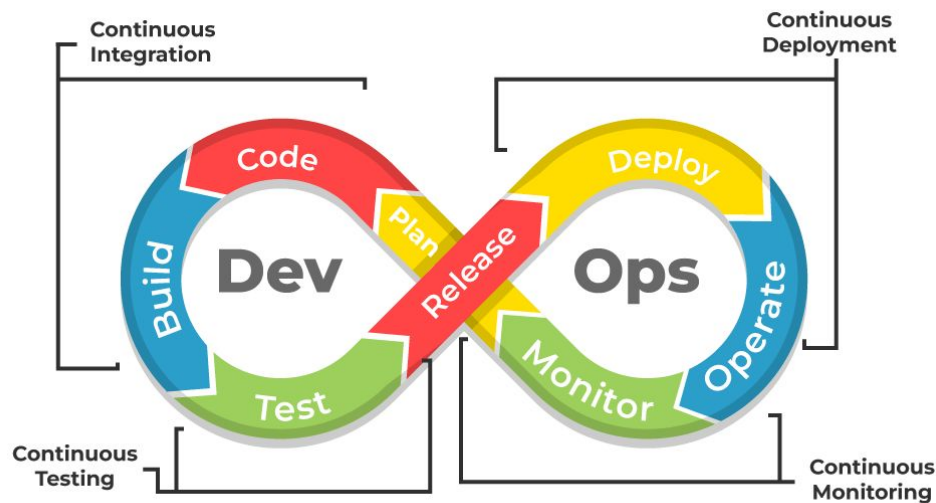










An aerial night view of a city, likely Shanghai, with numerous skyscrapers and a complex highway interchange. Overlaid on the cityscape are glowing yellow lines that form a network of arcs and connections, suggesting a digital or infrastructure theme. The text "DevOps and Infrastructure as Code" is centered in white, bold, sans-serif font.









DevOps and Infrastructure as Code









Introduction



- DevOps is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development.

	Ansible	▼
	Kubernetes	▼
	GitHub	▼
	Bamboo	▼
	Terraform	▼
	Jira	▼
	Splunk	▼
	Automation	

	Jenkins	▼
	Puppet	▼
	Prometheus	▼
	GitLab	▼
	Bitbucket	▼
	Apache Maven	▼
	Configuration managem...	▼
	Automation	

	Docker	▼
	Chef	▼
	Git	▼
	Nagios	▼
	CircleCI	▼
	Selenium	▼
	System integration	▼
	Automation	

DevOps Tools

DevOps Tools

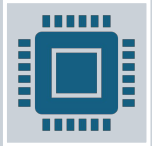
- Git
- GitHub
- Jenkins
- Docker
- Maven
- Jira
- Slack
- Kubernetes



- **Continuous integration (CI)**
- **Continuous delivery (CD)**
- **Continuous deployment**



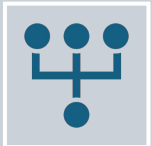
Getting started with DevOps



The term *DevOps* was introduced in 2007-2009 by Patrick Debois, Gene Kim, and John Willis, and it represents the combination of **Development (Dev)** and **Operations (Ops)**.



DevOps culture is a set of practices that reduce the barriers between **developers** and **operations**.



DevOps culture is also the extension of agile processes.

The DevOps movement is based on three axes:

