DevOps und SaaS mit Datenbank-Application-Containern

Dr.-Ing. Holger Friedrich, sumIT AG

Management, Entwicklung und Betrieb von Datenmodellen und datenbankbasierten Applikationen haben heute zwei Herausforderungen zu bewältigen: zum einen die Bewirtschaftung vieler Instanzen, sei es im Rahmen des Entwicklungs- und Testprozesses oder des Betriebs von Multi-Mandantensystemen (SaaS-Anwendungen), und zum anderen besteht die Notwendigkeit, agil und mit immer kürzeren Release-Zyklen hin zu Continuous Delivery zu kommen. Multi-Tenant-Applikations-Container sind ein neues Feature der Oracle-Datenbank, das bei der Bewältigung beider genannten Herausforderungen unterstützt.

Virtualisierung ist mittlerweile fast schon ein alter Hut. Zu Beginn waren es virtuelle Maschinen, die die Provisionierung einzelner Umgebungen und Systeme vereinfachten und die sichere, effiziente Nutzung geteilter Ressourcen ermöglichten. Mittlerweile zieht sich die Virtualisierung durch alle Aspekte eines Rechenzentrums. Virtualisiertes Netzwerk, virtueller Storage – der Entwicklung scheinen keine Grenzen gesetzt.

Während der letzten Jahre gab es allerdings Entwicklungen in der IT-Industrie, bei denen die Virtualisierung von Servern an ihre Grenzen stößt, ja schon zu ineffizient, zu altbacken geworden ist. Insbesondere sind dies die agile Software-Entwicklung sowie Software-as-a-Service/Cloud.

Agile Software-Entwicklung mit all ihren Bestandteilen, den kurzen Entwicklungs-Sprints, dem hohen Grad der Test-Automatisierung und schließlich dem Ziel des kontinuierlichen Deployments, erfordert den schnellen Auf- und Abbau zahlreicher Applikations-Entwicklungs- und Test-Umgebungen. Software-as-a-Service wiederum verlangt nach der schnellen Provisionierung von Umgebungen für neue Mandanten. In beiden Anwendungsfällen sollten Bereitstellung, Betrieb, aber auch Dekommissionierung möglichst hoch automatisierbar sein. Selbstbedienung ist erwünscht. Die Cloud lässt grüßen.

Gegenüber diesen hohen Ansprüchen hat die klassische Server-Virtualisierung mit virtuellen Maschinen deutliche Nachteile. Diese beruhen vor allem darauf, dass sie in jeder Instanz eine vollständige Betriebssystem-Installation beherbergen. Das macht VMs wartungsintensiv, groß und sehr Ressourcen-hungrig.

Zur Lösung dieses Problems wurde die Container-Technologie entwickelt. Hier beherbergt die virtuelle Maschine, also der Container, nur die Komponenten der Kunden-Applikation sowie einige Schnittstellen-Binaries und -Libraries. Diese kommunizieren mit der Server-Plattform, die eine Betriebssystem-Installation sowie die Container-Engine für alle gehosteten Container enthält. Die Applikationen kommunizieren über die Schnittstellen-Libraries in den

Containern mit der Container-Engine und dadurch mit der Betriebssystem-Installation. Aus dieser Architektur ergibt sich eine Reihe von Vorteilen von Containern gegenüber klassischen VMs *(siehe Tabelle 1)*.

Mittlerweile hat sich die Welt weitergedreht; selbst Container erscheinen bereits zu groß und zu unflexibel. Serverless-Architekturen sind der neueste Trend. Dabei ruft die eigene Applikation, natürlich als eine orchestrierte Menge von Micro-Services, nur noch APIs auf. Der Code lebt und läuft irgendwo in der Server-Landschaft eines Cloud-Providers, ohne dass Entwickler und Applikations-Owner sich um Provisionierung und Pflege kümmern müssen.

