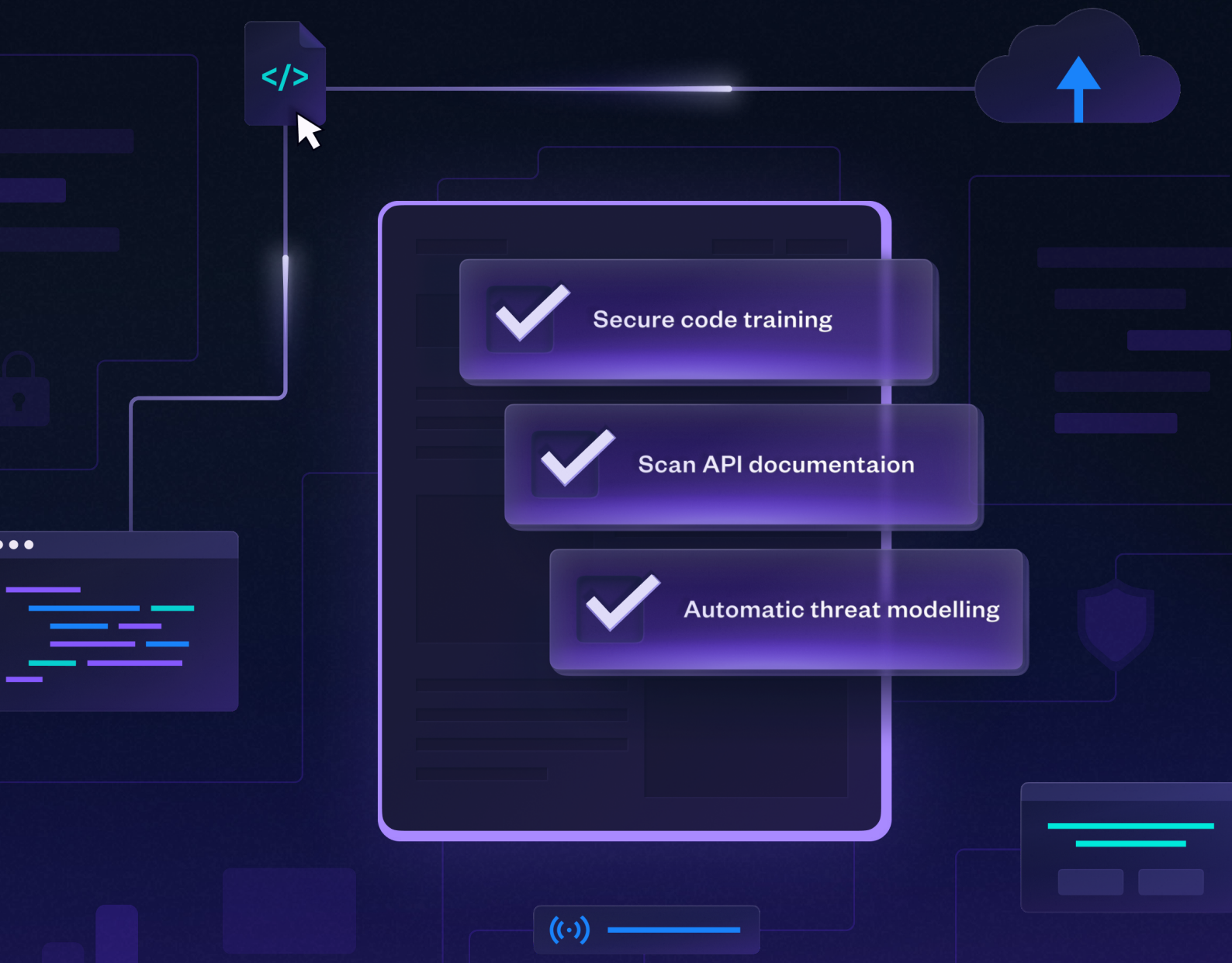
Checkmarx

CHECKLIST

The Ultimate

CODE TO CLOUD SECURITY CHECKLIST



What is

Code to Cloud

Code to cloud is a framework that centers around requirements for cloud-native application development and builds application security into every step of the software development lifecycle (SDLC) – from the first line of code to deployment and runtime in the cloud.

