# Security principles for DevOps and cloud

## Delivering value frequently while balancing cyber risk

Martijn Sprengers MSc
is Senior Manager at KPMG Netherlands.
sprengers.martijn@kpmg.nl

Wouter van der Houven MSc
is Consultant at KPMG Netherlands.
vanderhouven.wouter@kpmg.nl

**Many organizations across different sectors are increasing their digitization efforts, ultimately to deliver their products and solutions faster to the market. The technology, quality and, especially, the security functions struggle to keep up with the pace. While traditional organizations have difficulties moving their heterogeneous IT landscape to the cloud, less traditional technology-oriented companies struggle to embed security as a process in their development lifecycle, as they perceive it to impair their valuable time to market. Rather than implementing security as a stage gate at the end of the development and operations lifecycles, it should be a continuous process through the value delivery stream. This should ultimately help position the security function as a business enabler.**

## INTRODUCTION

Agile Methodology, Continuous Integration, Continuous Delivery, SecDevOps, DevSecOps. All terms that describe a change in mindset towards bringing (software) ideas to customers faster. Lately, organizations are adopting ways to increase the speed of development even further by composing teams with both development and operation engineers (DevOps teams) ([Brum19]). This rapid deployment of changes to production environments has security practices struggling to keep up with the pace. This, combined with the cloud transformation of the past decade, has led to rethinking security principles and dealing with risks ([Stur12], [Beek15]). However, these are mostly focused on the risks that are associated with bringing your IT and data to the cloud, rather than maintaining a sufficiently secure state of solution delivery in these rapidly changing environments.

This article addresses some of the key challenges that organizations face when applying this shift in mindset towards DevOps, and how to deal with those in a cloud-enabled world. Rather than focusing on how to directly control the technical risks, we will address how to apply security principles in the development process to deliver value faster while meeting security, privacy, and compliance needs.

In order to arrive at these security principles, we will first address the DevOps principles and how, in combination with cloud technology, these can be used to achieve an unparalleled time to market. We will then address how a secure state can be achieved and maintained with a combination of DevOps principles and elements of the application development lifecycle.

**Table 1.** Cloud principles that can help improve security in a DevOps environment.

| | |
|---|---|
| Serverless computing | The cloud provider manages underlying infrastructure such that users can focus on writing and deploying code. |
| Infrastructure as Code | The cloud provider manages underlying infrastructure based on (templated) code written and deployed by DevOps engineers. |
| Security Centralization | Leveraging data-output of the cloud provider's security capabilities to get a holistic view on the security posture. |

## ORGANIZING DEVOPS

DevOps is all about organizing three basic principles which we will briefly outline below ([Hütt12], [Kim14], [Kim16]):

1. The principle of flow. This principle emphasizes the performance of the system, instead of the performance of a specific subprocess, department or individual contributor (e.g., a developer, system administrator). It should focus on all value streams that IT delivers, from requirement identification, development, testing, transition to operations and delivered to end-users and customers.
2. The principle of feedback. This principle is about getting feedback as soon as possible in the software development lifecycle. Developers and practitioners refer to this as 'shifting left': the earlier you detect an error or issue, the more inexpensive it is to fix the issue. The goal is to shorten and increase feedback loops so necessary corrections can be continually made.
3. The principle of learning. This principle focuses on creating a culture for continual experimentation, taking risks and learning from failure. 'Fail fast' and 'fail often' are key terms to this principle, yet very hard to get right in practice. This also includes making the team responsible for their successes and failures and providing enough means to grow.

Many organizations have adopted this way of working. However, the principles themselves do typically not provide practical recommendations on how to organize secure development processes. Research has been conducted on applying these principles in practice, for example through implementing 'Continuous Integration' ([Fowl16], [Duva17]) and later 'Continuous Delivery' ([Humb10]). Also, organizations have embraced agile development processes, such as 'SCRUM' ([Schw02]), following different maturity levels ([Lepp13]). Although these principles provide some guidelines, we still see that many organizations struggle to embed security in the development process, to become so-called 'Secure by Design'.

