RAMAIAH
Institute of Technology

Taking DevOps Further

RAMAIAH
Institute of Technology

Security in the DevOps Process with DevSecOps

Introduction

- The DevSecOps culture or approach is the union of developers and operations with the integration of security as early as possible in the implementation and design of projects.
- The DevSecOps approach is also the automation of compliance and security verification processes in CI/CD pipelines, to guarantee constant security and not slow down application deployment cycles.
- Today, DevOps culture must absolutely integrate security teams but also all security processes, whether on tools, infrastructure, or applications.
- This is to provide better quality but also more secure applications.

Testing Azure infrastructure compliance with Chef InSpec

- One of the important practices of DevOps culture is IaC, which consists of coding the configuration of an infrastructure and then being automatically deployed via CI/CD pipelines.

- IaC allows cloud infrastructure to be deployed and provisioned very quickly.

- But the question that often arises is: Does the automatically-provisioned infrastructure meet functional compliance and security requirements?

Testing Azure infrastructure compliance with Chef InSpec

- To answer this question, we'll have to write and automate infrastructure tests that will verify the following:
  - The infrastructure deployed corresponds well to the application and enterprise architecture specifications.
  - The company's security policies are properly applied to the infrastructure.

- These tests can be written in any scripting language that can interact with our cloud provider and if we have an Azure subscription, then use Azure CLI or PowerShell Azure command to code the tests of our Azure resources.

Overview of InSpec

- InSpec is an open source tool written in Ruby that runs on the command line, and is produced by one of the leading DevOps tools, Chef.
- It allows users writing declarative-style code, to test the compliance of a system or infrastructure.
- To use InSpec, it's not necessary to learn a new scripting language.
- Should already have enough knowledge to write the desired state of the infrastructure resources or the system we want to test.
- With InSpec, we can test the compliance of remote machines and data and, since the latest version, it is also possible to test a cloud infrastructure such as Azure, AWS, and GCP.

Configuring Azure for InSpec

- Before writing test cases to test the compliance of our Azure infrastructure, need to create an Azure service principal that has read permission on the Azure resources that will be tested.
- To create this Azure service principal, use the same procedure that was discussed with Configuring Terraform for Azure.
- Using the Azure CLI tool, execute the following az cli command:

```
az ad sp create-for-rbac -name="<SP name> -role="Reader" -
scopes="/subscriptions/<subscription Id>"
```

Configuring Azure for InSpec

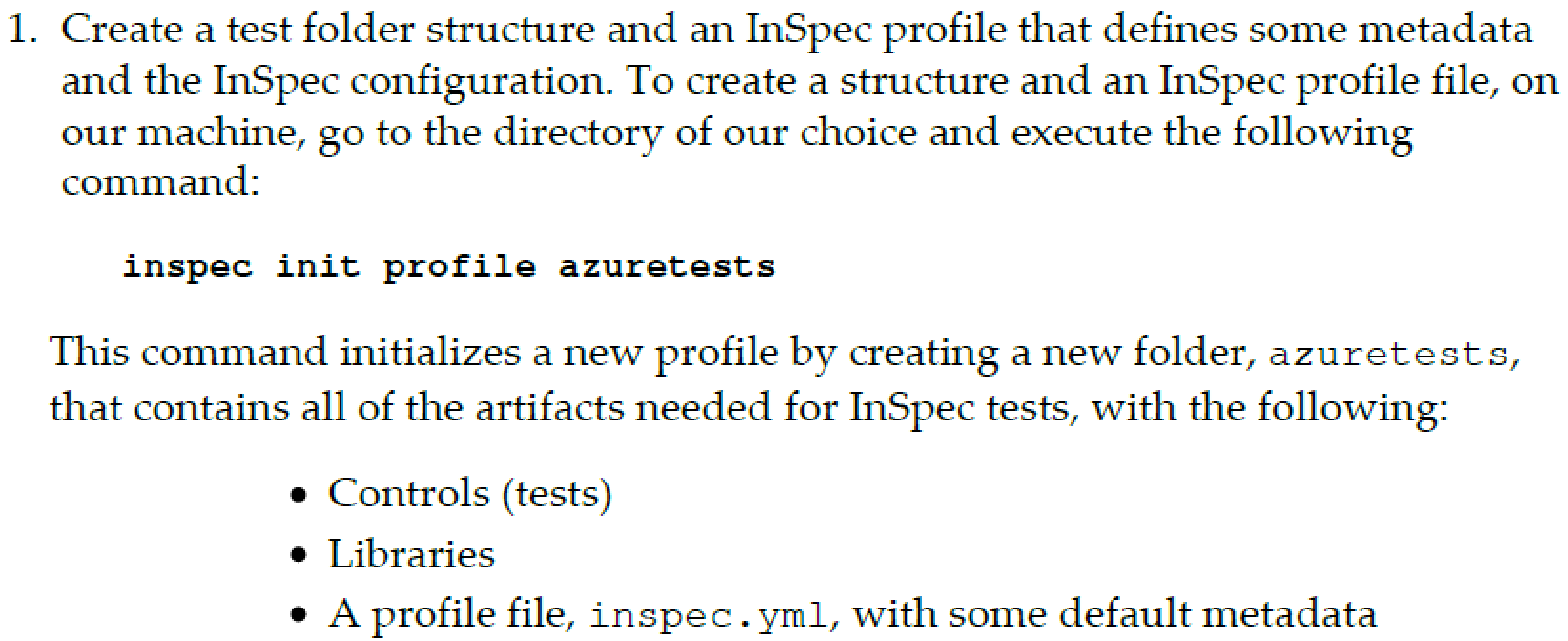
- This preceding command requires the following parameters:
  - **name** is the name of the Azure service principal to be created.
  - **scopes** is the ID of the subscription (or other scopes) in which the Azure resources will be present.
  - **role** is the role name that the service principal (SP) will have on the specified resource scope.
- The execution of this preceding command returns the three pieces of authentication information of the created service principal:
  - The client ID
  - The client secret
  - The tenant ID

Writing InSpec tests

- To show an example of InSpec tests, write tests that will check that the Azure infrastructure that was provisioned with Terraform is compliant with the specifications of our Azure infrastructure.
- This must be composed of the following:
- One resource group named bookRg
- One VNet with one subnet inside it named book-subnet
- One virtual machine named bookvm

Creating an InSpec profile file

- To create an InSpec profile file, generate the test directory structure, then modify the InSpec profile file that was generated.



1. Create a test folder structure and an InSpec profile that defines some metadata and the InSpec configuration. To create a structure and an InSpec profile file, on our machine, go to the directory of our choice and execute the following command:

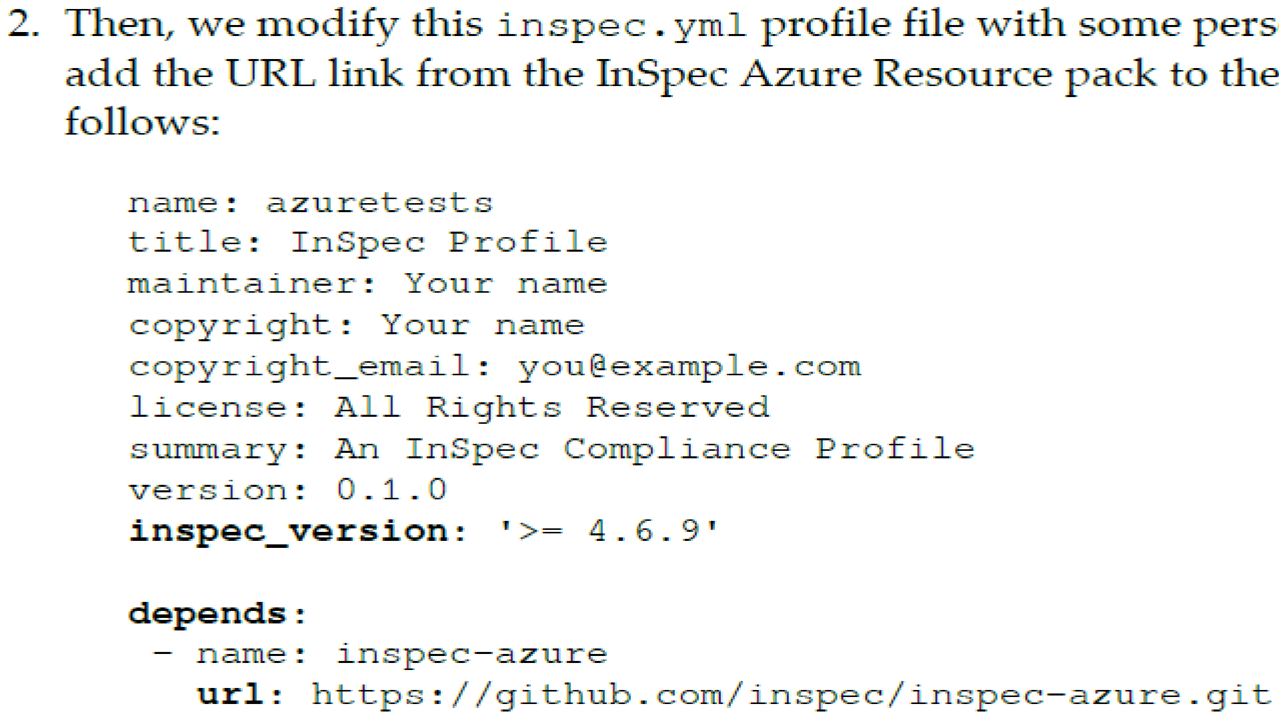
```
inspec init profile azuretests
```

This command initializes a new profile by creating a new folder, `azuretests`, that contains all of the artifacts needed for InSpec tests, with the following:

- Controls (tests)
- Libraries
- A profile file, `inspec.yml`, with some default metadata

Creating an InSpec profile file

- To create an InSpec profile file, generate the test directory structure, then modify the InSpec profile file that was generated.



