Erstellen eines Self-Service Framework für SQL Server Administratoren

Kathegorie
SQL Server

Aaron
Priesterroth

Das Leben eines DBA ist wirklich kein leichtes Spiel.
Wer kennt es nicht: von "Kannst du mal eben die Abfrage XYZ ausführen und mir die Ergebnisse als Excel-Dokument zukommen lassen?" bis "Hier, alle diese Skripte müssen in die Vor-Produktion!".

Alles keine großen Herausforderungen für einen DBA – aber sie kosten Zeit und / oder können uns schnell aus dem Konzept bringen.
Um solche allgemeinen Aufgaben in Zukunft vermeiden zu können, bietet es sich an, einen Self-Service Mechanismus einzurichten, der ohne viel Zutun des DBAs kleinere, evtl. öfter wiederholte Aufgaben ausführen kann, wie etwa Abfragen ausführen, Skripts veröffentlichen oder Umgebungen aktualisieren, ohne bzw. mit nur geringem Aufwand für den DBA.

Wie man einen solchen Self-Service Mechanismus einrichten kann, möchte ich in den folgenden Abschnitten erklären. Basis ist ein Mechanismus, mit dessen Hilfe ein beliebiger Benutzer ein SQL-Skript in eine Umgebung, die nicht der Produktionsumgebung entspricht, veröffentlichen kann.
Um einen solchen Self-Service zu implementieren, muss als Erstes ein Protokoll für die Kommunikation definiert werden, damit eine Anfrage für einen Service auch genau so verarbeitet wird, wie wir es erwarten würden. Das bedeutet, dass das Protokoll darüber bescheid wissen sollte, welcher Benutzer was für eine Aktion ausführen möchte. Diese Informationen können dann zusätzlich verwendet werden, um solche Anfragen von beliebigen Benutzern entweder freizugeben oder zu verweigern.

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

Das Protokoll

Beim Aufbau des Protokolls sind wir sehr flexibel. Wir können es genau so gestalten, dass es alle Informationen, die wir als wichtig erachten, enthält und das Protokollformat verwenden, das uns am meisten zusagt. Egal ob JSON, XML, csx, ini file oder vielleicht auch einfach eine Tabelle einer Datenbank.

Aber bevor wir anfangen unser Protokoll zu erstellen, müssen wir uns darauf festlegen, was wir eigentlich erreichen wollen. In unserem Fall formuliere ich die Anforderung an unser Protokoll folgendermaßen:
Wenn ein Benutzer des Systems (z.B. ein Entwickler, QA-Tester, o.Ä.) ein SQL-Skript in einer Nicht-Produktionsumgebung ausführen möchte, sollte er dafür nicht um die Hilfe eines DBAs bitten, sondern die Möglichkeit über einen Self-Service nutzen, indem er die Informationen über die Lage der Skript-Datei (der Pfad) und der Ziel-Umgebung an den Self-Service weiter gibt.

Das Wichtigste für unser Protokoll ist, dass es gut verständlich ist für jeden, der es evtl. benutzen muss. Aus diesem Grund beschränken wir uns auf die folgenden vier Eigenschaften, die unser Protokoll besitzt:

```
# Mit einem #-Zeichen am Anfang einer Zeile kann ein Kommentar eingefügt werden
# Ein Benutzer muss die Informationen rechts vom "=" ausfüllen
[ScriptPath]=[pfad_zur_skript_datei]
[requestor]=[email_adresse_des_benutzers]
[Target]=[name_der_umgebung_oder_sql_instance_name]
[Tag]=[ein_tag_zur_identifikation_dieser_anfrage]
```

Für jede dieser vier Eigenschaften muss vom Benutzer eine Angabe bereitgestellt werden.

Der Arbeitsablauf des Self-Service

Der Arbeitsablauf für den Self-Service ist fast genau so simpel wie der Aufbau unseres Protokolls. Er besteht aus den folgenden drei Arbeitsschritten:

1. Ein Benutzer möchte ein Skript veröffentlichen und bereitet sich darauf vor. Er speichert sein Skript in einem Ordner auf einem geteilten Laufwerk (engl. shared drive).
2. Als Nächstes muss der Benutzer das Protokoll ausfüllen, indem er seine Informationen in eine Datei auf dem geteilten Laufwerk schreibt, z.B. mit dem Namen **DeployConfig.txt** (bzw. **\\shared_drive\request\DeploymentConfig.txt**)
3. Diese **DeployConfig.txt** Datei kann dann von einem PowerShell-Skript, das z.B. durch einen SQL Server Agent Job ausgeführt wird, verarbeitet werden.