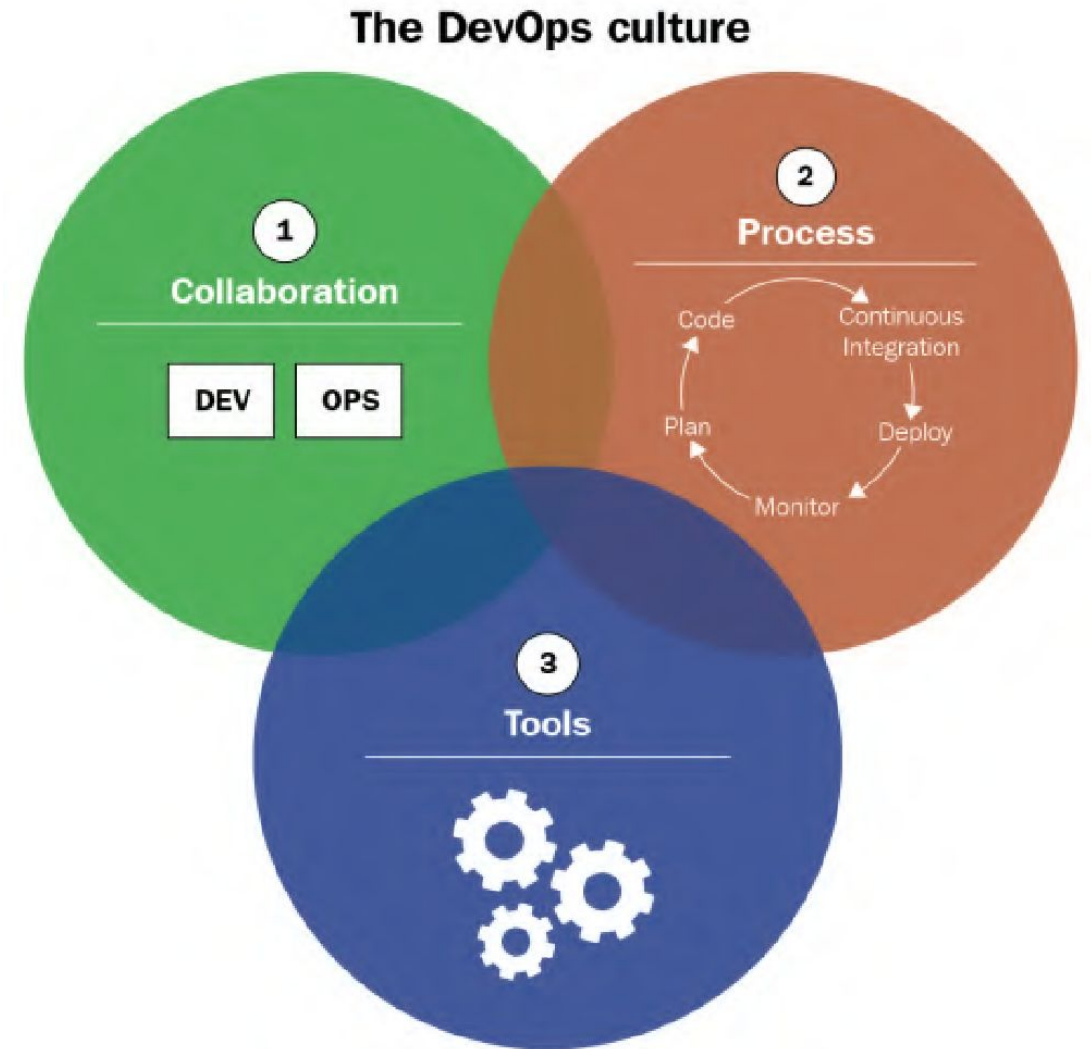
The culture of collaboration: people are brought together by making multidisciplinary teams that have the same objective.


Processes: teams must follow development processes from agile methodologies with iterative phases that allow for better functionality, quality, and rapid feedback.

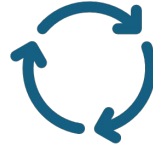
The DevOps process is divided into several phases:

- A. Planning and prioritizing functionalities
- B. Development
- C. Continuous integration and delivery
- D. Continuous deployment
- E. Continuous monitoring

Tools: The choice of tools and products used by teams is very important in DevOps.



- 
- The benefits of establishing a DevOps culture within an enterprise are as follows:
 - Better **collaboration and communication** in teams, which has a human and social impact within the company
 - Shorter lead times to production, resulting in better performance and **end user satisfaction**
 - **Reduced infrastructure costs** with IaC
 - Significant **time saved** with iterative cycles that reduce application errors and automation tools that **reduce manual tasks**, so teams focus more on developing new functionalities with added business value.



Implementing CI/CD and continuous deployment

There are three practices:

- **Continuous integration (CI)**
- **Continuous delivery (CD)**
- **Continuous deployment**

Continuous integration (CI)

"Continuous integration is a software development practice where members of a team integrate their work frequently... Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

