Explain the Fundamentals and Benefits of CI/CD to Achieve, Build, and Deploy Automation for Cloud-Based Software Products (by Huy Luu)

Context :

You are leading a team to develop the UdaPeople product, a revolutionary concept in Human Resources which promises to help small businesses care better for their most valuable resource: their people. Before implementing CI/CD for the UdaPeople product, you need authorization from the people who write the checks. Create a proposal in document or presentation form that "sells" the concept of CI/CD to non-technical decision-makers in the UdaPeople organization.

While writing this proposal document/presentation, step out of your technical role and step into the world of revenue and costs. You will need to translate the benefits of CI/CD from technical language to the business's values. To appeal to what makes business people tick, you'll need to focus on benefits that create revenue, protect revenue, control costs, or reduce costs.

The deliverable should be "near-production-quality", but you should try to time-box your work to about 30 minutes. In other words, it should be good enough to submit to a manager in a real job. It should not be a messy or last-minute submission. You may use public domain or open source templates and graphics if you'd like. But please make sure the content is your own. Your presentation should be no longer than 5 slides. Your manager prefers to view presentations that are short and sweet!

Defining CI/CD:

Initially, let's just be familiar with the definition of CI/CD :

- Firstly, CI/CD in DevOps is a strategy for regularly delivering apps to clients. Continuous Integration and Continuous Delivery/ Continuous Deployment are the three core CI/CD concepts.
- Secondly, The CI/CD Process aims to create a pipeline that incorporates each developer's code edits into a shared chain, tests the updates against criteria in a demo context, packages the verified code into a shared repository, and automatically publishes a new version of the program.
- Thirdly, CI/CD introduces continuous monitoring and ongoing automation across the whole lifecycle of apps, from the integration and testing phases to the delivery and deployment steps. Development and operations teams collaborate in an agile manner using a DevOps or site reliability engineering (SRE) methodology, and these interconnected processes of Continuous Integration and Continuous Delivery are called a "CI/CD pipeline."
- Fourthly, while CD simplifies the release and deployment processes for the operations team, CI optimizes the coding and build processes for the development team.

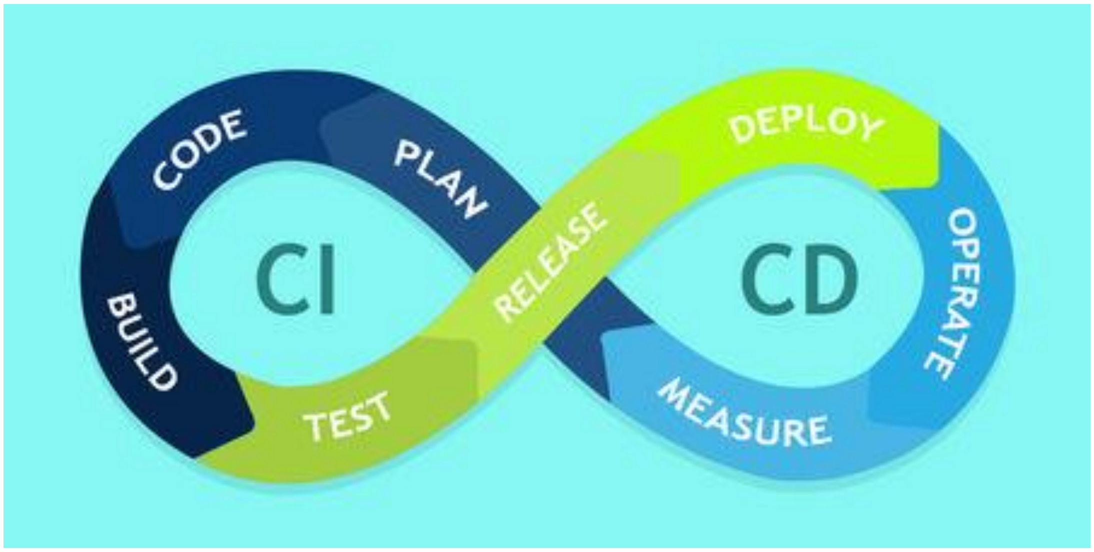
CI/CD fundamentals :

From my own perspective, I do think that the CI/CD will have approximately 8 fundamentals which supports effectively all project applied with CI/CD. Besides that, in general, I suppose that CI/CD will have some core fundamental which was explained like below:

