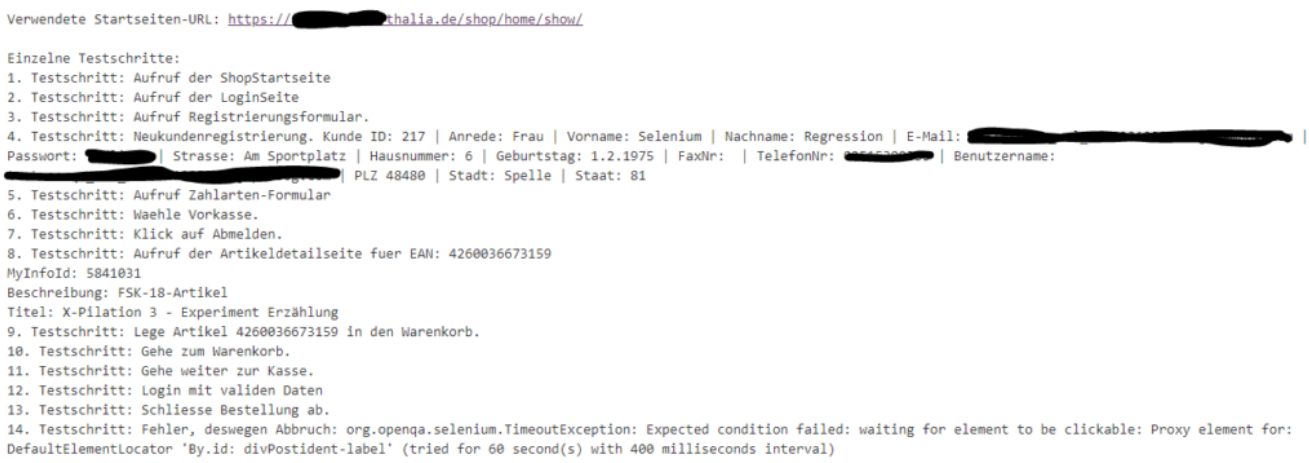
True Grid oder wie wir unsere Testausführung durch Parallelisierung 6x schneller machten

Bei Thalia gibt es nicht nur crossfunktionale Teams mit QAlern, es gibt auch die zentrale QA. Wir testen prozesskettenübergreifend – d.h. während die Kollegin aus Team Kaufen alle Eingabemöglichkeiten der Zahlarten durchtestet oder der Kollege aus Team Kunde alle möglichen (und unmöglichen) Lieferadressen hinterlegt, legen wir in einem Test einen Neukunden an, ändern die Zahlart, hinterlegen eine Lieferadresse, suchen einen Artikel und kaufen ihn dann. Alles natürlich automatisiert, und für 4 verschiedene Mandanten.



