DevOps-First oder Container-First — Strategien für den erfolgreichen Einstieg

Motivation

Das Geschäftsziel Nummer Eins in der digitalen Transformation ist heute Businessagilität. Sie hat in erster Linie das Ziel, neue Produkte schneller auf den Markt zu bringen und damit Wettbewerbsvorteile zu erlangen. Plattformökonomie, neue Geschäftsbeziehungen und attraktive neue Betriebsmodelle haben dazu geführt, dass es sich bei den Themen DevOps und Containertechnologien nicht mehr um einen Hype handelt. Der immer stärker umkämpfte Markt um IT-Arbeitskräfte, die ein optimales, modernes Arbeitsklima fordern und mit aktuellen Technologien arbeiten wollen, trägt dazu bei, dass es eigentlich keine Alternative mehr gibt.

Dies stellt kein großes Problem dar, wenn es sich um ein Start-up handelt oder neue Geschäftsbereiche gegründet werden. Wie sieht es aber aus, wenn man mit einer historisch gewachsenen Softwarearchitektur und -Infrastruktur oder auch einer klassisch aufgestellten Softwareentwicklungsabteilung auf die Infrastruktur-Technologien Container und Cloud wechseln möchte?

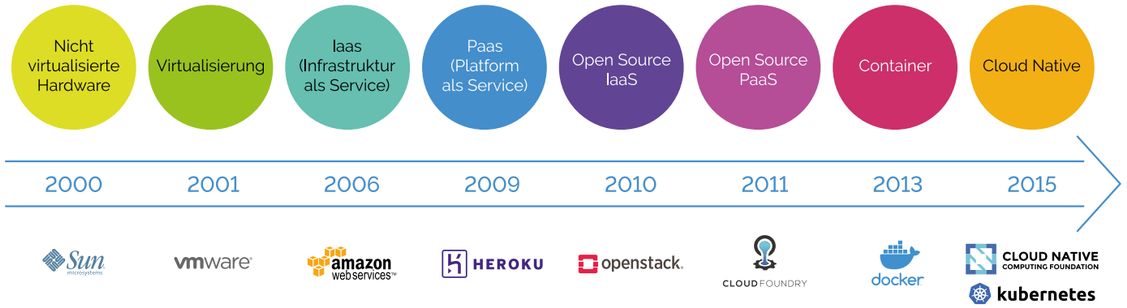
Dieser Artikel soll einen soliden Überblick geben, wie ein Einstieg oder Umstieg gelingen kann und welche Strategien existieren, um die Vorteile von DevOps und Container-Technologien zu nutzen

DevOps-First oder Container-First

Der Begriff DevOps tauchte das erste Mal im Jahr 2009 auf der Velocity Conference in San José auf, wo ihn Patrick Debois einführte. Lässt man den Strategieaspekt dieses Artikels einen kleinen Augenblick außen vor, so macht zeitlich gesehen DevOps das Rennen.

Container-Technologien kamen nämlich erst mit Docker im Jahr 2013 auf, und dieser Name hat sich – analog zu Tempo bei den Taschentüchern – bis heute gehalten. Zwar gab es schon erste Ansätze in den UNIX-Betriebssystemen, Teilbereiche der Dateisysteme zu isolieren oder ein Betriebssystem innerhalb des Betriebssystems zu starten, allerdings spricht man hier noch nicht von Containern.

Abbildung 1:
Cloud-native – Timeline
(Quelle: Cloud Native Foundation)



Die DevOps-Bewegung mit den Schwerpunkten Automatisierung, Metriken und Kultur hat vollkommen neue Sichtweisen auf die Softwareentwicklung mit sich gebracht. Sie sorgte außerdem dafür, dass die Entwicklung der Container-Technologien und eines Ökosystems um diese neue Form der Virtualisierung herum einen entscheidenden Impuls bekam, welche sich deshalb sehr schnell zur heutigen Produktvielfalt und -reife entwickelten.