So, what does that mean for the teams involved?

In a code to cloud AppSec program, developers and AppSec managers must be aligned, since they must work together to identify and remediate vulnerabilities found at in every stage of development.

In theory this close alignment, visibility into the SDLC, and ability to remediate vulnerabilities before deployment can help enterprises protect their business-critical applications – however, this approach has its unique challenges.

45%

of data breaches happen in the cloud

Source: IBM

73%

of businesses have cloud-based applications

76%

of organizations say the number of point tools they use cause blind spots

Evolution of

Cloud-Native Applications Impact and Significance

Cloud-native applications have changed the way software is developed, deployed, and managed. These applications differ significantly from traditional monolithic architectures in several ways:

1 Microservices

Microservice-based architectures, also known as composable applications, make it easier to scale applications across cloud environments. Organizations can deploy and scale individual components of an application as needed, rather than the entire application. In addition, a microservices-based architecture allows for multiple development teams to develop and release different parts of the application in parallel.

2 Application Programming Interfaces (APIs):

Microservice-based architectures have driven an increase in the use of APIs. APIs are used to communicate and improve interoperability between microservices and third parties. They also enable multiple development teams to work on different components of the application, knowing how those microservices will communicate with each other once deployed.

3 Containers

The need to rapidly scale applications across multiple cloud environments requires increased standardization in how applications are packaged. Container platforms, like Docker, provide an abstraction layer that enables developers to focus on the code specific to their application. Containers package an application and all its dependencies in a way that it can be consistently deployed across multiple environments, including different cloud providers.

4 Infrastructure as Code (IaC)

Deploying an application can require configuring many different infrastructure components. With the cloud enabling applications to be rapidly scaled up – and scaled down – these configuration tasks are potentially performed on a repeated basis. IaC automates infrastructure deployment using code, enhancing consistency, and reducing manual errors. For example, AWS CloudFormation enables defining and provisioning AWS infrastructure in a reproducible and efficient manner.

5 DevOps

Cloud-native application development has driven adoption of DevOps methodologies. With 100s or 1000s of microservices being developed in parallel, organizations need to operationalize how applications are developed, built, delivered, and deployed consistently at high scale and velocity. While DevOps means much more than just required technical capabilities, it has driven increased automation of different development steps and tasks across the SDLC.

73% **of organizations** have more than 50% of their apps deployed in the cloud.

The Challenge

Securing Cloud-Native Applications

The impact that cloud-native applications have can be immediate – facilitating speedier innovation cycles, deployments, and communication between development and AppSec teams. However, the transition to cloud-native applications has some challenges.

The Microservices Challenge

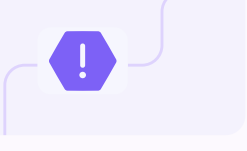
The agile development model enabled by microservices can challenge AppSec teams. A single application can now be comprised of dozens, or even hundreds, of microservices – each an independent software project worked on by different developers on a different release schedule, and that must be independently scanned for vulnerabilities.

Proliferation of APIs

The proliferation of APIs has created new security challenges, including data exposure caused by shadow and zombie APIs. Shadow APIs are undocumented and unknown to security teams, while zombie APIs are abandoned, while remaining in production. Both pose security and compliance risks.

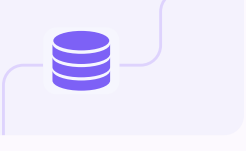
Attack Surface Expansion

Containerized environments can expand the attack surface and require individual component protection and micro-segmentation, particularly in the Kubernetes control plane. The short-lived nature of containers makes it difficult to investigate security issues. And their reliance on open source components increases security risks due to vulnerabilities that are potentially exploitable by attackers.



