This project has received funding from the European's Union Horizon 2020 research innovation programme under Grant Agreement No. 957258

Architecture for Scalable, Self-*, human-centric, Intelligent, Secure, and Tactile next generation IoT

D6.1 ASSIST-IoT DevSecOps Methodology and tools

Deliverable No.	D6.1	**Due Date**	30-APR-2021
Type	Report	**Dissemination Level**	Public
Version	0.8	**WP**	WP6
Description	This deliverable includes ASSIST-IoT Guidelines on DevSecOps Methodology.		

Copyright

The ASSIST-IoT consortium consists of the following 15 partners:

UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Spain
PRODEVELOP S.L.	Spain
SYSTEMS RESEARCH INSTITUTE POLISH ACADEMY OF SCIENCES IBS PAN	Poland
ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS	Greece
TERMINAL LINK SAS	France
INFOLYSIS P.C.	Greece
CENTRALNY INSTYUT OCHRONY PRACY	Poland
MOSTOSTAL WARSZAWA S.A.	Poland
NEWAYS TECHNOLOGIES BV	Netherlands
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS	Greece
KONECRANES FINLAND OY	Finland
FORD-WERKE GMBH	Germany
GRUPO S 21SEC GESTION SA	Spain
TWOTRONIC GMBH	Germany
ORANGE POLSKA SPOLKA AKCYJNA	Poland

Disclaimer

Authors

Name	Partner	e-mail
Ignacio Lacalle	P01 UPV	iglaub@upv.es
Eduardo Garro	P02 PRODEVELOP	egarro@prodevelop.es
Iordanis Papoutsoglou	P04 CERTH	ipapoutsoglou@iti.gr
Georgios Stavropoulos	P04 CERTH	stavrop@iti.gr
Oscar López	P13 S21SEC	olopez@s21sec.com
Mikel Uriarte	P13 S21SEC	muriarte@s21sec
Jordi Blasi	P13 S21SEC	jblasi@s21sec

History

Date	Version	Change
26-Feb-2021	0.1	Table of Contents proposed
03-March-2021	0.2	Table of Contents updated and assignments during 2nd Plenary
18-March-2021	0.3	PRODEVELOP contribution to assigned section
01-Aprl-2021	0.4	UPV and CERTH contribution to assigned section
15-April-2021	0.5	S21SEC compiled 1st draft version
20-April-2021	0.6	S21SEC version sent for internal review
29-April-2021	0.7	S21SEC version addressing comments from internal review
30-April-2021	0.8	S21SEC version addressing comments from PIC review

Key Data

Keywords	DevSecOps, DevOps, software development life cycle SDLC, plan, code, build, test, release, accept, deploy, operate, continuous integration CI, continuous delivery CD
Lead Editor	P013 S21SEC - Oscar López
Internal Reviewer(s)	P02 PRODEVELOP, P15 OPL

Executive Summary

DevSecOps is the natural extension of DevOps that advocates shift-security-left, security-by-design, and continuous security testing by building automated security controls in DevOps workflow.

The applications and services where data security and privacy are of prime concern can use the DevSecOps for delivering business application and services with agility, velocity, and assured security.

In this deliverable, a methodology based on a continuous security model for DevSecOps is presented based on the following three supporting pillars: continuous workflow, open-source tools and cloud deployment technologies for supporting workflow activities.

DevSecOps is defined as DevOps embedded with security controls providing continuous security assurance.

Following best DevSecOps practices with concrete target activities, the methodology proposed for ASSIST-IoT will enable the activation of security controls using appropriate open-source tools to perform security assurance tasks along the DevSecOps workflow.

The purpose of the deliverable D6.1 is the specification of technologies, tools and environment needed to be put in place for instantiation of the DevSecOps approach of ASSIST-IoT.

Disclaimer

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
AWS	Amazon Web Services
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
CaC	Compliance as Code
CI	Continuous Integration
CD	Continuous Deployment
CDE	Continuous Delivery
CF	Continuous Feedback
CT	Continuous Testing
DAST	Dynamic Application Security Testing
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAST	Interactive Application Security Testing
IDE	Integrated Developed Environment
OWASP	Open Web Application Security Project
OSS	Open Source Software
PaaS	Platform as a Service
SaC	Security as Code
SAST	Static Application Security Testing
SbD	Security by Design
SCA	Static Composition Analysis
SDLC	Software Development Life Cycle
SSH	Secure Shell Protocol
UAT	User Acceptance Testing
VM	Virtual Machine
RASP	Runtime Application Self-Protection
XaaS	Anything as a Service
XSS	Cross-site scripting

1. About this document

The main objective of this document is to set the foundations of the ASSIST-IoT DevSecOps methodology and the tools suite for the proposed continuous security model with DevSecOps.

DevSecOps continuous security practices in ASSIST-IoT will help to reduce, mitigate, and avoid potential threats on the software delivery lifecycle, and to secure operations on NG-IoT environments like those envisioned in the project.

1.1. Deliverable context

Keywords	Lead Editor
Objectives	**O1**: ASSIST-IoT Architecture supported by DevSecOps methodology and tools. **O2**: Smart networking components supported by DevSecOps methodology. **O3**: Definition and implementation of decentralized security and privacy, supported by DevSecOps methodology and tools. **O4**: Federation of smart AI enablers supported by DevSecOps methodology and tools. **O5**: Definition and implementation of human-centric tools interfaces supported according to DevSecOps methodology and tools. **O6**: Definition, deployment, and evaluation of real-life pilots supported by DevSecOps methodology and tools.
Work plan	DevSecOps methodology and tools indicated in this report will be the foundation for secure development and later secure operation for software components delivered in WP4 and WP5. The DevSecOps methodology proposed for ASSIST-IoT will be followed as much as possible during the integration WP6 (running as a parallel activity providing further outputs) and the deployment of pilots in WP7.
Milestones	This deliverable does not mark any specific milestone completion. However, it contributes towards *MS4 – Pilots deployed. Initial deployment of the three pilots* on M18.
Deliverables	The deliverable is the output for the task T6.1 DevSecOps Methodology. T6.1 DevSecOps methodology will run in parallel with other WP6 Integration activities until to M30.

1.2. The rationale behind the structure

The main objectives of D6.1 are (i) to define DevSecOps foundations and methodology in ASSIST-IoT, and (ii) to identify the DevSecOps tools to be used in the project. Section 3 and Section 4 details the traditional context of DevOps and the phases from development to operation. Section 5 explains the evolution and different approaches for DevOps and DevSecOps. Next, while Section 6 describes the DevSecOps methodology for implementing a continuous security model based on DevSecOps principles, DevSecOps workflow, and Section 7 lists a large number of available tools that cover the targeted activities that will enable to activate security controls and related practices as previously identified, and Section 8 describes a scenario use case approach for DevSecOps in ASSIST-IoT. Finally, Section 9 proposes the first guidelines to use GitLab in ASSIST-IoT.

2. Introduction

This document presents the project's methodology, which will follow the DevSecOps approach as a continuous model. It will contribute to develop security and operation in systems that requires continuous security, and thus, enabling delivery pipelines to deliver the security ready applications and services, capitalizing the cloud infrastructure and technologies. In particular, all applications and services where data security and privacy will be the prime concern will use the DevSecOps continuous model proposed here for delivering business application and services with agility, velocity, and assured security.

The DevSecOps approach for a continuous model is possible to be used in NG-IoT (Internet of Things) systems, including fog computing, edge computing, cyber physical system (CPS), VNF (virtual network function) applications of network communication or telecommunication, future 5G applications, etc., as these applications are closely tied with cloud infrastructure technologies [1][2][3][4][5][6][7][8][9].

3. DevOps definition

Historically, the lack of cooperation among the development and operations teams in software production often resulted in facing a lot of challenges along the software development lifecycle. For instance, if the deployment strategy of the software suite of a company plans big releases every e.g., six months, there is a lot of risk involved. For instance, if one thing goes wrong, it may break everything. Hence, the plan of deploying so many changes at once leads to a very hard forensics processing on identifying what, where and why are located those bugs that crashes the new release available.

This is where DevOps came into play. The term coined by Patrick Debois, in October 2009 [11], is referred to "*a movement of people who think it is time for a change in the IT Industry - time to stop wasting money, and time to start delivering great software, and building systems that scale and last*". Hence, DevOps was about fast, flexible development and provisioning of business processes, which by efficiently integrating development, delivery, and operations, facilitates a lean, fluid connection of these traditionally separated silos [10].

Even though the DevOps term and movement has been discussed for nearly a decade, it lacks a widely accepted definition. By consolidating the most cited definitions of DevOps, we will adopt the following statement from [12]:

> *DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability.*

To be more precise, DevOps is a methodology that is helping organisations (and their production teams) to build software in a way that enables continuous rapid deployment. In that sense, DevOps might be applied to very different delivery sectors, although it but must be tailored to the environment and product architecture (e.g., even in very constrained environments like embedded systems, upgrades can be planned and delivered quickly and reliably). Additionally, besides highly secured cloud-based delivery, DevOps need dedicated architecture and hardware approaches that fulfil the agile and lean delivery methodologies.

Following the systematic analysis of the literature carried out in [12], the conceptual framework of DevOps is composed of a conceptual map outlining four categories: ***process*** (which encompasses business-related concepts), ***people*** (which covers skills and concepts regarding the culture and collaboration), ***delivery*** (related with the CI/CD concept – Continuous Integration / Continuous Delivery-Deployment), and ***runtime*** (that synthesizes concepts needed to guarantee the stability and reliability of services in a continuous delivery environment).