2. Then, we modify this `inspec.yml` profile file with some personal metadata and add the URL link from the InSpec Azure Resource pack to the sample code, as follows:

```
name: azuretests
title: InSpec Profile
maintainer: Your name
copyright: Your name
copyright_email: you@example.com
license: All Rights Reserved
summary: An InSpec Compliance Profile
version: 0.1.0
inspec_version: '>= 4.6.9'

depends:
 - name: inspec-azure
   url: https://github.com/inspec/inspec-azure.git
```

Executing InSpec

- To execute InSpec, perform the following steps:

1. Configure the InSpec authentication to Azure; for this, create environment variables with the values of the Azure Service Principal information.

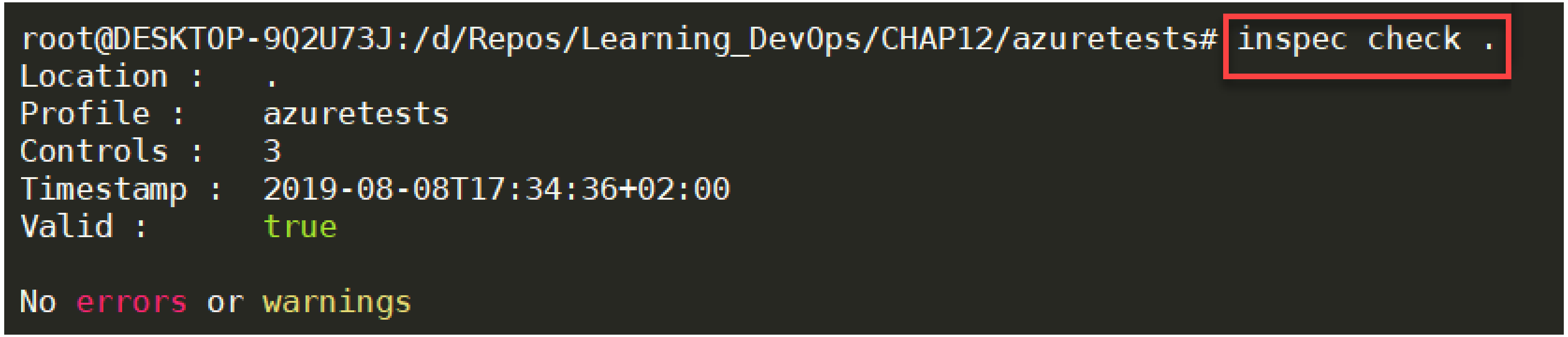
- The four environments variables and their values are as follows:
- AZURE-CLIENT-ID with the client ID of the service principal
- AZURE-CLIENT-SECRET with the secret client of the service principal
- AZURE-TENANT-ID with the tenant ID
- AZURE-SUBSCRIPTION-ID with the ID of the subscription that contains the resources and whose service principal has reader permissions.

Executing InSpec

- To execute InSpec, perform the following steps:

2. Then, in a Terminal, we'll place ourselves in the directory containing the profile file, inspec.yml, then execute the following inspec command to check that the syntax of the tests is correct:

- *inspec check*



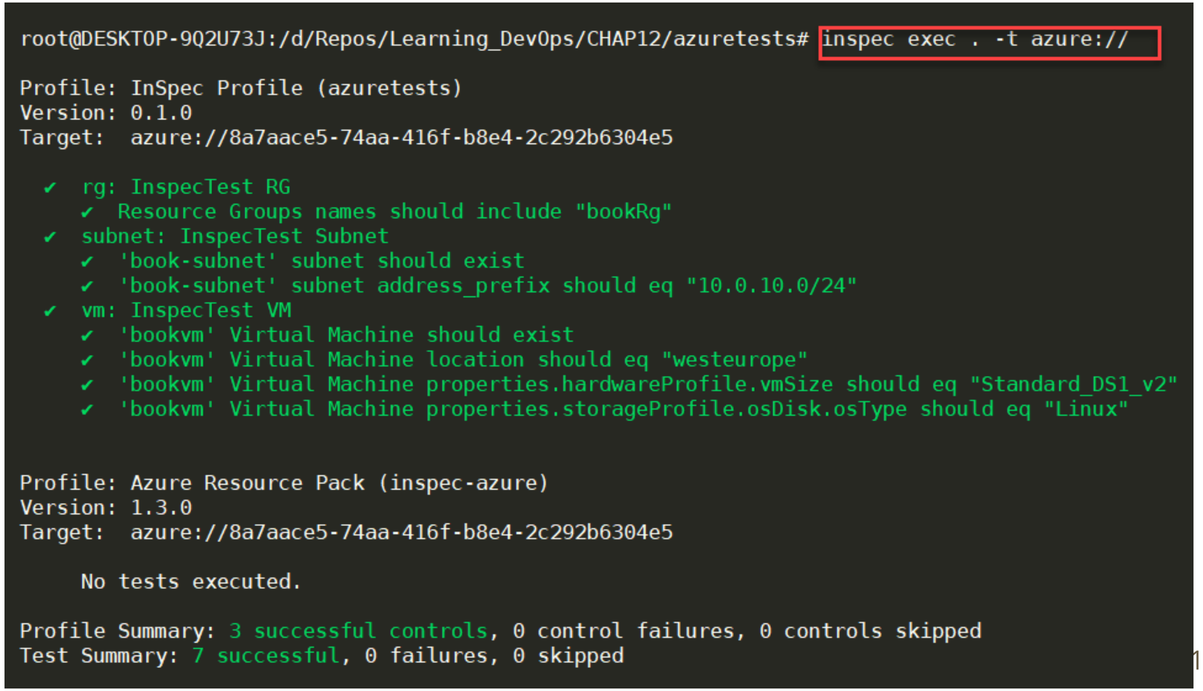
```
root@DESKTOP-9Q2U73J:/d/Repos/Learning_DevOps/CHAP12/azuretests# inspec check .
Location :    .
Profile :     azuretests
Controls :    3
Timestamp :   2019-08-08T17:34:36+02:00
Valid :       true

No errors or warnings
```

Executing InSpec

- To execute InSpec, perform the following steps:

2. Finally,execute InSpec to execute the tests with the inspec exec command, as follows:

inspec exec . -t azure://



```
root@DESKTOP-9Q2U73J:/d/Repos/Learning_DevOps/CHAP12/azuretests# inspec exec . -t azure://

Profile: InSpec Profile (azuretests)
Version: 0.1.0
Target:  azure://8a7aace5-74aa-416f-b8e4-2c292b6304e5

  ✔  rg: InspecTest RG
     ✔  Resource Groups names should include "bookRg"
  ✔  subnet: InspecTest Subnet
     ✔  'book-subnet' subnet should exist
     ✔  'book-subnet' subnet address_prefix should eq "10.0.10.0/24"
  ✔  vm: InspecTest VM
     ✔  'bookvm' Virtual Machine should exist
     ✔  'bookvm' Virtual Machine location should eq "westeurope"
     ✔  'bookvm' Virtual Machine properties.hardwareProfile.vmSize should eq "Standard_DS1_v2"
     ✔  'bookvm' Virtual Machine properties.storageProfile.osDisk.osType should eq "Linux"


Profile: Azure Resource Pack (inspec-azure)
Version: 1.3.0
Target:  azure://8a7aace5-74aa-416f-b8e4-2c292b6304e5

     No tests executed.

Profile Summary: 3 successful controls, 0 control failures, 0 controls skipped
Test Summary: 7 successful, 0 failures, 0 skipped
```

Using the Secure DevOps Kit for Azure

- Another very interesting tool that allows to check the security of Azure infrastructure resources.
- It is a tool provided by Microsoft, called Secure DevOps Kit for Azure (AzSK).
- Unlike InSpec, AzSK does not verify the compliance of Azure infrastructure with architectural requirements but rather will verify that the recommendations and good security practices are applied to Azure subscription and resources.
- AzSK also integrates seamlessly into a CI/CD pipeline and thus allows developers and operational staff to continuously ensure that their Azure resources are secure and do not open security vulnerabilities to unwanted people.

Installing the Azure DevOps Security Kit

- To install the Azure DevOps Security Kit, the following actions must be performed:

1. In a PowerShell Terminal, execute the following command:

- Install-Module AzSK -Scope CurrentUser
- Executing this command installs the AzSK module from the PowerShell public package manager called PowerShell Gallery.

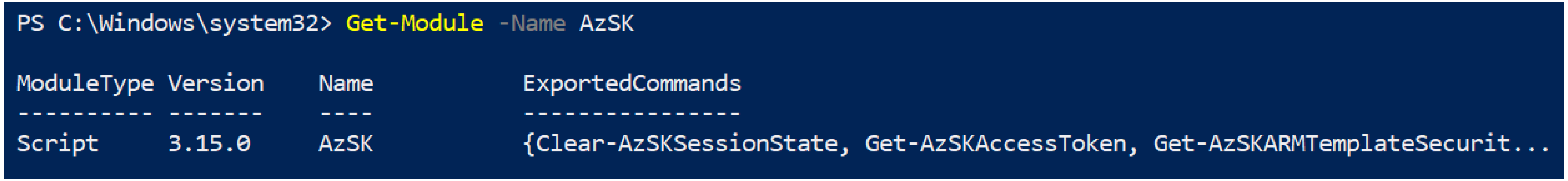
2. Then, for importing the module, we execute the following command:

- Import-Module -Name AzSK
- This command imports the module into the user context of the PowerShell execution.

