BETREUTES WOHNEN FÜR ALTANWENDUNGEN

Effiziente Migration und Betrieb von Alt-Verfahren in IT-Dienstleistungszentren mittels Containerisierung

| von TIM TIM POMMERENING

Altanwendungen von einem Drittthersteller – vielleicht vor Jahren von Fachbereichen im eigenen Rechenzentrum in Betrieb genommen – sollen nun in das Rechenzentrum eines behördenübergreifenden IT-Dienstleistungszentrums (IT-DLZ) übernommen werden: Dieses Szenario dürfte vielen Lesern und Leserinnen bekannt vorkommen. Wartungsverträge und Herstellersupport sind mittlerweile ausgelaufen, das IT-DLZ setzt längst neuere Versionen seiner Laufzeitumgebungen und Middleware voraus und eine Hebung der Anwendungen auf die neueren Versionen ist aus verschiedenen Gründen nicht zeitnah möglich. Darüber hinaus strebt das IT-DLZ eine Standardisierung der in ihm betriebenen Anwendungen an. Eine Überführung in einen geregelten IT-Betrieb scheint daher unmöglich.

Dieses Szenario kommt so oder in Abwandlungen immer wieder vor. In diesem Artikel beschreiben wir ein Vorgehen, wie solch eine Überführung von Altanwendungen Schritt für Schritt priorisiert stattfinden kann, um betroffene Anwendungen mithilfe der Container-Technologie in einen geregelten IT-Betrieb auf standardisierten Plattformen weiterbetreiben zu können. Der Container dient dabei als Adapter zwischen der Standardplattform des Betreibers und der Altanwendung. Der Vorteil dabei ist, dass der IT-Betrieb keine Sonderbehandlungen für seine standardisierten Betriebsumgebungen fahren muss, während aus Sicht der Anwendungen aber die ursprünglichen Systemanforderungen weiterhin erfüllt bleiben.

RISIKOBASIERTES VORGEHEN

Situationen wie oben beschrieben ergeben sich häufig durch organisatorische Änderungen wie die Zusammenlegung von Abteilungen oder Umstrukturierung von Zuständigkeiten. Abbildung 1 zeigt ein schrittweises Vorgehen zum Überführen von Altanwendungen in ein Standard-RZ, nachfolgend zuerst auf hoher Ebene skizziert, um dann in den weiteren Abschnitten dieses Artikels den Fokus auf die Analyse und Überführung der Anwendung in eine Containerlandschaft zu legen.

Bei Umorganisationen muss häufig zunächst herausgefunden werden, welche Anwendungen betroffen sind. Dies ist die Aufgabe des Architekturmanagements (vgl. Abbildung 1, rechts oben). Zuerst werden die betroffenen Anwendungen in einem EAM-Tool oder im einfachsten Fall auch in Excel-Tabellen erfasst.

Mit der Liste an Anwendungen lassen sich durch Auswertung zuvor festgelegter Kriterien (siehe Infobox 1) Risikobetrachtungen anstellen und somit Hand-

lungsbedarfe feststellen, die in einer priorisierten Liste münden. Darüber kann gesteuert werden, welche Altanwendungen in welcher Reihenfolge in die standardisierte Betriebslandschaft überführt werden sollen.

Eine über die Kriterien hinausgehende, tiefere Analyse (vgl. Abbildung 1, rechts unten) jeder Anwendung findet statt, nachdem die Anwendung aus der priorisierten Liste für eine Überführung in die standardisierte IT-Landschaft ausgewählt wurde. Je nach Dringlichkeit und Anzahl hochpriorisierter Anwendungen können Überführungen nacheinander oder auch parallel von mehreren Teams durchgeführt werden. Eine Überführung resultiert üblicherweise in verschiedenen Ergebnisartefakten (vgl. Abbildung 1, links unten):

- Die Festlegung zukünftiger Verantwortlichkeiten (Rollen)
- Die für den IT-Betrieb erforderliche Dokumentation
- Die in der Standardbetriebsumgebung laufende Anwendung

Die Ziellandschaft für die überführte Anwendung ist die Standardbetriebsumgebung. Falls es nicht oder nur unter hohem Aufwand möglich wäre, die Anwendung in der Ziellandschaft zu betreiben, wird sie in eine Transitionsarchitektur[1] überführt, die in einer Zwischenlandschaft lauffähig ist. Sie soll nach außen hin den Anforderungen der Ziellandschaft genügen und nach innen den Anforderungen der

