

Vom IT-Betrieb zu Platform Engineering. Ein Reisebericht (2/3)



Vor einer Woche habe ich berichtet, woher wir gekommen sind, welche Probleme wir hatten und wie wir mit Veränderungen wie z.B. der Einführung einer [SOA-Architektur](#) umgegangen sind.

Wir hatten schon einen wichtigen Schritt gemacht, indem wir unsere Arbeit sichtbar gemacht haben. Dadurch konnten wir priorisieren. Wir haben bewusst Themen abgemeldet bzw. erst für später eingeplant, um wiederum andere Themen mit einer höheren Priorität früher zu machen.

Wir hatten aber immer noch Probleme. Die Ticketanzahl stieg, wir waren immer noch zu langsam, und von Weiterentwicklung wollen wir gar nicht reden. Die Reise war also noch nicht zu Ende...

Kapitel 2: Basistechnologien, SelfServices & Automation

Wir wussten, dass wir nicht schnell genug waren. Die Anzahl der Tickets stieg

und acht Wochen Vorlauf für einen neuen Service in Produktion klingt irgendwie nicht zeitgemäß, oder? An welchen Schrauben sollten wir drehen? Die Teams, die wir unterstützen sollen, warteten zu oft auf uns. Immer wieder kam z.B. die Aussage, dass ein Server in der Cloud in wenigen Minuten verfügbar ist. Wir als Team waren mit operativer Arbeit überlastet und hatten nur wenig Zeit für Weiterentwicklung. Irgendwie mussten wir im Tagesgeschäft schneller werden, um mehr Zeit für die Weiterentwicklungen unserer eigenen Themen zu bekommen. Wenn wir einen Mehrwert bieten wollen, dann müssen wir uns mit neuen Technologien auseinandersetzen. Nicht zuletzt machen neue Technologien und Weiterentwicklung auch mehr Spaß als das hoch standardisierte und wiederholte Einrichten von neuen Servern, Monitoring, Datensicherung, Logging,

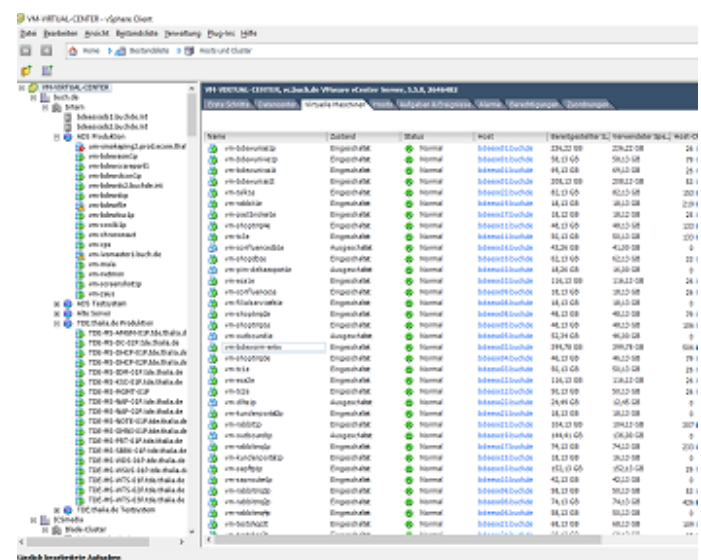


Was fehlte uns, um schneller zu werden? Wir hatten das Glück, dass wir die Chance hatten, Einblicke in die Arbeitsweise und das Mindset von Otto.de zu bekommen (vielen Dank und viele Grüße an die Kollegen von Otto.de!). Ein Satz ist mir besonders hängen geblieben, weil ich ihn anfangs erschreckend schlimm fand. Kurz zusammengefasst: Kommunikation macht langsam! Das meinten die Jungs jetzt nicht ernst, oder? Gute Kommunikation ist doch die Basis! Klar habe ich es erst missverstanden. Gute Kommunikation ist natürlich sehr wichtig. Gemeint war damit die Idee, dass **wenn zwei Teams miteinander reden müssen** bzw. Themen ein Team verlassen, im nächsten Team bearbeitet werden und dann wieder zurück müssen, dort geprüft werden, Fehler gefunden, wieder zurück an das andere Team zur Nachbesserung, zurück zum Test, Kommt das jemandem bekannt vor? Klingt das nach Topperformance? Solche Kommunikation an Schnittstellen macht einfach langsam. Versuche, Schnittstellen zu reduzieren. Versetze den Anforderer in die Lage, seine Anforderung selber umzusetzen, ohne Spezialisten-KnowHow zu haben. **Biete SelfServices an!** Ein SelfService basiert dabei auf einem Automaten mit einem einfachen Frontend. Über das Frontend ist

der Anforderer in der Lage, seine wiederkehrenden Aufgaben einzugeben. Der Automat erledigt dann die hoch standardisierte Umsetzung der Aufgabe in einer fehlerfreieren Qualität als ein Mensch es könnte.

Nur wo fange ich an? Leichter gesagt als getan. Wir haben uns daran orientiert, wo wir als IT-Betrieb die größten Schmerzen hatten und wo wir den neuen Produkt-Teams den größten Nutzen bringen konnten. Hier drei technische Beispiele, die uns nach vorne gebracht haben.

Beispiel 1: Automatisierte Service Bereitstellung (ASB)



