Automation and Scripting for System Admin and Troubleshooting

By,

Mr. Kumar Pudashine, (MEng, AIT, Bangkok)
PMP, CISA, CISM, CRISC, CNDA, CDCP, COBIT 5, CCNP (Enterprise), JNCIA, CEH v9, ITIL, ISO 27001:2022, AcitivIdentity Certified
Founder and President, Global Cybersecurity Community Forum
IEEE, ISACA, PMI Member

June 2025

Top 10 Scripting Languages in DevOps

Scripting Language: Definition ?

- Scripting languages are high-level programming languages designed for integrating and communicating with other programming languages.
- They are often interpreted rather than compiled, making them easy to write and execute quickly.
- Scripting languages are commonly used for automating tasks, manipulating data, and managing systems or web content.

Scripting Language: Characteristics

- **Interpreted:** Scripting languages are typically interpreted rather than compiled. The code is executed directly by an interpreter.
- **Ease of Use:** They are often easier to write and require fewer lines of code compared to compiled languages.
- **Dynamic Typing:** Variables in scripting languages are usually dynamically typed, meaning that the type of a variable is determined at runtime.
- **Rapid Development:** They facilitate quick development and prototyping.
- **Integration:** They are designed to integrate and communicate with other programming languages and systems.

Scripting Language: Windows PowerShell ISE

Opening PowerShell ISE

1. **Start PowerShell ISE:**
   - Press `Win + R` to open the Run dialog.
   - Type `powershell_ise` and press Enter.
   - Alternatively, you can search for "Windows PowerShell ISE" in the Start menu and open it from there.

Scripting Language: Windows PowerShell ISE

Writing and Running a Script in PowerShell ISE

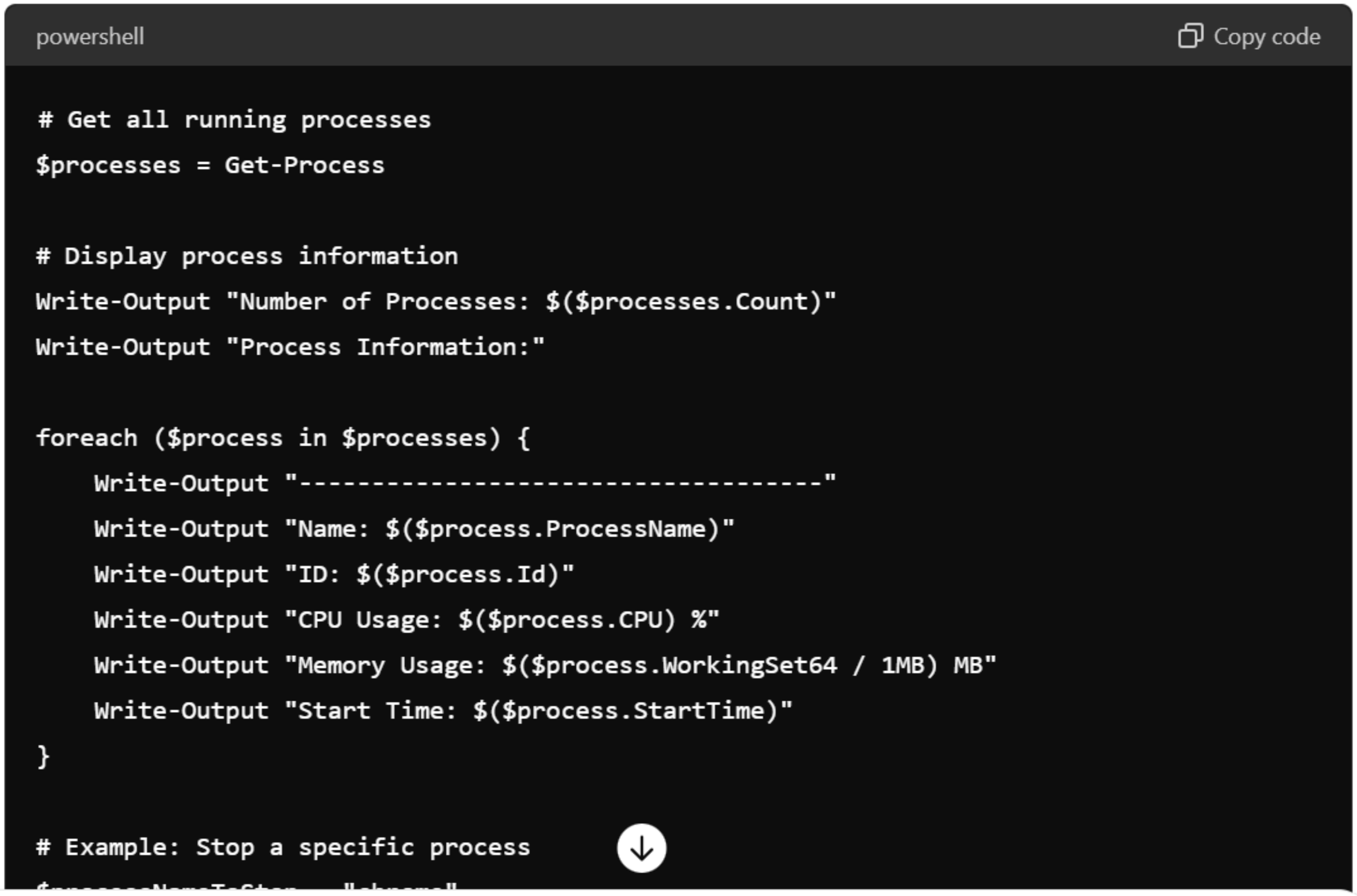
Step 1: Writing a Script

In the Script Pane, type the following example script:

```powershell
# Script to display system information

# Get the computer name
$computerName = $env:COMPUTERNAME

# Get the operating system
$operatingSystem = Get-WmiObject -Class Win32_OperatingSystem

# Display the information
Write-Output "Computer Name: $computerName"
Write-Output "Operating System: $($operatingSystem.Caption)"
Write-Output "OS Architecture: $($operatingSystem.OSArchitecture)"
Write-Output "Version: $($operatingSystem.Version)"
Write-Output "Build Number: $($operatingSystem.BuildNumber)"
```

