Interactive Grid oder für die JET Charts. Es lohnt sich, das Release auf apex.oracle.com genauer anzusehen oder es für eine eigene Installation herunterzuladen.

Zu allen hier vorgestellten neuen Features sind detaillierte Blog Postings auf dem Oracle Application Express Blog (siehe unter „Weitere Informationen") verfügbar.

Weitere Informationen

- Informationen zu Application Express und Demo-Umgebung *http://apex.oracle.com*
- Oracle Application Express Blog – mit vielen Postings zu APEX 19.1 *http://blogs.oracle.com/apex*

Carsten Czarski
carsten.czarski@oracle.com

Schnellstart – Versionskontrolle für existierende Oracle-(APEX-)Projekte

Ottmar Gobrecht, Linde

Viele Oracle-(APEX-)Projekte setzen bis heute keine Versionskontrolle ein. Die Gründe dafür sind vielfältig. Meist geht man wohl davon aus, dass die Datenbank ein sicherer Ort für den Quellcode ist. Mit einem funktionierenden Backup ist das auch richtig, man verliert aber auf jeden Fall die komplette Historie der Änderungen. Oft fehlt in laufenden Projekten auch einfach die Zeit, sich zusätzlich noch mit der Einführung einer Quellcode-Versionierung zu beschäftigen, weil auf den ersten Blick kein direkter Nutzen zu sehen ist. Wer wagt da unter Zeitdruck den zweiten Blick? Dieser Artikel will die Hürde für die Einführung einer Versionsverwaltung ein wenig tiefer legen.

Einleitung

Der erste Schritt ist der wichtigste, das wissen alle, die sich schon einmal nach langem Zögern endlich auf den Weg gemacht haben – worum auch immer es ging. Das Gleiche gilt für die Quellcode-Versionierung. Wenn man erst einmal auf dem Weg ist, fragt man sich, wie man so lange ohne auskommen konnte. Eine versionierte Quellcode-Basis vergrößert die Komfortzone bei der Entwicklung und die investierte Zeit zahlt sich mehrfach wieder aus. Doch wie geht man möglichst einfach und schnell den ersten Schritt? Wie gestaltet man die Struktur eines Repository?

Workflow-Änderung: dateibasiertes Arbeiten

Eine der wichtigsten Fragen lautet: Kann ich weiterarbeiten wie bisher? Die Antwort hängt ganz davon ab, wie man bisher gearbeitet hat. Grundsätzlich bedeutet das Arbeiten mit einer Quellcode-Versionierung auch einen Wechsel weg vom Entwickeln in der Datenbank. Das direkte Ändern und Kompilieren von Code in der Datenbank oder das Ändern von Tabellen mit dem Tool seiner Wahl muss sich hin zu einem dateibasierten Arbeiten entwickeln. Das bedeutet, mit einer Datei im lokalen Quellcode-Repository mit dem Tool der Wahl zu arbeiten. Die Veränderungen werden dann aus der Datei heraus kompiliert. Für Schema-Änderungen müssen Skripte den notwendigen DDL-Code enthalten. Je nach bisher verwendeten Tools ist das beim Bearbeiten von Logik eher nur ein subtiler, beim Verändern von Datenbankobjekten wie Tabellen dann doch schon ein großer Unterschied. Wurde bisher ein Schema-Diff-Tool verwendet, um

Änderungen der Entwicklungsumgebung in die Produktion zu transportieren, dann sieht das vielleicht auf den ersten Blick sogar wie ein Rückschritt aus.

Der Punkt ist hier, dass man nur dann eine verwertbare Quellcode-Historie bekommt, wenn man konsequent auf das dateibasierte Arbeiten setzt. Es kann auch einen Zwischenschritt hin zu diesem Ansatz geben. Man arbeitet weiter wie bisher und exportiert regelmäßig alle Objekte des Schemas in die Quellcode-Verwaltung. Auf diesem Weg erhält man wenigstens eine gewisse geordnete Transparenz der Änderungen. Werden dann im Laufe des Projektes die Vorteile dieses Vorgehens ersichtlich, fehlt einem meist noch die Information, wer denn die jeweiligen Änderungen ausgeführt hat. Außerdem werden durch die gewonnene Transparenz schnell Wünsche geweckt wie „Wenn ich den alten Code im Repository habe, dann darf ich ja mal etwas probieren, ich kann ja notfalls zurück". Schon ist die erste Hürde genommen und die Offenheit, vielleicht doch einmal richtig damit loszulegen, da.

