Imhotep Vidensformidling inden for Softwareudvikling

Novicell Workshop: DevOps and Docker

April 2016

Henrik Bærbak Christensen
Associate Professor
Computer Science
University of Aarhus

Imhotep Vidensformidling inden for Softwareudvikling

The speaker...

Associated Professor since 2003

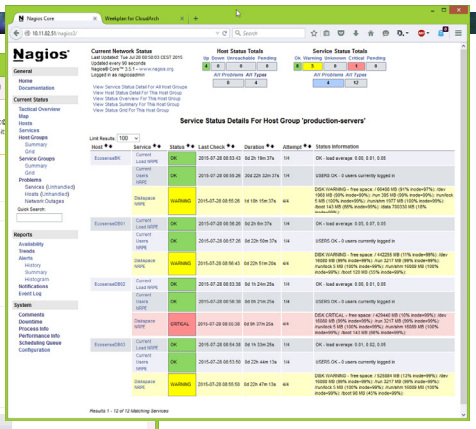
- Dept. of computer science / University of Aarhus
- Interests: Software architecture & Teaching
- Leader for SWK part-time education at AU

Industrial experience

- Architect *and* developer for a product suite of meteorological systems for Danish airports.
- Collaborations with Danish companies: Danfoss, Systematic A/S, B&O, Jyske Bank, TDC, Mjølner, KMD, and many others.
- Recently software intensive projects: Net4Care, EcoSense, CloudArch

… have SW in Production ☺

Imhotep Vidensformidling inden for Softwareudvikling



EcoSense
Karibu

Characteristics

Imhotep Vidensformidling inden for Softwareudvikling

- Designed for 24/7/365 operations
  - *No service windows at all*
    - *Last week we extended disk space from 12 TB -> 15 TB without service interruption*
    - *All security updates and reboots – without service interruption*
    - *All services can be updated and restarted – without service interruption [MQ is only exception!]*
  - *Services to be added at run-time*
    - *New dataformats, new apps, new producers – without service interruption*
  - *Performant and Scalable*
    - *~65 kB/s – and about 1% CPU load*
    - *More power? Add machines!*

But cannot survive: power cuts on all machines ☺

Agenda: Take Away Points

Imhotep Vidensformidling inden for Softwareudvikling

DevOps:

- Agility in development as well as in production
- *Full stack development:* Teams do the full stack

MicroServices:

- Decentral data and governance, *products* not projects
- Design for failure

Docker:

- *Infrastructure as code*
- Lightweight virtualization: *containers ship anywhere*

Software Architecture!

- *Technology fix will not help if architecture is wrong*

Literature

Imhotep Vidensformidling inden for Softwareudvikling

[Anderson, 2015] Charles Anderson (2015) *Docker*, IEEE Software, May/June 2015. (Online)

[Bahga and Madisetti, 2014] A. Bahga and V. Madisetti (2014). *Cloud Computing - A Hands-On Approach* Published by >A. Bahga and V. Madisetti

[Bernstein, 2014] David Bernstein (2014) *Cloud: From LXC to Docker to Kubernetes*, IEEE Cloud Computing, September, 2014. (Online)

[Buschmann, 2011] Frank Buschmann, 2011 *Tests: The Architect's Best Friend* In IEEE Software, Maj/June 2011 (Online)

[Fink, 2014] John Fink (2014) *Docker: a Software as a Service, Operating System-Level Virtualization Framework*, code4lib journal, Issue 25, 2014. (Online)

[Fowler, 2006] Martin Fowler, 2006 *Continuous Integration* (Online)

[Lewis et al., 2014] James Lewis and Martin Fowler (2014) *Microservices*. (Online)

[Nygard, 2007] Michael T. Nygard (2007). *Release It! Design and Deploy Production-Ready Software* Pragmantic Bookshelf

[Rosenblum et al., 2005] Rosenblum, M., Garfinkel, T. (2005) *Virtual Machine Monitors: Current Technology and Future Trends* IEEE Computer, May 2005 (vol. 38 no. 5) (Online)

[Smith et al., 2005] Smith, J. E., Nair, R. (2005) *The Architecture of Virtual Machines*. IEEE Computer, May 2005 (vol. 38 no. 5). (Online)

[Christensen, 2010] Henrik Bærbak Christensen *Flexible, Reliable Software -- Using Patterns and Agile Development* CRC Press, 2010

DevOps

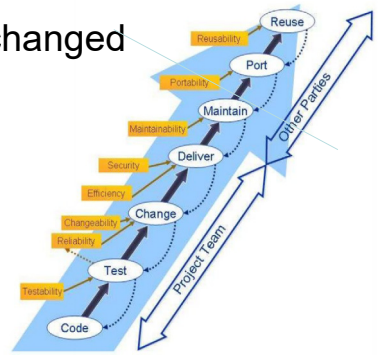
The Problem and a Solution

DevOps ?

