CONTINUOUS INTEGRATION

Continuous Integration und Continuous Delivery für Access mit Continua-CI

Tutorial zum Vortrag auf der AEK 21

REQUIREMENTS

Um CI mit Access realisieren zu können werden einige Dinge benötigt. In diesem Tutorial wird das mit den folgenden Tools realisiert:

- Subversion (SVN)
  Continuous Integration macht ohne eine Quellcodeverwaltung keinen Sinn. Für den Vortrag habe ich eine VM aufgesetzt die über Visual-SVN-Server einen Subversion-Server bietet.
  https://www.visualsvn.com/server/
- OASIS SVN
  Damit der CI-Server eine neue Datenbank erzeugen kann, wird OASIS-SVN verwendet.
  Für einen Test mit weniger als 50 Objekten reicht die kostenlose Demoversion aus.
  https://dev2dev.de
- TortoiseSVN
  OASIS arbeitet direkt mit Subversion oder Git zusammen. Ich persönlich arbeite noch immer mit Subversion und nicht mit Git.
  https://tortoisesvn.net
- Continua-CI
  Der eigentliche Continuous-Integration-Server. Mit 1 Agenten kostenlos einsetzbar.
  https://www.finalbuilder.com/continua-ci

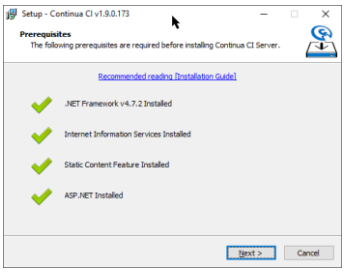
INSTALLATION

Die genannten Tools werden mit ihrem jeweiligen Installer installiert.

Ein wenig Beachtung benötigt hier lediglich TortoiseSVN. Hier ist darauf zu achten, dass die Command-Line-Client-Tools mit installiert werden.

OASIS und TortoiseSVN werden auf dem Entwicklungsrechner installiert. Für Continua-CI und Visual-SVN-Server sollte man einen eigenen Server aufsetzen. Es kann auch beides auf derselben Maschine installiert werden.
Am Ende dieses Tutorials werde ich den Umgang mit OASIS und TortoiseSVN noch kurz erklären.

Der Installer von Continua prüft selbst, ob die Voraussetzungen zur Installation gegeben sind und macht mit passenden Meldungen auf Probleme aufmerksam.



- .NET Framework
- IIS (Internet-Information-Services)
- Static Content
- ASP.NET

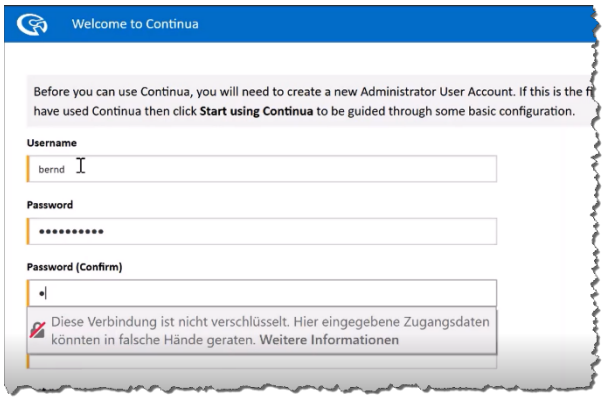
Bei Bedarf lassen sich die einzelnen Optionen in den Windows-Features aktivieren und installieren.

Der Installer fragt die benötigten Einstellungen wie z.B. Datenbankverbindung, Benutzeraccount und den zu verwendenden Port während der Installation ab.

Nach Abschluss der Installation sollte überprüft werden, ob der zu verwendende Port auch in der Firewall freigegeben ist. Ansonsten wird ein Zugriff von anderen Computern nicht möglich sein.

INBETRIEBNAHME

Nach Abschluss der Installation wird automatisch der Standardbrowser gestartet, da zunächst ein Administrator-Account eingerichtet werden muss.



Hier gilt es, kleinere Fallstricke zu beachten:

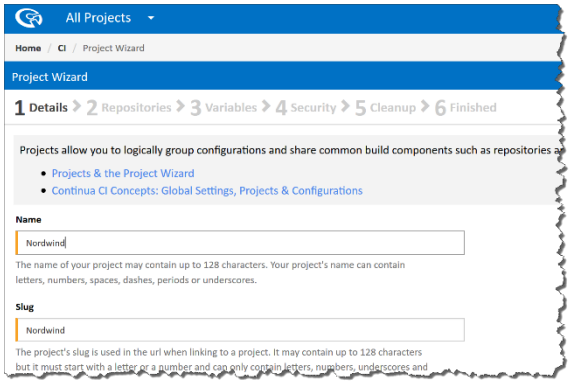
- Komplexes Passwort erforderlich
  Bei der Einrichtung des Accounts wird ein komplexes Passwort benötigt.
  Nach der Einrichtung kann es über die Benutzeroberfläche allerdings beliebig geändert werden.
- Sonderzeichen
  Wird ein Passwort verwendet das die Sonderzeichen „%“ oder „$“ enthält, müssen diese Zeichen verdoppelt werden da sie für Continua besondere Bedeutung haben.

PROJEKT ANLEGEN

Um sinnvoll mit dem CI-Server arbeiten zu können, muss zunächst ein Projekt angelegt werden.

SCHRITT 1 – DETAILS

Im ersten Schritt wird lediglich ein Name für das neue Projekt eingegeben.
ACHTUNG: Keine Umlaute verwenden!



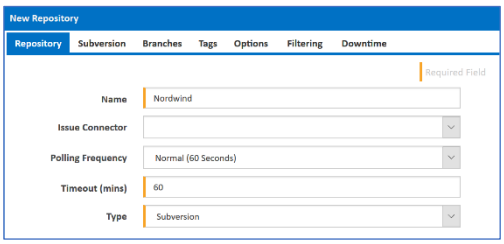
Das Feld „Slug“ wird für den Zugriff per URL benötigt und automatisch gefüllt. Leerzeichen werden durch Unterstriche ersetzt.

Wer möchte, gibt noch eine zusätzliche Beschreibung ein.

Die Option „Set project-wide versioning“ bleibt nicht aktiviert.

SCHRITT 2 – REPOSITORY

Im zweiten Schritt wird die Verbindung zum Quellcode-Repository eingerichtet.

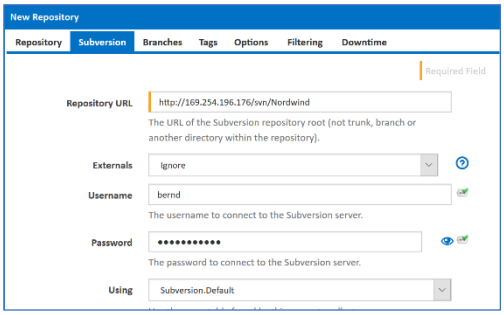


Um nicht gleich zu Beginn Flexibilität einzubüßen, verzichten wir hier auf ein „Globales Repository“ sondern legen über den Link „Create“ ein projektspezifisches Repository an.

Zunächst muss ein Name für das Repository vergeben und der Typ der Quellcodeverwaltung werden.

Der Wert für die „Polling Frequency“ bestimmt, in welchem Intervall das Repository auf Änderungen abgefragt wird. Der hier gewählte Wert hat später Auswirkungen auf die Reaktionszeit beim Einchecken von Änderungen. Wird hier ein hoher Wert gewählt, dauert es möglicherweise länger bis der Server mit seinen Aktionen beginnt.

