CI/CD für Oracle-APEX-Anwendungen mit GitLab

Johannes Michler und Simon Grossmann, Promatis Gruppe, Ettlingen (TechnologieRegion Karlsruhe)

Oracle Application Express (APEX) ist eine Low-Code-Plattform zur Erstellung von datenbasierten Webanwendungen, die direkt in einer Oracle-Datenbank installiert wird. Die Entwicklung einer solchen Webanwendung findet in der APEX-eigenen Entwicklungsumgebung statt. Dieser Artikel zeigt eine Möglichkeit auf, wie die Entstehung einer APEX-Applikation anhand CI/CD automatisiert auf verschiedene Umgebungen (TEST und PROD) ausgerollt werden kann.

Einleitung

Die Low-Code-Plattform Oracle APEX wird von Oracle seit vielen Jahren als Teil aller Datenbankeditionen entwickelt. Sie beinhaltet eine grafische Benutzeroberfläche zur Entwicklung von datenbasierten Webanwendungen und besteht neben der Benutzeroberfläche in der Regel aus zusätzlichen Oracle-Datenbank-Objekten (wie Tabellen, Packages usw.).

Ein häufiges Problem bei der Nutzung von Oracle APEX in größeren Projekten ist der aufwendige, manuelle Prozess, um eine Applikation und alle dazugehörigen Komponenten von einer Umgebung, beispielsweise der Entwicklungsumgebung, auf eine andere Umgebung, wie die Testumgebung, zu übertragen. Hierzu müssen alle Datenbankobjekte, statische Daten sowie die Applikation selbst manuell von der Entwicklungsumgebung auf die Testumgebung und später auf die Produktivumgebung übertragen werden.

Oracle APEX besitzt hierfür seit der Version 20.1 ein eingebautes Feature namens Remote Deployment, das den Deployment-Prozess von einer Umgebung auf eine andere vereinfachen soll. Dennoch besteht der Remote-Deployment-Prozess aus vielen einzelnen Schritten, auf die bereits während der Entwicklung geachtet werden muss. So müssen beispielsweise Skripte zur Aktualisierung des Datenbankschemas manuell geschrieben und in der APEX-Applikation als Update-Skripte hinterlegt werden. Der Prozess des eigentlichen Deployments ist hier ebenfalls ein händischer Prozess. In der APEX-Entwicklungsoberfläche muss eine APEX-Applikation mit manuellen Klicks durch mehrere Masken auf der Zielumgebung installiert werden (vgl. *[1]*). Hierbei können sich durch die manuelle Durchführung und Komplexität der Weboberfläche immer wieder Fehler einschleichen.

Diese und weitere Probleme werden in der modernen Softwareentwicklung typischerweise anhand einer Automatisierung durch CI/CD gelöst. CI/CD beschreibt die Automatisierung des Tests, der Auslieferung und der Überwachung von Software. Der Prozessablauf hilft dabei, die Änderungen der Entwickler in kürzeren Intervallen auf den Haupt-Entwicklungszweig zu bringen. Darüber hinaus sorgt CI/CD mit automatischen Tests und Deployments für eine bessere Qualität sowie häufigere Releases einer Software *(siehe Abbildung 1)*.

Im weiteren Verlauf dieses Artikels zeigen wir auf, wie wir es geschafft haben, eine CI/CD-Pipeline für APEX-Applikationen aufzubauen, um diese erfolgreich bei Kunden einzusetzen.

Tools

Zur Umsetzung des automatisierten Deployments sind einige Tools erforderlich, die zunächst kurz vorgestellt werden, bevor der Ansatz erläutert wird.

Git

Git ist ein verteiltes Versionskontrollsystem, das von der Open-Source-Community vorangetrieben wird. Heutzutage ist Git das am meisten eingesetzte Tool für Versionskontrolle in der Softwareentwicklung und verfügt dabei über ein Command-Line-Interface – ist aber auch mit einer grafischen Benutzeroberfläche in vielen gängigen Entwicklungsumgebungen integriert. Für jede Entwicklungsaufgabe oder jedes Feature wird in Git ein neuer Entwicklungszweig (Branch) erstellt und nach Abschluss der Entwicklung einer Aufgabe (z.B. Bug oder Feature) wird dieser Branch zurück in den Master-Branch gemerged *(siehe Abbildung 2)*.

