Implementations of DevSecOps Tools in WIMMA Lab

Case: WIMMA Lab 2019

Hannu Oksman

Bachelor's Thesis
May 2021
Information and Communication Technologies
Degree Programme in Information and Communication Technology

jamk.fi

Description

Author(s) Oksman, Hannu	Type of publication Bachelor's thesis	Date May 2021
		Language of publication English
	Number of pages 45	Permission for web publication: x

Title of publication
Implementations of DevSecOps Tools in WIMMA Lab
Case: WIMMA Lab 2019

Degree Programme
Information and Communication Technology

Supervisor(s)
Manninen, Pasi; Kotikoski, Sampo

Assigned by
JAMK University of Applied Sciences – WIMMA Lab

Abstract

WIMMA Lab 2019 was a highly autonomous software development project environment and simulated as closely as possible how work is done at the software development companies. Students who applied were interviewed and were chosen to have four multidisciplinary and motivated virtual companies. The virtual companies had varying internal and external tasks to create content, develop software, research, and create proof on concepts.

Each virtual company had multiple continuous integration, delivery, and deployment pipelines with various tools. A pipeline represents a process from committing code to a source code repository to deploying the built and tested the application to a production environment. The software development lifecycle was divided into ten phases, and the pipelines were contrasted to these phases. The goal was to identify how each phase was implemented, or not, in the virtual companies' pipelines.

The virtual companies' GitLab projects were analyzed to gather a picture of the implementations. GitLab projects consisted of GitLab CI, GitLab Container Registry, GitLab Pages, GitLab Runners, and various settings. The projects also utilized multiple third-party platforms and tools such as Docker, Google Cloud Platform, and Robot Framework.

Each of the software development lifecycle phases was covered by the projects, though no single pipeline incorporated a tool(s) for every phase. Some phases were implemented very lightly, such as security, monitor, and protect phases.

WIMMA Lab 2019 can be considered a very advanced software development environment regarding automation, modern tooling usage, and providing in-demand skills to work life.

Keywords/tags (subjects)
Continuous Integration/Deployment, DevOps, DevSecOps, Software Development Lifecycle

Miscellaneous (Confidential information) jamk.fi

Kuvailulehti

Tekijä(t) Oksman, Hannu	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Toukokuu 2021
		Julkaisun kieli Englanti
	Sivumäärä 45	Verkkojulkaisulupa myönnetty: x
Työn nimi **WIMMA Labin DevSecOps-työkalutoteutukset** Tapaus: WIMMA Lab 2019		
Tutkinto-ohjelma Tieto- ja viestintätekniikka		
Työn ohjaaja(t) Manninen, Pasi; Kotikoski, Sampo		
Toimeksiantaja Jyväskylän ammattikorkeakoulu JAMK – WIMMA Lab		

Tiivistelmä

WIMMA Lab 2019 oli itseohjautuvuutta korostava projektiympäristö, joka simuloi ohjelmistokehitysyrityksiä mahdollisimman paljon. Hakeutuneet opiskelijat haastateltiin ja heistä valikoitiin sopiva joukko siten, että voitiin muodostaa neljä monialaista ja motivoitunutta virtuaaliyritystä. Virtuaaliyritykset saivat sisäisiltä ja ulkoisilta asiakkailta toimeksiantoina luoda sisältöä, kehittää ohjelmistoja, tutkia ja tehdä toteutuskelpoisuusselvityksiä.

Jokaisella virtuaaliyrityksellä oli useita jatkuvan integroinnin, -toimituksen ja -julkaisun putkia. Putkissa on erilaisia työkaluja, joilla prosessoidaan ohjelmiston julkaisua. Putki alkaa koodin lisäämisestä lähdekoodijärjestelmään ja päättyy siitä koostetun sekä testatun ohjelman julkaisuun tuotantoympäristöön. Ohjelmistokehityksen elinkaari jaettiin kymmeneen vaiheeseen ja putket kuvastavat näitä vaiheita. Tavoitteena oli tunnistaa miten eri vaiheet oli toteutettu virtuaaliyritysten putkissa.

Virtuaaliyritysten GitLab-projekteja analysoitiin, joista saatiin kuvaus toteutuksista. Näiden GitLab-projektit koostuivat GitLab CI, GitLab Container Registry, GitLab Pages ja GitLab Runners -palveluista sekä muista asetuksista. Projektit hyödynsivät myös useita kolmannen osapuolen alustoja ja työkaluja kuten Docker, Google Cloud Platform ja Robot Framework.

Ohjelmistokehityksen elinkaaren jokainen vaihe oli mukana projektien putkissa, joskin yhdessäkään yksittäisessä putkessa ei ollut työkalutoteutusta jokaisessa vaiheissa. Yleisesti ottaen jotkut vaiheet oli toteutettu hyvin kevyesti, kuten turvallisuus-, monitorointi- ja suojausvaiheet.

WIMMA Lab 2019 oli korkeasti kehittynyt ohjelmistokehitysympäristö, joka antoi opiskelijoille työelämässä kysyttyjä ja tarvittavia taitoja.

Avainsanat (asiasanat)
Jatkuva integrointi ja julkaisu, DevOps, DevSecOps, ohjelmistokehitys

Muut tiedot (salassa pidettävät liitteet)

Contents

Figures

1 Introduction

1.1 Background

DevSecOps is an abbreviation of Development, Security, and Operations. It can be contrasted to DevOps, which comes from Development and Operations. Both are software development life cycle processes, which include human resources as well as technical solutions. In the DevOps, security concerns are not emphasized as much as in the DevSecOps. For example, security requirements are studied, demanded, and prioritized at the beginning of a project, which is done by using the DevSecOps methodology. (Mar 2018, 16)

The DevOps (and the DevSecOps) is well suited for long-term product or service development, such as web applications, because the software development life cycle is looping. The process is continuous, and the goal is to make quick releases to cater to stakeholders' needs. For short-term project development, an agile process is more suitable as it aims to plan, design, and launch the solution. The agile process is more dynamic and available for quick changes and iterating the design phase towards the final output (See Figure 1.). In contrast, the agile process does not explicitly emphasize the operations experts. The DevOps software development life cycle is often abstracted as a loop emphasizing development and operations life cycle phases. (DevOps vs Agile – Understand The Difference, 2020)

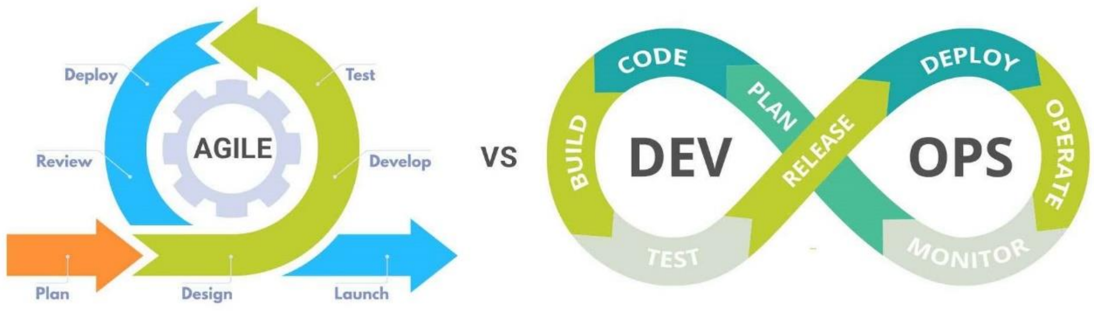


Figure 1. DevOps software development life cycle, modified (adapted from DevOps vs Agile – Understand The Difference, 2020)

The DevOps identifies software development life cycle phases and an integral connection between development and operations. However, the DevSecOps promote the importance of security at each phase. Security is of paramount priority as various breaches and incidents are regularly in the news. ETEK proposes DevSecOps Framework (See Figure 2.) with examples of what kind of security methods, tools or processes can be utilized in each phase. This analysis adds a new layer, the security, to the DevOps and further reduces distinctions between development and operations by replacing them with "Culture", "Quality", "Infrastructure", and "Standardization." The DevSecOps Framework also emphasis a motion for the software development life cycle. Even though it is a loop, it still advances an idea towards a full-fledged product and continues to enhance the product after the launch.

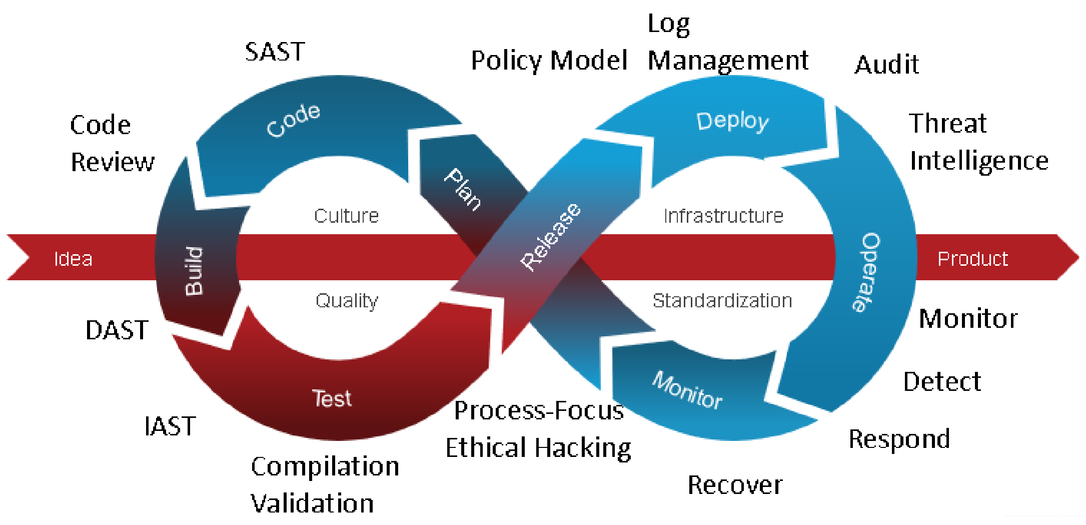


Figure 2. DevSecOps Framework (Development-Operation Cycle Security (DevSecOps). N.d.)

Furthermore, the DevSecOps approach to development may reduce expenses because fixing issues later is more expensive than fixing them earlier. Still, integrating security experts into a team requires teams to learn how to manage all security points of view at every development phase. That might cause some expenses, but overall, it benefits the team and the project. (Carter 2017, 95)

In the past, security experts tended to slow down development processes because they did not share their expertise and information. It was regarded as dangerous to let developers or operations experts have more information about vulnerabilities and attacks. The DevSecOps mentality entails that the security experts provide access to information and an added value of doing it securely. Thus cooperation, sharing knowledge, and educating a team are considered more valuable than keeping development, security, and operations separate entities. (ibid., 94-95)

DevSecOps Manifesto (Manifesto N.d.) reinforces concepts introduced in the background section. The DevSecOps is all about the mindset and attitude of cybersecurity experts. The focus of the Manifesto is how the security experts contribute to project teams' goals. The security experts should collaborate with their project team and take a proactive stance on development and operations. They should share their knowledge with the development team and solve problems during development with their expertise instead of merely discovering them. During development and operations phases, they should not rely only on automatic tools but should actively use their skills to attack the product. This way, they can discover and fix security vulnerabilities before malicious actors could take advantage. The security experts should support the operations by actively monitoring the product's state to detect anomalies, indicating security penetration.

