This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101069732

aerOS

EUROPEAN IOT-EDGE-CLOUD

D2.4 - DevPrivSecOps methodology specification (1)

Deliverable No.	**D2.4**	**Due Date**	**31-May-2023**
Type	**Report**	**Dissemination Level**	**Public**
Version	**1.0**	**WP**	**WP2**
Description	*Specification of technologies, tools and environment needed to be put in place for instantiation of the DevPrivSecOps approach of aerOS*		

The aerOS consortium consists of the following 27 partners::

UNIVERSITAT POLITÈCNICA DE VALÈNCIA	ES
NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"	EL
ASOCIACION DE EMPRESAS TECNOLOGICAS INNOVALIA	ES
TTCONTROL GMBH	AT
TTTECH COMPUTERTECHNIK AG (*third linked party*)	AT
SIEMENS AKTIENGESELLSCHAFT	DE
FIWARE FOUNDATION EV	DE
TELEFONICA INVESTIGACION Y DESARROLLO SA	ES
COSMOTE KINITES TILEPIKOINONIES AE	EL
EIGHT BELLS LTD	CY
INQBIT INNOVATIONS SRL	RO
FOGUS INNOVATIONS & SERVICES P.C.	EL
L.M. ERICSSON LIMITED	IE
SYSTEMS RESEARCH INSTITUTE OF THE POLISH ACADEMY OF SCIENCES IBS PAN	PL
ICTFICIAL OY	FI
INFOLYSIS P.C.	EL
PRODEVELOP SL	ES
EUROGATE CONTAINER TERMINAL LIMASSOL LIMITED	CY
TECHNOLOGIKO PANEPISTIMIO KYPROU	CY
DS TECH SRL	IT
GRUPO S 21SEC GESTION SA	ES
JOHN DEERE GMBH & CO. KG*JD	DE
CLOUDFERRO SP ZOO	PL
ELECTRUM SP ZOO	PL
POLITECNICO DI MILANO	IT
MADE SCARL	IT
NAVARRA DE SERVICIOS Y TECNOLOGIAS SA	ES
SWITZERLAND INNOVATION PARK BIEL/BIENNE AG	CH

Disclaimer

Authors

Name	Partner	e-mail
Prof. Carlos E. Palau	P01 UPV	cpalau@upv.es
Ignacio Lacalle Úbeda	P01 UPV	iglaub@upv.es
Rafael Vaño	P01 UPV	ravagar2@upv.es
Raúl San Julián	P01 UPV	rausanga@upv.es
Christos Xenakis	P10 IQB	chris@inqbit.io
Ilias Politis	P10 IQB	ilias.politis@inqbit.io
Panagiotis Bpountakas	P10 IQB	Panagiotis.Bpountakas@inqbit.io
Katarzyna Wasielewska-Michniewska	P13 IBSPAN	katarzyna.wasielewska@ibspan.waw.pl
Maria Ganzha	P13 IBSPAN	maria.ganzha@ibspan.waw.pl
Marcin Paprzycki	P13 IBSPAN	marcin.paprzycki@ibspan.waw.pl
Tarik Taleb	P14 ICT-FI	tarik.taleb@ictficial.com
Tarik Benmerar	P14 ICT-FI	tarik.benmerar@ictficial.com
Michalis Michaelides	P18 CUT	michalis.michaelides@cut.ac.cy
Herodotos Herodotou	P18 CUT	herodotos.herodotou@cut.ac.cy
Oscar Lopez	P20 S21Sec	olopez@s21sec.com
Saioa Ros	P20 S21Sec	sros@s21sec.com
Jon Egaña	P20 S21Sec	jegana@s21sec.com
Rafael Borne	P20 S21Sec	rborne@s21sec.com
Pablo Patús	P26 NASERTIC	ppatusdi@nasertic.es

History

Date	Version	Change
20/01/2023	0.1	TOC definition
03/04/2023	0.2	Compilation of partners' contributions in the first consolidated version
10/04/2023	0.3	Correction in the first consolidated version
05/05/2023	0.4	Compilation of partners' second contributions
12/05/2023	0.5	Peer review ready version
26/05/2023	1.0	Final version

Key Data

Keywords	DevPrivSecOps methodology specification
Lead Editor	P20 – S21SEC
Internal Reviewer(s)	P13 – IBSPAN (Katarzyna Wasielewska-Michniewska) // P05 – SIEMENS (Korbinian Pfab)

Executive Summary

This deliverable D2.4 presents the first version of the DevPrivSecOps methodology being designed within the aerOS project. This methodology will allow the internal developers of the project to bring the right practices in the lifecycle of the software developed during the project. Especially, and considering that the DevOps methodology is known and applied by most of the developers, in this first version the efforts have been directed to the implementation of security and privacy within the DevOps methodology.

First, an analysis was conducted on how security should be included in different phases of DevOps, analysing the specialities of each of these and presenting the most advanced options in the state of the art. This inclusion of security develops the DevSecOps methodology where the main intention is to guide in the development of secure software by design.

In response to the growing emphasis on privacy considerations, aerOS proposes the integration of privacy controls into the existing DevSecOps methodology, leading to the development of DevPrivSecOps. This novel approach aims to advance beyond the current industry standard and will incorporate privacy requirements, enabling aerOS developers to design software that is both secure and privacy-conscious from the beginning of the process. The document primarily focuses on exploring various privacy techniques and their potential application to the project, while the forthcoming version (D2.5) will provide specific guidelines for integrating privacy into the software development lifecycle.

In addition to introducing the DevPrivSecOps methodology, this document has presented the tools that will be used within the project in order to implement this methodology.

Finally, a guide has been created to help aerOS developers to implement the designed methodology.

This methodology is intended to teach, not only aerOS developers, but also the entire software development landscape how to implement it. For this reason, the deliverable is public and anyone interested will have access to it. In D2.5 we will analyse in concrete terms how this methodology has been applied during the aerOS project by analysing the good practices and the challenges it has.

.

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CPD	Continuous Planning Design and development
CSPM	Cloud Security Posture Management
CTF	Capture the Flag
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevPrivSecOp	Development, Privacy, Security and Operations
DevSecOps	Development, Security and Operations
DFD	Data Flow Diagram
DoS	Denial of Service
DORA	DevOps Research and Assessment
DPIA	Data Protection Impact Assessments
DRM	Digital Rights Management
DSAST	Deep Static Application Security Testing
ENISA	European Network and Information Security Agency
GDPR	General Data Protection Regulation
IaC	Infrastructure-as-a-Code
IDE	Integrated Development Environment
IAST	Interactive Application Security Testing
IT	Information technology
KSPM	Kubernetes Security Posture Management
LINDUNN	Linking, Identifying, Non-repudiation, Detecting, Data Disclosure, Unawareness, Non-compliance.
OWASP	Open Web Application Security Project
PASTA	Process for Attack Simulation and Threat Analysis
PIA	Privacy Impact Assessment
PPI	Personally Identifiable Information
SCA	Software Composition Analysis
SAST	Static application software testing
SCA	Software Composition Analysis
SDLC	Software Development Lifecycle
TLS	Transport Layer Security

VAST	Visual, Agile, and Simple Threat modelling
VCS	Version Control Systems

1. About this document

The main objective of this document is to specify a preliminary DevPrivSecOps methodology that will be used in the development and the operation of all the software created (or used) in the aerOS project. The aim of this methodology will be to generate security and privacy by design software. This methodology will also take into account the infrastructure-as-a-service paradigm that will be developed, and also how this will be operated. The main intention of the D2.4 would be to present the defined DevPrivSecOps methodology to the consortium and also to elaborate the guidelines for the implementation of the methodology, together with the tools that will be used in the project.

1.1. Deliverable context

Item	Description
Objectives	This deliverable is directly related with the objective "O3 Definition and implementation of decentralised security, privacy and trust". With this deliverable (D2.4) it is expected to create a cookbook and a good practices manual for the DevPrivSecOps implementation (KVI-3.3). As this methodology will be used as a baseline of all the software developments and operation in the project, will be implemented to reach the objectives O2, O4, O5 and O6.
Work plan	Deliverable D2.4 summarizes work done in the task T2.4. This deliverable brings together all the work done from month M3 to month M9. In this period of time a first version of the methodology has been defined. From the delivery of this document until the month M21, the methodology will be improved, adapting it to the needs that arise in the project in this period and making the necessary adjustments after receiving the feedback from the other workpackages.
Milestones	The submission of deliverable D2.4 is directly related to the completion of milestone MS2: Use cases and requirements defined. With the submission of D2.4 together with D2.2, the MS2 is fully achieved. 2 | Use cases and requirements defined | WP2 | 1-UPV | First iteration of use cases and requirements (D2.2a) | 9
Deliverables	This deliverable takes as input the already delivered D2.1 and the D2.2 that is delivered at the same time. Moreover, a second version of this deliverable, D2.5, is expected to be released for the month M21, where the defined DevPrivSecOps methodology will be updated.

1.2. The rationale behind the structure

Deliverable D2.4 has been structured as follows:

- **Section 2:** In this section a short introduction has been made to justify the need to create the DevPrivSecOps methodology for the aerOS project.
- **Section3:** This section introduces the DevOps methodology, which is the basis of the methodology presented in this document.
- **Section 4:** This section has analysed how security is introduced to move from DevOps to DevSecOps methodology, thus achieving a methodology that allows secure by design software development.
- **Section 5:** Within this section, the reader will find an in-depth analysis of the DevSecOps methodology, with particular emphasis on how it is implemented and how security testing is introduced in each of these phases in order to achieve a secure by design applications.

- **Section 6:** In this section, how privacy can be included in the DevSecOps methodology to turn it into DevPrivSecOps is described.
- **Section 7:** In this section a first analysis of how to include privacy in the software development lifecycle is made. This section is one of the most important sections of the document, as it goes beyond the state of the art. It presents the first steps that have been taken to design the DevPrivSecOps methodology.
- **Section 8:** This section, together with the previous one, form the core of the document and of the methodology defined in it. It specifies how the DevPrivSecOps methodology will be implemented within the aerOS project.
- **Section 9:** This section presents the first version of the toolset used within the project to implement the defined methodology.
- **Section 10:** In this section a guide for developers and integrators of aerOS developments has been included so that they can comply with the methodology described in the document.
- **Section 11:** Finally, this section presents conclusions and future work related to task T2.4.

1.3. Outcomes of the deliverable

- Definition of the preliminary version of the DevPrivSecOps methodology.
- Preliminary guidelines for developers and integrators.
- First toolset to implement the DevPrivSecOps methodology.

1.4. Version-specific notes

This document represents the first version of a series of two deliverables describing the DevPrivSecOps methodology designed by aerOS. This first version has been based on the state-of-the-art knowledge at the moment of writing. The second version is expected to be influenced by the feedback from work in workpackages WP2,3,4 related to the aerOS architecture and software components.

An update of this methodology will be presented in D2.5 (M21), adding the needs that have been identified during the design and implementation of the project. By its delivery date it is expected to have defined the final architecture of the aerOS edge-cloud-continuum (will be described in the deliverable D2.6 and D2.7) and its components.

2. Introduction

Recent years have seen the increasing need for data privacy. Several regulations have been published related to the data protection. Specifically, in Europe the main regulation for the data protection is the General Data Protection Regulation (GDPR), published in 2016 [1]. This is now widely regarded as a privacy law not just for the EU [2], but for the world.

This standard should be applied to all environments where data is used Although in the development of the SW the developers do not use any private data, they need to ensure that once the SW is deployed, the operation complies with the previously mentioned standard. This adds an important requirement for software developers.

Different analyses have identified that privacy and software security were silos in the software development lifecycle [3] [4]. Currently the responsibility lies with a specific group, in this case security and privacy experts, and not with all developers involved in the implementation of the software. In this context, iterations in development can increase risk as privacy security experts cannot review every change that is deployed quickly [5] [6], and sprints often do not plan for security issues [7].

Analysing software development showed that the measures to implement privacy in the lifecycle of software development are applied at the end of this process. But in several analyses, it has come to light that this is not enough. Privacy, in the same way as security in DevSecOps, must be considered from the beginning of software development and analysed at each stage of development, thus ensuring that developments are free of privacy problems. This need leads to the evolution of the software development methodology from DevSecOps to DevPrivSecOps. The aim of this is to increase the security and privacy knowledge of developers, testers, and operations staff and increase the partnership of privacy and security experts.
To create a suitable DevPrivSecOps methodology that meets the needs of the aerOS project, the first step is to analyse which software components and with what technologies will be developed in the project, what data will be used within the project and finally where the developed software components will be deployed.

In order to be able to analyse these needs, Figure 1 depicts dependencies of task T2.4, where the methodology is developed with tasks from where the input is collected, and workpackages this methodology will be implemented.

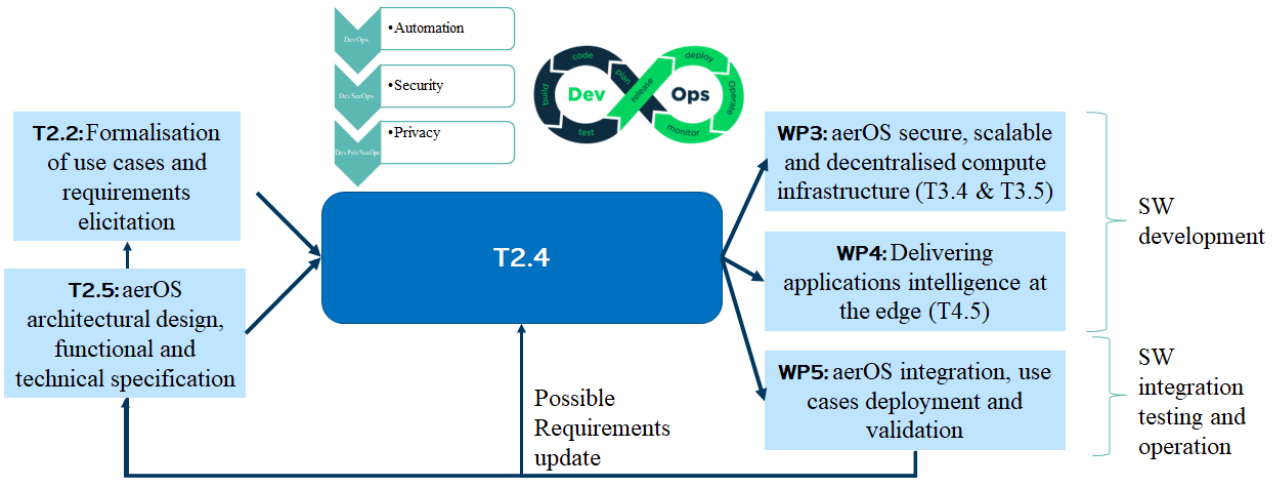

Figure 1 T2.4 relations with other aerOS tasks and workpackages

On the one hand, the task receives input from tasks T2.2 where the requirements of each use case are analysed and T2.5 where the aerOS architecture and the technical and functional specifications are described. These two tasks allow us to have the vision of the components that will be developed, the data that will be used by these components and where they will be deployed. Among other things, these entries allow us to start analysing the privacy status of the data to be used, the security status of the architecture and consequently, which components can be used to implement the DevPrivSecOps methodology and which components should be added.

