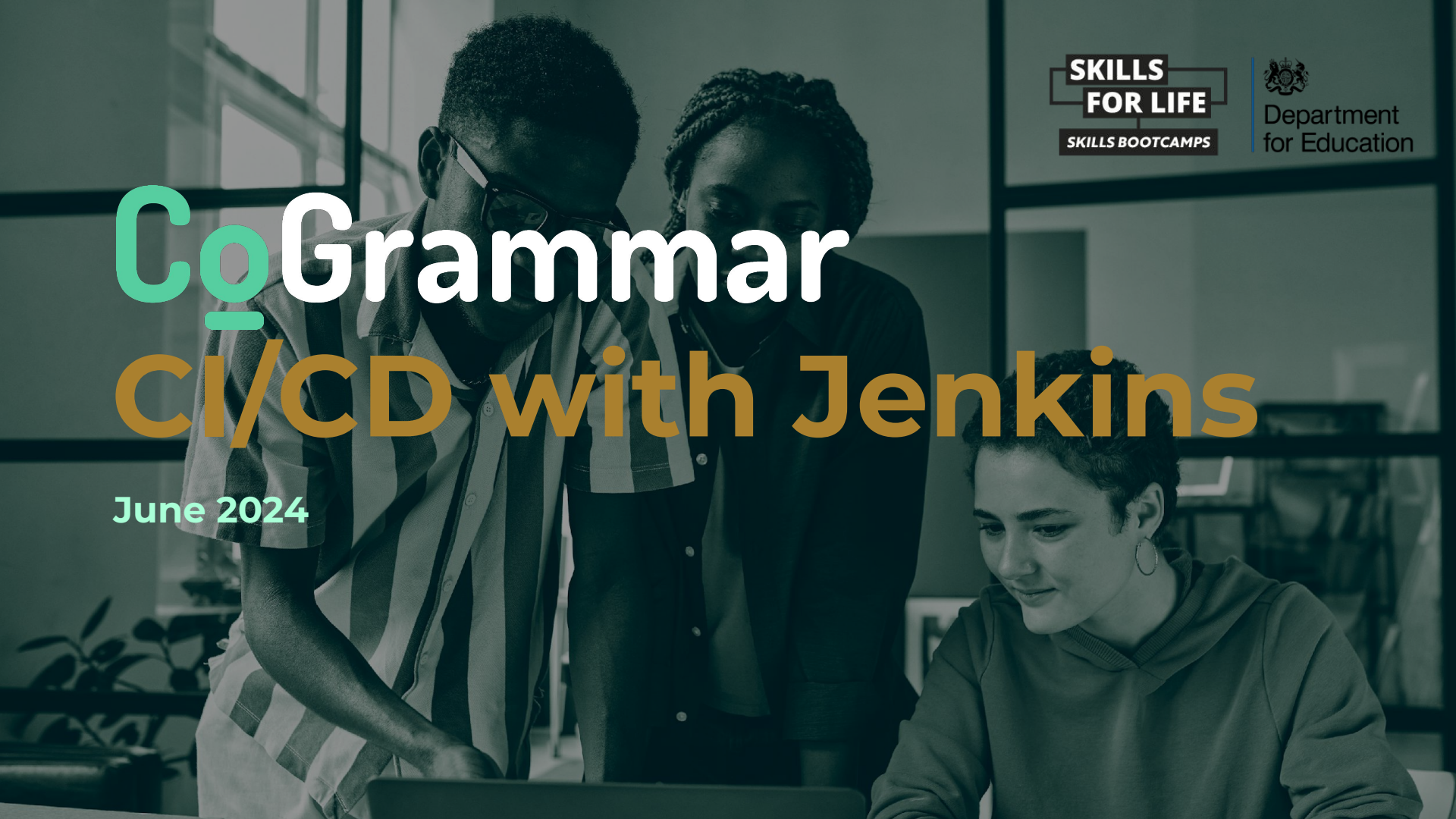
# Welcome to the CoGrammar CI/CD with Jenkins

## The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
CI/CD with Jenkins
June 2024

# Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

# **Software Engineering Session Housekeeping** cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

# **Software Engineering Session Housekeeping** cont.

- "Please check your spam folders for any important communication from us. If you have accidentally unsubscribed, please reach out to your support team.“
- Rationale here: Career Services, Support, etc will send emails that contain NB information as we gear up towards the end of the programme. Students may miss job interview opportunities, etc.

