AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Continuous Delivery/Deployment

Henrik Bærbak Christensen

AARHUS UNIVERSITET

An opinion…

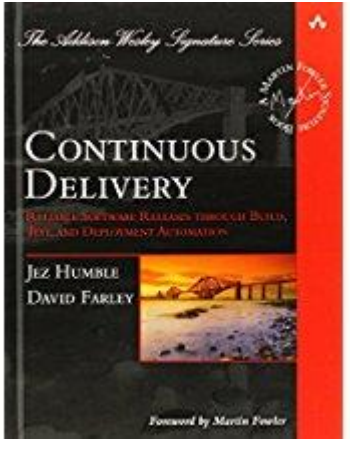
Agile Manifesto

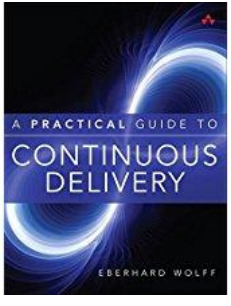
- Highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Hence *'continuous delivery'* was coined as term…

The Sources

- Sam Newman
  - (2nd Edition Sep 2018)
- Michael T Nygard
- Jez Humble et al.
- Eberhard Wolff





AARHUS UNIVERSITET

Why Continuous Delivery

Rephrased Talk by *Jez Humble*

Source

- This is a slidification of *Jez Humble / Continuous Delivery* talk in 2012…
  - https://www.youtube.com/watch?v=skLJuksCRTw

AARHUS UNIVERSITET

Deliver Frequently

- Deliver frequently because it ensures:
- 1. Build the right thing
- 2. Reduce risk of release
- 3. Real project progress

AARHUS UNIVERSITET

1: Built the Right Thing

AARHUS UNIVERSITET

Wasted Time

- More than 50% of functionality in SW is rarely or never used!

AARHUS UNIVERSITET

Build the Right Thing

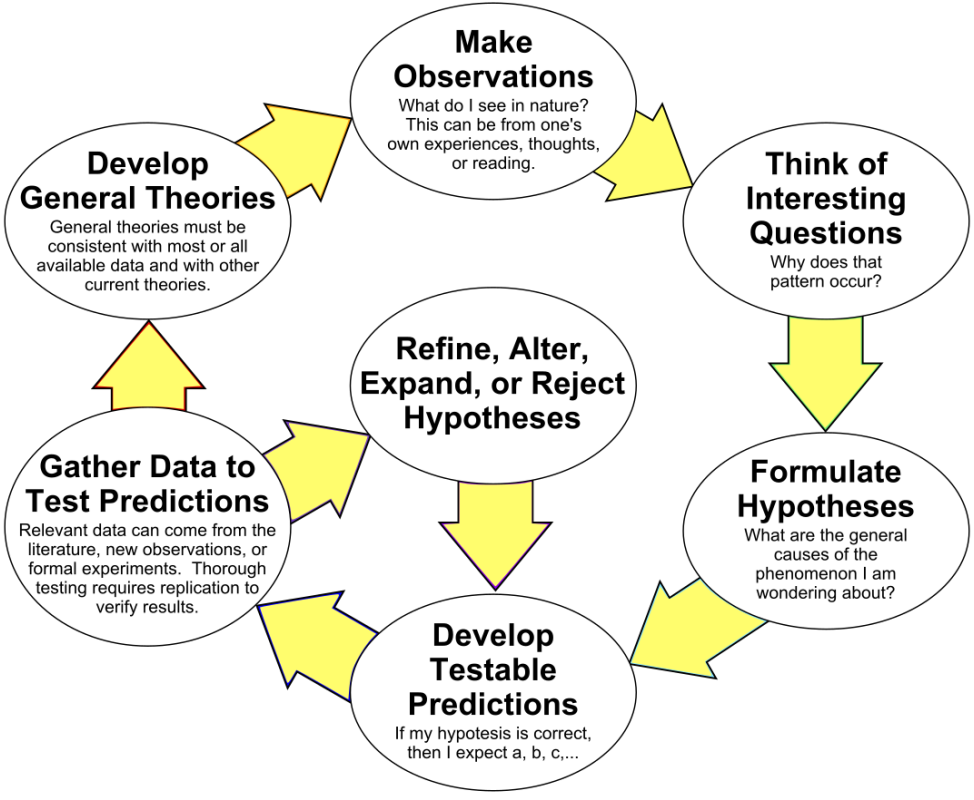
- Steve Jobs: *You can't ask customers what they want. By the time you deliver, they want something new…*
  - Henry Ford: They would have asked for faster horses…