Alle diese Entwicklungen sind spannend und sehr gut für Betrieb und Entwicklung von Applikationen im Middle-Tier einsetzbar. Wie steht es jedoch mit dem Back-End von Applikationen, typischerweise relationalen Datenbank-Systemen? Auch diese müssen ihre Rolle in der agilen Software-Entwicklung und in Saas-Architekturen spielen und ebenfalls automatisiert provisioniert und betrieben werden können.

Ist es eine gute Idee, relationale Datenbank-Managementsysteme wie eine Oracle-Datenbank ebenfalls im Container zu installieren? Außer für sehr kleine Installationen wahrscheinlich eher nicht. Was dagegen spricht, ist die Tatsache, dass ein relationales Datenbank-Managementsystem nicht nur Applikationslogik enthält, sondern auch die Daten persistiert. Diese sind bei ernsthaften Anwendungen wie ERP-Systemen oder gar Data Warehouses umfangreich und wachsen mit der Zeit. Container-Technologie hingegen ist, wie gezeigt, für flexible, schlanke Installationen gedacht. Man kann natürlich die Datenbank-Software im Container installieren, die Server-Prozesse darin laufen lassen und den persistenten Storage, also die Datenbank-Dateien, außerhalb des Containers verwalten. Dies schafft allerdings Engpässe beim Zugriff und Abhängigkeiten, die das einfache Umherschieben und Provisionieren verhindern.

So lässt sich also das Fazit ziehen, dass Virtualisierung und Container-Technologie für Applikations-Entwicklung und -Betrieb hervorragend geeignet sind, sich für Datenbank-Installationen aber nur begrenzt eignen. Andererseits werden mehr Flexibilität, höherer Automatisierungsgrad und mehr Agilität auch im Datenbank-Umfeld benötigt.

Virtuelle Maschinen	Container
Benötigt Hypervisor und eine vollständige OS-Installation in jeder VM	Spricht mit dem Host-Kernel (keine OS-Installation im Container)
Größerer Platzbedarf (RAM und nicht-flüchtiger Speicher)	Kleiner Footprint
In der Regel größer als 1 GB	Oftmals nur wenige MB
Startup in Minuten	Startup in Sekunden
komplexes Deployment (Netzwerk-Bandbreite etc.)	Einfaches Deployment
Langsamer	Schneller
Sicherheitsprobleme mit installiertem OS	Sicherheitsprobleme ausschließlich mit eigener Applikation