I am a software developer! I develop software, and then someone else puts it into production

– Makes sense for ‘Word’ and ‘Excel’

However, ‘software’ has changed

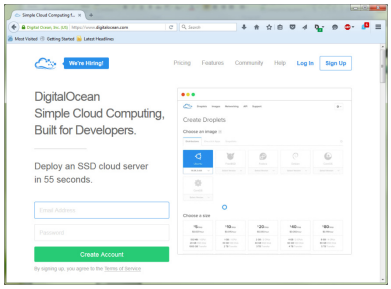


Software Anno 2016

Imhotep Vidensformidling inden for Softwareudvikling

Software no longer comes on a CD. It is already live on the internet

Windows Small Business Server 2011

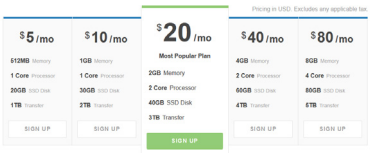


Computing as Utility

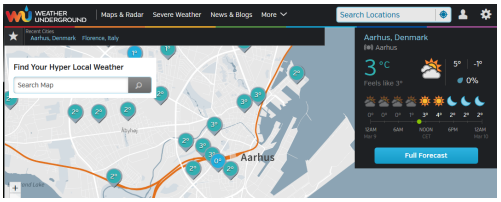
Software Anno 2016

Imhotep Vidensformidling inden for Softwareudvikling

I *buy the capacity* I need, not the machinery to execute on…



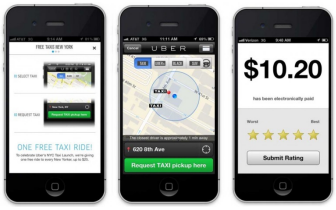
And why develop services, if they are *already there*?



Computing as Utility

Software Anno 2016

Imhotep Vidensformidling inden for Softwareudvikling



New Years eve:
- 60 bookings per second

Multiple data centers
- 1000s of servers

Millions of connected devices

Example: Uber

Services
- Fare estimation
- Tracking car movement
- Select nearest car
- Fare splits
- Supply positioning
  - Heatmaps
- Fraud detection
- Dynamic pricing
- Backend processing
  - Optimization

All made using open-source software!

But How to Develop?

Imhotep Vidensformidling inden for Softwareudvikling

From 2015 Master of IT thesis
- A typical development team includes UI designers/developers, backend developers, and database experts. […] and many other companies split the development process into several components, such as UI, backend, database and mobile applications.

Why does this make sense?

UI designers

Application Server developers

Database designers

Bottlenecks

Crossing the *boundaries of environments* for the full development-testing-deployment cycle is costly…

- Reconfigurations
- Setup
- Changed architecture
- Changed HW

Even worse

- Different people
- *Someone else's problem*

DevOps

Charles Anderson: Docker

> The DevOps movement, for example, emerged from one of the classic stumbling blocks in a lot of organizations. Developers build code and applications and ship them to the operations people, only to discover that the code and applications don't run in production. This is the classic "it works on my machine; it's operations' problem now."

So

The problems

- Boundaries between *people*
  - *Is it the fault of the UI guys, or the DB guys*
- Boundaries between *environments*
  - *But it works on my machine ???*

DevOps Movement

DevOps

From Wikipedia, the free encyclopedia

DevOps (a clipped compound of "development" and "operations") is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.[1][2] It aims at establishing a culture and environment where building, testing, and releasing software, can happen rapidly, frequently, and more reliably.[3][4][5]

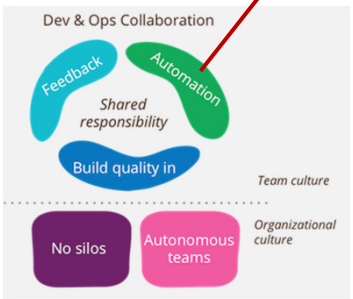
Rouan Wilsenach
9 July 2015

Agile software development has broken down some of the silos between requirements analysis, testing and development. Deployment, operations and maintenance are other activities which have suffered a similar separation from the rest of the software development process. The DevOps movement is aimed at removing these silos and encouraging collaboration between development and operations.

DevOps has become possible largely due to a combination of new operations tools and established agile engineering practices [1], but these are not enough to realize the benefits of DevOps. **Even with the best tools, DevOps is just another buzzword if you don't have the right culture**.

The primary characteristic of DevOps culture is **increased collaboration** between the roles of development and operations. There are some important cultural shifts, within teams and at an organizational level, that support this collaboration.

Virtualization, cloud computing, automated configuration management



http://martinfowler.com/bliki/DevOpsCulture.html

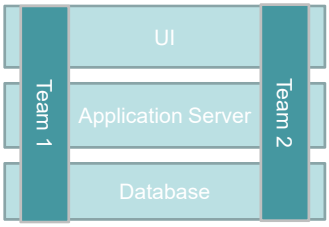
DevOps

Full stack developer

– Small teams do *everything*: db design, code development, UI, config, testing, deploy, monitor, production

Uber development

– 08-12 I develop it
– 12-16 I deploy it
– 16-08 I monitor it
  • (pager at bed table!)



