dbFlow – mit Git zu CI/CD in der Datenbank-Entwicklung

Maik Michel, Opitz Consulting Deutschland

CI/CD ist ein heiß begehrter Begriff. Doch im Datenbankumfeld ist die Umsetzung einer kontinuierlichen Integration gar nicht so einfach umzusetzen. Wie bekommen wir alle Änderungen für eine Applikation in die nächste Umgebung? Wie stelle ich eine fehlerfreie Auslieferung bereit und stelle sicher, dass alle Objekte und Abhängigkeiten in der richtigen Reihenfolge eingespielt werden?

Als Softwarearchitekten und Entwickler müssen wir in der Lage sein, die Perspektive zu wechseln. Sowohl *Bottom Up* als auch *Top Down* sind dabei wichtig. Das bezieht sich natürlich nicht nur auf die Applikation selbst, sondern auch auf die Art und Weise, wie man eine Applikation betrachtet. Betrachten wir unsere Applikation als Produkt. Welche Stufen durchläuft unser Produkt, bis es schließlich vom Benutzer bedient wird? Unser Produkt ist ständigen Neuerungen ausgesetzt und wird im Prinzip nie fertig. Es gibt immer wieder neue Ideen, die in Form von Features implementiert werden sollen. Zusätzlich ändern sich immer wieder Anforderungen. Bestehende Systeme unterliegen einem permanenten Wandel.

Anders als bei anderen Softwareprodukten haben wir in der Datenbankwelt das Problem vorhandener Daten und deren Struktur. Während also in anderen Bereichen „nur" Quellcode kompiliert und ausgeliefert wird, müssen wir auch noch auf die bereits vorhandenen Daten und Strukturen eingehen. Das ist, als würden wir Komponenten und Features an einem Auto erneuern, das voll beladen auf der Autobahn fährt.

dbFlow

Im Datenbankumfeld sind mit Liquibase und Flyway zwei Lösungen im Umlauf, die sich dieser Problematik widmen. Beide Lösungen kümmern sich in Form von Changesets um die Migration verschiedener Versionsstände von Datenbankschemas. Das Migrieren von Tabellen beziehungsweise das Einspielen der Änderungen ist nur ein kleiner Teil einer Softwareeinspielung. Im APEX-Bereich gibt es viel mehr zu berücksichtigen, damit das Deployment gelingt. Applikationen sollten während der Einspielung offline gesetzt, REST-Services sollten deaktiviert werden. Während der Einspielung sollten automatisierte Tests (nicht auf Produktion) durchgeführt werden. Bestimmte Automatismen, wie das Erstellen von beispielsweise Table-APIs, sollten ausgeführt werden. Changelogs über die Änderungen sollten gesammelt und bestimmten Benutzerkreisen zur Verfügung gestellt werden.

Mit dbFlow stelle ich Ihnen eine Lösung vor, die sich neben der Veränderung von Schemas beziehungsweise Versionsständen um all diese Themen kümmert und somit einen ganzheitlichen Lösungsansatz in der Auslieferung und Einspielung von Datenbank-basierten Anwendungen bietet.

dbFlow ist ein Open-Source-Tool unter der MIT-Lizenz. dbFlow läuft unter Bash und benötigt zum Bauen der Deployments zusätzlich nur noch Git. Zum Einspielen auf den jeweiligen Datenbanken kann wahlweise SQL*Plus oder SQLcl konfiguriert werden.

dbFlow baut im Kern auf 4 Konzepten auf:

1. Versionsverwaltung über Git und dem damit verbundenem Branch-Modell **Gitflow**
2. **2-Phasen-Deployment** für das Deployment als neues oder geändertes Produkt
3. **smartFS** als standardisierte Grundlage der Verzeichnisstruktur
4. Eigenes **Depot** als Artifactory

Gitflow

dbFlow setzt auf das Branch-Modell Gitflow auf. Gitflow beschreibt dabei einen Weg, wie Features und Änderungen in den Haupt-Entwicklungs-Branch (develop) gemergt werden. Damit wir diese Strategie in der Datenbankwelt benutzten können, legen wir Folgendes fest.

Für jede Zielstage unseres Produktes gibt es einen Branch, zum Beispiel test, acceptance, master (Production).