1.2 Three Examples of How to Implement the DevSecOps Mentality

The human aspect is essential in the DevSecOps software development process. It means that everyone in a development team, not just designated security experts, should take responsibility for the software's security. Developers may use automatic tools to identify vulnerabilities while the security experts support the development team at essential development phases. For example, security experts may use deception (a human component) to lure malicious attackers to interesting-looking files. When the attackers access the files, an alarm is triggered, and operations experts detect the attack. The rationale is that eventually, attackers may get into a system, in which case it is paramount to detect and react to the attack. (Mar 2018, 17)

Threat modeling is one concrete advantage that security experts can bring to a development team. The security can model different kinds of threats into automatic or static tests. Then, developers have to ensure that the tests pass (they are solved and do not cause unwanted issues) before a product is released. (Carter 2017, 95)

About utilizing human resources, one way to enhance security is to implement continuous authentication. Instead of just relying on a user's knowledge, that is, a username and a password, the user herself may be factored in. This approach may have privacy issues, but it would be beneficial to gather biometric data such as how the user typically operates her input devices, such as a keyboard and a mouse. The user could be authenticated with these kinds of behavioral patterns. (Mar 2018, 17)

1.3 The Client of the Thesis

The thesis client is JAMK University of Applied Sciences and WIMMA Lab. The contact person is senior lecturer Mr. Marko Rintamäki, the product owner of WIMMA Lab. The theme of the WIMMA Lab 2019 was DevSecOps: researching, experimenting, evaluating, and building various tools that could be used in continuous integration and delivery/deployment (CI/CD) pipelines. Each team in the WIMMA Lab had separate pipelines (series of steps from committing code to a repository to delivering an application to a production).

1.4 Continuous Integration, Delivery, and Deployment

Continuous integration means that when code is committed to a repository, an application is built and tested to detect any issues. The continuous delivery follows the CI, and the application is reviewed and staged for production. An operations expert evaluates the staged solution and deploys it to the production environment if the solution is sufficient, passing all the tests and analysis. The continuous deployment is the same as the continuous delivery, except the application is automatically pushed to production. (GitLab Continuous Integration (CI) & Continuous Delivery (CD) N.d.; Pittet, S. N.d.)

Software development's evolution and how each term correlates to the other is illustrated in the following figure (See Figure 3.). DevOps contains Operate and

Monitor phases and thus is more comprehensive than continuous deployment. Kuuva (2020, 5) proposes that testing is not part of continuous integration and identifies it as part of software development. Figures 1. and 2. support this interpretation as testing is firmly within the domain of development. However, Pittet (N.d.) argues that continuous integration is about testing, merging, and verifying pull requests and branch merges to avoid integration challenges such as broken builds and merge conflicts. In Figure 3., a dotted arrow covering the testing is added to the continuous integration to accommodate Pittet's point of view.

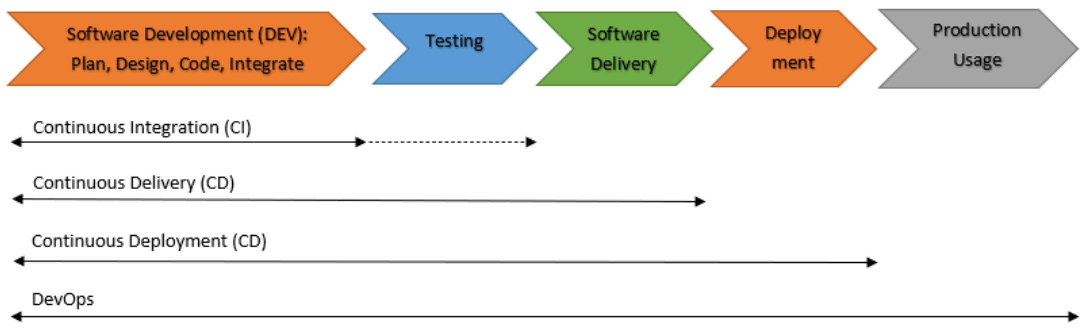

Software Development (DEV): Plan, Design, Code, Integrate
Testing
Software Delivery
Deploy ment
Production Usage
Continuous Integration (CI)
Continuous Delivery (CD)
Continuous Deployment (CD)
DevOps

Figure 3. Evolution of continuous software development (adapted and modified from Kuuva, 2020)

1.5 The Assignment

The research and development focus of the WIMMA Lab 2019 was DevSecOps and building continuous integration and delivery (CI/CD) pipelines. Each of the WIMMA Lab's virtual companies had its pipelines. In particular, Mysticons virtual company was tasked to create a CI/CD template. The template could be used in the future WIMMA Labs and software development project courses during semesters.

Furthermore, the Mysticons were assigned to write WIMMA Lab Green Book. The Green Book describes virtual companies' technical processes and development environments, such as GitLab. It also includes tutorials on how to use the CI/CD template.

The assignment was to observe, document, study, and analyze how the virtual companies implemented the DevSecOps practices and tools in their projects.

1.6 The Goal

WIMMA Lab is continuously improved to retain and enhance its status as an advanced project learning environment. The primary goal is to contribute by analyzing how various DevSecOps tools were used in the WIMMA Lab 2019. Thus providing information about which DevSecOps tools are applicable in the following WIMMA Lab instances and how software development lifecycle phases are implemented. Evaluating the status quo allows lecturers and stakeholders to modify the WIMMA Lab concept. This insight can shift focus to underutilized phases and seek new tools and methods to create more mature pipelines. This information may also be utilized to market the WIMMA Lab to students and stakeholders from local information and communication technology companies.

1.7 WIMMA Lab

WIMMA Lab software development project environment is arranged annually since 2011. Over the years, almost every aspect has varied: name, funding, the scope of projects, number of projects, duration of projects, number of students involved, how the students organized, et cetera. The current name, WIMMA Lab, has been used since 2017. (Mäkäläinen 2018, 6-9)

Students have to apply to the WIMMA Lab, and they are interviewed. They are assessed based on motivation and attitude, teamwork and communication skills, and expected growth as a professional (increased knowledge, experience, networks). To some degree, the students' technical aptitude matters because the WIMMA Lab's virtual companies require different skill sets, such as cybersecurity or front-end development.

The WIMMA Lab has been organized around virtual companies since 2016 (Mäkäläinen 2018, 9). The WIMMA Lab 2019 had four virtual companies: Iotitude (eight-person software development team), Mysticons (nine-person DevSecOps, robotics, and proof of concept team), Overflow (eight-person software development team), and Pengwin Media (five-person design, graphics, and communications team).

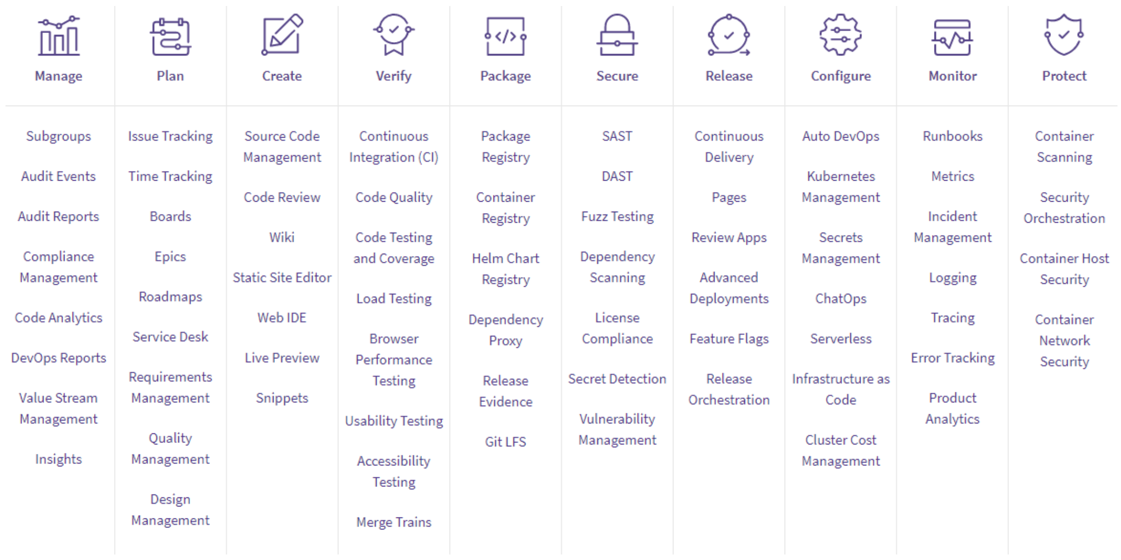
The WIMMA Lab has a head coach who is primary responsibilities are manifold. For example, the head coach arranges visits and visitors to provide networking opportunities for students, facilitates workshops, communicates with clients, and supports the virtual companies in social, technical, and management challenges. From 2017-2019 the head coach has been a student.

The clients of the WIMMA Lab provide assignments for virtual companies. For example, in the WIMMA Lab 2019, one of the clients was Ehasa ry (https://www.ehasa.org/), which organize outdoor military roleplay events. Ehasa ry wanted the WIMMA Lab to develop situational awareness and tactical command systems for their events. The clients are essential for the WIMMA Lab because their contribution guarantees the assignments are not trivial or mundane. They provide real challenges, support, and sparring to virtual companies.

2 DevSecOps Development Phases

2.1 The Ten Phases of DevSecOps Software Development Life Cycle

Various products, technologies, and tools are available for DevSecOps teams. The technologies can be categorized into ten phases by their purpose or role in the software development lifecycle. Figure 4. illustrates the phases with examples of types of tools and technologies associated with the phases. The ten phases are one version of the software development life cycle (SDLC) model, and each phase is further described in appendices 5. to 14. with a name, primary responsibility group, description, and an example tool for each phase. The phases are analyzed in this chapter. (DevOps Tools Landscape N.d.; The entire DevOps lifecycle in one application N.d.) Developing software is a very demanding and complex endeavor that requires various skill sets to be successful. Thus every phase benefits from development, security, and operations experts' skills and knowledge. While different experts are in primary charge of the phases, it is valuable to take advantage of the team's collective knowledge, and assertive involvement should be encouraged. (ibid.)