Infrastructure as Code

The new type of *code* on the block

Infrastructure Code

```
1  regservice:
2    build: .
3    ports:
4      - 4567:4567
5    links:
6      - mongodb:mongo
7
8  mongodb:
9    image: mongo:latest
10
```

- *Coding application logic*: The programming of logic for the core business functionality under the assumption that all services work correctly. Example: Developing application server code to access an external inventory service and retrieve the price and stock availability of an item.
- *Coding quality attribute logic*: The programming of logic for architectural quality attributes [4] (non-functional requirements) like availability or performance, such as to handle the situations where one or several services fail, are slow, or produce unexpected results. Example: Developing code that implement graceful failure modes in case the inventory service is too slow to respond due to a peak load or simply unavailable due to network failure.
- *Coding infrastructure logic*: The programming of logic for the deployment of services. Traditionally handled by manual procedures (installing, configuring, and linking services), but in face of large-scale deployments, this too must be coded. Example: Developing scripts that start the application server, inventory service and associated database, initialize them, and connect them correctly—i.e. create a *staging environment*.

Teaching DevOps and Cloud Computing using a Cognitive Apprenticeship and Story-Telling Approach

Henrik Bærbak Christensen
Computer Science / Aarhus University
8200 Aarhus N - Denmark
hbc@cs.au.dk

MicroServices

Micro Services

The agony of being an old man…

- Same ideas, new packaging, pops up every 10 years
  - Modular programming, object-orientation, component-based, service-oriented,… and now *micro services*
- All the same idea, dating to David Parnas
  - *Information hiding* in *modular programming*
- Gamma et al.
  - ❶ *program to an interface*
  - ❷ *favor object composition*
  - ❸ *encapsulate what varies*

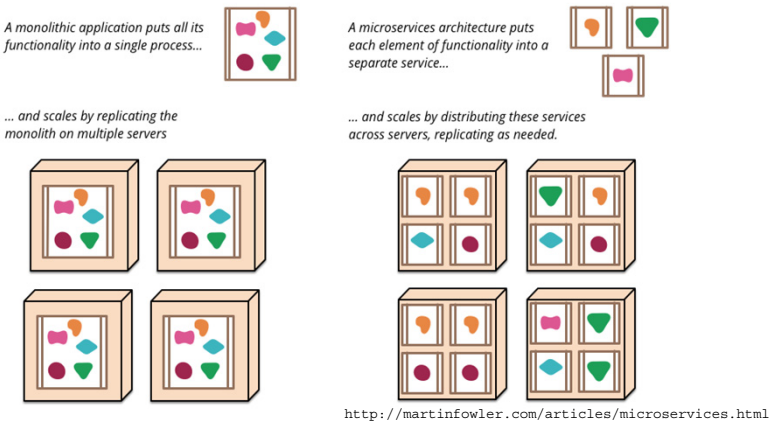
What varies?

The idea is the same, only the packing and wiring varies

- Modula 2 module = compilation unit, static link
- Class = compilation unit, static link
- Components = deployment/static unit, static link
- SOA = deployment/run-time unit, dynamically linked
- MicroService = deployment/run-time unit, dynamically linked

What is New?

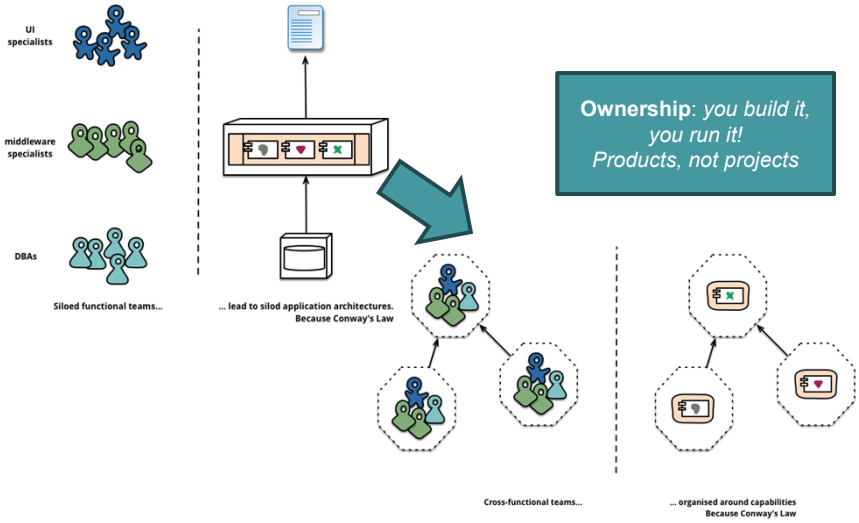
The *real* new concept here, IMO, is the *self-containedness* and *autonomy* of a MS



http://martinfowler.com/articles/microservices.html

MS = Business Capability

Imhotep Vidensformidling inden for Softwareudvikling



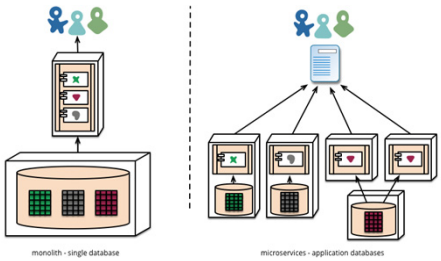
Decentralizing…

Imhotep Vidensformidling inden for Softwareudvikling

Governance
- Choice of platform and technology by the team
  - *Pick the right tool for the job*

Data management
- MS = Products *own* their data

Communication
- Typical
  - Network
  - RESTish
    - Lightweight
    - Extensible
    - Fault tolerant



Design for Failure !