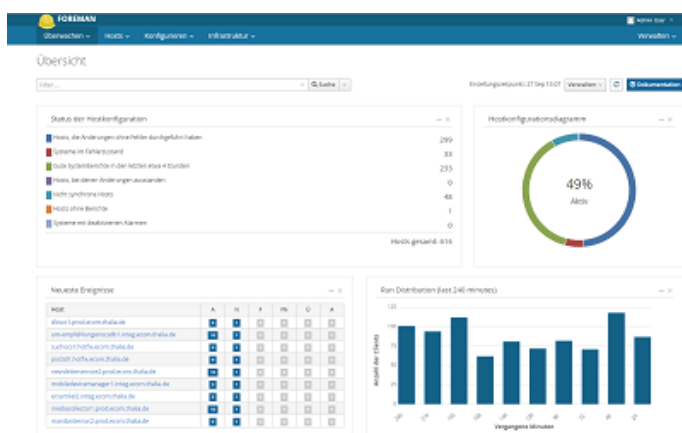
| Name | Status | Power | Host | Storage | Memory | CPUs |
|---------------|---------|--------|------------|------------|------------|------------|
| vm-1000000000 | Engaged | Normal | 1000000000 | 1000000000 | 1000000000 | 1000000000 |
| vm-1000000001 | Engaged | Normal | 1000000001 | 1000000001 | 1000000001 | 1000000001 |
| vm-1000000002 | Engaged | Normal | 1000000002 | 1000000002 | 1000000002 | 1000000002 |
| vm-1000000003 | Engaged | Normal | 1000000003 | 1000000003 | 1000000003 | 1000000003 |
| vm-1000000004 | Engaged | Normal | 1000000004 | 1000000004 | 1000000004 | 1000000004 |
| vm-1000000005 | Engaged | Normal | 1000000005 | 1000000005 | 1000000005 | 1000000005 |
| vm-1000000006 | Engaged | Normal | 1000000006 | 1000000006 | 1000000006 | 1000000006 |
| vm-1000000007 | Engaged | Normal | 1000000007 | 1000000007 | 1000000007 | 1000000007 |
| vm-1000000008 | Engaged | Normal | 1000000008 | 1000000008 | 1000000008 | 1000000008 |
| vm-1000000009 | Engaged | Normal | 1000000009 | 1000000009 | 1000000009 | 1000000009 |
| vm-1000000010 | Engaged | Normal | 1000000010 | 1000000010 | 1000000010 | 1000000010 |
| vm-1000000011 | Engaged | Normal | 1000000011 | 1000000011 | 1000000011 | 1000000011 |
| vm-1000000012 | Engaged | Normal | 1000000012 | 1000000012 | 1000000012 | 1000000012 |
| vm-1000000013 | Engaged | Normal | 1000000013 | 1000000013 | 1000000013 | 1000000013 |
| vm-1000000014 | Engaged | Normal | 1000000014 | 1000000014 | 1000000014 | 1000000014 |
| vm-1000000015 | Engaged | Normal | 1000000015 | 1000000015 | 1000000015 | 1000000015 |
| vm-1000000016 | Engaged | Normal | 1000000016 | 1000000016 | 1000000016 | 1000000016 |
| vm-1000000017 | Engaged | Normal | 1000000017 | 1000000017 | 1000000017 | 1000000017 |
| vm-1000000018 | Engaged | Normal | 1000000018 | 1000000018 | 1000000018 | 1000000018 |
| vm-1000000019 | Engaged | Normal | 1000000019 | 1000000019 | 1000000019 | 1000000019 |
| vm-1000000020 | Engaged | Normal | 1000000020 | 1000000020 | 1000000020 | 1000000020 |
| vm-1000000021 | Engaged | Normal | 1000000021 | 1000000021 | 1000000021 | 1000000021 |
| vm-1000000022 | Engaged | Normal | 1000000022 | 1000000022 | 1000000022 | 1000000022 |
| vm-1000000023 | Engaged | Normal | 1000000023 | 1000000023 | 1000000023 | 1000000023 |
| vm-1000000024 | Engaged | Normal | 1000000024 | 1000000024 | 1000000024 | 1000000024 |
| vm-1000000025 | Engaged | Normal | 1000000025 | 1000000025 | 1000000025 | 1000000025 |
| vm-1000000026 | Engaged | Normal | 1000000026 | 1000000026 | 1000000026 | 1000000026 |
| vm-1000000027 | Engaged | Normal | 1000000027 | 1000000027 | 1000000027 | 1000000027 |
| vm-1000000028 | Engaged | Normal | 1000000028 | 1000000028 | 1000000028 | 1000000028 |
| vm-1000000029 | Engaged | Normal | 1000000029 | 1000000029 | 1000000029 | 1000000029 |
| vm-1000000030 | Engaged | Normal | 1000000030 | 1000000030 | 1000000030 | 1000000030 |
| vm-1000000031 | Engaged | Normal | 1000000031 | 1000000031 | 1000000031 | 1000000031 |
| vm-1000000032 | Engaged | Normal | 1000000032 | 1000000032 | 1000000032 | 1000000032 |
| vm-1000000033 | Engaged | Normal | 1000000033 | 1000000033 | 1000000033 | 1000000033 |
| vm-1000000034 | Engaged | Normal | 1000000034 | 1000000034 | 1000000034 | 1000000034 |
| vm-1000000035 | Engaged | Normal | 1000000035 | 1000000035 | 1000000035 | 1000000035 |
| vm-1000000036 | Engaged | Normal | 1000000036 | 1000000036 | 1000000036 | 1000000036 |
| vm-1000000037 | Engaged | Normal | 1000000037 | 1000000037 | 1000000037 | 1000000037 |
| vm-1000000038 | Engaged | Normal | 1000000038 | 1000000038 | 1000000038 | 1000000038 |
| vm-1000000039 | Engaged | Normal | 1000000039 | 1000000039 | 1000000039 | 1000000039 |
| vm-1000000040 | Engaged | Normal | 1000000040 | 1000000040 | 1000000040 | 1000000040 |
| vm-1000000041 | Engaged | Normal | 1000000041 | 1000000041 | 1000000041 | 1000000041 |
| vm-1000000042 | Engaged | Normal | 1000000042 | 1000000042 | 1000000042 | 1000000042 |
| vm-1000000043 | Engaged | Normal | 1000000043 | 1000000043 | 1000000043 | 1000000043 |
| vm-1000000044 | Engaged | Normal | 1000000044 | 1000000044 | 1000000044 | 1000000044 |
| vm-1000000045 | Engaged | Normal | 1000000045 | 1000000045 | 1000000045 | 1000000045 |
| vm-1000000046 | Engaged | Normal | 1000000046 | 1000000046 | 1000000046 | 1000000046 |
| vm-1000000047 | Engaged | Normal | 1000000047 | 1000000047 | 1000000047 | 1000000047 |
| vm-1000000048 | Engaged | Normal | 1000000048 | 1000000048 | 1000000048 | 1000000048 |
| vm-1000000049 | Engaged | Normal | 1000000049 | 1000000049 | 1000000049 | 1000000049 |
| vm-1000000050 | Engaged | Normal | 1000000050 | 1000000050 | 1000000050 | 1000000050 |
| vm-1000000051 | Engaged | Normal | 1000000051 | 1000000051 | 1000000051 | 1000000051 |
| vm-1000000052 | Engaged | Normal | 1000000052 | 1000000052 | 1000000052 | 1000000052 |
| vm-1000000053 | Engaged | Normal | 1000000053 | 1000000053 | 1000000053 | 1000000053 |
| vm-1000000054 | Engaged | Normal | 1000000054 | 1000000054 | 1000000054 | 1000000054 |
| vm-1000000055 | Engaged | Normal | 1000000055 | 1000000055 | 1000000055 | 1000000055 |
| vm-1000000056 | Engaged | Normal | 1000000056 | 1000000056 | 1000000056 | 1000000056 |
| vm-1000000057 | Engaged | Normal | 1000000057 | 1000000057 | 1000000057 | 1000000057 |
| vm-1000000058 | Engaged | Normal | 1000000058 | 1000000058 | 1000000058 | 1000000058 |
| vm-1000000059 | Engaged | Normal | 1000000059 | 1000000059 | 1000000059 | 1000000059 |
| vm-1000000060 | Engaged | Normal | 1000000060 | 1000000060 | 1000000060 | 1000000060 |
| vm-1000000061 | Engaged | Normal | 1000000061 | 1000000061 | 1000000061 | 1000000061 |
| vm-1000000062 | Engaged | Normal | 1000000062 | 1000000062 | 1000000062 | 1000000062 |
| vm-1000000063 | Engaged | Normal | 1000000063 | 1000000063 | 1000000063 | 1000000063 |
| vm-1000000064 | Engaged | Normal | 1000000064 | 1000000064 | 1000000064 | 1000000064 |
| vm-1000000065 | Engaged | Normal | 1000000065 | 1000000065 | 1000000065 | 1000000065 |
| vm-1000000066 | Engaged | Normal | 1000000066 | 1000000066 | 1000000066 | 1000000066 |
| vm-1000000067 | Engaged | Normal | 1000000067 | 1000000067 | 1000000067 | 1000000067 |
| vm-1000000068 | Engaged | Normal | 1000000068 | 1000000068 | 1000000068 | 1000000068 |
| vm-1000000069 | Engaged | Normal | 1000000069 | 1000000069 | 1000000069 | 1000000069 |
| vm-1000000070 | Engaged | Normal | 1000000070 | 1000000070 | 1000000070 | 1000000070 |
| vm-1000000071 | Engaged | Normal | 1000000071 | 1000000071 | 1000000071 | 1000000071 |
| vm-1000000072 | Engaged | Normal | 1000000072 | 1000000072 | 1000000072 | 1000000072 |
| vm-1000000073 | Engaged | Normal | 1000000073 | 1000000073 | 1000000073 | 1000000073 |
| vm-1000000074 | Engaged | Normal | 1000000074 | 1000000074 | 1000000074 | 1000000074 |
| vm-1000000075 | Engaged | Normal | 1000000075 | 1000000075 | 1000000075 | 1000000075 |
| vm-1000000076 | Engaged | Normal | 1000000076 | 1000000076 | 1000000076 | 1000000076 |
| vm-1000000077 | Engaged | Normal | 1000000077 | 1000000077 | 1000000077 | 1000000077 |
| vm-1000000078 | Engaged | Normal | 1000000078 | 1000000078 | 1000000078 | 1000000078 |
| vm-1000000079 | Engaged | Normal | 1000000079 | 1000000079 | 1000000079 | 1000000079 |
| vm-1000000080 | Engaged | Normal | 1000000080 | 1000000080 | 1000000080 | 1000000080 |
| vm-1000000081 | Engaged | Normal | 1000000081 | 1000000081 | 1000000081 | 1000000081 |
| vm-1000000082 | Engaged | Normal | 1000000082 | 1000000082 | 1000000082 | 1000000082 |
| vm-1000000083 | Engaged | Normal | 1000000083 | 1000000083 | 1000000083 | 1000000083 |
| vm-1000000084 | Engaged | Normal | 1000000084 | 1000000084 | 1000000084 | 1000000084 |
| vm-1000000085 | Engaged | Normal | 1000000085 | 1000000085 | 1000000085 | 1000000085 |
| vm-1000000086 | Engaged | Normal | 1000000086 | 1000000086 | 1000000086 | 1000000086 |
| vm-1000000087 | Engaged | Normal | 1000000087 | 1000000087 | 1000000087 | 1000000087 |
| vm-1000000088 | Engaged | Normal | 1000000088 | 1000000088 | 1000000088 | 1000000088 |
| vm-1000000089 | Engaged | Normal | 1000000089 | 1000000089 | 1000000089 | 1000000089 |
| vm-1000000090 | Engaged | Normal | 1000000090 | 1000000090 | 1000000090 | 1000000090 |
| vm-1000000091 | Engaged | Normal | 1000000091 | 1000000091 | 1000000091 | 1000000091 |
| vm-1000000092 | Engaged | Normal | 1000000092 | 1000000092 | 1000000092 | 1000000092 |
| vm-1000000093 | Engaged | Normal | 1000000093 | 1000000093 | 1000000093 | 1000000093 |
| vm-1000000094 | Engaged | Normal | 1000000094 | 1000000094 | 1000000094 | 1000000094 |
| vm-1000000095 | Engaged | Normal | 1000000095 | 1000000095 | 1000000095 | 1000000095 |
| vm-1000000096 | Engaged | Normal | 1000000096 | 1000000096 | 1000000096 | 1000000096 |
| vm-1000000097 | Engaged | Normal | 1000000097 | 1000000097 | 1000000097 | 1000000097 |
| vm-1000000098 | Engaged | Normal | 1000000098 | 1000000098 | 1000000098 | 1000000098 |
| vm-1000000099 | Engaged | Normal | 1000000099 | 1000000099 | 1000000099 | 1000000099 |
| vm-1000000100 | Engaged | Normal | 1000000100 | 1000000100 | 1000000100 | 1000000100 |

Vorher: Server klonen mit VMware

Wie ihr wisst, haben wir acht Wochen Vorlauf für einen Service in Produktion benötigt. Das muss natürlich auch schneller mit weniger Vorlauf gehen. Warum haben wir acht Wochen Vorlauf gebraucht? Primär aufgrund der Abstimmung zwischen IT-Betrieb und Entwicklung. Von den acht Wochen haben wir drei Wochen für die Abstimmung eingeplant. Eine Woche ist für den Bau der gesamten Infrastruktur eingeplant worden, so dass nach vier Wochen die Infrastruktur für den neuen Service bereitstand. Aber warum dann acht Wochen? Vier Wochen vor Produktion startet das erste Testdeployment. Hier muss die Infrastruktur fertig sein, damit die Tests starten können. Auch in der Testphase ist immer wieder aufgefallen, dass die bereitgestellte Infrastruktur in sich nicht immer konsistent war. So waren Server trotz Standard VMware Image gerne mal unterschiedlich konfiguriert, es fehlten Berechtigungen oder Benutzer waren nicht angelegt. Um die Fehler in der Massenbereitstellung zu reduzieren, hilft nur eines wirklich:

Automatisiere! Stelle Services automatisch und ohne Expertenwissen bereit.