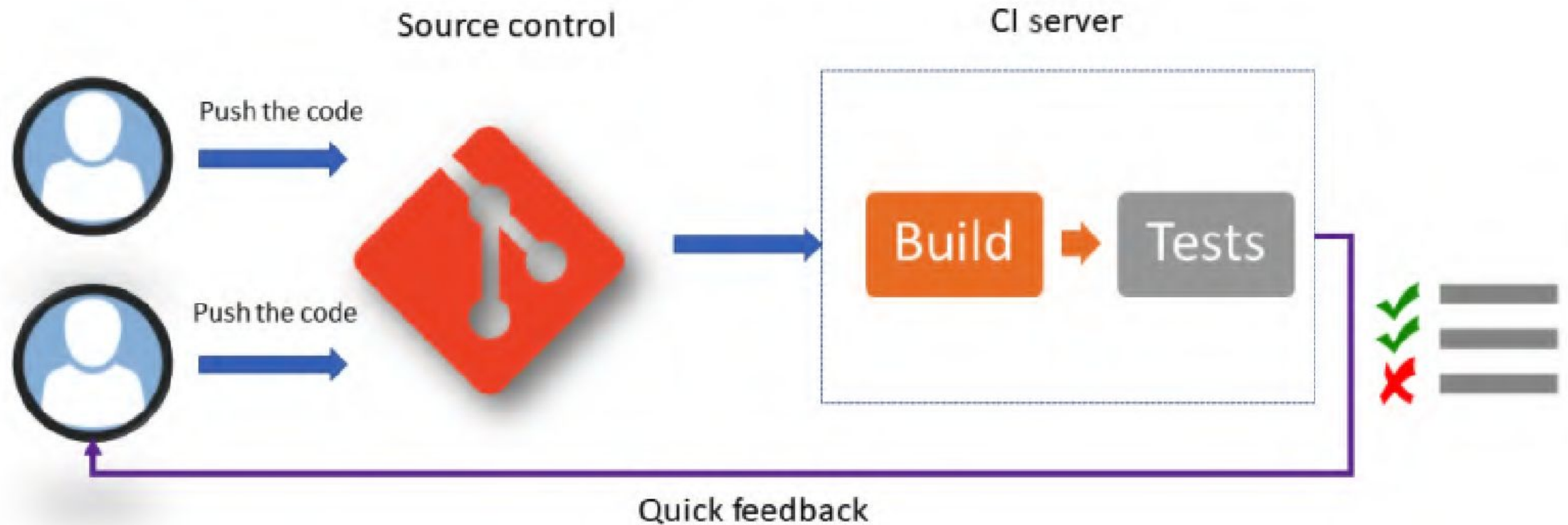
CI is an automatic process that allows you to check the completeness of an application's code every time a team member makes a change. This verification must be done as quickly as possible.

Implementing CI

Therefore, to set up CI, it is necessary to have a **Source Code Manager (SCM)** that will centralize the code of all members. This code manager can be of any type: Git, SVN, or **Team Foundation Version Control (TFVC)**. It's also important to have an automatic *build manager* (CI server) that supports continuous integration, such as Jenkins, GitLab CI, TeamCity, Azure Pipelines, GitHub Actions, Travis CI, and Circle CI.

The continuous integration workflow

The continuous integration (CI) pipeline





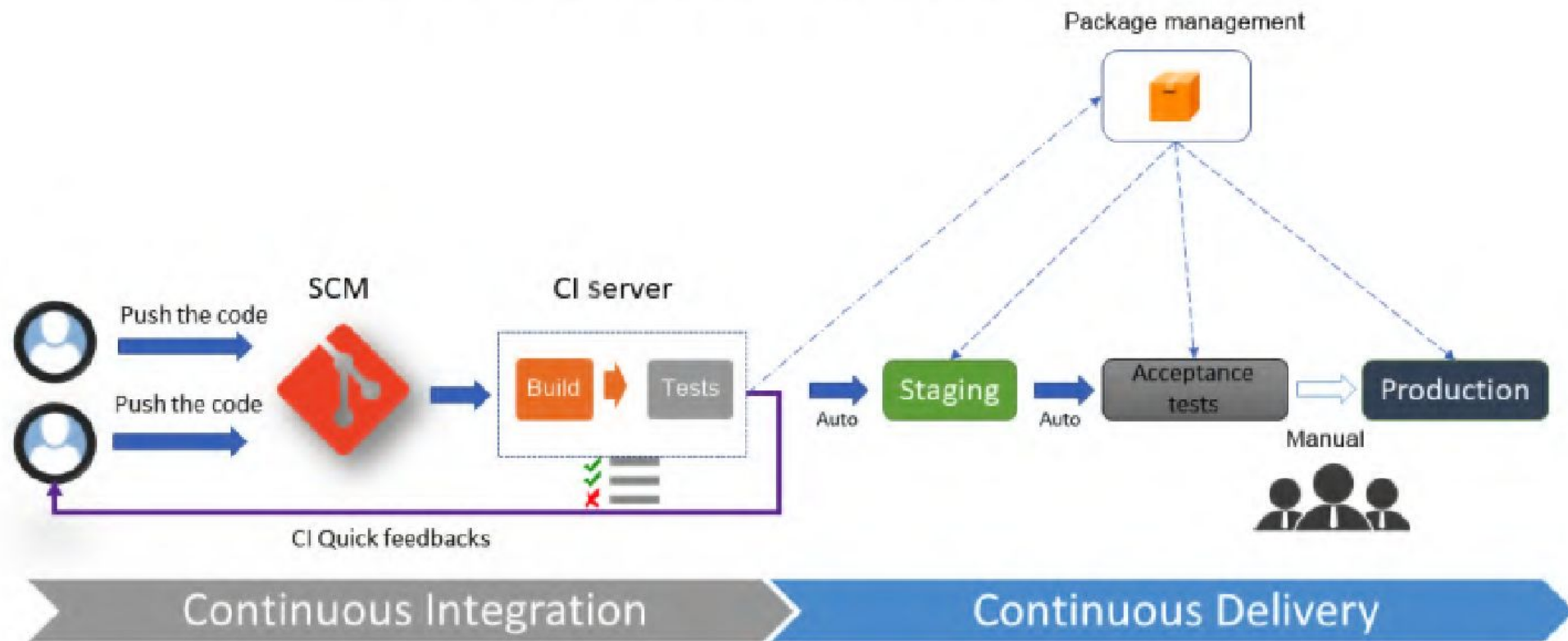
Continuous delivery (CD)

Once continuous integration has been completed, the next step is to deploy the

application automatically in one or more non-production environments, which is called **staging**. This process is called **continuous delivery (CD)**.