# Skills Bootcamp
# 8-Week Progression Overview

## Fulfil 4 Criteria to Graduation

### ✅ Criterion 1: Initial Requirements

- ***Timeframe:*** First 2 Weeks
- ***Guided Learning Hours (GLH):*** Minimum of 15 hours
- ***Task Completion:*** First four tasks

### ✅ Criterion 2: Mid-Course Progress

- ***Guided Learning Hours (GLH): 60***
- ***Task Completion:*** 13 tasks

CoGrammar

# Skills Bootcamp Progression Overview

## ✅ Criterion 3: Course Progress

- ***Completion:*** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- ***Interview Invitation:*** Within 4 weeks post-course
- ***Guided Learning Hours:*** Minimum of 112 hours by support end date (10.5 hours average, each week)

## ✅ Criterion 4: Demonstrating Employability

- ***Final Job or Apprenticeship Outcome:*** Document within 12 weeks post-graduation
- ***Relevance:*** Progression to employment or related opportunity

CoGrammar

# Learning Objectives

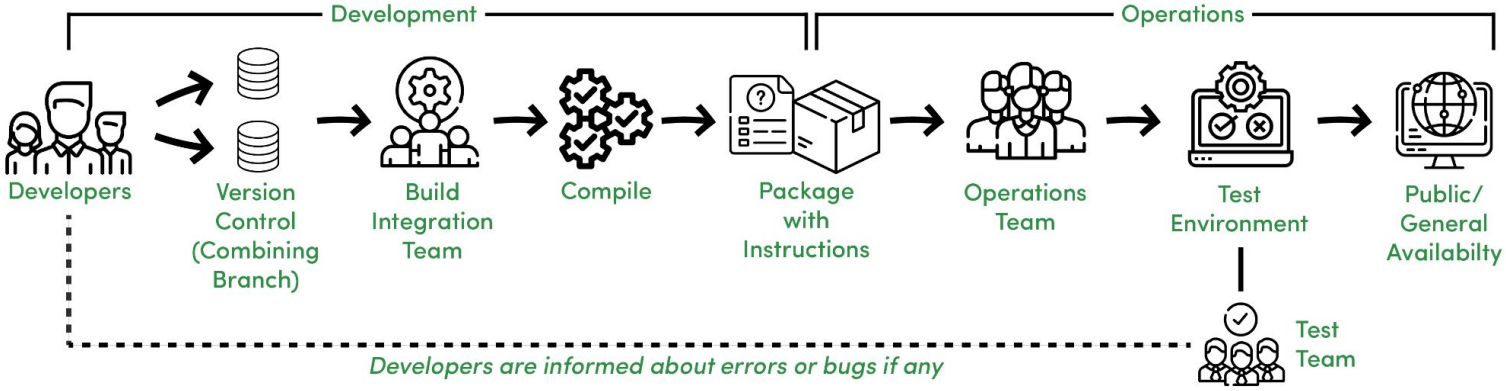
- Define and differentiate **Continuous Integration** (CI) and **Continuous Delivery/Deployment** (CD).
- Identify **common CI/CD tools** and understand their role in automating the pipeline.
- Describe the **stages** of a typical CI/CD pipeline and how they **interconnect**
- Provide some **example** of CI/CD pipeline in **real life** scenarios

# Introduction

CI/CD: Streamlining the Software Release Process

CoGrammar

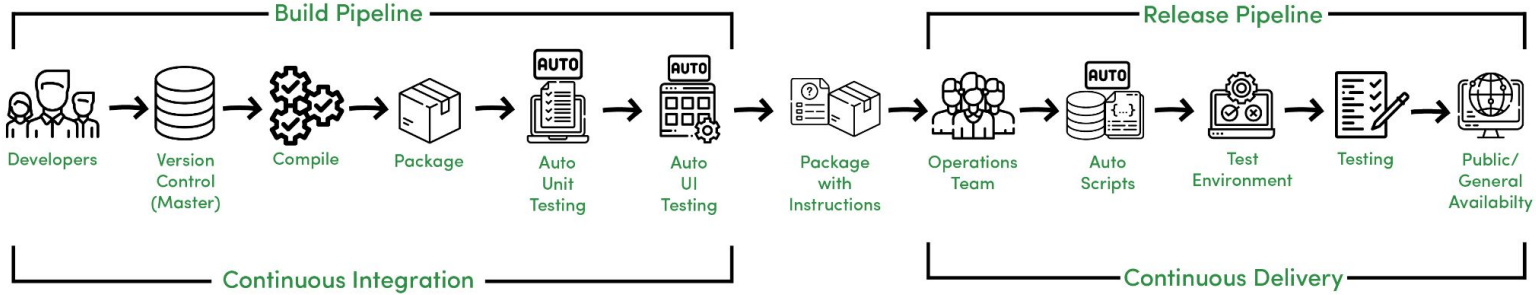
# Tradition Software Development Life Cycle (SDLC)