Tabelle 1: Klassische Virtualisierung und App-Container im Vergleich

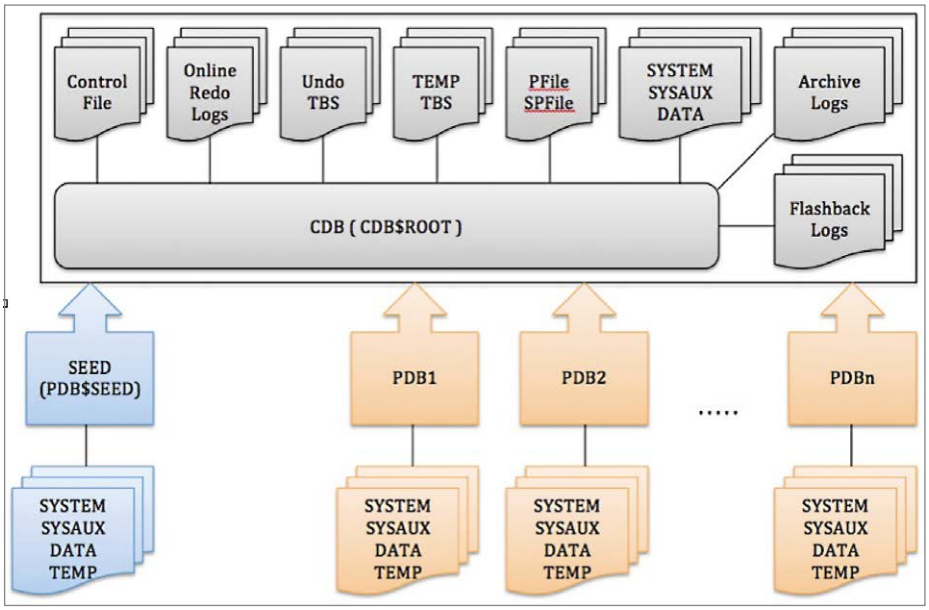

Abbildung 1: Architektur der Oracle-Container- und Pluggable-Datenbanken

Oracle-Datenbank-Applikations-Container

Oracle hat den Bedarf erkannt und veröffentlichte mit der Datenbank-Version 12c eine Technologie, die Container für Datenbanken und Datenbank-Applikationen bereitstellt. In Release 12.1 war es die Container-Technologie für Datenbanken in Form von Container- und Pluggable-Datenbanken, in Release 12.2 kamen Datenbank-Applikations-Container hinzu. Die Architektur von Container- und Pluggable-Datenbanken beruht auf der Trennung allgemeiner Datenbank-Managementprozesse, -Metadaten und -Daten von applikationsspezifischen Datenstrukturen und Logik *(siehe Abbildung 1)*.

Die Container-Datenbank ist in Grau dargestellt. Sie stellt für alle angeflanschten Pluggable-Datenbanken die zentralen Management-Prozesse sowie dafür benötigte Strukturen und Daten zur Verfügung. Blau ist die Seed-Datenbank, aus der einfach neue Datenbanken geklont werden können. In orangener Farbe sind die Nutzer-Datenbanken angezeigt. Die Architektur erlaubt das Management vieler Datenbanken in einem Container, der CDB. Typische Wartungsprozesse, wie Backup und Recovery, Upgrades, Patching, aber auch die Provisionierung neuer Datenbanken und die Migration von Datenbanken auf andere Umgebungen, sind so stark vereinfacht. Immer mehr dieser Operationen sind auch online, also ohne Unterbrechung des Applikationsbetriebs, möglich.

Diese Architektur war für Tätigkeiten auf Datenbank-Administrationsebene bereits hervorragend. Bis Release 12.2 fehlte jedoch die Möglichkeit, Datenbank-Applikations-Instanzen, ihre Strukturen, Pro-

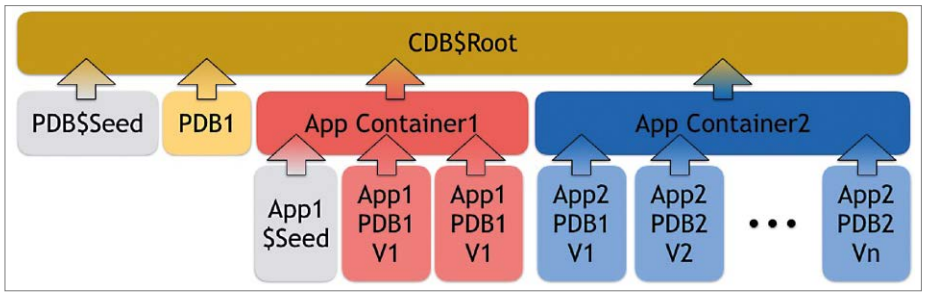

Abbildung 2: Architektur der Datenbank-Applikations-Container

gramm-Pakete und -Logik ähnlich komfortabel zu verwalten. Dafür hat Oracle das Konzept der Container-Datenbank um eine Ebene erweitert. Neben der Container-Datenbank und darunter angeflanschter Pluggable-Datenbanken gibt es nun auch Applikations-Container- und Applikations-Datenbanken *(siehe Abbildung 2)*.

Unter der Container-Datenbank (CDB) können außer der Seed-Datenbank und beliebiger Pluggable-Datenbanken (PDBs) nun auch Applikations-Container-Datenbanken eingehängt werden. App-Container sind nichts Weiteres als reguläre PDBs, die bei der Erstellung als App-Container gekennzeichnet und initialisiert werden. Mit dieser Initialisierung erhält die PDB die Funktionalität, Applikationen zu verwalten und sie in angeschlossenen Applikations-PDBs zu bewirtschaften.