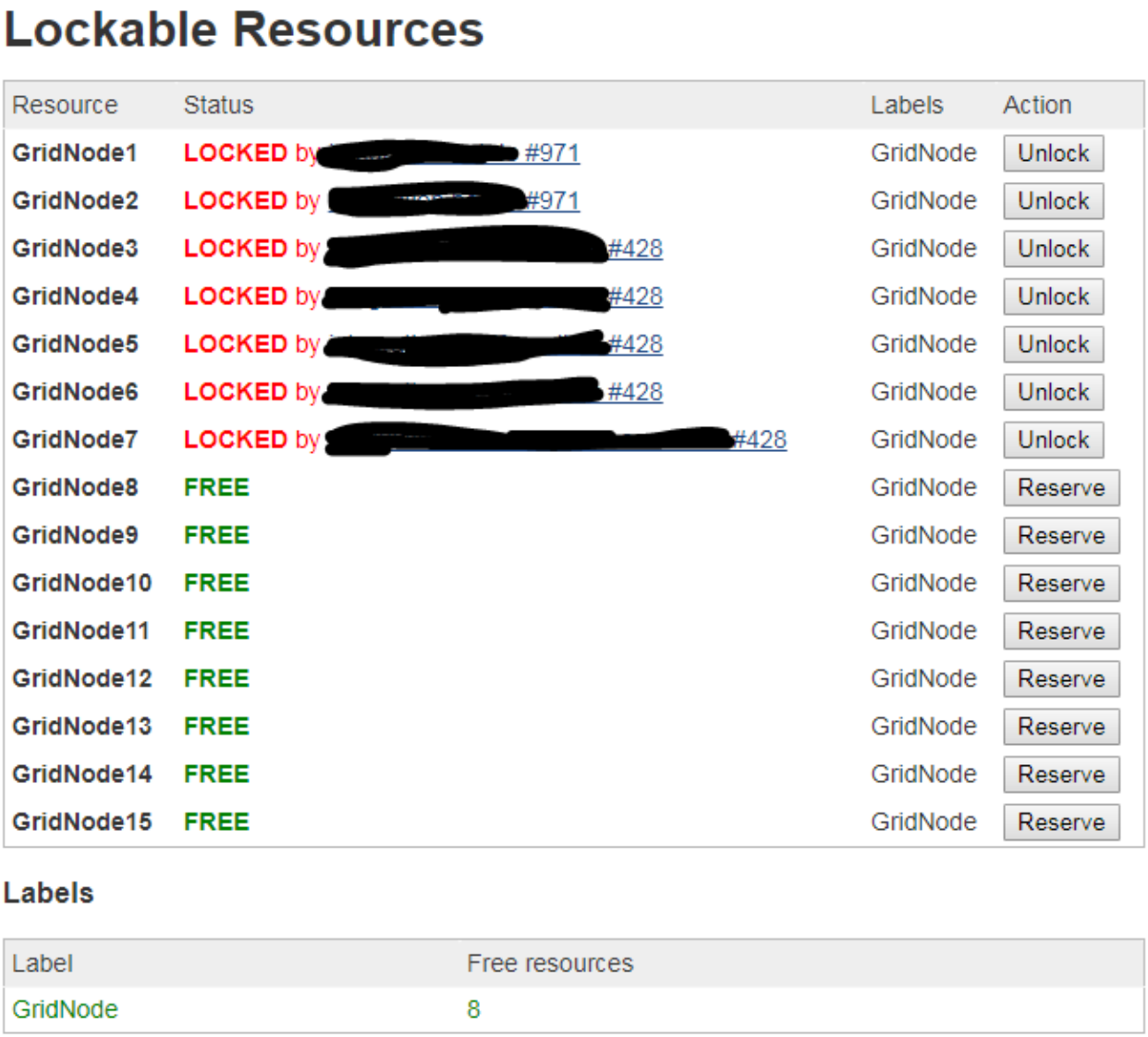
Insgesamt haben wir rund 100 Tests, die wir auch noch in 2 Browsern mit unterschiedlichen Auflösungen ausführen. Bis vor wenigen Wochen noch dauerte der größte Integrationstest ca. 6 Stunden… Wenn dann aber ein Hotfix aufgespielt werden soll und jemand schnell die Antwort haben möchte, ob denn die automatischen GUI-Tests fehlerfrei durchgelaufen sind, ist „Frag morgen nochmal nach“ keine angenehme Antwort – unsere Tests mussten schneller werden.

Wir überlegten kurz, ob weniger Tests in weniger Browsern oder ohne Screenshots eine Alternative wären, stellten aber schnell fest, dass unser Problem leichter mit mehr Rechnern und paralleler Testausführung zu lösen war. Selenium Grid mit allen Möglichkeiten zur einfachen Parallelisierung von

ferngesteuerten Browsern war die Antwort.

Alle machen Docker, warum nicht wir? Bisher hatten wir unsere Tests auf Windows durchgeführt (die Mehrzahl der Kunden nutzt Windows), aber wenn man z.B. Zalenium (einen Docker-basierten, dynamischen Selenium Grid) nutzen will, muss man Linux als Plattform wählen. Leider scheint es mit Firefox unter Linux bei automatisierten Tests manchmal Probleme mit unserem Hauptmenü zu geben, was sich an unerwarteten Stellen über die wichtigen Elemente legt und damit den Testweitergang behindert. Außerdem kann man mit Zalenium Chrome nicht in der kleinsten Auflösung unserer Webseite benutzen – das ist uns aber sehr wichtig, daher nahmen wir wieder Abstand von einer Docker- und Linux-basierten Lösung.