The four conceptual categories of DevOps, and their interrelation with engineering and management perspective, as well as with Development and Operations roles are depicted in Figure 1

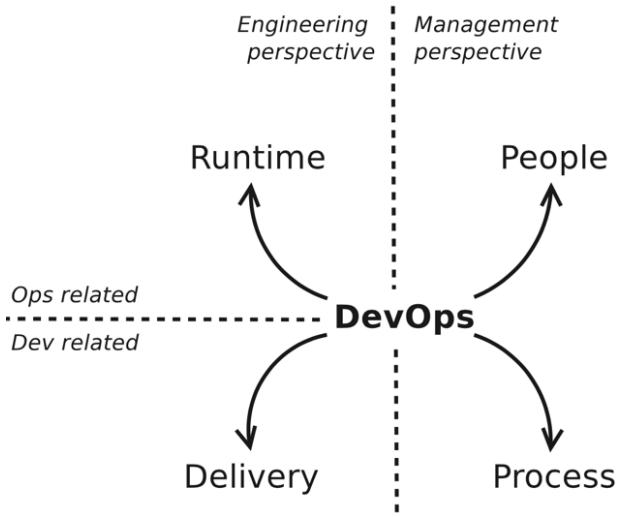

Figure 1. DevOps overall conceptual map [12]

From the figure, it can be considered that while the ***process*** and ***people*** categories relate more to the *management perspective*, ***runtime*** and ***delivery*** relate more to an *engineering* perspective. Moreover, within the engineering perspective, while ***delivery*** concepts relate more to *developers*, ***runtime*** concepts relate more to *operations* role.

As a summary, DevOps integrates the two worlds of development and operations, using automated development, deployment, and infrastructure monitoring. It is an organizational shift in which, instead of distributed siloed groups performing functions separately, cross-functional teams work on continuous operational feature deliveries. This approach helps to deliver value faster and continuously, reducing problems due to miscommunication between team members, and accelerating problem resolution.

4. DevOps Phases

Although there are several approaches aiming to identify which are the different DevOps stages or phases, those that are most frequently adopted in DevOps culture are presented in Figure 2. It includes eight phases: Plan, Code, Build, Test, Release, Deploy, Operate, Monitor.

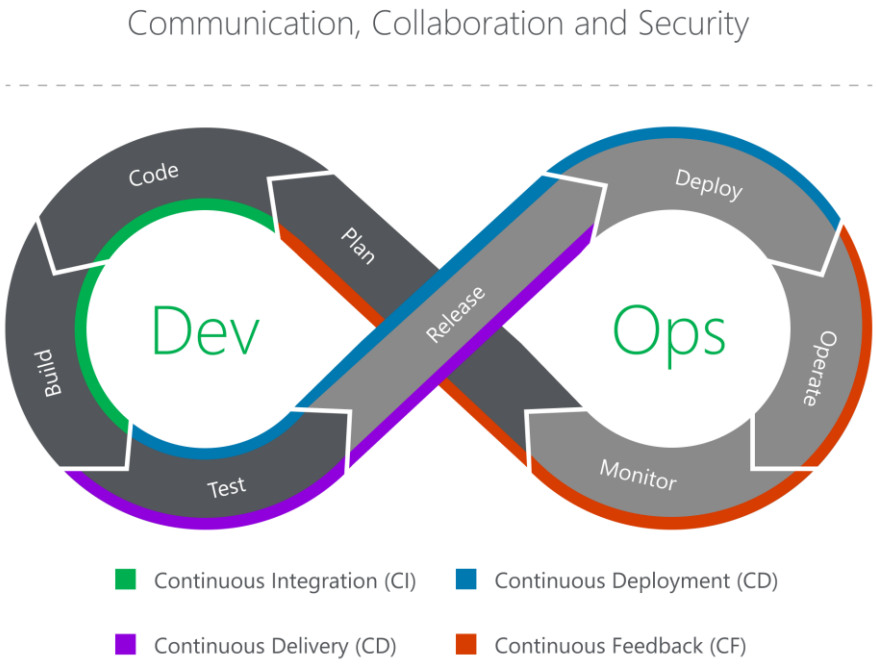

Figure 2. DevOps phases [13].

A short description of what is the objective of each phase as extracted from [13] is described next:

1. **Plan**: The Plan stage covers everything that happens before the developers start writing code, and it is mainly relate with the Product/Project Manager role. Requirements and feedback are gathered from stakeholders and/or customers and used to build a product roadmap to guide future development. The product roadmap can be broken down in different ways like features or user stories, creating a backlog of tasks that lead directly to the customers' requirements. The tasks on the backlog can then be used to plan sprints and allocate tasks to the team to begin development.
2. **Code**: In addition to the standard toolkit of a software developer, the DevOps team has a set of plugins installed in their development environments to aid the development process, including consistent code-styling and avoiding common security flaws. This set of plugins helps to teach developers good coding practice while boosting collaboration providing some consistency to the codebase. These tools also help to resolve issues later in the testing pipeline, resulting in fewer failed builds.
3. **Build**: Once a developer has finalized a task, he/she commits his/her code to a shared code repository. There are many ways this can be done, but typically the developer submits a pull request (i.e., a request to merge his/her new code with the shared codebase). Another developer then reviews the changes he/she has made, and once there are no issues, the pull-request is approved. This manual review is supposed to be quick and lightweight, but it is effective at identifying issues early. Simultaneously, the pull request triggers an automated process, which builds the codebase and runs a series of end-to-end, integration and unit tests to identify any regressions. If the build fails, or any of the tests fail, the pull-request fails, and the developer is notified to resolve the issue. By continuously checking code changes into a shared repository and running builds and tests, traditional integration issues that arise when working on a shared codebase are minimised.
4. **Test**: Once a build succeeds, it is automatically deployed to a staging environment for deeper, out-of-band testing. The testing that is performed during this phase is up to the organisation and what is relevant to the application, but this stage can be considered a testbed that lets to plug in new testing without interrupting the flow of developers or impacting the production environment. The staging environment may be an existing hosting service, or it could be a new environment provisioned as part of the deployment process. This practice of automatically provisioning a new environment at the time of deployment is referred to as Infrastructure-as-Code (IaC) and is a core part of many DevOps pipelines. Once the application is deployed to the test environment, a series of manual and automated tests are performed. Manual testing can be traditional User Acceptance Testing (UAT), where people use the application as the customer to highlight any issues or refinements that should be addressed before deploying into production. At the same time, automated tests might run security scanning against the application, check for changes to the infrastructure and compliance with hardening best-practices, test the performance of the application or run load testing.
5. **Release**: The Release phase is a milestone in a DevOps pipeline, as it is the point at which a build is ready for deployment into the ***production*** environment. By this stage, each code change has passed a series of manual and automated tests, and the operations team can be confident that breaking issues and regressions are unlikely. Depending on the DevOps maturity, it is possible to automatically deploy any build that makes it to this stage of the pipeline. Developers can use feature flags to turn off new features, so they cannot be seen by the customers until they are ready for action. Alternatively, in case to looking for a more control of when builds are released to production, it could be proposed to have a regular release schedule, or only release new features once a milestone is met. It can also include a manual approval process at the release stage, which only allows certain people to authorise a release into production.
6. **Deploy**: This stage is when a build is released into production. The same IaC that built the test environment can be configured to build the production environment. As it is already confirmed that the test environment has been built successfully, the DevOps practitioner can be confident that the production release will go satisfactorily on. A blue-green deployment will let to switch to the new production environment with no outage. Then the new environment is built, and it sits alongside the existing production environment. When the new environment is ready, the hosting service points all new requests to the new environment. If at any point, an issue is found with the new build, by simply

telling the hosting service to point requests back to the old environment, the practitioner can come up with a fix.

7. **Operate**: The new release is now live and being used by the customers. In this stage, the operations team should make sure that everything is running smoothly. Based on the configuration of the hosting service, the environment automatically scales with load to handle peaks in the number of active users. The organisation has also built a way for the customers/stakeholders to provide feedback on their service, as well as tooling that helps to collect feedback for helping on shaping next releases of the future development of the product. This feedback loop is important, as the customer is the best testing team, donating many more hours to testing the application than the DevOps pipeline itself.
8. **Monitor**: The final phase of the DevOps cycle is to monitor the environment, sustained by the customer feedback, by collecting data and providing analytics on customer behaviour. It can also include monitoring for potential performance bottlenecks in the DevOps pipeline, which are impacting the productivity of the development and operations teams. All of this information is then fed back to the Product Manager and the development team to close the loop on the DevOps process. Although it can be seen that is the beginning of a new loop, it should be considered that the DevOps is a continuous process (i.e., there is no start or end, just the continuous evolution of a service throughout its lifespan).

4.1. CI/CD

Ideally, and with the goal of agile and rapid deployment, DevOps software shall move continually through the aforementioned eight DevOps stages in an infinity loop. In this sense, some of the previous defined stages are grouped within the so-called Continuous everything concept, which is also known as Continuous Integration and Continuous Delivery and Deployment (CI/CD) concept. CI/CD are the foundational component of modern software DevOps development, as they involve the **Code, Build, Test, Release and Deploy** phases of the DevOps lifecycle. A breakdown of these terms and how they are related to the phases of the pipeline are described next.

Continuous Integration: One of the biggest difficulties in coordinating a software development team is managing the collaboration of many developers on a single codebase. A shared code repository is key to solving this problem However, there can be issues when merging the changes made by multiple people on the same piece of code. Continuous integration aligns with the ***Code*** and ***Build*** phases of the DevOps pipeline. It generally refers to performing all of code tests, unit tests, and integration tests. By merging smaller changes more regularly, the issues become smaller and easier to manage, improving overall productivity.

Continuous Delivery: It can be seen as an extension of Continuous Integration, which automates the process of deploying a new build into production. The goals of Continuous Delivery are (i) to perform automated testing on each new build in order to verify that builds are ready for release into production; (ii) to manage the automatic provisioning and configuration of deployment environments as well as testing of these environments for stability, performance, and security compliance; and (iii) to deploy a new release into production when approved and manually triggered by the organisation. As it can be seen in Figure 2, Continuous Delivery embraces the ***Test*** and ***Release*** phases of the pipeline, allowing organisations to manually trigger the release of new builds as regularly as they choose.

Continuous Deployment: It is a more advanced version of Continuous Delivery. The goals are the same, but the manual step of approving new releases into production is now automated. It involves the ***Test***, ***Release***, and ***Deploy*** phases of the pipeline. In a Continuous Deployment model, each build which passes all the checks and balances of the pipeline are automatically deployed into the production environment.