Manage	Plan	Create	Verify	Package	Secure	Release	Configure	Monitor	Protect
Subgroups	Issue Tracking	Source Code Management	Continuous Integration (CI)	Package Registry	SAST	Continuous Delivery	Auto DevOps	Runbooks	Container Scanning
Audit Events	Time Tracking	Code Review	Code Quality	Container Registry	DAST	Pages	Kubernetes Management	Metrics	Security Orchestration
Audit Reports	Boards	Wiki	Code Testing and Coverage	Helm Chart Registry	Fuzz Testing	Review Apps	Secrets Management	Incident Management	Container Host Security
Compliance Management	Epics	Static Site Editor	Load Testing	Dependency Proxy	Dependency Scanning	Advanced Deployments	ChatOps	Logging	Container Network Security
Code Analytics	Roadmaps	Web IDE	Browser Performance Testing	Release Evidence	License Compliance	Feature Flags	Serverless	Tracing	
DevOps Reports	Service Desk	Live Preview	Usability Testing	Git LFS	Secret Detection	Release Orchestration	Infrastructure as Code	Error Tracking	
Value Stream Management	Requirements Management	Snippets	Accessibility Testing		Vulnerability Management		Cluster Cost Management	Product Analytics	
Insights	Quality Management		Merge Trains						
	Design Management								

Figure 4. Ten phases of DevSecOps software development lifecycle. Each step has examples of operations it contains. Adapted from GitLab. (GitLab. N.d.)

2.2 Manage

The management phase is continuous for the entire software development life cycle. The central management categories are portfolio management, project management, time management, financial management, resource management, and demand management. (Best Practices in Project and Portfolio Management. N.d.) They encompass managing projects' authorizations, compliances, contracts, environments, expenditures, policies, specifications, stakeholders, teams, delivery timeline. These lists are not exhaustive. For example, managing people entails granting authorizations to different environments, such as cloud services, to team members, while managing scope entails negotiating with stakeholders on contracts, specifications, and delivery timelines.

A project manager is responsible for managing a project to success, which requires implementing changes, extract actionable information, inspire people, and have visibility to the project's state. For example, in Kanban methodology, the project manager may monitor the throughput of tasks. (7 Lean Metrics to Improve Flow. N.d.) If tasks are not proceeding at desirable velocity, the project manager can identify bottlenecks and suggest changes to the team to improve the throughput. Implementing the changes requires astute communication skills as the project manager has to motivate the team and ask advice on how to gain the best throughput.

2.3 Plan

Planning is essential for a software project to be successful. A specification is negotiated with a client (external or internal) and based on it, and a plan is prepared. In agile development, the planning phase also continues through the project. This phase requires software, operations, and security experts' collaboration to make informed decisions, avoid potential issues, and mitigate risks. The plan is a road map to the final product or service. For example, in Scrum methodology, the project's timeline is split into sprints, milestones, sprint plannings, retrospectives, and the project's scope to epics, user stories, and tasks.

The team decides what user stories or tasks are selected to work on in the next sprint and estimates how long they take in the sprint planning. The sprint is usually several weeks time period for developing the decided workload. After the sprint, a retrospective might be held, and the team discusses the past sprint to improve the process and review what was accomplished. The milestones demarcate actual events during the project, such as releasing new features or deployments to new environments. The epics are essential features of the product or service. A virtual queuing service might have the ability to create queues, see nearby queues on a map, and manage the queue. The epics consist of user stories that state users' behavior and expectations, such as the user wants to add a picture to the queue to make it more appealing or rename it to be more descriptive. The epics are divided into tasks such as create a user interface component for adding a picture or save a new picture of the queue to a database.

Most of the virtual companies' projects didn't have a product owner. Instead, the company's development team generated product backlog items, or PBI's, planned the next sprint, and pulled PBI's to the sprint backlog. Figure 5. illustrates the scrum framework.

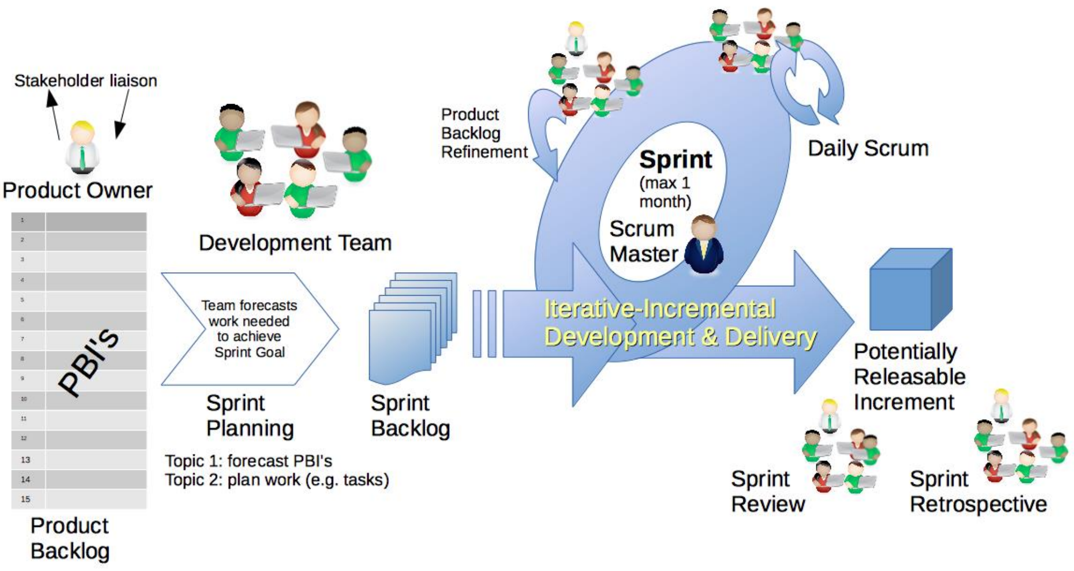


Figure 5. Scrum Framework illustrated. In WIMMA Lab, each project didn't have a Product Owner, and sprints were mainly two weeks long. (Mitchell, I. 2015)

2.4 Create

In the create phase, software development tasks are completed, environments are set up, source code management is in place, and continuous integration and deployment/delivery pipelines (CI/CD) are built. Developers are crucial during the creation phase as they create the code and complete tasks, but operations and security experts also have significant contributions. They consult developers to avoid security vulnerabilities, set up CI/CD pipelines, set up and harden environments. The hardening refers to, for example, enforcing policies, authentications, authorizations to users, or on servers closing unnecessary ports on servers to limit attack vectors.

Source control tools are crucial to managing the codebase among multiple developers and development teams. The source control empowers concurrency by branching. The codebase can be branched from the master/mainline branch to many branches to be developed independently. For example, each feature may have its branch, and when a feature is done, it can be merged back to the master. The merge can cause merge conflicts in source files if they are changed in different branches, and the source control tool cannot determine which version of the source file is current. In this case, the developers have to resolve the conflict manually.

The branching also enables developing new features and fixing existing bugs while the software is released. The new features and bug fixes are first merged to testing or staging branch, and if the source code in the testing branch passes quality assurance (for example, automatic tests, code analyzers, manual testing), then it is merged to release branch, which is used to build and deploy the release version. This process protects the main branches (master, mainline, release) from suboptimal code or even broken functionality, thus reducing risks and enhancing product quality.

2.5 Verify

The verify phase is all about quality assurance and providing feedback to the development team. Therefore, this phase is concurrent with the create phase, and ideally, verification tools are implemented to the continuous integration and deployment (CI/CD) pipeline as soon as possible to test the code. Manual testing such as concept testing may be executed parallel to automated verify tests such as code quality analyzers, unit tests, static analysis security tests, dynamic analysis security tests, and load tests, which run in the CI/CD pipeline.

The CI/CD platforms may have integrated verification tools, and additional tools can be integrated into the CI/CD pipeline by the operations (and the development and the security) team. For example, SonarQube is a static code analysis tool that can be implemented in the CI/CD pipeline, where it scans the code for issues and reports them. The static code analyzer contributes to maintaining a clean code base and avoid cumulating technical debt. Robot Framework has many applications, but its primary function is to run regression tests for web and mobile applications. The regression tests inform whether the application's prior features work as intended after changes are committed to the code base. If not, the application has regressed to a lower state where the previously working features are working no longer.

2.6 Package

In this phase, products and their dependencies are prepared for release, that is, packaged. This phase can be interpreted as the end of the continuous integration pipeline as now the product is packaged (integrated) for deployment. The product is compiled and built with production configurations instead of development or test parameters. Then the packages can be tagged to differentiate release candidates and versions of each other. Finally, the tagging can be used to lock the dependencies versions. The dependencies are external libraries or components, which are likely continuously developed. The dependencies might use semantic versioning, but that does not guarantee backward compatibility. Therefore it is vital to use specific versions of the packages' dependencies to mitigate the risk that a newer dependency causes issues.

Whenever a company uses external dependencies, it is essential to maintain dependencies in their environment or version control. It is a minor but business-critical risk that the dependency becomes unavailable in the future. Thus having them in-house is crucial for continuous business operations. The Amazon Web Services Elastic Container Registry (ECR) mentioned above is a fitting example of managing package resources. The ECR is used to store, manage, and deploy Docker images (Amazon Elastic Container Registry. N.d.), whether developed in-house or retrieved from external sources to own registry. The dependencies which are in the Docker image format are now available. The images can also be tagged to ease their management.

2.7 Secure

In this phase, the package is scanned for vulnerabilities and tested by the project team's security experts. This phase can be interpreted as the beginning of the continuous deployment and delivery pipeline, as the package exists by now and is ready for release. The secure phase is of utmost importance to mitigate security risks, such as hostile breaches, to retrieve sensitive customer information. The secure phase is also about securing the deployment environment, whether on-premises servers or a cloud platform.

A myriad of security tools can be integrated into continuous integration and deployment (CI/CD) pipelines. For example, Amazon Web Service has a native container image scanning for ECR (Elastic Container Registry), scans Docker images for vulnerabilities (Hausenblas, M. 2019). The security tool categories include, for example, license compliance tools to check licenses in project dependencies, secret detection tools to check for credentials, fuzzing tools to test the application with arbitrary inputs and payloads, container scanners, dependency scanners to check the dependencies for vulnerabilities (Secure. N.d.).

2.8 Release

In the release phase of the continuous integration and deployment (CI/CD) pipeline, the product is released. The product can be released to different environments such as testing or production and multiple servers or cloud platforms. The pipeline is a continuous integration and delivery pipeline if the product is released to production without the development team's operations experts. (Release. N.d.)

The release phase has multiple techniques, such as incremental rollout and tools available. In the incremental rollout, the production release is done in increments. If the production environment consists of multiple servers, then the release is made several times and incrementally releases to more servers. This process provides more time for the operations experts to detect any new release issues in the production environment. (Release. N.d.)

2.9 Configure

The configure phase has possibly evolved most due to cloud platforms' adoption, infrastructure as code tools, and a serverless deployment model. Each cloud platform has mechanisms to manage at least some portion of the cloud infrastructure as code.