Man kann DevOps also als Wegbereiter für die heutigen Cloud- und Container-Plattformen betrachten. Dieser Kulturansatz lebt von schnellen Feedback-Zyklen, einer einheitlichen Arbeitsweise mit standardisierten Werkzeugen und umfassender Automatisierung. Virtuelle Maschinen und Hypervisor waren der erste Schritt, und die neuen, leistungsfähigen Plattformen bringen eine erhebliche Optimierung mit sich. Erste Cloud- und „As-a-Service"-Angebote gab es bereits 2006, aber der Ruf nach Self-Services und besseren Optionen, die Konfiguration der Infrastruktur zu automatisieren und wie Code zu behandeln, beschleunigte fortan die Bestrebungen, Entwicklung und Betrieb näher zusammenzubringen.

Alles in allem muss man sich eingestehen, dass all diese Technologien und auch die DevOps-Bewegung noch relativ jung sind und dass sich Standards erst langsam konsolidieren.

DevOps-First-Strategien

Am vielversprechendsten und wirksamsten sind Strategien, die auf DevOps-Prinzipien aufbauen und damit auch die Organisation nachhaltig beeinflussen. Technologien sind oft nur Katalysatoren, die den Einstieg erleichtern und vor allem eine größere Flexibilität und damit Umsetzungsgeschwindigkeit verleihen.

Kulturwandel als Initialzündung

Mit der Einführung von DevOps in der Organisation ist ein Kulturwandel verbunden. Nimmt man ihn ernst und stattet ihn mit einer ordentlichen Portion

Engagement vonseiten des Managements aus, ist das eine solide Basis für den erfolgreichen Start in die aktuelle dynamische und hoch automatisierte wWelt der Softwareentwicklung – oft initiiert durch Container-Technologien.

Abbildung 2 zeigt wesentliche Faktoren, die man mit dem Begriff DevOps verbindet und es wird deutlich, dass sich die Aspekte Entwicklung und Betrieb auf viele Bereiche der DevOps-Kultur verteilen.

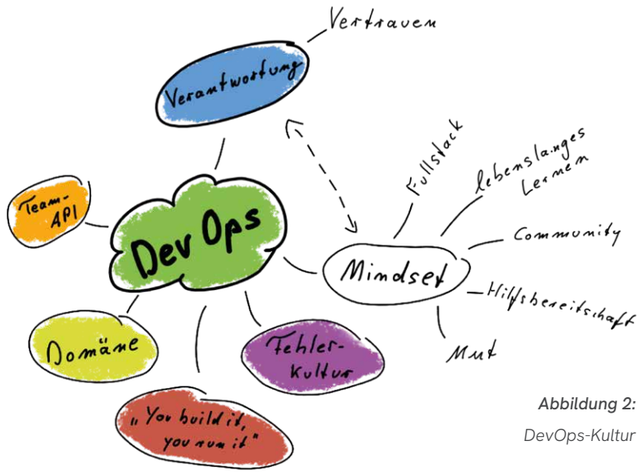


Abbildung 2:
DevOps-Kultur

Verantwortung und Mindset

Verantwortung gab es in der Softwareentwicklung schon immer, aber mit den zusätzlichen Aufgaben aus dem Ops-Umfeld kommt eine neue Dimension hinzu. Die klassischen Entwickler aus dem Mainframe-Umfeld werden hier müde lächeln. All die Entwickler hingegen, die sich in der verteilten Post-Mainframe-Welt mit schwergewichtigen Applikationsservern, mit zentralisierten Datenbank-Monolithen wie z. B. Oracle-Enterprise-Servern, mit z/DB2 oder mit Serverclustern und dedizierten Loadbalancern herumschlagen müssen, können ein Lied davon singen, was Operations bedeutet. Da ist es nicht verwunderlich,dass große Vorbehalte und Ängste geäußert werden, wenn der Begriff DevOps fällt.

