Jenkins Multi-Branch Build Pipeline mit Docker und SQL Server

Kathegorie
SQL Server, Docker

Aaron
Priesterroth

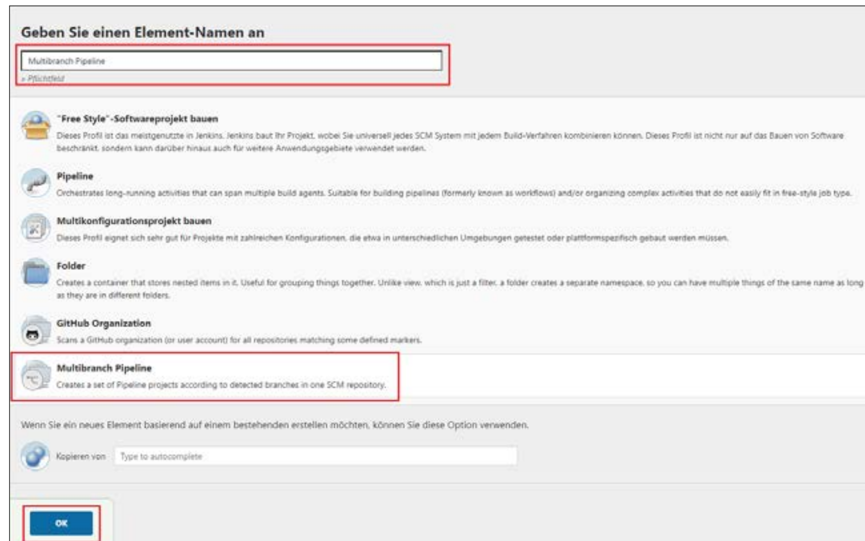
Der erste Schritt jedes Systems für kontinuierliche Integration ist, das Projekt unter Source Code Control zu bringen. Danach kann man sich eine Branching-Strategie überlegen, die dem Projekt angemessen ist. Ein "Branch" beschreibt dabei eine Evolutions-Linie einer Code Base in einem Source Control Repository. Ein Source Code Control System wie z.B. Git hat immer einen **master**-Branch als Grundlage der Veränderung der Code-Base. Da in den meisten Fällen mehr als nur ein Entwickler an einem Projekt arbeitet und das Arbeiten mit nur einem Branch sich als sehr unpraktisch erweist, benötigen wir die zuvor erwähnte Branching-Strategie.

An dieser Stelle kommt das Jenkins **Multi-Pipeline** Feature ins Spiel! In den folgenden Abschnitten werden wir die im Letzten Artikel erstelle CI Pipeline zu einer Multi-Branch Build Pipeline erweitern. Der Artikel kann hier gefunden werden.