Dass so etwas noch viel besser funktionieren kann, darauf kommen die meisten Entwickler dann selbst. Zum Beispiel das Anlegen von einem Branch (Entwicklungszweig) im lokalen Repository. Damit kann man etwas auszuprobieren, ohne Gefahr zu laufen, aufgrund eines unerwartet zu lösenden Bugs die gemachten Änderungen wieder verwerfen zu müssen. Man wechselt bequem in den (parallelen) Hauptzweig, arbeitet am Bugfix, aktualisiert seinen Branch danach auf den geänderten Hauptzweig und kann dann einfach weiter ausprobieren. Das geschieht alles ohne die Gefahr, Dinge zu verlieren oder andere Kollegen zu beeinflussen. Habe ich jetzt irgendjemanden abgehängt? Kein Problem, einfach loslegen – hier hilft nur Praxis.

Toolvergleich: DDL-Export

Wie bewerkstelligt man den ersten Schritt? Wie exportiert man die Schema-Objekte seiner Anwendung in eine wohlsortierte Verzeichnisstruktur? Die meisten der verwendeten grafischen Entwicklungstools bringen dafür eine Export-Funktion mit. Diese ist je nach Tool mehr oder weniger gut dafür geeignet, ein Quellcode-Repository aufzubauen. Deshalb schauen wir uns an, wie die am häufigsten verwendeten Tools SQL-Developer, PL/SQL-Developer und Toad das können. Ein Wort vorweg: Die Intention des Schema-Exports dieser Tools ist eigentlich, die Objekte in ein anderes Schema, vielleicht sogar in eine andere Datenbank zu bringen, und nicht, ein Quellcode-Repository zu erstellen. Folgende Fragen bewegen uns, wenn es darum geht, ein solches aufzubauen:

- Ist eine Skript-Datei pro Objekt möglich?

- Sind Unterverzeichnisse pro Objekttyp möglich?
- Sind eigene Dateien für Fremdschlüssel möglich? (Nützlich für einfachere Master-Skripte)
- Können „Object already exists“-Fehler verhindert werden? (Oder anders gefragt: Sind die Skripte wiederanlauffähig?)
- Können Daten exportiert werden? (Möglichst im CSV-Format zur Verfolgung von Stammdatenänderungen)
- Ist eine APEX App exportierbar? (Womöglich auch zerlegt in die Einzelteile wie Pages, Shared Components usw.)

Kriterium	SQL-Dev.	PL/SQL-Dev.	Toad
Datei pro Objekt	Ja	Ja	Ja
Unterverz. pro Typ	Ja	Nein	Ja
FK Constr. extra	Ja	Nein	Ja
Verhi. „object exist”	Nein	Nein	Nein
Export Daten	Ja	Nein	Jein
Export APEX App	Jein	Nein	Nein

Abbildung 1: Toolvergleich DDL-Export (Quelle: Ottmar Gobrecht)

Abbildung 1 zeigt das Ergebnis bezogen auf die Fragestellung.

Anmerkungen zum SQL-Developer:

- Ist am übersichtlichsten
- Viele Formate für Datenexport (auch CSV)
- Umfangreich konfigurierbar
- Export APEX App nur mit Command-Line-Version SQLcl

Anmerkungen zum PL/SQL-Developer:

- Wenig konfigurierbar
- Enttäuscht für den Aufbau eines Quellcode-Repository

Anmerkungen zu Toad:

- Zwei Exportmöglichkeiten (mindestens)
  - Entweder Unterverzeichnisse pro Objekttyp…
  - … oder Daten
- Daten nur als Insert Statements
- Umfangreich konfigurierbar, relativ unübersichtlich

Keines der Tools liefert uns ein fertiges, gut strukturiertes Quellcode-Repository. Man muss mehr oder weniger viel nacharbeiten. Das heißt nicht, dass es nicht gehen würde, zumal das Einsortieren der Objekte in eine vernünftige Struktur eine einmalige Arbeit sein sollte. Lediglich beim Export der APEX-Anwendung wünscht man sich eine automatische Lösung, weil dieser Export in Zukunft häufiger gemacht werden muss. Das liegt in der Natur von APEX als deklarativem „Low Code Framework“.

