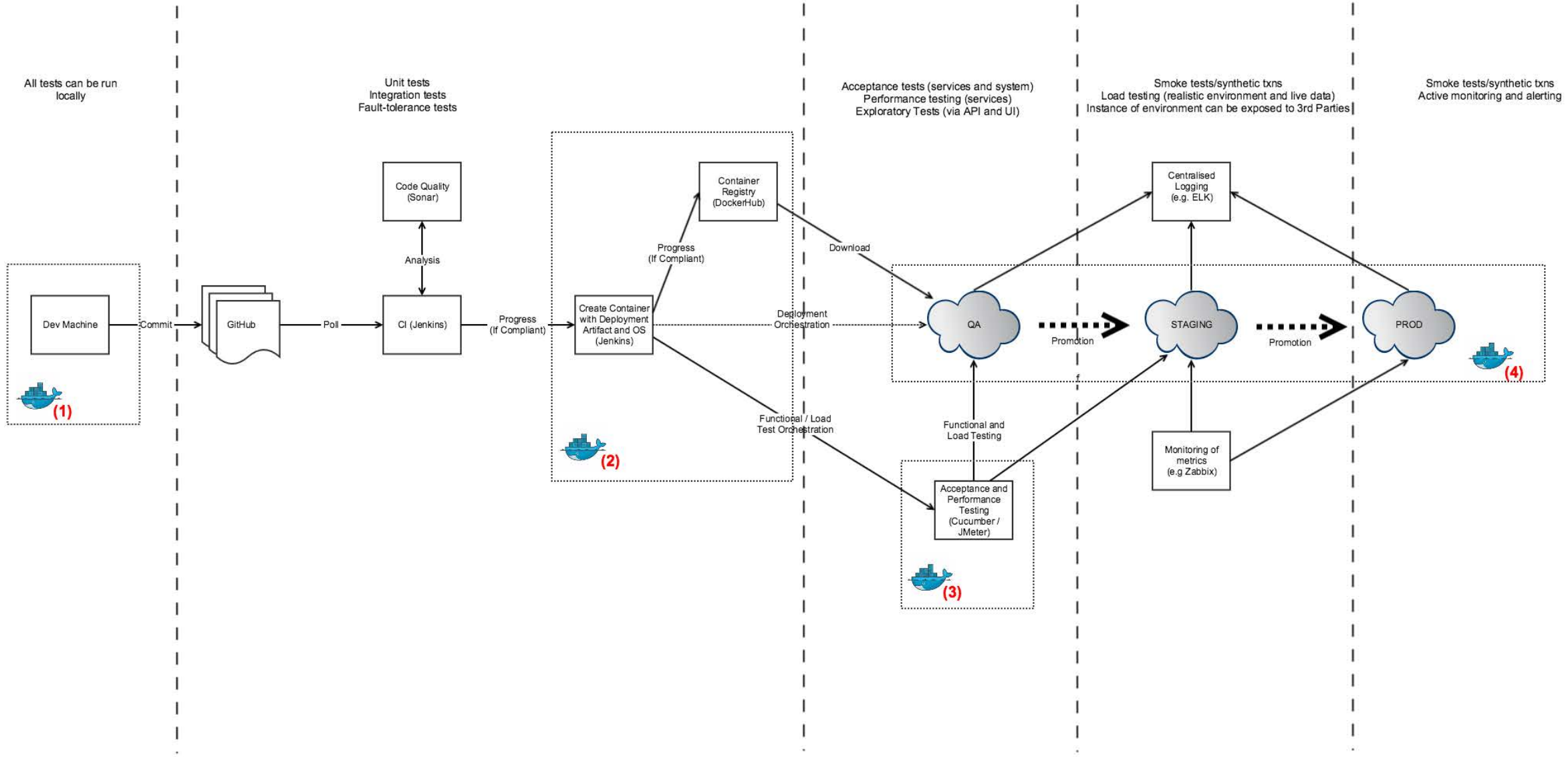


- Package and execute software

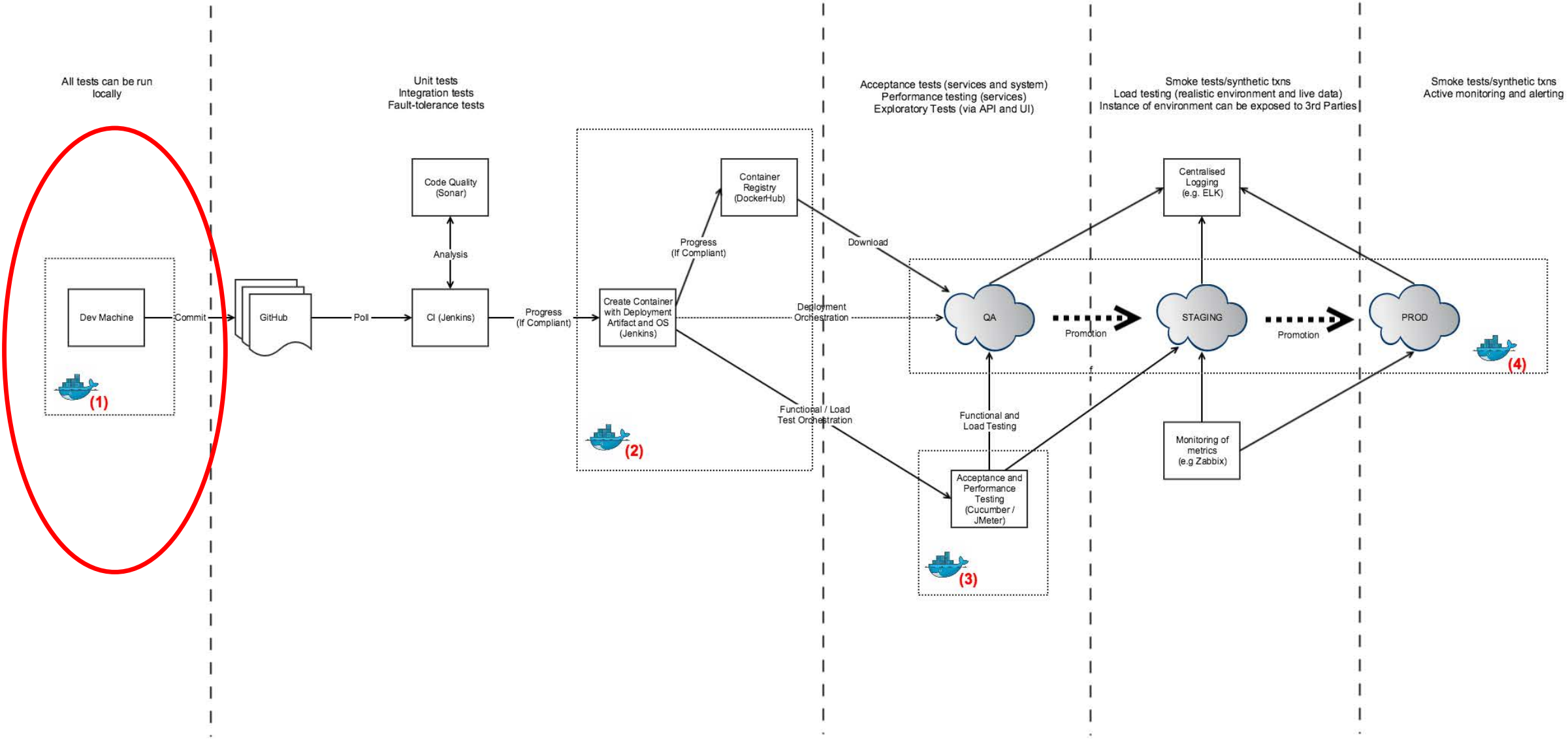
Windows Server 2016

- **Container image == ‘single binary’**


All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests
Performance testing
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Code Quality (Sonar)
Artifact Repo (Nexus)
Centralised Logging (e.g. ELK)
Analysis
Upload (If Compliant)
Download
Dev Machine
Commit
GitHub
Poll
CI (Jenkins)
Deployment Orchestration
QA
Promotion
STAGING
Promotion
PROD
Functional / Load Test Orchestration
Functional and Load Testing
Monitoring of metrics (e.g Zabbix)
Acceptance and Performance Testing (Cucumber / JMeter)

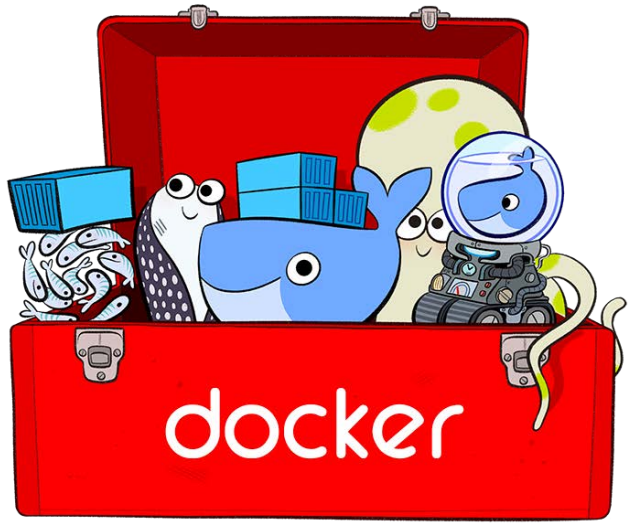

All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Progress (If Compliant)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)

Creating a pipeline for containers

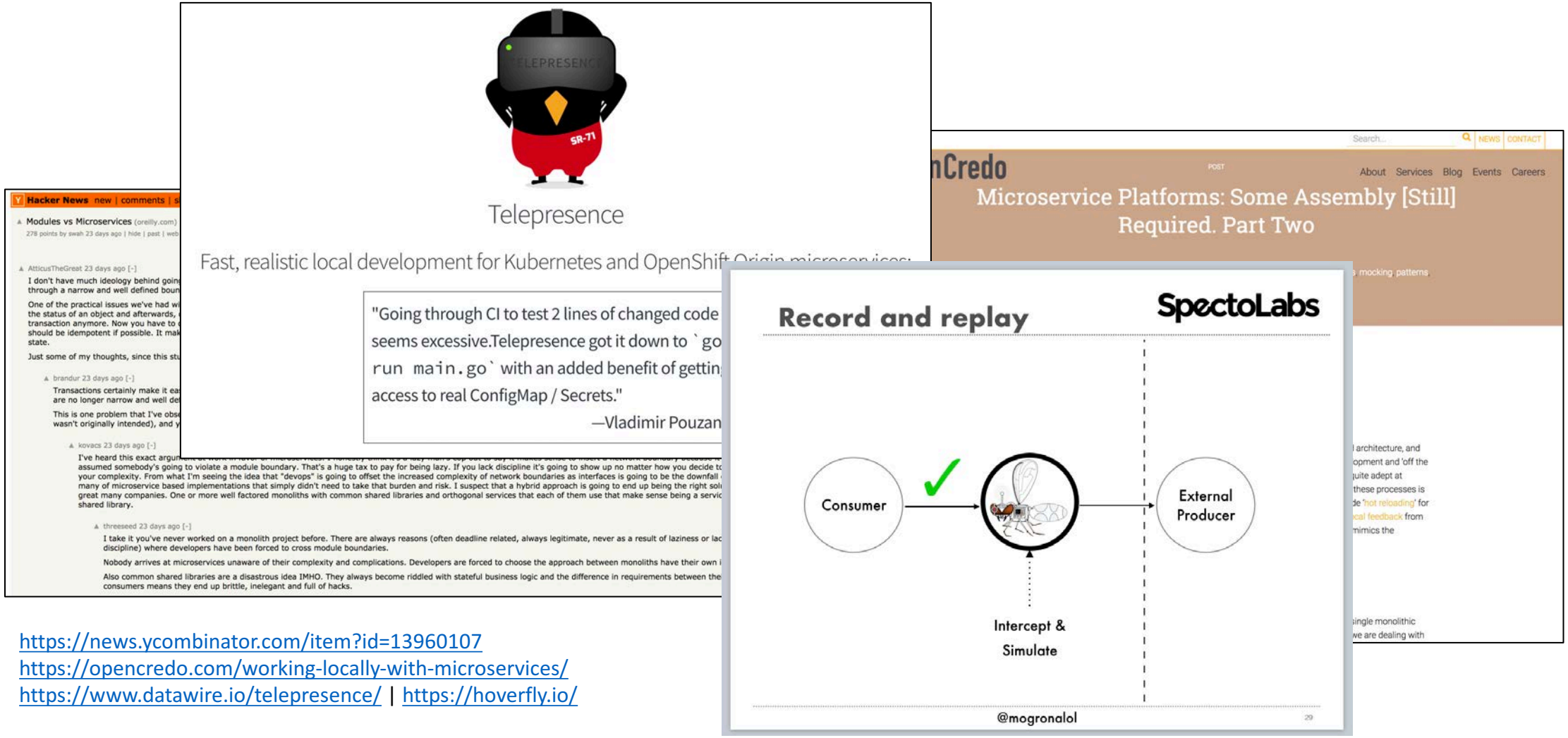

All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Progress (If Compliant)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)

Make your dev environment like production

- Develop locally or copy/code in container
- Must build/test containers locally
  - Perform (at least) happy path tests



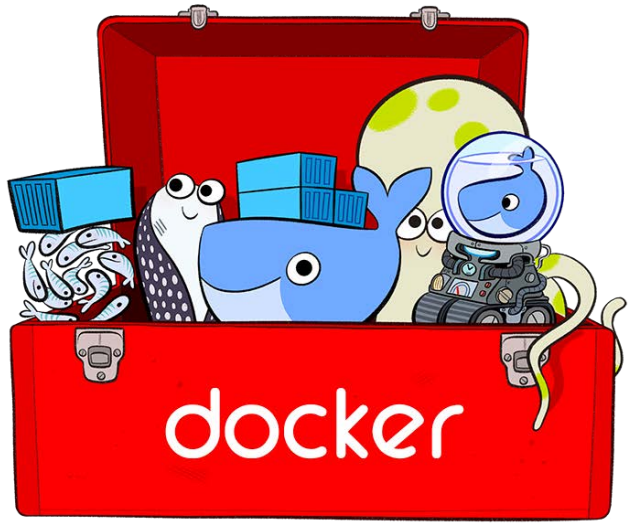
Quick Aside: Running *entire* system locally



https://news.ycombinator.com/item?id=13960107
https://opencredo.com/working-locally-with-microservices/
https://www.datawire.io/telepresence/ | https://hoverfly.io/