Baue einen Automaten mit einem Webfrontend. Gebe dem Entwicklungsteam einen Knopf, auf den man drücken kann und aus dem dann nach wenigen Minuten ein Server rausfällt. Ein Server? Wer will den einen Server haben? Erweitere den Knopf, so dass nach der automatischen Server-Installation (das kann eine public Cloud eh besser) auch gleich noch das Betriebssystem passend für den **Service** konfiguriert wird, dann der Service installiert und konfiguriert wird und in Produktion gestellt wird. Das kann eine public Cloud für unsere Services „out of the box“ nicht -> Cool, Mehrwert generiert. Klingt das gut? ☐



Nachher: Service Bereitstellung auf Foreman & Puppet Basis

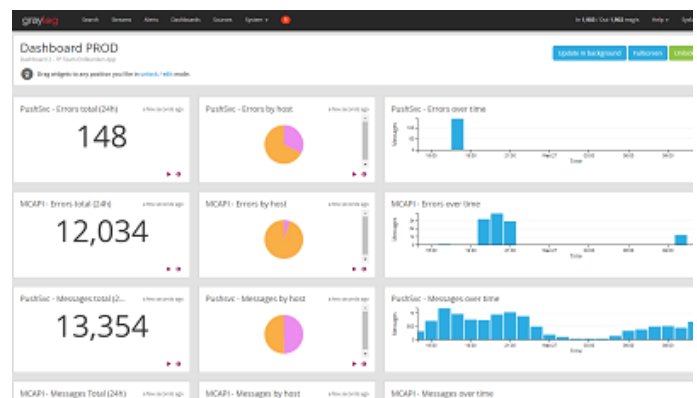
Leute aus den Entwicklungsteams und dem IT-Betrieb haben also damit begonnen, basierend auf Foreman, Puppet und ein paar anderen Helferlein eine automatisierte Service Bereitstellung (Intern kurz: ASB) aufzubauen. Der Fokus lag im ersten Schritt auf dem Basis Betriebssystem und JAVA/Tomcat Services. Davon hatten wir die meisten, und es würde somit den größten Nutzen bringen. In dieser Zeit haben wir auch die Basis-Architektur für den Automatismus definiert und aufgebaut. Darauf basieren bis heute alle neu entwickelten Automatismen z.B. für DNS oder Apache Webserver. Wir haben einen starken Fokus auf den Aufbau der ASB-Infrastruktur gegeben, wodurch sicherlich auch andere Arbeit liegen geblieben ist. Jedoch konnten wir nur so den Entwicklungsteams die Möglichkeit geben, Server inkl. Services ohne Rückfrage beim IT-Betrieb aufzubauen. Egal in welcher Stage. Konsequenz: Der SelfService wird so gut genutzt, dass wir heute über alle Stages weit über 1.000 Server betreiben.

Beispiel 2: Zentrales Logdaten Management (ZLM)

Vorher: Logdaten Analyse auf der Console

OK, ein SelfService, um Services automatisch bereitzustellen, das ist schon mal super. Aber auch nur ein Baustein. Oft ist uns in der Vergangenheit das Logdaten Management System auf ELK-Stack-Basis (ELK = Elastic Search, Logstash und Kibana) unter der Last zusammengebrochen. Das lag weniger an der Technologie „ELK-Stack“ sondern eher an der eigenen Architektur, die, als wir sie vor Jahren initial für IT-Betrieb Systeme designed haben, nicht für die inzwischen vorliegenden Last ausgelegt war. Das verursacht wieder unfrohliche Stimmung in den Entwicklungsteams, weil wichtige Daten fehlen und das System nicht verfügbar war. Gleichzeitig hat der IT-Betrieb wieder ein neues, ungeplantes, dafür dringendes Ticket, um das alte System mit viel Klebeband wieder ans Laufen zu bringen. OK, lass deine Arbeit *jetzt* liegen, kümmer dich (schon wieder) um einen Störfall im Logdaten Management. Macht weder glücklich noch bringt uns die Störung nach vorne. Von schneller werden reden wir hier auch nicht. Aber es

gibt ja nicht nur Störungen. SOA und Microservice sei Dank gibt es ja fast täglich neue Services, die auch in das Log-Management möchten. Mal abgesehen von den Kapazitäten im Logdaten Management System beginnt das traditionelle Spiel im Ticketsystem: Neues Ticket: „Ich benötige Logging“, klar baue ich dir gerne, so fertig, hmm ich sehe nichts, ups jetzt aber, OK funktioniert, aber warum kann ich nicht mehr Daten ins System geben, aus den Büchern der menschlichen Schnittstellenkommunikation □ . Wie kann ich solche Kommunikation vermeiden? Baue einen **SelfService**! Plane und implementiere eine Logdaten Management Infrastruktur, die leistungsstark genug ist, um dein Volumen locker zu handhaben und wenig störanfällig ist. Sorge dafür, dass die Infrastruktur skaliert (nach oben und nach unten). Stelle eine gut dokumentierte Schnittstelle bereit, über die ein Entwicklungsteam selber seine Daten in das Logdaten Management schreiben kann und die Möglichkeit hat, die Daten selber auszuwerten.



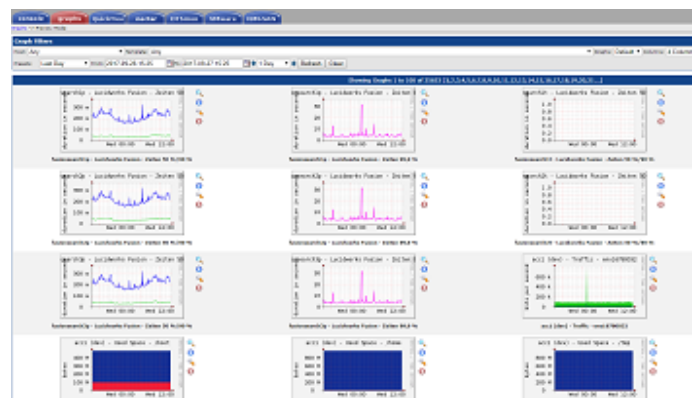
Nachher: Logdaten Management auf Graylog Basis

Auch den Aufbau eines neuen und leistungsfähigen Logdaten Managements konnten wir recht früh starten. Auch hier setzen wir auf das bewährte Prinzip, Entwicklerteams und IT-Betrieb zusammenzubringen. Schnell wurde aber klar, dass es nicht besonders schlau ist, alle Logdaten, die es gibt, ohne Sinn und Verstand in eine Datenbank zu pressen. Das ist zwar möglich, aber nicht schlau. So haben wir uns u.a. mit den Fragen auseinandergesetzt „Was und wie loggen wir überhaupt?“, „Wann brauchen wir eine Logausgabe?“ und „Was machen die Logdaten?“. Nach vielen Gesprächen über Anforderungen und Optionen haben wir einen Vorschlag erarbeitet, wie die Entwicklungsteams Logging in ihren Services nutzen können. Parallel dazu haben wir eine neue Logdaten Management Infrastruktur basierend auf Graylog aufgebaut. Graylog bietet

zusätzlich zum ELK-Stack noch ein paar nette Features wie z.B. ein User Management. Auch erschienen uns das Handling und der Betrieb im Vergleich zum ELK-Stack etwas einfacher. Gesized wurde die neue Infrastruktur für Spitzenlastzeiten. Bei uns ist das neben dem Schulbuchgeschäft natürlich das Weihnachtsgeschäft. So kamen wir auf eine Infrastruktur von u.a. zwei Elastic Search Master Nodes und acht Elastic Search Data Nodes in zwei Datacentern. Durch die Data Nodes ist die Infrastruktur jederzeit ohne viel Aufwand skalierbar. Im Peak kann die Infrastruktur bis zu 100.000 Messages die Sekunde verarbeiten. Um die künftigen Nutzer der Infrastruktur zu informieren und möglichst früh Feedback zu bekommen, haben wir zum Thema eine von uns so getaufte „Bier-Session“ angesetzt. Eine „Bier-Session“ ist in etwa vergleichbar mit einer Brownbag-Session, jedoch war es nicht in der Pause und es gab (alkoholfreies) Freibier. Das Feedback haben wir gerne aufgenommen und in die Infrastruktur eingebaut.

Was haben wir mit der neuen Infrastruktur erreicht? Die Produkt-Teams können nun ihre eigenen Anwendungen durch eine einfache Konfiguration in der „logback.xml“ anbinden. Auch die Dashboards zur Auswertung der Logdaten können sich die Teams ohne Unterstützung des Platform Engineering Teams einrichten. Ein weiterer schöner SelfService, der die Aufwände im IT-Betrieb reduziert und die Arbeit in den Produkt Teams beschleunigt.

Beispiel 3: Automatisiertes Monitoring



Vorher: CACTI Monitoring hat seinen Dienst getan

Monitoring war auch so eine super Sache. Wir monitoren ja echt viel – mit einigen

unterschiedlichen Monitoring- & Alarmierungssystemen bekommen wir einen super Überblick über den Zustand unserer Services aus technischer Sicht und aus Kundensicht. Jedoch mit was für einem Aufwand! Sämtliche Sensoren wurden von einem Menschen (immerhin haben wir für das Monitoring einen dedizierten Menschen) manuell in die Monitoring Systeme eingetragen. Also erstellt das Entwicklungsteam ein Ticket für das Monitoring, da fehlen aber noch Information im Ticket, wieso fehlen da noch Information, ich erkläre es dir, jetzt habe ich ein Ticket mit allen Information, die Sensoren wurden eingebaut, da fehlt aber noch ein Sensor, OK, warum hast du das nicht gleich gesagt, nun ist der Sensor auch eingebaut, aber wo ist der Aktionsplan für die Alarmierung, der kommt später, Hatten wir das mit den Schnittstellen nicht schon mal? Gibt es eine Lösungsidee? Klar: Bau eine **SelfService** Schnittstelle! Im gleichen Zuge, in dem wir die SelfService Schnittstelle gebaut haben, haben wir auch unser solides, jedoch in die Tage gekommenes CACTI durch eine moderne Lösung abgelöst. Von der neuen Lösung versprochen wir uns mehr Möglichkeiten, Dashboards, Graphen, Informationen etc. bedarfsorientiert zu visualisieren, um dadurch schneller und einfacher entstören und planen zu können.



Nachher: Performance Monitoring mit InfluxDB & Grafana

Dank der frühen Planung und der Rahmenbedingungen, die geschaffen wurden, konnten wir auch damit beginnen, einen weiteren SelfService für das automatisierte Monitoring zu bauen. Möglich wurde das u.a. dadurch, dass wir uns externe Unterstützung ins Team geholt haben, die uns den Rücken im Tagesgeschäft frei gehalten hat, so dass die internen Mitarbeiter sich um die Entwicklung der neuen Technologie kümmern konnten. Das klingt alles super und hat uns sehr geholfen, ein Überschuss an Mitarbeitern war aber dennoch nicht zu erkennen. Wir haben einige Themen parallel bearbeitet. Doch wir haben das

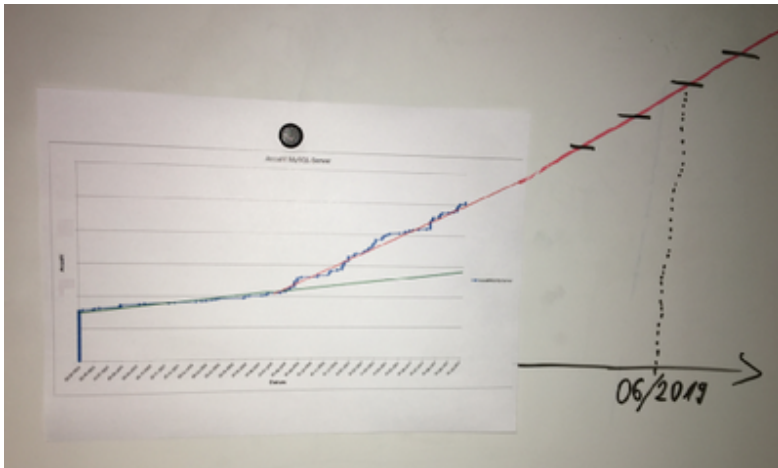
Beste daraus gemacht. Und so kam es z.B., dass wir u.a. einen unserer Datenbank Spezialisten überzeugen konnten, zusammen mit einem Monitoring-Spezialisten und Software-Entwicklern ein neues Performance Monitoring System basierend auf InfluxDB, Telegraph und Grafana zu bauen. Die Arbeiten wurden stark unterstützt von Produkt-Teams, die ihre Ideen und Anforderungen eingebracht haben. Nachdem der erste brauchbare Prototyp stand, haben wir vor der Pilotphase eine weitere „Bier-Session“ durchgeführt. Die Session hat allen Nutzern einen frühen Blick auf das künftige System gegeben und uns gutes Feedback gegeben, welches wir einarbeiten konnten.

