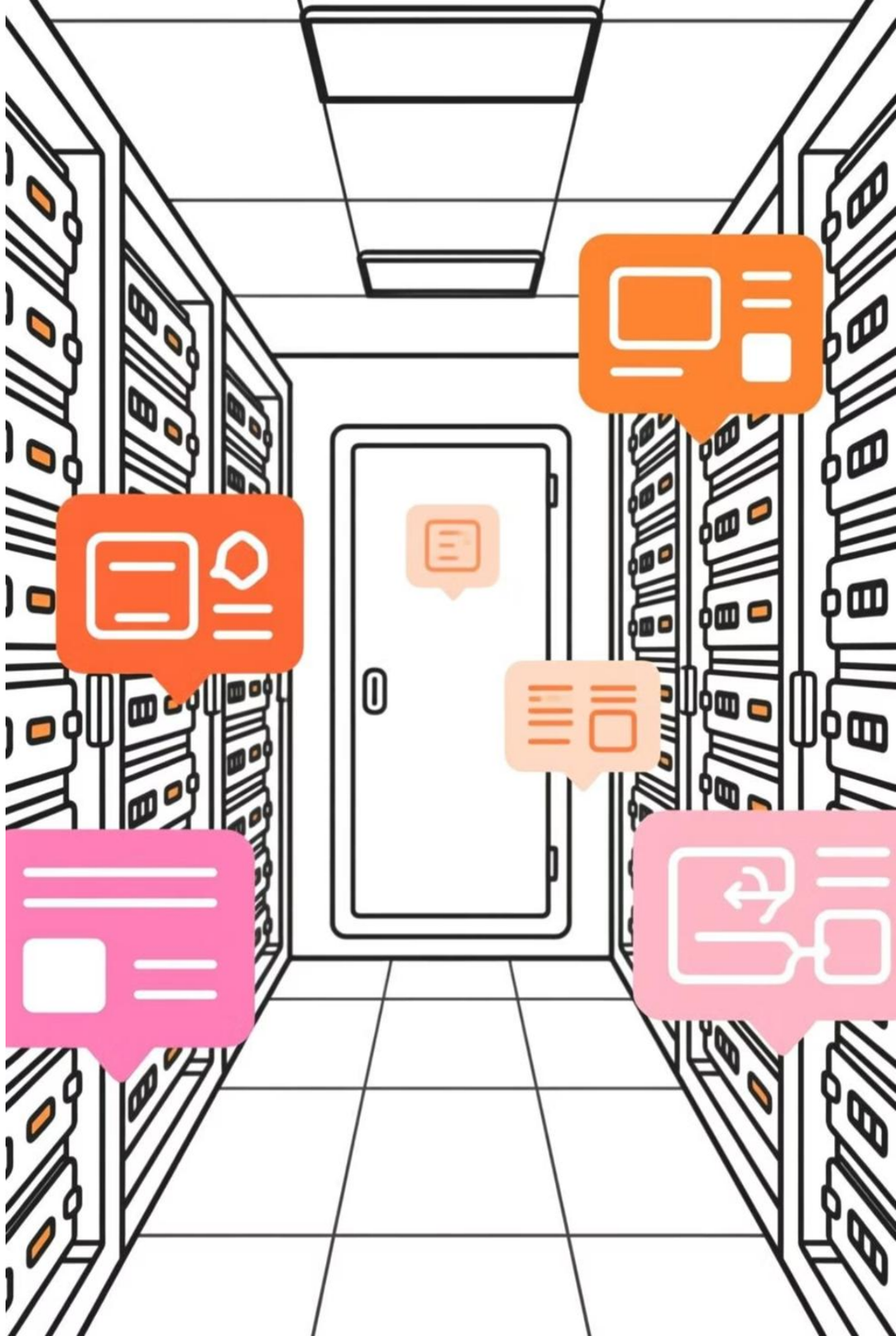
Automation & Scripting: PowerShell, Bash, Ansible & Infrastructure as Code

This comprehensive guide will introduce you to essential automation tools and scripting languages that power modern IT infrastructure. From basic scripting with PowerShell and Bash to configuration management with Ansible and Infrastructure as Code principles, you'll learn how to transform manual processes into efficient, scalable automation.



