SPECIALIST ARTICLE

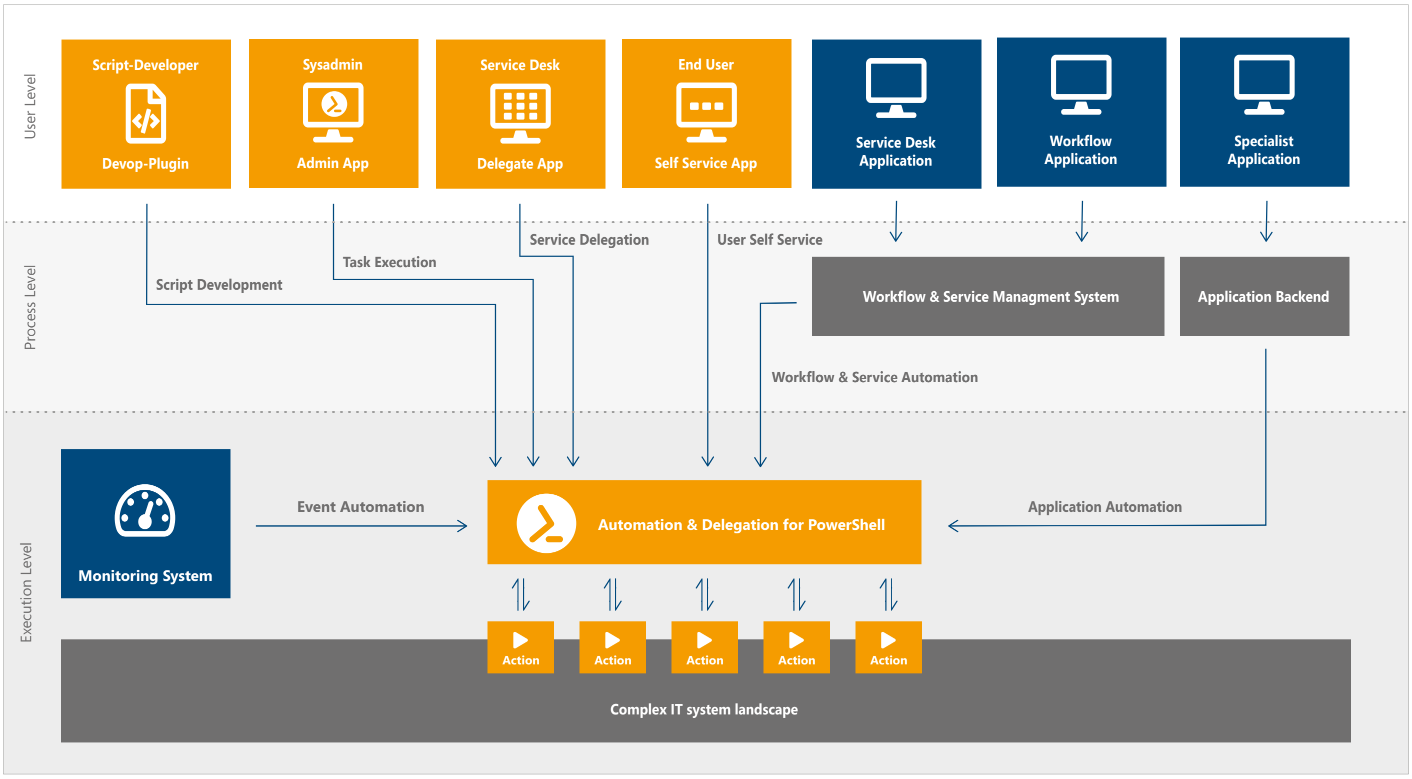
Automation with PowerShell

BY FRANK KRESSE

Administrators spend a large part of their work on recurring use cases. A lot of time can be saved by automating tasks and delegating tasks. The workshop describes the challenges and experiences of using PowerShell, and uses an example to explain how administrators go from cmdlet to script to process automation. We also discuss a possible security architecture for PowerShell in automated IT processes.

An essential part of any digitization strategy is the automation of repetitive tasks, routine processes, and even entire business processes, including the automatic response to repetitive events. IT is therefore intervening more deeply than ever before in the core business of every company. IT operations themselves are also in the focus of digitization and automation. Nevertheless, the degree of automation often lags far behind the possibilities.