Dabei hat sich auch hier die Welt weitergedreht. Mit Tools für die Automatisierung der Infrastruktur, etwa Puppet oder Ansible, gibt es bereits langbewährte Unterstützung, um Betriebstätigkeiten mit klassischer Infrastruktur den Schrecken zu nehmen und den Start in DevOps zu ermöglichen. All diese Automatisierungen lassen sich 1:1 in die Container- und Cloud-Welt übertragen und optimieren. Damit ist bereits eine große Hürde auf dem Weg zu mehr Verantwortung für Entwicklung und Betrieb genommen, denn jetzt bekommen die Softwareingenieure auch die Sicherheit, zu jeder Zeit in jeder Umgebung bis hin zur Produktion schnell und mit reproduzierbarer Qualität zu deployen.

Diese Kultur erfordert darüber hinaus ein gewisses Mindset. Wir sprechen heute nicht mehr von Softwareentwicklern, sondern von Softwareingenieuren mit Fullstack-Qualitäten. Besser geeignet ist in diesem Zusammenhang der Begriff des T-Shaped Professional. Damit bezeichnet man einen Generalisten mit Steckenpferd, der viel Breitenwissen besitzt und in vielen Bereichen gute Entscheidungen treffen kann, aber trotzdem noch Spezialisierungen mitbringt und anspruchsvolle Probleme lösen kann.

Technologien sind nur eine Seite der Medaille. Mut, Hilfsbereitschaft und die Akzeptanz des lebenslangen Lernens eröffnen den Softwareingenieuren die Möglichkeit, in diesem hochvolatilen technologischen Umfeld all die neuen Skills zu erwerben und in der täglichen Arbeit gewinnbringend einzusetzen. Dieser Aufwand wird sehr oft unterschätzt.

Im Schnitt sind zwanzig Prozent der Arbeitszeit für die Weiterbildung und die aktive Teilnahme an Communities zu veranschlagen Das beeinträchtigt zwar auf den ersten Blick die Performance des Teams, stellt aber durch bessere Verteilung des Wissens und offene Arbeitskultur gewaltige Gewinne in Aussicht.

Domäne – „You build it, you run it“

Verantwortung zu übernehmen kann eine Herausforderung werden, wenn die Anwendungen zu groß oder sehr viele eng gekoppelte Applikationen entwickelt oder betrieben werden sollen. In der Vergangenheit beantwortete man diese Herausforderung mit sehr langen Release-Zyklen, zeitraubender Qualitätssicherung und einer aufwendigen Fehlerbehebung, falls doch Fehler übersehen wurden.

DevOps allein hilft uns hier leider nicht weiter. Es sind Änderungen in der Softwarearchitektur angesagt, um die oft monolithische Anwendung in überschaubare und möglichst lose gekoppelte Bereiche aufzuteilen, damit kleinere Teams diese vollständig adaptieren und verantworten können.

Das Muster des Big Ball of Mud trifft man leider allzu häufig in historisch gewachsenen Softwaresystemen an. Mit etwas Voraussicht wurde die Software modular aufgebaut, doch oft ist auf den ersten Blick keine Trennung in einzelne fachliche Bereiche – die sogenannten Domänen – sichtbar und eine große Entwicklermannschaft schlägt sich mit einer schlecht strukturierten Codebasis, vielen Abhängigkeiten, organisatorischen Wissensinseln, schlechter Dokumentation und geringer Testabdeckung herum.

Domain-Driven Design bietet sich an, um Entwickler und Fachverantwortliche mit Methoden wie Event Storming oder Domain Storytelling in einem Raum zusammenzubringen und viel Neues über die Geschäftsdomäne zu erfahren. Die Domäne lässt sich dann in kleinere Sub-Domänen und sogenannte Bounded Contexts aufteilen, die schließlich mit kleinen Teams autonom weiterentwickelt werden können. Mit diesem Redesign öffnet man die Tür zum eigenverantwortlichen Handeln der Teams. Erst jetzt wird DevOps möglich, und der Weg ist frei für „You build it, you run it“.

Team-APIs

Mit dem Bild vor Augen, dass viele kleine Teams jeweils ihre eigenen Domänen der Software selbstständig und mit Produktcharakter entwickeln und betreiben, kommt man ganz schnell zu der Erkenntnis, dass sich irgendwo doch wieder alles zu einem großen Ganzen zusammenfügen muss, um den Kunden die gleiche umfassende Funktionalität liefern zu können. Fachliche Schnittstellen zwischen den Domänen werden jetzt zu Kommunikationsschnittstellen zwischen vielen kleinen Teams, über die ausgiebig, aber auch geregelt kommuniziert werden muss.