Somit werden Artefakte in der Testdatenbank eingespielt, die durch das Mergen in den Test-Branch mit dbFlow erstellt wurden. Das Gleiche gilt für eine mögliche Acceptance- oder Produktionsumgebung, in der Code aus dem Master-Branch eingespielt wird. Die Entwicklung findet im Haupt-Entwicklungs-Branch statt und kann gegebenenfalls noch auf weitere Feature-Branches verteilt werden. Der Develop-Branch sowie die Feature-Branches werden in der Entwicklungsdatenbank entwickelt. Ein zusätzlicher Release-Branch kann als Release-Schranke beziehungsweise Sperre fungieren, in der Anpassungen für das aktuelle Release nur noch im Release-Branch implementiert werden.

So kann die Entwicklung im Develop-Branch unabhängig vom aktuellen Release fortgesetzt werden.

dbFlow kann nun durch das Mergen in einen Branch das Deployment bauen. Im Grunde wird durch das Delta der veränderten Dateien ein Patch gebaut, der diese dann in die jeweilige Zieldatenbank einspielt. Durch die Nutzung von Git kann man sich dabei auf den exakten Zustand vor dem Mergen oder irgendeinen anderen in der Vergangenheit beziehen. Das Artefakt, das dbFlow nun erstellt (Tarball), kann jetzt in der Zieldatenbank eingespielt werden *(siehe Abbildung 1)*.

2-Phasen-Deployment

dbFlow bedient sich eines sogenannten 2-Phasen-Deployments. Dabei kann ein Artefakt gegen eine „leere" Datenbank oder als Patch auf eine bestehende Version eingespielt werden. Das funktioniert wie die Installation eines Office-Paketes. Entweder habe ich noch kein Office installiert, dann muss ich install.exe starten. Oder ich habe bereits eine Version installiert und muss nun update.exe ausführen.

Diese Art des Deployments bedeutet, dass man jede Version als Initial-Version und zusätzlich als passendes Update (Patch) für den jeweiligen Vorgänger erstellen kann. So kann man zum Beispiel Version 3 seiner Applikation initial deployen oder als Patch für die Version 2 (Delta) *(siehe Abbildung 2)*.

Mit diesem Konzept ist es ein Leichtes, die Release-Fähigkeit einer Applikation permanent feststellen zu lassen (Continious Integration). Wir haben hierfür eine entsprechende Pipeline in Jenkins hinterlegt. Erst, wenn sich alles einspielen lässt und alle Tests erfolgreich waren, ist ein Versionsstand releasefähig.

smartFS

Das Konzept von dbFlow setzt auf Git als Versionsverwaltung auf. Das heißt, es werden nur Dateien versioniert. Diese *MÜSSEN* im Dateisystem vorhanden sein!