A closer look at automation in various areas of the IT infrastructure reveals a mixed picture: In the past, there was a strong focus on automation solutions for software distribution. This subarea is now established in many companies. The second wave, currently still ongoing, concerns the automation of system monitoring with monitoring systems. One disadvantage, however, remains: the system still has to react manually to the reported events. When it comes to IT service management, ticket systems have been widely introduced. In Contrast, in the arena of service automation, including user self-services, there are various blank spots on the automation map.



TWO AUTOMATION STRATEGIES

There are two fundamentally different approaches to designing an automation strategy: **Top-down or bottom-up**. The top-down approach focuses on solutions and products for the full automation of IT processes. This approach is often driven in a very visionary way and often leaves out operational requirements and framework conditions. In practice, this often leads to very large, complex and escalating projects. These often address advanced stages of a transformation without having already mastered the initial stages.

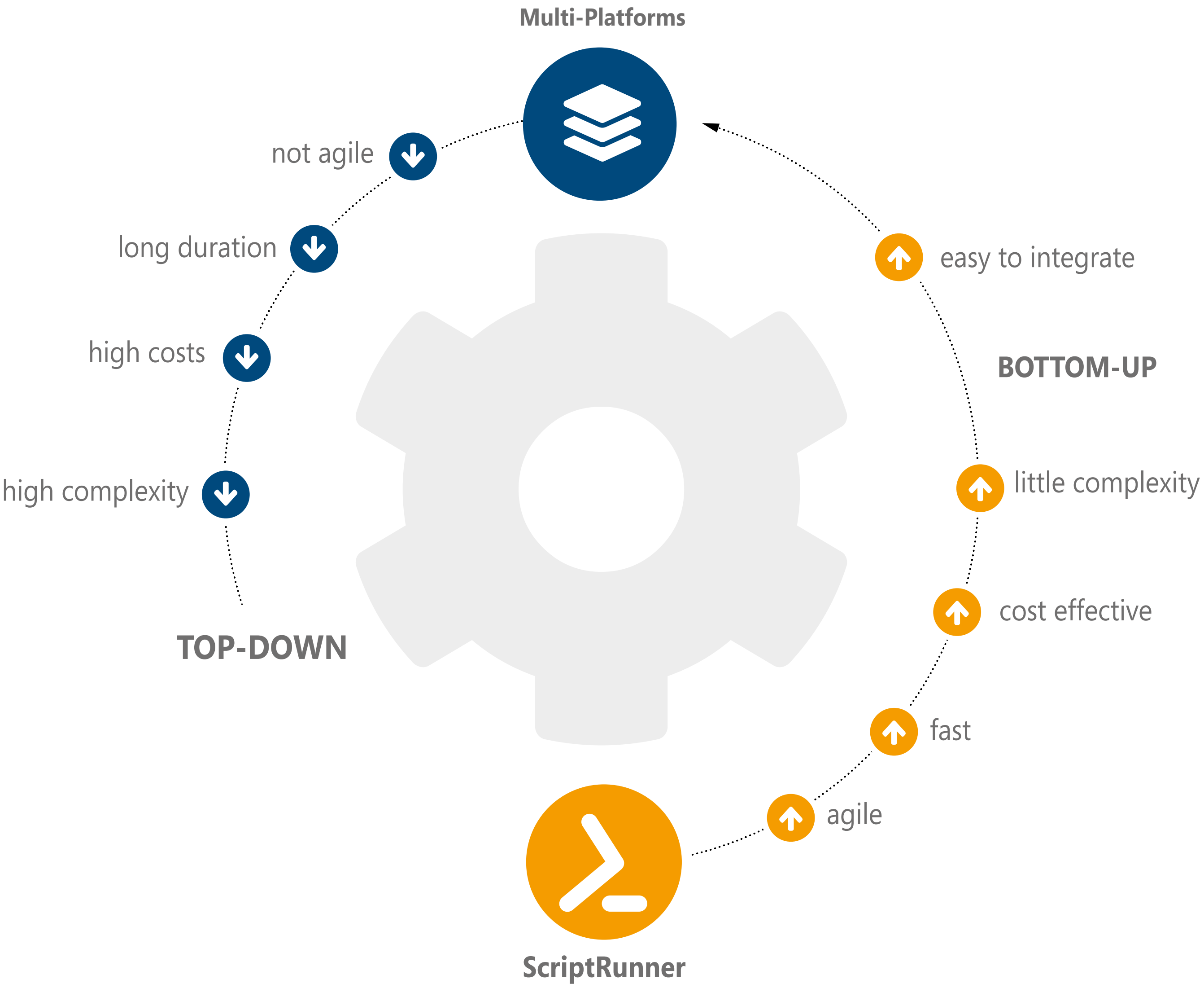
Typical for the top-down approach are:

- Few and very late results for the actual operational needs from day-to-day operations
- Very high costs and long running times, among other things due to the enormous complexity and massive scope
- Lack of agility with often poor cost-benefit ratio
- Additional high workload of the IT teams as well as many necessary new skills and at the same time a lot of specialist know-how
- High risk potential in the event of missed targets or failure of the project

The bottom-up approach, on the other hand, is based exclusively on the operational requirements and focal points of IT operations. It focuses on rapid improvements with an agile approach. With bottom-up strategies companies opt for a pragmatic approach, simpler platforms and clearer projects.

Therefore, the bottom-up approach usually shows the following picture:

- Fast results in daily operation
- Distributed and lower costs, significantly reduced complexity and a manageable scope of work in the individual stages of automation
- High agility with very good cost-benefit ratio
- Short project runtimes with few dependencies in each case
- Manageable workload and gradual development of skills and experience in the team
- More manageable risk on the way to a fully automated IT infrastructure



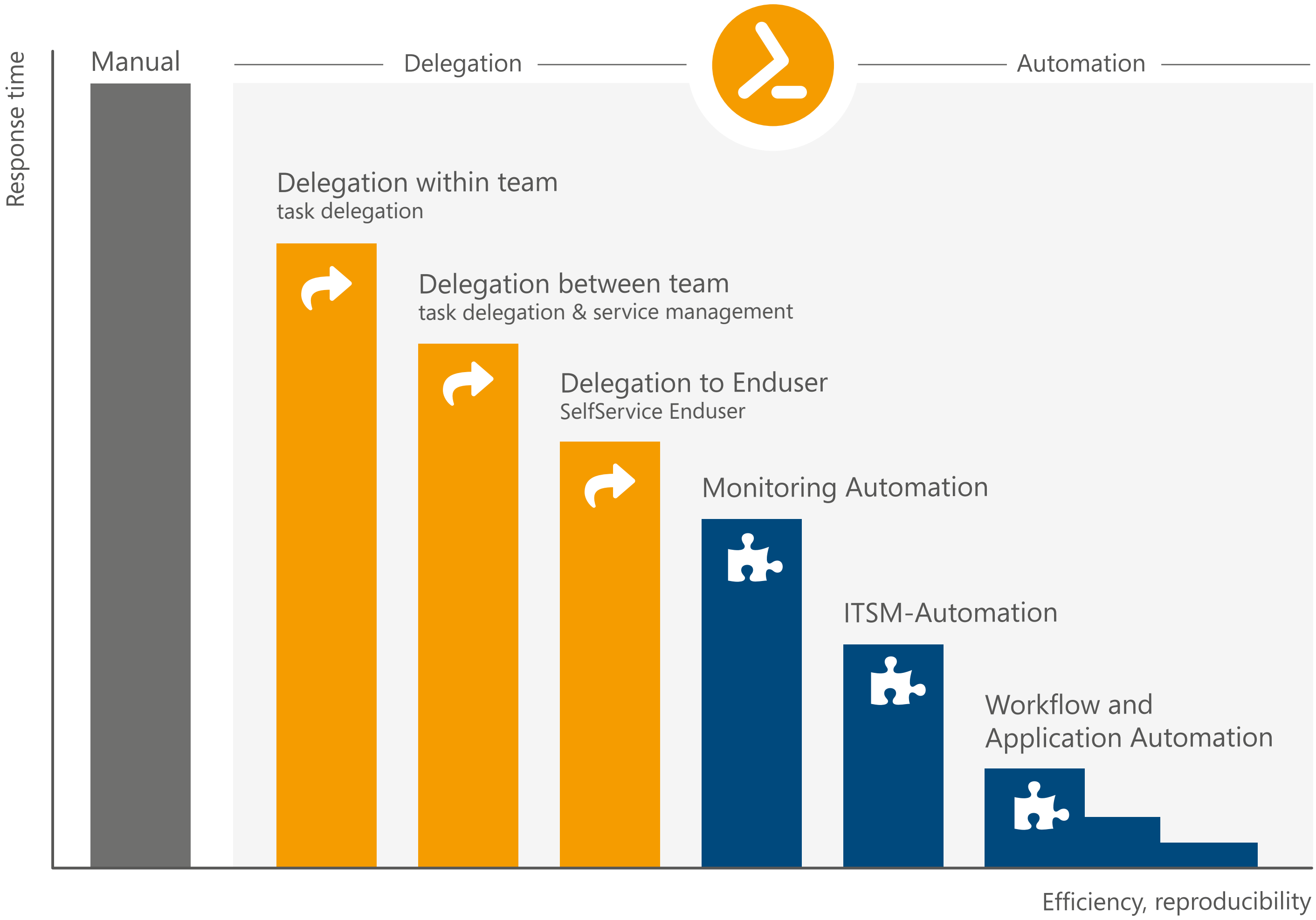
Both strategies ultimately lead to an overall automation solution, but in different ways and with very different intermediate results. While the bottom-up approach delivers fast results for daily tasks, the top-down approach first tries to define a comprehensive solution for the complete process level. The main advantage of the bottom-up strategy is that it opens up the control and process level step by step after the individual tasks have already been successfully automated. From the point of view of an agile cycle of designing, implementing, learning and improving, this is much easier to implement.