Eine Best-Practice-Lösung im Umgang mit Git und APEX ist, einen Branch pro Umgebung zu haben. Häufig wird dabei mit drei Umgebungen gearbeitet: einer Entwicklungsumgebung (Dev), einer Testumgebung und natürlich einem Produktiv-System. Die Entwicklungen werden dabei zunächst im Dev-Branch abgelegt. Sobald die Testumgebung aktualisiert werden soll, wird der Dev-Branch in den Test-Branch gemergt. Gleiches gilt für den Master-Branch, der den aktuellen Stand der Produktivumgebung widerspiegelt.

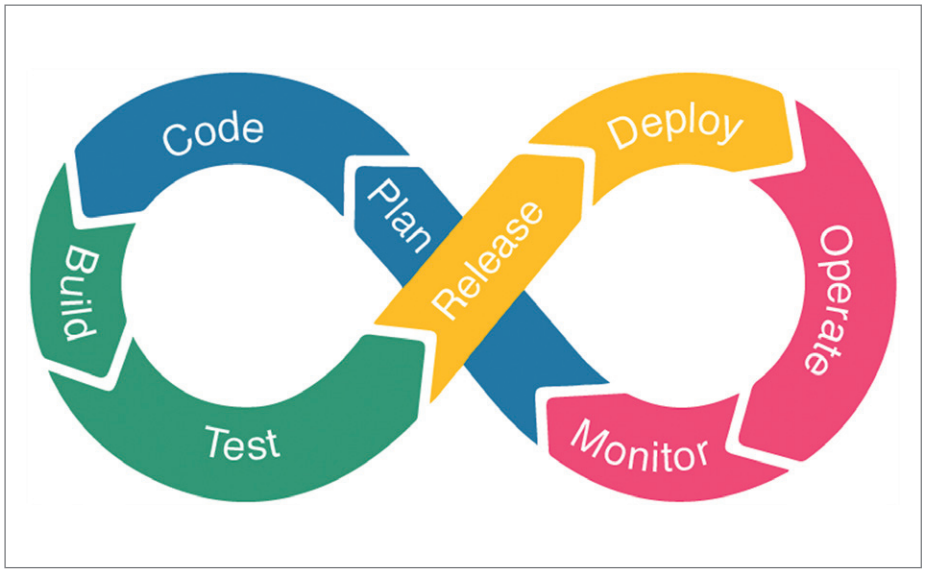

Abbildung 1: Veranschaulichung des Dev-Ops Lifecycle (Quelle: Yıldırım, Abdurrahim, 2019: DevOps Lifecycle: Continuous Integration and Development [2])

Zur Verwaltung von Projekten und Git-Repositories gibt es DevOps-Plattformen wie GitLab. Unser hier vorgestellter Ansatz würde auch auf anderen Plattformen wie GitHub oder Oracle Visual Builder Studio funktionieren. In unserem Fall konzentrieren wir uns allerdings auf GitLab. In GitLab lassen sich verschiedene Projekte anlegen, die als Git-Repository fungieren. Zusätzlich gibt es in GitLab die Möglichkeit, CI/CD Pipelines als Teil eines Projekts zu erstellen. Die Pipeline ist ebenfalls Teil des Quellcodes und kann somit von den Vorteilen der Versionskontrolle profitieren. Eine Pipeline kann dann beispielsweise bei jedem Push, also dem Laden der Änderungen, auf einen Branch ausgeführt werden.

