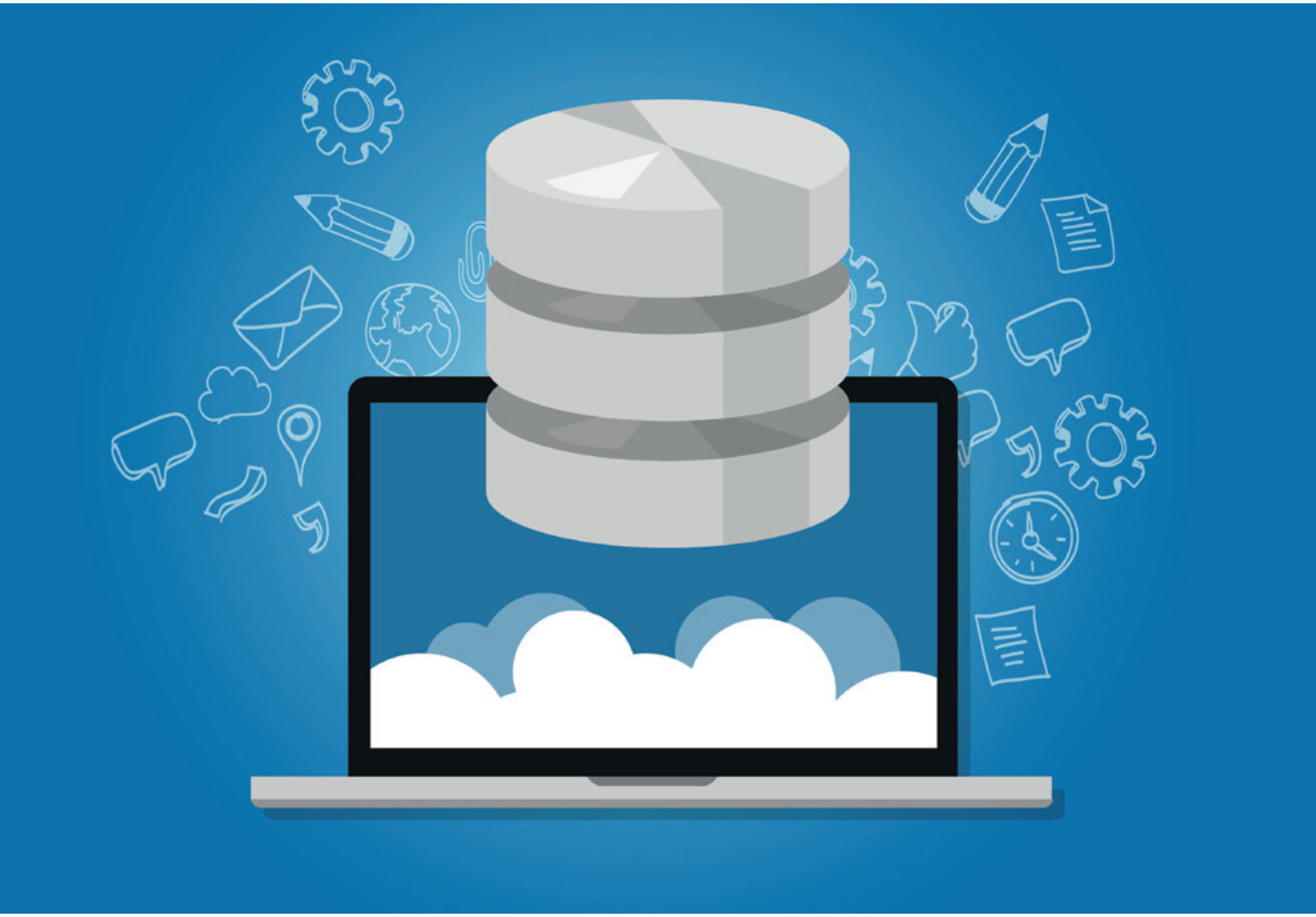

Oracle, PostgreSQL, Docker und Kubernetes bei der Mobiliar

Hans Eichenberger, Mobiliar Versicherungen, und Daniel Westermann, dbi-services

„Agile Development", „Docker", „Cloud", „DbaaS" – diese und ähnliche Begriffe werden auch bei der Mobiliar Versicherung seit ein paar Jahren immer häufiger genannt. Kann Oracle als strategisches RDBMS-Produkt dies alles abdecken oder gibt es Gründe, auch Open-Source-Datenbank-Systeme genauer anzuschauen?