Mit dem API-Management hat man ein Werkzeug in der Hand, das neben den technischen Aspekten der Service-APIs immer auch die fachlichen Aspekte der Schnittstellen beschreiben sollte. Existiert ein gutes Domänenmodell, definiert man in der Regel sehr gute Schnittstellen zwischen Domänen, Subdomänen und den Bounded Contexts. Diese Schnittstellen sollte sehr schnell dokumentiert und mit einem konsistenten API- bzw. API-Lifecycle-Management versehen werden.

API-Management hilft somit auch, Team-APIs zu identifizieren und zu dokumentieren, um organisatorische Maßnahmen ergreifen zu können, damit diese gut miteinander kommunizieren und sich nicht gegenseitig behindern.

Fehlerkultur

Der Aspekt Fehlerkultur sollte nicht verschwiegen werden und zählt zu den kontroversesten Themen in der DevOps-Kultur. Die Erfahrung der letzten Jahre zeigt, dass eingespielte DevOps-Teams zwar nicht weniger Fehler machen als traditionelle Teams, sie diese aber deutlich schneller beheben, sodass Fehler oft gar nicht in der QS oder gar beim Kunden aufschlagen. Dies erfordert jedoch einen offenen Umgang mit Fehlern und die Bereitschaft, Fehler zu akzeptieren, sich zu ihnen zu bekennen und aus ihnen zu lernen.
Hier sind nicht nur die Entwickler gefragt, sondern auch das Management und die Führungskräfte! Die Strategie muss ganz klar lauten: Etablieren Sie eine offene Fehlerkultur, in der man aus Fehlern lernen kann, anstatt dafür bestraft zu werden.

Erkenntnisse

Automatisierung, Continuous-X, Monitoring, Feedback-Zyklen und Metriken sind die Kernaspekte einer DevOps-Kultur. Alle diese Aspekte müssen adressiert werden, wenn man DevOps in der Organisation einführen möchte. Das geht nicht ohne Technik oder eine geeignete Plattform. Folgende Strategien können darüber hinaus helfen, in Ihrer Organisation eine DevOps-Kultur zu etablieren:

- Ein Umfeld schaffen, in dem Verantwortung und Vertrauen zentrale Säulen bilden.
- Eine Lernkultur etablieren und den Community-Gedanken stärken, damit sich Wissen breit verteilt und Zusammenarbeit über Teamgrenzen hinweg attraktiv wird.
- Früh auf DDD setzen. So entsteht eine fachlich strukturierte Softwarelösung mit konsistenten APIs, die mit Teamgrößen von maximal 15 Softwareingenieuren betrieben werden können.

Um den Grad der DevOps-Fähigkeiten zu beurteilen, sind nachfolgende Metriken nützlich:

- Lead Time – durchschnittliche Zeit, um Code und Tests zu entwickeln und auszurollen.
- Deployment Frequency – Häufigkeit, wie oft neue Versionen ausgerollt werden.
- Mean Time to Restore – wie lange dauert es im Durchschnitt, bis ein System nach einem Ausfall wieder verfügbar ist.
- Change Fail Ratio – das Verhältnis zwischen erfolgreichen und fehlerhaften Deployments.

Container-First-Strategien

Im ersten Teil des Artikels ging es um Strategien aus der Sichtweise von Dev-Ops-First. Es lohnt sich aber durchaus, das Thema auch von der Technologieseite aus zu beleuchten, denn häufig wollen oder müssen Unternehmen ihre Betriebsplattformen modernisieren. In Abbildung 3 sind einige Charakteristika von Container- und Cloud-Plattformen dargestellt, auf die wir im Folgenden näher eingehen, um aufzuzeigen, dass diese Plattformen ihre Power ohne DevOps nicht voll entfalten können.

