

Der Flying Circus Projektleitfaden

Fragen und Tipps für einen gelungenen Projektstart
aus der DevOps-Perspektive beim Flying Circus

Der Erfolg digitaler Projekte hängt von vielen Faktoren ab:

neben guten Ideen und einem kompetenten Entwicklungsteam braucht es auch die richtige Unterstützung, um eine Anwendung zu betreiben. Die Themenvielfalt, die es dabei zu berücksichtigen gibt, ist in den letzten 20 Jahren stark angewachsen und fordert neben guter Technik auch ein partnerschaftliches und konstruktives Miteinander von allen Beteiligten.

Galt früher "never touch a running system" müssen digitale Produkte heute deutlich schneller auf sich ändernde Anforderungen reagieren: Neben der Implementation neuer Features muss auch kontinuierlich auf steigende Nutzerzahlen, Sicherheitsaspekte, regulatorische Anforderungen und vieles mehr reagiert werden.

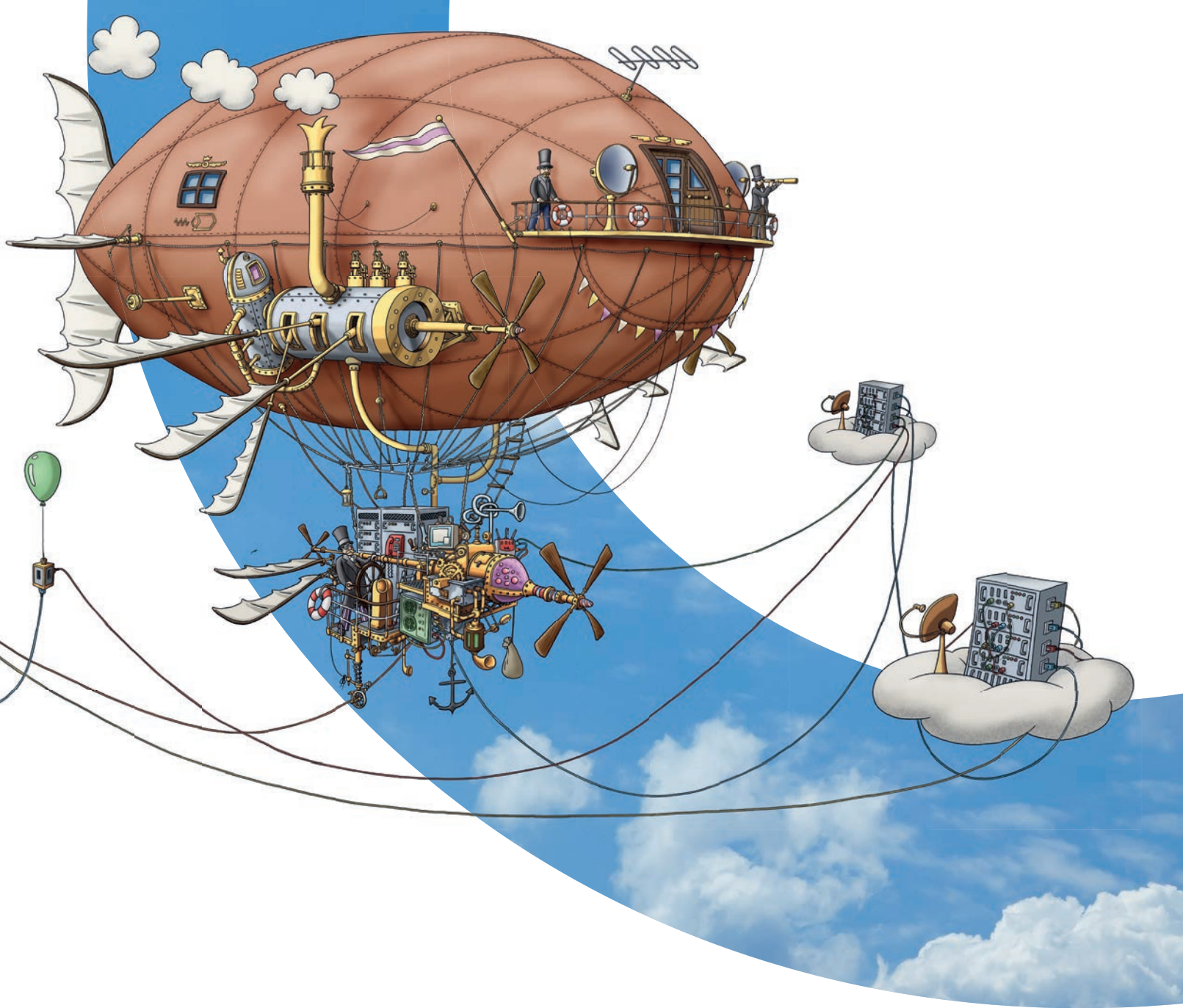
Unsere Begleitung auf Basis der DevOps-Philosophie hat zum Ziel, dass Projekte schnell und professionell starten können und dann in agiler Weise sich schnell weiterentwickeln können ohne dabei darauf vertrauen, dass die hier genannten betriebsrelevanten Aspekte immer mitziehen. Diesen Anspruch erfüllen wir vom Flying Circus als "System Reliability Engineers" (SREs).

Dieser Leitfaden soll Kund*innen, Projektleiter*innen und Entwickler*innen schon früh einen Eindruck verschaffen und eine Hilfestellung sein, um die wichtigsten Aspekte aus Sicht eines DevOps-Ansatzes konkret und pragmatisch zu beleuchten. Nicht alle Fragen müssen dabei in jedem Projekt relevant sein – aber der umfassende Blick soll auch dazu anregen, über Aspekte nachzudenken, die beim Fokus auf die Features einer Anwendung manchmal zu kurz kommen.

Viel Fragen lassen sich nur gemeinsam als Team beantworten - statt über Team-Grenzen fertige Dokumente auszutauschen, wollen wir so die Kommunikation zwischen Kund*innen, Entwickler*innen und System Reliability Engineers (SREs) fördern und damit Projekte schneller besser machen.

Christian Theune
CEO im Flying Circus





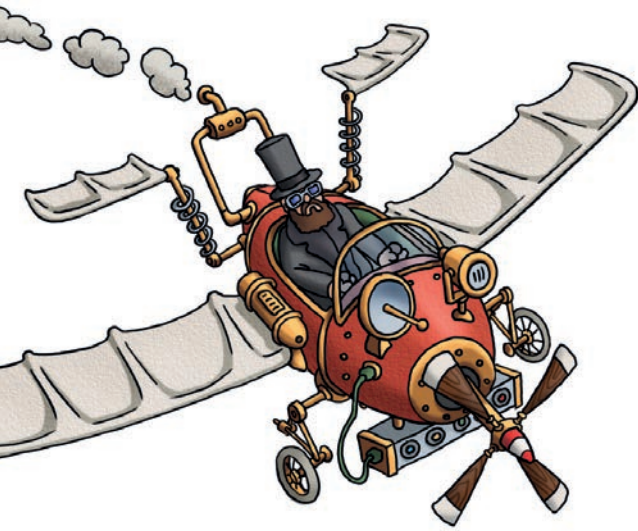
Tipp: Zur praktischen Umsetzung finden sich die Inhalte der folgenden Seiten auch als praktische Markdown-Vorlage zum Einsatz im eigenen Ticketsystem – einfach im PDF den Link-Button klicken oder in der Print-Broschüre den QR-Code scannen. Sie gelangen auf eine Seite, die das jeweilige Thema als Markdown zum Kopieren für Ihr Ticketsystem bereit hält – egal ob Jira, Redmine, Github, Gitlab oder YouTrack ...

[Link zum Markdown](#)



Inhalt

Das Projekt	5
Kund*in und Team	6
Zeitplan und Roadmap	7
Datenschutz, Sicherheit und Compliance	8
Architektur und Handwerkliches	10
Deployment und Konfiguration.....	12
Last- und Kapazitätsplanung.....	14
Performance und Skalierbarkeit.....	17
Netzwerk, Internet, externe Dienste und APIs.....	18
Monitoring, Zuverlässigkeit, Business Continuity.....	21



Das Projekt

Um digitale Projekte erfolgreich zu starten und weiterzuentwickeln, braucht es heutzutage ein übergreifendes Verständnis bei allen Beteiligten. Um unsere Erfahrung möglichst gut einbringen zu können, ist es wichtig, dass wir Projekte nicht nur aus der Perspektive betrachten, ob sie viel oder wenig Technik brauchen. Am meisten Wirkung entfaltet unser Team in den Projekten, in denen wir den weiteren Kontext kennen und wissen, welche Bedeutung das Projekt für unsere Kund*innen und die Nutzer*innen hat.

Wie heißt das Projekt?

Worum geht es in diesem Projekt?

Wer sind die Nutzer*innen?

Welche Funktionen bietet das Projekt seinen Nutzer*innen?

Wie sieht der bestmögliche Erfolg für dieses Projekt aus?



[Link zum Markdown](#)

Kund*in und Team

Neben dem Verständnis für das Projekt selbst müssen viele Absprachen getroffen werden und Beziehungen aufgebaut werden. Insbesondere ist es uns wichtig, Zugang zu unterschiedlichen Perspektiven zu haben, um Bedürfnisse aus unternehmerischen und technischen Perspektiven berücksichtigen zu können. Offenheit und Transparenz heißt auch möglichst „Stille Post“-Situations zu vermeiden, die häufig zu Fehlinformation, Missverständnissen und Verzögerungen führen.

Wie lauten die Kundendaten einschließlich Firmierung und Anschrift?