Make your dev environment like production

- Develop locally or copy/code in container
- Must build/test containers locally
  - Perform (at least) happy path tests
- Use identical base images from production
  - With same configuration



Lesson learned: Dockerfile content is **super** important

```
1 FROM openjdk:8-jre
2 ADD target/shopfront-0.0.1-SNAPSHOT.jar app.jar
3 EXPOSE 8010
4 ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

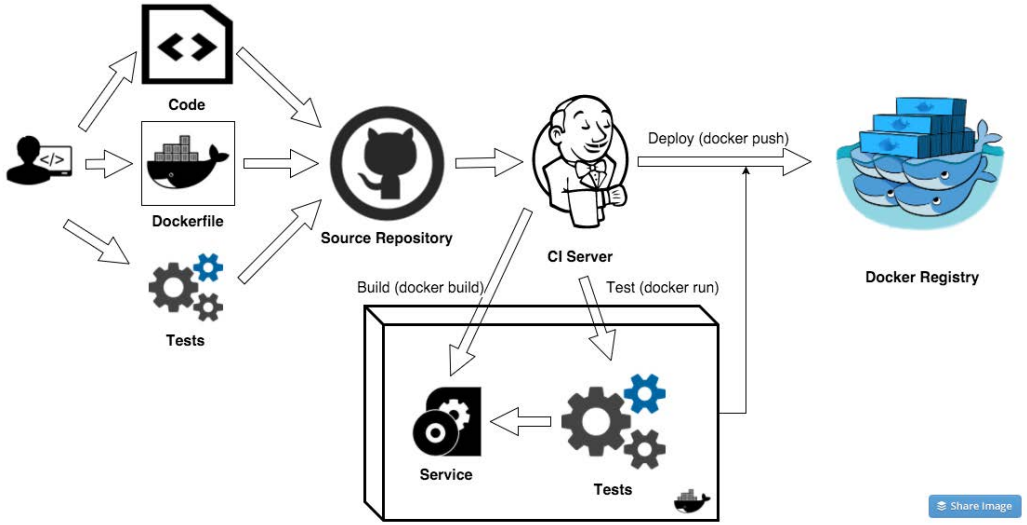
- OS choice
- Configuration
- Build artifacts
- Exposing ports

- Java
  - JDK vs JRE and Oracle vs OpenJDK?
- Golang
  - Statically compiled binary in scratch?
- Python
  - Virtualenv?

Please talk to the sysadmin people:
Their operational knowledge is invaluable

Different test and prod containers?

- Create “test” version of container
  - Full OS (e.g. Ubuntu)
  - Test tools and data
- Easy to see **app/configuration drift**
- Use test sidecar containers instead
- ONTEST proposal by Alexi Ledenev



http://blog.terranillius.com/post/docker_testing/

Docker multi-stage builds


```
FROM golang:1.7.3 as builder

WORKDIR /go/src/github.com/alexellis/href-counter/

RUN go get -d -v golang.org/x/net/html
COPY app.go    .

RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates

WORKDIR /root/

COPY --from=builder /go/src/github.com/alexellis/href-counter/app    .

CMD ["./app"]
```

http://blog.alexellis.io/mutli-stage-docker-builds/
https://github.com/moby/moby/pull/31257
https://github.com/moby/moby/pull/32063

Java specific stuff…

This repository Search Pull requests Issues Marketplace Explore

oracle / docker-images

Watch 247 Star 1,486 Fork 1,361

Code Issues 32 Pull requests 8 Projects 1 Insights

Branch: master docker-images / OracleJava / Create new file Upload files Find file History

aureliogrb committed with brunoborges Updated files from 8u131 to 8u144 (#570) Latest commit ce15b1e 25 days ago

java-8	Updated files from 8u131 to 8u144 (#570)	25 days ago
windows-java-8	Updated files from 8u131 to 8u144 (#570)	25 days ago
README.md	Removed references to Server JRE 7u80 (#385)	5 months ago

README.md

Oracle Java on Docker

Build a Docker image containing Oracle Java (Server JRE specifically).

The Oracle Java Server JRE provides the same features as Oracle Java JDK commonly required for Server-side Applications (i.e. Java EE application servers). For more information about Server JRE, visit this release notes.

Java 8

Download Server JRE 8 `.tar.gz` file and drop it inside folder `java-8`. Build it:

github.com/oracle/docker-images/tree/master/OracleJava

jdk.java.net

GA Releases
JDK 9

Early-Access Releases
JDK 9 for Alpine Linux
JDK 8

Reference Implementations
Java SE 9
Java SE 8
Java SE 7

Feedback
Report a bug

JDK 9 Early-Access Builds for Alpine Linux

Documentation

- Supported platforms
- Installation
- Migration
- Tool & command reference
- Release Notes
- API Javadoc

Most recent build: jdk-9+181

Summary of changes

License agreement

Thank you for accepting the Early Adopter Development License Agreement. You may now download this software.

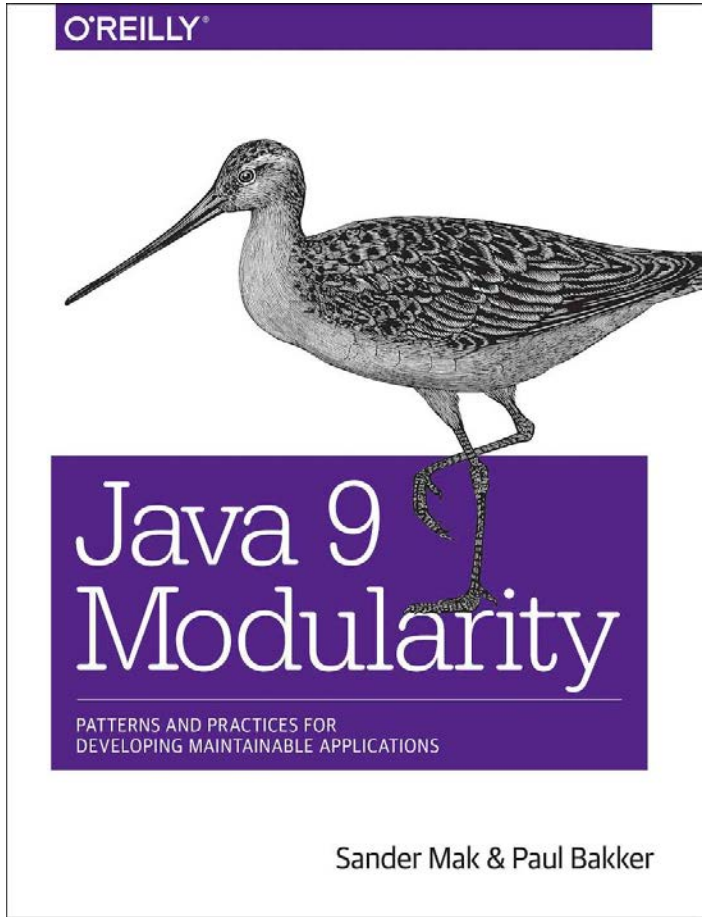
Downloads

Builds

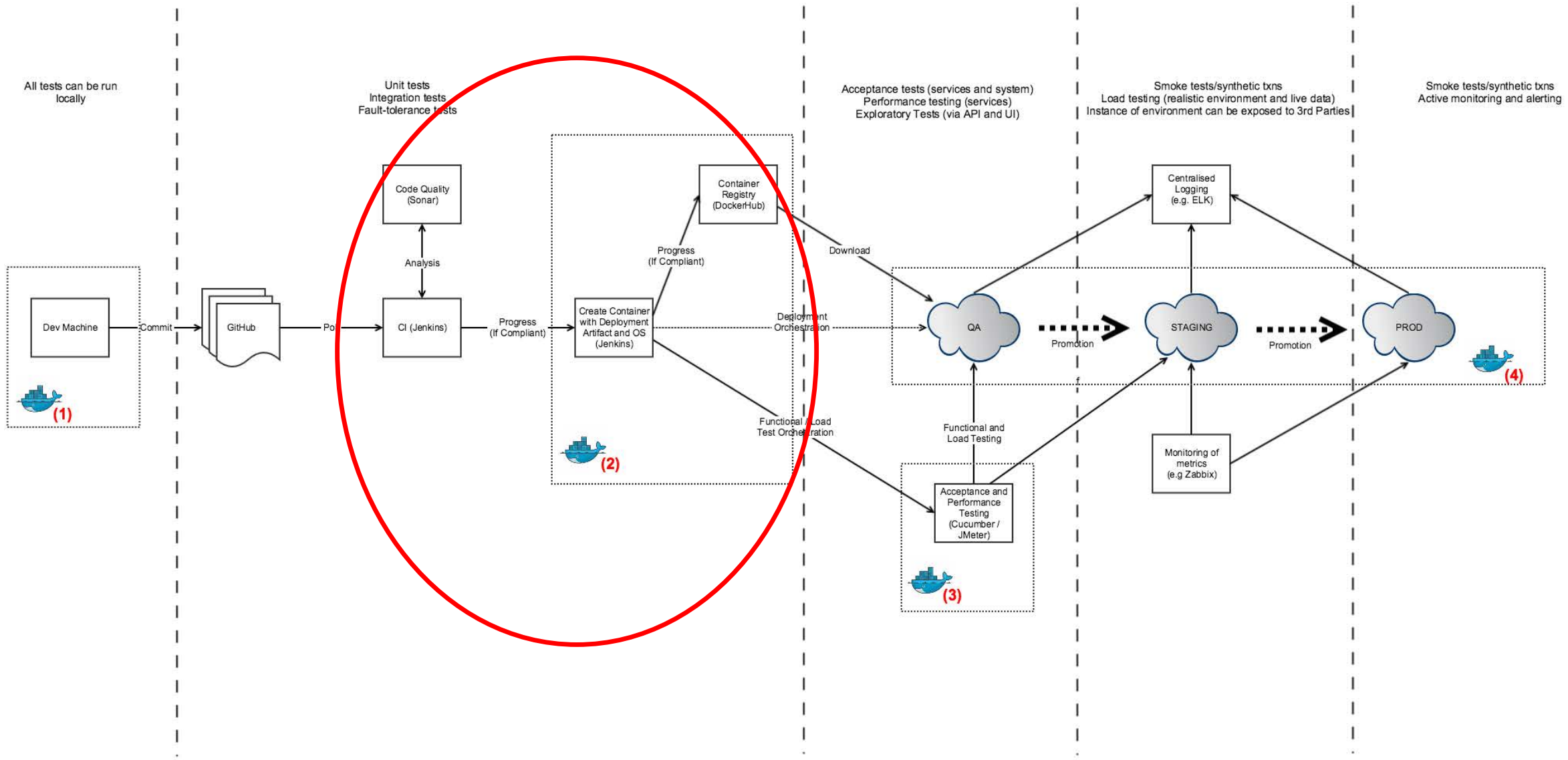
		Server JRE	JDK
Alpine Linux	**64-bit**	tar.gz (sha256) 56.8 MB	tar.gz (sha256) 201.00 MB

jdk.java.net/9/ea

Hot off the press: Modularity

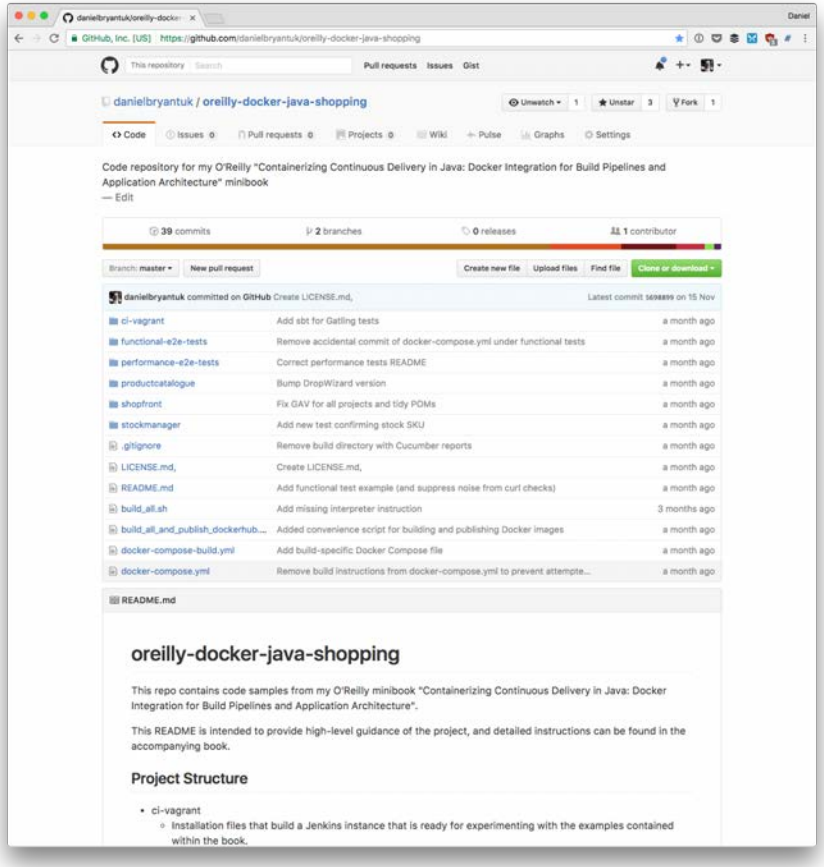


- Create minimal runtime images
- "jlink delivers a self-contained distribution of your application and the JVM, ready to be shipped."
- Benefits:
  - Reduced footprint
  - Performance
  - Security

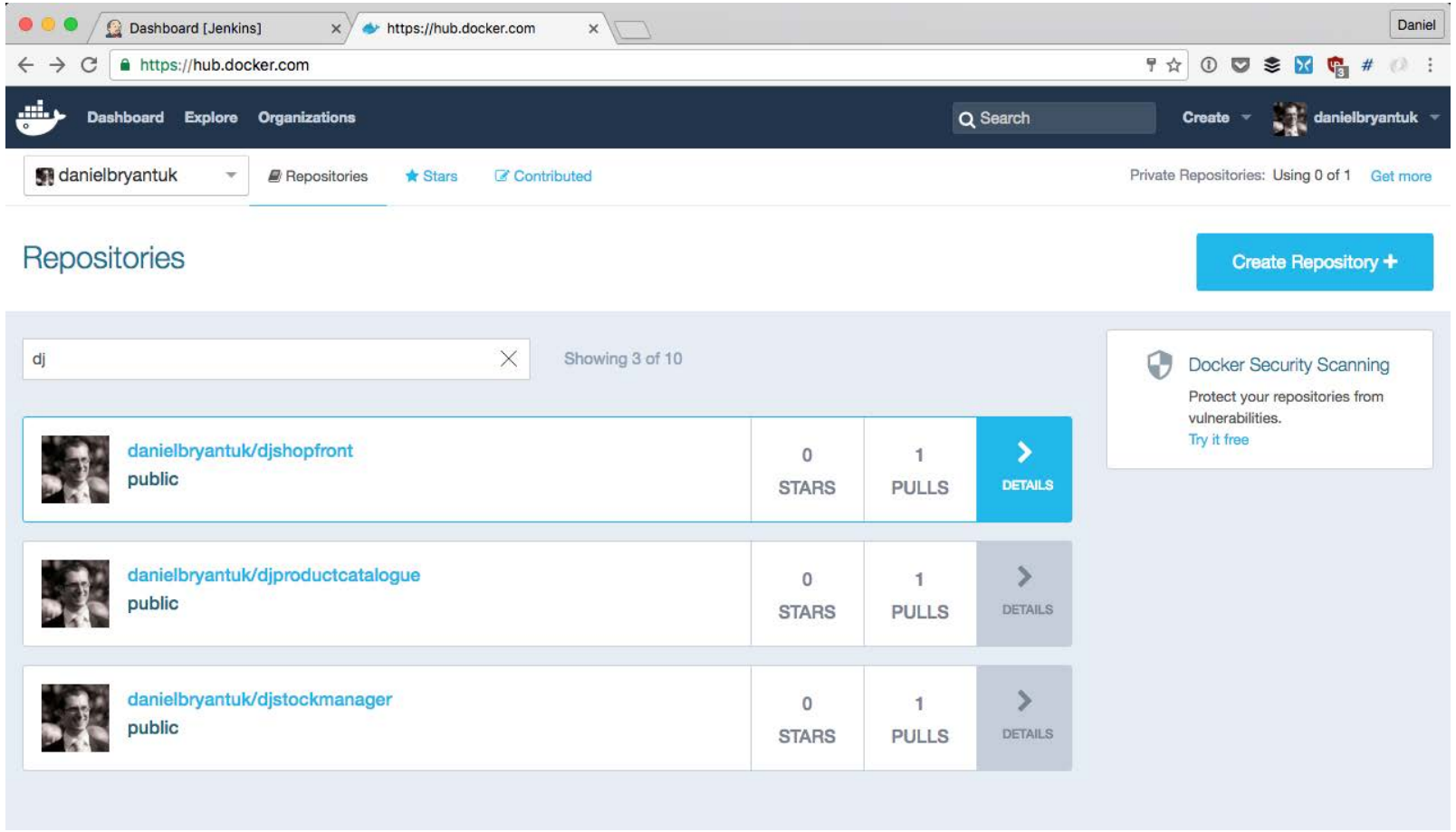

All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Progress (If Compliant)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)

Building images with Jenkins

- My report covers this
- Build as usual…
- Build Docker Image
  - [Cloudbees Docker Build and Publish Plugin]
- Push image to registry

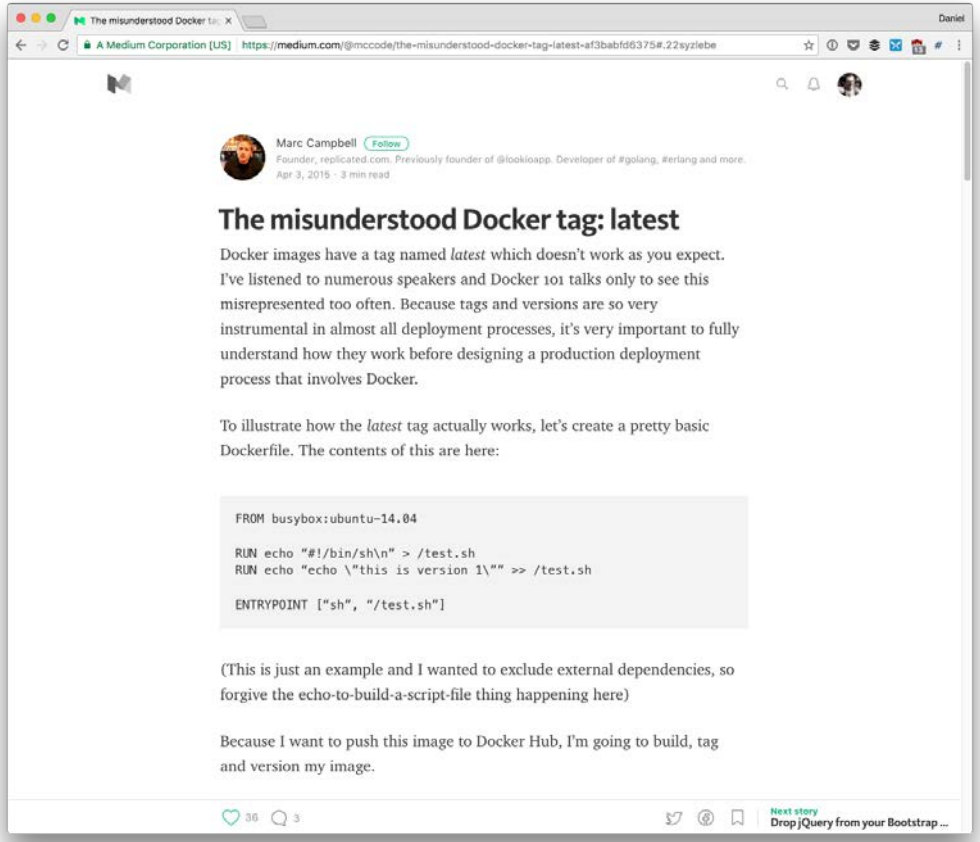


Storing in an image registry (DockerHub)



Metadata – Beware of “latest” Docker Tag

- Beware of the ‘latest’ Docker tag
- “Latest” simply means
  - the last build/tag that ran without a specific tag/version specified
- Ignore “latest” tag
  - Version your tags, every time
  - danielbryantuk/test:2.4.1



Marc Campbell Follow
Founder, replicated.com. Previously founder of @lookloapp. Developer of #golang, #erlang and more.
Apr 3, 2015 · 3 min read

The misunderstood Docker tag: latest

Docker images have a tag named *latest* which doesn’t work as you expect. I’ve listened to numerous speakers and Docker 101 talks only to see this misrepresented too often. Because tags and versions are so very instrumental in almost all deployment processes, it’s very important to fully understand how they work before designing a production deployment process that involves Docker.

To illustrate how the *latest* tag actually works, let’s create a pretty basic Dockerfile. The contents of this are here:

```
FROM busybox:ubuntu-14.04

RUN echo “#!/bin/sh\n” > /test.sh
RUN echo “echo \”this is version 1\”” >> /test.sh

ENTRYPOINT [“sh”, “/test.sh”]
```

(This is just an example and I wanted to exclude external dependencies, so forgive the echo-to-build-a-script-file thing happening here)

Because I want to push this image to Docker Hub, I’m going to build, tag and version my image.

Lesson learned: Metadata is valuable

- Application metadata
  - Version / GIT SHA
- Build metadata
  - Build date
  - Image name
  - Vendor
- Quality metadata
  - QA control, signed binaries, ephemeral support
  - Security profiles (AppArmor), Security audited etc

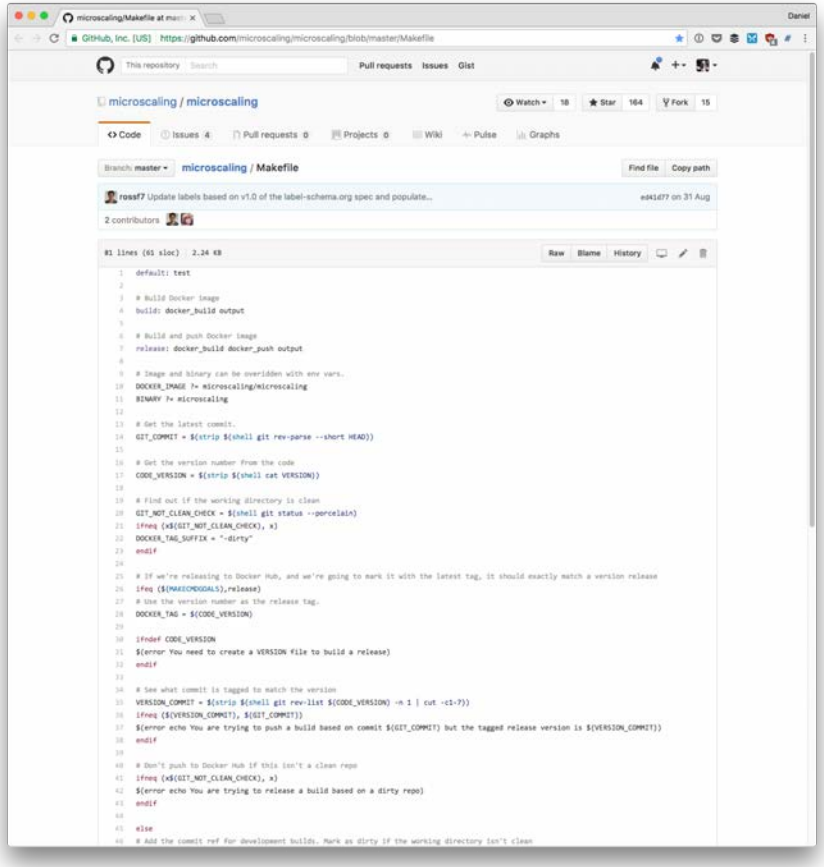
Metadata - Adding Labels at build time

- Docker Labels
- Add key/value data to image

```
FROM alpine:3.3
MAINTAINER Ross Fairbanks "ross@microscaling.com"

ENV BUILD_PACKAGES ca-certificates

RUN apk update && \
    apk upgrade && \
    apk add $BUILD_PACKAGES && \
    rm -rf /var/cache/apk/*

# Add binary and Dockerfile
COPY microscaling Dockerfile /

RUN chmod +x /microscaling

# Metadata params
ARG VERSION
ARG VCS_URL
ARG VCS_REF
ARG BUILD_DATE

# Metadata
LABEL org.label-schema.vendor="Microscaling Systems" \
      org.label-schema.url="https://microscaling.com" \
      org.label-schema.name="Microscaling Engine" \
      org.label-schema.description="Optimal resource util" \
      org.label-schema.version="1.2.3" \
      org.label-schema.vcs-url=$VCS_URL \
      org.label-schema.vcs-ref=$VCS_REF \
      org.label-schema.build-date=$BUILD_DATE \
      org.label-schema.docker.schema-version="1.0"
```

Metadata - Adding Labels at build time

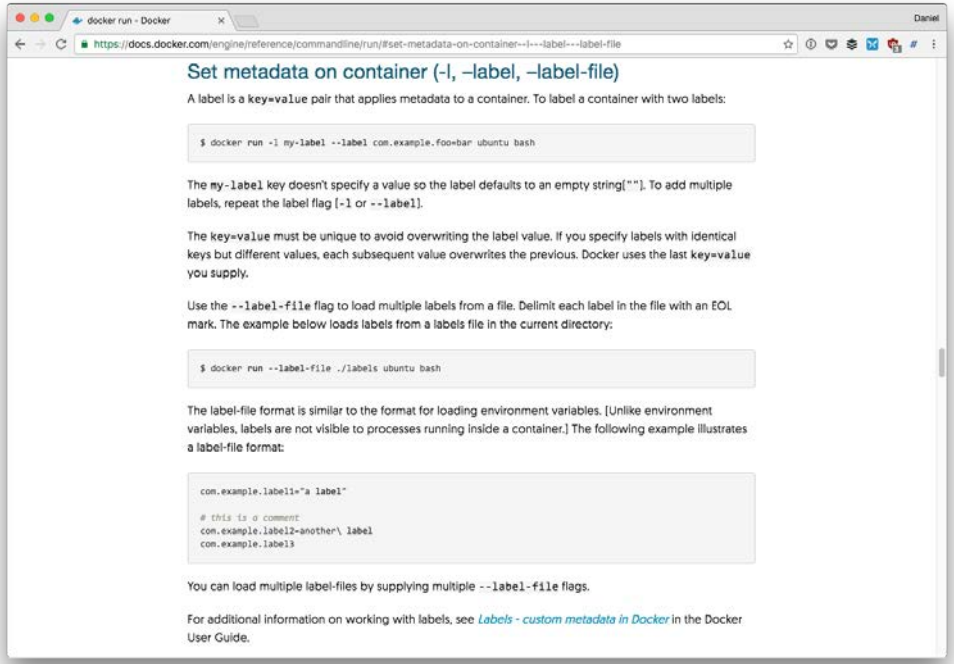
- Microscaling Systems' Makefile
- Labelling automated builds on DockerHub (h/t Ross Fairbanks)
  - Create file '/hooks/build'
- label-schema.org
- microbadger.com



Metadata - Adding Labels at runtime