5. Evolution from DevOps to DevSecOps

DevSecOps is defined in [26] as DevOps embedded with security controls providing continuous security assurance. DevSecOps is a natural extension of DevOps that advocates for shifting security-left to security-by-design and continuous security testing by making use of automated security controls in the DevOps workflow. Figure 3 illustrates how DevSecOps adds a continuous security assurance embedding security controls across a DevOps workflow.

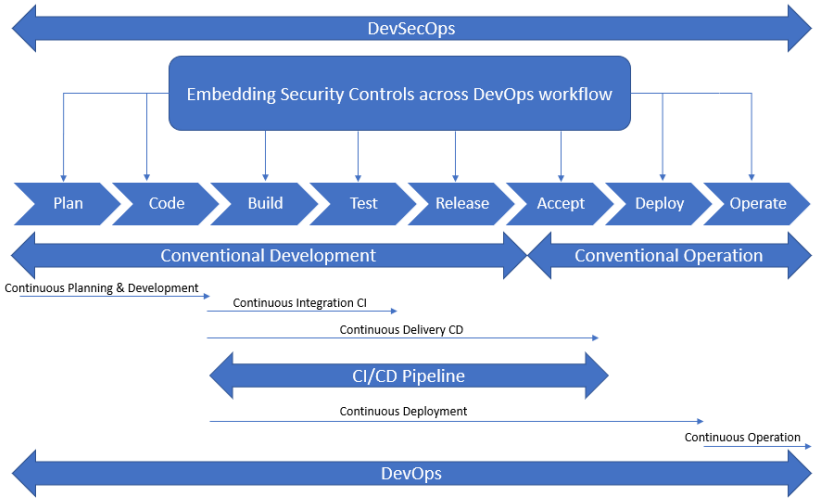

Figure 3. DevSecOps embedded security control [26]

Therefore, DevSecOps is DevOps with security. Hence, in DevSecOps models, DevOps teams should also collaborate with security team, developing a culture wherein development and operations team include security as integral component in their work products [14]. In addition, DevSecOps aims to bring cultural transformation in an organization by changing people's mindset that building and delivering security enabled applications is everybody responsibility and not just a tick mark towards end of the completed work. Automation plays a pivotal role in bringing this cultural transformation and mindset change. OSS drive this automation capitalizing the cloud infrastructure and related technologies.

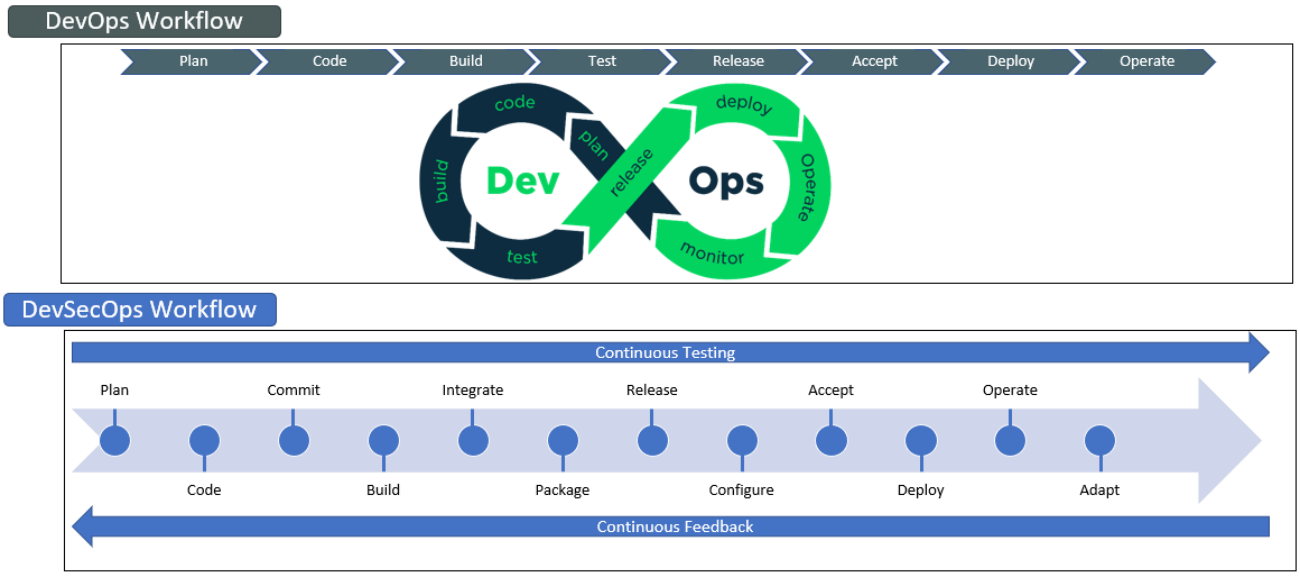

Figure 4: DevOps and DevSecOps comparative workflow

In Section 6 we propose a methodology for implementing a continuous security model for DevSecOps.

6. DevSecOps continuous security model

This section presents the general overview for each of the steps of the DevSecOps Methodology. Following the research work on [26] we propose a methodology for a continuous security model for DevSecOps based on the following three supporting pillars.

- DevSecOps continuous workflow applying DevSecOps practices and principles embedded with security controls.
- ***Open Source software tools:*** to support continuous workflow activities as necessary technologies.
- ***Cloud infrastructure technologies***: to support continuous workflow activities.

Figure 5 describes a conceptual framework that is the basis for designing security assurance controls for DevSecOps.

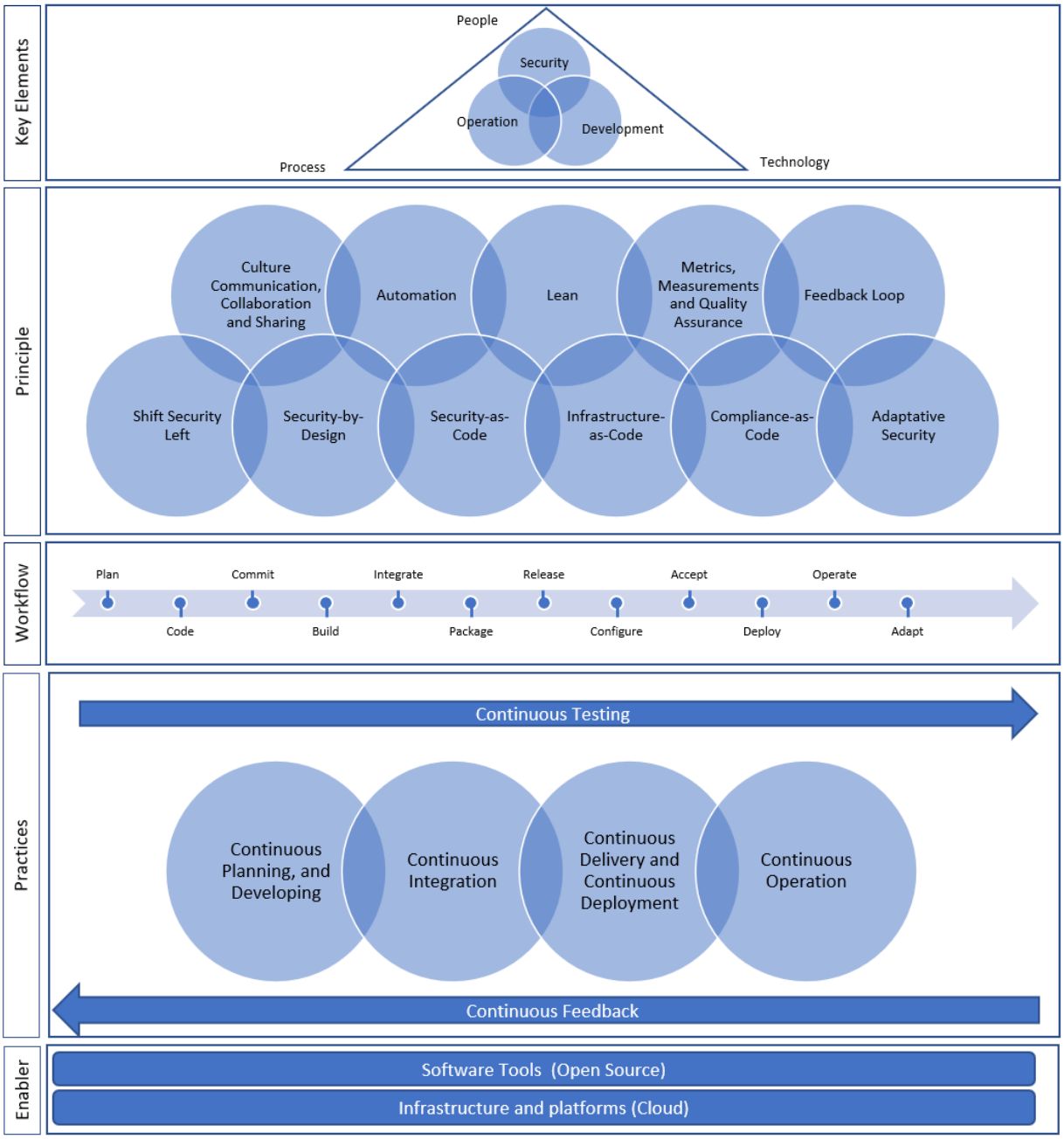

Figure 5: DevSecOps conceptual framework for continuous security model [26]

As it can be seen, the framework extends the fundamental principles and practices of DevOps [27][28][29] and it consists of five parts:

1) ***Fundamental elements.*** Organizational core units of development, security, and operations in a team, applying principles and practices by the organization for the people, process and technology.
2) ***DevSecOps Principles.*** A set of guidelines providing foundation for security controls in the continuous security model.
3) ***DevSecOps Workflow.*** Different stages to deliver application and services.
4) ***DevSecOps Practices.*** Different activities executed along the workflow that activate security controls.
5) ***Open-Source Software tools*** and ***cloud technologies*** as enablers for supporting the continuous security model.

The fundamental or key elements mentioned above as people, process and technology will work according to business requirements to contribute to development, security, and operation. The next sections describe in detail the rest of the proposed DevSecOps framework of the project.