Once the inputs are collected, in task T2.4 the DevPrivSecOps methodology is designed. This methodology will serve as the basis for the developments expected to be performed in WP3 and WP4 and in the integration and testing to be performed in WP5.

The methodology presented in this deliverable has been designed with the intention of fulfilling objective **O3: Definition and implementation of decentralised security, privacy and trust.** To this end, different methods have been analysed to ensure that the developments carried out within the project are secure and have privacy by design. In addition, a procedure is presented to ensure that the design, development and deployment of the software is done in a secure and privacy-compliant manner. Finally, the toolset that will be used in aerOS to achieve this objective is presented.

It is intended to create a dynamic methodology, which can be adapted during the lifecycle of the project, with the intention of covering the needs of this. Therefore, and through the feedback from WP3, WP4 and WP5, it is expected that modifications to the methodology will be introduced. It should be noted that the inclusion of privacy in the software development lifecycle is innovative, and for this reason, it is also intended to continue investigating new methods to analyse privacy and implement corrective actions during the duration of the task T2.4. With this, it is expected to have an improved version of the methodology for the month M21 of the project.

3. DevOps Methodology

The term DevOps is a combination of the two terms development and operations, representing a collaborative approach to the tasks that are performed by development and IT operations teams. DevOps is a collection of several tools and practices that help automate and integrate the processes between IT professionals and software. It further focuses on team collaboration, team empowerment, cross-team communication, and technology automation. DevOps can co-exist with IT service management frameworks, Agile software development, project management directives, and other IT infrastructures. DevOps can be simply defined as *a methodology that helps optimizing the process of software development and operation* [8].

3.1. DevOps definition

DevOps is a well-known methodology that is adopted to enhance the work throughout the software development lifecycle. The DevOps process can be visualized as an infinite loop, comprising multiple steps, e.g., plan, code, build, test, deploy, release, operate, monitor, and feedback-based plan, which resets the loop. To align software with expectation, stakeholders and developers communicate about the project, and developers work on small updates that go live independently of each other [9].

To avoid waiting times, IT teams employ continuous integration/continuous deployment pipelines and other automation methodologies in order to move between different steps in SW development. To ensure a seamless deployment to production, DevOps often use containers as execution environments or other similar methods to ensure the same behaviour of software from development to testing and from testing to production. The changes are deployed individually, so any issues can be easily traced. IT teams rely on configuration management for consistent deployment and hosting environments. Issues they discover in live operations lead to code improvements, often through a blameless post-mortem investigation and continuous feedback channels. IT operations administrators are involved in the software design meetings and provide guidance on how to use resources efficiently. Furthermore, in blameless post-mortems, anyone can contribute to software development and deployment process. The strong collaborations among experts and professionals fosters the DevOps culture.

The main DevOps advantages are:

- Efficient communications between IT teams,
- Fast marketing time for software,
- Easy improvements based on feedback,

- Less downtime,
- Less manual work,
- Improvement to the entire software delivery pipeline through builds, validation, and deployment,
- Broader rules and skills, and
- Streamlined development processes through increased responsibility and code ownership in development.

3.2. DevOps guidelines

DevOps can be divided in eight phases and every phase can be repeated several times across the SW development and deployment until it is completed [8] [10]. The fundamental goal is to create an environment where the code can move quickly through the pipeline while maintaining high quality standards.

Continuous improvement: This is crucial for determining the overall strategy for the software development lifecycle. It is based primarily on program design and programming. Program needs are identified and discussed with partners. In addition, for continuous software development, the backlog is managed depending on customer response and divided into shorter versions and targets.

Continuous integration: One of the most important features of the DevOps lifecycle is ongoing integration. There, updated programming, additional functions, and some additions are created and integrated into the current code. Moreover, bugs in the program in each phase are detected and tracked down by the testing phase and the source code is updated accordingly. This transforms the integration into a continuous process where the code is evaluated after each commit. In addition, the necessary tests are planned during this period. Jenkin, GitLab CI, and Circle CI are some of the DevOps tools used to make the programming process more efficient.

Ongoing testing: Quality testers use tools like Docker instances to test the software for bugs and issues. In the event of a bug or error, the program is sent back to the integration phase to be fixed. Automated screening also reduces resources required to produce high-quality results. At this stage, companies use tools such as Selenium. In addition, test automation improves test assessment reporting and reduces the cost of deploying and maintaining the test environment. Selenium is perhaps the most widely used open-source automated testing software that is compatible with various platforms and devices.

Continuous deployment: After the testing is completed, the finished program is released to operational servers, which is of utmost importance and is integrated into the DevOps lifecycle. Configuration control is part of continuous deployment to ensure that deployment runs correctly and smoothly. The Ansible and Chef setup management tools are used to ensure a seamless and constant workflow throughout the production process.

Feedback regularly: Constant feedback is created to examine and improve the program code. User behaviour is analysed for each distribution to improve upcoming updates and implementations. Organizations can solicit advice in either a controlled or an uncontrolled manner. Surveys and quizzes are used to solicit feedback on the structural approach. In an uncontrolled manner, input is gathered via social media. Ultimately, this procedure is critical to enable continuous deployment to introduce better output from the program. Pendo is a business monitoring platform that collects and analyses consumer feedback. An additional tool, TED from Qentelli, tracks the entire DevOps workflow and gathers valuable information about bugs and vulnerabilities.

Continuous monitoring: The operation and attributes of the application are periodically checked to detect network errors such as a low RAM, an unreachable host, and so on. The process helps IT staff immediately to identify application execution issues and underlying causes. When IT teams discover a serious issue, the app is re-run through the DevOps lifecycle to find a solution. However, at this stage, security vulnerabilities can be identified and fixed immediately. TestNG, Selenium, JUnit, and TestSigma are some of the DevOps tools used for continuous monitoring.

Operational continuity: Lastly in the DevOps lifecycle, it is critical to reducing planned downtime, such as planned maintenance. To make the changes, programmers typically need to shut down the system, which increases downtime and can result in a large loss to the business. Finally, continuous operation simplifies the process of starting and updating the application. In addition, interruptions are avoided by using unit management solutions such as Kubernetes and Docker.

4. Evolution from DevOps to DevSecOps

The introduction of DevOps culture in software production teams led to much more agile software developments and to a higher frequency of deployments. However, the security of the developed software was not taken into consideration within the DevOps continuous pipeline. Instead, security was added in the final phases of development, totally uncoupled from CI/CD pipeline. Security was only considered right before deploying the service to production.

Having the security incorporated at the end of development of a piece of software is detrimental to the philosophy that DevOps pursues. This is the case because in order to deploy a secure service, prior to its release the software must be analyzed searching for potential vulnerabilities [7]. Fixing these potential threats implies the modification of the original source code, rewriting the insecure snippets to adopt security good practices that were not considered right from the start. This brings the whole continuous DevOps pipeline to a halt, as instead of proceeding with deployment the team must first iterate the development until achieving a secure application for production. This poses a bottleneck in the development pipeline that misses the point of agile developments and deployments that DevOps brings to the table.

Incorporating security practices into development and operations from the starting point eliminates the necessity of implementing security features at the end of the build, hence removing the bottleneck that this causes in the CI/CD pipeline [7]. Dealing with security from day one not only makes development more agile, but also prevents accumulating security flaws overtime, which makes the implementations much safer compared to those that would be deployed if security were added at the end of the build. This also leads to an improved cost-efficiency when tackling unexpected security threats [7].

The consideration of security features from the start of development is commonly referred to as “security-by-design”. Incorporating security-by-design within the DevOps continuous pipeline leads to the evolution from DevOps to DevSecOps. Thus, DevSecOps can be conceived as an improvement of DevOps, where security is pondered in all phases of the continuous lifecycle.

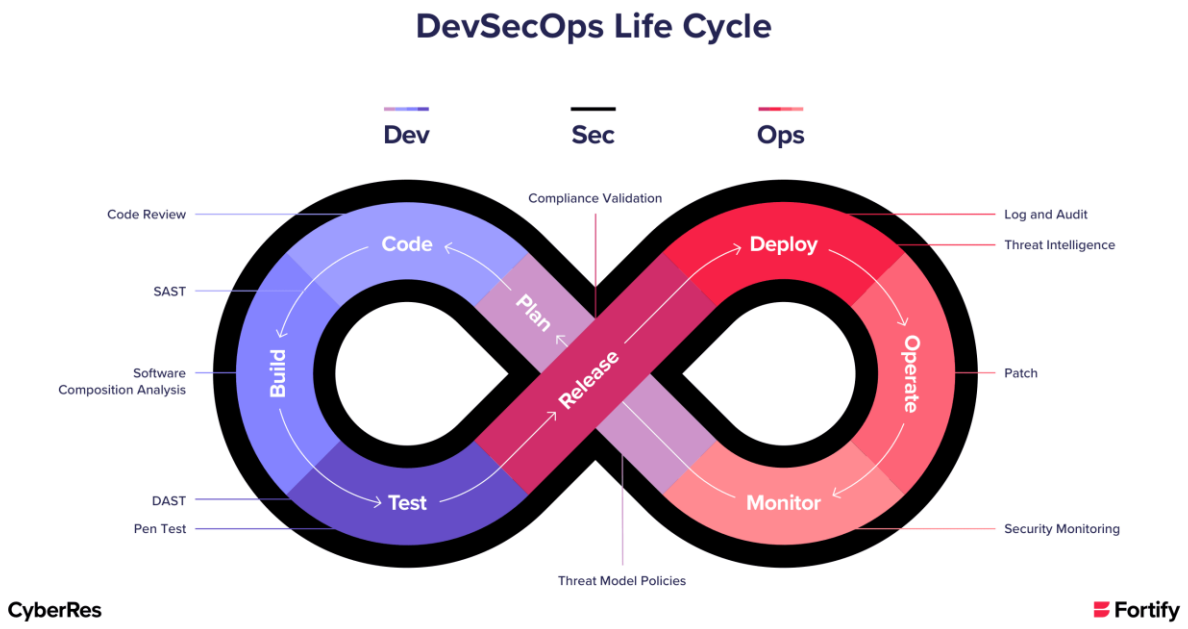

Figure 2. DevSecOps lifecycle [11]

Figure 2 depicts all the different courses of action integrated into the standard DevOps phases to tackle security. This means that security concerns now bring to the table further automated tests and monitoring tasks to be included in the usual DevOps pipeline, resulting in the DevSecOps lifecycle. Some of the procedures that

DevSecOps incorporates in its pipeline [12] involve the use of plugins while coding to detect vulnerable snippets in advance, automatic code analysis (looking for exploitable security breaches in the build) and threat modelling, among others. Section 5 offers a highly detailed description of all the procedures taking place within each of the DevSecOps' phases.

5. DevSecOps Methodology

DevSecOps means thinking about application and infrastructure security from the beginning and also embedding DevOps with security controls providing continuous security assurance [13]. DevSecOps is a natural extension of DevOps to include security-by-design and continuous security testing by automating some security controls in the DevOps workflow. Figure 3 presents how DevSecOps embeds security controls across the DevOps lifecycle phases.

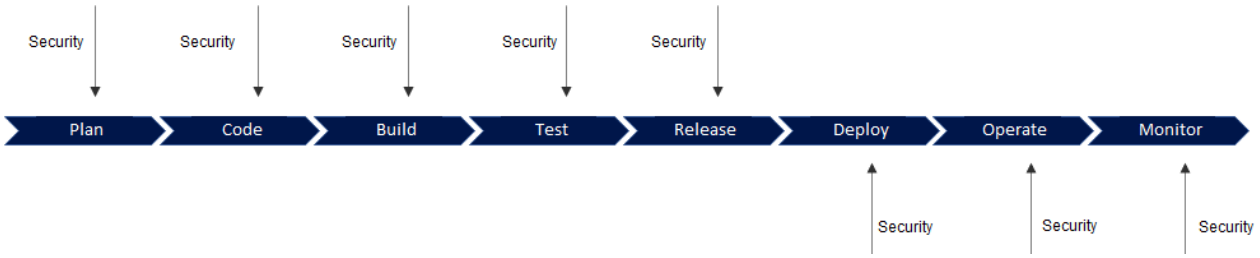

Figure 3 DevSecOps pipeline

The core concept of DevSecOps is that each stage is responsible for security. Developers must incorporate it into all facets of code and specifications. Code quality assurance professionals must test for security in addition to functionality. Finally, operations teams must monitor software behaviour and respond quickly to problems.

5.1. DevSecOps Principles

To start implementing DevSecOps, five key principles must be followed:

- deliver small, frequent releases using agile methodologies,
- wherever possible, make use of automated testing,
- empower developers to influence security changes,
- ensure a continuous state of compliance,
- be prepared for threats.

The DevSecOps principles are a set of guidelines for providing the foundation for creating different security controls in the DevSecOps continuous security model [14].

- **Culture Communication, Collaboration and Sharing**
  - The DevSecOps is a techno-cultural transformation that requires mind shift of the development, operations, and security teams to collaborate, communicate and share information to deliver security ready applications with velocity and agility.
- **Automation**
  - Automation is the backbone of DevSecOps workflow implementation and enables the implementation of DevSecOps principles and practices.
- **Metrics, Measurements and Quality Assurance**
  - Metrics for performance and quality measurement for an automated delivery flow (i.e. agility, velocity, security, and quality).
- **Shift Security Left**
  - Shifting security to left advocates building security controls into the applications at earlier stages of the development cycle.
- **Security-by-Design**

- SbD is an approach to system implementation that focuses on minimizing the vulnerabilities and reducing the attack surface of the system through designing and building security controls at every stage of the system implementation.