Projektverantwortliche Person:

Name	Organisation	Kontakt Daten (E-Mail, Telefon)	Funktion oder Rolle

Verantwortliche Person aus der Software-Entwicklung:

Name	Organisation	Kontakt Daten (E-Mail, Telefon)	Funktion oder Rolle

Weitere im Projekt relevante Personen:

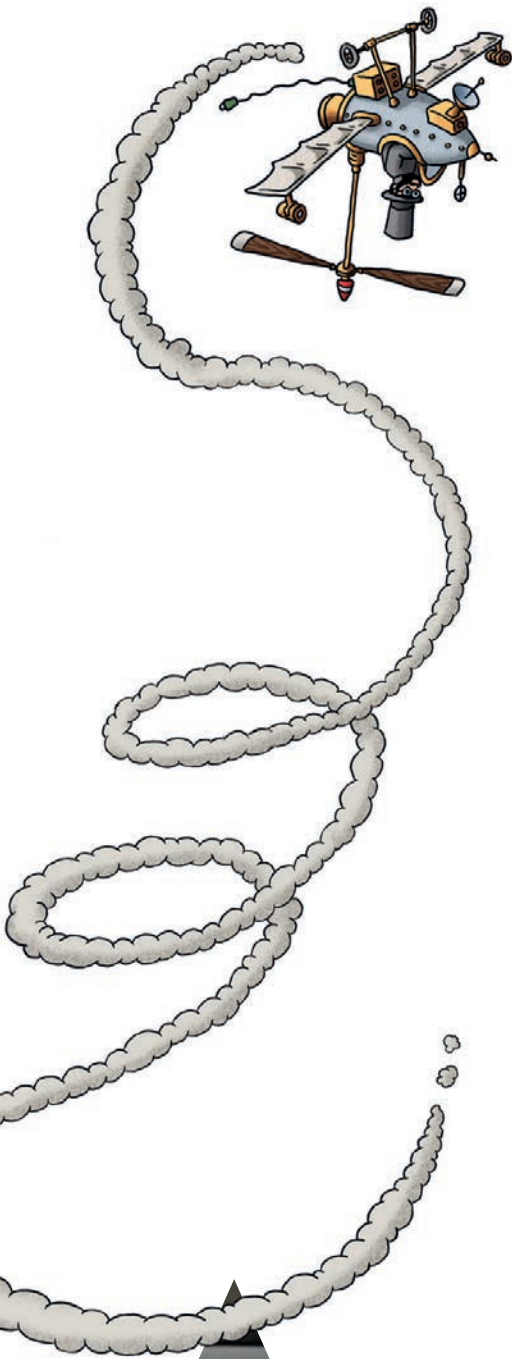
Name	Organisation	Kontakt Daten (E-Mail, Telefon)	Funktion oder Rolle

Welche bisher mit dem Betrieb vertrauten Personen sind unterstützend für die Migration ansprechbar?

Name	Organisation	Kontakt Daten (E-Mail, Telefon)	Funktion oder Rolle



Link zum Markdown



Zeitplan und Roadmap

Unterschiedliche Projekte können eine ganz unterschiedliche Dynamik entwickeln – häufig gibt es viele Unbekannte. Um möglichst proaktiv Entscheidungen treffen und sinnvolle Optionen vorschlagen zu können, hilft es, wenn wir die Situation des Projektes gut einschätzen können, wissen welche Termine unserer Kund*innen relevant sind und wo die Reise in Zukunft hingehen soll.

Welchen Charakter hat das Projekt?

Im Groben unterscheiden wir zwischen Neuentwicklungen und Migrationen. Weitere Unterschiede ergeben sich, wenn es sich um ein “Minimal Viable Product” handelt, wenn bestehende Systeme weiterentwickelt werden oder es sich um einen vollständigen Relaunch handelt.

In welchem Stadium befindet sich das Projekt zur Zeit?

Unabhängig ob das Projekt neu entwickelt wird oder bereits existiert: in welchem Entwicklungszustand befindet sich das Projekt zur Zeit? Wird noch das grundlegende Konzept besprochen? Gibt es bereits funktionierenden Code? Ist die Entwicklung bereits abgeschlossen und es soll “nur noch” live gehen?

Zeitplan:

Was	Bis wann
Entwicklungsumgebung	
Staging-Umgebung	
Produktionsumgebung	
Ready-to-Launch	
Livengang (Migration)	

Gibt es weitere Termine:

Deadlines, festgelegte Termine und andere relevante Ereignisse im weiteren Kontext: Produkt-Launches, Werbemaßnahmen, Versammlungen, Kunden-Events etc.

Was	Bis wann

Welche Weiterentwicklungen sind in den nächsten 2 Jahren vorgesehen?

Welches Wachstum wird in den nächsten 24 Monaten erwartet?

[Link zum Markdown](#)





Datenschutz, Sicherheit und Compliance

Wir haben uns beim Flying Circus entschieden, alle Aspekte rund um Sicherheit umfassend, aber auch pragmatisch zu behandeln. Der Flying Circus ist selbst vollständig nach ISO 27001 zertifiziert und wir begleiten unsere Kund*innen dabei, das notwendige Sicherheitsniveau auch im Betrieb der eigenen Anwendung zu erreichen.

Welche personenbezogenen Daten werden verarbeitet oder gespeichert?

Welche sensiblen personenbezogene Daten mit erhöhtem Schutzbedarf werden verarbeitet?

Die Verarbeitung von personenbezogenen Daten mit erhöhtem Schutzbedarf bedarf neben der Regelung einer AV eine Datenschutzfolgeabschätzung.

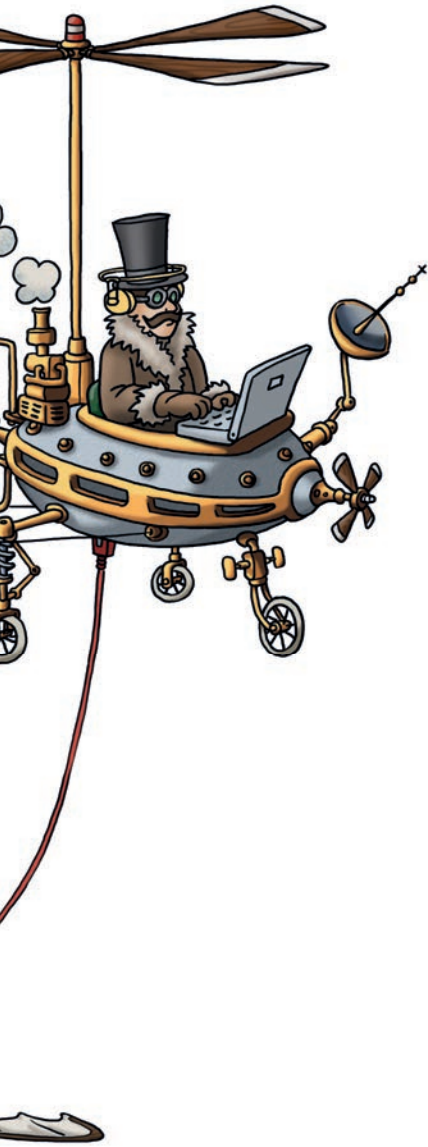
Welche Maßnahmen zur IT- und Datensicherheit sind implementiert?

Im allgemeinen sind die technischen und organisatorischen Maßnahmen aus den Anforderungen an die Sicherheit der Datenverarbeitung der EU-DSGVO bzw. dem BDSG zu berücksichtigen. Einige dieser Anforderungen setzt der Flying Circus durch Maßnahmen der Infrastruktur und Plattform vollumfänglich um – die übrigen Themen sind von uns weitestgehend vorbereitet, benötigen aber anwendungs- und kundenspezifische Betrachtungen, um sie vollständig zu erfüllen.

Vollständig umgesetzte Anforderungen: wie Zugangskontrolle, Transportkontrolle, Datenträgerkontrolle

Anforderungen, die weitestgehend umgesetzt sind, aber anwendungsspezifisch ergänzt werden müssen: wie Speicherkontrolle, Benutzerkontrolle, Zugriffskontrolle, Übertragungskontrolle, Eingabekontrolle, Wiederherstellbarkeit, Auftragskontrolle, Verfügbarkeitskontrolle, Datenintegrität.





Gibt es besondere Anforderungen an Backups?

Unser Standardschema sieht tägliche Backups aller virtuellen Festplatten vor. Dabei halten wir jeweils die Backups der letzten 7 Tage, 4 Wochen und 3 Monate vor. Optional können Backups auch stündlich angefertigt werden (und werden 24 h aufbewahrt). Für individuelle Varianten können wir miteinander sprechen. Die konkrete Uhrzeit, zu der Backups angefertigt werden, kann nicht festgelegt werden, ist aber gleichbleibend.

Werden Offsite-Backups benötigt?

Unsere Backups werden primär zur schnellen Wiederherstellung innerhalb des RZs aufbewahrt. Optional können Backups an unseren Backup-Standort in Halle gespiegelt werden.

Gibt es besondere Anforderungen an SSL-Zertifikate?

Standardmäßig werden Let's-Encrypt-Zertifikate mit automatischer Aktualisierung eingesetzt. Falls andere Zertifikate eingesetzt werden sollen oder Zertifikats-Pinning eine Rolle spielt, sollten wir dies wissen.

Welche Standards sind zu berücksichtigen oder welche Zertifikate sind zu erbringen?