```
$ docker run -d --label
uk.co.danielbryant.lbname=frontdoor nginx
```

- Can ’docker commit’, but creates new image
- Not possible to update running container
- Docker Proposal: Update labels #21721



docker run - Docker

https://docs.docker.com/engine/reference/commandline/run/#set-metadata-on-container--l---label---label-file

Set metadata on container (-l, –label, –label-file)

A label is a `key=value` pair that applies metadata to a container. To label a container with two labels:

```
$ docker run -l my-label --label com.example.foo=bar ubuntu bash
```

The `my-label` key doesn't specify a value so the label defaults to an empty string(`""`). To add multiple labels, repeat the label flag (`-l` or `--label`).

The `key=value` must be unique to avoid overwriting the label value. If you specify labels with identical keys but different values, each subsequent value overwrites the previous. Docker uses the last `key=value` you supply.

Use the `--label-file` flag to load multiple labels from a file. Delimit each label in the file with an EOL mark. The example below loads labels from a labels file in the current directory:

```
$ docker run --label-file ./labels ubuntu bash
```

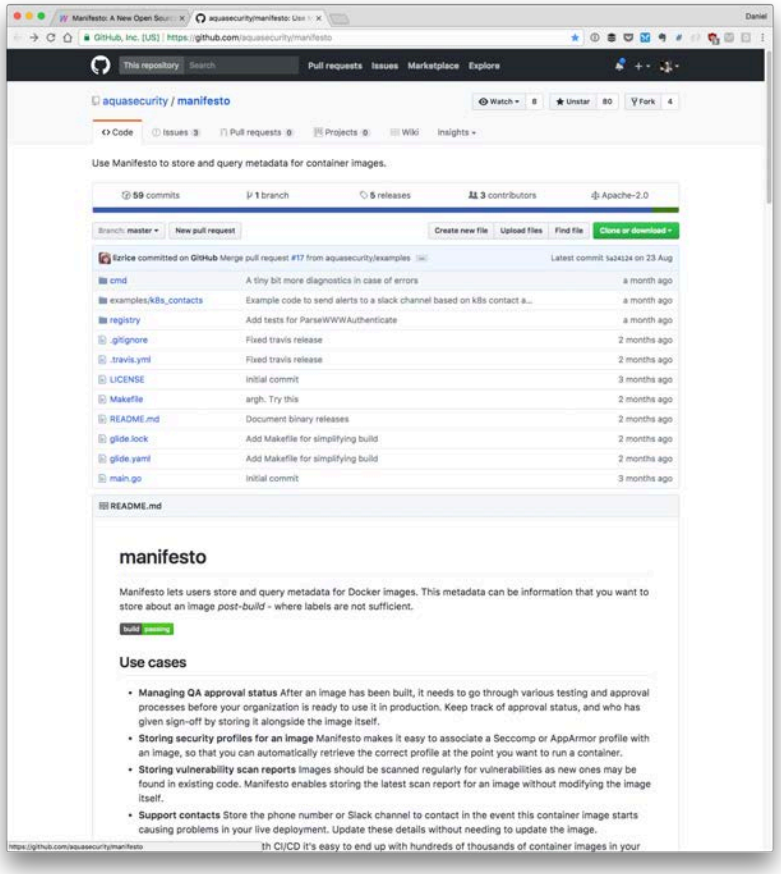
The label-file format is similar to the format for loading environment variables. [Unlike environment variables, labels are not visible to processes running inside a container.] The following example illustrates a label-file format:

```
com.example.label1="a label"

# this is a comment
com.example.label2=another\ label
com.example.label3
```

You can load multiple label-files by supplying multiple `--label-file` flags.

For additional information on working with labels, see *Labels - custom metadata in Docker* in the Docker User Guide.

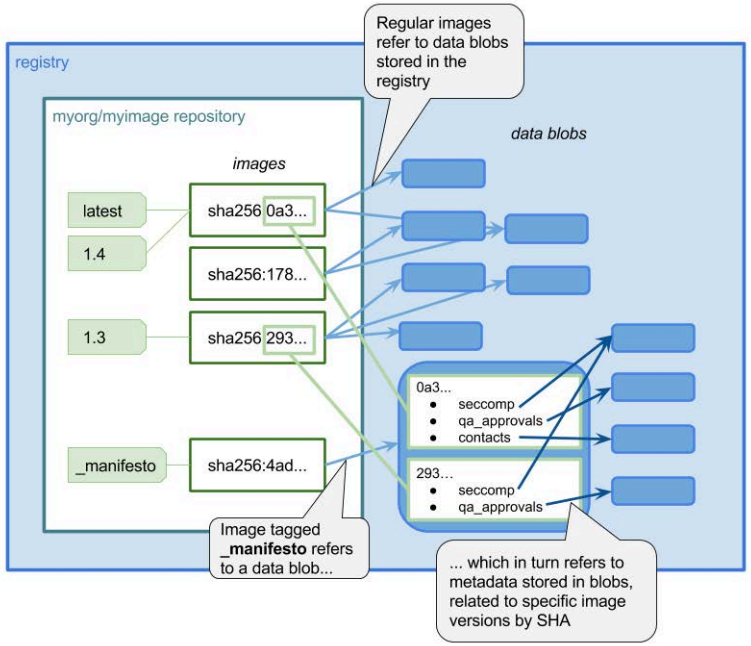
Liz Rice (and Aqua) to the rescue!



Proof of concept status

In this proof of concept:

- we store arbitrary metadata within the Docker registry
- we store a "manifesto" for the repository with the fixed tag "_manifesto"
- the manifesto is a json file with references to all the metadata stored for this repository



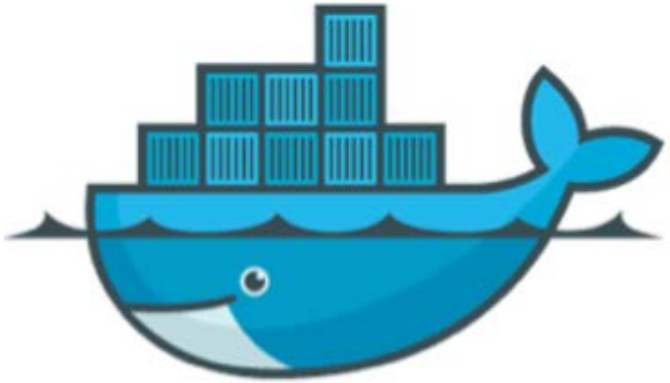
github.com/aquasecurity/manifesto

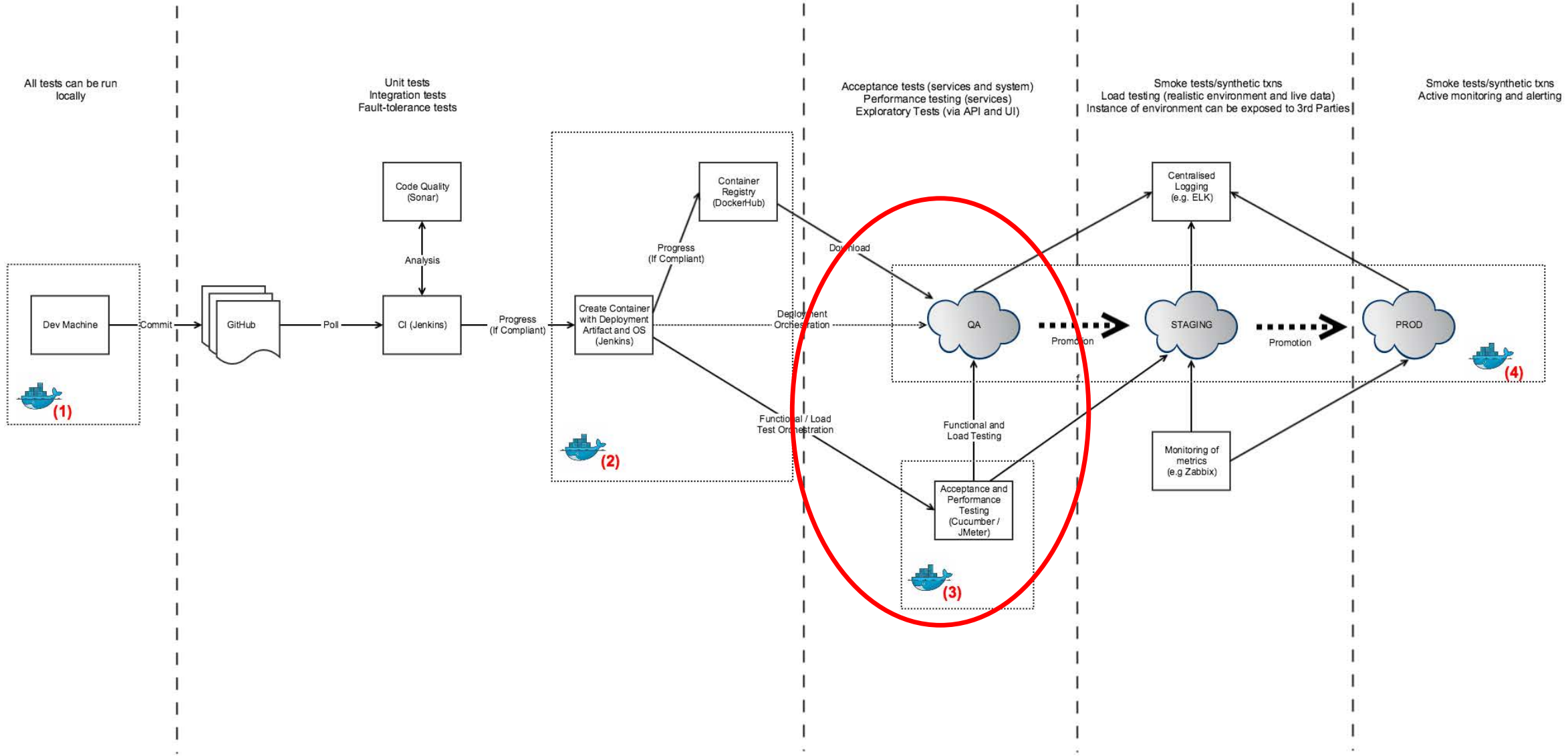
External registry with metadata support

JFrog Artifactory + Docker

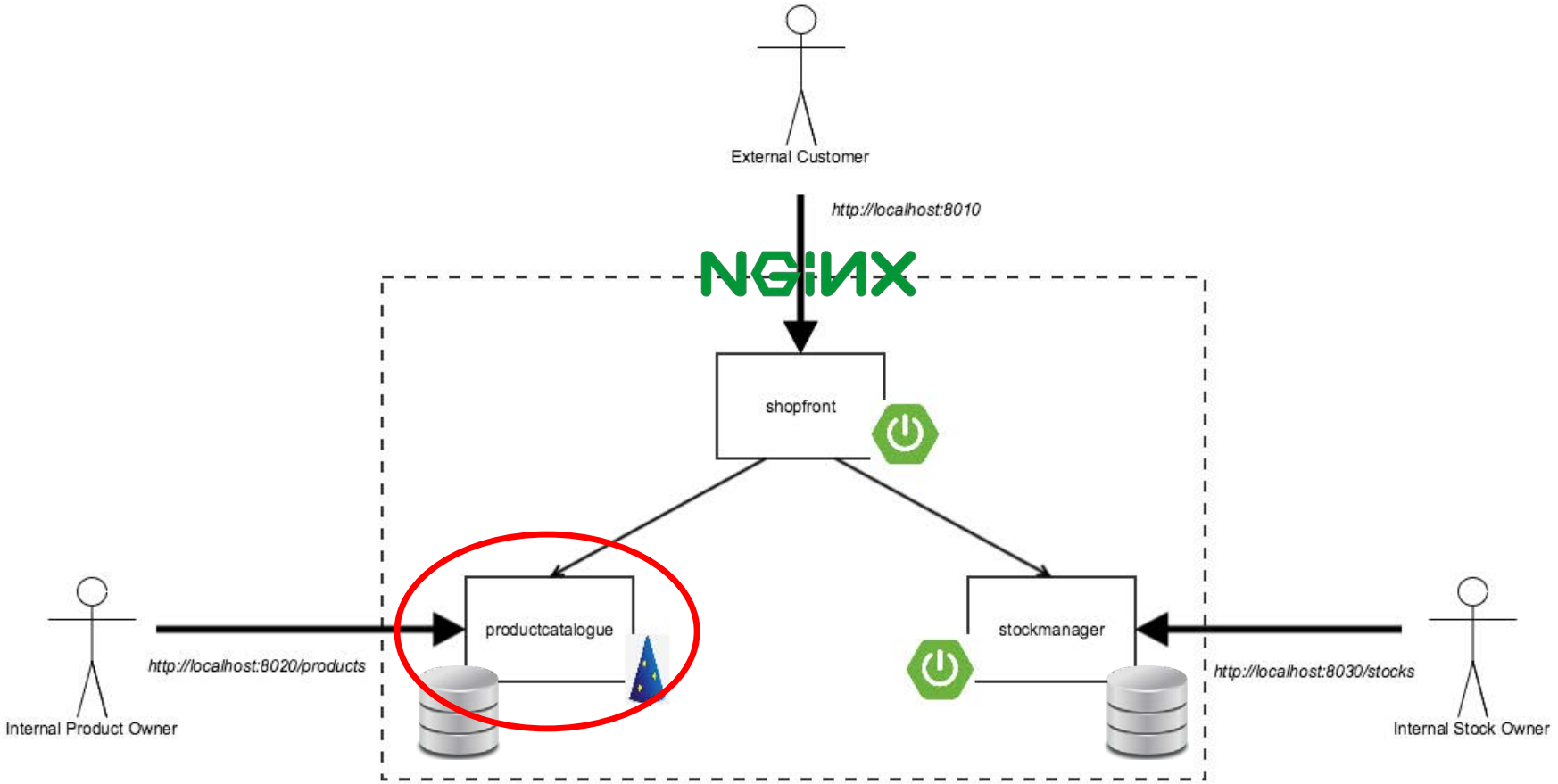
NexusOSS

your private docker registry




All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Progress (If Compliant)
Container Registry (DockerHub)
Download
Deployment Orchestration
Functional / Load Test Orchestration
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)

Component testing



Testing: Jenkins Pipeline (as code)

Jenkins ▸ e2e tests ▸

General | **Build Triggers** | Advanced Project Options | Pipeline

☐ Quiet period

☐ Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Advanced...

Pipeline

Definition: Pipeline script

Script