- **Security-as-Code**
  - SaC is about implementing security checks and controls into the workflow through codes.
- **Infrastructure-as-Code**
  - IaC treats infrastructure, both physical servers and virtual resources, as programmable unit and uses software development approach for their provisioning and configuration.
- **Compliance-as-Code**
  - CaC advocates using code, to define, implement and validate security policy and controls in the workflow.
- **Adaptative Security**
  - An adaptive security system does not wait for incident to happen but anticipates before it happens and acts proactively to prevent system from any possible security breach.

5.2. DevSecOps Workflow

In this section, each of the DevOps phases are presented and it is explained how they are modified so that the methodology conforms to DevSecOps principles [13].

5.2.1. Plan

The planning phase is the first automated phase of DevSecOps, and involves collaboration, discussion, review and strategy for software development and deployment with security in mind. A security analysis must be performed and a plan created that describes where, how and when security testing will be performed. In this phase, threat modelling is done, analysing the entire software development lifecycle, with the intention of detecting security threats in all phases. Although a plan is designed at the beginning of the cycle, it must be adapted to the needs that arise throughout the development lifecycle.

5.2.2. Code

Through the use of plugins installed in code programming environments, code reviews, static code analysis and precommit hooks should be performed. This set of plugins helps developers apply good coding practices while encouraging collaboration by bringing some consistency to the code.

These technologies support a variety of integrated development environments and many programming languages.

5.2.3. Commit

In this phase, the code is registered in a local repository where it is automatically version controlled.

In the commit phase, code review and verification of security guidelines, source code version control, unit and integration security testing and software composition analysis (SCA) are performed.

5.2.4. Build

When the code is submitted to the source code repository by the developers, the build phase begins. The build operation is performed by automated scripts that make such transformation (e.g., dockerfile to convert a codebase into a docker image), combining source code into runnable code. At the same time, this triggers an automated process that builds the code base and runs a series of end-to-end, integration and unit tests to identify any regressions. An automated security analysis of the compilation output artifact is the main focus of the DevSecOps compilation tools. Static application software testing (SAST), unit testing and software component analysis are crucial safety procedures that are performed automatically in this phase.

External code dependencies used when developing code can unintentionally or maliciously imply vulnerabilities and exploits. Therefore, reviewing and testing these dependencies for possible security flaws during the development phase is crucial.

5.2.5. Integrate

The integration of the application software components as a single system, including consideration of the hardware infrastructure, characterises this stage. System integration and security testing is carried out to look for issues that render the integration invalid, including the integration of any third-party software. Infrastructure configuration codes are validated against application requirements, verifying that everything works as expected. DevSecOps security controls for the integration phase include container and infrastructure analysis as code (IaC), software composition analysis (SCA), static application security testing (SAST), dynamic application security testing (DAST) and fuzzy testing.

5.2.6. Package

The different functionalities developed will be packaged together with release notes, installation and configuration scripts or instructions, resulting in a packaged executable code. An artifact repository security management will be implemented to control the security when packaging the software.

5.2.7. Release

The packaged application is delivered to a staging environment that is a replica of the production environment and the expected behaviour of the application in production is tested. In this environment, user acceptance and security testing and security management of the artifact repository is analysed. This focuses on the analysis of the protection of the architecture of the execution environment by reviewing the configuration settings of the environment, including user access control, network firewall access and personal data management. This includes a combination of checking access tokens and API keys to limit owner access.

5.2.8. Configure

All builds arriving at this point will have passed a SAST, a DAST and the usual checks and tests, and the artifacts will be (signed) and ready in the shared repository of the project/organisation.

In this phase, the application installed in the test environment is configured with the configuration data required for acceptance testing.

The code awaits final validation and release to a near-production environment/repository.

5.2.9. Accept

In this phase, functional and non-functional tests are carried out on the software, validating that it complies with the defined characteristics and that the performance is as expected. The results of these tests are analysed and communicated to the development team so that the necessary improvements can be made in the next iteration. In addition, different security tests such as penetration testing, DAST, Deep SAST, fuzz testing and security smoke testing are performed to identify and rectify any security breaches. In case all tests are passed successfully, the SW is accepted for deployment.

5.2.10. Deploy

The accepted application is deployed to the production environment and smoke tests are performed to identify functional and non-functional problems. Because security has been tested in all phases, problems are not expected in this phase, as they are more costly to fix in this phase than in previous phases.

Security issues that only affect the live production system should be addressed during deployment. For example, it is essential to carefully examine any configuration variations between the current production environment and the initial test and development configurations. In addition, production TLS and DRM certificates should be reviewed and validated in preparation for their upcoming renewal.

The deployment phase is a good time to collect data from a live system to assess whether it is functioning as intended. Chaos engineering principles can also be applied by testing a system to increase the system's confidence in its resilience to turbulence. Real-world events, such as hard drive failures, network connection losses and server crashes, can be replicated.

Security controls related to secrets management, infrastructure provisioning and orchestration, security patching, infrastructure hardening and security testing, and container and infrastructure security testing are carried out in this phase.

5.2.11. Operate

In this phase, operations staff usually perform regular maintenance tasks. Deployed applications and the production environment are continuously monitored for performance and malicious activities allowing stakeholders to verify whether the deployed security controls are working as expected and to take the necessary actions to correct deviations. Especially regarding security, zero-day vulnerabilities should be monitored, and mitigation methods implemented as soon as possible so that they do not affect the system. DevSecOps can use Infrastructure-as-Code (IaC) tools to protect the organisation's infrastructure quickly and effectively with patch updates, while preventing human error from creeping in.

Red teams (the team that acts as an adversary, attempting to identify and exploit potential weaknesses within the organization's cyber defenses using sophisticated attack techniques) [15] and bug bounties are essential in the operational phase for creating a feedback loop between security teams and developers. Some of the security controls applied in this phase are as follows: application and system logging, continuous monitoring and alerting, intrusion prevention detection and response, security incident management, security metric measurement and analysis, security audit and compliance, penetration testing, dynamic application security testing (DAST), fuzzy testing, interactive application security testing (IAST), run-time application self-protection, continuous vulnerability scanning, security smoke testing, infrastructure hardening and security testing, container and infrastructure security testing, red, blue and purple team testing and monkey testing.

5.2.12. Adapt

When security is constantly monitored for anomalies, a breach can be avoided. Therefore, it is essential to have a continuous monitoring tool that operates in real time to monitor system performance and detect any failures at an early stage. In addition to the ability to monitor, systems require, in order to recover from an attack, the ability to scale the infrastructure on demand and to replace a compromised environment in a matter of minutes by boosting adaptability, which is provided by the use of the right set of automation tools.

5.3. DevSecOps Practises

DevSecOps is a convergence of development, security and operations as an organisational pattern to adopt security throughout the whole software development lifecycle (SDLC). Specifically, it is the seamless integration of security processes into the development and delivery pipeline [16] [17] . Practices to help ensure effective DevSecOps implementation include [18] [19]:

- **Shifting security left** - moving security related processes from the end of the delivery process to the beginning placing security at the start of the development lifecycle, requiring software and security engineers to collaborate early on in the SDLC.
- **Security training** - all teams involved in the software delivery should be familiar with basic application security principles, security testing, and software engineering best practices. Developers must understand threat models, compliance assessments and should be able to identify and measure security risks and exposures as well as to apply security controls. The goal of aforementioned is to promote a culture of security testing in an organisation.
- **Implement continuous integration and continuous delivery** - automatically build, test, and deploy code changes to ensure that they are integrated and delivered quickly and efficiently reducing the risk of human error.

- **Automate security scanning and implement security scanning governance** - adopting the approach of automating everything that is related to CI/CD pipelines and implementing policies to enforce and document usage of selected solutions. The goal is to keep pipelines fast and productive also by leveraging tools such as, e.g. SAST, DAST, and AppSec.
- **Add security gates or guardrails** - it is not only important to use the tools but also analyse the output of such scanners.
- **Threat modelling** - the process of identifying, quantifying, and prioritizing the risks to your systems and data using, e.g. STRIDE method.
- **Incident management** - workflows and action plans should be created in advance to ensure the response to an incident is consistent, repeatable, and measurable.
- **Red teams, blue teams, bug bounties** - red teams hunt for threads testing effectiveness of security, whereas blue teams defend against attacks. Bug bounties are rewards given for identifying a problem.
- **Security as code** - need to constantly check the coding standards against security recommendations.

5.3.1. CI/CD

One of the critical component of DevSecOps is continuous integration (CI) and continuous delivery (CD) pipelines used for automating the delivery process from code development to product deployment. Continuous integration builds and tests code automatically, while continuous delivery automates the entire software release process up to production. This all is actually part of a broader DevOps movement. The following are benefits of CI/CD: speed up the delivery, faster time-to-resolution for problems, reduction of a risk of errors and vulnerabilities and improving code quality. Moreover, DevSecOps addresses vulnerabilities in software development by inserting security audits and penetration testing at every step of the process.

In [20] by the Carnegie Mellon University's Software Engineering Institute, the following can be implemented to improve CI/CD pipeline safety:

- Strong physical access controls,
- Clear change management processes,
- Be able to attribute actions to individuals,
- Track security controls for each delivery,
- Compliance metrics,
- Security alerts,
- Automatic vulnerability fixes,
- Clear incident response procedures.

DevSecOps CI/CD pipelines should contain the following continuous security-oriented phases (besides traditional DevOps stages):

- **Threat modelling**: modelling risks, attacks and methods of their mitigation.
- **Security testing:** scanning, reviewing and testing code with SAST and DAST tools.
- **Security analysis and prioritisation**: analysing and prioritising vulnerabilities, detected based on information from previous phases, for remediation.
- **Remediation**: addressing discovered vulnerabilities.
- **Monitoring**: security monitoring of deployed workloads.

DevSecOps CI/CD tools include:

- **Static application security testing (SAST)** - scanning source code for issues such as common vulnerabilities from the OWASP.
- **Dynamic application security testing (DAST)** - scanning applications during runtime to detect security issues.
- **Interactive application security testing (IAST)** - combines SAST and DAST into a single solution.
- **Source composition analysis (SCA)** - identifying libraries and dependencies within an application and list the associated vulnerabilities.
- **Vulnerability scanners** - detecting configurations problems that can compromise security and compliance.

Examples of security tools for specific solutions include: Kubernetes Security Posture Management (KSPM) for container clusters and Cloud Security Posture Management (CSPM).

5.4. Security-by-design software development in DevSecOps

The idea of applying the different actions to monitor and ensure security in the workflow of the software development lifecycle (presented in the section 5.2), aims to create secure code by design.

In software development, "Security-by-Design" refers to the practice of designing and implementing secure products from the very beginning of a project. This approach emphasizes the importance of integrating security controls throughout the entire software development lifecycle, from the initial planning stages to deployment and maintenance. In the context of a DevSecOps environment, security-by-design is a fundamental principle. This means that every member of the development team is responsible for ensuring that the product is secure throughout the entire lifecycle of the project[21]. The aim is to produce products that can withstand modern cyberattacks and ensure the security of the user's data.

The Open Web Application Security Project (OWASP) has described a set of fundamental principles for achieving Security-by-Design [22]. These principles are:

- **Minimize attack surface area**: Reduce the number of ways an attacker can target the system.
- **Principle of least privilege**: Users, processes, and systems should have only the minimum privileges necessary to perform their functions.
- **Fail securely**: The system should be designed to fail gracefully and securely.
- **Secure defaults**: The system should be configured securely by default.
- **Separation of duties**: Different parts of the system should be designed and implemented by different teams to prevent a single point of failure.
- **Defence in depth**: The system should have multiple layers of security to prevent attacks.
- **Open design**: The security of the system should not rely on the secrecy of the design or implementation.
- **Privacy by design**: The system should be designed to protect the privacy of its users and their data.
- **Secure communication**: All communication between the components of the system should be secure.

6. Evolution from DevSecOps to DevPrivSecOps

New privacy regulations require companies to incorporate privacy principles by design, including the need to include privacy in software development [23].

In the same way that the idea of the DevSecOps methodology is to shift security to the left and that security analysis is done in each phase of the methodology, in DevPrivSecOps the idea is not only to analyse security, but also to add privacy analysis in the different phases of the software development lifecycle.

It should be noted that the inclusion of privacy analysis in the DevSecOps methodology goes beyond the state of the art. As discussed in article [24], the inclusion of privacy is a necessity and it is also necessary for DevOps teams to consider both security and privacy, with development and operations teams working together with security and privacy experts. In works such as [25] we can see how privacy analysis has been implemented in the software development lifecycle.

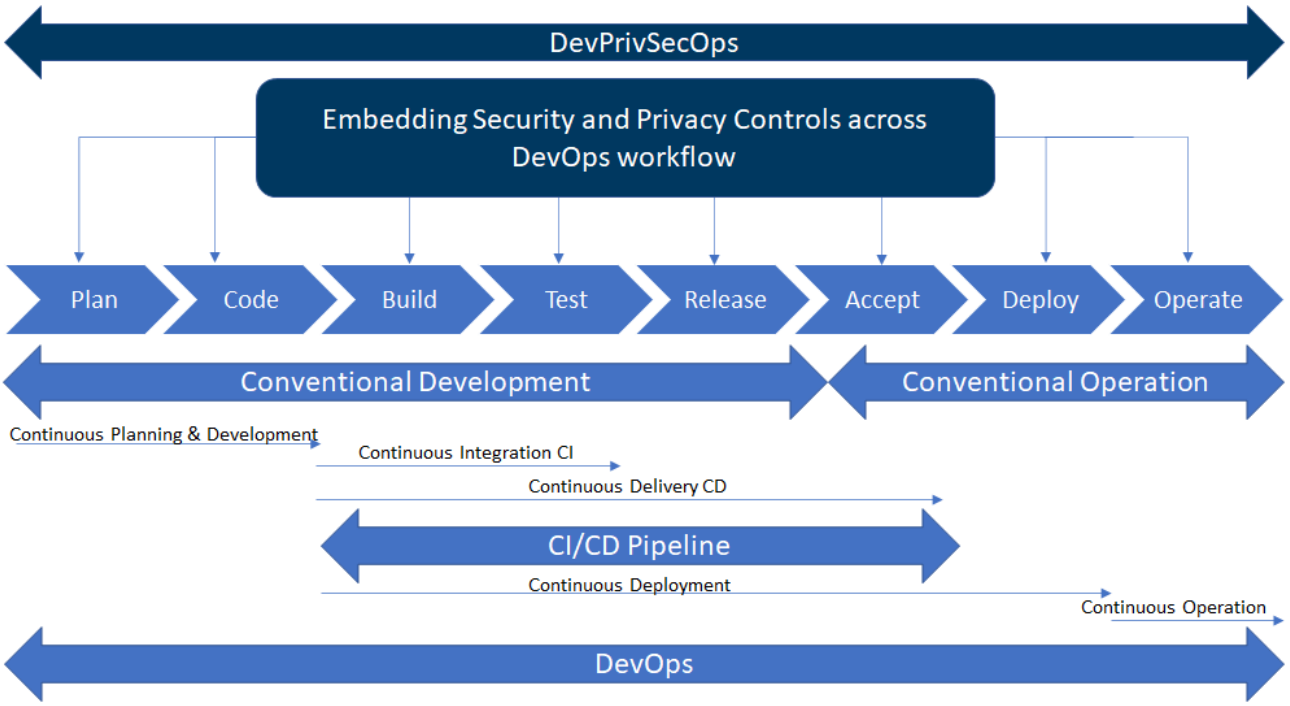

Figure 4 DevPrivSecOps software development lifecycle

7. DevPrivSecOps Methodology: Privacy threat analysis

Privacy threat modeling methodologies are used to identify and assess privacy risks associated with the processing of personal information. Here are some commonly used privacy threat modelling methodologies:

LINDDUN (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unlinkability, and Non-Compliance) [26]is a privacy threat modelling methodology developed by KU Leuven and admitted and included by the European Network and Information Security Agency (ENISA) to help organizations identify and address privacy risks associated with the processing of personal information[27].