Liquibase

Liquibase ist eine Software, die sich um Versionsmanagement von Datenbankschemata kümmert. Änderungen an Datenbankschemata werden in sogenannten Change-Sets auf XML-Basis dargestellt und können durch Liquibase auf Datenbanken durchgeführt werden. Liquibase erleichtert durch diese Technologie die Migration von Änderungen an Datenbankschemata zwischen verschiedenen Datenbanken. Das Übertragen von Änderungen von einer Datenbankumgebung auf eine andere Umgebung wird somit deutlich erleichtert und ist durch die Automatisierung weniger fehleranfällig.

SQLcl

Um den Code und die APEX-Applikation aus der Datenbank zu extrahieren, wird SQLcl verwendet – der Nachfolger von SQLPlus ist ein Command-Line-Interface für PL/SQL Code. Der Vorteil von SQLcl ist, dass Liquibase nativ mitgeliefert wird. Mithilfe von Liquibase können Datenbankobjekte jeglicher Art sowie APEX-Applikationen aus einer Datenbank extrahiert und in einer anderen Datenbank eingespielt werden. Dabei speichert Liquibase das Datenbankschema als XML-Dateien, die in unser Git-Repository eingecheckt werden können. Eine XML-Datei besteht dabei aus Change-Sets, wobei ein Change-Set eine Änderung an einem Datenbankschema beschreibt.

Als Besonderheit der in SQLcl integrierten Liquibase-Version sind zusätzliche Features im Umgang mit den Spezifika einer Oracle-Datenbank hervorzuheben *(siehe [3])*. Die von Liquibase generierten XML-Dateien sind Oracle-spezifisch und können dadurch einfacher mit Paketen, Jobs oder anderen Oracle-Datenbankobjekten umgehen.

Liquibase speichert in der Zieldatenbank Change-Sets, die bereits angewendet wurden, und kann so automatisiert Änderungen eines Schemas von einer Datenbank auf eine andere übertragen. Mit SQLcl kann darüber hinaus auch die APEX-Applikation extrahiert werden. Die daraus resultierenden SQL-Dateien können ebenfalls im Git-Repository eingecheckt werden. SQLcl kann mittels dieser SQL-Dateien die APEX-Applikation wieder auf einer Datenbank installieren.

SQLcl wird ständig weiterentwickelt und kürzlich wurde die Version 22.3 veröffentlicht. Unsere Code-Beispiele beziehen sich allerdings auf die Version 22.2, da Version 22.3 eine überarbeitete Syntax besitzt, die aber aufgrund einer veränderten Herangehensweise bei der Verwaltung der generierten Dateien für uns nicht infrage gekommen ist. Dadurch sind wir – trotz diverser Show Stopping Bugs – bei der Version 22.2 geblieben.

Methode

Wie oben im Git-Überblick bereits gezeigt, arbeitet unsere Methode mit drei Branches:

- Dev: Beinhaltet den aktuellen Stand der Entwicklungsumgebung. Hier wird nach einem Export der aktuelle Stand eingecheckt.
- Test: Beinhaltet den Stand der Testumgebung. Wann immer neue Entwicklungen auf die Testumgebung installiert werden, muss der Dev-Branch in den Test-Branch gemergt werden.
- Master: Beinhaltet den Stand der Produktion. Der Test-Branch wird bei einem neuen Release in den Prod-Branch gemergt.

Abbildung 3 stellt den Prozess dar, der durchgeführt werden muss, um eine APEX-Applikation von der Entwicklungsumgebung auf die Testumgebung zu übertragen. Hierbei müssen im ersten Schritt das Datenbankschema und die APEX-Applikation exportiert werden. Um den Export zu erleichtern, haben wir ein Skript entwickelt, das alle notwendigen Datenbankobjekte, statische Daten sowie die APEX-Applikation selbst, automatisch exportiert.