In der Mobiliar werden bereits diverse Applikationen als Microservices in einem Docker-/Kubernetes-Umfeld entwickelt und betrieben. Vor zwei Jahren hat die Mobiliar-IT eine REST-Schnittstelle für die automatisierte Bereitstellung von Pluggable Databases (PDB) gebaut. Diese erlaubt es dem Unternehmen, während des Deployments einer Applikation bei Bedarf in Kubernetes auch die Datenbank vollautomatisch bereitzustellen. Diese Lösung funktioniert inhouse zuverlässig und erstellt beziehungsweise löscht die PDBs ohne manuelle Eingriffe durch den Datenbank-Administrator.

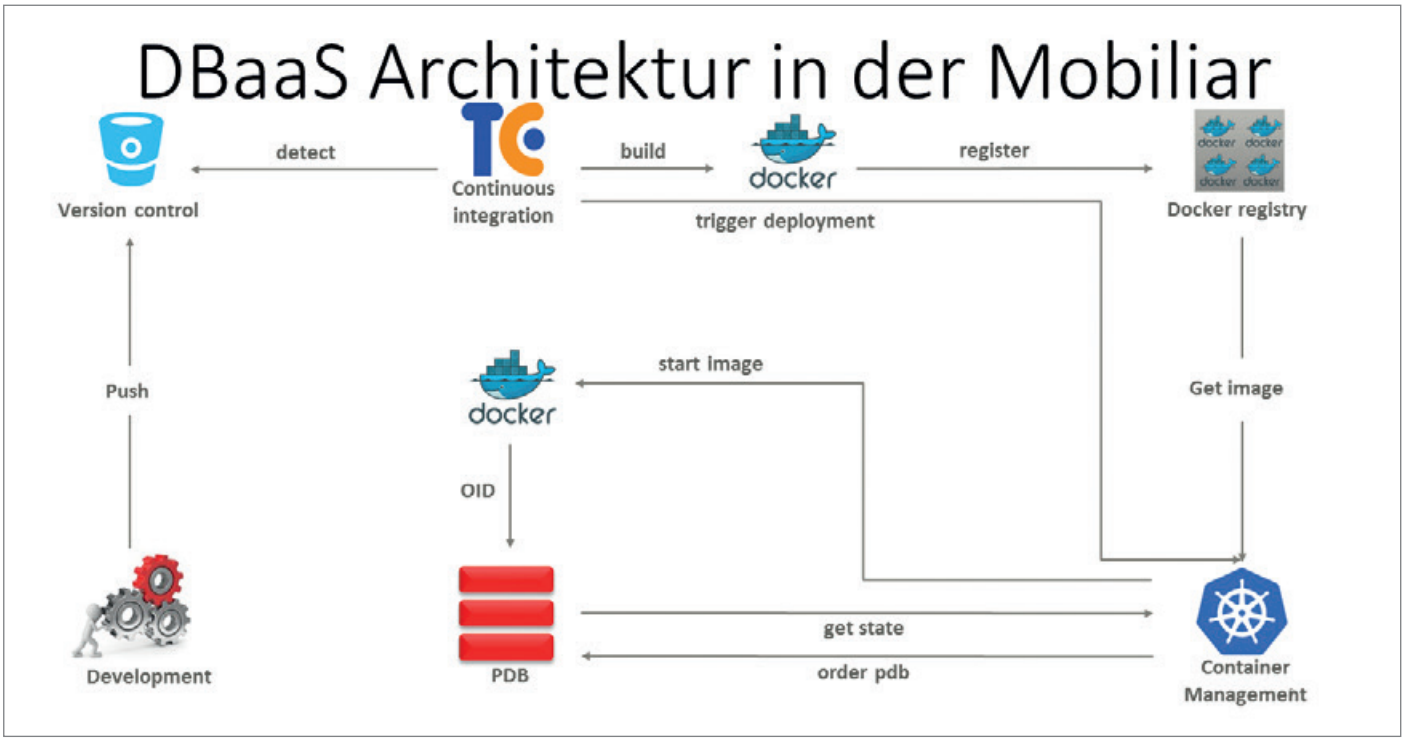

Abbildung 1: Schematische Darstellung der DBaaS-Architektur in der Mobiliar

Auch in der Mobiliar wird die Anforderung immer wichtiger, dass etwa für Test-Installationen sowohl Inhouse- als auch Cloud-Infrastrukturen einsetzbar sein sollen. Die Bedürfnisse der Entwicklung sollen vollumfänglich abgedeckt werden können. Ein zentraler Punkt dabei sind neben der Geschwindigkeit vor allem auch kalkulierbare Kosten. Mit der aktuellen PDB-Order-Schnittstelle erzwingt dieser Punkt, dass beispielsweise für Tests auf den Entwickler-Notebooks nicht eine EE-Installation genutzt, sondern stattdessen die lange nicht mehr aktualisierte XE-Version eingesetzt werden muss.

Technisch gesehen verwenden Microservices meist nur die Grund-Features eines Datenbank-Systems. Damit sind die Persistenz und der schnelle Zugriff auf die Daten sichergestellt. Das Deployment in Kubernetes lässt sich vereinfachen, wenn alle notwendigen Komponenten direkt von Kubernetes gemanagt und keine Schnittstellen zu externen Ressourcen gebaut werden müssen, beispielsweise zum Erstellen von PDBs. Darum musste man eine Lösung mit der Datenbank im Docker-Container suchen. Die folgenden Abschnitte zeigen den Weg von der aktuellen Lösung bis zur Entwicklung der Architektur der PostgreSQL-Docker-Implementierung, wie sie aktuell bei der Mobiliar im Einsatz ist.

