

Efficient Software Development Workflow: Version Control, GitHub, Build Images and Swagger Design for Design for RESTful APIs

LEARNER'S GUIDE

All rights reserved. This document is provided for the explicit use and guidance of parties approved by NTUC LearningHub Pte Ltd as information resource only. Any other use of this document or parts thereof, including reproduction, publication, distribution, transmission, re-transmission or storage in a retrieval system in any form, electronic or otherwise, for purposes other than that expressly stated above without the express permission of NTUC LearningHub Pte Ltd is strictly prohibited. Printed in Singapore.

Our Essence

Transforming People

We empower people to gain work ad life skills, transforming their lives, improving their employability, and inspiring them to grow through lifelong learning.

Vision

To be the leader and trusted lifelong partner in Continuing Education and Training

Mission

To provide learning that makes Every Worker a Better Worker, Every Job a Better Job, Every Company a Better Company



Our Values

People are our priority

- Respect and teamwork
- Listen to customer needs

Passion for our goals

- Passion for lifelong learning
- Transforming people through learning
- Heart and hunger to serve our customers

Performance is our business

- Deliver exceptional / sustainable value
- Unmatched choice for our people and customers



Table of Contents

Introduction to the Learner's Guide	1
Introduction to the Course	5
Course Objective	6
Warm Up Ice Breaker/ Introduction	7

SECTION 1:

INTRODUCTION TO EFFICIENT SOFTWARE DEVELOPMENT WORKFLOW

Overview of efficient software development practices	8
Role of version control, GitHub, image building, and API design	9
Understanding the benefits of streamlined workflows	10

SECTION 2:

MASTERING VERSION CONTROL WITH GIT AND GITHUB

Deep dive into Git: branches, commits, merges, and pull requests	11
Collaborative development using GitHub: forks, pull requests, code reviews	12
Best practices for managing code changes and resolving conflicts	13

SECTION 3:

BUILDING EFFICIENT CONTAINER IMAGES FOR DEPLOYMENT

Introduction to containerization and its benefits	14
Building Docker images: Docker files, layers, and optimization	14
Leveraging Docker Compose for local development environments	15
Best practices for creating efficient and secure container images	16

SECTION 4:**SWAGGER DESIGN FOR RESTFUL APIS**

Understanding the importance of API design and documentation	18
Introduction to Swagger/Open API for designing RESTful APIs	19
Defining API specifications using Swagger Editor	20
Generating client SDKs and server stubs with Swagger Codegen	23

SECTION 5:**EFFICIENT DEVELOPMENT WORKFLOW INTEGRATION**

Integrating version control, image building, and API design	25
Implementing automated CI/CD pipelines for continuous integration	26
Utilizing version-controlled code for building container images	27
Generating API documentation from Swagger specifications	27

SECTION 6:**HANDS-ON WORKSHOP: IMPLEMENTING EFFICIENT SOFTWARE DEVELOPMENT WORKFLOW**

Collaborative exercises: Setting up Git/GitHub, building container images, Swagger design.	28
Applying efficient workflow practices to a sample project	32
Integrating automated CI/CD pipelines for version-controlled code	34

SECTION 7: FUTURE TRENDS IN SOFTWARE DEVELOPMENT WORKFLOW

Exploring emerging trends in software development practices	36
Recognizing the impact of DevOps, automation, and cloud-native development	38
Reflecting on the evolving landscape of software development workflows	39

INTRODUCTION TO THE COURSE

Overview

This Learner's Guide aimed to present the content and activities aligned to the Workforce Skills Qualification (WSQ) Offer Efficient Software Development Workflow: Version Control, GitHub, Build Images and Swagger Design for Design for RESTful APIs

Target Audience

This 2-day, 16-hour course on "Efficient Software Development Workflow: Version Control, GitHub, Build Images and Swagger Design for Design for RESTful APIs" is a comprehensive course designed for experienced developers and DevOps engineers. Participants will gain mastery over version control, source code management, container image building, and API design to optimize their software development practices.

This advanced course empowers participants to streamline their software development workflow using version control, GitHub, image building, and API design with Swagger. Learn to create efficient and collaborative development processes.

please advise your trainers.

Assumed Skills and Knowledge

Assumed skills and knowledge means what we expect you to know and be able to do already before you start the course. We assume you have:

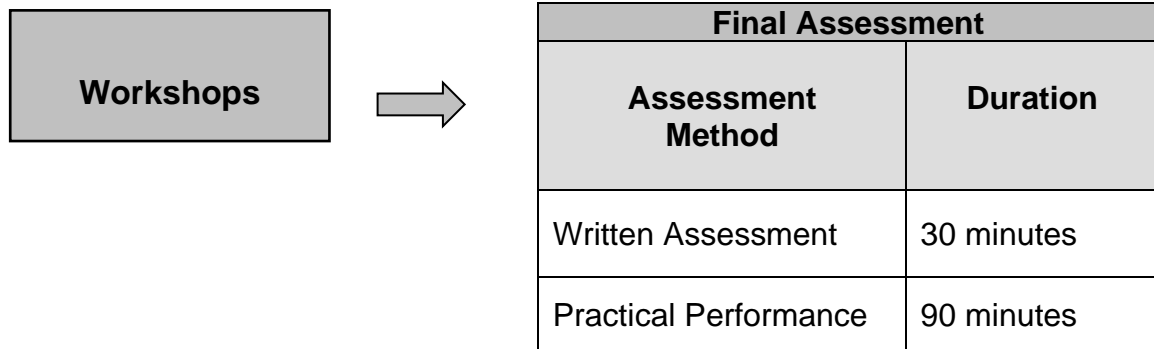
- Proficiency in at least one programming language.
- Familiarity with basic Git commands and RESTful API concepts.

Course Objectives

At the end of this course, learners will be able to:

- Master version control with Git and GitHub for collaborative development.
- Learn to build container images for efficient deployment and scalability.
- Understand the principles of Swagger for designing and documenting RESTful APIs.
- Streamline software development workflows for improved efficiency.

Course Components



Course Duration

Classroom Training	Final Assessment	Total
14 hours	2 hours	16 hours

Warm-Up and Introduction

“Two Truths and a Bean” Icebreaker Activity

- Adapt the classic icebreaker game "Two Truths and a Lie."
- Participants take turns sharing two true statements about themselves related to Spring (e.g., "I've used Spring Boot for a web application" and "I've attended a Spring conference") and one false statement.
- The rest of the group guesses which statement is false

Share the following with your group and then the class:

1. Your name
2. Company you are from
3. What do you do at your workplace?

Notes

Module 1: Introduction to Efficient Software Development Workflow

- Efficient software development workflow is essential for delivering high-quality software in a timely manner.
- It involves a systematic and organized process that optimizes collaboration, code management, and deployment.
- In this overview, we'll discuss key practices and tools that contribute to an efficient software development workflow.

Overview of Efficient Software Development Practices

Agile Methodology:

Agile methodologies, such as Scrum or Kanban, promote iterative development, collaboration, and flexibility in responding to changing requirements.

