Journal of Computer Science and Cybernetics, V.37, N.3 (2021), 321–337
DOI 10.15625/1813-9663/37/3/16245

DEVSECOPS METHODOLOGY FOR NG-IOT ECOSYSTEM DEVELOPMENT LIFECYCLE
ASSIST-IOT PERSPECTIVE

ÓSCAR LÓPEZ[1], JORDI BLASI[1], MIKEL URIARTE[1], IGNACIO LACALLE[2], GONZALO GALIANA[2], CARLOS E. PALAU[2], EDUARDO GARRO[3], MARIA GANZHA[4], MARCIN PAPRZYCKI[4,*], PIOTR LEWANDOWSKI[4], KATARZYNA WASIELEWSKA[4], KONSTANTINOS VOTIS[5], GEORGIOS STAVROPOULOS[5], IORDANIS PAPOUTSOGLOU[5]

[1]*Research and Development Department, S21Sec, Zamudio, Spain*
[2]*Communications Department, Universitat Politècnica de València, Valencia, Spain*
[3]*Research and Development Department,Prodevelop, S.L., Valencia, Spain*
[4]*Systems Research Institute Polish Academy of Sciences, Warsaw, Poland*
[5]*Information Technologies Institute, Centre for Research and Technology Hellas, Thessaloniki, Greece*

Abstract. Current software projects require continuous integration, during their whole lifetime. In this context, different approaches regarding introduction of DevOps and DevSecOps strategies have been proposed in the literature. The present contribution proposes introduction of DevOps, an agile methodology for the development and instantiation of software platforms, with minimal impact in any kind of operations environment, for the Next Generation IoT deployments. Moreover, the novelty of the proposed approach lies in leveraging DevSecOps in different stages and layers of the architecture. In particular, the present work describes the different DevSecOps methodology tasks, and how the security is included within pre-design activities, such as planning, creation or adaptation, design and implementation activities, as well as in post-implementation activities such as detection and response. Without proper consideration of security and privacy best practices identified in this article, the continuous delivery of services using DevOps methodologies may create risks and introduce different vulnerabilities for the Next Generation IoT deployments.

Keywords. Devops; Devsecops; IoT, NG-IoT, Security controls; Software development.

1. INTRODUCTION

Over the last few years, the Internet of Things (IoT) has emerged as a promising technology paradigm to be potentially applied to almost any type of Industrial environment.

Dedicated to Professor Phan Dinh Dieu on the occasion of his 85th birth anniversary.

*Corresponding author.
E-mail addresses: olopez@s21sec.com ; jblasi@s21sec.com ; muriarte@s21sec.com (M. Uriarte); iglaub@upv.es; gongafor@inf.upv.es; cpalau@dcom.upv.es; egarro@prodevelop.es; maria.ganzha@ibspan.waw.pl; paprzyck@ibspan.waw.pl (M. Paprzyck); piotr.lewandowski@ibspan.waw.pl; kvotis@iti.gr; stavrop@iti.gr; ipapoutsoglou@iti.gr.

Benefits of its application might range from enhanced productivity and quality, up to reduction of costs, or more efficient use of resources. Nevertheless, due to the increase of the number of interconnected devices, the rise of IoT brings new challenges [10]. These challenges are, among others: (i) significant growth of volume of unstructured data sent by the IoT devices; (ii) high degree of heterogeneity of this large volume of data, leading to interoperability issues; (iii) scalability problems caused by explosive growth of the number IoT devices; (iv) increased need for near-real-time reactions, close to the places of data creation and/or data consumption, and (v) dependencies between applications developed in technological silos for individual IoT deployments. Additionally, it should be observed that traditional, centralised IoT architectures lack the necessary capabilities to handle new requirements for human-centric applications, decentralization, and moving intelligence to the far edge of the deployment. New trends include moving from the widespread use of cloud-based infrastructure models, which are dominated by leading Internet companies, towards IoT edge, supporting mesh distributed processing, low latency, fault tolerance and increased scalability, security, and privacy. Furthermore, the Alliance of Internet of Things Innovation (AIOTI) [3], also foresees a change in the models of managing and controlling the flow and transmission of data.

Those rapid advances in IoT ecosystems require more secure and private approaches, in terms of continuous integration and deployment. In particular, for the Next Generation IoT (NG-IoT) scenarios [37], there is a demand for highly decentralised ecosystems that need to be supported by security, privacy, and trust enablers to ensure proper protection. Furthermore, the human-centric approach that will characterise NG-IoT systems will require new ways of interacting with legacy IoT ecosystems as well as with humans, posing a whole new set of security and privacy related challenges. Overall, the security and privacy by design aspects will be the crucial pillars to maintain trust in NG-IoT technologies. Hence, they will need to be extended along the lifecycle of involved systems, by making use of secure software development practices and security operations.