## CLOUD SECURITY OPPORTUNITIES

Utilizing cloud technologies and shifting towards a DevOps organization go hand in hand. New cloud developments like serverless computing and Infrastructure as Code have impacted organization's security landscape in the same way DevOps organization has by blurring lines between the development and operations of solutions. As a result, organizations now have a myriad of opportunities to use new (security) capabilities and technologies. The following (non-exhaustive) list provide an overview of principles that could help improve cyber security (see Table 1).

## Serverless computing

With the introduction of the cloud, the respective cloud providers are responsible for the (security of the) services that they offer which reduces the total 'security surface area' that the organization's security experts need to manage themselves. The usage of 'Infrastructure as a Service' (IaaS) and 'Platform as a Services' (PaaS) patterns allows organizations to better focus on their key strengths, rather than managing the complexities of hardware and software. Serverless computing concerns this transfer of responsibility for part of the security surface in solution development, such that users can focus on writing and deploying code. This helps reduce the risks associated with managing infrastructure components, such as data centers, virtual machines, databases and configuration of (network) components. For example, KPMG Digital Risk Platform's architecture is entirely constructed with serverless components, drastically reducing the overhead of patching and configuration management.

## Infrastructure as Code

This is the process of managing and provisioning software and hardware configuration through machine-readable definitions, rather than error-prone manual and interactive provisioning through configuration tools ([Arta17]). Infrastructure as Code can be used for platform as well as infrastructure components. All major cloud providers support this process. Another advantage is that the (changes to) definitions can be treated as code. This allows for managing changes to the infrastructure in the same way, with the same tools as managing changes to regular (application) code, using known software best practices for designing, implementing, and deploying applications infrastructure. Deployments are therefore less error-prone, the environment is more homogeneous and security configurations can be managed as part of the regular secure development lifecycle.

## Security centralization

Cloud environments allow for security capabilities such as encryption, key management, privileged identity management, auditability and security monitoring to be centralized. Although organizations tend to view this as an increased risk (i.e. "one place to rule them all", [Moll19]), it provides more visibility, opportunities for automation and simplicity. Leveraging the economy of scale, every security requirement brought forward by another cloud customer can improve the security of the overall cloud provider, as these provides have a major incentive to keep the cloud secure. Centralization in a cloud also allows to connect platforms like Azure DevOps, Gitlab and Atlassian to easily track work, collaborate on code and integrate continuous deployment. This greatly improves transparency and allows development and engineering teams to focus on the DevOps principles.

**Table 2.** Typical security challenges while implementing DevOps.

| | |
|---|---|
| View 'security' as a single stage gate | We see organizations trying to put all risk mitigating activities at the end of the 'development process' majorly impacts the 'DevOps' flow. |
| Get overwhelmed with the output and feedback of security solutions | As security solutions typically report many non-compliances, potential risks and incidents, activities such as triage, false positive reduction and follow-up require time and effort. |
| Failure to keep up with the speed of business | New tools and methodologies allow teams to rapidly develop, modify, and deploy new solutions towards production environments, while security practices are struggling to keep up with the pace. |
| Use 'time to market' as an excuse | Security challenges in keeping up with the speed of business are often addressed with the excuse that 'time to market' is the primary priority, resulting in security measures being postponed. |

## CLOUD AND DEVOPS SECURITY CHALLENGES

On the other hand, we also see organizations struggle with the overwhelming possibilities that cloud environments and new development methodologies offer in relation to security. Table 2 lists some examples.

### Security as a stage gate

We see that organizations are trying to put all risk-mitigating activities at the end of the 'development process', leading to feedback that is obtained only at the end of the software development lifecycle. This decreases the flow as it will take longer to follow up on the identified bugs and issues than when they would be detected earlier in the development process. In traditional companies with many legacy systems this results in delays in IT projects, as the security function of the organization is overwhelmed with activities to complete the security 'stage gate' at the end of the project.