In die App-Container können nun App-PDBs eingesteckt werden, analog zu den regulären PDBs, die direkt am „CDB$Root" hängen. App-PDBs können dabei aus einer App-Seed-PDB geklont, neu erstellt oder auch von beliebigen anderen PDBs geklont werden.

Über einen App-Container können Applikationen zentral installiert, upgegradet, gepatcht und deinstalliert werden. Wann Upgrades in angehängten App-PDBs wirksam werden, ist frei wählbar. Darüber hinaus kann der App-Container zentral Daten für alle angehängten App-PDBs verwalten. Auf diese Weise sind Daten allgemeinen Interesses schreibgeschützt allen provisionierten App-PDBs zugänglich.

Zwei Voraussetzungen müssen erfüllt sein, um DB-App-Container nutzen zu können. Die eine ist lizenzrechtlicher Natur, die andere betrifft die Konfiguration der Container-Root-Datenbank:

- Zur Nutzung des DB-App-Container-Features sind eine Enterprise-Edition-Datenbank-Lizenz und zusätzlich die Lizenzierung der Multi-Tenant-Option erforderlich.
- Der Datenbank-Root-Container, in dem DB-App-Container und App-DBs betrieben werden sollen, muss zur Anwendung von Oracle Managed Files (OMF) konfiguriert sein. Dies ist notwendig, da das DBMS im Verlaufe der Nutzung neue Datenbank-Files und PDBs erzeugen und dafür die Namen selbstständig generieren und verwalten können muss. OMF muss spätestens vor der Erstellung des ersten DB-App-Containers eingeschaltet sein.

Einen Datenbank-Applikations-Container erstellen

Die Erstellung eines DB-App-Containers ist denkbar einfach. Sie besteht aus dem Einrichten einer PDB, erweitert um die Klausel „AS APPLICATION CONTAINER". Dafür müssen der Anwender die SYSDBA-Berechtigungen haben und der Kontext der Session auf dem „CDB$Root"-Container stehen. *Listing 1* löst die Erstellung aus.

Der Applikations-Container ist ausschließlich zur Verwaltung von Applikationen bestimmt. Um eine Applikation zu nutzen, ist mindestens eine Applikations-PDB erforderlich. Diese kann als Klon einer bestehenden PDB, als Klon einer „APP$Seed"-PDB oder als neue App-PDB erzeugt werden. Ein User mit SYSDBA-Rechten oder der Admin-User des Applikations-Containers löst die Erstellung der App-PDB aus. Dabei muss der Session-Kontext zunächst auf den DB-App-Container gesetzt werden *(siehe Listing 2)*. Nachdem die App-PDB erstellt und geöffnet wurde, können bereits im App-Container installierte Applikationen in ihrer aktuellen Version in die App-PDB synchronisiert werden *(siehe Listing 3)*.

Im Zuge der Synchronisation werden in der PDB Applikations-User, Datenbankdateien, Datenstrukturen und auch Inhalte angelegt beziehungsweise mit dem App-Container verlinkt. Dabei gibt es verschieden Möglichkeiten, um Strukturen und Inhalte zu organisieren und zwischen App-Container und App-PDBs zu (ver-)teilen.