- Problem solved: Scientific method
  - Create hypothesis — *We make this cool feature*
  - Deliver MVP — **Minimum Viable Product**
  - Get feedback — Reject/Approve — *was it cool?*
  - (repeat)
- *Morale: Do not ask, run experiments and measure*

AARHUS UNIVERSITET

Science

The Scientific Method as an Ongoing Process



AARHUS UNIVERSITET

Lead Time

- Optimize our software delivery process
  - **Lead Time**: Time from
    - ‘one single line edited in code’ (or cool feature made)
    - to
    - ‘running in production, delivering value to customers’

- Question:
  - Do you know the lead time in your organization?
  - In case you do, what is it?

AARHUS UNIVERSITET

2: Reduce Risk of Release

‘Release’ as in ‘Users use it’

AARHUS UNIVERSITET

Release Frequently

- Large release every 3-6 months
  - Lots of stuff means **lots of stuff can go wrong**
    - Akin to ‘Big Bang Integration’

- Small release every 4-8 hours
  - Small amount of stuff means **easy to trace root cause**
  - And you practice releasing – you get good at it

AARHUS UNIVERSITET

Monitoring

- Does something go wrong?
  - ‘Crash and burn’ are obvious, but…
    - Throughput lowered by 0.8 % by defect in code
    - Customer’s actually buy something lowered by 8%
    - Service x crashes 50% more often

- Monitoring production systems is crucial
  - (Area that I have little practical large-scale experience with ☹)
  - Example: Nagios, ELK, Humio, …

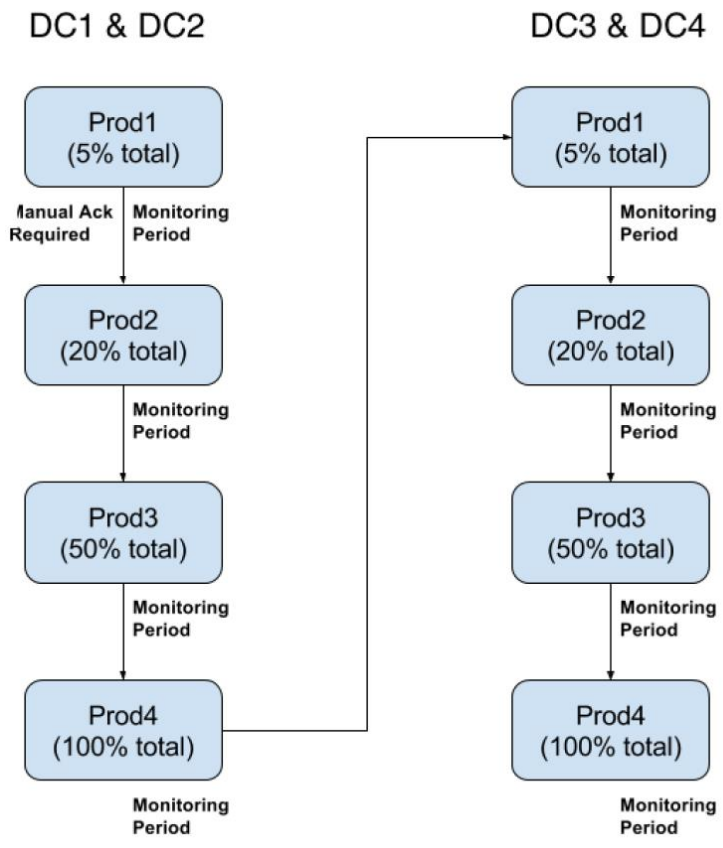
AARHUS UNIVERSITET

Availability Measure

- Mean Time Between Failure MTBF
  - The BMW
    - Never fails (but really expensive when it does)
- Mean Time to Restore Service MTRS
  - The Jeep
    - Fails often (but really fast and easy to repair)
- Continuous Delivery strives to improve MTRS
  - (Do not use it for space aircraft software ☺)