IaC Risks

IaC templates allow for potential misconfigurations, which can result in exposed sensitive data or compromised access controls. Storing IaC templates in insecure repositories may invite malicious manipulation, resulting in security breaches. Additionally, challenges like configuration drift, unintended changes in templates over time, and "ghost" resources, untagged assets with potential cost and maintenance can result in even more security implications if not monitored and addressed.



The Need For

A Secure Cloud-Native Application Lifecycle

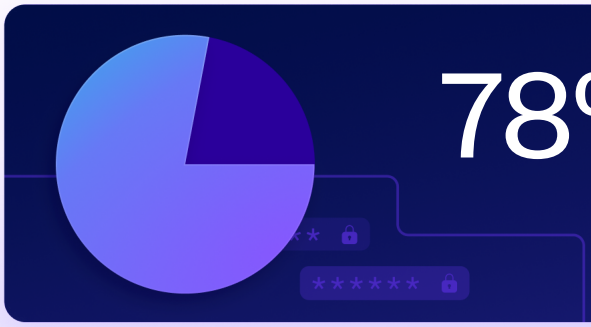
Securing cloud-native applications presents a unique set of challenges. As the pace of development continues to increase, it is becoming harder for AppSec teams to keep up with development. AppSec teams often find themselves with multiple security tools to use at different stages of the SLDC. This introduces another layer of complexities, since these tools all must be managed separately, and many don't integrate with each other. Without that unified view and overwhelmed by the number of potential vulnerabilities, AppSec managers often try to slow down the development process to remediate issues – or choose to release applications with known vulnerabilities.

One of the most impactful ways organizations build trust between AppSec and development teams, while also managing tool sprawl, is consolidating their tools into a unified platform – like Checkmarx One. Having teams work with a unified platform not only gives them visibility into the entire SDLC, but it also can help teams become more efficient with vulnerability identification, prioritization, and remediation. When enterprises invest in building trust across their teams, they often can develop a more secure and effective SDLC that aligns with the dynamic nature of cloud-native applications.

90%

of organizations have **deployed known-vulnerable code in production** to meet business or feature deadlines

Checkmarx, 2024 The Future of AppSec Report



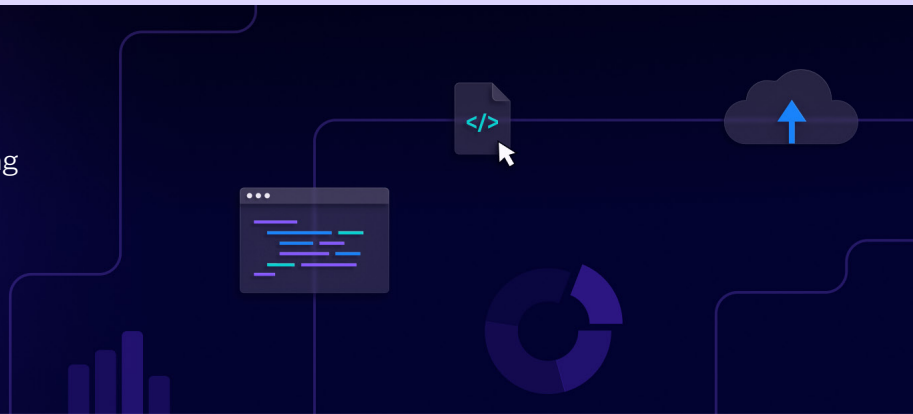
78%

of organizations experienced at least one breach in the last year **as a result of a vulnerable application they developed**

Checkmarx, 2024 The Future of AppSec Report

Discover a comprehensive checklist that addresses the unique challenges of securing cloud-native applications.

Keep reading



Application Security in the

Cloud-Native Application Lifecycle

AppSec teams have known for years that different vulnerabilities can be introduced or become apparent at different stages of the SDLC (such as with SAST and DAST). More recently, application security has embraced a "shift everywhere" approach to securing application development.