(Meta-)Daten teilen oder lokal halten

Sowohl Metadaten, also Struktur- und Logik-Definitionen, als auch Nutzdaten können zwischen App-Containern und ihren App-PDBs geteilt oder lokal in den App-PDBs gehalten werden. Dafür gibt es die neue Klau-

```
-- erstellen und öffnen des Application Containers
CREATE PLUGGABLE DATABASE appcon1 AS APPLICATION CONTAINER
   ADMIN USER appcon_admin IDENTIFIED BY ac1manager;
ALTER PLUGGABLE DATABASE appcon1 OPEN;
```

Listing 1

```
ALTER SESSION SET container = appcon1;
-- Erstelle Application PDB
CREATE PLUGGABLE DATABASE app1pdb1
ADMIN USER pdb_admin IDENTIFIED BY a1p1manager;
ALTER PLUGGABLE DATABASE app1pdb1 OPEN;
```

Listing 2

```
-- initialisere neue Application PDB
ALTER SESSION SET container = app1pdb1;
ALTER PLUGGABLE DATABASE APPLICATION ALL SYNC;
```

Listing 3

```
ALTER PLUGGABLE DATABASE APPLICATION
{ { app_name
    { BEGIN INSTALL 'app_version' [ COMMENT 'comment' ]
    | END INSTALL [ 'app_version' ]
    | BEGIN PATCH number [ MINIMUM VERSION 'app_version' ] [ COMMENT 'comment' ]
    | END PATCH [ number ]
    | BEGIN UPGRADE 'start_app_version' TO 'end_app_version' [ COMMENT 'comment' ]
    | END UPGRADE [ TO 'end_app_version' ]
    | BEGIN UNINSTALL
    | END UNINSTALL
    | SET PATCH number
    | SET VERSION 'app_version'
    | SET COMPATIBILITY VERSION { 'app_version' | CURRENT }
    | SYNC  }
  |
  { ALL SYNC }
}
```

Abbildung 3: API zur Applikations-Erstellung und -Verwaltung

sel „SHARING" für „Create DDL"-Statements. Damit wird festgelegt, wo Objekte und Daten lokalisiert sein sollen und welche Art des Zugriffs gestattet wird. Es gibt vier Arten der Definition und Berechtigungen.

- *Metadata*
  Dies ist die Default-Option. „Metadata-gesharte Objekte" bedeutet, dass die Definition im App-Container liegt und alle App-PDBs diese nutzen können, ohne sie allerdings verändern zu können. Die Definition einer Metadata-gesharten Tabelle ist in den App-PDBs unveränderlich. Jede App-PDB kann jedoch ihre eigenen Daten lokal darin verwalten.

- *Data*
  Sind Strukturen DATA-geshart, ist nicht nur die Definition im App-Container hinterlegt, sondern auch der Daten-Inhalt. App-PDBs können die Tabelle lesen, ihren Inhalt durch Constraints von lokalen Daten referenzieren, aber weder Struktur noch Inhalt verändern. Diese Option ist insbesondere für Stammdaten geeignet, die zentral gewartet werden sollten, etwa ein Postleitzahlen-Verzeichnis.

- *Extended Data*
  Diese Sharing-Option entspricht der „Data"-Option, mit der Erweiterung, dass App-PDBs eigene Daten in Form von Zeilen in die Struktur einbringen können. Die eigenen Daten sind nur innerhalb der betreffenden PDB sichtbar.

- *None*
  Die letzte Option ist für Strukturen gedacht, die lokal in jeder App-PDB erstellt werden sollen und von dieser danach in vollem Umfang verändert werden können.

Management-Views

Zur Kontrolle des Installationszustands, der Version und des Patch-Levels von Applikationen stehen im App-Container und den App-PDBs verschiedene Data-Dictionary-Views zur Verfügung. Insbesondere sinnvoll sind diese:

- *dba_applications*
  Gibt Auskunft über installierte Applikationen und deren Version beziehungsweise Installationsstatus

- *dba_app_patches*
  Enthält Informationen über installierte Applikationspatches

Release 12.2 ist das initiale Release der DB-App-Container-Funktionalität. Das API zur Erstellung und Pflege von Applikationen ist sehr übersichtlich und einfach zu nutzen. *Abbildung 3* zeigt das vollständige API mit Stand 12.2.