Es gilt: Das Dateisystem gewinnt. Immer. *Immer!*

Dieser Ansatz führt zu einem Paradigmenwechsel. Und genau diesen kann

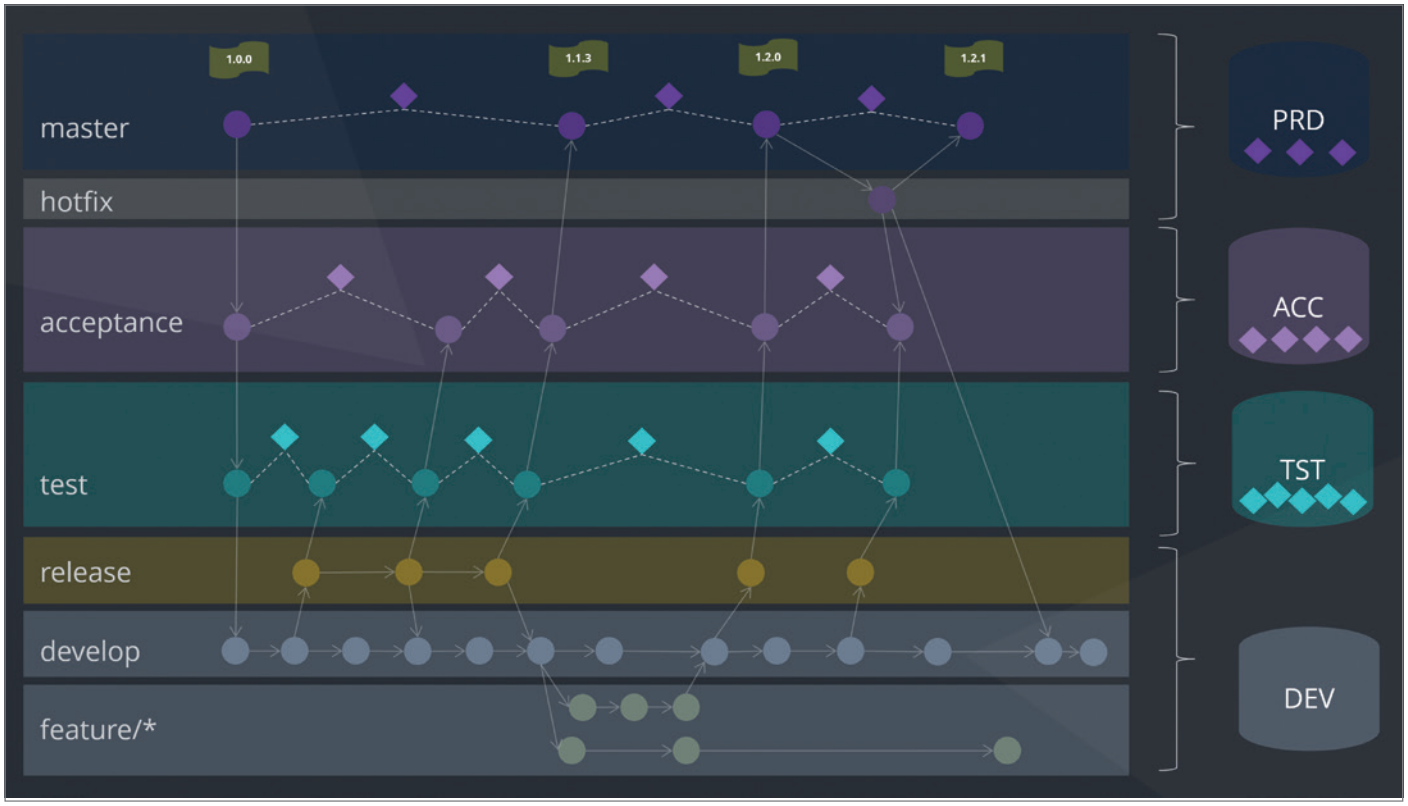

Abbildung 1: Gitflow-Branch-Modell mit Zuordnung zu Stage-Datenbanken (Quelle: Maik Michel)

ich Ihnen, liebe Leserin / lieber Leser, nur wärmstens ans Herz legen. Nutzen Sie zur Entwicklung einen Quellcode-Editor, wie zum Beispiel VSCode in Verbindung mit dbFlux (siehe Red Stack 01/2022, S.64). Wie schnell hat man seine Änderungen im SQL-Developer oder TOAD umgesetzt und vergessen, in die Versionskontrolle einzuchecken. Arbeiten Sie lieber von außen nach innen (Dateien schreiben und in die DB einspielen) als von innen nach außen (Packages, Tabellen anlegen und in das Dateisystem exportieren/speichern).

smartFS stellt hier eine standardisierte Verzeichnisstruktur dar. Auf der obersten Ebene stehen Verzeichnisse für verschiedene APEX-Applikationen, REST-Module sowie Datenbankschemas bereit. Ferner werden hier Konfigurationsdateien abgelegt, die das Projekt sowie die jeweilige Zieldatenbank definieren (build.env, apply.env). smartFS unterstützt drei verschieden Projektarten, die sich von der Anzahl der benutzten Schemas herleiten.

Das SingleSchema kennt nur ein Schema, während das MutliSchema drei Schemas kennt und sich hier des klassischen 3-Schichten-Modells bedient (DATA für Tabellen, LOGIC für die Geschäftslogik und APP für die Applikationslogik). Dann gibt es noch das FlexSchema, das n Schemas kennt. Zuordnungen zum Workspace und Schema werden im SingleSchema- und MultiSchema-Mode in der Konfiguration hinterlegt. Beim FlexSchema-Mode werden diese Informationen als zusätzliches Level in der Verzeichnishierarchie dargestellt.