### Get overwhelmed with the output and feedback of security solutions

Although cloud environments provide new tools and methodologies, we see organizations struggle to use them adequately. All three major cloud providers (Amazon, Google, Microsoft) provide many security solutions, ranging from DDOS protection to threat and vulnerability monitoring. However, more monitoring capabilities do not necessarily improve security. As these solutions typically report many non-compliances, potential risks and incidents, activities such as triage, false positive reduction and follow-up require time and effort. Determining what is actually important requires a sound threat model. With many potential risks reported by these solutions, companies fail to determine the actual business risk of potential security issues and vulnerabilities, thereby typically focusing on the wrong corrective actions and behaviors. Some examples are:

- Failure of the solutions to understand business context, reporting vulnerabilities in development environments that are segmented from the production environment as 'critical risks';
- Failure to understand usage patterns and behaviors, such as reporting the shared use of test accounts as impersonation attacks;
- Failure to understand the application or environment context, such as reporting licenses that are used in test tools (which are not distributed to end-users) as 'policy violations'.

We have seen clients that struggle with this output volume, particularly when they have many cloud security solutions that run frequently. Typically, these solutions report thousands of potential (high risk) security issues, while only few of them really affect the business continuity.

### Failure to keep up with the speed of business

We see IT functions struggle with the opportunities cloud environments provide. Due to the lack of business understanding, business and IT goals are diverging. Where IT functions try to keep the application portfolio to a minimum, with increased control, business users often procure and use their own IT. They praise the flexibility, frequency of functionality updates and possibilities of (cloud) Internet services. A credit card is usually enough to buy an application or server. An example is dealing with so-called 'shadow IT' ([Kuli16]): applications that are used by business users that are not or very little under the control of IT functions. Examples are marketing outings via servers not controlled by IT, sending sensitive to cloud storage providers and connecting business identities with third party applications.

### Use 'time to market' as an excuse

We have seen organizations that are quite capable of implementing the principle of flow but lack the appropriate checks and balances for security. Typically, they perceive time-consuming and compliance-driven security controls as an impediment to their time to market and execution speed. Frequently delivering new versions to end users is considered valuable. However, it must be done in accordance with a clear risk appetite and a sound associated threat model. We acknowledge that trying to cover all potential security risks is time-consuming, and often also undesirable as it always comes with a tradeoff (e.g. with usability). For example, patching a specific API endpoint that allows for SQL injection can take quite some developer resources. If this endpoint is only reachable by administrators and only after a two-factor authenticated login, it greatly reduces the attack surface and probability of a successful exploitation by an unauthenticated user. If this is in line with the company's risk appetite, it can be decided not to patch and continue the deployment. Three important elements play a role here:

- Major stakeholders, such as the board of management, should have set a sound risk appetite.
- DevOps teams should be aware of this risk appetite and how they can apply its boundaries in practice.
- DevOps teams and stakeholders should be aware of the (potential) risks that are present and the risks they would like to take. Organizing risk management is beyond the scope of this article, we refer to other resources that are available (such as [Baut19]).

## ORGANIZING SECURITY CAPABILITIES IN THE DEVELOPMENT LIFECYCLE

Organizations can apply security principles in the development and operation processes to deliver value faster, while also leveraging the benefits of cloud transformation. As cloud infrastructure is commonly continuously developed, a shift from manual processes to automated controls is required in order to maintain a consistent security posture while still maintaining frequent value delivery.

Furthermore, the infrastructure itself also changes. Application development produces not only an application, but also the underlying infrastructure and configuration thereof. This includes e.g. virtual machines, firewalls, databases, etc. Developing new infrastructure introduces the requirement to enroll in other security capabilities like monitoring, networking (VNet, WAN, VPN, DNS) and delivery (CDN, Load balancers, Application Gateways).

We will, based on the phases in the DevOps process, discuss how to embed security principles and capabilities. Figure 1 fairly represents common steps in secure development lifecycle process.

### Plan

As many of the security issues originate from human failure, it is important to enable DevOps engineers with the right knowledge and tools to make risk decisions as early in the software development lifecycle as possible, i.e. during the 'plan' phase. An important aspect is to make the team itself responsible for security, not just the IT security function or team in the organization. This requires that developers are allowed to take security training. Also, the (traditional) security function should provide modern product management and engineering disciplines (such as Product Owners) with advice and recommendations during the planning of functionality.

In addition to the functional requirements set for an application iteration, it is also necessary to define non-functional requirements, such as:

- performance and availability requirements;
- code quality and license requirements;
- data confidentiality and integrity requirements; and
- personal identifiable information requirements.