BEISPIELKRITERIEN ZUR RISIKOBETRACHTUNG VON ANWENDUNGEN

- **Versionen von Betriebssystem, Middleware und Anwendung**
  Häufig wurde Software lange nicht aktualisiert oder entspricht nicht dem Standard im Rechenzentrum. Es bestehen Sicherheitsrisiken.
- **Zustand von Supportverträgen**
  Drohen diese auszulaufen oder sind vielleicht schon ausgelaufen, steigen Risiken bei der Verfügbarkeit.
- **Dokumentation und Zuständigkeiten**
  War bisher ein Experte aus einer Fachabteilung für die Anwendung zuständig, existiert häufig keine für den IT-Betrieb standardisierte Dokumentation. Auch das ist ein Verfügbarkeitsrisiko.
- **Häufige Ausfälle**
  Ist die Anwendung heute bereits sehr wartungsintensiv, kann auch das ein Kriterium für eine Priorisierung sein.
- **Updateaufwand**
  Ist es einfach, die Anwendung zu aktualisieren, kann das als Sofortmaßnahme durchgeführt werden und damit Risiken und die Priorisierung einer notwendigen Überführung verringern.

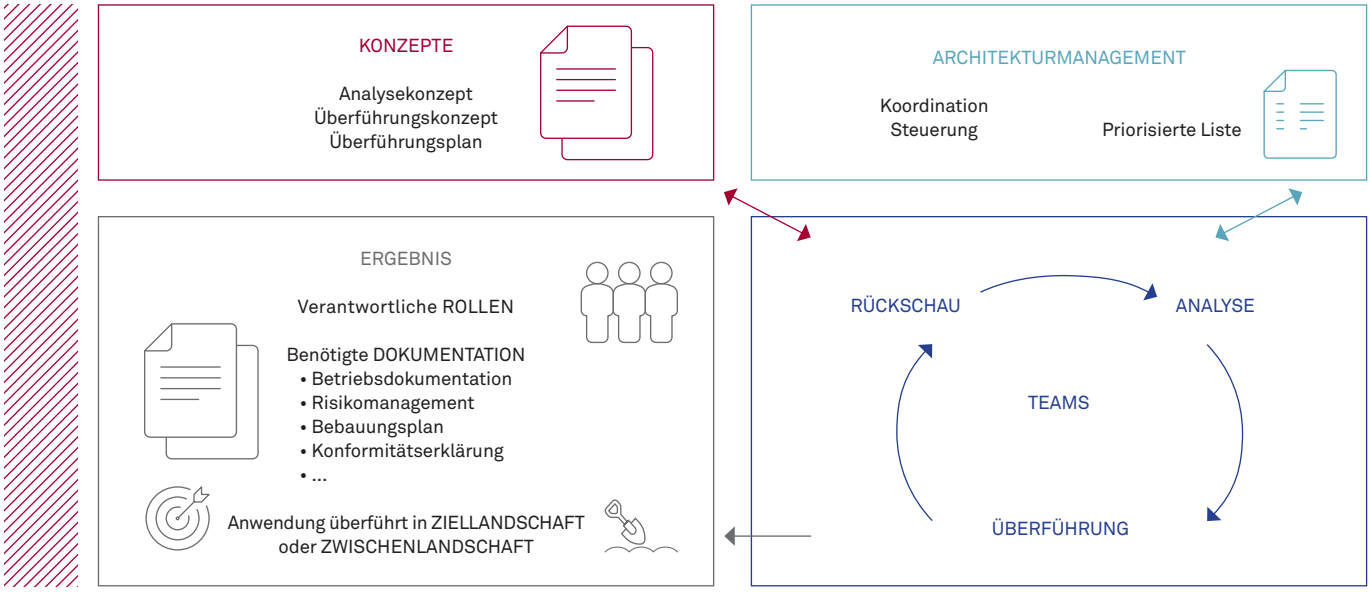

Abbildung 1: Risikobasierte, schrittweise Überführung von Altanwendungen in eine standardisierte Betriebsumgebung

überführten Anwendung gerecht bleiben. Hier kommt die Containerisierung ins Spiel. Nach jeder überführten Anwendung findet eine Rückschau statt. Die während der Überführung abgeleiteten Erkenntnisse aus der Arbeit mit dieser Anwendung fließen in die Konzepte (vgl. Abbildung 1, links oben) ein, um diese für zukünftige Überführungen schrittweise weiter zu verbessern. Das Team kann nun mit der Überführung der nächsten Anwendung aus der Liste starten.

