Introduction to DevOps

UNIT-5 QB

1. What is DevSecOps? How is it important in improving the DevOps practice?

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.

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?

2. What is Chef InSpec? Give an overview of Chef InSpec.

Chef InSpec is an open-source testing framework for infrastructure with a human- and machine-readable language for specifying compliance, security and policy requirements. **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. One must have prior knowledge of the language.

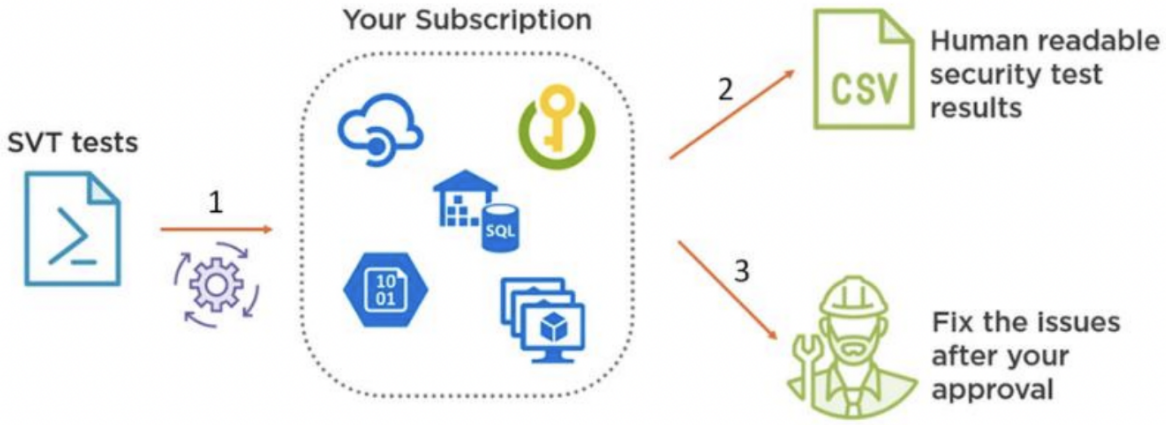
With InSpec, we can test the compliance of remote machines and data and, since the latest version, it is also possible to test cloud infrastructure such as Azure, AWS, and GCP.

3. What is a Secure DevOps kit (AzSK)? Discuss.

- It is a tool that allows users 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.

4. Explain the working of AzSK with a neat diagram.



1. Scan the Azure Resources
2. Get test results
3. Apply the needed fixes

5. What is the need for protecting the sensitive data between different components? List and explain the different tools used for securing sensitive data.

- 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.
- Tools :
    - Ansible Vault
    - Vault by Hashicorp
    - KeyPass
    - LastPass

6. List and explain the 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.**

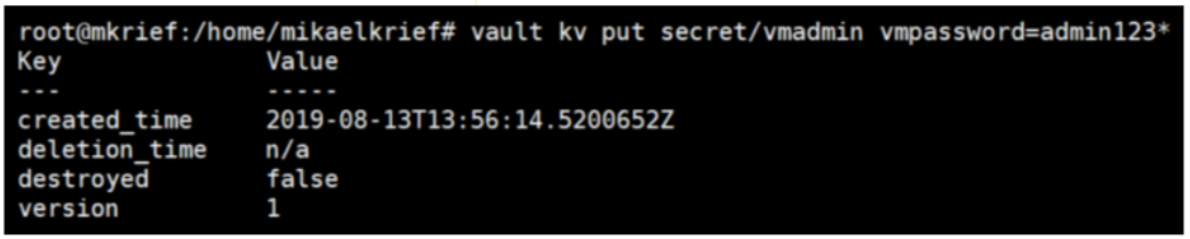
- It also has a **web interface** that allows the management of secrets.
- It integrates with a multitude of authentication systems.
- All secrets are stored in a single **centralised** tool.
- It allows you to be independent of your architecture by being **accessible** with all major cloud providers, Kubernetes, or even on internal data centres (on premises).