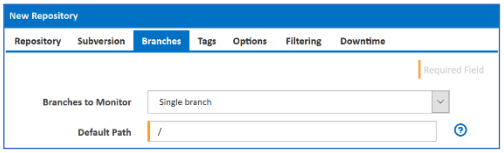
Die eigentliche Verbindung zum Repository wird erst nach der Auswahl von Subversion als Typ auf der zweiten Seite eingerichtet.



Liegt das Repository auf einem separaten Server, wird der Zugriff über den URL eingerichtet. Falls Continua und das Repository auf dem gleichen Computer liegen, kann der Zugriff auch über einen Datei-URL nach dem Muster file:///XYZ realisiert werden.

Auf dieser Seite geben wir lediglich noch Benutzernamen und Passwort ein.
ACHTUNG: Wird ein Passwort verwendet das die Sonderzeichen „%“ oder „$“ enthält, müssen diese Zeichen verdoppelt werden da sie für Continua besondere Bedeutung haben.

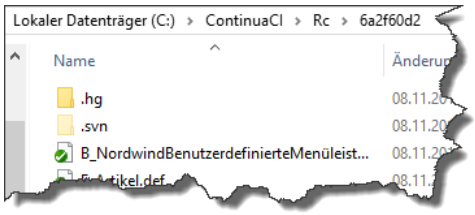
Auf Seite 3 wird eingerichtet, welcher Branch des Repository verwendet werden soll.
Falls wie im Beispiel keine Branches existieren, reduziert sich die Angabe auf „/“.



Abschließend folgt auf Seite 4 die Einrichtung des gewünschten Tag. Im Beispiel werden keine Tags verwendet, daher stellen wir die Auswahl auf „Ignore“ ein.

Nun kann durch einen Klick auf „Validate“ der Zugriff auf das Repository überprüft werden. Sollte hier noch etwas im Argen liegen, weist eine entsprechende Fehlermeldung darauf hin.

Nachdem der Dialog mit „Save“ beendet wurde, wird der Server sofort das Repository ansprechen und sämtliche Dateien aus dem Repository abholen.
Das Repository wird unter einem eindeutigen (und leider nicht sprechenden) Namen im Unterverzeichnis „Rc“ der bei der Installation angegebenen Freigabe abgelegt.



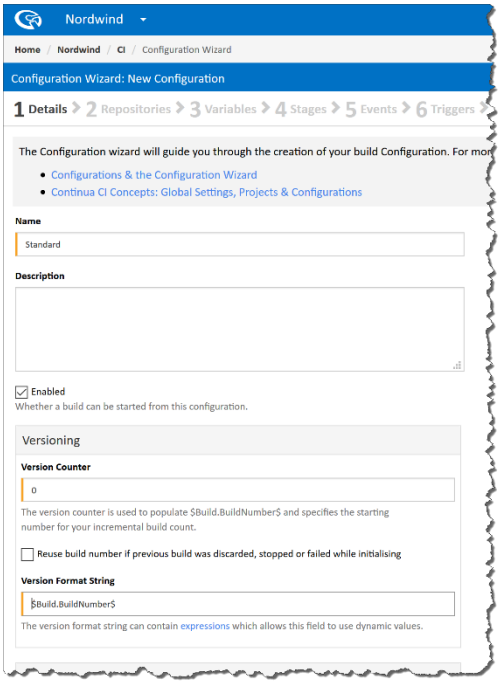
Mit Einrichtung des Repository ist die Konfiguration des Projektes abgeschlossen. Die Schritte 3 – 6 werden im Beispiel nicht benötigt.

KONFIGURATION ANLEGEN

Zu jedem Projekt können beliebig viele Konfigurationen angelegt werden. So könnte z.B. eine Konfiguration „NightlyBuild“ zu einem festgelegten Zeitpunkt das Projekt täglich neu bauen.
Wir legen im Beispiel eine Konfiguration an, die per Repository-Trigger auf Änderungen in der Quellcodeverwaltung reagieren wird.

SCHRITT 1 – DETAILS

Im ersten Schritt werden die grundsätzlichen Details der neuen Konfiguration eingestellt:



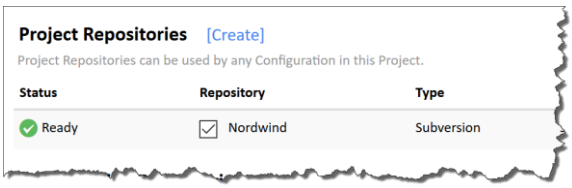
- **Name**
  Wir vergeben einen sinnvollen Namen für die Konfiguration
- **Description**
  Wer möchte, kann hier einen beschreibenden Text für die Konfiguration eingeben.
- **Enabled**
  Die Konfiguration wird natürlich aktiviert
- **Version Counter**
  Hier kann der Startwert für die laufenden Buildnummer verändert werden.
- **Version Format String**
  An dieser Stelle reicht uns als Versionsnummer die laufenden Nummer des Build.
  Wir ändern die Vorgabe entsprechend ab auf `$Build.BuildNumber$`.

Alle weiteren Optionen belassen wir auf den vorgegebenen Standardwerten. Sie haben vorerst keine Bedeutung für das Beispiel.

Wir klicken auf „Save & Continue“ um mit Schritt 2 fortzufahren.

SCHRITT 2 – REPOSITORY

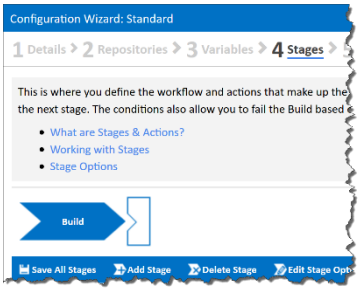
Das Repository haben wir bereits bei der Einrichtung des Projekts eingerichtet.



Schritt 2 reduziert sich daher auf das Auswählen des gewünschten Repository.

Wir klicken auf „Continue“ um mit Schritt 3 fortzufahren. Da wir keine Variablen verwenden, klicken wir erneut auf „Continue“ und gelangen zu Schritt 4.

SCHRITT 4 – STAGES

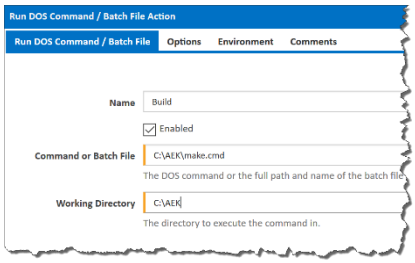


Schritt 4 ist der wichtigste und umfangreichste, denn hier werden die eigentlichen Aktionen konfiguriert.

Die Aktionen werden in sogenannten „Stages“ zusammengefasst, die einzelne Gruppen von Aktionen unter einem gemeinsamen Namen zusammenhalten.

Die erste Stage namens „Build“ hat der Server bereits angelegt. Das passt soweit auch gut, so dass wir es zunächst dabei belassen.

Die erste Aktion die wir benötigen, ist das Zusammenbauen eine neuen accdb-Datei aus den Objekten in der Quellcodeverwaltung.



Da es leider keine vorkonfigurierten Aktionen gibt die uns hier unterstützen würden, bleibt uns nur eine kleine Batchdatei um das Gewünschte zu erreichen.
Wir wählen also aus der Aktionsgruppe „Execution“ die Aktion „Run DOS Command / Batchfile“.