CoGrammar

Source

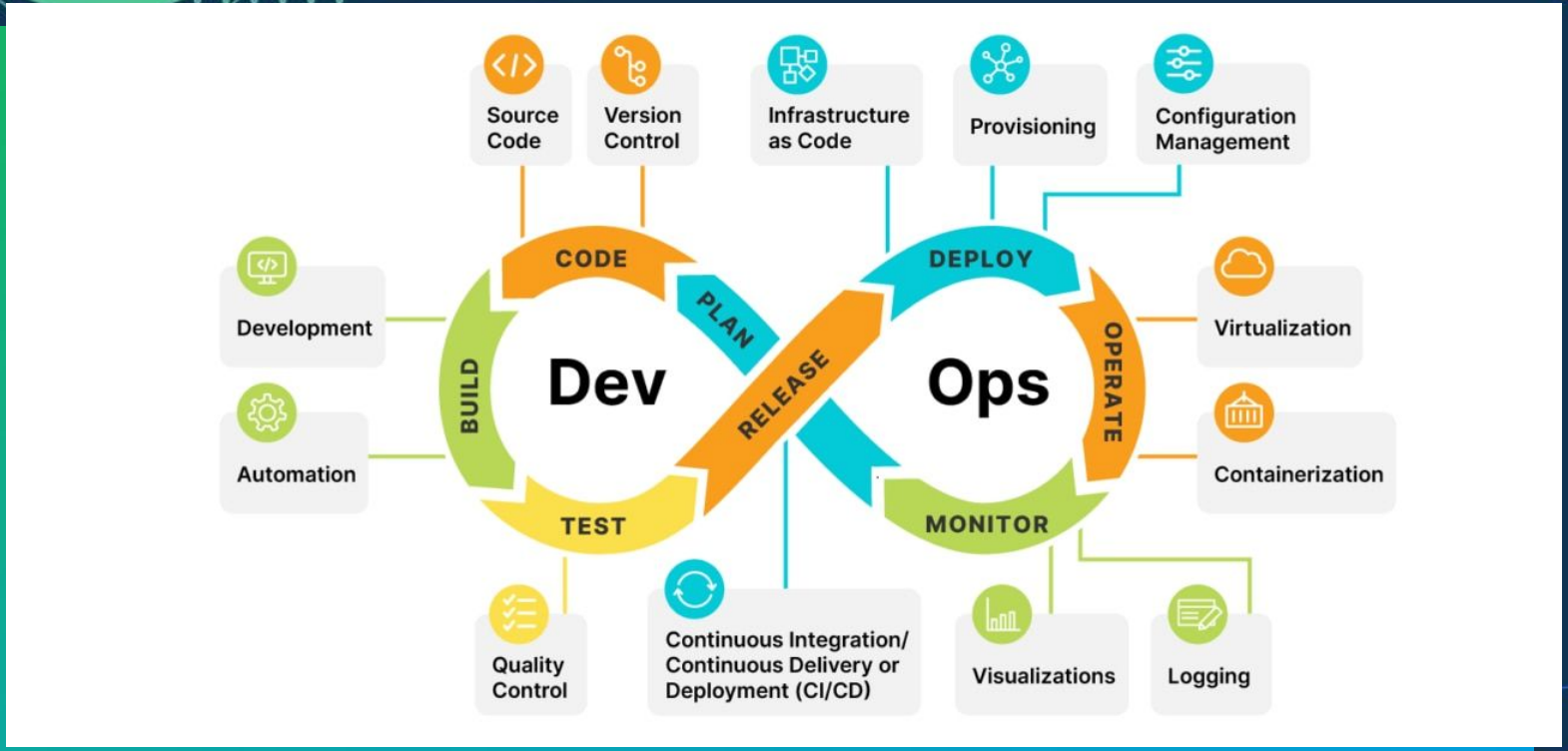
# CI/CD Pipeline




CoGrammar

Source

# CI/CD Pipeline



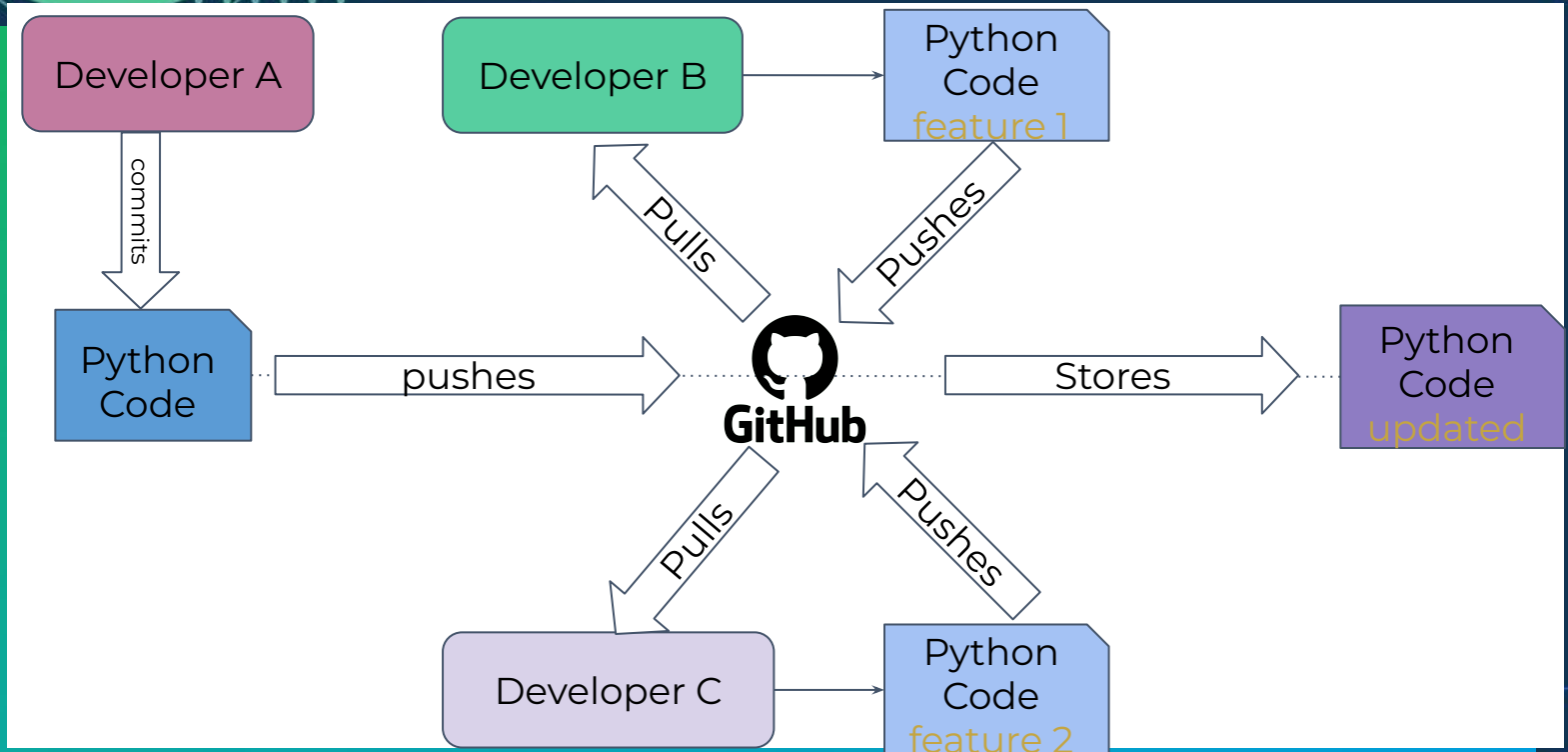

CoGrammar

Source

# Continuous Integration (CI)

CoGrammar

# What is Integration?

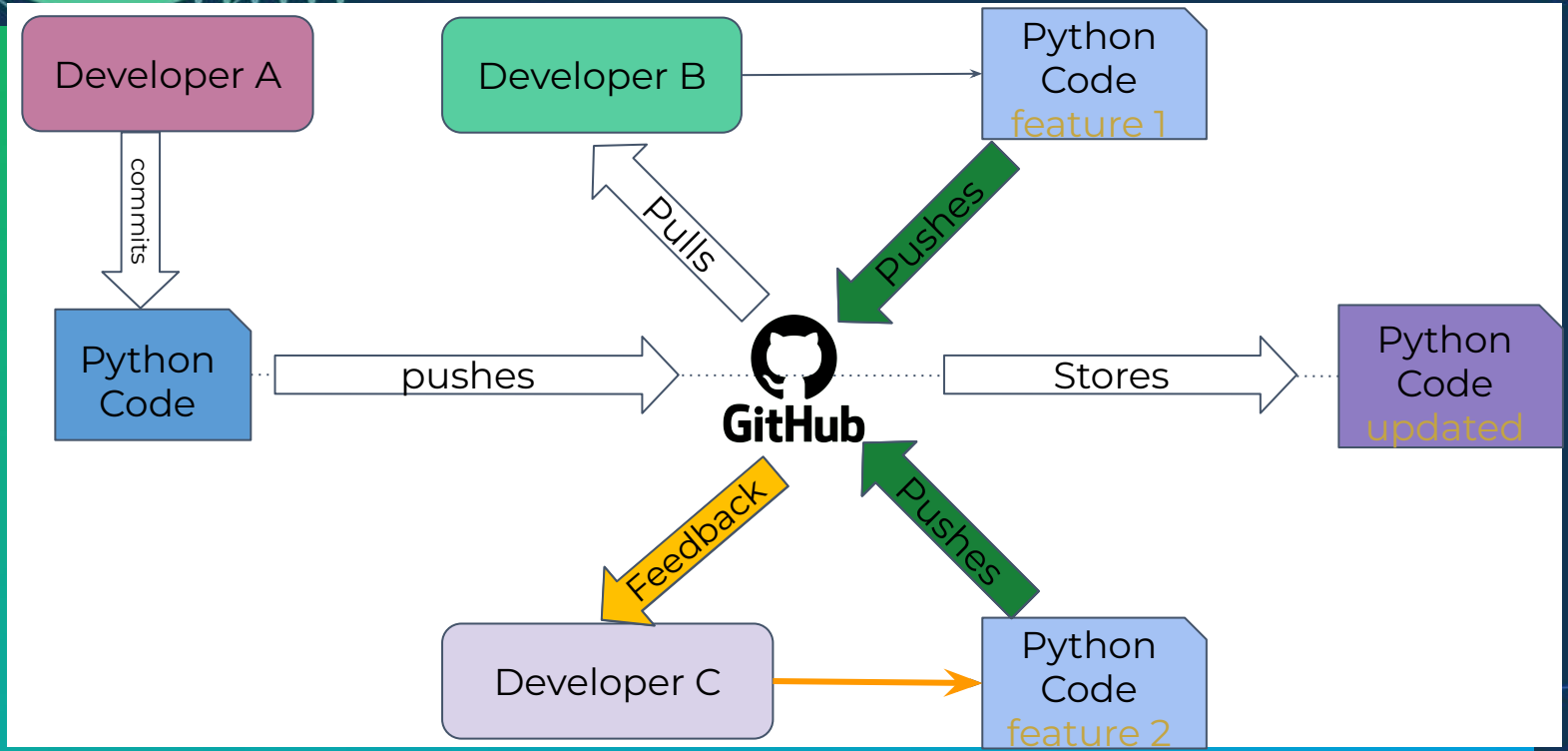



CoGrammar

# Continuous Integration: Building Quality In

- **Continuous Integration** (CI) is a software development practice where developers frequently merge their code changes into a shared repository.
- Developers usually integrates several times a day.
- Executes **automated tests** to **identify bugs** and **regressions** early.
- Each integration is verified by an **automated build**: compile the code and also **run automated tests**
    - Question: Why are automated tests run?