What Is Automation?

Using code and tools to perform repetitive cloud tasks automatically.

Purpose: Improves efficiency, reduces human error, and ensures consistency.

Why Automate ?

Consistency: Standardize resource creation and management.
Speed: Rapidly deploy and manage infrastructure.
Scalability: Handle large environments with minimal effort.
Cost Savings: Reduce manual labor and downtime.
Integration: Connect automation with CI/CD pipelines.

Example: Turning off VMs at night instead of clicking manually

Common Areas of Automation

- **IT & System Administration:** creating users, managing updates, restarting services.
- **Business Operations:** sending reports, processing invoices, data entry.
- **Cloud & DevOps:** deploying servers, configuring networks, managing backups.
- **Education:** grading submissions, generating student reports.
- **Home Automation:** smart lights, alarms, thermostats, and security cameras.

Tools Used

- **Windows:** PowerShell, Task Scheduler
- **Linux:** Bash, cron jobs
- **Cloud:** Azure CLI, AWS scripts, Ansible
- **General:** Python, Node.js, Docker, Jenkins

Why Automation & Scripting Matter Today

In today's fast-paced IT landscape, automation isn't just convenient—it's critical. Organizations are managing increasingly complex infrastructure across on-premises and cloud environments while facing:

Resource Constraints

IT teams are expected to accomplish more with the same or fewer resources

Speed Requirements

Business needs demand faster deployment and configuration

Scaling Challenges

Managing hundreds or thousands of systems manually is impossible

Automation and scripting provide the foundation for modern DevOps practices, enabling teams to break free from tedious manual work and focus on innovation.

Manual Infrastructure Deployment: The Old Reality

1. **Manual Server Setup**
   Technicians physically install hardware, configure BIOS, and install OS from installation media
2. **Software Installation**
   Administrators manually install applications, patches, and dependencies one-by-one
3. **Configuration**
   Settings applied manually through GUI interfaces or basic scripts, often following lengthy documentation
4. **Testing & Validation**
   Manual checks to verify proper configuration, often missing subtle issues

The High Cost of Manual Deployment

A typical enterprise server deployment could take 2-5 days of work hours, with a 15-30% chance of human error requiring troubleshooting and rework. Organizations with hundreds of servers faced enormous operational overhead.

Automation Transforms IT Operations

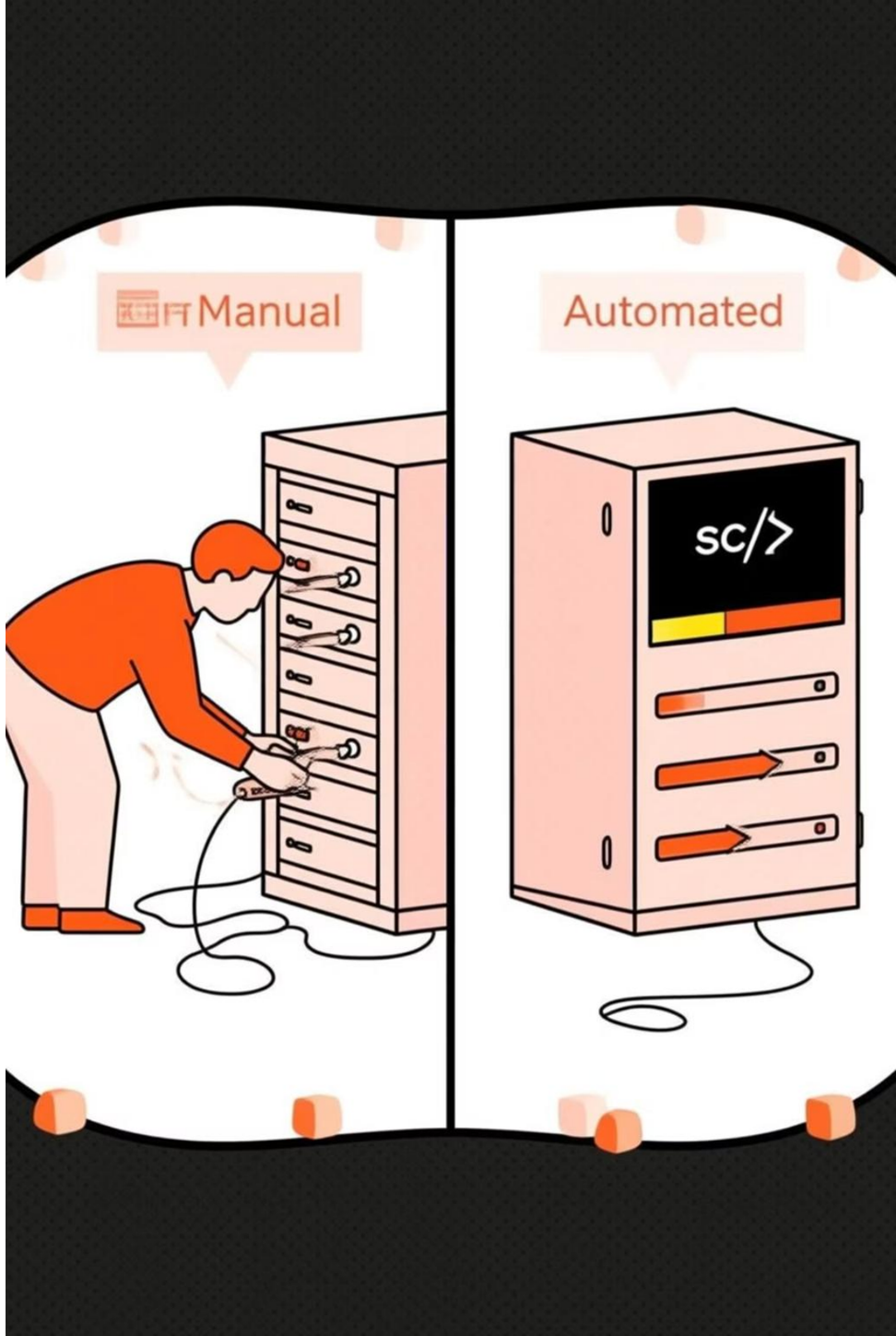
Before Automation

- Server provisioning takes days
- Configuration drift between environments
- Unpredictable deployment outcomes
- Limited ability to scale operations
- High operational costs for routine tasks

After Automation

- Server provisioning in minutes
- Consistent environments guaranteed
- Predictable, repeatable deployments
- Effortless scaling to thousands of systems
- Low operational overhead for routine tasks

> "Infrastructure automation isn't just about efficiency—it's about creating a reliable foundation that enables innovation while reducing the operational burden on IT teams."



Scripting

Scripting means **writing small programs (called scripts)** that tell a computer exactly what to do — step by step.
It's the foundation of automation because the script is what the computer runs automatically.

A **script** is a short list of commands or instructions written in a special language (like PowerShell, Python, or Bash) that a computer follows to perform a task.

Popular Scripting Languages

System	Language
Windows	PowerShell, Batch
Linux / macOS	Bash, Shell Script
Cross-platform	Python, Node.js
Cloud & DevOps	YAML (Ansible), Terraform

What You Can Do with Scripts

Category	Example
🗂️ File Management	Create, rename, copy, delete files or folders
🧮 Data Processing	Read a file, calculate values, generate reports
🖥️ System Admin	Add users, restart services, monitor resources
🌐 Networking	Check connections, send data between computers
📦 Software Tasks	Install apps, run updates, clean cache

Windows Automation (PowerShell & Batch Scripts)

Automating tasks on your Windows computer using simple scripts.

Create folders automatically:

```
# PowerShell script
for ($i=1; $i -le 5; $i++) {
  New-Item -Path
"C:\Users\$env:USERNAME\Documents\Folder$i" -
ItemType Directory
}
```

This creates 5 folders named `Folder1` to `Folder5`.