The aim of this contribution is to outline how the DevSecOps methodology can be applied to NG-IoT deployments in general, and within the ASSIST-IoT project [1] that aims to tackle the mentioned challenges, in particular. Figure 1 outlines (on the meta-level) the multilevel architectural approach of ASSIST-IoT, which is proposing a decentralized architecture for NG-IoT, combining the deployment of components on the edge and even near the far edge. As a summary, the ASSIST-IoT project designs, implements, and validates an open, decentralised reference architecture, with its associated enablers, services and tools, to assist human-centric IoT applications in multiple verticals. ASSIST-IoT will deliver, in a realistic, measurable, and replicable way, a unified innovative multi-plane (semi-) autonomous edge-to-cloud-continuum architecture for the future IoT deployments. ASSIST-IoT proposes to be primarily based on open source software technologies, relying on the most recent trends on microservices, containerisation, and orchestration, supplemented by cross-cutting digital enablers. ASSIST-IoT proposed architecture supports continuous integration and long-term sustainability of domain-agnostic, interoperable, self-* capable, intelligent, distributed, scalable, secure, and trustworthy IoT ecosystems.

Technical components of the architecture will rely on smart distributed software (and/or hardware) components (namely enablers), providing self-managed and automated capabilities. Those provisions will allow ASSIST-IoT-powered systems to respond with a more

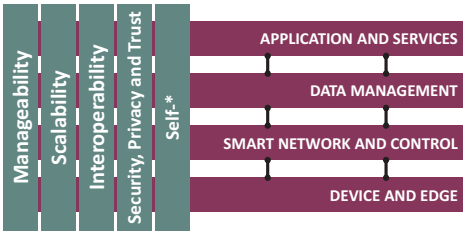

Figure 1: ASSIST-IoT multilevel approach architecture [4]

user-centric approach to the technology, with a decentralized security and privacy by default. Here, ASSIST-IoT's perspective will be applied to provide the foundations for the proposed DevSecOps approach.

The rest of the paper is structured as follows: Section II presents the methodologies in software development, and efforts in formulating requirements for IoT security. Afterwards, while Section III provides an overview of the strategies of DevOps and DevSecOps in general, Section IV specifies the different stages that constitute both methodologies and their application to NG-IoT deployments. Next, Section V describes the approach that has been proposed for ASSIST-IoT, in terms of security controls and DevSecOps tools to be used. Finally, Section VI concludes the paper with the presentation of the achievements and proposals of future work.

2. SOFTWARE DESIGN APPROACHES

Software development is a complicated process that calls for cooperation between professionals from multidisciplinary fields. For this reason, methodologies are conceived to establish the fundamentals for the efficient management of such projects. Therefore, organizations have different methodologies to choose from, based on their requirements and focal points.

As different methodologies exist in software development, Gitlab has documented the use of each methodology in a survey [12]. The most practiced methodologies are DevOps/DevSecOps and Agile, while methodologies like Kanban, Waterfall, and Lean are significantly lagging behind the most widely adopted ones.

One of the earliest software development methodologies is the *Waterfall*, introduced by Dr Royce [31]. As the methodology's name implies, it is a linear sequential flow of the processes involved in software development. Each phase commences only after the conclusion of the previous phase, which results in having a working software late into the cycle. The methodology has reported drawbacks like high cost and effort, along with late integration and testing [27].

Kanban is a methodology originating from the Japanese manufacturing sector, while its name translates into a signboard [2]. Kanban was initially associated with software development by David J. Anderson, in his work with smalls teams in Microsoft [24]. The methodology's beliefs are (i) work visualization; (ii) limitation of the work in progress; (iii) explicitness of management policies; (iv) adoption of scientific methods; and (v) continuous improvement.

The ***Lean*** methodology sets principles for software development by applying the Toyota Product Development System [28]. In brief, the Lean principles are the elimination of

waste, quality delivery, knowledge creation, fast delivery, respect for people, and continuous optimization.

Agile is one of the most practiced methodologies in software development. It incorporates multiple frameworks, such as Scrum. The methodology was popularized by the publication of the Agile manifesto [18]. The principles set in the manifesto focus on individuals and interactions, working software, customer collaboration, and response to changes. Similarly, other works share the same principles [36], or define the term agility [15].

Despite the comparable popularity, DevOps differs from Agile methodology. The first one focuses on pipeline optimization, while the latter is a project management strategy [32]. DevOps is a term that has been around for a bit over a decade, but a generally acknowledged definition of the approach is absent. The most cited definition for DevOps, available in the literature, describes the methodology as *"a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability"* [21].

In the efforts to continuously improve the DevOps methodology, an extension emerged to cover also issues related to security, leading to the so-called DevSecOps methodology. Security was traditionally considered at the final stages of the DevOps pipeline. In contrast, DevSecOps is a conscious effort to swift the security considerations towards the development pipeline, to address adequately and timely any security considerations. Specifically, in [30] DevSecOps is defined as DevOps embedded with security controls, aiming to provide continuous security assurance in all stages of the workflow. Hence, a set of additional secure rules to aid organizations to implant security deep in their DevOps development and operations processes is composing the DevSecOps [34].