Example: Uber

- uDeploy and uOrchestrate



AARHUS UNIVERSITET

3: Real Project Progress

AARHUS UNIVERSITET

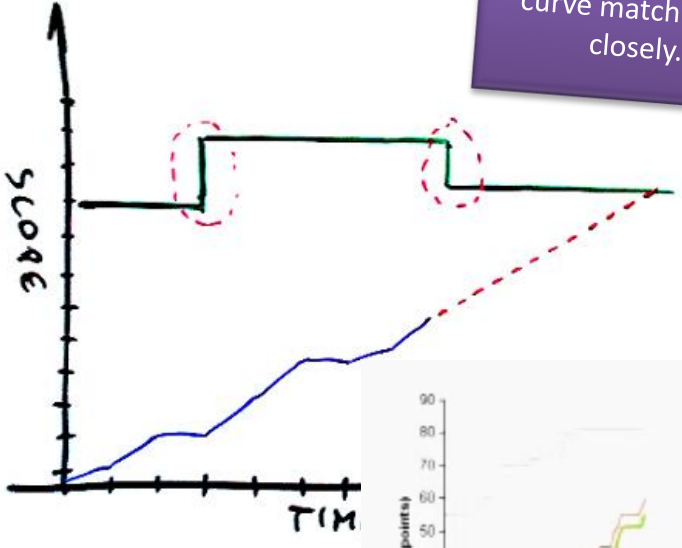
Windows Progress Metaphor

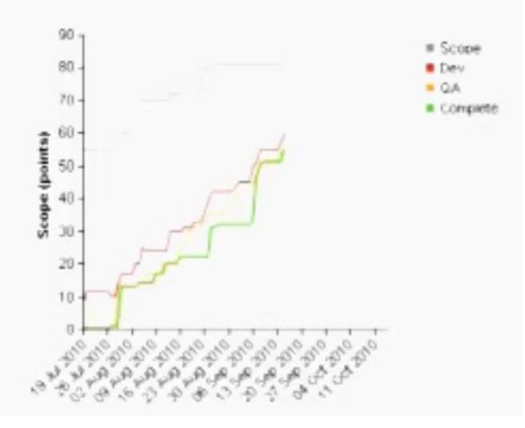
- The good old Windows progress bar
  - Spent 8 seconds to reach 95%
  - Spent 15 minutes to reach the last 5%
- So what does it mean to a project manager that developers state that the software is 90% finished?
  - If it is just *development* that is 90% done
    - Never tested with users, never tested with real sized database contents, never tested with 10.000 concurrent users, never tested for …
- Humble: “Done” versus “Done done”
- Nygard: “*feature complete* versus *production-ready*”

Real Project Progress

- Burn down chart
  - http://brodzinski.com/2012/10/burn-up-better-burn-down.html
- Time graph
  - Scope on Y
    - Will evolve up (or down)
  - Time on X
  - Plot ‘features in production’
    - Blue line = **real live features**
    - Dotted red = projection

Note: Using MVP the scope and progress curve match more closely.





AARHUS UNIVERSITET

Summary

- Do it, in order to
- **1. Build the right thing**
  - Fast feedback from users, focus attention to aspects that deliver value to them
- **2. Reduce risk of release**
  - Release often means you get very good at it
  - Release often means broken release is easier to roll back, and easier to find the defect
- **3. Show real project progress**
  - “Done” is not “Done-Done”
    - Feature made = Done. Feature used = Done-Done

Terminology

I find it muddles a bit…

AARHUS UNIVERSITET

Terminology

- Writing these slides, I found that I actually use terms like
  - Deploy, Release, Delivery, Production

- … in a rather unprecise way ☹
  - *I deploy SkyCave? I release SkyCave? Put SkyCave in prod.?*
- *… (and authors seems to mix it up a bit as well)*

- So – What do you mean by
  - Release x? Deploy x? Deliver x?

AARHUS UNIVERSITET

Humble: CD

Chapter 10. Deploying and Releasing Applications

Introduction

There are differences between releasing software into production and deploying it to testing environments—not least, in the level of adrenaline in the blood of the person performing the release. However, in technical terms, these differences should be encapsulated in a set of configuration files. When deployment to production occurs, the same process should be followed as for any other deployment. Fire up your automated deployment system, give it the version of your software to deploy and the name of the target environment, and hit go. This same process should also be used for all subsequent deployments and releases.

