CONTENTS

PART I

FOUNDATIONS

CHAPTER 1

INTRODUCTION: DEVELOPER AND OPERATION TEAMS CONVERGE AND BOTH USE SOFTWARE ENGINEERING PRACTICES

Many people talk about DevOps as well as there are multiple definitions and interpretations of the term DevOps. DevOps is referred as a philosophy, a culture, practices and specific tools. For my research, I will focus on two different aspects of the term DevOps:

The first one is the perspective of operation teams. Operation teams traditionally modeled infrastructure by installing physical hardware and by manually installing software components. With the rise of virtual machines and the cloud, it became possible to model infrastructure in software[1]. Modelling via software enables operation teams to use tools and practices[2] as seen in software engineering. Infrastructure code is version controlled, tested and can be automatically deployed.

The other aspect of DevOps[3] is the perspective of developer teams. Previously developer teams were only responsible for developing new features. Software engineering practices got established and proven. One of those practices is the continuous delivery pipeline[4]. The last step of the continuous delivery pipeline is the deployment. Formerly operation teams were responsible for deploying new features. The deployment as last step of the continuous delivery pipeline shifts a responsibility from operation to development. This shows that developer teams are becoming more and more responsible for running the software, they built.

[1]"Infrastructure as Code" describes different dynamic infrastructure types [4, p. 30] and how to model those by code [4, p. 42].
[2]In the chapter "Software Engineering Practices for Infrastrucure" [4, p. 179-194] practices like version controlling, continuous integration are described.
[3]The book "DevOps" [1] is written in the view of a developer running a system.
[4]For theoretical details on the continuous delivery pipeline read Part II of "Continuous Delivery" [3, p. 103-140] or a more practical approach by Wolff [5].

CHAPTER 2

DEVELOPERS USE THE CONTINUOUS DELIVERY PIPELINE

2.1 The Continuous Delivery Pipeline consists of commitment, continuous integration and deployment

2.2 Software Deployment approaches evolved from manual to automated

2.2.1 Blue-Green Deployment allows Zero Downtime releases

The approach of blue green deployment involves two environments. One is the environment which serves production traffic, the other environment is in standby. You deploy to the other environment the new version. You then switch routing from the environment, which runs the old version, to the environment with the new version.

This has the major disadvantage that it is a waste of resources. Just one environment doing work with serving production traffic as the other environment is just idling. Even if you use the idling environment as a staging it would be oversized and still wasting resources.

2.2.2 Automation leads to resource saving Phoenix Deployment and Rolling Deployments

In times of dynamic resource allocation, automation and virtual machines, it became easy to automatically spawn new servers. In for a deploy of a new version you would not change the servers, but automatically create new ones with the version to deploy, then switch the traffic from the old servers to the new servers, and in the end just destroy the old servers. This procedure is called phoenix replacement.

2.2.3 Canaries test releases with a small amount of traffic

Canary releasing is a way to test new versions of the application in production. But testing in production is risky, because when there is an error which leads to a defect or failure the users will get affected. And this costs money in any way.

There comes canary releasing into play. A single canary can't do much harm and it's not a big catastrophe if the small canary is malicious. Let me explain it at the example of a typically scaled web application. Usually you do not have just a single web server, but multiple servers. So when releasing in the canary style, you change just a small proportion of the twenty webservers to the new version. Now two versions of the webservers are running at the same time.

Usually you want exactly the same version of webservers in production with the exact same configuration. This makes systems easier to debug in case of an error. The maximum count of different versions during canary releasing is two. But why go for two different versions in production with the canary releasing technique, when it makes debugging in general harder.

It's because with canary releasing you can achieve different goals. The first one is truly when an error occurs. Just because less users are affected by the error. The majority of webservers is still in the old version, so just a small portion of all the users have a poor experience with the erroneous small canary version fraction.

In case of success, when everything works as expected, it is proven that there is less risk of errors, even under real production conditions.

2.2.4 continuous deployment is not continuous delivery

CHAPTER 3

OPERATORS TURNED INTO SITE RELIABILITY ENGINEERS

Companies, which run software on their servers, usually have a specialized operations team. A team of developers programs the software and after the team is finished, the software is handed to another team, which runs the software and keeps it running. Those teams are usually referred as operations teams. So what are their duties and responsibilities exactly?

At first the already developed software needs to be deployed on an infrastructure. Therefore the operations team must provide an infrastructure. Infrastructure generally consists of hardware like racks, servers and networking devices. But in most cases, software depends on other software components and services, which are for example programming languages, dns or databases. Software usually changes. Most importantly there are security patches, but also software updates with new features are mandatory to install. Hence the operations team is responsible for installing, configuring and updating hardware and software components.

To provide a certain quality of service, infrastructure as well as the software must be maintained. That means identifying any problems and moreover fixing the identified problems. To identify problems in the first place, there must be any kind of monitoring. A good monitoring system[1, p. 127/8] finds failures and resulting faults, recognizes performance problems, usual workloads and it detects intruders. More obvious problems can be detected early, even before the problem leads to failures. For example a disk will soon be full or utilization of resources like cpu, ram, network or disk i/o is too high. Other failures can not be detected in advance and lead to a defective system.