Ein neues Performance Monitoring hilft aber nur in Teilen weiter. Es ist leistungsstark, voller Features und kann ohne Hilfe des IT-Betriebs genutzt werden. Jedoch fehlt noch eine Lösung für unser Alarmierungssystem und dessen Aktionspläne. Daher haben wir zusätzlich zusammen mit Software Entwicklern, Qualitätstestern und Administratoren einen komplett neuen Service gebaut, der es erlaubt, aus Informationen aus einer YAML-Datei die Sensoren in der Alarmierung sowie dem Performance Monitoring zu erstellen und zu konfigurieren. Dadurch haben wir den Entwicklungsteams eine Schnittstelle gegeben, die sie aus ihrem Tagesgeschäft kennen, haben die Monitoring Konfiguration zu Code gemacht und versioniert, die manuellen Aufwände reduziert und die Qualität erhöht. Klingt nach einem coolen SelfService? Ist er auch! ☐

Was haben wir mit den Änderungen an den Basistechnologien erreicht?

Wir haben eine Menge Arbeit in die Bereitstellung von SelfServices und Automaten investiert. Dadurch haben wir es geschafft, die aufwändige Abstimmung zwischen Entwicklungsteams und IT-Betrieb zu reduzieren. Wir haben durch Automation und technische Schnittstellen die Qualität in der Umsetzung von Anforderungen erhöht und den Bedarf für Nacharbeiten reduziert. Dadurch haben wir einen wichtigen Betrag zur Beschleunigung der Entwicklungsteams geleistet. Die Aufwände für wiederkehrende Arbeit wurden im IT-Betrieb reduziert, wodurch mehr Zeit für Weiterentwicklung entstanden ist.

Gibt es auch Schattenseiten?



Wachstum DB-Systeme der letzten 2 Jahre inkl.
„Prognose“

Natürlich ist nicht immer alles rosarot, und auch wir haben durch Schmerzen gelernt. So wird z.B. unser SelfService zur automatischen Service Bereitstellung sehr gerne und viel genutzt, was erst einmal sehr positiv ist. Hier ist ein Beispiel, wie sich die Anzahl unserer Datenbanksysteme über die letzten 2 Jahre geändert hat und welches Wachstum wir noch erwarten. Ratet einfach mal wann die automatische Service Bereitstellung verfügbar war □ Eines der verbundenen Probleme ist, dass die Anforderung an alle Komponenten der Plattform (Beispiel: Storage, CPU, RAM, Virtualisierung, ...) deutlich gestiegen sind. Auch die Anforderungen an die für den Betrieb notwendigen Systeme (Beispiel: Datensicherung, Monitoring, Logging, ...) sind davon nicht ausgenommen. Natürlich haben wir mit einem spürbaren Anstieg der Systeme und somit auch mit den Anforderungen gerechnet. Dennoch wurden wir ein Stück vom Erfolg überrascht. Fehlende Ressourcen in der Plattform haben zu Störungen in den Services geführt, was wir nachträglich im laufenden Betrieb mit Schmerzen in allen Teams korrigieren mussten. Auch die steigenden Kosten sowie die Kostenkontrolle für Plattform Ressourcen müssen bei der Einführung und dem Ausbau von SelfServices berücksichtigt werden. Aktuell verrechnen wir die Kosten nicht an die Teams, jedoch stellen wir über Reporting sicher, dass jedes Team weiß, was ein System kostet und welche Kosten für Plattform Ressourcen ein Team produziert.

Kapitel 3: Mindset, Methoden, Prozesse und mehr...

In einer Woche werde ich im dritten und vorerst letzten Kapitel unserer Reise darüber berichten, dass neben der Technologie auch kulturelle und methodische Änderungen notwendig waren. Z.B. war unser Mindset das eines klassischen IT-Betriebs. Das Mindset unserer wichtigsten Kunden, der Entwicklungsteams, war jedoch das einer agilen Software-Entwicklung. Und was ist eigentlich dieses DevOps? Hier gab es also noch etwas zu tun. Abgerundet wird das dritte Kapitel durch ein Fazit, was wir gelernt haben. To be continued ...

Alle drei Kapitel im Überblick



[Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf](#)



[Kapitel 2: Basistechnologien, SelfServices & Automation](#)



[Kapitel 3: Mindset, Methoden, Prozesse und mehr...](#)

Vom IT-Betrieb zu Platform Engineering. Ein Reisebericht (1/3)



Es gibt einige Artikel auf unserem Tech Blog, die über unseren Wandel zur Produkt-Organisation berichten. Auch wenn der IT-Betrieb bei solchen Aktivitäten gerne mal vergessen wird, so war es bei unserem Wandel zur Produkt-Organisation nicht der Fall. Als Learning aus anderen Umstellungen und Unternehmen wurde der IT-Betrieb sehr früh in den Prozess einbezogen. Nach gut einem Jahr kann man sagen: Das hat sich bezahlt gemacht!

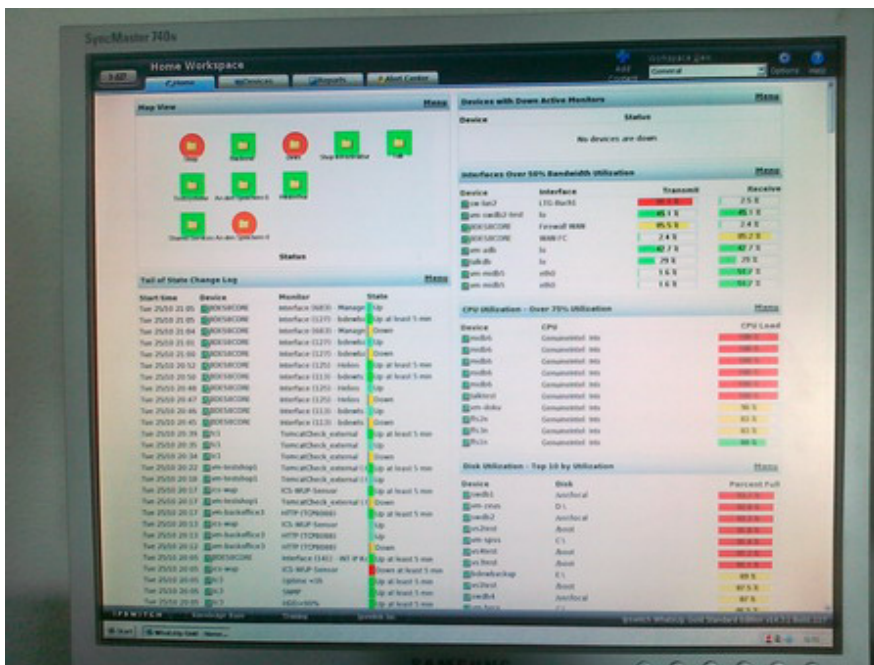
Die folgenden Zeilen sind ein Reisebericht aus dem Blickwinkel von IT-Operations. Woher kommen wir? Warum wollten wir uns verändern? Wie haben wir das gemacht? Was haben wir dabei gelernt? Diese kleine Artikelserien in drei Teilen gibt einen Überblick über die Themenblöcke, deren Hintergründe und Inhalte. Ich gehe dabei jedoch nicht auf jedes einzelne Detail ein. Eventuell folgen diese ebenfalls sehr spannenden Details in späteren Artikeln.

Kapitel 1: Woher kommen wir? 6 Jahre im

Schnelldurchlauf

Die Reise beginnt

Vor rund sieben Jahren haben wir nach und nach den IT-Betrieb aufgebaut. Dazu haben wir über die Zeit Datenbank-, Linux- und Windows-Spezialisten an Bord geholt und zu einem Team formiert. Eines war dabei immer klar: Bleib nah an der eigenen Entwicklung, die unsere eCommerce Software baut. Da die gesamte IT damals noch eine überschaubare Größe hatte und auf einem Flur saß, war das auch recht einfach. Genau wie heute war bereits damals die Stimmung sehr positiv, der Zusammenhalt und die gegenseitige Unterstützung sehr groß.



Monitoring & Alarmierung (2010)

Im Jahr 2010 hatten wir für thalia.de/ch/at lediglich rund 100 Server, die große Monolithen und einige wenige Services beherbergten. Eine vergleichsweise überschaubare Systemlandschaft mit Linux, Tomcat, JAVA, Apache, MySQL. Schon damals hatten wir kein eigenes Datacenter, sondern haben Datacenter as a Service (DCaaS) genutzt. Wir waren nicht darauf aus, unsere Zeit primär mit einem Schraubendreher im Rechenzentrum zu verbringen. Uns war immer klar, dass wir als Team den größten Mehrwert nah an den vom Kunden genutzten Services bringen können. Server in ein Rack einschrauben und Betriebssysteme installieren können viele Dienstleister und Cloudanbieter besser als wir.

Unsere Aufgabe war es, den Betrieb der Systeme und der Services rund um die Uhr sicherzustellen. In guter IT-Betriebs-Tradition haben wir durch Prozesse, Standardisierung und Professionalisierung einen guten Beitrag zur Verbesserung der Verfügbarkeit unseres Gesamtsystems und somit zum Umsatz geleistet. Auch wenn längst nicht alles perfekt war, so machten wir doch gute Fortschritte, und wir wurden stetig besser – was man u.a. auch an der Verfügbarkeit der Shop Systeme erkennen konnte.