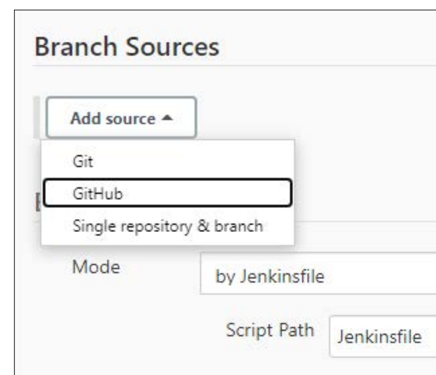
Multi-Branch Build Pipeline erstellen

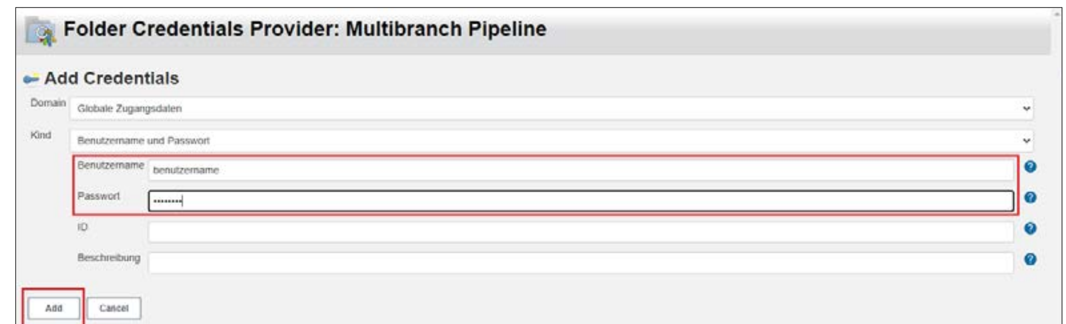
Für den folgenden Abschnitt wird eine Installation von Jenkins, Docker und dem SQL Server Dacfx Framework vorausgesetzt. Alle Informationen und Anleitungen zur Installation können im vorausgehenden Artikel gefunden werden.

Über die Default-URL http://localhost:8080/ muss auf die Startseite von Jenkins navigiert werden. Dort angekommen kann mit einem Klick auf **New Item** in der oberen linken Ecke ein neues Item erzeugt werden. Als Nächstes muss der Name und der Typ des Items, das wir erstellen möchten, angegeben werden. Hier geben wir den Namen **Muli-branch build pipeline** an und wählen den Item-Typ **Multibranch Pipeline** aus. Mit einem Klick auf **OK** wird die Eingabe bestätigt.



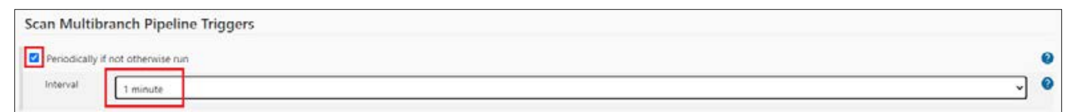
Für die Branch Sources wählen wir das Source Control System unserer Wahl aus (in diesem Beispiel GitHub) und authentifizieren uns, indem wir ein neues Credential-Set anlegen. Hier müssen nun **Benutzername** und **Passwort** angegeben werden. Mit einem Klick auf **Add** wird das Credential-Set angelegt. Nun kann es über das Drop-Down Menü ausgewählt werden.





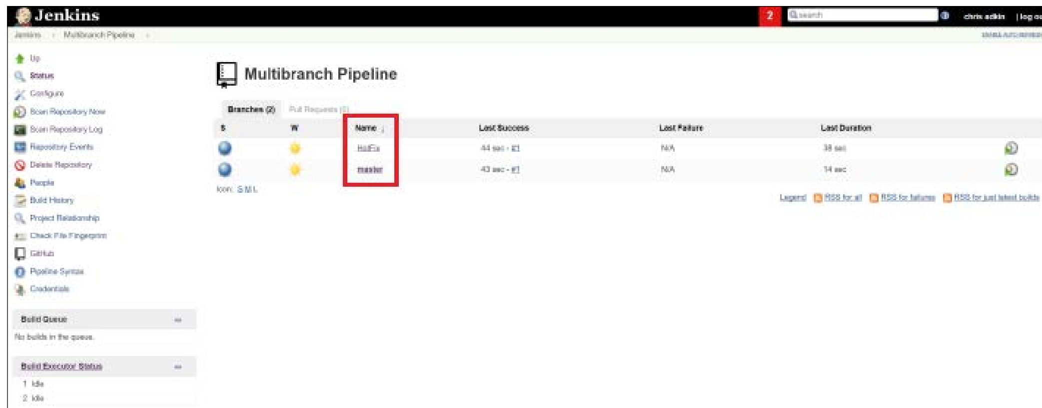
Als Nächstes muss der **Owner** (Besitzer) des Repository angegeben und das gewünschte Repository für die Pipeline aus dem Drop-Down Menü ausgewählt werden.

Als Nächstes möchten **wir**, dass Jenkins periodisch das Repository überprüft und ggf. automatisch neue Pipelines erzeugt, sollten neue Branches erkannt werden. Um diese zu konfigurieren, muss die Checkbox unter **Scan Repository Triggers** ausgewählt werden. Das Intervall kann beliebig gewählt werden. Mit einem Klick auf **Save** können die Änderungen und Einstellungen gespeichert werden.



Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Auf dem neuen Screen können nun mit einem Klick auf **Scan Respository Now** die Pipelines für jeden Branch erzeugt werden. Sobald Jenkins das Repository überprüft und die Anzahl der Branches bestimmt hat, können wir uns selbst vom Ergebnis überzeugen:



Build Pipeline als Code

Eine der besonderen Stärken von Jenkins ist, dass dem Benutzer erlaubt wird, Pipelines in Form von Codes zu spezifizieren. Den folgende Code im **Groovy script Syntax** werden wir für die Spezifikation unserer Pipeline verwenden. Wir nehmen dabei an, dass pro Branch ein Container erzeugt wird, wobei jeder Container auf dem selben Host erzeugt wird. Wir müssen also darauf achten, dass jeder Container seinen eigenen Port bekommt und dieser innerhalb des Containers dann auf Port 1433 zugewiesen wird.

```
def BranchToPort(String branchName) {
def BranchPortMap = [
[branch: ‘master’ , port: 15565],
[branch: ‘Release’ , port: 15566],
[branch: ‘Feature’ , port: 15567],
[branch: ‘Prototype’, port: 15568],
[branch: ‘HotFix’ , port: 15569]
]
BranchPortMap.find { it[‘branch’] == branchName }[‘port’]
}

def StartContainer() {
bat “docker run -e \”ACCEPT_EULA=Y\” -e \”SA_PASSWORD=P@ssword1\” --name
SQLLinux${env.BRANCH_NAME} -d -i -p ${BranchToPort(env.BRANCH_NAME)}:1433
microsoft/mssql-server-linux”
}

def DeployDacpac() {
def SqlPackage = “C:\\Program Files\\Microsoft SQL Server\\140\\DAC\\bin\\
sqlpackage.exe”
def SourceFile = “SelfBuildPipeline\\bin\\Release\\SelfBuildPipeline.dacpac”
def ConnString = “server=localhost,${BranchToPort(env.BRANCH_
NAME)};database=SsdtDevOpsDemo;user id=sa;password=P@ssword1”

unstash ‘theDacpac’
bat “\”${SqlPackage}\” /Action:Publish /SourceFile:\”${SourceFile}\” /
TargetConnectionString:\”${ConnString}\” /p:ExcludeObjectType=Logins”
}

node {
stage(‘git checkout’) {
git ‘https://github.com/chrisadkin/SelfBuildPipeline’
}
stage(‘build dacpac’) {
bat “\”${tool name: ‘Default’, type: ‘msbuild’}\” /p:Configuration=Release”
stash includes: ‘SelfBuildPipeline\bin\\Release\\SelfBuildPipeline.dacpac’,
name: ‘theDacpac’
}
stage(‘start container’) {
StartContainer()
}
stage(‘deploy dacpac’) {
try {
DeployDacpac()
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
catch (error) {
throw error
}
finally {
bat "docker rm -f SQLLinux${env.BRANCH_NAME}"
}
}
}
```

Der wichtigste Punkt ist die **BranchToPort** Funktion, die die einzelnen Branches ihren jeweiligen Ports zuweist, und die Verwendung der Environment-Variables ${env.BRANCH_NAME}. Strukturell besteht unsere Spezifikation aus insgesamt fünf unterschiedlichen Branches:

- master
- Release
- Feature
- Prototype
- HotFix

In Zeile 13 und 19 wird der Branch mit Hilfe von ${env.BRANCH_NAME} in die Funktion übergeben, damit zum Start des Containers der richtige Port verwendet wird. Gleichzeitig wird die Umgebungsvariable dazu verwendet, den Namen des Containers und den Connection-String zu erzeugen.

Damit haben wir unsere Multi-Branch Build Pipeline erzeugt und wiederum die Basis für den dritten und letzten Teil der Artikel-Reihe, in der wir mit Hilfe von Jenkins, Docker und SQL Server automatisierte Unit-Test durchführen wollen, geschaffen.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612