Continuous Integration (CI) and Continuous Deployment (CD):

CI/CD practices involve regularly integrating code changes into a shared repository, automating testing, and deploying code to production environments. This helps catch and fix issues early in the development cycle.

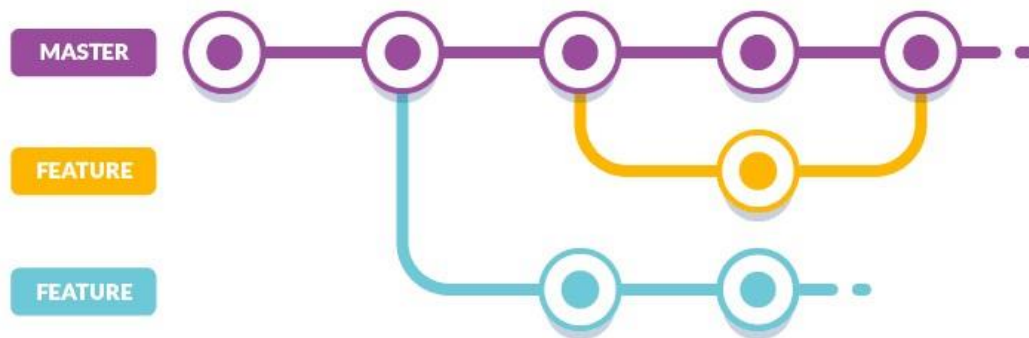
Automated Testing:

Implementing automated testing, including unit tests, integration tests, and end-to-end tests, ensures the reliability and functionality of the software throughout development.

Code Reviews:

Regular code reviews facilitate knowledge sharing, identify potential issues, and maintain code quality. Peer reviews help ensure that code adheres to coding standards and best practices.

Role of version control, GitHub, image building, and API design



Version Control:

Version control systems (e.g., Git) track changes to source code, enabling collaboration among developers, easy rollback to previous versions, and the maintenance of a clean and organized codebase.

GitHub:

GitHub is a web-based platform built on top of Git, providing features like pull requests, issue tracking, and collaboration tools. It enhances team collaboration and project management.

Image Building:

Containerization tools like Docker facilitate image building, allowing developers to package applications and their dependencies into lightweight, portable containers. This ensures consistent environments across different stages of development and deployment.

API Design:

Well-designed APIs (Application Programming Interfaces) are crucial for effective communication between software components. A clear and consistent API design enhances interoperability and ease of integration.

Understanding the Benefits of Streamlined Development Workflows

Faster Time-to-Market:

Efficient workflows reduce development time, enabling faster delivery of software products to end-users.

Improved Collaboration:

Streamlined processes and collaborative tools foster better communication and teamwork among developers, leading to a more cohesive and productive development environment.

Enhanced Code Quality:

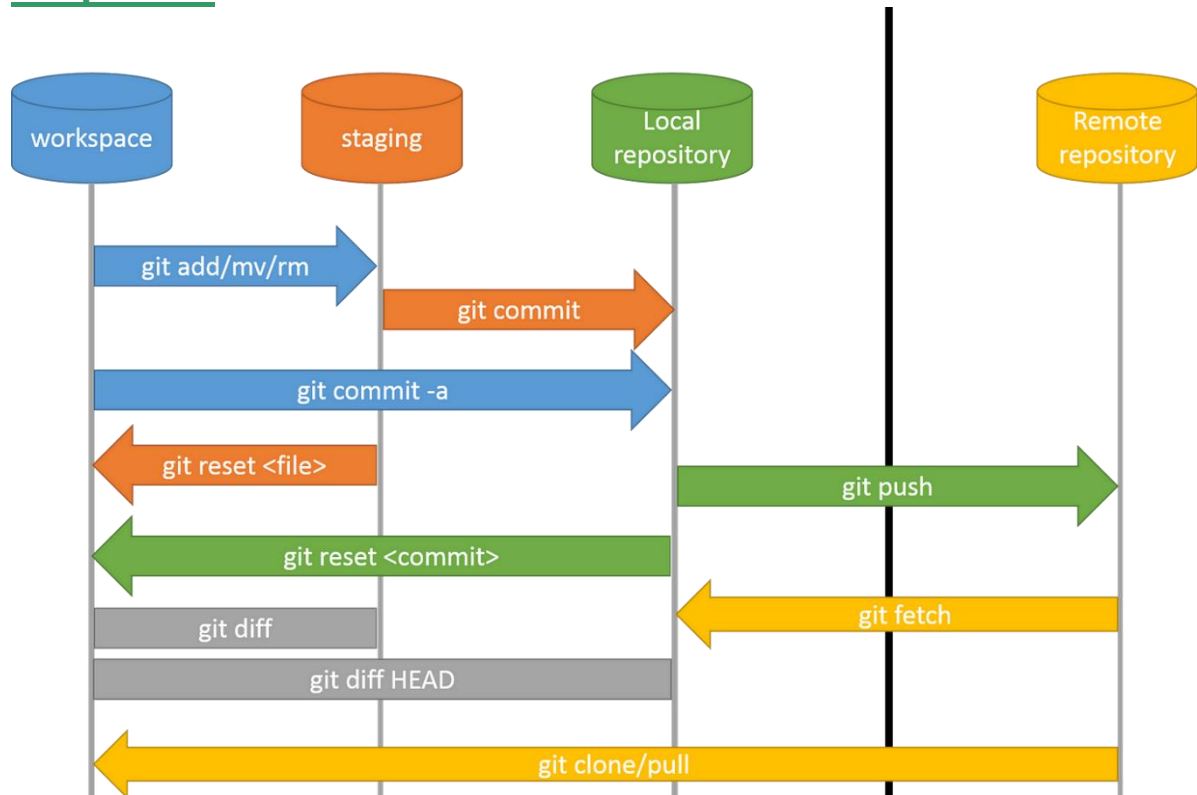
Continuous integration, automated testing, and code reviews contribute to improved code quality, reducing the likelihood of bugs and errors in production.

Scalability and Maintenance:

A well-organized codebase and deployment process make it easier to scale applications and maintain them over time. This is particularly important as software evolves, and new features are added.

Module 2: Mastering Version Control with Git and GitHub

Deep Dive Into Git: Branches, Commits, Merges, and Pull Requests.



Branches:

Git allows developers to create branches to work on features or fixes independently. Branching enables parallel development without affecting the main codebase.

Common operations include creating, switching, and merging branches.

Commits:

Commits represent individual changes to the codebase. Each commit has a unique identifier and a commit message describing the changes made.

Git provides a snapshot of the code at a specific point in time.

Merges:

Merging is the process of combining changes from one branch into another. Git automatically handles simple merges, but conflicts may arise in complex scenarios. Strategies like fast-forward, recursive, and three-way merges are commonly used.

Pull Requests:

Pull requests (PRs) in Git are proposals to merge changes from one branch into another. They are commonly used in collaborative development environments. PRs include a summary of changes, discussions, and automated checks before merging.

Collaborative Development on GitHub: Forks, Pull Requests, and Code Reviews

Forks:

Forking a repository creates a copy of it under your GitHub account. Forks are often used when contributing to open-source projects.

Changes made in a fork can be submitted back to the original repository through pull requests.