Installing the Azure DevOps Security Kit

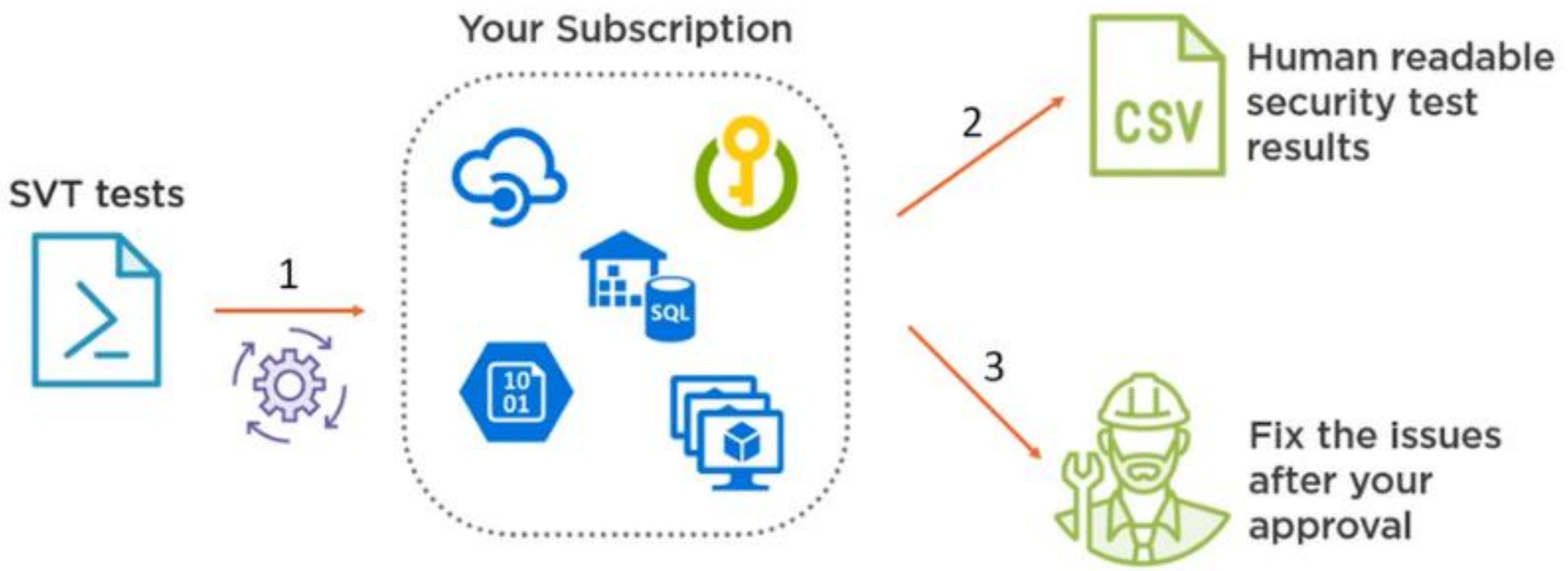
3. Finally, to check that the AzSK module is installed correctly, we run the following PowerShell command:

- Get-Module -Name AzSK



```
PS C:\Windows\system32> Get-Module -Name AzSK

ModuleType Version    Name             ExportedCommands
---------- -------    ----             ----------------
Script     3.15.0     AzSK             {Clear-AzSKSessionState, Get-AzSKAccessToken, Get-AzSKARMTemplateSecurit...
```

Working of AzSK

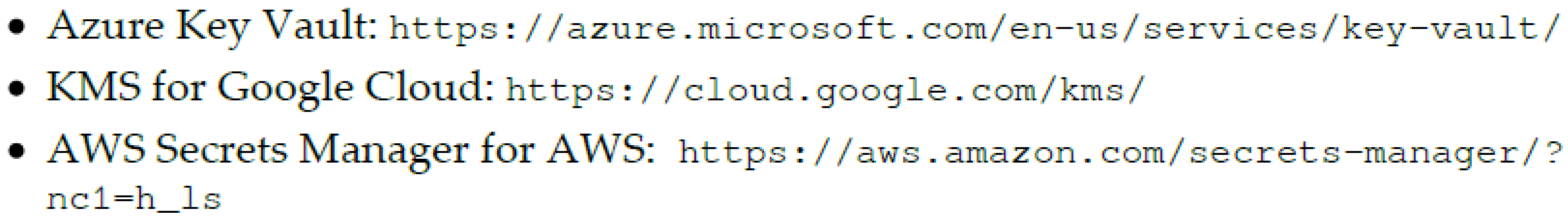

Your Subscription
SVT tests
1
2
3
CSV
SQL
10 01
Human readable security test results
Fix the issues after your approval

Preserving data with HashiCorp's Vault

- Today, when we talk about security in information systems, the most expected topic is the protection of sensitive data between different components of the system.
- This sensitive data that needs to be protected includes server access passwords, database connections, API authentication tokens, and application user accounts.
- Indeed, many security attacks occur because this type of data is decrypted in the source code of applications or in poorly protected files that are exposed to local workstations.

Preserving data with HashiCorp's Vault

- There are many known tools that can be used to secure this sensitive data, such as these:
  - KeyPass (https://keepass.info/)
  - LastPass (https://www.lastpass.com/)
  - Ansible Vault, the use of which we discussed Chapter 3, Using Ansible for Configuring IaaS Infrastructure
  - Vault from HashiCorp
- Also, cloud providers offer secret protection services such as the following:



- Azure Key Vault: `https://azure.microsoft.com/en-us/services/key-vault/`
- KMS for Google Cloud: `https://cloud.google.com/kms/`
- AWS Secrets Manager for AWS: `https://aws.amazon.com/secrets-manager/?nc1=h_ls`

Main Features and Benefits of Vault

- It allows the storage of static secrets as well as dynamic secrets.
- It also has a system for rotating and revoking secrets.
- It allows data to be encrypted and decrypted without having to store it.
- It also has a web interface that allows the management of secrets.
- It integrates with a multitude of authentication systems.
- All secrets are stored in a single centralized tool.
- It allows you to be independent of your architecture by being accessible with all major cloud providers, Kubernetes, or even on internal data centers (onpremises).

Installing Vault locally

- If you decide to use Vault, it is important to know that Vault is a tool that is responsible for the security of your sensitive infrastructure and application data.
- In practice, Vault is not just a tool, and before installing it in production, you need to understand its concepts and its different architectural topologies.
- The purpose of this chapter is therefore not to go into the details of the concepts and architecture of Vault but to explain the installation and use of Vault in development mode.
- In other words, we'll install Vault on a local workstation to have a small instance that's used for tests and development.

Installing Vault locally

- Vault can be installed either manually or via a script.
- To **install Vault manually**, the procedure is exactly the same as that of installing Terraform and Packer, so you need to do the following:
    1. Navigate to the download page: https://www.vaultproject.io/downloads.html.
    2. Download the package related to your OS in the folder of your choice.
    3. Unzip the package and update the PATH environment variable with its path to this folder.
- To **install Vault automatically**, we'll use a script, the code of which depends on our OS.

Installing Vault locally

For **Linux**, in a Terminal, run the following script:

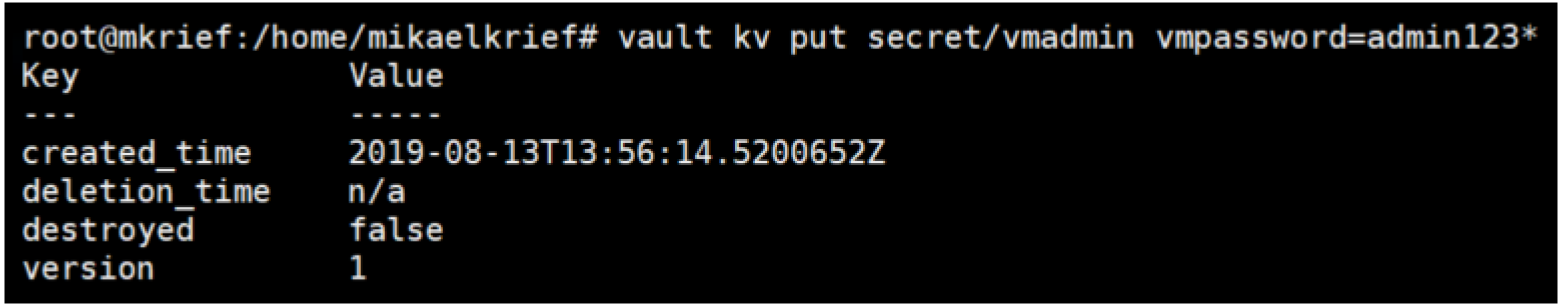
```
VAULT_VERSION="1.2.1"
curl --silent --remote-name
https://releases.hashicorp.com/vault/${VAULT_VERSION}/vault_${VAULT_VERSION
}_linux_amd64.zip
unzip vault_${VAULT_VERSION}_linux_amd64.zip
sudo mv vault /usr/local/bin/
```

This script performs the following actions:

1. It initializes a variable that contains the version of the vault to download.
2. It downloads Vault with curl. Curl is a tool for downloading any URL content, and can be downloaded from `https://curl.haxx.se/`.
3. It unzips the package.
4. It copies the vault binary into the `/usr/local/bin` folder (which is already filled in the `PATH` environment variable).