All of these require the stakeholders and business owners to agree on the risk appetite of the software, based on threat modeling, usage of the software, reputation, and the type of data being stored. This is all about reviewing risk scenarios. For example, fixing a SQL injection vulnerability in a part of the application only accessible to functional administrators could have less priority than replacing code that introduces a license infringement, as legal and reputational damages can directly impact the business.

## Code

A very important step during the 'Plan' and 'Code' phases is the validation of requirements by the developers. In agile development methodologies this is called the 'refinement': meetings in which developers demonstrate their understanding of the requirements (e.g. features, product backlog items) to be implemented to the product owner and/or business analysts. This is an ideal place to discuss potential security and privacy aspects/impact of the (non-)functional requirements, as new functionality also always introduces additional attack surface. Discussion thereof fosters ownership of issues and makes all parties involved work towards solutions that are acceptable to all. This should help developers gain a better understanding of the context and 'shift left' security activities: during the implementation of code they need to be informed about potential issues (such as security bugs) as early as possible.

## Build and Test

In order to uphold the (non-)functional requirements set out in the previous stages of application development, testing and failing to pass tests must happen as early as possible in the development process. To work towards this goal, various types of tests can be executed in an automated fashion, both during and after the 'build' phase.

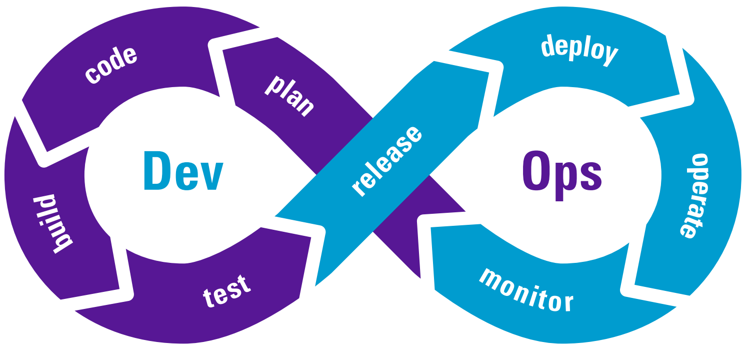
- Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) ([Gold19]) identify and address issues in proprietary software, such as bugs, potential security flaws, code coverage, code quality and technological debt.
- Software Composition Analysis (SCA) covers the identification of open source components with known vulnerabilities and potential license infringements in imported libraries.
- Implementing Security Monitoring in critical flows of the application, as identified in the requirements and design phases to improve incident response, and forensic capabilities as well as auditability of the application.

Also, developers should be encouraged to participate in offensive security activities against their own environment and developed code. The understanding of circumstances that introduce security vulnerabilities can help developers anticipate and solve security issues beforehand. Usage of security tools, such as OWASP ZAP, Burp, Nikto and Metasploit by developers is encouraged in order to facilitate the automation of security testing in the development process. But remember: "a fool with a tool is still a fool".

In the 'test' phase, the development team validates if the software build matches the requirements. A so-called 'Pull request' is one of the most important security measures in this phase, as it brings together all elements of performing risk assessment: product backlog items or user stories with the requirements, trackable/auditable work through work items, code and commit messages, results of the tests and Static Application Security Testing (SAST). We have depicted an example flow of the Pull Request in the picture.

A crucial element of the pull request is a peer review by another developer. However, to be properly executed, the IT staff involved in the pull request should have suffi-

**Figure 1.** The DevOps lifecycle ([Otey18]).



Use your cloud transformation to balance security activities and budget with time to market

cient security knowledge to make the decision. Only if all criteria are met, will the newly developed code be propagated towards the main branch (i.e. 'master' in Figure 2).