- **Name**
  Wir vergeben einen sinnvollen Namen.
- **Enabled**
  Natürlich wird der Schritt aktiv gesetzt.
- **Command or Batch File**
  Wir tragen Pfad und Dateiname der auszuführenden Batchdatei ein.
  ACHTUNG: Hier handelt es sich selbstverständlich um ein Verzeichnis auf dem Build-Server.
- **Working Directory**
  Als Arbeitsverzeichnis tragen wir das Verzeichnis ein in dem sich die Batchdatei befindet.

Wir speichern unsere Eingaben mit „Save“ und schließen damit den Dialog.

Durch einen Klick auf „Save & Continue“ speichern wir die Änderungen im Schritt 4 und gelangen zu Schritt 5. Da wir keine Events benötigen klicken wir auf „Continue“ und gelangen zu Schritt 6.

Die Batchdatei enthält lediglich eine einzige Zeile:

```
"C:\Program Files (x86)\Microsoft Office\...\MSACCESS.EXE" /cmd OASIS#C:\AEK\Build\Nordwind.accdb
```

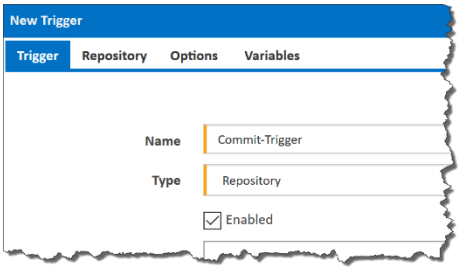
Der Aufruf startet die installierte Access-Version und übergibt als Kommandozeilenparameter den Namen der zu erzeugenden Datenbankdatei an OASIS.

OASIS wertet diesen Parameter aus und ist in der Lage, eine leere Datenbankdatei zu erzeugen und die im Repository abgelegten Objekte zu importieren.

SCHRITT 6 – TRIGGERS

Über die Trigger wird gesteuert, wann unsere Konfiguration aktiviert wird und die konfigurierten Aktionen ausführt. Grundsätzlich wird hier zwischen zeitgesteuerten und Repository-basierten Triggern unterschieden.

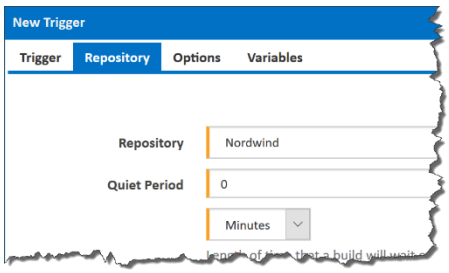
Wir erzeugen einen neuen Trigger durch einen Klick auf „Create“.



- **Name**
  Wir vergeben einen sinvollen Namen, z.B. „CommitTrigger“.
- **Type**
  Da wir auf Änderungen im Repository reagieren möchten, wählen wir als Typ „Repository“.

Alle weiteren Optionen belassen wir auf den Standardwerten.

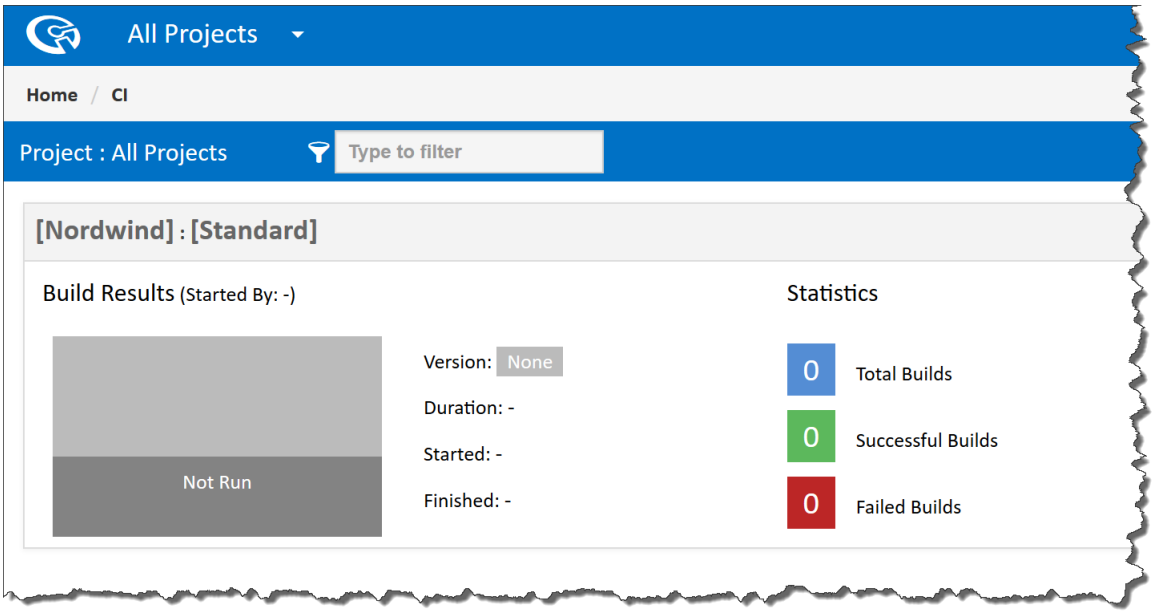
Auf der zweiten Seite des Dialogs stellen wir die Eigenschaften für das Repository ein:



- **Repository**
  Wir wählen das für diese Konfiguration eingerichtete Repository aus.
- **Quiet Period**
  Hier wird festgelegt, wie lange der Trigger wartet bevor er die Aktionen ausführt.
  Da wir möglichst schnell zu Ergebnissen kommen möchten, stellen wir die Ruhezeit auf 0 ein.

Alle weiteren Optionen belassen wir auf den Standardwerten und schließen den Dialog mit „Save“.

Die weiteren Schritte der Konfiguration werden nicht benötigt. Wir können die Einrichtung der Konfiguration daher mit einem Klick auf „Complete Wizard“ beenden.



Wir haben nun erfolgreich ein Projekt und eine Konfiguration eingerichtet. Auf der Übersichtsseite werden alle Projekte und Konfigurationen angezeigt.

Noch wurde die Konfiguration noch nie ausgeführt, daher sind noch keinerlei Details erkennbar.

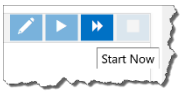
Bevor die Konfiguration zum ersten Mal gestartet werden kann, müssen zunächst noch einige Einstellungen vorgenommen werden:

- Die Einstellungsdatei für OASIS (im Beispiel „Nordwind.oasis“) muss in das Arbeitsverzeichnis kopiert werden.
- In der Einstellungsdatei muss auf den korrekten Pfad verwiesen werden.