Die Anzahl unserer Tests ändert sich nicht so häufig, und wie viele Nodes wir parallel brauchen, können wir einfach pro Testinstanz im Jenkins festlegen – daher entschlossen wir uns, mit einer festen Anzahl Windows-Rechner zu arbeiten, die sich jeweils mit 4 Node-Prozessen an einem Selenium-Grid (der ausnahmsweise unter Linux läuft) anmelden. Um die genaue Anzahl von verfügbaren Browser-Handles und damit die Anzahl von Threads, die pro Test verwendet werden können, zu verwalten, sei hier noch löblich auf das Lockable Resource-Plugin im Jenkins verwiesen, mit dem wir recht einfach Flaschenhälse bei der gleichzeitigen Ausführung des Tests verhindern konnten.



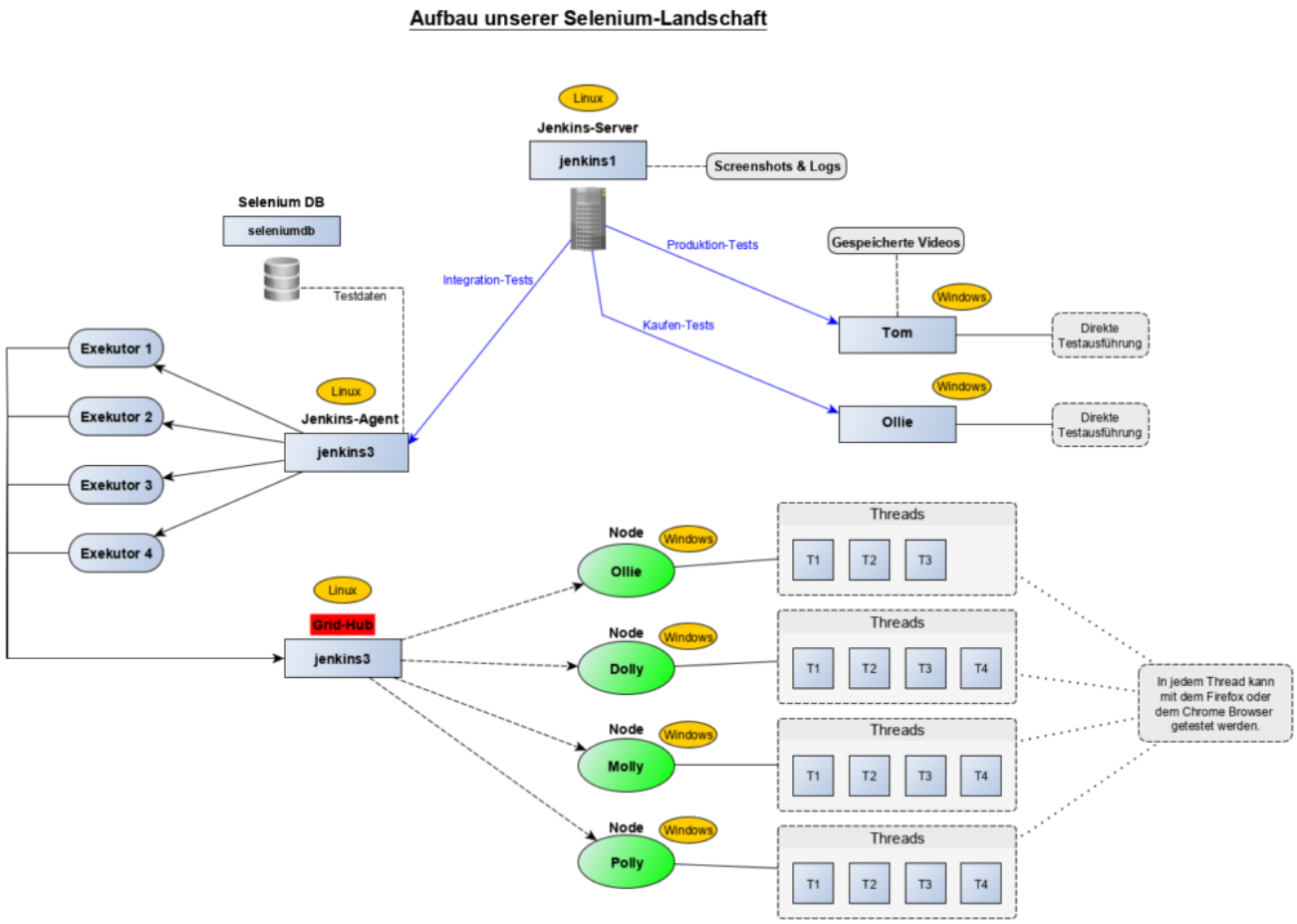
Lockable Resources

Resource	Status	Labels	Action
GridNode1	LOCKED by [redacted] #971	GridNode	Unlock
GridNode2	LOCKED by [redacted] #971	GridNode	Unlock
GridNode3	LOCKED by [redacted] #428	GridNode	Unlock
GridNode4	LOCKED by [redacted] #428	GridNode	Unlock
GridNode5	LOCKED by [redacted] #428	GridNode	Unlock
GridNode6	LOCKED by [redacted] #428	GridNode	Unlock
GridNode7	LOCKED by [redacted] #428	GridNode	Unlock
GridNode8	FREE	GridNode	Reserve
GridNode9	FREE	GridNode	Reserve
GridNode10	FREE	GridNode	Reserve
GridNode11	FREE	GridNode	Reserve
GridNode12	FREE	GridNode	Reserve
GridNode13	FREE	GridNode	Reserve
GridNode14	FREE	GridNode	Reserve
GridNode15	FREE	GridNode	Reserve

Labels

Label	Free resources
GridNode	8