Der Flying Circus ist umfassend nach ISO 27001 zur IT-Sicherheit zertifiziert. Wir evaluieren laufend, ob weitere Standards oder Zertifizierungen in Projekten relevant sind, benötigen hier aber gegebenenfalls entsprechende Vorlaufzeiten und Projektvolumina, um diese zu realisieren.

Gibt es spezifische weitergehende Sicherheitsanforderungen, die berücksichtigt werden müssen?

[Link zum Markdown](#)



Architektur und Handwerkliches

Viele Anwendungen folgen ähnlichen architekturellen Ansätzen und verwenden Bibliotheken und Frameworks, die bestimmte Themen einheitlich regeln. In Web-orientierten Projekten gibt es typischerweise eine Kombination von Webserver, Frontend, Backend, Datenbank und weiteren Diensten wie Message-Queues oder API-Anbindungen. Daten-fokussierte Anwendungen erfordern meist eine Cluster-Technologie zur Koordination von Jobs sowie unterschiedliche Datenbanken. Die folgenden Fragen sollen uns helfen einen Überblick zu bekommen, welche spezifischen Aspekte in diesem Projekt eine Rolle spielen und geben Denkanstöße, um typische Fallen zu vermeiden.

Aus welchen Software-Komponenten besteht die Anwendung, welche begleitenden Dienste werden benötigt und wie kommunizieren diese untereinander?

Wünschenswert wären ein Diagramm der wichtigsten Komponenten und eine möglichst vollständige Liste der konkreten Software.

Welche Bibliotheken, Programme oder Funktionen sind einen zweiten Blick wert, da diese im Betrieb besondere Anforderungen haben könnten?

Hier denken wir an Themen wie PDF-Generierung, exotische oder ggfs. veraltete Bibliotheken wie beispielsweise wkhtmltopdf, Bilderkennung (OCR), Kryptografie oder der Bedarf an speziellen CPU-Flags wie aes, sse oder Vektor-Operationen.

Werden große oder viele Dateien beim Upload, beim Download und in der Speicherung verarbeitet?

Wir empfehlen in diesem Kontext die Speicherung in unserem S3-kompatiblen Object Storage und raten dringend von der Verwendung von lokalen oder Netzwerk-Dateisystemen ab.

Um beim Einsatz von S3 spätere Probleme zu reduzieren, haben wir einen separaten Guide für Entwickler zur Strukturierung von Bucket-Namen und den Key-Namensräumen innerhalb der Buckets entwickelt.

Gibt es Eigenschaften der Anwendung, die eine horizontale Skalierung über mehrere Server verhindern? Wird NFS benötigt?

Typischerweise ist dies der Fall, wenn die Anwendung das Dateisystem für bewegliche Daten Sessions, Bilder, Caches o. Ä. verwenden. Wenn dies der Fall ist und keine Möglichkeit besteht, die Anwendung durch den Einsatz von S3, Redis, memcache und anderen Diensten in eine „Shared-Nothing“-Architektur zu überführen, muss ein erhöhtes Augenmerk auf die möglichen Strategien zur Skalierbarkeit gelegt werden.

Häufig werden Sessions lokal gespeichert. Diese sollten in Redis oder einer bereits genutzten Datenbank gespeichert werden. memcached sollte nicht für Sessions genutzt werden, da das Eviction-Verhalten für diesen Zweck nicht gut kontrollierbar ist.





Gibt es aufwendige Funktionen, die synchron bearbeitet werden und dadurch Threads oder Worker blockieren können?

Welche asynchrone Job-Queues kommen zum Einsatz und für welche Jobs werden diese genutzt?

Wir empfehlen den Einsatz spezialisierter Werkzeuge (z.B. Celery, RabbitMQ, oder ähnliches), um Job-Queues zu implementieren, da diese generell schon umfassende Möglichkeiten zum Monitoring und Management bei problematischen Jobs bieten. Allgemein sollte davon abgesehen werden, generische Datenbanken wie MySQL zur Implementation von Job-Queues zu verwenden, hier wäre als Ersatz Redis zu empfehlen.

Wie werden regelmäßige Tasks verwaltet?

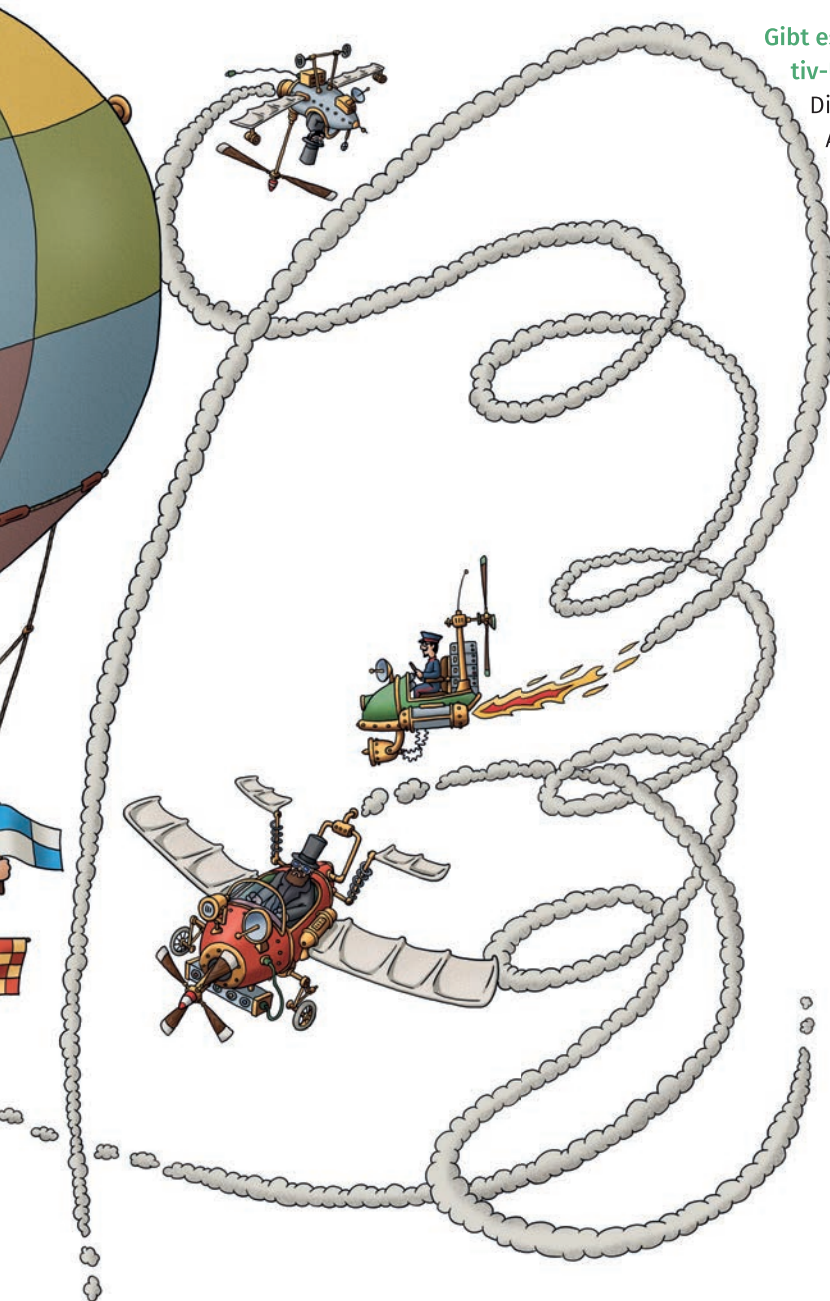
Regelmäßige Tasks sollten mindestens davor geschützt werden, parallel zu laufen (Locking), sollten mit Timeouts versehen werden und müssen erlauben, Fehler zu erkennen (Monitoring) und zu diagnostizieren (Logging). Bei fehlenden Alternativen wandeln wir Cron-Jobs in systemd-Timer um, sodass diese Anforderungen zumindest grundlegend erfüllt werden.

Gibt es Anforderungen an Plattform und Infrastruktur, um Mandantenfähigkeit zu realisieren?

Gibt es Gründe, die Anwendung in unterschiedliche Produktiv-Umgebungen aufzutrennen?

Die Umgebungen im Flying Circus bilden Einheiten, um Anforderungen an Sicherheit und Verfügbarkeit einheitlich zu verwalten. In komplexen Anwendungen, die mehrere Dutzend VMs verwenden, kann es sinnvoll sein, diese in unterschiedliche Einheiten aufzutrennen, um separate Zugriffsrechte zu pflegen oder unterschiedliche SLA-Level vorzusehen.

[Link zum Markdown](#)



Deployment und Konfiguration

Einer der wichtigsten Aspekte, mit denen wir ein Projekt erfolgreich machen können, ist die Etablierung eines automatisierten Deployments. Dabei ist es wichtig, dass die Anwendung dies auch gut unterstützt: neue Funktionen und Fehlerbehebungen können dann äußerst schnell und zuverlässig den Nutzer*innen zur Verfügung gestellt werden. Wenn hier gut gemeinsam gearbeitet wird, dann stellen auch mehrere Releases pro Tag keine Herausforderung für Entwickler*innen und SREs im DevOps-Ansatz dar. Im Gegenzug versteckt sich hier aber auch eins der größten Risiken: Lässt sich die Anwendung nicht zuverlässig vollständig automatisiert deployen, dann kann schnell eine Kette von negativen Auswirkungen das Projekt gefährden: Wichtige Änderungen können nicht zeitnah ausgerollt werden, es entstehen Fehler und Downtimes, Arbeitszeit von Projektleiter*innen, Entwickler*innen und SREs werden in komplizierten Abstimmungen, Bürokratie und Fehlerbehebungen verschwendet – statt das Projekt gemeinsam voranzubringen. Mit den folgenden Fragen wollen wir den Blick schärfen, um das Projekt rechtzeitig auf einen erfolgversprechenden Pfad zu lenken.