Deploy

b : to place in battle formation or appropriate positions
// *deploying* troops to the region

Release

4 : to give permission for publication, performance, exhibition, or sale of

also : to make available to the public
// the commission *released* its findings

// *release* a new movie

Source: Merriam Webster online dictionary

So…

- Deploy service
  - Bring it in an executable form into a given environment
    - Which may be a testing env or a production env
  - (normally also means ‘and start its execution’)

- Release service
  - Make its features available to the ‘public’ (customers)
    - Which (almost) equate ‘deploy and start it on production env’
      - **Deliver ?**
    - (unless the feature is ‘toggled off by a feature toggle’)

https://www.martinfowler.com/articles/feature-toggles.html

AARHUS UNIVERSITET

Release is not Release

- Perhaps because I am an old-school guy…
- In *the old days* release meant
  - Tag a release tag onto a specific version and call that ‘the one to use’
  - Ala ‘upgrade your build.gradle to include frds.broker, v 1.14’
- Different from *this* notion of release…
  - “Push release” versus “Pull release” ?

AARHUS UNIVERSITET

Newman Part II

- (Newman: “Monolith to MicroService”, p 80)
  - Deployment does not mean accessible by customers (!)
    - I.e. not released; may run with no user load in production
  - Released does not mean deployed in production (!)
    - i.e. we have only released it in staging, for testing
- Martin Schmidt’s Master’s Thesis, p 22

> **Continuous Deployment**: Every commit to main branch results in a new release.
> **Continuous Delivery**: Ready and able to deploy any version to any supported platform at any time.

AARHUS UNIVERSITET

The Pipeline Again

Just adding stages…

Pipeline

- *Deployment Pipeline:* An automated implementation of your application's build, test, *deploy* and *release* process.

Figure 1.1 The deployment pipeline

- Every change that is made in any artefacts of the application *triggers* the creation of a new instance of the pipeline.
  - Series of tests, each more demanding, each giving confidence in that *it will work*. Passing all tests means ready for release.

AARHUS UNIVERSITET

Production-Ready Software

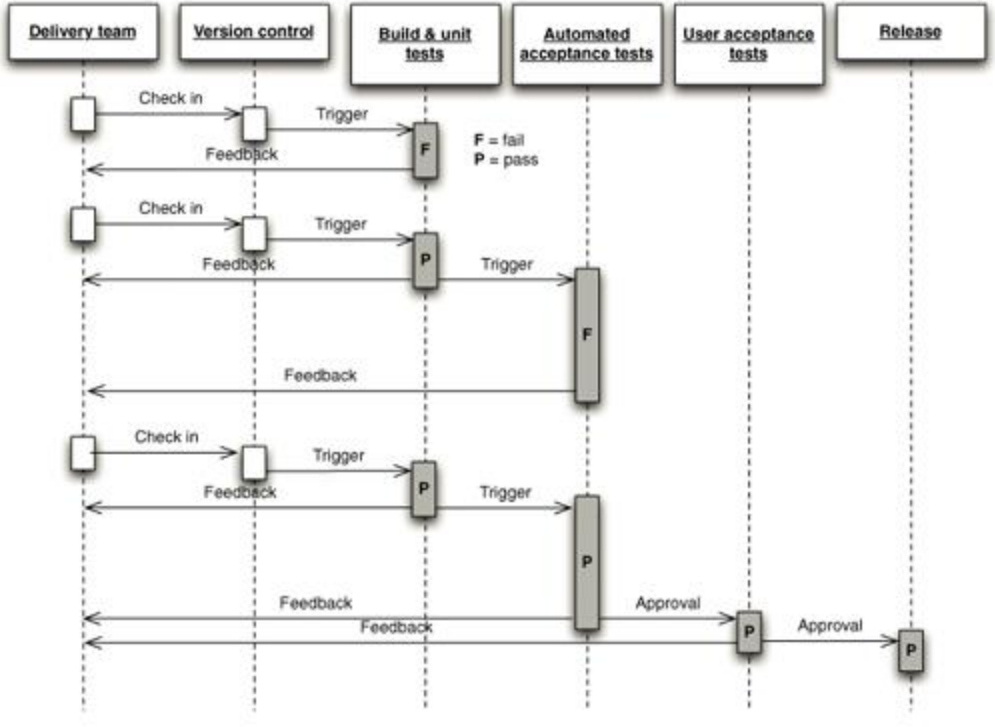
- Automation of (almost) everything
  - Software is put into production by the ‘push of a button’
  - Software is always working (that is, services customers!)