Bevor wir mit dem Export beginnen, muss zunächst die Verbindung zur Datenbank aufgebaut werden. Dafür wird erst das Datenbankschema exportiert, das – im Voraus mithilfe von durch Namenskonvention festgelegten Präfixen – gefiltert wird, sodass nur die relevanten Objekte exportiert werden (und dadurch z. B. verschiedene Teil-Applikationen getrennt abgelegt werden können). Dies

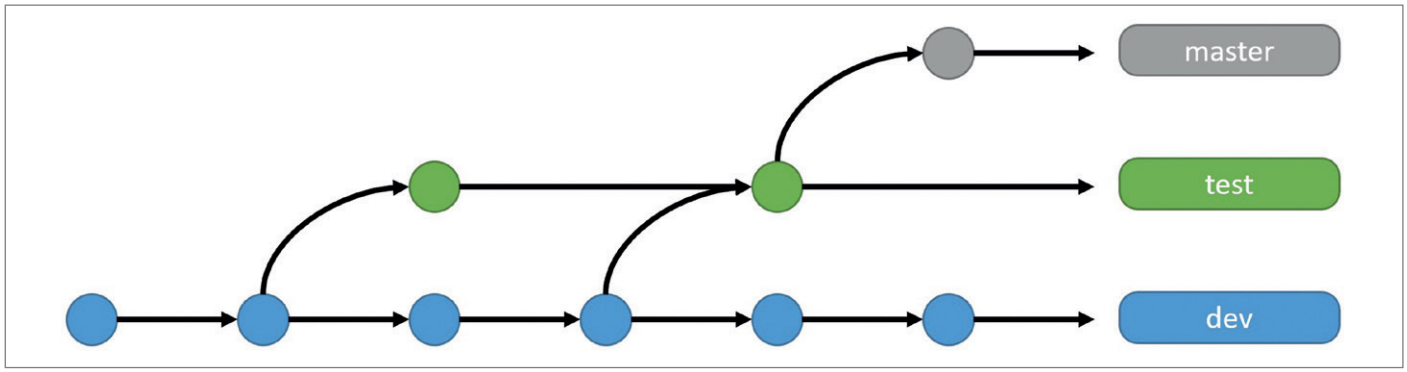

Abbildung 2: Überblick der Git-Branches für unsere APEX-Applikationen (Quelle: PROMATIS)