7. Discuss the different methods of installing the 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.

8. Illustrate how to protect sensitive data by writing secrets in the 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****



- 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.
- 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.

9. With necessary commands discuss how to read the 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.
- 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*
```

- 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

```
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#
```

the desired version number, as in this example:

- **vault kv get -version=1 secret/vmadmin**

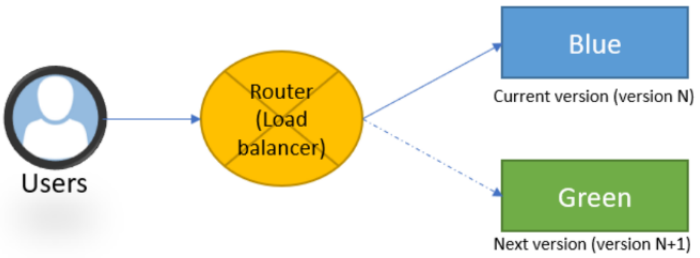
10. With the help of a necessary code explain how to reduce the deployment downtime with Terraform.

- To solve this problem of downtime, we can add the Terraform create-before-destroy option:
- 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.

```
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}
}
```

11. Illustrate the blue-green deployment concepts and patterns with the help of a neat diagram. How it helps in improving the production environment.

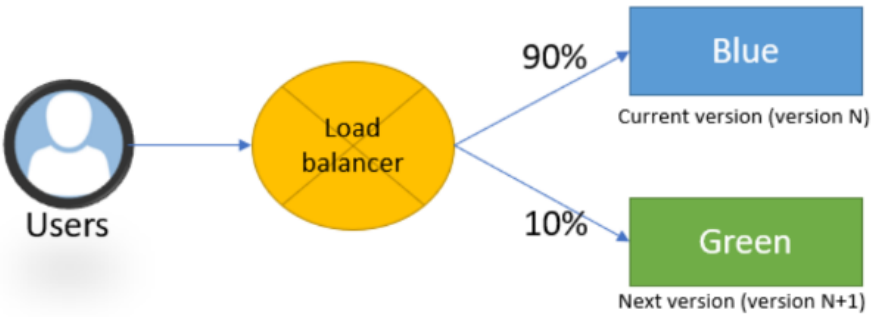
- Blue-green deployment is a practice that allows us to deploy a new version of an application in production without impacting the current version of the application.
- In this approach, the production architecture must be composed of two identical environments; one environment is known as the blue environment 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.
- The following diagram shows a simplified schematic of a blue-green architecture:



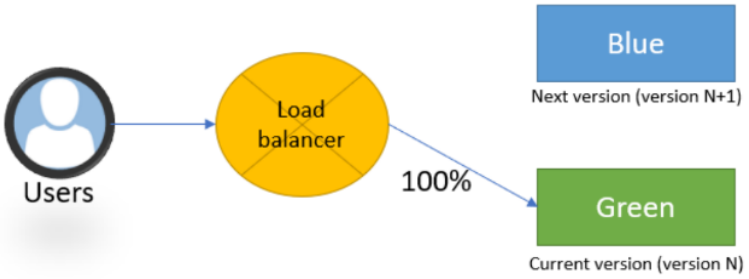
- As we can see, there are two identical environments: the environment called blue, which is the current version of the application, and the environment called green, which is the new version or the next version of the application.
- We can also see a router, which redirects users' requests either to the blue environment or the green environment.
- 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.

12. Discuss the canary release pattern with the neat diagram.

- 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.
- The following is a sample diagram of 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:



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

13. What is the dark launch pattern? How is it different from the canary release pattern? Discuss.

- 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.

14. What are feature flags? Discuss the different technical solutions used in implementing feature flags in an application.

- 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
}
```

- 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.
- 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.
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.

