Continuous Integration

Improve software quality with Continuous Integration.
By Bart de Best

Context:
This blog is derived from my experiences as a DevOps trainer, coach and auditor with the concept of Continuous Integration. As a trainer, coach and auditor, I see many variants of Continuous Integration applications. This blog describes my experiences with this part of Continuous Everything.

Challenge:
The challenge of applying Continuous Integration is that it cuts deeply into the way software programming works. This has been a difficult part of quality management for decades because every DevOps engineer wants to program in his or her own way and derives energy from that. On the other hand, the quality of the software written by DevOps engineers varies widely. For example, readability, maintainability, extensibility, integrability, scalability, security, performance quality aspects are important factors that play a role in the time-to-market and achieving SLA standards. Continuous Integration focuses mainly on the integrability of the software of DevOps engineers and the acceleration of the time-to-market by reducing waste.

Solution:
The solution to this challenge has been found in the concept of Continuous Integration in which DevOps engineers share their programming work more times a day and check whether they work well together. This blog discusses the concept of Continuous Integration through the following steps:

1. The Definition
2. The principles
3. The method
4. The experiences

1. The definition

Continuous integration can be defined as follows:

Continuous Integration
Continuous Integration is a holistic Lean software development approach that aims to produce and put into production continuous software in an incremental and iterative manner, with waste reduction being a top priority.

The key words are Lean and continuous. The word 'Lean' refers to the reduction of waste that arises, for example, from defects or incidents that are only discovered late in the CI/CD secure Pipeline.
The word 'continue' refers to the aim of a smooth branch (making a local copy of source code) and merge (merging with the central source code). This means that a DevOps engineer can adjust or expand part of the software and that it can be easily and without problems merged (integrated) with the rest of the software, even if it is

simultaneously modified or expanded by other DevOps engineers. If the branch leads to merge conflicts, it is also referred to as merge hell. This means that it can be very complex to merge the mutations and extra functionality. Compare it to a 1-page MS Word document that has been edited by 20 reviewers and needs to be merged. Software development projects have been stopped because solving merge hell would take longer than reprogramming everything.

2. The principles

The following principles apply to Continuous Integration:

- Only small chunks of software are developed
- Locally isolated software development
- Use of a central (remote) version control repository
- High frequency of source code merges
- Short lifespan of a branch

Small chunks of software

With Continuous Integration, DevOps engineers work with small, manageable pieces of software. This makes it easier to integrate changes and quickly identify and fix problems.

Local

Every DevOps engineer has their own environment to work in. This enables them to develop new features or fixes without directly impacting fellow DevOps engineers or the stable version of the software. Software is only shared if it has been tested.

Central

A shared repository, such as GitHub, is used as the central source where all source code comes together. This provides a single point of integration and facilitates collaboration and change tracking.

Frequency

Merging changes frequently, such as several times a day, minimises or prevents integration problems. This ensures that defects and conflicts are quickly discovered and resolved.

Branch lifetime

Long branch lifespans can lead to complex merge conflicts and integration issues. Short branch lifecycles promote rapid integration and reduce the likelihood of discrepancies between different chunks of source code.

3. The Way of Working

Figure 1 shows the value stream of Continuous Integration. Steps 1 to 6 are completed cyclically.

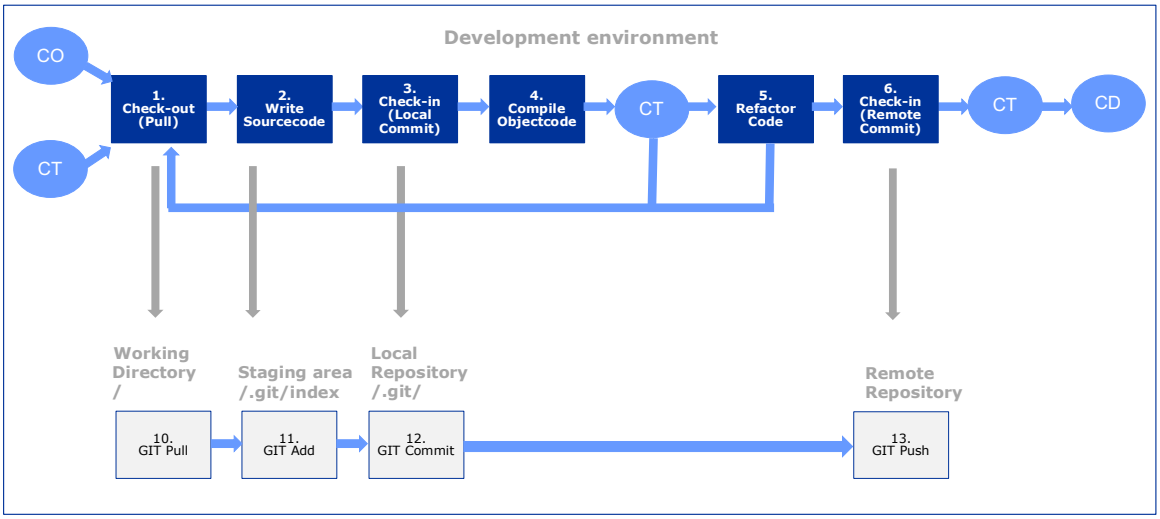


Figure 1, Continuous Integration value stream.

The start of the value stream is Continuous Design (CN), which provides various design objects as well as the requirements. Continuous Testing (CT) is the value stream that plays an essential role in Continuous Integration (see also the Blog Productivity increase through Continuous Testing). Finally, the application is deployed and released by Continuous Deployment (CD).

Step 1. Check-out
The first step is to clone the source code stored in a central repository to a local repository. Then a branch is created locally.

Step 2. Write Sourcecode
The DevOps engineer can now get started in his or her own repository. This means that the change can only be seen locally by your own DevOps engineer and not by colleagues.

Step 3. Check-in (local commit)
The local commit means that the DevOps engineer merges the change in the branch into the software in the local repository.

Step 4. Compile
The compilation step translates the source code into object code. This allows the application to be executed locally. This allows the application to be tested using Continuous Testing. Steps 1, 2, 3, 4 are repeated until the test cases are successful.
Step 5. Refactoring
The source code is cleaned up and auxiliary source code (scaffolding) is removed.

Step 6. Check-in (remote commit)
Finally, the location repository is synchronised with the central repository. This is the moment of integration. If this integration happens more than once a day, it is called Continuous Integration.

4. The experiences

Over the years I have had various experiences with Continuous Integration that I would like to share with you.

Own experiences
Working locally on source code gives a safe feeling because no work from colleagues can be lost. The tendency is that you focus so much on the realisation of the design and requirements that you forget to check whether everything works together with what others are making.

The use of pair programming where two DevOps engineers work together behind the screen to write software ensures a large reduction in errors. One DevOps engineer types software while the other indicates what needs to be coded. This does not detract from the requirement for a high degree of integration.

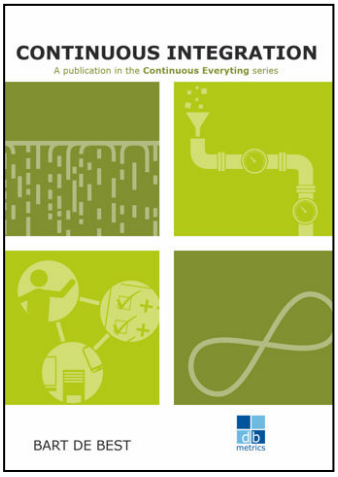
Merging into the central repository does require that system integration tests and system tests are simultaneously performed that test the entire application. If this is not the case, everyone should stop programming until the build and subsequent testing are successful. This is called recovering from a broken build. Allowing everyone to cooperate also creates a positive learning effect on which mistakes should be avoided.

Training experiences
Discussions often take place in training as to whether it is necessary to integrate often. Especially if you use microservices for which the interfaces are defined, and colleagues can use mocking to simulate each other's microservices. This is indeed a situation where fewer conflicts will arise in the event of a merge. But the operation of the application is more than software compatibility. The behavior of the different parts of the application and the way in which information is processed also requires good and high-frequency integration.

Audit experiences
During audits I notice that questions regarding branching and merging often lead to discussions. DevOps teams often appear to not have a well-defined development process. It is then more the discretion of the DevOps engineer.
One DevOps engineer has a high merge frequency, and another doesn't merge anything for days or weeks. This is of course not what is meant by Continuous Integration. The entire DevOps team working on an application must adhere to the principles and the Continuous Integration value stream to be successful.

With this application of Continuous Integration it is possible to continuously determine whether the application is functioning properly, and errors are quickly found and can also be resolved quickly. This also allows the frequency of deployment to be increased. That is why this method of software development is a good example of the application of Continuous Integration.

By Bart de Best
DutchNordic.Group



https://www.dbmetrics.nl/ce-en/continuous-integration-en/