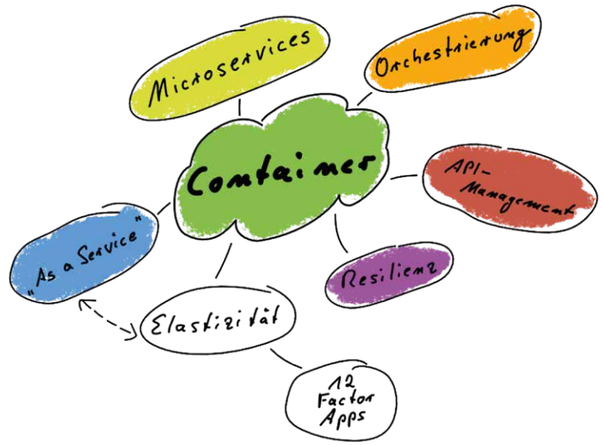


Abbildung 3:

Impulse durch Container-Technologie

Orchestrierung

Für den Start mit einer neuen Plattform sind bestehende, historisch gewachsene Anwendungen oft schlecht geeignet. Der Ansatz Lift and Shift funktioniert nur bis zu einer gewissen Größe der Anwendungen. Es ist also in der Regel keine gute Idee, nur die Plattform auszutauschen und bei der bestehenden Anwendungsarchitektur zu bleiben.

Container-Plattformen nutzen als Betriebsmodell Orchestrierung; verbreitete Beispiele sind Kubernetes und Rancher. Diese Werkzeuge bringen viele Good Practices in Form von Automatisierung, CI/CD, Sidecars und Konfigurationsmanagement mit und unterstützen massiv Self-Services. Ohne DevOps-Arbeitsweise bringen diese Plattformen keine Vorteile! Der traditionelle Betrieb, wie wir ihn aus der traditionellen IT kennen, wäre schlichtweg überfordert, für eine große Anzahl von Entwicklerteams tausende Container auszurollen, zu betreiben und zu überwachen.

Auf der Seite der Entwickler ist dies allerdings mit einer mehr oder weniger steilen Lernkurve verbunden, denn auf diesen Plattformen gilt die Devise: „You build it, you run it“. Das Team muss sich unweigerlich mit Betriebsaspekten beschäftigen, die früher im Betrieb gebündelt waren. Die Einführung einer Container- oder Cloud-Plattform bedingt eine Neustrukturierung der Anwendungslandschaft, die von kleineren Teams eigenverantwortlich weiterentwickelt werden kann. Das führt in der Konsequenz zu neuen Kommunikationsbeziehungen in der Organisation und ist in der einschlägigen Literatur unter dem Begriff Inverse Conway Maneuver zu finden.

„As a Service“ und Microservice-Architekturen

Ein anderer Grund, mit einer Container-Plattform zu starten, kann die Einführung von Microservice-Architekturen sein. Auslöser ist häufig der As-a-Service-Ansatz, mit dem man fachliche Geschäftsprozesse in lose gekoppelte unabhängige Services verpackt, die dann von kleineren Teams in Eigenverantwortung entwickelt und betrieben werden können.

Weitere Gründe für diesen Architekturstil sind Skalierbarkeit und Flexibilität. Die modernen Plattformen punkten mit Elastizität, einem hohen Automatisierungsgrad und fest eingebauten Metriken für die Überwachung, womit sie alle erdenklichen Anforderungen an Performance mit vergleichsweise geringem Aufwand erfüllen. Die Plattformen bringen außerdem eine sogenannte Registry mit, die es erlaubt, Templates zu hinterlegen, mit denen flexible – zu einem gewissen Maß auch standardisierte –, polyglotte Services erstellt werden können. So ist es möglich, die optimalen Ablaufumgebungen und Programmiersprachen für die Microservices zu nutzen, ohne eine große Anwendung komplett neu schreiben zu müssen.

Vergleicht man diese beiden Aspekte mit den DevOps-Paradigmen in Abbildung 2, so korrespondieren die Microservices mit fachlichen Domänen, während der As-a-Service-Aspekt auf Verantwortung für eine Domäne abzielt.

