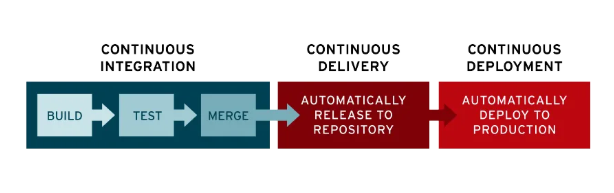
CI/CD

CI/CD, which stands for continuous integration and continuous delivery/deployment, aims to streamline and accelerate the software development lifecycle.

[Continuous integration](https://www.redhat.com/en/topics/integration) (CI) refers to the practice of [automatically](https://www.redhat.com/en/topics/automation) and frequently integrating code changes into a shared source code repository. [Continuous delivery](https://www.redhat.com/en/topics/devops/what-is-continuous-delivery) and/or deployment (CD) is a 2 part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.



Taken together, these connected practices are often referred to as a ["CI/CD pipeline"](https://www.redhat.com/en/topics/devops/what-cicd-pipeline) and are supported by development and operations teams working together in an agile way with either a [DevOps](https://www.redhat.com/en/topics/devops) or [site reliability engineering (SRE)](https://www.redhat.com/en/topics/devops/what-is-sre) approach.

Why is CI/CD important?

CI/CD helps organizations avoid bugs and code failures while maintaining a continuous cycle  
of software development and updates.As apps grow larger, features of CI/CD can help decrease complexity, increase efficiency, and streamline workflows.

Because CI/CD automates the manual human intervention traditionally needed to get new code from a commit into production, downtime is minimized and code releases happen faster. And with the ability to more quickly integrate updates and changes to code, user feedback can be incorporated more frequently and effectively, meaning positive outcomes for end users and more satisfied customers overall.

## What is continuous integration?

The "CI" in CI/CD always refers to continuous integration, an automation process for developers that facilitates more frequent merging of code changes back to a shared branch, or “trunk.” As these updates are made, automated testing steps are triggered to ensure the reliability of merged code changes. In modern application development, the goal is to have multiple developers working simultaneously on different features of the same app. However, if an organization is set up to merge all branching source code together on one day (known as “merge day”), the resulting work can be tedious, manual, and time-intensive.   
  
That’s because when a developer working in isolation makes a change to an application, there’s a chance it will conflict with different changes being simultaneously made by other developers. This problem can be further compounded if each developer has customized their own local [integrated development environment (IDE)](https://www.redhat.com/en/topics/platform-engineering/what-is-ide), rather than the team agreeing on one [cloud](https://www.redhat.com/en/topics/cloud-computing)-based IDE.

CI can be thought of as a solution to the problem of having too many branches of an app in development at once that might conflict with each other. Successful CI means that once a developer’s changes to an application are merged, those changes are validated by automatically building the application and running different levels of automated testing, typically unit and integration tests, to ensure the changes haven’t broken the app. This means testing everything from classes and function to the different modules that comprise the entire app. One of the benefits of CI is that if automated testing discovers a conflict between new and existing code, it is easier to fix those bugs quickly and often.

## What is continuous delivery?

 Continuous delivery automates the release of validated code to a repository following the automation of builds and unit and integration testing in CI. So, in order to have an effective continuous delivery process, it’s important that CI is already built into your development pipeline.  
  
In continuous delivery, every stage—from the merger of code changes to the delivery of production-ready builds—involves test automation and code release automation. At the end of that process, the operations team is able to swiftly deploy an app to production.  
  
Continuous delivery usually means a developer’s changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment by the operations team. It’s an answer to the problem of poor visibility and communication between dev and business teams. To that end, the purpose of continuous delivery is to have a codebase that is always ready for deployment to a production environment, and ensure that it takes minimal effort to deploy new code.

## What is continuous deployment?

The final stage of a mature CI/CD pipeline is continuous deployment. Continuous deployment is an extension of continuous delivery, and can refer to automating the release of a developer’s changes from the repository to production, where it is usable by customers.  
  
CD addresses the problem of overloading operations teams with manual processes that slow down app delivery. It builds on the benefits of continuous delivery by automating the next stage in the pipeline.

In practice, continuous deployment means that a developer’s change to a cloud application could go live within minutes of writing it (assuming it passes automated testing). This makes it much easier to continuously receive and incorporate user feedback. Taken together, all of these connected CI/CD practices make the deployment process less risky, whereby it’s easier to release changes to apps in small pieces, rather than all at once.   
  
However, because there is no manual gate at the stage of the pipeline before production, continuous deployment relies heavily on well-designed test automation. This means that continuous deployment can require a lot of upfront investment, since automated tests will need to be written to accommodate a variety of testing and release stages in the CI/CD pipeline.

## What is CI/CD security?

[CI/CD security](https://www.redhat.com/en/topics/security/what-is-cicd-security)is used to safeguard code pipelines with automated checks and testing to prevent vulnerabilities in software delivery. Incorporating security into your pipeline--via methods like [shift left and shift right](https://www.redhat.com/en/topics/devops/shift-left-vs-shift-right)security--helps to protect code from attack, prevent data leaks, comply with policies, and ensure quality assurance.  
  
The rapid nature of development and deployment without proper security can expose the pipeline to risks, such as:

* Exposure of sensitive data to outside sources
* Use of insecure code or third party components
* Unauthorized access to source code repositories or build tools

Identifying and mitigating vulnerabilities throughout the software development cycle assures that code changes are thoroughly tested and adhere to security standards before being deployed to production.

## What are some common CI/CD tools?

CI/CD tools can help a team automate their development, deployment, and testing. Some tools specifically handle the integration (CI) side, some [manage](https://www.redhat.com/en/topics/management) development and deployment (CD), while others specialize in continuous testing or related functions.

Tekton Pipelines is a CI/CD framework for Kubernetes platforms that provides a standard cloud-native CI/CD experience with containers.

Beyond Tekton Pipelines, other open source CI/CD tools you may wish to investigate include:

* [Jenkins](https://cloud.redhat.com/blog/deploying-jenkins-on-openshift-part-1?extIdCarryOver=true&intcmp=7013a000002wBnmAAE&sc_cid=7013a000002DgC5AAK%27%5d%5d&cicd=32h281b), designed to handle anything from a simple CI server to a complete CD hub
* [Spinnaker](https://spinnaker.io/), a CD platform built for [multicloud](https://www.redhat.com/en/topics/cloud-computing/what-is-multicloud) environments.
* [GoCD](https://www.gocd.org/), a CI/CD server with an emphasis on modeling and visualization.
* [Concourse](https://concourse-ci.org/), "an open-source continuous thing-doer."
* [Screwdriver](https://screwdriver.cd/), a build platform designed for CD.

Teams may also want to consider managed CI/CD tools, which are available from a variety of vendors. The major [public cloud providers](https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-providers) all offer CI/CD solutions, along with [GitLab](https://about.gitlab.com/), [CircleCI](https://circleci.com/), [Travis CI](https://www.travis-ci.com/), [Atlassian Bamboo](https://www.atlassian.com/software/bamboo), and many others.