As the principles for software development are set by all the aforementioned methodologies, there are efforts in defining security requirements in IoT deployments. The following statements are briefly outlining these considerations, as stated in in IoT Security Compliance Framework [13]:

- IoT sensors in machines that provide data are assumed to be reliable;
- Workstations used to access collected data from machines are assumed to be reliable and operative;
- Software on the servers deployed in the IoT service framework is reliable and continuously updated;
- Servers and IoT services framework are reliable with data protection and privacy concerns, and reliable applying appropriate security mechanisms at network level;
- Users of the software are authenticated and authorized, and the owner of the provided service is also considered to be trustworthy.

Secure software and firmware updates are a technical measure for IoT, associated with authorization, as described in ENISA Good Practices for IoT [8]. They aim at mitigating different threats associated with failures, malfunctions among other operational threats related to IoT devices. Software distribution should be controlled in IoT environments, not

only associated with authorization but also on the software update process. Overall, DevOps paradigm deals with the above statements related to software distribution and secure operation.

3. DEVOPS AND DEVSECOPS IN IOT

The following section describes how DevOps and DevSecOps methodologies can be applied and materialized in the deployment of NG-IoT ecosystems, considering the needs of permanent updates that NG-IoT components will require, and also the needs related to collaboration and automation. DevOps and DevSecOps methodologies have as their main objective to facilitate integration, delivery and deployment in a continuous cycle. This approach, understood as a continuous loop methodology is to enable the delivery of NG-IoT security ready applications and services.

3.1. DevOps in IoT

It has been noticed that IoT environments are usually dependent on smart networks and network functions, built upon Infrastructure as a Code (IaC), and with strong software dependencies on the deployment. Smart IoT devices need to manage efficiently the frequent software and firmware updates, since there are several threats associated with software distribution that should be controlled in IoT environments. In parallel, edge and cloud deployments need tools to support automation for implementing a workflow that can be automated, tested in a continuous workflow, reproducible and repeatable. Security by design principle is one of the most relevant approaches to be considered for software development and to perform a secure operation of the software in the deployment phase [9].

Following the systematic analysis of the literature carried out in [21], the conceptual framework of DevOps is composed of a conceptual map outlining four categories: (i) process which encompasses business-related concepts, (ii) people which covers skills and concepts regarding the culture and collaboration, (iii) delivery related to the Continuous Integration / Continuous Delivery and Continuous Deployment (CI/CD) concept[1] , and (iv) runtime that synthesizes concepts needed to guarantee the stability and reliability of services in a continuous delivery environment. Consequently, DevOps methodology deals with the statements related to software development, and software distribution in a project management. Traditionally, DevOps [20] is the combination of best practices in software development and IT operations, provided to shorten software development, or System Development Life Cycle (SDLC), enabling continuous delivery without impacting quality [22]. In addition, DevSecOps methodology introduces security by design among other principles to be applicable in the former DevOps methodology.

The DevOps practices are organized as follows: Continuous Planning, Continuous Integration, Continuous Delivery, Continuous Deployment, Continuous Operation and Continuous Feedback ([4, 30]):

[1]The “CD” in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they are sometimes used separately to illustrate just how much automation is happening.

- **Continuous Planning**: In continuous planning, all the stages of a workflow are taken into consideration. It involves planning, execution, and monitoring of different activities of development, testing, release, deployment, and operations phases. The tasks related to people, process, and technologies, like priorities management, budget allocation, resource management (allocation, education, collaboration, etc.), process enforcement (threat modelling, design review, code reviews, access and privileges, change management, vulnerability and patch management, etc.), environment setup, integration and infrastructure availability, communication, sharing, governance, etc., are identified and planned for execution at relevant stages in the software development workflow.

- **Continuous Integration**: DevOps aims at supporting collaboration among Development and Operations teams. However, coordinating a software development team, where many developers work simultaneously on a single codebase, can become an issue. A shared code repository is the most natural solution to this problem. Continuous integration (CI) aligns with the Code and Build phases of the DevOps pipelines. Generally, it refers to performing all of code tests, unit tests, and integration tests at each stage. By merging smaller changes more regularly, the issues become smaller and easier to manage, improving overall productivity.

- **Continuous Delivery**: Automates the process of deploying new builds 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 stability, performance, security compliance testing of these environments; and (iii) to deploy a new release into production, when approved and manually triggered by the organisation. Continuous Delivery embraces the Test and Release phases of the pipeline (described later in more detail), allowing organisations to manually trigger the release of new builds as regularly as possible.