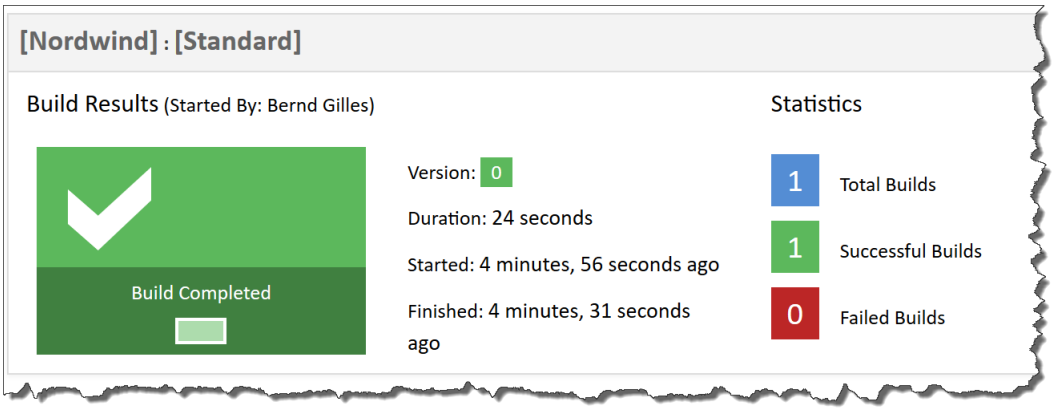
```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<OASIS>
  <ProjectSettings>
    <LocalPath value="C:\ContinuaCI\Rc\6a2f60d2\"/>
    <CompatibilityMode value="false"/>
```

Siehe hierzu: Schritt 2 – Repository (Seite 2)

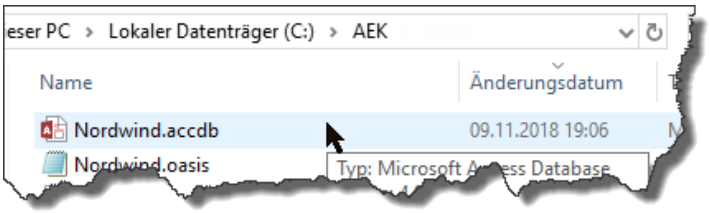
Durch einen Klick auf den passenden Button auf der rechten Seite können wir die Konfiguration nun manuell starten.



Haben wir alles korrekt eingestellt, erhalten wir nach wenigen Sekunden „grünes Licht“.



Im Arbeitsverzeichnis des CI-Servers wurde eine neue Datenbankdatei erzeugt:



KONFIGURATION ERWEITERN

Bislang wurde lediglich eine neue Datenbankdatei erzeugt. Ob diese Compilerfehler enthält, kann ohne die Erzeugung einer ACCDE nicht festgestellt werden.
Außerdem wurde noch keine Benachrichtigung eingerichtet, um z.B. per Mail auf Fehler im Prozess hingewiesen zu werden.

Daher werden wir die Konfiguration nun erweitern und zusätzlich noch die Benachrichtigungen einrichten.

ACCDE-DATEI

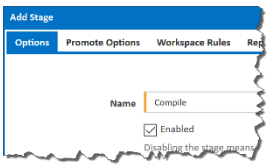
Zunächst erweitern wir die Konfiguration um in einem weiteren Schritt (Stage) aus der Datenbankdatei eine ACCDE-Datei zu erzeugen.

Dazu benötigen wir zunächst eine kleine VB-Script-Datei, die per PowerShell aus einer Datenbankdatei eine ACCDE erzeugt.

```
Dim app
Dim strDBName
Dim strADEName
Dim Result

  strDBName = "C:\AEK\Build\Nordwind.accdb"
  strADEName = "C:\AEK\Build\Nordwind.accde"
  Set app = CreateObject("Access.Application")

  Result = app.SysCmd(603, CStr(strDBName), CStr(strADEName))

  Set app = Nothing

  If Result = 1 Then
    WScript.Echo "ACCDE successfully created"
    WScript.Quit 0
  Else
    WScript.Echo "Compile error. Could not create ACCDE"
    WScript.Quit 7
  End If
```

Dieses Script speichern wir unter dem Namen „CreateMDE.vbs“ im Arbeitsverzeichnis des CI-Servers.

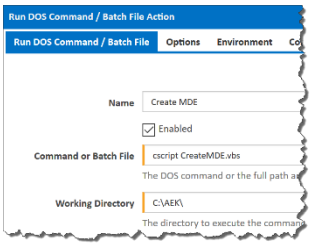
Nun bearbeiten wir die Konfiguration und fügen über „Add Stage“ eine neue Stage hinzu.



Wir tragen hier lediglich einen sinnvollen Namen für die neue Stage ein und speichern diese mit einem Klick auf „Save“.

Nun müssen wir dafür sorgen, dass in dieser Stage unser VB-Script aufgerufen wird.

Dazu verwenden wir erneut die Aktion „Run DOS Command / Batch File“.



- **Name**
  Hier vergeben wir einen sinnvollen Namen.
- **Enabled**
  Natürlich wird der Schritt aktiv gesetzt.
- **Command or Batch File**
  Wir rufen mit `cscript CreateMDE.vbs` unser Script auf.
- **Working Directory**
  Als Arbeitsverzeichnis legen wir unser bekanntes Verzeichnis auf dem CI-Server fest.

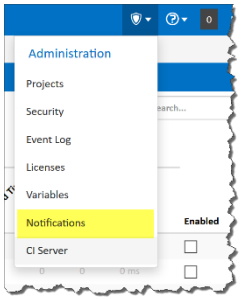
Wir speichern die Eingaben mit einem Klick auf „Save“ und können die Änderungen an der Stage nun mit „Save & Complete Wizard“.

Bevor wir die neue Stage testen, richten wir zunächst die Benachrichtigungen ein.

BENACHRICHTIGUNGEN EINRICHTEN

Eines der Hauptziele die wir mit dem Continuous-Integration-Prozess erreichen möchten, ist die Benachrichtigung über Erfolg und natürlich vor allen Dingen den Misserfolg des Prozesses.

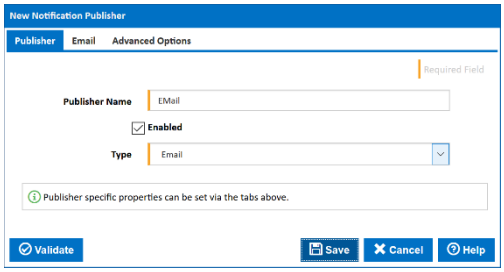
Dazu müssen wir entsprechende Benachrichtigungen einrichten.



Dazu muss zunächst eine passende Benachrichtigungsmöglichkeit erzeugt werden.
In Beispiel werden wir zu diesem Zweck eine Benachrichtigung per Email einrichten.

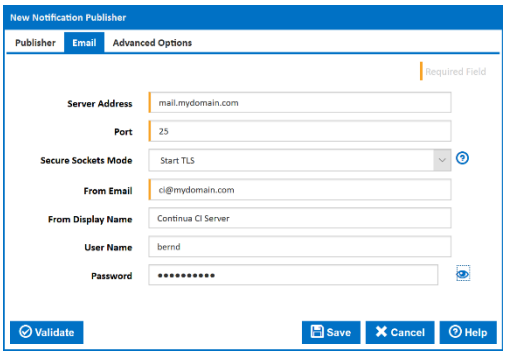
Die Einrichtung starten wir über das Administrationsmenü über den Menüpunkt „Notifications“.

Auf der linken Bildschirmseite erscheint nun eine umfangreiche Liste mit den Einstellmöglichkeiten des Servers.



Wir wählen unter der Überschrift „Notifications“ den Menüpunkt „Publishers“ und erzeugen mit dem Link „Create“ einen neuen Publisher.

- **Name**
  Wir vergeben einen sinnvollen Namen für den Publisher.
- **Type**
  Hier wählen wir aus den vorhandenen Möglichkeiten den Eintrag „Email“.