API-Management

Das Thema API-Management hat auch aus Technologiesicht mit Blick auf die vielfältigen Lösungen für API-Gateways viel Aufmerksamkeit erfahren. Es ist in der Tat ein Antrieb, auf eine moderne Plattform zu setzen. Zum Beispiel bei Banken, Versicherungen und Behörden werden Services und Self-Services immer wichtiger.

Das führt dazu, dass die IT dieser Unternehmen sich stark für API-Gateways interessiert und sich viel von diesen Technologien verspricht.

Doch Technologie sollte nicht der einzige Antrieb sein. Man muss feingranulare Geschäftsprozesse modellieren, die Fachlichkeit kapseln und fachliche Schnittstellen bzw. APIs mit Produktcharakter definieren. Nur dann können diese Services unabhängig genutzt und von vielen anderen Services, Benutzeroberflächen und letztendlich Kunden konsumiert werden. In diesem Zusammenhang ist der Schlüsselfaktor der Produktcharakter für die Services hinter den APIs, denn er schlägt die Brücke zu Team-APIs und passt ideal in eine DevOps-Kultur.

Resilienz

Ein letzter Aspekt, der oft als Grund für die Einführung von Container-Plattformen genannt wird, ist Resilienz. Wir wollen hier nicht verschweigen, dass dies ein komplexes Thema ist, welches in der klassischen Softwareentwicklung oft vernachlässigt wurde bzw. eine untergeordnete Rolle spielt.

Ein Monolith läuft, oder er läuft nicht. Microservice-Architekturen bestehen hingegen aus vielen Services, die in ihrer Gesamtheit eine Funktionalität erfüllen. In diesem Szenario kann es durchaus passieren (und es wird passieren), dass einzelne Services ausfallen, mit großer Verzögerung antworten oder eine Zeit lang gar nicht verfügbar sind. Auch die Verteilung der Last auf mehrere Instanzen eines einzelnen Service ist kein triviales Thema, auch wenn die Plattformen hier komfortable Lösungen bieten.

Alle diese Überlegungen laufen darauf hinaus, dass man viele neue Skills aufbauen und diese in allen Teams, die Services entwickeln, verankern muss. DevOps ist auch hier eine gute Antwort, denn Full-stack-Mentalität zielt nicht nur auf Entwicklung im Front- und Backend ab, sondern umfasst auch Betrieb, Architektur, Monitoring und das Wissen über nichtfunktionale Qualitäten wie Resilienz, Kommunikation und Verfügbarkeit.

Erkenntnisse

Digitale Transformation mit der Einführung einer modernen Container- oder Cloud-Plattform einzuleiten ist weit verbreitet. Dies wird allerdings nicht gelingen bzw. keine Vorteile mit sich bringen, wenn man nur die technischen Aspekte betrachtet und die Organisationskultur außen vor lässt.

Folgende Strategien bieten sich an, wenn man mit einer Container-Plattform starten möchte:

- Produkte und Weiterentwicklungen konsequent auf Container-Plattformen ausrichten. Das funktioniert nur, wenn man bereit ist, die Softwarearchitektur auf diese Plattformen auszurichten und eine DevOps-Kultur zu etablieren.
- Servicebasierte Architekturen errichten, wenn die Anwendungslandschaft modernisiert werden soll oder muss. Container-Plattformen bieten mit vielen Good Practices für autonome DevOps-Teams das ideale Fundament.
- APIs als Geschäftsmodell nutzen. Container-Plattformen sind dafür ein Katalysator. API-Gateways sind elementare Bausteine dieser Plattformen, bieten komfortable Skalierungs- und Steuerungsmechanismen und unterstützen autonome Team-APIs.
- Microservice-Architekturen brauchen Resilienz. Diese Plattformen bringen das technische Fundament mit. Für die Organisation bedeutet dies aber eine neue Fehlerkultur, wofür die DevOps-Kultur ebenfalls eine gute Antwort darstellt.