Neben den jeweiligen Schemas wird im *db*-Ordner auch ein *_setup*-Ordner bereitgestellt. Hier werden die Skripte abgelegt, die dann eingesetzt werden zur Erstellung der Schemas, der jeweiligen DB-Benutzer sowie der abhängigen Features, wie beispielsweise Logger oder utPLSQL. In der Projektdefinition wird hinterlegt, mit welchem Admin-Account diese Skripte ausgeführt werden sollen. Jedes Datenbankschemaverzeichnis hält für jeden in der DB befindlichen Objekttyp ein eigenes Unterverzeichnis bereit. Manche Verzeichnisse sind noch weiter unterteilt. Diese Unterteilung bestimmt später die Reihenfolge der jeweiligen Dateien während der Einspielung.

In jedem Level des Verzeichnisbaums gibt es sogenannte Hook-Ordner (*.hooks*), die sich abhängig von der jeweiligen Installationsart (*init* oder *patch*) und dem jeweiligen Einspielungszeitpunkt während eines Deployments unterteilen. Diese Hook-Ordner können Skripte enthalten, die während der Installation aufgerufen werden. Somit können zum Beispiel TableAPIs erstellt oder UnitTests ausgeführt werden.

Da dbFlow das sogenannte 2-Phasen-Deployment unterstützt, werden Tabellenänderungen doppelt gepflegt. Das bedeutet, es gibt ein Tabellenskript im Verzeichnis *tables*, das das komplette Create-Table-Statement enthält. Ein weiteres Skript enthält die Änderungen an der jeweiligen Tabelle und liegt im Unterverzeichnis *tables/tables_ddl*.

Beispiel: Fügen wir eine neue Spalte an die Tabelle employees ein, dann wird das create statement in der Datei employees.sql angepasst und es wird eine neue Datei employees.1.sql (1 steht für die erste Änderung) im Verzeichnis *tables_ddl* erstellt, die das Alter-Table-Statement enthält. Später, während des Deployments, wird dbFLow feststellen, dass es Änderungen an Dateien mit dem gleichen Dateistamm im *tables*-Verzeichnis sowie im *tables_ddl*-Verzeichnis gibt, und daher nur das Skript aus dem *tables_ddl*-Verzeichnis ausführen, sofern es sich beim Deployment um einen Patch handelt. Im Initfall wird nur das Create-Table-Skript ausgeführt.

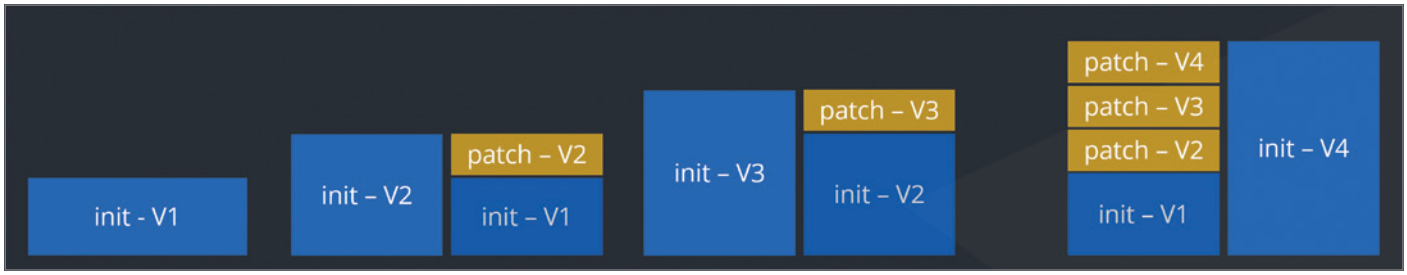

Abbildung 2: Version 3 seiner Applikation initial deployen oder als Patch für die Version 2 (Delta) (Quelle: Maik Michel)

Für das Ändern von Daten oder Spezialfälle wie Aufräumarbeiten stehen die Ordner *ddl* beziehungsweise *dml* bereit. Hier werden ähnlich wie bei den Hooks Skripte hinterlegt, die abhängig von der Installationsart oder dem Einspielungszeitpunkt aufgerufen werden.

depot

Das Depot stellt eine Art Artifactory dar und wird als Zielverzeichnis konfiguriert. Abhängig vom Git-Branch, von dem aus ein Deployment erstellt wird, wird das einzuspielende Artefakt, das dbFLow erstellt (Tarball), in ein Unterverzeichnis des Depots abgelegt. Zum Beispiel: *depot/test/patch_4.7.12.tar*