- Collaboration
  - DevOps
  - Full stack developers

AARHUS UNIVERSITET

Deployment Pipeline

- Deployment like a Processor pipeline
  - Like our hero passing a series of test to marry the princess



Jenkins, Concourse, Go, etc.

That is, a *clear set of stages, moving towards valuable software*

AARHUS UNIVERSITET

What Artefacts?

- *Software service customers*
  - What is the ‘unit of deployment’?
  - What is the ‘execution context’?

Both need automation!

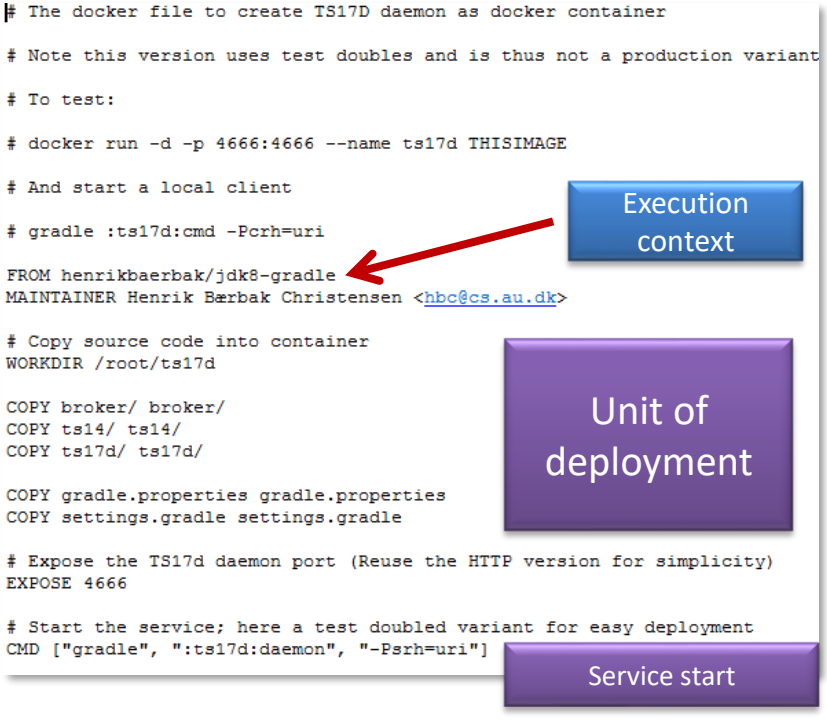
- Units of deployment
  - Ruby gems, Java jars and war, .NET dlls, Node.js npm, …
  - Ubuntu deb, CentOS RPM, Windows MSI, …

- Execution context
  - Apache tomcat, nagios, nginx, …

AARHUS UNIVERSITET

Containers - as Artifacts

- The artefacts we use: **Containers**
- Ex: Docker container
  - Provides execution context
    - Ex: Ubuntu 18.04, Java8, Gradle
  - Provides unit of deployment
    - Ex: Cave daemon service
- Can be automatically built
  - Dockerfile
- Can be released easily
  - Docker push (imagename)




```
# The docker file to create TS17D daemon as docker container

# Note this version uses test doubles and is thus not a production variant

# To test:

# docker run -d -p 4666:4666 --name ts17d THISIMAGE

# And start a local client

# gradle :ts17d:cmd -Pcrh=uri

FROM henrikbaerbak/jdk8-gradle
MAINTAINER Henrik Bærbak Christensen <hbc@cs.au.dk>

# Copy source code into container
WORKDIR /root/ts17d

COPY broker/ broker/
COPY ts14/ ts14/
COPY ts17d/ ts17d/

COPY gradle.properties gradle.properties
COPY settings.gradle settings.gradle

# Expose the TS17d daemon port (Reuse the HTTP version for simplicity)
EXPOSE 4666

# Start the service; here a test doubled variant for easy deployment
CMD ["gradle", ":ts17d:daemon", "-Psrh=uri"]
```

AARHUS UNIVERSITET

Immutable Servers

- Reusing servers leads to *Configuration drift*
- *Configuration drift:* The configuration settings in a given immutable version in our SCM does not match those in effect on our running host.
  - Typically, *someone* logged into the host and changed some parameters by *fiddling*

- **Solution:** Running servers are always the result of a deployment pipeline operation.
  - Error in config? No fiddling, but *fix configuration file, check-in, instantiate deployment pipeline!*

Nygard

- … says the very same

- *Immutable and Disposable Infrastructure:* Start from a known base image, apply fixed set of changes, and then never attempt to patch or update that machine.