Fazit

Wie heißt es so schön: „Viele Wege führen nach Rom!“ So ist es auch mit DevOps- und Containerstrategien – es gibt nicht den einen oder den besten Weg. Oder wie sagt Kent Beck so gerne: „It depends.“

Was man allerdings mit Sicherheit sagen kann: Man ist schlecht beraten, eine Container- oder Cloud-Plattform einzuführen und bei der Anwendungsentwicklung auf klassische Organisationsstrukturen zu setzen. Dann wird man viel Geld für eine State-Of-The-Art-Plattform ausgeben, ihr Potenzial aber nicht nutzen können. Die Software wird durch eine neue Plattform nicht besser, Qualität und Entwicklungsgeschwindigkeit werden nicht steigen und im schlimmsten Fall werden die Erwartungen der Kunden nicht erfüllt, sodass am Ende ein Scherbenhaufen übrigbleibt und die Entwickler das Weite suchen.

Wägt man alle Argumente gründlich ab, kommt man zu der Erkenntnis, dass es sich lohnt, Container und DevOps gleichzeitig zu verfolgen. Es ergibt keinen Sinn, mehrere Jahre lang eine DevOps-Kultur aufzubauen und sich mit „alten“ Plattformen und Technologien herumzuquälen. Die modernen Plattformen sind die natürlichen Habitate für DevOps-Teams und -Ingenieure, und der Arbeitsmarkt wird über kurz oder lang nur noch diesen Entwicklertyp bieten.

Es stellt sich also nicht die Frage: *„DevOps-First oder Container-First?“*, sondern vielmehr: *„Wie gelingt ein erfolgreicher Einstieg mit DevOps und Containern?“*.

ARS
Passion for your digital future

Über ARS

Wir sind ARS, und wir leben für die digitale Zukunft unserer Kunden – das ist mehr als nur ein Claim für uns, es ist unser Versprechen. In einer Welt, die zunehmend digitalisiert wird, sind wir vertrauenswürdiger Partner, der Unternehmen durch den gesamten Lebenszyklus ihrer Softwareprojekte begleitet. Von der strategischen Planung über die effiziente Umsetzung bis hin zur Transformation der Arbeitsweisen, Prozesse und Infrastruktur – wir begleiten unsere Kunden mit maßgeschneiderten Lösungen und echter Begeisterung für Technologie.

Unser vielseitiges Angebot umfasst Beratung in der Architektur, Anwendungsentwicklung, Qualitätssicherung, DevOps-Betriebsmodelle, API-Management bis hin zu Cloud-Technologien und künstlicher Intelligenz. Doch wir gehen weiter als nur die Bereitstellung technischer Expertise. Wir stellen sicher, dass Agilität in den Unternehmen nicht nur ein Schlagwort bleibt, sondern zur gelebten Realität wird – und das von der Softwareentwicklung bis zum operativen Betrieb.

Unser Herz schlägt für die DevOps-Kultur, die darauf abzielt, technische und organisatorische Hindernisse aus dem Weg zu räumen. Wir machen relevante Informationen verfügbar, die zur Wertschöpfung beitragen, und stärken die kontinuierliche Weiterentwicklung der Teams. Unsere Arbeit ist geprägt von einem interdisziplinären Ansatz, bei dem wir die Erfahrungen und Perspektiven aller Beteiligten berücksichtigen. In gemeinsamen, iterativen Schritten entwickeln wir eine DevOps-Kultur, die perfekt zum individuellen Unternehmensumfeld passt.

So unterstützen wir unsere Kunden nicht nur dabei, sich den Herausforderungen der digitalen Transformation zu stellen, sondern vor allem darin, sie aktiv zu gestalten. Wir sind überzeugt, dass unsere Leidenschaft für Technologie sie voranbringen wird.
Lassen Sie uns gemeinsam eine Zukunft gestalten, die so dynamisch und innovativ ist wie die digitale Welt, in der wir leben.

Mit Leidenschaft, mit Expertise – mit ARS.

ARS Computer und Consulting GmbH
Garmischer Str. 7, 80339 München ▪ T +49 89 324 68-0 ▪ modernize@ars.de ▪ www.ars.de


ART-DEV-20210701-V2
DEVIDEN
Café au Lait
BOSS
BOSS