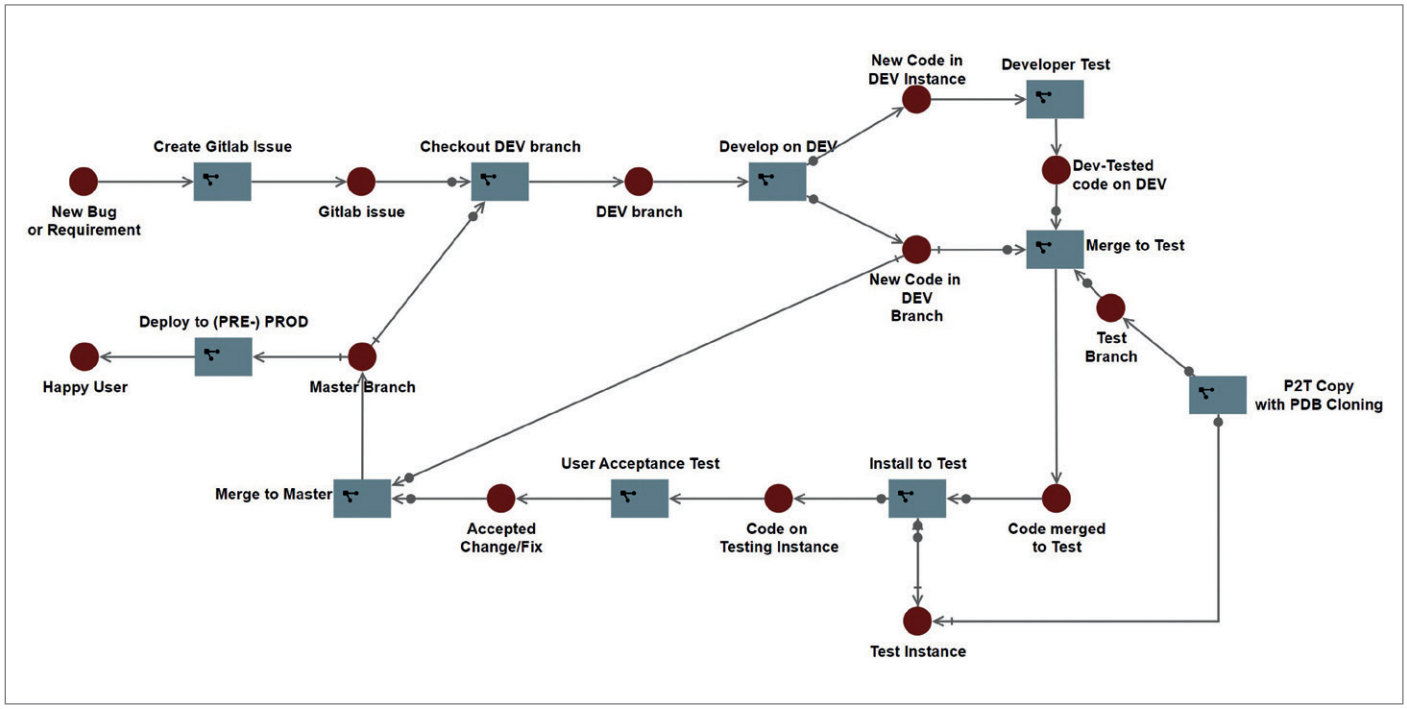

Abbildung 3: Überblick der Methode für automatisiertes Deployment für APEX-Applikationen (Quelle: PROMATIS)

wird mit dem Ausführen eines einzigen Befehls erreicht. Anschließend werden die statischen APEX-Daten mittels Spooling in CSV-Dateien exportiert. Dabei wird für jede Tabelle eine eigene CSV-Datei angelegt. Zuletzt wird nun die APEX-Applikation anhand des Standard-APEX-Export-Befehls exportiert *(siehe Listing 1)*. Wir setzen jedoch bewusst nicht auf die SQLcl-Liquibase-APEX-Integration, da diese für unseren Anwendungsfall zu viele Probleme verursacht hat. Um Komplikationen mit dem Überschreiben von Datenbank-Sequenzen in den Zieldatenbanken vorzubeugen, werden alle Änderungen an Sequenzen nach dem Export rückgängig gemacht.

Anschließend müssen die Änderungen in Git eingecheckt werden. Diesbezüglich ist zu beachten, dass vor dem Export der richtige Branch ausgecheckt wurde – in unserem Beispiel ist dies der DEV-Branch. In der vorliegenden Konfiguration ist es für jeden Nutzer möglich, direkt auf den DEV-Branch zu pushen; TEST- und PROD-Branch können nur via Merge Request aktualisiert werden. Dies ist so konfiguriert, um zusätzliche Sicherheit bei der Aktualisierung der Test- und Prod-Umgebung zu erreichen *(siehe Listing 2)*.

Mit den auf den Dev-Branch in Git eingecheckten und gepushten Änderungen kann in GitLab ein Merge Request vom Dev-Branch auf den Test-Branch erstellt und gemergt werden. Wir haben in unserem GitLab-Projekt die CI/CD-Pipeline so konfiguriert, dass bei einer Änderung auf dem Test-Branch automatisch diese auf die Test-Instanz ausrollt *(siehe Abbildung 4)*. Das Ausrollen der Änderungen auf die Prod-Instanz für den Prod-Branch wird hingegen (aus Sicherheitsgründen noch) nicht vollautomatisch ausgeführt, sondern muss „per Knopfdruck" über die GitLab-Weboberfläche gestartet werden.

Nachdem der Merge durchgeführt worden ist, führt unsere CI/CD-Pipeline die Installation durch. Nun werden automatisch die folgenden Schritte aus *Listing 3* in einer Kommandozeile durchgeführt.

Nachdem die CI/CD-Pipeline erfolgreich durchlaufen ist, sind alle Datenbankobjekte, Setup-Daten und die APEX-Applikation auf der Testumgebung installiert *(siehe Abbildung 5)*. Die Testumgebung ist damit auf dem gleichen Stand wie die Entwicklungsumgebung. Einen kleinen Unterschied gibt es dennoch: Liquibase erstellt auf der Zielumgebung bei der ersten Ausführung drei weitere Tabellen.