Wir entwickeln die APEX-Anwendung im Browser und die Meta-Daten der Anwendung liegen im APEX-Repository der Datenbank. Ein Export der Anwendung kann auch mit dem Browser gemacht werden, dabei erhält man jedoch nur eine einzige große SQL-Datei mit der gesamten Applikation. Wünschenswert für ein Quellcode-Repository wäre ein Export der Einzelteile wie zum Beispiel Pages, Shared Components, Plug-ins usw. Damit lässt sich dann für die Anwendung verfolgen, wie sie sich entwickelt hat, und auch eine Suche im Repository etwa nach einem bestimmten Package-Aufruf ergibt bei einer zerlegten APEX-Anwendung wesentlich mehr Sinn. In der Vergangenheit konnte man dafür den APEX-Export-Splitter verwenden, im Grunde eine Java-Klasse, die im Lieferumfang jeder APEX-Version bis heute enthalten ist. Der Nachteil ist, dass man die erstellte Verzeichnisstruktur so nehmen muss, wie sie vom Splitter erstellt wurde, oder man verändert sie im Nachgang mit lokalen Skripten auf dem PC. Schöner wäre allerdings, die Verzeichnisstruktur schon während des Exports anpassen zu können, um alle Master-Skripte in einem zentralen Ordner des Repository speichern zu können.

Seit APEX Version 5.1.4 gibt es zu diesem Zweck das APEX_EXPORT Package. Mit diesem kann man entweder das große Gesamtskript oder eben die zerlegte Variante exportieren. Das Rückgabefor-

```
WITH
   FUNCTION backapp RETURN BLOB IS
   BEGIN
       RETURN plex.to_zip(plex.backapp(
           p_app_id                  => 100,
               /* If null, we simply skip the APEX app export. */
           p_include_object_ddl      => true,
               /* If true, include DDL of current user/schema and
               all its objects. */
           p_include_templates       => true,
               /* If true, include templates for README.md, export
               and install scripts. */
           p_include_runtime_log     => true,
               /* If true, generate file plex_backapp_log.md with
               runtime statistics. */
           p_include_data            => true,
               /* If true, include CSV data of each table. */
           p_data_max_rows           => 1000,
               /* Maximum number of rows per table. */
           p_data_table_name_like => ,DEMO_PRODUCT_INFO,DEMO_STATES‘
               /* A comma separated list of like expressions to filter
               the tables. Example: 'EMP%,DEPT%' will be translated to
               "where ... and (table_name like 'EMP%' escape '\' or
               table_name like 'DEPT%' escape '\')". */
       ));
   END backapp;

SELECT backapp FROM dual;
```

Listing 1: Möglicher Erst-Export

mat ist in beiden Fällen eine Collection – jede Zeile der Collection beinhaltet eine Pfadangabe für die jeweilige Exportdatei und ein CLOB-Feld mit dem Inhalt. Man könnte also in einem Export-Skript diese Collection bearbeiten und die Pfadangaben an die eigenen Bedürfnisse anpassen. Genau das macht das vom Autor des Artikels als Open Source veröffentlichte Tool-Package PLEX – der Name steht für PL/SQL Export Utilities. Des Weiteren kann PLEX jede der genannten Fragestellungen zum DDL-Export mit Ja beantworten – kein Wunder, es ist dafür entwickelt worden.

Schnellstart: Package PLEX

PLEX hat zum Ziel, die erste Version des Repository mit einer einfachen Query als BLOB selektieren zu können. Nach Speichern des BLOB in das lokale Verzeichnissystem auf dem PC mit der Endung „.zip" kann man es entpacken und findet in der generierten Verzeichnisstruktur dann auch Skript-Beispiele für zukünftige halb- oder vollautomatische Exporte und Deployments per Shell-Befehl beziehungsweise SQL*Plus. Ein möglicher Erst-Export könnte so aussehen wie in *Listing 1* dargestellt.