Welche Release-Artefakte gibt es und wie werden diese versioniert?

Alle Abhängigkeiten müssen umfassend definiert und gepinnt sein, sodass Builds und Deployments zuverlässig wiederholbar und auch zurückrollbar sind.

Bei Containern sollten die Versionsnummern alle auf konkrete Versionen gepinnt sein. Der Einsatz von „latest“ ist nicht zulässig.

Wie werden die Abhängigkeiten der Anwendung oder die Container-Images aktuell gehalten?

Der Flying Circus liefert Security-Updates für die Plattform automatisch aus. Beim Aktualisieren von Container-Images oder Anwendungsdependencies ist die Build- und Test-Chain sowie die Kompetenz der Entwickler gefragt und wir müssen uns abstimmen, wie dieser Aspekt gemeinsam gehandhabt wird.

Werden dauerhafte oder temporäre Umgebungen abweichend von „Staging“ und „Production“ benötigt?

Wir gehen von mindestens zwei getrennten Umgebungen aus: Staging und Production. Staging dient sowohl dazu, Änderungen in der Anwendung als auch die ganzheitliche Funktionalität mit regelmäßigen Updates unserer Plattform zu testen. In einigen Situationen haben wir die Erfahrung gemacht, dass es eine einzelne Staging-Umgebung nicht ausreicht, weil Entwickler*innen, SREs, Kund*innen und externe Tester*innen gleichzeitig darauf zugreifen müssen. Wir können je nach Bedarf weitere Umgebungen für Integrations-tests, Abnahmen oder auch Entwickler bereitstellen.

Gibt es derzeit Abläufe, die ein vollautomatisiertes Deployment verhindern und regelmäßig manuelle Eingriffe notwendig machen?

Unser Ziel ist es, dass wir gemeinsam Änderungen schneller und häufiger in Produktion nehmen, ohne dass bei jedem Release manuelle Arbeit notwendig ist. Dazu investieren wir gern in die Automatisierung der Pipeline sind, aber auch darauf angewiesen, dass nicht jedes Release manuelle Änderungen an Umgebungsvariablen, Secrets, Versionsnummern etc. benötigt, sondern dass sich dies auch geeignet automatisieren lässt.



Sind alle Konfigurationsparameter über Konfigurationsdateien oder Umgebungsvariablen steuerbar?

„Hard coded“ Konfiguration gilt es zu vermeiden, da sonst Verwechslungen zwischen Staging und Production entstehen können oder uns im Notfall schnelle Eingriffsmöglichkeiten fehlen. Beispiele für Konfigurationsparameter sind: Verbindungen zu Datenbanken oder externen Diensten, IP-Adressen und Ports, Benutzernamen, Datenbanknamen, Passwörter, API-Verbindungsdaten, API-Endpunkte, Domains, URLs, S3-Bucket-Namen, Keys, Listen-Adressen und Ports und viele weitere mehr.

Ein weiteres mögliches Anti-Pattern ist auch, dass Konfigurationsparameter in Datenbanken gespeichert werden. Dies kann im Speziellen auch Content-Management-Systeme betreffen, bei denen Links zu internen Dokumenten mit absoluter URL/Domain gespeichert werden, sodass bei der Replikation von Daten aus der Produktiv- in die Staging-Umgebung Verwirrung für die Anwender entstehen kann, der zu unbeabsichtigten Schäden in der Produktivumgebung führen kann.

Beim Einsatz von Docker ist zu beachten, dass die Konfiguration vollständig von außerhalb des Containers parametrisierbar sein muss und es keine abweichenden Images je nach Umgebung (Staging, Production) geben darf.

Wie werden Datenbank-Schemata migriert?

Automatisierte Deployments benötigen auch automatisierte, wiederholbare und kontinuierliche Migrationskripte für Schema-Migrationen. Hier ist auch zu beachten, ob Migrationen online laufen können und ob die Anwendung einen kontinuierlichen Roll-Out verträgt.

Welche Interaktionen für SREs sind vorgesehen?

Auf Systemebene kann es Skripte geben, die von SREs regelmäßig oder in besonderen Situationen benötigt werden. Hier wünschen wir uns eine Übersicht an Skripten oder Anleitungen, die bereits bekannt sind.

Aber: Ein regelmäßiges Eingreifen durch SREs um Standard-Geschäftsfälle zu behandeln ist nicht zulässig. Der flexible DevOps-Ansatz ersetzt nicht die Investition in wichtige Basis-Funktionalität.

(Spezifisch für PHP-Anwendungen): Kann die Anwendung mit einem „Eternal Opcache“ umgehen?

Eine Eigenart von PHP besteht darin, dass der Quelltext bei potentiell jedem Request vollständig neu geparkt wird. Das kostet wertvolle Zeit und Ressourcen und macht Anwendungen potentiell sehr langsam - zumal sich der Quelltext ohne Deployment nicht ändert. Wie setzen standardmäßig hier den sogenannten „Opcache“ ein, der den Quelltext nur einmal parst und dann für folgende Requests cached. Dieser Cache wird von uns standardmäßig nur bei Neustarts des Servers oder einem Deployment gelöscht. Manche Anwendungen können damit Schwierigkeiten haben, speziell wenn dynamisch Code generiert wird.

[Link zum Markdown](#)





Last- und Kapazitätsplanung

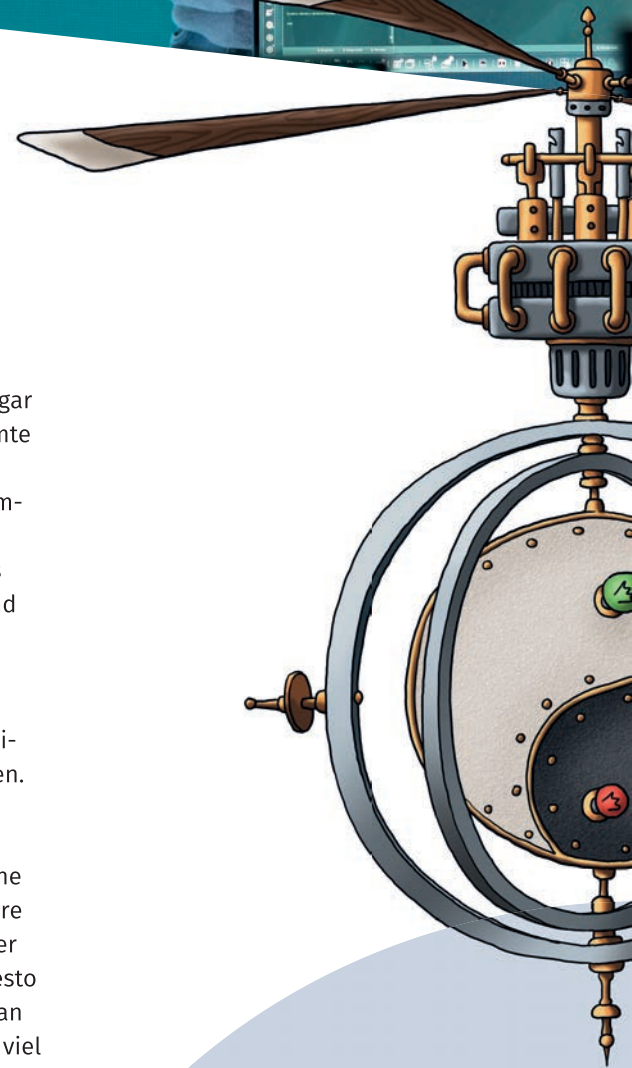
Wir wollen die für das Projekt notwendige Infrastruktur bereits im ersten Schritt möglichst passend dimensionieren: zu große bedeutet unnötige Kosten; zu kleine bedeutet das Risiko von schlechter Performance oder sogar Downtimes. Unsere Infrastruktur lässt sich zeitnah skalieren, aber bestimmte strukturelle Aspekte müssen wir rechtzeitig berücksichtigen, um dann angemessen flexibel reagieren zu können. Im gemeinsamen Austausch bestimmen wir gern aufgrund unserer Erfahrung anhand der vorhandenen Daten das konkrete Setup. Wenn ein Projekt bereits eine Historie hat, dann ist es sehr hilfreich, Daten aus verfügbaren Abrechnungen, Zugriffsstatistiken und Systemkonfigurationen einzubeziehen.

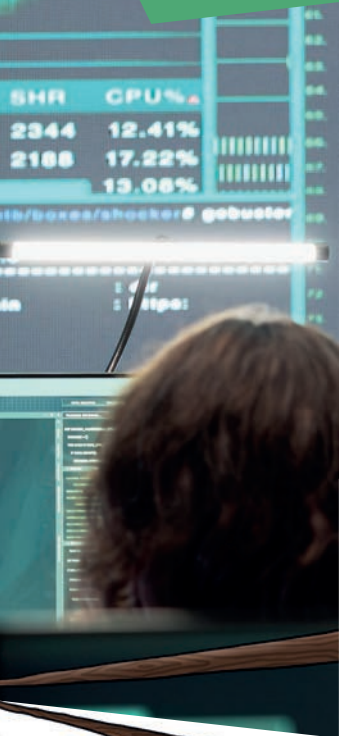
Für wie viele Nutzer*innen ist die Anwendung ausgelegt?

Je nach Anwendungsart kann es - zum Beispiel im Umfeld IoT - auch technische Clients geben, die ähnlich wie Nutzer*innen eingeplant werden sollten.