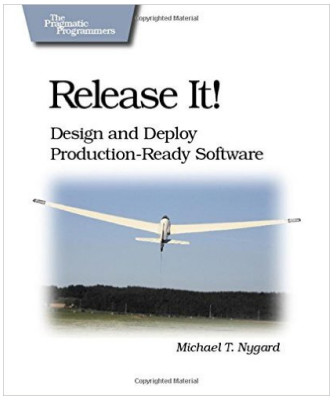
Get that book!

Safe failure modes

- *Failing services*
- *Slow responses*
- *Cascading failures*

Redundancy

Horizontal scaling



CAP theorem

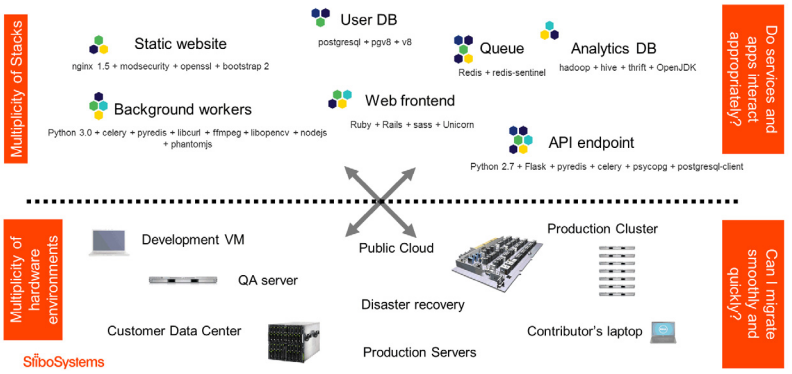
- *Consistency* (all nodes see the same data at the same time)
- *Availability* (a guarantee that every request receives a response about whether it succeeded or failed)
- *Partition tolerance* (the system continues to operate despite arbitrary partitioning due to network failures)

Containers and Docker

The Problem Again

Imhotep Vidensformidling inden for Softwareudvikling

Crossing boundaries, that is, *moving code*



Source: Torben Haagh, StiboSystems

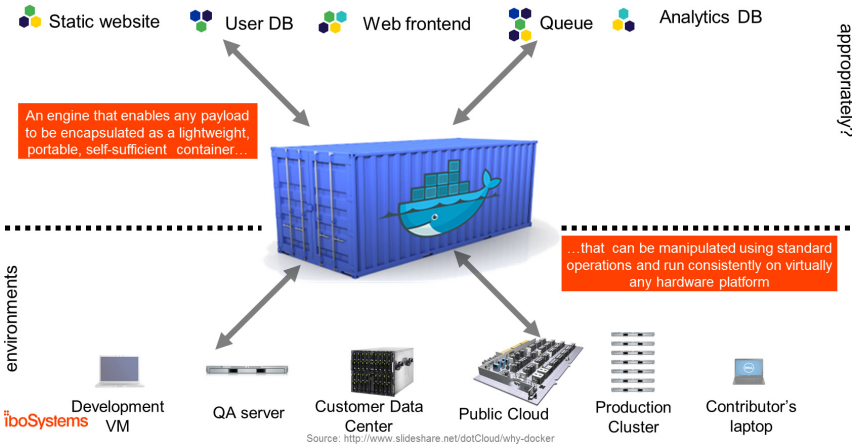
Was Solved in 1960'ies

Imhotep Vidensformidling inden for Softwareudvikling

Docker = Container

Imhotep Vidensformidling inden for Softwareudvikling

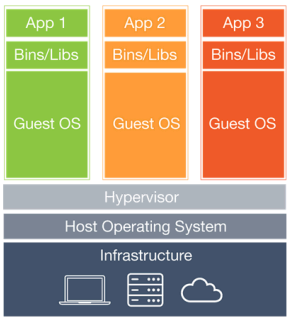
Docker is a shipping container system for code



Light-Weight virtualization?

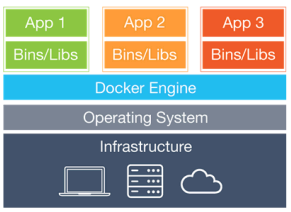
Imhotep Vidensformidling inden for Softwareudvikling

Moving the virtualization boundary



Traditional VVMs

What consequences do Docker then dictate on the VMs it hosts?