- **Continuous Deployment**: It is an advanced version of Continuous Delivery, where the manual step of approving new releases into production is now automated. It involves the Test, Release, and Deploy phases of the pipeline. In the Continuous Deployment model, each build that passes all the checks and balances of the pipeline is automatically deployed into the production environment.

- **Continuous Operation**: Helps to monitor and to adapt changes of the stages of the workflow. The objective of continuous operation is to ensure service continuity. It involves automated continuous logging, scanning, monitoring, event correlation and analysing system and application events. Continuous operation requires also continuous scanning for the vulnerabilities and anomalous events. Infrastructure orchestration tools are used to scale according to the service demand and enables continuous operation with an infrastructure prepared to respond to different failures of services deployed.

- **Continuous Feedback**: Traditionally included in DevOps in the operation and monitor phase, continuous feedback practices during operation and monitoring is important to get the relevant feedback based on monitoring observation and performance data analytics.

Figure 2: DevOps workflow

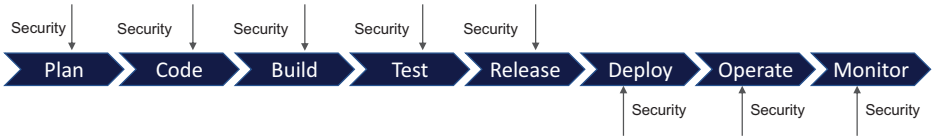

Figure 3: DevSecOps workflow ("shifting security left")

3.2. From DevOps to DevSecOps

DevOps has been mainly centred on harmonizing the interplay of development and operations, with the goal of institutionalizing CI/CD. However, as it can be seen in Figure 2 [16], since security is not a priority in DevOps processes, it only involves the security team in the latter stages of development.

Conversely, DevSecOps [14] considers and elevates security as a key element, to be considered during all stages of the workflow and is devoted to the fundamental principle of "shift-security left". In other words, it introduces security controls at early stages of the development cycle (also considering security by design), while involving security as early as the planning phase. The change is clear in the DevSecOps workflow (see Figure 3) that injects security into the traditional DevOps operations processes. In this fashion, security is integrated along the continuous DevOps workflow, with specific security activities involved in the earlier development processes, moving security to the left part of the workflow (without losing the benefits or controls of classic DevOps approach).

DevSecOps principles are broadly covered in [5], and also agreed in [30], and can be captured using the following key-phrases: "Shift-security left", "Security by Design", Culture, Automation, Metrics, Security as Code, Infrastructure as Code, Compliance as Code, and Adaptative security. In that sense, DevSecOps application in NG-IoT architectures has the potential to contribute in reducing and mitigating threats during software delivery in IoT deployments, while helping guarantee secure operation by applying its most relevant principles such as: shift-security left, and security-by-design. The main extensions of DevSecOps over the DevOps principles are summarised below:

- **Continuous Testing**: Testing controls run in parallel with other security controls in different stages of the workflow. While traditional DevOps tests included functional and non-functional testing techniques, DevSecOps introduces additional security testing techniques, like Software Composition Analysis (SCA), Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), and Runtime Application Security Protection (RASP). These tests are applied at different stages of the DevSecOps workflow using different tools. In this way they cover all stages of software development process.
- **Continuous Feedback**: Additional to DevOps, DevSecOps also extends the continuous feedback to other stages of the workflow, to obtain feedback from the Build and

Test phase, in order to include relevant security reports.
The security "addons", conceptualized within the DevSecOps, infused into the rest of the DevOps principles cited before as Continuous Integration, Continuous Delivery, Continuous Deployment and Continuous Operation are described in the next section, and summarized in the implementation of concrete security controls associated to each of the stages of the workflow.

4. DEVSECOPS STAGES FOR NG-IOT

Before analysing in-depth the proposed DevSecOps strategy for ASSIST-IoT, it is worth mentioning how are usually those plans tackled in DevOps. The most frequently phases adopted in DevOps culture are Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor, and they will be also applied in ASSIST-IoT.

4.1. Plan, Code and Build phases

The ***Plan stage*** covers everything that happens before the developers start writing the code. In NG-IoT environments, security requirements' analysis needs to be deeply considered during the planning phase in order to avoid late mistakes (incurring in inefficiencies and additional temporal and resource costs). Therefore, the DevSecOps perspective fits perfectly at this point, considering the proposal of "security by design" and "shift-security left". Software threat modelling approaches should also be applied, as a way of understanding the risk that the developed software will face. A software threat modelling will identify security threats that apply to NG-IoT software components and will study how to mitigate them, during the deployment phase. Use of threat modelling [26], Data Flow Diagrams (DFD) [29], and well-known vulnerability frameworks like OWASP TOP 10 vulnerabilities [25], or MITRE ATT&CK [23], can help avoid the most common vulnerability issues, when developing and exposing APIs, or end-user applications. These considerations on studying the way that software will be used before the coding phase commences will build a late understanding of security use and misuse of the application[2] supporting the development of abuse test use cases. It must bear in mind that the Plan stage will not be fixed but dynamic, evolving during the lifecycle of the application. Thus, it must be designed flexible enough to be adaptable to the deployment/integration environment.