When thinking about a code to cloud AppSec strategy, it's important to fully understand the complexity of the SDLC. Securing applications effectively requires a holistic AppSec strategy that focuses on building security from the first line of code through deployment and runtime in the cloud.

Design

A code to cloud approach to application security starts with proper design. The design phase enables multiple development teams to work on all the different microservices in parallel, by defining how the components, sub-components, and architectural elements come together. From a security perspective:

API security: Proper API design allows developers to scan documentation files to identify potential security risks and misconfigurations that will appear once in production, before writing a single line of code. Documentation can be then shared with security teams to apply the proper runtime security controls once the application is live in production environments. Maintaining and updating API documentation as code changes over time can help identify inadvertent introductions of security risks into the environment.

Threat modeling: Threat modelling can provide security recommendations and requirements upfront where they be easily integrated once in the coding phase. In addition, using threat modelling can enable AppSec and development teams to reach agreement on how the application will be built – including when and where security controls are applied.

Code

Once the application is designed, developers then translate the blueprint into functional reality by creating the code base through a combination of source code that they write, supplemented with open source and third-party software components.

Secure code training: Most developers have not been sufficiently trained on application security risks or implementing secure coding techniques to mitigate risks like injection attacks, buffer overflows, and other common vulnerabilities. Improving developers' security skills can not only benefit developers but also reduce the number of vulnerabilities that make their way into code.

Static Application Security Testing (SAST): Scanning source code for vulnerabilities is a foundation for any AppSec program. However, many SAST solutions require tuning for each application to maximize accuracy – i.e., finding the most true positives with the least number of false positives. When applying a SAST solution, organizations must consider the requirements of each application. Business-critical applications can benefit from tuning to identify maximum risk. However, the need to cover every application requires low false positives out of the box for less critical applications, even at the expense of lower true positives.

API security: For a cloud-native application, understanding risk often requires seeing vulnerabilities through an API lens. Displaying detected vulnerabilities per API allows AppSec and development teams to prioritize risk by API. Discovering APIs while scanning code can be much more effective at inventorying all the APIs in an application. The global API inventory can then be compared with the API documentation to identify shadow and zombie APIs that were previously missed – and likely not protected - in the production environment.

Secrets detection: In modern development teams, developers within and across different teams must collaborate when developing applications. This often involves sharing of secrets, which can include passwords, API keys, cryptographic keys, and other confidential data that developers need to collaborate, but should not be exposed to unauthorized users. Software supply chain security should start in the coding phase by identifying secrets shared in collaboration tools and preventing accidental leakage.

Build

This phase is where the application starts to take shape. Source code is compiled into executable files and creates the foundation of the application. Developers often build tools and systems to automate this process, to ensure consistency in the generation of deployable artifacts in the later stages of development.

Software Composition Analysis (SCA): Compiling code and building the application starts to pull in open source libraries and other dependencies. Securing these third-party components and mitigating potential vulnerabilities is essential for robust application security. An SCA solution will identify vulnerabilities in open source libraries and present AppSec teams and developers with remediation options.

Malicious package protection: Open source security has evolved beyond just identifying vulnerabilities inadvertently introduced to an open source project. New threats include malicious code intentionally contributed by attackers to be exploited once the application is built. Modern open source security must now identify malicious code to guard against potential threats.