6.1. DevSecOps Principles

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

- Culture Communication, Collaboration and Sharing.
  - The DevSecOps is a techno-cultural transformation that necessitates mind shift of the development, operations, and security people 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 anticipate before it can and act proactively to prevent system from any possible security breach.

6.2. DevSecOps Workflow

6.2.1.Plan

Both functional and non-functional security requirements, including security, are analysed by the experts from business, development, security, and operations. The features are broken into working software, deliverable in small duration. Based on threat modelling, an adaptive security architecture and design approach is adopted to build incremental security controls in every development cycle. An end-to-end planning that involves integration of all the stages of workflow is performed.

Security controls in DevSecOps for Plan phase are:

- Security requirement analysis
- Threat modelling
- Adaptative security architecture and design
- Security use, misuse, and abuse test cases
- Code review and security guidelines check
- Software inventory management

6.2.2. Code

Developers are provided with secure code development guidelines. The coding guidelines are checked and enforced through available plug-ins for the Integrated Development Environment (IDE) used by the developers.

Security controls in DevSecOps for Code phase

- Security use, misuse, and abuse test cases
- Code review and security guidelines check
- Source code version control security
- Unit and integration security testing
- Static application security testing (SAST)

6.2.3. Commit

At this stage, developers check in their code in a source repository, which is automatically version controlled. A commit is followed by a build that provides immediate feedback for any break in the application source code base.

Security controls in DevSecOps for Commit phase are:

- Code review and security guidelines check
- Source code version control security
- Unit and integration security testing
- Static application security testing (SAST)
- Software composition analysis (SCA)

6.2.4. Build

The build stage starts with compiling the changed source code while resolving all the dependencies. During compilation it can trigger static code analysis and SCA, using appropriate tools.

After the *Commit* step, which provide as outcomes a series of isolated software elements (binaries/services/virtualised components) available (tagged and versioned) in the shared code repository, sub-divided in proper sub-project folders, with clear pull/push mechanisms (based on *Git* protocol), the *Build* mechanisms aim to convert the code pieces into actionable software featuring functionalities in the system.

However, this process may differ depending on the type of software development conducted and, more specifically, to the language and technologies used (e.g. compiling binaries, composing a service, building a *docker* image, etc.).

Technically, the actual **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. If the code presents any flaw (e.g., code does not compile, it cannot be executed, it does not properly interpret environment variables or path, some libraries break, dependencies are not up-to-date), the build tool informs the user or the committer developer to modify the commit and the process starts over again.

The "Build" stage of the DevOps pipeline has turned into a **security-powered step in DevSecOps**, embedding security controls whenever generating the image/compiled binary/service and executing individual micro-testing actions over them before verifying for deployment. As a first step, DevSecOps introduces Static Application Security Testing (SAST) [15], consisting of isolated security tests applied over the source code (e.g., SQL injection or XSS) before proceeding with actual build. The objective of SAST [15] [16] is to look for vulnerabilities in the source code at different stages of the DevSecOps workflow and perform continuous testing as a continuous security practices into the different phases of DevSecOps practices. In addition, SAST tools allow organisation to incorporate ad-hoc rules for finding specific vulnerabilities in the code (e.g., the known OWASP Top 10 [17] or others previously defined after risk analysis) and in the third-party dependencies. The SAST process also generates a series of useful metrics and logs that are kept for potential further analysis and record. Afterwards (once SAST are successfully checked in the piece of code), the traditional (DevOps) build process takes place. In addition, after packaging and prior to store in an artifact repository package code it should consider including a digital signature.

Build result is crucial in the **DevSecOps** pipeline as it performs the first verification and testing at a very early stage of the cycle. This effect (promoted in DevSecOps) contributes to the so-called *shift left* philosophy, consisting of breaking the application in smaller pieces with regards to security issues identification, so that it is easier and cheaper to solve those after a more accurate location of the breach.

Security controls in DevSecOps for Build phase are:

- Software inventory management
- Source code version control security
- Unit and integration security testing
- Container and Infrastructure as Code (IaC) analysis
- Software composition analysis (SCA)
- Static application security testing (SAST)
- Security smoke testing

6.2.5. Integrate

This stage is characterized as integration of application software components as one system, including hardware infrastructure consideration. The system integration and security tests are conducted looking for any interface that breaks between two different interacting components, including any third-party software. The infrastructure configuration codes are validated as per the application requirement.

Security Controls in DevSecOps for Integrate phase are:

- Source code version control security
- Unit and integration security testing
- Container and Infrastructure as Code (IaC) analysis
- Artifact Repository Security Management
- Software composition analysis (SCA)
- Static application security testing (SAST)
- Dynamic application security testing (DAST)
- Fuzzy testing

- Interactive application security testing (IAST)
- Run-time application self-protection
- Continuous vulnerability scanning
- Security patch application
- Security smoke testing

6.2.6. Package

All the required binaries used by an application to deliver the expected features, like web server, database, third-party libraries, are bundled together in one application package along with release notes, installation and configuration scripts or instructions. This packaged software is stored, mostly, in a separate artifact repository.

Once the checks for building the application successfully advance, the result of this step (in DevOps flow) consists of a **"packaged"** executable coded. In ASSIST-IoT (as presented in deliverable D3.5), the pieces of source code will be packaged following a virtualisation approach, meaning that those will be either Docker Images, Kubernetes (or similar) Pods and/or Virtual Network Functions. Finally, the build step includes performing single automated testing of the functioning of the runnable code. The way that Package stage of DevSecOps will be performed in ASSIST-IoT is described in Section 9.

Security Controls in DevSecOps for Package phase are:

- Artifact repository security management

6.2.7. Release

The packaged application is delivered from artifact repository to a staging environment for the user acceptance. The staging environment is replica of production environment and the application is tested for expected behaviour in production.

Security Controls in DevSecOps for Release phase are:

- User acceptance and security testing
- Artifact repository security management

6.2.8. Configure

The application installed at the staging environment is configured with necessary configuration data for acceptance testing.

As mentioned earlier, the release and later configure step in DevSecOps pipeline is the point at which a build is ready for deployment into the production environment. By this stage, each code change has passed a series of manual and automated tests, and the operations team can be confident that breaking issues and regressions are unlikely.

All builds arriving at this point would have passed a SAST, DAST and usual controls and tests, and the artifacts will be (signed) and ready in the shared repository of the project/organisation, but still at the "staged repository", waiting for final validation and release to a production-close environment/repository.

Security Controls in DevSecOps for Configure phase:

- Dynamic application security testing (DAST)
- Interactive application security testing (IAST)

6.2.9. Accept

The users and the quality assurance team execute different functional and non-functional testing to validate application features and performance as per expectations. The user experiences are shared to the development

team for resolution and improvisation in next iteration. The different security tests, like penetration test, fuzz test, DAST, etc., are performed to identify and rectify any security gaps.

The acceptance stage in DevOps workflow [13] and concerns the integration of testing to ensure the security as described in the Component analysis of DevOps and DevSecOps [18] in Hong work.

Mayoral-Vilches et al. [19] have described four methods of software security testing in this present phase. The methods consist of black-box and machine learning security methods that pinpoint the potential vulnerabilities. These methods are Dynamic Application Security Testing (DAST), Deep Static Application Security Testing (Deep SAST), Fuzz testing, and Penetration Testing. The methods are included in Björnholm [20] work with the addition of Dependency Scans. All these methods are different and are important to uncover pitfalls prior to deploying the software.

Security Controls in DevSecOps for Accept phase are:

- User acceptance and security testing
- Artifact repository security management
- Penetration testing
- Software composition analysis (SCA)
- Static application security testing (SAST)
- Dynamic application security testing (DAST)
- Fuzzy testing
- Interactive application security testing (IAST)
- Run-time application self-protection
- Continuous vulnerability scanning
- Security patch application
- Security smoke testing

6.2.10. Deploy

The accepted application is deployed over production environment and smoke testing is performed to identify any functional and non-functional issues. The issues identified at this stage is more expensive to fix as compared to previous stages.

The deployment phase is not different to the DevOps as it incorporates the configuration and system integration as task [19]. The authors regard this phase as the most critical and the phase calls for the definition of secure deployment mechanisms and systematic configuration practices. The deployment phase is envisioned to be consisted of smaller parts. Morales et al. [24], work names these parts as application security monitoring, secure deployment process, secure environment, and secure operational enablement.

Security Controls in DevSecOps for Deploy phase are:

- Secrets management
- Infrastructure provisioning and orchestration
- Security patch application
- Infrastructure hardening and security testing
- Container and infrastructure security testing

6.2.11. Operate

The deployed applications and the production environment are monitored continuously for performance and malicious activities. This allows stakeholders to verify if the deployed security controls are functioning as expected. and take necessary action to address the deviations. Zero-day vulnerabilities are patched as quickly as possible to minimize exposure time. The team leverages infrastructure-as-code tools for patch updates, and

configuration changes to target machines at scale, providing speed, accuracy, and consistency with no human error.

The operation phase comes after the systems deployment in production. Hong [18] mentions that the verification compliance and control are tasks in the operational phase. The tasks in the operational phase are described by Mao et al. [25] and include automated security checks and monitoring feedback loops. In more details, red teams and bug bounties are essential in the operational phase to create a feedback loop between security teams and developers.

Security Controls in DevSecOps for Operate phase are:

- 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
- Monkey testing

6.2.12. Adapt

The ability to scale infrastructure on demand and replacing a compromised environment in minutes, drives the adaptability which is facilitated through by using right set of automation tools.

6.3. DevSecOps Practices

DevSecOps practices are a set of different activities that will activate security controls to perform security assurance tasks.