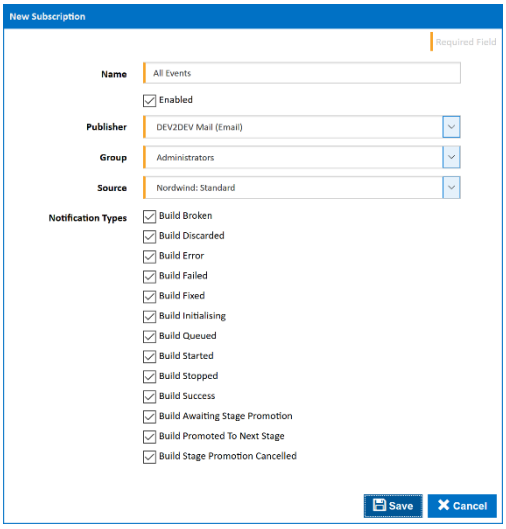


Nach der Auswahl des Typs erscheint eine weitere Seite im Dialog, auf der wir nun die Einstellungen für den Mailversand vornehmen.

- **Server Address**
  Hier tragen wir die Adresse unseres Mailservers ein.
- **Port**
  Der zu verwendende Port.
- **Secure Sockets Mode**
  Hier wählen wir die evtl. zu verwendende Verschlüsselung aus.
- **From Email**
  Die Mailadresse die zum Versand verwendet werden soll.
- **From Display Name**
  Absendername. Hier können wir einen sprechenden Namen verwenden, um die Mails im Posteingang leichter identifizieren zu können.
- **Username / Passwort**
  Die Zugangsdaten für den Mailserver und das verwendete Konto.

Mit einem Klick auf „Validate“ kann der Zugang getestet werden. Der Server versendet im Erfolgsfall eine Mail an die Adresse des Administrator-Accounts.

Als Nächstes richten wir eine Benachrichtigung ein, indem wir auf der linken Seite unter der Überschrift „Notifications“ den Menüpunkt „Subscriptions“ wählen.



Mit einem Klick auf den Link „Create“ öffnen wir den Einstellungsdialog und richten die Benachrichtigung ein.

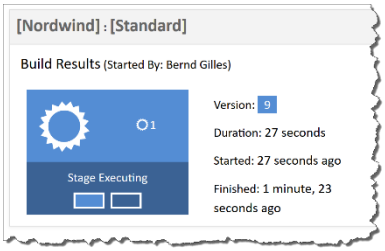
- **Name**
  Wir vergeben einen sinnvollen Namen.
- **Enabled**
  Natürlich wird die Benachrichtigung aktiviert.
- **Publisher**
  Wir wählen den im letzten Schritt eingerichteten Publisher aus.
- **Group**
  Wir wählen die Benutzergruppe aus, die diese Benachrichtigung erhalten soll.
  Über die Benutzerverwaltung lassen sich beliebige Gruppen anlegen und Benutzeraccounts zuordnen. Damit lassen sich die Benachrichtigungen noch feiner steuern.
- **Source**
  Hier wird eingestellt, auf welches Projekt/Konfiguration reagiert werden soll. Im Beispiel möchten wir nur auf Ereignisse einer bestimmten Konfiguration benachrichtigt werden.
- **Notification Types**
  Hier können wir nun auswählen, auf welche Art von Ereignissen die Benachrichtigung erfolgen soll.

Alle Einstellungen lassen sich später jederzeit auf die persönlichen Bedürfnisse und Vorlieben ändern.

Mit einem Klick auf „Save“ speichern wir die Einstellungen und haben nun unsere Benachrichtigungskette eingerichtet.

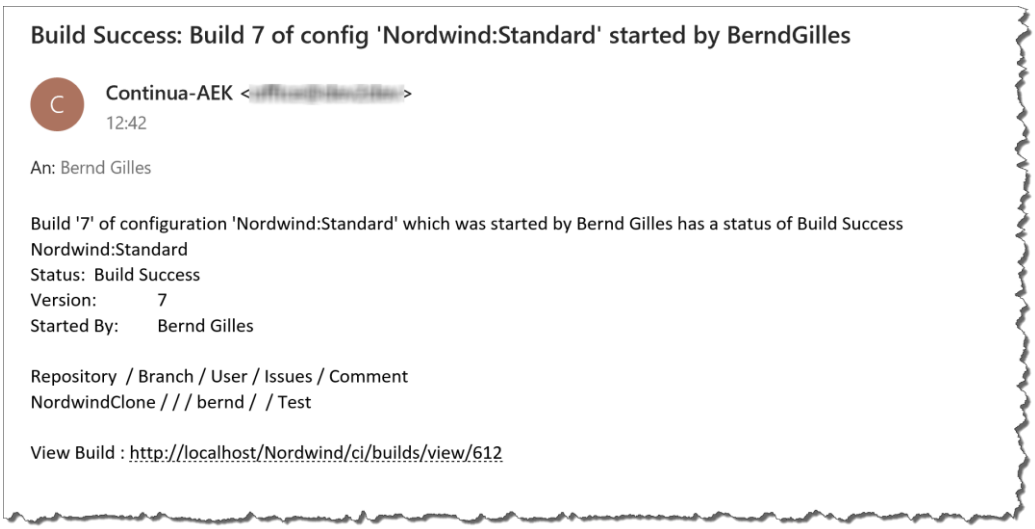
Von der Startseite aus können wir nun unsere Konfiguration erneut starten.

Wir stellen zunächst fest, dass sich offensichtlich etwas geändert hat, da das Status-Icon nun 2 kleine Rechtecke enthält.

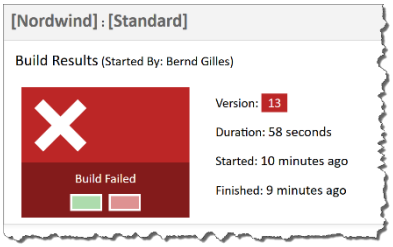


Grund dafür ist die neu hinzugefügte Stage. Für jede Stage wird ein eigenes Rechteck gezeichnet und fungiert im Verlauf des Build-Prozesses als Fortschrittsbalken. Hierüber kann man gut erkennen, an welcher Stelle sich der Build-Prozess während der Bearbeitung befindet.

Wurde alles korrekt konfiguriert, werden wir nun außerdem per Mail über die von uns gewählten Build-Ereignisse informiert.

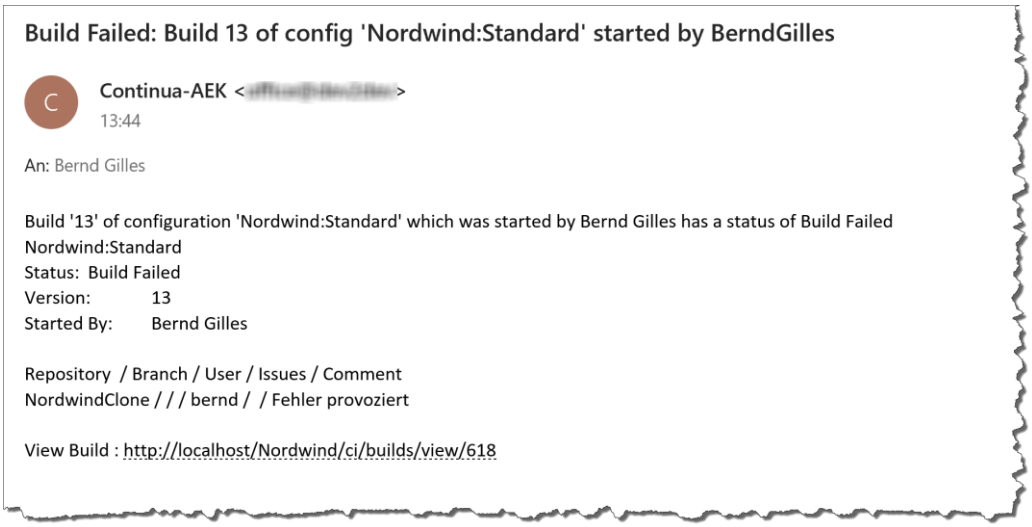


Provozieren wir nun einen Fehler, indem wir z.B. fehlerhaften VBA-Code in das Repository einschecken, wird von unserem kleinen VB-Script ein von 0 abweichender Return-Code zurückgegeben.