Der Fokus unseres Arbeitsablaufes liegt klar auf dem dritten Schritt, also der Verarbeitung durch das PS-Skript. In den folgenden weiteren vier Schritten wollen wir nun die Logik dieses Verarbeitenden PS-Skripts genauer betrachten:

1. Überprüfe, ob alle benötigten Informationen in der **DeployConfig.txt** Datei vorhanden sind. Falls nicht, beende den Vorgang und gib eine Warnung aus. Falls doch, mach mit dem nächsten Schritt weiter.
2. Überprüfe anhand des angegebenen **[Tag]** aus der **DeployConfig.txt**, ob diese Konfiguration bereits ausgeführt wurde. Falls ja, verschicke eine Email an den Benutzer. Falls nicht, mach mit dem nächsten Schritt weiter.
3. Überprüfe, ob eine Datei zum angegebenen **[ScriptPath]** existiert und der angegebene **[TargetServer]** erreicht werden kann. Falls nicht, verschicke eine Email an den Benutzer mit der Aufforderung die Konfiguration zu korrigieren. Falls ja, mach mit dem nächsten Schritt weiter.
4. Iteriere über alle Dateien, die im Ordner von **[ScriptPath]** enthalten sind und ordne sie basierend auf ihrem vollen Namen in alphanumerischer Reihenfolge. Führe das gewünschte Skript auf dem **[TargetServer]** aus und registriere die Veröffentlichung in

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

der **[DeploymentHistory]** Tabelle. Falls es zu einem Fehler kommt, stoppe die Verarbeitung und schicke eine Email an den Benutzer mit der Nachricht des Fehlers. Ansonsten beende die Verarbeitung und schicke anschließend eine Email an den Benutzer mit der Nachricht, dass seine Aufgabe beendet wurde.

Der Prozess – Voraussetzungen

Damit der unten stehende Code verwendet werden kann, müssen wir vorher einige Annahmen über unser System formulieren:

1. Als **[TargetServer]** wird immer ein SQL Server Instanz Name verwendet.
2. Sollen mehrere Skripte auf einmal ausgeführt werden, müssen sich die Skripte in ihrer Ausführungsreihenfolge nach dem Namen ordnen lassen.
3. Da es keine Parallelisierung in unserem System gibt, kann immer nur ein Benutzer nach dem anderen bedient werden. Erst nachdem eine Anfrage verarbeitet wurde, kann die Nächste begonnen werden.

Der Code für die Engine

Als Grundlage für unser System wird eine Datenbank mit dem Namen **[DBA]** benötigt. Auf dieser Datenbank müssen Tabellen zum Loggen der ausgeführten Anfragen existieren. Mit dem folgenden Skript können die benötigten Tabellen auf einer existierenden Datenbank **[DBA]** erstellt werden:

```
USE [DBA]
GO

drop table if exists [dbo].[DeploymentHistory], [dbo].[DeploymentConfig]; --SQL
2016 and later syntax
go

-- log the details of each Deployment request
CREATE TABLE [dbo].[DeploymentConfig](
   [InitFile] [varchar](2000) NOT NULL,
   [InitFileDate] [datetime] NOT NULL,
   [TargetServer] [varchar](100) NOT NULL,
   [ScriptFolder] [varchar](600) NULL,
   [Requestor] [varchar](100) NOT NULL,
   [Tag] [varchar](150) not NULL,
   [id] [int] IDENTITY(1,1) primary key,
)

GO
-- log the deployment result of each script
CREATE TABLE [dbo].[DeploymentHistory](
   [FullPath] [varchar](800) NULL,
   [Tag] [varchar](150) NULL,
   [TargetServer] [varchar](60) NULL,
   [Status] [varchar](20) NULL,
   [Message] [varchar](6000) NULL,
   [DeployDate] [datetime] NULL,
   [ConfigID] [int] NULL,
   [id] [int] IDENTITY primary key
);

GO
ALTER TABLE [dbo].[DeploymentHistory] ADD  DEFAULT ('not started') FOR [Status];
ALTER TABLE [dbo].[DeploymentHistory] ADD  DEFAULT (getdate()) FOR [DeployDate];
ALTER TABLE [dbo].[DeploymentHistory]  WITH NOCHECK ADD FOREIGN KEY([ConfigID])
REFERENCES [dbo].[DeploymentConfig] ([id]);
GO
```