# What is Integration?

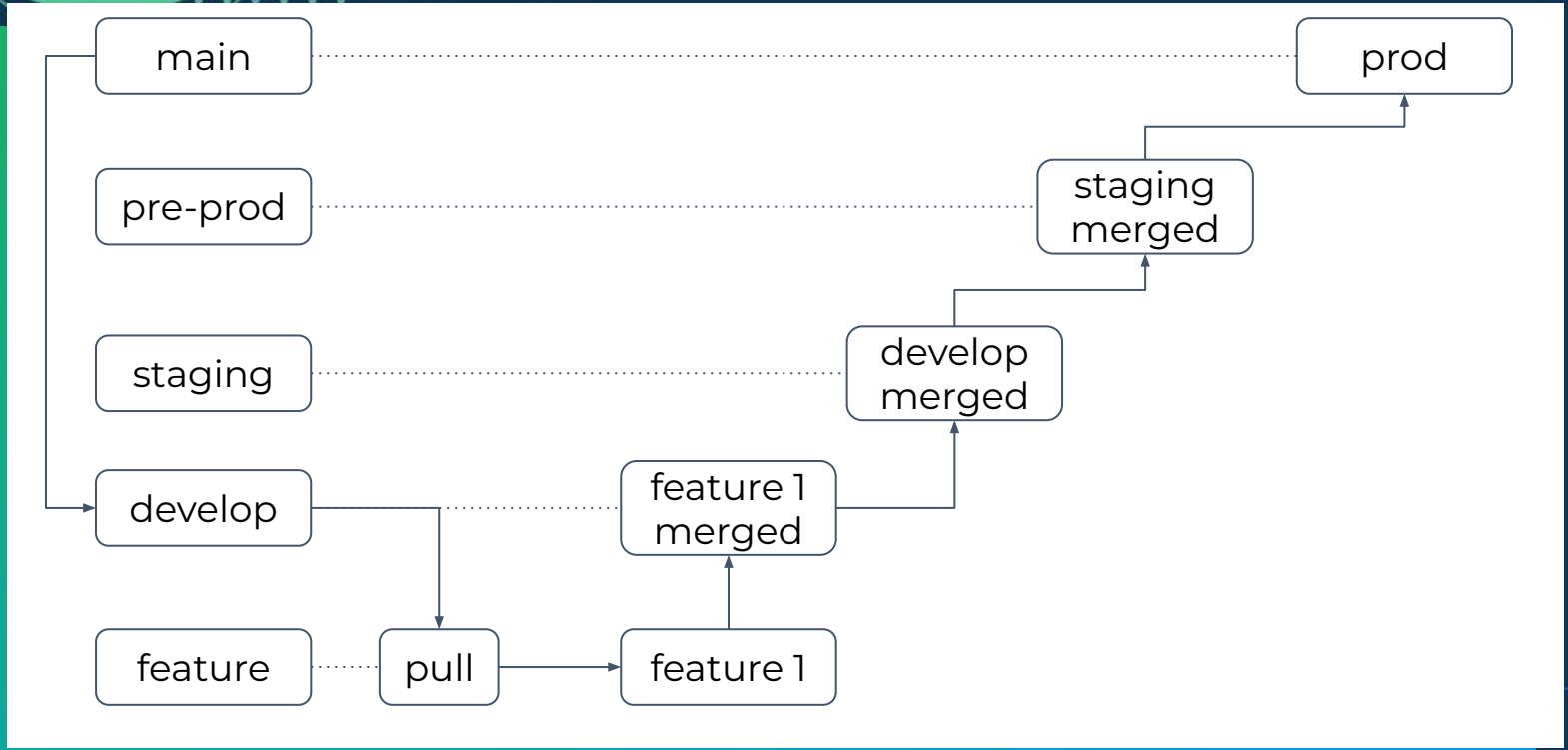



CoGrammar

# Branching Strategy

CoGrammar

# Branching Strategy



# Continuous Deployment (CD)

CoGrammar

# What is Continuous Delivery/Deployment?

- Builds upon CI by automating deployments after code passes all tests.
  - **Continuous Delivery**:
    - Deploys **approved** code changes to a **staging environment**, ready for **release**.
    - Provides a final **manual** approval step before pushing to **production**.
  - **Continuous Deployment**:
    - **Automatically** deploys **approved** code changes directly to production.
    - **Eliminates** the need for **manual intervention** and approvals.

# Deployment Strategies

- **Blue-Green Deployment:** Two identical environments, one active (**blue**) and one idle (**green**). Traffic is switched to the green environment after **successful deployment**.
- **Canary Releases: Gradually** roll out the new version to a small subset of users before a full-scale release.
- **Rolling Updates:** Update servers **incrementally**, reducing downtime and risk.