Scripting Language: Windows PowerShell ISE

Example: Retrieving and Managing Processes

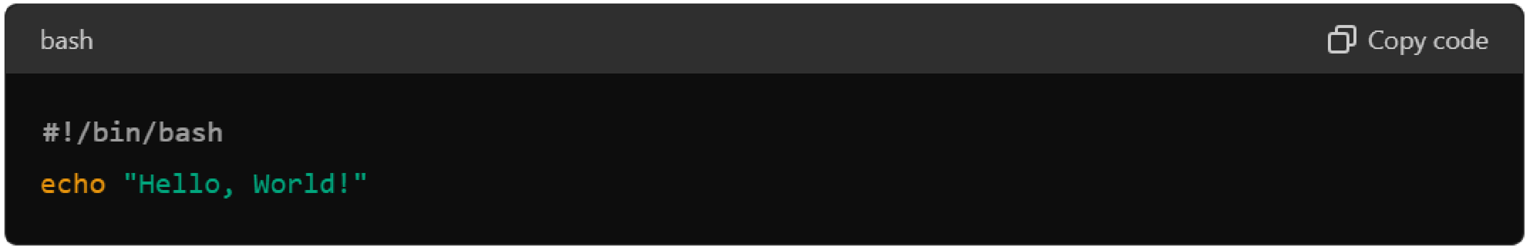


```powershell
# Get all running processes
$processes = Get-Process

# Display process information
Write-Output "Number of Processes: $($processes.Count)"
Write-Output "Process Information:"

foreach ($process in $processes) {
    Write-Output "------------------------------------"
    Write-Output "Name: $($process.ProcessName)"
    Write-Output "ID: $($process.Id)"
    Write-Output "CPU Usage: $($process.CPU) %"
    Write-Output "Memory Usage: $($process.WorkingSet64 / 1MB) MB"
    Write-Output "Start Time: $($process.StartTime)"
}

# Example: Stop a specific process
$processNameToStop = "chrome"
```

Scripting Language: Bash Shell Scripting

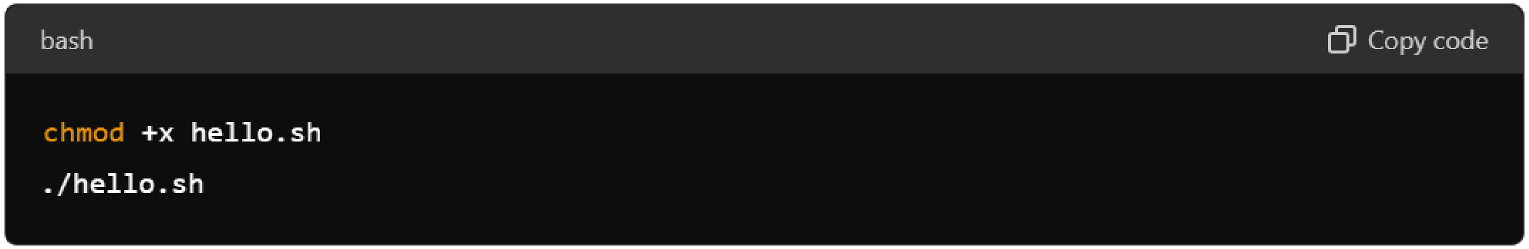
Example: Hello World Script

Create a file named `hello.sh` with the following content:



```bash
#!/bin/bash
echo "Hello, World!"
```

Make the script executable and run it:



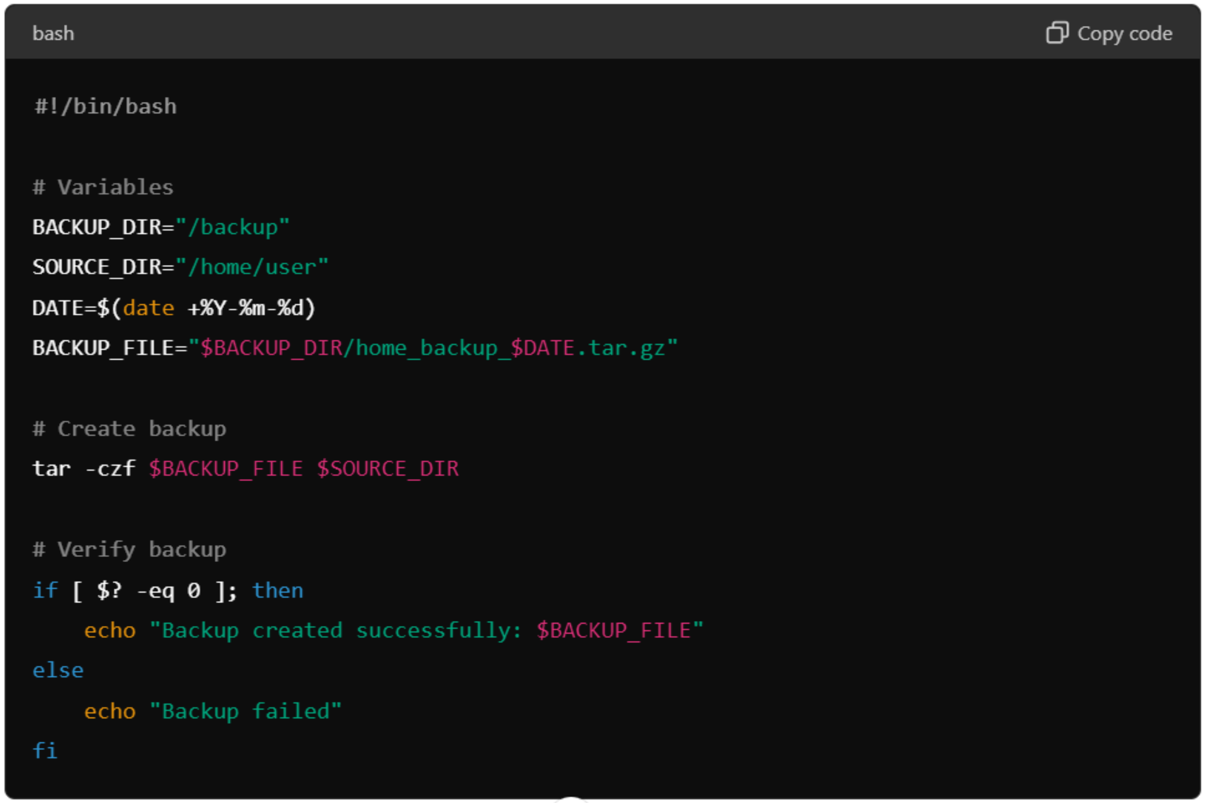
```bash
chmod +x hello.sh
./hello.sh
```

Scripting Language: Bash Shell Scripting

1. Backup Script

Example: Backup Home Directory

This script creates a compressed backup of the user's home directory.