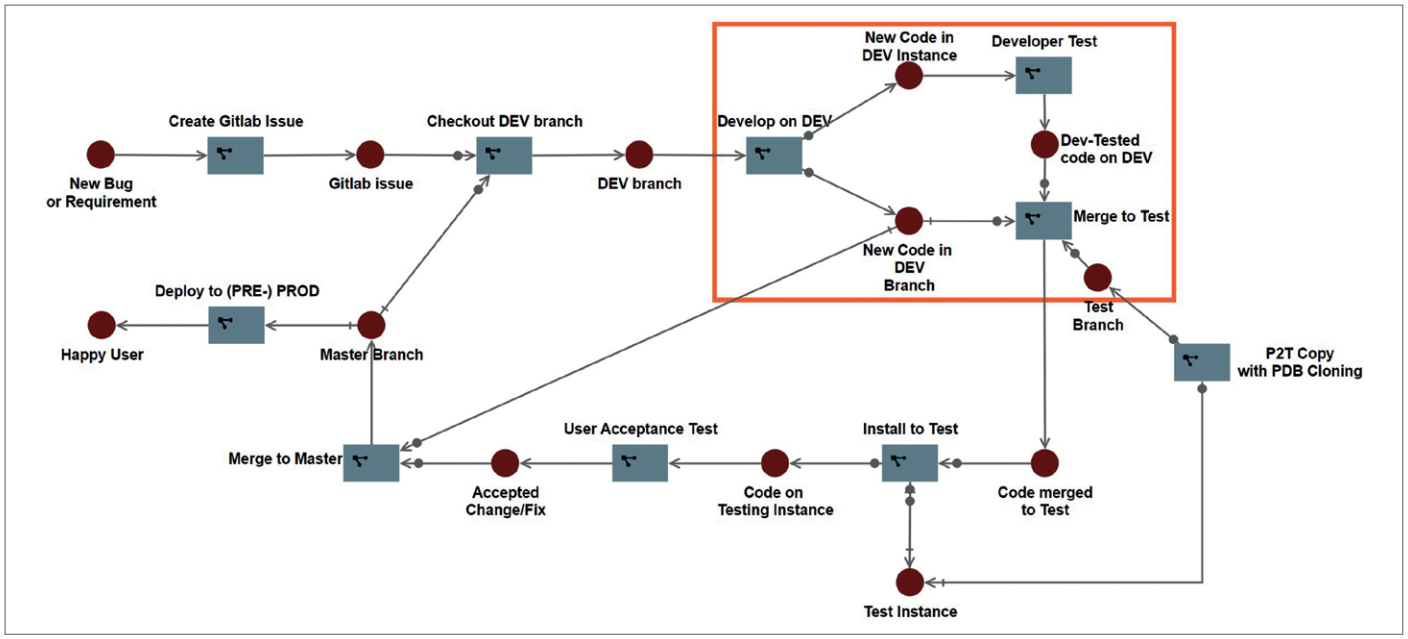

Abbildung 4: Überblick der Methode für automatisiertes Deployment für APEX-Applikationen mit der Markierung des aktuellen Prozessschritts.(Quelle: PROMATIS)

```
$ cd <pfad_zu_git>
$ sql user/password@//host:port/service
sql> cd database/mul;
sql> lb genschema -split -filter "LIKE 'MUL_%'";
sql> cd ../..;
sql> set sqlformat csv;
sql> set feedback off;
sql> alter session set nls_date_format = 'YYYY-MM-DD';
sql> spool 'data/mul/<tabelle>.csv';
sql> select * from <tabelle>;
sql> spool off;
sql> alter session set nls_date_format = 'DD.MM.RR';
sql> set sqlformat ansiconsole;
sql> set feedback on;
sql> cd apex;
sql> apex export -workspaceId <workspace_id> -applicationId <app_id>
-skipexportdate -exportoriginalids -split;
sql> cd ..;
sql> exit;
$ git checkout HEAD -- ./database/mul/sequence/
```