Jetzt haben wir also 4 Windows-Rechner als Selenium Nodes. Ein weiterer Windows-Rechner wird für stündliche Tests gegen die Produktion verwendet. Und auf denen soll man jetzt Firefox- bzw. Chrome-Versionen im Blick behalten und die passenden Treiber für Selenium (geckodriver oder chromedriver) updaten? Sowas macht man heutzutage nicht mehr per Hand, es musste eine automatisierte Update-Lösung her. Dank Git und einem Batch-Update-Skript, was nächtlich läuft, kann ich auf meinem lokalen Rechner das Zusammenspiel von Browser-Treiber und Browser testen, einchecken und am nächsten Tag davon ausgehen, dass die Updates auf alle Rechner verteilt sind.



Und wie lange brauchen unsere Tests jetzt? Wir sind tatsächlich mit Tests für alle 4 Mandanten und noch einigen wenigen komplexen Testszenarien jetzt in rund 1 Stunde mit allen Tests durch – wenn alles gut läuft und die Testumgebung nicht zu sehr verändert wurde.

Build ↑	Duration	Agent
#1186	1 hr 1 min	um-qajenkins3
#1185	1 hr 1 min	um-qajenkins3
#1184	1 hr 20 min	um-qajenkins3
#1183	1 hr 3 min	um-qajenkins3

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

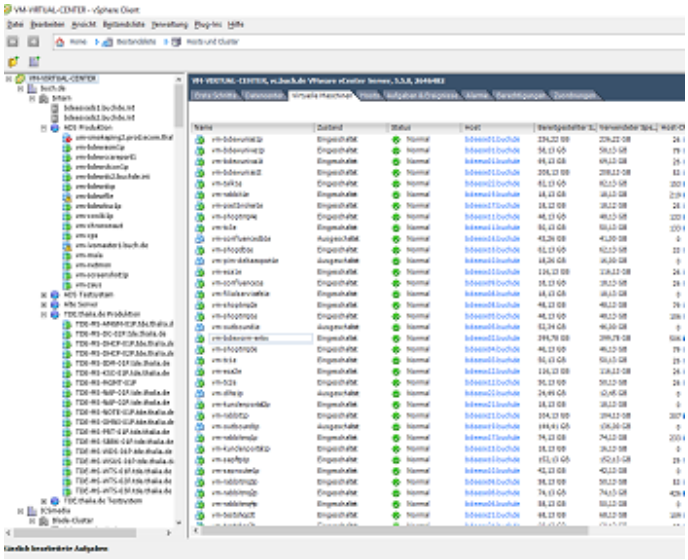
wir unterstützen sollen, warteten zu oft auf uns. Immer wieder kam z.B. die Aussage, dass ein Server in der Cloud in wenigen Minuten verfügbar ist. Wir als Team waren mit operativer Arbeit überlastet und hatten nur wenig Zeit für Weiterentwicklung. Irgendwie mussten wir im Tagesgeschäft schneller werden, 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)