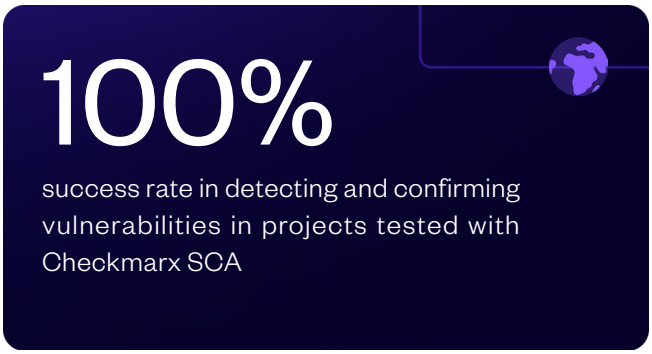
Software Bill of Materials (SBOM): Upon disclosure of a new zero-day vulnerability, organizations need to quickly identify if any application in their environment includes a vulnerable version. This is complicated by cloud-native application architectures, simply by the increase in number of different application components that must be validated. Having SBOMs for every software project, along with proper SBOM storage and operational processes, can greatly simplify response. This is where the application is deployed into the production environment.

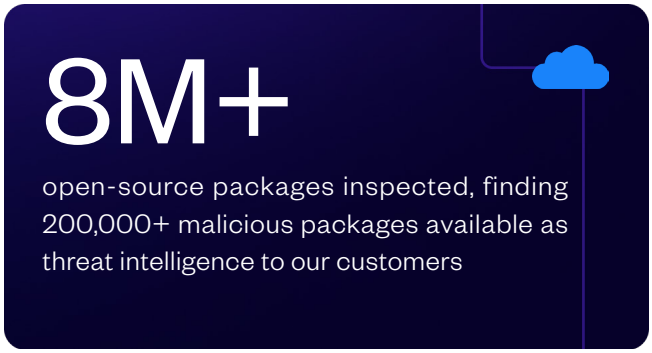
Test

Here, we test the functionality, performance, and security of the applications. Developers utilize various methodologies, such as unit testing, integration testing, system testing, and penetration testing to identify and fix any issues. This is another place where automated tools help – since they can help ensure comprehensive test coverage and expedite feedback to the development team.

Dynamic Application Security Testing (DAST): Testing of the compiled code is crucial to pinpoint vulnerabilities and weaknesses in the application logic and codebase. Neglecting dynamic testing could expose the application to attacks that only become apparent once compiled.

Penetration testing: Penetration testing, whether using DAST or other tools, will help uncover vulnerabilities and weaknesses in the application, such as vulnerable third-party components, data leakage, and session management issues. Being aware of these issues at this stage will allow for proactive remediation before deployment.





Deploy

With cloud-native applications, this stage involves configuring servers, setting up databases, and deploying application components on cloud infrastructure. Continuous integration and deployment (CI/CD) pipelines automate this process, which streamlines deployment workflows and minimizes human errors.

Container security: While container security is a practice that spans multiple SDLC stages, security risks become pronounced in the deploy stage. To reduce risk, static container images should be scanned for vulnerabilities – both in proprietary source code and open source libraries – prior to building and deploying the containerized application.

IaC Security: Improperly configured access controls during deployment may result in unauthorized users gaining access to sensitive data or functionalities. Ensuring the deployment environment is properly configured is important to prevent misconfigurations that could lead to security vulnerabilities. Like scanning static container images, IaC templates should be scanned for security issues and other misconfigurations earlier in the SDLC, so they can be called upon and executed as needed in the deploy phase.

Go-live

The newly developed, or updated, application is now released for full-scale operation. Here the application transitions out of the testing environment into a live production environment. This phase requires meticulous planning and coordination to ensure a seamless transition and often includes data migration, user training, and monitoring system performance to flag any potential issues during the initial rollout.