During the ***Code phase*** (see Table 1), the coding guidelines are checked and enforced through the available plug-ins, for the Integrated Development Environment (IDE) used by the developers. Software inventory management is something that will allow quality results, while security will have to be implemented on the Source Code version control system.

Additionally, peer reviews of code are also considered as best practices to reduce software vulnerabilities. Functional testing should be expected to focus on the functional requirements of the software. Unit, Integration, and other functional testing processes should be done also during the code phase. Finally, test driven development considers performing tests during the Plan and Code phase. More concretely, unit testing will examine individual methods and

[2]Note that when using the term "application", authors aim at expressing any software artifact within an IoT system worthwhile to be controlled under a DevSecOps methodology (a piece of software, SDN rules, specifications, program configuration, etc.). This comment applies to the whole manuscript.

Table 1: DevSecOps security controls for Plan, Code and Build phases adapted from [30]

Security Controls for Plan, Code and Build phases
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
Source code version control security
Unit and integration security testing
Static application security testing (SAST)
Unit and integration security testing

functions, components and modules used by the software. Unit tests are produced by the software developers who will write different test cases to test the code. Integration testing verifies that different modules or services are working well together and are more expensive due to require multiple parts of the application up and running. The different type of tests mentioned will need to be integrated and automated into the continuous integration process.

The ***Build phase*** starts with compiling the changed source code, while resolving all the dependencies. Under the DevSecOps perspective, it is recommended to apply Static Composition Analysis (SCA) as a process for identifying the use of third party and open-source software components. Additionally, Static Application Security Testing (SAST) will analyse source code to find security related vulnerabilities.

4.2. Test, and Release phases

The ***Test stage*** consists of the integration of application software components and hardware infrastructure into a single system. In the case of edge devices, which is a key aspect in the NG-IoT scenarios, integration should be carefully tested, considering associated technologies, e.g. a container scanning analysis if the application is packaged in a container, and the IaC analysis used for the integration and automation. Besides, system integration security tests need to be conducted (see Table 2) at this stage, such as Fuzzy testing, which has a similar approach to Dynamic Application Security Testing (DAST), and it is black box oriented with no access to the software code. Furthermore, penetration testing needs to be considered, along with security smoke testing, and security patching. Going beyond, Interactive Application Security Testing (IAST), which will also be performed at this stage, follows the same line as the previous by introducing an instrumented app environment. Once the software is ready to be deployed to production, the final system should be tested to check that it is valid. This should include a wide range of tests, such as load tests, stress tests and usability tests. In terms of security, monkey testing deals with applying a series of random interactions with the application, and finally providing new validation status.

Afterwards, the application is packaged and stored in an artifact repository, in order to be delivered and accepted later at the ***Release stage***. Therefore, in DevSecOps methodology, an artifact repository security management should be also considered. In this sense, the introduction of Runtime Application Self-Protection (RASP) technologies can be also placed to improve security whenever the used framework supports it. This technology, combined

Table 2: DevSecOps security controls for Test and Release phases adapted from [30]

Security Controls for Test and Release phases
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

with traditional perimetral protection, will enhance the application protection by: (i) monitoring the inputs received, (ii) considering the contextual environment, and (iii) protecting them from those that may result in a threat to the application and the environment.

Finally, as code is progressing from different environments, and credentials, or keys, vary from one to another, a proper "secrets management" strategy and tool should be included to guarantee the software code does not reveal credentials or secrets.

4.3. Deployment, and Configure phases

DevSecOps methodologies will use different environments when implementing the pipeline. Typical environments examples could be development, staging, preproduction, and production. The most important reason to have different environments is the performance of different tests and the validation acceptance. The ***Deployment*** and, later, ***Configure phase*** (see Table 3) should be done in the staging environment, which is a replica of the production environment, where the application is configured with necessary configuration data, for acceptance testing. At Configure 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 preferably 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.

4.4. Operation, and Monitoring phases

During the ***Operation phase***, the developed software will be deployed in a production environment. To provide a reliable environment, the infrastructure, being host-based or container-based, should go through a hardening process and one more round of security operation. Therefore, in order to ensure a proper Operation execution, it is required to follow a formal DevSecOps methodology to be security compliant.

At the ***Monitor phase***, different practices for continuous monitoring with logging, analysis, visualization, and notification tools need to be considered. Continuous Monitoring and

Table 3: DevSecOps security controls for Deployment and Configure phases adapted from [30]

Security Controls for Deployment and Configure phases
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
Secrets management
Infrastructure provisioning and orchestration
Infrastructure hardening and security testing
Container and infrastructure security testing

Security Information and Event Management along with penetration testing, DAST, IAST, fuzzy testing, continuous vulnerability scanning already introduced before. Later and advanced DevSecOps activities include external security Red Team and internal security Blue Team activities, as an effective way to test that the system implemented, presents detection and response capabilities in production. Adequate monitoring concludes with incident management, main objective of which is to give a proper treatment to every detected abnormality. This should include a detection and response process as well as metric analysis.