Docker VVM

Docker Engine

Imhotep Vidensformidling inden for Softwareudvikling

Core concepts

- Image Deployment Unit – a Virtual Machine
  - Naming: **ownername/imagename:tag**
- Container Executing process(es) in an Image
  - Naming: '**trusty_thar**' or give it your own name

Lifecycle *classic* for building *your* image

- container = instantiate(image)
  - docker run –d –name c1 baerbak/dockerdemo:v1
- modify container
  - Install software, change files, add stuff, …
- commit container → image2
  - docker commit c1 baerbak/dockerdemo:v2

Tech Glimpse

Imhotep Vidensformidling inden for Softwareudvikling

Onion file system: *Copy-on-Write*

- Every operation basically creates a new *file layer*
  - *Changing 'hans.txt' in layer N creates a (modified) copy of 'hans.txt' in layer N+1*

Base images = 'prebaked file system'

- All layers up-till N forms an Image

I.e. henrikbaerbak/cloudarch:base_v2

- Ubuntu 14.04 LTS server base image
- Java, Ant, Ivy, Git, … are all layered on top

Docker Engine

But but – *infrastructure code ???*

Lifecycle *scripted* for building *your* image

Dockerfile

- *Infrastructure code* at the image level
- Defines the contents of the image

- docker build –t (name)
  - creates the image from local 'Dockerfile'

```
FROM henrikbaerbak/cloudarch:base_v1

# CaveReg docker file. Requires a running mongodb container
# named 'mongo' running.

MAINTAINER Henrik Bærbak Christensen <hbc@cs.au.dk>

# Create the /root/regservice

RUN mkdir /root/regservice
WORKDIR /root/regservice

# Copy local stuff there
COPY src/ /root/regservice/src/
COPY test/ /root/regservice/test/
COPY resource/ /root/regservice/resource/

COPY build.xml /root/regservice/build.xml
COPY ivy.xml /root/regservice/ivy.xml

# Setup the entry point
COPY entry-point.sh /root/regservice/entry-point.sh
RUN chmod 755 /root/regservice/entry-point.sh

# Default port exposed, standard spark-java port
EXPOSE 4567

# Entrypoint, you need to supply the 'web' as the ant target to run
ENTRYPOINT ["/root/regservice/entry-point.sh"]
CMD ["web"]
```


Docker Hub

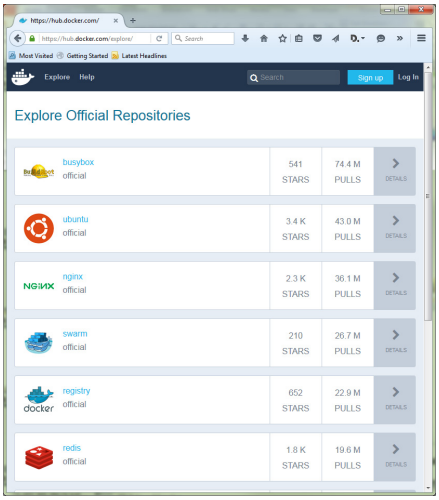
Similar to Maven Repo

Repo of open-source images

- docker pull (name)

Free hosting

- Register and
- docker push (name)



Example

Imhotep Vidensformidling inden for Softwareudvikling

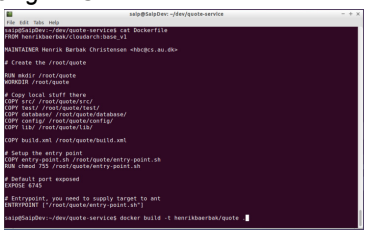
Exam exercise

- *Integrate with a 'Einstein Quote' service using REST*

Step 1: *develop… and make Dockerfile*

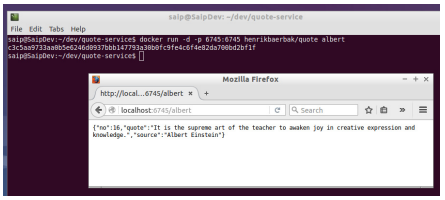
Step 2: *build an image*

Step 3: *push to docker hub*



Step 4: Now all students can pull and start it

- docker run -d -p 6745:6745 henrikbaerbak/quote albert
- Browse: localhost:6745/albert



Why not try yourself?