Listing 1: Befehle zum Export aus der Entwicklungsumgebung

```
$ cd <pfad_zu_git>
$ git add .
$ git commit -m "<commit_nachricht>"
$ git push
```

Listing 2: Befehle zum Einchecken der Änderungen in das Git-Repository

```
$ cd <pfad_zu_git>
$ sql user/password@//host:port/service
sql> cd database
sql> lb update -changelog controller.xml
sql> cd ../data/<tabelle>
sql> lb update -changelog data.xml
sql> cd ../../apex/<applikation>
sql> @install.sql
```

Listing 3: Befehle zur Installation auf der Testumgebung

Anhand dieser Tabellen merkt sich Liquibase, welche Änderungen (Change-Sets) bereits angewandt worden sind, sodass diese nicht erneut angewendet werden.

Fazit

Deployments von APEX-Applikationen lassen sich in nur wenigen Schritten fast vollständig automatisieren und sorgen so dafür, dass die Deployments einfacher und in kürzeren Zyklen stattfinden können. Durch die Automatisierung des gesamten Deployments lassen sich Fehler minimieren und somit positive Auswirkungen auf die Qualität der Software ermöglichen. Oracle hat für die Zukunft weitere Verbesserungen hinsichtlich Built-in CI/CD angekündigt *(siehe [4])*.

Darüber hinaus kann die CI/CD-Pipeline um automatisiertes Testen erweitert werden. Denkbar wäre hier beispielsweise das Ende-zu-Ende-Testen der APEX-Applikation mittels Selenium. So könnte die Qualität der Software weiter verbessert werden. Des Weiteren wird durch das automatisierte Testen sichergestellt, dass durchgeführte Änderungen keine Fehler in der Software verursachen.