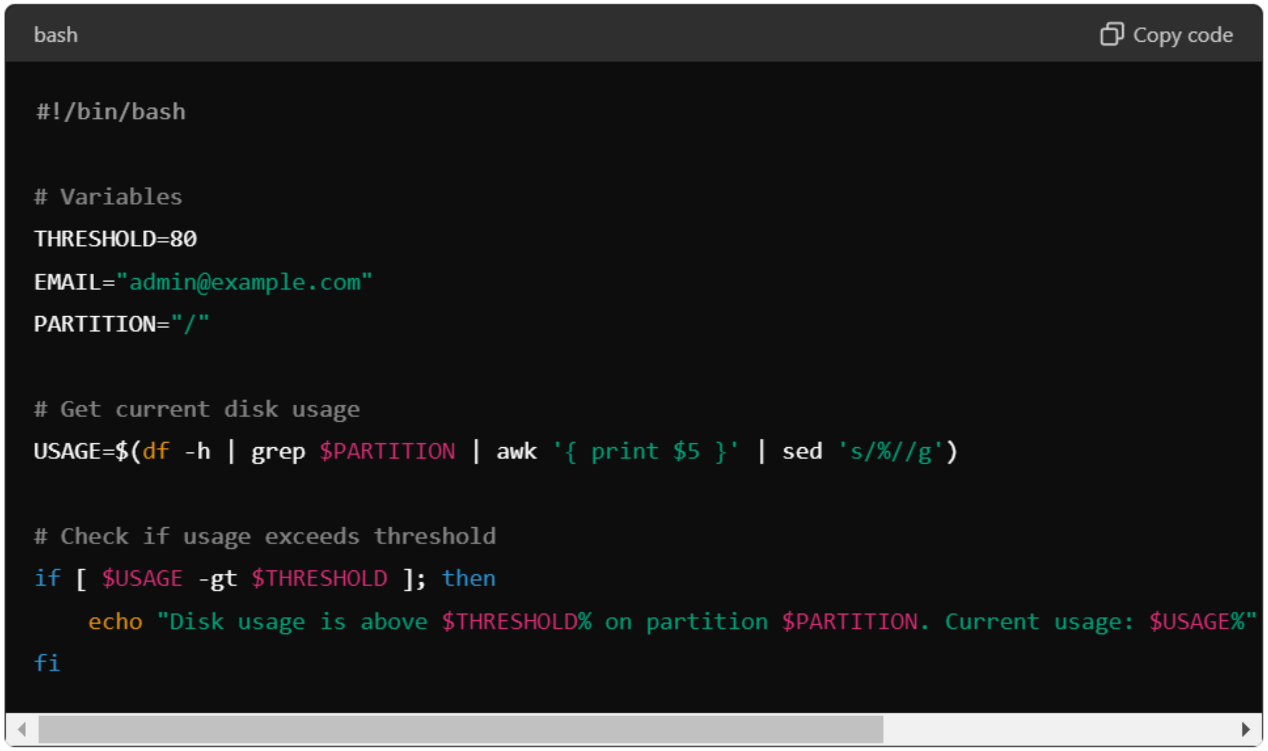
```bash
#!/bin/bash

# Variables
BACKUP_DIR="/backup"
SOURCE_DIR="/home/user"
DATE=$(date +%Y-%m-%d)
BACKUP_FILE="$BACKUP_DIR/home_backup_$DATE.tar.gz"

# Create backup
tar -czf $BACKUP_FILE $SOURCE_DIR

# Verify backup
if [ $? -eq 0 ]; then
    echo "Backup created successfully: $BACKUP_FILE"
else
    echo "Backup failed"
fi
```

Scripting Language: Bash Shell Scripting

2. Disk Usage Monitoring

Example: Check Disk Usage and Send Alert

This script checks the disk usage of a specified directory and sends an alert if usage exceeds a threshold.



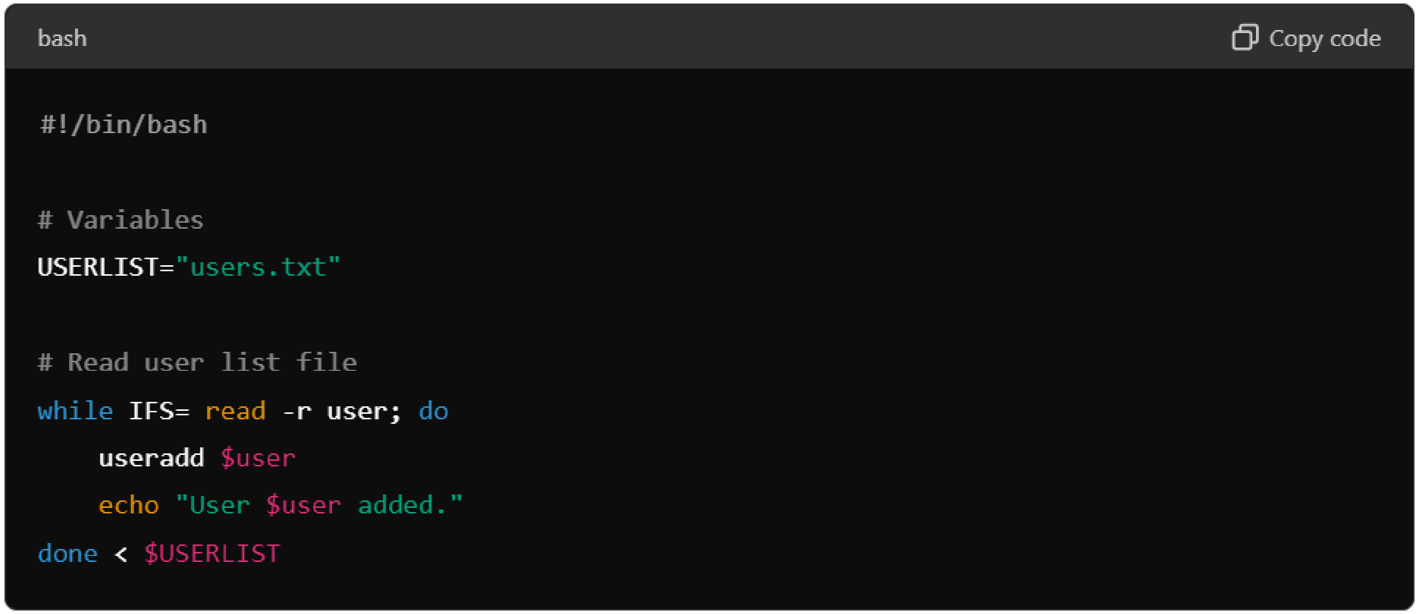
```bash
#!/bin/bash

# Variables
THRESHOLD=80
EMAIL="admin@example.com"
PARTITION="/"

# Get current disk usage
USAGE=$(df -h | grep $PARTITION | awk '{ print $5 }' | sed 's/%//g')

# Check if usage exceeds threshold
if [ $USAGE -gt $THRESHOLD ]; then
    echo "Disk usage is above $THRESHOLD% on partition $PARTITION. Current usage: $USAGE%"
fi
```