TRANSITIONSARCHITEKTUR DURCH CONTAINERISIERUNG

Nachdem das Vorgehen zur Überführung von Anwendungen in eine Standardbetriebsumgebung erläutert wurde, geht es nun an einem konkreten Beispiel darum, wie die Überführung in die zuvor beschriebene Zwischenlandschaft mittels Containerisierung durchgeführt werden kann.

Im Beispiel bietet der IT-Betrieb als Ausprägung der Standardbetriebsumgebung eine Docker-Landschaft[2] an. Eine Webanwendung mit Namen Classic-App auf Basis von PHP 5, die bisher auf einem Server von einer Fachabteilung betrieben wurde, soll nun an den IT-Betrieb übergeben werden. War es früher möglich, die PHP-5-Plattform ohne größere Eingriffe in den Quellcode von Classic-App zu aktualisieren, wurde das mit PHP 5.6 bereits problematisch und nach Supportende dieser Version im Dezember 2018 ist ein Umstieg auf PHP 7 nun ohne größere Codeänderung nicht mehr möglich. Da dafür kein Herstellersupport mehr besteht, bietet das Rechenzentrum keine standardisierte Plattform für PHP 5 mehr an. Weil Classic-App zur Risikominimierung aber schnellstens ins Rechenzentrum überführt werden soll, fiel die Entscheidung auf die Überführung in eine Zwischenlandschaft.

PHASEN DER ÜBERFÜHRUNG

Um das Ziel zu erreichen, die Anwendung in einer Standardbetriebsumgebung zu betreiben, werden die Schritte Analyse und Überführung in fünf Phasen unterteilt und anschließend anhand des Classic-App-Beispiels durchgespielt:

1. **Anwendung analysieren** mit dem Ziel, die architektonisch relevanten Besonderheiten zu identifizieren, aus denen Anforderungen an die Containerisierung entstehen
2. **Überführungsstrategie entwerfen** mit dem Ziel, anhand der Anforderungen einen Entwurf zu erstellen, wie die Überführung stattfinden soll
3. **Überführung umsetzen** mit dem Ziel, die Anwendung zu containerisieren
4. **Containerisierung testen** mit dem Ziel, die Qualität der Umsetzung festzustellen
5. **Containerisierung in Betrieb nehmen** mit dem Ziel, die Altanwendung in einer standardisierten Betriebsumgebung zu betreiben und die vorherige nicht mehr dem Standard entsprechende Umgebung abzuschalten

Phase 1: Anwendung analysieren

In dieser ersten Phase geht es darum, die Anforderungen der Anwendung an ihre Betriebsumgebung zu identifizieren und zu dokumentieren, um damit in der nächsten Phase einen Entwurf für die Containerisierung zu erstellen, der dann auch Teil der Dokumentation der Systemarchitektur wird. Die Analyse durchläuft mehrere Schritte:

- Technologien und Versionen identifizieren: Um entscheiden zu können, wie die Anwendung containerisiert wird, müssen zuerst die eingesetzten Technologien (Datenbanken, Laufzeitumgebung, Middleware ...) identifiziert und die benötigte Version dokumentiert werden.
- Bewegungsdaten im Dateisystem identifizieren: Zum Konzept von Containern gehört es, dass sie sich bei jedem Neustart wieder in einem sauberen Initialzustand befinden, also keine Systemreste zurücklassen. Während der letzten Laufzeit im Dateisystem angelegte oder geänderte Dateien – die Bewegungsdaten – sind verloren, sofern sie nicht in sogenannten data-volumes[3] oder in speziell eingehängten Ordnern des darunterliegenden Servers abgelegt wurden. Daher müssen Bewegungsdaten, die über den Neustart eines Containers persistiert werden sollen, aus dessen Ordnerstrukturen herausgelöst werden.[4]
- Ports/Schnittstellen identifizieren: Für die Umsetzung der Containerisierung ist es wichtig zu wissen, welche Ports innerhalb des Containernetzwerks verwendet werden und welche Ports nach außen freigegeben werden sollen.
- Mögliche Versionsupdates identifizieren: Auch, wenn es Ziel der Containerisierung ist nicht standardisierte alte Versionen weiterhin betreiben zu können, ist es doch sinnvoll zu prüfen, welche Standardkomponenten auf neuere Versionen gehoben werden können – schon aus dem praktischen Grund, dass zu alte Versionen im Paketmanagement der zukünftig verwendeten Umgebung fehlen und daher bei der Installation Mehraufwand bedeuten könnten.

