Oracle, PostgreSQL, Docker und Kubernetes bei der Mobiliar Versicherung

Hans Eichenberger
Die Mobiliar
Bern, Schweiz

Daniel Westermann
dbi services
Delémont, Schweiz

Schlüsselworte
Multitenant, RDBMS, PostgreSQL, Docker, Kubernetes, DevOps, Lifecyle, Cloud

Einleitung

Agile Development, Docker, Cloud, DBaaS: Diese und ähnliche Begriffe werden auch bei der Mobiliar seit ein paar Jahren immer häufiger genannt. Kann Oracle als strategisches RDBMS-Produkt dies alles abdecken - oder was können Gründe dafür sein, dass auch Open Source-Datenbanksysteme genauer angeschaut werden?

In der Mobiliar werden bereits diverse Applikationen als Microservices in einem Docker/Kubernetes- Umfeld entwickelt und betrieben. Vor zwei Jahren ist dazu von der Mobiliar IT eine REST-Schnittstelle gebaut worden. Diese erlaubt es uns, während des Deployments einer Applikation in Kubernetes bei Bedarf auch die Datenbank vollautomatisch bereitzustellen. Diese Lösung funktioniert in-house zuverlässig und erstellt bzw. löscht die PDBs ohne manuelle Eingriffe seitens DBA.

Die Anforderung, dass z.B. für Testinstallationen sowohl In-House- als auch Cloud-Infrastrukturen einsetzbar sein sollen, um die Bedürfnisse der Entwicklung vollumfänglich abdecken zu können, wird auch in der Mobiliar immer wichtiger. Ein zentraler Punkt in diesem Thema sind neben der Geschwindigkeit vor allem auch kalkulierbare Kosten. Mit der aktuellen PDB-Order-Schnittstelle erzwingt dieser Punkt, dass z.B. für Tests auf den Entwickler-Notebooks nicht eine EE-Installation gemacht werden kann, sondern die lange nicht mehr aktualisierte XE-Version eingesetzt werden muss.

Technisch gesehen verwenden Microservices meist „nur“ die Grundfeatures eines Datenbanksystems. Damit sind die Persistenz und der schnelle Zugriff auf die Daten sichergestellt. Das Deployment in Kubernetes kann vereinfacht werden, wenn alle notwendigen Komponenten direkt von Kubernetes gemanaged werden und keine Schnittstellen zu externen Ressourcen gebaut werden müssen, z.B. zum Erstellen von PDBs. Darum mussten wir eine Lösung mit der Datenbank im Docker-Container suchen.

In diesem Vortrag zeigen wir den Weg auf von der aktuellen Lösung bis zur Entwicklung der Architektur der PostgreSQL-Docker-Implementierung.

Automatisierte PDB-Bereitstellung via REST-Schnittstelle

Damit das agile Entwickeln nicht beim manuellen, sprich langsamen, konventionellen Deployment scheitert, haben wir in der Mobiliar vor ca. zweieinhalb Jahren eine Lösung gesucht, wie Datenbanken möglichst automatisiert und schnell bereitgestellt werden können. Es stellt sich relativ schnell heraus, dass die Multitenant-Option, die seit Oracle 12.1 verfügbar ist, die Anforderungen an die DBs im agilen Umfeld am besten abdecken kann. Mit

der Modularisierung der Applikation, sprich dem Schneiden der bestehenden „Monolithen“ in mehrere Microservices, wurde seitens der Architektur definiert, dass Microservice zu DB eine 1:1-Beziehung darstellen soll. Das bedeutet auf der einen Seite eine Entflechtung der Schnittstellen auf DB-Ebene. Auf der anderen Seite aber steigt die Zahl der notwendigen DBs stark an. Mit der Multitenant Option kann diese Anforderung abgedeckt werden, ohne dass die Server-Ressourcen aufgestockt werden müssen.

Wie eine solche DB auszusehen hat, war also klar. Jetzt ging es noch darum, diese DB in quasi „no-time“ zur Verfügung zu stellen. Als Lösungsansatz wurde eine Container-DB (CDB) gebaut und darin eine Template PDB erstellt, die bereits alle standardmässigen Komponenten enthielt (App-Tablespace, App-User-Rolle, etc.). Via REST-Schnittstelle wird auf eine zentrale Verwaltungs-DB connected und eine Funktion aufgerufen, die im Hintergrund via DB-Link auf die entsprechende CDB connected und dort via scheduled-job die Host-Scripts: clone_pdb_4_app, add_oid_entry, register_pdb_in_oem aufruft. Die aufrufende Schnittstelle erhält zum Schluss das Resultat, d.h. ob die PDB fehlerfrei angelegt werden konnte bzw. den aufgetretenen Fehler.