That is, the operations team can script configurations, which modify the cloud. Infrastructure as code enables source controlling the configurations, the continuous integration and deployment (CI/CD) pipeline can set up the infrastructure, and the operations team can be confident that the infrastructure's state is desired. Say a company has a fleet of thousands of virtual machines, manually configured by different operators for several years. Likely, some machines are not configured as intended, which could cause issues or enable attack vectors for hostile parties. Managing the fleet by infrastructure as code, the same configurations can be issued to the whole fleet.

The serverless entails an abstraction level with no servers. It can be argued to be a platform as a service. For example, Amazon Web Services has AWS Lambda, and Google Cloud Platform has Cloud Functions, which execute source code's functions when they are invoked. In the serverless, the platform allocates function calls to their data centers to execute. Thus the development team does not manage the servers at all. The serverless is potentially, but not necessarily, cheaper than managing an on-premises server environment or infrastructure as a service virtual machines, as the platform clients are billed on how much resources (CPU time, memory, for example) the functions take. For instance, in a scenario where a company's servers are on low demand most of the time, the serverless can be cheaper as it is billed for actual use, not for the capacity.

2.10 Monitor

The monitor phase entails a product is released to a production environment, but monitoring testing and staging environments are feasible to detect issues. The kinds of monitoring tools are multitude, for example, gathering metrics, cluster monitoring, incident management, logging, tracing, and error tracking (Monitor. N.d.). The monitoring provides information about points of congestion, runtime errors, access, and change history, resource usage of servers, or cloud platform. The list is not exhausting.

For example, AWS X-Ray is a tracing tool that monitors activity in a system. One use case is a distributed cloud-native application utilizing a cloud platform's services and microservice architecture to the full extent. In such an environment, it is demanding to track and monitor user activity, for example, for audit trail purposes. A correlation id can be implemented, which identifies each trace and AWS X-Ray enables visibility to the traces' paths in the cloud and what function calls the traces associated within the microservices. (AWS X-Ray. N.d.)

Thus invocation chains caused by each users' actions (the traces) can be filtered precisely from the logs, which provides data to solve bugs and detect anomalies in the system. The traces generate audit trails that provide evidence of who has accessed and what actions were taken against specific data, which is paramount when a high degree of privacy is expected, for example, in a hospital environment (Nielsen, A. 2017).

2.11 Protect

The protect phase is tightly coupled with the monitoring to detect anomalies, intrusions, and threats to application runtime, cluster/network, or platform. Essential protection elements are not limited to monitoring and include authentication, authorization, access policies, and role-based account management. Other aspects concern, for example, container scanning, container host security, and container network security (Protect, N.d.). Other network-related protection mechanisms are IP filtering, firewalls, virtual networks, API gateways. Intrusion forensics, investigation, and incident reporting are in the protection domain. This domain is wildly varying and is relatively detached from developers' core expertise, though with cloud platforms such as Azure, the developers may manage many of the above aspects. While operations are likely more familiar with the protection domain, especially regarding administrative and network-related aspects, there is arguable a need for specialized security experts. For example, forensics is a remote skill set for developers and operations.

3 Methodology and Design

3.1 The Issue to be Solved

In the WIMMA Lab 2019, each virtual company implemented at least one continuous integration and deployment (CI/CD) pipeline. None was an enterprise production-ready pipeline in terms of robustness or coverage. In chapter 2 DevSecOps Development Phases is analyzed what phases can be in a CI/CD pipeline, and all of the phases were not used. The issue to be solved is identifying which phases were implemented, which were omitted, and which would be attractive candidates to implement in the future WIMMA Labs or similar project environments.

3.2 Goal

The goal is to enhance WIMMA Lab's software development processes by providing visibility to the state of continuous integration and deployment (CI/CD) implementations of the WIMMA Lab 2019. The processes benefit from identifying what key CI/CD phases were not implemented, and the WIMMA Lab steering group may choose to implement some of them. This thesis allows the steering group an opportunity to make informed decisions. These findings may be used in a similar project environment or other courses.

3.3 Initial environment

The WIMMA Lab 2019 source code is managed by a GitLab service on the client's LabraNet environment. The LabraNet is operated and maintained by JAMK's operations specialists. The GitLab is provided as a platform as a service (PaaS) to students, faculty members, and other stakeholders. Each virtual company of the WIMMA Lab has its subgroup organized under a group called "WIMMA Lab 2019" (https://gitlab.labranet.jamk.fi/wimma-lab-2019). In addition to the subgroups, the group has a "core" project used by the WIMMA Lab 2019 steering group.

3.4 Virtual Companies' Tasks

Each virtual company varied in its focus, type of work, team role composition, size, role, and technologies. Some tasks were from third-party clients, some from the WIMMA Lab steering group, some from the other virtual companies, and some were internal identified development or research needs.

3.4.1 Iotitude's Tasking

Iotitude was a virtual development company and developed a scalable Progressive Web Application (PWA) with React and Firebase called "Jonoxi." The application allowed users to create virtual queues anywhere. The application utilized mobile phones' GPS to get their location and show nearby queues. The queues could be searched and filtered. The unique business proposal was to allow persons, non-profits, companies, and services to improve user experience, image, and resource management. Say a shop and give away 100 items for the first customers at a specific location. The shop could create a virtual queue with 100 positions, insert information and metadata about the queue and post it. The customers could join the queue remotely by their mobile devices, which allows them to spend less time waiting, see when it is their turn to pick up the item, receive messages from the shop (i.e., the event is canceled), change their position in the queue and more.

3.4.2 Mysticons' Tasking

Mysticons was a virtual company that did the proof of concepts, research, and documentation. Mysticons created Green Book, which described the WIMMA Labs development processes and translated Black Book to English. The Black Book explains the WIMMA Lab concept. Mysticons also contributed to OPF (Open Project Framework). See http://wimmalab.pages.labranet.jamk.fi/guides-and-info/ for more information.

Mysticons also researched and tried out some test automation tools, had a robot with ROS (Robot Operating System), various sensors and components to develop machine vision, autonomous movement, and human-machine interaction, and finally made a proof of concept of an online music pedagogy application.

3.4.3 Overflow's Tasking

Overflow was a virtual development company that developed a web service for outdoor airsoft and paintball games. Mysticon's client wanted a situational awareness and tactical command application called "TACS" for team leaders and commanders. The most prominent games are played over several days and may have over 800 players. Players' mobile phone's GPS tracks them, and they receive orders to it from the commanders. The commanders could also order fire support and other game mechanics to the game area. Overflow's technology stack was React, React Native, NestJS, PostgreSQL, Google Cloud, and Robot Framework.

3.4.4 Pengwin's Tasking

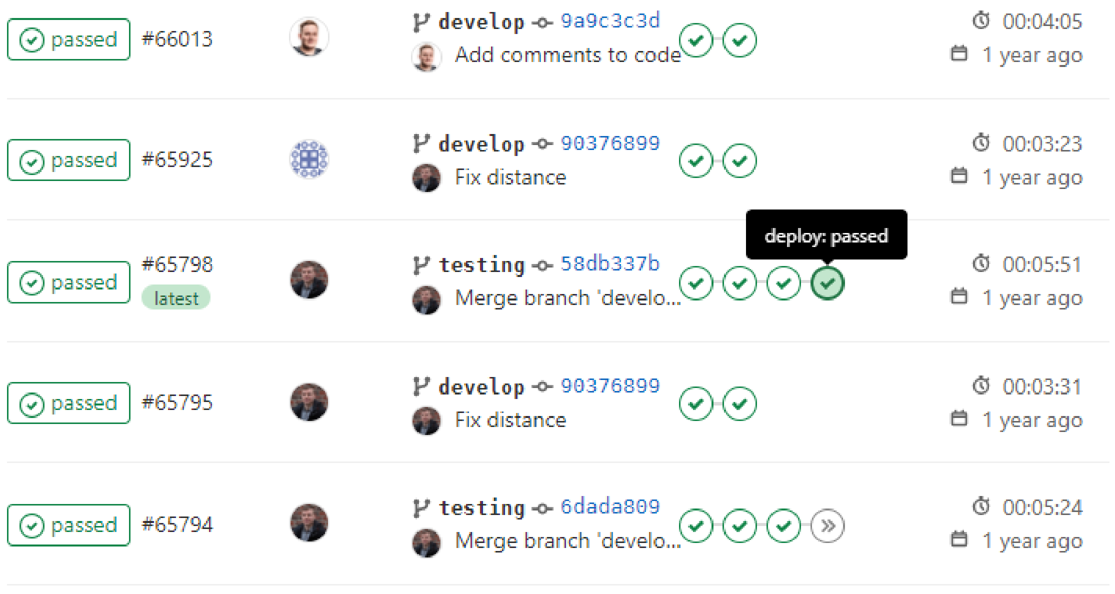
Pengwin was a virtual contractor company and provided services to the other virtual companies and the WIMMA Lab. Pengwin's services included facilitating UI designs, UI designing, websites, marketing, promoting, graphics, photos, videos, and audio. As such, they did not utilize DevSecOps-tools except to build and publish their website automatically.

3.5 Continuous Integration & Continuous Deployment Implementations

This chapter examines the configurations, CI/CD pipelines, and tools used by virtual companies. Each virtual company used GitLab for documentation, presentation, project management, and version control.

3.5.1 Iotitude's Implementations

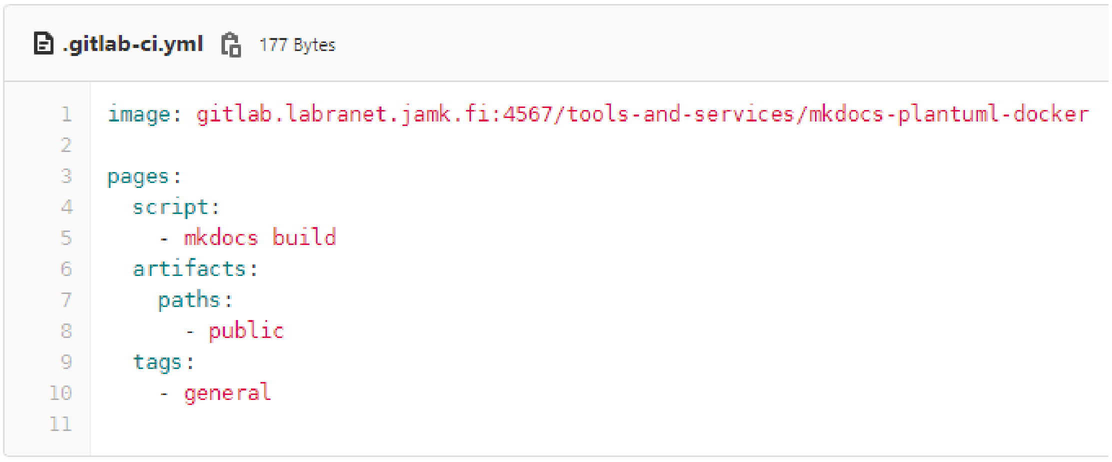
Iotitude had multiple projects in their GitLab group for different purposes, such as developing components, experimenting with new tools, documentation, and websites. Each project utilized GitLab's GitLab CI/CD feature, which performs CI/CD pipeline operations when a new commit is made to the projects' repository. The pipeline consists of a series of stages (such as integrate, deploy) consisting of parallel jobs (such as run scripts, copy files). GitLab runners perform the jobs. See Appendix 1. *Iotitude's GitLab Projects* for more information and links to the projects. Figure 6. is a screen capture from some of the performed pipeline runs. Notice commits to different branches initiate pipelines with a different number of jobs (check-marks next to timestamps). (GitLab CI/CD, N.d.)



passed	#66013	develop 9a9c3c3d Add comments to code		00:04:05 1 year ago
passed	#65925	develop 90376899 Fix distance		00:03:23 1 year ago
passed	#65798 latest	testing 58db337b Merge branch 'develo...	deploy: passed	00:05:51 1 year ago
passed	#65795	develop 90376899 Fix distance		00:03:31 1 year ago
passed	#65794	testing 6dada809 Merge branch 'develo...		00:05:24 1 year ago

Figure 6. Screen capture from Iotitude's the *jonoxi-front* project's GitLab pipelines run history. The *develop* and the *testing* branches had different pipelines. The former had four jobs and the latter two. See Appendix 1. and the *core* project for more information.