POWERSHELL AS THE BASIS FOR IT AUTOMATION

The evolution of PowerShell in daily use has become more visible in recent years. After more and more admins got tired of the daily click orgies, they were able to use the PowerShell cmdlets to perform many tasks on the command line – more and more of them are only available there. The next step was to use PowerShell scripts to better map tasks in their entirety. These scripts are also used on the command line, today often still locally, after the administrator has connected to the system via RDP. Another evolutionary step is the use of PowerShell remoting. This allows commands and scripts to be executed on remote target systems without first having to log on to the system there.

Against this background, PowerShell is perfectly suited for the bottom-up approach because admins use it:

- to achieve fast improvements and results for their daily work
- to gradually gain personal experience
- as an inexpensive introduction with lower complexity
- without having to become a PowerShell professional immediately



Each of these points is a prerequisite and necessary experience for later automation with PowerShell.

Further questions are essential for the actual automation in the IT operations team:

- How do administrators use the scripts in the team?
- How can I ensure centralized deployment of scripts?
- Who can run which scripts on which systems, when and how?
- How is it ensured that the scripts are only controlled, policy-driven and securely processed on the target systems at all times?
- How transparent is the whole thing? Where and how does central reporting take place and which audit options are necessary?
- What should the development and test process for scripts look like and how can it be established in the company?

In this context, for example, secure delegation of tasks could be one of the first stages of automation with PowerShell.

EXAMPLE USER PROVISIONING

User provisioning simplifies the process of automatically creating, modifying, disabling, and removing user accounts in the IT infrastructure and enterprise applications. At the same time, the appropriate authorizations must be assigned or withdrawn. Experience shows that managing user accounts and authorizations is time-consuming. But that's not all. Delegating the associated tasks to the service desk, for example, is also a challenge.

To enable secure delegation, the Active Directory (AD) and Exchange, for example, can be provided with Role Based Access Control (RBAC). Service Desk users are then trained for two administrative tools, the AD User and Computers Console and the Exchange Management Console. However, this is relatively time consuming, and security aspects also pose challenges. For example, an effective security shell model does not provide direct access by service desk employees to central systems. Instead, access is always indirect, controlled and monitored.