Die folgende Grafik zeigt, wie die automatische PDB-Erstellung in den Build/Deployment-Prozess integriert wurde.

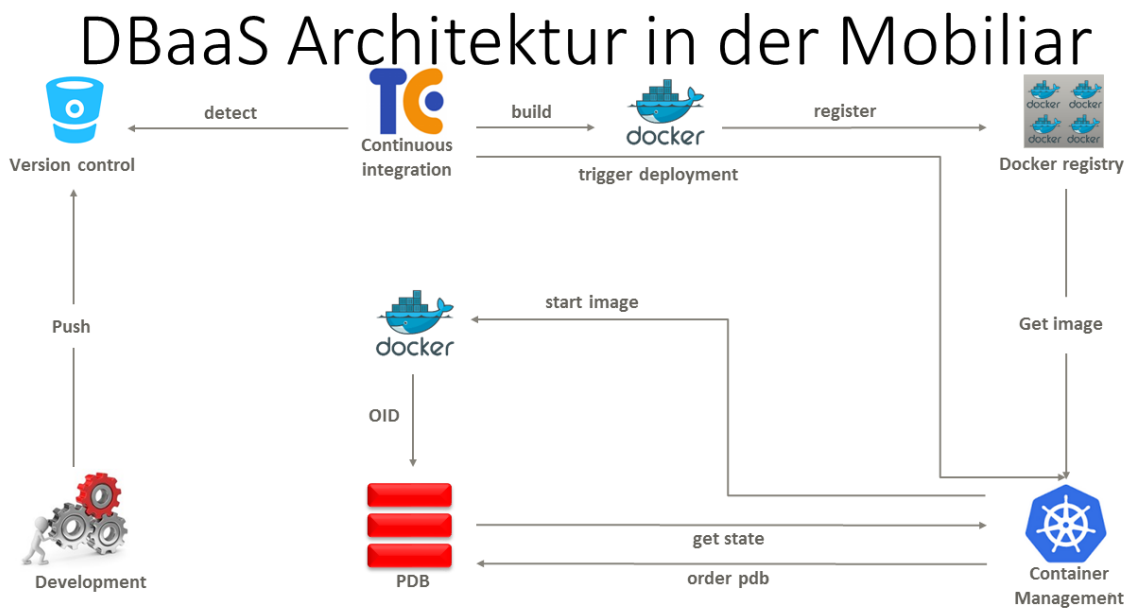


Abbildung 1 Schematische Darstellung DBaaS Architektur in der Mobiliar

Oracle oder PostgreSQL für Docker Kubernetes

Dies sind die wichtigsten Vorgaben für den Betrieb von Datenbanken in einem Docker-Container, der von Kubernetes gemanaged wird:

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

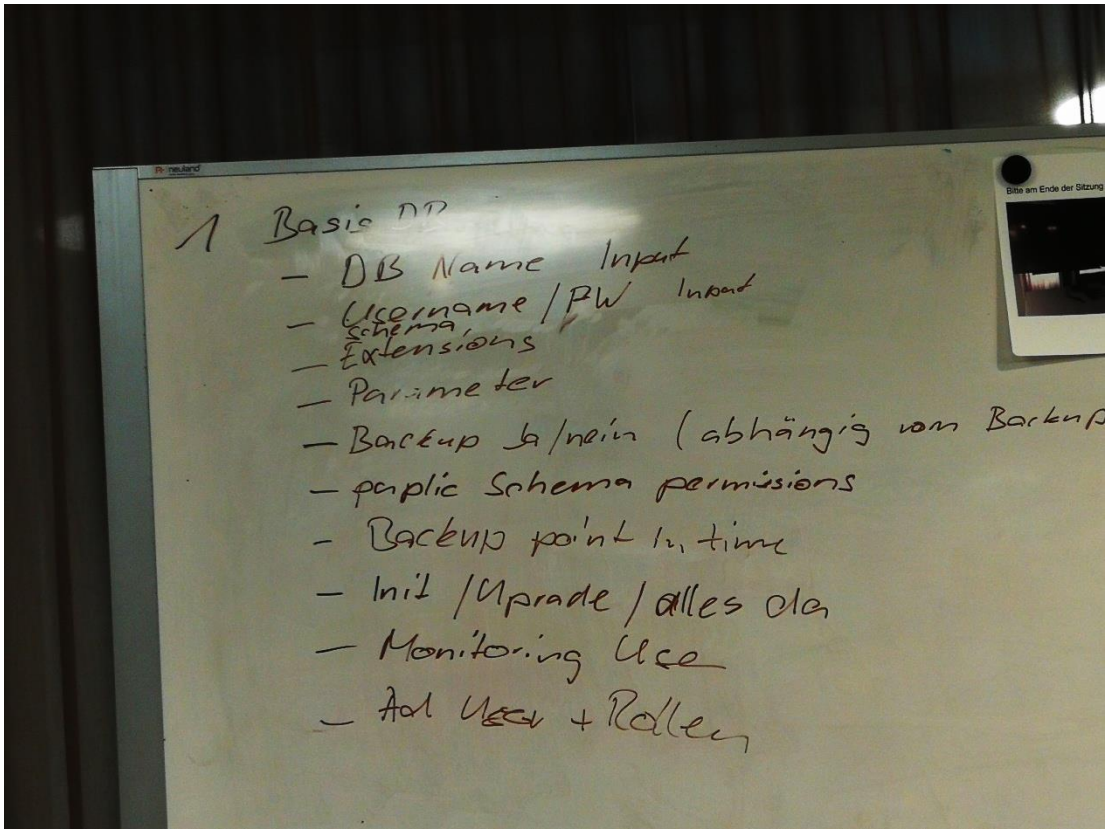
- Kalkulierbare Kosten unabhängig von der eingesetzten Infrastruktur
- Performant und skalierbar
- Support durch Community und Spezialisten
- Security-Vorgaben müssen eingehalten werden können
- Möglichst kleines Docker-Image
- Entwicklung / Test / Produktion mit identischer SW

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

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 externer Unterstützung durch Daniel Westermann von dbi.

Entwicklung der Architektur

Die Entwicklung einer neuen Architektur ist immer eine Herausforderung - und irgendwo muss man halt einfach mal anfangen. Unser Ausgangspunkt war dieser:



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 geben werden?
- Sollen Benutzername und Passwort als Parameter in den Container gegeben werden?
- Wie behandeln wir die PostgreSQL-Parameter? Soll es ein Standard-Set für alle geben? Brauchen wir verschiedene Templates für verschiedene Anforderungen?
- Wie macht man in der Container-Welt ein Backup und vor allem einen Restore der Instanz? Brauchen wir das überhaupt?
- Was ist mit PITR? Wohin wollen wir archivieren?
- Was machen wir mit Minor- und Major-Versionsupgrades von PostgreSQL? Soll das automatisch laufen?
- Wie überwachen wir in der Container-Welt?
- Nehmen wir einen vorgefertigten PostgreSQL-Container oder bauen wir ihn selbst?

Die Entscheidung, den PostgreSQL-Container selbst zu bauen, wurde aus unterschiedlichen Gründen sehr schnell getroffen: Einerseits war es uns wichtig, die volle Kontrolle über das Container Image zu haben, damit Optionen die wir nicht brauchen, auch nicht in den Container kommen. Andererseits soll das Container Image so klein als möglich werden und wir wollen vor allem den Upgrade Prozess selbst nach unseren 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
- De-installation alle Pakete, die es nicht mehr braucht, um das Image klein zu halten

Zusammengefasst gehen folgende Variablen in den Container:

```
ENV PG_MAJOR=10 \

PG_VERSION=10.4 \

PG_SHA256="1b60812310bd5756c62d93a9f93de8c28ea63b0df254f428cd1cf1a4d9020048
" \

PGDATA=/u02/pgdata \
```

```
PGDATABASE="" \
PGUSERNAME="" \
PGPASSWORD="" \
REGBACKUP="yes" \
LANG=en_US.utf8
```

Sobald der Container startet, werden diese Variablen 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, haben wir dieses Thema schon erledigt: Kommt ein neuer PostgreSQL Minor Release, bauen wir den Container mit diesem Release neu. Sobald ein Datenbank Container neu startet, wird er die neue Version des Images anziehen und der minor Upgrade ist erledigt.

Major Version Upgrades gestalten sich schwieriger, konnten aber so gelöst werden: Sobald ein neuer Major Release zur Verfügung steht, wird das entsprechende Container Image mit dieser Version gebaut. Zusätzlich enthält dieses Image aber auch den letzten Minor Release der Vorgängerversion. Somit kann beim Starten des Containers geprüft werden, ob ein neuer Major Release zur Verfügung steht und der Upgrade dann automatisiert durchgeführt wird.