```
1  node {
2
3      stage ('build') {
4          git url: 'https://github.com/daniel-bryant-uk/oreilly-docker-java-shopping.git'
5          // do other build stuff
6      }
7
8      stage ('end-to-end tests') {
9          timeout(time: 60, unit: 'SECONDS') {
10             try {
11                 sh 'docker-compose up -d'
12                 waitUntil {
13                     def r = sh script: 'curl http://localhost:8010/health | grep "UP"',
14                     return (r == 0);
15                 }
```

☑ Use Groovy Sandbox

Pipeline Syntax

Save | Apply

e2e tests [Jenkins]

localhost:8080/job/e2e%20tests/

Jenkins | Try Blue Ocean UI ... | search | Daniel Bryant | log out

Jenkins › e2e tests › ENABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Move
- Full Stage View
- Pipeline Syntax

Build History — trend

find

#8	Oct 19, 2016 9:04 PM
#7	Oct 19, 2016 9:03 PM
#6	Oct 19, 2016 9:02 PM
#5	Oct 19, 2016 8:56 PM
#4	Oct 19, 2016 8:52 PM
#3	Oct 19, 2016 8:48 PM
#2	Oct 19, 2016 8:48 PM
#1	Oct 19, 2016 8:32 PM

Pipeline e2e tests

add description

Recent Changes

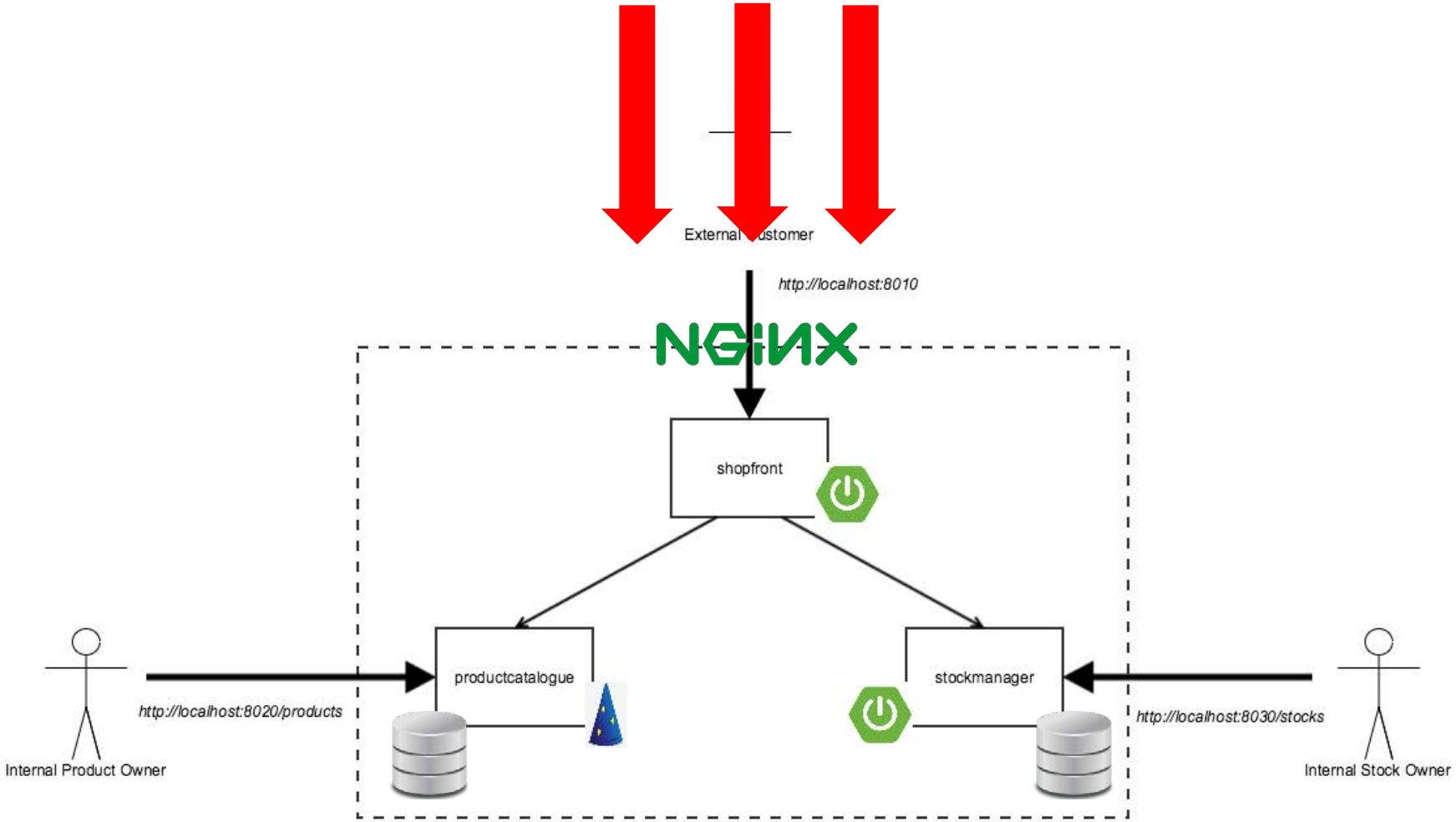
Stage View

	build	end-to-end tests	deploy
Average stage times:	35s	27s	7ms
#8 Oct 19 22:04 No Changes	905ms	18s	7ms
#7 Oct 19 22:03 No Changes	986ms	20s	7ms
#6 Oct 19 22:02 No Changes	614ms	22s	8ms
#5 Oct 19 21:56 No Changes	1s	47s	8ms

Testing individual containers

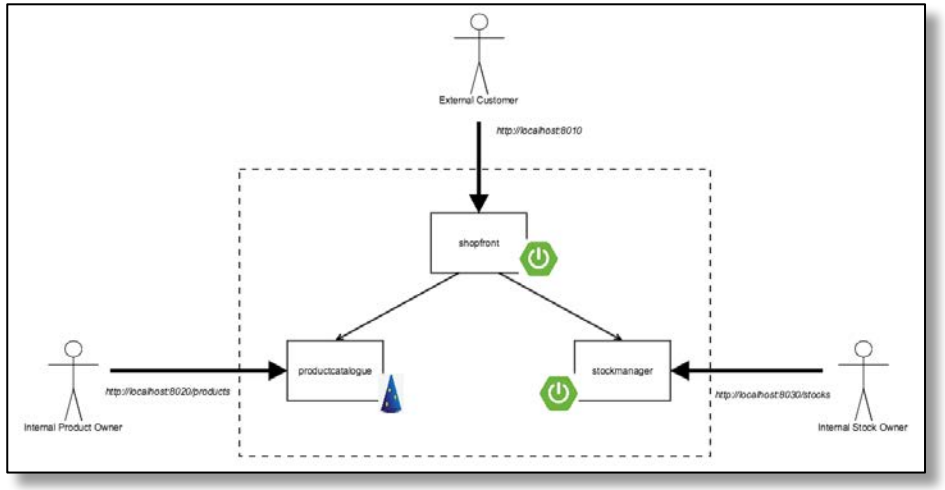
```
1   node {
2           stage ('Successful startup check') {
3               docker.image('danielbryantuk/djshopfront')
4               .withRun('-p 8010:8010') {
5                   timeout(time: 30, unit: 'SECONDS') {
6                       waitUntil {
7                           def r = sh script: 'curl http://localhost:8010/health | grep "UP"', returnStatus: true
8                           return (r == 0);
9                       }
10                  }
11              }
12          }
13  }
```

Integration testing



Introducing Docker Compose

```yaml
version: '2'
services:
  shopfront:
    build: shopfront
    image: danielbryantuk/djshopfront
    ports:
     - "8010:8010"
    links:
      - productcatalogue
      - stockmanager
  productcatalogue:
    build: productcatalogue
    image: danielbryantuk/djproductcatalogue
    ports:
     - "8020:8020"
  stockmanager:
    build: stockmanager
    image: danielbryantuk/djstockmanager
    ports:
     - "8030:8030"
```

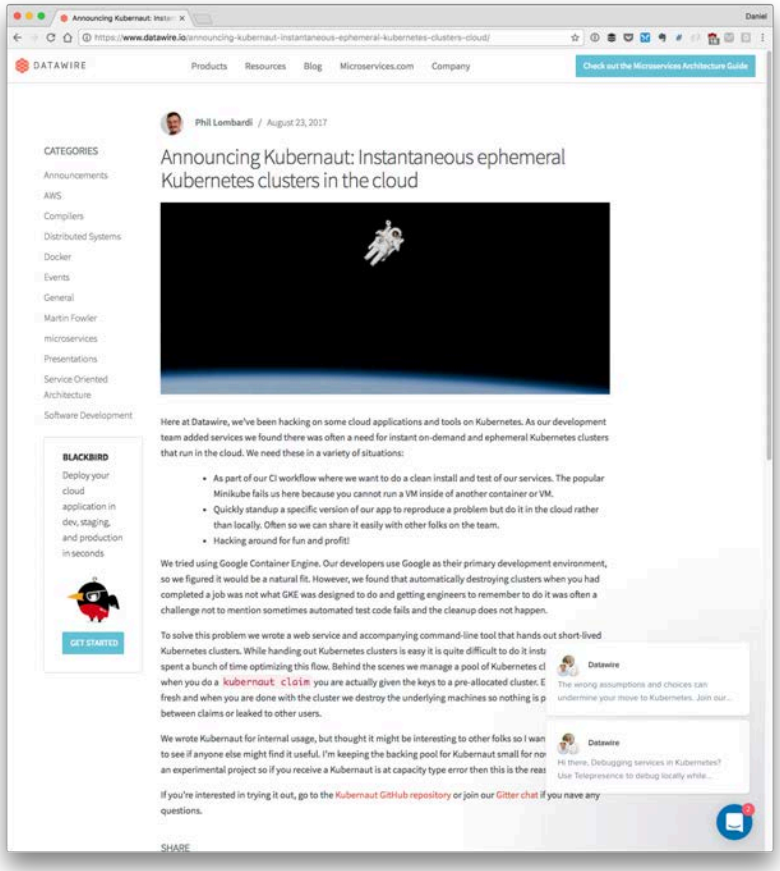


Docker Compose & Jenkins Pipeline

```
node {

    stage ('build') {
        git url: 'https://github.com/daniel-bryant-uk/oreilly-docker-java-shopping.git'
        // do other build stuff
    }

    stage ('end-to-end tests') {
        timeout(time: 60, unit: 'SECONDS') {
            try {
                sh 'docker-compose up -d'
                waitUntil {
                    def r = sh script: 'curl http://localhost:8010/health | grep "UP"', returnStatus: true
                    return (r == 0);
                }

                //other tests
                sh 'curl http://localhost:8010 | grep "Docker Java"'

            } finally {
                sh 'docker-compose stop'
            }
        }
    }