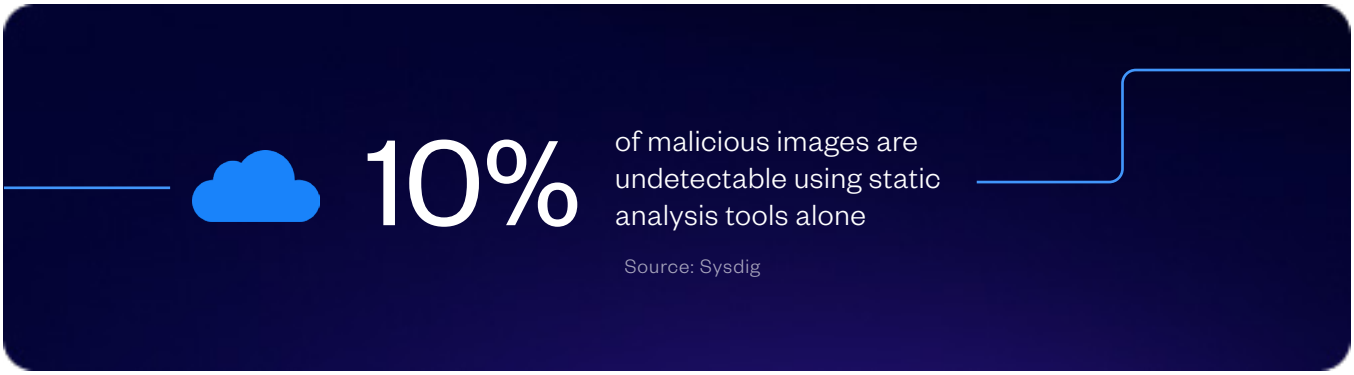
Protecting cloud-native applications requires a different approach than protecting traditional applications – and securing application development. A single application can be made up of different microservices that are running in multiple cloud regions or even different cloud providers. Application workloads are continuously in flux, scaling up and down to meet demand. And ownership of infrastructure security is shared between enterprises and cloud providers. Some of the required capabilities include:

Container security: Prior to the go-live stage, container security focuses primarily on scanning static containers images. But in production environments, it must do much more to protect container workloads running in the cloud. Including scanning running containers for vulnerabilities, posture management, forensics, threat detection, and blocking.

Cloud Workload Protection Platform (CWPP): CWPP solutions focus on the application workload running in a cloud environment. This can include a range of capabilities, from network security (like firewalling and micro-segmentation), to monitoring running the application workloads, to detecting and investigating anomalies in application behavior, and even anti-malware scanning.

Cloud Security Posture Management (CSPM): Cloud-native applications are designed to run on cloud infrastructure with many different infrastructure components that must be properly configured. CSPM monitors the cloud infrastructure and configuration and identifies misconfigured resources that must be addressed.

Web Application and API Protection (WAAP): Cloud WAAP solutions protect applications running in production environments from runtime attacks. It includes traditional capabilities like web application firewall (WAF), distributed denial of service (DDoS) protection, bot management, and API Security.



Developers and Security Teams

Achieving Synergy

A fundamental requirement in cloud-native application security is the ability to integrate with the code to cloud development workflow. Application security must adapt to how organizations are developing cloud-native applications, which includes implications on scale and velocity. Integrating security into the development process can look like:

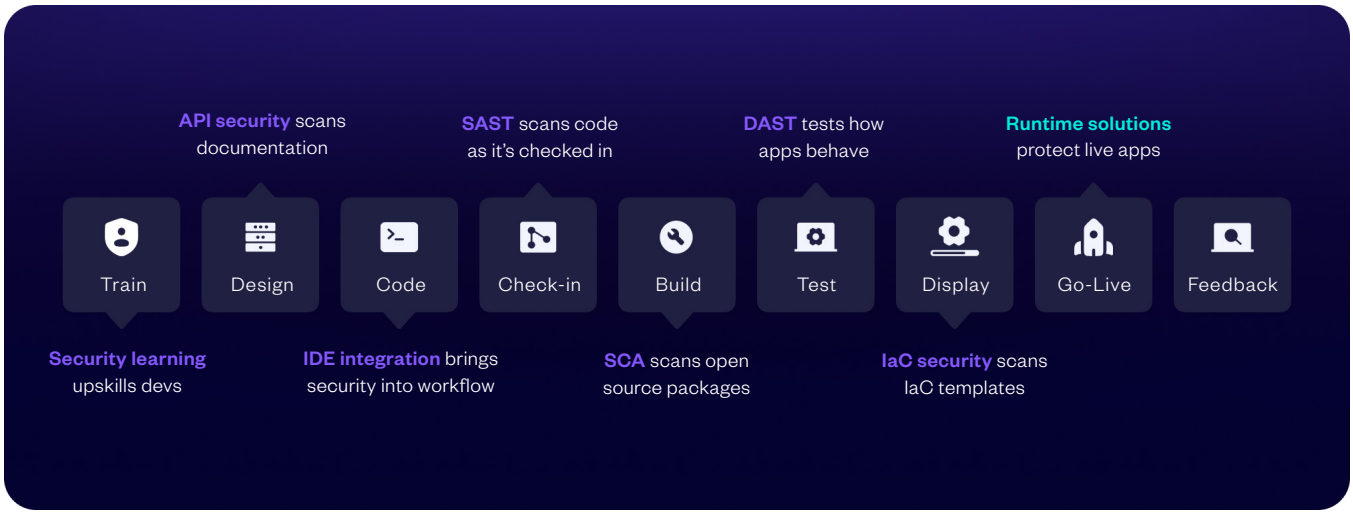
IDE integrations: Rather than forcing developers to learn and use new security-specific tools, meet developers where they are. Bringing security findings and remediation guidance into their Integrated Development Environment (IDE) can make it easier for developers to participate in an application security program.

SCM integration: Integrating directly with Source Code Management (SCM) systems allows for security scans to be automatically triggered as part of the development process, as early as check-in. This allows vulnerabilities to be found while developers are still working on the code, making it easier to address any issues discovered.

Pull request decoration: Integrating with the SCM also enables the ability to decorate the pull request, inserting security findings into the change documentation and facilitating security awareness as part of the development process.

Feedback tool integration: Integrating with bug ticketing systems, like JIRA, and collaboration systems, like Slack, allow for security issues to be communicated via channels already used by developers.

A code to cloud strategy is only effective when developers and AppSec teams trust each other and can work together effectively to create a proactive security culture within an organization. By streamlining security practices into the development workflow, Checkmarx believes that security should be an integral part of the development process, and not an afterthought.



The Checkmarx

Code to Cloud Approach

While many vendors approach cloud-native application security from an infrastructure, network, or workload perspective and then shift left, we believe that it must start from the very first line of code. Our industry leading Checkmarx One platform offers all the capabilities you need to secure every stage of the SDLC, correlate security findings, and then prioritize remediation for developers to make the biggest business impact.

Unified AppSec platform

Checkmarx One was built from the ground up to provide all the capabilities needed to secure applications from code to cloud. Our Fusion engine correlates security data from every tool to help prioritize remediation. Unified analytics and reporting provide a comprehensive view of the application risk across your entire application footprint – including both cloud-native and traditional non-cloud applications.

Comprehensive AppSec capabilities

Checkmarx One provides the full suite of capabilities required to secure development of cloud-native applications, including SAST, SCA, software supply chain security, API security, DAST, container security, IaC security, and secure code training – all on a unified, consolidated application security platform.

Seamless integration across the SDLC

Checkmarx One offers the broadest set of integrations to bring security into the SDLC, including IDEs, SCM tools, CI/CD tools, and feedback tools. This enables security scans to be automatically applied as every application progresses from code, to build, to deploy, and go-live in the cloud. With a consolidated AppSec platform, you can seamlessly integrate your security tools into your SDLC once.

Visibility from code to cloud

Checkmarx One starts with unified analytics and reporting to provide a single view into all your vulnerabilities, but the promise of code to cloud is greater. By correlating security data across every stage in the SDLC – including runtime insights – Checkmarx One can provide true visibility into the vulnerability lifecycle. This helps AppSec teams identify and prioritize the most exploitable vulnerabilities in cloud-native applications, while pointing developers to the exact line of code to fix withactionable remediation guidance.