- Right, not always possible
  - Example: Crunch machine needs configuring the Docker engine's network creation algorithm ☹

AARHUS UNIVERSITET

Again: Container Technology

- Do *not security-patch a container*
  - Instead, make a pipeline stage that produce the base image…
- Following stages will then produce the service container, based up the updated base image

Jenkins enabled JDK8+Gradle

```
# The docker file to create execution container for
# TS17-D on a Jenkins CI server.

FROM ubuntu:16.04
MAINTAINER Henrik Bærbak Christensen <hbc@cs.au.dk>

RUN apt-get -y update
RUN apt-get -y upgrade

RUN apt-get install -y openjdk-8-jdk
RUN apt-get install -y gradle

# docker ce
RUN apt-get install -y apt-transport-https
RUN apt-get install -y ca-certificates
RUN apt-get install -y curl
RUN apt-get install -y software-properties-common

RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

RUN add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

RUN apt-get -y update

RUN apt-get install -y docker-ce
```

AARHUS UNIVERSITET

Environments and Configuration

- The execution environment must be *configurable*
  - Development environment — Unit Tests
    - Use test doubles for fast development of new features
    - Fast unit-test execution
  - Staging environment — Service Tests, Journeys
    - Drilled down production-like environment
      - No firewall, fewer services
      - Docker images, created by TestContainers
      - Mountebank imposters instead of certain services
  - Production environment
- Solution: Create *one artifact* and *manage configuration separately*

Example

- SkyCave
  - Dependency injection using AbstractFactory+ObjectMgr patterns
    - Reading CPF property files

- Other recommendations
  - Environment variables (is cross platform, but reuse difficult)
    - And JUnit code *cannot set it* directly
  - Injection frameworks (Spring, Guice, “Bærbak CPF ☺”…)

AARHUS UNIVERSITET

Deployment Interface

- Newman's mantra: *… the most sensible way to trigger any deployment is via a single, parameterizable command-line call.*
- Three parameters
  - Artifact — *what service to deploy*
  - Version — *… in which version*
  - Environment — *… in which service configuration ?*
- `$ deploy artifact=ts17d environment=production version=b456`
- `docker run -d hbc/ts17d:b456 -Pcpf=production.cpf`

Humble

- Humble says something along the same lines…

Chapter 10. Deploying and Releasing Applications

Introduction

There are differences between releasing software into production and deploying it to testing environments—not least, in the level of adrenaline in the blood of the person performing the release. However, in technical terms, these differences should be encapsulated in a set of configuration files. When deployment to production occurs, the same process should be followed as for any other deployment. Fire up your automated deployment system, give it the version of your software to deploy and the name of the target environment, and hit go. This same process should also be used for all subsequent deployments and releases.

AARHUS UNIVERSITET

Deployment Interface

- But how does it relate to IaC?
- *Infrastructure-as-code*
  - Codify your infrastructure
- Exercise:
- Compare *deployment interface*
  - *Artifact?*
  - *Version?*
  - *Environment?*