Automatisierte PDB-Bereitstellung via REST-Schnittstelle

Damit das agile Entwickeln nicht beim manuellen (sprich langsamen) konventionellen Deployment scheitert, hat die Mobiliar vor etwa zweieinhalb Jahren eine Lösung gesucht, um Datenbanken möglichst automatisiert und schnell bereitstellen zu können. Es stellte sich relativ rasch heraus, dass die Multitenant-Option, die seit Oracle 12.1 verfügbar ist, die Anforderungen an die Datenbanken im agilen Umfeld am besten abdecken kann.

Mit der Modularisierung der Applikation, also dem Schneiden der bestehenden Monolithen in mehrere Microservices, wurde seitens der Architektur definiert, dass Microservice zu Datenbank eine „1:1"-Beziehung darstellen soll. Das bedeutet auf der einen Seite eine Entflechtung der Schnittstellen auf Datenbank-Ebene; auf der anderen Seite steigt allerdings die Zahl der notwendigen Datenbanken stark an. Mit der Multitenant-Option kann diese Anforderung abgedeckt werden, ohne die Server-Ressourcen aufstocken zu müssen.

Wie eine solche Datenbank auszusehen hat, war also klar. Jetzt ging es noch darum, diese quasi in „no time" zur Verfügung zu stellen. Als Lösungsansatz baute man eine Container-Datenbank (CDB) und erstellte darin eine Template-PDB, die bereits alle standardmäßigen Komponenten enthält (Applikations-Tablespace, Applikations-User-Rolle etc.).

Via REST-Schnittstelle wird auf eine zentrale Verwaltungs-Datenbank verbunden und eine Funktion aufgerufen, die im Hintergrund per Datenbank-Link auf die entsprechende CDB verbindet. Dort ruft sie mit „scheduled-job" die Host-Skripte „clone_pdb_4_app", „add_oid_entry" und „register_pdb_in_oem" auf. Die aufrufende Schnittstelle erhält zum Schluss das Resultat, also die Meldung, ob die PDB fehlerfrei angelegt werden konnte, beziehungsweise den aufgetretenen Fehler. Die *Abbildung 1* zeigt, wie die automatische PDB-Erstellung in den Build-/Deployment-Prozess integriert wurde.