Applikationen werden zunächst im App-Container installiert. Dabei wird im Kontext des App-Containers die Installation initiiert und ein Name vergeben. Anschließend können alle Operationen zur Erstellung der Applikation ausgeführt werden. Dies beinhaltet die Erstellung von Datenbank-Benutzern, Tablespaces, Strukturen, Programmpaketen etc. *Listing 4* zeigt die beispielhafte Installation einer Applikation. Danach kann die neue Applikation dann in App-PDBs synchronisiert werden *(siehe Listing 5)*.

Upgrade von Applikationen

Applikations-Upgrades verlaufen prinzipiell gleich wie die Installation. Dazu wird das Upgrade im App-Container initialisiert. Dann kommen alle Upgrade-Skrip-

```
-- Beginn der Installation initiieren
ALTER PLUGGABLE DATABASE APPLICATION demoapp BEGIN INSTALL '1.0';
-- Ausführen aller Installtionsskripte
CREATE TABLE demoapp.bundesland SHARING=DATA (
   id             NUMBER,
   name           VARCHAR2(50),
   einwohnerzahl  NUMBER(22)
   CONSTRAINT bundesland_pk PRIMARY KEY (id));
ALTER TABLE demoapp.bundesland ADD CONSTRAINT bundesland_pk PRIMARY KEY
(id);
INSERT INTO demoapp.bundesland (id, name, einwohnerzahl)
   values (1, ,'Baden-Württemberg', 10879618);
INSERT INTO demoapp.bundesland (id, name, einwohnerzahl)
   values (2, 'Bayern', 12843514);
COMMIT;
-- Ende der Installation bestimmen
ALTER PLUGGABLE DATABASE APPLICATION demoapp END INSTALL '1.0';
```

Listing 4

te für die Applikation zur Ausführung, die die Entwickler bereitgestellt haben. Dabei können auch Strukturen, Benutzer etc. gelöscht werden. Zu guter Letzt wird wieder in die App-PDBs synchronisiert, wann immer es dafür Bedarf und ein Wartungsfenster gibt. Wie verhindert nun das Datenbank-Managementsystem, dass Änderungen im App-Container nicht sofort auf die App-PDBs durchschlagen?

Um dies zu bewerkstelligen, erzeugt das System im Hintergrund vollautomatisch und transparent einen Klon des App-Containers. Dieser enthält die bisherige Applikationsversion. Dann werden alle App-PDBs an diesen Schatten-Container umgehängt. All dies geschieht bei der Initiierung des Upgrades. Zum Ende des Upgrades ist der primäre App-Container im neuen Zustand, aber ohne eingehängte PDBs. Diese werden während des Synchronisierens upgegradet und vom Schatten-Container zum aktuellen Container umgehängt. *Abbildung 4* zeigt die drei Phasen des Upgrades.

```
ALTER SESSION SET container = app1pdb1;
ALTER PLUGGABLE DATABASE APPLICATION demoapp SYNC;
```