The continuous delivery workflow

The continuous delivery (CD) pipeline



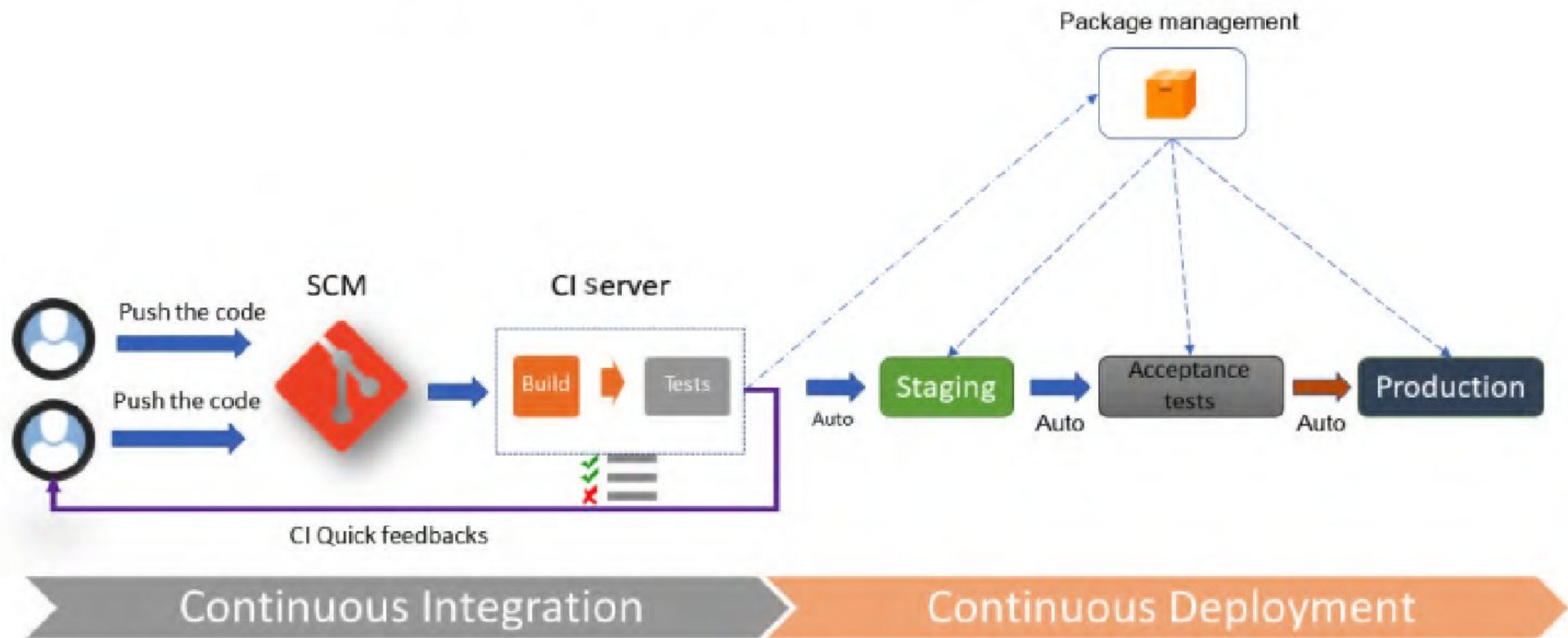
Continuous deployment

Continuous deployment is an extension of CD, but this time, with a process that automates the entire CI/CD pipeline from the moment the developer commits their code to deployment in production through all of the verification steps.



The continuous deployment workflow

The continuous deployment pipeline



Understanding IaC practices

IaC is a practice that consists of writing the code of the resources that make up an infrastructure.

IaC is the process of writing the code of the provisioning and configuration steps of infrastructure components, which helps automate its deployment in a repeatable and consistent manner.