Final Words

Understanding the phases of an application's lifecycle is necessary when thinking about how you can implement security throughout your SDLC. The code to cloud approach we've outlined relies on proactive AppSec at every step, from the first line of code to ongoing runtime monitoring.

This approach not only addresses vulnerabilities at each stage, but also helps organizations develop a strong culture of security within development and security teams.

See why a code to cloud approach is

critical for business success

Speak With Us ↗

Checkmarx

Checkmarx is the leader in application security and ensures that enterprises worldwide can secure their application development from code to cloud. Our consolidated platform and services address the needs of enterprises by improving security and reducing TCO, while simultaneously building trust between AppSec, developers, and CISOs. At Checkmarx, we believe it's not just about finding risk, but remediating it across the entire application footprint and software supply chain with one seamless process for all relevant stakeholders.

We are honored to serve more than 1,800 customers, which includes 60 percent of all Fortune 100 companies including Siemens, Airbus, SalesForce, Stellantis, Adidas, Wal-Mart and Sanofi.

Printable Checklist

Design

- [ ] Scan API documentation files to identify vulnerabilities and misconfigurations
- [ ] Automatic threat modelling to identify security recommendations and requirements upfront

Code

- [ ] Secure code training to upskill developers on application security practices
- [ ] Static Application Security Testing (SAST) to scan source code for vulnerabilities and can:
  - Support every programming language and development framework used by your development teams
  - Offer flexibility to tune SAST coverage for specific mission-critical applications
  - Provide low false positives (FPs) out of the box for developer experience
  - Integrate with Source Code Management (SCM) tools to automate scanning at code check-in and decorate pull requests (PRs) with security findings
  - Integrate with developer Integrated Development Environments (IDEs) to bring remediation into developer workflow
  - Integrate with bug ticketing tools to integrate remediation with the developer workflow
- [ ] "Shift-left" API security that can:
  - Discover APIs in source code (and not rely on API documentation)
  - Provide an API-centric view into vulnerabilities
  - Identify shadow and zombie APIs
- [ ] Secrets detection to identify and prevent secrets leakage in collaboration tools used during the development process

Build

- [ ] Software Composition Analysis (SCA) to secure the use of open source software and can:
  - Identify known vulnerabilities in open source libraries
  - Identify malicious code in open source dependencies
  - Integrate with build tools to automate scanning as part of the build process
  - Integrate with developer Integrated Development Environments (IDEs) to bring remediation into developer workflow
- [ ] Software Bill of Materials (SBOM) to track use of open source software in applications

Test

- [ ] DAST or penetration testing to test compiled applications in your test / dev environment prior to deploying

Deploy

- [ ] Container security to scan static container images for vulnerabilities and can:
  - Integrate with SCA to identify vulnerabilities in open source software
  - Integrate with container development tools like Docker Desktop
  - Integrate with runtime container security solutions to correlate and prioritize remediation
  - Infrastructure as Code (IaC) security to scan IaC templates for potential security risks and misconfigurations

Go-Live

- [ ] Container security to protect running containerized applications
- [ ] Cloud Workload Protection Platform (CWPP) to protect application workloads running in cloud environ
- [ ] Cloud Security Posture Management (CSPM) to monitor cloud infrastructure and identify resource misconfigurations ments
- [ ] Web Application and API Protection (WAAP) to protect against runtime attacks, like web application, DDoS, bot, and API attacks

Compliance and Documentation

- [ ] Compliance adherence
- [ ] Documentation

Visibility and Insight

- [ ] Unified dashboard to display vulnerabilities from all AppSec tools in one place
- [ ] Vulnerability management to analyze and triage vulnerabilities from all AppSec tools with one process
- [ ] Correlation of vulnerabilities across the SDLC stages
- [ ] Prioritization to focus remediation on the most exploitable vulnerabilities first