Listing 5

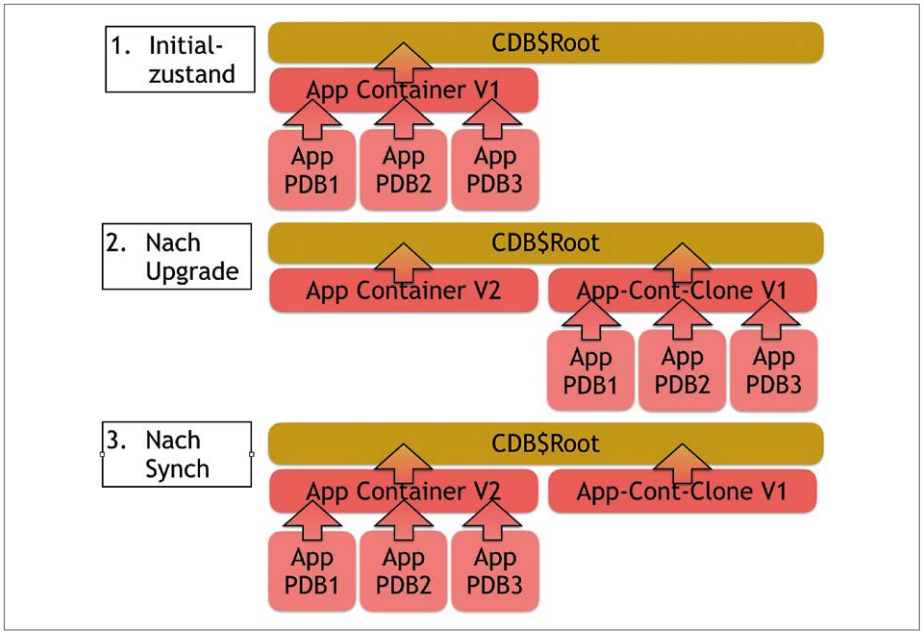

Abbildung 4: Evolution der App-Container und -PDBs beim Upgrade

Patching

Der Vorgang beim Patching gleicht für den Applikations-Verantwortlichen dem des Upgrades. Im Hintergrund geschieht allerdings etwas anderes, weswegen es Einschränkungen gibt. Beim Patching wird eine Patchnummer statt einer Versionsnummer vergeben. Wichtiger ist jedoch, dass kein Schatten-Container erzeugt wird und die App-PDBs nicht umgehängt werden. Daraus resultieren Einschränkungen bei den möglichen Operationen innerhalb des Patching. Destruktive Operationen wie das Löschen von Spalten, Tabellen oder Usern sind nicht gestattet und führen zu Fehlermeldungen. Statt innerhalb eines Patches müssen sie in ein Upgrade verpackt und mit diesem Mechanismus ausgebracht werden.

Das Entfernen von Applikationen erfolgt nicht selbstständig durch die Datenbank. Auch hierfür muss der Applikations-Entwickler Skripte bereitstellen. Wie bei einer Installation werden die Deinstallations-Skripte zunächst im App-Container ausgeführt. In den App-PDBs ist die Applikation dann allerdings noch vorhanden. Hinter den Kulissen wird eine Deinstallation also wie ein Upgrade behandelt. Ein Schatten-Container entsteht zu Beginn, die App-PDBs werden an diesen gehängt und die Deinstallation innerhalb der App-PDBs erfolgt durch Aufruf der Synchronisation *(siehe Abbildung 4)*.

Fazit

Datenbank-Applikations-Container können für das persistente Backend das sein, was Container-Technologie für den Middle-Tier Layer bereits sind, nämlich eine Methode, um schnell, ressourcensparend und kostengünstig viele Applikations-Instanzen zu erzeugen, zu verwalten und zu pflegen. Zusätzlich zu den beschriebenen Eigenschaften können auch geografisch verteilte Setups aufgebaut werden. Bei diesen sind mehrere DB-App-Container durch DB-Links miteinander verbunden. Mit solch einem Setup lassen sich alle angehängten App-PDBs mit einem Master-DB-App-Container synchronisieren. Dies ermöglicht einfaches Applikations-Management über Rechenzentrums- und Ländergrenzen hinweg.

Die Nutzung all dieser Eigenschaften ist im Rahmen agiler Software-Entwicklung, zur Unterstützung automatischen Testens und von Continuous Delivery ein großer Vorteil. In Software-as-a-Service-Anwendungs-Szenarien, bei denen neue Instanzen von Applikationen für Mandanten automatisch erzeugt und provisioniert werden müssen, können Datenbank-Applikations-Container ebenfalls eine Schlüsselrolle spielen.

Der Funktionsumfang in Release 12.2 ist bereits gut, um die Funktionalität gewinnbringend zu nutzen. Mit Spannung sind die weitere Entwicklung und zukünftige Erweiterungen in Release 18c zu erwarten.

Dr.-Ing. Holger Friedrich
holger.friedrich@sumit.ch