DevOps tasks **before** automation



my-app



Prepare servers



- ▶ setup servers
- ▶ configure networking
- ▶ create route tables
- ▶ install software
- ▶ configure software
- ▶ install DB

DevOps tasks **before** automation



my-app



Setup



high human resources cost



more effort and time



more human errors possible

DevOps tasks **before** automation



my-app



Setup

Maintenance



- ▶ update versions
- ▶ deploy new release
- ▶ DB backups/updates
- ▶ recover app
- ▶ add/recover servers

DevOps tasks **after** automation



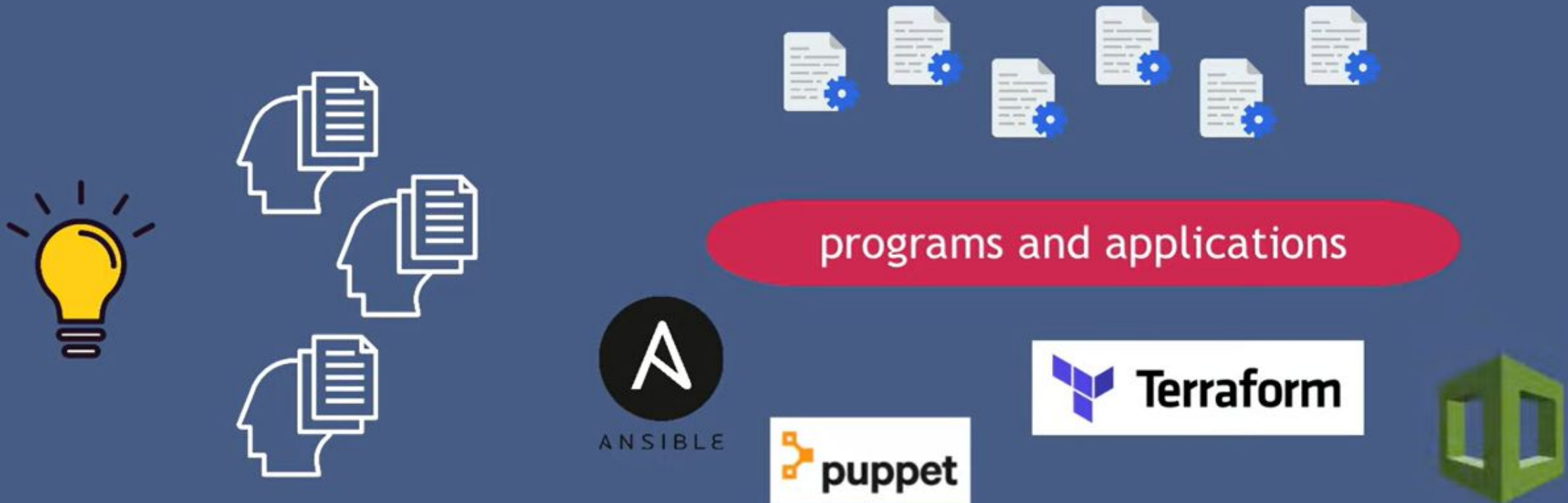
Automate complete process with

Infrastructure as Code

IaC

What is Infrastructure as Code? 🤔

- ▶ IaC is a **concept**
- ▶ IaC tools/programs, which carry out these tasks



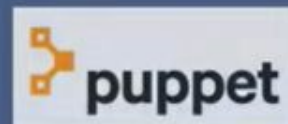
What is Infrastructure as Code? 🤔

Why so many tools? 🤔

- ▶ no tool that can do everything
- ▶ each one is good in a specific area



programs and applications



The benefits of IaC

The benefits of IaC are as follows:

- The **standardization** of infrastructure configuration reduces the risk of errors.
- The code that describes the infrastructure is **versioned** and **controlled** in a source code manager.
- The code is integrated into **CI/CD** pipelines.
- Deployments that make infrastructure **changes** are **faster** and more **efficient**.
- There's better management, control, and a reduction in infrastructure **costs**.

laC languages and tools

The languages and tools that are used to write the configuration of the infrastructure can be of different types; that is, **scripting**, **declarative**, and **programmatic**.

Scripting types

Scripting types

These are scripts such as Bash, PowerShell, or others that use the different clients (SDKs) provided by the cloud provider; for example, you can script the provisioning of an Azure infrastructure with the Azure CLI or Azure PowerShell.

Using the Azure CLI (the documentation is available at <https://bit.ly/2V1OfxJ>), we have the following:

```
az group create --location westeurope --resource-group  
MyAppResourcegroup
```

--verbose

default subscription using 'az account set -s
NAME_OR_ID'.
: Increase logging verbosity. Use --debug for full
debug logs.

Examples

Create a new resource group in the West US region.

```
az group create -l westus -n MyResourceGroup
```

To search AI knowledge base for examples, use: az find "az group create"

```
C:\Users\VVCE>az group create --location --help
```

argument --location/-l: expected one argument

Examples from AI knowledge base:

```
az group create --location westus --resource-group MyResourceGroup
```

Create a new resource group in the West US region.

```
az group create --location westeurope --resource-group MyResourceGroup --tags {tags}
```

Create a new resource group. (autogenerated)

```
az account list-locations
```

List supported regions for the current subscription. (autogenerated)

```
https://docs.microsoft.com/en-US/cli/azure/group#az\_group\_create
```

Read more about the command in reference docs

```
C:\Users\VVCE>
```

Using Azure PowerShell (the documentation is available at <https://bit.ly/2VcASeh>), we have the following:

```
New-AzResourceGroup -Name  
MyAppResourcegroup -Location  
westeurope
```



Declarative types

These are languages in which it is sufficient to write the state of the desired system or infrastructure in the form of configuration and properties.

For example, the following Terraform code allows you to define the desired configuration of an Azure resource group:

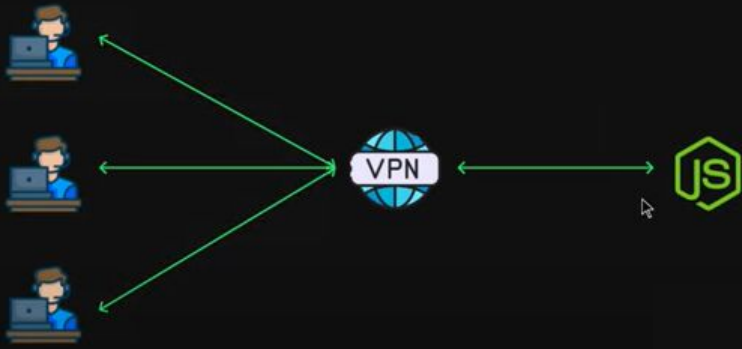
```
resource "azurerm_resource_group" "myrg" {  
  name = "MyAppResourceGroup"  
  location = "West Europe"  
  
  tags = {  
    environment = "Bookdemo"  
  }  
}
```


NGINX

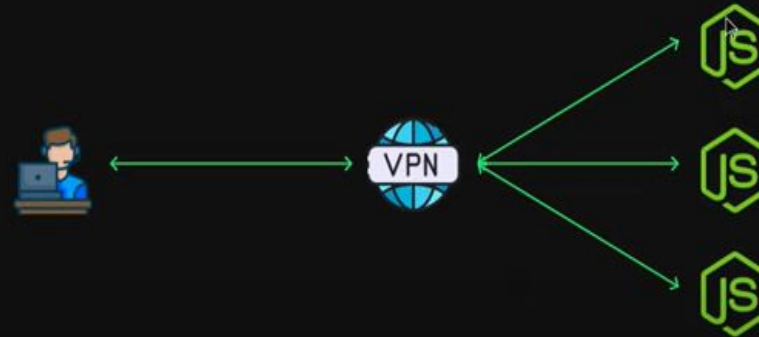
NGINX is a powerful web server and uses a non-threaded, event-driven architecture.

It can also do other important things, such as **load balancing**, and **HTTP caching**, or be used as a **reverse proxy**.

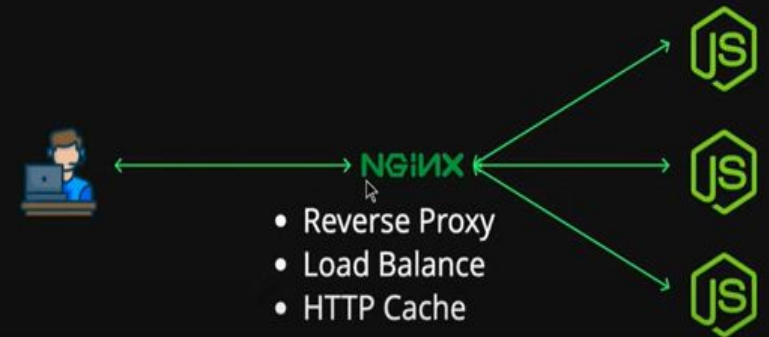
Forward Proxy



Reverse Proxy



Reverse Proxy



Here is another example that allows you to install and restart nginx on a server using Ansible:

```
---
- hosts: all
  tasks:
    - name: install and check nginx latest version
      apt: name=nginx state=latest
    - name: start nginx
      service:
        name: nginx
        state: started
```

- The first task installs Nginx using the **apt** module with the **state=latest** option, ensuring it's the latest version available in the repository.
- The second task starts the Nginx service using the **service** module with **state: started**. This ensures that Nginx is running on the server after installation.

Example

```
---  
- hosts: all  
  tasks:  
    - name: stop nginx  
  service:  
    name: nginx  
    state: stopped  
    - name: check nginx is not installed  
  apt: name=nginx state=absent
```

In this example, it was enough to change the `state` property to indicate the desired state of the service.

Programmatic types

more union between developers and operations so that we see the emergence of IaC tools that are based more on languages known by developers, such as

TypeScript, Java, Python, and C#.