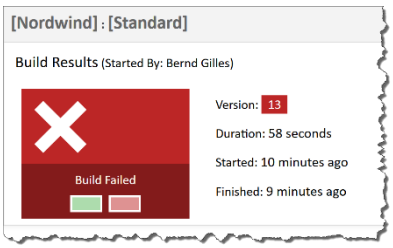


Der Build-Prozess läuft nun erwartungsgemäß auf einen Fehler und wird mit dem Status „Failed“ beendet. Die als Fortschrittsbalken ausgeführten Rechtecke im Status-Icon geben bereits Aufschluss darüber, dass der Fehler in Stage 2 aufgetreten ist.

Haben wir eine entsprechende Benachrichtigung eingerichtet, erhalten wir eine Fehlermeldung per Email.

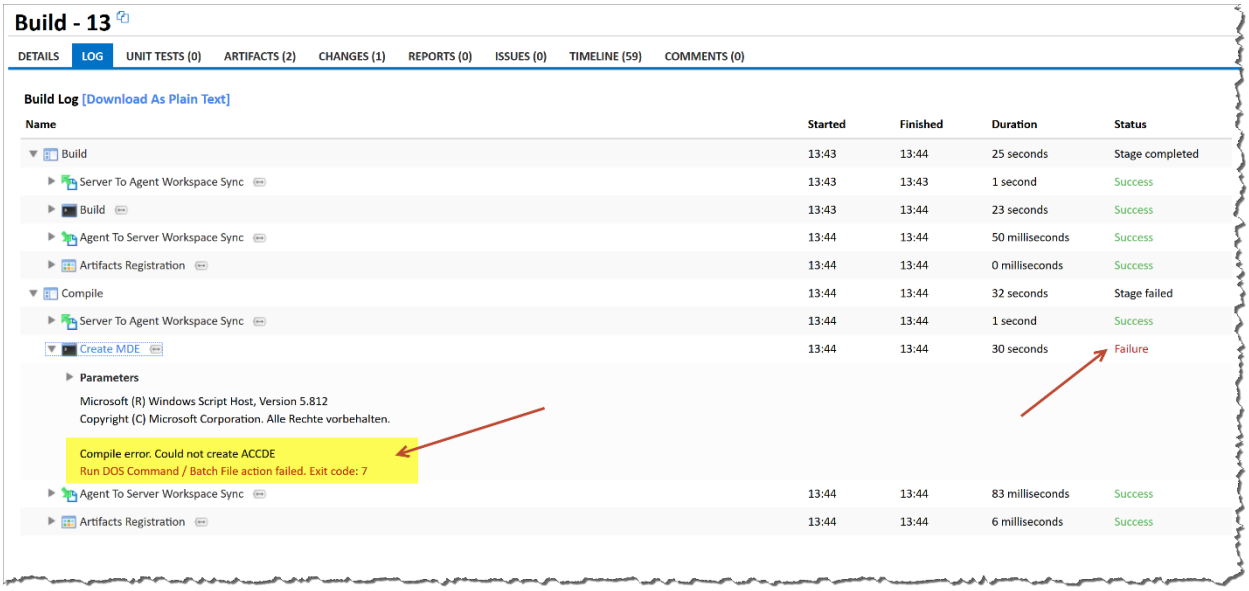


Leider erhalten wir an dieser Stelle keine detaillierte Information darüber, welcher Fehler aufgetreten ist.
Wir müssen dazu die Web-Oberfläche des CI-Servers aufrufen.



Durch einen Klick auf die Versionsnummer gelangen wir in die Detailansicht des fehlerhaften Build.

Um den aufgetretenen Fehler identifizieren zu können wechseln wir hier auf die Seite „Log“.



Wir erkennen, dass der Fehler in unserer zweiten Stage „Compile“ aufgetreten ist und finden dort die von unserem VB-Script ausgegebene Fehlermeldung.

CONTINUOUS-DELIVERY

Wir besitzen nun einen funktionsfähigen Continuous-Integration-Prozess.

Was liegt da näher, als diesen zu einem Continuous-Delivery-Prozess zu erweitern?

Ziel ist es, nicht nur den Build-Prozess zu automatisieren, sondern die dadurch erzeugte ACCDE-Datei auch gleich an die Endanwender (oder die Tester) zu verteilen.

Dies funktioniert allerdings nur, wenn der Start der Applikation über eine kleine Loader-Anwendung realisiert wird. Diese hat die Aufgabe, die Version der lokalen Access-Applikation zu ermitteln, um sie mit der aktuellen Versionsnummer zu vergleichen. Ist die lokale Applikation veraltet, soll vor dem Start die aktuellste Version von einer zentralen Stelle heruntergeladen und erst dann gestartet werden.

Dazu sind einige Vorarbeiten notwendig.

VORBEREITUNG DER APPLIKATION

Damit der Loader überhaupt in die Lage versetzt wird die Version der lokalen Applikation zu ermitteln, muss die Versionsnummer irgendwo in der Applikation abgelegt werden.

Dies kann über verschiedene Wege erreicht werden:

- Lokale Tabelle in der Applikation
- Datenbankeigenschaft
- etc.

Im Beispiel wird eine lokale Tabelle „Version“ in der Applikation angelegt, die nur über ein einziges Feld „Version“ enthält.

Außerdem muss ein Mechanismus implementiert werden, der nach dem Build-Prozess die neue aktuelle Versionsnummer in dieser Tabelle speichert.

Dazu wird im Beispiel ein Stückchen VBA-Code in Verbindung mit dem AutoExec-Makro verwendet.
Dieser Code wertet die Kommandozeilenparameter aus und speichert die dort ggf. übergebene Versionsnummer in der Tabelle.

Alternativ könnte man dies z.B. auch über ein VB-Script oder eine kleine Applikation realisieren.

```
Option Compare Database
Option Explicit

Sub CheckCommandLineParams()
Dim Params() As String
Dim SQL As String

 If Command$ <> "" Then
   Params = Split(Command$, "#")
   If Params(0) = "SaveVersion" Then
     SQL = "UPDATE Version SET VERSION = " & Params(1)
     CurrentDb.Execute SQL
     DoEvents
     DoCmd.Quit
   End If
 End If

End Sub
```

VERSIONSINFO

Damit überhaupt festgestellt werden kann welche Versionsnummer die aktuelle ist, sind wiederum verschiedene Wege denkbar:

- Zentrale Datenbank im Netz
- XML- oder Textdatei im Netz
- XML- oder Textdatei auf einem Webserver
- etc.