Henrik Bærbak

MicroServices

Imhotep Vidensformidling inden for Softwareudvikling

Many systems depend upon other services

- Ex: LAMP stack:
  - Linux/Apache Server/MySQL/PHP

Docker Compose

- *Infrastrucure code* for multi-container deployments

```
1  regservice:
2    build: .
3    ports:
4      - 4567:4567
5    links:
6      - mongodb:mongo
7
8  mongodb:
9    image: mongo:latest
10
```

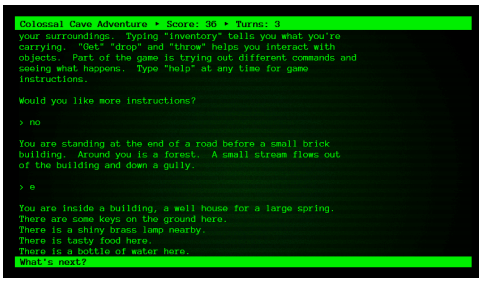
Docker in Action

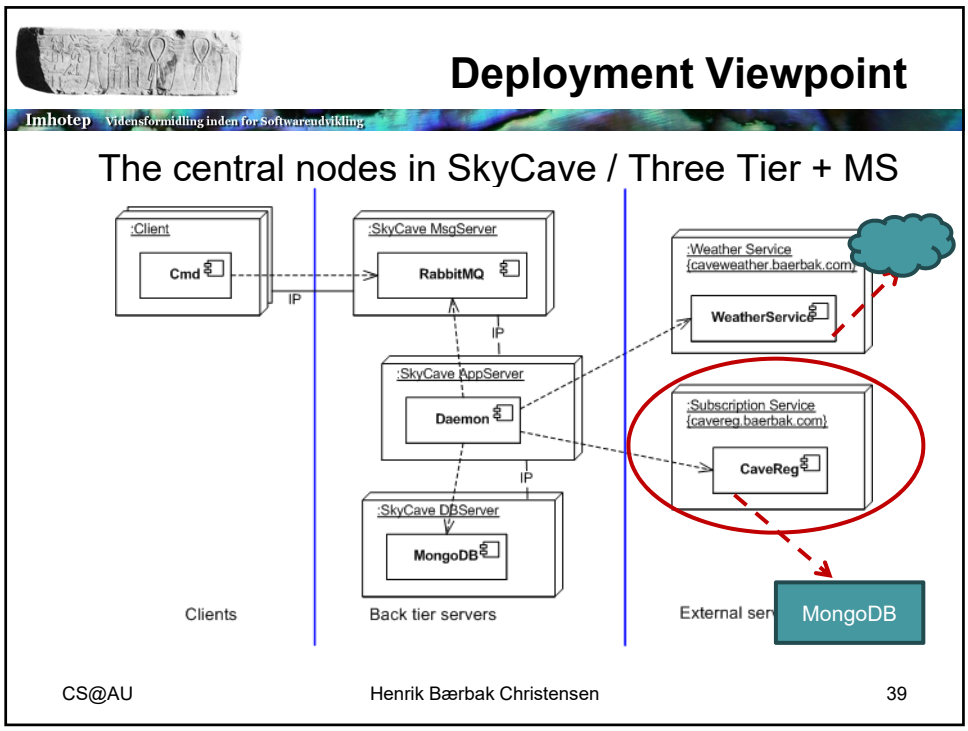
From my
Cloud Computing Course

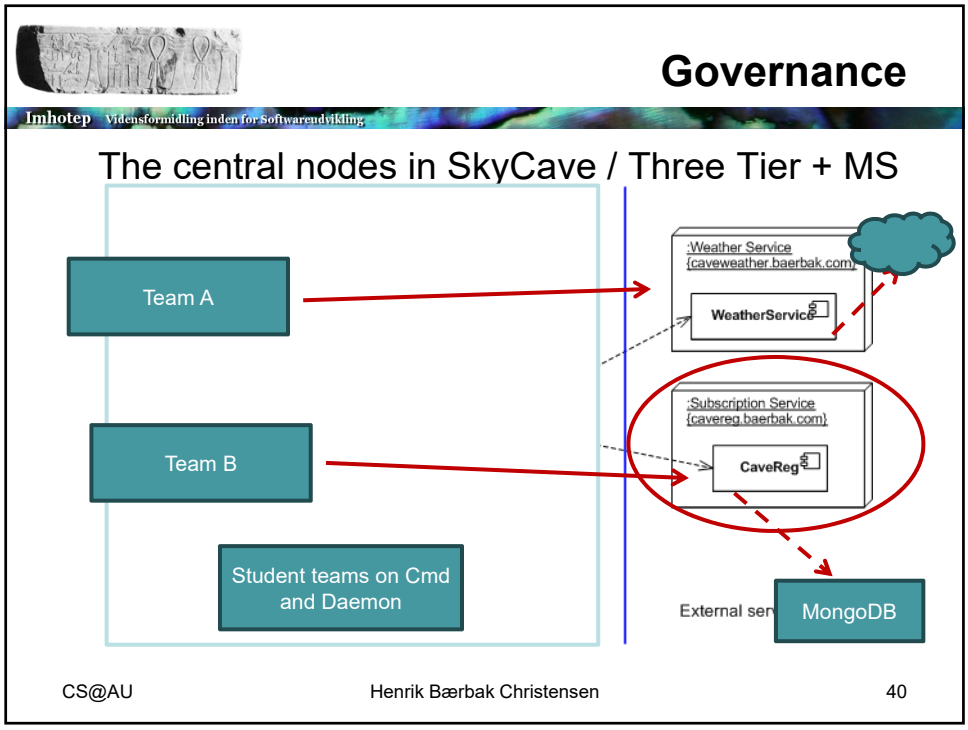
SkyCave

Massive Multiplayer Online (MMO)
Social Networking
Domain: ‘Colossal Cave’ – reimagined