The following is an example of some *TypeScript* code written with the Terraform CDK:

```
import { Construct } from 'constructs';
import { App, TerraformStack, TerraformOutput } from 'cdktf';
import {
  ResourceGroup,
} from '../.gen/providers/azurerm';
class AzureRgCDK extends TerraformStack {
  constructor(scope: Construct, name: string) {
    super(scope, name);
    new AzurermProvider(this, 'azureFeature', {
      features: [{}],
    });
    const rg = new ResourceGroup(this, 'cdktf-rg', {
      name: 'MyAppResourceGroup',
      location: 'West Europe',
    });
  }
}
const app = new App();
new AzureRgCDK(app, 'azure-rg-demo');
app.synth();
```

An overview of Packer

Packer is part of the HashiCorp open source suite of tools, and this is the official Packer page: <https://www.packer.io/>. It's an open source command-line tool that allows us to create custom VM images of any OS (these images are also called **templates**) on several platforms from a JSON file.

Installing Packer

Packer, like Terraform, is a cross-platform tool and can be installed on Windows, Linux, or macOS. The installation of Packer is almost identical to the Terraform installation and can be done in two ways: either manually or via a script.



Installing manually

To install Packer manually, use the followings steps:

1. Go to the official download page (<https://www.packer.io/downloads.html>) and download the package corresponding to your operating system.
2. After downloading, unzip and copy the binary into an execution directory (for example, inside c:\Packer).
3. Then, the PATH environment variable must be set with the path to the binary directory.

Installing by script

It is also possible to install Packer with an automatic script that can be installed on a remote server and be used on a CI/CD process.

Installing Packer by script on Windows

On Windows, we can use **Chocolatey**, which is a software package manager. Chocolatey is a free public package manager, like NuGet or npm, but dedicated to software. It is widely used for the automation of software on Windows servers or even local machines. Chocolatey's official website is here: <https://chocolatey.org/>, and its installation documentation is here: <https://chocolatey.org/install>.



Install Packer

Once Chocolatey is installed, we just need to run the following command in PowerShell or in the CMD tool:

choco install packer -y

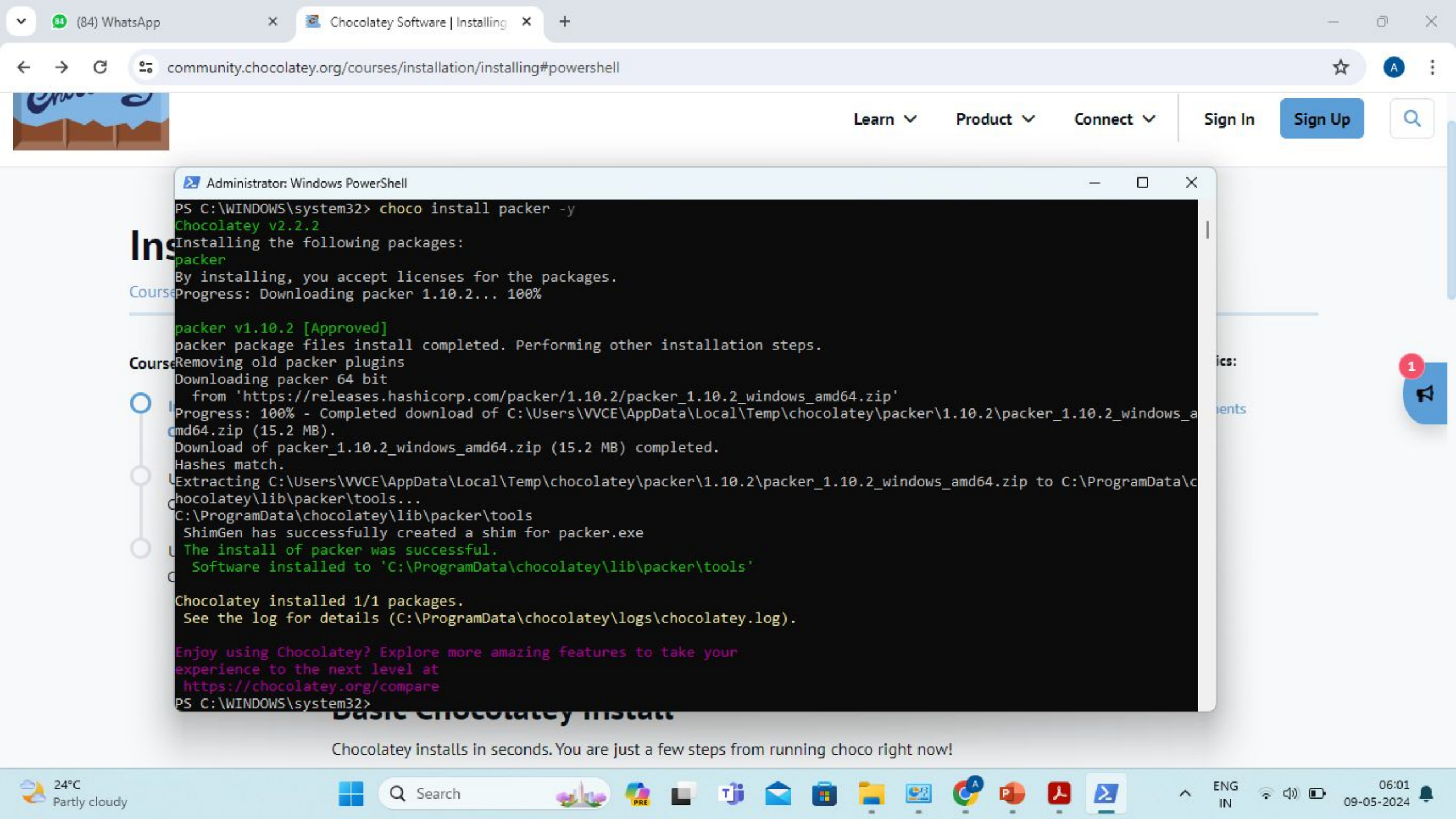
The following is a screenshot of the Packer installation for Windows with Chocolatey:

The execution of `choco install packer -y` installs the latest version of Packer from Chocolatey.

```
PS C:\Windows\system32> choco install packer -y
Chocolatey v0.10.11
Installing the following packages:
packer
By installing you accept licenses for the packages.
Progress: Downloading packer 1.4.0... 100%

packer v1.4.0 [Approved]
packer package files install completed. Performing other installation steps.
Removing old packer plugins
Downloading packer 64 bit
  from 'https://releases.hashicorp.com/packer/1.4.0/packer_1.4.0_windows_amd64.zip'
Progress: 100% - Completed download of C:\Users\MikaelKRIEF\AppData\Local\Temp\chocolatey\packer\1.4.0\packer_1.4.0_windows_amd64.zip (33.61 MB).
Download of packer_1.4.0_windows_amd64.zip (33.61 MB) completed.
Hashes match.
Extracting C:\Users\MikaelKRIEF\AppData\Local\Temp\chocolatey\packer\1.4.0\packer_1.4.0_windows_amd64.zip to C:\ProgramData\chocolatey\lib\packer\tools...
C:\ProgramData\chocolatey\lib\packer\tools
ShimGen has successfully created a shim for packer.exe
The install of packer was successful.
  Software installed to 'C:\ProgramData\chocolatey\lib\packer\tools'

Chocolatey installed 1/1 packages.
  See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\Windows\system32>
```

Administrator: Windows PowerShell

```
PS C:\WINDOWS\system32> choco install packer -y
Chocolatey v2.2.2
Installing the following packages:
packer
By installing, you accept licenses for the packages.
Progress: Downloading packer 1.10.2... 100%

packer v1.10.2 [Approved]
packer package files install completed. Performing other installation steps.
Removing old packer plugins
Downloading packer 64 bit
from 'https://releases.hashicorp.com/packer/1.10.2/packer_1.10.2_windows_amd64.zip'
Progress: 100% - Completed download of C:\Users\VVCE\AppData\Local\Temp\chocolatey\packer\1.10.2\packer_1.10.2_windows_a
amd64.zip (15.2 MB).
Download of packer_1.10.2_windows_amd64.zip (15.2 MB) completed.
Hashes match.
Extracting C:\Users\VVCE\AppData\Local\Temp\chocolatey\packer\1.10.2\packer_1.10.2_windows_amd64.zip to C:\ProgramData\c
chocolatey\lib\packer\tools...
C:\ProgramData\chocolatey\lib\packer\tools
ShimGen has successfully created a shim for packer.exe
The install of packer was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\packer\tools'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
https://chocolatey.org/compare
PS C:\WINDOWS\system32>
```

Chocolatey installs in seconds. You are just a few steps from running choco right now!

Checking the Packer installation

- Once installed, we can check the installed version of Packer by running the following
- command:
- **packer --version**
- This command displays the installed Packer version:



```
PS C:\Users\mkrief> packer --version
1.7.3
```

```
packer --help
```

After executing, we will see a list of available commands, as shown in the following screenshot:

```
PS C:\Users\mkrief> packer --help
Usage: packer [--version] [--help] <command> [<args>]

Available commands are:
  build      build image(s) from template
  console    creates a console for testing variable interpolation
  fix        fixes templates from old versions of packer
  fmt        Rewrites HCL2 config files to canonical format
  hcl2_upgrade transform a JSON template into an HCL2 configuration
  init       Install missing plugins or upgrade plugins
  inspect    see components of a template
  validate   check that a template is valid
  version    Prints the Packer version
```