Mit wie vielen gleichzeitigen Nutzer*innen wird gerechnet?

Gleichzeitige Nutzung kann unterschiedliche Auswirkung haben: viele offene Verbindungen, hohe Zugriffszahlen, höherer Speicherverbrauch oder andere Ressourcen (Job-Queues), die zum Flaschenhals werden. Je genauer wir hier die Anforderungen und die reale Situation der Implementation kennen, desto besser können wir einschätzen, ob ein einfacheres Setup (das nicht spontan skalieren muss) oder ein anspruchsvolleres Setup (das sehr plötzlich sehr viel skalieren muss) benötigt wird. Gleichzeitig sollte das auch den Entwickler*innen entsprechend helfen zu planen, welche Funktionen und Code-Pfade hier möglicherweise ein gutes Skalieren begünstigen oder erschweren könnte.





Mit wie vielen Zugriffen wird im Monat gerechnet?

Um die Last zu planen, helfen uns sowohl Zugriffszahlen, die einzelne URLs widerspiegeln (wie man sie aus Webserver-Logs erhält) oder auch größere Zahlen zu Seitenabrufen (wie man sie aus Analyse-Tools erhält). Auch unique vs. wiederholte Besuche und Abrufe sind interessant.

Welche Datenmengen werden gespeichert und wie sieht das Wachstum bei konstanter und steigender Nutzung aus?

Hier wäre wissenswert, in welcher Größenordnung (wenige Megabyte, einige Gigabyte, Dutzende oder Hunderte Gigabyte oder mehr) welche Arten von Daten in welchen Funktionen oder Teilsystemen anfallen. Auch ob die Daten im Verlauf der Nutzung wieder gelöscht werden und sich dadurch ein Sättigungspunkt einstellen sollte oder ob die Daten potenziell unendlich lange aufbewahrt werden und dadurch mit potenziell unendlichem Speicherbedarf zu rechnen ist.

Insbesondere wenn es sich um besonders große oder zahlreiche Bilder, Videos, Dokumente, Archive handelt, sollten wir gemeinsam schauen, wie diese gehandhabt werden.

Welche Datenmengen werden durchschnittlich pro Monat aus dem oder in das Internet übertragen?

Um die Kosten gut abzuschätzen, benötigen wir eine grobe Zahl in Gigabytes (GiB) oder Terabytes (TiB) welche Daten in Summe übertragen werden. 100 GiB stehen jedem Projekt monatlich ohne zusätzliche Kosten zur Verfügung.

Daten, die zwischen VMs innerhalb eines Rechenzentrums übertragen werden, sind immer inklusive.

Welche Spitzenauslastung im Traffic zum Internet werden erwartet?

Traffic im Internet wird in MBit/s oder GBit/s angegeben. Hilfreich sind Erfahrungswerte, um mögliche Engstellen frühzeitig abschätzen zu können.

Welche spezifischen Anforderungen oder Erfahrungen in Bezug auf CPU, RAM und Storage liegen vor?

Bei Projekten, die produktiv betrieben werden oder die bereits während der Entwicklungs- oder Testphase entsprechende Erkenntnisse erbracht haben.

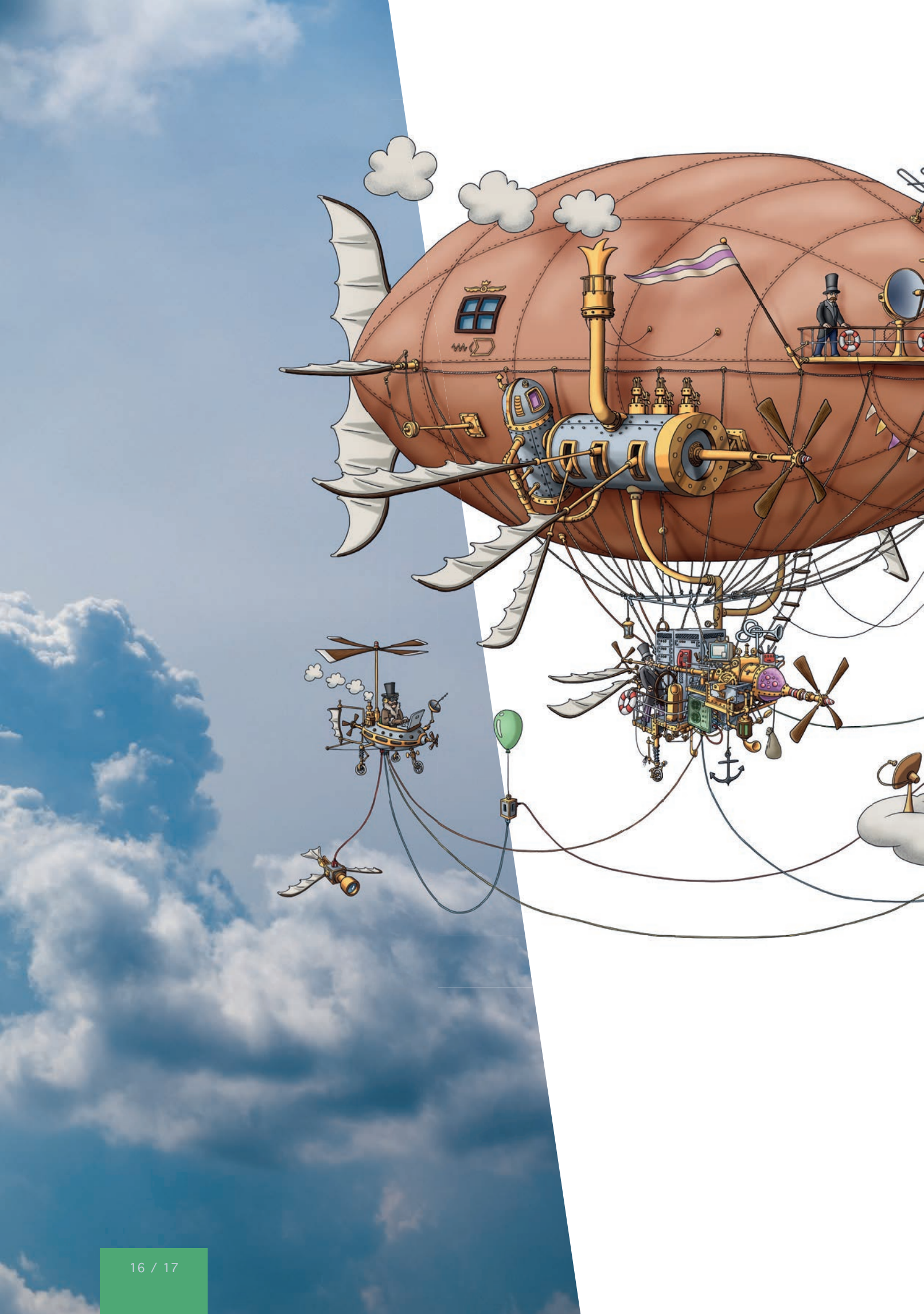
Neben der Menge an CPUs und Arbeitsspeicher ist hier auch die Größe und die Art der Storages relevant. Sind normale schnelle Disks (250 IOPS) oder schnellere Disks (10.000 IOPS) angemessen?

Welche regelmäßigen saisonalen Lastprofile und Lastprognosen spielen eine Rolle?

Lastprofile unterscheiden sich häufig je nach Tageszeit, aber auch über Wochen oder Monate hinweg können starke Unterschiede bestehen, zum Beispiel durch die Aussendung von Newslettern, Pushnachrichten, Live-Events oder im Zusammenhang mit Werbung und Marketing-Aktionen.

[Link zum Markdown](#)







Performance und Skalierbarkeit

Neben einer passenden Auslegung der Infrastruktur gibt es unterschiedliche Detailspekte von Anwendungen, die beeinflussen, wie wir im Fall von steigender Last reagieren können. Bestimmte Funktionen und Komponenten müssen bei Überschreiten von kritischen Grenzen neu ausgelegt und anders aufgebaut werden. Horizontale Skalierung liefert auf Anwendungsebene hier häufig die besten Hebel – gleichzeitig hilft es nur wenig eine Funktion, die aufgrund ungünstiger Algorithmen oder Datenbankstrukturen beruht, auf mehrere Server zu verteilen.

Welche Performance-Probleme sind bekannt?

Wir können hier proaktiv tätig werden, wenn wir grob wissen, bei welchen Funktionen oder Komponenten und in welchen Situationen es zu Performance-Problemen kommt.

Welche langlaufenden Requests oder Jobs gibt es?

Langlaufende Requests können zu Verfügbarkeitschwierigkeiten führen, wenn diese Worker-Prozesse blockieren. Hier können wir zum einen darauf achten, diese Schwierigkeiten durch geschicktes Load-Balancing nicht eskalieren zu lassen.

Gleichzeitig können wir, wenn diese Langläufer in Summe zu Problemen führen, gemeinsam diagnostisch vorgehen und beispielsweise Profiling und Datenbank-Query-Analysen anwenden, um fehlende Indices oder ungünstige Datenstrukturen und Algorithmen zu erkennen und mögliche Lösungsstrategien aufzeigen.

Welche Caching-Konzepte werden benötigt?

Für Anwendungen, die weitestgehend Lesezugriffe statt Schreibzugriffe verzeichnen, empfehlen wir den Einsatz entsprechender Caching-Werkzeuge. In den meisten Fällen ist hier Varnish das Werkzeug der Wahl. Gleichzeitig gibt es gegebenenfalls weitere Caching-Anforderungen von Teilkomponenten der Anwendung, diese sollten wir ebenfalls klären.

