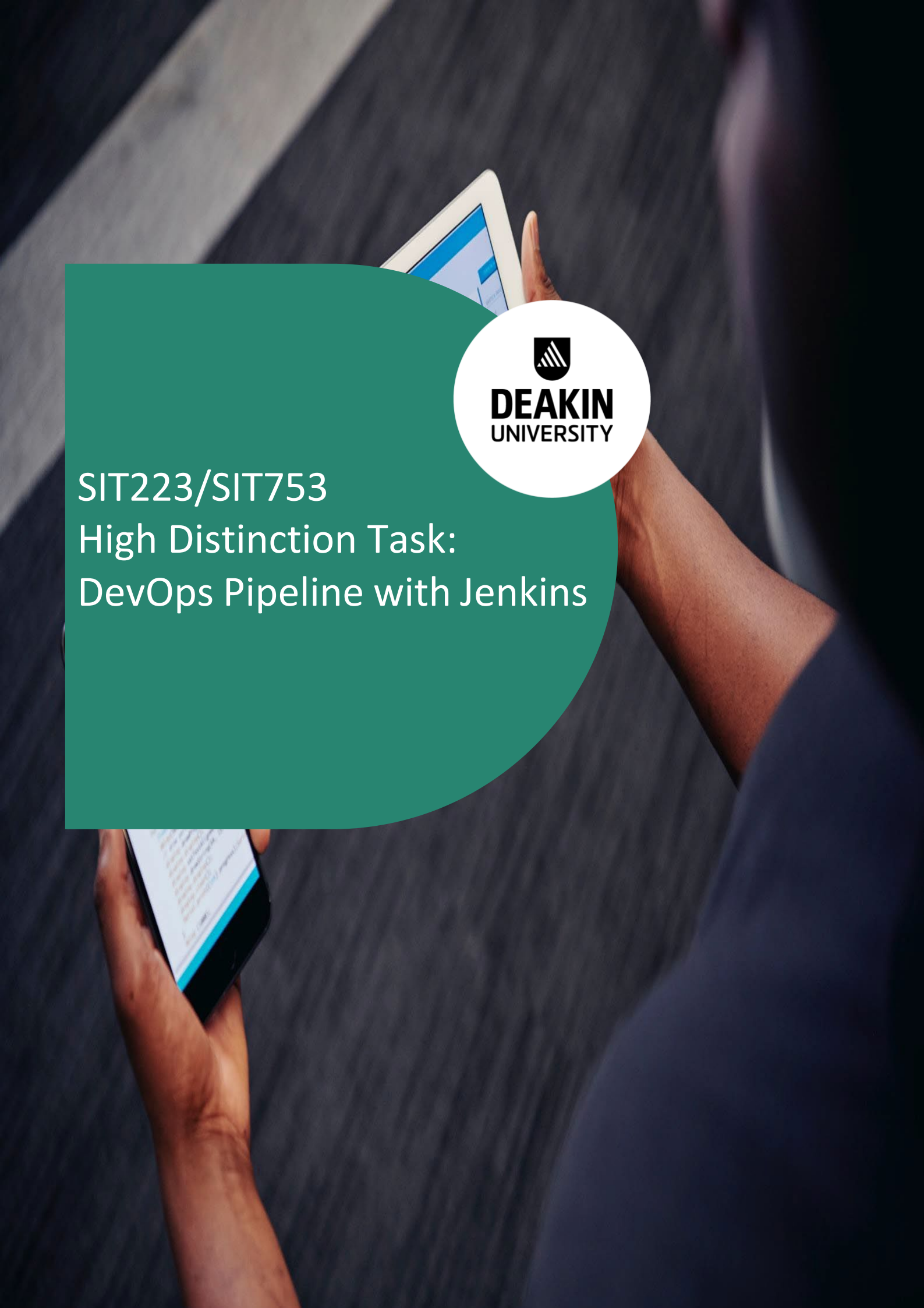

DEAKIN UNIVERSITY

SIT223/SIT753
High Distinction Task:
DevOps Pipeline with Jenkins

Overview

In this task, you will create a DevOps pipeline for your own project or code using Jenkins to ensure that your code is built, tested, and deployed in a reliable and consistent manner. The pipeline will include automated testing, code quality analysis, and deployment. Continuous delivery and deployment tools will be used to ensure that changes are automatically deployed to test and production environments, while monitoring and alerting tools will be included to ensure that any issues are quickly identified and addressed.