Da PLEX boolesche Parameter besitzt, benutzen wir eine Inline-Funktion in der With-Klausel. Wer eine Datenbankversion kleiner als 12c verwendet, erstellt sich eine Hilfsfunktion analog dem Beispiel. PLEX hat noch einige Parameter mehr, um zum Beispiel den APEX App Export zu konfigurieren. Weitere Details sind auf der offiziellen Projektseite *[1]* zu finden. Je nach Größe des Schemas und der APEX App kann dieser erste Aufruf zwischen wenigen Sekunden und mehreren Minuten dauern. Das liegt daran, dass im Hintergrund für jedes Objekt das Package DBMS_METADATA bemüht wird, um das DDL des Objektes zu generieren. Wer an Laufzeit-Infos interessiert ist, findet im Hauptverzeichnis des Exports eine Logdatei mit Informationen darüber, was PLEX so alles gemacht hat und wie lange die jeweiligen Schritte gedauert haben. PLEX selbst existiert jetzt in einer ersten Version – Verbesserungsvorschläge oder Fehlermeldungen sind willkommen und können über die Projektseite als Issue gemeldet werden.

```
BEGIN
   FOR i IN (
       SELECT 'DEMO_STATES' AS object_name
              FROM dual
           MINUS
       SELECT object_name
              FROM user_objects
   ) LOOP
       EXECUTE IMMEDIATE q'[
----------------------------------------
CREATE TABLE "DEMO_STATES" (
   "ST"           VARCHAR2(30),
   "STATE_NAME"   VARCHAR2(30)
)
----------------------------------------
       ]';
   END LOOP;
END;
/

BEGIN
   FOR i IN (
       SELECT ,STATE_DESCRIPTION‘ AS column_name
              FROM dual
           MINUS
       SELECT column_name
       FROM user_tab_columns
           WHERE table_name = ,DEMO_STATES‘
               AND column_name = ,STATE_DESCRIPTION‘
   ) LOOP
   EXECUTE IMMEDIATE q‘[
----------------------------------------
ALTER TABLE demo_states ADD (
   state_description VARCHAR2(255)
)
----------------------------------------
       ]‘;
   END LOOP;
END;
/
```

Listing 2: Wiederanlauffähiges Skript

Ab hier hängt es stark von den Bedürfnissen des jeweiligen Projektes ab, ob und wie häufig man einen DDL-Export ausführt. Der Normalweg sollte ein regelmäßiges Exportieren der APEX-Anwendung sein. Alles andere sollte ab jetzt lokal bearbeitet werden und es infolgedessen kein Bedürfnis für einen Export geben. Wie schon eingangs erwähnt, mag es aber auch Situationen geben, in denen man regelmäßig alle Schema-Objekte exportiert, um die Anwendung zu dokumentieren. Dies sollte allerdings nur als Zwischenschritt hin zu einer dateibasierten Arbeitsweise verstanden werden.

Einen Sonderfall stellt generierter Code dar: Also zum Beispiel das Nutzen von Quick-SQL in APEX oder das Generieren von Tabellen-APIs. Hier kann man überlegen, den generierten Code mit entsprechend konfigurierten Objekt-Filtern zu exportieren. Bei APIs sollte man darüber nachdenken, ob man nicht lieber den Generator anstelle des generierten Codes versioniert – das hängt jedoch ganz davon ab, ob der generierte Code noch manuell verändert wird oder nicht. Jeder manuell erstellte Code sollte versioniert werden.

Geschwindigkeit: Immer Skripte

Wie man schon an den von PLEX mitgelieferten Skript-Beispielen erkennen kann, sollte es das Ziel sein, jedwedes Deployment per Skript auszuführen. Dazu gehört dann auch wieder die Überlegung, das Objekt-DDL nur beim allerersten Mal zu exportieren – denn was passiert zum

Beispiel beim Export von Tabellen-DDL? Hat man eine weitere Spalte per Alter-Table-Anweisung in das Tabellenskript geschrieben und dafür gesorgt, dass es wiederanlauffähig ist *(siehe Listing 2)*, dann überschreibt ein weiterer DDL-Export die Alter-Anweisung und unser Tabellenskript enthält einfach die neue Spalte, gelistet in der Create-Table-Anweisung. Diese wird jedoch aufgrund der Wiederanlauffähigkeit niemals mehr ausgeführt, da unsere Tabelle ja schon existiert.