Wie verhält sich die Anwendung unter Last und einem Cold-Cache-Szenario, also bei spontanem Verlust der Caches?

Wie werden vorhandene Caches invalidiert?

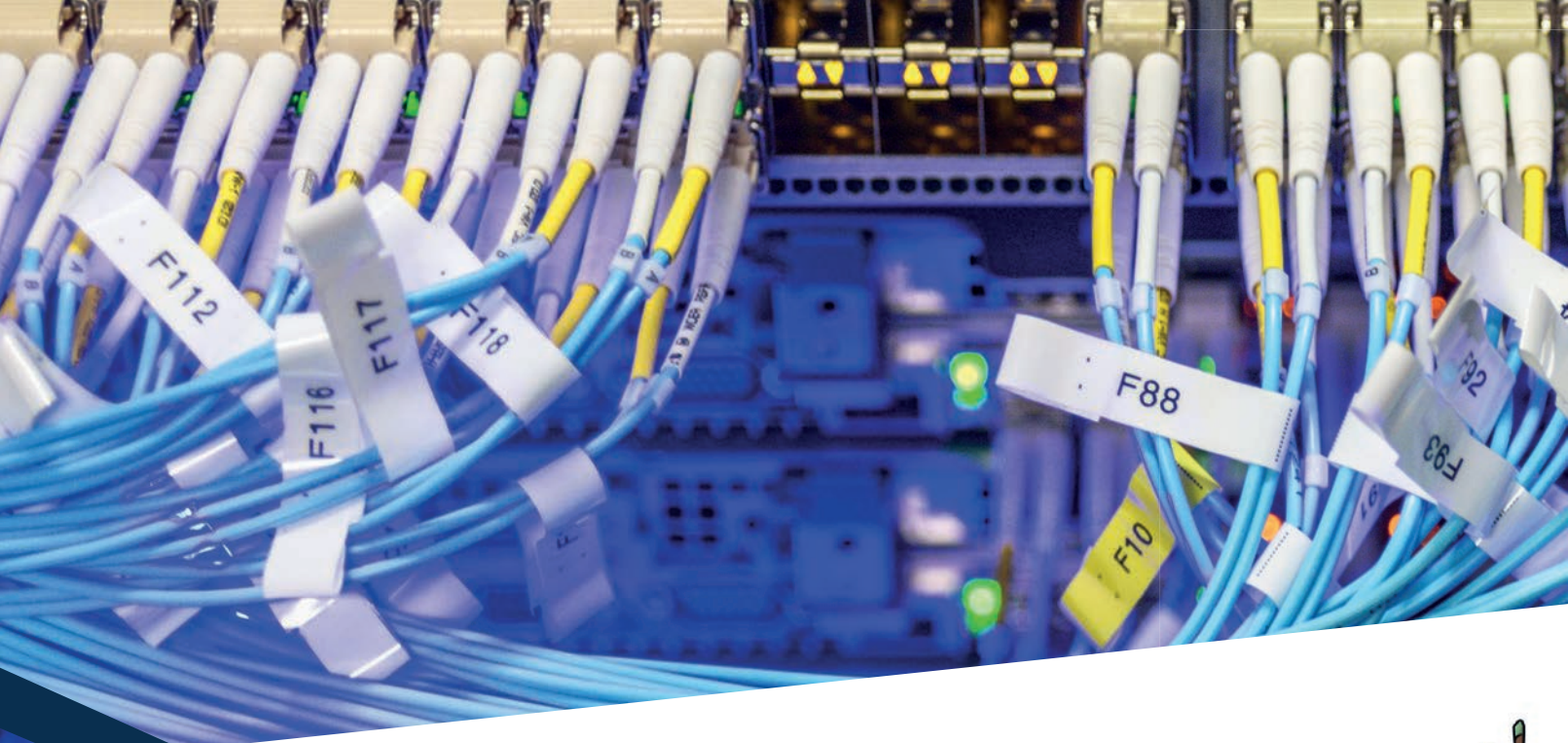
Welche Integration mit spezifischen Caching-Regeln muss berücksichtigt werden?

Varnish kann in einigen Situationen mit sehr allgemeinen Cache-Regeln verwendet werden, wenn die Anwendung entsprechende Header ausliefert und (siehe oben) entsprechende Invalidierungs-Nachrichten schickt.

Gleichzeitig können wir spezielle Regeln vorsehen, um komplexere Caching-Strategien zu implementieren, beispielsweise um korrektes Caching auch für angemeldete Benutzer sicherzustellen.

[Link zum Markdown](#)





Netzwerk, Internet, externe Dienste und APIs

Das Netzwerk im Flying Circus entworfen worden, um unterschiedlichste Anwendungen sicher und performant zu betreiben, sodass Entwickler sich darüber weitestgehend keine Gedanken machen müssen. Gleichzeitig gibt es aus unserer Erfahrung eine Reihe von neuralgischen Punkten, die frühzeitig geklärt werden sollten, um böse Überraschungen zu vermeiden.

Welche Anforderungen bestehen hinsichtlich der Netzwerkarchitektur?

Die Netzwerk-Architektur im Flying Circus sieht eine Trennung zwischen externem und internem Traffic vor und beinhaltet Firewalls an der RZ-Grenze und auf jeder einzelnen virtuellen Maschine.

Gleichzeitig gibt es manchmal relevante Anforderungen in Bezug auf externe Firewalls, NAT-Traversal oder öffentliche IP-Adressen.

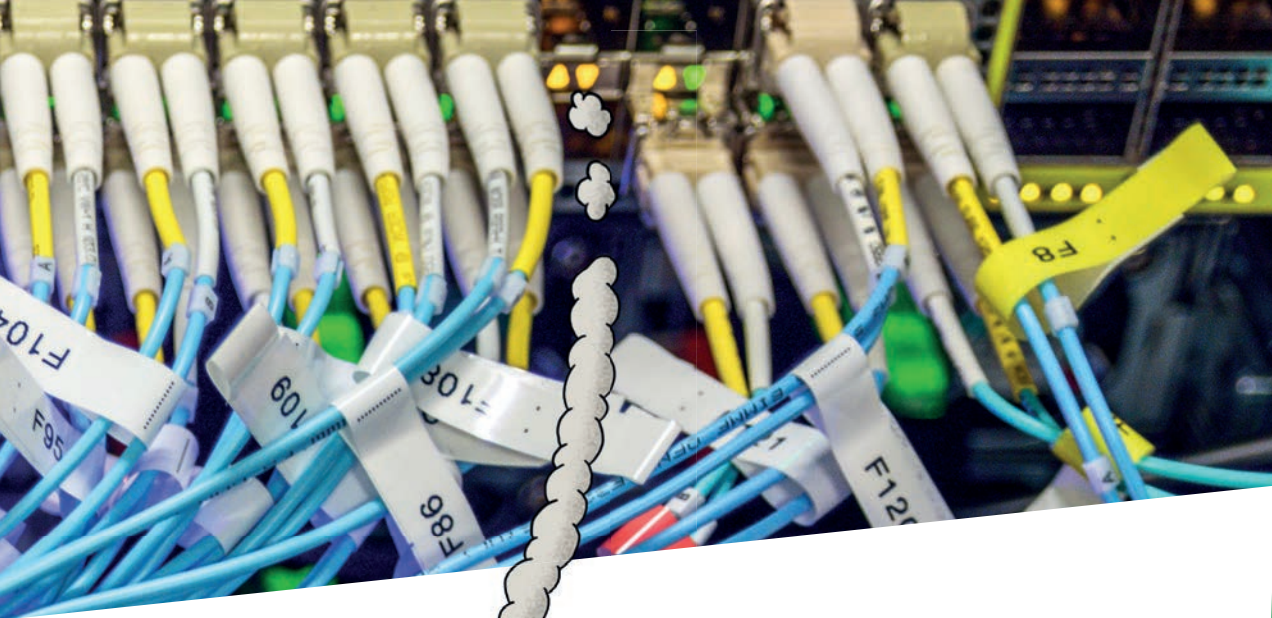
Spricht etwas gegen die Verwendung von IPv6?

Wir bieten alle Kundendienste standardmäßig per Dual-Stack (IPv4 + IPv6) an. Zwar ist in Deutschland die Nutzung von IPv4 bisher uneingeschränkt für Endkunden möglich, so bietet IPv6 jedoch inzwischen speziell bei mobilen Nutzer*innen deutliche Vorteile in Bezug auf Geschwindigkeit und Zuverlässigkeit.

Welche VPN-Verbindungen werden benötigt?

Um Nutzer*innen, Standorte oder APIs anzubinden, können unterschiedliche Technologien zum Einsatz kommen. Wir empfehlen grundsätzlich je nach Einsatzzweck Wireguard und OpenVPN. Wenn keine anderen Optionen zur Verfügung stehen, unterstützen wir auch IPsec – allerdings ist dies häufig mit Schwierigkeiten bei der Einrichtung und teilweise auch – je nach Unterstützung durch die Gegenstelle – Herausforderungen bei eventuell notwendiger Problembehebung im Fehlerfall verbunden.





Wer betreibt den relevanten DNS-Server?

DNS-Betrieb bieten wir vom Flying Circus als Inklusiv-Service mit umfassendem Self-Service-UI an. Allgemein sind unsere SLAs umfassender abgedeckt, wenn wir im Fehlerfall auch bei DNS-Themen 24/7 handlungsfähig sind und nicht auf externe Hilfe angewiesen sind, die gegebenenfalls nicht passend zum gewählten SLA reaktionsfähig ist.