Insourcing der Entwicklung & SOA-Architektur

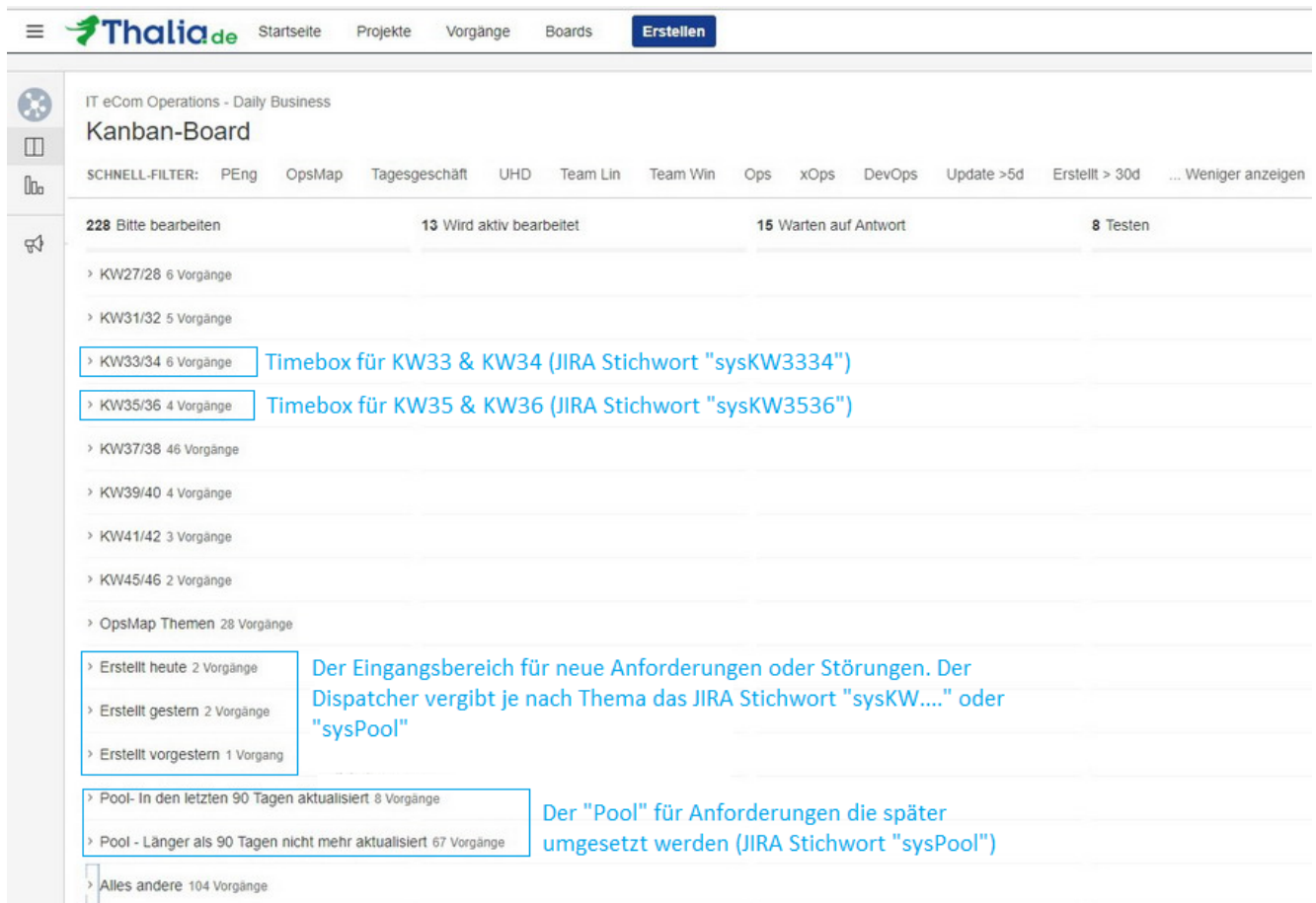
Doch nicht nur der Betrieb entwickelte sich weiter. Auch die Software-Entwicklung machte große und tolle Fortschritte. Damals wurde Software für unser eCommerce System zu einem großen Teil extern entwickelt. Diese Entwicklung wurde ins Haus geholt (hmm, das klingt gerade leichter als es wirklich ist – bestimmt schiebt einer der Kollegen mal einen Artikel dazu ☐). In diesem Zusammenhang wurde auch das Deployment, welches zuvor vom externen Dienstleister durchgeführt wurde, in den noch jungen IT-Betrieb übergeben. Ungefähr zur gleichen Zeit hatte die Software-Entwicklung eine echt gute Idee: [SOA-Architektur](#)! Tötet die Monolithen! Nach einem kurzen Termin mit unserem Software-Architekten, der die Idee vorstellte, sagten wir völlig ahnungslos dafür aber um so selbstbewusster: Klar, kein Problem! Und so kam was kommen musste. Der IT-Betrieb wurde von Anforderungen überrollt und konnte die Software-Entwicklung nicht bedarfsgerecht bedienen. Hinter Prozessen und dem Ticketsystem haben wir versucht, in Deckung zu gehen und die Anforderungen zu verstehen, zu priorisieren und sequenziell abzuarbeiten. Waren wir damals schnell? Nun ja, es gab da durchaus Verbesserungspotential.



Die gesamte Schnittstelle zwischen der Entwicklung und dem IT-Betrieb hatte einen massiven Overhead und verursachte hohe Reibungsverluste. Klar waren wir (Entwicklung & IT-Betrieb) immer noch gute Kollegen, haben uns geholfen wo es ging und haben freudig (inzwischen auf zwei Etagen) zusammengearbeitet. Events unterschiedlichster Art wie z.B. den Sysadminday haben wir gerne gefeiert und machen das immer noch. Der Stresspegel auf beiden Seiten stieg jedoch permanent an. Jeder im IT-Betrieb hatte im Ticket System seine 100 Vorgänge und mehr zu tun als er schaffen konnte. Transparenz für den Anforderer und für den Bearbeiter gab es trotz Ticket System jedoch schon lange nicht mehr. Klares Priorisieren der Masse war nahezu unmöglich. Zeit für eine Veränderung!

IT-Operations meets Kanban

Überrollt von Anforderungen aus der Entwicklung sowie aus dem IT-Betrieb mussten wir mehr Struktur und Transparenz in unsere Arbeit bringen. Wir waren ein klassischer IT-Betrieb und hörten zum ersten Mal etwas von „Kanban“. Hmm, was ist das? Was kann das? Sieht erst einmal spannend aus. Da wir damals schon JIRA mit dem AGILE Plugin im Einsatz hatten, haben wir einfach mal unsere Tickets in einem Kanban Board anzeigen lassen. Toll, 500 Tickets auf einem Board. Das soll jetzt helfen? Es hat viele Anläufe gebraucht, bis wir es tatsächlich geschafft haben, ein Kanban Board zu entwickeln, welches uns als IT-Betrieb den Fokus auf die „jetzt“ wirklich wichtigen Tickets gegeben und dem Anforderer ein Feedback gegeben hat, wann ein Ticket aller Wahrscheinlichkeit nach umgesetzt wird. Von der Grundidee haben wir unsere Themen sichtbar gemacht und Timeboxes von 2 Wochen eingeführt. Jedes Ticket wurde einer Timebox zugewiesen oder gezielt in den Pool gelegt, wenn es später bearbeitet werden sollte. Auch Tickets unbearbeitet jedoch kommentiert schließen kann sinnvoll sein, führt aber gerade am Anfang zu Diskussionen.



Damit das Board funktioniert, haben wir einen Filter auf alle offenen Tickets des IT-Betriebs gelegt. Das Board selber hat drei Kategorien von Swimlanes: (1) **Timeboxen** für die Bearbeitung, (2) den **Eingangsbereich** und (3) den **Pool**.

Eine **Timebox** dauert immer zwei Wochen. Tickets in einer Timebox werden mit dem JIRA Stichwort „sysKWxxxx“ getagged wobei „xxxx“ die jeweiligen zwei Kalenderwochen angeben (Beispiel sysKW0102, sysKW0304, sysKW0506, ...). Wird ein Ticket z.B. mit dem Stichwort „sysKW3334“ versehen, so erscheint das Ticket in der Swimlane der Timebox „KW33/34“. Diese Timebox gibt dem Bearbeiter die Chance sich nur auf Tickets in der Timebox zu fokussieren (und nicht auf alle 500 offenen Tickets). Der Anforderer kann am Stichwort erkennen, wann sein Ticket gemäß Planung bearbeitet werden soll.

Neue Anforderungen oder Störungen landen automatisch im **Eingangsbereich**. Ein Dispatcher entscheidet nun anhand von Prioritäten (ggf. sind hierfür Rückfragen beim Anforderer notwendig), in welcher Timebox ein Ticket planmäßig abgearbeitet werden soll. Um ein Ticket in eine Timebox zu packen, muss er lediglich das JIRA Stichwort „sysKWxxxx“ vergeben.

Ist die Priorität einer Anforderung oder eine Störung aktuell nicht besonders

hoch, so wird das zugehörige Ticket im **Pool** abgelegt. Dazu vergibt das Dispatcher das Stichwort „sysPool“. Auch ist es der Dispatcher, der immer wieder den Pool prüft, ob Tickets aufgrund veränderter Prioritäten in eine Timebox geschoben werden sollten. Der Anforderer wird automatisch per Mail über die Vergabe von JIRA Stichwörtern und somit den Planungsstatus des Tickets informiert. Ist er mit der Timebox oder dem Pool nicht einverstanden, so kann er beim Dispatcher die Priorität besprechen und ggf. verändern.

Was haben wir bis hier erreicht?

Wir hatten nun eine gute Übersicht über alle Themen und in welchem Status die Themen waren. Wir konnten Themen priorisieren, abmelden oder auf später verschieben. So konnten wir die wichtigen Themen früher machen und die vermeintlich unwichtigeren Themen später oder gar nicht. Im Team hatten wir mit einem Mal Struktur in den Themen. Die Team-Mitglieder hatten nicht mehr den Druck, alles auf einmal machen zu müssen. Sie hatten einen klaren Blick auf eine handhabbare Anzahl an Tickets. Dadurch konnten sie fokussierter an den wichtigen Themen arbeiten.

