DevOps in Practice

A case study of the OpenJDK repo

Daniel Hartler dhartler@kth.se
Michael Ask micask@kth.se

We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct

1 Introduction

The process of software development has developed over the years from something of a niche that was mostly done in universities many years ago, to something that is an integral part to many companies and their workflows. As our society has gone through a process of digitalization, we rely more and more on the software that underpins this digitalization, and by extension we become reliant on these processes being done in a correct and safe manner.

DevOps, or development operations, is something that has developed as a set of theoretical principles for what makes for a good software development process. These are not mandatory for making software, but rather a tool for a good software development process. Exactly what constitutes these principles can differ from project to project, however, in this report we will focus on the DevOps principles that have been developed for an **agile approach** to development. That is, a process whereby the software is continually updated through feedback and new ideas.

We generally categorize these into a **development part** that consists of a planning, coding, building and release sections. And an **operations part** that consists of deployment, operate, monitoring and testing sections [1]. Although these are the abstract principles we want to manage with our DevOps, how we implement these in practice will differ.

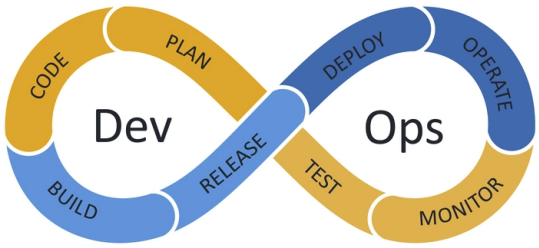

Figure 1: DevOps cycle, source: browserstack.com

In this project we will look at how these processes work in a open source project, that is, a project that is managed by a limited group of people who control the workflow, i.e the DevOps, but is continually developed by anyone who wish to contribute. Specifically, we are interested in looking at how these principles work in the open source project OpenJDK.

2 Background

2.1 DevOps methodology

The concept of an agile approach is one where the software is continuously developed through an iterative process where each piece of the DevOps cycle is a piece in the puzzle of making this process of continuous development and continuous integration work[2]. In this section we will describe each one of these pieces and explain their individual purpose towards creating a holistic development approach.

2.1.1 Planning

The development process begins with some planning phase, this is where there should be some notion of integrating the ideas and feedback that arise from the testing phase of the program. There are many tools available for analyzing the data gathered from testing and this will differ from project to project, but some form of data analysis tool could be useful for the planning phase in order to come up with the tasks that we want to address.

2.1.2 Coding

The coding phase is where the tasks from the planning phase gets solved and integrated into the code base. The actual code editor is generally not that important in terms of DevOps, but rather we would want some uniform environment where the code gets integrated into one code base. One example of such an environment could be Git.

2.1.3 Build

The building phase is usually where we integrate the new code updates into the previous code base in order to get a new version of the code ready for release. The tools used for this will differ based on the programming languages used. An example of a tool that automates the build process for java is Maven.

2.1.4 Testing

The testing phase is meant to make sure that the newly built code version works as intended. Although it is impossible to make sure that the code base is completely bug free, we would preferably expect a pretty extensive test suite to automatically run on the code in order to find any bugs[3]. There are many tools for building unit test for various different languages, but they generally tend to want to assert expect behaviour and test it.

2.1.5 Release and Deploy

The release and deploy step is sometimes one integrated step and sometimes two different steps depending on the program. We would expect this to have

some system for managing the release and deployment of the new version into the production version of the code. One crucial step of this stage is to make sure that we have an easy process for the users of the software to get integrated into the new version.

2.1.6 Operate

Once the new version is up and running we would want to have some sort of system for making sure the everything runs smoothly. What is required here is usually some sort of server to be the link between the system and the users, unless the software is purely ran on the users computer. In which case we would only want to host the software for the users to download. Regardless, this will differ greatly from project to project.

2.1.7 Monitor

In order to close the DevOps cycle we would ultimately like some monitoring of the system to make sure that we can find any flaws or get feedback on the preformance of the system. This is a crucial step to an effective DevOps cycle as it will be what allows the planning phase to develop the software further. There are many tools that help with monitoring systems and getting a good overview, one such tool to track application performance would be DataDog.