Scripting Language: Bash Shell Scripting

3. User Account Management

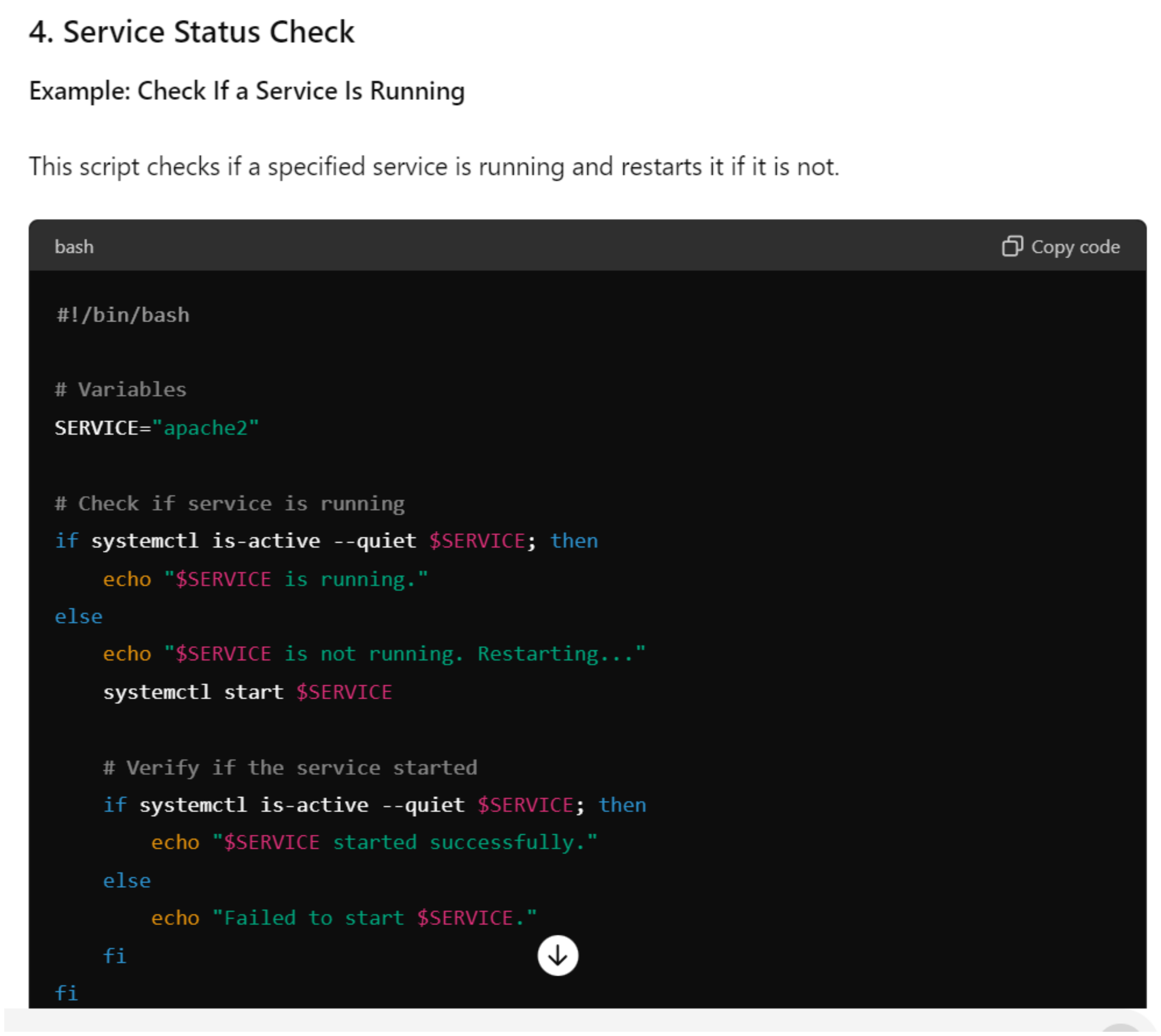
Example: Create Multiple User Accounts

This script reads a list of usernames from a file and creates user accounts for each.



```bash
#!/bin/bash

# Variables
USERLIST="users.txt"

# Read user list file
while IFS= read -r user; do
    useradd $user
    echo "User $user added."
done < $USERLIST
```

Scripting Language: Bash Shell Scripting



4. Service Status Check

Example: Check If a Service Is Running

This script checks if a specified service is running and restarts it if it is not.

bash Copy code

```bash
#!/bin/bash

# Variables
SERVICE="apache2"

# Check if service is running
if systemctl is-active --quiet $SERVICE; then
    echo "$SERVICE is running."
else
    echo "$SERVICE is not running. Restarting..."
    systemctl start $SERVICE

    # Verify if the service started
    if systemctl is-active --quiet $SERVICE; then
        echo "$SERVICE started successfully."
    else
        echo "Failed to start $SERVICE."
    fi
fi
```

Ansible

- Ansible is an open-source automation tool used for IT tasks such as configuration management, application deployment, orchestration, and task automation.
- It allows you to manage and configure systems using simple, human-readable YAML files called playbooks.
- Ansible operates without requiring an agent on the target machines, relying instead on SSH for communication.

Ansible: Key Concepts ??

Inventory: A list of hosts or nodes that Ansible manages.

Playbook: A YAML file containing a series of tasks that define the desired state of the system.

Task: An action that Ansible performs, such as installing a package or copying a file.

Module: A pre-defined script that Ansible runs to perform specific tasks.

Role: A way to group related tasks and configurations into reusable components.