Pull Requests:

Pull requests allow contributors to propose changes to the main project. They provide a space for discussion, code reviews, and automated testing. Maintainers review the changes and decide whether to merge them into the main branch.

Code Reviews:

Code reviews involve systematically examining code changes before merging. Reviewers provide feedback on style, logic, and potential issues. GitHub's interface supports inline comments, making it easy to discuss specific lines of code.

Best Practices for Managing Code Changes and Resolving Conflicts

Branching Strategy:

Adopt a clear branching strategy, such as Gitflow or GitHub flow, based on the project's needs.

Use feature branches for new features, hotfix branches for critical patches, and develop/main branches for ongoing development.

Commit Guidelines:

Follow commit message conventions to maintain a clear and informative Git history. Use meaningful commit messages that describe the purpose and context of the changes.

Regular Pull Requests:

Create smaller, focused pull requests instead of large, monolithic ones. This makes it easier for reviewers to understand and approve changes.

Regularly update your branch with the latest changes from the main branch to avoid conflicts.

Conflict Resolution:

When conflicts arise during merges or pull requests, address them promptly. Communicate with collaborators to understand conflicting changes and find consensus on resolutions.

Automated Testing:

Integrate automated testing into your workflow to catch issues early.

GitHub Actions or other CI/CD tools can be used to run tests automatically on pull requests.

Module 3: Building Efficient Container Images for Deployment

Introduction to Containerization and its Advantages

Containerization Overview:

Containerization is a lightweight, portable, and efficient packaging of applications and their dependencies. It encapsulates an application and its runtime environment, ensuring consistency across different environments.

Advantages of Containerization:

- **Portability:** Containers run consistently across different environments.
- **Isolation:** Applications and dependencies are isolated, reducing conflicts.
- **Scalability:** Easily scale applications up or down based on demand.
- **Resource Efficiency:** Containers share the host OS kernel, minimizing overhead.
- **Version Control:** Container images allow versioning and easy rollbacks.

Building Docker images: Docker files, Layering, and Optimization Techniques

Docker files:

Docker files are scripts that define how a Docker image should be built. They include instructions for installing dependencies, copying files, and configuring the environment.

Best practices include using a minimal base image and combining commands to reduce image layers.

Image Layers:

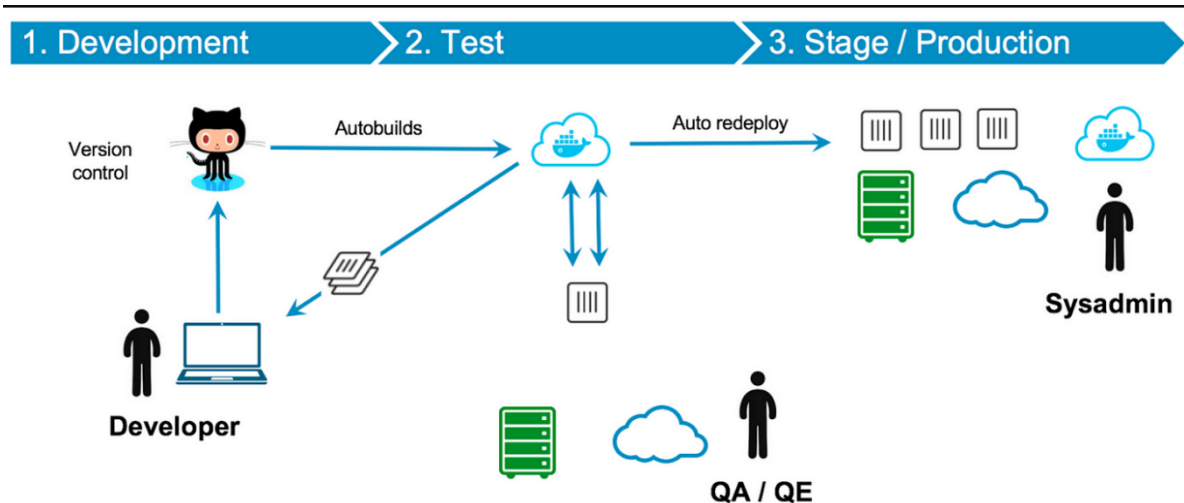
Docker images are composed of layers, each representing a set of file changes. Leveraging layers enhances image build speed and facilitates caching.

Optimize layers by grouping related commands and minimizing layer size.

Optimization Techniques:

- **Multi-stage Builds:** Use multi-stage builds to reduce the size of the final image by discarding unnecessary build artifacts.
- **Alpine Images:** Choose lightweight base images like Alpine Linux to minimize the image size.
- **Caching:** Leverage caching strategically to speed up image builds.

Leveraging Docker Compose for Local Development Environments



Docker Compose Overview:

Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define services, networks, and volumes in a YAML file.

Compose simplifies the orchestration of interconnected containers for local development.

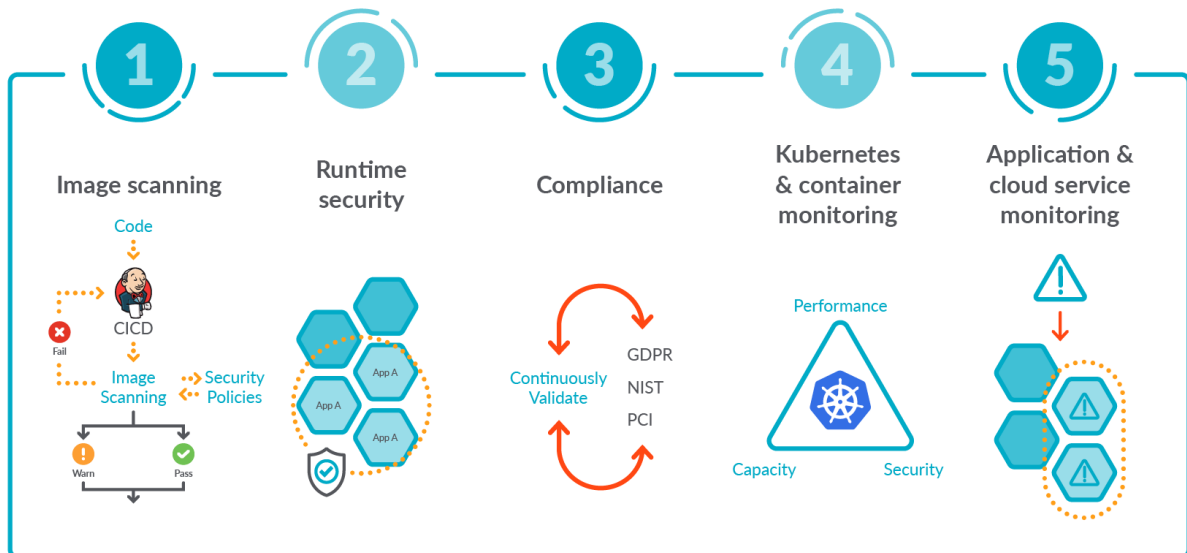
Local Development Workflow:

Create a `docker-compose.yml` file to define the services, dependencies, and configurations needed for local development.

Use commands like `docker-compose up` to start services and `docker-compose down` to stop them.