2.2 DevSecOps

The bigger a project grows, the more important the term security becomes. With the growth of the DevOps workflow in software development, security has become all the more integral [4]. In some contexts, it is a given that security is included when using the term DevOps, but it is often specified separately with the term DevSecOps. The core of the concept is to integrate security as part of the overall workflow. The motivation behind this movement is that when security is kept isolated, related issues that crop up during DevOps are handled too late, undermining the agile approach that the workflow encompasses [5].

A core principle of DevSecOps is the so called *shift left* mantra. The idea is to take security and shift it to the left (to the beginning) of the workflow [6]. An important way to achieve this is through educating developers and building a culture around security, in hopes to make everyone aware of security and its concers throughout the whole DevOps pipeline. Due to the nature of DevOps, security automation is another way to integrate it into the agile workflow. This may include techniques such as automated security testing with static analysis and automatic patching of known vulnerabilities [5].

2.3 OpenJDK

OpenJDK is a collaborative effort of developing an open source JDK (Java Development Kit) implementation of the Java Standard Edition platform (Java SE). Java SE is the platform for developing, running and maintaining Java applications and adheres to the Java specification JSR, *Java Specification Requests*. What began as an internal project in 2006 at Sun Microsystem inc quickly expanded into a open project involving both private individual actors in the Java community as well as large companies such as Red Hat, IBM, Apple, and Oracle [7].

Today the OpenJDK project is hosted on a GitHub repository where it currently has more than 70k commits on its master branch, which itself consists of over 67k files [8]. It integrates several pull requests per day, sometimes even several per hour.

3 Analysis of the OpenJDK repository

When analyzing the OpenJDK project we have primarily analyzed their GitHub page and the information available about the project that can be found there. We will draw some conclusions about their workflow based on previous activity and their stated policies. We will also look at GitHub's insights feature in order to get a more holistic view of how the project is managed and operated from a DevOps perspective.

3.1 Monitoring and Planning

Issues that arise in the OpenJDK project are reported to the OpenJDK team that manages these through an internal website they call the *JDK Bug System*. This is somewhat similar to the issues functionality in GitHub. However, due to scale of the project, it seems that they have wanted to build their own version of this that is more extensive. This also allows for more flexibility and discussion of any new ideas or issues that arise as anyone can discuss the issues publicly, and moderation of what should and should not be done can be managed by the team of administrators. This type of forum-like discussion around development and planning also seems quite natural for an open source project like OpenJDK as the typical hierarchy that you would normally find in a corporation working on a software project does not exist for open source projects.

3.2 Continuous Integration

The continuous integration, i.e the coding, building, and testing phase, is managed through GitHub. The contributors to the project can create a local clone of the repository on their own machine and update the code as they wish. When that is done they can wish to have their updates pushed through to GitHub where there is a comprehensive set of tests that gets ran on each update. These tests are ran in multiple different environments to make sure that the code base still works as intended for each one.

GitHub Provides an easy overview of the test results for each code update as you can monitor the run-time and results of each update directly through the GitHub website. GitHub also provides an easy way to build the code as you can simply have your code integrated into the existing code base through a code merge.

3.3 Continuous deployment

The OpenJDK does continually update their JDK through multiple steps. Once development of a new version gets closer to being released they do a *rampdown of phase one* followed by a *rampdown of phase two*. These are meant to narrow the scope of what updates should be included into the new version. They then proceed to release an *initial release candidate* followed by a *final release*

candidate. These are meant to be tested by a broader userbase before finally releasing a *generally available version*.

They do not follow the typical DevOps principle of continually deploying new versions as they are integrated into the code base, but rather an iterative approach where they do multiple deployments before settling on an official version that is robust enough to be fully deployed.