Quellen

[1] Chaitanya Koratamaddi, 2020: Remote Deployment of your APEX App is just

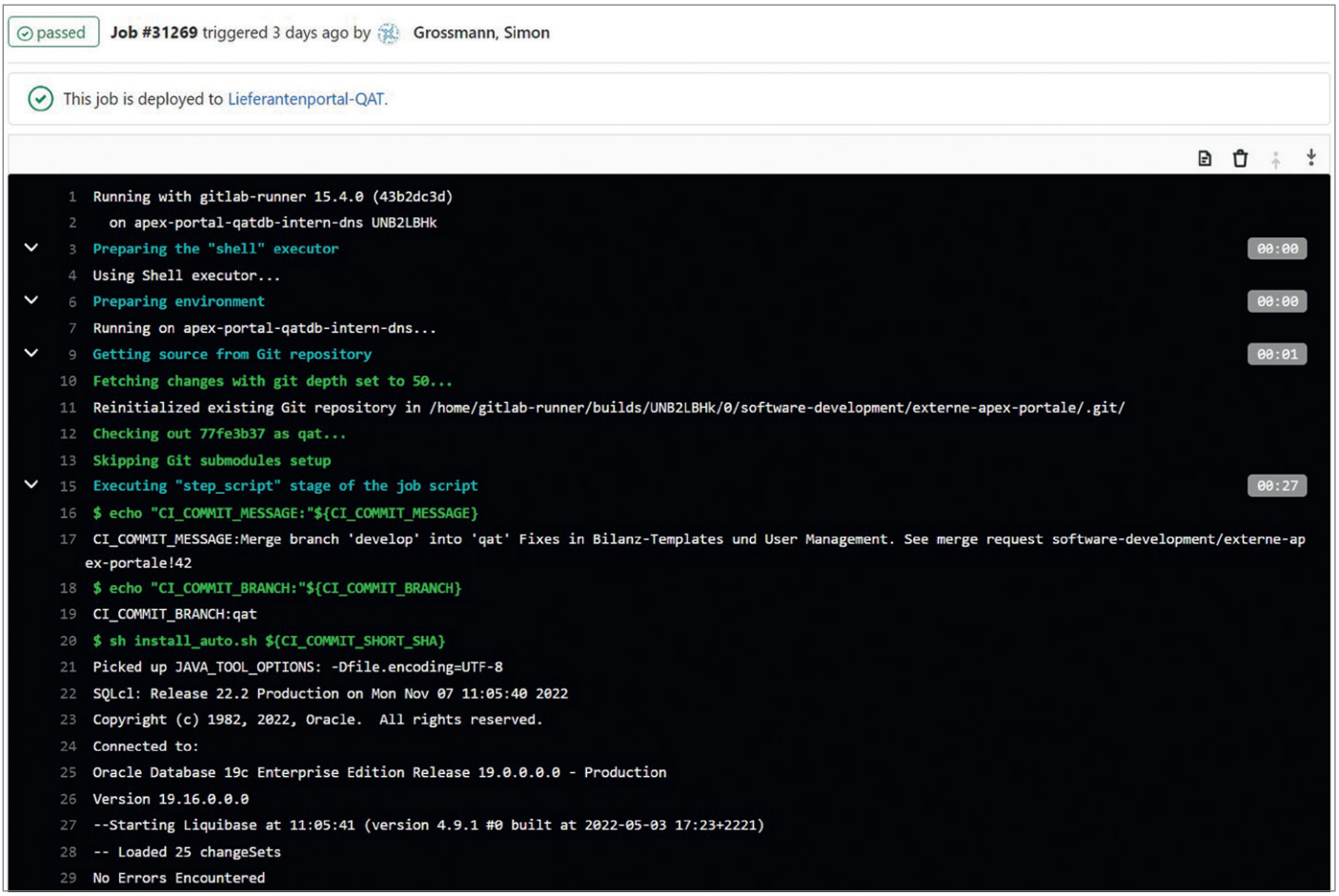

Abbildung 5: GitLab Job Log eines Deployments auf die Testumgebung. (Quelle: PROMATIS)

One Click Away! *https://blogs.oracle.com/apex/post/remote-deployment-of-your-apex-app-is-just-one-click-away*

[2] Abdurrahim Yıldırım, 2019: DevOps Lifecycle: Continuous Integration and Development *https://medium.com/t%C3%BCrk-telekom-bulut-teknolojileri/devops-lifecycle-continuous-integration-and-development-e7851a9c059d*

[3] Jeff Smith, 2022: Oracle Change Management with SQLcl and Liquibase *https://speakerdeck.com/thatjeffsmith/oracle-change-management-with-sqlcl-and-liquibase*

[4] Oracle: Oracle APEX Roadmap *https://apex.oracle.com/en/learn/resources/roadmap/*

Über die Autoren

Johannes Michler ist Senior Principal Consultant, Systemarchitekt und Projektleiter für die Promatis-Gruppe mit Fokus auf serviceorientierte Architekturen (SOA), Web-Portale (insbesondere mit ADF und APEX) sowie Prozessautomatisierung. Als Mitglied im Management Board bekleidet er die Funktion „Executive Vice President – Head of Platforms & Development“ und ist seit 2010 für die DOAG als Referent und Autor mit wissenschaftlichen und anwendungsnahen Beiträgen aktiv. Er nimmt als Referent zahlreiche Veranstaltungen der Oracle Community (IOUG & OATUG) wahr und ist als „ACE Director“ Teil der Oracle-ACE-Community.

Simon Grossmann ist seit 2015 bei der Promatis-Gruppe und besitzt als technischer Consultant mehrjährige Erfahrung in der Java-Anwendungsentwicklung und der Realisierung prozessorientierter Informationssysteme mit Oracle-Komponenten sowie fundierte Kenntnisse im Einsatz moderner Web-Technologien (APEX, JEE) in Verbindung mit leistungsfähigen Software-Entwicklungsumgebungen, Application-Servern und Service-orientierten Architekturen (SOA).

Johannes Michler
johannes.michler@promatis.de

Simon Grossmann
simon.grossmann@promatis.de