Somit haben wir jetzt eine Quellcode-Datei, die für das Deployment nicht mehr geeignet ist. Schade eigentlich, denn bis eben hätten wir mit unserem Master-Skript einfach immer alle Objekt-Skripte aufrufen können und nur die Änderungen (wie z.B. unsere neue Spalte) wären wirklich ausgeführt worden – ein einfaches Deployment und eine übersichtliche Historie im Repository. Demgegenüber steht das automatische Erstellen von Diff-Skripten, die mitunter nicht wiederanlauffähig sind und auch bei etwas komplizierteren Schema-Änderungen die Gefahr eines Datenverlustes beinhalten. Man kommt auch bei sehr teuren Tools dieser Klasse nicht umhin, die generierten Skripte auf Richtigkeit zu überprüfen und manuelle Anpassungen für etwaige Datenmigrationen auszuführen. Die Entscheidung, welchen Weg man geht, nimmt einem niemand ab. Jetzt sind wir mittendrin in der Diskussion darüber, wie man ein Deployment in das Zielsystem durchführt. Da kann dann auch PLEX nicht weiterhelfen – außer, dass das Package per se versucht, alle notwendigen Objekt-DDLs wiederanlauffähig zu extrahieren. Hier sind wir im weiten Feld der individuellen Bedürfnisse eines Projektes angekommen. Die von PLEX gelieferten Skriptbeispiele müssen auf jeden Fall an die Bedürfnisse des jeweiligen Projektes angepasst werden. Der Autor beispielsweise unterscheidet fast immer zwischen Skripten für das Backend und solchen für das Frontend – sowohl für den Export als auch für das Deployment.

Überlegung: Verzeichnisstruktur im Repository

Wie soll man sein Repository strukturieren? Hier ein Vorschlag, der sich im Laufe der Zeit für mich persönlich als vorteilhaft herausgestellt hat. Jedes Projekt mag da andere Vorstellungen haben. Ich verwende hier eine einfache Auflistung – die ersten Wörter jedes Listenpunktes stellen den Verzeichnisnamen dar und etwaige Erklärungen finden sich in Klammern dahinter:

- app_backend (unser Schema DDL)
  - constraints (ein Ordner pro Objekttyp)
  - package_bodies
  - packages
  - ref_constraints
  - sequences
  - tables
  - …
- app_data (Verfolgung von Master-Daten, optional)
- app_frontend (unsere APEX app)
  - pages
  - shared_components
  - …
- docs (die Dokumentation)
- reports
- scripts (alle Master-Skripte vereint)
  - logs (Export- und Installations-Logs)
    - export_app_100_from_MYSCHEMA_at_DEV_20190315_132542
    - install_app_100_into_MYSCHEMA_at_TEST_20190318_083756
    - …
  - misc (häufig genutzte Skripte, etwa zum Rekompilieren des Schemas)
  - 1_export_app_from_DEV.bat (Shellskript mit Parametern für das eigentliche SQL-Exportskript)
  - 2_install_app_into_TEST.bat (Shellskript mit Parametern für das eigentliche SQL-Deploymentskript)
  - 3_install_app_into_PROD.bat
  - export_app_custom_code.sql (Export-Master-Skript mit individuellen Anpassungen)
  - install_app_custom_code.sql (Deployment-Master-Skript mit individuellen Anpassungen je Release)
  - install_backend_generated_by_plex.sql (wird aufgerufen vom Master-Skript)
  - install_frontend_generated_by_apex.sql (wird aufgerufen vom Master-Skript)
- tests (Unit- und Frontend-Tests)
- README.md (generelle Informationen zum Projekt und Links in die Dokumentation)

Das Repository sollte logisch aufgebaut sein und keine extrem tiefen Verzeichnisstrukturen beinhalten, um den Umgang damit zu erleichtern. Alle Master-Skripte sind vereint in einem Skript-Ordner, auch das von APEX generierte Installationsskript für das Frontend. Alle Master-Skripte sollten während der Verwendung die jeweiligen Rückmeldungen in entsprechende Log-Dateien im Unterordner „logs" ablegen – damit sind Exporte von Quellcode und Deployments in Zielsysteme nachvollziehbar.

Ausblick: CI/CD & Schema-Migration

Die skizzierte Diskussion um das Deployment lässt es schon erahnen: Einfach ein Quellcode-Repository aufsetzen und man ist fertig, ist nicht die Lösung. Das war nur der erste Schritt zum Aufwärmen. Denn hat man erst einmal alles wohl strukturiert und per Skript lauffähig gemacht, dann kann man das Deployment auch im zweiten Schritt automatisieren. Der Inhalt dieser Diskussion sprengt allerdings bei Weitem den Rahmen dieses Artikels, der ja den ersten Schritt im Fokus hat. Daher legt der Autor jedem nahe, nach diesem ersten Schritt nicht stehen zu bleiben und weitere Schritte ins Auge zu fassen. Einen sehr guten Grundlagenartikel hat beispielsweise Martin Fowler zum Thema verfasst – er betrachtet jede Änderung an der Datenbank als eine Migration. Zum weiteren Studium seien daher folgende Einstiegspunkte genannt:

- Antti Kirmanen: Git vs. Subversion (SVN): Welches Versionskontrollsystem sollten Sie nutzen? *[2]*
- Martin Fowler: Evolutionary Database Design *[3]*
- Samuel Nitsche:
  - There is no clean (database) development without Version Control *[4]*
  - „One does not simply update a database" – migration based database development *[5]*
- Blain Carter:
  - Tips to help PL/SQL developers get started with CI/CD *[6]*
  - CI/CD for Database Developers – Export Database Objects into Version Control *[7]*
- Jeff Smith: 19.X SQLcl Teaser: LIQUIBASE *[8]*

Quellen

[1] *https://github.com/ogobrecht/plex*
[2] *https://entwickler.de/online/development/git-subversion-svn-versionskontrollsystem-579792227.html*
[3] *https://www.martinfowler.com/articles/evodb.html*
[4] *https://cleandatabase.wordpress.com/2017/09/22/there-is-no-clean-database-development-without-version-control*
[5] *https://cleandatabase.wordpress.com/2017/11/28/one-does-not-simply-update-a-database-migration-based-database-development*
[6] *https://learncodeshare.net/2018/04/30/tips-to-help-pl-sql-developers-get-started-with-ci-cd*
[7] *https://learncodeshare.net/2018/07/16/ci-cd-for-database-developers-export-data-base-objects-into-version-control*
[8] *https://www.thatjeffsmith.com/archive/2019/01/19-x-sqlcl-teaser-liquibase/*

Ottmar Gobrecht
ottmar.gobrecht@linde.com

Ansible oder Puppet: Which way to go?

Raphael Daum (DBConcepts) und Florin-Catalin Enache (T-Systems Austria)

Was verbindet DevOps, Cloud, Serverless Computing und Continous Delivery? Hinter all diesen Begriffen verbirgt sich die Notwendigkeit der Automatisierung, die Notwendigkeit, Abläufe deterministisch und reproduzierbar zu machen. Wir stellen zwei prominente Vertreter dieses Automatisierungsansatzes vor: Der Newcomer Ansible trifft auf Puppet, den Darling der letzten Dekade. Was verbindet sie, was trennt sie und welchen Nutzen bieten sie?

Alt gegen Jung, so könnte ein Vergleich zwischen den Tools Puppet und Ansible zusammengefasst werden. Während der frühere Platzhirsch scheinbar schon mit dabei war, als das Internet gerade anfing, ein Massenphänomen zu werden, betritt mit Ansible ein Werkzeug zu einem Zeitpunkt die Bühne des Geschehens, an dem nur mehr von Cloud und Terraforming die Rede ist. Somit sollte auch schon klar sein, welchem der beiden Konfigurationsmanagement-Tools die Zukunft gehört.

Werkzeugkoffer: Meiner oder deiner?

Damit würden wir es uns allerdings zu einfach machen, denn beide trennt nicht nur zeitlich die eine oder andere Dekade, sondern auch architektonisch unterscheiden sie sich in manchen Bereichen grundlegend. Und somit kann die Frage, ob sich der Einsatz von Puppet oder Ansible eher lohnt, auch nicht beantwortet werden. Obwohl eine Konfigurationsmanagement-Software im Grunde nichts anderes als ein Werkzeugkoffer ist, der einen in die Lage versetzen soll, spannende IT-Dinge zu bauen, und somit eigentlich in den Hintergrund treten soll, ist es dann doch der Werkzeugkoffer der Wahl. Und niemand will, gerade in den Zeiten von „Agile Development“ und „Continous Delivery“, seinen Werkzeugkoffer im Quartalsrhythmus austauschen, denn das verursacht unnötige Kosten und Reibungsverluste. Deshalb sollte die Entscheidung mit Bedacht getroffen und hierbei insbesondere herausgearbeitet werden, welche Probleme die Konfigurationsmanagement-Software lösen soll (und welche nicht gelöst werden können).

Ansible: Speed Transformation

Ansible ist eine Open-Source-Automatisierungsplattform. Mithilfe der einfachen Automatisierungssprache lässt sich eine IT-Anwendungsinfrastruktur in Ansible-Playbooks perfekt beschreiben. Ansible ist auch eine Automatisierungs-Engine zur Ausführung von Ansible-Playbooks *[5]*.

Die jetzt Red Hat gehörende Software kann leistungsfähige Automatisierungsaufgaben verwalten und lässt sich an viele verschiedene Workflows und Umgebungen anpassen. Gleichzeitig kann