Process for Attack Simulation and Threat Analysis (PASTA) [28] is a threat modelling methodology developed by the Open Web Application Security Project (OWASP) to identify and assess security and privacy risks associated with software applications. PASTA includes a structured approach for identifying and prioritizing potential threats to an application's security and privacy.

Visual, Agile, and Simple Threat modelling (VAST) [29] is a threat modelling methodology that focuses on visualization and agile techniques to identify and assess security and privacy risks associated with software applications. VAST includes a structured approach for identifying and prioritizing potential threats to an application's security and privacy, and it emphasizes simplicity and flexibility in the threat modelling process.

The Threat Representation and Knowledge Exchange (Trike) [30] methodology is a framework for threat modeling that was developed by the U.S. Department of Defense. It provides a structured approach for identifying and assessing potential threats to systems, including security and privacy threats. Trike emphasizes collaboration between stakeholders in the threat modelling process.

Privacy Impact Assessment (PIA) [31] is a privacy threat modelling methodology that is commonly used in Canada, Europe, and Australia. PIA is a process for identifying and assessing privacy risks associated with the processing of personal information. PIA includes a structured approach for identifying and assessing privacy risks and evaluating the effectiveness of privacy controls.

These are just a few examples of privacy threat modelling methodologies. Each methodology has its own strengths and weaknesses, and organizations may choose to use one or a combination of these methodologies based on their specific needs and objectives.

7.1. LINDDUN based privacy threat analysis

This chapter describes the methodology designed in aerOS to include privacy in software development. For this design, in this first version of the methodology, it has been decided to follow the steps of LINDUNN privacy threat modelling.

This threat modelling will allow us in the plan phase to analyse the needs for privacy inclusion throughout the software development lifecycle. To do so, the first step will be to analyse the possible privacy problems that the software to be developed may contain, analysing different weak points and also analysing the privacy problems of the scenario where this software is going to be deployed. Once the possible problems have been analysed, the idea is to be able to find solutions for them, with the intention that the development of the software is carried out considering them, thus minimising the privacy problems in the deployment phase. The search for possible problems and solutions is carried out in six phases, which can be seen in the following Figure 5:

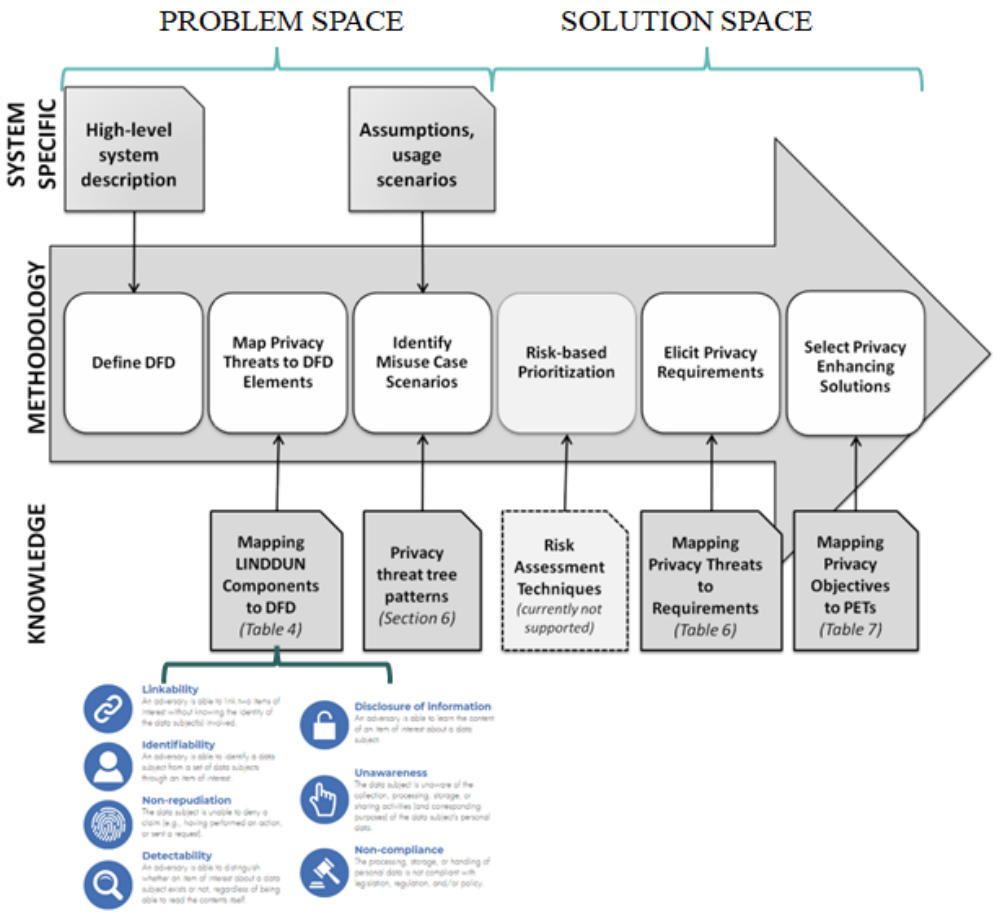

Figure 5 LINDDUN phases [26]

In this first definition of the DevPrivSecOps methodology, for the analysis and inclusion of privacy, aerOS has decided to follow the LINDDUN methodology. LINDDUN is a model-based approach that leverages a data flow diagram (DFD) as a representation of the system to be analysed by systematically examining it in depth for privacy threats. The methodology is also knowledge-based with an overview of the most common attack paths associated with the set of privacy threat categories contained in the LINDDUN acronym (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, Non-compliance). This methodology is based on representations such as threat trees that detail the possible causes of threats. Each tree presents the attack paths related to a threat category that applies to a particular type of DFD element (entity, data flow, data store or process).

7.1.1. System privacy threat modelling

The LINDDUN methodology proposes to follow the 3 phases shown in Figure 6: modelling the system, eliciting threats/risks and managing threats.

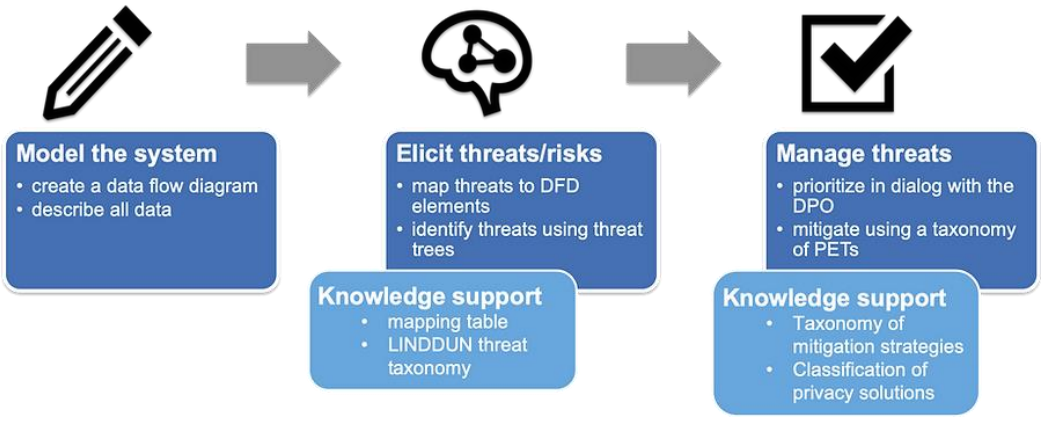

Figure 6 LINDDUN phases [26]

7.1.1.1. Model the system

In order to perform a good system modelling, it is first necessary to know very well the system to be modelled, the data used within it and how the data moves within the system (the data flows). LINDDUN uses a Data Flow Diagram (DFD) as a model to collect the most relevant knowledge of the system for privacy analysis.

A DFD is a structured graphical representation of the system using 4 main types of building blocks: external entities (i.e. users or third party services external to the system), data stores (i.e. passive containers of information), processes (i.e. computational units) and data flows (indicating how information is propagated through the system). In addition, trust boundaries can be used to indicate a logical or physical division of the system. In the Figure 7 can be seen an example of how to represent graphically a DFD.

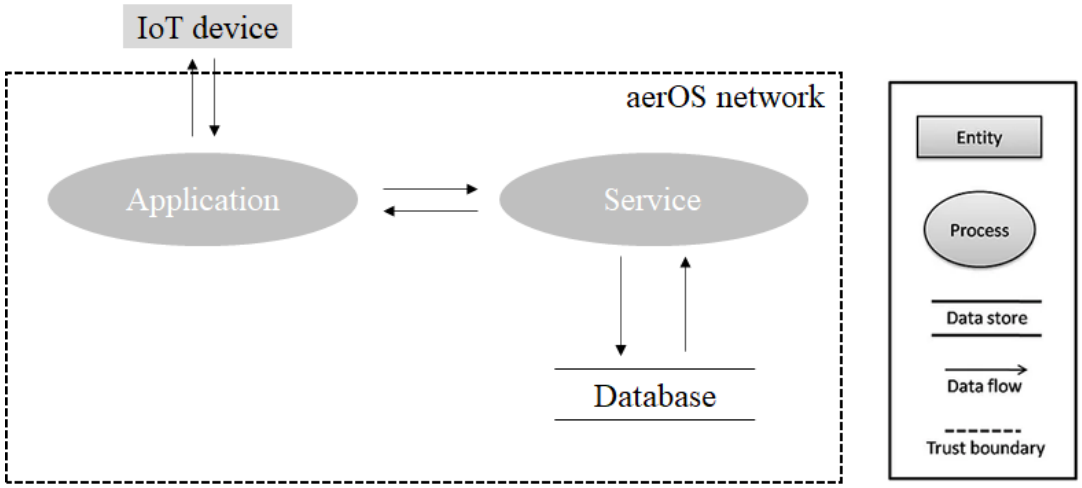

Figure 7 Graphical representation example of a DFD

7.1.1.2. Elicit threats/risks

Once the system is described, each DFD element should be systematically analysed for privacy threats.

Map DFD elements to threat categories

In this step a custom table is created based on the LINDDUN components for each software component where privacy will be analysed. In this table, a row will be created for each component (the components of the diagram created in the previous step) of the system to be analysed. Using this table, you check (by adding an X in the

corresponding square) whether the components may have a potential threat in one of the LINDDUN components. The Table 1 shows how the example described in the previous section can be mapped to the table.

DFD element	L	I	N	D	D	U	N
IoT device	X	X				X	
Application	X	X	X	X	X		X
Service	X	X		X	X		
Database		X	X		X		X

Table 1 LINDDUN mapping table

Elicit and document threats

For each of the Xs in the mapping table of the previous step, it is analysed to determine whether they pose a threat to the system. For this purpose, LINDDUN provides a set of privacy threat tree patterns. These trees represent the most common attack paths for a specific LINDDUN threat category and DFD element type.

LINDDUN threat trees can be found in the LINDDUN threat tree catalogue [32]. An example of a threat tree can be seen in the Figure 8.

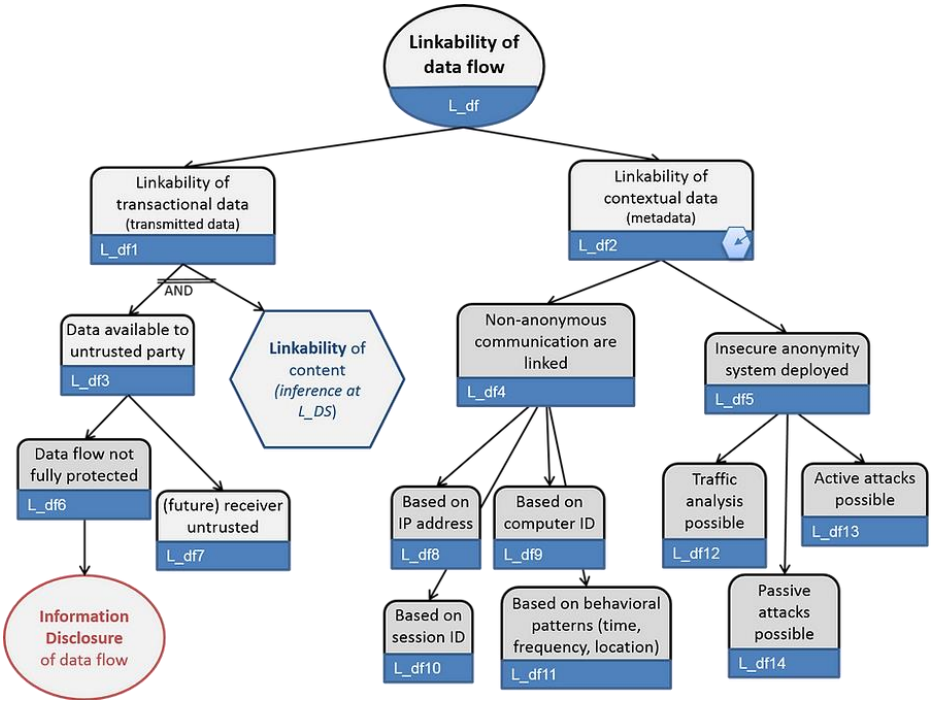

Figure 8 Threat tree example (Linkability)

The analyst must examine each branch of the tree with the specified DFD element in mind. Each applicable leaf node (or branch) is considered as a threat.