Let us look at a concrete example:
A new user is to be created in the AD and a mailbox that authorizes access to files and printers, updates the distribution lists and sends a first mail with further information. In the classic click, copy & paste, the administrator can expect a wealth of windows, tabs and clicks. Reproducibility and traceability are not really given, susceptibility to errors is high and therefore much time is spent.

After gaining first experiences with PowerShell commands, the administrator can create a script that bundles individual tasks. Using the script on the PowerShell console now leads to higher reproducibility with significantly fewer errors, and logging also improves traceability. Noticeable time savings are achieved. After the positive experiences, the administrator shares the script application with the team and makes it available to everyone in the next step. In addition, an employee from the Service Desk agrees to participate in the experiment with the user provisioning script.

After a short time, however, serious problems arise:

- There are different versions of the script because some colleagues have made personal adjustments.
- The execution and results of the different scripts are no longer reproducible and comparable.
- The execution of the scripts in the personal PowerShell Console prevents uniform logging and thus traceability.
- Some colleagues in the team would like to use the script, but ask for a graphical input mask, because they don't always have the command line and the many necessary parameters for the call ready immediately.
- When the Service Desk employee uses the script, it turns out that he would need too many administrative permissions. Alternatively, these permissions could only be stored in the script code. A much too high security risk.
- In addition, the script has been further developed and improved. A process that ensures the development, testing and transfer of scripts into production would t herefore be necessary.

A closer look reveals the two sides of the coin: On the upside, it should be noted that a start has been made for automation and that the PowerShell script provides reproducible results for complex user provisioning. The downside is that applying the solution only seems to work as a one-man process. Team use fails due to organizational and technical hurdles, and secure delegation is unthinkable.

The administrator could try to store the scripts in a common directory. But the experiences tend to be bad here and the number of personal variants doesn't really decrease. Also, to be able to develop more professionally, the decision is often made in favor of Visual Studio code with a central git. If an administrator tries to implement graphical user elements for his colleagues with on-board tools, he usually finds that this is far too complicated and that the requirements for the service desk cannot be fulfilled in this way.

Instead, he now has to distribute compiled EXE files to the individual employees. However, their execution requires the same administrative rights and constraints as a script. In addition, the scripts still cannot be executed centrally. The execution authorizations still pose a problem, a coherent and comprehensive reporting is not manageable with the distributed log files. It is impossible to define which script should be executed how and where.

COMPONENTS OF AN AUTOMATION SOLUTION

The described experiences are often the normal case when entering the automation world with PowerShell. It is therefore advisable to use a specific automation solution for PowerShell.

The following basic requirements or components of such a solution can be derived:

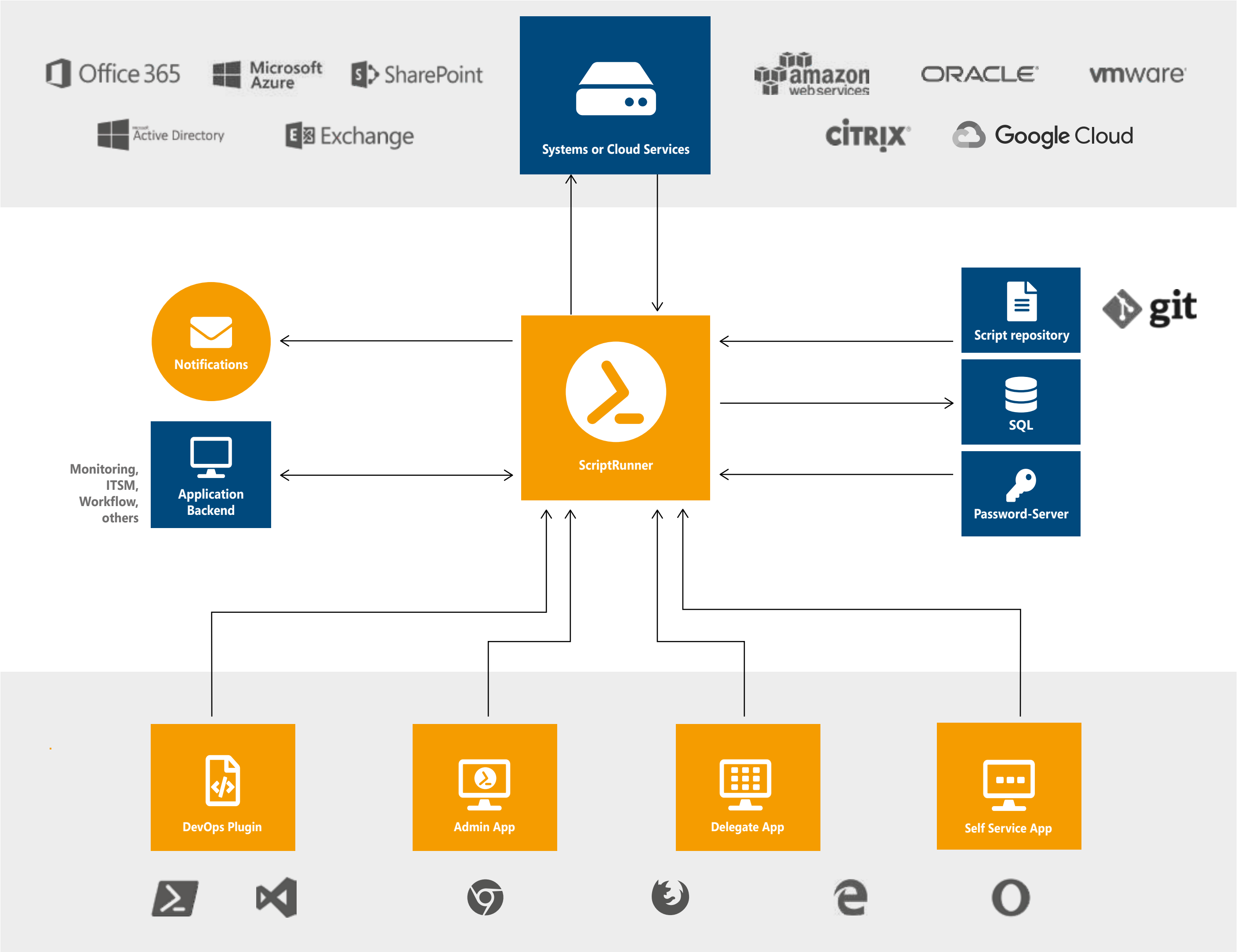
1. Scripts as well as self-written modules at a later date: These implement the necessary steps to complete simple and complex tasks. Functions that are always necessary can be bundled in a library of function scripts or own modules to enable reusability of PowerShell code without copy & paste.

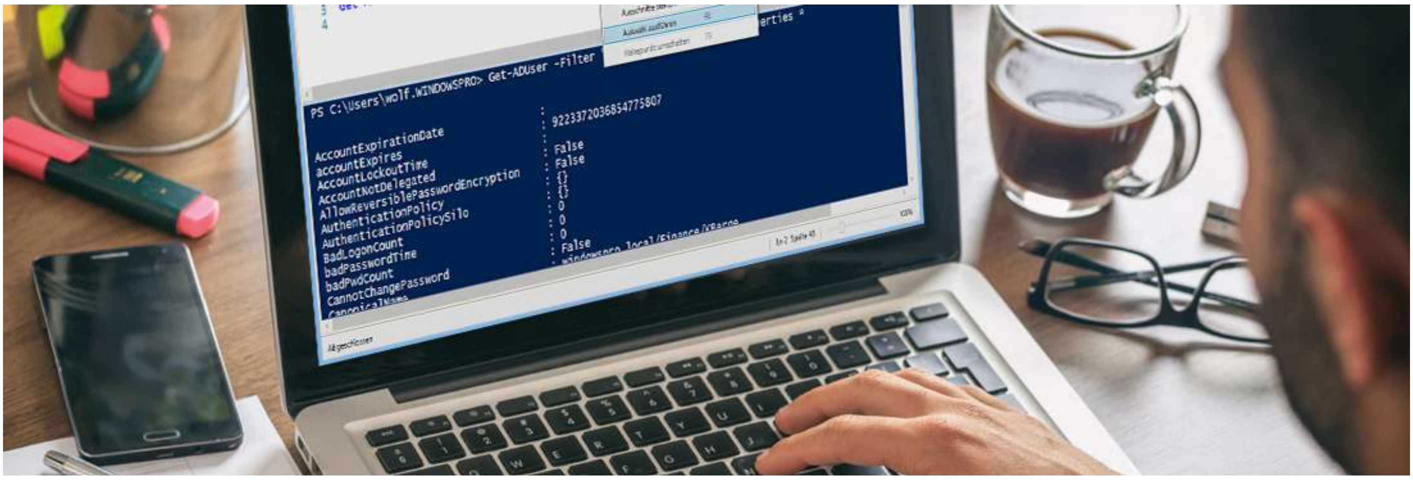
2. Central script repository: A central repository of scripts should ensure that the same script is always used for the same task. By definition, it is forbidden to create and use personal incarnations of a script.