## Release

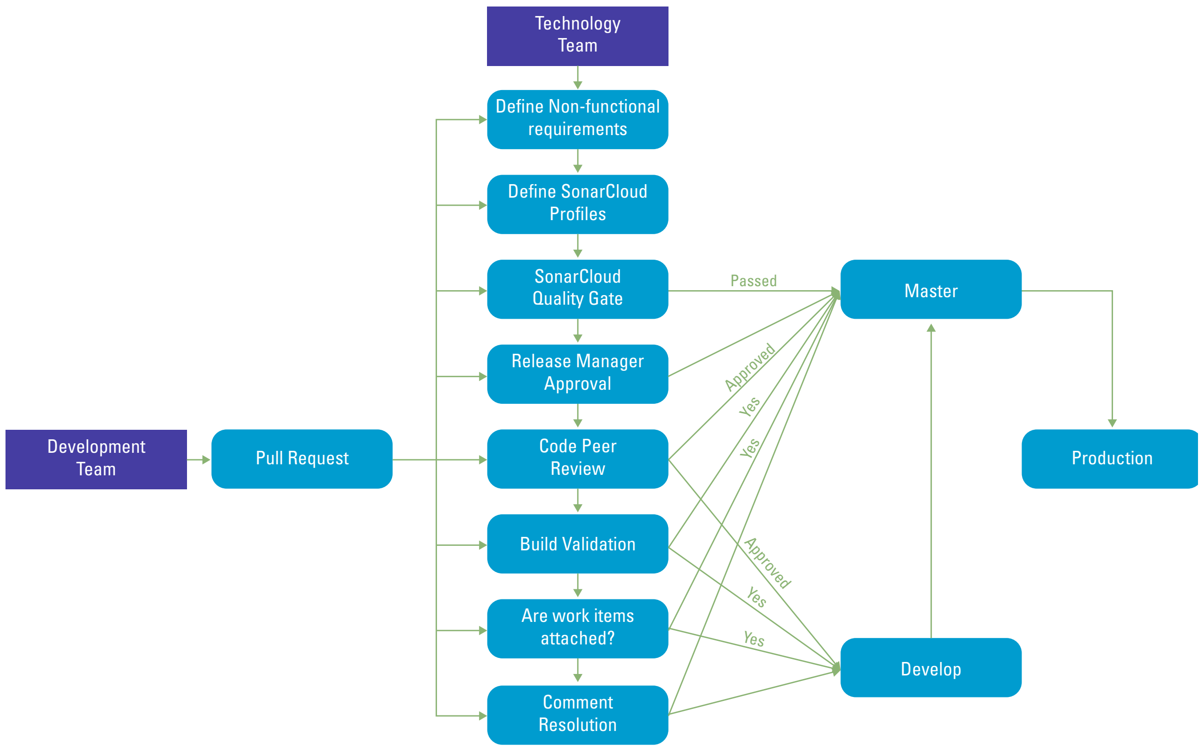
The final step before pushing changes to production environments is the release phase. This includes the final security review, in which the risks associated with the deployment of changes are assessed by the business, given the defined (non-)functional requirements and test results. It is important that the business stakeholders can make an informed security decision. The attack surface should match the threat model and risk appetite set forth during the 'requirements' phase. Any residual risks of operation should be addressed through the definition of an incident response plan.

After approval, the release and release approval are archived to help create an auditable trail from the definition of requirements to production release.

## Deploy

When a release is approved by the relevant stakeholders, the changes should be pushed to the production environment. This is called the 'deploy' phase. A great way to minimize the amount of errors, is to maximize automation. Modern cloud environments support the configuration of automated release pipelines that build upon the principle of Infrastructure as Code. This ensures that the infrastructure required to run the application is launched and configured through predefined scripts, which in itself can be treated like any other application change. An important security aspect during the 'deploy' phase is to ensure environments (such as development, test and production environments) are separated through the usage of 'key vaults'. These vaults store the secrets (such as password and keys) of the application and under-

**Figure 2.** Example approval process for deploying code to production, used in KPMG's Digital Risk Platform.



lying infrastructure and can be automatically populated and used in a deployment pipeline.

### Operate

The operate phase involves maintaining and troubleshooting applications in production environments. The DevOps teams maintain roles such as 'designated responsible individuals' and 'site reliability engineers' to ensure system reliability, high availability and performance while reinforcing security. They try to identify issues before these affect the end-user experience and respond to issues quickly when these occur.

### Monitor

Given the defined (non-)functional requirements, operational and monitoring use cases can be defined. By implementing monitoring, a production-first DevOps mindset is fostered and impact on end users can be limited by taking proactive actions. Impact can be evaluated through observation, testing, analysis of telemetry, and user feedback. This then feeds the 'plan' phase of the next iteration of product development in the DevOps process.