Oracle oder PostgreSQL für Docker Kubernetes

Um diese Frage beantworten zu können, hat man sich überlegt, was die wichtigsten Vorgaben für den Betrieb von Datenbanken in einem Docker-Container sind, der von Kubernetes gemanagt wird:

- Sicherstellung der Persistenz (Backup/Recovery, Desaster-fähig)

```
ENV PG_MAJOR=10 \
PG_VERSION=10.4 \
PG_SHA256="1b60812310bd5756c62d93a9f93de8c28ea63b0df254f428cd1cf1a4d9020048" \
PGDATA=/u02/pgdata \
PGDATABASE="" \
PGUSERNAME="" \
PGPASSWORD="" \
REGBACKUP="yes" \
LANG=en_US.utf8
```

Listing 1

- Kalkulierbare Kosten unabhängig von der eingesetzten Infrastruktur
- Performant und skalierbar
- Support durch Community und Spezialisten
- Security-Vorgaben müssen eingehalten werden können
- Möglichst kleines Docker-Image
- Entwicklung/Test/Produktion mit identischer Software

Zwei Punkte sind mit den aktuellen Oracle RDBMS unter den aktuell gültigen Lizenzbedingungen nur schwierig zu erreichen: kalkulierbare Kosten und die Größe des Docker-Image. Das gilt auch für die anderen etablierten Datenbank-Systeme, die bei der Mobiliar eingesetzt werden. Einzig PostgreSQL deckt die geforderten Punkte alle ab.

Nach mehreren Diskussionen mit den Entwicklern, die PostgreSQL als RDBMS-Alternative zu Oracle bevorzugen, wurde vom DBA-Team ein PoC mit PostgreSQL im Docker-/Kubernetes-Umfeld gestartet, in enger Zusammenarbeit mit dem Container-Solutions-Team und mit externer Unterstützung durch Daniel Westermann von dbi.

Entwicklung der Architektur

Die Entwicklung einer neuen Architektur ist immer eine Herausforderung – aber irgendwo muss man halt einfach mal anfangen. Ausgangspunkt war die nachfolgende Liste, die in einem Brainstorming zusammengetragen wurde:

- Datenbank-Name (Input)
- User-/Schema-Name und Passwort (Input)
- Extensions
- Parameter
- Backup ja/nein (abhängig vom Backup-Konzept)
- Public Schema Permissions
- Backup point in time
- Init/Upgrade, alles da
- Monitoring User
- Ad User und Rollen

Das sieht ziemlich simpel aus, fasst aber schon quasi alles zusammen, über was man sich Gedanken machen muss:

- Soll der Name der Datenbank als Parameter in den Container gegeben werden?
- Sollen Benutzername und Passwort als Parameter in den Container gegeben werden?
- Wie behandelt man die PostgreSQL-Parameter? Soll es ein Standard-Set für alle geben?
- Sind verschiedene Templates für verschiedene Anforderungen notwendig?
- Wie macht man in der Container-Welt ein Backup und vor allem einen Restore der Instanz? Braucht man das überhaupt?
- Was ist mit PITR? Wohin will man archivieren?
- Was macht man mit Minor- und Major-Versionsupgrades von PostgreSQL?
- Soll das automatisch laufen?
- Wie überwacht man in der Container-Welt?
- Verwendet man einen vorgefertigten PostgreSQL-Container oder baut man ihn selbst?

Die Entscheidung, den PostgreSQL-Container selbst zu bauen, wurde aus unterschiedlichen Gründen sehr schnell getroffen: Einerseits war wichtig, die volle Kontrolle über das Container-Image zu haben, damit nicht benötigte Optionen auch nicht in den Container kommen. Andererseits soll das Container-Image so klein wie möglich sein und man wollte vor allem den Upgrade-Prozess selbst nach eigenen Vorstellungen steuern können.