3. Guidelines for execution: the guidelines for script execution should specify which script can be executed how and on which target system in which rights context, in order to enable comparability of results.

4. Rights control for execution: rights control defines various issues. Which credentials are required to execute the script? Which additional credentials must be added to the PowerShell process for processing, and who is authorized to start a script action at all?

5. Execution instance for scripts: This instance ensures reproducibility, i.e. the scripts are always executed in the same way on the target systems and services using the script policies and permission control.





```
PS C:\Users\wolf.WINDOWSPRO> Get-ADUser -Filter                              erties
AccountExpirationDate             :
accountExpires                    : 9223372036854775807
AccountLockoutTime                :
AccountNotDelegated               : False
AllowReversiblePasswordEncryption : False
AuthenticationPolicy              : {}
AuthenticationPolicySilo          : {}
BadLogonCount                     : 0
badPasswordTime                   : 0
badPwdCount                       : 0
CannotChangePassword              : False
CanonicalName                     : windowspro.local/Finance/VBarge
```

6. Central reporting: If scripts are executed, it is important to know which results and errors occurred. With a uniform reporting database that centrally logs each execution, it is possible to track exactly what happened at any time. This is an important prerequisite for audits.

7. Delegation possibility: Against the background of an increasing division of labor and the peculiarities of command line administration, it becomes clear that many employees in IT operations do not or do not want to become PowerShell professionals so quickly. A delegation mechanism should make it possible to assign individual script actions to different user groups based on roles. With this assignment, PowerShell administrators have the ability to allow third parties to perform scripts securely.

8. Graphical application: Although PowerShell is a scripting language, a graphical user interface is available as a browser app for setting up policies, rights, and delegation, right up to viewing the reports. A graphical browser app is ideal for service desk staff and administrators to make scripting widely available. It can be used to enter and validate PowerShell parameters and start a script action.

9. Interfaces: A PowerShell automation solution is usually not alone in the field. Monitoring, ITSM, workflow systems, and others are also important components of the IT infrastructure. For a future overall automation solution, it is enormously important to offer appropriate interfaces for integration. Web services, especially REST, are the interface standard.

10. Development environment for scripts: Administrators developing PowerShell scripts face new challenges. These are divided into two groups. First, there is the development tool including version management and, secondly, mapping the process from design to production. While there is a selection of technical tools on the market, such as Visual Studio Code and Git, process design is heavily dependent on internal organizational constraints. Basically, the different phases of development, testing and production have to be mapped.

CONCLUSION

It makes sense to enter the PowerShell universe at an early stage in order to better understand the concepts, relationships and details. PowerShell offers many possibilities for automation in IT operations. The flexibility of the scripting language in conjunction with the functionality of commands and modules allows daily tasks to be accurately mapped in scripts. For a secure overall solution for all employees in the IT team and beyond, additional technical and organizational support functions are helpful, ideally with a comprehensive platform solution.

Frank Kresse is General Manager at ScriptRunner Software GmbH and creator of ScriptRunner.

This article was also published online on it-administrator.de in May 2018

ScriptRunner Software GmbH
Ludwig-Erhard-Straße 2
76275 Ettlingen
Germany

Tel: +49 (0) 7243 20715-0
Fax: +49 (0) 7243 20715-99
E-Mail: info@scriptrunner.com
Web: www.scriptrunner.com