Writing secrets in Vault

- When you want to protect sensitive data that will be used by an application or infrastructure resources, the first step is to store this data in the secret data manager that has been chosen by the company.
- To protect data in Vault, we go to a Terminal and execute the following command:
- **vault kv put secret/vmadmin vmpassword=admin123***



```
root@mkrief:/home/mikaelkrief# vault kv put secret/vmadmin vmpassword=admin123*
Key                 Value
---                 -----
created_time        2019-08-13T13:56:14.5200652Z
deletion_time       n/a
destroyed           false
version             1
```

Writing secrets in Vault

- The command, with the put operation, creates a new secret data in memory with the title vmadmin of the key-value type, which in this example is the admin account of a VM, in the secret/ path.
- In Vault, all protected data is stored in a path that corresponds to an organizational location in Vault.
- The default path for Vault is secret/, and it is possible to create custom paths that will allow better management of secret rights and better organization by domain, topic, or application.
- About the secrets stored in Vault, one of its advantages is that it is possible to store multiple data in the same secret; for example, we'll update the secret data that we have created with another secret, which is the login admin of the VM.

Writing secrets in Vault

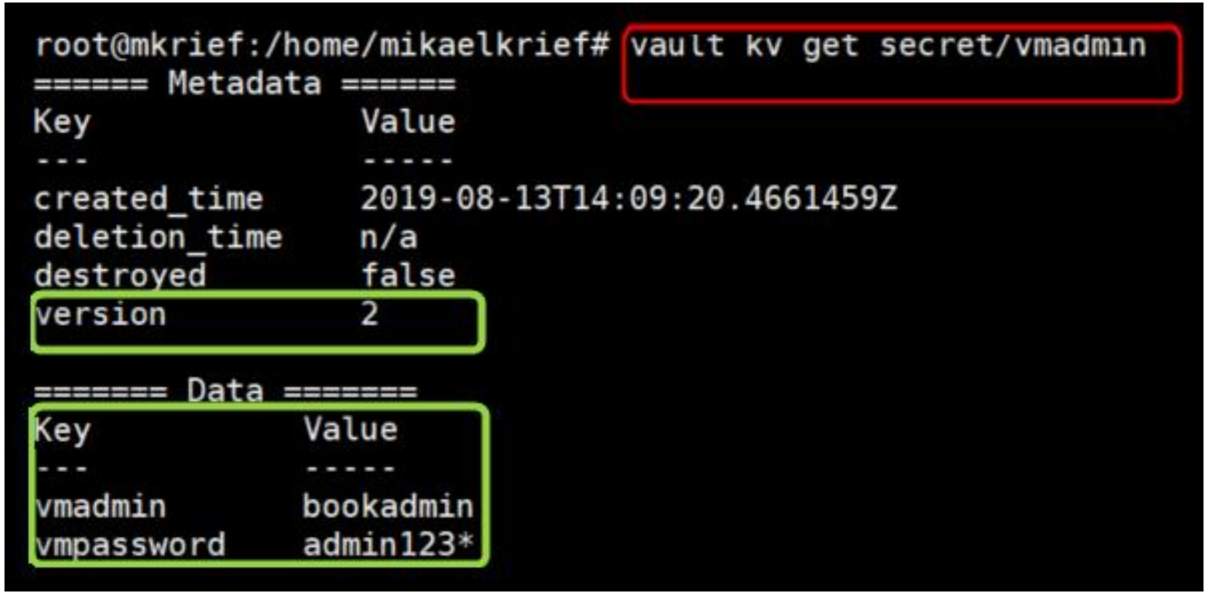
- For this, we'll execute the following command that adds another key-value secret in the same Vault data:
- **vault kv put secret/vmadmin vmpassword=admin123* vmadmin=bookadmin**
- As we can see in this execution, we used exactly the same command with the same secret, and we added new key-value data, that is, vmadmin.

Reading secrets in Vault

- Once we have created secrets in Vault, we'll have to read them to use them in our applications or infrastructure scripts.
- To read a key that is stored in a vault, we go to a Terminal to execute this command:
- **vault kv get secret/vmadmin**
- In this command, we use the kv operation with the get operator, and we indicate in the parameter the complete path of the key to get the protected value within our example, secret/vmadmin.

Reading secrets in Vault

- The following screenshot shows the command execution, as well as its output:

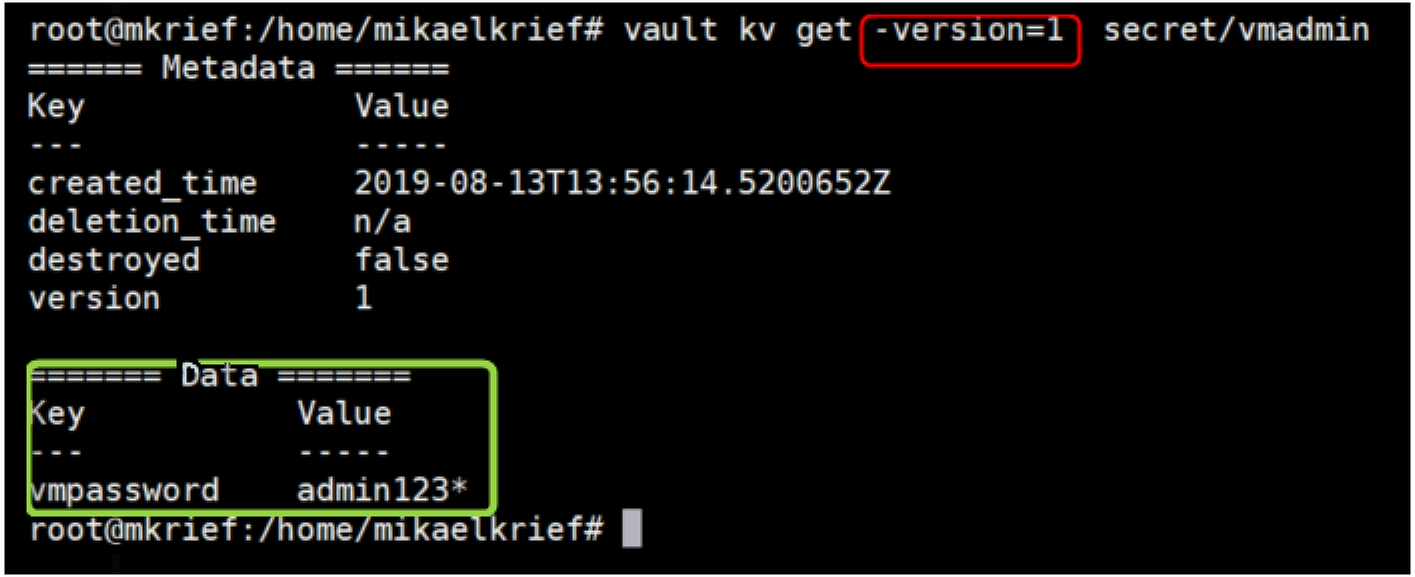


```
root@mkrief:/home/mikaelkrief# vault kv get secret/vmadmin
====== Metadata ======
Key              Value
---              -----
created_time     2019-08-13T14:09:20.4661459Z
deletion_time    n/a
destroyed        false
version          2

======= Data =======
Key           Value
---           -----
vmadmin       bookadmin
vmpassword    admin123*
```

Reading secrets in Vault

- What we notice in the output of this command is the following:
    - The version number of the secret here is 2 because we executed the kv put command twice, so the version number was incremented at each execution.
    - There are two key-value data items that we protected in secret in the previous section, Writing secrets in Vault.
- If you want to access the data stored in this secret but from an earlier version, we can execute the same command by optionally specifying the desired version number, as in this example:
- **vault kv get -version=1 secret/vmadmin**

Reading secrets in Vault



```
root@mkrief:/home/mikaelkrief# vault kv get -version=1 secret/vmadmin
====== Metadata ======
Key              Value
---              -----
created_time     2019-08-13T13:56:14.5200652Z
deletion_time    n/a
destroyed        false
version          1

======= Data =======
Key           Value
---           -----
vmpassword    admin123*
root@mkrief:/home/mikaelkrief#
```