3.4 DevSecOps

Given the nature of open source, security can often be harder to control due to varying levels of experience among contributors. One way the OpenJDK project manages this is by only allowing a few select members, called committers, to actually accept pull requests and commit to the master branch. This is detailed on their *contributing to the JDK* page [9] [10]. For a non committer to contribute, they must first find a sponsoring committer that is willing to commit their code. This sponsor will go through and make sure that the code is up to the OpenJDK standard, improving security before the code is even committed into the branch. In their own words: "The JDK is used daily by millions of people and thousands of businesses, often in mission-critical applications, and so we can't afford to accept anything less than the very best". This mindset indicates a culture that focuses on security and stability.

What is missing however, is the automation of security testing, such as a static analysis security test. As mentioned earlier in (3.2), some forms of automated testing is done, but these tests are a form of integration and unit testing, not security testing. Patching of known vulnerabilities is also not automated. Publicized vulnerabilities are manually patched and integrated into the master branch by a trusted sub group of the committers [11], with back porting performed in exceptional cases.

4 Reflection

In general, it is clear that some of the DevOps principles do apply in how OpenJDK operate as an open source software project. However, there are some things that do differ due to the requirements put on the project.

The continuous integration part is very well worked out with clear guidelines for how the process is structured, which we feel is quite natural for a project that is open source and of this scale. It seems to us that the actual integration of the code is a somewhat more rigorous process than through traditional software project as there are risks associated with letting anyone directly contribute 3.4. A majority of the testing is automated through the aforementioned test suite 3.2 and the actual integration code integration part follows a very typical DevOps workflow.

The deployment of their versions do differ quite a bit from a traditional DevOps deployment workflow. This does slow down the timeline quite a bit for getting new updates deployed, but we do think that this is a sensible approach as there are many projects that rely on their JDK, and it is crucial that they only release stable versions as not doing so can have large ramifications in other projects.

One issue found with the workflow was in regards to the security, mainly the automation of it. Since the project features no automated security tests, such as a static analysis, every vulnerability has to be found and patched manually, hurting agileness of contributing. One reason for this may be the scale and complexity of the project, making it difficult to assess deep security issues.

In conclusion, the typical DevOps principles are well suited for a project like OpenJDK and are largely applied in practice. However, there are some clear adjustments done to parts of the process to facilitate certain requirements of an open source project.

References

[1] Sourojit Das. *DevOps Lifecycle : Different Phases in DevOps*. 2023. URL: https://www.browserstack.com/guide/devops-lifecycle.

[2] Sikender Mohsienuddin Mohammad. "DevOps automation and Agile methodology". In: *International Journal of Creative Research Thoughts (IJCRT), ISSN* (2017), pp. 2320–2882.

[3] Mohd Ehmer Khan. "Different forms of software testing techniques for finding errors". In: *International Journal of Computer Science Issues (IJCSI)* 7.3 (2010), p. 24.

[4] Håvard Myrbakken and Ricardo Colomo-Palacios. "DevSecOps: A Multivocal Literature Review". In: *Software Process Improvement and Capability Determination*. Ed. by Antonia Mas et al. Cham: Springer International Publishing, 2017, pp. 17–29. ISBN: 978-3-319-67383-7.

[5] Red Hat. *What is DevSecOps?* 2023. URL: https://www.redhat.com/en/topics/devops/what-is-devsecops (visited on 05/05/2023).

[6] IBM. *What is DevSecOps?* URL: https://www.ibm.com/topics/devsecops (visited on 05/05/2023).

[7] OpenJDK. *OpenJDK main website*. URL: https://openjdk.org/ (visited on 05/05/2023).

[8] OpenJDK. *OpenJDK GitHub Repository*. URL: https://github.com/openjdk/jdk (visited on 05/05/2023).

[9] OpenJDK. *Contributing to the JDK*. URL: https://github.com/openjdk/jdk/blob/master/CONTRIBUTING.md (visited on 05/05/2023).

[10] OpenJDK. *How to contribute*. URL: https://openjdk.org/contribute/ (visited on 05/05/2023).

[11] OpenJDK. *OpenJDK Vulnerability Group*. URL: https://openjdk.org/groups/vulnerability/ (visited on 05/05/2023).