The list below describes DevSecOps Practices and target activities associated:

- Continuous Testing (CT)
  - SCA (Static Composition Analysis)
  - SAST (Static Application Security Testing)
  - Unit and integration testing
  - DAST (Vulnerability scan, PenTest, Exploit Test)
  - Acceptance, Smoke, Load and Performance Testing
  - IAST (Interactive Application Security Testing)
  - Infrastructure Configuration and Security Testing

- Continuous Planning, Design and Development (CPD)
  - Development Environment
  - Threat modelling and Security-by-Design
  - Source Version Control
  - Development Management
- Continuous Integration
  - Integration Automation
  - Build Automation
  - Artifact Repository
- Continuous Delivery
  - Configuration Management
  - Delivery Automation
- Continuous Deployment
  - Deployment Automation
- Continuous Operation
  - Logging, Analysis, Visualization & Notification
  - Continuous Monitoring
  - IDS, IPS & Security Information and Event Management
  - RASP (Runtime Application Self-Protection)
  - Infrastructure orchestration
  - Secret management
- Continuous Feedback
  - Collaboration & Communication Environment
  - Quality & Performance Measurements, Analytics, Trending & Alerting

7. Open-Source Software and tools

As indicated in the Section 6, software tools are mandatory in automating the DevSecOps methodology. In particular, the proposed DevSecOps framework for ASSIST-IoT will rely on open-source tools. **¡Error! No se encuentra el origen de la referencia.** lists a large number of tools that covers target activities that will enable to activate security controls and related practices as described in Section 6.3.

Table 1. OSS tools associated with DevSecOps practices

Practices		Target Activities	DevSecOps OSS tools by licence			
			Apache	GNU GPL	MIT	Others
Continuous Testing	CT	Code review	Gerrit		Review Board	
		SCA	Dependency Check, Synk	Bundler-Audit (GPL-3.0)	Bundler	FOSSA (MPL-2.0)
		SAST	Retire.js, Anchore, Docker, Bench, Clair	Find Security Bugs (GPL-2.0), SonarQube (GPL-2.0), SpotBugs (GPL-2.0)	DevSkim	PMD (Multiple)
		Unit and integration testing	TestNG			JUnit (EPL-2.0), xUnit Framework (Multiple)
		DAST (Vulnerability scan, PenTest, Exploit Test)	Zed Attack Proxy	Wfuzz (GPL-2.0), Wapiti (GPL-2.0), Nmap (Modified GPL-2.0), SQLMap (Modified GPL-2.0), Vuls (GPL-3.0), Sslyze (AGPL-3.0), BDD-Security (AGPL-3.0)		Metasploit (Multiple), Vega (BSD-3-Clause), Arachni
		Acceptance, Smoke, Load & Performance Testin	JMeter, Selenium		Cucumber	
		IAST (Interactive Application Security Testing)	Hdiv			
		Infraestructure Configuration and Security Testing	Cloud Custodian, InSpec, ChaosMonkey	Oscap (GPL-2.0)	Ansible-Lint, Puppet-Lint, Foodcritic, Serverspecs	Dev-Sec.io(Multiple)
Continuous Planning, Design and Development	CPD	Development Environment	NetBeans	Kdevelop (GPL-2.0)		Eclipse IDE(EPL-2.0), Eclipse Che(EPL-2.0)
		Threat modeling and Security-by-Design	ThreatDragon, CAIRIS			SeaSponge (MPL-2.0)
		Source Version Control	Subversion	Git (GPL-2.0), Mercurial (GPL-2.0)	GitLab	
		Development Management	Phabricator	Redmine (GPL-2.0), Tuleap (GPL-2.0), OpenProject (GPL-3.0)		
Continuous Integration	CI	Integration Automation	Concourse CI		GitLab, Jenkins	
		Build Automation	Maven, Gradle, Ant		GitLab	
		Artifact Repository	Archiva		GitLab	Nexus OSS
Continuous Delivery	CD	Configuration Management	Puppet, Chef, Saltstack	Ansible (GPL-3.0)	GitLab	Terraform(MPL-2.0), CFEngine (Multiple), BCFG2 (BSD-2-Clause)
		Delivery Automation	Spinaker, GoCD, Docker		GitLab	
Continuous Deployment	CDP	Deployment Automation	Drone	Foreman (GPL-3.0)	Capistrano	Fabric (BSD-2-Clause)
Continuous Operation	COP	Logging, Analysis, Visualization &Notification	Elasticsearch, Logstash, Kibana, ElastAlert, Grafana,	Graylog2 (GPL-3.0), Rsyslog (GPL-3.0)		
		Continuous Monitoring	Sysdig Falco, Prometheus	Nagios Core (GPL-2.0), Zabbix (GPL-2.0), Munin (GPL-2.0), Cacti (GPL-2.0), Icinga (GPL-2.0)	StatsD	CollectD (Multiple), OpenNMS (Multiple),Gangila (BSD-3-
		IDS, IPS & Security Information and Event Management		Fail2Ban (GPL-2.0), Snort (GPL-2.0), Suricata (GPL-2.0), OSSIM (GPL-3.0)		OSSEC (Multiple), wazuh (Multiple)
		RASP (Runtime Application Self-Protection)	Hdiv		AppSensor	
		Infraestructure orchestation	Kubernetes, OpenStack, Mesos, Swarm, Cloudify		Vagrant	
		Secret management	Git secrets		Blackbox, Transcrypt	Harshicorp Vault (MPL-2.0)
Continuous Feedback	CF	Collaboration & Communication Environment	Phabricator	Mattermost (AGPL-3.0), Tuleap (GPL-2.0)	GitLab	
		Quality & Performance Measurements, Analytics, Trending & Alerting	Elasticsearch, Logstash, Kibana, ElastAlert, Grafana, Graphite, seyren,	Graylog2 (GPL-3.0), Zabbix (GPL-2.0), Cacti (GPL-2.0), Icinga (GPL-2.0), LimeSurvey (GPL-2.0)		CollectD (Multiple), Sentry (BSD-3-Clause)

A different overview to analyse software tools for DevSecOps is *DevOps Periodic Table* [23], with a well-known list of more than 100 software tools, that provides context of the use and DevSecOps practices used by each tool as shown below in Figure 6.

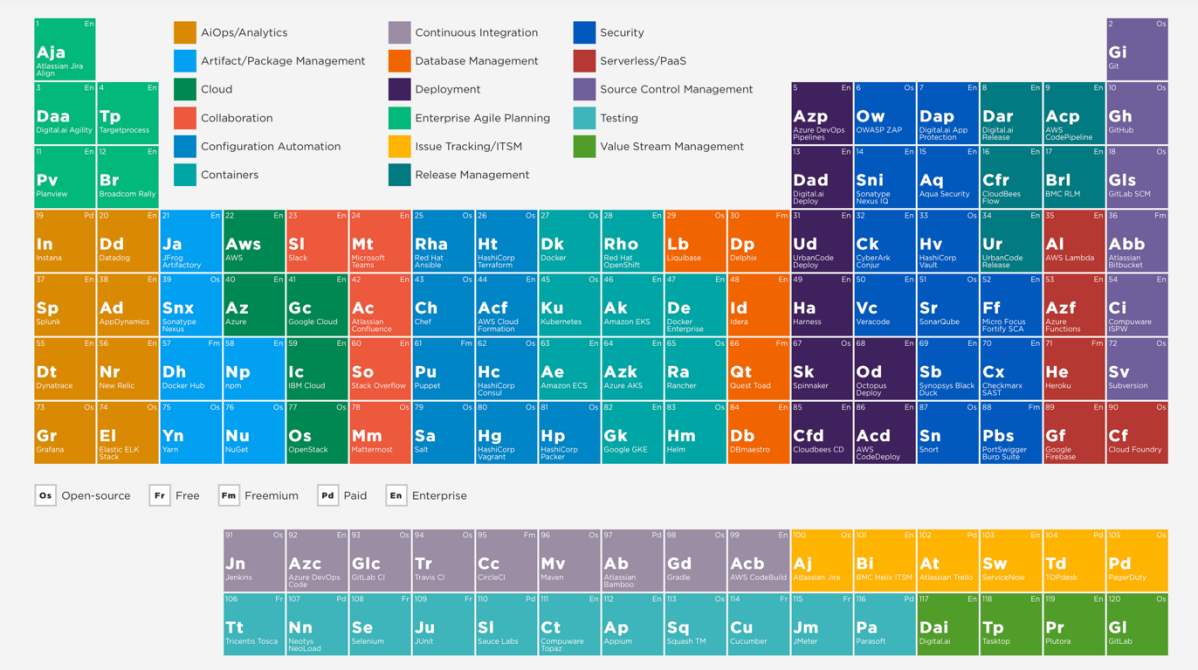

Figure 6. The Periodic DevOps table [23].

The next sections identify different tools associated with the different DevSecOps stages of ASSIST-IoT.

7.1. Tools for collaboration and communication environment

The DevOps strategy focuses on human collaboration across silos. Concretely, this entails different departments sharing knowledge, process, and practices, which requires sharing specific tools.

On the one hand, ***Gitlab*** not only provides a code repository but also has a wiki system that enables developers and operators to share knowledge. Gitlab offers an issue management system, which is typically used by developers and business analysts to share knowledge through agile practices like issues comments and pull requests processes. Making operators also familiar with these issues helps them to understand the product and project contexts. Moreover, documenting production problems known by operators in the issue management system is an essential step toward handling non-functional requirements prioritization in the development flow.

7.2. Tools for source version control and CPD

Tools for source version control are one of the key activities integrated in the practices for continuous planning design and development (CPD). Source code management tools usually intend to promote collaboration among developers. These tools are basic blocks to implement continuous integration and, therefore, continuous delivery. From this traditional perspective, one could relate source code management solely to the delivery category of concepts. However, source code management can also be used by operators to store artifacts and automation scripts, and to access software information that can impact operations activities. For example, a development bug may cause a memory leak that impacts operations. When storing infrastructure-related artifacts, whether in the infrastructure-as-code style or just as plain text, operators provide developers with better insight into how the software is executed. Therefore, source code sharing among developers and operators becomes a real point of collaboration. The most traditional tools in this category are ***SVN*** and ***Git***. More complete platforms, such as ***GitLab*** and ***GitHub***, can also wrap Git, providing easier-to-use visualizations for code changes, as well as integrating with additional tools like the issue manager.

7.3. Tools for build automation

Build tools are highly developer-centered. Their goals relate to enabling continuous delivery and the delivery category of concepts. This group does not only include tools that generate deployable packages, also called builds, but also tools that generate essential artifacts and feedback using the source code as input. Each programming language has some tools to cope with the build process by supporting dependencies resolution, implementation of custom tasks, or generation of deployable packages. Examples of dependency managers, also called package managers, are ***PIP*** for Python, ***RubyGems*** for Ruby, ***NuGet*** for .NET, and ***Maven*** and ***Gradle***, which compete in the Java landscape. ***Gradle*** eases the implementation of custom tasks during the build, as also done by GNU Make, which is often used for GNU/Linux packages, or ***Rake*** for ***Ruby***.

Some of the cited tools, such as Maven, are also responsible for producing the deployable package, such as WAR files for Java environments. However, for some languages, such as Python and Ruby, there is no need for producing a deployable package as a single-file artifact. It is also possible to use more generic package tools coupled to the target environment, such as Debian packages. Each programming language also has some unit-test frameworks that provide vital feedback for developers about the correctness of the software. Some examples are ***JUnit*** and ***Mockito*** for Java, ***RSpec*** for Ruby, and ***NUnit*** for .NET. More sophisticated testing that automates the end-user behaviour for web applications is possible with browsing automation tools, such as ***Selenium***. Another type of feedback for developers is regarding source code quality, which is provided by source-code static analysis tools, such as ***SonarQube*** and ***Analizo***. While SonarQube classifies code problems and evaluates coverage metrics as well as technical debts in several programming languages via its plugins, ***Analizo*** supports the extraction of a variety of source code metrics for C, C++, C#, and Java codes.

In general, source-code analysis tools can point to issues in source-code, such as non-compliance to the standard style, problems in maintainability, risk of bugs, and even vulnerability breaches. Source-code static analysis tools vary in supported programming languages and in the way they are delivered. They can be provided as a service, or even within the developer environment through tools such as PMD for the Eclipse IDE.

7.4. Tools for continuous integration

Continuous integration tools orchestrate several automated actions that, together, implement the deployment pipeline pattern. Among the stages orchestrated by the pipeline are package generation, automated test execution for correctness verification, and deployment to both development and production environments. These tools are related to the delivery category of concepts. The main actors responsible for defining the pipeline structure are typically developers. Operators usually collaborate on defining the deployment stages; they are also in charge of maintaining the continuous integration infrastructure running as a service to developers. For this, operators run continuous integration tools such as ***Jenkins*** or ***GitLab CI***. Deployable packages can be stored on repositories like ***Nexus*** for enabling future rollback. Since downtime of continuous-integration infrastructure results in the interruption of continuous delivery, it is common to use highly available third-party services, such as ***GitLab*** and ***Travis***.

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

Deployment automation tools are employed in the deployment stages of the pipeline to make the continuous delivery process viable. They enable frequent and reliable deployment processes, as well as other concepts related to the delivery category. Although the use of deployment automation tools is a joint effort between developers and operators, the primary mission of continuous delivery is putting the deployment schedule under business control. Every automated deployment approach relies on the concept of IaC. It requires engineers to specify servers, networking, and other infrastructure elements in files modelled after software code. In this model, deployment and infrastructure definitions are shared among developers and operators, allowing effective cooperation between them. Automated deployment can encompass not only the deployment of the package to the production environment, but also the provisioning of the target environment. Such provisioning is usually performed in cloud environments, such as ***Amazon Web Services (AWS), Google Cloud,*** and ***Azure***. These platforms deliver a vast amount of infrastructure services via Infrastructure as a Service (IaaS) model, such as

launching VMs, databases, queues, and so on. The creation of all these resources can also be orchestrated, on e.g., Amazon's platform, through the usage of AWS Cloud Formation. Operators and developers can share the task of using IaaS services. It is also possible to use a Platform as a Service (PaaS), such as ***Heroku***. In the PaaS approach, the platform is responsible for the deployment, and developers do not know the underlying virtual infrastructure.

Configuration management tools are important in cloud deployments when using IaaS, where the environment provisioning is followed by the execution of scripts that effectively install the application in the target environment. A deployment script can be written using Shell Script, but configuration management tools such as ***Chef*** and ***Puppet*** offer advantages by leveraging OS portability and idempotence mechanisms, which are difficult to achieve with Shell Script. An example of language construct of ***Chef*** that leads to portability is using the "package" resource, which is resolved to a concrete package manager, such as ***Apt***, only at execution time. An example of an idempotent mechanism of ***Puppet*** in the "service" resource is declaring the desired final state of the service as "running" rather than writing a command to start the service.

Another alternative for deployment is containerization, primarily implemented by ***Docker***. ***Docker*** containers resemble VMs. The main difference is that they are more lightweight, considering they share the kernel of the host. Docker and related tools, such as ***Docker Compose***, ***Kubernetes***, and ***Rancher***, allow the specification of containers and their dependency relationships. These specifications generate container images in the build stage, which are later instantiated in the target environment. ***Docker*** has been used not only for deploying applications but also for deploying the underlying infrastructure. Containerization could be seen as complementary to the deployment script strategy (done by ***Chef*** or ***Puppet***). However, in practice, these strategies seem to compete. A ***Chef*** script is executed continuously on the target node, and its success depends on the previous target node state. However, in containerization the whole containerized environment is generated in the build, so the environment is destroyed and rebuilt at each new software version. When compared to the ***Chef*** strategy, ***Docker*** yields faster and more reliable deployment, but at the expense of bigger builds. Complementary usage of configuration management and containers entails the setup of ***Docker*** environment and update of the OS software, like SSH. Another usage is to manage configuration that varies across diverse environments (testing, staging, production). Such a configuration cannot be embedded in container images since the same image must be deployed in every environment.

As the application evolves, the database structure evolves. Traditionally, "database administrators" maintain the database structure. The adoption of emergent design and the need for frequent releases have encouraged the use of database migrations tools such as ***Flyway***, which controls the automated application of schema updates across different environments, thus enabling developers to manage database structure themselves.

7.6. Tools for monitoring

Monitoring tools usually track applications' non-functional properties, such as performance, availability, scalability, resilience, and reliability. Self-healing, alerts, log management, and business metric follow-up are example tasks performed by monitoring tools; they relate to the runtime category of concepts. Examples of tools for monitoring and alerting are ***Nagios***, ***Zabbix***, and ***Prometheus***. Examples of log management tools are ***Graylog*** and ***Logstash***. Cloud services also play an essential role in guaranteeing non-functional properties of applications, since they provide elastic resources that can be allocated on demand. It is also typical for cloud services to provide monitoring and alerting services. The tendency toward cross-functional, full-stack teams, combined with the expectation that developers must be accountable for the product pushes the use of these tools to the development team.

8. DevSecOps ASSIST-IoT use case

The application of the DevSecOps methodology through the continuous security model is described in the following use case, that will need to be adapted and mapped to concrete tools but that is compliant with the approach and maps with software tools previously described.

The following figure presents the continuous approach for DevSecOps in ASSIST-IoT, from development environment to production environment. DevSecOps generic features tools are identified and highlighted in the diagram as previously identified in Section 7. The concrete tools will be further detailed during the next steps of the project within the context of WP6 Integration task.

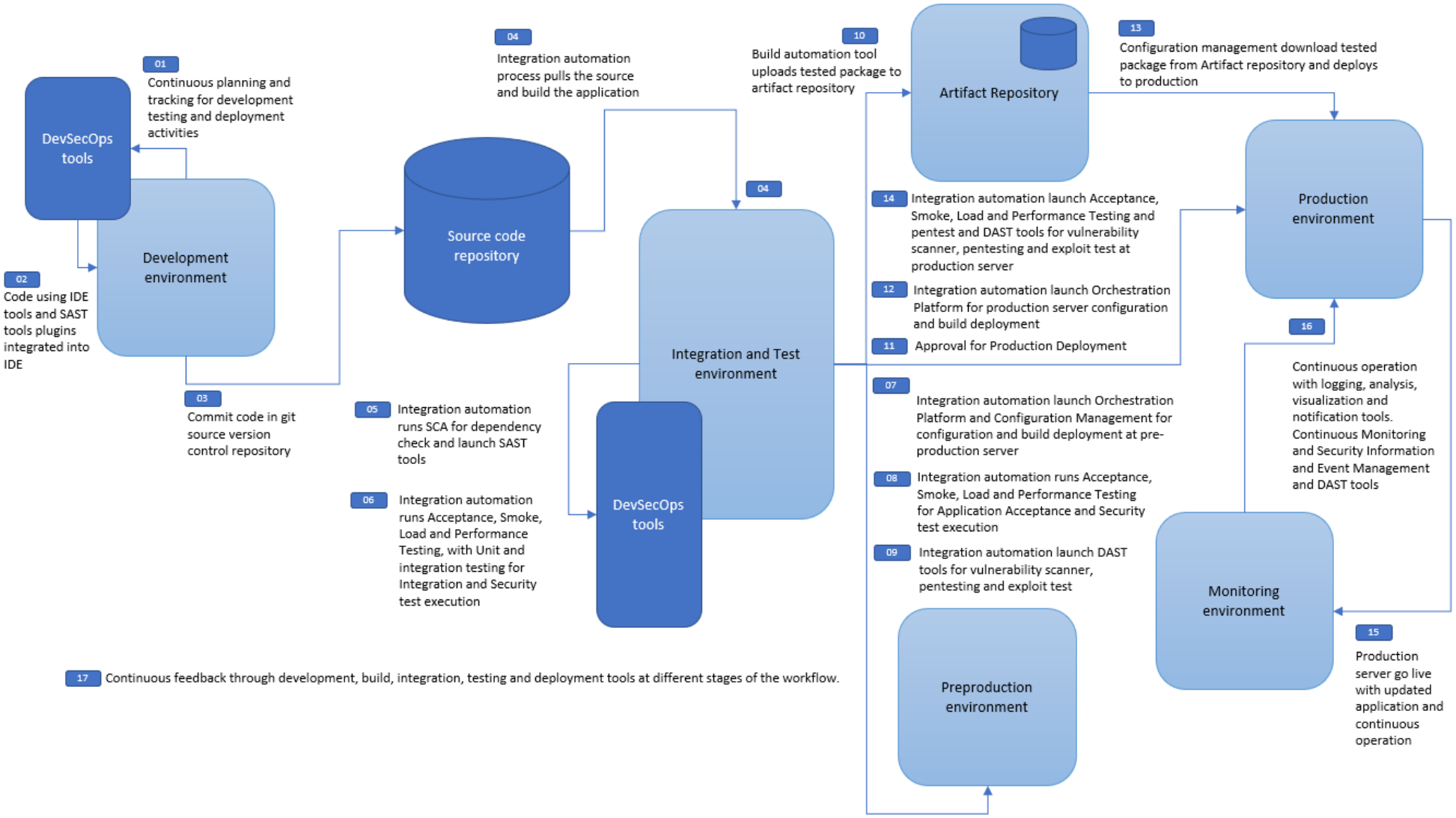

Figure 7: DevSecOps methodology for ASSIT-IoT use case.