Da man PostgreSQL problemlos selbst vom Source-Code kompilieren kann, war das initiale Container-Image schnell gebaut. Die vereinfachte Prozedur dazu ist:

- Installation der benötigten Betriebssystem-Pakete
- Download des PostgreSQL-Source-Codes
- Makefile erzeugen
- Kompilieren und installieren
- Deinstallation aller Pakete, die es nicht mehr braucht, um das Image klein zu halten

Zusammengefasst gehen folgende Variablen in den Container *(siehe Listing 1)*. Sobald der Container startet, werden sie verarbeitet und bestimmen die Version von PostgreSQL, den Ort, an dem die Datenbank-Dateien gespeichert werden, den Namen der Datenbank, den Benutzernamen und das zugehörige Passwort. Zuletzt gibt es noch die „REGBACKUP"-Variable, die steuert, ob für diesen Container Datenbank-Backups erstellt werden sollen. Da ein PostgreSQL-Minor-Upgrade nichts anderes bedeutet, als die neuen Binaries zu installieren und die Instanz mit diesen zu starten, ist dieses Thema schon erledigt: Sobald ein neues PostgreSQL-Minor-Release herauskommt, baut man den Container mit diesem Release neu. Sobald ein Datenbank-Container neu startet, wird er die neue Version des Image anziehen und der Minor-Upgrade ist erledigt.

Major-Version-Upgrades gestalten sich schwieriger: Sobald ein neues Major-Release zur Verfügung steht, wird das entsprechende Container-Image mit dieser Version gebaut. Zusätzlich enthält dieses Image aber auch das letzte Minor-Release der Vorgänger-Version. Somit lässt sich beim Starten des Containers prüfen, ob ein neues Major-Release zur Verfügung steht und der Upgrade dann automatisiert durchgeführt wird.