Document threats

Once threats have been detected, they should be documented by exposing all the information gathered during the phase of the analysis. This information will allow a third party to easily identify and understand the threats.

7.1.1.3. Manage threats

This section outlines how responses to the detected threats should be made with the intention of migrating these threats.

In the case where the system is running in production, and several privacy threats have been detected in the system, it is important to prioritise these threats by addressing the most critical ones at the earliest stage, thus reducing the damage that the system's privacy may suffer.

Each identified threat must be addressed. LINDDUN provides a taxonomy of mitigation strategies (Figure 9) to delimit the solution space. A mapping between the LINDDUN threat trees and the taxonomy of mitigation strategies is provided [33] by the methodology to facilitate the selection of an appropriate strategy.

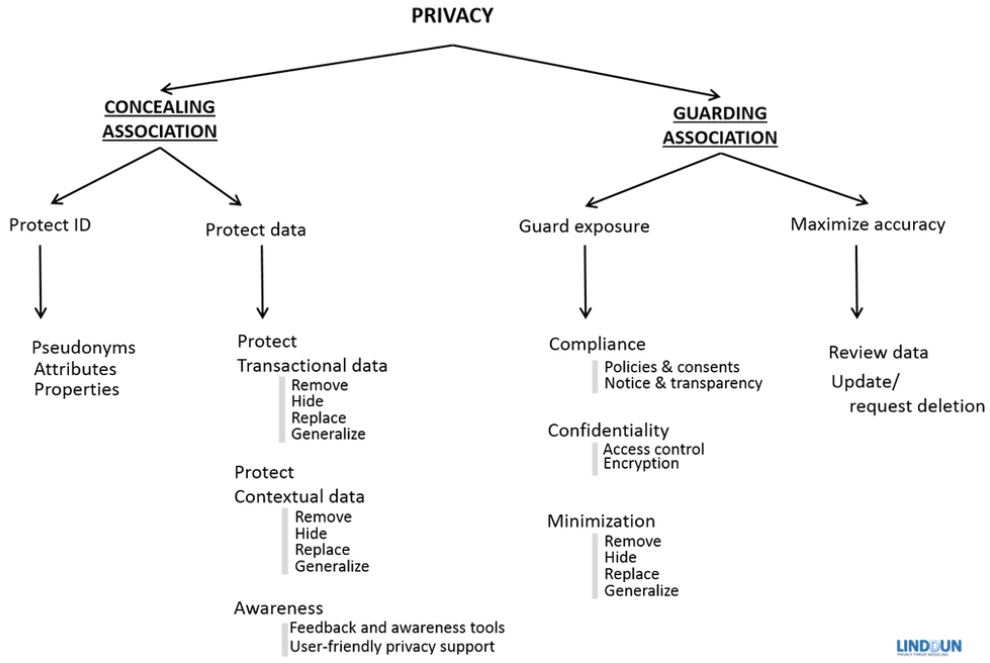

Figure 9 LINDDUN mitigation strategies taxonomy [33]

Finally, the classification of privacy-enhancing technologies according to the mitigation strategies they support allows for a more focused selection of appropriate privacy-enhancing solutions. If necessary, mitigation strategies can also be translated into privacy requirements, rather than directly applied as solutions.

7.2. Privacy threat management

The privacy threat management is a process deployed to prevent cyberattacks that aim to compromise the data privacy of a system, identify cyberthreats, respond to cybersecurity incidents, and apply recovery activities to minimize the risk of privacy-related cyberthreats [34]. When applied to aerOS, the DevPrivSecOps methodology will develop a privacy threat management strategy that will focus on tackling privacy-related risks. The management of potential risks to an individual's privacy will include five primary processes, which are outlined in the following:

- **Privacy threat identification**: The privacy threat identification refers to the process of identifying functions and processes during the DevPrivSecOps phase that have the potential to lead to privacy issues (e.g., data leakage). One example would be the identification of functions that deal with personally identifiable information.
- **Assessment of potential threats to privacy**: This process involves doing a risk assessment on the privacy risks that have been previously discovered, assigning a risk score to each potential risk, and then classifying each potential risk according to its risk score.
- **Privacy threat mitigation**: The privacy threat mitigation process applies privacy protection measures in the privacy risks that have been detected in the previous step. The following counter measures should be included into the privacy risk reduction strategy:
    - Privacy-by-design refers to the practice of including privacy and data protection strategies into the first stages of the system's development.

  - Privacy Policies: In case that personal data is handled, the privacy policy will reveal what personal information is gathered, how that information is obtained and deployed, and whether or not it is shared with any third parties.
  - Privacy controls are the application of measures that try to safeguard the privacy of personally identifiable information and sensitive data. For instance, by using various anonymization methods to the personal data, we may make it less identifiable.
- **Recovery activities**: The recovery activities create measures to guarantee the privacy resilience and continuation of the processes in the event that there is a cybersecurity incident (e.g., data breach). Upon the completion of this stage, the implementation and acceptance of alternative techniques that will be put into use following the occurrence of a cybersecurity event will be guaranteed.

8. DevPrivSecOps in aerOS

An innovation of aerOS is the introduction of the DevPrivSecOps methodology, where the DevSecOps operations are extended to also include privacy-by-design at the same level as security. The DevPrivSecOps methodology will assure fast and frequent development cycles that provide security and privacy by design ensuring agile long-term evolution and support for instantiation of aerOS architecture in the different demonstration use cases. This section elaborates on various aspects of the DevPrivSecOps, such as privacy preserving code development, inclusion of privacy in the aerOS architecture, monitoring of privacy and security compliance, privacy and security training for the aerOS development teams and users, and correlation of the DevPrivSecOps methodology with the aerOS technological and architectural design.

8.1. Privacy preserving code development

Privacy preserving refers to the practice of protecting the privacy and confidentiality of sensitive information or data during its processing, storage, and transmission. It involves the use of various techniques and measures to ensure that personal or sensitive data is not accessed, disclosed, or used in ways that violate the privacy rights of the individuals associated with that data. Privacy-preserving techniques may include various methods including but not limited to data encryption, anonymization, access controls, data minimization as well as other methods designed to protect sensitive or personal data. The goal of privacy preserving is to strike a balance between data utility and privacy protection, so that data can be employed for legitimate purposes without compromising the privacy of the individuals whose data is being used.

Privacy preserving during the code development can be performed in numerous ways by implementing methods aiming to ensure that sensitive information or data is protected. These techniques are discussed below:

- **Data anonymization**: A well-known privacy-preserving technique is data anonymization, which removes or obfuscates any personally identifiable information (PII) from the data that will be employed by the developer. This can be achieved through methods like hashing, pseudonymization, or generalization.
- **Differential privacy:** Differential privacy is a type of data anonymization but is more suitable for datasets that regularly incorporate new data. Differential privacy is a technique that adds noise to the data in such a way that the statistical properties of the data remain intact while protecting the privacy of individuals in the data set [35]. This technique is implemented to protect sensitive data during code development by ensuring that any output generated by the code does not reveal sensitive information.
- **Data encryption:** Data encryption is another popular method to protect personal data during code development. By encrypting the data at rest or in transit, even if it is accessed by unauthorized users, they will not be able to extract any useful information.
- **Access controls:** Access control is another important technique for preserving privacy during the code development process. By restricting access to sensitive data or code to only authorized users the unauthorized access or misuse of data can be prevented.
- **Code review:** Code review is an important aspect of privacy-preserving code development. By having multiple developers review the code for potential privacy vulnerabilities, organizations can identify and remediate any issues before the code is deployed. This can include reviewing the code for any unnecessary data collection, data leakage, or insecure storage and transmission of data.

The aforementioned privacy preserving methods can be applied throughout the code development lifecycle to ensure that sensitive information or data is protected.

8.2. Privacy inclusion in the aerOS architecture

aerOS envisions to build a resilient platform-agnostic meta operating system for the IoT edge-cloud continuum. As one can understand, aerOS processes various types of data (e.g., personal, IoT generated, etc.) that should be protected from malicious actions, such as unauthorized access and data leakage. To this end, privacy protection is thoroughly considered from the beginning of the project, namely from the design of the architecture

following the privacy-by-design approach. Privacy along with security and trust by design are the anchors of establishing confidentiality, integrity, and availability; hence, aerOS focuses on evolving the DevSecOps operations by including actions that will establish privacy. This has as a result a major contribution of the project, which is the DevPrivSecOps methodology. The DevPrivSecOps methodology is an essential element of aerOS that includes security and privacy in the DevOps processes. More specifically, aerOS deploys an agile approach including several iterations and sprints, extending the DevSecOps methodology to add also privacy-by-design at the same level as security.

An important question that arises is *how privacy will be included in the aerOS architecture?*

Privacy will be included and preserved in aerOS mainly in three ways that are described below:

The implementation of privacy-preserving techniques such as encryption, anonymization, and differential privacy are the first step towards including privacy in the aerOS architecture. On the one hand, sensitive data can be encrypted at the edge and decrypted only when needed in the cloud, reducing the risk of data exposure. On the other hand, anonymization can be deployed to protect the identities of users or devices generating data, while differential privacy can help to protect the privacy of individuals in aggregated data.

Moreover, privacy is included in aerOS by utilizing privacy policies, laws, and regulations across the aerOS ecosystem. This includes policies that outline how data is collected, stored, processed, and shared, as well as the roles and responsibilities of different parties in the ecosystem. Regulations such as GDPR can provide guidelines for implementing privacy policies and ensuring compliance.

Finally, identity, access, and trust management mechanisms are also employed to guarantee the inclusion of privacy in aerOS. These facilitates the system administrators to manage establishing various roles and restricting users' access to aerOS procedures, accomplishing strong and secure authentication to the system by deploying strong authentication mechanisms (e.g., OpenID Connect), and ensuring trust among the various entities of aerOS (e.g., IoT devices).

In summary, privacy inclusion in aerOS is essential for ensuring the protection of sensitive data and addressing the privacy and security issues associated with edge-cloud continuum. Thus, aerOS evolves the DevSecOps procedures including privacy. Furthermore, aerOS implements from an early-stage privacy-preserving techniques, considers privacy policies, laws, and regulations as well as deploys identity, access, and trust management mechanisms transparency and user control mechanisms to achieve this goal and accomplish high-levels of privacy.

8.3. Monitoring of privacy and security compliance

The monitoring of privacy and security compliance is the process of checking and evaluating the systems, devices, and networks within aerOS ecosystem to comply with European regulations and cybersecurity standards. Monitoring the privacy and security compliance is not an easy task since regulations and standards change often in order to follow the progress of threats and vulnerabilities. However, the aerOS consortium is comprised by key members that have long experience both in privacy and security compliance and are capable to effectively tackle this issue. The security and privacy compliance are essential for maintaining the confidentiality, integrity and availability of the data and ensuring the uninterruptedly execution of the aerOS activities. This section elaborates on the methods and techniques that employed to monitor the security and privacy compliance of aerOS.

In aerOS in order to monitor the privacy and security compliance numerous proactive methods and techniques are implemented. Each method facilitates the monitoring process by contributing on different tasks (e.g., user training, vulnerability scans, etc.), but the integration of all the methods constitute a holistic approach to ensure the uninterrupted monitoring of security and privacy compliance. The first step towards monitoring the privacy and security compliance are the regular audits, namely regular privacy and security audits will be performed to identify potential privacy and security risks and assuring that aerOS is complying with GDPR and other security

and privacy standards [36] [37]. Vulnerability scans are also an important proactive action that helps in identifying weaknesses in the aerOS processes. Hence, regular vulnerability scans will be performed using the MITRE CVE database [38] to ensure that all the systems are up to date and vulnerability-free. Moreover, Data Protection Impact Assessments (DPIAs) [39] will be performed to assess the privacy risks associated with the data deployed in aerOS. DPIAs will assure that personal and sensitive data are collected, used, transferred, and maintained according with GDPR principles.

The aerOS developers and users will be trained to cultivate a privacy/security-by-design mindset as well as to be informed about the EU regulations regarding the processing of personal and sensitive data, best practices regarding the data collection, deployment, transfer, and storing as well as the code development process. Furthermore, aerOS employs security and privacy policies to enhance its resilience against cyberattacks; however, the deployment of these policies is not a one-time job, in the context of the monitoring of security and privacy compliance the security/privacy policies will be reviewed periodically to assure that they are up to date and meet the compliance requirements. In addition, aerOS implement access controls to avoid unauthorized access in aerOS sensitive data and processes. The access controls will be also monitored to ensure that only authorized users will have access to sensitive information.

By monitoring security compliance in these ways, is guaranteed that aerOS is protecting the confidentiality, integrity, and availability of its sensitive information, avoiding unexpected security incidents, and meeting regulatory requirements.

8.4. Privacy and security training for aerOS development teams

Privacy and security are two crucial factors that should be considered while a development team designs and implements and when a user deploys a software application. Privacy refers to ensuring confidentiality of personal records and security refers to protecting the software and hardware against malicious attacks like unauthorized access, phishing attacks, denial of service (DoS) attack, and many more. Therefore, it is essential to train development/integration teams and users about security and privacy to protect the integrity of confidential data and avert any potential risks of the systems.

aerOS development team can evolve their privacy and security knowledge by following a structured approach that includes education, best practices, and ongoing monitoring. The training should start by educating the aerOS development team about the importance of privacy and security as well as about the risks associated with not addressing them. The aerOS team should also have knowledge of the relevant laws and regulations that apply to their work. The training should then cover best practices for secure coding, secure storage and transmission of data, and securing APIs and cloud-based applications. The aerOS developers should be encouraged to implement security measures during every stage of the development process, from design to testing to deployment. Providing a training program in the context of aerOS project can foster a culture of security and privacy within development teams. Such a culture, when effectively established, could significantly decrease the probability of security breaches and privacy violations[40].