The different steps describe the further work to possible selection of OSS tools, and will be useful to provide an overview on the idea of the implementation of the DevSecOps methodology.

1. Continuous planning and tracking for development testing and deployment activities.
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 runs SCA for dependency check and launch SAST tools.
6. Integration automation runs 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 runs Acceptance, Smoke, Load and Performance Testing for Application Acceptance and Security 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 launches Acceptance, Smoke, Load and Performance Testing and DAST tools for vulnerability scanner, pentesting 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 and Security Information and 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.

9. DevSecOps ASSIST-IoT guidelines

9.1. ASSIST-IoT GitLab

GitLab is a common and broadly used tool in DevSecOps corporate approaches for implementing and supporting DevSecOps methodology. As mentioned, and detailed in **¡Error! No se encuentra el origen de la r eferencia.**, GitLab takes part and contributes widely in different parts of DevSecOps methodology.

As cited in **¡Error! No se encuentra el origen de la referencia.**, GitLab is a tool that provides features to s upport DevSecOps methodology and practices described in this deliverable. Generically to CI/CD, but also in other practices mentioned generically in the methodology such as: CPD providing support to source version control activities, CI providing support to CI activities (integration automation, build automation, and artifact repository), CD and CDP with CF due to features supported. The DevSecOps practices and associated activities that could not be handled using GitLab CE will use additional software tools integrated and automated as much as possible as described in the **¡Error! No se encuentra el origen de la referencia.**.

Previous experience of some partners inside the consortium like S21Sec and CERTH, using GitLab in other funded H2020 projects like FORTIKA, where GitLab was used as a central source code repository, also as container registry, and to implement CI/CD pipeline to deploy a cloud marketplace of services, has been also a crucial point to assume the decision.

ASSIST-IoT developments will configure and use the GitLab Workflow, with the purpose to help ASSIST-IoT teams to work cohesively and effectively from the first stage of implementing software development in WP4 and WP5 tasks from developing software components associated to each of the subtasks, towards the WP6 Testing integration and support where the last stage, deploying implementation to pilot use cases will be done in WP7.

GitLab Community Edition [30] will be used as the first approach in ASSIST-IoT and the purchase of different features will be evaluated during the timeline of WP6 Testing, integration, and support tasks.

GitLab allows to support DevSecOps methodology described to cross-check committed code and work on the code collaboratively before moving it to the CI/CD pipeline. GitLab offers git repository management, code reviews, issue tracking, activity feeds and wikis. Contextualising the role in ASSIST-IoT's DevSecOps flow (as shown in Figure 8), GitLab has been chosen as it is a highly modular tool that allows the incorporation of "features" and "applications" to be seamlessly integrated in the DevOps flow. GitLab also incorporates a "DevOps overview" dashboard that allows to realise, in a glance, the activity taken place with regards to commits, deployments, etc.

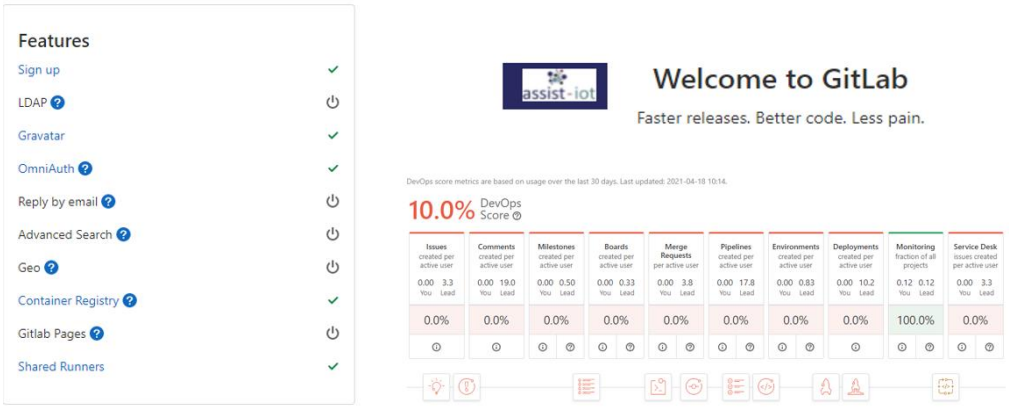

Figure 8: GitLab GUI

GitLab Community Edition [30] will be used as the first approach in ASSIST-IoT and the purchase of different features will be evaluated during the timeline of WP6 Testing, integration, and support tasks.

ASSIST-IoT team has agreed on applying DevSecOps methodology described in this deliverable with the support of Open-Source tools identified and selecting GitLab as DevSecOps open platform identifying and using the following main features

- GitLab CI/CD pipelines as top level components for continuous integration delivery and deployment.
- GitLab Package Registry: ASSIST-IoT will use package registry integrated into GitLab, registry for a variety of common package managers, enabling publish and share packages, which can be easily consumed as a dependency in downstream projects
- GitLab Container Registry: ASSIST-IoT will use container registry integrated into GitLab, to maintain docker images project developments associated to different GitLab projects associated to the different software developments and enablers, making possible that each of the ASSIST-IoT developments can have its own space to store its Docker images.
- GitLab Runner. ASSIST-IoT will use GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline.
- SAST tool for analysing known vulnerabilities on the code managed in the repository. This tool directly tackles the step of Continuous Testing practices outlined previously in the document

As a general guideline for ASSIST-IoT CI/CD, each software component should implement a CI/CD pipeline in GitLab. Some of the expected tasks for each of the software components developed and integrated into the CI/CD pipeline will be the following.

- Use and refine use cases in CI/CD pipeline using *gitlab-ci.yml*
- Provision a test environment (i.e. using docker container, and configuring environment variables)
- Perform a check out of the code using SAST scan
- Configure dependencies needed to test environment.
- Execute a test cases created by the code developers
- Create an environment to provide continuous delivery/continuous deployment using GitLab (i.e. from GitLab CD provision a docker container containing the software or application developed and deploy to staging, preproduction or production

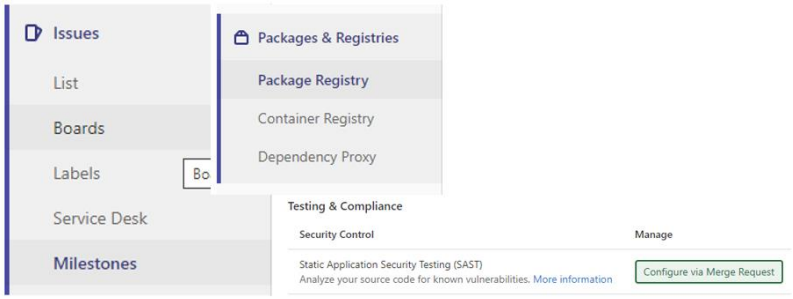

Figure 9: GitLab features to be used following DevSecOps practices in ASSIST-IoT

The ecosystem of DevSecOps software tools will be enhanced later in the project (scope of WP6) to cover all the steps arranged for the ASSIST-IoT DevSecOps methodology described in the present document.

9.2. ASSIST-IoT GitLab WP organization

With respect to the use of GitLab in ASSIST-IoT as a code repository, it has been devised as follows:

- On the top level, is the GitLab itself, dedicated to the ASSIST-IoT project, and accessible under: https://gitlab.assist-iot.eu/.

    - A thorough descriptive introduction (in the form of a README) will be provided, indicating all the enablers developed by plane and vertical, and including general guidelines regarding their deployment (related to infrastructure, e.g., Kubernetes based).
- Individual groups will be created (one per each WP –WP4, WP5, WP6, WP7 initially) within the ASSIST-IoT GitLab.
    - WP4 – Horizontal enablers (belonging to architectural construction)
    - WP5 – Vertical enablers (belonging to architectural construction)
    - WP7 – Pilot-specific enablers
- Inside the groups, different subgroups will also be created associated to each task, depending on the nature of the WP:
    - WP4 (in the sub-groups below will there only be the enablers associated to a specific plane).
        - Device and Edge plane enablers (T4.1) – *device-edge*
        - Smart and Network Control plane enablers (T4.2) – *network*
        - Data Management plane enablers (T4.3) – *data-mgmt*
        - Application and Service Plane enablers (T4.4) – *device-edge*
    - WP5 (in the sub-groups below there will be the enablers that fall under the scope of various planes, classified according to their functionality nature).
        - Self-* enablers (T5.1) – *self*
        - Federated Machine Learning enablers (T5.2) – *federated-ml*
        - Cybersecurity enablers (T5.3) – *security*
        - Privacy and trust enablers (T5.4) – *privacy-trust*
        - Manageability enablers (T5.5) – *manageability*
        - Other enablers. If needed, the sub-groups “Interoperability” and “Scalability” might be created at some point. However, this is not planned at this moment as (according to D3.5), those verticals will not have specific enabler implementations but rather are “properties” of the system.
    - WP7 (as software components developed for a specific pilot, such as GPS trackers for the port automation pilot). Although potentially falling under WP5 or WP4 scope, in this sub-group there will be the software components that are only created with the purpose of fulfilling one pilot – and not as own enablers of the architecture).
        - Port Automation Pilot (T7.1)
        - Smart Safety of Workers Pilot (T7.2)
        - Cohesive Vehicle Monitoring and Diagnostics Pilot (T7.3)
- Inside each subgroups of WP4, WP5, and WP7, the actual Git repositories will be created, associated to each software enabler. All the enablers will be accompanied by a description file, or pointing to it, detailing its requirements (software and hardware), deployment and usage. Examples:
        - wp4/network/sdn-controller
        - wp5/privacy-trust/auditing-tool
        - wp7/port-automation/gps-tracker