Example 1: Simple Playbook to Install Nginx

Create a playbook (`` `install_nginx.yml` ``) to install Nginx on web servers:

```yaml
---
- name: Install Nginx on web servers
  hosts: webservers
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present

    - name: Start Nginx service
      service:
        name: nginx
        state: started
```

Run the playbook:

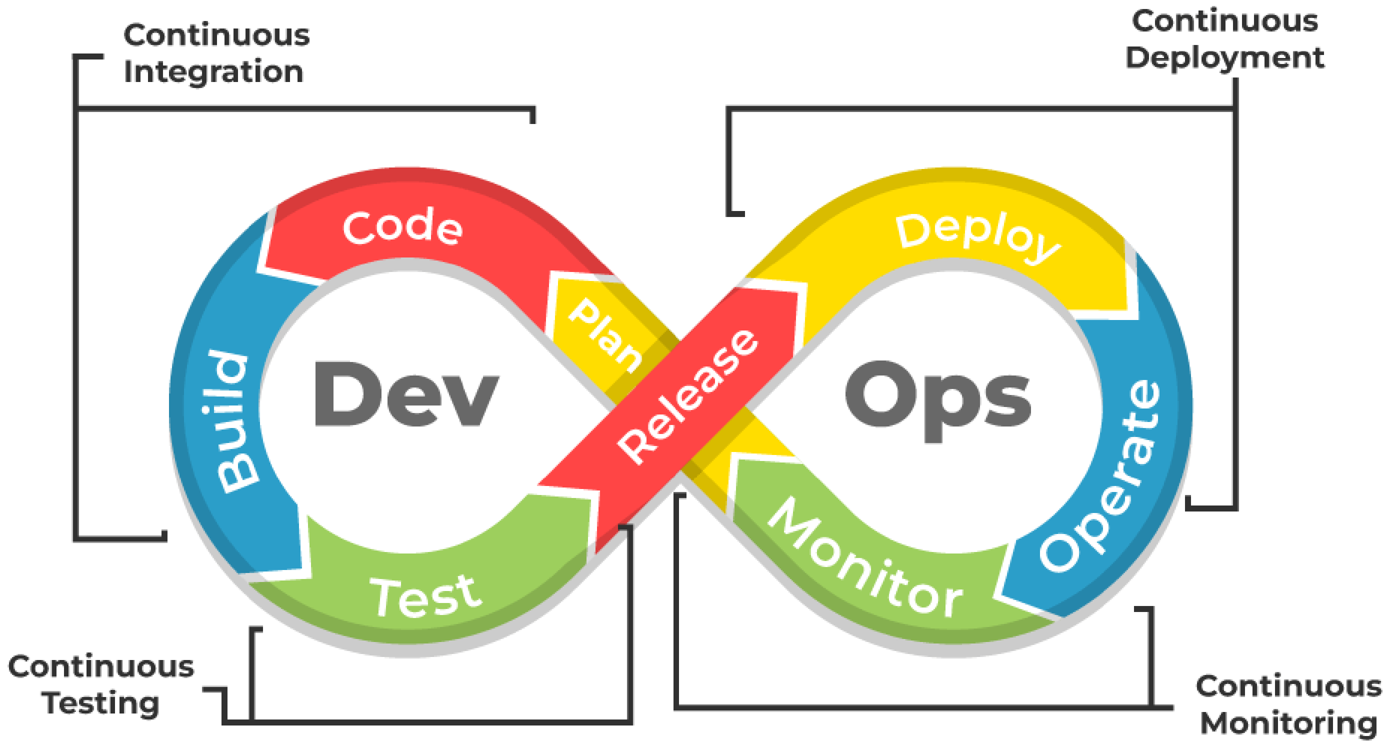
```bash
ansible-playbook -i hosts.ini install_nginx.yml
```

DevOps Principles and Practices

- DevOps, a blend of "Development" and "Operations," is a set of practices, cultural philosophies, and tools designed to improve an organization's ability to deliver applications and services at high velocity.

- It aims to create a seamless workflow between software development (Dev) and IT operations (Ops), enabling faster development cycles, increased deployment frequency, and more reliable releases.

DevOps: Development and Operations



DevOps Principles and Practices: Key Concepts ??

1. Continuous Integration (CI) and Continuous Deployment (CD)

- Continuous Integration: Developers frequently integrate code into a shared repository, usually multiple times a day. Each integration is verified by an automated build and automated tests to detect integration errors quickly.

- Continuous Deployment: Automated deployment of code changes to a production environment after passing the CI pipeline, ensuring that software can be released to users quickly and sustainably.

2. Infrastructure as Code (IaC)

- Managing and provisioning computing infrastructure through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. Common IaC tools include Terraform, Ansible, and Puppet.

DevOps Principles and Practices: Key Concepts ??

3. Micro services Architecture

- An architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. This approach allows each service to be developed, deployed, and scaled independently.

4. Monitoring and Logging

- Implementing robust monitoring and logging systems to track the performance of applications and infrastructure.
- Tools like Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), and Splunk are commonly used for this purpose.

DevOps Principles and Practices: Key Concepts ??

5. Collaboration and Communication

- Fostering a culture of shared responsibility and collaboration between development and operations teams.
- Tools such as Slack, Microsoft Teams, and Jira are often used to facilitate communication and project management.

6. Automation

- Automating repetitive tasks to increase efficiency and reduce human error.
- Automation spans various aspects, including code testing, deployment, infrastructure provisioning, and configuration management.