    stage ('deploy') {
        //deploy stuff
    }
}
```

Ephemeral Kubernetes Clusters

- Kubernaut (WIP)
- Manages a pool of clusters
- ”Claim” a fresh cluster
- Use Helm to install dependencies



Testing NFRs in the build pipeline

- Performance and Load testing
  - Gatling / jmeter
  - Flood.io

- Security testing
  - Findsecbugs / OWASP Dependency check
  - Bdd-security (OWASP ZAP) / Arachni
  - Gauntlt / Serverspec
  - Docker Bench for Security / CoreOS Clair

Delaying NFRs to the ‘Last Responsible Moment’

- Newsflash!
  - Sometimes the last responsible moment is **up-front**!
- Containers / microservices don’t make this easier
  - Sometimes more difficult…

Mechanical sympathy: Docker and Java

- Watch for JVM cgroup/taskset awareness
  - getAvailableProcessors() may incorrectly report the number of cpus in Docker (JDK-8140793)
  - Runtime.availableProcessors() ignores Linux taskset command (JDK-6515172)
  - *Default fork/join thread pool sizes (and others) is based from host CPU count*

- Set container memory appropriately
  - JVM requirements = Heap size (Xmx) + Metaspace + JVM overhead
  - Account for native thread requirements e.g. thread stack size (Xss)

- Entropy
  - Host entropy can soon be exhausted by crypto operations

Deployment

Menu aws https://twitter.com/danielbryantuk/lists/circle English Sign In to the Console

AWS Elastic Beanstalk
Developer Guide (API Version 2010-12-01)

Documentation - This Guide
Search

- What Is AWS Elastic Beanstalk?
- Getting Started
- How Elastic Beanstalk Works
- Elastic Beanstalk Platforms
- Tutorials and Samples
- Managing Applications
- Managing Environments
  - The Environment Management Console
  - Creating Environments
  - **Deployments**
    - Deployment Options
    - Blue/Green Deployments

Deployment Methods

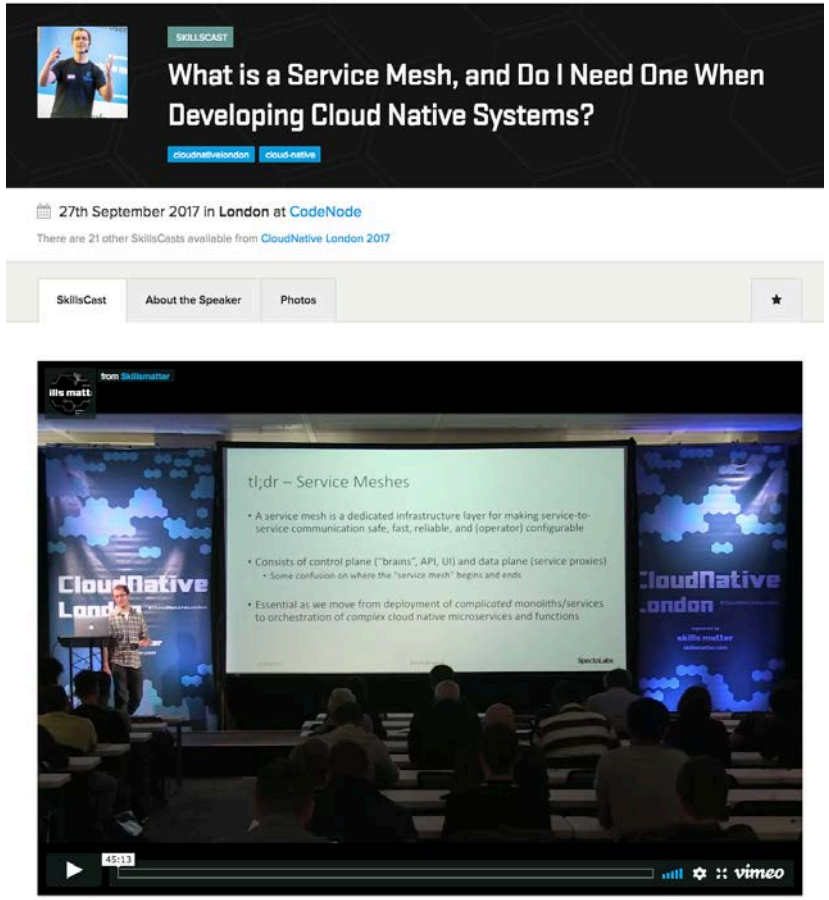
Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All at once	Downtime	◷	x	✓	Re-deploy	Existing instances
Rolling	Single batch out of service. Any successful batches prior to failure running new application version.	◷ ◷ †	✓	✓	Re-deploy	Existing instances
Rolling with additional batch	Minimal if first batch fails, otherwise similar to **Rolling**.	◷ ◷ ◷ †	✓	✓	Re-deploy	New & existing instances
Immutable	Minimal	◷ ◷ ◷ ◷	✓	✓	Re-deploy	New instances
Blue/green	Minimal	◷ ◷ ◷ ◷	✓	x	Swap URL	New instances

† *Varies depending on batch size.*

On this page:

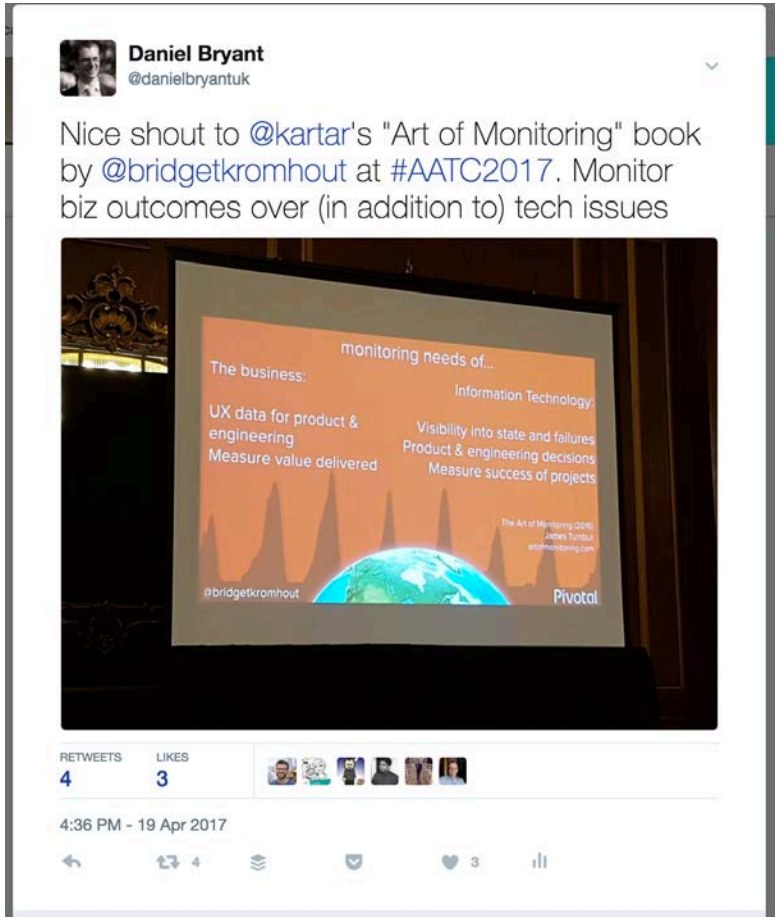
- Deploying a New Application Version
- Redeploying a Previous Version

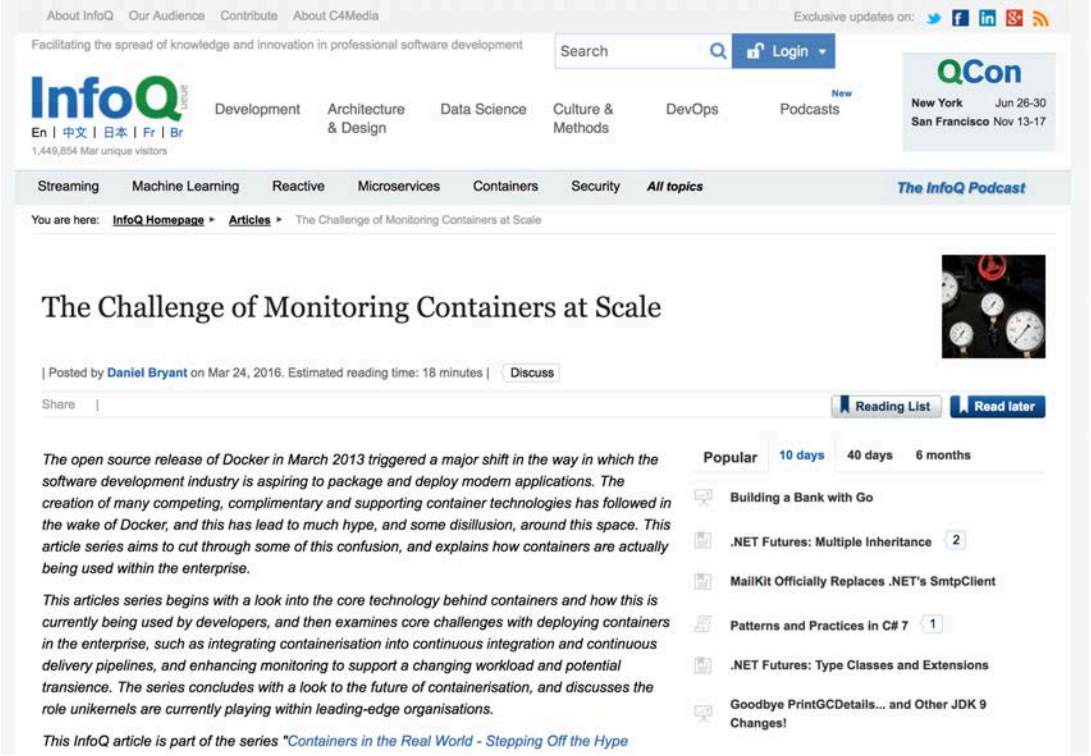
docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html



skillsmatter.com/skillscasts/10668-looking-forward-to-daniel-bryant-talk

