**CI/CD Pipelines in DevOps**

The topic of this essay is the importance of CI/CD Pipelines within DevOps, how its implemented, challenges, growing trends and what CI/CD Pipelines are. CI/CD is a software development practice which combines the practices Continuous Integration and Continuous Delivery or Deployment, which is what the CI and CD stands for. It is a series of steps for developers to follow in order to deliver software updates. Continuous Integration (CI) involves regularly combining code changes into a shared repository after which automated tests are performed to ensure code quality and find potential problems. Continuous Delivery (CD) resumes the process of CI by automating the preparation of code for deployment so that code can be deployed and released at any time. Continuous Deployment deploys the code changes into a production environment after those changes pass the automated test. CI/CD pipelines are important when discussing DevOps as they are both built around automating the process of integrating code. One of the goals of DevOps is to provide faster delivery of software by integrating development and operations. With CI/CD tools and automated pipelines, developers can build, test and deploy updates for application at a much quicker rate than what would be possible using manual processes. Any code delivered can be ensured a considerably higher quality than what would be delivered manually.

A modern CI/CD pipeline requires three basic steps to implement. Step one is to set up version control, mostly done through Git platforms. This will help implement branch protection rules and code reviews. The next step is to configure a CI pipeline, allowing for automated builds and tests. The final step is to incorporate security into the continuous delivery/deployment section which enables automated vulnerability detection when being deployed. Most CI/CD pipelines include four phases. The first is the source phase and is based around the management of source code which is created or altered by developers and pushed to a version control system, with Git being the most notable system. After this, we have the build phase. This is triggered when there is new code pushed to the repository and is when source code gets modified into a product which is eligible for use. Next is the testing phase which is composed of many different types of tests for the code such as integration tests, functional tests, performance tests and security tests. After that is the deployment phase where the app is released and is accessible to end users. Deployment does not require any user input and is fully automated. Something with such a number of stages and steps would require tools to properly manage and there is a vast array of tools and practices based around CI/CD Pipelines that are available. The main tools for CI/CD include Spacelift, Azure DevOps and GitHub Actions. All three of these tools are generally considered easy to use for newcomers and they are also all highly scalable. Each of these tools bring their own benefits. Spacelift allows users to control what happens before and after every runner phase, making it highly flexible. It is able to be self-hosted within AWS. It also allows them to integrate with any third-party tool they desire, as well as popular VCS providers like GitHub, GitLab, BitBucket and Azure DevOps. Any deployed resources are easy to keep track of and find details about. Infrastructure can be quickly provisioned and configured in a single integrated workflow and enables control to manage risk and comply with standards. Overall Spacelift provides the user with collaboration, infrastructure governance and flexibility. Azure DevOps provides users with an all in one experience with the most relevant one being Azure Pipelines. Azure Pipelines is, of course, built around CI/CD automation. It provides faster deployment due to its Continuous Integration model and can deploy to many types of targets simultaneously. It can work alongside any language or platform and integrates smoothly with Azure deployments and GitHub. In general, Azure Pipelines is good for collaborative environments thanks to its ability to integrate with other services and provides fast deployment. Lastly, GitHub Actions is renowned for its many features which allow for automation, customization and execution of software development workflows within the repositories themselves. Such features include caching, shared storage between steps, community built marketplace, built-in container registry and the ability to run tests on many versions and platforms at the same time, which is referred to as matrix builds. Users have the ability to have many tasks triggered automatically in response to any events which may happen within GitHub. GitHub Actions can easily integrate with other GitHub features making it easier to manage workflows. Setting up CI/CD pipelines are simple and dedicated resources, webhooks and hardware aren’t needed to set up or maintain pipelines. Besides tools, there are a number of best practices recommended by DevOps experts in regard to CI/CD Pipelines. One is to consolidate CI/CD tools into a unified DevSecOps platform as this will reduce maintenance overhead, minimize context switching and make collaboration easier. In addition, having fewer tools which belong to the same platform makes integration a less complex process. Another is to keep everything automated continuously including testing, deployment and provisioning of infrastructure. As was mentioned several times previously, the automation of processes will help to improve accuracy due to less human error and productivity due to lower pressure on developers to due repetitive tasks. Another practice is to regularly commit any code you produce. Not only will it prevent losing any work in case of a technical error, but smaller changes are much simpler to review and test. The next practice is to always run tests early. The earlier the test is run, the quicker errors can be identified. It is always recommended to run the tests that will take the shortest time first. The final practice that I will mention here is to ensure everything is monitored which can be done with application metrics, logs and business KPIs.