Some points on how to train the aerOS development team in privacy and security concerns are:

- **Webinars**: Regular webinars should be conducted to keep the aerOS development team updated with the latest security and privacy trends, risks, and regulations.
- **Provide hands-on experience**: Hands-on experience is an effective way to train the aerOS team on secure coding practices, secure storage and transmission of data, and securing APIs and cloud-based applications. This can be done by creating cybersecurity exercises [41], namely “a process to train for, assess, practise, and improve performance in an organization” based on the description that provided in ISO Guidelines for Exercises [42]. A common type to perform a cybersecurity exercise is via cybergames, like Capture the Flag (CTF) and security/privacy-focused hackathon. A CTF game typically involves teams competing against each other to find and solve security vulnerabilities in a controlled and safe environment. The teams are usually presented with various challenges, puzzles, and tasks that

simulate real-world security threats. In a hackathon, various developer teams are competing towards creating a secure and efficient application and evaluated based on whether they accomplished the pre-defined objectives.

- **Document and communicate policies and procedures**: Document and communicate policies and procedures related to privacy and security. These documents should be regularly updated and made easily accessible to the aerOS development team.

Training aerOS developers in security and privacy is essential to help them protect the personal information and prevent data breaches. The training should start by educating developers about the risks associated with sharing personal information, such as identity theft and financial fraud. aerOS developers should also be trained on how to identify common cybersecurity threats, such as phishing emails and social engineering scams. Best practices for creating strong passwords and regularly updating them should also be covered. Additionally, aerOS developers should be trained in the importance of keeping software and applications up-to-date with the latest security patches and updates. Finally, aerOS developers should be encouraged to report any security incidents or suspicious activity to the appropriate authorities. This training can be provided through a internal project webinars (that can also be for the external audience). By providing effective training to developers can significantly reduce the risk of security breaches and protect sensitive data for every possible implementation of the aerOS Project.

Some points on how to achieve training for privacy and security for aerOS developers:

- **Develop user-friendly training materials**: Develop training materials that are easy to understand and user-friendly, such as videos, infographics, and interactive tutorials.
- **Use real-life examples**: Use real-life examples of security breaches in edge-cloud and IoT environments and their consequences to help developers understand the importance of privacy and security as well as the severity of cyberattacks.
- **Test user knowledge**: Conduct regular assessments to test user knowledge and identify areas where additional training may be necessary.
- **Use simulated attacks**: Use simulated attacks, such as phishing simulations, to help developers identify potential security risks and enhance their cybersecurity habits.
- **Webinars**: Keep aerOS developers informed about the latest security threats and privacy updates. These webinars can serve as an effective means of educating developers on how to protect their sensitive information and mitigate potential security threats.
- **Provide ongoing support**: Provide ongoing support to aerOS developers to answer any questions and help them resolve any security/privacy-related issues.

8.5. Correlation of the DevPrivSecOps methodology with the aerOS technological and architectural design

To accomplish its objectives, the aerOS technological and architectural design follows the agile methodology including iterative work cycles with parallel streams of time to market context. The agile methodology that is followed in aerOS contains several iterations that rely on requirement gathering and planning, analysis and design, implementation, testing, and feedback. In this agile methodology the well-known DevSecOps procedures (thoroughly discussed in Chapter 5) will be extended to also encompass privacy in the same manner as security to overcome privacy related challenges at an early stage before leading to more serious issues (e.g., data breach).

aerOS envisions a modular architecture to exploit encapsulated functionalities and offer the ability to easily incorporate new modules. In this way, it aims to be scalable, hierarchical, decentralized, and adaptable to different necesities. In the aerOS architecture the continuum will contain different layers with various distributed groups. DevPrivSecOps will contribute to the development of each aerOS service by offering to the developers well-defined methodological steps that take into account both security and privacy by design principles to enhance the security and privacy of the whole aerOS ecosystem and reduce the risk of introducing security and privacy related issues during the development of aerOS components. For instance, during the

development the code is regularly checked for bugs that might lead to code injection attacks (i.e., adversaries insert arbitrary input to an application to cause unexpected actions). The tools described in the section 9 will be used to implement this first version of the DevPrivSecOps methodology, using for the different phases the necessary techniques.

9. Open-Source Software and tools to be used in aerOS to provide DevPrivSecOps

aerOS is a research project whose one of the most important final objectives is to reach a strong collaboration with open-source community's projects and initiatives. Therefore, aerOS plans to deliver most of its software outputs as open-source code, in addition to use existing open-source tools (and even contribute to them, if it is possible or an opportunity arises) to move forward its objectives. In consequence, preliminary research about the available tools to realize the DevPrivSecOps methodology was needed.

With the help of more than 18,000 developers, Digital.ai has designed an interactive landscape in form of a chemical periodic table that includes a set of the most popular DevOps tools grouped by purpose [43]. This table includes both open-source and proprietary tools, thus focus must be put on those being open-source and fitting the methodology. In the following subsections, the selected tools are described to provide an overall knowledge of their features and a better understanding about the reason why they have been chosen over other alternatives.

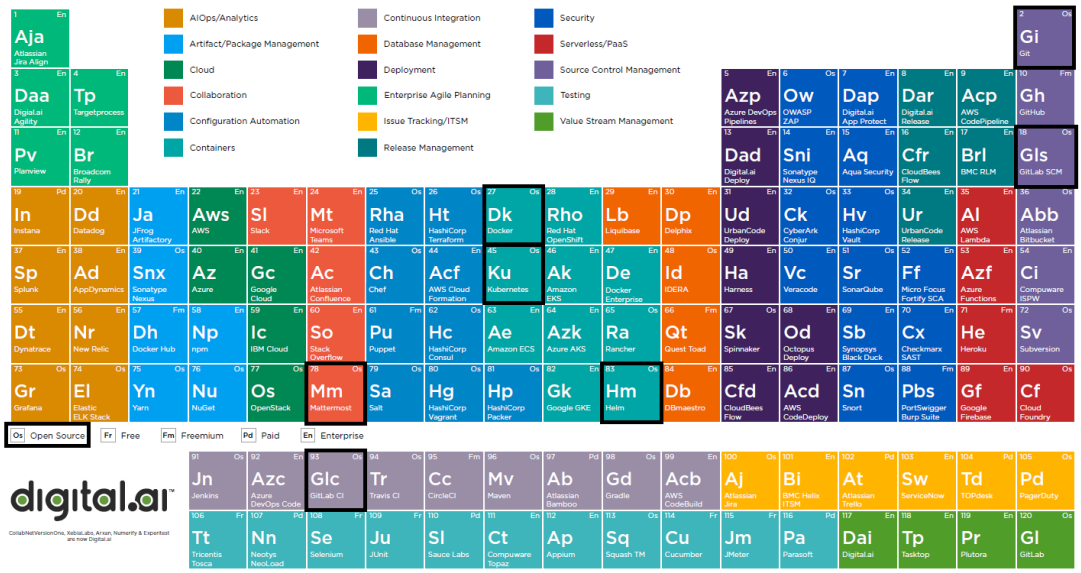

Figure 10 DevOps periodic table with the selected tools to be used in aerOS [43]

9.1. Tools for collaboration and communication environment

Mattermost is an open-source platform that encompasses instant messaging capabilities, collaborative workflows and project management. This software is a tool designed as the central core of communication and collaboration between the technical teams of the projects, designed for digital operations [44]. For that reason, Mattermost has been chosen in order to have agile, continuous, private and secure communication and collaboration between aerOS consortium's teams. It is possible to use Mattermost through a web browser, desktop applications (available for GNU/Linux, Windows and macOS) and mobile applications (available for Android and iOS). Finally, Mattermost has a large number of features and options, but only those most relevant to this project (and that are being used) are described below.

- **User management**: Mattermost allows to manage teams, users, channels, and permissions through its administration console. Each user has associated a specific role, which can be a system administrator or a member of one or more teams. Roles have different permissions associated with them that can be modified by Mattermost administrators in order to establish a hierarchy of users. At the end, every user belongs to at least one team, which is a set of users who can communicate through an almost unlimited number of channels or chats.
- **Channels**: in Mattermost, the communication between users is divided into channels, when the

conversation involves two or more users, and direct messages for peer-to-peer communications. Both types of communication include a chat that allows file sharing and the exchange of text messages with common features such as basic editing of the text style (use of bold, italics, lists, insertion of links, code statements, …), reaction to messages and an individual reply to them, which generates conversation threads within the chats (these threads help maintain channels cleaner) that can be managed in a specific view. For organizing channels to which a user belongs, it is possible to create categories. For instance, if a project is organized into several workpackages and each workpackage has several tasks with their corresponding channels (which is how aerOS has decided to structure Mattermost's instance), the administrator can create categories for each workpackage and include in each one its respective channels.

- **Notifications**: to keep up to date with the messages and files that are exchanged in the different channels or to which a user belongs or in private chats, Mattermost offers the possibility of receiving desktop, email or mobile app notifications, which can be configured independently for each user.
- **Plugins**: the default features provided by Mattermost can be extended by installing a set of plugins available in its official plugin store such as Jira, GitLab or GitHub. None of them have been utilized so far, but the Consortium will make use of specific plugins as long as the technical works will evolve.
- **Security**: one of the most important features of Mattermost is the high level of security it offers to its users, both on the server side and on the client side. Conceived as one of the most secure communication and collaboration platforms for development teams, Mattermost allows connection encryption, system monitoring, compliance with data protection laws, constant code reviews, continuous security updates, etc.

Below there is a screenshot of the current instance of Mattermost being used in aerOS:

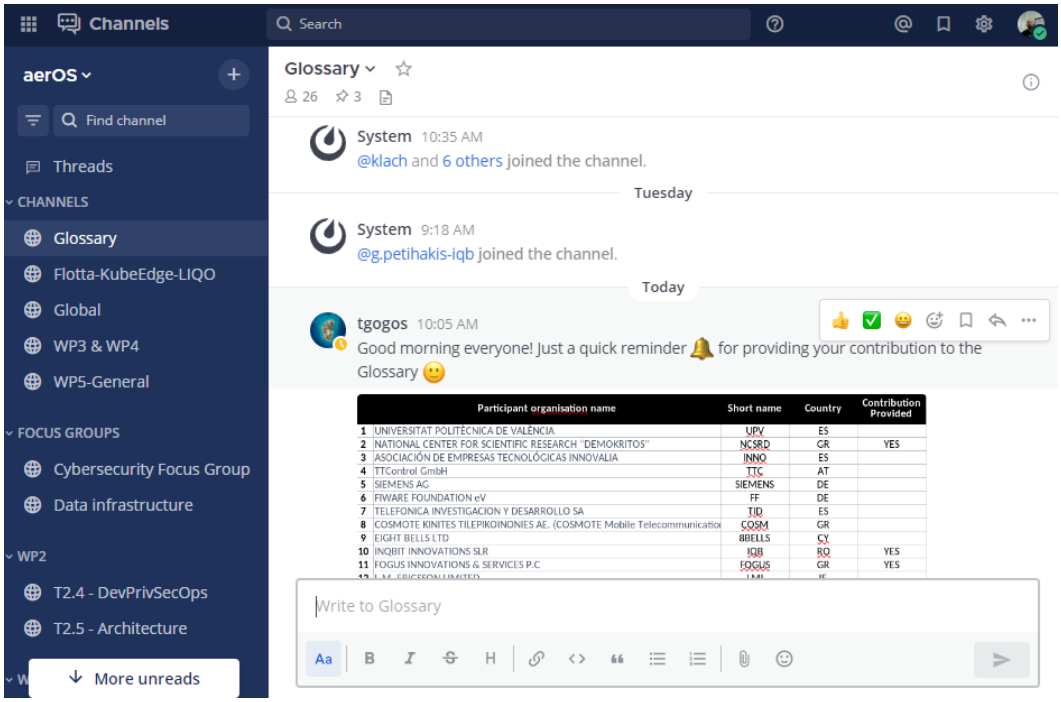

Figure 11 Overview of the Mattermost user interface

9.2. Tools for source version control and CPD

Version Control Systems (VCS) allow to track and manage changes in source code during the development stage of the software production, therefore, these systems play a main role in the practices for continuous planning design and development (CPD). Git and SVN (Apache Subversion) are the most common open-source

approaches, being the former distributed and the latter centralized. Git will be used as the VCS standard in the aerOS project based on previous positive experiences and its dominant position in the market.

Nowadays, a number of tools are available that can add extra functionalities to the Git standard – including IDE and user-oriented functionalities, so after extensive research backed up by the partners' knowledge, two of them have been selected: GitHub and Gitlab. GitHub presents the advantage of being a Software as a Service (SaaS) platform which is available from the public internet via website or desktop application. Thus, there is a cloud support, skipping the need to be self-hosted on client premises. Usage of GitHub is not free, but it does offer a basic free plan for organizations. However, this free plan presents some limitations: while the number of users and repositories is unlimited (even the private ones), the time used by CI/CD jobs in private repositories is limited to 2,000 minutes per month for all users in the organization, which can be a definitive limitation in case of partners' early code sharing policies (that might be restrictive). DevPrivSecOps methodology and selection of recommended tools must bear these scenarios in mind.

On the other hand, the Gitlab company also offers a SaaS solution with paid plans for organizations as its free plan is only available for single users. Nevertheless, the source code of the Gitlab tool has been delivered under an open-source MIT License, so it can be installed in private infrastructures while maintaining the vast majority of the features and benefits of the SaaS paid version [45]. This is the approach best fitting aerOS, allowing unlimited number of private repositories, full control over the stored resources and no restrictions on the CI/CD pipelines to carry out the DevPrivSecOps methodology (for instance, package repository, SAST tools, etc.).

In Gitlab, code repositories are also named projects and can be organized by groups. This is a feature of interest to the aerOS project, considering that groups can be created for each technical workpackage or area of interest involving interrelated tasks of them. Furthermore, the repositories have common features such as tracking of issues, merge requests, code releases and a complete Web IDE (Integrated Development Environment) for online code development.

Gitlab code repositories do not only manage and store versions of developed source code, but they include a registry to store container images, packages and infrastructure definitions related with the development. Since aerOS will root on cloud-native approaches (containers and their orchestration leveraging K8s), developed software must be packaged into container images. These images can be stored in the "Container registry" of each repository, suitably substituting inconvenient private container registries (require separated installation) and overcoming limitations of public repositories (e.g., DockerHub). Following with the cloud-native approach, at this stage of the project the consortium is considering the packaging of applications into Helm charts to be then deployed in a K8s cluster. Thus, the "Package registry" feature will be used to store such charts (along with other code packages such as Maven, npm or PyPI), presenting the same advantages over public repositories as the "Container registry".