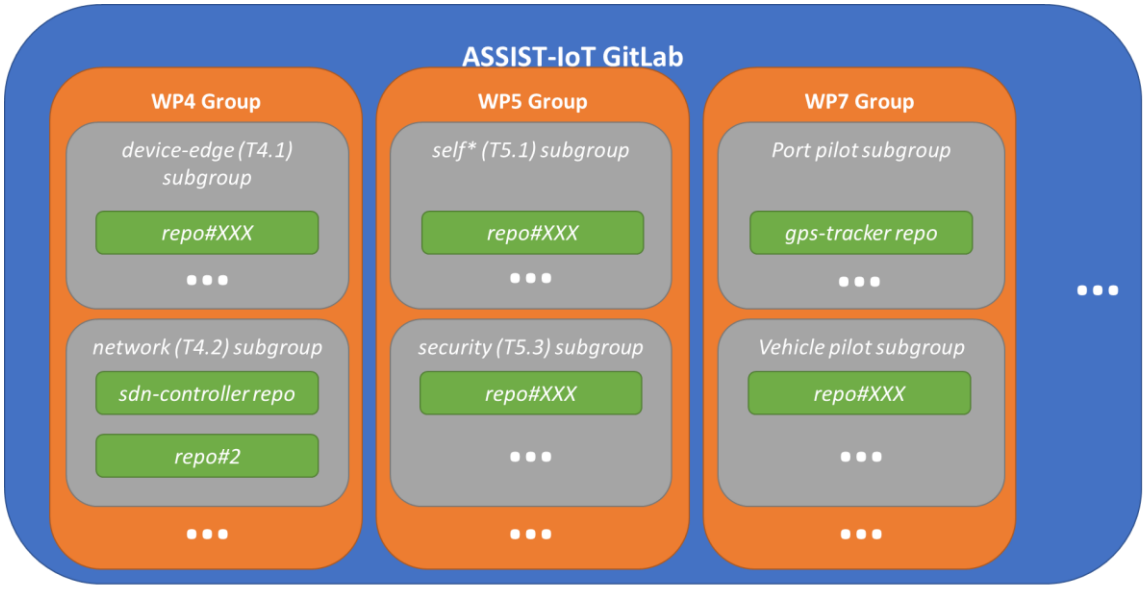

Figure 10: ASSIST-IoT GitLab internal structure/organisation

The software code provided by ASSIST-IoT in all sub-groups above will be continuously and automatically checked against any problems (syntactical, semantical, or functional). Additionally, GitLab also allows to issue quality assurance tests over the repositories. This part fits the step of *Continuous Testing* practices as part of the methodology exposed in this document, achieving then a seamless development-environment testing within the same space.

10. Conclusions

The deliverable described DevSecOps methodology to be used in the implementation of the ASSIST-IoT architecture and the work and further activities to be performed mainly on WP6 and WP7 applying continuous security on all the phases from development to operation that takes part in a SDLC.

The deliverable has analysed DevOps evolution to DevSecOps providing a methodology for implementing a continuous security framework and identifying target activities and security controls associated to each of the DevSecOps phases workflow. The methodology based on a continuous security model for DevSecOps is presented based on the following three supporting pillars: continuous workflow, open-source tools and cloud deployment technologies for supporting workflow activities.

Software tools are mandatory in automating the DevSecOps methodology. In particular, the proposed DevSecOps framework for ASSIST-IoT will rely on open-source tools. Target activities, and concrete security controls for each state of the DevSecOps workflow have been listed and these activities have been associated with DevSecOps practices from continuous testing to continuous feedback and including CPD, CI, CD, CDP, CO, where finally identifying open-source tools with related activities.

ASSIS-IoT use case for DevSecOps presents the continuous approach for DevSecOps in ASSIST-IoT, from development environment to production environment. DevSecOps generic features tools are identified and highlighted in a use case diagram identified in Section 7. Additional tools will be further detailed during the next steps of the project within the context of WP6 Integration task.

ASSIST-IoT team has agreed on applying DevSecOps methodology described in this deliverable with the support of open-source tools identified and selecting GitLab as DevSecOps open platform, identifying and planning to use the main core features provided to set up WP4 and WP5 software developments and further WP6 testing integration and support activities.

As a general guideline for ASSIST-IoT CI/CD, each software component should implement a CI/CD pipeline in GitLab.

The ecosystem of DevSecOps software tools will be enhanced later in the project (scope of WP6) to cover all the steps arranged for the ASSIST-IoT DevSecOps methodology described in the present document.

References

[1] Banica, L., Polychronidou, P., Radulescu, M., Stefan, C., 2018. When IoT meets devops: fostering business opportunities. KnE Soc. Sci. 3 (10), 250–264.

[2] Donkers, P. , 2019. MYST: Automated DevOps for distributed applications across heterogeneous Cloud, Fog and Edge infrastructures. University of Amsterdam Ph.D. thesis

[3] Hoque, S., De Brito, M.S., Willner, A., Keil, O., Magedanz, T., 2017. Towards container orchestration in fog computing infrastructures. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), 2, pp. 294–299.

[4] John, W., Marchetto, G., Nemeth, F., Skoldstrom, P., Steinert, R., Meirosu, C., Papafili, I., Pentikousis, K., 2017. Service provider devops. IEEE Commun. Mag. 55 (1), 204–211.

[5] Meirosu, C., John, W., Opsenica, M., Mecklin, T., Degirmenci, F., Dinsing, T., 2017. Devops: fueling the evolution toward 5g networks.https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/devops-fueling-the-evolution-toward-5g-networks

[6] Mimidis, A., Ollora, E., Soler, J., Bessem, S., Roullet, L., Van Rossem, S., Pinneterre, S., Paolino, M., Raho, D., Du, X., Chesterfield, J., Flouris, M., Mariani, L., Riganelli, O., Mobilio, M., Ramos, A., Labrador, I., Broadbent, A., Veitch, P., Zembra, M., 2018. The next generation platform as a servicecloudifying service deployments in telco-operators infrastructure. In: 2018 25th International Conference on Telecommunications (ICT), pp. 399–404.

[7] Oracle, 2019. A cloud-native journey for telecommunications.https://www.oracle.com/a/ocom/docs/industries/communications/cloud-native-journey-telecomm-wp.pdf

[8] Truong, H.-L., Berardinelli, L., 2017. Testing uncertainty of cyber-physical systems in IoT cloud infrastructures: Combining model-driven engineering and elastic execution. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Test- ing Embedded and Cyber-Physical Systems. Association for Computing Machinery, pp. 5–8

[9] Truong, H.-L., Klein, P., 2020. Devops contract for assuring execution of IoT microservices in the edge. Internet Things 9, 1–17.

[10] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. 2016. DevOps. IEEE Software 33, 3 (2016), 94–100. Code: A54

[11] The Incredible True Story of How DevOps Gots Its Name. 2014. https://newrelic.com/blog/nerd-life/devops-name

[12] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles, “A survey of DevOps concepts and challenges”, ACM Computing Surveys, Vol. 52, No. 6, Article 127. Nov. 2019

[13] The Eight Phases of a DevOps Pipeline https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba

[14] Carter, K., 2017. Francois Raynaud on DevSecOps. IEEE Softw. 34 (5), 93–96

[15] A Quick Guide to DevSecOps Pipeline.Nov. 2020. https://www.xenonstack.com/insights/devsecops-pipeline/

[16] DAST, SAST, IAST and SCA: Which security technology is best for me? Feb 2020 https://www.kiuwan.com/blog/application-security-tools-comparison/

[17] OWASP Top 10 Web Application Security Risks https://owasp.org/www-project-top-ten/

[18] Hong, J. K. (2019). Component Analysis of DevOps and DevSecOps. *Journal of the Korea Convergence Society*, *10*(9), 47-53.

[19] Mayoral-Vilches, V., García-Maestro, N., Towers, M., & Gil-Uriarte, E. (2020). DevSecOps in robotics. arXiv preprint arXiv:2003.10402.

[20] Björnholm, J. (2020). Performance of DevOps compared to DevSecOps: DevSecOps pipelines benchmarked!.

[21] Department of Defense. DoD Enterprise DevSecOps Reference Design. Aug 2019. https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf

[22] Apache License, version 2.0 https://www.apache.org/licenses/LICENSE-2.0

[23] V4 of the Periodic Table of DevOps Tools is LIVE!. Jun 2020. https://digital.ai/catalyst-blog/v4-periodic-table-of-devops-tools-is-live

[24] Morales, J., Turner, R., Miller, S., Capell, P., Place, P., & Shepard, D. J. (2020). Guide to implementing DevSecOps for a system of systems in highly regulated environments.
[25] Mao, R., Zhang, H., Dai, Q., Huang, H., Rong, G., Shen, H., ... & Lu, K. (2020, December). Preliminary Findings about DevSecOps from Grey Literature. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)* (pp. 450-457). IEEE.
[26] Rakesh Kumar, Rinkaj Goyal, July 2020 "Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud ADOC"
[27] Appdynamics, 2015. Keep calm and embrace devops. https://kapost-files-prod.s3.amazonaws.com/published/555271a4c12539dc18000118/ebook-keep-calm-and-embrace-devops.pdf
[28] Willis, J., 2010. What devops means to me. https://blog.chef.io/2010/07/16/ what- devops- means- to- me/
[29] Willis, J., 2012. Devops culture (part 1). http://itrevolution.com/devops-culture- part-1/
[30] GitLab Self-Managed Feature Comparison https://about.gitlab.com/pricing/self-managed/feature-comparison/