Falls Domains nicht komplett von uns verwaltet werden sollen, können wir auch anbieten, eine delegierte Sub-Zone zu verwalten und das Thema dadurch etwas zu entschärfen.

Welche Arten von E-Mails werden versendet oder empfangen?

E-Mail ist ein in sich breites Thema und kann hier nur angerissen werden. Wir unterscheiden grundsätzlich den Versand von Massenmails und Transaktionsmails oder den Bedarf an "Enduser-Mailservern".

Welcher Typ Mailserver soll eingesetzt werden?

Wir können externe Mailserver anbinden, APIs von Dienstleistern verwenden oder ein Mailserver beim Flying Circus betrieben werden. Dies hat auch jeweils Auswirkungen, wie die Anbindung von den Entwicklern realisiert ist (SMTP, IMAP, API) und welches Monitoring und welche Handlungsmöglichkeiten im Fehlerfall dem SRE-Team zur Verfügung stehen.

Werden dynamische oder statische Mail-Accounts verwendet?

Wer ist für die korrekte Konfiguration von E-Mail-Domains im DNS zuständig?

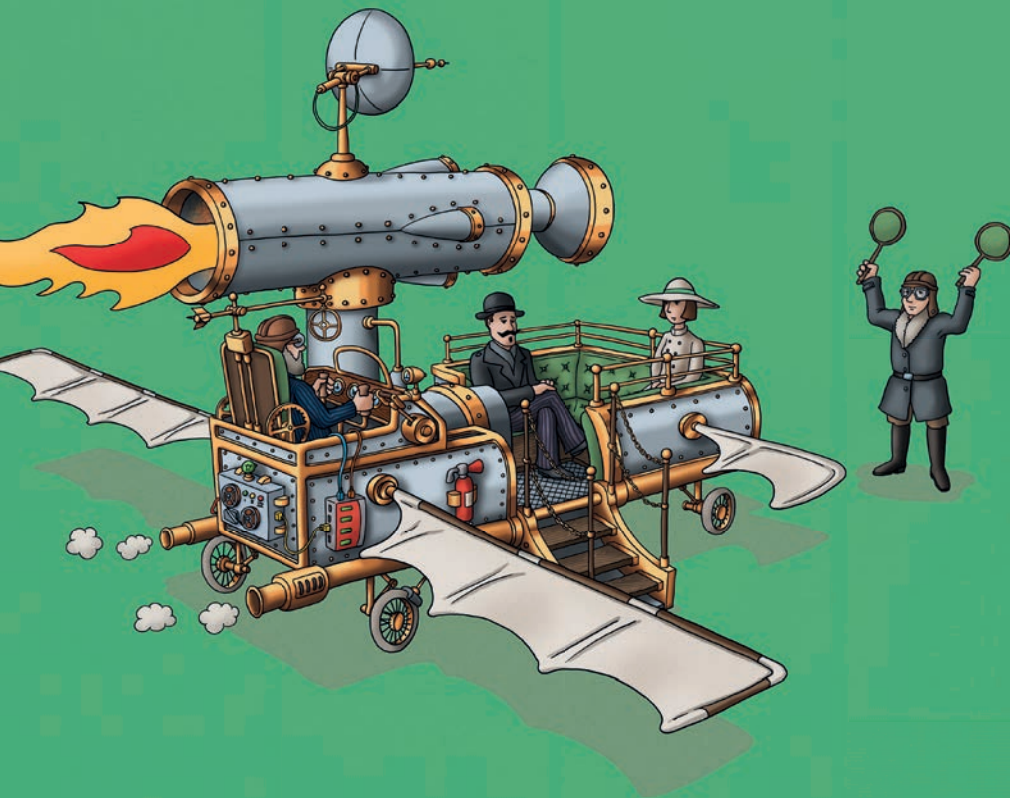
Um eine gute Zustellrate für versendete E-Mails zu erzielen, müssen eine Reihe spezifischer Domain-Authentifizierungsprotokolle (DMARC, DKIM, SPF) gesetzt werden. Dies geschieht im DNS-Server. Wenn der Flying Circus den DNS für die relevanten Domains verwaltet, übernehmen wir dies.

Welche dritten Parteien werden direkt oder indirekt in die Anwendung eingebunden?

Abseits von den Nutzerinnen und Nutzern der Anwendung kann es sein, dass eingehende oder ausgehende API-Calls eine Rolle spielen, die besonders betrachtet werden sollten. Speziell wenn diese synchron oder im Kontext von Nutzeranfragen stattfinden.

[Link zum Markdown](#)





Monitoring, Zuverlässigkeit, Business Continuity

Grundsätzlich sollen Systeme im Idealfall natürlich keine Fehler aufweisen. Gleichzeitig ist die Erkenntnis aus dem Betrieb komplexer Systeme: Fehler passieren. Man kann deshalb nur zu einem Teil Fehler verhindern und muss sich auf der anderen Seite darauf vorbereiten, mit Fehlern umzugehen. Wir wägen zwischen diesen beiden Maßnahmen ständig ab, um Stabilität und Flexibilität in Einklang zu bringen. Um für Fehlerfälle vorbereitet zu sein, klären wir die an das System gestellten Erwartungen, stellen sicher, dass wir im Ernstfall handlungsfähig sind, wollen dass kleinere Fehler möglichst keine größeren Fehler nach sich ziehen und geben Anregungen wie auch Entwickler*innen dazu beitragen können.

Welche Fehler haben in der Vergangenheit zu Problemen, Downtimes oder Störungen geführt?

Welche drei Aspekte haben im laufenden Betrieb bisher am meisten Herausforderungen bereitet?

Welche potenziellen Probleme würden am meisten Sorgen bereiten?

Reagiert die Anwendung bei kurzen Störungen oder unklaren Fehlern korrekt und erholt sich selbstständig oder „stürzt sauber ab“?

Manche Frameworks neigen dazu, beispielsweise bei kurzen Verbindungsverlusten zur Datenbank ohne externen Eingriff in einen teilweise schlecht erkennbaren inkonsistenten Zustand zu geraten.

Unser grundsätzlicher Ansatz zur Optimierung der „Mean Time To Recovery“ sieht vor, dass es in komplexen Systemen immer zu kurzzeitigen Fehlern kommen kann und die Anwendung als Ganzes solche Zustände ohne manuellen Eingriff überstehen kann.

Wie wir mit Vorgängen umgegangen, die nicht durch Transaktionen geschützt sind und in inkonsistente Zustände geraten können?

Speziell bei geplanten oder ungeplanten Reboots oder spontanen Crashes von Anwendung oder Betriebssystem sollte es nicht generell notwendig sein, manuell einzugreifen, um das System wieder in einen betriebsfähigen Zustand zu versetzen.

Wie kann die Kommunikation mit externen Diensten beobachtet werden?

Anfragen an externe Dienste und APIs müssen in den Logs ersichtlich sein und Request-Beginn und Request-Ende mit eindeutigen IDs und Zeitstempeln ausweisen. Gleichzeitig sollte es eine Option geben, die Nutzdaten von Requests an externe Dienste bei Bedarf zu loggen.

Wie reagiert die Anwendung, wenn externe Dienste down sind oder undefiniert nicht reagieren?

Ein sauberes Verhalten der Anwendung wäre gegeben, wenn „graceful degradation“ dazu führt, dass sich zum einen keine Requests unendlich lang aufstauen und dadurch andere Teile der Anwendung blockieren und zum anderen, dass Nutzer zeitnah eine Antwort mit entsprechendem Hinweis, dass bestimmte Funktionen nicht zur Verfügung stehen erhalten. Gleichzeitig sollten Nutzerdaten in diesem Fall nicht verloren gehen.

Welche Endpunkte für Metriken und Monitoring stehen in der Anwendung zur Verfügung?

Idealerweise stellt die Anwendung einen Prometheus-kompatiblen Endpunkt per HTTP zur Verfügung, von dem wir Metriken rund um das Anwendungsverhalten in unsere Telemetrie-Infrastruktur integrieren können.

Außerdem benötigen wir einen Endpunkt, der per HTTP angesprochen werden kann und beispielsweise mit JSON-Output antwortet und maschinen- und menschenlesbare Statusinformationen bereitstellt. Es sollten spezifische Diagnosen, wie beispielsweise „Datenbank nicht erreichbar - Fehlermeldung: [Details]“ enthalten sein.

Wie kann die Bereitschaft der Anwendungsprozesse nach Start abgefragt werden?

Readiness-Probes also Abfragen zur Bereitschaft dienen dazu um Prozessmanagern die Unterscheidung korrekter oder fehlerhafter Starts zu ermöglichen und um zu verhindern, dass Load-Balancer Anfragen an Ports weiterleiten, die noch nicht antwortbereit sind.

Welcher Support-Level und Reaktionszeiten sind erwünscht?

Der Flying Circus bietet zwei Service-Level an: Guided und Managed.

Im Guided-Service-Level unterstützen wir bei der Migration, Konfiguration und Wartung auf unserer Plattform, entwickeln gemeinsam einen Migrationsplan und definieren den Ablauf. Die Umsetzung übernehmen Sie.

Im Managed-Service-Level übernehmen wir die vollständige Betriebsverantwortung, entwickeln ein voll-automatisiertes Deployment Ihrer Anwendung auf unserer Plattform und betreiben Ihre Anwendung stabil und sicher.