Best Practices for Creating Secure and Effective Container Images

Five Essential Workflows for Secure DevOps



Efficiency Best Practices:

- **Minimize Layers:** Reduce the number of layers in your Docker File to optimize image size and build speed.
- **Use Efficient Base Images:** Choose base images that are lightweight and tailored to the specific needs of your application.
- **Clean Up Unnecessary Dependencies:** Remove unnecessary files and dependencies from the final image.

Security Best Practices:

- **Regularly Update Base Images:** Keep base images up to date to patch security vulnerabilities.
- **Implement Least Privilege:** Run containers with the least necessary permissions to reduce the attack surface.
- **Scan for Vulnerabilities:** Utilize tools like Clair or Anchore to scan container images for security vulnerabilities.

Documentation:

Include README Files: Clearly document the purpose, usage, and configuration of your container images.

Versioning: Clearly define and document versioning for your images.

Module 4: Design for RESTful APIs with Swagger

Changelt	
GET	/resources/try Retrieve Deleteme object created
POST	/resources/try Does the same as the method below
GET	/resources/try/littletest a GET Test
default	
PUT	/resources/{version}/{context}/entity/{type}
POST	/resources/{version}/{context}/entity/{type}
GET	/resources/{version}/{context}/entity/{type}/{id}
DELETE	/resources/{version}/{context}/entity/{type}/{id}

Swagger, now known as OpenAPI Specification (OAS), is a framework for describing, producing, consuming, and visualizing RESTful web services. It offers a standardized approach to documenting and defining APIs, making it easier for developers to understand and integrate them. Swagger tools include a mix of open source, free, and commercial solutions to help with API development and documentation.

Understanding the importance of API design and documentation

API (Application Programming Interface) design and documentation are critical elements in the development of modern software applications, especially in the era of microservices and cloud computing. Swagger, now known as the OpenAPI Specification (OAS), plays a pivotal role in this domain. Let's delve into the importance of API design and documentation, and how Swagger contributes to it.

Importance of API Design and Documentation

- **Clarity and Usability:** A well-designed API with clear documentation is easier for developers to understand and use. Good documentation acts as a guide, explaining how to effectively interact with the API.
- **Consistency:** Consistent API design helps maintain a standard across different services within an organization, making it easier for developers to work with various APIs.
- **Efficiency in Development:** Proper API design and documentation can significantly speed up the development process, as developers spend less time figuring out how to use the API and more time actually using it.
- **Integration and Scalability:** Well-documented APIs simplify the process of integrating different systems or services, especially in complex architectures like microservices.
- **Error Reduction:** Clear documentation reduces the chances of misunderstandings and errors in API usage, leading to more robust applications.
- **Community and Ecosystem Growth:** For public APIs, good documentation can attract external developers, fostering a community and potentially leading to an ecosystem around the API.

Introduction to Swagger/OpenAPI for RESTful APIs Design

Swagger provides a language-agnostic way to document your API's operations, inputs, outputs, and errors.

How Swagger (OpenAPI) Helps


- **Standardization:** Swagger provides a standard way of describing RESTful APIs. This standardization ensures that APIs are described in a consistent and understandable format.
- **Interactive Documentation:** Swagger tools like Swagger UI create interactive documentation from an OpenAPI specification. This allows developers to explore APIs, send test requests, and view responses directly from the documentation.
- **Code Generation:** Swagger Codegen can automatically generate server stubs, client libraries, and API documentation, which speeds up development and ensures that implementations stay in sync with documentation.
- **Design-First Approach:** Swagger encourages a design-first approach to API development, where the API's design is thought through and documented before being implemented. This often leads to better API design decisions.
- **Validation and Testing:** Swagger tools can validate an API's specification against the OpenAPI standard, ensuring that the API documentation is accurate and error-free.

Impact of Swagger

- **Improved Developer Experience:** Developers find it easier and faster to work with APIs that are documented using Swagger, leading to increased productivity.
- **Enhanced Communication:** Swagger's standardized format improves communication among team members and between teams, especially in large organizations or in projects with many contributors.
- **Easier Adoption:** For public APIs, Swagger documentation makes it easier for external developers to adopt the API, thereby potentially increasing its reach and impact.
- **Quality Assurance:** The ability to test and validate APIs against their Swagger documentation helps maintain high quality and reliability of the API and its implementation.

Defining API Specifications using Swagger Editor

Defining API specifications using Swagger Editor is a crucial step in designing and documenting RESTful APIs. Swagger Editor provides an intuitive interface for writing OpenAPI (formerly Swagger) specifications.

Step 1: Access Swagger Editor	Visit https://swagger.io/tools/swagger-editor/ 
Step 2: Understand the Structure of an OpenAPI Specification	<ul style="list-style-type: none"> • OpenAPI Object: The root of the document. It must include openapi (the version), info (metadata about the API), and paths (the available paths/endpoints). • Info Object: Contains metadata like the API's title, description, version, etc. • Paths Object: Describes the available paths and operations (GET, POST, PUT, DELETE, etc.) on those paths.
Step 3: Define Basic API Information	Start by providing some basic information about your API: <pre> paths: /items: get: summary: List all items responses: '200': description: Successfully retrieved items </pre>
Step 4: Define API Paths and Operations	Defining a Path: Add a new path to your API.

	<pre>paths: /items: get: summary: List all items responses: '200': description: Successfully retrieved items</pre> <p>Adding Operations: For each path, you can define operations (HTTP methods). The above example defines a GET operation on /items.</p>
Step 5: Specify Operation Details	<p>Add details for each operation, such as responses, parameters, request bodies, etc.</p> <pre>responses: '200': description: A list of items content: application/json: schema: type: array items: \$ref: '#/components/schemas/Item'</pre>
Step 6: Define Data Models (Schemas)	<p>Use the components/schemas section to define data structures.</p> <pre>components: schemas: Item: type: object properties: id: type: integer name: type: string</pre> <p>This defines an Item object with id and name properties.</p>
Step 7: Add More Details and Operations	<p>Parameters: Define path parameters, query parameters, etc.</p>

	<pre>parameters: - in: query name: filter schema: type: string description: Filter for the list of items.</pre> <p>Other HTTP Methods: Add operations for POST, PUT, DELETE, etc., along with their request bodies and responses.</p>
Step 8: Validate and Export the API Specification	<ul style="list-style-type: none"> • Validation: Swagger Editor automatically validates your specification. Look out for any errors or warnings. • Export: Once your API is defined, you can export the YAML or JSON file for use in Swagger Codegen or other tools.
Step 9: Explore Swagger Editor Features	<ul style="list-style-type: none"> • Swagger Editor provides several features like syntax highlighting, error reporting, and a real-time preview of your API documentation. • Experiment with these features to become more familiar with OpenAPI specifications and the capabilities of Swagger Editor.

Defining API specifications in Swagger Editor involves carefully structuring your API using OpenAPI standards, from basic information to detailed endpoint descriptions. It's a process of incrementally building up the API, ensuring accuracy and completeness in the specification. This structured approach helps create clear, comprehensive, and usable API documentation, facilitating better development and integration experiences.