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

Als Nächstes folgt das PowerShell-Skript. Die Arbeitsweise lässt sich in drei simplen Schritten zusammenfassen:

1. Lese die **DeployConfig.txt** Datei und validiere die Eingabe. Falls erfolgreich, schreibe die Konfiguration der Anfrage in die **DeploymentConfig** Tabelle.
2. Veröffentliche das Skript und schreibe die Ergebnisse der Ausführung in eine Log-Datei und in die **DeploymentHistory** Tabelle.
3. Benachrichtige den Benutzer (und den DBA falls notwendig) über die Ergebnisse der Verarbeitung.

Im folgenden Skript müssen evtl. die angegebenen Werte mit deinen eigenen ausgetauscht werden. Darunter fallen vor allem:

- $log_file
- $central_svr
- $IniFile
- @mycompany.com

```
#Function: this is to deploy scripts based on one config file
#$log_file, $central_svr and $IniFile, also @mycompany.com need to be changed

import-module sqlserver;
$log_file ="d:\dba\log_$((get-date).ToString('yyyyMMdd_HHmm')).txt" # deployment
log
$central_svr = 'MyServer'; # from this central server, the deployment will be
made
$IniFile = '\\<sharefolder>\DeployConfig.txt'; #requestor prepares this deployment
request file

$deploy_status = 'SUCCEEDED';

if (test-path -Path $log_file)
{ remove-item -Path $log_file; }
if (-not (test-path -Path $IniFile))
{ throw "[$IniFile] does not exist, please double check";}

$dtbl = new-object System.Data.DataTable;
$dc = new-object System.Data.DataColumn ('InitFile', [System.string]);
$dtbl.Columns.add($dc);

$dc = new-object System.Data.DataColumn ('InitFileDate', [System.DateTime]);
$dtbl.Columns.add($dc);

$dc = new-object System.Data.DataColumn ('TargetServer', [System.string])
$dtbl.Columns.add($dc);

$dc = new-object System.Data.DataColumn ('ScriptPath', [System.string])
$dtbl.Columns.add($dc);

$dc = new-object System.Data.DataColumn ('Requestor', [System.string]);
$dtbl.Columns.add($dc);

$dc = new-object System.Data.DataColumn ('Tag', [System.string]);
$dtbl.Columns.add($dc);

#read the DeployConfig.txt file
$ini = @{};
switch -Regex -file $IniFile
{
    "\[(.+)\]=\[(.+)\]"
    {
        $name, $value = $matches[1..2];
        $ini[$name]=$value;
    }
}

# make it mandatory the [Requestor] has a valid email address
if ($ini.Requestor -notmatch '@mycompany.com' -or $ini.tag -eq '' -or $ini.
ScriptPath -eq 'the sql script path in a shared folder' -or $ini.TargetServer -eq
'sql server name')
```