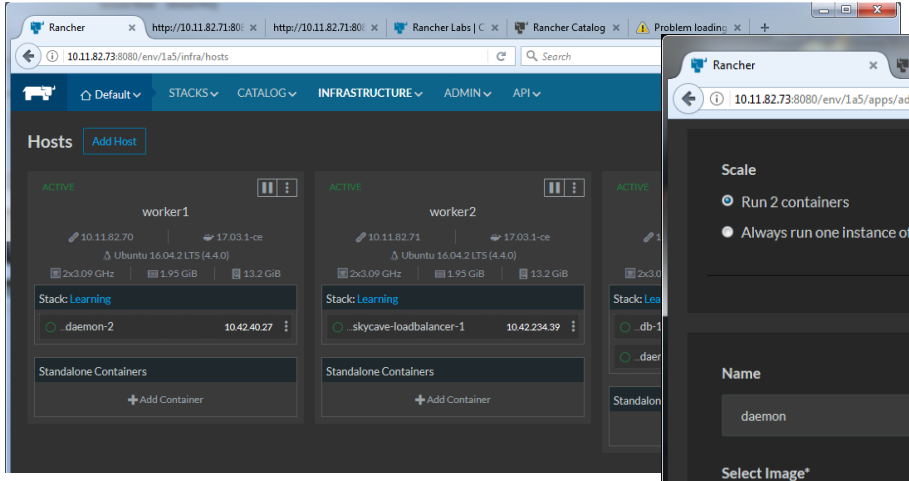
```yaml
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

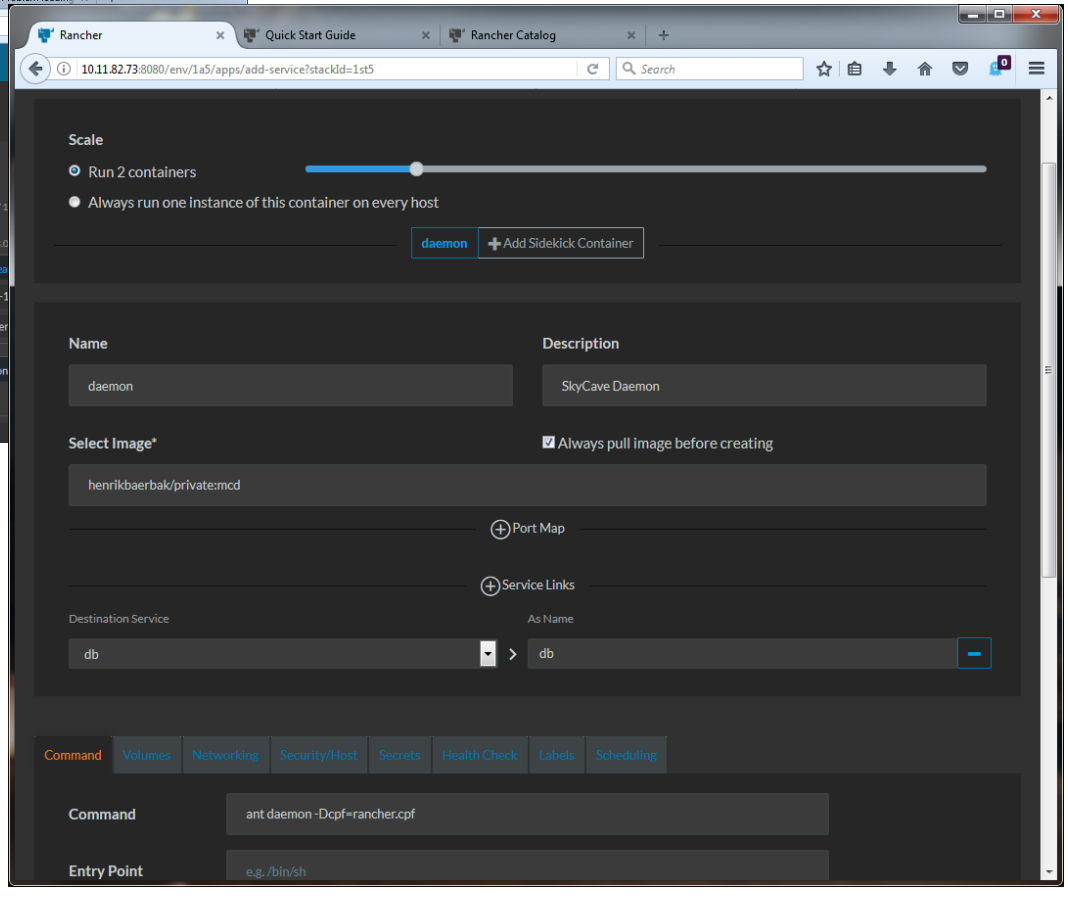
Discussion

- So – the question is…
  - Is a deployment interface the same as IaC
  - If not
    - Benefits of deployment interface?
    - Liabilities?
    - Benefits of IaC?
    - Liabilities?

AARHUS UNIVERSITET

A GUI *Depl-Intf*: Rancher





Point'n'Click hell ☹

PhoenixServer

- Fowler: PhoenixServer
  - Burn all you production equipment, and then measure the time until you are completely back in *normal operations*