5. ASSIST-IOT DEVSECOPS APPROACH AND ESSENTIAL TOOLS

Collaborative work is a centrepiece of NG-IoT architectures design and deployment. The short concept-to-market and development-to-production times, altogether with strong software dependencies between different components of the architecture, force NG-IoT teams to work in distributed environments. The ASSIST-IoT project proposes a DevSecOps methodology for NG-IoT, which is based on two fundamental pillars [30]:

- Applying practices and principles embedded with security controls;
- Selection of open-source tools that will perform target activities for each of the DevSecOps practices.

In the following paragraphs, an overview of the selected DevSecOps practices specified for the ASSIST-IoT's strategy is outlined:

1. During the implementation of DevSecOps methodology in any scenario, and particularly in the case of NG-IoT- the actual "staging" environments are as follows: 1) Integration and test environment, 2) Preproduction environment, and 3) Production environment.
2. Some of the expected tasks, for each of the software components developed and integrated into the CI/CD pipeline, are likely to be the following:

Table 4: DevSecOps security controls for Operation and Monitoring phases adapted from [30]

Security Controls for Operation and Monitoring phases
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

(a) Using and refining use cases in the CI/CD pipeline, using a sequence of steps or jobs normally described in a YAML file for describing the pipeline in GitLab or CircleCI or in a Jenkins file that will describe pipeline steps;
(b) Provisioning of a test environment;
(c) Performing a check out of the code using SAST scan;
(d) Configuring dependencies needed to test the environment;
(e) Executing test cases created by the code developers;
(f) Creating an environment to provide continuous delivery/continuous deployment using appropriate automation tools for testing with DAST; and, finally
(g) Building artifacts in different forms (i.e., as containers or other software packages) containing the software or application developed to further deploy to staging, preproduction and production.

3. Associated and related to the described pipeline steps, and considering the DevSecOps methodology, there are some essential features that need to be covered with the appropriate tools (to address the following needs):

   - IDE tools that facilitate the first building and debugging of the code and the integration with SAST tools. As already mentioned, and crucial for the paradigm of "shift-security left", it is very important that developers could detect security errors in the code they are developing in the Code phase and even previous to their first commit to the repository. This test could be performed by integrating security tools like SAST into the IDE environment. As a reference tool for IDE, it is worth mentioning, among others, Visual Studio Code [35] as the code editor for writing, building, and debugging web and cloud applications. It has the advantage of tight integration with a broad range of cloud service providers (e.g., AWS, MS Azure, etc.), and works with a vast ecosystem of extensions, with the

choice to include SAST scan tools like Shift-Left SAST [33] that includes features such as an integrated multi-scanner based design, to scan and detect various kinds of security flaws.

- Version control is a crucial method of tracking and managing changes to code that must be followed in all cases. Version control allows developers to see the complete revision history of a project and revert to a former version or file if needed. Git [6], the most utilised technology, is a free distributed Version Control System (VCS) that utilizes branching and merging. Moreover, as already mentioned, one of the DevSecOps principles is the development in a collaborative environment, where developers make daily/regular updates on the main branch, including changes completed by the rest of the team. Branching and merging code are the main Git features that stand out from other software code management tools. Their availability means that Git supports the branching model. Git is a widely used tool as it is admittedly used by almost 85% of respondents in the Gitlab's report. There are also different Git implementation alternatives, but all of them require a hosting service for the software repositories, that can be done on premises, using internal servers, or using external cloud deployments, depending on the strategy that better suits the deployment requirements. GitHub [11] is a software code management tool platform where hundreds of millions of private, public, and open-source repositories are posted and reviewed. GitLab is another alternative broadly extended as the most-used hosting service.
- As a general guideline for NG-IoT software deployments, which follow the DevSecOps methodology, they will consequently need to implement CI/CD processes, with their associated CI/CD tools. In particular, to implement pipelines to Build, Test and Deploy facilitating continuous integration and continuous delivery. GitHub or GitLab include different alternatives to implement CI/CD pipelines. More in detail, GitHub is increasingly expanding its offerings to align with more and more processes in the DevOps workflow implementing CI/CD features using GitHub actions [11]. GitLab was one of the first hosting services to fully embrace DevOps and has since been on a mission to create a complete DevOps platform. GitLab provides everything to manage, plan, create, verify, package, release, configure, monitor, and secure your applications. Nevertheless, there are more alternatives that can make use of other specific tools like CircleCI [7], or Jenkins [17], which will cover the automation feature to implementing CI and CDel pipelines.
- Adequate tools for packaging, taking into consideration the business needs for each environment and using a package registry to facilitate the software deployment. GitLab and GitHub also offers several package registry solutions that will enable the uses for a variety of packet registries for Docker and other package distributions and common package managers, enabling publishing and sharing packages. Further work will also concentrate on ways to continuously deliver and automate deployments based on solutions to orchestrate container environments (i.e., approach for packaging applications inside containers) deployed on the edge. As the most predominant technology in the field of container orchestration is Kubernetes (K8s) [19], a paramount step that must be taken in ASSIST-