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

```
{
   write-host 'nothing to run';
   return;
}

$c = get-content -Path $IniFile | ? { $_ -match '^\s?\[.+'}; # excluding comment
and only extracting the key data
$FileDate = dir -Path $IniFile | Select-Object -Property LastWriteTime;

$c | out-file -FilePath $log_file -Encoding ascii -Append;

"`r`n`r`nImplementation starts at$(get-date)..`r`n `r`nDeployment Starts" | out-
file -FilePath $log_file -Encoding ascii -Append;

# when an error is encountered, the script will stop, you can rerun the whole
script, and it will skip the failed script and contine to next one

[string]$requestor = $ini.Requestor
[string]$ScriptPath = $ini.ScriptPath;
[string]$tag = $ini.Tag;

$target_server = $ini.TargetServer;

if (test-path $ScriptPath)
{
    $script_folder = $ScriptPath
}
else
{
    invoke-sqlcmd -ServerInstance . -Database msdb -Query "exec dbo.sp_send_
dbmail @recipients='$($requestor)',@copy_recipients='DBATeam@mycompany.com', @
subject='Cannot find Script folder', @Body='[$($ScriptPath)] is invalid'";
    throw "Invalid Folder [$ScriptPath]"
}
```

```
#check whether the $Target_server is correct
try
{
  invoke-sqlcmd -ServerInstance $target_server -Database master -query "select
getdate()" | Out-Null
}
catch
{
    invoke-sqlcmd -ServerInstance . -Database msdb -Query "exec dbo.sp_send_
dbmail @recipients='$($requestor)',@copy_recipients='DBATeam@mycompany.com', @
subject='Cannot connect to$($target_server)', @Body='The server$($target_server)
cannot be accessed'";

  throw "The server$target_server cannot be accessed";
}

#check whether the $Tag is already there, if so, we need to change it

    $qry = @"
   if exists (select * from dbo.DeploymentHistory where Tag='$($Tag)')
       select isTagInside = 1;
   else
       select isTagInside = 0;
"@

    $result = invoke-sqlcmd -ServerInstance $central_svr -Database dba -Query
$qry -OutputAs DataRows;
if ($result.isTagInside -eq 0)
{
    #we save the DeploymentConfig.txt
    $r = $dtbl.NewRow();
    $r.InitFile = $c -join "`r`n";
    $r.InitFileDate = $FileDate.LastWriteTime ;

    $r.TargetServer = $ini.TargetServer;
    $r.ScriptPath = $ini.ScriptPath;
```


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

```
    $r.Requestor = $ini.Requestor;
    $r.tag = $ini.tag;
    $dtbl.Rows.Add($r);

    Write-SqlTableData -ServerInstance $central_svr -DatabaseName dba -SchemaName
dbo -TableName DeploymentConfig -InputData $dtbl;
}

[string]$deployment_name = $ini.tag; # choose your own name if needed,my pattern
is: Application_Date_VerNum

$continue = 'N'; #adding another layer of protection in case the prod server is
used...
IF ($target_server -in ('prod01', 'prod02', 'prod03')) #adding your prod list
here
{
   $continue = 'n' ;
   throw "we do not allow to deploy to production [$target_server] at this time";
}
else
{ $continue ='y';}

if ($continue -ne 'y')
{ throw "you are going to deploy to prod, not continuing";}

$dt = New-Object System.Data.DataTable;
$col = New-Object System.Data.DataColumn('FullPath', [system.string]);
$dt.Columns.Add($col);
$col = New-Object System.Data.DataColumn('Tag', [system.string]);
$dt.Columns.add($col);
$col = New-Object System.Data.DataColumn('TargetServer', [system.string]);
$dt.Columns.add($col);

dir *.sql -Path $Scriptpath -Recurse -File  |
Sort-Object { [regex]::replace($_.FullName, '\d+', { $args[0].value.padleft(10,
'0')})}  |

ForEach-Object {
    $r = $dt.NewRow();
    $r.FullPath = $_.FullName;
    $r.Tag = $deployment_name;
    $r.TargetServer = $target_server;
    $dt.Rows.add($r);  }

#check whether we need to populate the table again
$qry = @"
if exists (select * from dbo.DeploymentHistory where Tag='$($deployment_name)')
   select isRunBefore = 1;
else
   select isRunBefore = 0;
"@

$result = invoke-sqlcmd -ServerInstance $central_svr -Database dba -Query $qry
-OutputAs DataRows;

if ($result.isRunBefore -eq 0) # the deployment never run before
{
    Write-SqlTableData -ServerInstance $central_svr -DatabaseName dba -SchemaName
dbo -TableName DeploymentHistory -InputData $dt;
}

$qry = @"
select FullPath, id, TargetServer, Tag, [Status] from dbo.DeploymentHistory
where Tag = '$($deployment_name)' and [Status] = 'not started' --'success'
order by id asc
"@;

$rslt = Invoke-Sqlcmd -ServerInstance $central_svr -Database dba -Query $qry
-OutputAs DataRows;

foreach ($dr in $rslt)
{
    try
```