- **A single-source repository**: I think that in source code management, all the documents and scripts needed to produce builds are kept (SCM). The repository contains all of the materials required for the creation. It includes the source code, libraries, properties files, version control, and database design. It must also have scripts for writing applications and running tests.
- **Frequent check-ins to the main branch:** Early and frequently, you should integrate code into your trunk, mainline, or master branch. As regularly, you should merge little sections of code into the branch.
- **Automated builds:** Scripts should have all the components you need to create something from a single command. Web server files, database scripts, and application software are all included. The code should be automatically packaged & compiled into a usable application by the CI processes.
- **Self-testing builds:** Testing scripts should ensure that when a test fails, the build also fails. Static pre-build testing scripts can be used to verify the integrity, caliber, and code security.
- **Frequent iterations:** There are fewer places for conflicts to hide when the repository has multiple commits. Instead of making radical changes, make incremental, frequent iterations. If there is an issue or conflict, it is simple to roll back changes by performing this.
- **Stable testing environments:** Always use a clone of the production environment to test the code & the operational production version cannot be used to test new code. Make a replica of the real world that is as accurate as you can. To find and track down bugs that evaded the first pre-build testing phase, you can use thorough testing scripts.
- **Maximum visibility:** Every developer should have access to the most recent executables and be able to observe any repository modifications. The repository's information ought to be accessible to everyone. Manage handoffs using version control so that developers know the most recent version. Everyone can monitor progress and spot potential issues when there is maximum visibility.
- **Predictable deployments anytime:** The team feels comfortable carrying out deployments whenever they like because they are so frequent and low-risk. Verification and testing procedures for CI/CD should be thorough and reliable, giving the team more confidence to deploy updates.

CI/CD - Why is it important?

- In this part, from my own view, I do suppose that there are some features that make CI/CD is vital including:



- **Enhances “Quality at Speed”**
  - A successful CI/CD process is an effective way to hasten deployment and increase the value of each release for users. Allowing constant contact between teams and employing automated processes, shortens the deployment period.

- **Free up developers' time**
    - The team has more time for rewarding initiatives because more of the deployment process is automated. Developers spend between 35 and 50 percent of their time evaluating, validating, and fixing code. Developers may increase their productivity by automating these operations.
- **Improved code quality**
    - The CI/CD pipeline offers a way for developers to exchange their codes with team members and integrate them more often to prevent conflicts in upcoming builds. It will gradually increase the code quality for all upgrades and lower the cost of addressing bugs.
- **Recover faster**
    - Fixing problems and recovering from incidents is made simpler with CI/CD. Frequent modest software upgrades made possible by Continuous Integration and Continuous Delivery in DevOps make it simpler to identify defects when they arise. The customer can swiftly resume working if developers want to roll back the modification or quickly resolve bugs.

CI/CD Benefits :

There are some amazing benefits of CI/CD including:

- **Superior Code Quality**
    - CI/CD helps to enhance the overall code quality
    - Enables developers to integrate their code into a common repository in small batches.
    - In addition, this helps to share stable builds more frequently, free of any critical bug, and thus bad code rarely makes it to production.
- **Reduced Changes & Review Time**
    - A CI environment can be integrated to communicate with a Version Control System as well. This means any change pushed to the merge triggers a CI run, automatically checking the coverage code and whether all tests are passed.
    - This highly reduces the time spent on reviewing the changes and shipping the code.
- **Accelerated Release Cycles**
    - CI/CD enables accelerated release rates. Any software development system can support recurring releases only if code is developed in a continued automated testing pipeline.
    - In addition, CI/CD serves this purpose by continuously merging codes and deploying it to production-like systems regularly to keep the code in a release-ready state. This also

enables the organization to establish a standardized delivery mechanism that runs repeated processes for each change, trusted by all.

- **Reduced Backlog**
    - By implementing CI/CD into your development process, you have the chance to decrease the number of non-critical defects in your team's backlog. Such defects are often fixed before they become a critical issue.
    - Any such defect is highlighted and is fixed before it makes it to production or impacts end-users.
- **Improved Mean Time To Resolution (MTTR)**
    - One of the main benefits of CI/CD is that it helps you bring down this number. Smaller code changes and quicker fault isolation play a significant role in keeping failures to a bare minimum.
    - It also helps recover from any setback within no time as the CI/CD pipeline ensures the fix is quickly tested in integration with the whole code before deploying to production.
- **Cost Deduction**
    - Using a CI/CD pipeline limits the potential impact and loss that a deployment problem can cause by allowing it to be deployed in non-critical business hours.
    - Also, repeated automated deployments during the development phase help developers catch the errors early before causing any significant damage. Such a pipeline implementation increases code quality, thus increasing overall ROI for the organization.