RAMAIAH
Institute of Technology

Reducing Deployment Downtime

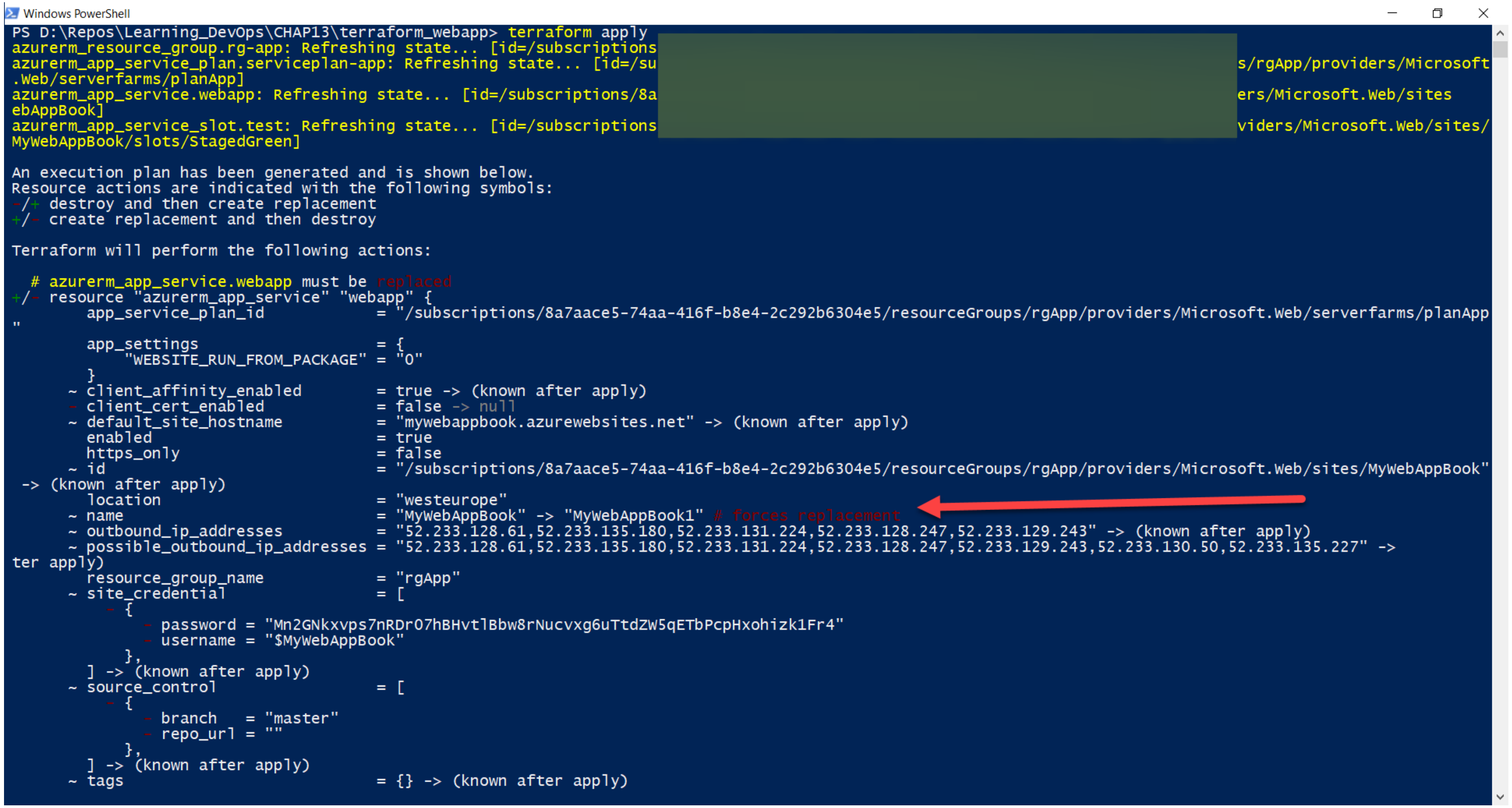
Introduction

- We will now go deeper into DevOps practices by looking at how to ensure the continuous availability of your applications even during your deployments, and how to deliver new versions of these applications more frequently in production.
- Often, what we see is that deployments require your applications to be interrupted by, for example, infrastructure changes or service shutdowns.
- Moreover, what we also see is that companies are still reluctant to deliver more frequently in production.
- They are not equipped to test the application in the production environment or they are waiting for other dependencies.

Reducing deployment downtime with Terraform

- In Chapter 2, Provisioning Cloud Infrastructure with Terraform, we detailed the use of Terraform by looking at its commands and life cycle and put it into practice with an implementation in Azure.
- One of the problems with Terraform is that, depending on the infrastructure changes that need to be implemented, Terraform may automatically destroy and rebuild certain resources.
- To fully understand this behavior, let's look at the output of this following Terraform execution, which provisioned a web app in Azure and has been modified with a name change:

Reducing deployment downtime with Terraform



```
PS D:\Repos\Learning_DevOps\CHAP13\terraform_webapp> terraform apply
azurerm_resource_group.rg-app: Refreshing state... [id=/subscriptions[redacted]
azurerm_app_service_plan.serviceplan-app: Refreshing state... [id=/su[redacted]s/rgApp/providers/Microsoft
.Web/serverfarms/planApp]
azurerm_app_service.webapp: Refreshing state... [id=/subscriptions/8a[redacted]ers/Microsoft.Web/sites
ebAppBook]
azurerm_app_service_slot.test: Refreshing state... [id=/subscriptions[redacted]viders/Microsoft.Web/sites/
MyWebAppBook/slots/StagedGreen]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement
+/- create replacement and then destroy

Terraform will perform the following actions:

  # azurerm_app_service.webapp must be replaced
+/- resource "azurerm_app_service" "webapp" {
        app_service_plan_id            = "/subscriptions/8a7aace5-74aa-416f-b8e4-2c292b6304e5/resourceGroups/rgApp/providers/Microsoft.Web/serverfarms/planApp
"
        app_settings                   = {
            "WEBSITE_RUN_FROM_PACKAGE" = "0"
        }
      ~ client_affinity_enabled        = true -> (known after apply)
      - client_cert_enabled            = false -> null
      ~ default_site_hostname          = "mywebappbook.azurewebsites.net" -> (known after apply)
        enabled                        = true
        https_only                     = false
      ~ id                             = "/subscriptions/8a7aace5-74aa-416f-b8e4-2c292b6304e5/resourceGroups/rgApp/providers/Microsoft.Web/sites/MyWebAppBook"
 -> (known after apply)
        location                       = "westeurope"
      ~ name                           = "MyWebAppBook" -> "MyWebAppBook1" # forces replacement
      ~ outbound_ip_addresses          = "52.233.128.61,52.233.135.180,52.233.131.224,52.233.128.247,52.233.129.243" -> (known after apply)
      ~ possible_outbound_ip_addresses = "52.233.128.61,52.233.135.180,52.233.131.224,52.233.128.247,52.233.129.243,52.233.130.50,52.233.135.227" ->
ter apply)
        resource_group_name            = "rgApp"
      ~ site_credential                = [
          - {
              - password = "Mn2GNkxvps7nRDr07hBHvtlBbw8rNucvxg6uTtdZW5qETbPcpHxohizk1Fr4"
              - username = "$MyWebAppBook"
            },
        ] -> (known after apply)
      ~ source_control                 = [
          - {
              - branch   = "master"
              - repo_url = ""
            },
        ] -> (known after apply)
      ~ tags                           = {} -> (known after apply)
```

Reducing deployment downtime with Terraform

- Here, we can see that Terraform will destroy the web app and then rebuild it with the new name.
- Although destruction and reconstruction are done automatically, during this period of time in which Terraform will destroy and rebuild the web app, the application will be inaccessible to users.
- To solve this problem of downtime, we can add the Terraform create-before-destroy option:

```
resource "azurerm_app_service" "webapp" {
    name = "MyWebAppBook1" #new name
    location = "West Europe"
    resource_group_name = "${azurerm_resource_group.rg-app.name}"
    app_service_plan_id = "${azurerm_app_service_plan.serviceplan-app.id}"
    app_settings = {
       WEBSITE_RUN_FROM_PACKAGE = var.package_zip_url"
    }
     lifecycle { create_before_destroy = true}
 }
```

Reducing deployment downtime with Terraform

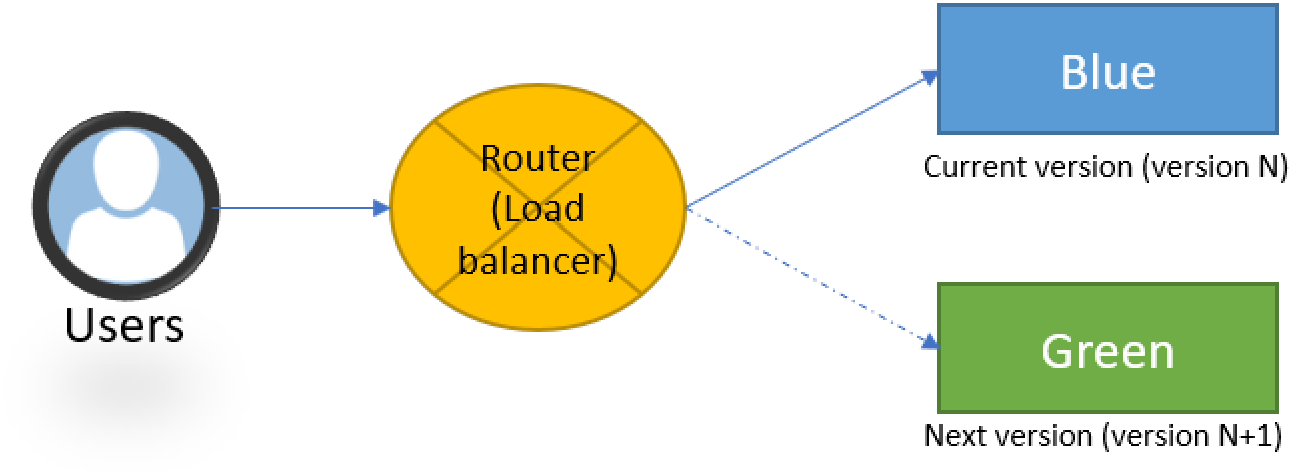
- By adding this option, Terraform will do the following:
    1. First, Terraform creates the new web app with a new name.
    2. During the provisioning of the new web app, it uses the URL for the application package in ZIP format that's provided in the app_settings property. Use WEBSITE_RUN_FROM_PACKAGE to launch the application.
    3. Then, Terraform will destroy the old web app.
- Using the Terraform create_before_destroy option will ensure the viability of our applications during deployments.
- Be careful, though: this option will only be useful if the new resource that's being created allows us to have the application running very quickly at the same time as it's provisioning so that a service interruption doesn't occur.