Observability is core to continuous delivery





www.infoq.com/articles/monitoring-containers-at-scale

Containers are not a silver bullet

Moving to containers: Going all-in?

OR

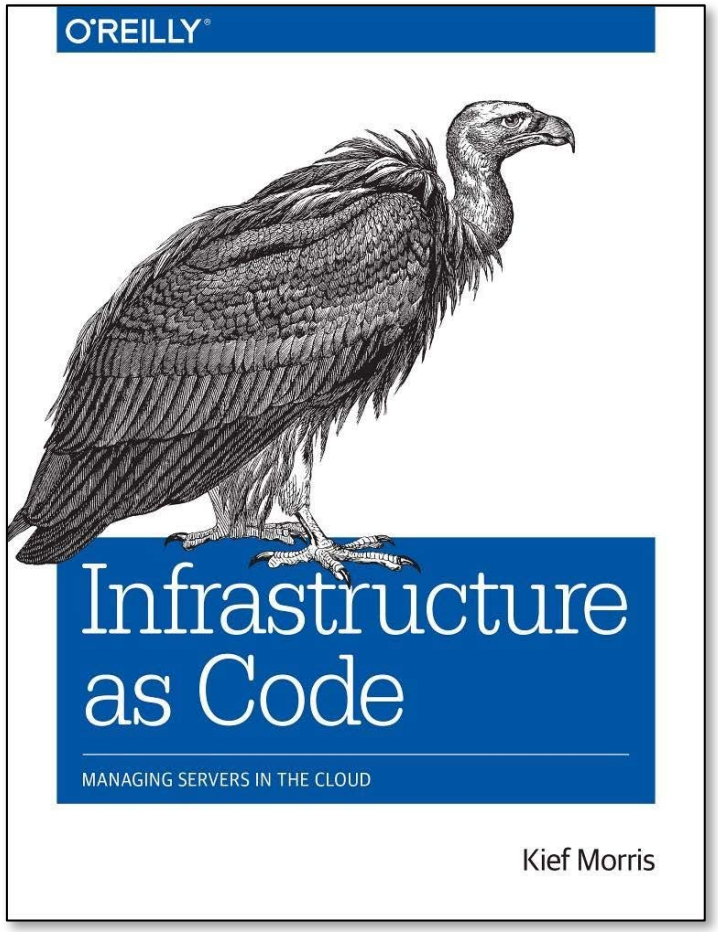
Should I build my own container platform?

Probably not

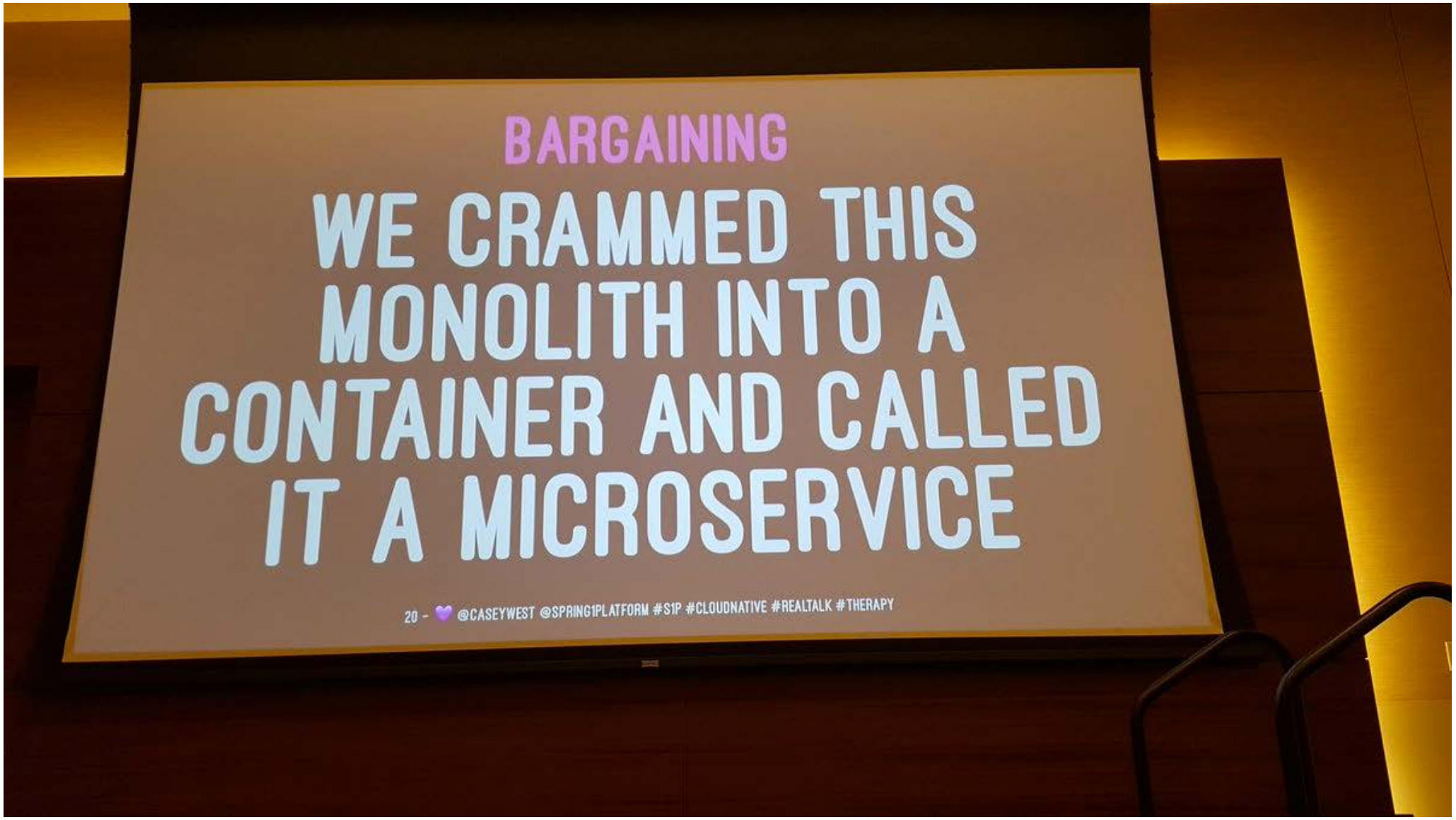
(Unless you are Google, AWS or IBM)

Whatever you decide…

push it through a pipeline ASAP!



Using containers does not obviate the need for good architectural practices



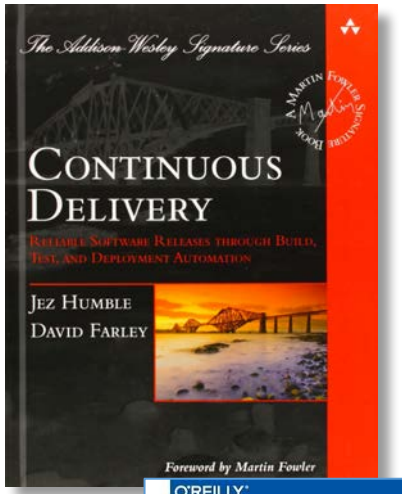
https://speakerdeck.com/caseywest/containercon-north-america-cloud-anti-patterns

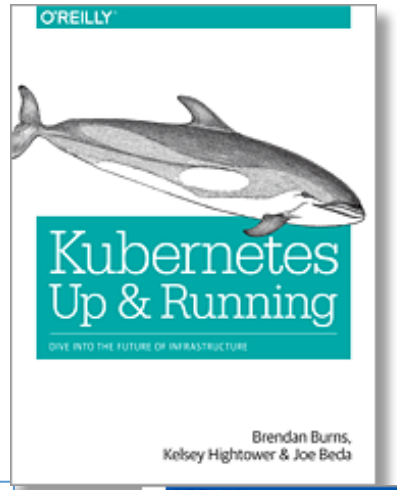
Summary

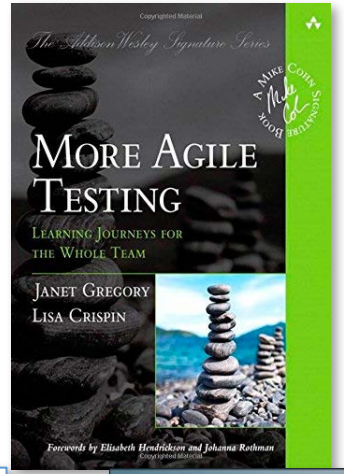
In summary

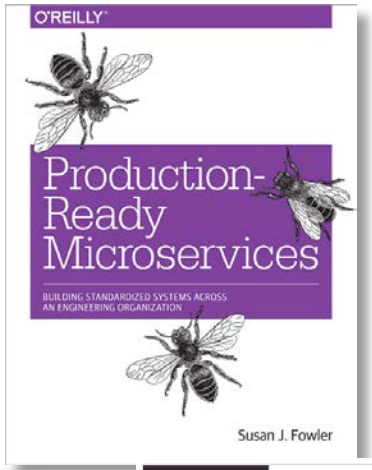
- Continuous delivery is vitally important in modern architectures/ops
- Container images must be the (single) source of truth within pipeline
  - And metadata added as appropriate…
- Mechanical sympathy is important (assert properties in the pipeline)
  - Not all developers are operationally aware
- The tooling is now becoming stable/mature
  - We need to re-apply existing CD practices with new technologies/tooling

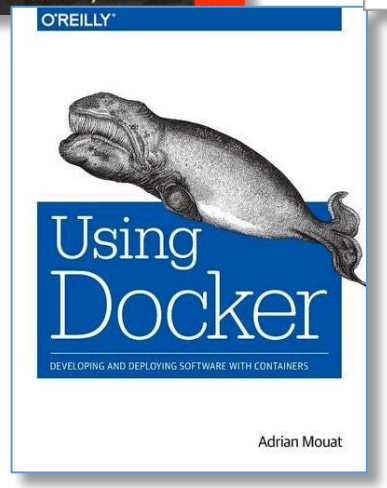
Bedtime reading

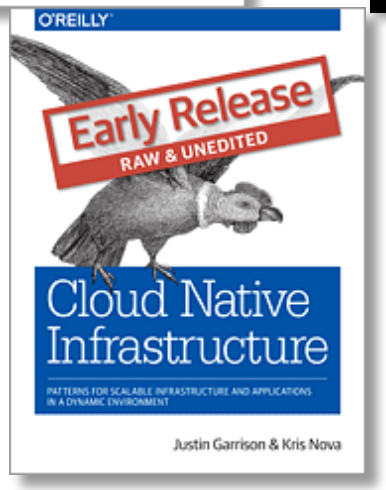


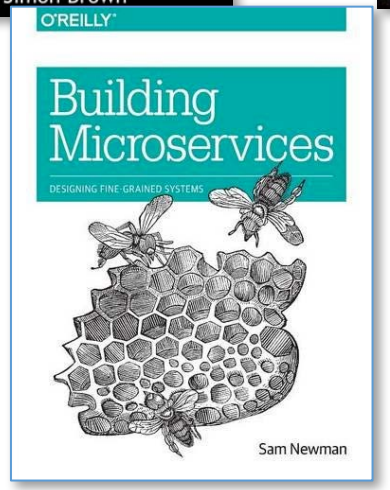


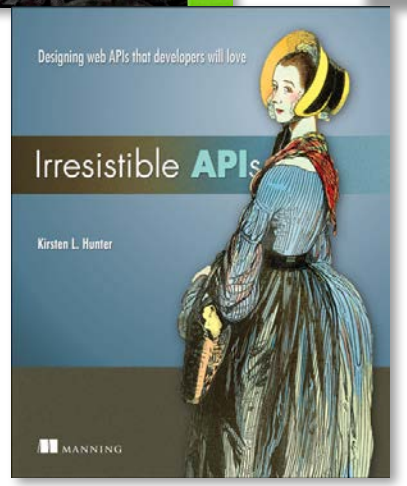


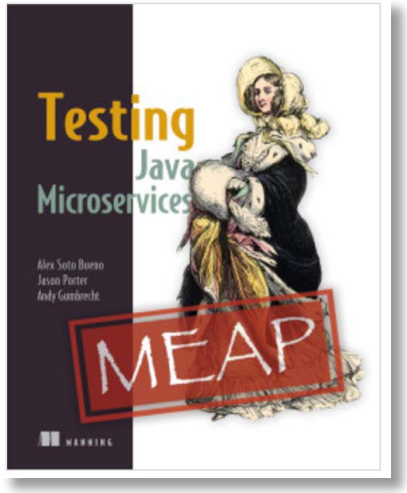








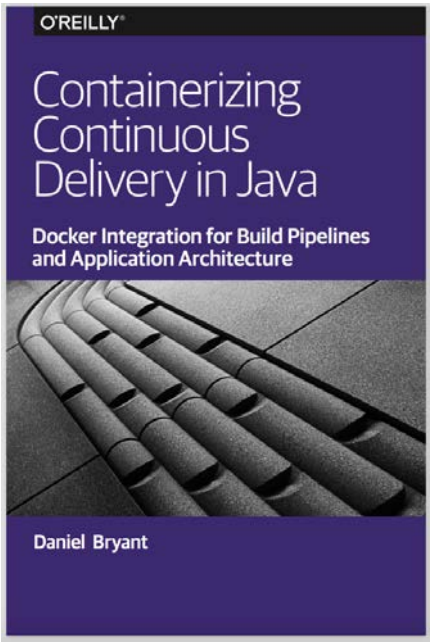




Thanks for listening

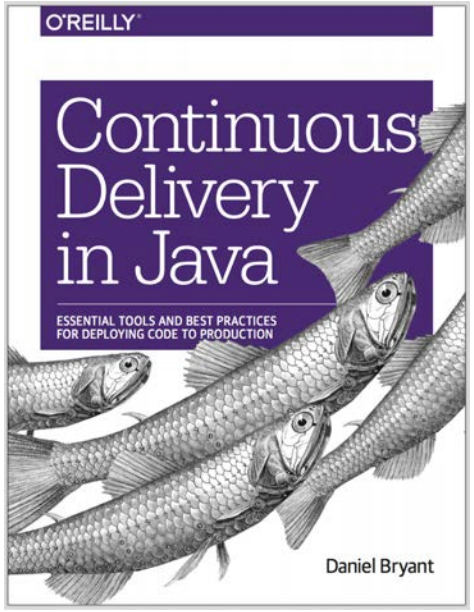
- Any questions?

- Feel free to contact me
  - @danielbryantuk
  - daniel.bryant@tai-dev.co.uk



bit.ly/2jWDSF7

Coming soon!



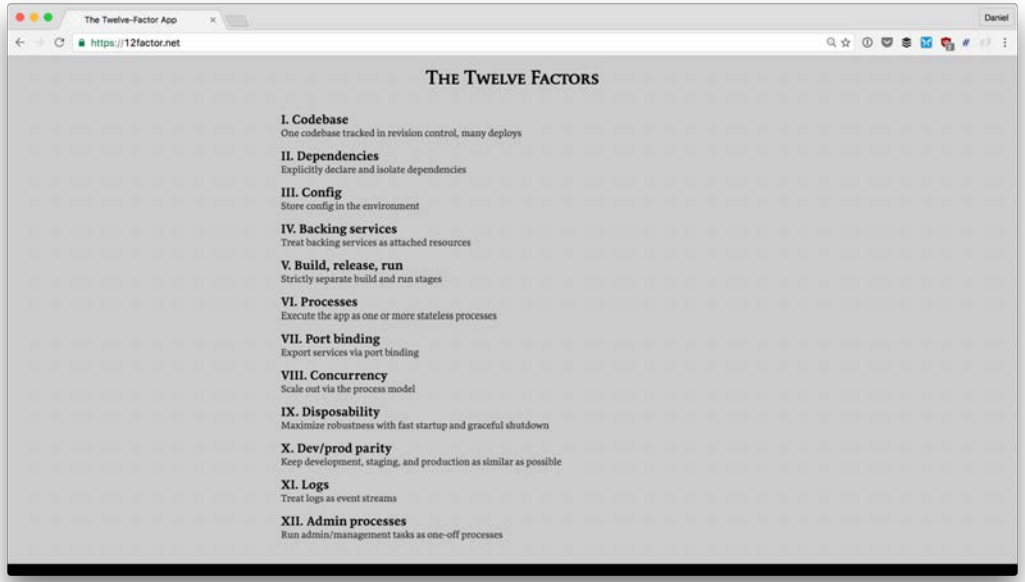
Bonus slides (for extra context)

Containerise an existing (monolithic) app?

- For
  - We know the monolith well
  - Allows homogenization of the pipeline and deployment platform
  - Can be a demonstrable win for tech and the business
- Against
  - Can be difficult (100+ line scripts)
  - Often not designed for operation within containers, nor cloud native
  - Putting lipstick on a pig?

Key lessons learned

- Conduct an architectural review
  - Architecture for Developers, by Simon Brown
  - Architecture Interview, by Susan Fowler
- Look for data ingress/egress
  - File system access
- Support resource constraints/transience
  - Optimise for quick startup and shutdown
  - Evaluate approach to concurrency
  - Store configuration (secrets) remotely



New design patterns

Design patterns for container-based distributed systems

Brendan Burns *David Oppenheimer*
Google

1 Introduction