Im Beispiel verwenden wir eine XML-Datei auf einem Webserver.Da wir mit Continua-CI sowieso schon einen Webserver betreiben, bietet es sich hier an die XML-Datei auch dort abzulegen.

Das Root-Verzeichnis des Webservers befindet sich (bei Standardinstallation) im Verzeichnis:
`C:\Program Files\VSoft Technologies\ContinuaCI\Web\`

Der Inhalt dieser XML-Datei ist relativ trivial. Sie enthält lediglich die aktuelle Versionsnummer und einen Download-URL, den unsere Loader-Applikation benötigt.

```
1 <?xml version="1.0"?>
2 <application>
3   <version value="7" url="http://169.254.254.45:8080/downloads/Nordwind.accde" />
4 </application>
```

LOADER-APPLIKATION

Um vor dem Start der eigentlichen Access-Anwendung bei Bedarf die aktuelle Version laden zu können, wird eine kleine Loader-Applikation verwendet, die vier Aufgaben übernimmt:

- Feststellen der aktuellen Versionsnummer
- Vergleich mit der Versionsnummer der lokalen Anwendung
- Bei Bedarf: Download der aktuellen Version der Anwendung
- Starten der Anwendung

Idealerweise schreibt man sich eine solche Applikation z.B. in Delphi, C# oder VB.NET. Dazu können die jeweils kostenlosen Community-Versionen von Delphi oder Visual-Studio verwendet werden. Auch das Open-Source-Projekt „Lazarus“ mit dem Free-Pascal-Compiler kann hier durchaus eine Alternative sein.

Für den Einstieg verwenden wir im Beispiel kurzerhand eine Minimallösung in Access.

VERSIONSNUMMER FESTSTELLEN

Die erste Aufgabe des Loaders ist es, die aktuelle Versionsnummer unserer Access-Anwendung zu ermitteln.

Dazu liest der Loader die oben beschriebene XML-Datei per XMLHttp-Request.

```
1 Public Function GetCurrentVersionInfo(ByVal AUrl As String) As TVersionInfo
2 Dim XMLDocument As MSXML2.DOMDocument
3 Dim Node As IXMLDOMElement
4 Dim XMLText As String
5
6   XMLText = GetServerXML(AUrl)
7
8   Set XMLDocument = New MSXML2.DOMDocument
9   XMLDocument.loadXML XMLText
10
11  Set Node = XMLDocument.documentElement.selectSingleNode("version")
12
13  GetCurrentVersionInfo.VersionNumber = Node.getAttribute("value")
14  GetCurrentVersionInfo.DownloadUrl = Node.getAttribute("url")
15
16  Set Node = Nothing
17  Set XMLDocument = Nothing
18
19
20 End Function
21
22 Private Function GetServerXML(ByVal AUrl As String) As String
23 Dim ARequest As MSXML2.XMLHttp
24
25  Set ARequest = New MSXML2.XMLHttp
26
27  ARequest.Open "GET", AUrl & "?" & GetTickCount, False
28  ARequest.setRequestHeader "Content-Type", "text/xml"
29  ARequest.send
30
31  GetServerXML = ARequest.responseText
32
33  Set ARequest = Nothing
34
35 End Function
```

VERSIONSNUMMER DER ANWENDUNG

Als Nächstes muss der Loader die Versionsnummer der lokalen Anwendung ermitteln.

```
Private Function GetLocalVersion() As Integer
Dim db As DAO.Database
Dim rs As DAO.Recordset

  On Error GoTo ErrHandler

  Set db = OpenDatabase(CurrentProject.Path & "\Nordwind.accde")
  Set rs = db.OpenRecordset("Version", dbOpenForwardOnly, dbReadOnly)

  If Not rs.EOF Then
    GetLocalVersion = rs!Version
  Else
    GetLocalVersion = -1
  End If

  Set rs = Nothing
  Set db = Nothing

End Function
```

DOWNLOAD DER ANWENDUNG

Stellt der Loader nun fest, dass die lokale Access-Anwendung veraltet ist, wird per HTTP-Download die neue Version der Anwendung vom Server geladen.

```
Public Function DownloadFile(ByVal AUrl, AFilename As String) As Boolean
Dim ARequest As MSXML2.XMLHttp
Dim AStream As ADODB.Stream

  Set ARequest = New MSXML2.XMLHttp
  ARequest.Open "GET", AUrl & "?" & GetTickCount, False
  ARequest.send

  If ARequest.status = 200 Then
    Set AStream = New ADODB.Stream
    AStream.Open
    AStream.Type = adTypeBinary
    AStream.Write ARequest.responseBody
    AStream.SaveToFile AFilename, adSaveCreateOverWrite
    AStream.Close
    Set AStream = Nothing
    DownloadFile = FileExists(AFilename)
  End If

  Set ARequest = Nothing

End Function
```

Besondere Beachtung verdient hier der markierte Teil des Quellcodes.
Hier wird dem URL über GetTickCount ein eindeutiger Zahlenwert als zusätzlicher Parameter angehängt. Grund dafür ist der Cache des Browsers. Der würde nämlich ohne diesen Parameter möglicherweise davon ausgehen, dass die zu ladendende Datei sich noch im Cache befindet und die Datei dann aus dem Cache statt vom Server laden.

Dieser Trick wird beim Lesen der XML-Datei ebenfalls angewendet.

STARTEN DER ANWENDUNG

Das Starten der Anwendung ist nun mit wenigen Zeilen Code erledigt.

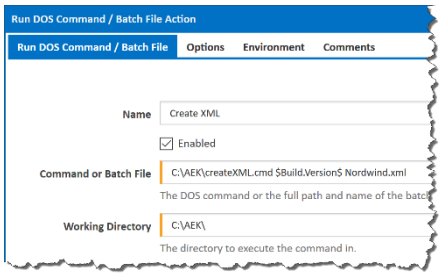
```
  Set acc = New Access.Application
  acc.OpenCurrentDatabase strDbName, False
```

NEUE STAGE: DEPLOY

Begeben wir uns nun daran, den CI-Prozess zu erweitern und zu einem Continuous-Delivery-Prozess auszubauen.

Wir bearbeiten die „Stages“ unserer Konfiguration und fügen mit „Add Stage“ eine neue Stage namens „Deploy“ hinzu.

XML-DATEI SCHREIBEN



Der erste Schritt besteht darin, die XML-Datei zu erzeugen.
Das übernimmt wieder eine kleine Batchdatei.

Wir wählen aus der Aktionsgruppe „Execution“ erneut die Aktion „Run DOS Command / Batchfile“.

Diese Batchdatei verarbeitet zwei Parameter:

- Versionsnummer des Build
  `$Build.Version$`
- Name der XML-Datei
  `Nordwind.xml`

```
1 @echo off
2 if exist %2 del %2
3 echo ^<?xml version="1.0"?^> > %2
4 echo ^<application^> >> %2
5 echo   ^<version value="%1" url="http://169.254.254.45:8080/downloads/Nordwind.accde" /^> >> %2
6 echo ^</application^> >> %2
```

XML-DATEI KOPIEREN

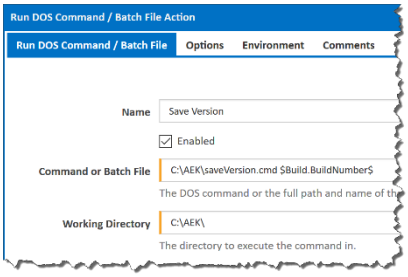
Die nächste Aktion kopiert die soeben erzeugte XML-Datei in das Root-Verzeichnis des Webservers.