In case monitoring identifies a problem, the operations team needs to take care of the problem. Operations teams need to be on call and manage incidents. A new Incident must be rated how severe it is. In advance detected problems potentially leading to failures like a disk going to be full in few days, must not be handled immediately. Other incidents like an unavailable database service can be business critical and must be fixed as rapidly as possible. But often the cause of a failure is not obvious and the system must be debugged with specific tools. Critical failures mostly affect users, so another part of incident management is to inform the affected parties.

Google states that the traditional approach of an operations team[1] is not efficient. It is expensive, it does not scale very well and it creates tension in interests[2, p. 3/4].

A growing system means there will be more hardware, more software components and a more complex configuration. More people are needed to install, monitor and maintain the infrastructure and services. The traditional operations team approach has the disadvantage, that it does not scale very well and produces more costs, the bigger the system becomes. Another disadvantage is conflicting interests of the developer team and operation team. Operations teams are payed to create stability and provide a certain quality of service. But change is the biggest source of failure[2, p.10]. That's why operations teams work under the motto of 'Never change a running system'. On the other hand, a good service changes to the needs of its users. Developers get paid to deliver as much features as possible, so they want frequent change. Those are conflicting interests and they harm the whole product. For Google it's clear, that the different assumptions and tension in interests of operations and developer teams produce a lot of hidden costs.

3.1 Site Reliability Engineers maintain systems like software engineers

Operations teams developed a lot of practices to both integrate the fundamentally conflicting interests of change and stability, but also scale the operations team approach. This reduces obvious and hidden costs and makes the whole work more efficient. A lot of those practices became possible, because the infrastructure turned from pure hardware into a dynamic software defined infrastructure. Google refers to this approach as the site reliability engineering approach. It is operations how an software engineer would do it.

New technologies made it possible to define all infrastructure in code. One of the first who provided an dynamic infrastructure is amazon with ec2. The technology of virtualization made provisioning and configuration to software tasks instead of hardware tasks. This is what it made automate-able. Also Docker made it easier to package software and to deploy and configure the services[2].

With the possibility to automate the tasks an operations team has to do, there will come many benefits. The obvious value is that is scalabillity. When a task is automated and can be done by a machine, it is very cheap and easy to execute the task a lot more often. But there is a much bigger advance, which automation gives. When a task is automated it is a well defined process, which will consistently performed, where a human can easily make a mistake by manually executing the task. It can also be extended, measured and done at inconvenient times for humans[3].

The before one off tasks of provisioning, deploying and configuring have now well defined practices and processes. Those tasks are now reproducible because the definition or configuration of the infrastructure can now be stored in a version control system. And with automate-able deployment processes like zero downtime releases, phoenix deployment or rolling updates, those processes can be done continuously.

Operations teams have collected great knowledge on monitoring. What are the key indicators for monitoring a system. What does a good monitoring infrastructure provide. Those questions I am going to examine in detail in the upcoming sections. But very important is that monitoring can notify a human

[1]Google actually calls it the sysadmins approach
[2]See Chapter 2 'Platforms' in [4] for those technologies
[3]See Chapter 'Value of Automation' in [2] for complete discussion of the advantages.

or even better trigger automated tasks, that humans do not need to be involved at all, when a problem occurs.

Also incident management can partly be automated. Distributed systems like databases have nowadays automatic failovers. Issues which have been serious, major incidents before and a humans had to manually intervene can, are now automatically triggered and the systems heal themselves. Disaster recovery is now easily done, because you can automatically reprovision the whole infrastructure as mentioned before.

With those practices it means that in the end this means that the conflicting interests of change and stability can nowadays be more and more integrated.

3.2 Monitoring to identify Problems

3.2.1 Health checks measure availability

3.2.2 Measuring Latency, Traffic, Errors and Saturation identifies failures and performance problems

3.2.3 Incident Management (/Notifications) for appropriate and fast actions in case of Problems

CHAPTER 4

METRICS MEASURE TEAM EFFICIENCY AND SOFTWARE QUALITY

4.1 Velocity and cycletime are efficiency metrics for an agile team

cycle time measures quality of delivery engine

4.1.1 Deploys/Week indirectly measures velocity

and it measures the effect of a quality delivery engine we want many deploys per week

4.1.2 Deploy Duration is import for cycle time

4.2 MTTR and Failurerate measure the quality of a software

we want low risk per deploy to achieve MTTR and low Failure Rate

4.2.1 LOCS/Deploy indirectly measures the risk per deploy

PART II

NEW PRACTICES

CHAPTER 5

POST RELEASE TESTING EXTENDS THE CONTINUOUS DELIVERY PIPELINE TO SUPPORT MAINTAINING A SYSTEM

The three main steps in the continuous delivery pipeline are: commitment to version control, continuous integration and release. After releasing, the new version will be operated. This includes monitoring, logging, security aspects and incident management[1]. To enable developers to take more responsible in running the software, it is necessary to extend the practices of the continuous delivery pipeline.