In beiden Varianten legen Sie fest, welche maximalen ungeplanten Downtimes monatlich zulässig sind und ob wir bei anwendungsspezifischen Fehlern während der normalen Geschäftszeiten (Mo.-Fr., 8-16 Uhr, 3h Reaktionszeit), zu erweiterten Zeiten (Mo.-So., 7-21 Uhr, 1h Reaktionszeit) oder 24/7 (1h oder 15 Min.) reagieren.

In welchem Zeitfenster dürfen automatisierte Wartungsarbeiten durchgeführt werden?

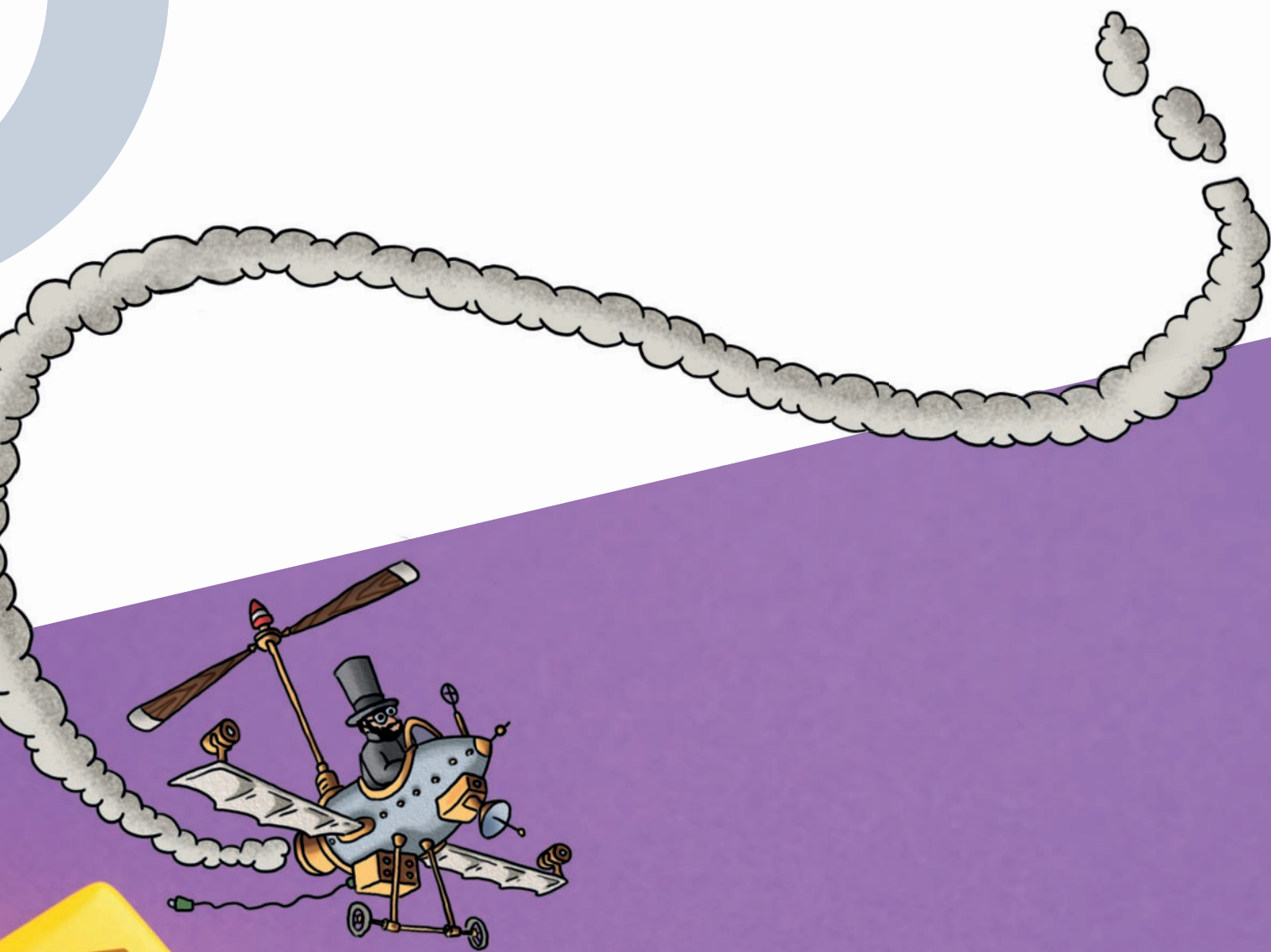
Unser Standardzeitfenster liegt zwischen 22 Uhr und 5 Uhr nachts und meldet notwendige Wartungsarbeiten mindestens 12 Stunden vorher per E-Mail an die technischen Ansprechpartner. Das Zeitfenster kann frei an eigene Bedürfnisse angepasst werden, darf aber nicht zu kurz gefasst sein, da Wartungsarbeiten über mehrere Maschinen hinweg nacheinander eingeplant werden. Außerdem sollte die Vorankündigungszeit nicht mehr als wenige Tage betragen, da sonst Sicherheitsupdates nicht zeitnah eingespielt werden können.

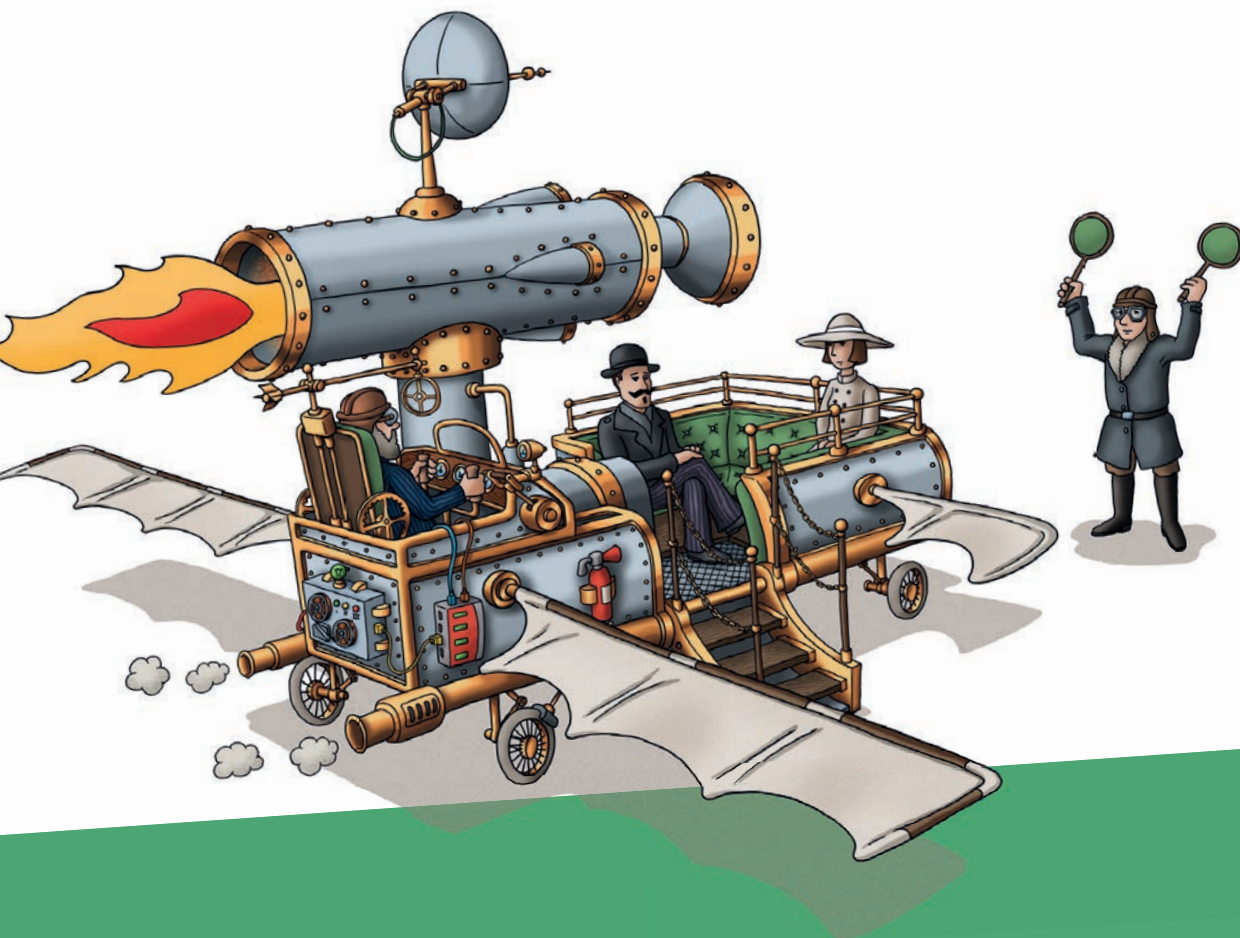
Welche Schritte sind zur Vorbereitung und Nachbereitung von Wartungsfenstern notwendig?

Es kann erforderlich sein, bestimmte Prozesse vor Unterbrechungen zu schützen, die Anwendung in einen Wartungszustand zu versetzen oder wieder zu aktivieren. Unser Wartungsmanagement erlaubt hier eine Automatisierung. Manuelle Eingriffe sind für regelmäßige Wartungsfenster nicht zulässig.



[Link zum Markdown](#)





Haben Sie Fragen oder möchten Sie Ihr Projekt mit einem Experten besprechen?

Wir helfen gerne weiter unter 0345 219 401-11

Flying Circus Internet Operations GmbH

Leipziger Str. 70/71
06108 Halle (Saale)
Deutschland

Kontakt

mail: mail@flyingcircus.io

fon: +49 345 219 401 0

fax: +49 345 219 401 28