Ergebnis: Die Zufriedenheit bei Anforderer und Bearbeiter stieg. Die Zahl der Eskalationen reduzierte sich. Ein großer Schritt nach vorne!

Warum mussten wir uns dennoch weiterentwickeln?

Waren wir jetzt schnell genug? Nein! Auf keinen Fall!

Per Prozessdefinition musste ein virtueller Server, der in Produktion eingesetzt werden sollte, rund acht Wochen vor Deployment angemeldet werden (3 Wochen Planung, 1 Woche bauen, 4 Wochen die neue Software testen). Die echte Arbeitszeit in den acht Wochen lag jedoch nur bei ~3PT. Ein Großteil der Zeit während der acht Wochen ging für die Terminfindung, Warten, Abstimmung, Rückfragen, ... bei den beteiligten Teams drauf. Nicht jeder fand das super. Auch wurden die Aufgaben (insbesondere aus dem Tagesgeschäft) immer mehr. Wir hatten kaum Zeit für Innovation und Weiterentwicklung von Themen aus dem

Bereich IT-Betrieb. Wir waren zwar entspannter mit einem besseren Blick – jedoch immer noch Getriebene.

Die Reise geht also weiter ...

Kapitel 2: Basistechnologien, SelfServices & Automation

In einer Woche werde ich im zweiten Kapitel unserer Reise darüber berichten, was wir im Bereich der Technologien gemacht haben. Wie konnten wir Werkzeuge nutzen, um uns zu beschleunigen und die Schnittstellen zu den Produkt-Teams genauer definieren? To be continued ...

Alle drei Kapitel im Überblick



[Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf](#)

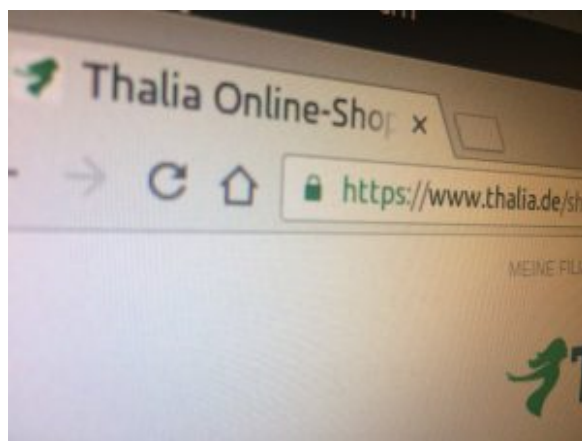


[Kapitel 2: Basistechnologien, SelfServices & Automation](#)



[Kapitel 3: Mindset, Methoden, Prozesse und mehr...](#)

Alles auf „grün“ – Unser Weg hin zu „SSL für Alles“



Alles auf „grün“: hier ist die komplette Umstellung unserer Mandanten / Domain thalia.de gemeint

HTTPS ist nicht nur ein von Google bestätigter Rankingfaktor, auch einige Internet-Browser weisen inzwischen Webseiten, die noch nicht über HTTPS erreichbar sind, als unsicher aus. Aus Nutzersicht macht es also ganz klar Sinn Websites auf HTTPS umzustellen. Niemand möchte, dass sensible Daten von Dritten abgefangen werden können. Sieht ein Nutzer, dass eine Website vom Browser als unsicher eingestuft wird, überlegt er es sich zweimal, ob er einen Kauf abschließt oder seine Daten für einen Newsletter oder andere Dienste hinterlegt. Eine Einstufung als sicher kann ihn hingegen in seiner Kaufabsicht bestärken. Natürlich sind bei Thalia die Seiten, auf denen sensible Daten sind bzw. erhoben werden, schon immer verschlüsselt. Wir haben uns jedoch bewusst entschieden alle Seiten zu verschlüsseln was neben den erwähnten positiven Faktoren für den Kunden auch die Performance optimiert als auch die Administration der Seite vereinfacht.

Auch im Bezug auf die **Sichtbarkeit** in Suchmaschinen ist die Nutzung von HTTPS anzuraten. Schon seit August 2014 ist HTTPS als Rankingfaktor bei Google etabliert. Seit Ende 2015 versuchen sämtliche Suchmaschinen automatisch Websites unter HTTPS aufzurufen. Ist eine Website unter HTTP und HTTPS aufrufbar, wird in der Regel die HTTPS-Version indexiert. Eine Studie des SEO-Tool-Anbieters Searchmetrics von 2015 bestätigt einen Zusammenhang zwischen der Nutzung von HTTPS und der Sichtbarkeit in Suchmaschinen einer Website. Laut der Studie haben Websites mit HTTPS tendenziell bessere Rankings.

Doch nicht nur die Sichtbarkeit und / oder Indexierung von Inhalten ist maßgeblich für eine erfolgreiche Umstellung auf HTTPS. Unsere gesamte eCommerce Plattform unterliegt regelmäßigen PCI ([Payment Card Industry Data Security Standard](#)) Audits. Mithilfe dieser extern beauftragten Audits wird sichergestellt, dass sämtliche Zahlungsmodalitäten (Visa, Paypal, etc.), sowie Verfahren innerhalb unserer eCommerce Plattform, die [Anforderungen und Regeln des PCI](#) erfüllen. Einer der wichtigsten Punkte ist dabei die Absicherung sämtlicher Kommunikation zwischen Plattform, Endbenutzer sowie Drittanbieter. Dieses ist für uns ein Baustein um Kundendaten maximal zu schützen.

Es ist ziemlich sinnvoll SSL auf allen Webseiten zu haben. Lasst es uns angehen! Aber nicht ohne die Fachabteilungen.

Für die Umsetzung haben wir uns bewusst für den Wasserfall-Ansatz entschieden. Gestartet mit dem Thema „SSL für Alles“ sind wir im März 2016 mit einer großen Kickoff Veranstaltung, bei der alle Fachbereiche (insgesamt 12), die Berührung mit der eCommerce Plattform hatten, eingebunden waren. Auch wenn es schnell und einfach möglich gewesen wäre auf dem Apache ein HTTPS-Forwarding einzurichten, ohne die einzelnen Fachbereich wäre es auf keinen Fall gegangen. Der Shop wäre dann zwar technisch auf HTTPS gewesen, jedoch hätten sich die Kollegen gefragt, wo plötzlich die Umsätze hin sind.

Ausgehend von der großen Kickoff Veranstaltung begann anschließend die erste Planung der für die Umstellung notwendigen Ressourcen. Dabei kamen unglaubliche Zahlen zutage: 195 PT und 31.09.2016. So viel schon mal vorweg:

das Datum konnten wir nicht halten ☐

„Vorbereitung & Planung ist alles“...

...hat mal ein schlauer Mensch gesagt und lag damit verdammt richtig. Doch auch eine noch so genau Planung und Vorbereitung, in der man(n) denkt, auch wirklich an alles, absolut alles gedacht zu haben, relativiert sich, wenn die Umstellung naht und der „Schalter umgelegt“ wird.

Es versteht sich von selbst, dass eine solche Umstellung von HTTP auf HTTPS nicht auf den produktiven Systemen „per Hand“ durchgeführt werden kann und darf. Zur Absicherung der Plattform und aus Gründen der Stabilität sowie aus Sicherheitsgründen haben wir uns dafür entschieden die bisherige produktive Webstrecke bestehend aus diversen Webservern, Loadbalancern und Firewalls nicht zu verändern. Stattdessen haben wir parallel eine komplett neue Webstrecke mit den eben genannten Komponenten aufgesetzt. Mithilfe dieser sind wir in der Lage die Umstellung schnell, sicher und ohne Ausfall zu planen und im Vorfeld testen.

Mit der Bereitstellung einer dedizierten Webstrecke — bestehende aus neuen IP Adressen, frischen Webservern auf Basis von Apache — sowie einer Vielzahl von Erweiterung / Änderungen an Firewall sowie Loadbalancer haben wir die Möglichkeit geschaffen, diese neue Strecke mit Anbindung an das produktive Backend ausgiebig zu testen und weiter zu optimieren, ohne dabei den eigentlichen produktiven Traffic in irgendeiner Weise negativ zu beeinflussen. Eine gute Ausgangsbasis, wenn man bedenkt, dass wir durch den Aufbau einer dedizierten bzw parallelen Strecke jederzeit hin und her schalten könnten.

Unsere Domain thalia.de sowie die dazugehörigen (virtuellen) Webserver sind nicht nur der Endpunkt für den Einstieg in den bereits genannten Online-Shop. Vielmehr bietet thalia.de eine Vielzahl von Diensten für die unterschiedlichsten Geräte bzw. Konsumenten — wie beispielsweise OAuth, API, Partner-Integration, Mobile Apps (XCA), Hörbuch Downloads, etc... — und natürlich auch sämtliche Microservices, die in den letzten Monaten durch die einzelnen Teams bereitgestellt worden sind. Nicht zu vergessen sind hier auch die Abhängigkeiten zu externen Partnern, wie Paypal, Payback, Visa, usw., um nur

einige zu nennen.

Die Umstellung ist nicht nur ein simples HTTPS-Forwarding auf dem Apache

Und spätestens an dieser Stelle dürfte jedem klar sein, dass eine scheinbar triviale Anforderung (HTTP zu HTTPS) weitreichende Auswirkungen haben kann. Daher ist es umso wichtiger das Gesamtbild aller Funktionalitäten und Abhängigkeiten vor Augen zu haben. Hand auf's Herz: Ist dazu wirklich irgendjemand in der Lage? Die Antwort darauf ist so nüchtern wie erschreckend: Nein. Aus der Theorie wissen wir, dass eine oder maximal eine handvoll Personen einen sehr guten Überblick haben. Der Rest *weiß* einfach, an wen man sich wenden kann und muss, wenn der Fall eintritt. Aus diesem Grund haben wir uns dafür entschieden alle wichtigen Vertreter der einzelnen Services (Shop-Management-System, App Services, IT Betrieb,) für den Tag der Tage in einem Raum zu vereinen. Die Umstellung ist ein Team-Erfolg.

SSL und SEO - so passt beides zusammen

Um eine möglichst Suchmaschinen-konforme Umstellung zu erreichen, mussten die folgenden Punkte zwingend eingehalten werden. Wie gut, dass wir dafür unseren Spezialisten am Tag der Umstellung an Board hatten. Eine zentrale Rolle spielt dabei die Google Search Console (ehemals Google Webmaster Tools). Hier ist es unbedingt notwendig, direkt nach der Umstellung auch die neue HTTPS-Version der Webseite einzurichten.