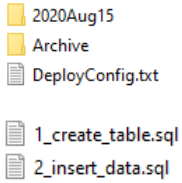

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
    {
        write-host "Processing [$($dr.FullPath)] with id=$($dr.id)"
-ForegroundColor Green;
       "Processing [$($dr.FullPath)] with id=$($dr.id)" | Out-File -FilePath
$log_file -Encoding ascii -Append
        [string]$pth = $dr.FullPath;
        invoke-sqlcmd -ServerInstance $dr.TargetServer -Database master
-InputFile $dr.FullPath  -QueryTimeout 7200 -ConnectionTimeout 7200 -ea Stop ;

        [string]$myqry = "update dbo.DeploymentHistory set [Status]='Success'
where id =$($dr.id);"
        invoke-sqlcmd -ServerInstance $central_svr -Database dba -Query $myqry;
    }
    catch
    {
        $e = $error[0].Exception.Message;
        $e = $e.replace("'", '"');
       # [string]$myqry = "update dbo.DeploymentHistory set [Status]='Error',
[Message]='$($e)' where id = $($dr.id);"
        if ($e.Length -gt 6000)
        { $e = $e.Substring(1, 6000);}
        [string]$myqry ="update dbo.DeploymentHistory set [Status]='Error',
[Message]='" + $e + "' where id =$($dr.id);"
        write-host "Error found on id=$($dr.id) with message =`r`n [$e]";
        "`r`nError occurred `r`n`r`n$($e)"| out-file -filepath $log_file -Encoding
ascii -Append;

        $deploy_status = 'FAILED';
        invoke-sqlcmd -ServerInstance $central_svr -Database dba -Query $myqry
-QueryTimeout 7200 -ConnectionTimeout 7200;
        write-host "error found, plese get out of here";
        break;
    }
}

$qry = @"
set nocount on;
UPDATE h set ConfigID = c.id
from dba.dbo.DeploymentHistory h
inner join dba.dbo.DeploymentConfig c
on h.Tag = c.Tag
where h.ConfigID is null;

exec msdb.dbo.sp_send_dbmail @recipients ='$($requestor)',@copy_
recipients='DBATeam@mycompany.com', @subject='Deployment of [$($deployment_name)]
on [$Target_server]$($deploy_status)', @body='please verify and let DBA know if
there is any issue', @file_attachments='$($log_file)';
"@;

invoke-sqlcmd -ServerInstance $central_svr -Database DBA -Query $qry;

#move the origina DeployConfig to an archive folder for later verification
Move-Item -Path $IniFile -Destination "\\<share_folder>\DeployConfig_Archive\
DeployConfig_$((get-date).tostring('yyyyMMdd_hhmm')).txt";

$txt = @"
#config file, please fill the [] on the equation right side WITHOUT any quotes
#[TargetServer] should be the sql instance name, such as [MyQA_1], [PreProd_2]
etc
#[ScriptPath] is where you put the sql script, such as [\\<share_folder>\
Deployment\2020Aug20_QA\]
#[Requestor] should be the requestor's email, so once the deployment is done, an
notification email will be sent out
#[Tag] this is more like a deployment name, can be anything to indicate the
deployment, for example [QA_V1.2.3]

[TargetServer]=[sql server name]
[ScriptPath]=[the sql script path in a shared folder]
[Requestor]=[your email addr]
[Tag]=[]
"@;
#re-generate the DeployConfig.txt
$txt | out-file -FilePath $IniFile  -Encoding ascii;
```


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

Demonstration

Für die Ausführung benötigen wir eine Ordner-Struktur. Diese ist recht simpel aufgebaut und befindet sich an einer beliebigen Stelle. In unserem Beispiel hier betrachten wir den Ordner mit dem Pfad "**D:\Deployment**". Innerhalb dieses Ordners benötigen wir einen Ordner für die Skripte, die ausgeführt werden sollen. In unserem Beispiel ist die Ordnerstruktur wie folgt:



Der Ordnern **2020Aug15**, in den wir die verarbeiteten Anfragen legen, enthält zwei beispielhafte SQL-Skripte. Diese haben den folgenden Inhalt:

```
-- 1_create_table.sql
use DBA;
go

drop table if exists dbo.tblTest;
go

create table dbo.tblTest(id int identity, a varchar(120));
go

-- 2_insert_data.sql
use DBA;
go

insert into dbo.tblTest(a)
values('hello world 1'),('hello world 2')
,('hello world 3'),('hello world 4');
go
```

Die **DeployConfig.txt** Datei erweitern wir mit einer Erklärung für die Benutzer:

```
# Konfigurationsdatei! Bitte die [] auf der rechten Seite ohne "" ausfüllen.
# [TargetServer] Der Instanz-Name des Ziel-Servers
# [ScriptPath] Der Ordner der die SQL-Skripte enthält, die ausgeführt werden
sollen
# [Requestor] Die Email des Bentuzers der die Anfrage stellt
# [Tag] Eine Bezeichnung für die Anfrage
[TargetServer]=[sql server name]
[ScriptPath]=[the sql script path in a shared folder]
[Requestor]=[your email addr]
[Tag]=[tag]
```

Und eine zugehörige, beispielhafte Konfiguration könnte wie folgt aussehen:

```
# Konfigurationsdatei! Bitte die [] auf der rechten Seite ohne "" ausfüllen.
# [TargetServer] Der Instanz-Name des Ziel-Servers
# [ScriptPath] Der Ordner der die SQL-Skripte enthält, die ausgeführt werden
sollen
# [Requestor] Die Email des Bentuzers der die Anfrage stellt
# [Tag] Eine Bezeichnung für die Anfrage
[TargetServer]=[localhost\sql2016]
[ScriptPath]=[d:\deploy\2020Aug15\]
[Requestor]=[aaron@madafa.de]
[Tag]=[Test]
```

Wenn wir das Skript jetzt ausführen, können wir direkt in unsere Datenbank gehen und uns selbst von den Ergebnissen überzeugen. Als Erstes werfen wir einen Blick in die **DeploymentHistory** und **DeploymentConfig** Tabellen:

	FullPath	Tag	TargetServer	Status	Message	DeployDate	ConfigID	id
1	D:\Deploy\2020Aug15\1_create_table.sql	Test	localhost\sql2016	Success	NULL	2020-09-22 14:36:15.633	1	1
2	D:\Deploy\2020Aug15\2_insert_data.sql	Test	localhost\sql2016	Success	NULL	2020-09-22 14:36:17.743	1	2

	InitFile	InitFileDate	TargetServer	ScriptFolder	Requestor	Tag	id
1	[TargetServer]=[localhost\sql2016] [ScriptPath]...	2020-09-22 14:35:44.207	localhost\sql2016	D:\Deploy\2020Aug15\	aaron@madafa.de	Test	1

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

Wir können sehen, dass unsere Anfrage erkannt wurde. Durch das Überprüfen der Tabelle, die durch die ausgeführten Skripte erstellt wurde, können wir uns noch von der korrekten Ausführung der Anfrage überzeugen:

	id	a
1	1	hello world 1
2	2	hello world 2
3	3	hello world 3
4	4	hello world 4

...und die E-Mail ist auch noch angekommen!

Also sind wir schon fertig? Fast! Optional kann man jetzt noch einen SQL Server Agent Job dazu verwenden, das PowerShell-Skript regelmäßig (z.B. alle 5 Minuten) auszuführen. Dabei muss man darauf achten, einen Step vom Typen Operating system (CMDExec) mit dem folgenden Command zu konfigurieren:

```
powershell.exe -nologo -noprofile -f "path_to_ps_script"
```

Und wie geht's jetzt weiter?

Am besten wirst Du mit dem ganzen System erst mal warm und schaust Dir die vielfältigen Tools an. Als Nächstes könntest Du überlegen, an welchen Stellen das System, das wir hier besprochen haben, bezüglich Deiner Bedürfnisse verbessert oder erweitert werden könnte. In vielen Firmen gibt es einige DBA Aufgaben, die mit Leichtigkeit über eine solche Automatisierung durchgeführt werden könnten. Damit gestaltet sich der Arbeitsalltag als DBA angenehmer und es bleibt noch genug Zeit für die wichtigen Dinge.

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