The pipeline, stages, and jobs are configured in a .gitlab-ci.yml configuration file (ibid.), which uses YAML, or *YAML Ain't Markup Language*, format. The YAML is a semi-structured data format, such as XML or JSON, and the structure is defined by indentation and line separation. If a GitLab Runner is configured to run as a Docker container, then used Docker image is specified. More of Docker below and see Figure 7. of Iotitude's *core* project's repository's configuration file. The YAML format is more human-readable than XML or JSON, as they are based on different brackets for demarking elements. Thus they are more cluttered and harder to read. (Classify your data, 2021)



```yaml
image: gitlab.labranet.jamk.fi:4567/tools-and-services/mkdocs-plantuml-docker

pages:
  script:
    - mkdocs build
  artifacts:
    paths:
      - public
  tags:
    - general

```

Figure 7. Iotitude's *core* project's repository's .gitlab-ci.yml configuration file. See Appendix 1. and the *core* project for more information.

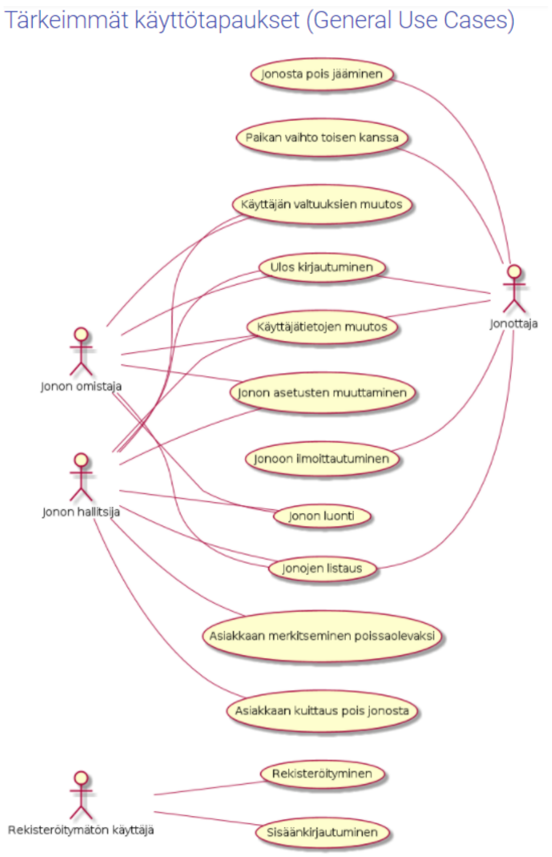
A GitLab Runner is an agent that runs the jobs. The runners are explicitly created to a project or shared across projects (shared runners were used in WIMMA Lab). If a job is initiated while the runner is busy performing another job, the job has to wait for the runner to complete its ongoing job. The GitLab CI/CD's coordinator API monitors projects' repositories and allocates runners to them as required. A GitLab environment may have multiple runners to avoid congestion and allow parallel execution. The runners can be configured differently, have different capabilities, and a tag may be assigned to a runner. Tag(s) are used in the .gitlab-ci.yml configuration file to specify which runners can run the job. In Figure 7., runners with a 'general' tag may run the job. (Configuring runners in GitLab, N.d.)

Iotitude's documentation, product, and team website projects used the GitLab Pages feature. The GitLab Pages support static websites, and the projects' GitLab CI/CD published them from their repositories. Each project has its own .gitlab-ci.yml describing the stages and jobs to be performed by the GitLab runner, which is launched when a commit is made to a specific or any of the repository's branches, which are monitored by the GitLab CI/CD's coordinator API. In Figure 7., 'pages' inform the GitLab Runner to publish to the GitLab Pages. (GitLab Pages, N.d.)

For example, Iotitude's project management and documentation project's .gitlab-ci.yml configuration utilized Docker, MkDocs, and PlantUML technologies to publish its static website. On a commit to the repository, a GitLab Runner runs the jobs in a Docker container. The .gitlab-ci.yml specifies which Docker image is used to create the container by GitLab CI/CD (The runner is set up as a Docker executor). The Docker is a containerization software, and a container is a standard unit of software that has packaged up all code and all its dependencies. Docker runs isolated containers on the host machine's operating system and is not a virtualization technique. It is lightweight compared to a virtual machine (it's an application process, not an operating system). Therefore it is suitable to create temporary and isolated execution contexts. In Figure 7., 'image' specifies which Docker image is used to run the job. Figure 9. line 3. indicates that the GitLab Runner is configured as a Docker executor with URI to the Docker image. (What is a Container?, N.d.)

MkDocs is a static site generator that generates static webpages from Markdown markup language and thus is supported by GitLab Pages. Similar to configuring GitLab CI/CD, the MkDocs generator is configured by a YAML file. Iotitude's configuration included static website settings like site name, used Markdown extensions (PlantUML), theme (i.e., logo, color palette), and Google Analytics to gather visitor metrics. (MkDocs, N.d.)

PlantUML is an open-source tool to create UML and many non-UML diagrams from a text-based language. The diagrams can be version controlled as they are text, and no graphic design tools are necessary to modify the diagrams. The diagrams are generated in PNG, SVG, or LaTeX format. The MkDocs generator produces the diagrams from the Markdown notation each time the GitLab Pages is built. See Figure 8. below for an example of a generated diagram and its source code. (PlantUML in a nutshell, N.d.)


Tärkeimmät käyttötapaukset (General Use Cases)
Jonosta pois jääminen
Paikan vaihto toisen kanssa
Käyttäjän valtuuksien muutos
Ulos kirjautuminen
Käyttäjätietojen muutos
Jonon asetusten muuttaminen
Jonoon ilmoittautuminen
Jonon luonti
Jonojen listaus
Asiakkaan merkitseminen poissaolevaksi
Asiakkaan kuittaus pois jonosta
Rekisteröityminen
Sisäänkirjautuminen
Jonon omistaja
Jonon hallitsija
Jonottaja
Rekisteröitymätön käyttäjä

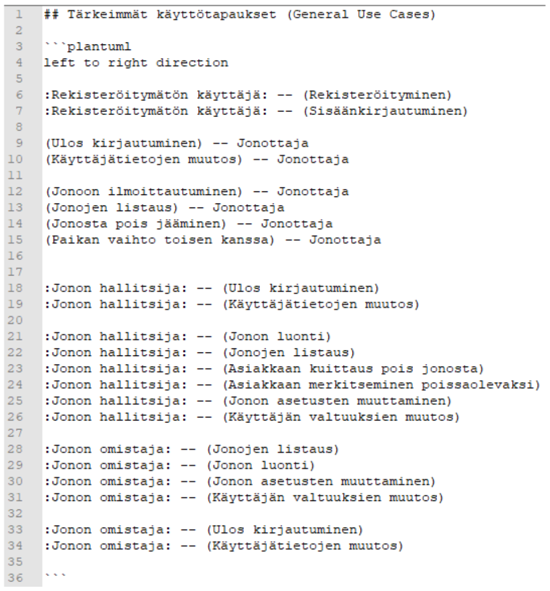
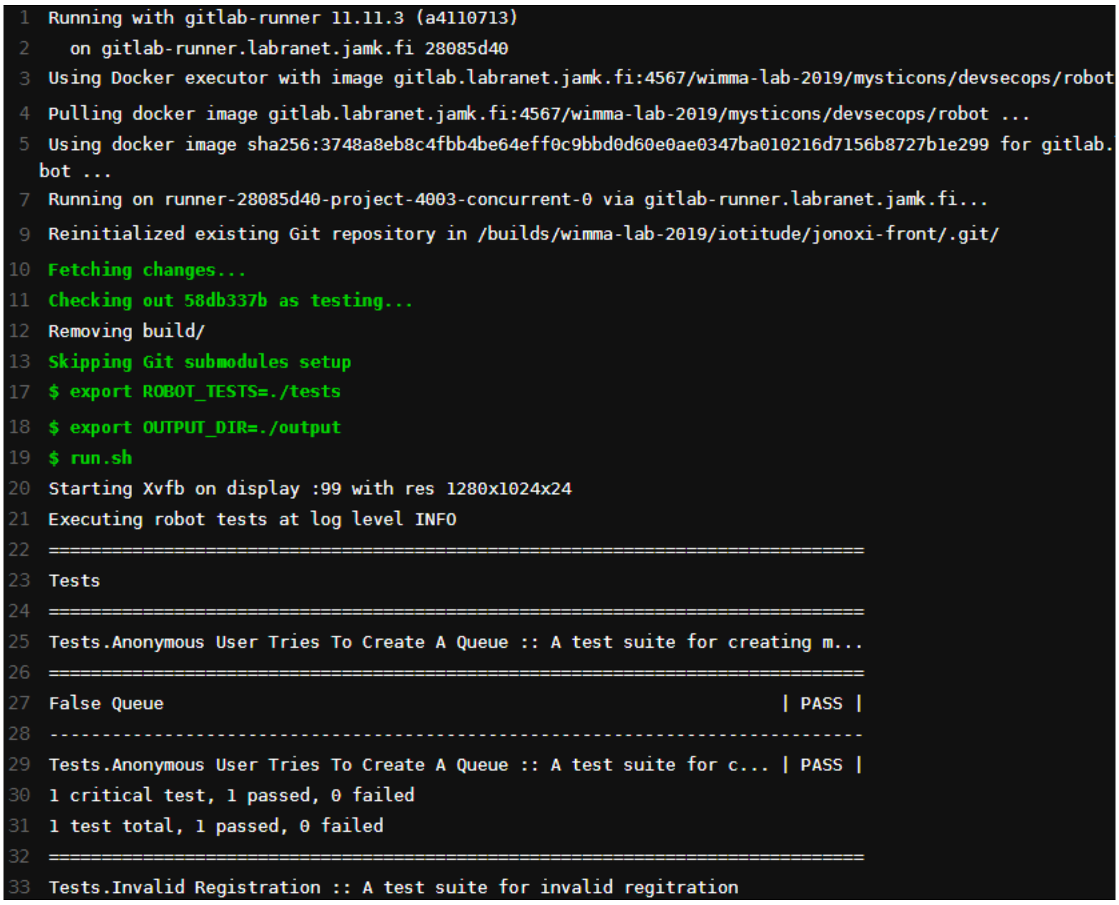

~~~
## Tärkeimmät käyttötapaukset (General Use Cases)