Assessment Instructions

To achieve **a low HD grade**, it is necessary to **successfully implement only four stages from steps 4-10**. Successful implementation of these four required stages will demonstrate the completeness of the pipeline functionality. However, **developing more than four stages can lead to a high HD grade**.

1. Choose a project of your choice. This could be your Capstone project or previous projects, such as a web application or mobile application (Please see Appendix I).

2. Create a Git repository for your project and push your code to the repository.

3. Create a Jenkins pipeline for your project with all stages: Build, Test, Code Quality, Security, Deploy, Release, and Monitoring.

4. In the **Build stage**, configure Jenkins to build your code and create a build artefact. This could be a JAR file, Docker image, or any other artefact that can be used to deploy your application.

5. In the **Test stage**, configure Jenkins to run automated tests on your code. You can choose any testing framework of your choice, such as JUnit, Selenium, or Appium, to test your application in the test environment to ensure that the deployment was successful, and that the application is working correctly.

6. Configure Jenkins to **run code quality analysis** on your code. This focuses on the structure, style, and maintainability of your codebase. You can choose tools such as SonarQube or CodeClimate and configure them to automatically detect issues such as code duplication, code smells, and poor complexity or design patterns.

   Note: *While some tools (e.g., SonarQube) may optionally report basic security-related issues, this stage is mainly concerned with code health,* ***NOT*** *in-depth security scanning.*

7. In the **Security stage**, configure Jenkins to perform automated security analysis on your codebase or dependencies. This ensures that vulnerabilities are detected and addressed early in the CI/CD pipeline. If vulnerabilities are found, you must briefly explain:
    - What the issue is.
    - Its severity.
    - Whether and how you addressed it (e.g., updating a library, excluding false positives).

    Tip:
    - *Think of Code Quality as improving how your code looks and works for developers.*
    - *Think of Security as protecting your app and users from hackers.*

8. In the **Deploy stage**, configure Jenkins to deploy your application to a test environment, such as a staging server or a Docker container. You can use any deployment tool of your choice, such as Docker Compose or AWS Elastic Beanstalk.

9. In the **Release stage**, configure Jenkins to promote the application to a production environment. You can use any release management tool of your choice, such as Octopus Deploy or AWS CodeDeploy.

10. In the **Monitoring and Alerting stage**, configure Jenkins to monitor the application in production for any issues and alert the team if any issues arise. You can choose a monitoring and alerting tool, such as Datadog or New Relic, and configure it to monitor the application in production for any issues and alert the team if any issues arise.

Submission Guidelines

- Create a demo video that showcases your pipeline in action. The video should be no longer than 10 minutes and should demonstrate the following:
    - How to clone the repository and set up the pipeline in Jenkins.
    - How the pipeline works and progresses through each stage.
    - The final deployed application and any additional features you wish to showcase.

- Submit a PDF document that includes the following, using the provided template:
    - A link to the demo video.
    - A link to your GitHub repository containing the Jenkins pipeline script. Please ensure that **BOTH your Marker AND the Unit Chair** have been granted appropriate **access permissions** to view the codebase.
    - Specify how many stages you have implemented.
    - A brief description of your project and the technologies used.
    - A screenshot of your Jenkins pipeline.
    - A brief description of each stage of your pipeline, including the frameworks/tools used.

See Appendix I – on the next page

Appendix I - Some Notes on How to Choose a Suitable Project for Your Jenkins DevOps Pipeline

1. **Choose a Project with Functional Depth**
   - Your project should include multiple features or components (e.g. user authentication, CRUD operations, or API endpoints).
   - It should be more than just static content — ideally, it should involve back-end logic, services, or interactions that can be built, tested, deployed, and monitored.