Die Analyse von Classic-App liefert folgendes Ergebnis: Classic-App war ursprünglich eine PHP-5.3-Anwendung, die im Rahmen früherer Updateaktivitäten in der Vergangenheit auf PHP 5.6 gehoben wurde. Eine Hebung auf PHP 7 ist ohne größere Umbauten nicht mehr möglich, da in dieser PHP-Version viele verwendete alte Programmierschnittstellen verändert oder entfernt wurden (Details siehe Kasten „Analyseergebnisse Classic-App“).

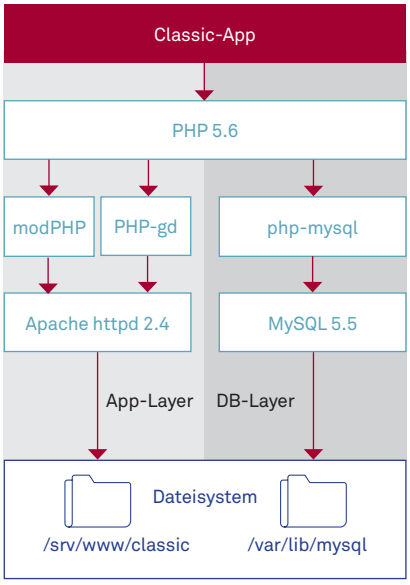

Abbildung 2: Analyseergebnis der Altanwendung Classic-App und ihrer Abhängigkeiten auf bisherigem Server

PHASE 1: ANALYSEERGEBNISSE CLASSIC-APP

Die Anwendung verwendet folgende Laufzeitumgebung:

- Einen Apache-Webserver 2.4 mit modPHP: Der Apache-Webserver bietet eine Plug-in-Architektur, für die ein PHP-Plug-in existiert. Bei anderen Webservern läuft die PHP-Laufzeitumgebung hingegen oft als eigener Dienst.
- Eine MySQL-5.5-Datenbank
- Zwei PHP-Erweiterungsbibliotheken: php-mysql zur Datenbankanbindung sowie php-gd zur Darstellung von Grafiken

Neben den Informationen in der Datenbank legt Classic-App hochgeladene Grafiken unterhalb ihres Installationsordners /srv/www/classic/ in einem Unterordner images/ sowie einige benutzerspezifische Daten im Unterordner users/ ab. Im Unterordner temp/ liegen zusätzlich temporäre Daten, die aber zu jeder Zeit gelöscht werden können, weil sie die Anwendung bei Bedarf neu anlegt. Die übrigen Unterverzeichnisse enthalten PHP-Dateien, die Teil der Anwendungsinstallation sind. Konfigurierbar sind die Datenverzeichnisse leider nicht.

Classic-App ist also eine typische Webanwendung und verwendet Port 80/TCP, der von außerhalb des internen Netzes erreichbar sein muss, um auf die Anwendung zugreifen zu können. Die MySQL-Datenbank verwendet Port 3306/TCP, der nur im internen Netz, aber nicht von außen erreichbar sein soll.

Phase 2: Überführungsstrategie entwerfen

Mit den Informationen der Analyse geht es nun in dieser Phase darum, die Zielarchitektur zu definieren. Die in Phase 1 identifizierten Technologien und deren Versionen sollen sich direkt in sogenannten „Images“[5] niederschlagen. Abbildung 3 zeigt auf der rechten Seite, wie Docker-Images hierarchisch aufgebaut sind und immer weiter spezialisiert werden.

Das *Docker-Image os (Operating System)* ist im Beispiel der Ausgangspunkt für alle weiteren Images. Es handelt sich um eine spezielle Linux-Variante mit Anpassungen, bereitgestellt vom IT-Betrieb. Darauf baut das Image *nginx*[6] auf und auf das wiederum das Image php56. Auf diese Weise lassen sich leicht etwa verschiedene PHP-Versionen als Images erstellen, die alle auf nginx basieren. Die bisher genannten Images stellen strukturierte hierarchisch wiederverwendbare Komponenten dar. Es existieren später davon keine Laufzeit-Varianten (Container).