## CONCLUSION

Organizations need to balance security activities and budget with time to market and user friendliness. Cloud transformations can help with embedding security principles and solutions, especially if these are implemented through the DevOps principles. Moving to the cloud is also a good opportunity to embed cyber security in daily processes such that organizations get more 'secure by design'. We have discussed common pitfalls in implementing cloud security solutions and have provided security principles and activities that can be embedded in (agile) development processes. The key take-away is to make the DevOps engineers feel responsible for the security decisions they take during development and provide them with the means and mandate to do so. This should help organizations to better balance security and usability, while still maintaining the ever increasing need to deliver value faster.

## References

**[Arta17]** Artac, M. et al. (2017). DevOps: Introducing Infrastructure-as-Code. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 497-98.

**[Baut18]** Bautista, M.C. & B. Krutzen (2018). Digitization of Risk Management. *Compact* 2018/2. Retrieved from: https://www.compact.nl/en/articles/digitization-of-risk-management/

**[Beek15]** Beek, J.J. van (2015). Assurance in the Digital World of the Future. *Compact* 2015/Special. Retrieved from: https://www.compact.nl/en/articles/assurance-in-the-digital-world-of-the-future/

**[Brum19]** Brummelen, J. van & T. Slenders (2019). Modern Software Development. *Compact* 2019/2. Retrieved from: https://www.compact.nl/articles/modern-software-development/

**[Duva07]** Duvall, P.M. et al. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. London: Pearson Education.

**[Fowl06]** Fowler, M. et al. (2006). Continuous integration. *Thought-Work*, 122, 14.

**[Gold19]** Goldstein, A. (2019, May 23). SAST vs. SCA: It's Like Comparing Apples to Oranges. *Whitesource Software*. Retrieved from: https://resources.whitesourcesoftware.com/home/sast-vs-sca

**[Howa06]** Howard, M. et al. (2006). *The Security Development Lifecycle*. Redmond: Microsoft Press Redmond.

**[Humb10]** Humble, J. et al. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. London:* Pearson Education.

**[Hütt12]** Hütterman, M. (2012). *DevOps for Developers*. New York: Apress.

**[Kim14]** Kim, G. (2014). *The Three Ways: The Principles Underpinning DevOps*. Portland: IT Revolution Press.

**[Kim16]** Kim, G. et al. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland: IT Revolution.

**[Kuli16]** Kulikova, O. (2016). Cloud Access Security Monitoring: To Broker or Not To Broker? *Compact*, 2016/3. Retrieved from: https://www.compact.nl/articles/cloud-access-security-monitoring-to-broker-or-not-to-broker/

**[Lepp13]** Leppanen, M. (2013). A Comparative Analysis of Agile Maturity Models. *Information Systems Development*, 329-343.

**[Moll19]** Mollema, D. (2019). Syncing yourself to Global Administrator in Azure Active Directory. *Fox-IT*. Retrieved from: https://blog.fox-it.com/2019/06/06/syncing-yourself-to-global-administrator-in-azure-active-directory/

**[Otey18]** Oteyowo, T. (2018, April 12). DevOps in a Scaling Environment. *Medium*. Retrieved from: https://medium.com/tech-tajawal/devops-in-a-scaling-environment-9d5416ecb928

**[Schw02]** Schwaber, K. et al. (2002). *Agile Software Development with Scrum*. Upper Saddle River: Prentice Hall.

**[Stur12]** Sturrus, E., Steevens, J. & Guensberg, W. (2012). Access to the cloud. *Compact* 2012/0. Retrieved from: https://www.compact.nl/en/articles/access-to-the-cloud/

**[Tunc17]** Tunc, C. et al. (2017). Cloud Security Automation Framework. *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 307-312.

## About the authors

**Martijn Sprengers MSc** is head of Technology at KPMG's Digital Risk Platform, a SaaS solution to digitize and automate risk functions in organizations. Martijn was an IT security advisor at KPMG Netherlands before. He has over ten years of experience with technical facets of cyber security.

**Wouter van der Houven MSc** is an IT security advisor at KPMG Netherlands. He has over two years of cloud security experience. He developed and automated the security controls for KPMG's Digital Risk Platform.