Another advantage of using self-hosted Gitlab is that the number of users is unlimited. The administrator is responsible for managing the users, who can be assigned different permissions on code projects or groups through predefined roles (from less to more permissions: Reporter, Developer, Maintainer and Owner) in order to perform different tasks during the software development process in addition to achieve security and privacy in it.

Gitlab has been the selected tool for version control and CPD in aerOS. In forthcoming deliverables (i.e., D2.5), a extensive report on the usage of Gitlab in the project will be provided.

9.3. Tools for build automation and continuous integration

In the last section, self-hosted Gitlab has been stated as the best solution for managing source code repositories in aerOS, so the tools for building automation and continuous integration offered by Gitlab must be explored and compared to other alternatives to state if it is worthy to deploy an alternative tool. In Gitlab, the CI/CD process is conducted by a runner, which runs a series of jobs listed in a YAML file (the *.gitlab-ci.yaml* file, which is located in the root path of a Gitlab code repository) and reports their final results to Gitlab in order to show them to the final user in a user-friendly dashboard.

For self-hosted Gitlab, runners must be manually configured, so to register an infrastructure (it can not be the same where it is installed Gitlab) as a runner in a Gitlab project, Gitlab Runner must be first installed in it. The

specifications of the infrastructure depends on the type of the job that is required to be executed in a CI/CD pipeline, because it must support the requisites of the selected executor that will run the job in a final instance. For example, a Linux server must be registered as a runner which uses the Docker executor to run a job that consists of executing commands in a Docker container. Moreover, Gitlab offers a complete set of executors: Shell, Docker, SSH, VirtualBox, Kubernetes… Finally, when a runner is registered, it is set up a communication between the sef-hosted Gitlab instance and the infrastructure where Gitlab Runner has been installed.

In Gitlab, the CI/CD process is referred to as CI/CD pipelines, which are comprised of jobs (define the action, e.g., compile or test code) and stages (group a series of jobs and define the exact moment to run them, for instance, stages which contains job tests are run after the stage that compiles the code). Jobs are the most important elements of a Gitlab pipelines because they are the elements that really take into action the required executions. These jobs are not limited by number inside a stage and can be defined with constraints that specify the conditions for their execution. Pipelines usually move to the next stage if all the jobs in a stage are successful, otherwise, the next stage is not executed, and the pipeline ends before the completion of all the stages. According to the Gilab official documentation, a typical pipeline consists of four stages: build, test, stating and production. In addition, pipelines can be started depending on a wide range of conditions: scheduled over the time, manually triggered (by HTTP requests, jobs of other pipelines, webhooks, …) or commit changes to a branch, among others.

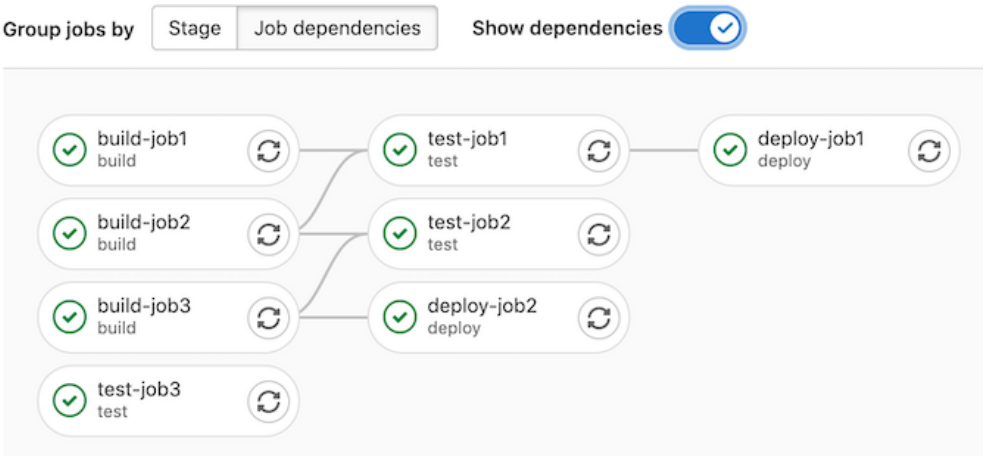

Figure 12 Gitlab CI/CD: jobs and stages ([46])

Furthermore, Gitlab includes an interesting functionality named Auto DevOps, which is able to detect the programming language used in a project and then use predefined CI/CD templates (*.gitlab-ci.yaml* files targeting common features of a set of programming languages that have been created by the Gitlab development team) to create and run pipelines which build and test the developed software.

In conclusion, Gitlab offers a complete toolset for achieving CI/CD pipelines in aerOS, in addition to other interesting functionalities that will be described in the following subsections. Therefore, the best solution is to take advantage of these built-in features of Gitlab instead of deploying a different tool to only be in charge of automation and continuous integration.

9.4. Tools for deployment automation, infrastructure automation and configuration management

Apart from the code CI/CD processes (covered in 9.3), aerOS would greatly benefit from automated means to put in place infrastructure where the software products could be run and tested over. Till now, the set up and arrangement of infrastructure has been done manually. The process of manually selecting infrastructure, configuring it, installing proper dependencies and setting it ready for deployment is tedious, failure-prone and might drastically reduce efficiency of code development teams although several tools are now allowing the structuration and preparation of (mostly, virtual) infrastructure in an automated way (what is known as

Infrastructure-as-Code – IaC), such as Ansible, Chef, Puppet or Terraform, those are not capable to live smoothly with the aforementioned CI/CD script promoted under the environment of Gitlab.

The partners of aerOS are more willing to find a solution (open-source) that would directly integrate with the rest of tools and processed explained in this section.

For that, the solution comes with GitOps. GitOps is a tool integrated in Gitlab that offers a way to automate and manage virtual infrastructure as long as it is available for the teams and it is virtualized and set up for cloud-native deployment, and it does this by using DevOps best practices that many organizations already use, and that coverage in aerOS is already guaranteed, such as version control (Git in Gitlab, section 9.2), code review, and CI/CD pipelines (Gitlab runners and jobs, section 9.3). GitOps allows to define infrastructure with code and parameterization, expressed in handy format and with high automation capabilities.

Thus, the goal of GitOps is to use CI/CD to automatically deploy resources by using code stored in owned Git repositories. GitOps allow the definition of infrastructure using JSON or YAML files stored in a git repository. This way, any change to the definition of the infrastructure to be arranged for deployment is tracked thanks to Git protocol usage. In addition, it allows to perform IaC code reviews, which can be crucial in cloud-native environments. It also allows the collaborative edition of IaC, via merge requests, enabling a much more dynamic and efficient management of the available infrastructure.

The functioning of GitOps orbits around four pillars:

1. The Git repository is the source of truth for the application configuration and code.
2. The CD pipeline is responsible for building, testing, and deploying the application.
3. The deployment tool is used to manage the application resources in the target environment.
4. The monitoring system tracks the application performance and provides feedback to the development team.

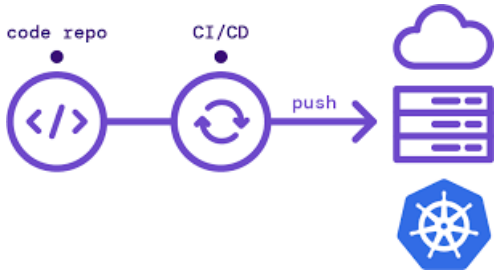

Figure 13 GitOps operative flow ([47])

In order to deploy GitOps, Gitlab has adopted the main tool FluxCD in order to allow the IaC automation over cloud-native environments (those based in Kubernetes). Up to today, the most used tool was ArgoCD, that, integrated in Gitlab, allowed GitOps to take place in overall environments (not cloud-native). However, with this new addition fits perfectly with the underlying container orchestration framework selected in aerOS (Kubernetes family).

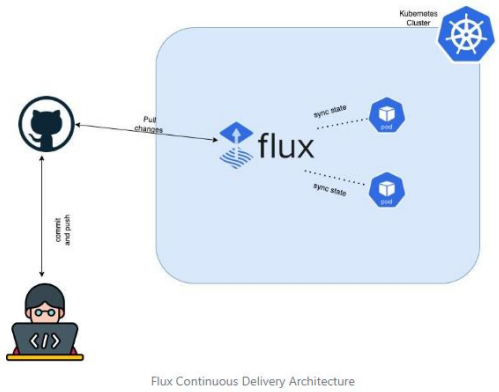

Figure 14 FluxCD as main implementation tool of GitOps in aerOS ([48])

With regards to the particular implementation of GitOps, it is worth remarking that is not only composed by a single tool, but of a combination of various technologies under the umbrella of GitLab plugins:

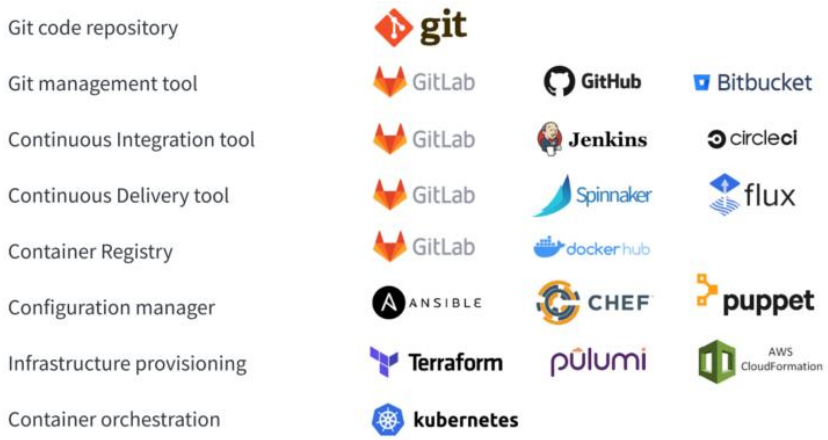

Figure 15 GitOps technological plugins ([49])

aerOS plans to use GitOps to automatically manage dynamic infrastructure that will be provided to the development teams in WP3 and WP4.

9.5. Tools for monitoring

Monitoring is an important part of the DevPrivSecOps process because it provides to users a clear vision of the actual status of it in real time. By default, GitLab provides through its dashboard three different types of analytics (divided into sections) to monitor interesting data and view statistics about repository usage:

- **Repository analytics**: in this section can be observed some graphs informing about the percentage of use of programming languages or commits in the code by month, day of the week and hours, among others.
- **Value stream analytics**: key and DevOps Research and Assessment (DORA) metrics filtered by date.
- **CI/CD analytics**: in the CI/CD flows section of a code repository are displayed overall statistics about pipeline durations, success rates and graphs for the last year, month or week.

As a complement to the aforementioned CI/CD flow analysis, there are other metrics that can be used to improve performance. These metrics are:

- **Code coverage**: to ensure sufficient coverage as the codebase grows, Gitlab calculates the amount of unit tests to measure the percentage of source code covered.
- **Failover time**: amount of time between when a build reports a test failure and the next build where the test passes.
- **Number of bugs**: bugs pending to be resolved in the Backlog (a list of GitLab bugs).
- **Deployment size**: used as a reference, this metric is used to monitor the size of applied changes to verify that they have been continuously applied without hitches.

Gitlab comes with the option of integrating Grafana, which is a tool for visualizing time series metrics through tables and graphs in information panels, to visualize the results of these measurement systems. To achieve this integration, GitLab deposits its performance data of the different components that make it up in Prometheus so that Grafana is able to obtain it and visualize it in its information panels. The GitLab team has published in their own repository a pre-set set of dashboards for Grafana in JSON format which can be imported one by one or all at once. Furthermore, it is possible to further integrate Grafana with the GitLab web dashboard by adding a shortcut to it via the sidebar.
In the last section, Flux CD has been described as the recommended approach to do GitOps in Gitlab. This GitOps solution can be integrated with Prometheus to monitor their processes using the kube-prometheus-stack,

a combination of Kubernetes manifests, Prometheus rules, Grafana dashboards and the Prometheus Operator. Flux CD is capable of displaying a large number of preconfigured dashboards through Grafana, such as the control plane dashboard, the cluster reconciliation dashboard or the control plane logs. Furthermore, it is possible to use own Prometheus and Grafana instances to be able to monitor other parameters of the Flux CD processes, as well as add annotations to the graphs of the Grafana dashboards or obtain error logs in JSON format.

10. DevPrivSecOps aerOS guidelines

This chapter presents the guide that has been created in the aerOS project to be followed by the project developers or any developer outside the project who wants to implement the designed methodology.

The Figure 16 presents the DevPrivSecOps methodology approach for aerOS, where a sequence of steps describes how it should be implemented from design through development to deployment and monitoring. The steps to be followed are as follows:

1. Continuous planning and tracking for development testing and deployment activities: security and privacy threat modelling
2. Code using IDE tools and SAST tools plugins integrated into IDE.
3. Commit code in git source version control repository.
4. Integration automation process pulls the source and build the application.
5. Integration automation may run SCA for dependency check and launch SAST tools.
6. Integration automation may run Acceptance, Smoke, Load and Performance Testing, with Unit and Integration Testing for Integration and Security test execution.
7. Integration automation launches Orchestration Platform and Configuration Management for configuration and build deployment at pre-production server.
8. Integration automation may run Acceptance, Smoke, Load and Performance Testing for Application Acceptance and Security and Privacy test execution.
9. Integration automation launches DAST tools for vulnerability scanner, pentesting and exploit tests.
10. Build automation tool uploads tested package to artifact repository.
11. Wait for approval for Production Deployment.
12. Integration automation launches Orchestration Platform for production server configuration and build deployment.
13. Configuration management download tested package from Artifact repository and deploys to production.
14. Integration automation may launch Acceptance, Smoke, Load and Performance Testing and DAST tools for vulnerability scanner, pentesting, privacy test and exploit test at production server.
15. Production server goes live with updated application and continuous operation.
16. Continuous operation with logging, analysis, visualization, and notification tools. Continuous Monitoring: Security Information, Privacy Information, Event Management and DAST tools
17. Continuous feedback through all the stages development, build, integration, testing and deployment tools at different stages of the workflow.

To implement this methodology, the tools presented in section 9 will be used. Since the project is still in the early stages, the set of tools that will be used to implement the methodology may vary.

