**PROG8850- Database Automation**

**Assignment 4 (Total: 20 points)**

**Question 1: Analysis and Integration of Database Automation Tools (8 Points)**

* 1. Select any two database automation tools from the following list: Jenkins, GitHub Actions, Azure DevOps, GitLab CI/CD, Liquibase, Flyway, Ansible. For each tool:

- Provide a brief overview and key features.

Jenkins

|  |
| --- |
| While Jenkins doesn't directly perform biological or chemical assays, its fundamental purpose—automation—applies powerfully to the realm of software testing, which we can conceptualize as "assays" of code quality and functionality. In this context, Jenkins orchestrates a comprehensive suite of automated tests, acting as a meticulous lab technician ensuring the integrity of the software product. Instead of pipettes and reagents, Jenkins wields scripts and test frameworks, meticulously examining each component and interaction.  Imagine a sophisticated laboratory where each test is a precisely defined assay. Jenkins, in this scenario, is the central automation system, managing the flow of samples (code changes) through a series of rigorous procedures. Unit tests, the equivalent of highly specific biochemical assays, scrutinize individual code modules, ensuring their isolated functionality aligns with expected outcomes. Integration tests, akin to assays examining the interplay of biological pathways, probe the interactions between different software components, verifying their seamless collaboration. Functional tests, mirroring assays that validate a system's overall performance, confirm that the software meets its specified requirements. End-to-end tests, the comprehensive clinical trials of software development, simulate real-world user interactions, validating the entire application's flow.  Jenkins' ability to manage and execute these diverse "assay types" is facilitated by its extensive plugin ecosystem. This ecosystem, much like a well-stocked laboratory supply cabinet, provides the necessary tools and integrations to handle a wide range of testing methodologies. Performance tests, akin to stress tests on a biological system, assess the software's resilience under various loads. Security tests, the equivalent of rigorous quality control measures, identify potential vulnerabilities, ensuring the software's robustness against threats.  The automation provided by Jenkins is not merely about speed; it's about consistency and reliability. Just as a laboratory strives for reproducible results, Jenkins ensures that each test is executed identically every time, eliminating human error and providing a consistent baseline for evaluating software quality. This repeatability is vital for continuous integration and continuous delivery, enabling teams to rapidly iterate and deploy software with confidence.  In essence, Jenkins transforms software testing into a highly automated and reliable process, akin to a well-oiled laboratory. By orchestrating a diverse array of "assay types," Jenkins ensures that software products are rigorously tested, consistently reliable, and ready for deployment. Thus, while the context differs from traditional scientific assays, the underlying principles of automation, precision, and reliability remain strikingly similar.  **Key feature**   * **Continuous Integration and Continuous Delivery:**   Automates the build, test, and deployment processes.   * **Plugin Ecosystem**   Extensive library of plugins for integration with various tools and technologies.   * **Easy Installation and Configuration**   User-friendly web interface for setup and management.   * **Distributed Builds**   Supports distributed builds across multiple machines for faster execution.   * **Pipeline as Code**   Jenkins Pipelines allow defining CI/CD workflows as code, enabling version control and reproducibility.   * **Cross-Platform Support**   Runs on various operating systems, including Windows, Linux, and macOS. |