```plantuml
left to right direction

:Rekisteröitymätön käyttäjä: -- (Rekisteröityminen)
:Rekisteröitymätön käyttäjä: -- (Sisäänkirjautuminen)

(Ulos kirjautuminen) -- Jonottaja
(Käyttäjätietojen muutos) -- Jonottaja

(Jonoon ilmoittautuminen) -- Jonottaja
(Jonojen listaus) -- Jonottaja
(Jonosta pois jääminen) -- Jonottaja
(Paikan vaihto toisen kanssa) -- Jonottaja


:Jonon hallitsija: -- (Ulos kirjautuminen)
:Jonon hallitsija: -- (Käyttäjätietojen muutos)

:Jonon hallitsija: -- (Jonon luonti)
:Jonon hallitsija: -- (Jonojen listaus)
:Jonon hallitsija: -- (Asiakkaan kuittaus pois jonosta)
:Jonon hallitsija: -- (Asiakkaan merkitseminen poissaolevaksi)
:Jonon hallitsija: -- (Jonon asetusten muuttaminen)
:Jonon hallitsija: -- (Käyttäjän valtuuksien muutos)

:Jonon omistaja: -- (Jonojen listaus)
:Jonon omistaja: -- (Jonon luonti)
:Jonon omistaja: -- (Jonon asetusten muuttaminen)
:Jonon omistaja: -- (Käyttäjän valtuuksien muutos)

:Jonon omistaja: -- (Ulos kirjautuminen)
:Jonon omistaja: -- (Käyttäjätietojen muutos)

```
~~~

Figure 8. A generated PlantUML diagram of Iotitude's *Jonoxi* services general use cases and the diagram's source code. See Appendix 1. and the *core* project for more information.

The "Jonoxi-Back" project utilizes its .gitlab-ci.yml to get a Docker image with Firebase installed. The configuration builds the Cloud Functions for Firebase from the source code and publishes it to Google Cloud Platform. The Firebase is a platform with multiple services to build serverless web and mobile applications. Iotitude used multiple Firebase products such as Firebase Authentication for authentication and authorization, Cloud Firestore as a database, and Cloud Functions for Firebase as a serverless backend. (Use Firebase in Progressive Web Apps, 2020)

The "Jonoxi-Front" project utilizes its .gitlab-ci.yml for two pipelines. The first is a continuous deployment pipeline for development purposes. It monitors the "develop" Git branch for changes, builds the source code, and deploys it to a Firebase development environment. The second is a continuous delivery/deployment pipeline and monitors the "testing" branch, builds the source code, and deploys it to a Firebase testing environment. Robot Framework tests are run against the testing environment, and if the tests pass, the last step is deployment to a Firebase production environment. The Robot Framework is an open-source automation framework for test automation and robotic process automation (Robot Framework. N.d.). The Robot Framework runs automated tests against registration and creating queues. Commits to the "testing" branch always initiate building and deployment to the testing environment (continuous deployment), but the deployment to the production environment requires a human to initiate the pipeline manually. Thus publish to the production is a continuous delivery pipeline as it delivers the build to the testing and runs the tests but does not deploy to the production. Figure 9. is a partial screen capture from a job that runs the Robot Framework tests. The tests start from line 20, and it has multiple tests.



```
Running with gitlab-runner 11.11.3 (a4110713)
  on gitlab-runner.labranet.jamk.fi 28085d40
Using Docker executor with image gitlab.labranet.jamk.fi:4567/wimma-lab-2019/mysticons/devsecops/robot
Pulling docker image gitlab.labranet.jamk.fi:4567/wimma-lab-2019/mysticons/devsecops/robot ...
Using docker image sha256:3748a8eb8c4fbb4be64eff0c9bbd0d60e0ae0347ba010216d7156b8727b1e299 for gitlab.
bot ...
Running on runner-28085d40-project-4003-concurrent-0 via gitlab-runner.labranet.jamk.fi...
Reinitialized existing Git repository in /builds/wimma-lab-2019/iotitude/jonoxi-front/.git/
Fetching changes...
Checking out 58db337b as testing...
Removing build/
Skipping Git submodules setup
$ export ROBOT_TESTS=./tests
$ export OUTPUT_DIR=./output
$ run.sh
Starting Xvfb on display :99 with res 1280x1024x24
Executing robot tests at log level INFO
==============================================================================
Tests
==============================================================================
Tests.Anonymous User Tries To Create A Queue :: A test suite for creating m...
==============================================================================
False Queue                                                           | PASS |
------------------------------------------------------------------------------
Tests.Anonymous User Tries To Create A Queue :: A test suite for c... | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Tests.Invalid Registration :: A test suite for invalid regitration
```

Figure 9. Partial screen capture from *rf-test* job's printing. The job runs the Robot Framework tests, and the GitLab Runner uses Docker executor. See Appendix 1. and the *jonoxi-front* project for more information.

The "Jonoxilanding" project utilizes its .gitlab-ci.yml configuration file. Instead of the "build-test" project, the files are not built but are instead copied and moved. Jonoxilanding also has a .travis.yml, which is for Travis CI continuous integration and deployment tool, but it is obsolete as GitLab CI is used instead of Travis CI.

The "pipeline-testi" project utilizes its .gitlab-ci.yml to prototype with Locust and OWASP Zap test automation tools. The Locust is used for load testing endpoints, and the OWASP Zap is a web application scanner for security testing. Mysticons provided the Docker images for both tools.

Iotitude used multiple environments, platforms, tools, and technologies to develop and deploy its solutions. Almost every project used the GitLab CI/CD feature to implement a continuous integration and deployment pipeline, and the "Jonoxi-Front" project has the most elaborate pipeline with four stages and deployments to different environments.

3.5.2 Mysticons’ Implementations

The "avoimet-ovet-demo" project has the most comprehensive .gitlab-ci.yml configuration file in terms of steps. Its pipeline has six steps and the last step, deployment to the production environment, requires that the pipeline is initiated manually. However, as this is a demo pipeline, not every step has actual actions. The first step builds a Docker image and pushes it to the project's GitLab Container Registry. The second step uses Google Cloud SDK Docker image to connect to the Google Cloud Platform, configure settings, create a container cluster, runs deployment.yml with Google Kubernetes Engine to run the Docker image on Kubernetes. Steps three to six ("run_robot_tests,” "run_locust_tests,” "run_zap_tests," and "deploy_prod_app") echoes a description of the step’s purpose; they do not contain actual logic or actions. Thus during the Open Doors event, the visitors do not have to wait unnecessarily, and they gain insight into the purpose of the pipeline. See Appendix 2. *Mysticons' GitLab Projects* for more information and links to the projects.

The "Cayac" is a group of seven projects. Its purpose is to provide a base group for new virtual companies, and it contains documentation, examples, and templates. For example, it describes how to register new GitLab Runners, which run the GitLab CI pipelines. The "Tools" project demonstrates how to use the Anchore tool to scan Docker containers, which was not used by any actual development project.

The "DevSecOps" project does not have pipelines, and it is used for prototyping. A new tool is fMBT, which generates graphs, execution paths, and tests from programmatic configuration files.

Mysticon's projects introduced new technologies (Anchore, fMBT, and Kubernetes) to their pipelines compared to Iotitude. Many projects use some of the technologies used by Iotitude, such as Docker, and GitLab Pages, and Robot Framework. Mysticons had a more extensive .gitlab-ci.yml configuration with six stages, and Mysticons used more tools, which is appropriate as Mysticons was the technology demonstrator virtual company.

3.5.3 Overflow's Implementations

Overflow used similar technologies as Iotitude and Mysticons. A new feature Overflow used is Docker Compose to define a multi-container Docker GitLab runner (Overview of Docker Compose. N.d.). The "ehasa-backend" project utilizes its .gitlab-ci.yml to purge old Docker containers and use Docker Compose to set up two Docker containers. The first container is for the backend application, and the other is for the Postgres database. See Appendix 3. *Overflow's GitLab Projects* for more information and links to the projects.

3.5.4 Pengwin's Implementations

Pengwin virtual company's GitLab projects do not feature new continuous integration and deployment usages or tools than previous virtual companies implementations. See Appendix 4. *Pengwin's GitLab Projects* for more information and links to the projects.

4 Findings and solutions

4.1 Implemented DevSecOps Development Phases in WIMMA Lab 2019

Most of the projects had a continuous integration, continuous delivery, or continuous deployment pipeline. Most of the pipelines did not have many steps and published websites instead of complete applications and environments. Some pipelines were used for learning and prototyping, and a few for testing the project's source code. Coverage of the phases is complete because some tools or technologies are incorporated into each of the phases.

This chapter contrasts the virtual companies' continuous integration implementations to the Ten Phases of DevSecOps Software Development Life Cycle presented in chapter 2.

4.1.1 Manage

In the WIMMA Lab 2019, every virtual company and the WIMMA Lab management used GitLab for project management. GitLab is a platform with multiple tools for every phase of the software development life cycle (GitLab N.d.). GitLab can be used as an online service, or it can be installed on-premises. The WIMMA Lab 2019 utilized on-premises GitLab hosted and maintained by JAMK University of Applied Sciences, and public repositories are available on https://gitlab.labranet.jamk.fi/wimma-lab-2019. It was used for communication through issues and tasks, and documentation.

Slack messaging service was used for more immediate and personal communication within and between the virtual companies and sharing WIMMA Lab-wide announcements.

Reportronic work hour tracking was used to manage work hours, even though the students of WIMMA Lab weren't paid.

4.1.2 Plan

GitLab was used to plan epics, issues, milestones, and sprints. The virtual companies held daily stand-up meetings, sprint plannings, retrospectives, among other meetings according to the Scrum Framework.

Each day virtual companies' lead and the WIMMA Lab head coach had a stand-up meeting to discuss common themes such as announcements, events, and team statuses.

Product backlog items and sprint backlog was maintained in GitLab.