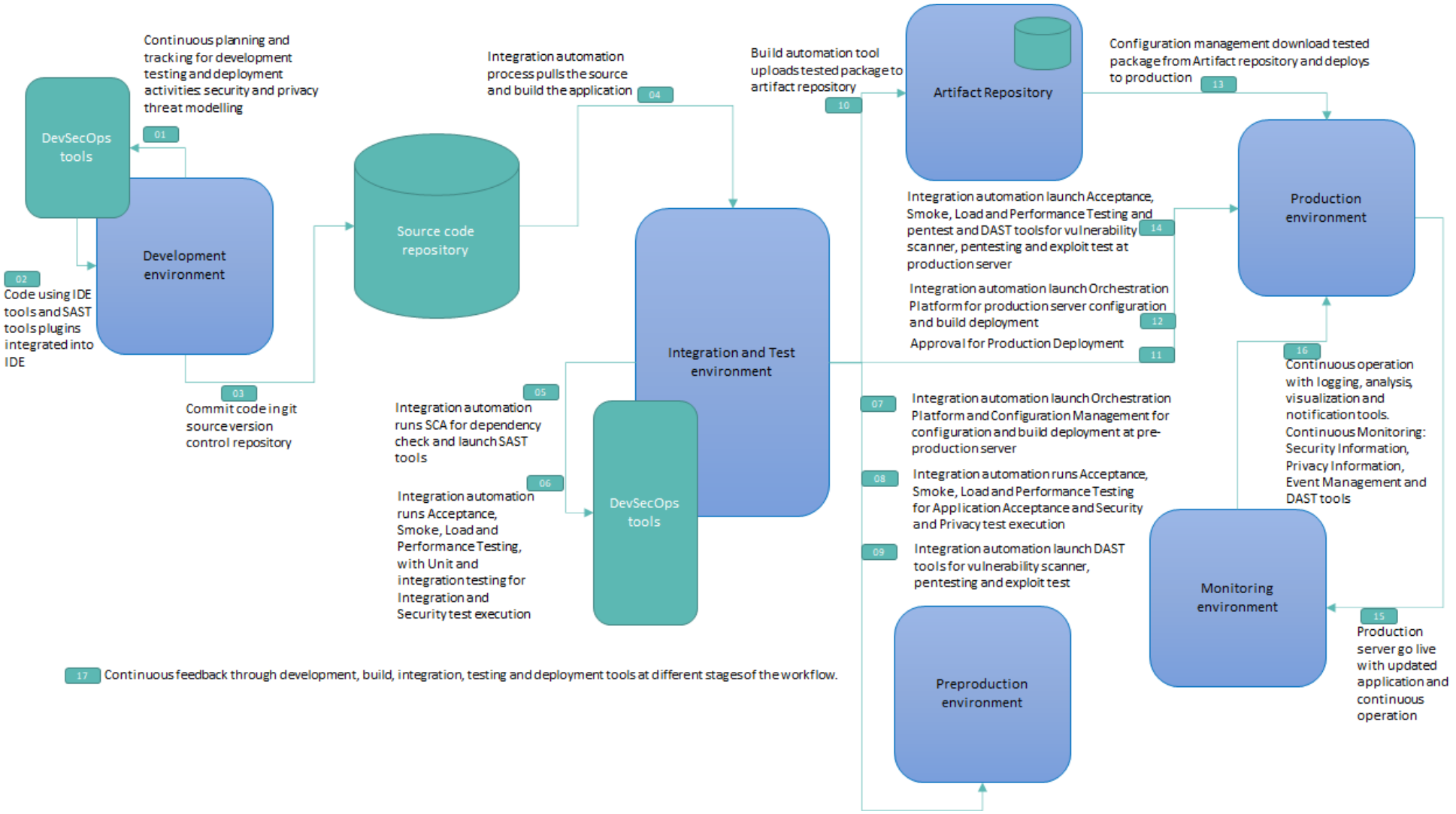

Figure 16 DevPrivSecOps methodology for aerOS

11. Future Work

The main goal of aerOS is to design and build a virtualised, platform-agnostic meta operating system for the IoT edge-cloud continuum. aerOS will: (i) offer common virtualised services to enable orchestration, virtual communication (network-related programmable functions), and efficient support for frugal and explainable AI and the creation of distributed data-driven applications; (ii) expose an API to be available anywhere and anytime (location-time independent), flexible, resilient and platform agnostic; and (iii) include a set of services and infrastructural features that address cybersecurity, reliability and manageability.

This set of functionalities that aerOS offers must be developed in an efficient and error-free manner so that the whole system can function without any problems.

The methodology presented in this deliverable will allow aerOS developers to follow the necessary guidelines to build secure and privacy aware by design software components, thus meeting the needs of the market.

By using the selected tools and following the presented methodology, developers will succeed in creating high quality software that will be deployed in a secure environment and monitored at all times.

Since this deliverable is delivered in a phase of the project where the architecture is not completely defined, the components to be developed are not yet finalised and the environment where it will be deployed is not yet available, it is expected that this first version of the methodology presented here will be modified during the life of the project, adjusting it to the needs that arise during the project.

Due to this, and related to the future work, it is expected to analyse the final aerOS architecture, the components to be developed and the deployment infrastructure, adapting the DevPrivSecOps methodology to the needs of these. In addition to integrating updates, new methods to analyse privacy in the software development lifecycle will be investigated during the duration of task T2.4. As mentioned in the document, this goes beyond the state of the art and through this project we intend to create a privacy aware methodology that can be used by the software development community.

References

[1] GDPR, “GDPR-info” [Online]. Available: https://gdpr-info.eu [Accessed 26th May 2023]

[2] Schwartz, P. M. (2019). Global data privacy: The EU way. NYUL Rev., 94, 771.

[3] Guerriero, M., Tamburri, D. A., Ridene, Y., Marconi, F., Bersani, M. M., & Artac, M. (2017, April). Towards DevOps for privacy-by-design in Data-Intensive applications: a research roadmap. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (pp. 139-144).

[4] Ahamed, S. F., Dhar, M., Kishore, S. K., Borawake, M. P., Thirupurasundari, D. R., & Thenmozhi, M. (2022, March). DevOps Security and Privacy in the Development of Multicloud Applications. In 2022 International Conference on Electronics and Renewable Systems (ICEARS) (pp. 1631-1635). IEEE.

[5] Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., ... & Williams, L. (2017). The top 10 adages in continuous deployment. IEEE Software, 34(3), 86-95.

[6] Z. Azham, I. Ghani, and N. Ithnin, "Security Backlog in Scrum Security Practices", Proc. 5th Malaysian Conf. Software Eng. (MySEC 11), 2011, pp. 414–417.

[7] Infoq, “The Top 10 Adages in Continuous Deployment” [Online]. Available: https://www.infoq.com/articles/cd-adages/ [Accessed 26th May 2023]

[8] Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. Ieee Software, 33(3), 94-100.

[9] TechTarget, “What is DevOps? The ultimate guide” [Online]. Available: https://www.techtarget.com/searchitoperations/definition/DevOps [Accessed 26th May 2023]

[10] Knowledge-hut, “DevOps Lifecycle: Definition, Phases” [Online]. Available: https://www.knowledgehut.com/blog/devops/devops-lifecycle [Accessed 26th May 2023]

[11] Microfocus, “DevOps and DevSecOps: Agile software development” [Online]. Available: https://www.microfocus.com/es-es/cyberres/use-cases/devsecops [Accessed 26th May 2023]

[12] Stackscale, “DevSecOps LifeCycle” [Online]. Available: https://www.stackscale.com/blog/devops-devsecops/ [Accessed 26th May 2023]

[13] Veritis, “What are the Phases of DevSecOps?” [Online]. Available: https://www.veritis.com/blog/what-are-the-phases-of-devsecops/ [Accessed 26th May 2023]

[14] Kumar, R., & Goyal, R. (2020). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). Computers & Security, *97*, 101967.

[15] XM Cyber, “What is a Red Team” [Online]. Available: https://www.xmcyber.com/glossary/what-is-a-red-team/ [Accessed 26th May2023]

[16] Z. Ahmed and S. C. Francis, "Integrating Security with DevSecOps: Techniques and Challenges," 2019 International Conference on Digitization (ICD), Sharjah, United Arab Emirates, 2019, pp. 178-182, doi: 10.1109/ICD47981.2019.9105789.

[17] Myrbakken, Håvard & Colomo-Palacios, Ricardo. (2017). DevSecOps: A Multivocal Literature Review. 17-29. 10.1007/978-3-319-67383-7_2.

[18] DevOps, “15 DevSecOps Best Practices” [Online]. Available: https://devops.com/15-devsecops-best-practices/ [Accessed 26th May 2023]

[19] Jaikumar Vijayan, “6 DevSecOps best practices: Automate early and often” [Online]. Available: https://techbeacon.com/security/6-devsecops-best-practices-automate-early-often [Accessed 26th May 2023]

[20] Lyndsi Hughes and Vanessa Jackson, “A Framework for DevSecOps Evolution and Achieving Continuous-Integration/Continuous-Delivery (CI/CD) Capabilities” [Online]. Available: https://insights.sei.cmu.edu/blog/a-framework-for-devsecops-evolution-and-achieving-continuous-integrationcontinuous-delivery-cicd-capabilities/ [Accessed 26th May 2023]

[21] S. A. Ebad, "Exploring How to Apply Secure Software Design Principles," in IEEE Access, vol. 10, pp. 128983-128993, 2022, doi: 10.1109/ACCESS.2022.3227434

[22] OWASP, “Secure Product Design Cheat Sheet” [Online]. Available: https://cheatsheet-series.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.html [Accessed 26th May 2023]

[23] Vidhani, K., Banahatti, V., & Lodha, S. (2021). Challenges in enabling privacy self management. CSI Transactions on ICT, 9(3), 185-191.

[24] Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., ... & Williams, L. (2017). The top 10 adages in continuous deployment. IEEE Software, 34(3), 86-95.

[25] Robles-González, A., Parra-Arnau, J., & Forné, J. (2020). A LINDDUN-Based framework for privacy threat analysis on identification and authentication processes. Computers & Security, 94, 101755.

[26] Wuyts, K., & Joosen, W. (2015). LINDDUN privacy threat modelling: a tutorial. CW Reports.

[27] ENISA, “Privacy and data protection by design” [Online]. Available: https://www.enisa.europa.eu/publications/privacy-and-data-protection-by-design/@@download/fullReport [Accessed 26th May 2023]

[28] OWASP, “Process for Attack Simulation and Threat Analysis (PASTA)” [Online]. Available: Retrieved from https://owasp.org/www-project-pasta/ [Accessed 26th May 2023]

[29] VAST, “A Visual, Agile, and Simple Threat modelling Methodology” [Online]. Available: https://www.synopsys.com/content/dam/synopsys/sig-assets/whitepapers/wp_vast_a_visual_agile_and_simple_threat_modeling_methodology.pdf [Accessed 26th May 2023]

[30] Trike, “Trike Methodology” [Online]. Available: http://www.trikeapps.com/methodology/ [Accessed 26th May 2023]

[31] PIA, “Privacy impact assessment” [Online]. Available: https://gdpr-info.eu/issues/privacy-impact-assessment/ [Accessed 26th May 2023]

[32] LINDDUN, “LINDDUN threat catalog” [Online]. Available: https://www.linddun.org/linddun-threat-catalog

[33] LINDDUN mitigation, “LINDDUN. Mitigation strategies and solutions” [Online]. Available: https://www.linddun.org/mitigation-strategies-and-solutions[Accessed 26th May 2023]

[34] NIST SP 800-37, “Risk Management Framework for Information Systems and Organizations: A System Lifecycle Approach for Security and Privacy” https://www.nist.gov/privacy-framework/nist-sp-800-37 [Accessed 26th May 2023]

[35] Tian Wang, Yaxin Mei, Weijia Jia, Xi Zheng, Guojun Wang, Mande Xie, Edge-based differential privacy computing for sensor–cloud systems, Journal of Parallel and Distributed Computing, Volume 136, 2020, Pages 75-85, https://doi.org/10.1016/j.jpdc.2019.10.009.

[36] ETSI TR 103 591, “Privacy study report; Standards Landscape and best practices ” [Online]. Available: https://www.etsi.org/deliver/etsi_tr/103500_103599/103591/01.01.01_60/tr_103591v010101p.pdf [Accessed 26th May 2023]

[37] ETSI GR SAI 001, “Securing Artificial Intelligence (SAI); AI Threat Ontology” [Online]. Available: https://www.etsi.org/deliver/etsi_gr/SAI/001_099/001/01.01.01_60/gr_SAI001v010101p.pdf [Accessed 26th May 2023]

[38] MITRE, “CVE database” [Online]. Available: https://cve.mitre.org/ [Accessed 26th May 2023]

[39] Binns, R. (2017). Data protection impact assessments: a meta-regulatory approach. International Data Privacy Law, 7(1), 22-35.

[40] Arain, M. A., Tarraf, R., & Ahmad, A. (2019). Assessing staff awareness and effectiveness of educational training on IT security and privacy in a large healthcare organization. Journal of multidisciplinary healthcare, 73-81.

[41] Yamin, M. M., & Katt, B. (2022). Modeling and executing cyber security exercise scenarios in cyber ranges. Computers & Security, *116*, 102635.

[42] ISO Central Secretary, “Societal security - guidelines for exercises” [Online]. Available: https://www.iso.org/standard/50294.html [Accessed 26th May 2023]

[43] Digital AI, “DevOps Periodic Table” [Online]. Available: https://digital.ai/learn/devops-periodic-table/ [Accessed 26th May2023]

[44] “Mattermost” [Online]. Available: https://mattermost.com/platform-overview/ [Accessed 26th May 2023]

[45] “GitLab” [Online]. Available: https://docs.gitlab.com/ [Accessed 26th May2023]

[46] GitLab, “CI/CD pipelines” [Online]. Available: https://docs.gitlab.com/ee/ci/pipelines/ [Accessed 26th May 2023]

[47] GitLab, "3 ways to approach GitOps”[Online]. Available: https://about.gitlab.com/blog/2021/04/27/gitops-done-3-ways/ [Accessed 26th May 2023]

[48] Saka-Aiyedun Segun, “Kubernetes GitOps with FluxCD” [Online]. Available: https://earthly.dev/blog/k8s-gitops-with-fluxcd/ [Accessed 26th May2023]

[49] GitLab, “Infrastructure automation using DevOps best practices” [Online]. Available: https://page.gitlab.com/resources-ebook-beginner-guide-gitops.html?_gl=1*1m0n6gp*_ga*MTI5ODQ3MjI1MC4xNjgzMjc2MTUx*_ga_ENFH3X7M5Y*MTY4MzI3NjE1MS4xLjEuMTY4MzI3NjI5NS4wLjAuMA.. [Accessed 26th May 2023]