Ansible

|  |
| --- |
| **Overview**   * **Automation Engine:**   + Ansible automates repetitive tasks across various IT environments, including servers, networks, and cloud infrastructure.   + It streamlines deployments, configurations, and management processes. * **Infrastructure as Code (IaC):**   + Ansible allows you to define your infrastructure as code, enabling version control, reproducibility, and consistency. * **Agentless Architecture:**   + It operates without requiring agents to be installed on target systems, simplifying deployment and management.   + It communicates via SSH or other standard protocols. * **Simplicity and Readability:**   + Ansible uses YAML, a human-readable language, making playbooks easy to understand and maintain.   **Key Features:**   * **Playbooks:**   + YAML files that define automation tasks and configurations.   + They describe the desired state of systems and the steps to achieve it. * **Inventory:**   + A list of target systems that Ansible manages.   + It can be static or dynamic, allowing for integration with cloud providers and other dynamic environments. * **Modules:**   + Pre-built tasks that perform specific actions, such as installing packages, configuring services, or managing files.   + Ansible has a large library of modules, and you can create custom modules. * **Roles:**   + A way to organize and reuse Ansible playbooks.   + They encapsulate related tasks and configurations, promoting modularity and reusability. * **Idempotency:**   + Ansible ensures that tasks are executed only when necessary, preventing unintended changes.   + Running a playbook, multiple times produces the same result. * **Community and Ecosystem:**   + Ansible has a large and active community, providing support, modules, and roles.   + Ansible Automation Platform is a red hat supported enterprise grade platform, that greatly increases the capabilities of the opensource ansible project. * **Network Automation:**   + Ansible can automate network devices, allowing for consistent and efficient network configurations. * **Cloud Automation:**   + Ansible can automate the provisioning and management of cloud resources on platforms like AWS, Azure, and Google Cloud. |

- Create a comparison table evaluating the two tools based on- Ease of Use, Integration with CI/CD Pipelines, and Supported Databases

|  |  |  |
| --- | --- | --- |
| Feature | Jenkins | Ansible |
| Ease of Use | \* Steeper learning curve, especially for complex pipeline configurations.  \* Web-based GUI can be complex.  \* Relies heavily on plugins, which adds complexity. | \* Relatively easy to learn due to YAML-based playbooks.  \* Agentless architecture simplifies setup.  \* Human-readable syntax. |
| Integration with CI/CD Pipelines | \* Core strength. Designed for CI/CD automation.  \* Extensive plugin ecosystem for integrating with various CI/CD tools.  \* Robust pipeline functionality. | \* Can be integrated into CI/CD pipelines, but not its primary focus.  \* Often used for deployment stages of a CI/CD pipeline, after Jenkins build and testing.  \* Less focus on the "CI" side of CI/CD. |
| Supported Databases | \* Indirectly through plugins. Can interact with databases through scripting and relevant plugins for tasks like database migrations during deployments.   \* Jenkins itself does not manage databases. | \* Can manage database configurations and deployments through modules. Supports various databases (MySQL, PostgreSQL, etc.).  \* Focus's on the configuration of databases. |
| Primary use case | Continuous integration and continuous delivery pipelines. | Infrastructure as code, and configuration management. |

* 1. Integration Strategy: Propose a strategy to integrate the two selected tools into a CI/CD pipeline for a software project.

**Pipeline Stages and Tool Roles:**

1. **Version Control (Git, etc.):**
   * Developers commit code changes to a version control system.
2. **Continuous Integration (Jenkins):**
   * **Trigger:** Jenkins is triggered by code commits.
   * **Build:** Jenkins builds the application (compiles code, packages dependencies).
   * **Unit Tests:** Jenkins runs unit tests to verify code functionality.
   * **Static Analysis:** Jenkins performs static code analysis to identify potential issues.
   * **Artifact Creation:** Jenkins creates deployable artifacts (e.g., Docker images, JAR files).
   * **Artifact Storage:** Jenkins stores the artifacts in an artifact repository (e.g., Nexus, Artifactory).
3. **Continuous Testing (Jenkins):**
   * **Integration Tests:** Jenkins deploys the application to a test environment and runs integration tests.
   * **Functional/E2E Tests:** Jenkins executes functional and end-to-end tests to validate application behavior.
   * **Performance Tests:** Jenkins runs performance tests to evaluate application performance under load.
   * **Security Tests:** Jenkins conducts security scans to identify vulnerabilities.
   * **Test Reporting:** Jenkins generates test reports and provides feedback to developers.
4. **Continuous Deployment (Ansible):**
   * **Trigger:** Upon successful testing in Jenkins, Jenkins triggers an Ansible playbook.
   * **Infrastructure Provisioning (Ansible):** If necessary, Ansible provisions the required infrastructure (e.g., servers, cloud resources).
   * **Configuration Management (Ansible):** Ansible configures the target environment (e.g., installs dependencies, configures services).
   * **Application Deployment (Ansible):** Ansible deploys the application artifacts from the artifact repository.
   * **Database Migrations (Ansible):** Ansible executes database migrations as part of the deployment process.
   * **Service Restart/Configuration (Ansible):** Ansible restarts services or applies final configuration changes.
   * **Verification (Ansible):** Ansible performs post-deployment verification checks.
5. **Continuous Monitoring (External Tools):**
   * External monitoring tools (e.g., Prometheus, Grafana) monitor the application and infrastructure.
   * Alerts are sent to the development and operations teams.