Indeed, there are many challenges and limitation related to CI/CD Pipelines and their relationship with DevOps. If there is any slight error in the implementation of CI/CD, this can result in certain performance issues such as slower page loading and server responses. Performance issues can be caused by a wide variety of errors such as limited hardware resources, lack of parallel tasks due to the size of certain files in database and bad caching mechanisms. An example of a performance issue is performance bottlenecks. Bottlenecks occur when the CI/CD pipeline becomes overflowed with builds, tests and deployments. They result in a slower pipeline and longer feedback loops. While automated testing systems generally succeed in increasing developer productivity, it is not always guaranteed. Should the automated testing be flawed in any way, this could lead to developers deploying code with numerous errors. In this case, always review test cases to make sure they aren’t flawed in any way. Developers cannot rely on automation to perform every task and one task that may need human intervention is debugging. Debugging takes a lot of time and effort, especially when a large piece of code is involved and if not done properly it can result in a slower resolution at best and at worst it can produce a poor quality product. Using better tools for reporting can help make the debugging process easier as finding the problem will take less effort. One of the most pressing issues, however, in relation to CI/CD Pipelines is the security risks associated with it. Insufficient flow control mechanisms allow attackers to push code that is not supposed to be pushed without the developer’s consent. Inadequate identity and access management can allow bad actors to be granted powerful capabilities within the environment and can access confidential information among other things. Dependency chain abuse describes an attacker’s ability to abuse the way software dependencies are sourced and managed and, as a result, can risk in credentials theft or the execution of malicious software on a host. Excluding these issues, there are many limitations of a CI/CD Pipeline that don’t count as errors but are merely part of the process. For example, the dependence of CI/CD on automation might include a learning curve, especially for those that are used to a more manual workflow and requires a degree of commitment. Intricate planning and management abilities are required to utilise the full potential of CI/CD Pipelines. The integration of CI/CD Pipelines within DevOps is a joint effort, and effective communication between developers and the stakeholder of the project is vital. Communication breakdowns will result in delays and even incorrect settings. Lastly, finding the correct tool for your CI/CD Pipeline can be a challenge especially since each tool offers something different and have their own benefits. It is simply a matter of which tool will bring out the most benefits of the CI/CD Pipeline in the DevOps project.

Being such an integral part of DevOps, there is a great deal of trends in regard to CI/CD Pipelines as well as possible future directions. GitHub Actions, which was discussed in a previous paragraph, seems to be by far the most popular tool for projects regarding CI/CD Pipelines as of 2025. JetBrains surveyed several developers about the use of CI/CD tools and 62% if respondents used GitHub Actions for personal projects and 41% of respondents claimed to use it in organizations. The use of GitHub Actions for personal projects was especially higher than the next most popular tool for CI/CD, GitLab CI, which the survey reported that only 24% of respondents used. As for organization tools, the results were a lot closer but ultimately boiled down to GitHub Actions and two other major tools being on top. Those two tools are GitLab CI and Jenkins. GitLab CI is based largely around organizations and their ability to implement continuous integration, delivery and deployment. It helps organizations identify code errors early on in the development cycle and ensures any code deployed adheres to compliance requirements and standards. GitLab is fast, reliable and automated. Jenkins is an open source server where developers can build, test and deploy software but is used in a DevOps context for implementing CI/CD Pipelines. A major factor in Jenkins’ popularity is the ability to integrate with most common DevOps tools, as well as its reliability. Tools aside, the increasing popularity and development of AI has a drastic impact on how effectively CI/CD Pipelines are run and it is unlikely to slow down. While most of the process around CI/CD Pipelines is already automated, a level of user interaction is still required. Users are mainly expected to solve complex problems while AI is intended for repetitive tasks. MLOps is an example of a recent development within the AI and Machine Learning environment. The aim of MLOps is to provide automated, scalable and dependable pipelines based around machine learning. CI/CD pipelines are highly beneficial within the realm of MLOps and the machine learning lifecycle. The implementation of CI/CD pipelines in MLOps accelerates the delivery of updates, ensures code changes and updates are tested reliably before deploying, makes managing and scaling machine learning models a simpler process and allows for easier collaboration. Machine learning pipelines follow most of the same design principles as CI/CD pipelines. Continuous integration for pipelines consists of automated building and automated testing. Continuous delivery refers to automated deployment for ML models. If those who develop machine learning pipelines utilised the full capabilities of CI/CD pipelines within that environment, they could see a drastic improvement in machine learning possibilities. The previously mentioned Jenkins tool is a widely used tool for CI/CD pipelines used in machine learning environments. Its primary function is to run tests on local hardware or heavily configured cloud services.

As this report should have proved, CI/CD pipelines are an essential part of DevOps. It allows for greater automation within the development cycle, has numerous tools to help integrate it with other components of DevOps project and is growing steady with new developments and updates. While there are certainly performance and security risks that need to be considered when discussing CI/CD, it is overall a firmly established pillar of the DevOps realm with countless tools, practices and trends backing it and is not showing any signs of slowing down soon.