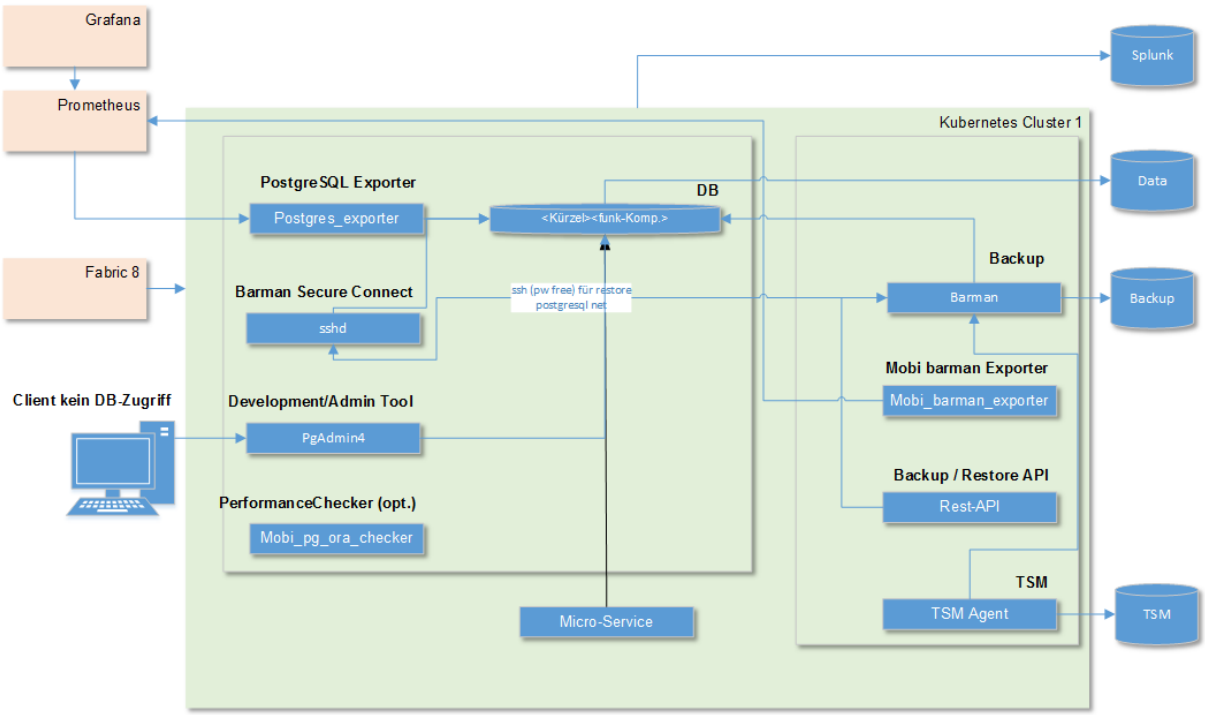
Das brachte uns zum nächsten Punkt: Wie wollen wir die Instanzen sichern und wieder herstellen?
Da wir voll auf Open Source setzen wollten, waren wir uns ziemlich schnell einig, Barman zu verwenden. Barman ist ein Community-Produkt, das in der PostgreSQL-Welt sehr verbreitet und beliebt ist. Die Lösung war, 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 könnten. Ein eigener pgadmin4-Container ist dafür vollständig ausreichend.
Darum haben wir pro Datenbank-Deployment einen 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 Logmeldungen 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 mussten wir nichts neu erfinden, sondern haben lediglich dafür gesorgt, dass Metriken aus PostgreSQL heraus geliefert werden können, die dann von Prometheus weiter verarbeitet werden. Darauf aufbauend kommt dann Grafana zum Einsatz zur Visualisierung der Daten.

Die komplette heute eingesetzte Architektur sieht so aus:



Kontaktadressen:

Hans Eichenberger
Schweizerische Mobiliar
Versicherungsgesellschaft AG
Monbijoustrasse 68
CH-3001 Bern

Telefon: +41 (0) 31-389 69 88

E-Mail hans.eichenberger@mobiliar.ch
Internet: www.mobiliar.ch

dbi services
Daniel Westermann
Rue de la Jeunesse 2
CH-2800 Delémont

Telefon: +41 32 422 96 00
Fax: +41 32 422 96 15
E-Mail daniel.westermann@dbi-services.com
Internet: www.dbi-services.com