In the late 1980s and early 1990s, object-oriented programming revolutionized software development, popularizing the approach of building of applications as collections of modular components. Today we are seeing a similar revolution in distributed system development, with the increasing popularity of microservice architectures built from containerized software components. Containers [15] [22] [1] [2] are particularly well-suited as the fundamental "object" in distributed systems by virtue of the walls they erect at the container bound-

libraries that made code more reliable and faster to develop.

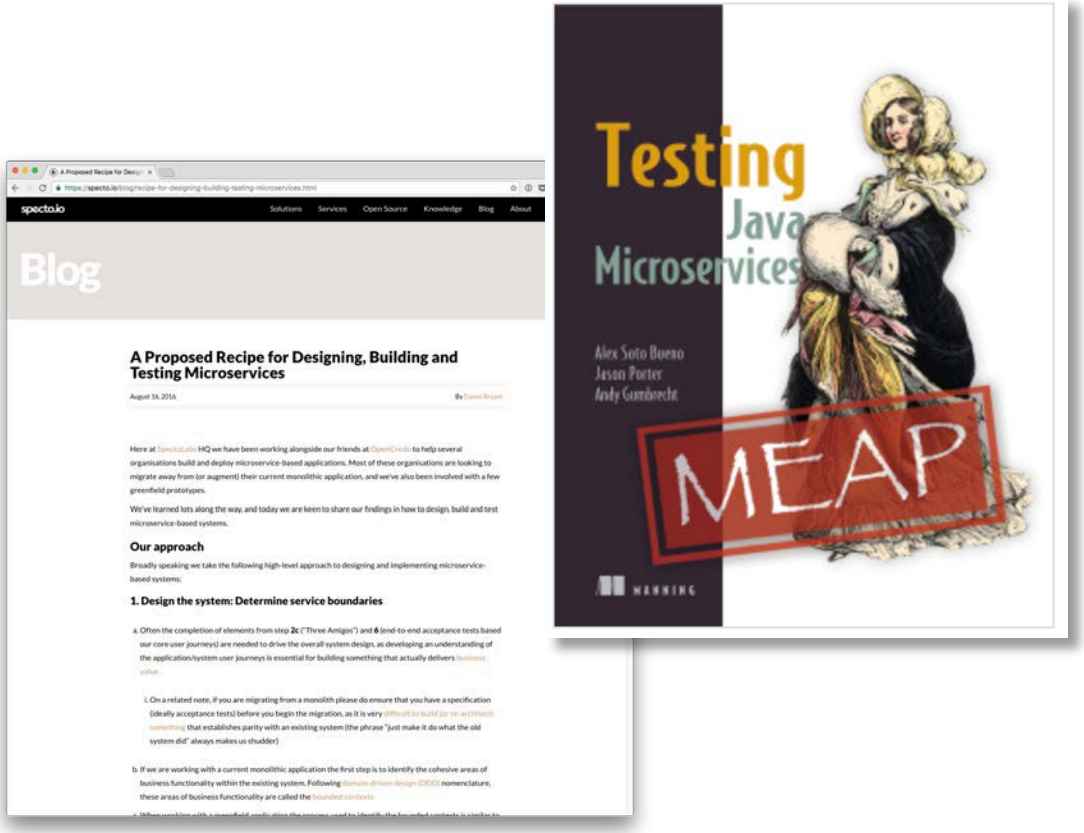
The state-of-the-art in distributed system engineering today looks significantly more like the world of early 1980s programming than it does the world of object-oriented development. Yet it's clear from the success of the MapReduce pattern [4] in bringing the power of "Big Data" programming to a broad set of fields and developers, that putting in place the right set of patterns can dramatically improve the quality, speed, and accessibility of distributed system programming. But even the

bit.ly/2efe0TP

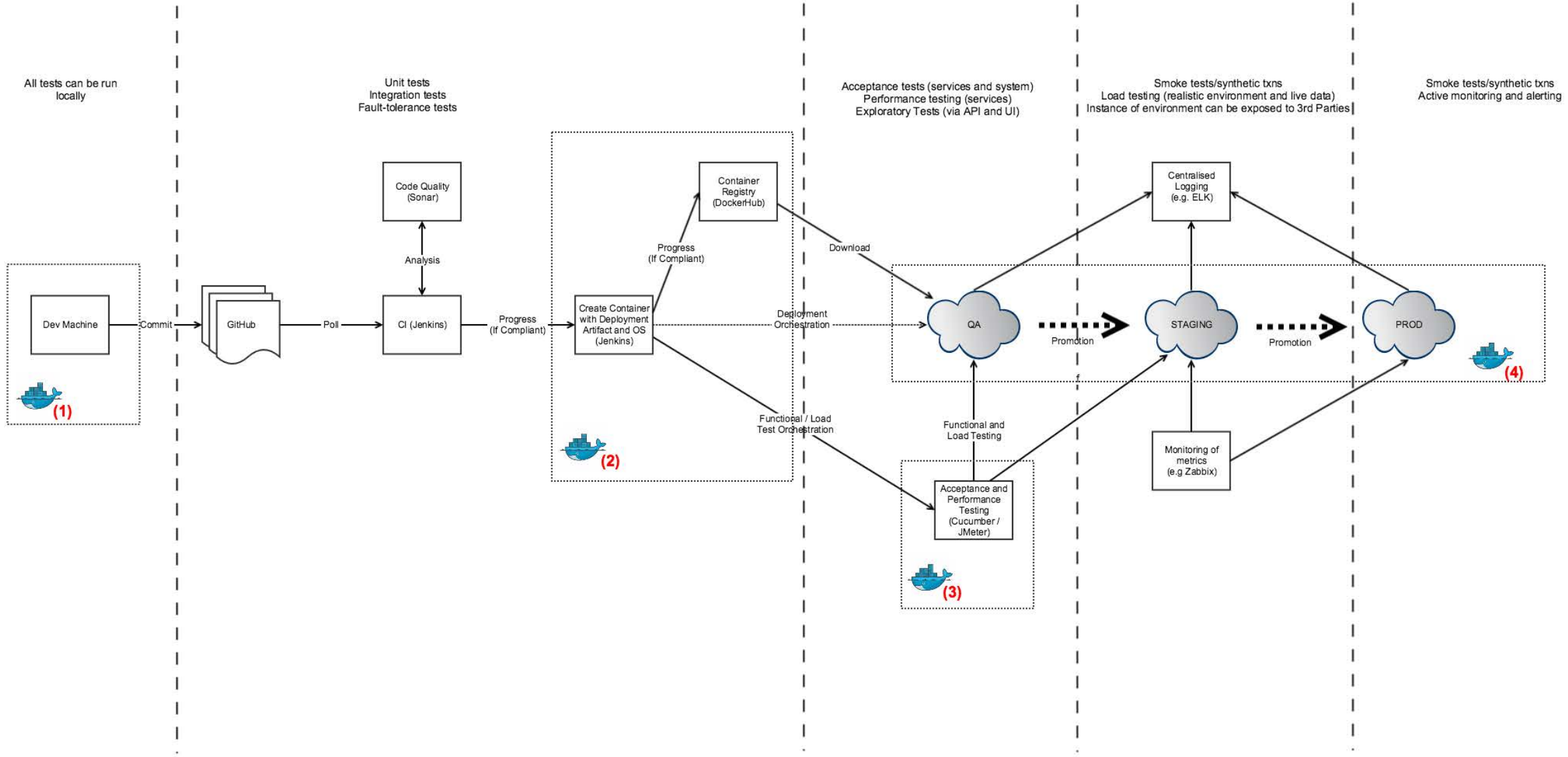
Microservices…

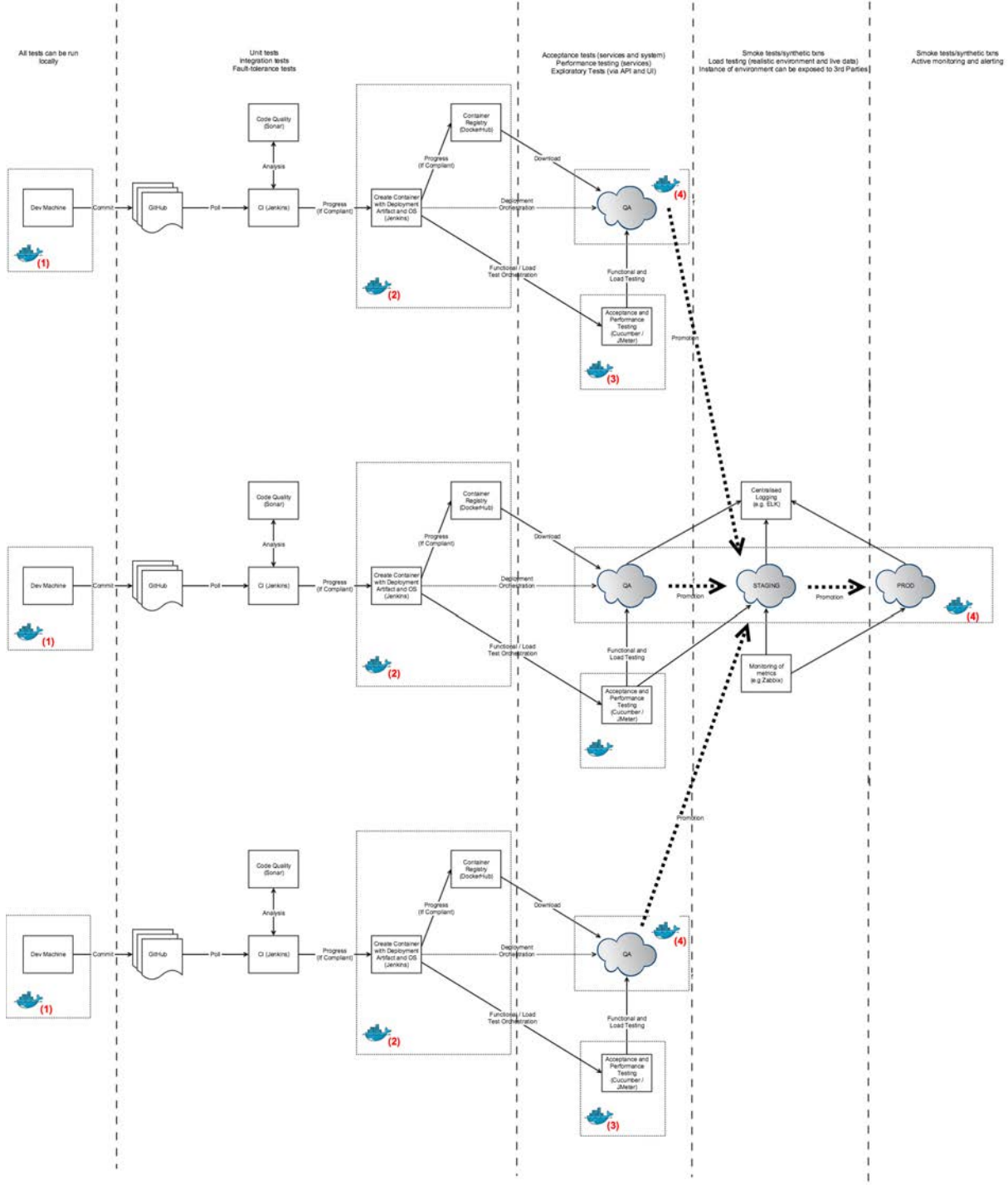
Containers and microservices are **complementary**

Testing and deployment change

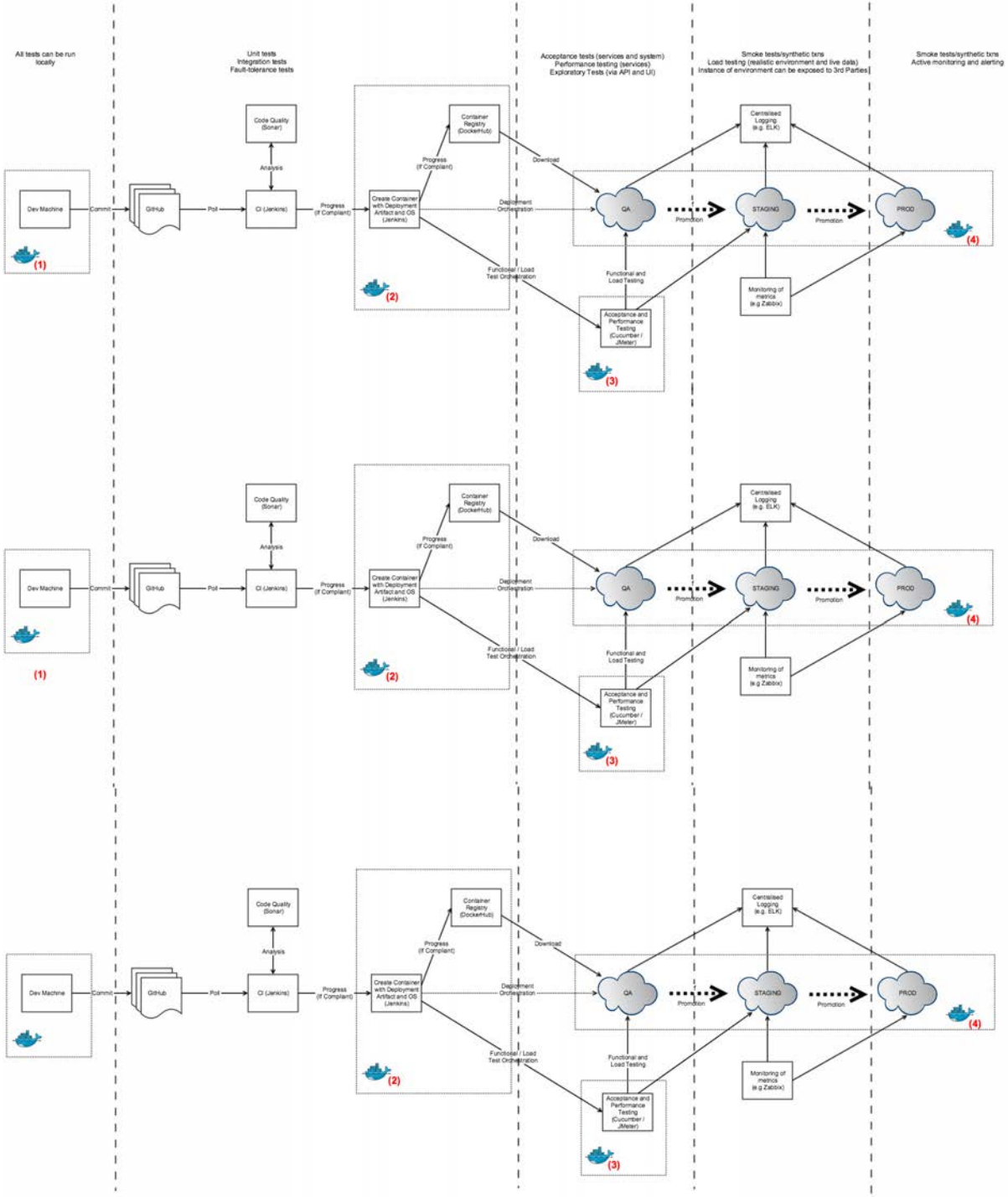


https://specto.io/blog/recipe-for-designing-building-testing-microservices.html


All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Progress (If Compliant)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Promotion
STAGING
Promotion
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)


All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
CI (Jenkins)
Code Quality (Sonar)
Analysis
Progress (If Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Promotion
Centralised Logging (e.g. ELK)
STAGING
PROD
Monitoring of metrics (e.g.Zabbix)
(1)
(2)
(3)
(4)

SpectoLabs


All tests can be run locally
Unit tests
Integration tests
Fault-tolerance tests
Acceptance tests (services and system)
Performance testing (services)
Exploratory Tests (via API and UI)
Smoke tests/synthetic txns
Load testing (realistic environment and live data)
Instance of environment can be exposed to 3rd Parties
Smoke tests/synthetic txns
Active monitoring and alerting
Dev Machine
Commit
GitHub
Poll
Code Quality (Sonar)
Analysis
CI (Jenkins)
Progress (if Compliant)
Create Container with Deployment Artifact and OS (Jenkins)
Container Registry (DockerHub)
Deployment Orchestration
Functional / Load Test Orchestration
Download
QA
Promotion
STAGING
PROD
Functional and Load Testing
Acceptance and Performance Testing (Cucumber / JMeter)
Centralised Logging (e.g. ELK)
Monitoring of metrics (e.g Zabbix)
(1)
(2)
(3)
(4)

SpectoLabs