4.1.3 Create

GitLab was used for source control management (Git repositories of the projects), and GitLab CI, GitLab Pages, GitLab Runners were used for continuous integration and delivery/deployment pipelines.

Created documentation was also source controlled to a GitLab project's repository. Some binary files, such as Microsoft Office documents, were source-controlled, but most of the created pictures, videos, and other binary file outputs were uploaded to Dropbox cloud service.

4.1.4 Verify

The virtual companies did manual testing and quality assurance while concurrently developing, especially Iotitude and Overflow, as they were developing a singular product with a large team. Iotitude and Overflow integrated Locust and Robot Framework to their continuous integration pipelines for load testing, integration, and functional testing. Mysticons studied fMTB for test case generation and testing.

Overflow used Postman to automate REST API endpoint regression testing by creating collections of requests. The collections with predetermined variables can be run with minimal effort.

4.1.5 Package

Every project used .gitlab-ci.yml configuration files to package the source code. The configuration controls GitLab CI, which utilizes GitLab Runners to perform build actions with a specific Docker image from GitLab Container Registry or external container registry.

MkDocs static website generator with PlantUML was used to build and package some of the websites and their diagrams.

4.1.6 Secure

Iotitude and Mysticons tested Anchorage for container scanning and OWASP Zap for web application vulnerability scanning.

Overflow had a cybersecurity student who hardened environments and improved security.

4.1.7 Release

Static websites were released by the GitLab Pages feature, and the source code was continuously deployed to testing environments. Some pipelines were configured as continuous delivery pipelines to build, deploy test environments, and run automated tests. Such pipelines required human decisions to deploy to the production environment. GitLab Runners, GitLab CI, GitLab Container Registry, and Docker were used to release source code and run the environments.

Iotitude and Overflow had significant applications. The former released their application to a serverless Google Cloud Firebase platform and the latter to Google Cloud. Figure 10. illustrates Overflow's release runtime plan.

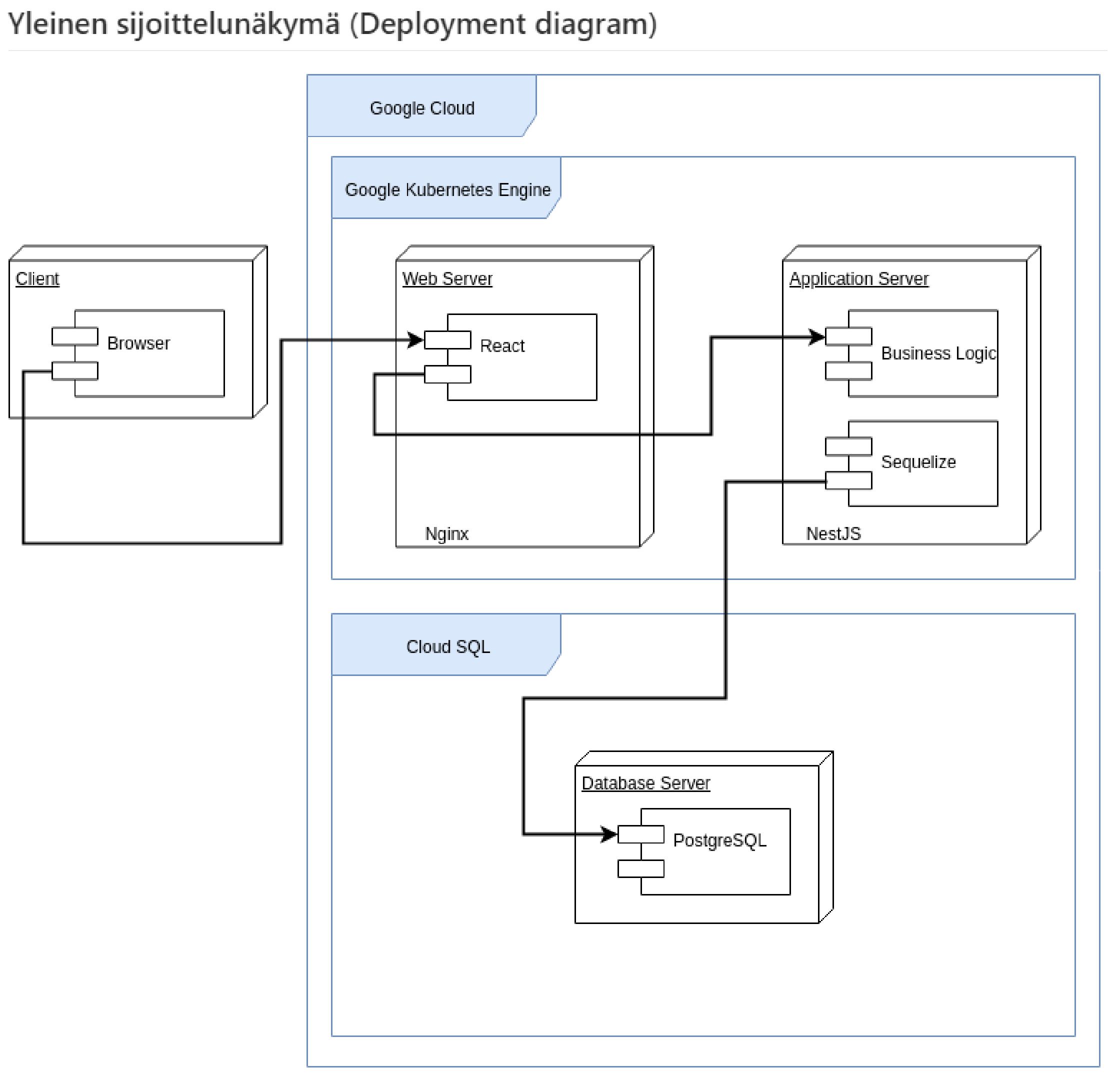


Figure 10. Deployment diagram of Overflow's Google Cloud environment. See Appendix 3. and the *core* project for more information.

4.1.8 Configure

Iotitude configured Google Firebase and Cloud Functions to have a serverless platform managed as infrastructure as code. Overflow configured necessary databases to Docker containers (refer to Figure 10.), managed as infrastructure as a code.

Configurations are mostly in .gitlab-ci.yml configuration files. Some Firebase-specific configuration formats are JSON and *.rules files. Overflow's NestJs and TypeScript configurations are also JSON files.

4.1.9 Monitor

The virtual companies monitored costs of running services on Google Cloud Platform and the services health, usage statistics, estimated costs, and configurations. The virtual companies' homepages and product pages used Google Analytics to monitor visitor traffic by multiple metrics.

Overflow monitored NGINX server logs and noticed a possible scripted intrusion attempt or an indifferent web crawler.

4.1.10 Protect

Overflow's cybersecurity student did some hardening to Overflow's environment, but otherwise, the virtual companies relied on GitLab's and Google Cloud Platform's security settings, user authentication, and authorization, for example, Firebase Authentication, firewalls, and other features.

Some of the virtual companies' project repositories are private and require specific user roles to be accessible.

5 Conclusion

Surprisingly, the virtual companies covered all of the DevSecOps lifecycle phases to some degree. This feat demonstrates that the WIMMA Lab is a very advanced project environment concept as it enables multidisciplinary teams with cybersecurity, media and web development, network, and programming students to spend time to enhance these business-critical processes. In most of the courses, the focus is very narrow, which entails it is unnecessary or even undesired to build elaborate continuous integration and delivery/deployment pipelines (CI/CD). Mysticons improved existing and created new documentation, examples, and tutorials on using CI/CD pipelines and various technologies and tools. Still, several phases are lacking, most noticeable the secure, monitor, and protect phases. Each phase is susceptible to improvements and further development. The CI/CD pipelines could, for instance, update epics, issues, and other planning artifacts as the pipeline proceeds.

Manage and plan phases would have benefited from a more rigorous and extensive project and sprint management tools usage. For example, only a few epics were used, and issues were mainly plain without assignees or iteration. These are cultural issues, not technical, and relatively easy to incorporate as the tooling is sufficient. Sometimes this kind of cultural accountability, aptitude, and experience is more important to get a first paid job than technical skills. That is, most student cannot express them and may appear not as professional. The next WIMMA Lab should focus on issue quality.

Create, verify, package, release, and configure phases were advanced and used multiple tools such as Docker, fMBT, Kubernetes, Locust, and Robot Framework. Also, many environment and platform-specific setups were done, especially on Google Cloud Firebase. Unit testing was minor, and test-driven development nonexistent. These are also cultural issues, to some extent, and improving them would be very beneficial to students. On the other hand, it may initially demand time which is a scarce resource. Especially during the first four weeks, virtual companies have to produce content for the Open Doors event. Still, the next WIMMA Lab should be considered this.

The secure, monitor, and protect phases were unsurprisingly most rudimentary. The product lines, that is, Iotitude and Overflow, focused on application development to meet their respective clients' requirements. Mysticons did some research and proof on concept work to these phases, and these could be improved in the successive WIMMA Lab instances or on other project learning programs. For example, the protect phase could include security as code to automatically create security rules, such as role-based account roles, to the environments. Extensive secure, monitor, and protect configurations take much advanced technical know-how and might be out of the scope of the WIMMA Lab. If the next WIMMA Lab has dedicated cybersecurity of network student who focuses on operations and security, these phases will improve.

Overall the level of implemented CI/CD automation and processes are way more advanced on comprehensive than in courses. Some courses touch these areas, such as *Cloud Services and Automation*, but most do not. The WIMMA Lab 2019 had many visitors and guests from the software industry, and most were impressed with what the virtual companies had done in terms of CI/CD. This reaction may indicate that the software industry has varying CI/CD automation levels, and it is likely that most companies could improve theirs. And that this kind of know-how is valuable for students. For example, one Mysticon student now works as a CI/CD specialist.

During the WIMMA Lab Open Doors event, Mysticons' DevOps demo garnered much attention from the visitors. The visitors were alumni and information technology professionals from local companies. In the discussions at the Open Doors event and other instances as interested professionals visited the WIMMA Lab regularly to see what we are doing, give lectures, or recruit, it became evident that this kind of knowledge about the CI/CD automatization is in high demand thus providing significant value to the students. This observation further validates the WIMMA Lab concept as an excellent platform to learn about communication skills, presenting, processes and tooling, soft skills, and teamwork.

References

7 Lean Metrics to Improve Flow. N.d. Article about Kanban. Accessed on 5 January 2020. Retrieved from https://leankit.com/learn/kanban/lean-flow-metrics/.

Amazon Elastic Container Registry. N.d. Accessed on 4 February 2020. Retrieved from https://aws.amazon.com/ecr/.

AWS X-Ray. N.d. Product information of a tracing tool. Accessed on 5 February 2020. Retrieved from https://aws.amazon.com/xray.