Delete old files automatically:

```
for ($i=1; $i -le 5; $i++) {
  Remove-Item "C:\Users\$env:USERNAME\Documents\Folder$i" -
Force
}
```

Create a scheduled task called **“DailyRestart5PM”** that restarts your computer **every day at 5 PM**.

```
$action  = New-ScheduledTaskAction -Execute "shutdown.exe"
-Argument "/r /f /t 0"
$trigger = New-ScheduledTaskTrigger -Daily -At 17:00
$settings = New-ScheduledTaskSettingsSet -
AllowStartIfOnBatteries -DontStopIfGoingOnBatteries -
StartWhenAvailable -WakeToRun

Register-ScheduledTask -TaskName "DailyRestart5PM" `
  -Action $action -Trigger $trigger -Settings $settings `
  -RunLevel Highest -User "SYSTEM"
```

Test It Immediately

```
schtasks /Run /TN "DailyRestart5PM"
```

To Delete the Task Later

```
schtasks /Delete /TN "DailyRestart5PM" /F
```

Create a script that will display a Warning popup at 4:55 PM (shows for up to 5 minutes)

PowerShell script that scans **all drives** and lists every file **≥ 1 GB**, prints a table, and saves a CSV on your Desktop.

Run **PowerShell as Administrator** for best results (to avoid access-denied folders).

```
# === Find files >= 1 GB across all drives and export a CSV ===
$ThresholdBytes = 1GB
$timestamp = Get-Date -Format 'yyyyMMdd_HHmm'
$outCsv = Join-Path $env:USERPROFILE "Desktop\LargeFiles_$timestamp.csv"

# Optional: exclude noisy/system folders (edit as you like)
$excludePattern = '\\Windows\\|\\Program Files(
\(x86\))?\\|\\ProgramData\\|\\AppData\\|\\Recovery\\|\\$Recycle\.Bin\\|\\System Volume
Information\\'

$results = foreach ($drive in (Get-PSDrive -PSProvider FileSystem)) {
  Write-Host "Scanning $($drive.Root) ..."
  Get-ChildItem -Path $drive.Root -Recurse -Force -File -ErrorAction SilentlyContinue |
    Where-Object {
      $_.Length -ge $ThresholdBytes -and ($_.FullName -notmatch $excludePattern)
    } |
    Select-Object @{
        Name='SizeGB'; Expression={ [math]::Round($_.Length/1GB, 2) }
      }, Length, LastWriteTime, FullName, DirectoryName, Name
}

$results |
  Sort-Object SizeGB -Descending |
  Tee-Object -Variable big |
  Format-Table SizeGB, LastWriteTime, FullName -AutoSize

$big | Export-Csv -NoTypeInformation -Path $outCsv
Write-Host "`nSaved CSV to: $outCsv"
```

Key Azure Tools:

Tool	Description	Difficulty
Azure CLI	Type commands in a terminal	⭐
Azure PowerShell	Windows-style commands	⭐
Bicep / ARM Templates	Define resources in files	⭐⭐
Azure Automation	Run scripts in the cloud	⭐⭐

Getting Started with Azure CLI

- **Command-line tool** for managing Azure resources.
- Cross-platform: Works on Windows, macOS, and Linux.
- Best for quick automation scripts.

Step 1: Open the **Azure Cloud Shell** (built into Azure Portal).
Step 2: Try these commands 👇

```
az login
az group create --name myGroup --location eastus
az vm create --resource-group myGroup --name
testVM --image UbuntuLTS
```

Activity: Create and delete your first resource group.

Azure Automation (No Coding Needed)

- Azure Automation is a Cloud-based automation platform that lets you **run scripts automatically**.
- Runs **PowerShell and Python runbooks**.
- You can schedule them (like alarms).

Capabilities:

- Automate VM management.
- Schedule tasks.
- Update management.
- Configuration tracking.

Runbook types:

- Graphical (drag-and-drop)
- PowerShell
- Python

Example:

Runbook to stop a VM:

```
Stop-AzVM -Name "testVM" -ResourceGroupName "myGroup" –Force
```

Activity: Create a runbook that shuts down your VM daily.

Bash: The Ubiquitous Unix Shell

Bash (Bourne Again SHell) is the default shell on most Linux distributions and macOS. Its ubiquity makes it an essential skill for anyone working in IT.