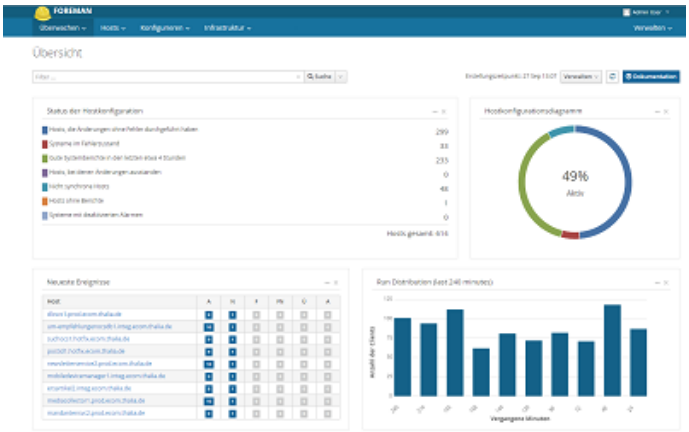


Vorher: Server klonen mit VMware

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?



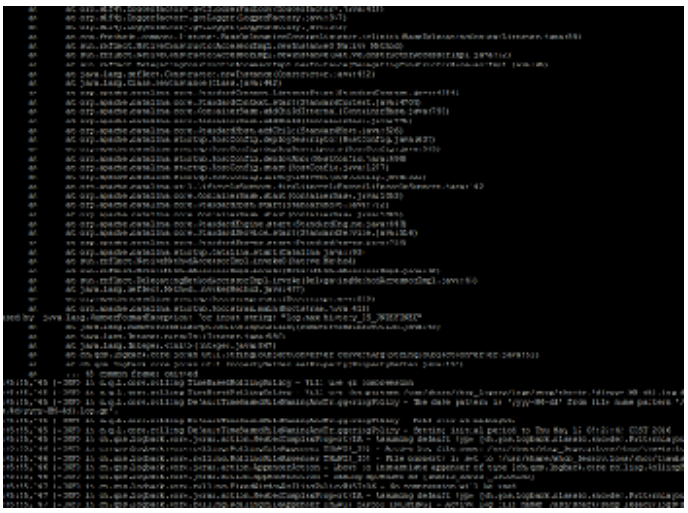
Nachher: Service Bereitstellung auf Foreman & Puppet Basis

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.

Was haben wir damit erreicht? Wir haben die Server- & Service-Installation nun via einfaches Webfrontend an die Entwicklungsteams gegeben. Wir müssen keine

Termine finden, es ist keine lange Abstimmung mehr notwendig, keine langen Rückfragen, keine Nachbesserung, keine acht Wochen Wartezeit… Der Aufwand für den IT-Betrieb und die Entwicklungsteams wurde stark reduziert, und die Durchlaufzeit für Server- und Service-Installationen wurde von Wochen auf wenige Minuten reduziert. Win/Win Situation!

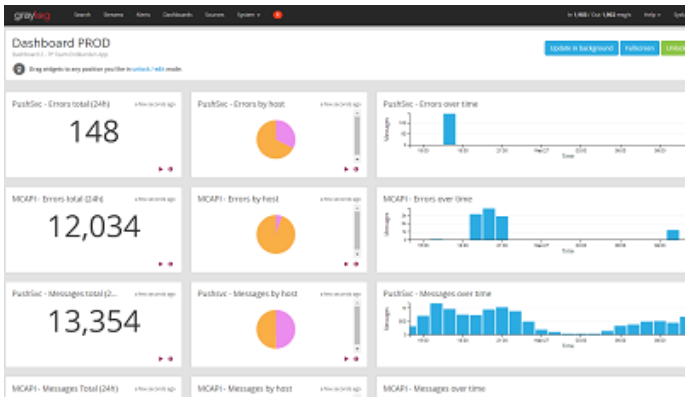
Beispiel 2: Zentrales Logdaten Management (ZLM)



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 unfröhliche Stimmung in den Entwicklungsteams, weil wichtige Daten fehlen und das System nicht verfügbar war. Gleichzeit 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, kümmere 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.



