CI And CD is the practice of automating the integration of code changes from multiple developers into a single codebase. It is a software development practice where the developers commit their work frequently to the central code repository (Github or Stash). Then there are automated tools that build the newly committed code and do a code review, etc as required upon integration.

The key goals of Continuous Integration are to find and address bugs quicker, make the process of integrating code across a team of developers easier, improve software quality, and reduce the time it takes to release new feature updates. Some popular CI tools are Jenkins, TeamCity, and Bamboo.

**Continuous Integration**

There could be scenarios when developers in a team, work in isolation for an extended period and only merge their changes to the master branch once their work is completed. This not only makes the merging of code very difficult, prone to conflicts, and time-consuming but also results in bugs accumulating for a long time which are only identified in later stages of development. These factors make it harder to deliver updates to customers quickly.

With Continuous Integration, developers frequently commit to a shared common repository using a version control system such as Git. A continuous integration pipeline can automatically run builds, store the artifacts, run unit tests, and even conduct code reviews using tools like Sonar. We can configure the CI pipeline to be triggered every time there is a commit/merge in the codebase.

**Continuous Delivery**

Continuous delivery helps developers test their code in a production-similar environment, hence preventing any last-moment or post-production surprises. These tests may include UI testing, load testing, integration testing, etc. It helps developers discover and resolve bugs preemptively.

By automating the software release process, CD contributes to low-risk releases, lower costs, better software quality, improved productivity levels, and most importantly, it helps us deliver updates to customers faster and more frequently. If Continuous Delivery is implemented properly, we will always have a deployment-ready code that has passed through a standardized test process.

**CD** or **Continuous Delivery** is carried out after Continuous Integration to make sure that we can release new changes to our customers quickly in an error-free way. This includes running integration and regression tests in the staging area (similar to the production environment) so that the final release is not broken in production. It ensures automation of the release process so that we have a release-ready product at all times and we can deploy our application at any point in time.

Continuous Delivery automates the entire software release process. The final decision to deploy to a live production environment can be triggered by the developer/project lead as required. Some popular CD tools are AWS CodeDeploy, Jenkins, and GitLab.

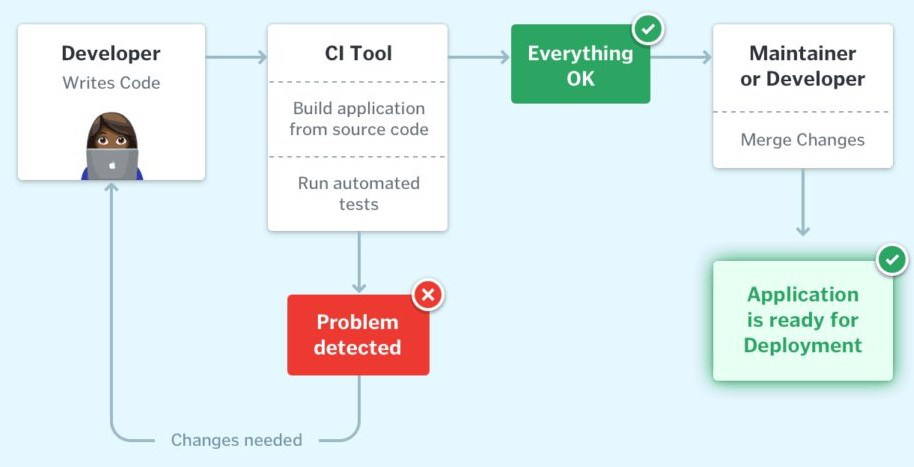
**Continuous Deployment**

the final stage of CI and CD will be continuous deployment. It’s an extension of continuous delivery, which automate the proper code to the code repository, continuous deployment will automate the related app for production purpose because there is not having any manual gate at the stage of the pipeline before production, continuous deployment relies on high automation.

in simple language, it is a change of application that goes through the cloud which is carried by the developer and it will live within a few minutes of writing pass with the automated testing.

**CI Workflow**

Below is a pictorial representation of a **CI pipeline**– the workflow from developers checking in their code to its automated build, test, and final notification of the build status.

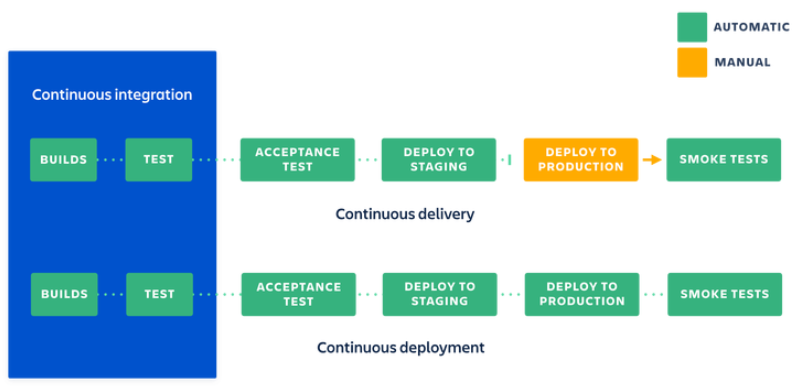


Once the developer commits their code to a version control system like Git, it triggers the CI pipeline which fetches the changes and runs automated build and unit tests. Based on the status of the step, the server then notifies the concerned developer whether the integration of the new code to the existing code base was a success or a failure.

This helps in finding and addressing the bugs much more quickly, makes the team more productive by freeing the developers from manual tasks, and helps teams deliver updates to their customers more frequently. It has been found that integrating the entire development cycle can reduce the developer’s time involved by ~25 – 30%.

**CI and CD Workflow**

The below image describes how Continuous Integration combined with Continuous Delivery helps quicken the software delivery process with lower risks and improved quality.



*CI / CD workflow*

We have seen how Continuous Integration automates the process of building, testing, and packaging the source code as soon as it is committed to the code repository by the developers. Once the CI step is completed, the code is deployed to the staging environment where it undergoes further automated testing (like Acceptance testing, Regression testing, etc.). Finally, it is deployed to the production environment for the final release of the product.

If the deployment to production is a manual step. In that case, the process is called Continuous Delivery whereas if the deployment to the production environment is automated, it is referred to as Continuous Deployment.

**Why is CI/CD Important?**

CI/ CD enables organizations to develop software quickly and efficiently. CI/CD enables an effective process for getting products and software to market faster than ever before, continuously moving code into production, and ensuring a steady flow of new features

**What are the Benefits of CI/CD?**

Automated testing enables continuous delivery that ensures software quality and safety and increases code profitability in production. CI/ CD pipelines enable a much shorter time-to-market for new product features, resulting in happier customers and reducing the burden on development.  
The significant increase in overall delivery speed enabled by CI/CD pipelines improves a company’s competitive advantage.  
Automation allows team members to focus on what they do best, resulting in the best end products. Companies with a successful CI/CD pipeline can attract outstanding talent. By moving away from traditional waterfall methods, engineers and developers are no longer engaged in repetitive activities that are often highly dependent on completing other tasks.

**What are Some Common CI/CD Tools?**

CI and CD tools can help to team with the development, deployment, and testing, some of highly recommended for the integration part and some are for the development and management of the testing and related functionality.

most of the famous tools for the CI and CD which is Jenkins. It is open source and it will help to handle all types of work and design a simple Ci server to complete the CD hub.

apart from the Jenkins, many more sources are available for the proper way of managing CI and CD which are listed below:

* [Concourse](https://concourse-ci.org/): It is an open-source tool to build the mechanics of CI and CD.
* [GoCD](https://www.gocd.org/): it’s used for the modeling and visualization.
* [Screwdriver](https://screwdriver.cd/) is a building platform for CD.
* [Spinnaker](https://spinnaker.io/): it’s a CD platform used to build a multi-cloud environment.

**CI/CD Pipeline Stages**

To define CI/CD pipelines, look at the basic steps:  **Develop → Build → Test → Deploy. ‍**‍

As we expand the pipeline we also have Monitoring → Feedback → Operations. A continuous orchestration platform can help you design such a pipeline - like Opsera’s [**Continuous Orchestration**](https://www.opsera.io/release-orchestration).

**Develop**

Here we are writing the code, meaning that we need a repository for storing and checking out code.

**Build‍**

When code is checked-in to the repository, that code is integrated into the master branch. Here is where version control is necessary. In older waterfall style workflows, integration/build would only occur after the completion of a major feature set or large volume code changes. When working in a more agile, continuous model, code is integrated into the feature times per day. Code is broken down into smaller working functions, allowing for iterative improvements and fast integration results.

**Test/Quality‍**

Once the code is built, the resulting application must be tested for errors, functional failures, and quality. These tests can and should be automated using any number of purpose-built tools. By testing at every build, feedback is received quickly and corrections can be implemented swiftly.

**Security Scan‍**

As part of our transition away from bolt-on and after-the-fact auditing, security scanning is a critical part of the continuous loop. Security scans can automatically detect vulnerabilities and insecure implementations before being released into the wild and exploited by bad actors. It is essential to include this step in the testing portion of your pipeline - as they say, “an ounce of prevention is worth a pound of cure.”

**Deploy‍**

The code is built, testing has provided the green light, and it’s time to push our changes to an environment, whether pre-production or production. With continuous deployment, regular automated push to non-production environments gives clear feedback and metrics for how the code will perform when released to a customer-facing environment. After passing the appropriate tests in pre-production, code can be automatically released to production using whatever method or model meets your requirements.

**CI/CD Tools in DevOps**

There are a large number of tools for facilitating CI/CD processes in DevOps-based teams. Some of the most popular ones include:

* **Configuration Management** - Ansible, Puppet, Chef
* **Code Management** - GitHub, GitLab, BitBucket
* **Build** - Jenkins, Bamboo, TeamCity
* **Testing** - Selenium, JUnit, SonarQube
* **Deployment** - Argo, Spinnaker, Octopus Deploy

‍

## Benefits of Kubernetes for CI/CD Pipelines

The secret to the success of a CI/CD pipeline is ensuring that application updates are performed quickly and in an automated manner. Teams typically face the following challenges when adopting CI/CD:

* **Manual steps in the release process**—Many CI/CD processes still use manual testing and deployment steps. This can cause delays and affect production schedules. Manual CI/CD processes can cause code merge conflicts and increase customer wait times for patches and updates.
* **Downtime risk**—manual infrastructure management processes can be a headache for DevOps teams because they create the risk of downtime. For example, unexpected traffic spikes that exceed capacity can cause downtime and require manual steps to restore applications.
* **Inefficient resource utilization**—applications are often deployed on servers in an inefficient way. This means organizations have to pay more for capacity. As applications are added, scaled up and down, it can be difficult to efficiently use available hardware resources. This is true whether the application is running in the cloud or on-premises.

Kubernetes can solve all three of these problems. It reduces the time and effort required to develop and deploy applications in a CI/CD pipeline. Its efficient resource management model increases hardware utilization, automates management processes, and reduces disruptions that negatively impact customers. Specifically, Kubernetes can:

* **Cluster Management** – Kubernetes takes the best practices for all previous clustering solutions and packages them in a vendor agnostic way. It comes bundled with several critical components such as schedulers and resource managers and contains plugin mechanisms for storage, networking, secrets etc. Writing distributed applications with Kubernetes is much easier compared to legacy clustering solutions as the environment it offers is standardized and without any proprietary mechanisms to closed systems
* **Orchestrate deployment and provisioning**—coordinating provisioning activities and simplifying deployment. Kubernetes handles hardware and storage resource configuration, software deployment, scalability, and health monitoring, and is fully customizable for specific needs.
* **Declarative constructs**—codifying the final state of the desired environment or application in simple, human readable code. This makes it possible to recover faster from downtime and production issues, better control scaling, and streamline disaster recovery processes.

## CI/CD in the Cloud

CI/CD in the cloud refers to the practice of using cloud-based services to perform Continuous Integration and Continuous Delivery/Deployment (CI/CD) of software. This enables developers to build, test, and deploy their software faster and more efficiently by leveraging the scalability, flexibility, and cost-effectiveness of the cloud.

### CI/CD in AWS

AWS offers a suite of services specifically designed to facilitate CI/CD practices, enabling developers to automate the software release process from code build to deployment:

* **AWS CodePipeline**, a continuous integration and continuous delivery service, orchestrates the workflow of pushing code through various stages of the release process.
* **AWS CodeBuild**compiles source code, runs tests, and produces ready-to-deploy software packages.
* **AWS CodeDeploy** automates the deployment of applications to any instance, including Amazon EC2, AWS Fargate, AWS Lambda, and on-premises servers.

### CI/CD in Microsoft Azure

Microsoft Azure facilitates CI/CD through Azure DevOps Services, offering a suite of development tools for software teams:

* With **Azure Pipelines**, teams can automatically build and test their applications to ensure code quality and consistency.
* **Azure DevOps** tooling supports multiple languages and platforms, including Windows, Linux, and macOS, and integrates with GitHub and container registries like Docker Hub.
* **Azure DevOps** also provides project planning, source code management, and reporting tools.

**Learn more in the detailed guide to**[**Azure automation**](https://spot.io/resources/azure-automation/azure-automation-runbooks-configuration-management-updates/)

## CI/CD Tools

Here is a brief review of popular CI/CD tools.

**Learn more about these and other tools in our detailed guide to**[**CI/CD tools**](https://codefresh.io/learn/ci-cd/ci-cd-tools-16-tools-delivery-pros-must-know-about/)

### Continuous Integration Tools

Popular CI tools include Codefresh, Bitbucket Pipelines, Jenkins, CircleCI, Bamboo, and GitLab CI.

#### Codefresh

Codefresh is a comprehensive GitOps continuous integration toolset designed for Kubernetes and modern applications. It is built from the ground up for flexibility and scalability around Argo Workflows and Argo Events. It takes the best of the open source toolset and provides essential enterprise features like a unified user interface, a single pane for cloud-wide management, security validated enterprise-grade runtime, end-to-end auditability, and cross-application single sign-on.

#### Bitbucket Pipelines

Bitbucket Pipelines is a CI tool that integrates directly into Bitbucket, a cloud-based source control system. It lets you manage pipelines as code and deploy your projects to production via CD tools. You can use Bitbucket pipelines to create pipeline definitions and kick off builds.

#### Jenkins

Jenkins is an open source automation tool that provides plugins to help develop, deploy, and deliver software. It is a server that lets developers distribute tasks across various machines and perform distributed tests and deployments. The Jenkins Pipeline offers several plugins to facilitate the implementation of a continuous integration (CI) pipeline.

***Learn more in the detailed guide to***[***Jenkins***](https://codefresh.io/learn/jenkins/)***.***

#### CircleCI

CircleCI is a CI tool that supports various container systems, delivery mechanisms, and version control systems like Github. CircleCI can run complex pipelines with caching, resource classes, and Docker layer caching. You can run this tool in the cloud and on-premises.

#### Bamboo

Bamboo is an automation server for continuous integration that can automatically build, test, integrate, and document source code to prepare apps for deployment. It offers a simple user interface for CI/CD and various features, including automated merging and built-in deployment support.

#### GitLab CI

GitLab CI is an open source CI tool. It lets you use the GitLab API to install and set up projects hosted on GitLab. GitLab CI can help you test and build projects and deploy your builds. It indicates areas that require improvement and lets you secure project data using confidential issues.

### Continuous Delivery and Deployment Tools

Popular CD tools include Codefresh, Argo CD, GoCD, AWS CodePipeline, Azure Pipelines, and Spinnaker.

#### Codefresh

Codefresh is a modern GitOps software delivery solution powered by Argo with support for advanced deployments like canary, blue-green, and experimental releases. It provides comprehensive dashboards that offer visibility from code to cloud while integrating with your favorite tools. A centralized dashboard gives insight into deployments at scale while providing the security and support enterprises need.

#### Argo CD

Argo CD is a Kubernetes-native CD tool optimized for [GitOps](https://codefresh.io/gitops/). It stores configuration in a Git repository and automatically applies it to Kubernetes clusters, making it easy to integrate with existing workflows. Argo CD can detect configuration drift, monitor application health, and roll back unwanted configuration changes. It also supports progressive delivery strategies like blue/green and canary deployment.

**Learn more in our detailed guide to**[**Argo CD**](https://codefresh.io/learn/argo-cd/)

#### GoCD

GoCD is an open source CD tool that helps automate the entire build-test-release process, including code check-in and all the way to deployment. It works with Git, Subversion, Mercurial, TFVC (TFS), and Perforce, and has an open plugin ecosystem. It is deployed on-premises.

#### AWS CodePipeline

AWS CodePipeline is a cloud-based CD service that helps model, visualize, and automate software release steps and continuous changes. Notable features include release process automation, establishing a consistent release process, and viewing pipeline history details.

**Learn more in our detailed guide to**[**CI/CD in AWS**](https://codefresh.io/learn/ci-cd/ci-cd-on-aws-the-basics-and-4-best-practices/)

#### Azure Pipelines

Azure Pipelines is a cloud-based service that helps automatically build, test, and ship code to multiple targets, through a combination of CI and CD mechanisms. It supports many languages, including Python, JavaScript, and Go, most application types, including Node.js and C++, and targets such as virtual machines (VMs), containers, on-premises, and cloud platforms.

**Learn more in our detailed guide to**[**CI/CD in Azure**](https://codefresh.io/learn/ci-cd/building-your-ci-cd-pipeline-in-azure/)

#### Spinnaker

Spinnaker is an open source CD platform for multi-cloud environments. It offers a pipeline management system and integrates with many cloud providers. Spinnaker provides a pipeline builder to automate releases, and lets you save and reuse existing pipelines as JSON files. It supports Kubernetes and integrates with tools like Prometheus, Datadog, and StackDriver.

#### GitHub Actions

GitHub Actions is a CI/CD tool from GitHub, the world’s most popular platform for hosting and collaborating on software projects. GitHub Actions allows developers to automate workflows directly from their GitHub repositories, making it convenient for teams already using GitHub.

GitHub Actions allows developers to create custom workflows using simple YAML syntax, and they can leverage a marketplace of community-contributed actions to extend their workflows.

**Learn more in our detailed guide to [GitHub Actions](https://codefresh.io/learn/github-actions/)**

#### Harness.io

Harness.io is a CI/CD platform that emphasizes simplicity of usage. It offers a visual interface for building and managing deployment pipelines and can automate various CI/CD processes, such as canary deployments and rollback decisions. Harness.io automatically verifies deployments in real-time to detect and mitigate issues before they impact end-users.

**Learn more in our detailed guide to**[**Harness.io**](https://codefresh.io/learn/harness-io/)

**Learn more about these and other tools in our detailed guides to**[**CI/CD tools**](https://codefresh.io/learn/ci-cd/ci-cd-tools-16-tools-delivery-pros-must-know-about/)

## CI/CD Security Risks

### Supply Chain Attacks

A supply chain attack is a cyber attack that targets weak links in an organization’s supply chain. A [supply chain](https://www.mend.io/resources/blog/software-supply-chain-security-the-basics-and-four-critical-best-practices/) is the network of all individuals, organizations, resources, activities, and technologies involved in creating and selling software products.

Modern software applications rely heavily on dependencies to provide their core functionality. The software ecosystem relies heavily on CI/CD to publish source code and binaries to public repositories. This allows attackers to bypass standard security measures and directly attack the supply chain, infecting many applications and websites simultaneously.

### Insecure System Configuration

A CI/CD environment consists of several systems from various vendors. To optimize CI/CD security, security teams must focus on the health and resilience of individual systems and the code and artifacts flowing through the pipeline.

Like any other system that stores and processes data, a CI/CD system includes a variety of security settings and configurations at the application, network, and infrastructure levels. These settings have a significant impact on the security posture of a CI/CD environment and its vulnerability to potential breaches. Attackers are on the lookout for ways to exploit potential CI/CD vulnerabilities and misconfigurations.

### Insecure Code

The demand for rapid software development and delivery has increased the use of open source third-party integrations. Some teams may bring third-party integrations into their deployments without properly scanning the source code for security vulnerabilities. Such integrations could lead to vulnerabilities in the CI/CD pipeline. Developers may not follow code security best practices, increasing the attack surface. Common code vulnerabilities include user input vulnerabilities, buffer overflows, error handling errors, and serialization issues.

### Exposure of Secrets

Automated processes are a key component of any DevOps infrastructure. CI/CD orchestration and configuration tools are increasingly being deployed into DevOps processes to automate processes and facilitate rapid deployment of software releases.

However, CI/CD tools make extensive use of secrets (like passwords and API access tokens). they access many sensitive resources, such as information from other applications and services, code repositories, and databases. The more secrets you have, the more difficult it is to securely store, transmit, and audit them.

Also, secrets are not only used for tool-to-tool authentication. In many cases, confidential information must be provided during the build and deployment process so that deployed resources can access it. This is especially important when deploying microservices using the auto-scaling capabilities of tools like Kubernetes.