Understanding blue-green deployment concepts & patterns

- Blue-green deployment is a practice that allows us to deploy a new version of an application in production without impacting the current version of the application.
- In this approach, the production architecture must be composed of two identical environments; one environment is known as the blue environment while the other is known as the green environment.
- The element that allows routing from one environment to another is a router, that is, a load balancer.

Understanding blue-green deployment concepts & patterns

- The following diagram shows a simplified schematic of a blue-green architecture:



Understanding blue-green deployment concepts & patterns

- As we can see, there are two identical environments: the environment called blue, which is the current version of the application, and the environment called green, which is the new version or the next version of the application.
- We can also see a router, which redirects users' requests either to the blue environment or the green environment.
- Now that we've introduced the principle of blue-green deployment, we will look at how to implement it in practice during deployment.

Using blue-green deployment to improve the production environment

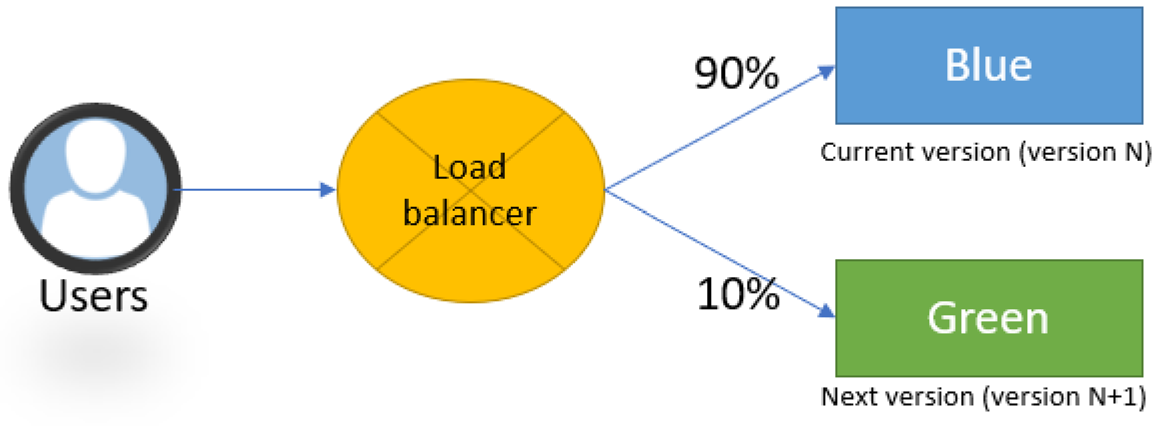
- The basic usage pattern of the blue-green deployment is that when we're deploying new versions of the application, the application is deployed in the blue environment (version N) and the router is configured in this environment.
- When deploying the next version (version N+1), the application will be deployed in the green environment, and the router is configured in this environment.
- The blue environment becomes unused and idle until the deployment of version N+2.
- It also will be used in the case of rapid rollback to version N.
- This practice of blue-green deployment can also be declined on several patterns, that is, the canary release and dark launch patterns.

Understanding the canary release pattern

- The canary release technique is very similar to blue-green deployment.
- The new version of the application is deployed in the green environment, but only for a small restricted group
- of users who will test the application in real production conditions.
- This practice is done by configuring the router (or load balancer) to be redirected to both environments.
- On this router, we apply redirection restrictions of a user group so that it can only be redirected to the green environment, which contains the new version.

Understanding the canary release pattern

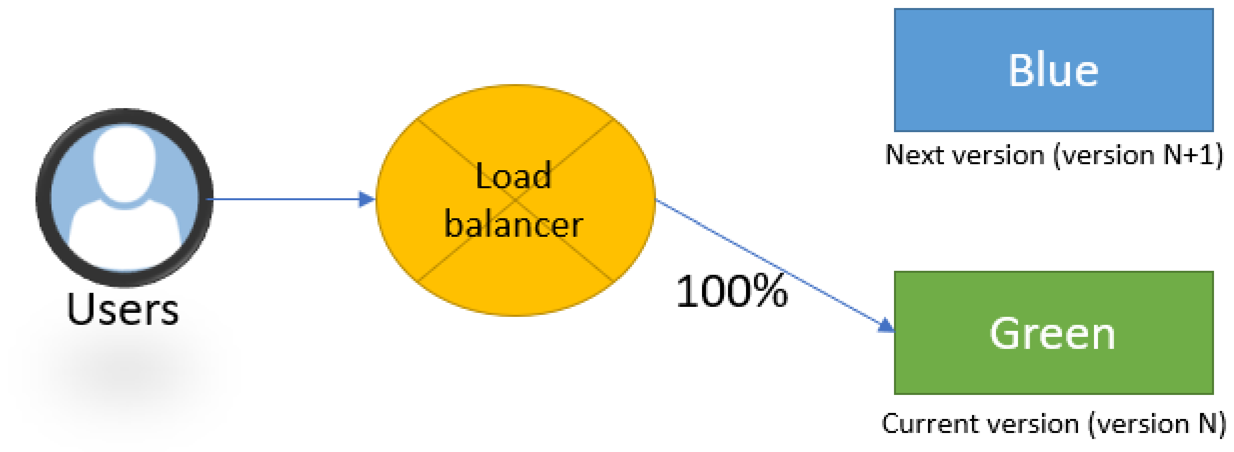
- The following is a sample diagram of the canary release pattern:



Understanding the canary release pattern

- In the preceding diagram, the router redirects 90% of users to the blue environment and 10% of users to the green environment, which contains the new version of the application.
- Then, once the tests have been performed by this user group, the router can be fully configured in the green environment, thus leaving the blue environment free for testing the next version (N+2).
- As shown in the following diagram, the router is configured to redirect all users to the green environment:

Understanding the canary release pattern



- This deployment technique, therefore, makes it possible to deploy and test the application in the real production environment without having to impact all users.

Exploring the dark launch pattern

- The dark launch pattern is another practice related to blue-green deployment that consists of deploying new features in hidden or disabled mode into the production environment.
- Then, when we want to have access to these features in the deployed application, we can activate them as we go along without having to redeploy the application.
- Unlike the canary release pattern, the dark launch pattern is not a blue-green deployment that depends on the infrastructure but is implemented in the application code.
- To set up the dark launch pattern, it is necessary to encapsulate the code of each feature of the application in elements called feature flags (or feature toggle), which will be used to enable or disable these features remotely.

Introducing feature flags

- Feature flags (also called feature toggle) allow us to dynamically enable or disable a feature of an application without having to redeploy it.
- Unlike the blue-green deployment with the canary release pattern, which is an architectural concept, feature flags are implemented in the application's code.
- Their implementation is done with a simple encapsulation using conditional if rules, as shown in the following code example:

```
if(activateFeature("addTaxToOrder")==True) {
    ordervalue = ordervalue + tax
}else{
    ordervalue = ordervalue
}
```

Introducing feature flags

- In this example code, the activateFeature function allows us to find out whether the application should add the tax to order according to the addTaxToOrder parameter, which is specified outside the application (such as in a database or configuration file).
- Features encapsulated in feature flags may be necessary either for the running of the application or for an internal purpose such as log activation or monitoring.
- The activation and deactivation of features can be controlled either by an administrator or directly by users via a graphical interface.

Introducing feature flags

- The lifetime of a feature flag can be either of the following:
    - **Temporary**: To test a feature. Once validated by users, the feature flag will be deleted.
    - **Definitive**: To leave a feature flagged for a long time.
- Thus, using feature flags, a new version of an application can be deployed to the production stage faster.
- This is done by disabling the new features of the release.
- Then, we will reactivate these new features for specific group of users like testers, who will test these features directly in production.

Introducing feature flags

- Moreover, if we notice that one of the application's functionalities is not working properly, it is possible for the features flags to disable it very quickly, without us having to redeploy the rest of the application.
- Feature flags also allow A/B testing, that is, testing the behavior of new features by certain users and collecting their feedback.
- There are several technical solutions when it comes to implementing feature flags in an application:

1. You develop and maintain your custom feature flags system, which has been adapted to your business needs. This solution will be suitable for your needs but requires a lot of development time, as well as the necessary considerations of architecture specifications such as the use of a database, data security, and data caching.

Introducing feature flags

2. You use an open source tool that you must install in your project. This solution allows us to save on development time but requires a choice of tools, especially in the case of open source tools. Moreover, among these tools, few offer portal or dashboard administration that allows for the management of features flags remotely. There is a multitude of open source frameworks and tools for feature flags.

3. You can use a cloud solution (PaaS) that requires no installation and has a back office for managing features flags, but most of them require a financial investment for large-scale use in an enterprise.

RAMAIAH
Institute of Technology

DevOps for Open Source Projects

Introduction

- Until a few years ago, the open source practice, which consists of delivering the source code of a product to the public, was essentially only used by the Linux community.
- Since then, many changes have taken place in relation to open source with the arrival of GitHub.
- Microsoft has since made a lot of its products open source and is also one of the largest contributors to GitHub.
- Today, open source is a must in the development and enterprise world, regardless of whether we wish to use a project or even contribute to it.