In my masterthesis I want to optimize the software engineering practices in order to empower developers in their bigger responsibilities. I want to focus on the process of deployment and enhance the continuous delivery pipeline. To achieve this, I want to examine operation practices like monitoring and incident management. And then extend the deployment process by three consecutive steps.

The first step is to completely automate deployments of software features and achieve continuous deployment. The next step is to sensibly monitor new releases to give automatically feedback. Third and lastly, identify incidents to automatically trigger rollbacks to self-heal the software system. I'm going to call these last two steps continuous post release testing.

5.1 Post Release Testing leads to lower time to market

cycle time

[1]In "Site Reliability Engineering" monitoring and notifying principles [2, p. 55-63] are well described.

5.2 It makes Releases consistent, measurable, fast and scalable

mttr, automation/automatic discussion

5.3 It is a new opportunity for risk management

identify test before release is a mttr of zero, after release still fast. easier to test in production (complexity of system)

5.4 Companies are already post release testing their software systems

5.4.1 Netflix uses Simian Army to live test their systems

5.4.2 Synthetic Monitoring tests a complex distributed system

5.5 Post Release Testing with Canaries is appropriate for testing non change

ErrorRate as monitoring measure for automation problems in error rate measure defect and failure solution a secific heuristic

5.5.1 Black-Box monitoring is only one part and monitoring change is difficult

5.5.2 Canary testing is important for maintenance but not feature deploys

5.5.3 Continuous Delivery is a requirement

5.5.4 Notifications in case a canary behaves different

5.5.5 Automated Rollbacks for a automatic self healing system

CHAPTER 6

IMPLEMENTING CANARY POST RELEASE TESTING

6.1 New technologies drive new techniques

6.1.1 Kubernetes is a Cluster OS

6.1.1.1 Resource Management in Kubernetes is made for high available services

6.1.1.2 Deployments implement Rolling Updates The extend to the canary releasing technique is a rolling update. It unifies phoenix replacement and canary releasing and extends it to a full deployment. You start with one server in the new version and spawn it automatically. Now it becomes part of the whole pool of servers and serves traffic as the server is ready. Now two different versions are serving traffic, just like with a canary release. But after the first step the deployment goes on and then destroys a server instance in the old version as you would do in the phoenix replacement. Then the procedure will be done multiple times until all servers of the old version are replaced by the new version.

6.1.2 DataDog is a Cluster Monitoring Systems as a sevice

alternativen Graphana, Prometheus

6.1.2.1 The main components are: Datacollection, Timeseriesdatabase, Graphing and Alerting

6.1.2.2 DataDog integrates well in Kubernetes

6.2 Deployer is a Service for Continuous Deployment and enables Canary Post Release testing

6.2.1 Deployer integrates into the Continuous Delivery Pipeline

6.2.2 Deployer integrates into Kubernetes and deploys itself

6.2.3 Deployer integrates into Monitoring and enables Canary testing

6.2.4 The main Deployer API features are Deployments and Canary deployments

depctl, curl

6.2.4.1 Deployments deploy a whole repository

6.2.4.2 One Canary per Replicated Pods can be deployed and monitored

6.2.4.3 Immediate Notifications in case of failure difference depctl and curl

6.2.4.4 Automatic Rollbacks in case defect Canary via datadog and triggers deployer future: staging deploys

PART III

EVALUATION

CHAPTER 7

GAPFISH IS THE COMPANY TO EVALUATE DEPLOYER

7.1 GapFish's services are complex and highly available

overview of Gapfish's services

7.1.1 GapFish's Operation Service is used by internal Staff and Customers

operation service == operation.gapfish.com

7.2 GapFish uses tools for continuous deployment

7.2.1 GapFish differentiates between development and operation

how is the deployment tradditionally done

7.2.2 Kubernetes enables GapFish to have development and operations in one team

which services are migrated to kubernetes

CHAPTER 8

METRICS ARE IMPLEMENTED AS LOGGING OUTPUT

8.1 Deploys/Week

8.2 Deploy Duration

8.3 LOCS/Deploy

CHAPTER 9

RESULTS

9.1 Metrics: traditional vs. new approach

9.2 theoretical/practical conclusion for deployer and cd

9.3 for Gapfish

9.4 Lessons learned and future

CHAPTER 10

RESUME

Bibliography

[1] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: a software architect's perspective*. Addison-Wesley, 2015.

[2] Betsy Beyer et al. *Site Reliability Engineering: how google runs production systems*. O'Reilly, 2016.

[3] Jez Humble and David Farley. *Continuous Delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley, 2010.

[4] Kief Morris. *Infrastructure as code : managing servers in the cloud*. O'Reilly, 2016.

[5] Eberhard Wolff. *Continuous Delivery: der pragmatische Einstieg*. 2. dpunkt.verlag, 2016.