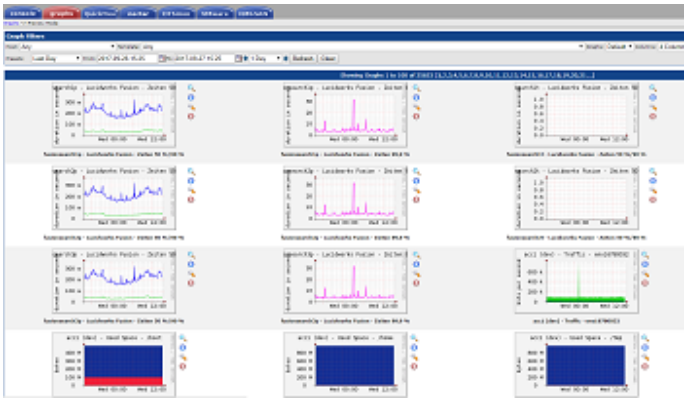
Nachher: Logdaten Management auf Graylog Basis

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 versprachen 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

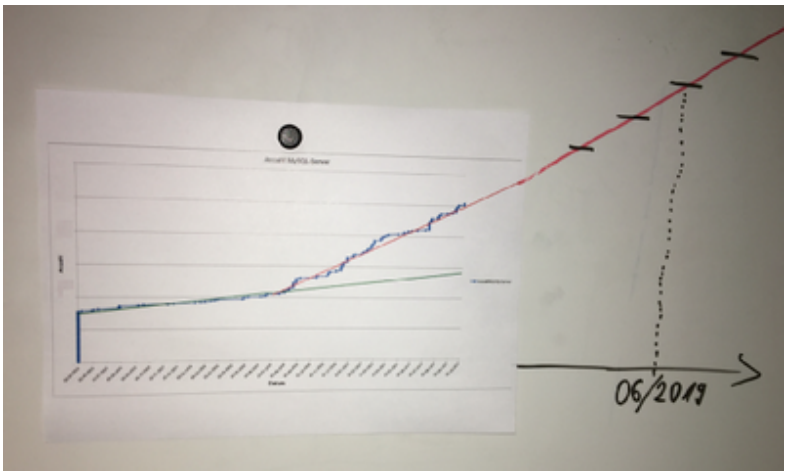
und Software-Entwicklern ein neues Performance Monitoring System basierend auf InfluxDB, Telegraph und Grafana zu bauen. Die Arbeiten wurden stark unterstützt von Produkt-Teams, die ihre Ideen und Anforderungen eingebracht haben. Nachdem der erste brauchbare Prototyp stand, haben wir vor der Pilotphase 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