**Integration Methods:**

* **Jenkins Plugins:**
  + Use the Ansible plugin for Jenkins to trigger Ansible playbooks from Jenkins pipelines.
  + This allows for seamless integration and parameter passing between Jenkins and Ansible.
* **Jenkins ssh or command steps:**
  + Jenkins can also call ansible-playbook commands directly, via ssh, or through local commands.
* **Artifact Repository:**
  + Use an artifact repository to store deployable artifacts, enabling both Jenkins and Ansible to access them.
* **Environment Variables:**
  + Pass environment variables between Jenkins and Ansible to share configuration and deployment parameters.
* **API Interactions:**
  + For more advanced integrations, use API interactions to exchange data and control between Jenkins and Ansible.

**Benefits:**

* **Separation of Concerns:** Jenkins handles CI and testing, while Ansible handles deployment and configuration.
* **Increased Efficiency:** Automation of the entire CI/CD pipeline reduces manual effort and accelerates releases.
* **Improved Reliability:** Consistent deployments and configurations ensure application stability.
* **Enhanced Collaboration:** Clear separation of roles and responsibilities facilitates collaboration between development and operations teams.
* **Infrastructure as Code:** Ansible enables infrastructure as code, promoting consistency and reproducibility.

**Example Pipeline Snippet (Jenkinsfile):**

|  |
| --- |
| pipeline {  agent any  stages {  stage('Build') {  steps {  // Build steps...  }  }  stage('Test') {  steps {  // Test steps...  }  }  stage('Deploy') {  steps {  ansiblePlaybook(playbook: 'deploy.yml', inventory: 'production')  }  }  }  } |

This strategy creates a well-defined and automated CI/CD pipeline, maximizing the benefits of both Jenkins and Ansible.

**Question 2: Hands-on Exercise Using Ansible (12 Points)**

**Task: Implement an up.yaml and down.yaml ansible-playbook that deploys a Mysql database, includes a schema update step, validates the update and creates a migration when down.yaml is run for the next time up.yaml is run.**

**Steps**

1) ***Initial Setup-*** Use the provided Git repository to create and maintain a database of subscribers and their email addresses:

* Script 1: **‘up.yaml’** – Starts mysql, initializes the database so that it can be used by flyway and run’s flyway to apply the migrations. up.yaml needs to be idempotent
* Script.2 dbtests.py - tests the schema to make sure that it contains the required structure for the current commit to the repository. You will need to continue adding to this every time you make a commit.
* Script 3: **‘down.yaml’** – makes a migration to seed the database with any data that was added since the last commit and stops the database. You will run down.yaml when you are done with the lab, and then commit the results for the next time you create the database.

1. **Change-** you will make a change to the database to add the subscription date to your subscriber table. The subscription date needs to be automatically populated when a new row is inserted:
   * Step 1: Make a migration and test manually by adding a new subscriber.
   * Step 2: Update dbtests.py with test(s) for your change.
   * Step 3: run down.yaml to shutdown and save state.
   * Step 4: run a commit and push so that your data is preserved
2. **Submission** 
   * Submit the zip of your repository, also including a .pdf of your answer to question 1**.**

**Points Breakdown of Question 2:**

* Correct setup and deployment of the up and down.yaml: 4 Points
* Execution of the flyway schema update: 4 Points • Validation of the schema update in dbtests.py: 4 Points **Notes for Submission:**
* Submit your assignment as a ZIP file including:
  + A Word/PDF document with your answers to Question 1 (overview, comparison, and strategy).
  + zip of your repository including pdf.
* Ensure your submission is organized and well-documented:
* Cite any references used
* Code should be clean and well-commented:

- Include comments in the ‘.yml’ file(s) to explain key steps.