Generating Client SDKs and Server Stubs with Swagger Codegen

Generating client SDKs (Software Development Kits) and server stubs using Swagger Codegen is a powerful feature that can greatly enhance the development process by automating the creation of boilerplate code for various programming languages and frameworks.

Step 1: Install Swagger Codegen	Download the Swagger Codegen CLI jar file from its GitHub Releases page .
Step 2: Prepare Your Swagger Specification	Make sure you have your Swagger/OpenAPI specification file ready. This file should be in either JSON or YAML format and contain the complete definition of your API.
Step 3: Generate Client SDK or Server Stubs Use the Swagger Codegen CLI to generate code in the language and framework of your choice.	<p>Generating a client SDK: Suppose you want to generate a client SDK for a JavaScript application. The command would look like this:</p> <pre>swagger-codegen generate -i path/to/api-spec.yaml -l javascript -o /path/to/output/dir</pre> <p>Here, -i specifies the input specification file, -l sets the language for the SDK, and -o determines the output directory.</p> <p>Generating Server Stubs: If you want to generate server stubs, for example, in Java using Spring, the command would be:</p> <pre>swagger-codegen generate -i path/to/api-spec.yaml -l spring -o /path/to/output/dir</pre>
Step 4: Explore Codegen Options	<p>Swagger Codegen supports a wide range of languages and frameworks. You can explore all available options with:</p> <pre>swagger-codegen langs</pre> <p>For specific configuration options for each language/framework, use:</p> <pre>swagger-codegen config-help -l [language]</pre>

Step 5: Customize the Generation Process	<p>You can customize the generation process to suit your project needs:</p> <ul style="list-style-type: none"> • Custom Templates: Swagger Codegen allows you to define custom templates to control how the code is generated. • Configuration File: For complex customizations, you can use a configuration file in JSON format to specify additional options.
Step 6: Integrate Generated Code into Your Project	<p>After generating the code, integrate it into your project:</p> <ul style="list-style-type: none"> • For client SDKs, this might involve adding the generated code as a module or library in your client application. • For server stubs, you would typically include the generated code in your server project, and then implement the business logic for each endpoint.

Best Practices

- Consistency in API Design: Be consistent with naming conventions, request/response formats, and error handling.
- Use Examples in Swagger Documentation: Provide examples in your Swagger documentation to clarify usage.
- Validation and Testing: Validate your Swagger specification and test generated SDKs and server stubs.
- Version Control: It's generally a good practice to keep the generated code out of version control and generate it as part of your build process.
- Customization: Utilize customization features of Swagger Codegen to align the generated code with your project standards and architecture.
- Regular Updates: Keep your API specification and generated code in sync, especially after making changes to the API.

Swagger Codegen is a versatile tool that automates the creation of client SDKs and server stubs, saving time and reducing the potential for human error. By following these steps, you can efficiently generate code for your APIs in the language.

Module 5: Efficient Development Practices

Creating an efficient development workflow is crucial in modern software engineering. It streamlines processes, enhances productivity, and reduces the likelihood of errors. This module will cover the integration of version control, image building, API design, automated CI/CD pipelines, and the generation of API documentation.

Combining Version Control, Image Building, and API Design

Version Control with Git:

- Use Git for version control to manage and track changes in your codebase.
- Create a repository (repo) for your project.
- Organize your code into branches for features, bug fixes, and releases.
- Regularly commit and push changes to the remote repository.

Image Building with Docker:

Use Docker to create container images of your application.

Step 1: Create a Dockerfile in your project's root directory.	<pre>FROM node:14 WORKDIR /app COPY package*.json ./ RUN npm install COPY . . EXPOSE 8080 CMD ["node", "app.js"]</pre>
Step 2: Build a Docker image and tag it appropriately.	<pre>docker build -t myapp:latest .</pre>

API Design with Swagger/OpenAPI:

- Design your API using Swagger (OpenAPI) specifications.
- Store the Swagger spec in your project, typically in a YAML or JSON file like api-spec.yaml.
- Keep the API spec version-controlled along with your code.

Implementing Automated CI/CD Pipelines for Continuous Integration

1. **Setting Up a CI/CD Tool** (like Jenkins, GitLab CI, CircleCI):
 - Choose a CI/CD tool compatible with your project and infrastructure.
 - Set up a CI/CD server or use a cloud-based service.
2. **Creating a CI/CD Pipeline:**
 - Define your pipeline in a configuration file (e.g., .gitlab-ci.yml for GitLab CI).
 - Specify pipeline stages: build, test, deploy.
 - Configure triggers for the pipeline (e.g., on push to specific branches).In yaml:

```
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - docker build -t myapp:$CI_COMMIT_REF_NAME .

test_job:
  stage: test
  script:
    - docker run myapp:$CI_COMMIT_REF_NAME npm test

deploy_job:
  stage: deploy
  script:
    - echo "Deploying application..."
```

3. **Integrating Automated Testing:**
 - Include automated tests in your pipeline.
 - Configure the pipeline to run tests every time code is pushed to the repo.

Utilizing Version-Controlled Code for Building Container Images

Automating Image Builds:

- Set up your CI/CD pipeline to automatically build Docker images from your codebase.
- Use tags to manage different versions of the images.
- Push built images to a container registry.

```
docker build -t myregistry.com/myapp:$CI_COMMIT_REF_NAME .
docker push myregistry.com/myapp:$CI_COMMIT_REF_NAME
```

Generating API Documentation from Swagger Specifications

- **Using Swagger Codegen:**
 - Use Swagger Codegen to generate API documentation automatically from your Swagger spec.
 - Install Swagger Codegen CLI.
- **Generating Documentation:**
 - Run Swagger Codegen CLI to generate documentation.

```
swagger-codegen generate -i api-spec.yaml -l html -o ./docs
```
 - This will generate an HTML documentation file from your api-spec.yaml.
- **Integrating Documentation Generation into CI/CD:**
 - Add a step in your CI/CD pipeline to generate and publish the API documentation whenever the API spec changes.

An efficient development workflow is key to successful software projects. By integrating version control with Docker for image building, designing APIs with Swagger, implementing automated CI/CD pipelines, and generating consistent documentation, you create a robust, scalable, and maintainable development ecosystem. This workflow not only improves productivity but also enhances the overall quality of the software development lifecycle.