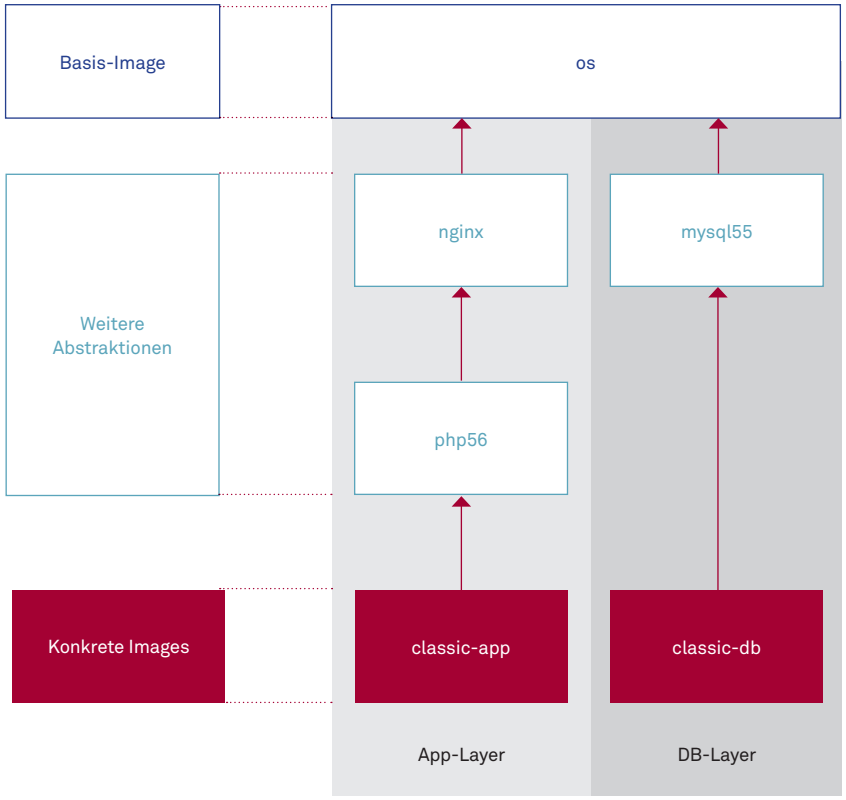

Abbildung 3: Die Docker-Images classic-app und classic-db leiten von abstrakten Docker-Images ab und erben damit deren Inhalte.

PHASE 2: ÜBERFÜHRUNGSSTRATEGIE CLASSIC-APP-STRUKTUR

Das Beispiel geht davon aus, dass in der Ziellandschaft der leichtgewichtigere Webserver nginx anstelle von apache http eingesetzt werden soll. Das nginx-Image stellt daher die Basis-Konfiguration für den Webserver bereit, während darauf aufbauend das php56-Image diese mit der Laufzeitumgebung für PHP anreichert. Da für nginx kein PHP-Plug-in existiert, wie bei Apache (mod_php), stellt das php56-Image mit phpfastcgi diese Laufzeitumgebung für PHP bereit und konfiguriert sie in nginx hinein.

Das Beispiel geht deshalb diesen Weg, um zu zeigen, dass in resultierenden Containern nun zwei Prozesse laufen: je einer für den Webserver und die Laufzeitumgebung für PHP, die miteinander kommunizieren. Das widerspricht erst einmal der Philosophie von Docker, einen Prozess pro Container zu kapseln, kann aber bei einer so engen Kopplung sinnvoll sein, weil es die Handlichkeit erleichtert.

Das unterste Image im Zweig App-Layer in Abbildung 3, classic-ap, installiert und konfiguriert die Programmdateien der Classic-App, setzt auf das php56-Image auf und kann die dort enthaltene PHP-Konfiguration je nach Systemanforderungen verändern. Grundsätzlich wäre es möglich, die Datenbank, also mysql, im gleichen Container laufen zu lassen. Auch wäre es möglich, dass ein allgemeinerer Datenbankcontainer ein gemeinsames DBMS (Datenbankmanagementsystem) für eine Reihe von Anwendungen bietet. Im Beispiel der Classic-App folgt die Entwurfsentscheidung hier der Philosophie von Docker und so enthält der DB-Layer ein vom mysql55-Basis-Image abgeleitetes classic-db-Image mit nur dem für Classic-App dedizierten mysql-DBMS, das, sollte es stoppen, von Docker automatisch neu gestartet werden kann.

Neben der statischen Struktur der zu migrierenden Anwendung müssen für die in Phase 1 identifizierten Bewegungsdaten auch Auslagerungsstrategien festgelegt werden. Docker kann Bewegungsdaten auf zwei verschiedene Arten auslagern, damit sie beim Neustart eines Containers erhalten bleiben: zum einen in data-volumes und zum anderen in Verzeichnisse auf dem Server, auf dem Docker läuft. Auch data-volumes sind am Ende nur Verzeichnisse auf diesem Server. Der Unterschied ist, dass Docker sie verwaltet. Beide Varianten können auch gemischt verwendet werden.