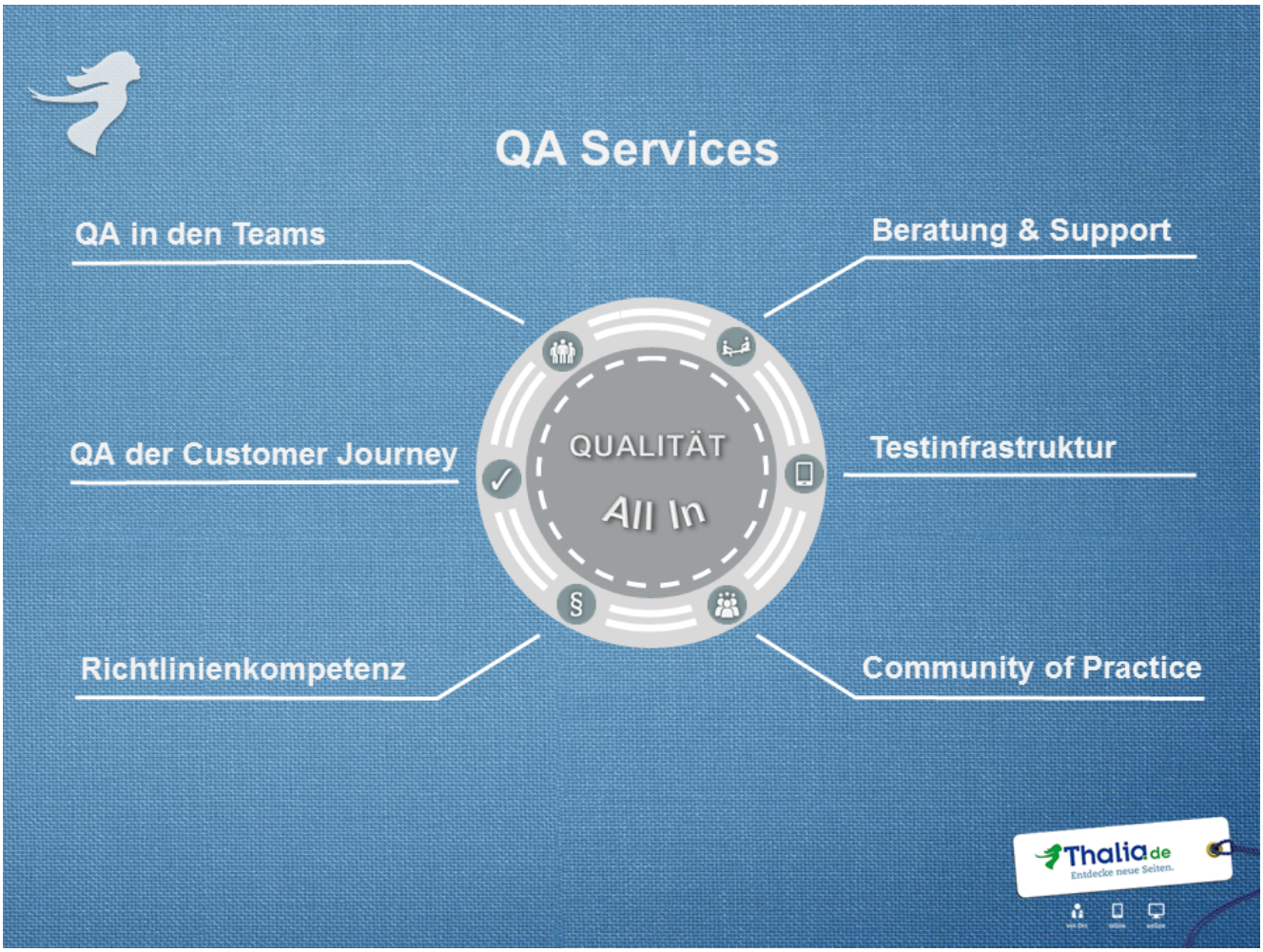
Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf

Kapitel 2: Basistechnologien, SelfServices & Automation

Kapitel 3: Mindset, Methoden, Prozesse und mehr…

QA Services – Mittendrin statt nur am Ende

Der Bereich Thalia eCommerce Quality Assurance Services unterstützt die anderen Bereiche bei dem Ziel, gegenüber den Kunden und Kundinnen ein hochwertiges Produkt zur Verfügung zu stellen. Wir sind dabei entweder direkt einem Team zugeordnet oder in der zentralen QA tätig. Durch die permanente Weiterentwicklung von Produkten und Prozessen werden wir stets gefordert, neue passende Lösungen zu entwickeln. Derzeit stehen bei uns sechs Themen im Fokus:



QA in den Teams

Wir sind als QA-Lead direkter Bestandteil eines cross-functional Teams und nehmen vielfältige Aufgaben wahr:

- Abstimmung der teaminternen Qualitätskriterien

- Erster Ansprechpartner zum Thema Qualitätssicherung und Test
- Themen zur Testautomation im Team bündeln und vorantreiben
- und natürlich das normale Testgeschäft

Beratung und Support

Wir beraten die einzelnen Teams und Fachbereiche bei Themen wie Testfallermittlung, Testkonzeption, Tooleinsatz etc.

QA der Customer Journey

Wir prüfen zyklisch, dass die Customer Journey für unsere Kunden keine Stolpersteine enthält. Die Prüfung durch automatisierte Tests bauen wir stetig aus.

Testinfrastruktur

Wo und wann kann ich testen? Woher bekomme ich Testdaten? Womit kann ich Lasttests ausführen? Das sind einige Fragestellungen, die wir beantworten. Und falls wir die Antwort nicht kennen, sorgen wir für Lösungen.

Richtlinienkompetenz

Wir definieren Spielregeln, damit die Flexibilität und hohe Innovationsgeschwindigkeit der agilen Produktentwicklung nicht zu Lasten von Stabilität und Qualität geht.

Community of Practice

Wir tauschen uns zu spannenden Themen im Bereich der QA aus, um voneinander zu lernen und besser zu werden.