Characteristics
- Subscription-based
- Single Sign-on
- NoSQL
- REST services



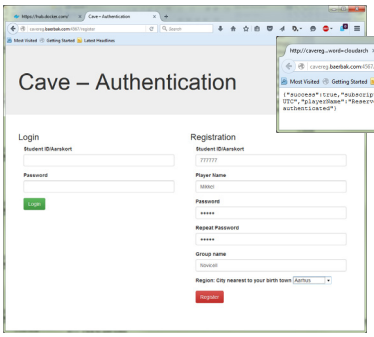

Deployment Viewpoint
Imhotep Vidensformidling inden for Softwareudvikling
The central nodes in SkyCave / Three Tier + MS
:Client
Cmd
IP
:SkyCave MsgServer
RabbitMQ
IP
:SkyCave AppServer
Daemon
IP
:SkyCave DBServer
MongoDB
:Weather Service
{caveweather.baerbak.com}
WeatherService
:Subscription Service
{cavereg.baerbak.com}
CaveReg
MongoDB
Clients
Back tier servers
External ser
CS@AU
Henrik Bærbak Christensen
39

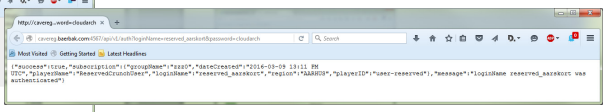

Governance
Imhotep Vidensformidling inden for Softwareudvikling
The central nodes in SkyCave / Three Tier + MS
Team A
Team B
Student teams on Cmd
and Daemon
:Weather Service
{caveweather.baerbak.com}
WeatherService
:Subscription Service
{cavereg.baerbak.com}
CaveReg
MongoDB
External ser
CS@AU
Henrik Bærbak Christensen
40

MicroService: Subscription

The Subscription service is a web site, backed up by a MongoDB database, which allows:

- Creating a new subscriber
- Authenticating through REST call





Thus, to run, a MongoDB container must be live first!

Example: Docker Compose

Multiple compose files

- Layering
  - Base: Core stuff for staging environment
  - Prod: Additional stuff for real production running

```
1  regservice:
2    build: .
3    ports:
4      - 4567:4567
5    links:
6      - mongodb:mongo
7
8  mongodb:
9    image: mongo:latest
10
```

```
1  mongodb:
2    volumes:
3      - /mongo_datadir:/data/db
4
5
```

For staging

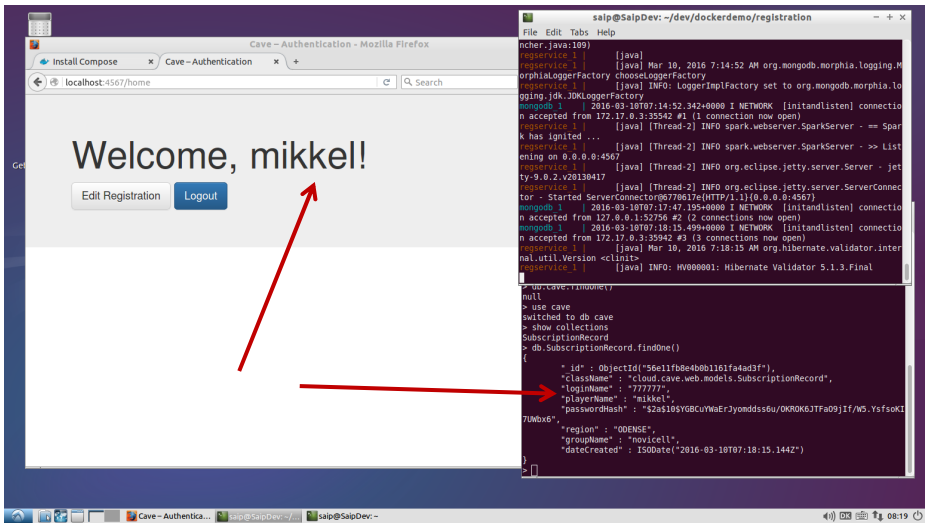
- docker-compose up
  - Read only docker-compose.yml