DevOps Principles and Practices: Popular Tools

Version Control: Git, GitHub, GitLab, Bitbucket

CI/CD Pipelines: Jenkins, Travis CI, CircleCI, GitLab CI/CD

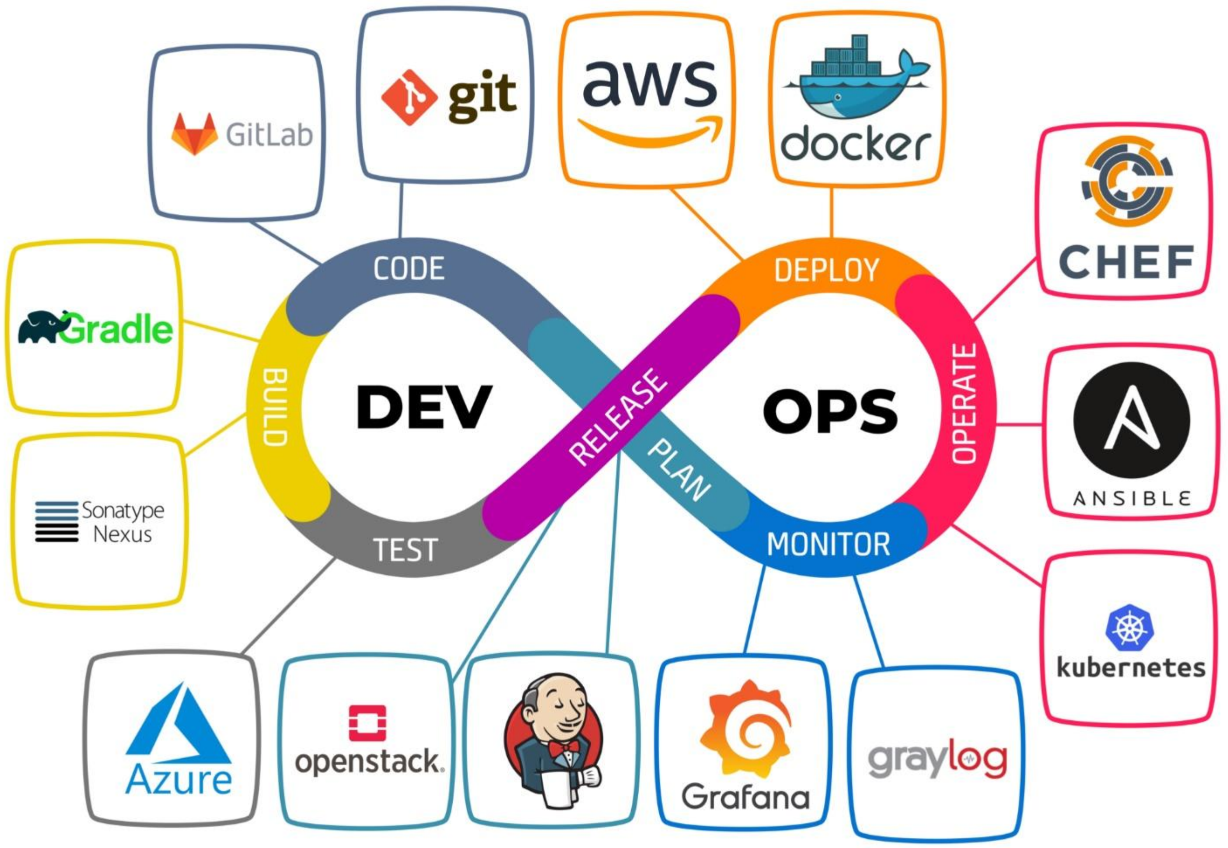
Configuration Management: Ansible, Puppet, Chef

Containerization: Docker, Kubernetes

Monitoring: Prometheus, Grafana, Nagios

Collaboration: Slack, Microsoft Teams, Jira

DevOps: Development and Operations



DevOps Principles and Practices: Benefits

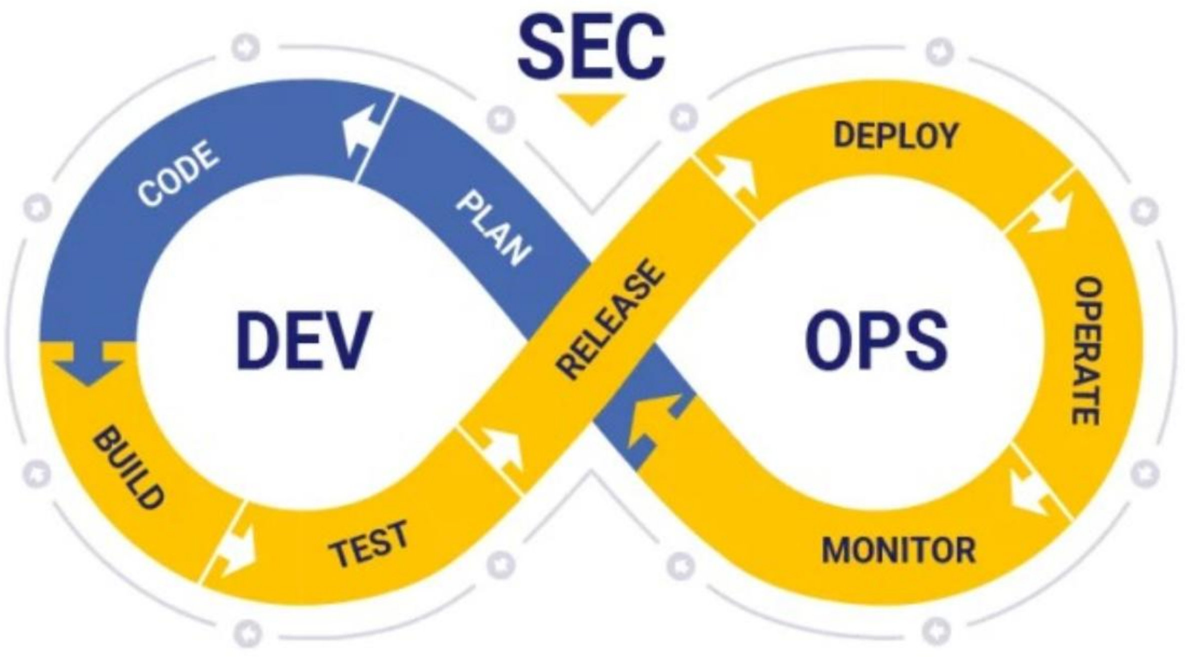
Accelerated Time to Market: Faster development and deployment cycles allow organizations to release new features and updates more quickly.