Best Practices in Project and Portfolio Management. N.d. A white paper by Microfocus. Accessed on 5 January 2020. Retrieved from https://www.microfocus.com/media/white-paper/best_practices_in_project_and_portfolio_management_wp.pdf.

Carter, K. 2017. *Francois Raynaud on DevSecOps*. IEEE Software, vol. 34, no. 5, pp. 93-96.

Classify your data. 2021. Accessed on 11 April 2021. Retrieved from https://docs.microsoft.com/en-us/learn/modules/choose-storage-approach-in-azure/2-classify-data.

Configure. N.d. GitLab configure tools overview. Accessed on 4 February 2020. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/configure/.

Configuring runners in GitLab. N.d. Accessed on 11 April 2021. Retrieved from https://docs.gitlab.com/ee/ci/runners/README.html.

Development-Operation Cycle Security (DevSecOps). N.d. Accessed on 14 February 2021. Retrieved from https://etek.com/devsecops/.

DevOps Tools Landscape. N.d. Page on GitLab's website. Accessed on 15 October 2019. Retrieved from https://about.gitlab.com/devops-tools/.

DevOps vs Agile – Understand The Difference. 2020. Accessed on 14 February 2021. Retrieved from https://reqtest.com/agile-blog/agile-vs-devops/.

Entire DevOps lifecycle in one application. N.d. Accessed on 15 October 2019. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/.

GitLab. N.d. Accessed on 28 April 2021. Retrieved from https://about.gitlab.com/.

GitLab CI/CD. N.d. Accessed on 11 April 2021. Retrieved from https://docs.gitlab.com/ee/ci/README.html.

GitLab Continuous Integration (CI) & Continuous Delivery (CD). N.d. Page on GitLab's website. Accessed on 27 July 2019. Retrieved from https://about.gitlab.com/product/continuous-integration/.

GitLab Pages. N.d. Accessed on 11 April 2021. Retrieved from https://docs.gitlab.com/ee/user/project/pages/.

Hausenblas, M. 2019. *Native Container Image Scanning in Amazon ECR*. Accessed on 21 January 2019. Retrieved from https://aws.amazon.com/blogs/containers/amazon-ecr-native-container-image-scanning/.

Kuuva, P. 2020. *A Brief History of DevOps*. Bachelor's Thesis. University of Jyväskylä. http://urn.fi/URN:NBN:fi:jyu-202011236721.

Manifesto. N.d. DevSecOps manifesto. Accessed on 5 August 2019. Retrieved from https://www.devsecops.org/.

Mar, S. 2018. *Bringing Cybersecurity into the Future: Internal auditors should consider whether CARTA is a smarter approach to addressing information security risks*. Internal Auditor. 2018;75(1):16-17.

Mitchell, I. 2015. *Scrum Framework*. Accessed on 18 April 2021. Retrieved from https://commons.wikimedia.org/wiki/File:Scrum_Framework.png.

MkDocs. N.d. Accessed on 11 April 2021. Retrieved from https://www.mkdocs.org/.

Monitor. N.d. GitLab release tools overview. Accessed on 5 February 2020. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/monitor/.

Mäkäläinen, M. 2018. *WIMMA Lab Black Book*. Description of WIMMA Lab concept. Accessed on 3 August 2019. Retrieved from http://wimmalab.pages.labranet.jamk.fi/site/media/WIMMALab-BlackBook-1.0.ec5daf40.pdf.

Nielsen, A. 2017. *GDPR: What can you prove?* Article. Accessed on 13 February 2020. Retrieved from https://gdpr.report/news/2017/11/23/gdpr-can-prove/.

Overview of Docker Compose. N.d. Accessed on 18 April 2021. Retrieved from https://docs.docker.com/compose/.

Pittet, S. N.d. *Continuous integration vs. continuous delivery vs. continuous deployment*. Accessed on 14 February 2021. Retrieved from https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

PlantUML in a nutshell. N.d. Accessed on 12 April 2021. Retrieved from https://plantuml.com/.

Protect. N.d. GitLab protect tools overview. Accessed on 24 January 2021. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/protect/.

Ramos, M. 2016. *GitLab Workflow: An Overview*. Article of GitLab usage. Accessed on 6 January 2020. Retrieved from https://about.gitlab.com/blog/2016/10/25/gitlab-workflow-an-overview/.

Release. N.d. GitLab release tools overview. Accessed on 4 February 2020. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/release/.

Robot Framework. N.d. Accessed on 18 April 2021. Retrieved from https://robotframework.org/.

Secure. N.d. GitLab security tools overview. Accessed on 4 February 2020. Retrieved from https://about.gitlab.com/stages-devops-lifecycle/secure/.

Tatiyants, A. 2012. *Impact of Continuous Delivery Beyond Engineering*. Accessed on 28 December 2019. Retrieved from http://tatiyants.com/impact-of-continuous-delivery-beyond-engineering/.

Use Firebase in Progressive Web Apps. 2020. Accessed on 18 April 2021. Retrieved from https://firebase.google.com/docs/projects/pwa.

Appendices

Appendix 1. Iotitude's GitLab Projects

Project	**Description**	**Source**
build-test	This project was used for prototyping the front end's build. Tools: GitLab CI, GitLab Runner, GitLab Pages.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/build-test
core	This project contains documentation, specifications and, an issue board. Tools: GitLab CI, GitLab Runner, GitLab Pages, Docker, Markdown, MkDocs, PlantUML, Google Analytics	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/core
core	This specification file includes an example of a generated PlantUML diagram. For example, see the General Use Cases diagram.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/core/-/blob/master/dokumentit/02-vaatimusmaarittely/vaatimusmaarittely.md
Jonoxilanding	The source code and assets of the "Jonoxi" applications landing page. NB: This is a private project.	http://wimma-lab-2019.pages.labranet.jamk.fi/iotitude/jonoxilanding/
Jonoxi-Back	The source code of the back end, the central technology is Cloud Functions for Firebase.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/jonoxi-back
Jonoxi-Front	The source code of the front end, the central technology is React. To see Pipelines click on	https://gitlab.labranet.jamk.fi/wimma-lab-

	the left side links CI/CD and then Pipelines.	2019/iotitude/jonoxi-front
pipeline-testi	This project was used for prototyping the GitLab CI pipelines.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/pipeline-testi
React-Proto	This project was used for prototyping with React.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/react-proto
site	The Iotitude virtual company's website.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/site
source	An example project and was not used.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/iotitude/source

Appendix 2. Mysticons' GitLab Projects

Project	**Description**	**Source**
avoimet-ovet-demo	Demonstration material for WIMMA Lab 2019 Open Doors event.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/avoimet-ovet-demo
Cayac	This purpose-built base template folder contains tools and documentation for future WIMMA Lab virtual companies.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/cayac
core	This project contains documentation, specifications and, an issue board.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/core

DevSecOps	This project was used for prototyping various DevSecOps tools.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/devsecops
opf-vanilla-en	This project was used to translate The Black Book into English.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/opf-vanilla-en
site	The Mysticons virtual company's website.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/site
source	An example project and was not used.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/source
Turtlebot	This project was used for prototyping the Turtlebot 2 robot.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/mysticons/turtlebot

Appendix 3. Overflow's GitLab Projects

Project	**Description**	**Source**
core	This project contains documentation, specifications and, an issue board.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/core
ehasa-backend	The source code of the back end, the central technology is Nest.js.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/ehasa-backend
ehasa-frontend	The source code of the front end, the central technology is React. To see Pipelines click on	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/ehasa-frontend

	the left side links CI/CD and then Pipelines.	
ehasa-mobile	The source code of the mobile application, the central technology is React Native.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/ehasa-mobile
site	The Overflow virtual company's website.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/site
source	An example project and was not used.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/source
Tacslanding	The source code and assets of the "TACS" applications landing page. NB: This is a private project.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/overflow/tacslanding

Appendix 4. Pengwin's GitLab Projects

Project	**Description**	**Source**
core	This project contains documentation, specifications and, an issue board. NB: This is a private project.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/pengwin/core
site	The Pengwin virtual company's website. NB: This is a private project.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/pengwin/site
source	An example project and was not used. NB: This is a private project.	https://gitlab.labranet.jamk.fi/wimma-lab-2019/pengwin/source

Appendix 5. Phase 1: Manage

Phase	**Manage**
Responsibility	Development, Security, Operations
Description	With management tools, project managers and product owners have enhanced awareness of project status and team management. Managing the project is a continuous process and can follow a specific method, such as Kanban.
Tool example	Microfocus Project and Portfolio Management.

Appendix 6. Phase 2: Plan

Phase	**Plan**
Responsibility	Development, Security, Operations
Description	Project workload can be arranged into smaller collections, such as epics and milestones, to provide a more granular and seminated understanding of the workload.
Tool example	Trello.

Appendix 7. Phase 3: Create

Phase	**Create**
Responsibility	Development
Description	These tools are used to manage the project's codebase. They can provide a repository for the code, branching, and merging tools to improve concurrent programming within a team.
Tool example	Subversion.

Appendix 8. Phase 4: Verify

Phase	**Verify**
Responsibility	Development, Operations

Description	Continuous integration (CI) pipeline can begin on, creation phase and verification tools are used to test and analyze produced code.
Tool example	Travis CI.

Appendix 9. Phase 5: Package

Phase	**Package**
Responsibility	Operation
Description	CI-pipeline can be extended to a continuous deployment pipeline with packaging tools. They build and package code to a deployable application.
Tool example	Sonatype Nexus Repository Pro.

Appendix 10. Phase 6: Secure

Phase	**Secure**
Responsibility	Development, Security, Operations
Description	These tools are used to find security vulnerabilities, dependency issues with other libraries and manage license compliance. This phase should be integrated into a continuous deployment/delivery pipeline.
Tool example	Snyk.

Appendix 11. Phase 7: Release

Phase	**Release**
Responsibility	Operations
Description	Continuous delivery (CD) pipeline release application to servers automatically. CD requires a very high degree of confidence in the pipeline because human verification is not present. (Tatiyants, A. 2012)
Tool example	Spinnaker.

Appendix 12. Phase 8: Configure

Phase	**Configure**
Responsibility	Operations
Description	These tools provide infrastructure as code. Environments are configured to source files and tools to build, manage and orchestrate the environment automatically. For example, they provision new Kubernetes pods to cope with increased traffic.
Tool example	Kubernetes.

Appendix 13. Phase 9: Monitor & Defend

Phase	**Monitor**
Responsibility	Security, Operations
Description	Monitoring tools provide information about applications in the production environment and of the state of the environment itself. These tools are helpful to recognize resource bottlenecks and contribute to detect malicious intrusions.
Tool example	AWS X-Ray.

Appendix 14. Phase 10: Protect

Phase	**Protect**
Responsibility	Security, Operations
Description	These tools are used to prevent, detect, contain, and deny intrusions, anomalies, and threats against applications or environments. This phase of the software development lifecycle includes security experts' penetration testing.
Tool example	FireEye.