For production

- docker-compose –f docker-compose.yml –f docker-compose.prod.yml up -d

Example

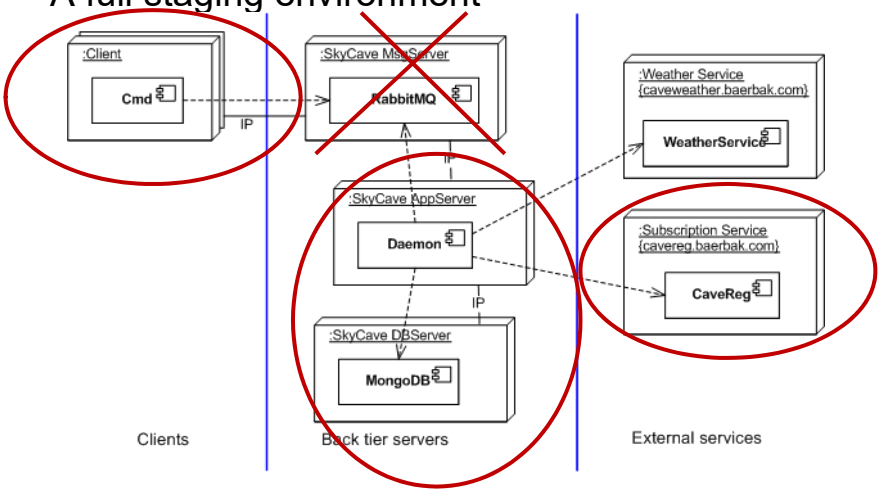
Imhotep Vidensformidling inden for Softwareudvikling



Deployment Viewpoint

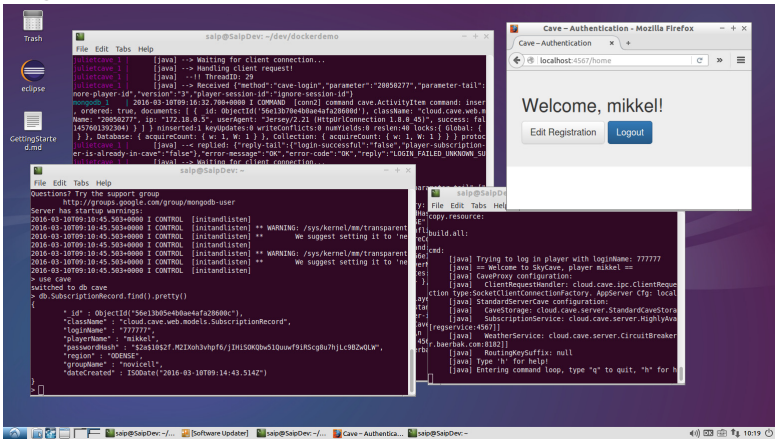
Imhotep Vidensformidling inden for Softwareudvikling

A full staging environment



Full Staging

Adding a SkyCave server with own database, staged subscription server with own database



Four server Docker-compose


```
# Docker compose for DockerDemo / Novicell workshop

# Demonstrate a four server staging environment, set up by
# docker-compose automatically

# by Henrik Bærbak Christensen / Aarhus University

version: '2'
services:

  # --- Juliets SkyCave server
  julietcave:
    build:
      context: ./juliet-cave
    ports:
      - "37123:37123"
    links:
      - cavedb
      - regservice

  # --- MongoDB used by SkyCave server
  cavedb:
    image: mongo:latest


  # --- Staging registration/subscription server
  regservice:
    build:
      context: ./registration
    ports:
      - "4567:4567"
    links:
      - mongodb

  # --- MongDB used by subscription server
  mongodb:
    image: mongo:latest

```

Juliet's server, depends on 'cavedb' and 'regservice'

Subscription server, depends on 'mongodb'

Discussion

Benefits:

- *Infrastructure as code*
  - Fast, reproducable execution of deployments
  - Excellent documentation of deployment view

Liabilities:

- Docker is a moving target
  - E15: Linking was the central composition tool; now deprecated!
  - Documentation erosion!
- Layered file system: Take care with security!
  - No, you cannot delete the credential file again…

… And software architecture

Discussion

Imhotep Videnformidling inden for Softwareudvikling

MicroServices and Docker are **patterns** and **technology**, they do not by themselves solve the deep architectural issues of 'hard to maintain monoliths'

- *Loose coupling* and *High cohesion* are challenging no matter what technology you use!
- Software Architecture =
  - Making the *right* decisions on
    - Splitting the behaviour in the proper units
    - Providing adequate connections between the units
    - Controling quality attributes
      - » Especially availability, modifiability, performance in MS

Discussion

Imhotep Videnformidling inden for Softwareudvikling

If a MS architecture

- Have bad boundaries between units
- Require all units' behaviour and the networks between them to operation correctly

… you still have a Monolith that is very hard to maintain and grow…

Discussion

Design for failure, means

- Introduce proper techniques to handle cascading failures at the integration points
  - Detect failure
- Consider how graceful degradation works
  - How to continue operation once the failure occurs?

Example:

- Player wants to log into SkyCave, but the subscription server is not responding
  - Reject login? (Loss of interest, bad reputation)
  - Accept limited login? (”Karl, I know him, let him in…”)

Summary

Agenda: Take Away Points

DevOps:

- Agility in development as well as in production
- *Full stack development:* Teams do the full stack

MicroServices:

- Decentral data and governance, *products* not projects
- Design for failure

Docker:

- *Infrastructure as code*
- Lightweight virtualization: *containers ship anywhere*

Software Architecture!

- *Technology fix will not help if architecture is wrong*

Thanks…

Questions?