Welche Variante bevorzugt wird, muss eine übergeordnete Strategie festlegen. Eine sinnvolle Strategie könnte dabei so aussehen, dass für alle Nutzdaten, für die ein Back-up benötigt wird, data-volumes verwendet werden, während Dateien, die vom IT-Betrieb verwaltet werden, wie etwa Logfiles, in bestimmte vom IT-Betrieb vorgegebene Verzeichnisse auf dem Docker-Server abgelegt werden, wo sie etwa dem Monitoring bereitgestellt oder rolliert werden können. Im fertigen Architekturentwurf zeigt Abbildung 4 die Abhängigkeiten zwischen Containern, Volumes und dem Dateisystem mit Pfeilen.

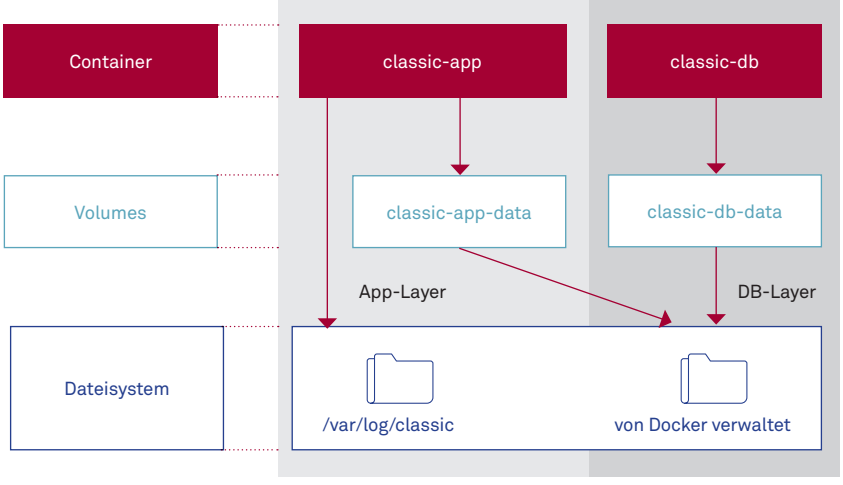

Abbildung 4: Der Architekturentwurf mit Docker-Images für Programm- und Docker-Volumes für Bewegungsdaten

Phase 3: Überführung umsetzen

Die Umsetzung des in Phase 2 erzeugten Architekturentwurfs erfolgt dann in mehreren „Dockerfiles“ und in einem „docker-compose file“. Ein Dockerfile ist die in einem einfachen Klartext-Beschreibungsformat beschriebene Bauanleitung für ein Image, also die statische Sicht. Ein docker-compose file definiert im Gegensatz zum statischen Dockerfile das Verhalten und Zusammenspiel mehrerer Container zur Laufzeit, konfiguriert damit die dynamische Sicht.

Die in Phase 2 getroffenen Entscheidungen zur Hierarchie und Aufteilung der Classic-App in ihre Komponenten und damit Images schlägt sich also direkt in den Bauplänen der Images nieder. Sie beschreiben die automatisierte Installation jeglicher Software und Konfiguration, die später während des laufenden Containers benötigt wird.

PHASE 3: ÜBERFÜHRUNGSSTRATEGIE CLASSIC-APP-BEWEGUNGSDATEN

Um innerhalb der Containersicht die Bewegungsdaten von statischen Daten zu trennen, sind zwei Wege möglich. Falls die Anwendung es erlaubt, die Pfade zu den Bewegungsdaten zu konfigurieren, können diese direkt auf die oben beschriebenen data-volumes oder Verzeichnisse konfiguriert werden (im Beispiel /data/images und /data/users). Sieht die Anwendung hingegen keine Konfigurationsmöglichkeit für Verzeichnisse vor, helfen symbolische Verknüpfungen im Dateisystem, um dynamische Teile aus den statischen herauszulösen.

Für die Classic-App ist /data der Ort, an dem das data-volume classic-app-data in das Dateisystem eingehängt wird. Somit liegen die beiden Ordner mit den Bewegungsdaten (images und users) an einem persistenten Ort.

Log-Dateien sollen im Beispiel nicht im Volume abgelegt werden, sondern direkt auf dem Server. Dazu hängt der classic-app-Container beim Start das Verzeichnis /var/log/classic vom Docker-Server ein und bildet es im Container ab.

Prinzipiell könnte natürlich auch die gesamte Anwendung im data-volume abgelegt werden. Das geht jedoch mit dem Verlust der Möglichkeit einher, eine Anwendung durch Restart des Docker-Images auf ihren Initialzustand zurückzusetzen, sie in den Image-Versionen zu historisieren oder sie mehrfach zu instanziieren. Daher ist von dieser Variante abzuraten.