Dazu wählen wir aus der Gruppe „File Operations“ die Aktion „Copy“

- **Name**
  Wir vergeben einen sinnvollen Namen.
- **Enabled**
  Natürlich wird die Aktion aktiviert.
- **Source Directory**
  Das Quellverzeichnis.
- **File Names / Patterns**
  Hier geben wir den Namen der zu kopierenden XML-Datei an.
- **Destination Directory**
  Ziel des Kopiervorgangs ist das Root-Verzeichnis des Webservers.
- **Overwrite existing files**
  Diese Option muss unbedingt markiert werden, damit der Vorgang nicht fehlschlägt falls die Datei im Ziel bereits vorhanden ist.

VERSIONSNUMMER SPEICHERN

Im nächsten Schritt kommt wieder eine Batchdatei zum Einsatz, die die Versionsnummer des aktuellen Build in die in Stage 2 erzeugte ACCDE-Datei „injiziert“.



Dazu wird die Versionenummer des Build über die Systemvariable `$Build.Version$` als Parameter an die Batchdatei übergeben.

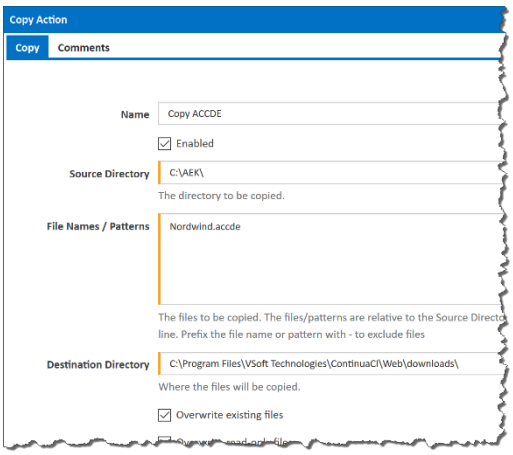
Die Batchdatei selbst ist reichlich trivial.
Sie startet lediglich MS-Access mit der Anwendung und übergibt im Kommandozeilenparameter den Befehl „SaveVersion“ und die Versionsnummer.
Unser kleines Stückchen VBA-Code übernimmt dann das Schreiben in die Tabelle.

```
0         10        20        30        40        50        60        70        80        90
1 @echo off
2 "C:\Program Files (x86)\...\MSACCESS.EXE" "C:\AEK\Nordwind.accdb" /cmd SaveVersion#%1
3
```

ANWENDUNG KOPIEREN

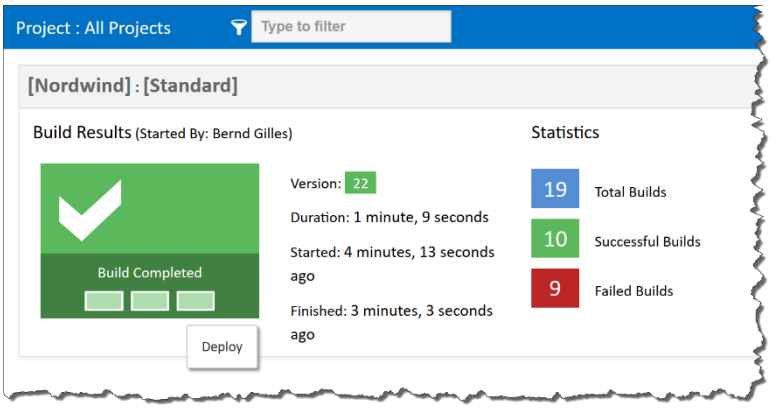
Zum guten Schluss fehlt nun lediglich das Kopieren der Anwendung in ein Verzeichnis auf dem Webserver.

Dazu kommt wieder die Aktion „Copy“ zum Einsatz.



- **Name**
  Wir vergeben einen sinnvollen Namen.
- **File Names / Patterns**
  Hier geben wir den Namen der ACCDE-Datei an.
- **Destination Directory**
  Ziel des Kopiervorgangs ist diesmal ein Unterverzeichnis des Web-Root.
- **Overwrite existing files**
  Wir achten natürlich auch hier darauf, dass die Option markiert ist.

Damit ist nun auch unser Continuous-Delivery-Prozess fertig.
Nachdem die Anwendung zunächst automatisch gebaut wurde, wird sie nun auch vollständig automatisiert zur Verfügung gestellt und im Bedarfsfall beim Anwender heruntergeladen.



FAZIT

Es bedarf sicher etwas Einarbeitungszeit, aber mit verhältnismäßig geringem Aufwand lässt sich Continua-CI sinnvoll und gewinnbringend einsetzen.

OASIS-SVN

OASIS-SVN ist selbst keine Versions- oder Quellcodeverwaltung, sondern bietet als COM-Add-In komfortablen Zugriff auf Funktionen, um eine Access-Anwendung mit einer Quellcodeverwaltung wie Subversion oder Git verwalten zu können.

TORTOISE-SVN

TortoiseSVN ist eine Shell-Erweiterung für den Windows-Explorer und bietet direkt aus dem Kontextmenü Funktionen zur Verwaltung von Dateien mit Subversion.

VISUAL-SVN-SERVER

Visual-SVN-Server ist ein vollständiger Subversion-Server für Windows und lässt sich in kürzester Zeit mit wenigen Klicks installieren

Nach der Installation bietet das System eine sehr einfache Verwaltungsoberfläche, mit der sich Repositories und Benutzeraccounts verwalten lassen.

Das System installiert einen eigenen Apache-Webserver und lässt sich sowohl über HTTP als auch über HTTPS ansprechen.

PREISE

OASIS-SVN

Private-Licence	Rein private Nutzung für einen einzelnen Entwickler	39,- €
Business-Licence	Kommerzielle Nutzung für einen einzelnen Entwickler	69,- €
Enterprise-Licence	Kommerzielle Nutzung für beliebig viele Entwickler eines Unternehmens	399,- €

Die Demoversion ist kostenlos. Hiermit können jedoch lediglich 50 Objekte am Stück verarbeitet werden.

TORTOISE-SVN

TortoiseSVN ist Open-Source und kostenlos.

VISUAL-SVN-SERVER

Visual-SVN-Server ist in der Standard-Edition auch für die kommerzielle Nutzung kostenlos.

Die Enterprise-Edition kostet 950,- $ je Server und bietet z.B. Backup-Jobs, Remotezugriff und AD-Integration.

CONTINUA-CI

Continua-CI ist für 1 Agenten und einen „Concurrent Build“ kostenlos.

Jeder weitere „Concurrent Build“ kostet 199,- $ und bietet gleichzeitig eine beliebige Anzahl von Agenten.

TIPPS

SUBVERSION

Wer in seinem Netzwerk ein Synology-NAS betreibt, benötigt keinen separaten SVN Server.

Das Paket wird ganz normal über die Paketverwaltung heruntergeladen, installiert und konfiguriert.

LINKS

OASIS-SVN
https://dev2dev.de

TortoiseSVN
https://tortoisesvn.net/

Visual-SVN-Server
https://www.visualsvn.com/server/

Synology-SVN
https://www.synology.com/de-de/dsm/packages/SVN

Continua-CI
https://www.finalbuilder.com/continua-ci