Module 6: Hands-on Workshop: Building Efficient Software Development Workflow

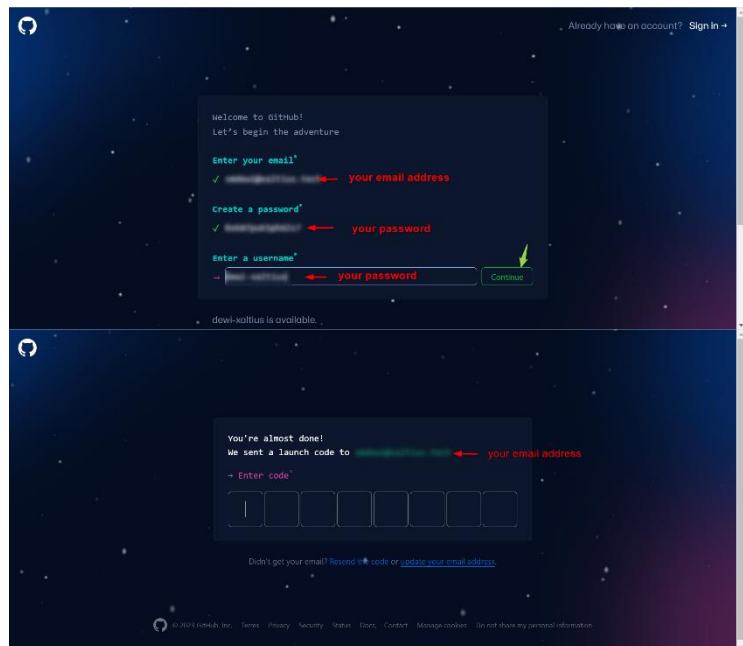
Collaborative Exercises: Setting up Git/GitHub, Building Docker images, Swagger Design

Setting up GitHub

Step 1: Create your GitHub account.

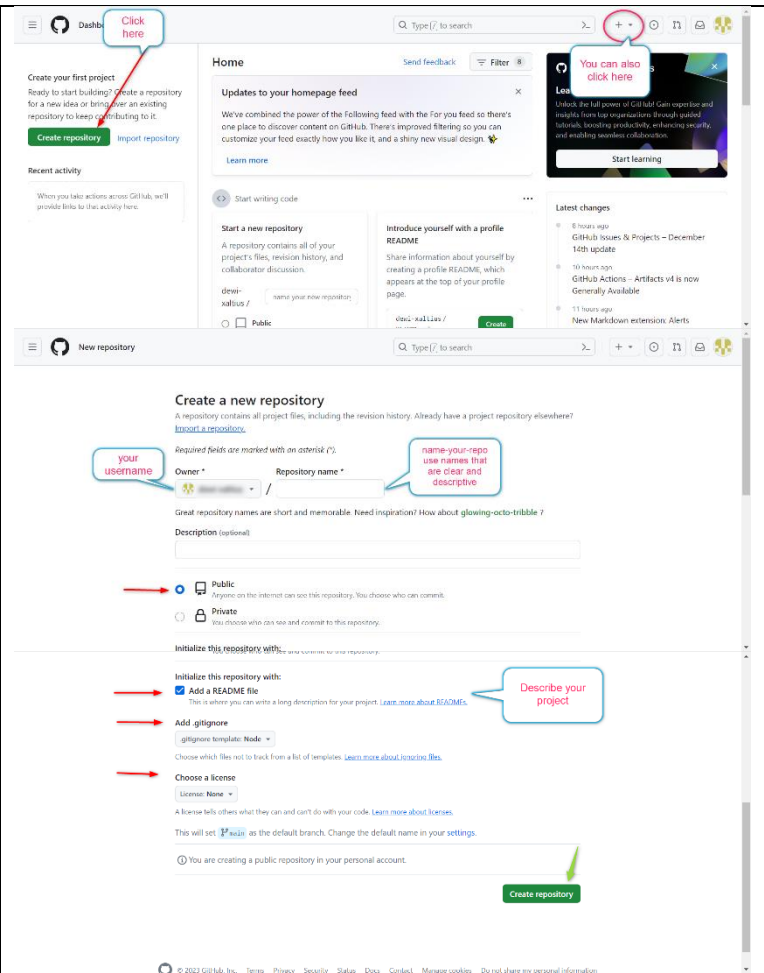
After you enter your email address, password and username, GitHub will ask you to verify your account. You will receive an email with your launch code.

Please skip personalization for now. You can always go back to this at a later stage.



The screenshot displays the GitHub account creation interface. The top section, titled 'Welcome to GitHub!', prompts the user to 'Let's begin the adventure' and provides fields for 'Enter your email*', 'Create a password*', and 'Enter a username*'. Each field has a red arrow pointing to it with a label: 'your email address', 'your password', and 'your password' respectively. A 'Continue' button is visible. Below this, a message states 'devkit-kali is available'. The bottom section, titled 'You're almost done!', informs the user that a launch code has been sent to their email and provides a field to 'Enter code'. A red arrow points to the email address field with the label 'your email address'. At the bottom, there is a link for 'Didn't get your email? Request a new code or update your email address'.

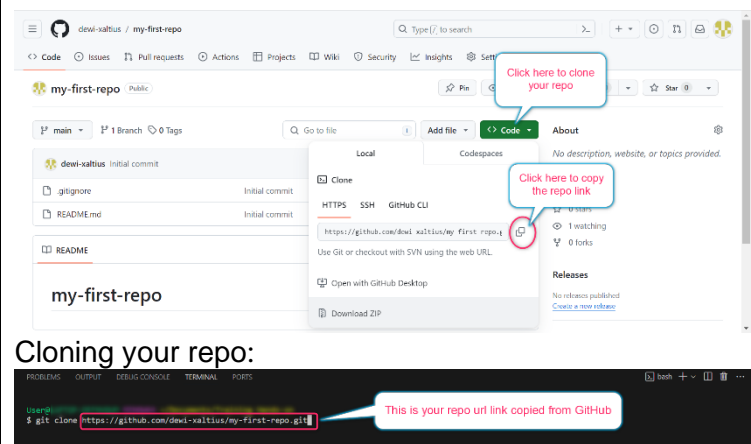
Step 2: Create a new repo



Step 3: Clone your repo into your local environment.

Click the Code Tab to open the option for local environment. Choose HTTPS and copy your repo link.

Open your terminal and type the git command in the appropriate folder:



Cloning your repo:

This creates a cloned copy of the repo in your local environment with the same name folder. If you would like to create a folder with a different name, you could specify the name of the new folder after the repo url.

Step 4: Commit your changes

First you need to configure your git set up:

```
User@ ~ /Documents/Training Hands-on/my-first-repo (main)
$ git config --global user.email "your email address"

User@ ~ /Documents/Training Hands-on/my-first-repo (main)
$ git config --global user.name "your user name"
```

To commit your changes to your repo, you first must save any changes you make to the file. Then you add the file:

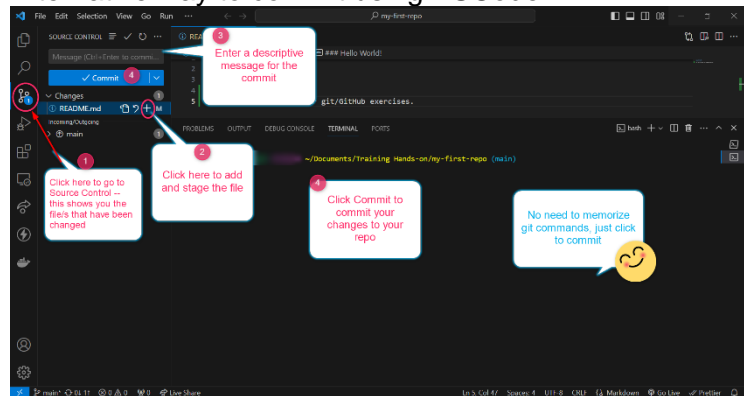
```
User@ ~ /Documents/Training Hands-on/my-first-repo (main)
$ git add README.md
```

Now you are ready to commit!

```
User@ ~ /Documents/Training Hands-on/my-first-repo (main)
$ git commit -m "add Hello World msg"
[main 1f67b9a] add Hello World msg
1 file changed, 3 insertions(+), 1 deletion(-)
```

Once you've committed your changes, you will see the details for the commit.