Die in Phase 2 hingegen getroffenen Entscheidungen zur Ablage der Bewegungsdaten, zur Analyse der Ports und zu Abhängigkeiten zwischen laufenden Containern werden direkt in der Laufzeitkonfiguration umgesetzt. Sie definiert etwa, dass classic-db laufen muss, damit classic-app starten kann, ob ein Container automatisch neu gestartet werden soll, falls er sich beendet, oder welches virtuelle Netzwerk sich die Anwendungsgruppen zur Laufzeit teilen.

Phase 4 und 5: Containerisierung testen und in Betrieb nehmen

Nachdem in Phase 3 die Docker-Images und Volumes konfiguriert und verknüpft wurden, muss das Ergebnis nun noch getestet und in Betrieb genommen werden. Die in der vorherigen Phase entstandenen Installationsartefakte erlauben es, die Anwendung praktisch automatisiert auf einem Docker-Server zu installieren und zu starten.

Abbildung 5 zeigt eine aus zwei Testinstanzen bestehende Laufzeitumgebung der Classic-App. Die verschiedenen Instanzen der gleichen Anwendung können für verschiedene Testszenarien verwendet und zum Beispiel mit unterschiedlichen Daten bestückt werden. Im Rahmen der Tests und Inbetriebnahme wird auch die festgelegte Betriebsdokumentation erstellt oder bestehende Dokumentation in die standardisierte Form gebracht. Getestete Images können im Rahmen der Tests versioniert und für den Echtbetrieb freigegeben werden.

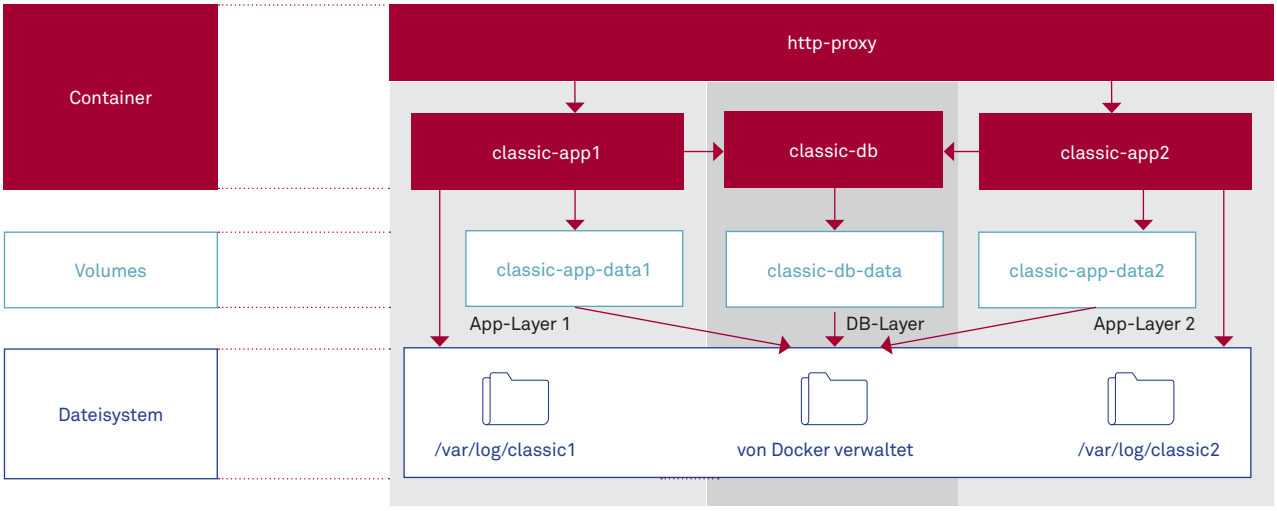

Abbildung 5: Zwei Testinstanzen von Classic-App mit je einem eigenen Data-Volume und geteilter DB-Instanz

PHASE 4 UND 5: CLASSIC-APP: CONTAINERISIERUNG TESTEN UND IN BETRIEB NEHMEN

Zu Testzwecken könnten auf einem Docker-Host mehrere Instanzen der Classic-App laufen, um verschiedene Testszenarien durchführen zu können. Dabei ist zu beachten, dass das classic-app-Image zwar beliebig oft als neuer Container instanziiert werden kann, für jede unterschiedliche Anwendungsinstanz aber auch eigene data-volumes erzeugt und konfiguriert werden müssen, da die Instanzen sich sonst die Bewegungsdaten teilen.