2. **The Codebase Should Support Automation**
   - Use a tech stack that can be easily integrated into Jenkins (e.g. Node.js, Java, Python, Docker).
   - Ensure it has a build process (e.g. npm run build, mvn package, or docker build).
   - The codebase should support tools such as:
     - SonarQube or CodeClimate for code quality
     - Snyk, Trivy, or Bandit for security scanning
     - Docker, AWS, or Heroku for deployment

3. **Testing Should Be Possible**
   - Include logic or modules that can be tested with unit or integration testing frameworks (e.g. JUnit, Jest, Mocha, or Postman for API testing).
   - Avoid projects that consist only of static HTML/CSS or simple front-end demos lacking testable logic.

4. **Ensure It Can Be Deployed**
   - The project should run as a deployable app or service such as a web app, mobile API back-end, or microservice.
   - If using containers, ensure you have a working Dockerfile or docker-compose.yml.

5 Support Monitoring (For High HD)
   - Your project should expose logs, endpoints, or performance metrics that can be monitored.
   - Choose something that can be run in a container or VM where tools like Prometheus, New Relic, or Datadog can track usage or failures.

Appendix II - Rubric

Criteria	80–85% (Low HD)	85–90% (low HD)	90–95% (High HD)	95–100% (High HD)
Pipeline Completeness	At least **3 core stages** implemented (e.g., Build, Test, Deploy).	**4–5 stages** implemented and function correctly.	**All 7 stages** implemented, minor integration issues.	**All 7 stages** implemented with full automation and smooth transitions between stages.
Project Suitability	Project runs and builds but is basic in scope.	Moderate functionality; limited modularity but testable.	Well-structured, modular project with multiple testable features.	Complex, production-like project suitable for a full pipeline (test, monitor, secure, deploy).
Build Stage	Generates a working artefact (e.g., Docker image, JAR).	Automated build with minor scripting.	Build integrated with versioning, triggers, and clean logs.	Fully automated, tagged builds with version control and artifact storage.
Test Stage	Basic test suite using a single framework.	Multiple units/components tested.	Good coverage; automated test feedback in pipeline.	Advanced test strategy (unit + integration); structured with clear pass/fail gating.
Code Quality Stage	SonarQube or equivalent used with default setup.	Custom rules or thresholds applied.	Configured quality gates with explained metrics.	Advanced config: thresholds, exclusions, trend monitoring, and gated checks.
Security Stage	Tool runs but shows minimal results or explanation.	Issues identified; some understanding of severity.	Scan output interpreted; vulnerabilities categorized and partially addressed.	Proactive security handling: issues fixed, justified, or documented with mitigation.
Deploy Stage	Deployed to local or test server manually.	Semi-automated to staging with Docker or scripts.	Fully automated deployment to reliable test infra.	End-to-end automated deployment using best practices (infra-as-code, rollback support).
Release Stage	Manual promotion to a final/stable version.	Semi-automated release using basic tool or Git tag.	Release process is repeatable and automated.	Tagged, versioned, automated release with environment-specific configs.
Monitoring Stage	Monitoring tool listed or shown without working setup.	Metrics partially monitored (e.g., CPU, health).	Dashboards or alerts triggered and explained.	Fully integrated system with live metrics, meaningful alert rules, and incident simulation.
Demo Video (≤10 min)	Pipeline shown but lacks fluency or misses some stages.	Walkthrough includes most stages with basic commentary.	Covers all pipeline steps clearly, with commentary and output review.	Professional and confident presentation, with deep insight and fluent narration.
Report Quality	Meets basic submission criteria; minimal explanation.	Clear structure with brief technical descriptions.	Detailed, well-written report explaining pipeline decisions and outputs.	Excellent documentation with diagrams, screenshots, and reflective technical insight.