Im Instanzverzeichnis, von dem aus jede Installation in der entsprechenden Stage ausgeführt wird, kann nun mit dbFLow dieses Artefakt installiert werden. dbFlow wird dabei im *depot* im Unterordner test den Patch mit der entsprechenden Version holen und einspielen, weil die Stage dem Branch *test* zugeordnet wurde. Sofern es keine Fehler beim Deployment gab, werden der Patch sowie die jeweiligen Logs in einem Unterverzeichnis mit dem Namen *success* passend zum Artefakt abgelegt. Einspielungen, die schiefgelaufen sind, werden dabei im Ordner *failure* abgelegt.

dbFlow hat die Möglichkeit, eine Installation an dem Punkt fortzusetzen, an dem etwas fehlgeschlagen ist. Dabei wird das jeweilige Logfile als Parameter übergeben. dbFlow wird nun mit der Datei fortfahren, die nicht zu einem positiven Feedback in der alten Logdatei geführt hat.

Ich empfehle, das Depot-Verzeichnis außerhalb des Entwicklungsverzeichnisses zu platzieren und ebenfalls als Git-Repository anzulegen. Somit stehen Automatismen zur Verfügung, um ein Artefakt automatisiert in eine Zielumgebung einzuspielen.

Prozess im Ganzen

In der Gesamtbetrachtung stellt sich der Workflow so dar: Sie entwickeln in einem Verzeichnisbaum, der dem smartFS-Aufbau entspricht, in einem Haupt-Entwicklungs-Branch oder Feature-Branch, auf einer Entwicklungsdatenbank und checken ihre jeweiligen Änderungen entsprechend mit Git in die Versionsverwaltung ein. Hier kann ein CI/CD-Tool, zum Beispiel Jenkins, den Entwicklungs-Branch komplett oder als Delta zum Test-Branch in eine Builddatenbank einspielen (Nightlybuilds). Der Releaseverantwortliche kann, sofern natürlich alle Tests erfolgreich abgeschlossen wurden, durch das Mergen über den Release-Branch (Release-Schranke) in den Test-Branch das Artefakt mit dbFlow erstellen und im depot ablegen. Automatisiert oder manuell kann nun, von einem Instanzverzeichnis aus, das Artefakt mit dbFlow eingespielt werden. Anfallende Logs werden von dbFlow in das Depot-Verzeichnis unter success wieder abgelegt *(siehe Abbildung 3)*.

Wie kann ich starten?

dbFLow wird als Submodul in einem bereits bestehenden Git Repository installiert. *(siehe Listing 1 für die Anlage eines dbFLow-Projekts)*.

dbFlow hat nun ein Verzeichnis inklusive Schemas, User, Features und APEX Workspace angelegt. Jetzt kann das Entwickeln auf dem Develop-Branch losgehen.

Sobald alle auszuliefernden Änderungen commitet wurden, kann in den Ziel-Branch, zum Beispiel test, gemergt werden. Um nun einen Patch mit der Version 1.0.0 aus dem Delta zu erzeugen, führt man folgenden Befehl aus: `.dbFlow/build.sh --patch --version 1.0.0`

dbFlow baut nun den Patch und legt diesen im Depot ab. Wechselt man nun in das Instanzverzeichnis oder etwa auf

```
# create a folder for your project and change directory into
$ mkdir demo && cd demo

# init your project with git
$ git init

# clone dbFlow as submodule
$ git submodule add https://github.com/MaikMichel/dbFlow.git .dbFlow

# generate and switch to your development branch
$ git checkout -b develop

# generate project structure
$ .dbFlow/setup.sh generate <project_name>

# after processing the wizard steps, just install
$ .dbFlow/setup.sh install
```