Der Webport kann nur von einem Webserver belegt werden. Sollen mehrere Container mit Webdiensten auf demselben Server laufen und ohne Angabe eines extra Ports über den Webbrowser erreichbar sein, ist ein http Reverse Proxy vorzuschalten.

Abbildung 5 schaltet deshalb den http-proxy, ebenso einen Docker-Container, vor die Webanwendungen classic-app1 und classic-app2. Jede der App-Instanzen nutzt für ihre Bewegungsdaten ihr eigenes Log-Verzeichnis und data-volume.

Für die Datenbank bestände die Möglichkeit, auch das classic-db Image mehrfach zu instanziieren und mit verschiedenen Volumes zu arbeiten. Die Abbildung zeigt als Alternative, dass classic-db nur einmal instanziiert wird und darauf verschiedene Datenbankschemata betrieben werden, je eine eigene pro classic-app-Instanz.

WAS WURDE ERREICHT?

Eine Altanwendung konnte mittels Containerisierung von einer kritischen, schwer zu betreibenden Hardware in eine standardisierte IT-Landschaft umgezogen werden.

In diesem Rahmen konnten Anforderungen wie Trennung von Anwendungs- und Bewegungsdaten umgesetzt werden, um etwa standardisiert Back-ups zu erzeugen und zukünftige Updates der Laufzeitumgebung unabhängig von den Daten in Betrieb nehmen zu können. Die Containerisierung spielt hier die Rolle eines „Adapters“ zwischen den Vorgaben der Standardplattform des IT-Betriebs und den Systemanforderungen der Altanwendung an ihre veraltete Betriebsumgebung. Sie kann so als Zwischenplattform dienen, bis die Anwendung abgelöst oder auf eine neuere Version, die auf einer Standardplattform lauffähig ist, aktualisiert werden kann.

WAS MUSS WEITERHIN BEDACHT WERDEN?

Dieser „Adapter“ bewirkt, dass sich der IT-Betrieb nicht mit den Eigenheiten der nicht standardisierten Umgebung befassen muss. Damit sind diese Eigenheiten nicht verschwunden, denn durch die Containerisierung wurde ein Großteil der Betriebssicht zur Anwendung verlagert. Die Komplexität bleibt, nur ist sie nun woanders. Und die adaptierte Umgebung ist noch immer veraltet.

Die Einführung der Abstraktion durch Containerisierung führt auch zu neuer Komplexität, was die Fehlersuche erschweren kann. Containerisierung zu beherrschen, bedeutet auch (neues) Know-how im IT-Betrieb. Daher ist ein wichtiger Teil der Überführung die Definition von Verantwortlichkeiten für alle Aspekte sowie standardisierte Dokumentation.

Wird Containerisierung – aus der Welt der Microservices kommend – zu ihrem eigentlichen Zweck, wie etwa dem automatischen Starten und Stoppen vieler gleicher Instanzen zur Lastverteilung, eingesetzt, müssen die Anwendungen exakt dafür gebaut worden sein. Es genügt nicht, einen Container um eine Anwendung zu packen, denn damit ist sie noch lange nicht beliebig oft instanzierbar. Das hier durchgespielte Beispiel zeigt aber auch, dass Containerisierung auf andere Weise eingesetzt werden kann: nämlich als leichtgewichtiger Adapter zwischen einer Betriebsumgebung und einer Anwendung, die ohne diesen Adapter nicht auf dieser Betriebsumgebung hätte betrieben werden können. ●

1 Im Sinne des EAM eine Zwischenarchitektur als Schritt auf dem Weg zur Zielarchitektur.
2 https://www.docker.com/ (abgerufen am 17.02.2020).
3 Ein data-volume bezeichnet bei Docker einen Ordner im Dateisystem, der zwischen Container-Restarts persistent bleibt. Es kann auch in mehreren Containern gleichzeitig eingebunden werden.
4 Ein Sonderfall von Bewegungsdaten sind Datenablagen auf Netzlaufwerken, denen unabhängig von der Containerisierung während der Analyse deshalb besondere Beachtung geschenkt werden muss, weil sich hier zusätzliche betriebliche Abhängigkeiten zu anderen Systemen verstecken (z. B. wichtig beim Finden von Wartungsfenstern).
5 Ein *Image* stellt im Kontext der Containerisierung den mit Software und Konfigurationen bestückten, aber nicht gestarteten Container dar, vergleichbar etwa mit einem Programm, das noch nicht gestartet wurde.
6 *nginx* ist ein Webserver – eine leichtgewichtigere Alternative zum bekannteren Apache-Webserver.