Advanced Enterprise Computing - Lecturenotes SoSe2016

Julius Hülsmann

20. Juli 2016

Inhaltsverzeichnis

1 Repetition

1.1 Containerization

encapsulate application together with its operating environment

1.2 Software-defined Networking

Encapsulation: gives access to configuration of "ntwork-layer"without being confronted with the hardware.
Separation into **Control Plane** and **Data Plane**

1.2.1 Control Plane

Make decision on data's destination
The control plane functions include the system configuration, management, and exchange of routing table information

1.2.2 Data Plane

Also known as Forwarding Plane
Forwards traffic to the next hop along

1.2.3 Example from sdntutorials.com

The protocol or application itself doesn't really determine whether the traffic is control, management, or data plane, but more importantly how the router processes it. Consider a 3 router topology with routers R1, R2 and R3. Lets say a Telnet session is established from R1 to R3. On both of these routers the packets need to be handled by the control/management plane. However from R2's perspective this is just data plane traffic that is transiting between its links.

1.3 Continuous Integration

Improvement, Automatization of the Deployment organization

1.3.1 Deployment Pipeline

Menge von Validierungen, die eine Software auf ihrem Weg zur Veröffentlichung bestehen muss

1.4 Continuous Delivery

Improvement of the Deployment organization
Wichtiger Begriff: Deployment pipeline

Laut Wikipedia „Weiterentwicklung“von Continuous Integration, indem extends CI by making sure the software checked in on the mainline is always in a state that can be deployed to users and makes the actual deployment process very rapid.

2 Dev-Ops

2.1 Definition and Goals

- Set of practices

Aim:

- Reduce Time between committing and normal Use of the software
- high Quality

Practices

- Iterative ⇒ ok 4 changing requirements, fast delivery and high-quality
- small teams, improve collabortation dev- & ops, reduce communication
- frequent commits, automatic deployment & testing, early problem detection

Influences

- the way systems are built
- (incl.) the Organization of Teams
- the structure of the built system

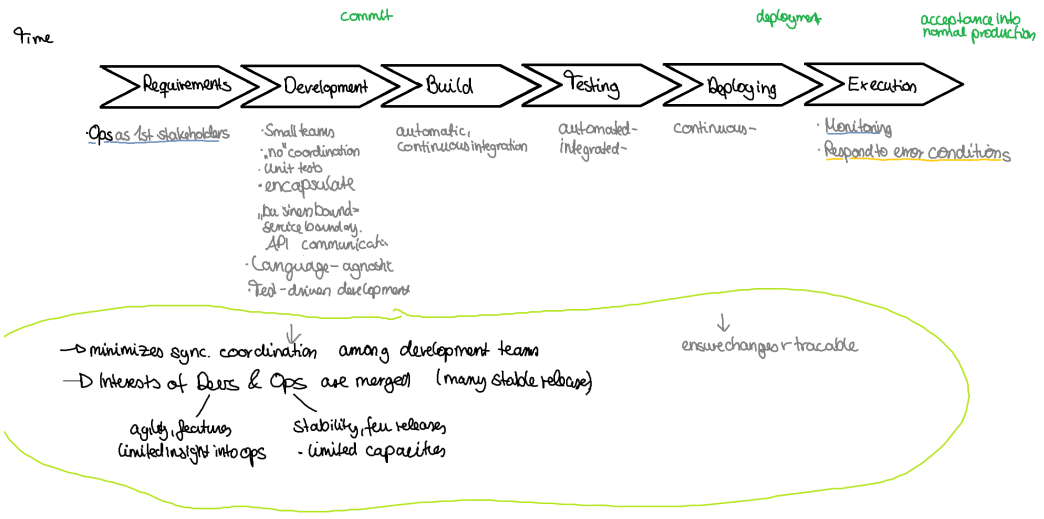

Abbildung 1:

3 Microservices

3.1 Definition and Motivation

Way of developping:

- architectural style
- single application = Set of small services (own processes, programming language, databases
- technologies); communication often via EXTERNALIZABLE APIS (often lightweight http)
- independently deployable, automatic
- are minimum of centralized management
- – no direct linking, no direct reads of another team's data store, – no shared-memory model, no back-doors whatsoever. – The only communication allowed is via service interface calls over the network.

Frage: Wie weit verschachtelt (depth) sind die Microservices?
Unterschied zu Monoliths (all functionallity in one application, scaled by replication) Microservice: (Scales by distributing services to replicas; Replication as needed)

3.2 Characteristics of Microservices

1. Componentization via Services
   Component = independant, upgradable
   Explicit dependencies
   No specific assumptions on execution environment
   Different possible realizations (programming lang.., code structure)
   Composable from other software components
   Libraries = Components linked into program; in-memory-function calls
   Service = out of process, Communicate f.i. via http
2. Organized around Business Capabilities
   Wenn das Team nach Gruppen sortiert ist Auswirkungen auf Program. Besser ggf mix
3. Products not Projects
   you build, you run it vs. nur build

4. Smart end points and dumb pipes
   as decoupled and as cohesive as possible, filters in Unix sense

5. Decentralized Governance
   Centralized ⇒ standartize technology used. Not good

6. Decentralized Data Management (= multiple DB with different content)

7. Infrastructure Automation
   (= Automated, continuous deployment and testing processes)

8. Design for failure
   design to tolerate failures of other services (could fail due to unavailability of the supplier)
   constantly reflect on how service failures affect the user experience

9. Evolutionary Design
   granular release planning. Monolithic: build and deployment of the entire application.
   microservices, only need to redeploy the service(s) you modified

4 Continuous Engineering

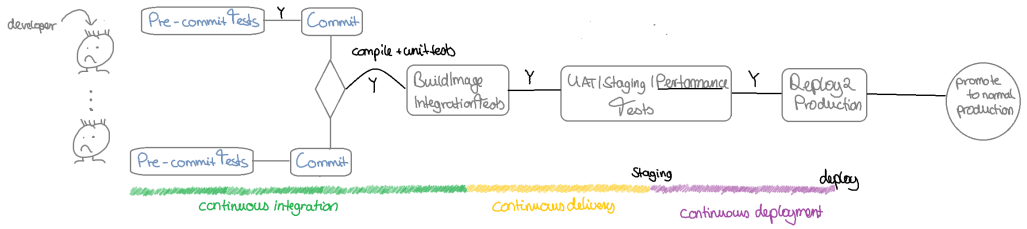

Abbildung 2:

Probleme Deployment: Es kann sich ändern, wie neu deployter Microservice andere - benutzt und benutzt wird. Any deployment team has to be able to deploy at any time it takes time to replace one VM Service to clients must be maintained

4.1 Strategies

4.1.1 All-or-nothing (alle Testenden Involved)

- Blue-Green or Big Flip Deployment leave N VMs with version A as they are, allocate and provision N VMs with version B, switch to version B; once stable and release VMs with version A.
- Rolling upgrade ()allocate a new VM with version B, release one VM with version A. Repeat N times)

Blue-Green	Rolling Upgrade
Only one version available at a time	2 versions available
2N VMs	N+1 VMs
Rollback easy	

4.1.2 Partial strategies - kleine Gruppe enduser

- Canary testing few servers of a new version, running in production in order to perform live testing in the real environment Testers. organization-based (dog-fooding) / geographically based / Rand Rout the specified requests to the testing image (e.g. Load balancer), observe
- A/B Testing Both systems in parallel, tested by someone wo wants new features
- – Chaos / LATENCY Monkey: randomly kills VMs in production