Storing the source code in GitHub

- If we want to share one of our projects in an open source fashion, we must version its code in a Git platform that allows the following elements:
- **Public repositories**; that is, we need to have access to the source code contained in this repository, but without necessarily being authenticated on this Git platform.
- **Features and tools** for code collaboration between the different members of this platform.
- There are two main platforms that allow us to host open source tools: GitLab and GitHub

Introduction

- Until a few years ago, the open source practice, which consists of delivering

RAMAIAH
Institute of Technology

DevOps Best Practices

Automating everything

- When you want to implement DevOps practices within a company, it is important to remember the purpose of the DevOps culture: it delivers new releases of an application faster, in shorter cycles.
- To do this, the first good practice to apply is to automate all tasks that deploy, test, and secure the application and its infrastructure.
- Indeed, when a task is done manually, there is a high risk of error in its execution.
- The fact that these tasks are performed manually increases the deployment cycles of applications.
- In addition, once these tasks are automated in scripts, they can be easily integrated and executed in CI/CD pipelines.

Automating everything

- Another advantage of automation is that developers and the operational team can spend more time and focus their work on the functionality of their business.
- It is also important to start the automation of the delivery process at the beginning of project development; this allows us to provide feedback faster and earlier.
- Finally, automation makes it possible to improve the monitoring of deployments by putting traces on each action and allows you to make a backup and restore very quickly in case of a problem.

Automating everything

- Automating deployments will, therefore, reduce deployment cycles and the teams can now afford to work in smaller iterations.
- Thus, the time to market will be improved, with the added benefit of better-quality applications.
- However, automation and orchestration require tools to be implemented, and the choice of these tools is an important element to consider in the implementation of a DevOps culture.

Choosing the right tool

- One of the challenges a company faces when it wants to apply a DevOps culture is the choice of tools.
- Indeed, there are many tools that are either paid for or free and open source and that allow you to version the source code of applications, process automation, implement CI/CD pipelines, and test and monitor applications.
- Along with these tools, scripting languages are also added, such as PowerShell, Bash, and Python, which are also part of the DevOps suite of tools to integrate.
- The important point of DevOps culture is the union of Dev, Ops, processes, and also tools.

Choosing the right tool

- That is to say, the tools used must to be shared and usable by both Devs and Ops, and should be integrated into the same process.
- In other words, the choice of tools depends on the teams and the company model.
- It is also necessary to take into account the financial system, by choosing open source tools, which are often free of charge; this is easier to use them at the beginning of the DevOps transformation of the company.
- That is not the case of paid tools, which are certainly richer in features and support, but require a significant investment.

Writing all your configuration in code

- Writing the desired infrastructure configuration in code offers many advantages both for teams and for the company's productivity.
- It is, therefore, a very good practice to put everything related to infrastructure configuration in code.
- Seen this in practice with Terraform, Ansible, and Packer, but there are many other tools that may be better suited based on needs.
- Apart from the major editors the use of JSON files, Bash, PowerShell, and Python scripts are also used for this configuration.
- The key is to have a description of your infrastructure in code, that is easy for a human to read, and tools that are adapted to you.

Writing all your configuration in code

- Moreover, this practice continues to evolve in other fields, as seen with the use of Inspec, which allows to describe the compliance rules of the infrastructure in code.
- Putting any configuration in code is a key practice of the DevOps culture to take into account from the beginning of projects for both Ops and developers.
- For developers, there are also good practices about the designing of the application and infrastructure architecture.

Designing the system architecture

- A few years ago, all the services of the same application were coded in the same application block.
- This architecture design was legitimate since the application was managed in waterfall mode, so new versions of the application were deployed in very long cycles.
- Since then, many changes have taken place in software engineering practices, starting with the adoption of the agile method and DevOps culture, then continuing with the arrival of the cloud.
- This evolution has brought many improvements, not only in applications but also in their infrastructure.

Designing the system architecture

- However, in order to take advantage of an effective DevOps culture to deploy an application in the cloud, there are good practices to consider when designing software architecture and also when designing infrastructure.
- First of all, cloud architects must work hand in hand with developers (or solution architects) to ensure that the application developed is in line with the different components of the architecture and that the architecture also takes into consideration the different constraints of the application.
- In addition to this collaboration, security teams must also provide their specifications, which will be implemented by developers and cloud architects.

Designing the system architecture

- In order to be able to deploy a new version of the application more frequently without having to impact all of its features, it is good practice to separate the different areas of the application into separate code at first, then into different departments at a later stage.
- Thus, the separate code will be much more maintainable and scalable, and can be deployed faster without having to redeploy everything.
- Once decoupled, however, there is still a need to control dependency in order to implement a CI/CD pipeline that takes into account all the dependencies of the application.
- Another good practice that allows deploying in production more frequently and it consists of encapsulating the functionalities of the application in feature flags.

Designing the system architecture

- These feature flags must also be taken into account when designing the application, as they allow the application to be deployed in the production stage, enabling/disabling its features dynamically without having to redeploy it.
- Finally, the implementation of unit tests and the log mechanism must be taken into account as soon as possible in the development of the application, because they allow feedback on the state of the application to be shared very quickly in its deployment cycle.
- DevOps culture involves the implementation of CI/CD pipelines and this requires changes in the design of applications, with the separation of functionalities to create less monolithic applications, the implementation of tests, and the addition of a log system.

Building a good CI/CD pipeline

- Creation of CI/CD pipelines using different tools such as GitLab CI, Jenkins, and Azure Pipelines etc.
- Building a good CI/CD pipeline is indeed an essential practice in DevOps culture and, together with the correct choice of tools, allows faster deployment and better-quality applications.
- One of the best practices for CI/CD pipelines is to set them up as early as the project launch stage.
- This is especially true for the CI pipeline, which will allow the code (at least the compilation step) to be verified when writing the first lines of the code.

Building a good CI/CD pipeline

- Then, as soon as the first environment is provisioned, immediately create a deployment pipeline, which will allow the application to be deployed and tested in this environment.
- The rest of the CI/CD pipeline process's tasks, such as unit test execution, can be performed as the project progresses.
- In addition, it is also important to optimize the processes of the CI/CD pipeline by having pipelines that run quickly.
- This is used to provide quick feedback to team members (especially for CI) and also to avoid blocking the execution queue of other pipelines that may be in the queue.

Building a good CI/CD pipeline

- Thus, if some pipelines take too long to run, such as integration tests, which can be long, it may be a good idea to schedule their execution for hours with less activity, such as at night.
- Finally, it is also important to protect sensitive data embedded in CI/CD pipelines.
- So if you use a configuration manager tool in your pipelines, do not leave information such as passwords, connection strings, and tokens visible to all users.
- To protect this data, use centralized secret management tools such as Vault or use Azure Key Vault.

Integrating tests

- Testing is, in today's world, a major part of the DevOps process, but also of development practices.
- Indeed, it is possible to have the best DevOps pipeline that automates all delivery phases, but without the integration of tests, it loses almost all its efficiency.
- The minimum requirement for a DevOps process is to integrate at least the execution of the unit tests of the application.
- In addition, these unit tests must be written from the first line of code of the application using testing practices such as **Test-Driven Development** and **Behavior-Driven Development** (BDD).

Integrating tests

- In this way, the automatic execution of these tests can be integrated into the CI pipeline.
- However, it is important to integrate other types of tests, such as functional tests or integration tests, that allow the application to be tested functionally from start to finish with the other components of its ecosystem.
- It is certainly true that the execution of these tests can take time; in this case, it is possible to schedule their execution at night.
- But these integration tests are the ones that will guarantee the quality of the application's smooth operation during all stages of delivery until deployment to production.