Listing 1: bash-Befehle für die Anlage eines dbFlow-Projekts

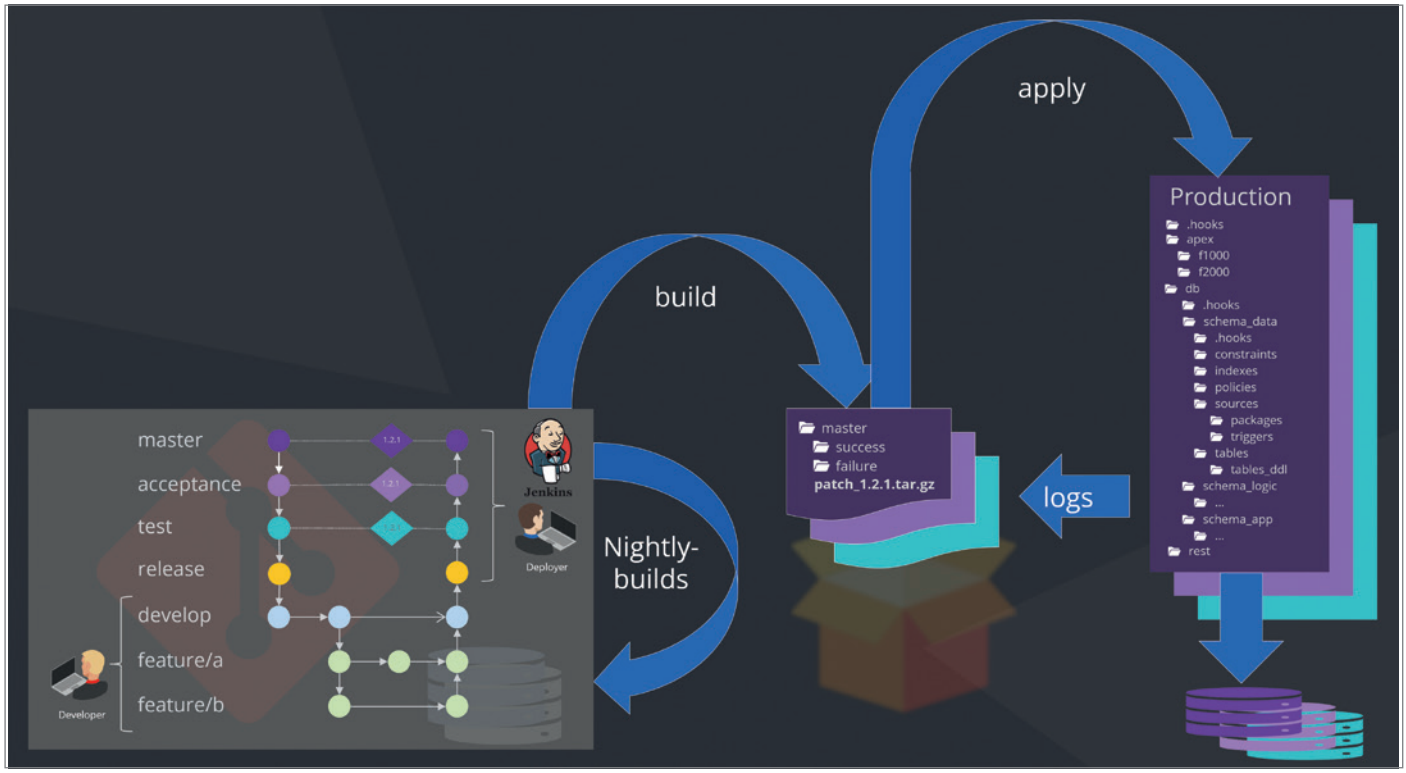

Abbildung 3: Zusammenfassung des Prozesses im Ganzen (Quelle: Maik Michel)

eine andere VM, kann man mit dem folgenden Befehl den Patch einspielen: `.dbFlow/apply.sh --patch --version 1.0.0`

Dieser Artikel kann nicht alle Funktionalitäten von dbFlow beschreiben. Ich denke, dass Sie einen ersten Eindruck von dbFlow gewonnen haben, und lade Sie herzlich ein, dbFlow einfach mal auszuprobieren.

dbFlow inklusive Dokumentation finden Sie auf GitHub unter folgendem Link:
https://github.com/MaikMichel/dbFlow

Quellen

1. Titelbild: Unsplash - *https://unsplash.com/photos/KPAQpJYzH0Y*

Über den Autor

Maik Michel ist LEAD Developer und Consultant bei der Opitz Consulting Deutschland GmbH. Hier verantwortet er verschiedene Projekte und Teams rund um das Thema Digitale Transformation mit der Low-Code-Plattform APEX. Daneben ist er Autor des Blogs micodify.de und als Sprecher auf verschiedenen Konferenzen zu finden.

Maik Michel
maik.michel@opitz-consulting.com