IoT is to align the overall DevSecOps approach with the different K8s variants and flavours. This will be specially considered when aiming at generalizing the methodology for a "generic" edge-cloud deployment as NG-IoT architectures require (as is the case of ASSIST-IoT).

- SAST tools for analysing known vulnerabilities on the managed code. They are preferable to be integrated into the source code version management repository, as already mentioned, and built and designed for DevSecOps workflow integration shifting security to the left and to be able to detect security code vulnerabilities at every stage of the process. SAST scan [33] can be integrated into GitLab CI/CD pipeline, implementing security policies that will break the Build phase when developer commit the code into the Git repository if parameters configured in the security policy are not fulfilled. The results are presented with a score on critical, high, medium and low level, associated to each test performed, inside a GitLab job. Apart from source code analysis to find security and code style issues, SAST analysis can include the following characteristics: credential scanning to detect accidental secret leaks, audit of open-source dependencies for known common vulnerability and exposures (CVEs), checking for license violation, and container image scanning for application CVE. SAST tools have also good performance in finding keys and certificates uploaded to git repositories. It is also able to detect the use of versions of libraries with security issues, and security threats related with the source code (e.g., with the taint checking). Finally it can detect data used in the software application that is exported directly to log files.
- DAST tools for analysing the software, looking for security vulnerabilities on runtime will be also used in ASSIST-IoT DevSecOps processes, so that the software can be tested from the outside. DAST analysis evolves checking not only Application server security configuration, but also testing the web Application Programming Interface (API), or endpoints to ensure that the security of the application cannot be compromised by a malicious use of the API, using parameters out of the API specification. DAST tools, can detect failures with the SSL connection, due to issues with certificates or HTTP server configuration, and also miss configuration of the HTTP server, for example containing multiple index files.

6. CONCLUSIONS AND FUTURE WORK

This work has explored the possibilities to manage Next Generation IoT CI/CD operations, development, and security, and has outlined some approaches towards increasing process control, while using DevSecOps, without losing the agility and benefits that DevOps offers. It has also reflected on how DevSecOps practices can be exploited in managing independently addressed verticals. Traditionally, DevOps and DevSecOps strategies have been applied to the development of different services, while this work provides strategies on how to apply them across forthcoming NG-IoT architectures.

Applying DevSecOps methodology (and selection of controls) along with use case implementation guidelines should enable reducing and mitigating threats to software delivery and operation of NG-IoT environments. The analysis has provided a comprehensive overview of

steps that can be applied in DevSecOps, in the context of NG-IoT. In the discussion, the approach championed by the ASSIST-IoT project provides the real-world anchor. Validation of the proposals presented in this contribution will take place during the ASSIST-IoT project implementation and is expected to identify the most relevant DevSecOps practices in this area, resulting in a conceptual model for NG-IoT components' development and deployment, following DevSecOps practices.

As future work, further steps in applying DevSecOps practices to NG-IoT environments in general, and to ASSIST-IoT in particular, will include the coordination and definition of the continuous delivery and automated deployments, and later implementing, monitoring, and observability methods and tools over the NG-IoT deployments. Finally, the validation of the methodology and strategy in verticals related to the automotive industry, transportation and logistics and safety at work is planned.

ACKNOWLEDGMENT

This work has been partially funded by H2020 project ASSIST-IoT with EC contract number 957258.

REFERENCES

[1] "ASSIST-IoT EU H2020 project," 2021, *Accessed on 03.08.2021*. [Online]. Available: https://assist-iot.eu

[2] M. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 2013, pp. 9–16.

[3] Alliance of Internet of Things Innovation (AIOTI). (2020) Strategic Foresight through Digital Leadership. *Accessed on 03.08.2021*. [Online]. Available: https://aioti.eu/wp-content/uploads/2020/10/IoT-and-Edge-Computing-Published.pdf

[4] Amazon Web Services. (2019) What is DevOps. *Accessed on 03.08.2021*. [Online]. Available: https://aws.amazon.com/devops/what-is-devops/?nc1=h_ls

[5] AppDynamics. (2015) Keep CALM and Embrace DevOps. *Accessed on 03.08.2021*. [Online]. Available: https://kapost-files-prod.s3.amazonaws.com/published/555271a4c12539dc18000118/ebook-keep-calm-and-embrace-devops.pdf

[6] S. Chacon and B. Straub, *Pro Git*. Apress, August 2021.