Alternative way to commit using VSCode:



- Go to Source Control
- Add and stage the file/s you want to commit
- Enter a commit message to describe the changes
- Click Commit

Step 5: Create a new branch

A new branch is usually created to keep changes separate from the main branch until it is tested and ready to be merged with the main branch.

```
User@ ~ /Documents/Training Hands-on/my-first-repo (main)
$ git branch feature1
```

```
User@LAPTOP-DEFWQUM MINGW64 ~/Documents/Training Hands-on/my-first-repo (main)
$ git branch
feature1
* main
```

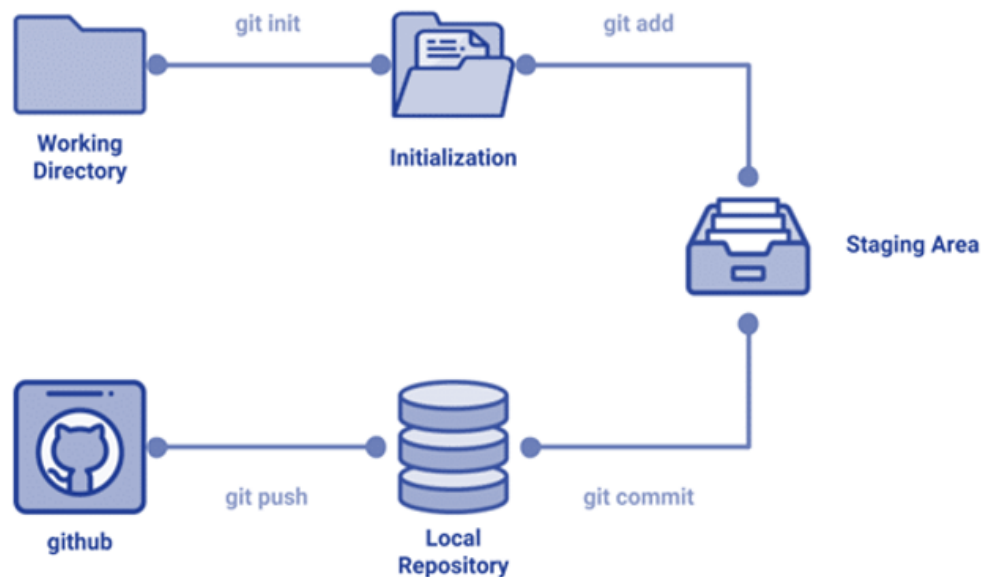

Building Container image with Docker

<p>Step 1: Create a Dockerfile:</p> <p>In the root of your project, create a Dockerfile.</p>	<p>Add the following content to Dockerize the React app:</p> <pre>FROM node:14 WORKDIR /usr/src/app COPY package.json ./ COPY package-lock.json ./ RUN npm install COPY . . EXPOSE 3000 CMD ["npm", "start"]</pre>
<p>Step 2: Build the Docker Image</p>	<pre>docker build -t react-docker-swagger-app .</pre>
<p>Step 3: Run the container</p>	<p>Run your React app inside a Docker container:</p> <pre>docker run -p 3000:3000 react-docker-swagger-app</pre> <p>The app should be accessible at http://localhost:3000.</p>

Designing a Mock API with Swagger

<p>Create a Swagger Specification for a Mock API</p>	<p>Use Swagger Editor (Swagger Editor) to design a mock API for your app.</p> <p>Define a simple API, like fetching a list of items:</p> <pre>openapi: 3.0.0 info: title: Mock API for React App version: 1.0.0 paths: /items: get: responses: '200': description: Returns a list of items.</pre> <p>Save the Swagger spec (swagger.yaml) in your project directory.</p>
--	--

Applying Efficient Workflow Techniques to a Sample Project



Applying efficient workflow practices to a sample project involves several key steps, from initial setup to continuous deployment. For this learner's guide, let's use a sample React project as an example. We'll go through setting up version control with Git, writing code, reviewing changes, automating tests and deployments with CI/CD, and more.

Setting Up a Version Control System with Git

Version control is essential for managing changes to the project over time and collaborating with others. Git is a widely used version control system that tracks changes and facilitates collaborative work. Initializing a Git repository in your project directory allows you to start tracking changes. A `.gitignore` file is crucial to exclude temporary or non-source code files (like `node_modules`) from being tracked. Regular commits with clear messages document the history of your project, making it easier to understand changes and revert to previous states if needed.

Setting Up the React Project

Setting up the project structure is key to maintaining organization and clarity. Using tools like `create-react-app` streamlines the initial setup by creating a standard React application structure with all necessary configurations and dependencies. Organizing the project into specific directories like `components`, `utils`, and `assets` helps in managing the codebase, making it more maintainable and navigable for developers.

Writing and Reviewing Code

Adopting a feature branch workflow allows developers to work on new features or bug fixes without disturbing the main codebase. Each new feature or fix is developed in a separate branch, which keeps the main branch stable. Regular commits keep track of changes, while pull requests provide a platform for code review. This practice enhances code quality and fosters collaboration among team members.

Automating Testing and Deployment with CI/CD

Continuous Integration (CI) and Continuous Deployment (CD) are practices that automate testing and deployment of your application. CI tools run tests automatically on every commit, ensuring that new changes do not break the existing code. CD automates the deployment of the application to a hosting platform, ensuring a seamless and consistent deployment process. This automation reduces manual errors, saves time, and ensures a high-quality, stable application.

Utilizing Containerization with Docker

Containerization, using Docker, encapsulates the application and its environment, ensuring consistency across different development, testing, and production environments. A Dockerfile defines the steps to create a container image of the application, which can then be run in any environment that supports Docker. This process eliminates the “it works on my machine” problem and facilitates easier deployment and scaling.

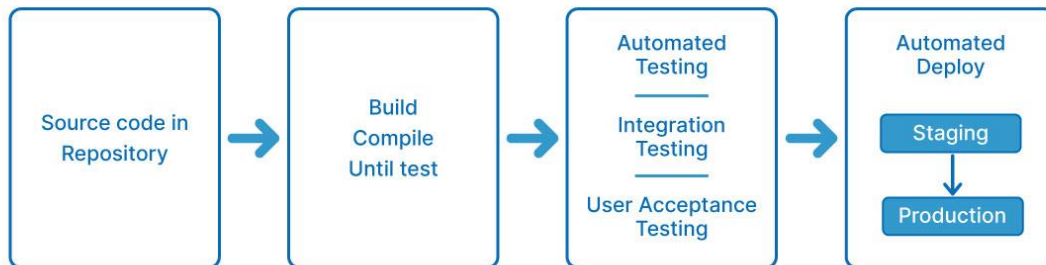
Monitoring and Feedback

Implementing monitoring and feedback mechanisms is crucial for maintaining the performance and health of your application post-deployment. Monitoring tools help track the application’s operational metrics and log data, which are invaluable for diagnosing issues. Gathering user feedback is equally important for understanding the user experience and identifying areas for improvement, ensuring the application continues to evolve and meet user needs effectively.