Key Bash Features

Text Stream Processing: Powerful text manipulation with tools like `grep`, `sed`, and `awk`

Pipeline Chaining: Connect commands with pipes (|) to create powerful processing workflows

Job Control: Manage background and foreground processes

Shell Expansions: Filename, variable, command, and arithmetic expansion

```
#!/bin/bash# Bash Example# Monitor disk space and alert if over 90%# Get filesystems over 90% usagecritical_fs=$(df -h | grep -v "Filesystem" |  awk '{print $5 " " $6}' | grep "^9[0-9]%" || true)# Send alert if any filesystems are criticalif [ ! -z "$critical_fs" ]; then echo "CRITICAL: Disk space alert!" echo "The following filesystems are over 90%:" echo "$critical_fs" # Would normally send email or notification hereelse echo "All filesystems have adequate space."fi
```

PowerShell vs Bash: Strengths & Weaknesses

PowerShell Strengths

- Superior object handling and structured data processing
- Deep integration with Windows systems and services
- Consistent parameter handling with robust help system
- Strong typing and error handling

Bash Strengths

- Extremely lightweight and fast
- Universal availability on Unix-like systems
- Rich ecosystem of text processing tools
- Concise syntax for common operations

PowerShell Limitations

- More verbose syntax than Bash
- Slower startup time compared to Bash
- Less mature on Linux/macOS platforms
- Steeper learning curve for beginners

Bash Limitations

- Limited structured data handling
- Inconsistent parameter handling across commands
- Weaker error handling capabilities
- Limited debugging facilities

Choosing the Right Tool

Select PowerShell for Windows administration, complex data manipulation, and environments where object handling is valuable. Choose Bash for Unix/Linux environments, text processing tasks, and situations where script portability and performance are critical.

What Is Infrastructure as Code (IaC)?

- Infrastructure as Code means **managing and provisioning IT infrastructure (like servers, networks, and databases) using code**, instead of clicking around in a dashboard.
- You write code to build your infrastructure automatically — just like writing a program that creates your computers, servers, and connections for you.

Why IaC Is Important

Benefit	Description
Consistency	Prevents “it works on my machine” issues — everyone gets the same setup.
Speed	You can deploy or rebuild infrastructure in minutes instead of hours.
Version Control	You can track changes and roll back just like code.
Scalability	Easy to create multiple environments (dev, test, production).
Automation	Reduces human error and repetitive manual work.

Core Concepts of IaC

- **Declarative** , You describe *what* you want (the desired end state).

  Example: “I need one server with Ubuntu in region eastus.”

  Tools like Terraform, CloudFormation, Bicep use this.

- **Imperative** ,You describe *how* to get there step by step.

  Example: “Create a VM, then install Ubuntu, then configure settings.”

  Tools like Ansible or Bash scripts can use this approach.

Common IaC Tools

Tool	Description
Terraform	Cloud-agnostic tool — works with AWS, Azure, GCP.
AWS CloudFormation	For Amazon Web Services infrastructure.
Azure Bicep / ARM Templates	For defining Azure infrastructure as code.
Ansible / Chef / Puppet	Configuration management tools — great for software setup.
Pulumi	Lets you use real programming languages (Python, TypeScript) for IaC.

Infrastructure as Code (IaC) to be done **locally** with PowerShell.

Tasks

You'll create a **PowerShell script** that:

- Automatically sets up a mini "infrastructure" on your PC (folders, logs, text files).
- It's repeatable and versionable — the same script gives the same setup every time (that's what IaC is about!).

✓ **Step 1: Create the Script**

Open Notepad, paste this in, and save it as `setup_environment.ps1` (on your Desktop).

Then run:

```
.\setup_environment.ps1
```

To verify:

```
Get-ExecutionPolicy -List
```

Lets create sample files and subfolders inside each of the environment folders (Logs, Data, Config)

- Save this as populate_environment.ps1
- Run it in PowerShell:

```
.\populate_environment.ps1
```

Automate Daily or Repetitive Tasks

You can write new PowerShell scripts that **run inside** this environment, such as:

A. Backup important files

```
Copy-Item "C:\Users\$env:USERNAME\Documents\*.docx"
"C:\MyLocalEnvironment\Data" -Recurse
```

Automatically copies your documents into the `Data` folder.

B. Use PowerShell to List Files

Run this command:

```
Get-ChildItem "C:\MyLocalEnvironment\Data" -Recurse | Where-
Object { $_.Extension -eq ".docx" } | Select-Object FullName
```

This will print out a list of all .docx files that were copied