Umstellung auf HTTPS: Die wichtigsten Tipps

- Alle alten URLs sollten mit einem 301-Redirect auf die HTTPS-Variante weitergeleitet werden
- Auch Canonical Tags sollten in die SSL-Variante umgeschrieben werden
- Sofern bei den internen Links keine relativen Links verwendet wurden, sollten diese ebenfalls umgebaut werden, um unnötige Redirects zu vermeiden
- Die robots.txt Datei sollte auch via HTTPS erreicht werden
- Es sollte sichergestellt sein, dass alle internen und externen Ressourcen (Bilder, Scripte, CSS etc.) über HTTPS laden - ansonsten werden im

Browser für den Nutzer abschreckende Warnungen angezeigt

- In der [Google Search Console](#) sollte eine neue Property für die HTTPS-Version der Seite angelegt sein
- Die Sitemap sollte neu erstellt und bei der Google Search Console eingereicht werden
- [HTTP Strict Transport Security \(HSTS\)](#) sollte eingeführt werden: Dieser Sicherheitsmechanismus schützt HTTPS-Verbindungen vor der Aushebelung der Verbindungsverschlüsselung durch downgrade-Attacken

„Harry, leg den Schalter um !“

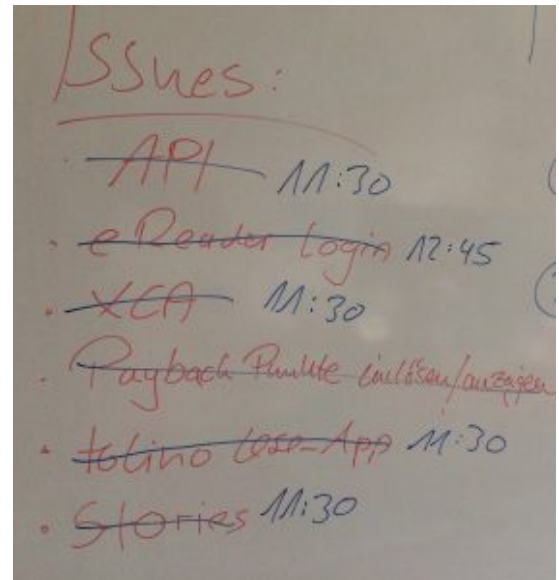
Eine Umstellung in der Nacht kam für uns nicht in Frage. Wir sind mutig und haben ein Top-Team welches am Morgen deutlich frischer ist. Ein beliebtestes Ritual darf an einem solchen Tag jedoch nicht fehlen: ausgiebig gemeinsam Frühstück.



Käse, Wurst, Lachs, Mett und 40
Brötchen waren es zu Beginn :-)

Gut gestärkt geht es dann in die letzte Vorbereitungsphase, in der alle anwesenden Teilnehmer(/-innen) noch einmal alle notwendigen Punkte durchgehen. Für die vorliegende Umstellung von HTTP auf HTTPS haben wir uns für einen klassisch-moderaten Ansatz entschieden: Anstatt Änderungen im DNS zu einem gegebenen Zeitpunkt durchzuführen und darauf zu warten, dass diese Änderungen weltweit propagiert sind (Notiz am Rande: Ein TTL von 5 Minuten kann auch sehr sehr lang sein), entschieden wir uns für eine Anpassung bei unserem vorgeschalteten DDoS-Schutz. Mit Hilfe des dazugehörigen Webportals kann im Handumdrehen der Upstream (in diesem Fall unser Endpunkt) aktualisiert werden, ohne dabei auf TTL oder sonstiges warten zu müssen, denn

diese Art von Änderungen werden sofort aktiv. Hinzu kommt der charmante Vorteil, dass wir jederzeit und auf direktem Wege wieder zur ursprünglichen Konfiguration hätten zurück kehren können. Das passt und befriedigt auch die verbleibenden kritischen Stimmen.



Auszug aus unserem Whiteboard vom Tag der Umstellung: Störungen sowie die Lösungszeit sind hier vermerkt.

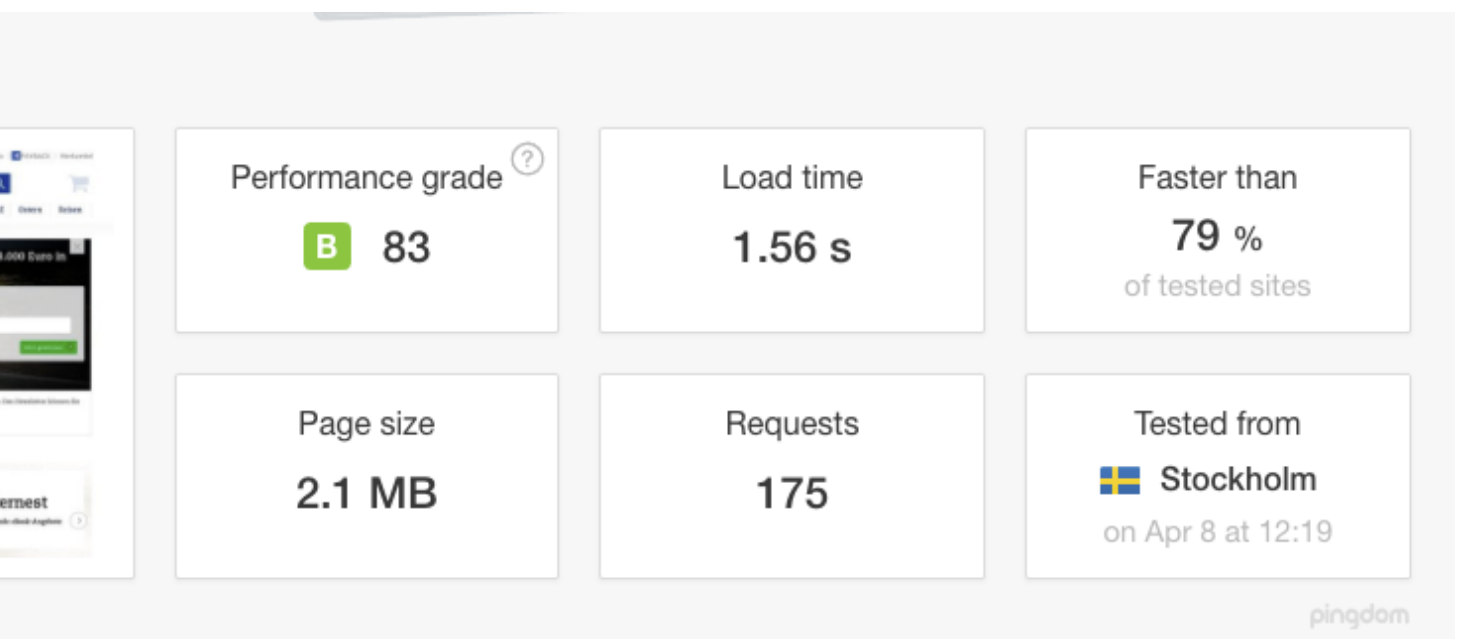
Wer jetzt erwartet hat, dass diese Umstellung ohne irgendwelche „Ruckler“ oder sonstigen Seiteneffekte vollzogen worden ist, den müssen wir hier enttäuschen. Auch Thalia kennt Murphys-Law. Es gibt immer Punkte, die zuvor nicht berücksichtigt werden konnten oder schlichtweg auf der Strecke geblieben sind. Doch eine gute Vorbereitung heißt auch, genau diese Punkte in kürzester Zeit anzugehen sowie diese Probleme aus dem Weg zu räumen. Denn hey, dafür haben wir doch „alle Mann an Board bzw. in einem Raum eingesperrt“.

Bereits Wochen zuvor haben wir in (in teils penetranter Art & Weise) sämtliche Bereichen, Abteilungen sowie interne und externe Partner auf diese Umstellung sensibilisiert, wodurch genügend Raum und Zeit gegeben worden ist, sämtliche Probleme zu klassifizieren, zu adressieren und schnellstmöglichst lösen zu können.

Fazit & Lesson's Learned

Unsere Vorgehensweise sämtliche Stakeholder und notwendigen Ressourcen an einem (physikalischen) Ort zu bündeln (bzw einzusperren) erwies sich wieder einmal als sehr nützliche Methode / Methodik, denn alle gemeldeten Problemchen konnten während des normalen Tagesbetriebs gelöst werden. Dank unserer ASB (=Automatische Service Bereitstellung) konnte ein Hotfix noch am gleichen Tag erstellt, getestet und in Produktion ausgerollt werden.

Die Umstellung auf HTTPS (SSL/TLS) hat darüber hinaus noch weitere Vorteile: Wir sind ab sofort in der Lage [HTTP/2](#) Verbindungen zu unterstützen, welches eine effizientere Nutzung der Netzwerk-Ressourcen und kürzere Latenzen verspricht (und auch einhält). Verbesserungen sollen aber auch eine Kompression der Header der IP-Pakete sowie Multiplex- und Push-Techniken bringen. Browser können mittels [Multiplexing](#) mehrere Messages gleichzeitig über eine Verbindung beziehen und Server können Elemente im voraus, also ohne Anfrage des Browsers pushen. Dies wird umso deutlicher, wenn wir uns vor Augen führen, wie viele Bilder beim Aufruf der sämtlicher Seiten geladen und übertragen werden müssen: HTTP/2 erweist sich hier als bemerkenswert effizient, da sämtliche Anfragen parallel und mit nur einer Verbindung übertragen werden. Dies wiederum hat zur Folge, dass die Ladezeit erheblich verbessert werden konnte.