[7] CircleCI. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://circleci.com/

[8] Enisa. (2019) Good practices for IoT and Smart Infrastructures Tool. *Accessed on 03.08.2021*. [Online]. Available: https://www.enisa.europa.eu/topics/iot-and-smart-infrastructures/iot/good-practices-for-iot-and-smart-infrastructures-tool/results#IoT

[9] ——. (2019) How to Implement Security by Design for IoT. *Accessed on 03.08.2021*. [Online]. Available: https://www.enisa.europa.eu/news/enisa-news/how-to-implement-security-by-design-for-iot

[10] European Commission. (2017) Digitising Eureopean Industry. *Accessed on 03.08.2021*. [Online]. Available: https://ec.europa.eu/futurium/en/system/files/ged/15_11_2017_digitising_european_industry_brochure_ec_final_web3.pdf

[11] GitHub. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://docs.github.com/en/github

[12] Gitlab. (2021) Gitlab DevSecOps 2021 Global Survey results - A Maturing DevSecOps Landscape. *Accessed on 03.08.2021*. [Online]. Available: https://learn.gitlab.com/c/2021-devsecops-report?x=u5RjB_

[13] IoT Security Foundation. (2020) Iot security compliance framework release 2.1. *Accessed on 03.08.2021*. [Online]. Available: https://www.iotsecurityfoundation.org/wp-content/uploads/2020/05/IoTSF-IoT-Security-Compliance-Framework-Questionnaire-Release-2.1.zip

[14] J. Bird. (2016) DevOpsSec: Securing software through continuous delivery. *Accessed on 03.08.2021*. [Online]. Available: https://www.oreilly.com/content/devopssec-securing-software-through-continuous-delivery/

[15] J. Erickson and K. Lyytinen and K. Siau, "Agile modeling, agile software development, and extreme programming: The state of research," *J. Database Manag.*, vol. 16, pp. 88–99, 2005.

[16] J. Pennington. (2019) The Eight Phases of a DevOps Pipeline. *Accessed on 03.08.2021*. [Online]. Available: https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba

[17] Jenkins. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://www.jenkins.io/

[18] K. Beck and others. (2001) Manifesto for Agile Software Development. *Accessed on 03.08.2021*. [Online]. Available: https://agilemanifesto.org/

[19] Kubernetes. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[20] L. Banica, and P. Polychronidou, and M. Radulescu, and C. Stefan, , "When IoT Meets DevOps: Fostering Business Opportunities," *KnE Social Sciences*, vol. 3, no. 10, p. 250–264, 2018.

[21] L. Leite, and C. Rocha, and F. Kon, and D. Milojicic, and P. Meirelles,, "A survey of devops concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, 2019.

[22] M.A. Lopez-Pena, and J. Díaz, and J.E. Pérez, and H. Humanes, , "DevOps for IoT Systems: Fast and continuous monitoring feedback of system availability," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 695–10 707, 2020.

[23] MITRE ATT CK. (2020) *Accessed on 03.08.2021*. [Online]. Available: https://attack.mitre.org/

[24] N. Kirovska, and S. Koceski,, "Usage of Kanban methodology at software development teams," *Journal of Applied Economics and Business*, vol. 3, no. 3, pp. 25–34, 2015.

[25] OWASP. (2020) OWASP top 10 web application security risks. *Accessed on 03.08.2021*. [Online]. Available: https://owasp.org/www-project-top-ten/

[26] OWASP. (2021) OWASP threat model cookbook. *Accessed on 03.08.2021*. [Online]. Available: https://owasp.org/www-project-threat-model-cookbook/

[27] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *Product-Focused Software Process Improvement*, F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 386–400.

[28] M. Poppendieck, "Lean software development," in *29th International Conference on Software Engineering (ICSE'07 Companion)*, 2007, pp. 165–166.

[29] PYTM. (2021) A Pythonic framework for threat modeling. *Accessed on 03.08.2021*. [Online]. Available: https://github.com/izar/pytm

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

[31] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th International Conference on Software Engineering*. IEEE Computer Society Press, 1987, p. 328–338.

[32] S. M. Mohammad, "DevOps automation and agile methodology," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 5, no. 3, pp. 946–949, 2017.

[33] SCAN. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://github.com/ShiftLeftSecurity/sast-scan

[34] V. Mohan, and L.B. Othmane, , "SecDevOps: Is it a marketing Buzzword? - mapping research on security in DevOps," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 2016, pp. 542–547.

[35] Visual Studio Code. (2021) *Accessed on 03.08.2021*. [Online]. Available: https://code.visualstudio.com/

[36] L. Williams and A. Cockburn, "Agile software development: it's about feedback and change," *Computer*, vol. 36, no. 6, pp. 39–43, 2003.

[37] Z. Zhou *et al.*, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

Received on July 01, 2021
Accepted on August 19, 2021