Efficient workflow practices in software development involve a well-organized version control system, regular and clear commit messages, code reviews, automated testing, continuous integration, optional containerization, and continuous deployment. Additionally, monitoring and user feedback are crucial for ongoing improvement. By following these steps, you can establish a robust and efficient workflow for your React project or any other software development project.

Integrating Automated CI/CD Pipelines for Version-Controlled Code

CI/CD PIPELINE



Integrating automated Continuous Integration/Continuous Deployment (CI/CD) pipelines into version-controlled code is a cornerstone of modern software engineering practices. This integration streamlines the process of software development, testing, and deployment, ensuring a more robust and efficient release cycle.

Understanding CI/CD and Version Control

- Continuous Integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. It involves automatically testing code changes in a shared repository.
- Continuous Deployment (CD) extends CI by automatically deploying all code changes to a production or staging environment after the build stage.
- Version Control Systems (VCS) like Git help manage and track changes to the codebase, making it easier to integrate CI/CD pipelines.

Best Practices

- Maintain a Clean Codebase: Ensure that the main branch is always deployable.
- Feature Branch Workflow: Use feature branches for new features or fixes and merge them into the main branch after testing.
- Automate as Much as Possible: Aim for full automation – from code integration to deployment.
- Security: Always keep security in mind, especially with dependencies and during deployment.
- Documentation: Keep your pipeline configuration and related documentation up to date.

Integrating automated CI/CD pipelines with version-controlled code is an essential strategy for modern software development. It enhances collaboration, speeds up

the release process, and reduces the potential for errors. By following these steps and best practices, developers can ensure a smooth, efficient, and reliable software development and deployment process.

Module 7: Future Trends in Software Development Workflows

The software development landscape is constantly evolving, and the workflows we use today will undoubtedly transform in the years to come. Let's explore these future trends in software development workflows, focusing on emerging practices, the impact of DevOps, automation, and cloud-native development, and the overall evolving landscape.

Exploring Emerging Trends in Software Development Practices

Microservices Architecture: Breaking down applications into smaller, independently deployable services. This approach offers flexibility, scalability, and facilitates continuous delivery and deployment.

Serverless Computing: Developers are increasingly adopting serverless architectures, where cloud providers dynamically manage the allocation of machine resources. This trend emphasizes a focus on code and services, reducing the need for traditional server management.

AI-powered Development: Artificial intelligence (AI) is rapidly infiltrating the software development space, offering capabilities like:

- Code generation and autocompletion: Tools like GitHub Copilot and Tabnine assist developers by suggesting code snippets and even entire functions based on context.
- Automated testing and bug detection: AI-powered testing frameworks can analyze code and identify potential bugs with greater accuracy and efficiency than traditional methods.
- Personalized learning and recommendations: AI algorithms can personalize developer workflows by recommending relevant tools, libraries, and learning resources based on individual needs and preferences.

Artificial Intelligence and Machine Learning Integration: AI and ML are being integrated into development processes for predictive analytics, intelligent automation, and enhancing user experiences.

Low-code/No-code Development: Platforms that allow building applications with minimal coding effort are becoming popular, especially for rapid prototyping and for users with limited programming expertise. Democratizing software development, low-code and no-code platforms enable individuals with limited coding experience to build basic applications through visual interfaces and drag-and-drop functionality. This expands the talent pool and fosters citizen development within organizations.

Blockchain Integration: Blockchain technology, known for its secure and transparent distributed ledger capabilities, is finding its way into software development in various ways, including:

- **Decentralized applications (dApps):** Building applications that run on peer-to-peer networks, eliminating the need for central servers and fostering greater user control.
- **Supply chain management and provenance tracking:** Ensuring the authenticity and traceability of goods and materials throughout their lifecycle.
- **Secure data storage and sharing:** Providing a tamper-proof and immutable way to store and share sensitive data.

Containerization and Kubernetes: The use of containers (like Docker) for deploying applications is a significant trend. Kubernetes has emerged as a leading container orchestration tool, managing containerized applications' deployment, scaling, and operations.

Recognizing the Impact of DevOps, Automation, and Cloud-native Development

DevOps

The DevOps approach, merging development (Dev) and operations (Ops) into a single team, fosters closer collaboration and accelerates software delivery. Automation plays a crucial role in DevOps, enabling tasks like infrastructure provisioning, configuration management, and continuous integration/continuous delivery (CI/CD) pipelines to be performed automatically, freeing up developer time for more creative work.

The fusion of Agile and DevOps practices is becoming more pronounced. This integration focuses on improving collaboration between development and operations teams, speeding up the delivery process, and enhancing software quality.

Automation

Increased Emphasis on CI/CD Pipelines: Continuous Integration (CI) and Continuous Deployment (CD) are key components of modern software workflows. They enable automatic software building, testing, and deployment, leading to faster release cycles.

Infrastructure as Code (IaC): Managing infrastructure through code rather than manual processes. This practice supports DevOps initiatives by automating the provisioning and management of infrastructure.

Cloud-native Development: Embracing cloud-native technologies for building and running applications to leverage the cloud computing delivery model's full potential. This includes microservices, containers, Kubernetes, and serverless architectures.

Reflecting on the evolving landscape of software development workflows

Increased Focus on Security (DevSecOps): Integrating security practices into the DevOps process (DevSecOps) to ensure continuous security consideration. This trend reflects a shift-left approach in security, integrating it earlier in the development cycle.

Remote and Distributed Teams: Advances in collaboration tools and practices are enabling more effective remote and distributed software development, a trend accelerated by global events like the COVID-19 pandemic.

Personalization and User Experience Focus: As user expectations grow, there's an increased emphasis on creating personalized and seamless user experiences, leveraging data analytics, and user feedback in the development process.

Sustainability in Software Development: Growing awareness of environmental impacts leads to a focus on sustainable software development practices, optimizing resource usage and energy efficiency.

The future of software development workflows is shaped by the need for faster delivery, higher quality, and more secure software, while also adapting to the latest technological advancements and changing global dynamics. The focus on DevOps, automation, cloud-native technologies, and emerging trends like AI, low-code platforms, and sustainability are driving significant changes in how software is developed, deployed, and maintained. As these trends continue to evolve, they will further influence and transform software development workflows, making them more efficient, collaborative, and user centric.

WITH OUR WEALTH OF INDUSTRY EXPERIENCE, NTUC LEARNINGHUB PROVIDES HIGH-QUALITY TRAINING PROGRAMMES THAT ARE INNOVATIVE, CONTEMPORARY AND AFFORDABLE.

These consist of programmes in:

- Infocomm Technology
- Employability & Literacy
- Business Excellence
- Human Resources
- Healthcare
- Security
- Workplace Security & Health
- Foreign Worker Training

For more information, please visit our website at <http://www.ntuclearninghub.com/> or call us at **6336-5482**