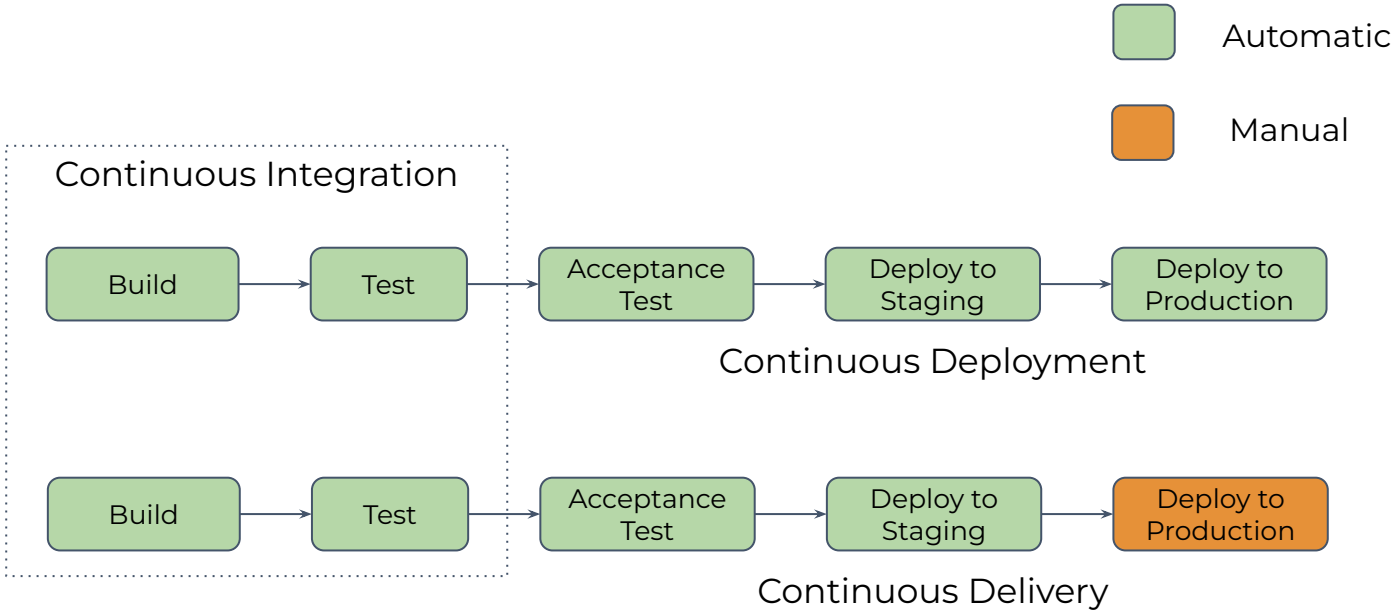
# Key Components of Continuous Deployment

- **Infrastructure as Code** (IaC)
  - Manage and provision infrastructure through code, making it scalable, repeatable, and consistent.
  - Tools: **Terraform,** AWS CloudFormation, Ansible.
- **Containerization Basics**
  - Use of **containers** to **package** and run applications in isolated environments.
  - Enables **consistency** across different stages of development and production.
  - **Tools**: Docker, Kubernetes.

# Key Components of Continuous Deployment

- Automated Deployments to Production
  - **Automate** the process of **deploying** code changes to **production environments**.
  - **Reduces** human **error** and **speeds** up the deployment cycle.
  - **Tools**: Jenkins, GitLab CI/CD, CircleCI.

# Continuous Deployment vs Continuous Delivery



# Key Components of a CI/CD Pipeline

CoGrammar

# Version Control Systems

- **Version Control Systems** (VCS): Manage code changes over time.
  - **Example**: **Git**: A popular distributed VCS.
  - **Branching**: Allows developers to work on isolated features without affecting the main codebase.
  - **Merging**: Integrates changes from branches back into the main codebase.

# Build Automation

- **Build Automation Tools**: Automate the process of **compiling** and **packaging** code.
- **Example**: Jenkins: A popular open-source build automation tool.
- Benefits:
  - Ensures **consistent** builds across environments.
  - **Reduces manual** effort and errors.
  - **Integrates** with other CI/CD tools.

