Integrating quality and security gates

Resist the initial temptation to rush on the keyboard! This is the time to pause and think as a team about your end-to-end deployment process that you would implement in this company.

Reminders:

- Use an InPrivate/Incognito window in your browser to avoid any confusion with any other credentials that you may use to access Azure resources.

At the completion of Challenge 5, you implemented workflow(s) to perform unit testing, continuous integration, and Blue/Green deployments. You performed basic gating to verify that the staging environment was online before performing the Blue/Green swap.However, there are alway opportunities to improve automated quality and security gates.

Depending upon business need, it is common to perform one or more of the following in your deployment workflows:

- **Dependency Scanning** - Open source components are used in the majority of modern applications. This greatly accelerates development but can

also introduce vulnerability or license issues. Dependency scanning can check for issues as part of your code commit or during CI/CD workflows.

- **Code Coverage** - Unit Tests can provide a safety net and indicator of code quality. However, unit tests require ongoing maintenance and can provide a false sense of security. Code Coverage helps you understand how much of your code is being tested and monitor whether testing declines as code changes.
- **Static Code Analysis** - Computers are useful in identifying patterns, including patterns in code that may be vulnerable, difficult to maintain, perform poorly or are otherwise buggy. Static Code Analysis can identify "code smells" and improve the value of your code reviews.
- **Variant Analysis** - Like Static Code Analysis, Variant Analysis can identify "code smells" and improve the value of your code reviews. However, Variant Analysis can also reduce false positives and otherwise provide more meaningful insights to your code. Variant Analysis has proven extremely useful in security analysis scenarios.
- **Integration Testing** - After deployment to a pre-production environment, automated tests can run against an integrated system, verifying not only "units" of code but also the completely integrated system.
- **Load Testing** - Code changes can cause unexpected performance issues. Load and Performance testing enables you to measure the scaling or performance impact of code changes.
- **Manual Approvals** - Although manual intervention is against some of the core tenants of automation, it is still required in some scenarios. In this case, moving from a pre-production environment to production requires the approval of a specific individual or team.

Challenge

In this challenge you will plan and improve your workflow to support ***one or more*** quality or security gates. You must complete implementation of one of these enhancements and should be able to share the design of a second enhancement.

- *Dependency Scanning* is frequently performed as part of your commit workflow or during PR or CI builds. If you're using GitHub, implement security alerts and automated security updates. For other repositories, integrate Whitesource Bolt or another vulnerability into your PR or CI workflow.
- *Code Coverage* is generally performed during unit testing to measure how much of your code is actually tested. Integrate code coverage reporting into your PR or CI workflow. You may also implement a gate such that the workflow fails if the amount of code coverage falls as code is changed.
- *Static Analysis* is generally performed as part of PR or CI builds and reporting is used to support peer reviews. Different computer languages may leverage different tools for static analysis.
- *Variant Analysis* is more intensive than static analysis an is frequently decoupled from the pipeline. Semmle (https://semmle.com) can be configured to support variant analysis on your public repositories.
- *Integration Tests* are generally run against a pre-production environment to verify your components integrate well across your code base. The technology will be different than unit testing (e.g., Selenium et al) and should be implemented after deployment.
- *Load Tests* are similar to integration tests in that they are run after deployment. Some implementations force hard gates when performance degrades, and others simply provide feedback for manual approvals.

- *Manual Approval* is very common in enterprise environments, relying on a specific individual or team to review an approve a pre-production deployment before pushing to production.

> **Note**: Different source repositories and orchestration tools will have different mechanisms to implement these capabilities. Depending upon your tools, some of these may be significantly easier or more difficult to achieve.

Success Criteria

- Demonstrate one or more of the above enhancements. This will frequently require pushing changes through your workflow and showing and explaining the steps. It may also require demonstrating integration with external tools.
- For a different enhancement, explain a design, the specific tools that you would leverage, how they would fit into your process, and where you would integrate them.

References

- Dependency Scanning
  - GitHub Security Alerts (https://help.github.com/github/managing-security-vulnerabilities/about-security-alerts-for-vulnerable-dependencies)
  - Automated Security Updates with GitHub (https://help.github.com/github/managing-security-vulnerabilities/configuring-automated-security-updates)
  - Whitesource Bolt with Azure Pipelines (https://bolt.whitesourcesoftware.com/azure/faq/)
  - OWASP Dependency Checker in the Azure DevOps marketplace (https://marketplace.visualstudio.com/items?itemName=dependency-check.dependencycheck)
- Code Coverage
  - CoverAlls (https://coveralls.io/)
  - CodeCov (https://codecov.io/)
  - Jacoco (https://www.jacoco.org/jacoco/)
  - CoverAlls for GitHub Actions (https://github.com/marketplace/actions/coveralls-github-action)
  - Azure Pipelines ecosystem (per-language code coverage docs) (https://docs.microsoft.com/azure/devops/pipelines/ecosystems/?view=azure-devops)
  - GitHub Coverage Reporter for Jenkins (https://github.com/jenkinsci/github-coverage-reporter-plugin)

- Static Code Analysis
  - <a href="https://github.com/marketplace?type=actions&query=lint" target="_blank">Linters in the GitHub Actions marketplace
  - Static Code Analysis tools (Wikipedia) (https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)
  - SonarCloud for GitHub Actions (https://github.com/marketplace/actions/sonarcloud-scan)
  - Integrate Visual Studio Team Services with SonarCloud (https://docs.microsoft.com/labs/devops/sonarcloudlab/index)
  - Using SonarCloud in an Azure DevOps pipeline (https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Extension+for+VSTS-TFS)
- Variant Analysis (Security)
  - Variant Analysis for Security (https://semmle.com/variant-analysis)
  - Continuous security analysis (with GitHub and SEMMLE) (https://lgtm.com/)
- Integration Testing
  - Integration testing (http://softwaretestingfundamentals.com/integration-testing/)
  - Integration tests vs Unit tests (https://www.guru99.com/unit-test-vs-integration-test.html)
  - Integration test with .Net Core (https://docs.microsoft.com/aspnet/core/test/integration-tests)
  - Integration testing in GO (https://blog.codeship.com/testing-in-go/)
  - Set up a go workspace (https://docs.microsoft.com/azure/devops/pipelines/languages/go#set-up-a-go-workspace)
  - Integration testing with Node.JS (https://www.codementor.io/olatundegaruba/integration-testing-supertest-mocha-chai-6zbh6sefz)
- Load Testing
  - Blazemeter with Azure Pipelines (https://guide.blazemeter.com/hc/en-us/articles/360004191957-Testing-via-Azure-DevOps-Pipeline-Testing-via-Azure-DevOps-Pipeline)
- Manual Approvals
  - Setting up Webhooks for GitHub Actions (http://www.btellez.com/posts/triggering-github-actions-with-webhooks.html)
  - GitHub Deployments (REST API) (https://developer.github.com/v3/repos/deployments/)
  - Release approval gates for Azure Pipelines (https://docs.microsoft.com/azure/devops/pipelines/release/approvals/?view=azure-devops)
  - Manual Intervention task for Azure Pipelines (https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/manual-intervention?view=azure-devops)