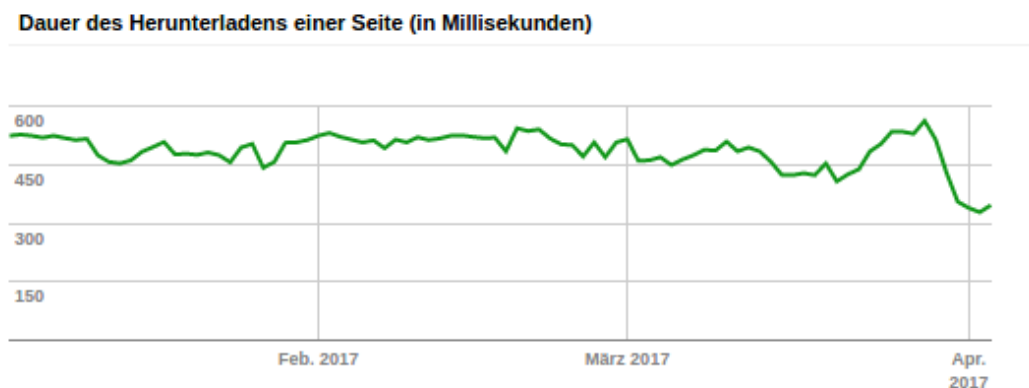
Umstellung auf HTTPS inkl HTTP/2 (Quelle: tools.pingdom.com)

[illegible]

Gut zu erkennen, wie mit nur einer Verbindung — Dank HTTP/2 — mehrere Bilder / Assets geladen und übertragen

Mittels [HSTS](#) ([HTTP Strict Transport Security](#)) wurde ein zusätzlicher HTTP-Header eingeführt, mit dem der Server dem Browser signalisiert, dass eine Seite künftig nur noch über HTTPS abgerufen werden soll. Der Browser speichert diese Information lokal für einen im Header festgelegten Zeitraum („Max. Age“), der im Prinzip wie ein Cache funktioniert. Ab diesem Moment wendet der Client sich direkt über HTTPS an den Host, auch wenn der Nutzer explizit „http://“ in der Adresszeile des Browsers eingibt oder einem „http“-Link dorthin folgt. Ein

weiterer Vorteil für den Client, über den Sicherheitsgewinn hinaus: Durch die entfallende Umleitung verkürzen sich die Antwortzeiten.



Auch Google indiziert die Domain www.thalia.de nun deutlich schneller als zuvor (Unsere Änderung wurde Ende März in Produktion ausgerollt)

Ausblick

Der erste Schritt in die richtige Richtung ist vollbracht. In den kommenden Wochen werden wir das Verhalten sowie die Performance weiter im Auge behalten und ständig weiter optimieren, um unseren Kunden und Besuchern eine best mögliche User Experience zu präsentieren. In Sachen Performance und User Experience stehen unsere Edge-Provider (MyraCloud Security sowie Akamai) ganz oben auf der Liste der Agenda. Hier gilt es zu bewerten, inwiefern wir das Caching weiter verfeinern können.

Ich@Thalia

Am Anfang

Im Frühjahr des letzten Jahres (2016) habe ich meine Tätigkeiten als *System Engineer* bei Thalia am Standort Münster begonnen. Jeder — wirklich jeder in meinem näheren Umfeld — kannte bereits den Namen bzw. die Marke Thalia und konnte damit zumindest eine Verbindung zum (lokalen) Buchhandel herstellen. „Aber wie in aller Welt soll denn meine Tätigkeit bei einem Buchhändler aussehen ... ?! „, habe ich mich — sowie mein Umfeld sicherlich auch — zurecht gefragt. Zum damaligen Zeitpunkt konnte ich diese Frage nicht wirklich beantworten, ließ mich aber gern auf dieses Abenteuer und neue Erfahrungen ein.

Ich gestehe, dass ich keine Ahnung davon hatte, was mich in den kommenden Wochen und Monaten erwarten würde, aber hey, ist das nicht immer so, wenn man(n) etwas Neues beginnt ?! Gleichzeitig konnte (und ein wenig auch „wollte“) ich nicht wahrhaben, dass Thalia bereits zu diesem Zeitpunkt ein durchaus gewachsenes eCommerce Business — neben dem klassischen Buchhandel — erschaffen hatte und dieses erfolgreich betreibt.

Plötzlich mittendrin, statt nur dabei

Niemals auf der Stelle zu treten liegt mir quasi schon von Berufs wegen (als sog. Externer Kollege) im Blut. Diesen ständigen Fortschritt sowie die Weiterentwicklung bereits existierender Produkte sind auch nach mehr als einem Jahr mein ständiger Begleiter innerhalb unseres/unserer Teams.

[Wikipedia](#) bezeichnet „*ein Team als einen Zusammenschluss mehrerer Personen zur Lösung einer bestimmten Aufgabe oder zur Erreichung eines bestimmten Ziels ...*“ (Quelle: [Wikipedia: Team](#)). Soviel zur Definition, doch mitten in ein bereits existierendes Team integriert zu werden, wahrgenommen zu werden, Aufgaben zu erledigen, Kritisch zu hinterfragen, „Tekki-Witze“ auf eine Pinnwand zu kritzeln, um sich sowie alle anderen zum Lachen zu bringen, Fehler zu machen, Spaß daran zu haben Neues auszuprobieren, etc. sind meine Gründe, warum ich meine Tätigkeit hier gern ausübe. Neben den sog. harten Fakten, die ich eben beschrieben habe, sind es jedoch immer die Menschen selbst, die für mich den deutlichsten Unterschied machen. Ich zitiere an dieser Stelle gern einen Satz, der mir selbst schon häufig um die Ohren geflattert ist und den ich nur doppelt und dreifach unterstreichen kann: „... Es sind die Kollegen und

Kolleginnen, die für mich den Unterschied machen“. Bingo, voll ins Schwarze ☐

Spielplatz auf 12 Uhr

Achtung Spoiler-Alarm: Wer glaubt, dass wir für unsere eCommerce Plattform nur längst überfällige Hardware und Low-Cost Komponenten einsetzen, dem kann ich hier nur folgende Buzzwords anbieten: [F5](#), [Palo Alto](#), [VMWare](#), [HP](#), [IBM](#), [Graylog](#), [Puppet](#) & Co sind nur ein paar Komponenten, die bei uns im täglichen Einsatz sind.

Das Thema „Datenverarbeitung in Echtzeit“ steht in unserer Liste der ToDo's schon länger ganz oben. Daraus erwachsen für unsere IT neue Herausforderungen, um die automatische Speicherung von Daten, ihre Verarbeitung und Analyse schnell und übersichtlich zu ermöglichen. [Graylog](#) ist ein Open-Source-Projekt für das Log-Management von Daten jeglichen Formats. Auf der Graylog-Plattform werden sowohl strukturierte als auch unstrukturierte Log-Daten von verschiedenen Servern und Anwendungen gesammelt, indiziert und analysiert. Ebenso möglich ist die Verwendung von Komponenten für Metadaten wie MongoDB oder Elasticsearch als Log-Daten-Speicher und Text-Suche. Auch hier stehen Dashboards zur Verfügung, denen Analyse- und Suchresultate als Widgets zugefügt werden können. Die Suche sowie die grafische Visualisierung sind somit ohne Vorkenntnisse oder lange Schulungen möglich.

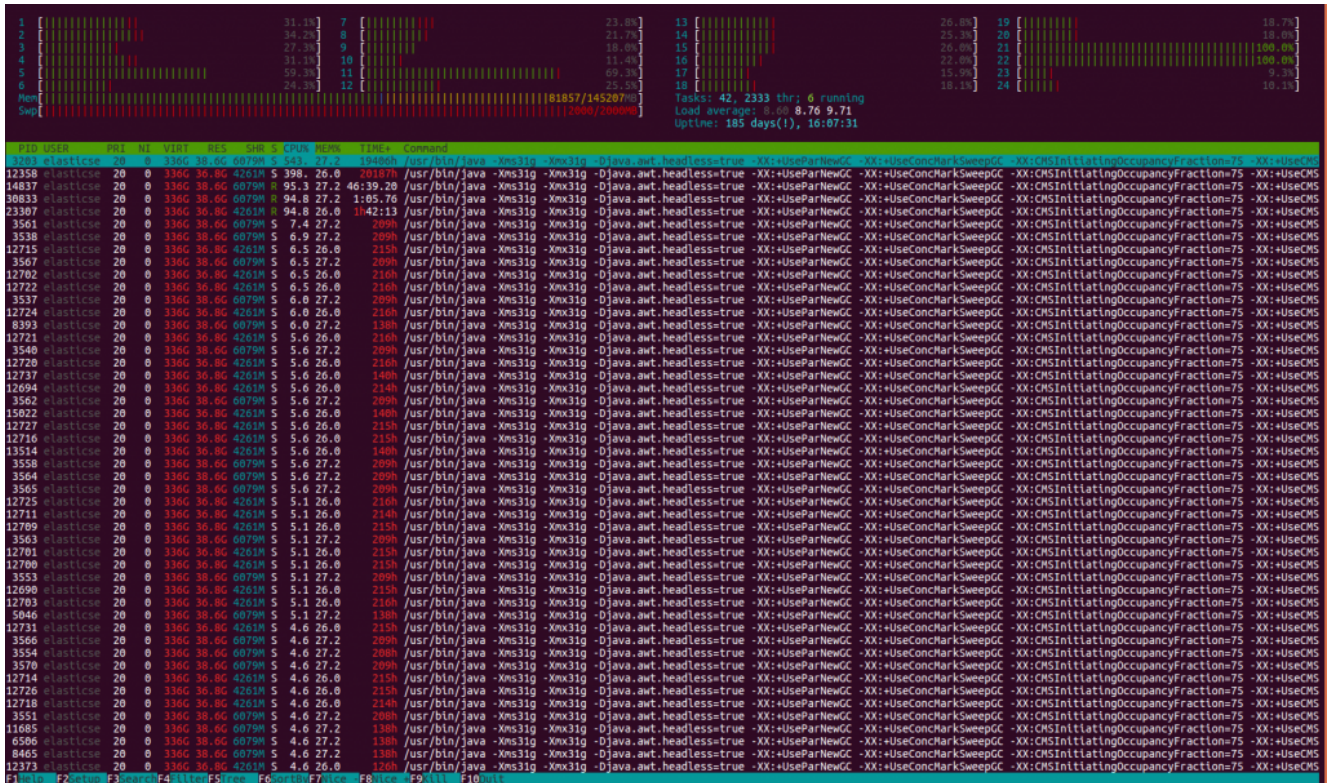


Bild 1: Einer der neuen LogServer auf Basis von Graylog kurz vor dem GoLive

Hand auf Herz: Wer hat sich nicht schon mal wie ein Kleinkind an Weihnachten gefreut, wenn plötzlich klar wird, dass die nagelneue, vor CPU und RAM strotzende Hardware betriebsbereit dasteht und direkt einem kleinen aber feinen Lasttest unterzogen wird. Jeder (in dieser Branche). Und ja, wir sind große Kinder auf einem Spielplatz, aber genau dafür ist dieser ja auch gedacht. Zum Lernen.

Bin ich richtig hier ?

Kurz und schmerzlos: Ja, absolut.