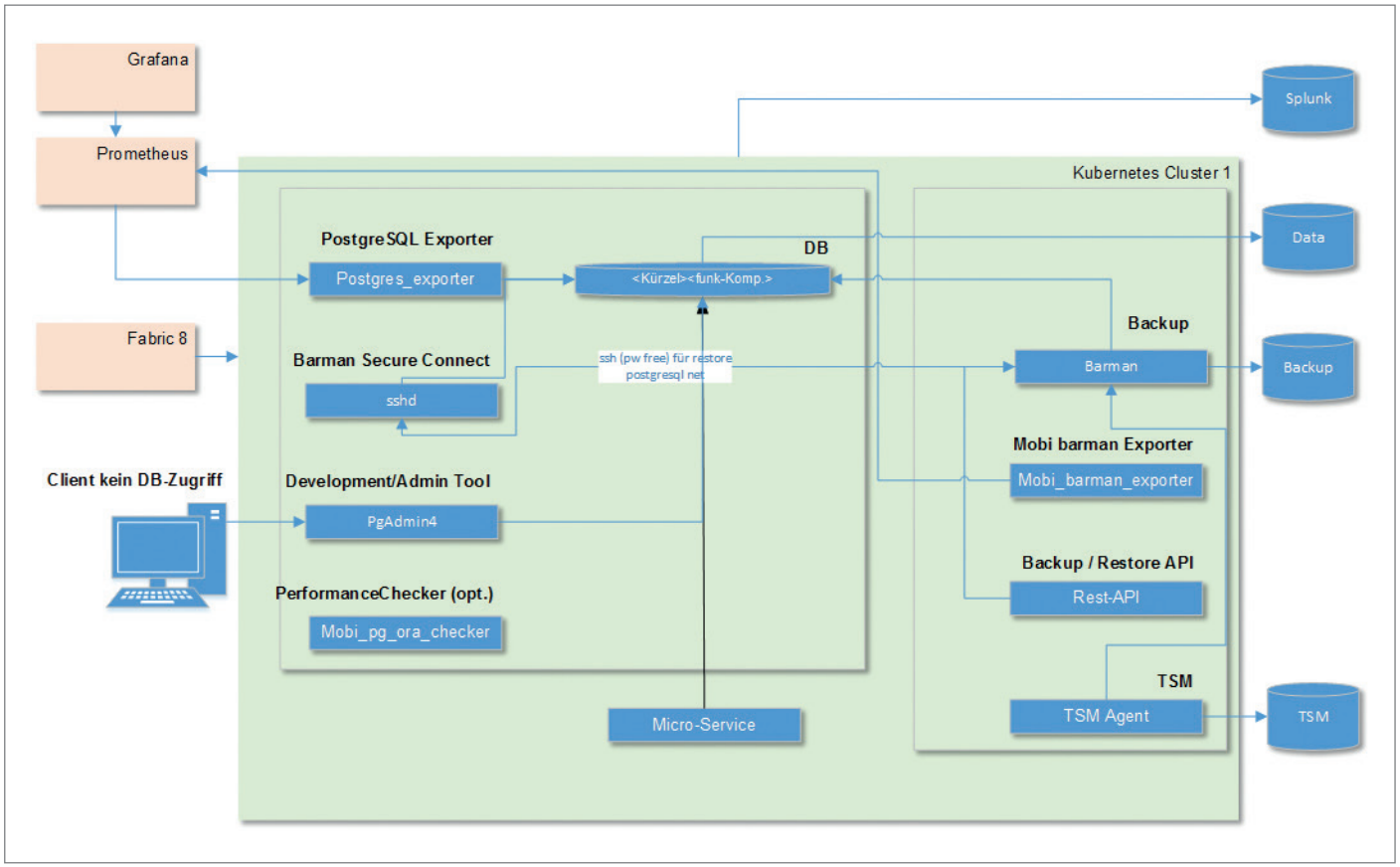

Abbildung 2: Die neue Architektur

Das führt zum nächsten Punkt: Wie will man die Instanzen sichern und wiederherstellen? Da voll auf Open-Source gesetzt wird, war man sich ziemlich schnell einig, Barman zu verwenden. Barman ist ein Community-Produkt, das in der PostgreSQL-Welt sehr verbreitet und beliebt ist. Die Lösung besteht darin, einen eigenen Barman-Container pro Kubernetes-Cluster laufen zu lassen, der dann auf alle Datenbank-Container Zugriff hat, und so ein zentrales Backup-Repository aufzubauen.

Zusätzlich brauchte es noch etwas, womit die Business-Analysten einfach Abfragen gegen die Datenbank machen können. Ein eigener „pgadmin4"-Container ist dafür vollständig ausreichend. Darum wurde pro Datenbank-Deployment ein „pgadmin4"-Container zur Verfügung gestellt.

Da im Docker-Umfeld Log-Meldungen nach „stdout" gehen, lag es auf der Hand, diese von Splunk abgreifen zu lassen. Somit werden alle Log-Meldungen zentral von Splunk verwaltet und auch das Alerting ist sichergestellt.

Blieb das Thema „Monitoring": Das Produkt, das im Kubernetes-Umfeld zum Einsatz kommt, ist Prometheus. Hier musste man nichts neu erfinden, sondern lediglich dafür sorgen, dass Metriken aus PostgreSQL heraus geliefert werden können, die dann von Prometheus weiterverarbeitet werden. Darauf aufbauend kommt dann Grafana zum Einsatz zur Visualisierung der Daten. Die komplette heute eingesetzte Architektur sieht wie in *Abbildung 2* aus.

Fazit

Auch wenn das Oracle-RDMS in Docker betrieben werden kann, liegen die Vorteile doch klar bei Open-Source-Lösungen wie PostgreSQL. Nicht nur ist das Produkt ausgereift, auch das Open-Source-Modell überzeugt, sodass der fehlende Hersteller-Support nicht als Show-Stopper angesehen werden muss. Die besonderen Herausforderungen im Docker-Umfeld wie ein performantes Storage-System sind unabhängig vom eingesetzten Datenbank-System zu lösen.

Hans Eichenberger
hans.eichenberger@mobi.ch

Daniel Westermann
daniel.westermann@dbi-services.com