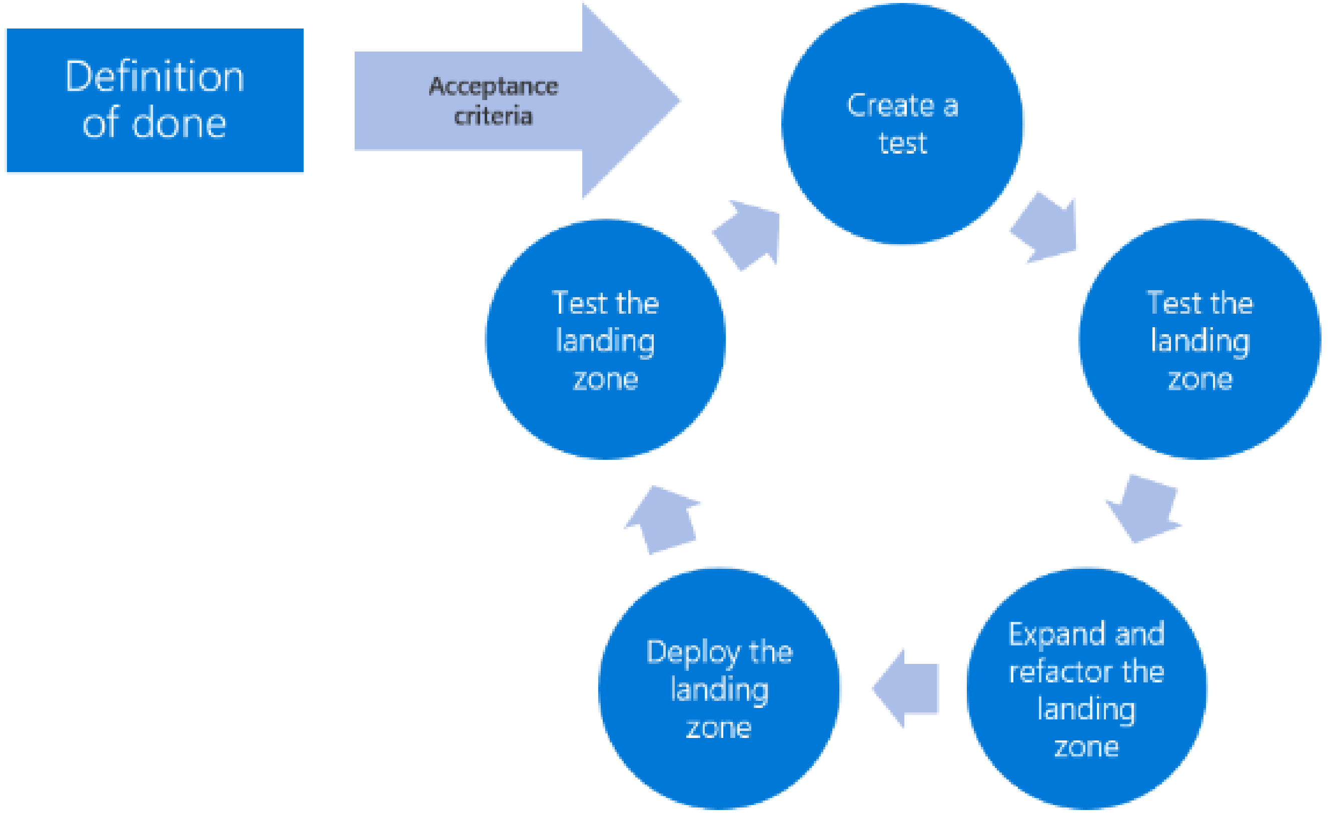
Integrating tests

- Testing is, in today's world, a major part of the DevOps process, but also of development practices.
- Indeed, it is possible to have the best DevOps pipeline that automates all delivery phases, but without the integration of tests, it loses almost all its efficiency.
- However, there is often a very bad practice of disabling the execution of tests in the CI pipeline in case their execution fails.
- This is often done to avoid the blocking of the complete CI process and thus deliver faster in production.

Integrating tests

- But keep in mind that the errors detected by the unit tests, including the ones that would have been detected by tests that have been disabled, will be detected in the production stage at some point and the correction of the failed code will cost more time than if the tests had been enabled during CI.
- Other types of tests, such as code analysis tests or security tests, which are not to be ignored.
- The sooner they are integrated into CI/CD pipelines, the more value will be added for maintaining code and securing our application.
- To summarize, you should not ignore the implementation of tests in your applications and their integration in CI/CD pipelines, as they guarantee the quality of your application.

Test Driven Development



Test Driven Development

- **Create a test:** Define a test to validate that acceptance criteria for a feature has been met. Automate the test as you develop, to reduce the amount of manual test effort, especially for enterprise-scale deployments.
- **Test the landing zone:** Run the new test and any existing tests. If the required feature isn't included in the cloud provider's offerings and hasn't been provided by prior development efforts, the test should fail. Running existing tests helps validate that your new feature or test doesn't reduce the reliability of existing landing zone features.
- **Expand and refactor the landing zone:** Add or modify source code to fulfill the requested value-add feature and improve the general quality of the code base.

Test Driven Development

- **Deploy the landing zone:** Once the source code fulfills the feature request, deploy the modified landing zone to the cloud provider in a controlled testing or sandbox environment.
- **Test the landing zone:** Retest the landing zone to validate that the new code meets the acceptance criteria for the requested feature. Once all tests pass, the feature is considered complete and the acceptance criteria are considered met.

Behavior-Driven Development

- Behavior-driven development (BDD) is an Agile software development methodology in which an application is documented and designed around the behavior a user expects to experience when interacting with it.
- By encouraging developers to focus only on the requested behaviors of an app or program, BDD helps to avoid bloat, excessive code, unnecessary features or lack of focus.
- This methodology combines, augments and refines the practices used in test-driven development (TDD) and acceptance testing.
- A typical project using behavior-driven development would begin with a conversation between the developers, managers and customer to form an overall picture of how a product is intended to work.

Behavior-Driven Development

- The expectations for the product's behavior are then set as goals for the developers, and once all of the behavior tests are passed the product has met its requirements and it ready for delivery to the customer.
- Behavior-driven development revolves around conducting behavior-specific tests, or functional specifications that outline executable scenarios for the application.
- This includes:
  - Applying the 5 Whys principle or the if-then scenario to generate user stories and clearly relate application features to a business purpose.
  - Identifying a single outcome for every behavior.

Behavior-Driven Development

- This includes:
  - Translating each scenario into domain specific language (DSL) to ensure accurate communication.
  - Gathering all behaviors into one set of documentation so it is accessible for all developers, testers and stakeholders.

Applying security with DevSecOps

- Security and compliance analyses must be part of DevOps processes.
- However, in companies, there is often a lack of awareness among development teams about security rules and this is why security is implemented too late in DevOps processes.
- To integrate security into processes, it is, therefore, necessary to raise awareness among developers of aspects of application code security, but also of the protection of CI/CD pipeline configuration.
- In addition, it is also necessary to eliminate the barrier between DevOps and security, by integrating security teams more often into the various meetings that bring together Developer and Operational teams.

Applying security with DevSecOps

- Thus ensuring better consistency between developers, operational team, and also security.
- Regarding the choice of tools, do not use too many different tools, because the goal is for these tools to also be used by developers and be integrated into CI/CD pipelines.
- It is, therefore, necessary to select a few tools that are automated, do not require great knowledge of security, and provide reports for better analysis.
- Finally, if you don't know where to start when it comes to analyzing the security of the application, work with simple security rules that are recognized by communities, such as the top 10 OWASP rules.

Applying security with DevSecOps

- You can use the ZAP tool, which uses these 10 rules, to perform security tests on a web application.
- You can also use the very practical checklist at **https://www.sqreen.com/checklists/devsecopssecurity-checklist** to check security points in a DevOps process.
- These are some good practices for integrating security into DevOps culture to in order to achieve DevSecOps culture.

Monitoring your system

- One of the main elements for the success of DevOps culture is the implementation of tools that will continuously monitor the state of a system and applications.
- The monitoring must be implemented at all levels of the system by involving all teams, with the aim of having applications with real added value for the end user.
- Indeed, the first component that can be monitored is the application itself, by implementing, as soon as possible, a log or trace system that will serve to gather information on the use of the application.
- Then, measure and monitor the state of the infrastructure, such as the RAM and CPU level of the VMs or the network bandwidth.

Monitoring your system

- Finally, the last element that must be monitored is the status of DevOps processes.
- It is therefore important to have metrics on the execution of CI/CD pipelines, such as information on the execution time of pipelines, or the number of pipelines that are in success or failure.
- With this data, for example, we can determine the deployment speed of an application.
- There are many monitoring tools, such as Graphana, New Relic, Prometheus, and Nagios, and others are integrated into the various cloud providers, such as Azure Application Insight or Azure Log Monitor.

Monitoring your system

- Concerning good practices for monitoring, it is important to target the KPIs that are necessary for you and that are easy to analyze.
- It is useless to have a monitoring system that captures a lot of data or an application that writes a lot of logs, as it's too time-consuming when it comes to analyzing this information.
- In addition, on the volume of the captured data, we need to take care in ensuring its retention.
- The retention time of the data must be evaluated and different teams should be consulted.

Monitoring your system

- Too much retention can cause capacity saturation on VMs or high costs for managed components in the cloud, and with too little retention, the log history is shorter and therefore you can lose track of any problems.
- Finally, when choosing the tool, you must make sure that it protects all the data that is captured, that the dashboards the tools present are understandable enough by all team members, and that it is integrated into a DevOps process.
- Monitoring is a practice that must be integrated into DevOps culture, taking into account some points of good practice that can improve communication between Dev and Ops and improve product quality for end users.

Evolving project management

- Previously discussed some good DevOps practices to apply to projects, but all this can only be implemented and realized with a change in the way that projects are managed and teams are organized.
- Here are some good practices that can facilitate the implementation of DevOps culture in project management within companies.
- First of all, it should be remembered that DevOps culture only makes sense with the implementation of development and delivery practices that will allow applications to be delivered in short deployment cycles.
- Therefore, in order to be applicable, projects must also be managed with short cycles.

Evolving project management

- To achieve this, one of the most suitable project management methods to apply DevOps culture and has proven its worth in recent years is the agile method, which uses sprints (short cycles of 2 to 3 weeks) with incremental, iterative deployments and strong collaboration between developers.
- DevOps culture just extends the agile methodology by promoting collaboration between several domains (Dev/Ops/Security/Testers).
- In addition, for a better application of DevOps implementations, it is important to change your organization by no longer having teams organized by areas of expertise, such as having a team of developers, another team of Ops, and a team of testers.

Evolving project management

- The problem with this organizational model is that the teams are compartmentalized, resulting in a lack of communication.
- This means that different teams have different objectives, which slams the brakes on applying good practices of the DevOps culture.
- One of the models that allows for better communication is feature team organization with multidisciplinary project teams that are composed of people from all fields.
- In a team, we have developers, operational staff, and testers, and all these people work with the same objective.

Thank You