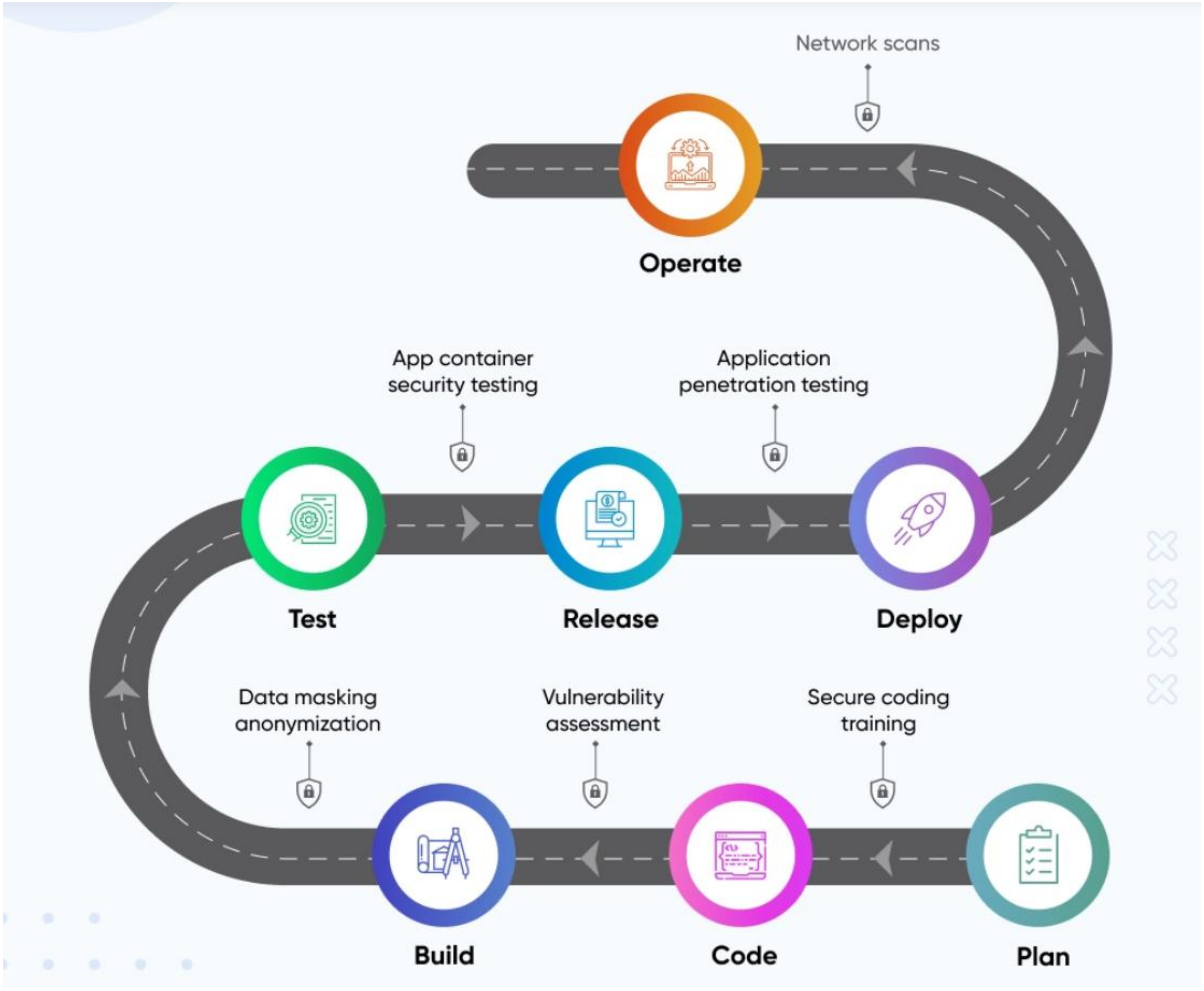
Improved Collaboration: Enhanced communication and collaboration between teams lead to better product quality and innovation.

Increased Efficiency: Automation of repetitive tasks reduces manual intervention and errors, leading to more efficient processes.

Enhanced Security: By integrating security practices into the DevOps workflow (DevSecOps), organizations can ensure that security is a continuous and integrated part of the development process.

DevSecOps: Development, Security, and Operations




Network scans
Operate
App container security testing
Application penetration testing
Test
Release
Deploy
Data masking anonymization
Vulnerability assessment
Secure coding training
Build
Code
Plan

Troubleshooting

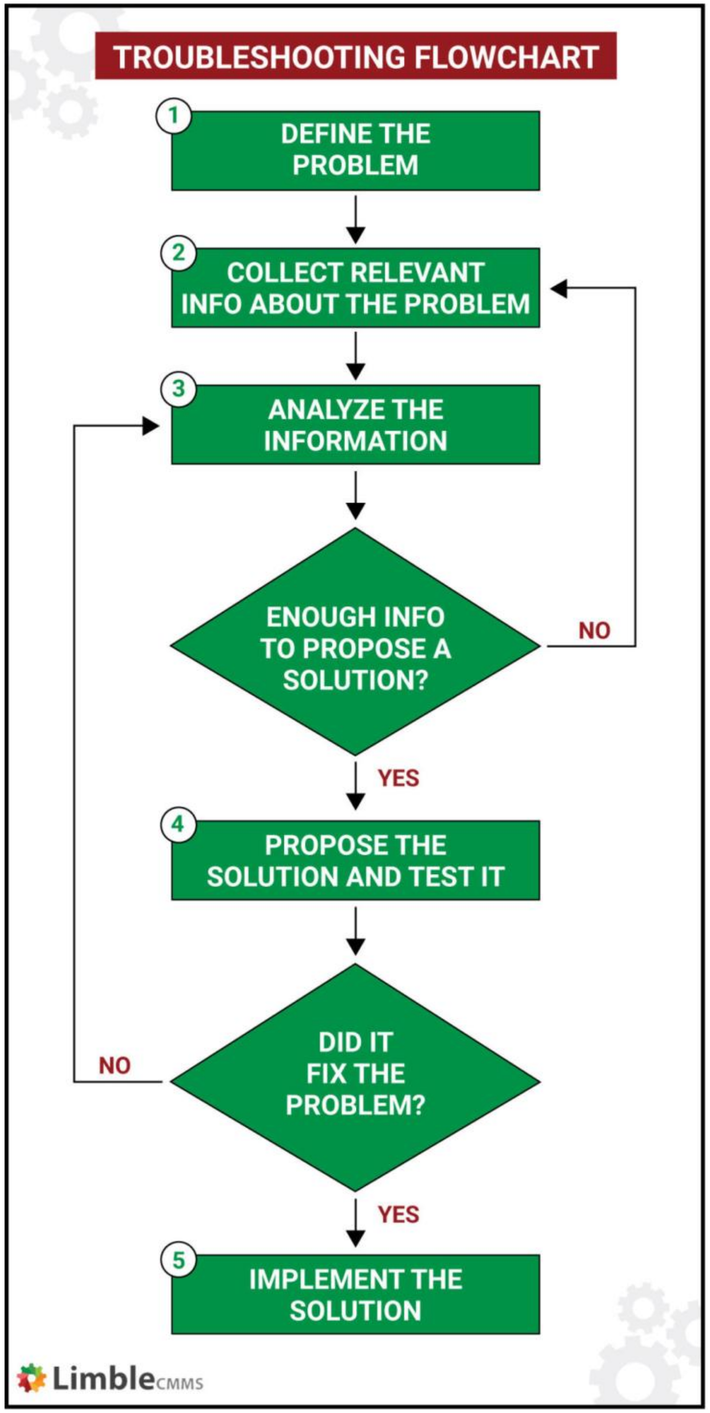
- ✓ Troubleshooting in system administration involves identifying, diagnosing, and resolving problems within a computer system, network, or IT environment.

- ✓ Structured approach to troubleshooting common issues in system administration are as follows:

  1. Identify the Problem
  2. Analyze the Problem
  3. Formulate Hypotheses
  4. Test Hypotheses
  5. Implement Solutions
  6. Verify the Solution
  7. Prevent Future Issues

Troubleshooting

WHEN TO TROUBLESHOOT?
device failure
unexpected operation
other anomalies
LimbleCMMS


TROUBLESHOOTING FLOWCHART
1
DEFINE THE PROBLEM
2
COLLECT RELEVANT INFO ABOUT THE PROBLEM
3
ANALYZE THE INFORMATION
ENOUGH INFO TO PROPOSE A SOLUTION?
NO
YES
4
PROPOSE THE SOLUTION AND TEST IT
DID IT FIX THE PROBLEM?
NO
YES
5
IMPLEMENT THE SOLUTION
LimbleCMMS

Remote Administration

- ✓ Remote administration refers to the ability to manage a computer or network from a remote location.
- ✓ This can involve a variety of tasks such as software installation, system updates, troubleshooting, monitoring, and managing user permissions.

Remote Administration: Benefits

1. **Convenience:** Administrators can manage systems from anywhere, reducing the need for physical presence.
2. **Efficiency:** Enables quick responses to issues, minimizing downtime.
3. **Cost-Effective:** Reduces travel and on-site support costs.
4. **Scalability:** Easier to manage multiple systems spread across different location

Remote Administration Tools

1. **Remote Desktop Protocol (RDP):** Allows users to connect to another computer over a network connection. It is commonly used in Windows environments.

2. **Secure Shell (SSH):** Provides a secure channel over an unsecured network, commonly used for remote command-line access in Unix-like operating systems.

3. **Virtual Network Computing (VNC):** A graphical desktop-sharing system that uses the Remote Frame Buffer (RFB) protocol to remotely control another computer.

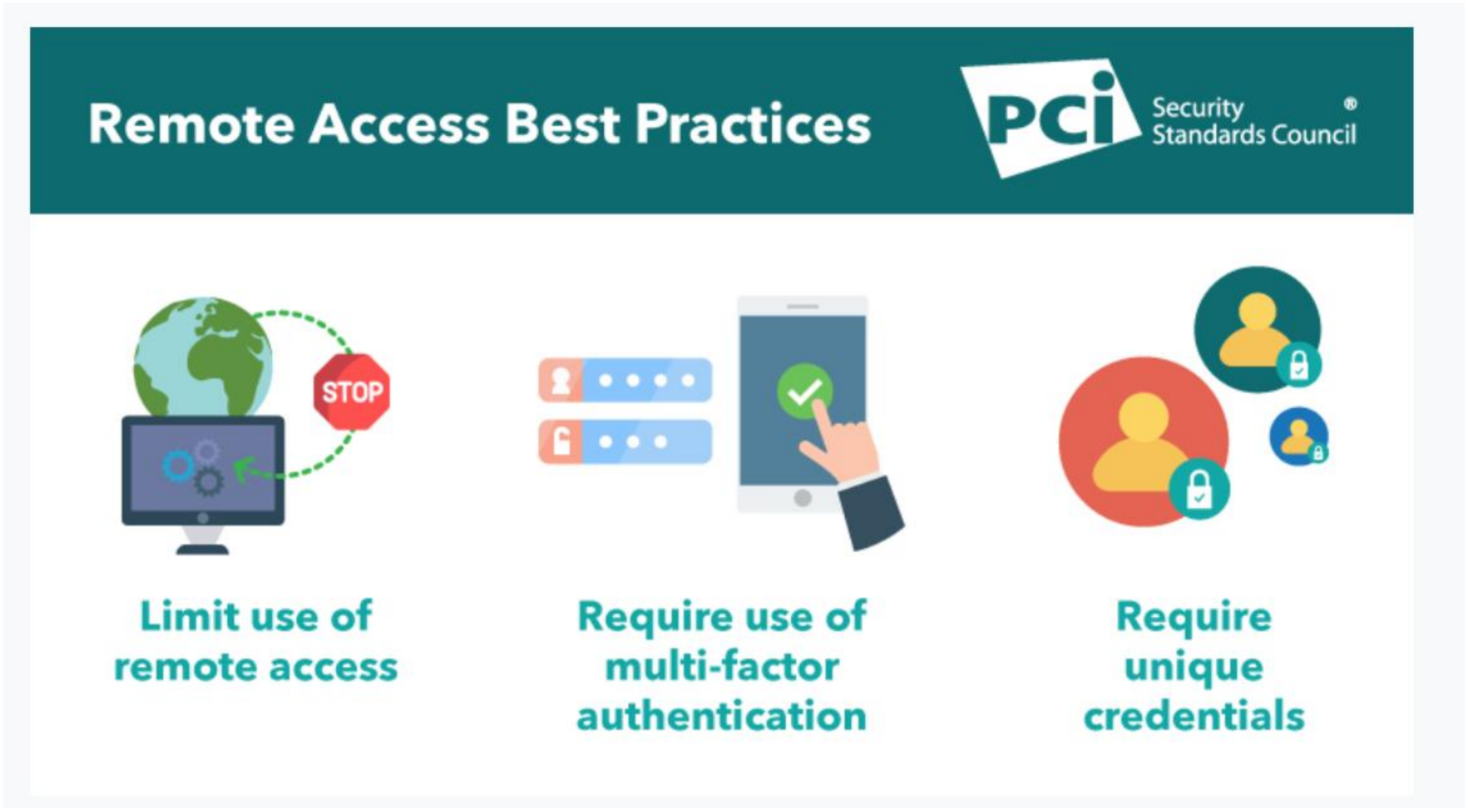
4. **Remote Management Tools:** Software applications like TeamViewer, AnyDesk, LogMeIn, and Microsoft Remote Desktop provide comprehensive remote access solutions.

5. **Web-Based Administration:** Many systems and applications offer web interfaces for remote management. This can include web servers, routers, and cloud services.

Remote Administration: Security Considerations

- **Encryption:** Use secure protocols like SSH and VPNs to protect data in transit.
- **Authentication:** Implement strong authentication mechanisms, such as multi-factor authentication.
- **Access Control:** Limit access to authorized users only and use role-based access controls.
- **Monitoring and Logging:** Keep logs of remote access sessions for auditing and monitoring purposes.
- **Regular Updates:** Ensure remote administration tools are up-to-date to protect against vulnerabilities.

Remote Administration: Remote Access Best Practices



Remote Access Best Practices

PCI Security Standards Council®

Limit use of remote access

Require use of multi-factor authentication

Require unique credentials

Thank You