15. Writing all your configuration in code is one of the best practices in DevOps. Justify the statement.

Writing all configuration in code has several benefits in the context of DevOps:

- **Version control**: By writing configuration in code, you can store it in a version control system like Git. This allows you to track changes to your configuration over time and roll back to previous versions if needed.
- **Collaboration**: Storing configuration in version control also makes it easier for multiple team members to collaborate on configuration changes. They can submit pull requests to make changes, which can be reviewed and approved by other team members before being deployed.
- **Reusability**: Code-based configuration can be easily reused across different environments, such as development, staging, and production. This can help to reduce errors and ensure consistency across environments.
- **Testing**: Code-based configuration can be tested using automated testing tools, which can help to catch errors before they are deployed.
- **Automation**: Code-based configuration can be automatically deployed using tools like Ansible, chef, and puppet, which helps to reduce the risk of errors and makes it easier to manage configuration across multiple servers.

16. Outline the good practices to consider when designing software architecture and designing infrastructure.

- Good communication between **cloud architects** and developers.
- Collaboration of **security teams** with developers. Security teams must provide their specifications, which will be implemented by developers and cloud architects.
- Writing code in separate places for separate parts of the project. This achieves scalability, maintainability and **modularity**.
- Use of **feature flags** as they allow the application to be deployed in the production stage, enabling/disabling its features dynamically without having to redeploy it.
- Unit and Integration **testing** on completion of project development, because they allow feedback on the state of the application to be shared very quickly in its deployment cycle.

17. Outline the best practices for 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.
- 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 optimise 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.
- 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, **centralised secret management tools** such as Vaultor use Azure Key Vault.

18. Describe the following

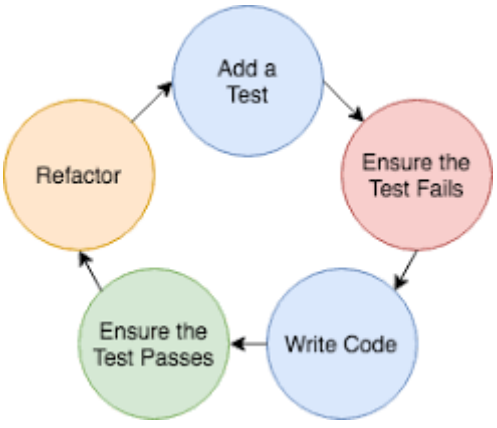
i. Test-Driven Development

Test-driven development (TDD) is a software development methodology in which tests are written for a piece of code before the code is actually written. The tests are designed to ensure that the code meets the desired specifications.

The process of TDD typically involves the following steps:

1. **Write a test**: The first step is to write a test for the desired functionality. The test should define the input and expected output for the code.
2. **Run the test**: The next step is to run the test. Since the code has not been written yet, the test should fail.
3. **Write the code**: The third step is to write the code to make the test pass. The code should be as simple as possible, and should only include the minimum amount of code needed to make the test pass.
4. **Run the tests again**: After the code has been written, the tests are run again. If the tests pass, then the code is considered to be correct.
5. **Refactor the code**: If the tests pass, the code can be refactored(restructured) to make it more efficient or maintainable. However, the tests should still be run to ensure that the refactored code still meets the desired specifications.

In the context of DevOps, TDD can be used to ensure that code changes are safe and do not introduce new bugs into the system. By writing tests before writing code, developers can be confident that their changes will not break the system.



ii. Behaviour-Driven Development

- A typical project using behaviour-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.
- The expectations for the product's behaviour are then set as goals for the developers, and once all of the behaviour tests are passed the product has met its requirements and it is ready for delivery to the customer.
- Behaviour-driven development revolves around conducting behaviour-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 behaviour.
    - Translating each scenario into domain specific language (DSL) to ensure accurate communication.
    - Gathering all behaviours into one set of documentation so it is accessible for all developers, testers and stakeholders.

19. Explain the steps taken to improve security in DevOps processes. ???

- It is necessary to raise awareness among developers of aspects of application code security, but also of the protection of CI/CD pipeline configuration.
- 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.
- It is 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 analysing the security of the application, **work with simple security rules that are recognized by communities,** such as the top 10 OWASP rules.

20. Discuss the 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.
- 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.
- 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.