# Testing Automation

- **Testing Automation**: **Automates** the **execution** of **test** suites to ensure code quality.
- Types of Tests:
  - **Unit Tests**: Verify the functionality of individual units of code (e.g., functions, classes).
  - **Integration** Tests: Test how different parts of the code work together.
  - **End-to-End Tests**: Simulate real user interactions to test the entire application flow.

# Deployment Automation

- **Deployment Automation Tools**: Automate the process of **moving code** changes to **production**.
- **Examples**:
  - **Docker**: Creates portable containers that package applications and their dependencies.
  - **Kubernetes**: Orchestrates the deployment, scaling, and management of containerized applications.
- **Benefits**:
  - **Reduces deployment** complexity and errors.
  - Enables **faster deployments** and rollbacks.
  - Improves **consistency** across environments.

# Setting Up a Basic CI/CD Pipeline in Jenkins

CoGrammar

# Step-by-Step Overview

- A **CI/CD** pipeline **automates** the software development **process**, **triggered** by a code **commit**.
- Typical stages include:
  - **Commit**: Developer **pushes** code changes to a version control system (e.g., Git).
  - **Build**: Jenkins fetches code, **compiles it**, and **runs** automated **tests**.
  - **Test**: If tests pass, the pipeline **proceeds** to **deployment**.
  - **Deploy**: Successful builds are automatically **deployed** to a **staging** or **production environment**.

# Example Scenario: Web Application

- Consider a simple **web application project**.
- Developers work on features and bug fixes in their local environments.
- Upon completion, they commit their code changes to a Git repository.

# CI/CD Pipeline in Action

- Code committed in Git triggers the **Jenkins pipeline**.
- Jenkins fetches the latest code from the Git repository.
- **Build stage**:
    - Code is compiled (e.g., Java code is compiled into bytecode).
    - Automated tests are executed (e.g., unit tests to ensure code functionality).

# Deployment (Success Scenario)

- If all tests pass in the build stage, **the pipeline proceeds to deployment**.
- Jenkins **deploys** the application to a **designated environment** (e.g., staging or production).
- Developers and testers can **validate** the **deployed application**.

# Deployment (Failure Scenario)

- If **tests fail** in the build stage, **the deployment is halted**.
- Developers are **notified** of the **failure** and can **investigate** the issue.
- Once the issue is **fixed**, the build **process starts** over

# Post-Deployment Actions (Success Scenario)

- Upon **successful deployment**, the application becomes **available** in the **target environment**.
- Developers and testers can perform thorough testing to validate functionality.
- **Monitoring** tools can be used to **track application** performance and **identify any issues** in production.

# Jenkins

CoGrammar

# Jenkins

# Key Features of Jenkins

- **Open-Source**: Freely available and customizable to specific needs.
- **Plugins**: Extensive library of plugins for various tools and integrations.
- **Build Automation**: Automates tasks like compiling code and running tests.
- **Continuous Integration/Deployment (CI/CD)**: Supports automating the CI/CD pipeline.
- **Pipeline as Code**: Defines workflows using declarative syntax for improved readability.
- **Job Scheduling**: Schedules builds and deployments at specific times or intervals.
- **Reporting and Monitoring**: Provides detailed reports and visualizations of build history and test results.

# Summary

- Code Review and Approval: Stage in the pipeline, promoting code quality and collaboration.
- Deployment Strategies: (e.g., blue/green deployment) used in CI/CD for controlled releases
- Containerisation: Using technologies like Docker to package applications and their dependencies for consistent deployment across different environments
- CI/CD Tools and Configuration: Focus on using a beginner-friendly CI/CD tool and the concept of configuration files (e.g., workflows) to define the pipeline stages.
- Benefits of CI/CD: Faster deployments, fewer errors, and improved developer productivity.

# Summary

- Continuous Integration (CI): Integrating code changes into a shared repository frequently to detect and fix integration issues early.
- Continuous Deployment/Delivery (CD): Automating the deployment process so that code changes can be released to production or a test environment seamlessly and frequently.
- Automated Builds and Testing: CI automates building code and running tests with each change, ensuring consistent quality, without manual intervention. Implementation of various test types (unit, integration, functional) that run automatically when code changes are pushed. Incorporating automated tests into the CI/CD pipeline to verify that code changes do not introduce new bugs or break existing functionality.
- Version Control Integration: Tracking code changes and triggering the CI/CD pipeline.

# Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar