CI stands for Continuous Integration. The sole purpose of CI is to give developers rapid feedback about their code quality with respect to the rest of the project. Once you push some changes to the git repository (or SVN or whatever you prefer) and you want to merge it to the main branch, you raise a PR (pull request). The PR triggers the CI pipeline which compiles and builds the code (if applicable) and performs some automated tests against it. Tests can be as simple or as complex as you want them to be (unit tests, code coverage, etc.) Basically, you’re examining whether your code “integrates” well with the application. If tests pass, the code can be merged into the main branch, usually with another team-member’s approval. Ideally, the CI pipeline should prepare an artifact and push it to an antifactory. The artifact can be as simple as a zip file containing the code files in compressed form or as complex as it needs to be (EXE file, JAR archive, Ruby Gem, Helm package, etc.) The artifactory is a special type of storage that allows versioning. For example, AWS S3, ECR, Nexus and others.  
Continuous Delivery (CD) makes sure that code not only integrates well with the application, but it is also deployable to an environment that is as close as possible to production. If the application follows the microservices architecture, the CD pipeline performs tests against the entire system (all the APIs) like end-to-end tests and User Acceptance Tests (UAT). The important thing to note here is that CD pulls the artifact from the artifactory and “delivers” it to one or more environments. The artifact remains the same in all stages of CD since it is the one where all tests were done against. When it comes to delivering the artifact to production, it must be done manually due to the obvious criticality. Someone selects which artifact version can be released to production and clicks a button that triggers the same CD process but against the live environment. CD involves working with deployment which ranges from just uploading files through FTP or rsync to using configuration management tools like Ansible or AWS SSM. If the environment is containerized, then we’d be working with kubectl or helm and kustomize. Environments can also be created and destroyed on the fly using IaC tools like Terraform, and Packer.  
In more advanced (and mature) scenarios, deployment to production is also automated, which is referred to as continuous deployment. However, it requires a whole suite of thorough tests to guarantee code and application quality in production. QA here involves functional testing, performance, and stress testing among several others. When it comes to deployment, CD should also feature the ability of rolling-back failed deployments. Some of those techniques are achieved via blue/green deployments and/or feature gates, which we can discuss in another post.

