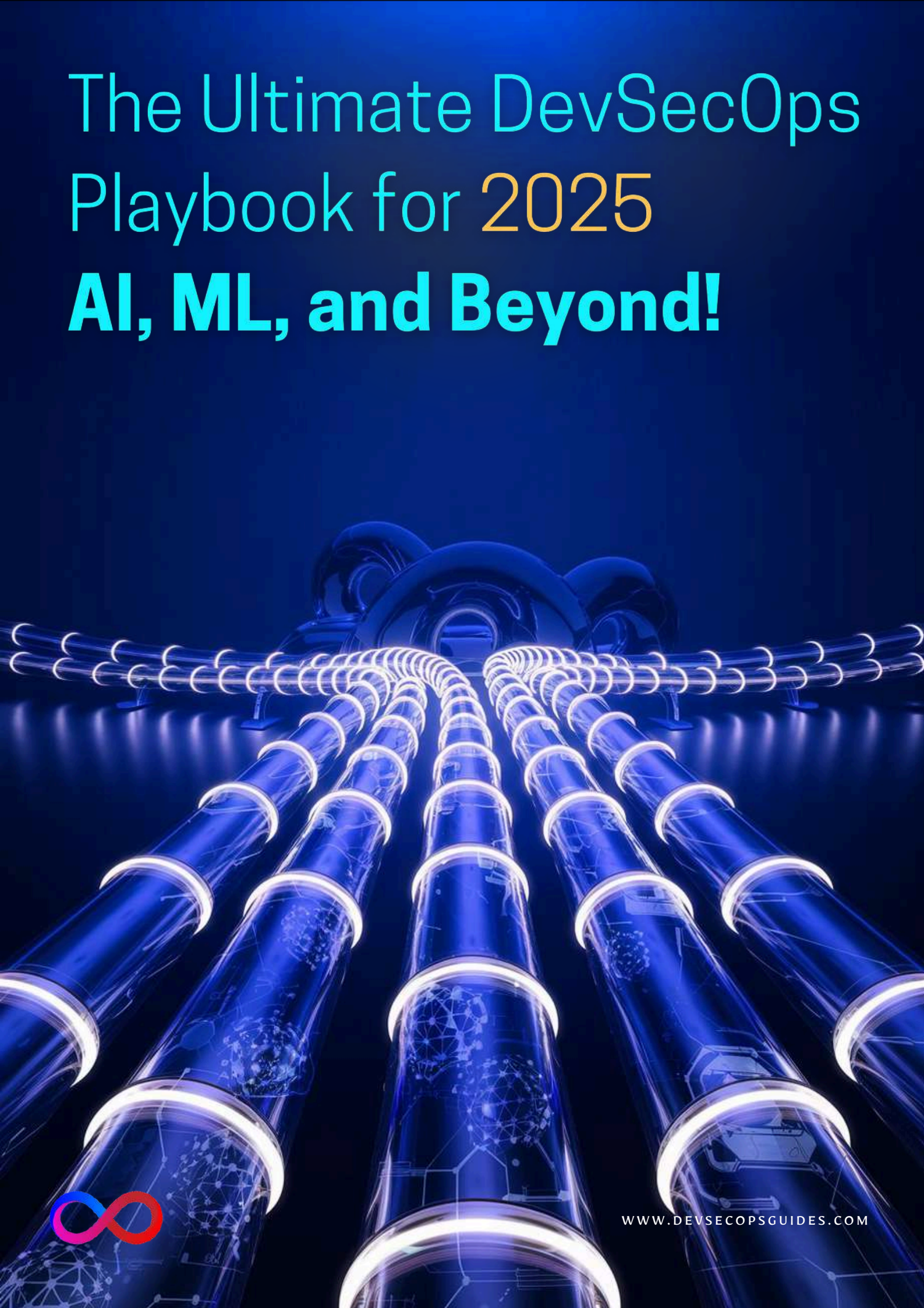
The Ultimate DevSecOps Playbook for 2025

AI, ML, and Beyond!



WWW.DEVSECOPSGUIDES.COM

FOREWORD

In an era where digital transformation continues to accelerate at an unprecedented pace, the intersection of development, security, and operations has become more critical than ever. As we venture into 2025, we find ourselves at a pivotal moment where artificial intelligence and machine learning are not just buzzwords, but fundamental pillars reshaping how we approach DevSecOps. This playbook represents countless hours of research, practical experience, and collaborative insights from industry leaders who are pioneering the future of secure software delivery.

The landscape of DevSecOps has evolved dramatically since its inception. What began as a movement to "shift security left" has transformed into a sophisticated ecosystem where AI-driven tools and machine learning algorithms work alongside human expertise to create more resilient and secure applications. Through this playbook, we aim to bridge the gap between traditional DevSecOps practices and emerging technologies, providing you with actionable strategies that can be implemented in organizations of any size or maturity level.

As practitioners on the frontlines of technological innovation, we understand the challenges you face daily – from managing complex technology stacks to implementing robust security measures while maintaining rapid delivery cycles. This playbook doesn't just focus on theoretical frameworks; it delves deep into practical implementations, real-world case studies, and concrete metrics that will help you measure and improve your DevSecOps initiatives. We've carefully curated content that addresses both the technical and cultural aspects of modern DevSecOps, recognizing that success in this field requires a holistic approach.

The integration of AI and ML into DevSecOps isn't just about automation or efficiency – it's about fundamentally reimagining how we approach security in the software development lifecycle. As you journey through this playbook, you'll discover how these technologies can enhance your team's capabilities, from automated threat detection to predictive analysis of potential vulnerabilities. Whether you're just beginning your DevSecOps journey or looking to elevate your existing practices, this guide will serve as your compass in navigating the exciting and complex landscape of modern secure software development.

DevSecOps Community

ACKNOWLEDGEMENT

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

Jérémy Lanfranchi - GitGuardian
Anna Nabiullina - GitGuardian
Amanda McCarvill - Semgrep
Sarah Nelson - Semgrep
Dan Barahona - APIsec University
Timo Pagel
Eslam Samy Hosney
Carol Valencia
Aristide Bouix
Charles Chibueze
Burcu YARAR
Sophie Edwards
Dan Wiliams

SECURE YOUR PIPELINE WITH

GitGuardian

GitGuardian is a leading security platform specializing in secrets detection and remediation for DevSecOps teams. By leveraging AI-driven scanning and real-time monitoring, it helps organizations detect and secure hardcoded secrets—such as API keys, credentials, and sensitive tokens—across source code, CI/CD pipelines, and Infrastructure as Code (IaC). A key focus of GitGuardian is securing Non-Human Identities (NHI) and their secrets, ensuring compliance with industry standards while preventing unauthorized access. The platform seamlessly integrates with GitHub, GitLab, Bitbucket, and enterprise security workflows, automating security checks directly within CI/CD pipelines using ggshield, GitGuardian's CLI tool. Additionally, GitGuardian incorporates security considerations at the earliest stages of the Software Development Life Cycle (SDLC), helping organizations proactively reduce risks and secure the software supply chain. Trusted by enterprises worldwide, GitGuardian plays a crucial role in preventing data breaches and strengthening DevSecOps resilience.

Manages credentials and secrets while ensuring compliance with standards. update the description with **"Secures Non-Human Identities and their secrets"** Automates security checks directly within CI/CD workflows - add ggshield by GitGuardian Detects secrets from source code in your CL Integrates security considerations into the earliest stages of the SDLC. add ggshield by GitGuardian Detects secrets from source code in your CL.

For more details, visit: GitGuardian DevSecOps Guides (https://s.gitguardian.com/devsecops-guides)

LEARN API SECURITY WITH

APISEC UNIVERSITY

APIs are the backbone of modern applications, enabling seamless integrations and rapid innovation. However, they have also become a primary target for cyber threats, leading to data breaches and security risks. APIsec University is dedicated to educating and equipping security professionals with the knowledge needed to identify, mitigate, and defend against API vulnerabilities. Through comprehensive courses, hands-on labs, and expert-led training, APIsec University helps security teams and developers stay ahead of evolving threats. Whether you're an experienced professional or just starting in API security, APIsec University offers free, **high-quality training** to strengthen your skills.

🎉 Get started today with APIsec University (https://www.apisecuniversity.com/)

Also for a free CASA voucher! Claim yours here:
APIsec University (https://www.linkedin.com/company/devsecopsguides/?viewAsMember=true)

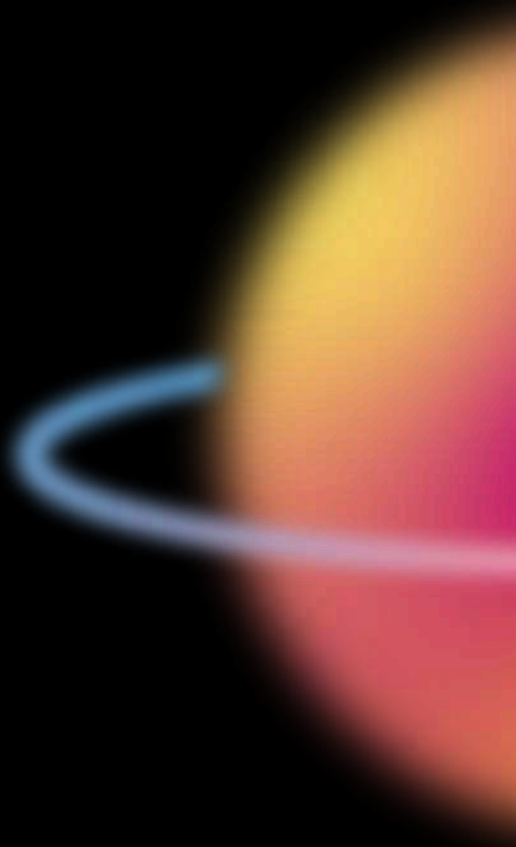
TABLE OF CONTENT

EXECUTIVE SUMMARY

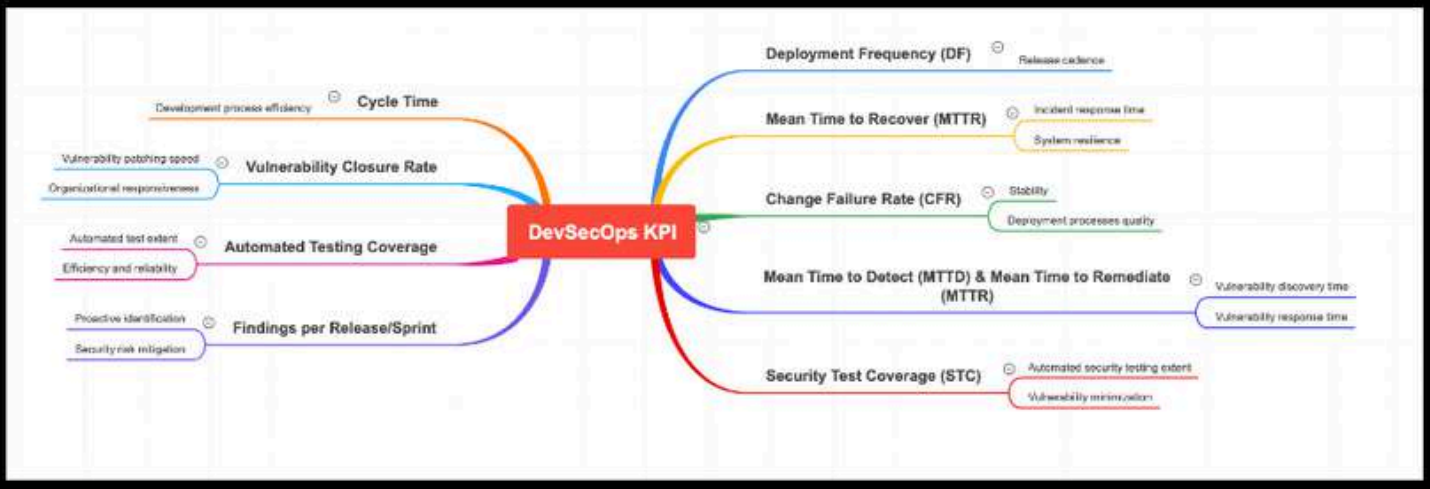
As cyber threats evolve and software development accelerates, DevSecOps is entering a new era driven by AI, Machine Learning (ML), and automation. The Ultimate DevSecOps Playbook for 2025 provides security leaders, CISOs, and DevSecOps professionals with a strategic roadmap to embed security into every stage of the Software Development Life Cycle (SDLC). This playbook explores AI-powered threat detection, ML-driven anomaly detection, and autonomous security workflows—enabling organizations to scale security operations, reduce risk, and accelerate secure software delivery. With real-world case studies, cutting-edge frameworks, and actionable best practices, this guide empowers teams to stay ahead of emerging threats while maintaining agility and innovation.

As we move into 2025, the integration of AI and ML in DevSecOps is no longer optional—it's a necessity. This playbook highlights how security automation, adaptive risk management, and intelligent compliance can fortify your organization against supply chain attacks, API threats, and AI-generated vulnerabilities. Whether you're a CISO strategizing security investments, a DevSecOps leader optimizing your pipeline, or a security engineer implementing next-gen defenses, this guide equips you with the insights, tools, and methodologies needed to build resilient, AI-driven security programs.

01 Most Important KPIs in DevSecOps Teams for 2025



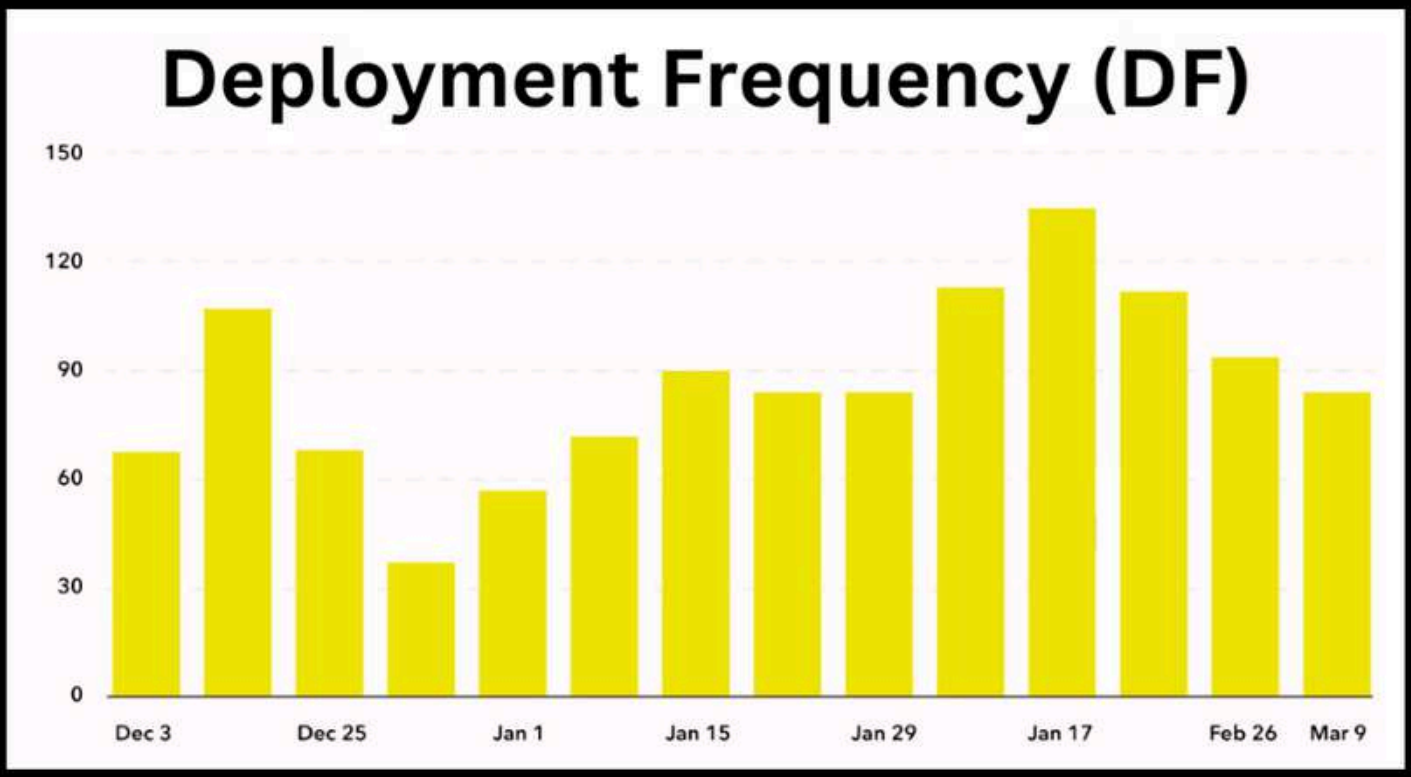
Most Important KPIs in DevSecOps Teams for 2025



KPI	Description
Deployment Frequency (DF)	Measures how often code is deployed to production. High frequency ensures agility and responsiveness.
Mean Time to Recover (MTTR)	Tracks the time needed to recover from an incident, reflecting system resilience and incident handling.
Change Failure Rate (CFR)	Percentage of deployments causing issues, indicating process quality and stability.
Mean Time to Detect (MTTD)	Average time to detect security vulnerabilities, crucial for proactive threat management.
Mean Time to Remediate (MTTR)	Average time to fix vulnerabilities, showcasing the team's ability to respond quickly to threats.
Security Test Coverage (STC)	Percentage of code covered by automated security tests, ensuring fewer blind spots.
Findings per Release/Sprint	Tracks the number of security issues per release/sprint, emphasizing preemptive security practices.
Automated Testing Coverage	Measures the extent of automated testing, enhancing efficiency and reliability.
Vulnerability Closure Rate	Measures how quickly vulnerabilities are patched, reflecting organizational responsiveness.
Cycle Time	The time taken to move a change from ideation to production, indicating process efficiency.

Deployment Frequency (DF)

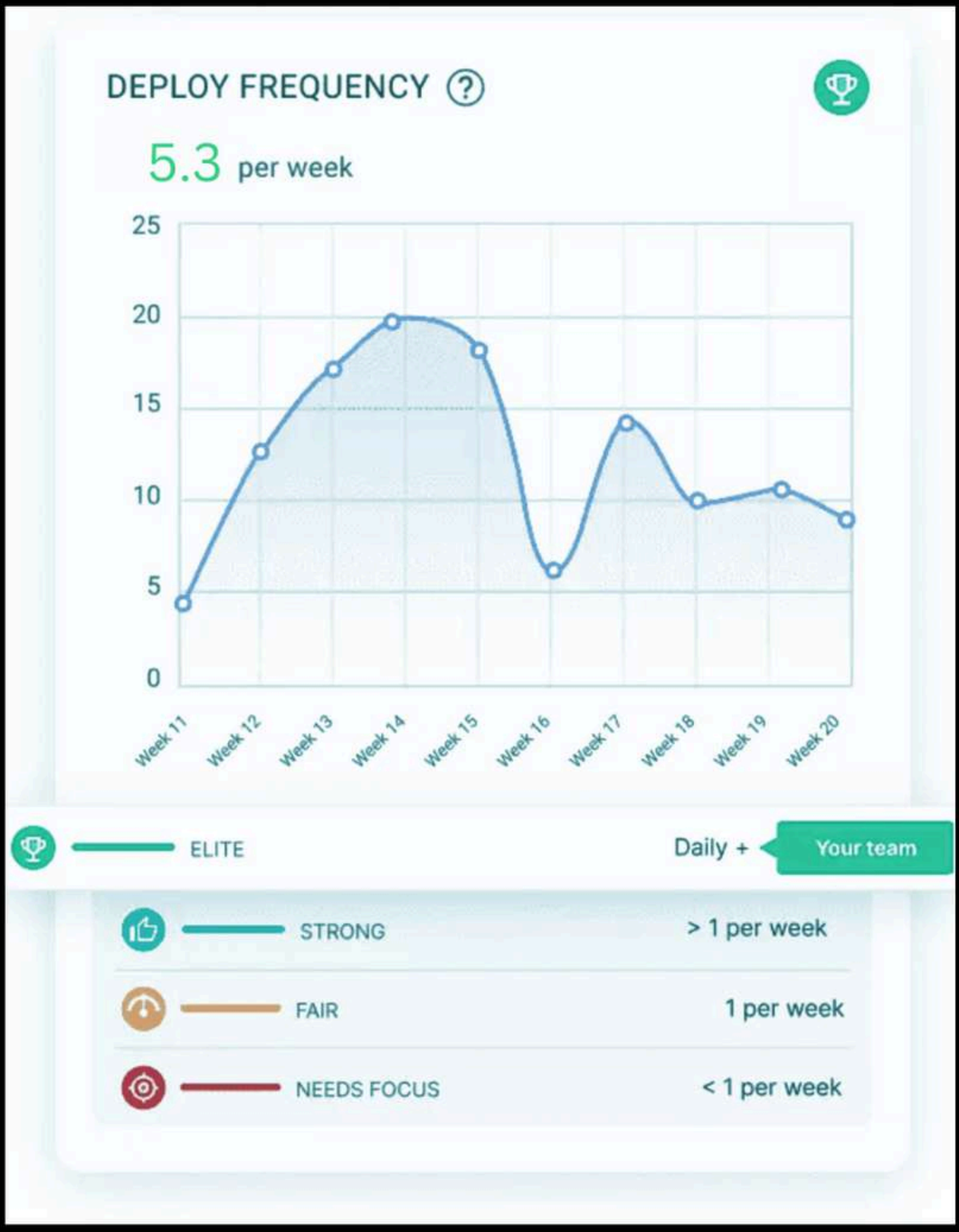
What is Deployment Frequency?



Deployment Frequency (DF) is a crucial DevSecOps Key Performance Indicator (KPI) that measures **how often code changes are deployed to production environments**. The sources provide insights into the benefits of DF, especially in the context of a DevSecOps approach. Here is a detailed discussion of those benefits and other considerations regarding DF.

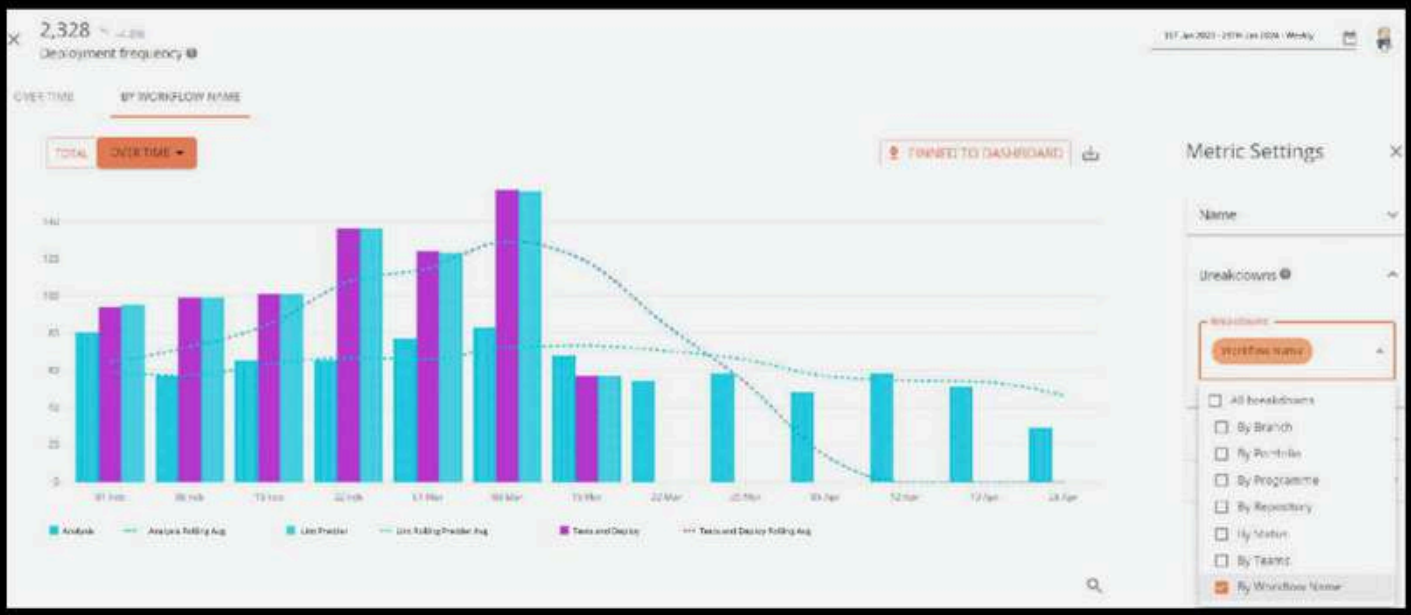
Deployment frequency (df) is a key performance indicator (KPI) in DevSecOps that measures how often software is deployed to production. It is a critical metric for evaluating the speed and efficiency of a development team.

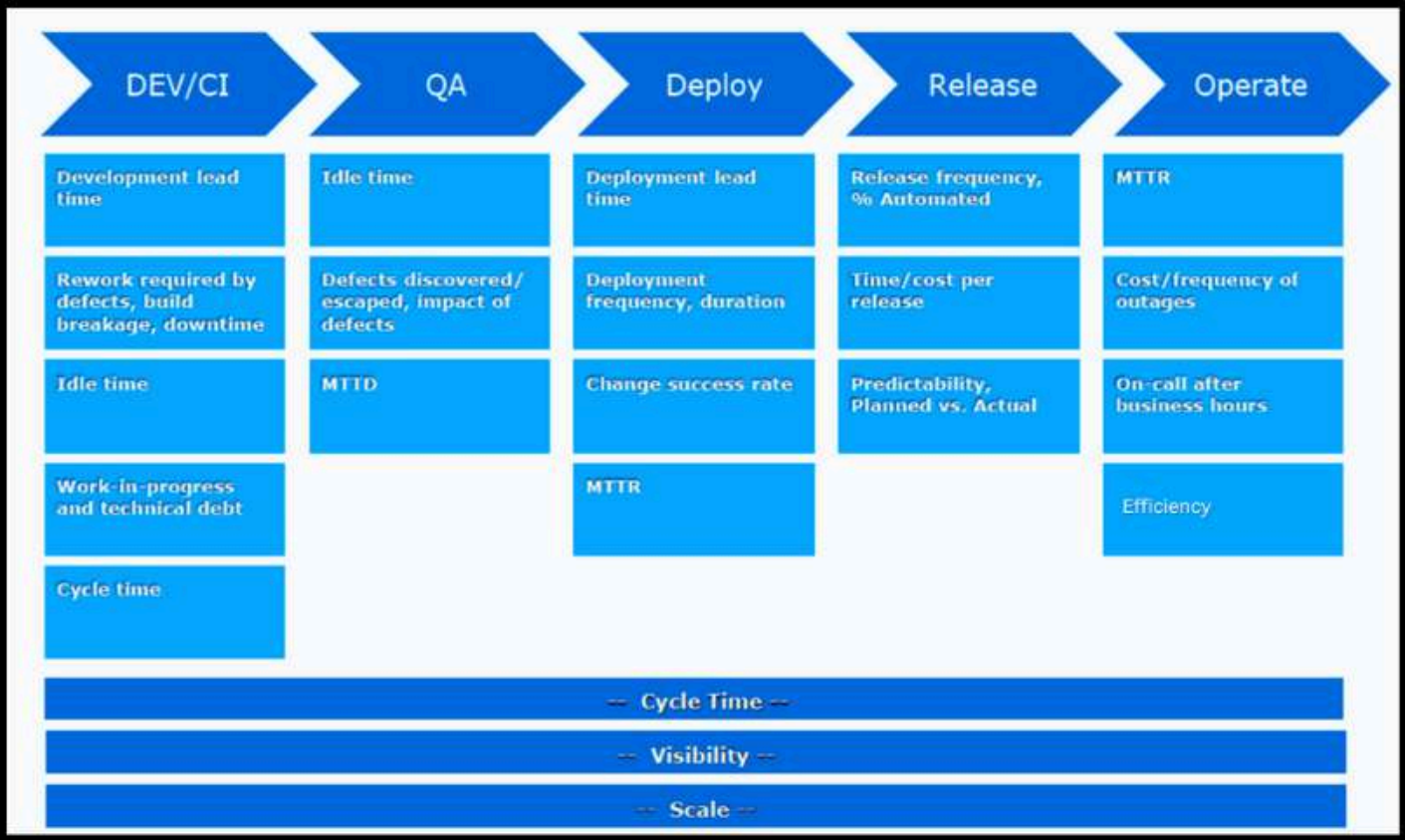
Why is Deployment Frequency important?



Deployment frequency is essential because it directly impacts the time-to-market for new features and bug fixes. High deployment frequency indicates a team's ability to quickly and reliably deliver software changes, which is a hallmark of successful DevOps adoption.

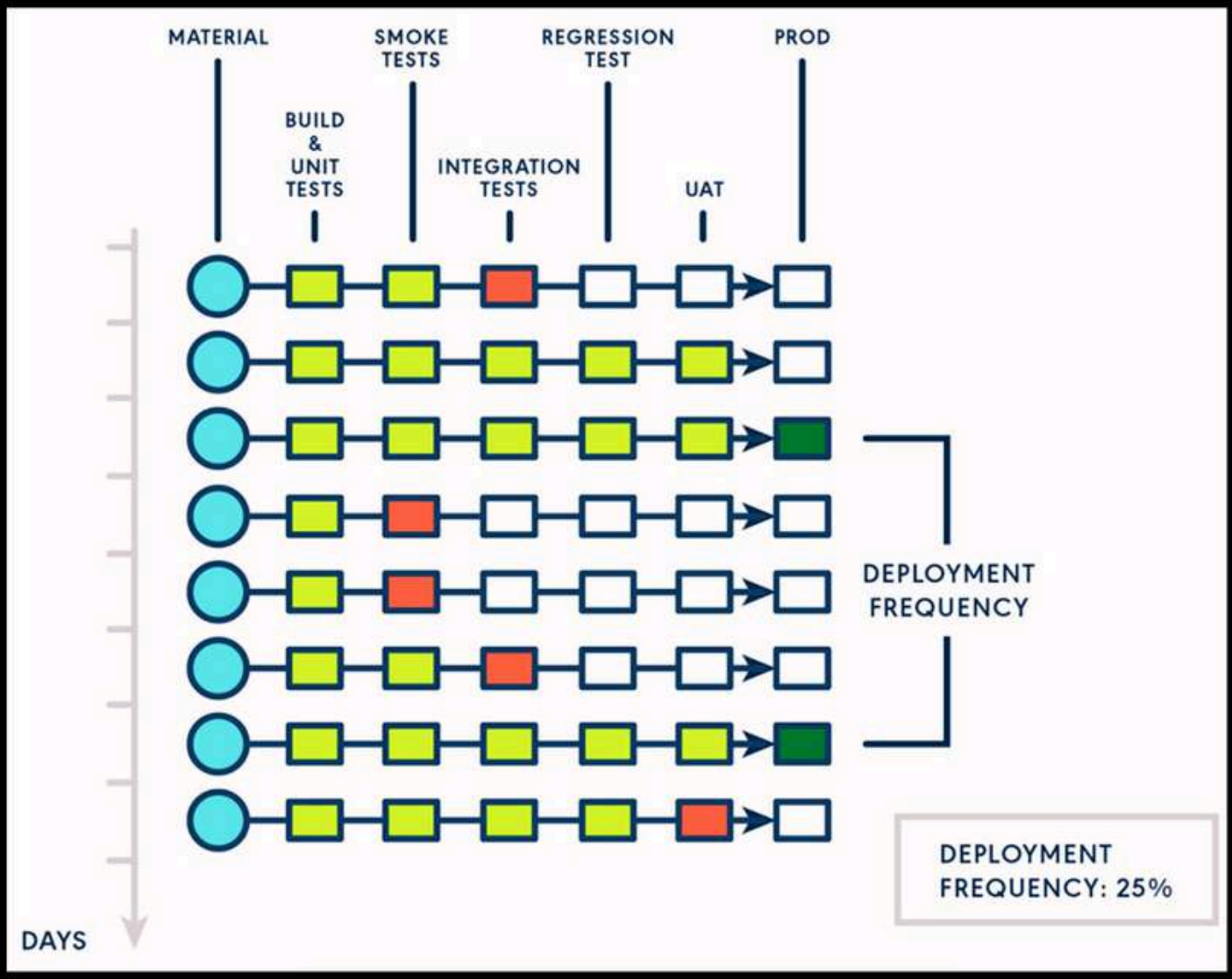
How to measure Deployment Frequency?





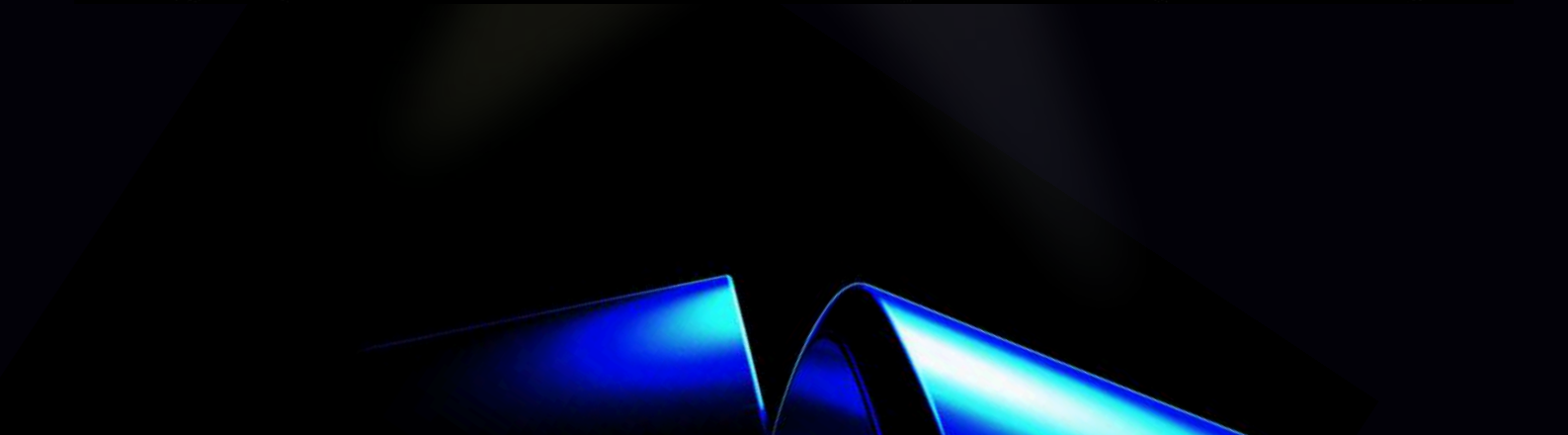
Deployment frequency can be measured by tracking the number of deployments per unit of time, such as deployments per day or week. This metric can be calculated using tools like Jenkins, GitLab CI/CD, or other continuous integration and continuous deployment (CI/CD) pipelines.

What are the benefits of high Deployment Frequency?



High deployment frequency offers several benefits, including:

- Faster time-to-market for new features and bug fixes
- Improved customer satisfaction through faster delivery of new features and bug fixes
- Increased agility and responsiveness to changing customer needs
- Reduced risk of technical debt and code rot
- **Faster Release Cycles:** High DF allows organizations to release new features and bug fixes to users more rapidly. This agility can provide a competitive edge, enabling companies to respond quickly to market demands and user feedback.
- **Increased Quality and Reliability:** Frequent deployments, coupled with continuous testing in a CI/CD pipeline, help identify and address bugs earlier in the development process. This leads to more reliable and higher-quality software, enhancing user satisfaction and trust.
- **Enhanced Developer Productivity:** By automating the deployment process and integrating security checks into the pipeline, developers can focus on coding rather than time-consuming manual tasks.
- **Rapid Feedback:** Frequent deployments allow for quicker feedback on code changes, enabling developers to identify and resolve issues more efficiently.
- **Improved Customer Satisfaction:** Frequent deployments allow organizations to provide a seamless user experience with less disruption during application updates. Addressing customer-reported issues quickly is a key aspect of good service, leading to greater customer satisfaction and loyalty.
- **Reduced Downtime:** A proactive approach to security, integrated with a high DF, can minimize the likelihood and impact of security-related outages.



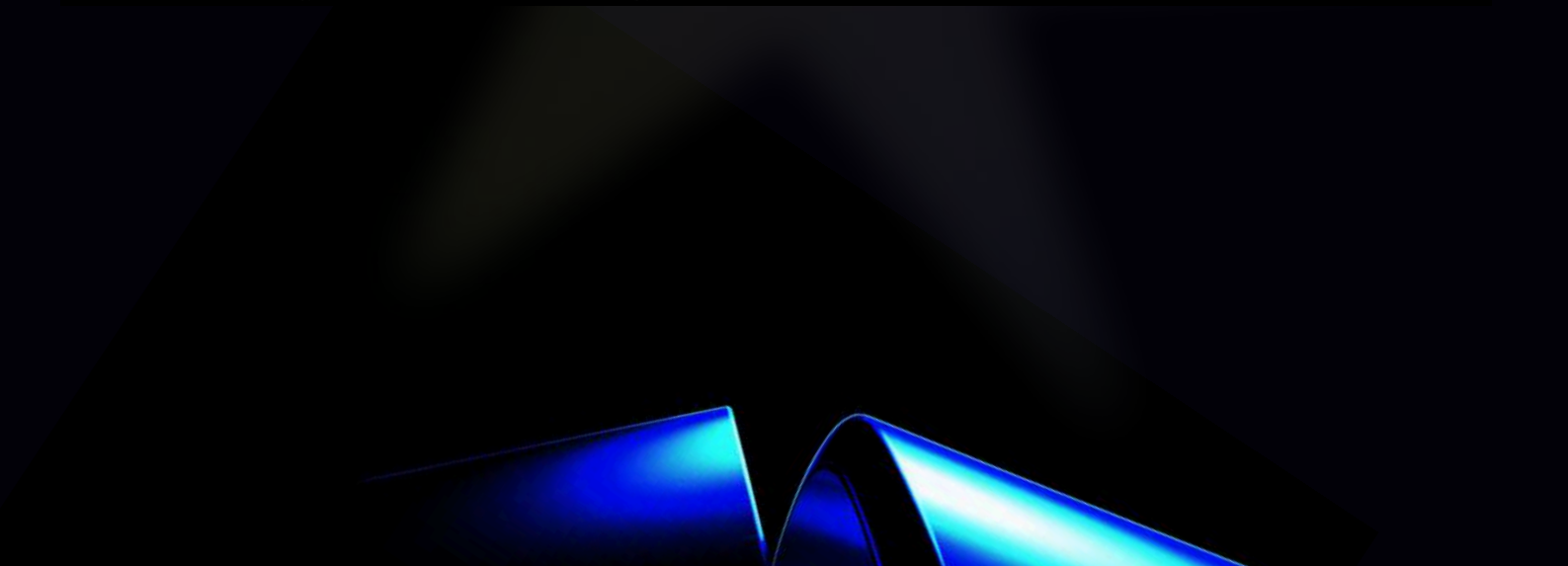
What are the challenges of achieving high Deployment Frequency?

Considerations for Deployment Frequency:

- **Maturity Level:** The appropriate deployment frequency for an organization depends on its DevSecOps maturity level. Organizations with mature DevSecOps practices and robust automated processes are better equipped to handle high DF.
- **Business Needs:** The desired deployment frequency should align with the specific goals and needs of the business. For example, a company focusing on rapid innovation might prioritize a higher DF than an organization working on a mature and stable product.
- **Risk Tolerance:** A higher DF inherently comes with a higher risk of introducing bugs or vulnerabilities. Organizations need to balance their desired speed with their tolerance for potential issues. Robust testing, monitoring, and rollback mechanisms are essential to mitigate these risks.
- **Team Collaboration and Communication:** Effective collaboration and communication between development, security, and operations teams are crucial to successfully handle a high DF. This includes regular feedback, knowledge-sharing sessions, and efficient conflict resolution strategies.

Achieving high deployment frequency can be challenging due to:

- Complexity of the deployment process
- Frequency of code changes
- Quality and reliability of the code
- Availability of resources and personnel

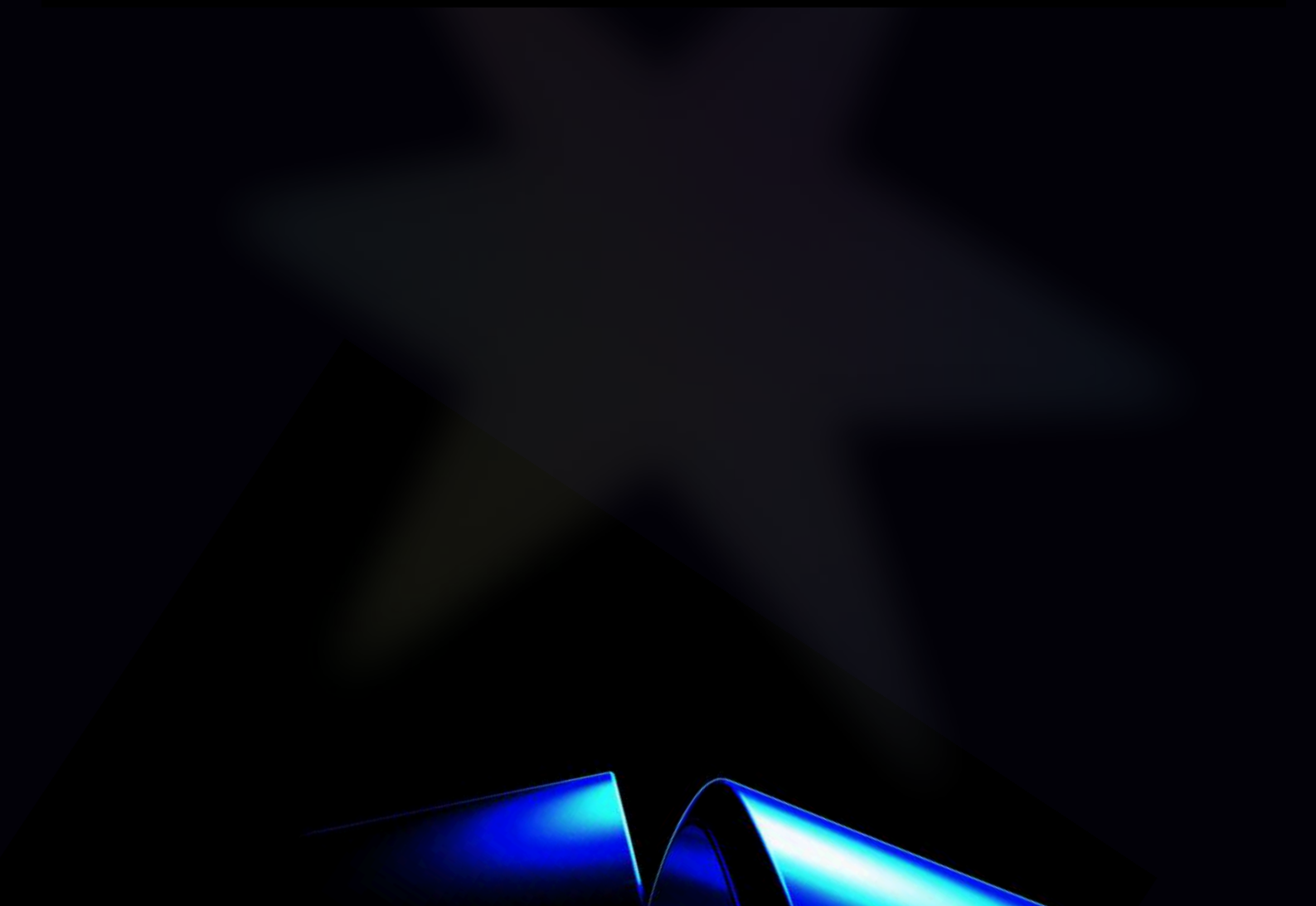


Recommendations for improving Deployment Frequency

To improve deployment frequency, consider the following recommendations:

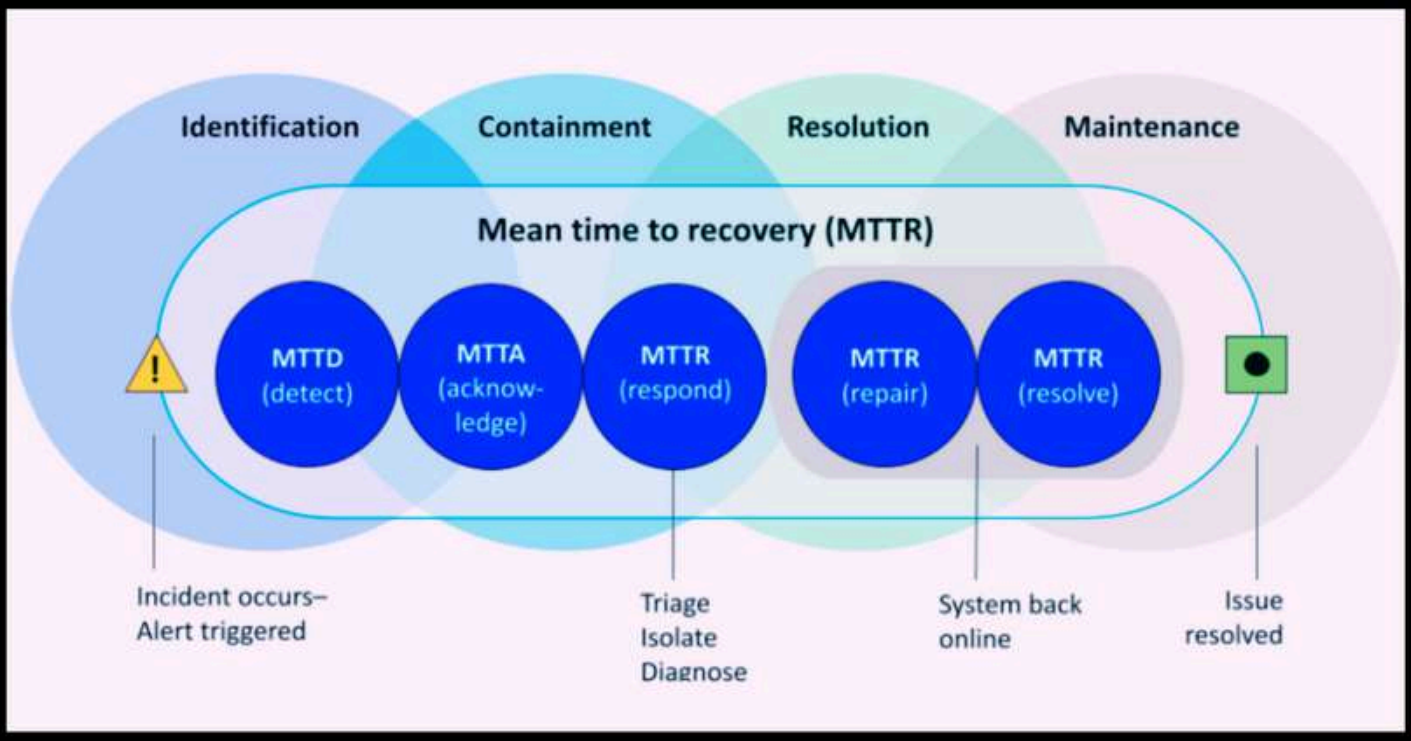
- Implement continuous integration and continuous deployment (CI/CD) pipelines
- Automate testing and validation processes
- Use containerization and orchestration tools like Docker and Kubernetes
- Implement a culture of continuous learning and improvement
- Monitor and analyze deployment metrics to identify areas for improvement

It's also important to note that the optimal deployment frequency is not about achieving a specific number, but about finding a **sustainable pace that aligns with business goals, risk tolerance, and team capabilities.** The emphasis should be on continuous improvement and adapting the deployment frequency based on data and feedback.



Mean Time to Recover (MTTR)

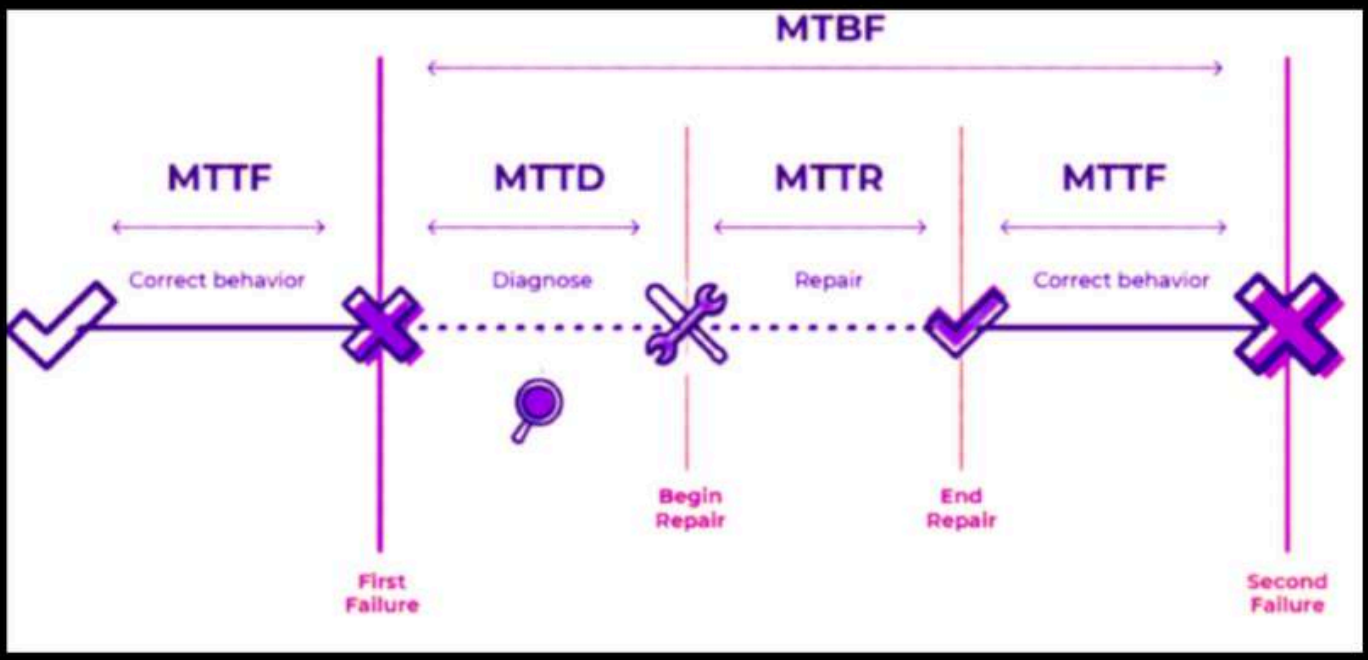
What is Mean Time to Recovery (MTTR) in DevOps?



Mean Time to Recover (MTTR) is a critical DevSecOps KPI that measures the average time it takes to restore a system to a fully functional state after a failure or incident. The sources provide valuable information about the benefits of focusing on MTTR as a performance metric within a DevSecOps approach.

MTTR is a key metric in DevOps that measures how quickly a system or service can be restored to a functional state after a failure or interruption. It is an essential indicator of a team's ability to respond to and resolve issues efficiently.

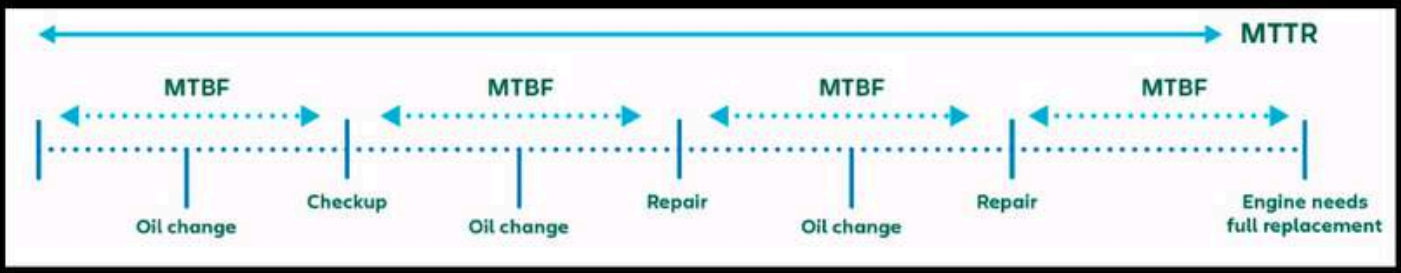
Why is MTTR important in DevOps?



MTTR is crucial in DevOps as it directly impacts the overall quality and reliability of a system or service. A low MTTR indicates that a team can quickly identify and resolve issues, reducing the impact on users and improving overall system stability.

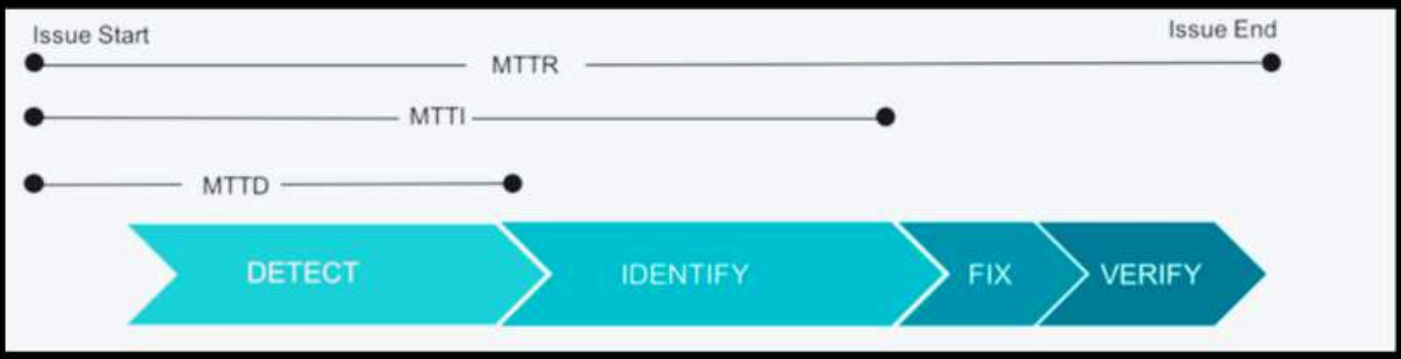
- **Increasing Complexity of Systems:** As software systems continue to become more complex and interconnected, the potential impact of failures increases, making quick recovery even more critical.
- **Growing Importance of Automation:** The trend toward automation in incident detection, diagnosis, and remediation will likely lead to further improvements in MTTR.
- **Focus on Continuous Improvement:** DevSecOps emphasizes continuous improvement, and MTTR is a metric that can be consistently monitored and optimized.
- **Emphasis on Observability:** A focus on observability—the ability to understand the internal state of a system by examining its external outputs—is gaining traction in the DevSecOps world. This enhanced visibility into system behavior is likely to contribute to faster and more efficient incident resolution, improving MTTR.

How to calculate MTTR?



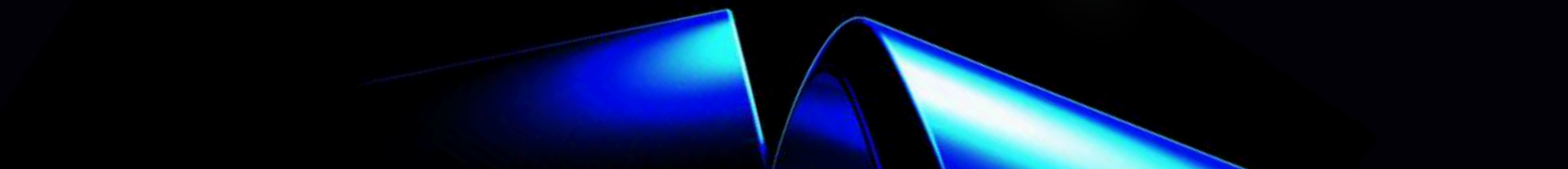
MTTR can be calculated by dividing the total time spent on recovery by the number of incidents. For example, if a team spends 10 hours recovering from 2 incidents, the MTTR would be 5 hours per incident.

Benefits of tracking MTTR



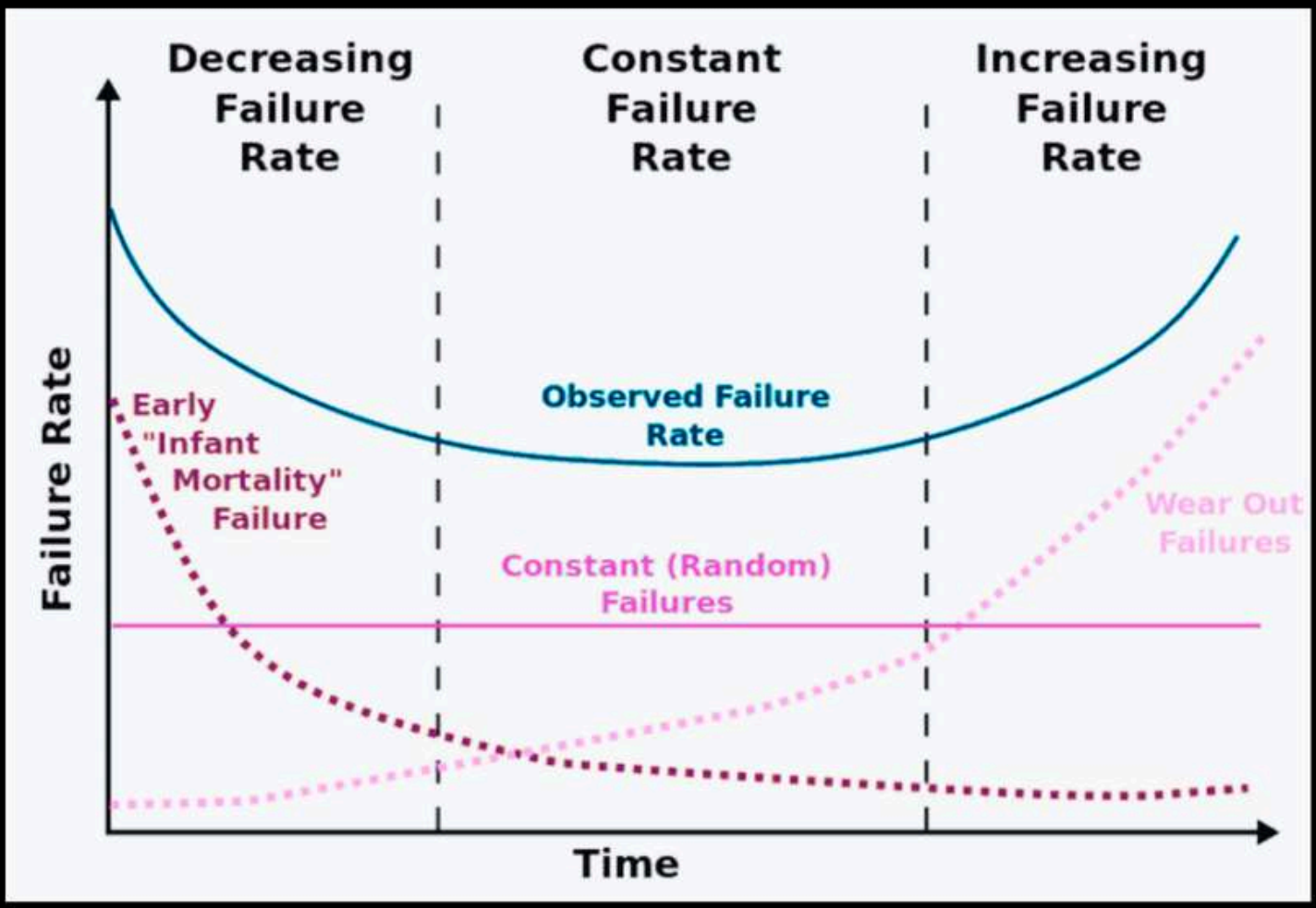
Benefits of a Low MTTR

- **Minimize Downtime:** A low MTTR is directly associated with minimizing downtime. Reducing downtime is crucial for businesses as it directly impacts service availability, customer satisfaction, and revenue.
- **Improved System Reliability and Resilience:** Focusing on MTTR encourages organizations to build systems that are more robust and capable of quick recovery. This leads to more reliable and resilient software that can withstand failures and disruptions, improving overall operational stability.
- **Faster Incident Response:** A low MTTR implies a well-defined incident response process and skilled teams that can quickly diagnose and resolve issues. This efficient incident response minimizes the impact of failures and helps maintain customer trust.
- **Reduced Costs:** Downtime is expensive. By reducing the time it takes to recover from failures, organizations can minimize financial losses and operational costs.
- **Improved Customer Experience:** Quick recovery from failures ensures minimal disruption to users, leading to a better overall customer experience.
- **Support Business Goals:** Fast and stable software delivery, which includes efficient recovery from failures, allows organizations to experiment, learn, and respond to market changes more effectively. This agility is crucial for achieving business goals and staying ahead of the competition.
- **Increased Team Confidence and Agility:** A low MTTR can boost team confidence in their ability to handle failures effectively. This confidence, coupled with efficient recovery mechanisms, can encourage greater agility in experimenting with new features and deployments.



Change Failure Rate (CFR)

What is Change Failure Rate (CFR)?



Change Failure Rate (CFR) is a key DevSecOps KPI that tracks the percentage of deployments to production that result in failures, requiring either an aborted deployment or a rollback to a previous working version. Sources highlight CFR as a critical measure of stability and a focal point of software development, helping to refine both software quality and the processes used to create it.

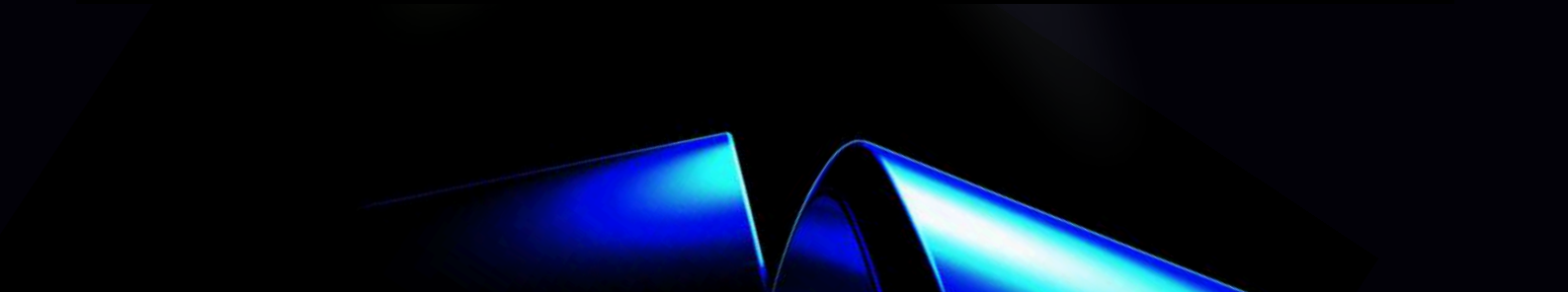
Change Failure Rate (CFR) is a metric that measures the percentage of changes that result in unintended consequences, such as downtime, errors, or negative impact on users. It is calculated by dividing the number of failed changes by the total number of changes made over a specific time.

Why is CFR important?

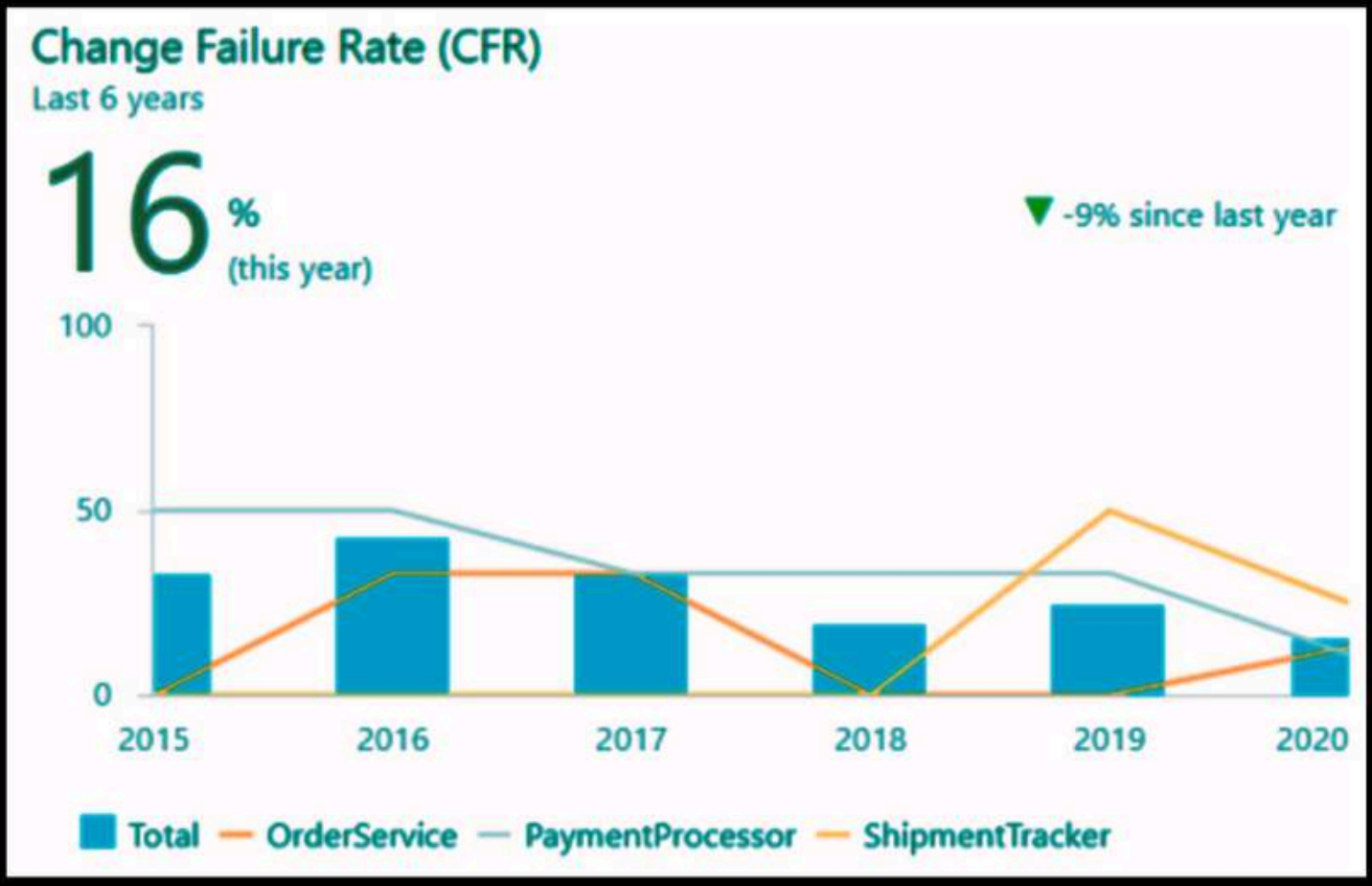
	Low	Mid	High
Change failure rate	<30%	16% - 30%	0% - 15%

CFR is an essential metric for organizations to measure the effectiveness of their change management processes and identify areas for improvement. It helps gain valuable insights into the stability of systems, processes, and technologies.

- **Understanding System Stability:** CFR directly indicates the stability of your deployment process and the overall reliability of your software releases. A high CFR suggests potential problems in various areas, prompting further investigation and improvement efforts.
- **Identifying Bottlenecks and Inefficiencies:** A high CFR can point towards underlying issues in the development pipeline, such as inadequate testing practices, poor code quality, insufficient automation, or unclear operational goals and processes. Analyzing CFR trends helps to pinpoint these bottlenecks and guide improvement initiatives.
- **Improving Software Quality and Customer Satisfaction:** By addressing the root causes of a high CFR, organizations can significantly improve the quality of their software releases. This, in turn, leads to fewer bugs and a more stable user experience, ultimately increasing customer satisfaction and trust.



How to calculate CFR?



The formula to calculate CFR is as follows:

CFR = (Number of Failed Changes / Total Number of Changes) x 100

Where:

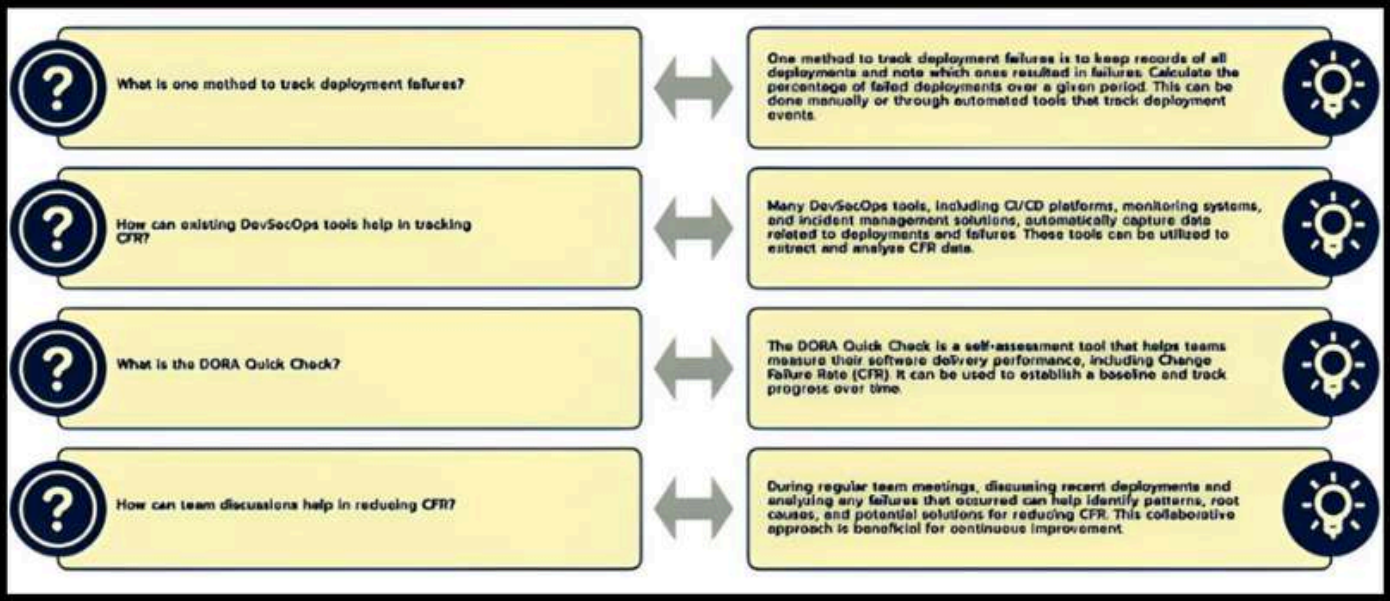
- Number of Failed Changes: The number of changes that resulted in unintended consequences or disruption.
- Total Number of Changes: The total number of changes made to the system or component over a specified time.

What is a good CFR?

Software delivery performance metric	Low	Medium	High
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	Between once per month and once every 6 months	Between once per week and once per month	On-demand (multiple deploys per day)
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Between one month and six months	Between one week and one month	Between one day and one week
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Between one week and one month	Between one day and one week	Less than one day
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	46%-60%	16%-30%	0%-15%

A "good" CFR depends on various factors, including the size and complexity of the IT system, the level of risk associated with changes, and the company's overall goals and objectives. However, as a general rule, organizations strive to keep their CFR as low as possible, ideally less than 5%.

Implementing CFR Tracking



Sources describe various methods for tracking and measuring CFR, including:

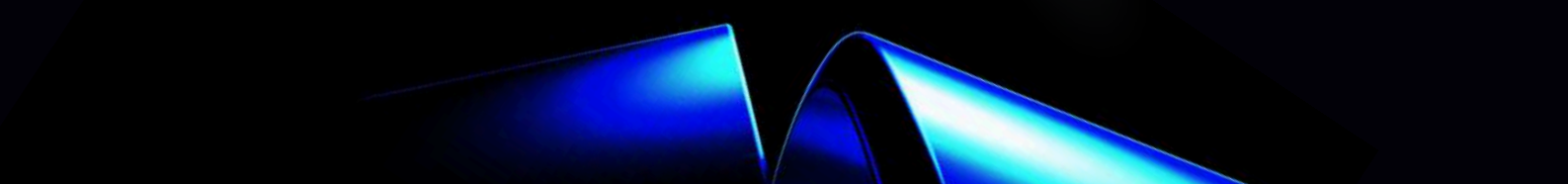
1. **Direct Tracking:** Keep records of all deployments and note which ones resulted in failures. Calculate the percentage of failed deployments over a given period. This can be done manually or through automated tools that track deployment events.
2. **Leveraging Existing Tools:** Many DevSecOps tools, including CI/CD platforms, monitoring systems, and incident management solutions, automatically capture data related to deployments and failures. Utilize these tools to extract and analyze CFR data.
3. **DORA Quick Check:** The DORA Quick Check is a self-assessment tool that helps teams measure their software delivery performance, including CFR. It can be used to establish a baseline and track progress over time.
4. **Team Discussions and Reflection:** During regular team meetings, discuss recent deployments and analyze any failures that occurred. This collaborative approach can help identify patterns, root causes, and potential solutions for reducing CFR.

Interpreting CFR Data

	Deployment frequency	Lead time for changes	Change failure rate	Mean time to recovery
Elite	on-demand	less than one day	5 percent	less than 1 hour
High	once a day to once a week	one day to one week	10 percent	less than 1 day
Medium	once a week to once a month	one week to one month	15 percent	1 day to 1 week
Low	once a week to once a month	one week to one month	64 percent	1 month to 6 months

While CFR provides a valuable measure of stability, interpreting it in isolation can be misleading. For a more holistic understanding of your DevSecOps performance, consider the following:

- **Contextualize CFR:** A low CFR might be acceptable for a mature and stable product, whereas a high-growth, rapidly evolving product might have a naturally higher CFR. What matters is understanding the acceptable threshold for your specific context and continuously striving for improvement.
- **Combine CFR with Other Metrics:** Analyze CFR in conjunction with other DevSecOps KPIs, such as deployment frequency, lead time for changes, mean time to recovery, and rework rate. This multi-dimensional view provides a more comprehensive picture of your overall performance.

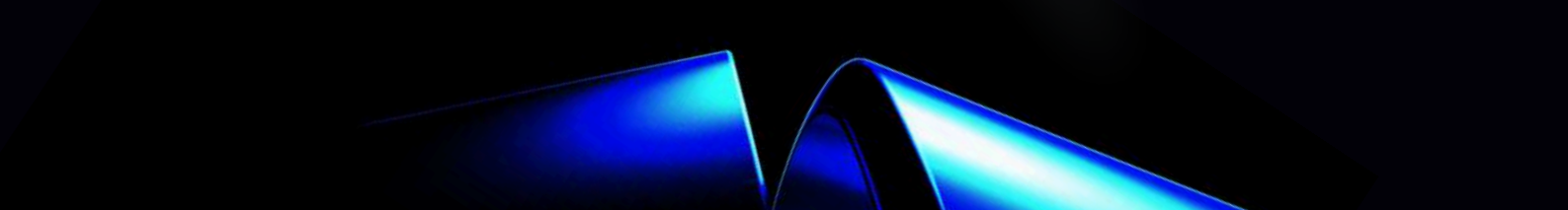


Mean Time to Detect (MTTD)

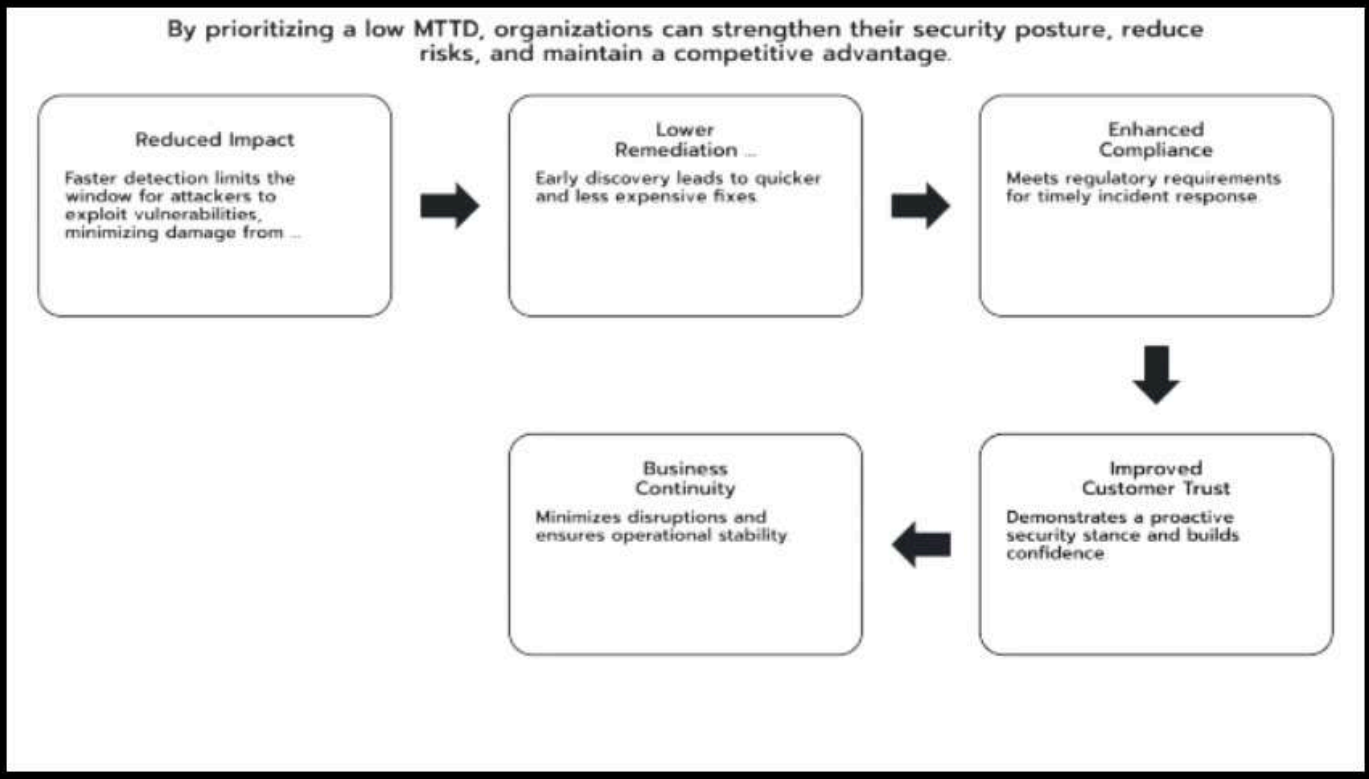
Start time	Detection Time	Time elapsed in minutes
8:20 AM	12:20 AM	240
9:05 AM	9:51 AM	46
7:00 AM	7:10 AM	10
3:37 PM	4:17 PM	40
2:42 PM	3:21 PM	39

Mean Time To Detect (MTTD) is a crucial security metric in DevSecOps that measures the average time it takes to identify a security issue from the moment it occurs. While the sources don't provide a precise definition of MTTD, they discuss various concepts and metrics related to security incident detection and resolution, offering insights into the significance of MTTD within a DevSecOps framework.

MTTD, or Mean Time to Detect, is a measure of how long a problem exists in an IT deployment before the appropriate parties become aware of it. It is also known as Mean Time to Discover or Mean Time to Identify. MTTD is a common key performance indicator (KPI) for IT incident management.



Why is MTTD Important?

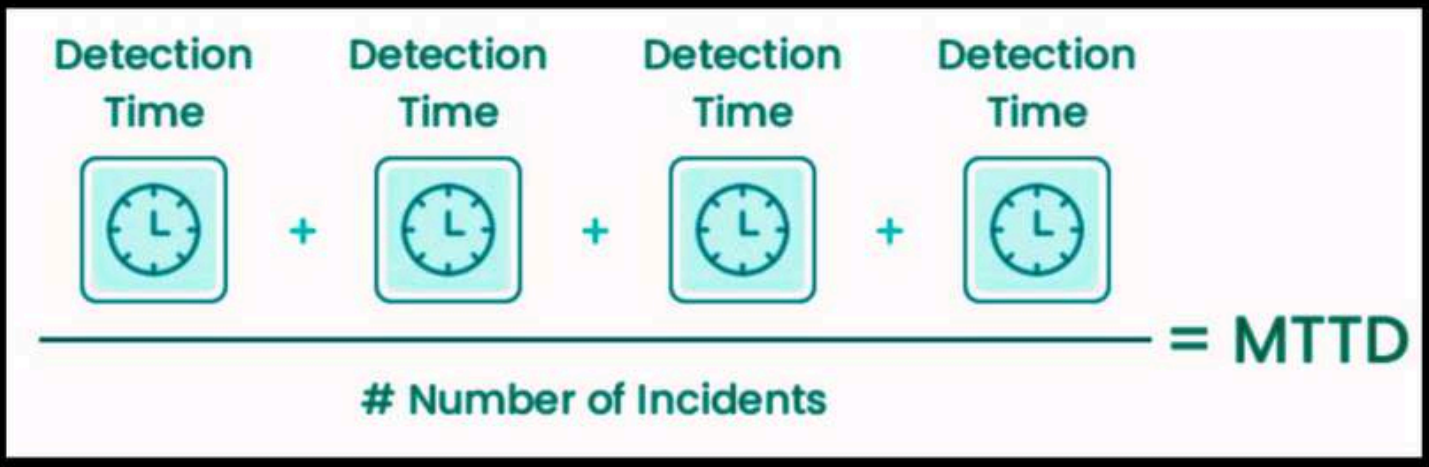


MTTD is important because it indicates how quickly an organization can detect and respond to IT issues. A shorter MTTD indicates that users suffer from IT disruptions for less time compared with a longer MTTD. IT organizations strive to detect issues before end users do in order to minimize disruption.

Minimizing MTTD is critical for effective security management in a DevSecOps environment. A low MTTD brings several benefits:

- **Minimize the Impact of Security Incidents:** The faster you detect a security issue, the less time attackers have to exploit it and cause damage. This reduces the potential impact of data breaches, system compromises, and other security incidents.
- **Reduce Remediation Costs:** Early detection of security issues typically translates to faster and less costly remediation. The longer a security vulnerability remains undetected, the more extensive and expensive the remediation efforts can become.

How to Calculate MTTD



The formula for MTTD is the sum of all incident detection times for a given technician, team, or time period divided by the total number of incidents. To gauge performance, IT teams can then compare the resulting MTTD with those for other time periods, other incident response teams, and so on.

Example of Calculating MTTD

For example, say the 24/7 IT operations support team for internal applications at a national bank tracks its MTTD monthly. In August, the team experienced eight incidents, and it determined each incident's start and discovery time based on system logs, the organization's intrusion detection system, and help desk tickets filed by users.

MTTD = (67 + 257 + 45 + 42 + 191 + 15 + 406 + 143) / 8 MTTD = 145.75 minutes

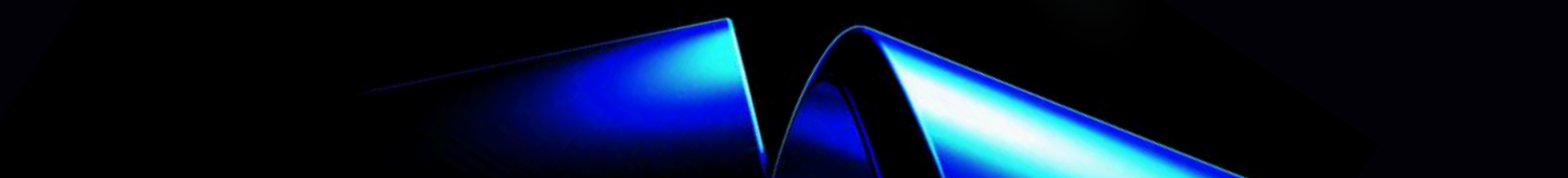
Some organizations might choose to remove outliers from the equation, as shown in Table 2. In this case, 406 minutes is the highest time to detect, and 15 minutes is the lowest. Without these outliers, the MTTD equals 124.17 minutes.

Implementing MTTD Tracking

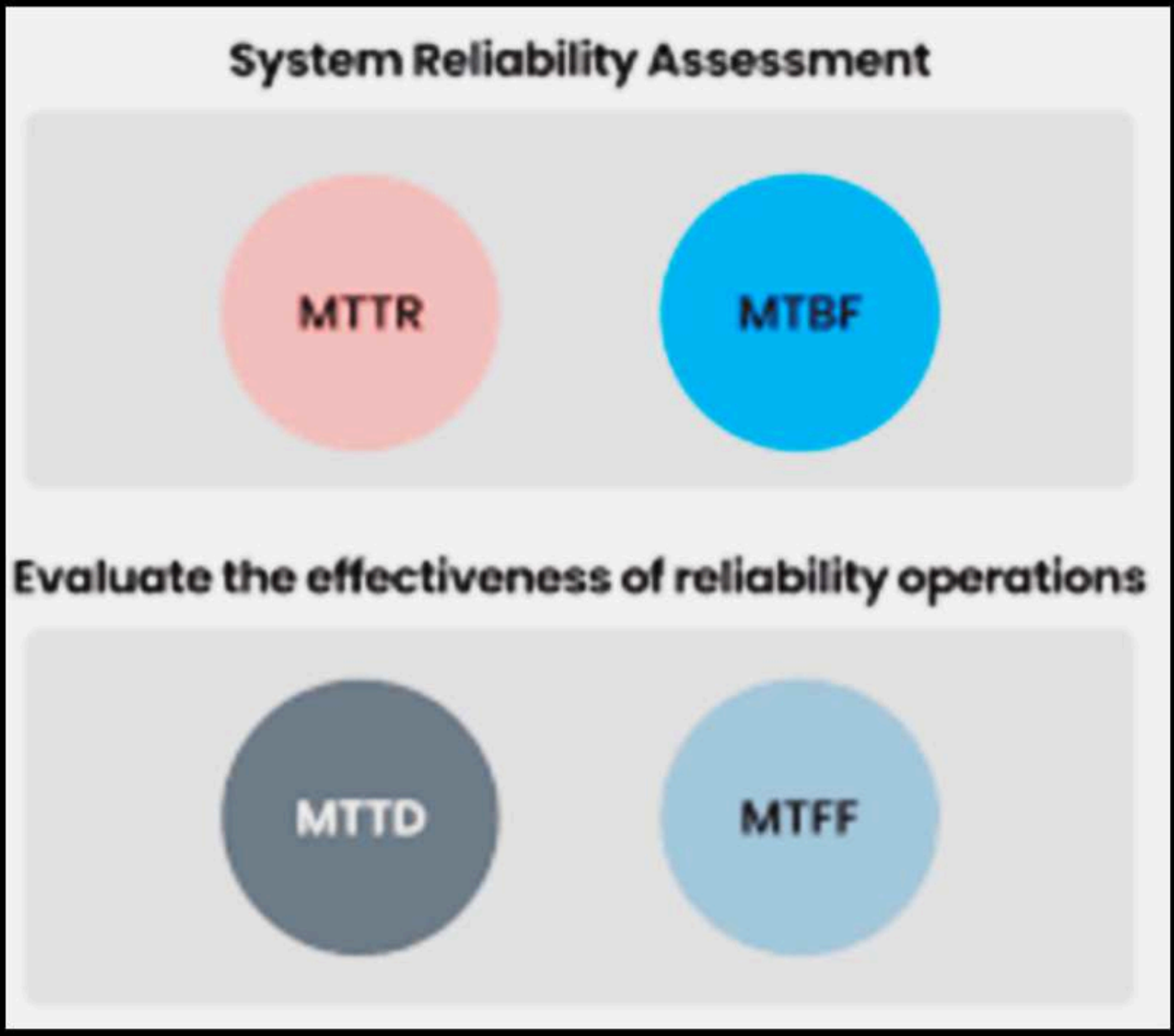
$$MTBF = \frac{\text{Period of component work time}}{\text{Number of failures per a period}}$$

Several methods can be employed to implement MTTD tracking within a DevSecOps environment:

1. **Log Analysis and Monitoring:** Implement robust logging and monitoring systems that capture security-related events and alerts from various sources, including applications, infrastructure, security tools, and network devices. Analyze these logs to identify patterns, anomalies, and potential security incidents. The ELK stack (Elasticsearch, Logstash, and Kibana) or Prometheus and Grafana are examples of open source tools that can be used for this purpose.
2. **Security Information and Event Management (SIEM):** Utilize a SIEM system to collect, aggregate, and correlate security data from multiple sources, providing a centralized platform for threat detection and analysis. SIEMs can help automate the process of identifying security incidents and reducing detection time.
3. **Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS):** Deploy IDSs and IPSs to monitor network traffic for suspicious activity and security threats. These systems can generate alerts and even take automated actions to block or mitigate potential attacks, contributing to faster detection.
4. **Threat Intelligence Feeds:** Integrate threat intelligence feeds into your security monitoring systems to gain insights into emerging threats and attack patterns. This proactive approach can help you identify and respond to security incidents more quickly.

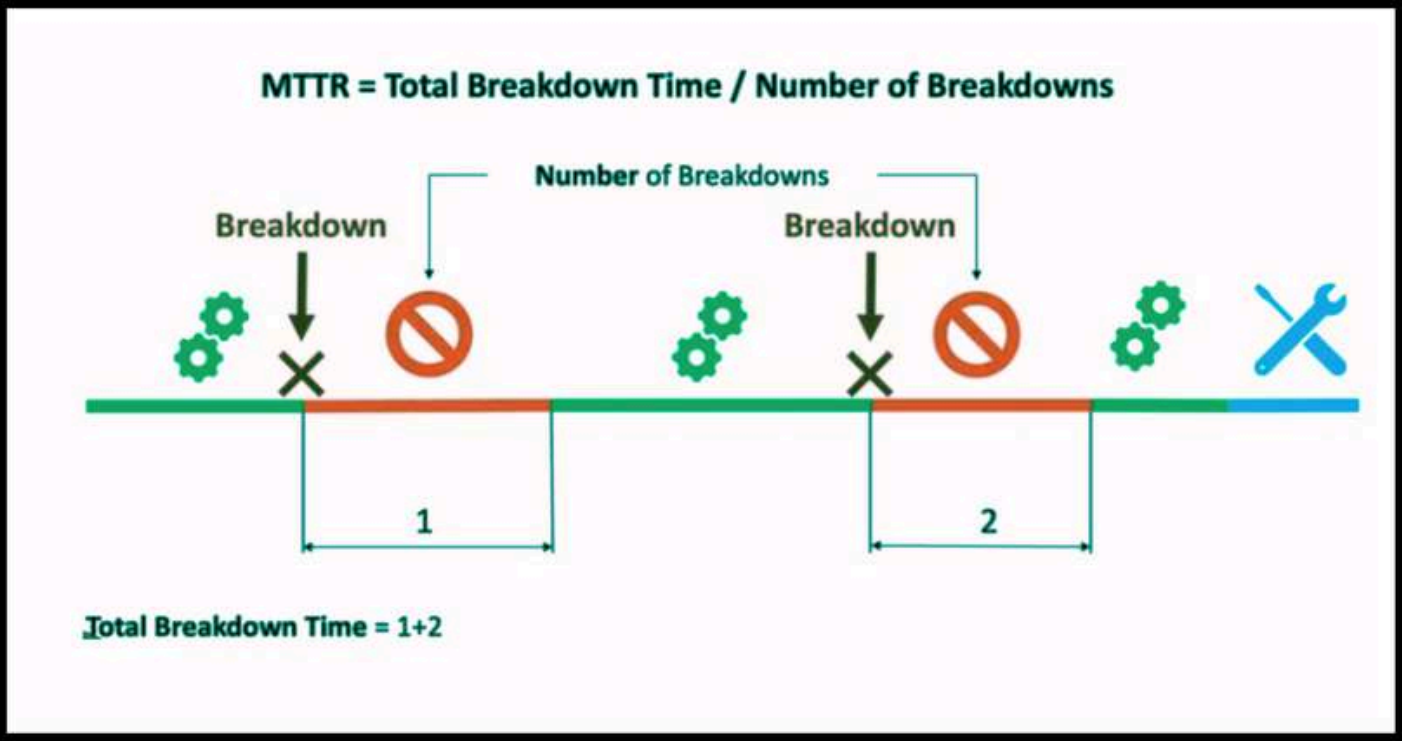


Mean Time to Remediate (MTTR)



Mean Time to Remediate (MTTR), a crucial DevSecOps KPI, measures the average time it takes to fix or resolve a security issue once it's been detected. Sources emphasize the importance of MTTR in understanding the effectiveness of security incident response and remediation efforts within the DevSecOps framework.

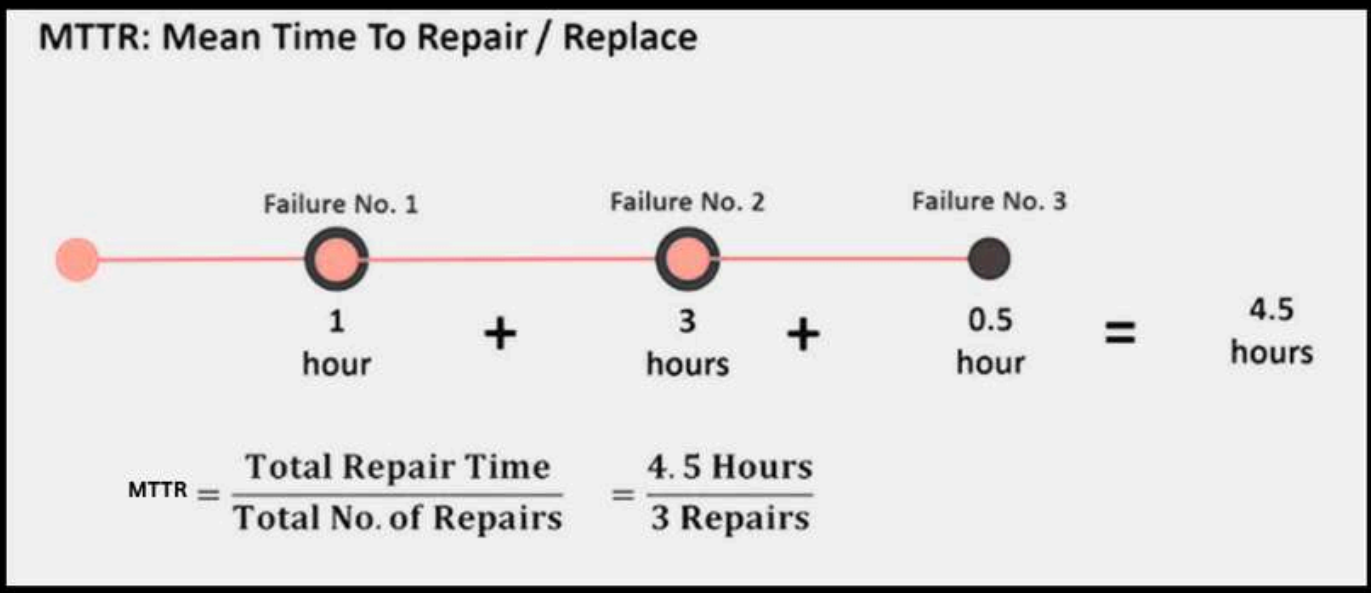
Importance of MTTR



MTTR is crucial for several reasons:

- **Minimize System Downtime and Service Disruptions:** A lower MTTR means that security issues are resolved faster, leading to reduced downtime for applications and services. This is particularly crucial for organizations that rely heavily on their digital infrastructure to deliver value to customers.
- **Reduce the Window of Exposure:** The longer a security vulnerability remains unpatched, the greater the risk of it being exploited by attackers. A low MTTR minimizes the window of exposure, reducing the likelihood of successful attacks.
- **Improve Overall Security Posture:** By tracking MTTR, organizations gain insights into the efficiency of their security incident response processes, allowing them to identify areas for improvement and streamline remediation efforts. A consistently low MTTR indicates a mature and effective security program.

Calculating MTTR



MTTR is typically calculated by dividing the total time spent on remediation by the number of incidents. The formula is:

MTTR = Total Remediation Time ÷ Number of Incidents

For example, if a team spends 10 hours on remediation for 2 incidents, the MTTR would be:

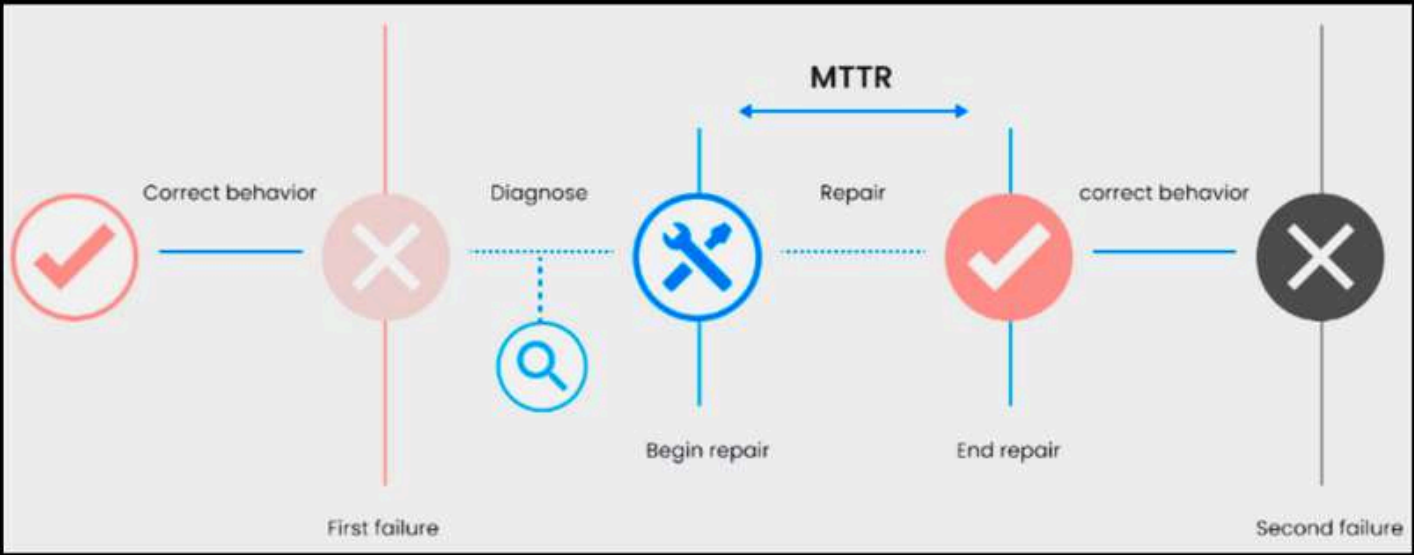
MTTR = 10 hours ÷ 2 incidents = 5 hours

Best Practices for Reducing MTTR

To reduce MTTR, IT teams can follow these best practices:

- Implement a robust incident management process that includes clear roles, responsibilities, and communication channels.
- Use automation tools to streamline incident response and remediation.
- Provide regular training and coaching to incident response teams.
- Continuously monitor and analyze incident data to identify areas for improvement.

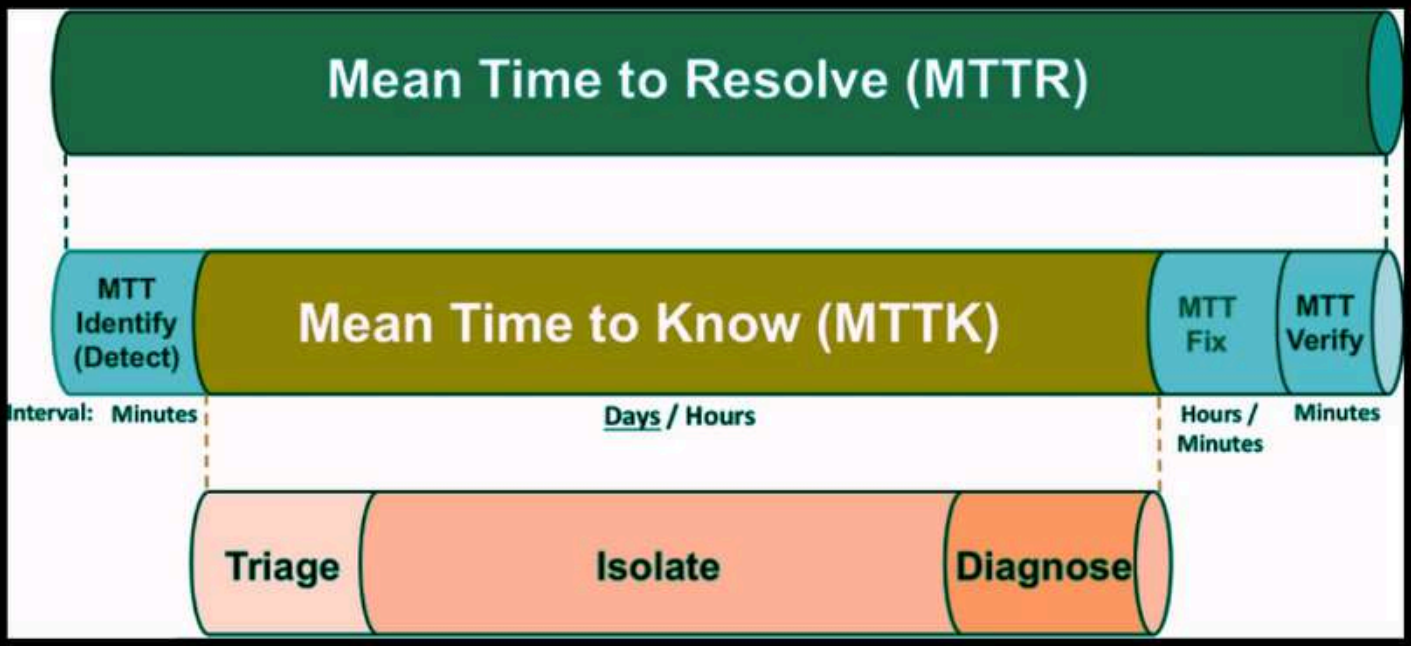
Implementing MTTR Tracking



Sources suggest various methods for effectively tracking and measuring MTTR:

1. **Track Incident Resolution Time:** Record the time it takes to resolve security incidents, starting from the moment the issue is detected to the point when it's fully remediated and verified. Calculate the average resolution time over a specific period to determine the MTTR.
2. **Utilize Incident Management Systems:** Implement incident management systems that track the lifecycle of security incidents, including detection time, response actions, remediation steps, and resolution time. These systems can provide valuable data for calculating MTTR and analyzing trends.
3. **Leverage Automation:** Automate tasks related to incident response and remediation, such as vulnerability scanning, patching, and configuration management. Automation can significantly reduce the time it takes to resolve security issues, leading to a lower MTTR.
4. **Continuous Monitoring and Alerting:** Implement robust monitoring and alerting systems to detect security issues early on. Faster detection allows for quicker response and remediation, contributing to a lower MTTR.

Factors Influencing MTTR



While the sources don't explicitly mention all factors that influence MTTR, based on the information provided and the nature of DevSecOps, several factors can be inferred:

- **Complexity of the Security Issue:** The time to remediate can vary significantly depending on the complexity of the vulnerability or security incident. Simple issues might be resolved quickly, while complex issues might require extensive investigation, code changes, and testing.
- **Availability of Skilled Resources:** Having skilled security professionals, developers, and operations personnel available to respond to and remediate security issues is crucial for reducing MTTR.
- **Effectiveness of Incident Response Processes:** Well-defined and efficient incident response processes, including clear communication channels, escalation procedures, and documented remediation steps, contribute to a lower MTTR.
- **Level of Automation:** Organizations with a higher degree of automation in their security processes tend to have lower MTTRs, as automation streamlines remediation tasks and reduces manual effort.

Security Test Coverage (STC)

/

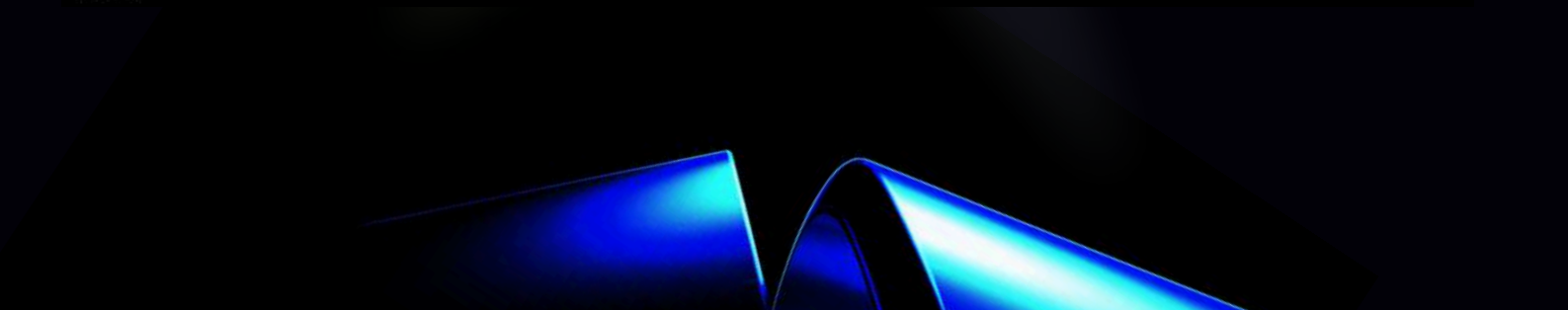
98.36% Statements 1855/1898 **93.92%** Branches 726/773 **100%** Functions 266/266 **99.73%** Lines 1816/1821 **37 statements, 21 branches** ignored

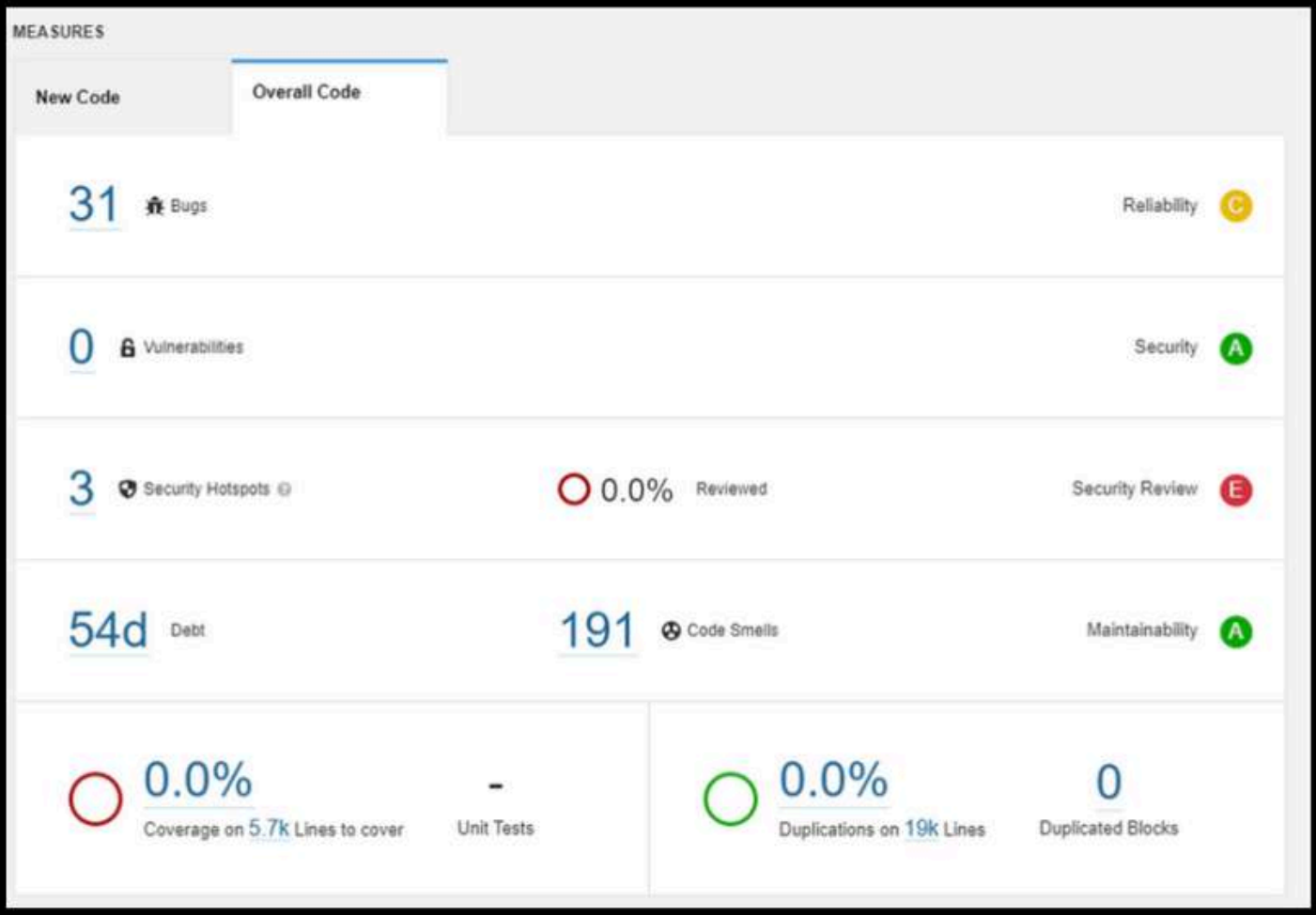
File		Statements		Branches		Functions		Lines	
express/		100%	1/1	100%	0/0	100%	0/0	100%	1/1
express/examples/auth/		93.75%	75/80	80.77%	21/26	100%	17/17	100%	71/71
express/examples/content-negotiation/		100%	32/32	100%	2/2	100%	13/13	100%	32/32
express/examples/cookie-sessions/		100%	12/12	100%	4/4	100%	1/1	100%	12/12
express/examples/cookies/		95.83%	23/24	87.5%	7/8	100%	3/3	100%	22/22
express/examples/downloads/		94.12%	16/17	87.5%	7/8	100%	3/3	100%	15/15
express/examples/ejs/		100%	11/11	100%	2/2	100%	1/1	100%	11/11
express/examples/error-pages/		97.14%	34/35	83.33%	10/12	100%	6/6	100%	34/34
express/examples/error/		90%	18/20	66.67%	4/6	100%	4/4	100%	18/18
express/examples/markdown/		95.24%	20/21	66.67%	4/6	100%	5/5	100%	20/20
express/examples/multi-router/		100%	9/9	100%	2/2	100%	1/1	100%	9/9
express/examples/multi-router/controllers/		100%	14/14	100%	0/0	100%	4/4	100%	14/14
express/examples/mvc/		95.45%	42/44	80%	8/10	100%	4/4	100%	42/42
express/examples/mvc/controllers/main/		100%	2/2	100%	0/0	100%	1/1	100%	2/2
express/examples/mvc/controllers/pet/		94.12%	16/17	50%	1/2	100%	4/4	100%	16/16
express/examples/mvc/controllers/user-pet/		92.86%	13/14	50%	1/2	100%	1/1	100%	13/13
express/examples/mvc/controllers/user/		100%	21/21	100%	4/4	100%	6/6	100%	19/19
express/examples/mvc/lib/		97.96%	48/49	80%	20/25	100%	2/2	100%	46/46

Security Test Coverage (STC) is a key metric in DevSecOps that measures the **extent to which an application's codebase has undergone security testing**. It helps identify areas that may need additional attention from a security standpoint. STC is like stargazing, sometimes you spot a comet, other times, a black hole.

Code coverage is a measure of how much of the code is executed during a test run. It's an essential metric in web security, as it helps determine whether a test is useful or not.

Code coverage is crucial in web security because it ensures that the test is not just finding vulnerabilities, but also executing the code that is being tested. Without code coverage, it's difficult to determine whether the test is effective or not.



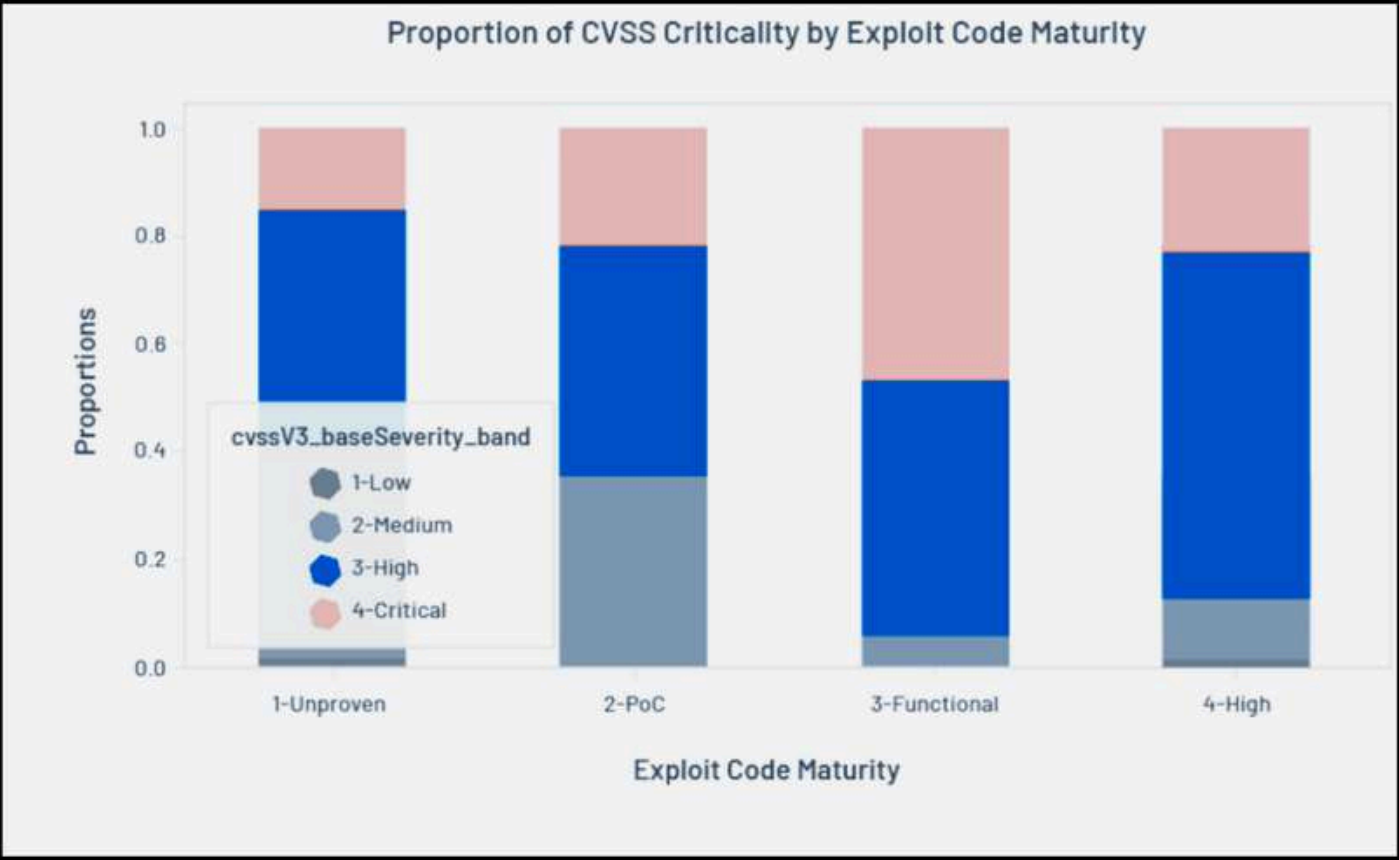


A high degree of STC is essential for ensuring the security of applications in a DevSecOps environment.

Some of the benefits of achieving high STC include:

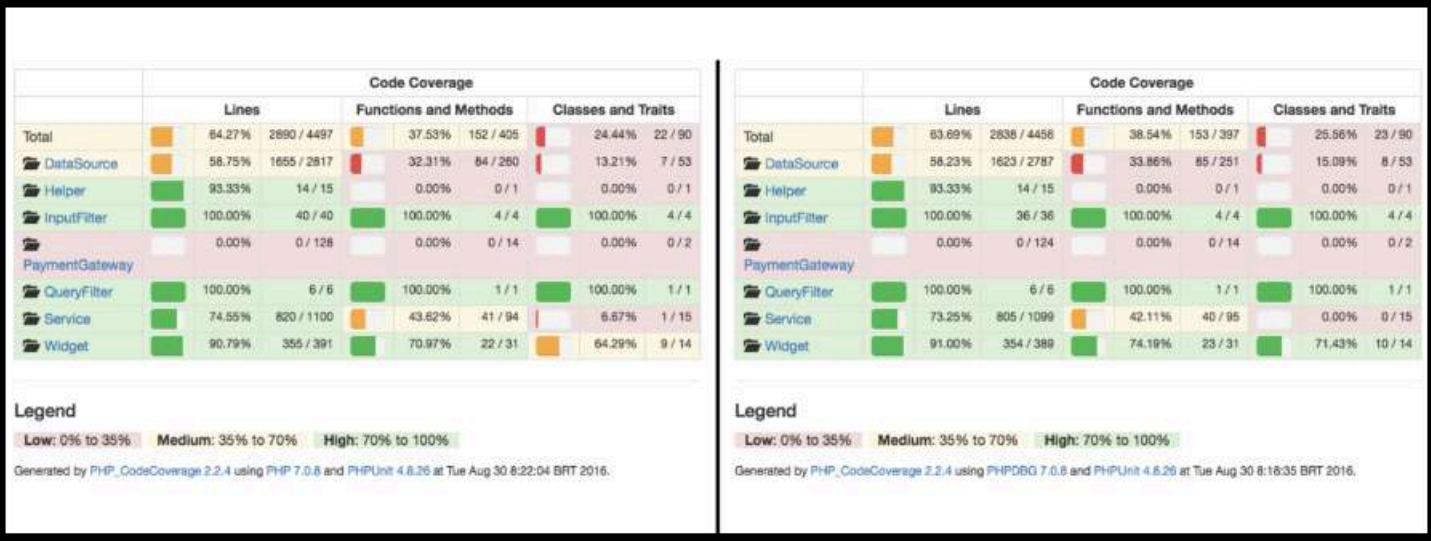
- **Early Detection of Vulnerabilities:** By conducting security testing throughout the development process, organizations can identify and remediate vulnerabilities before they make it into production.
- **Reduced Risk of Security Breaches:** A comprehensive security testing program helps reduce the overall risk of security breaches by identifying and mitigating vulnerabilities early on.
- **Improved Compliance with Security Standards:** Many industry regulations and security standards require organizations to conduct specific types of security testing. Achieving high STC helps organizations comply with these requirements.

Use Case: How Code Coverage Helped Us Find Critical Vulnerabilities



In a use case, code coverage helped find 3 critical vulnerabilities in a web application. The test was able to generate 9 bug findings, but the code coverage was only 16%. After logging in and rerunning the test, 22 new bugs were found, including 3 security-critical SQL injections.

Interpreting the Results: Why is Feedback on Code Coverage so Important?



Code Coverage

	Lines		Functions and Methods		Classes and Traits	
Total	64.27%	2890 / 4497	37.53%	152 / 405	24.44%	22 / 90
DataSource	58.75%	1655 / 2817	32.31%	84 / 260	13.21%	7 / 53
Helper	93.33%	14 / 15	0.00%	0 / 1	0.00%	0 / 1
InputFilter	100.00%	40 / 40	100.00%	4 / 4	100.00%	4 / 4
PaymentGateway	0.00%	0 / 128	0.00%	0 / 14	0.00%	0 / 2
QueryFilter	100.00%	6 / 6	100.00%	1 / 1	100.00%	1 / 1
Service	74.55%	820 / 1100	43.62%	41 / 94	6.67%	1 / 15
Widget	90.79%	355 / 391	70.97%	22 / 31	64.29%	9 / 14

Legend

Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

Generated by PHP_CodeCoverage 2.2.4 using PHP 7.0.8 and PHPUnit 4.8.26 at Tue Aug 30 8:22:04 BRT 2016.

Code Coverage

	Lines		Functions and Methods		Classes and Traits	
Total	63.69%	2838 / 4458	38.54%	153 / 397	25.56%	23 / 90
DataSource	58.23%	1623 / 2787	33.86%	85 / 251	15.09%	8 / 53
Helper	93.33%	14 / 15	0.00%	0 / 1	0.00%	0 / 1
InputFilter	100.00%	36 / 36	100.00%	4 / 4	100.00%	4 / 4
PaymentGateway	0.00%	0 / 124	0.00%	0 / 14	0.00%	0 / 2
QueryFilter	100.00%	6 / 6	100.00%	1 / 1	100.00%	1 / 1
Service	73.25%	805 / 1099	42.11%	40 / 95	0.00%	0 / 15
Widget	91.00%	354 / 389	74.19%	23 / 31	71.43%	10 / 14

Legend

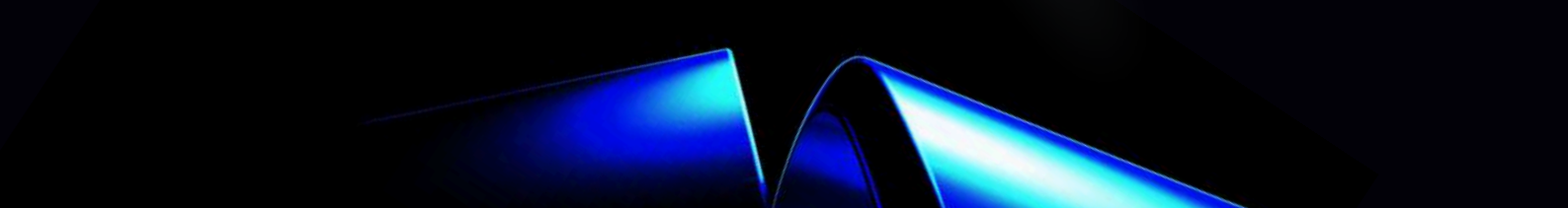
Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

Generated by PHP_CodeCoverage 2.2.4 using PHPDBG 7.0.8 and PHPUnit 4.8.26 at Tue Aug 30 8:18:35 BRT 2016.

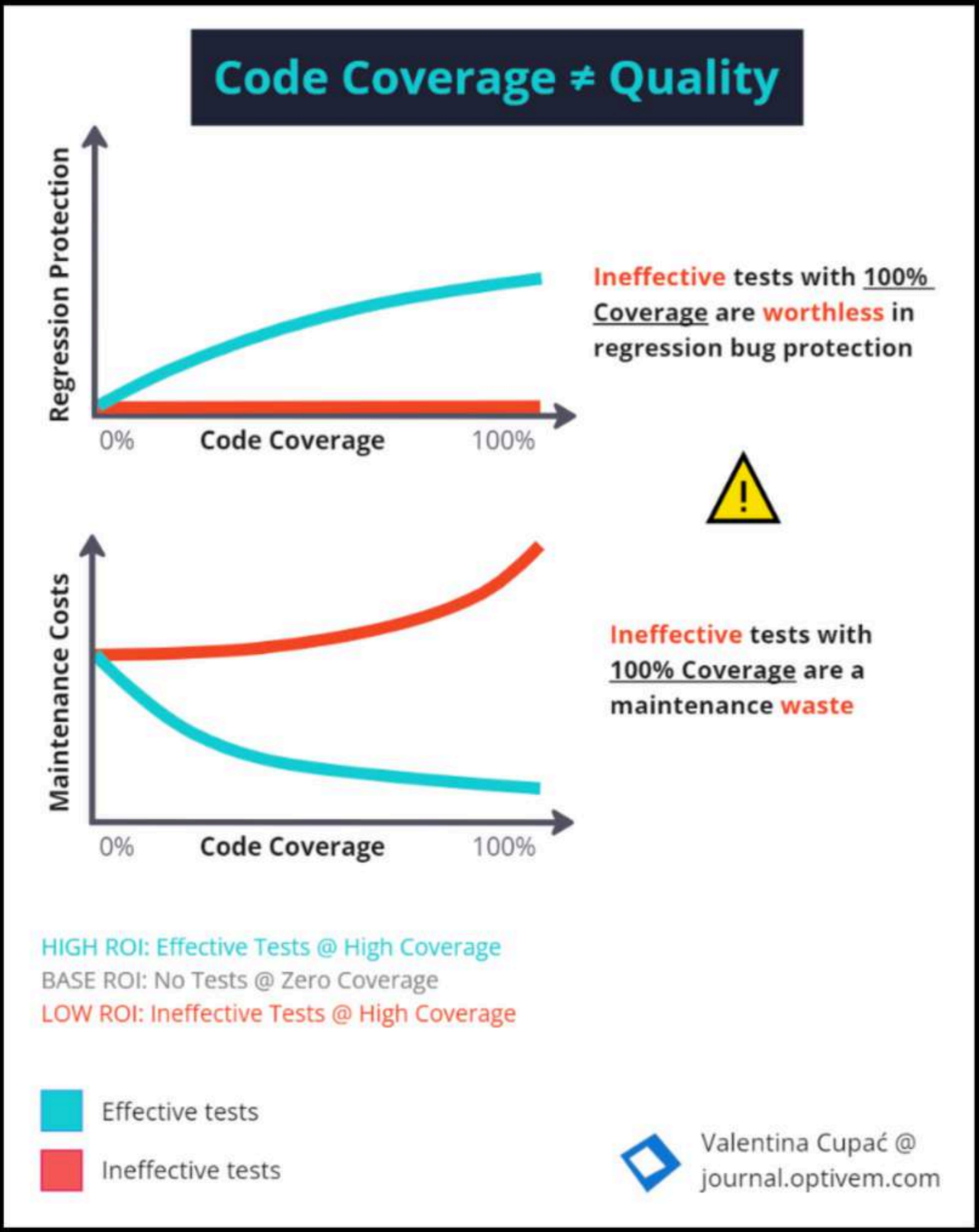
Feedback on code coverage is important because it helps identify areas of the code that are not being tested. This can include missing permissions, user groups with different access levels, and other road blockers that can lead to low coverage.

Coverage-Guided vs Black-Box Testing

TESTING TYPE	BLACK BOX TESTING	GRAY BOX TESTING	WHITE BOX TESTING
Granularity level	Low level	Medium level	High level
Known as	Opaque-box testing, closed-box testing, input-output testing	Traslucent box testing	Glass-box testing, clear-box testing, logic-based and code-based testing
Participants	Done by end-users and also by tester and developers	Done by end-users (called user acceptance testing) and also by testers and developers	Done by testers and developers
Design techniques	Decision table testing, all pairs testing, equivalence partitioning, error guessing	Matrix testing, regression testing, pattern testing, orthogonal array testing	Control flow testing, data flow testing, branch testing
Insight into internal working	External focus only	External with some internal focus	Complete internal focus
Insight into internal working	External focus only	External with some internal focus	Complete internal focus
Discovery of hidden errors	Challenging	Possible at the user level	Easier due to in-depth knowledge
Time consumption	Less exhaustive	Partially exhaustive	Most exhaustive

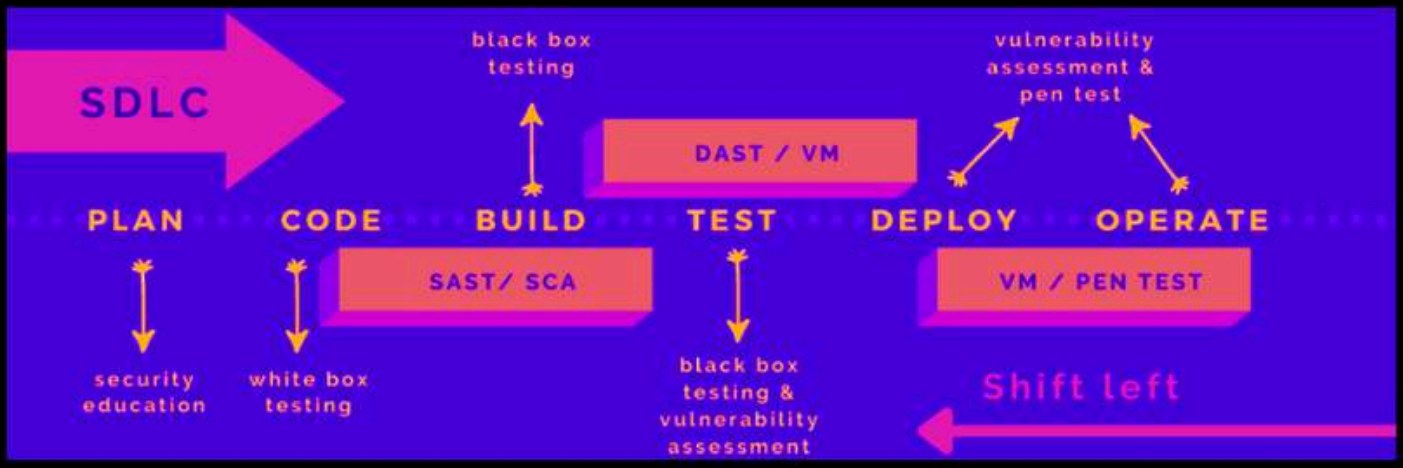


How to Measure and Report Code Coverage



Code coverage can be measured and reported using tools such as CI Fuzz. This platform uses modern fuzz testing approaches to automate security testing for web applications and continuously measures code coverage. It also comes with detailed reporting and dashboards that allow developers to monitor the performance of fuzz tests in real-time.

Implementing STC



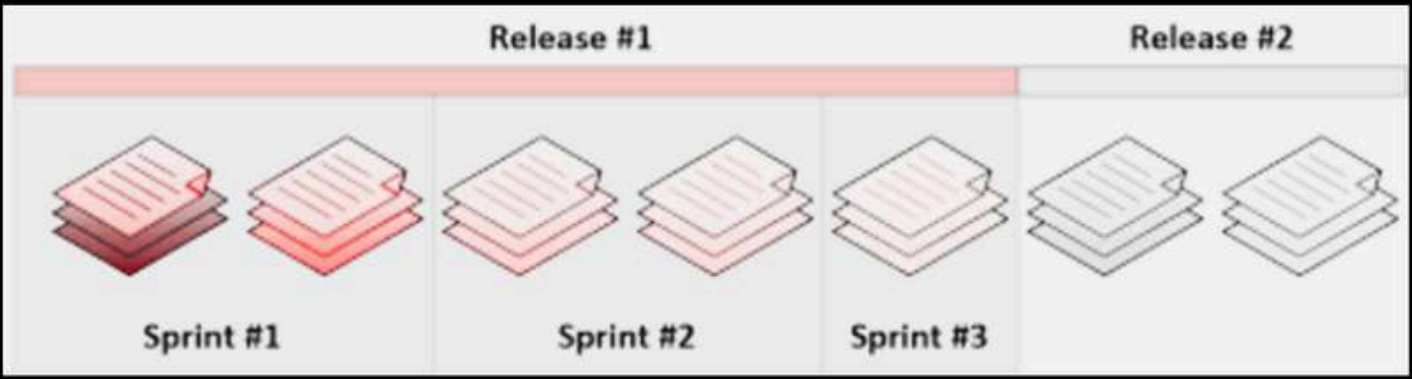
Although the sources do not provide specific methods for implementing STC, they do discuss a variety of security testing approaches and tools. These can be used to establish a comprehensive security testing program that provides a high degree of STC.

- **Static Application Security Testing (SAST)** analyzes the source code for potential security vulnerabilities. SAST is like having a grammar checker for your code.
- **Dynamic Application Security Testing (DAST)** identifies vulnerabilities in a running application. DAST is like a secret agent spying on the application, but for good reasons.
- **Software Composition Analysis (SCA)** examines the open source components used in the code, such as third-party libraries and dependencies, for known vulnerabilities. SCA is like a dedicated quality assurance team that covers security and compliance for your software's ingredients.
- **Infrastructure as Code (IaC) Scanning** looks for known vulnerabilities in your IaC configuration files.

Findings per Release/Sprint

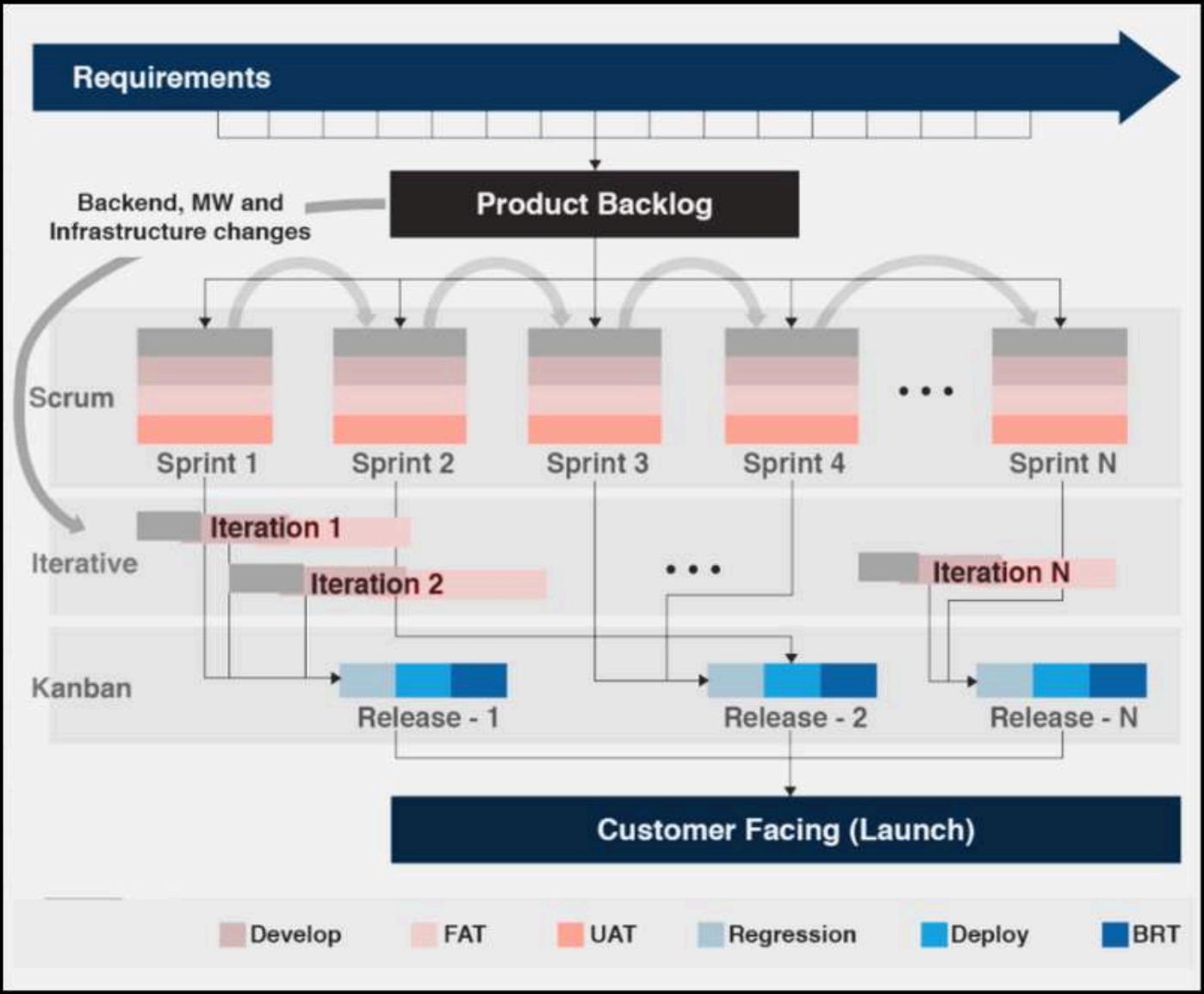
"Findings per Release/Sprint" is a vital KPI in DevSecOps that measures the average number of security issues found in each software release or sprint. Tracking this metric offers valuable insights into the effectiveness of security practices integrated into the development process and helps identify areas for improvement.

Implementing Findings per Release/Sprint Tracking



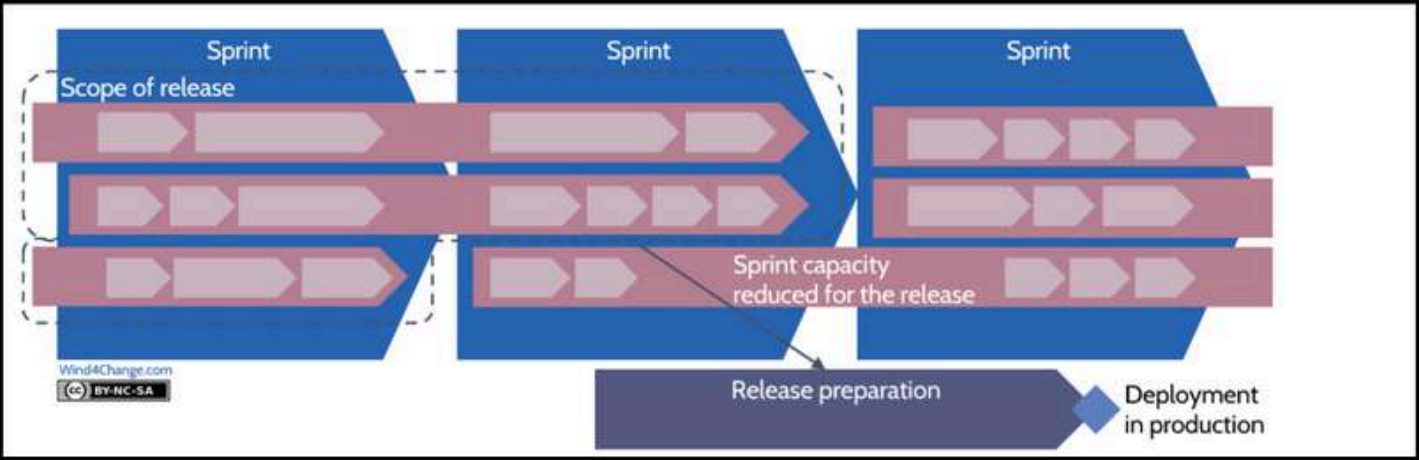
1. **Establish a Consistent Definition of a "Security Finding":** To ensure accurate measurement, it's crucial to define what constitutes a security finding. This could include vulnerabilities, misconfigurations, deviations from security policies, and other security-related issues.
2. **Integrate Security Testing into the Development Pipeline:** Regularly conduct security testing activities, such as SAST, DAST, SCA, and penetration testing, as part of the development workflow. This allows for continuous identification of security findings throughout the process.
3. **Track Security Findings in a Centralized System:** Use issue tracking systems or specialized DevSecOps platforms to log and manage security findings. This enables efficient tracking, prioritization, and remediation of identified issues.
4. **Categorize and Prioritize Findings:** Classify security findings based on severity level (e.g., critical, high, medium, low) to prioritize remediation efforts. This ensures that the most critical issues are addressed first.

Benefits of Tracking Findings per Release/Sprint



- **Identify Trends and Patterns:** Tracking this metric over time reveals trends and patterns in the types and frequency of security findings. This data helps pinpoint recurring issues and areas that require additional attention or training.
- **Measure the Effectiveness of Security Practices:** A decreasing trend in findings per release/sprint suggests that security practices are becoming more effective in preventing and detecting issues early on. Conversely, an increasing trend might indicate the need to improve security measures.
- **Proactive Risk Management:** By understanding the common types of security findings, organizations can proactively address potential risks and implement preventative measures to reduce the likelihood of similar issues occurring in future releases.

Findings per Release/Sprint and Continuous Improvement



Tracking findings per release/sprint is not just about measuring numbers but about driving continuous improvement in the DevSecOps process. Analyzing the data allows organizations to:

- **Refine Security Testing Strategies:** Adjust testing approaches based on the types of findings observed. For instance, if SCA consistently reveals vulnerabilities in specific third-party libraries, the organization might consider using alternative libraries or implementing stricter dependency management processes.
- **Enhance Security Training:** Tailor security training programs to address the specific weaknesses identified through findings. For example, if findings often relate to secure coding practices, developers can benefit from targeted training on secure coding techniques.
- **Foster a Culture of Security:** Regularly communicating findings per release/sprint data and involving developers in the analysis and remediation process helps embed security considerations into the development culture.

Considerations for Findings per Release/Sprint

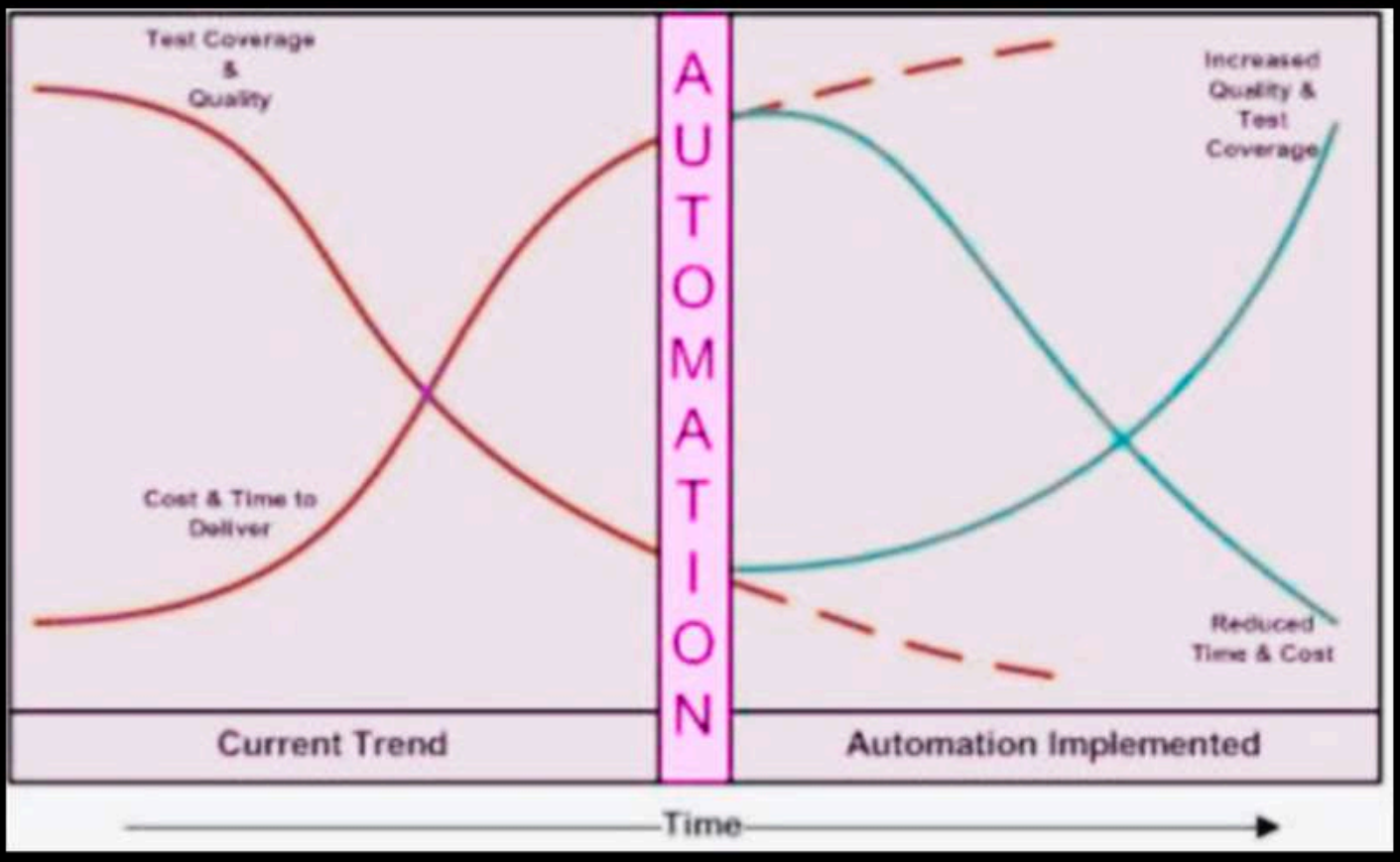
- **Contextual Interpretation:** The metric should be interpreted in the context of the application's complexity, size, and risk profile. A complex application might naturally have more findings than a simple one.
- **Focus on Trends, Not Absolute Numbers:** Instead of fixating on absolute numbers, prioritize analyzing trends over time to understand whether security practices are improving or require adjustments.

By continuously monitoring and analyzing this KPI, organizations can ensure that security remains an integral part of the development lifecycle and that applications are released with a higher level of security assurance.

Automated Testing Coverage

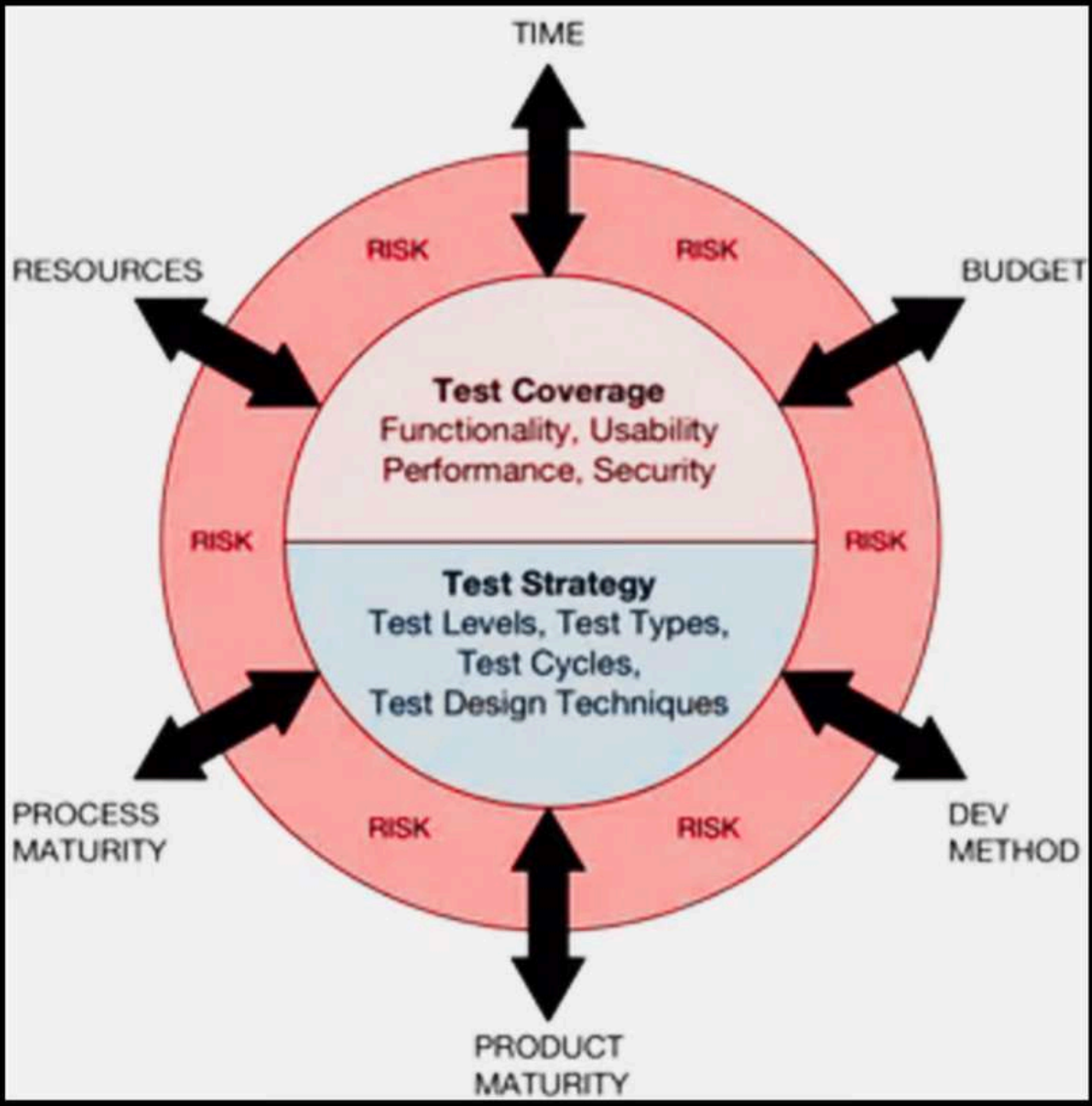
Automated Testing Coverage is a crucial DevSecOps KPI that assesses the **percentage of an application's codebase that is automatically tested for security vulnerabilities**. It acts as an indicator of the effectiveness and efficiency of security testing practices within the software development lifecycle. The higher the automated testing coverage, the more confident organizations can be in the security and resilience of their applications.

What is Test Coverage?



Test coverage is a technique used to determine whether test cases are actually covering the application code and how much code is exercised when running those test cases. It is calculated as a percentage of the total code covered by the test cases.

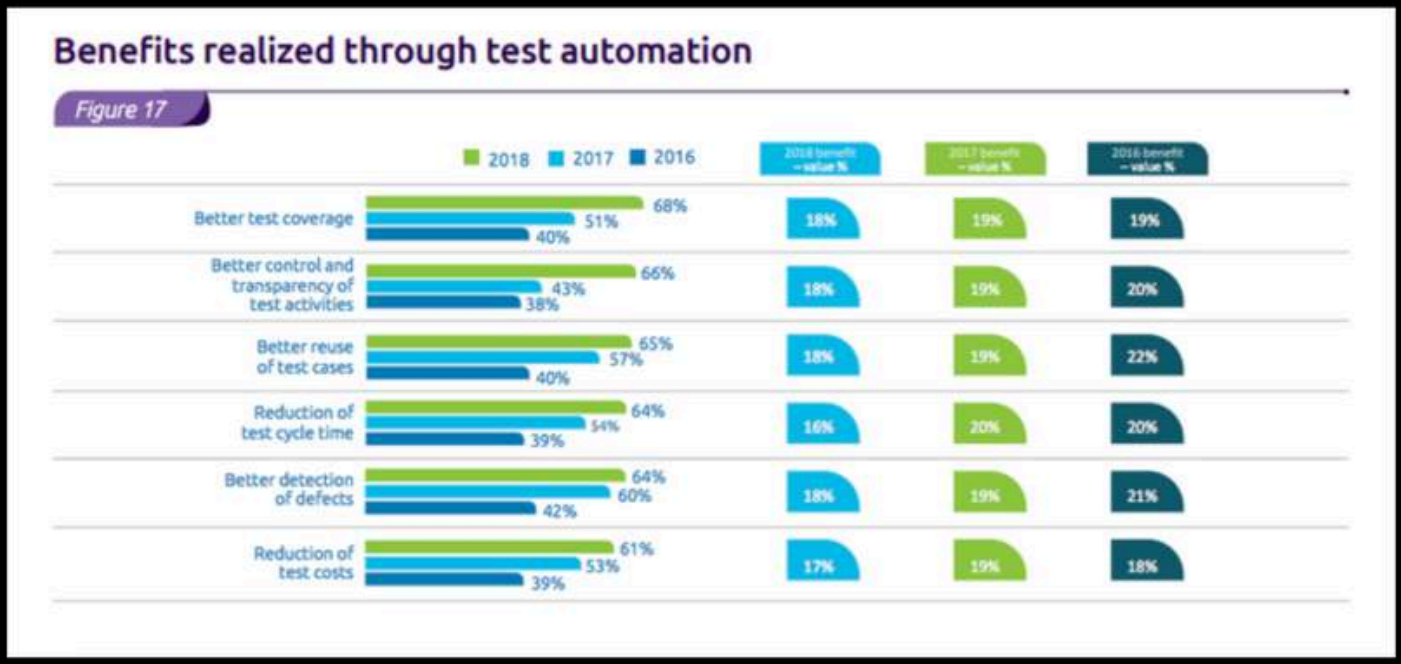
Test Coverage Techniques



Some popular test coverage techniques include:

- Product coverage
- Risk coverage
- Requirements coverage
- Compatibility coverage

Implementing Automated Testing Coverage



Implementing a robust automated testing strategy requires careful planning and execution. Here are key steps involved:

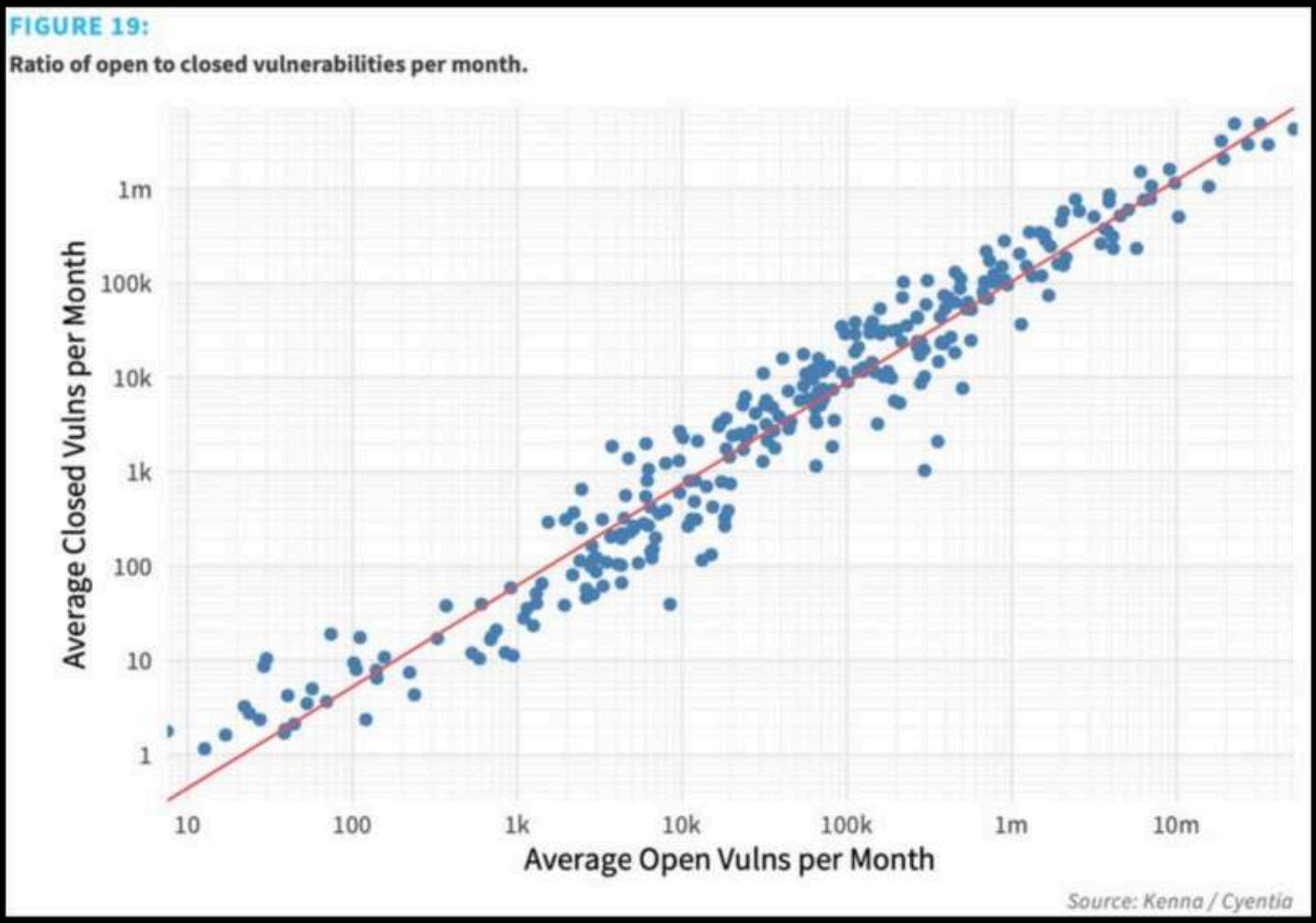
1. **Select Appropriate Testing Tools:** Choose tools that align with the application's technology stack, security requirements, and DevSecOps workflow. The sources mention several popular tools, including:

    - **SAST Tools:** Snyk, SonarQube, Brakeman, Bandit
    - **DAST Tools:** OWASP ZAP, Arachni, Nessus
    - **SCA Tools:** OWASP Dependency-Check, Snyk, Nexus Lifecycle by Sonatype
    - **IAC Scanning Tools:** Checkov, Terrascan
    - **Penetration Testing Tools:** While not explicitly mentioned, various penetration testing tools exist, and some DAST tools offer penetration testing capabilities.
2. **Integrate Testing into the CI/CD Pipeline:** Incorporate automated security testing into the CI/CD pipeline to ensure that tests are run automatically whenever code changes are made. This continuous testing approach helps catch vulnerabilities early in the development process.

Vulnerability Closure Rate

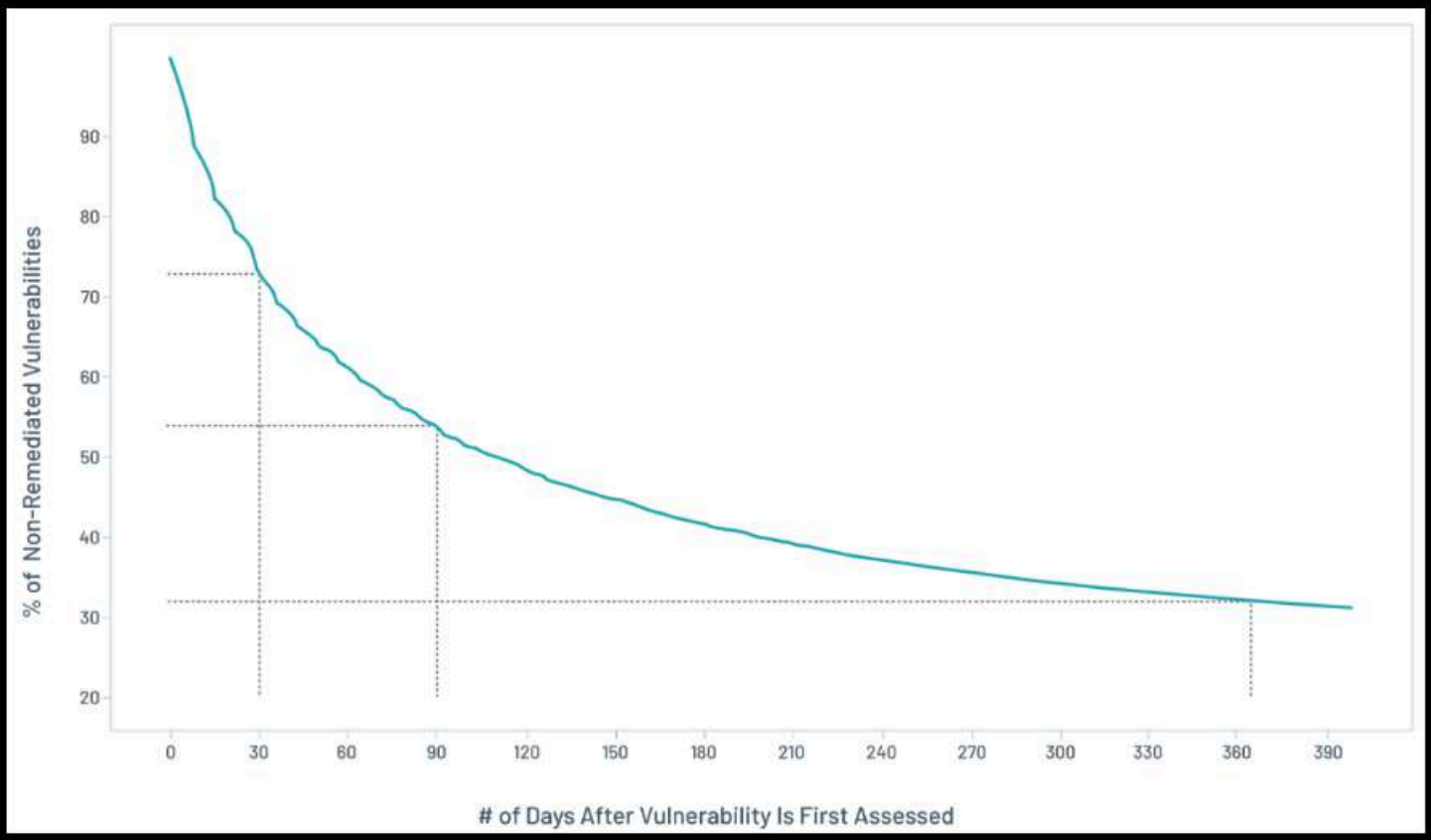
FIGURE 19:

Ratio of open to closed vulnerabilities per month.



Vulnerability Closure Rate (VCR) is a crucial KPI in DevSecOps, highlighting the effectiveness and efficiency of vulnerability management practices within the software development lifecycle. This metric **measures the speed at which identified security vulnerabilities are addressed and closed**, demonstrating an organization's commitment to proactively managing security risks and minimizing the window of exposure for potential exploits.

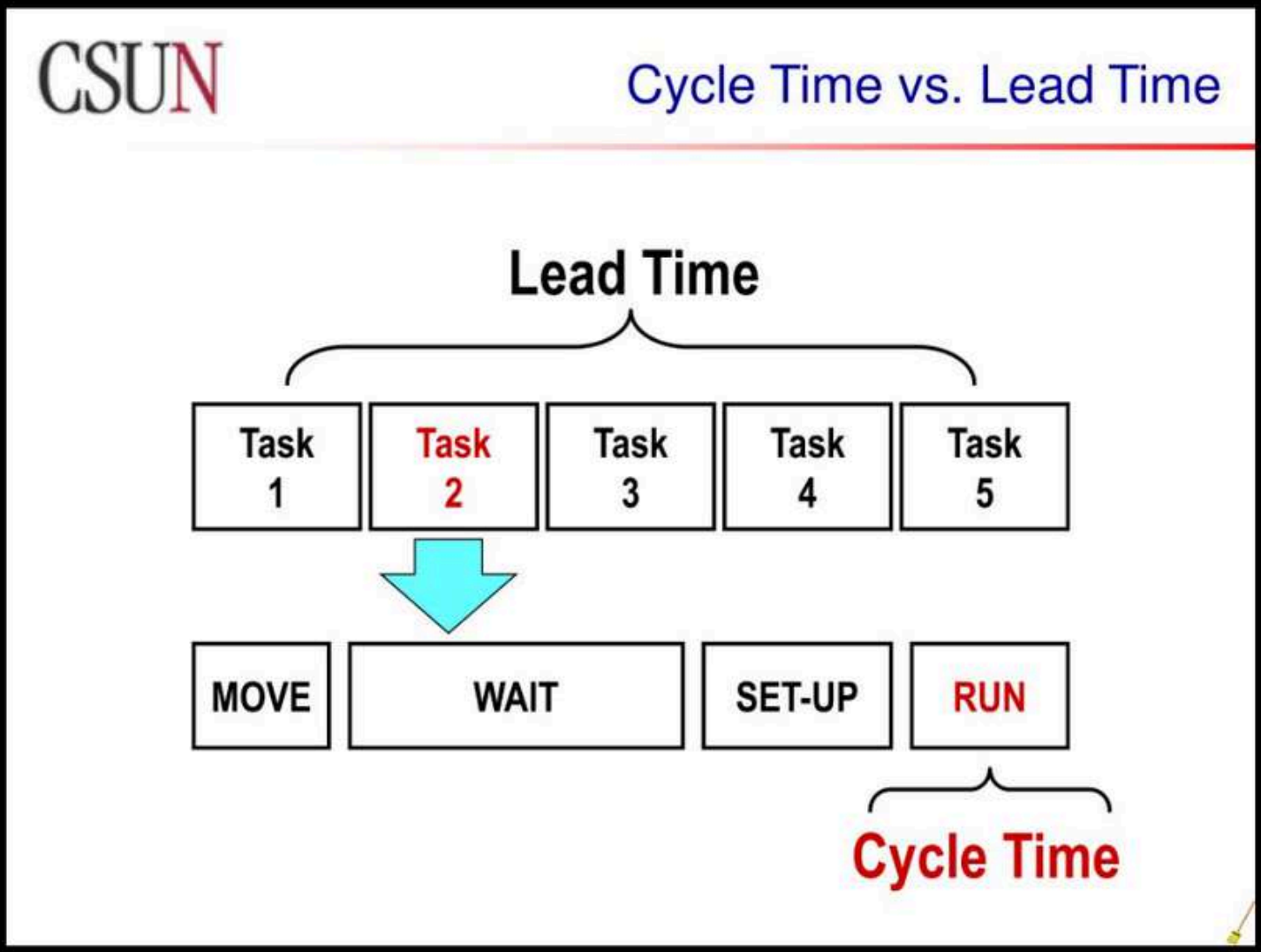
Considerations for Vulnerability Closure Rate



- **Contextual Interpretation:** VCR should be analyzed in the context of the application's complexity, size, risk profile, and industry regulations.
- **Focus on Trends, Not Absolute Numbers:** It's crucial to focus on trends in VCR over time rather than fixating on absolute numbers. A steady improvement in VCR indicates progress in vulnerability management.
- **Balance Speed and Quality:** While it's essential to address vulnerabilities quickly, organizations must also ensure that remediation efforts do not compromise the quality or functionality of the software.

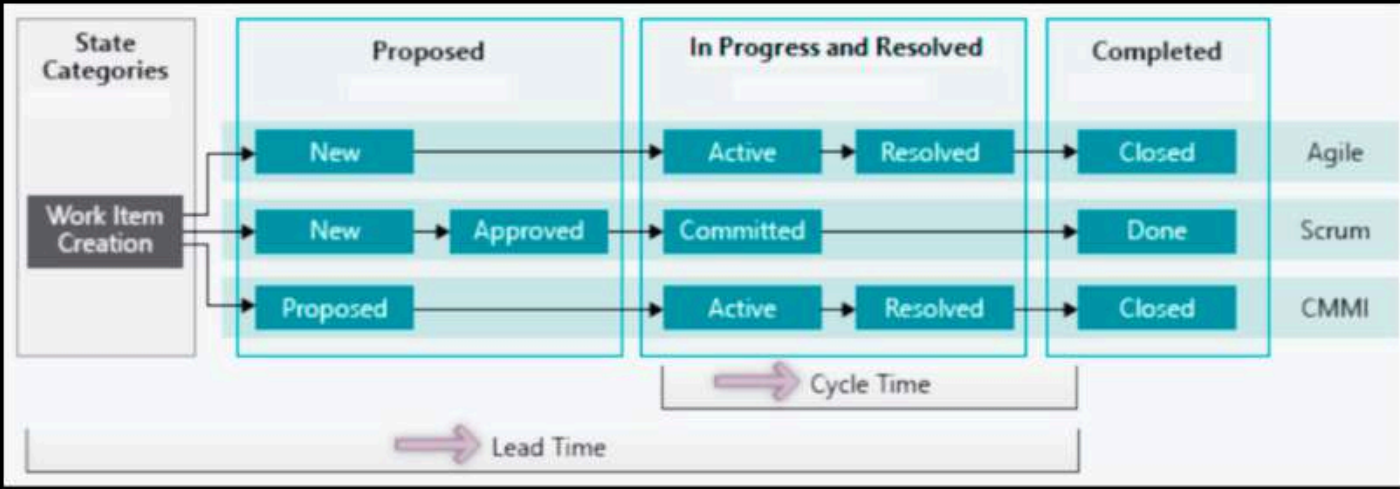
By consistently monitoring and optimizing their vulnerability closure rate, organizations can establish a robust security posture and ensure that their applications are released with a high level of assurance.

Cycle Time



Cycle time is a measure of how long it takes for a software development team to ship or fix a new software feature through the entire software development lifecycle. It measures the duration from picking up a feature sourced from customer requirements to delivering it into production—and all the steps in between, including design, development, testing, and deployment.

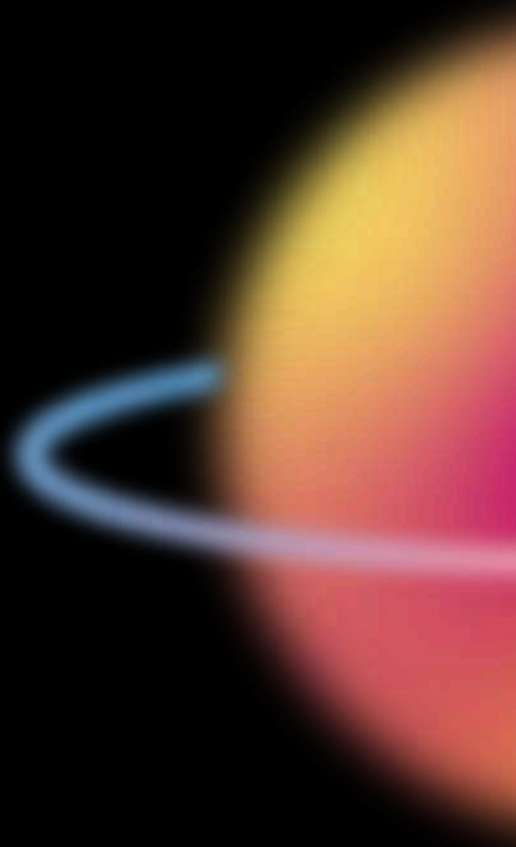
Factors Affecting Cycle Time

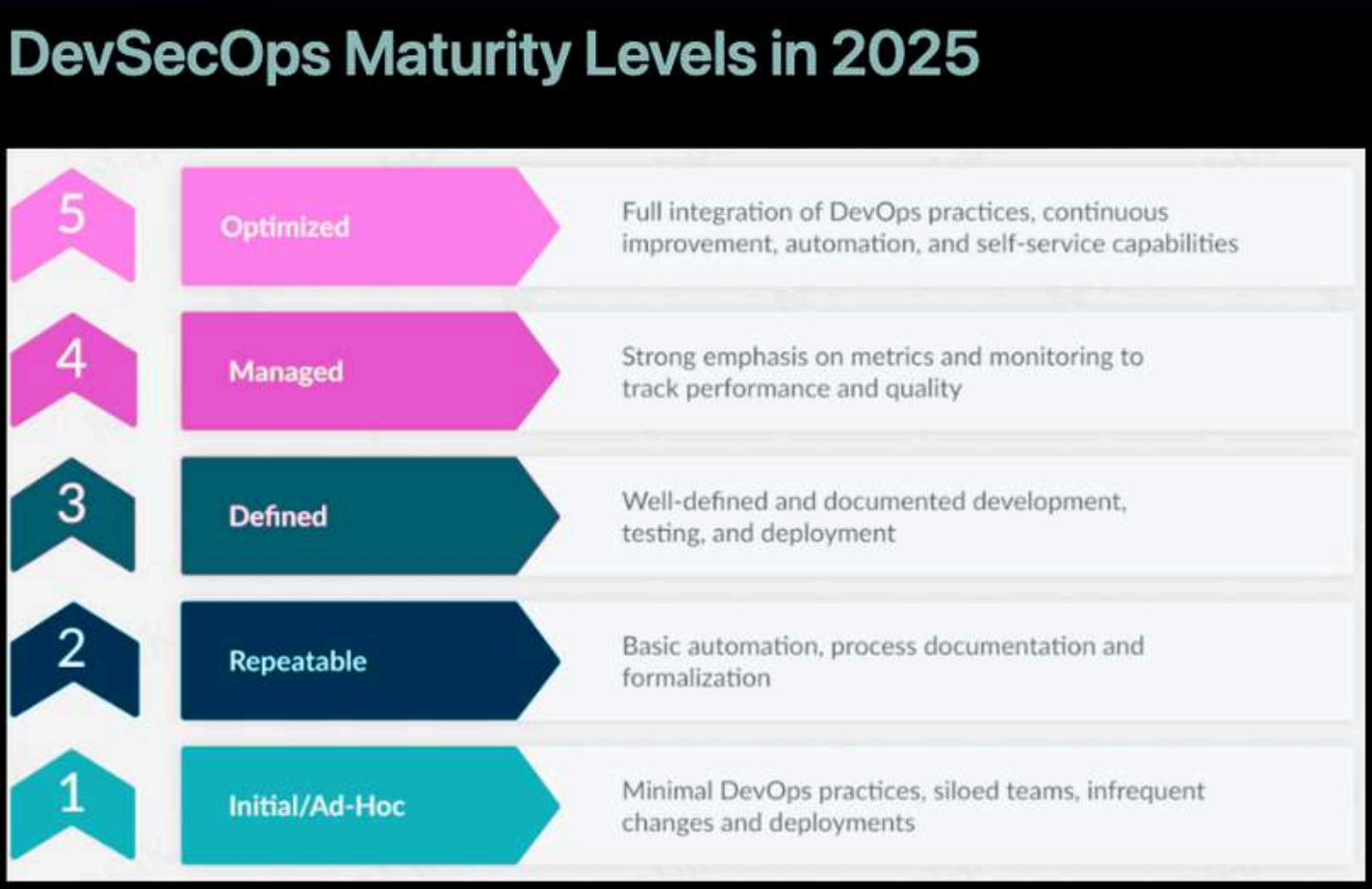


Several factors can lead to long cycle times, including:

- Overload: When the team has too much on its plate, engineers might spend too much time context-switching between tasks to get a single set of changes out the door.
- Lack of developers: A lack of developers can lead to long lead times between client requests and pickup.
- Long code review times: Code reviews are indispensable to ensuring software quality, but long review times for pull requests will kill deployment velocity.
- Tooling issues: CI/CD deployment pipeline issues, flaky tests, slow build times, and poor integration between internal developer tools can create further delays.
- Technical debt: Technical debt can also contribute to long cycle times.

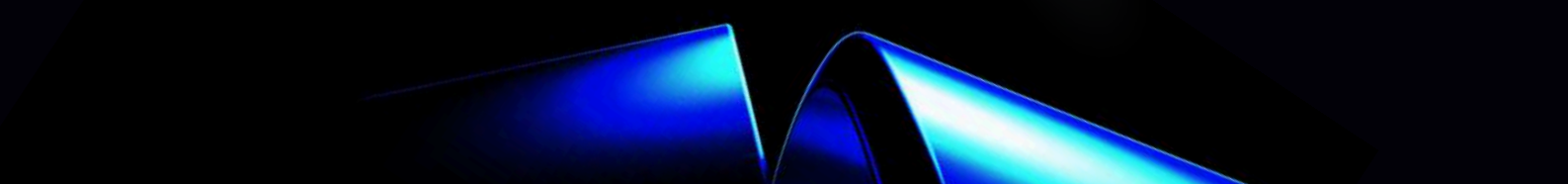
02 DevSecOps Maturity Levels in 2025





DevSecOps Maturity Levels in 2025 can help organizations evaluate and improve their integration of security practices within DevOps processes. Below is a proposed model for understanding these levels, along with references to OWASP's DSOMM and related resources:

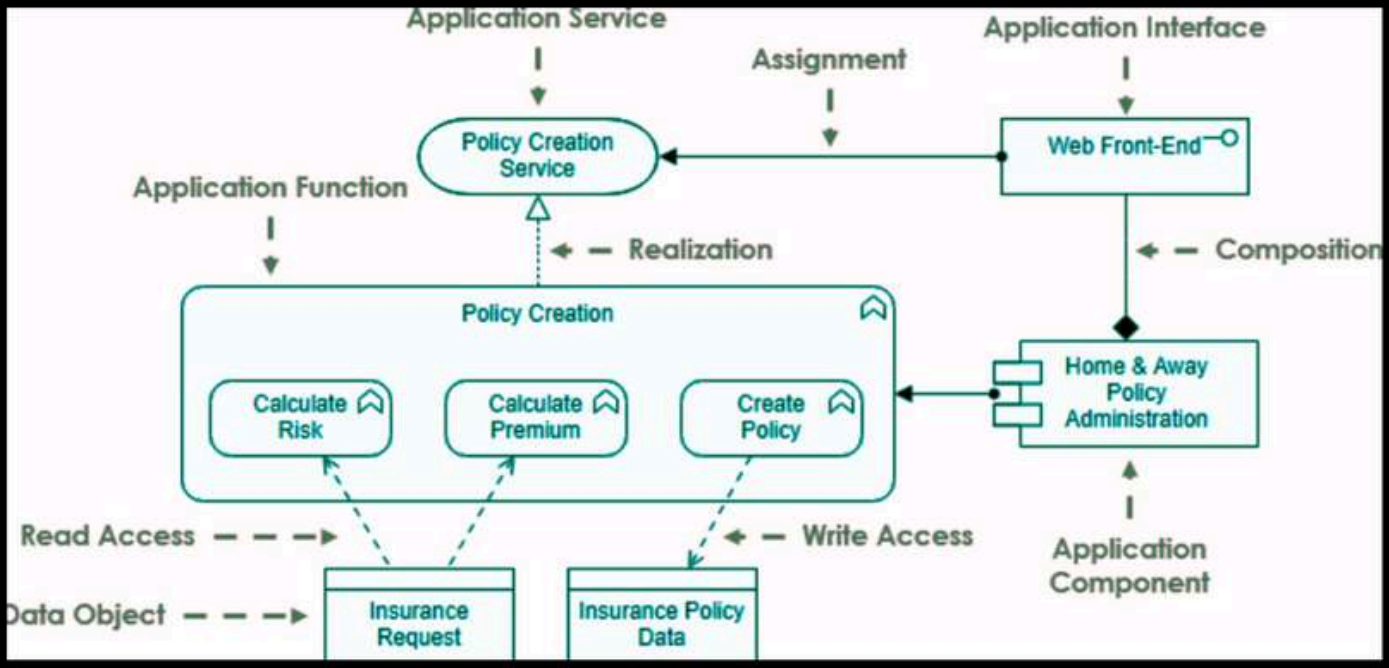
Maturity Level	Description	Key Characteristics	References
Level 1: Awareness	Basic understanding of DevSecOps principles.	- Ad hoc security checks - Minimal automation - Initial team education	DSOMM Overview
Level 2: Structured Adoption	Beginning implementation of practices.	- Documented processes - Simple automated tasks - Secure coding education begins	Usage Guidelines
Level 3: Integrated Practices	Security integrated into DevOps workflows.	- Consistent automation - Continuous monitoring - Advanced threat modeling	Mapping Levels
Level 4: Advanced Implementation	Proactive and scalable security measures.	- Full automation - Dynamic security testing - Regular training and updates	Heatmap Analysis
Level 5: Optimization and Resilience	Highest maturity with advanced adaptability.	- AI-driven threat detection - Self-healing systems - Continuous innovation	OWASP DSOMM



Each level reflects the gradual integration of security into DevOps practices, advancing from basic awareness to a state where security is a core aspect of every workflow.

For organizations aiming to assess and advance their DevSecOps maturity, OWASP's DSOMM (DevSecOps Maturity Model) provides a robust framework to align practices with modern security challenges.

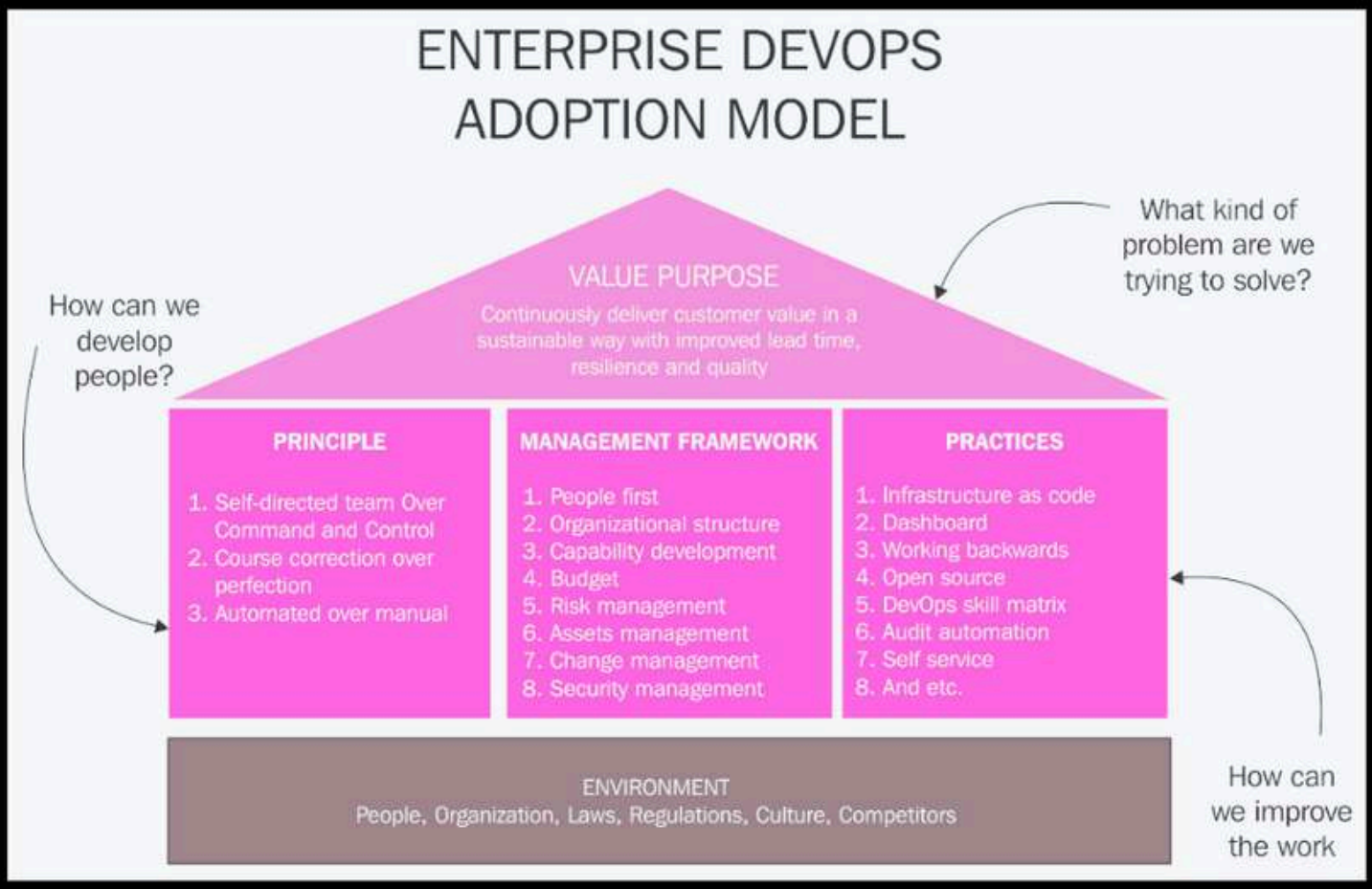
Level 1: Awareness (Foundational Understanding)



At this initial level, organizations have a basic understanding of DevSecOps principles but lack structured implementation. The focus is on building awareness and laying the groundwork for future adoption.

- **Ad-hoc security checks:** Security practices are implemented on a case-by-case basis without a consistent framework.
- **Minimal Automation:** Security tasks are largely manual, with little to no automation. Organizations at this level might be using tools from other domains, such as build and release tools (Git, Azure DevOps, Octopus Deploy, Jenkins), configuration management tools (Ansible, Puppet, Chef), test automation tools (Selenium, Worksoft, Kobiton), and deployment and monitoring tools (Nagios, Splunk, SolarWinds AppOptics). However, their application to security is limited.
- **Initial Team Education:** This involves introductory training on DevSecOps concepts, emphasizing the importance of security in the software development lifecycle.

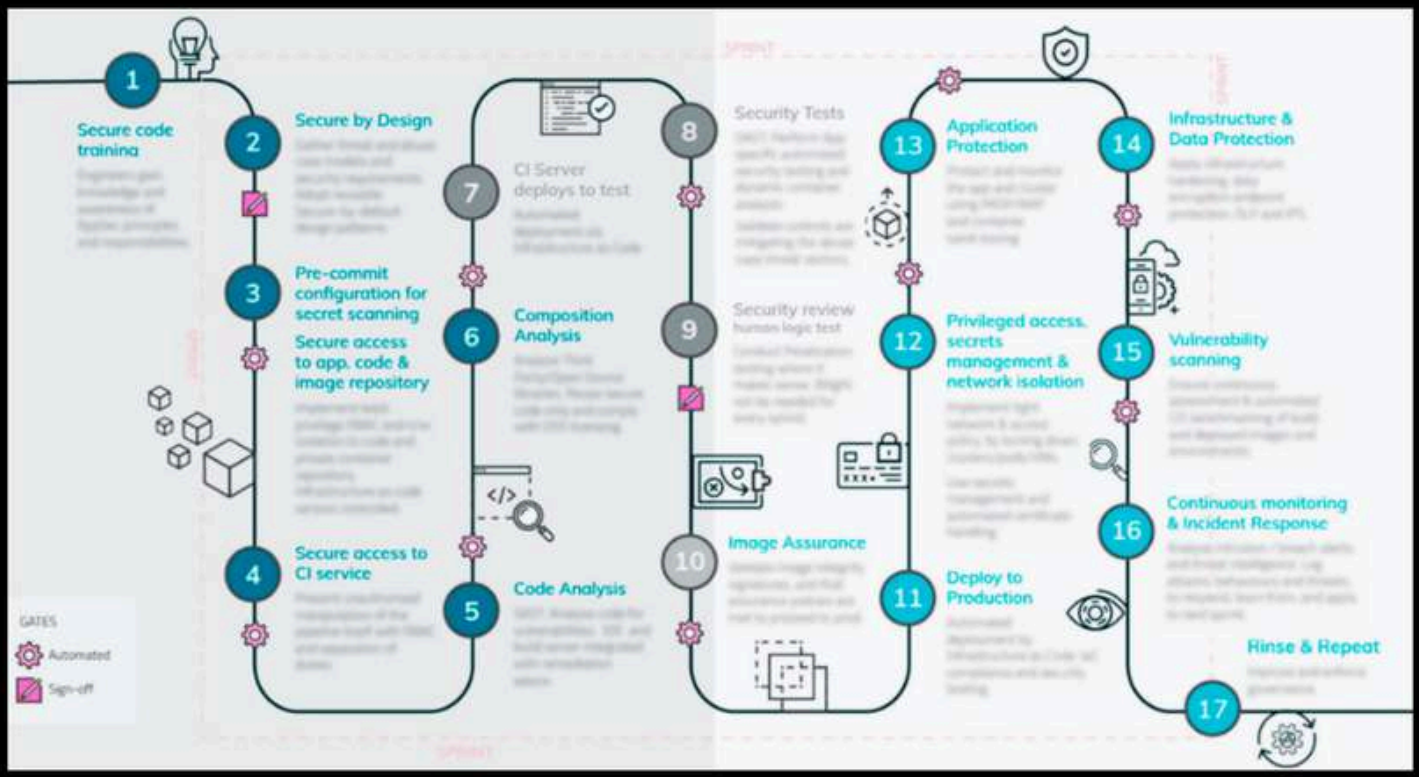
Level 2: Structured Adoption (Implementation Begins)



Organizations at this level begin to implement DevSecOps practices in a more structured and consistent manner. They start adopting best practices and incorporating automation into specific security tasks.

- **Documented Processes:** Security practices are defined and documented, creating a framework for consistent implementation.
- **Simple Automated Tasks:** Basic security tasks, such as static code analysis or vulnerability scanning, are automated using readily available tools.
- **Secure Coding Education Begins:** Developers receive formal training on secure coding practices and common security vulnerabilities. This might include teaching developers how to identify potential avenues of attack and take steps to mitigate those risks.

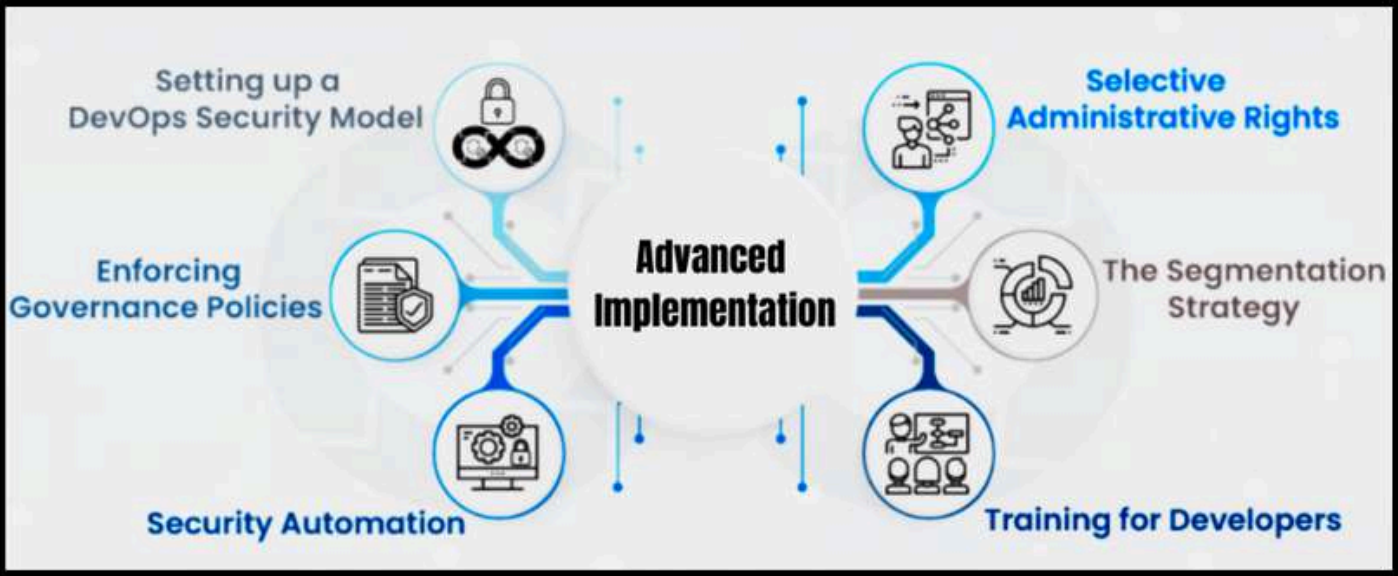
Level 3: Integrated Practices (Security Embedded in Workflows)



This level is marked by the integration of security practices into the core DevOps workflows, making security an integral part of the development process. Automation plays a key role, and continuous monitoring is established to ensure ongoing security.

- **Consistent Automation:** Security tasks are consistently automated across various stages of the software development lifecycle, improving efficiency and reducing human error.
- **Continuous Monitoring:** Systems are continuously monitored for security threats and vulnerabilities, enabling prompt detection and response to incidents. This can involve automating monitoring and incident generation.
- **Advanced Threat Modeling:** Organizations utilize structured threat modeling techniques to proactively identify and mitigate potential security risks. This involves examining applications through the eyes of an attacker to identify and highlight security flaws that could be exploited. Threat modeling helps teams better understand each other's roles, objectives, and pain points, resulting in a more collaborative and understanding organization.

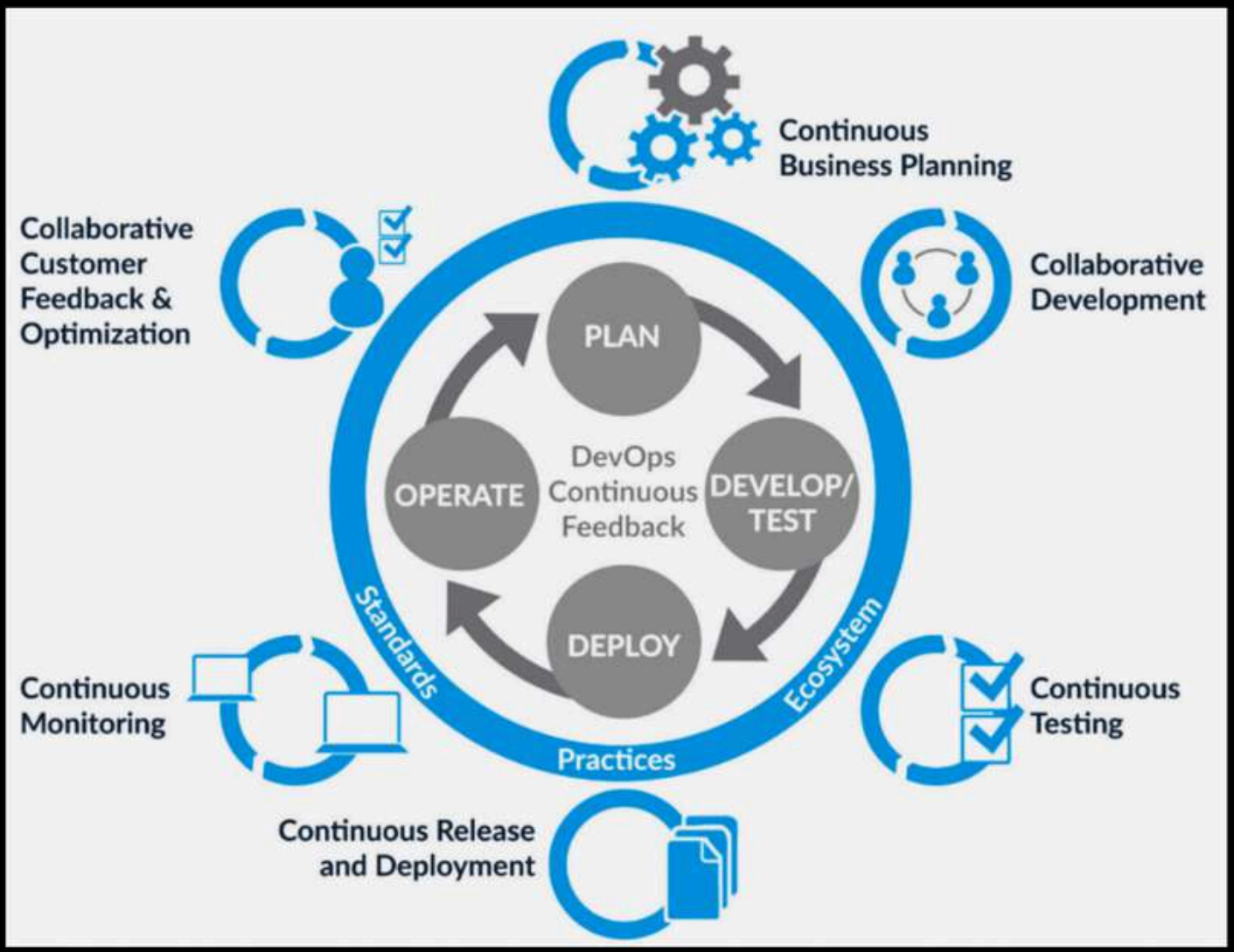
Level 4: Advanced Implementation (Proactive and Scalable Security)



Level 4 represents a mature DevSecOps implementation where security practices are proactive, scalable, and adaptable. Organizations at this level prioritize continuous improvement and stay abreast of emerging security threats and technologies.

- **Full Automation:** Security tasks are fully automated throughout the development pipeline, minimizing manual intervention and ensuring consistency.
- **Dynamic Security Testing:** Organizations implement dynamic security testing techniques, such as penetration testing and vulnerability scanning, to identify vulnerabilities in running applications.
- **Regular Training and Updates:** Teams undergo regular training to stay informed about evolving security threats, industry best practices, and the latest security tools and technologies. The focus is on educating developers about security processes and tools.

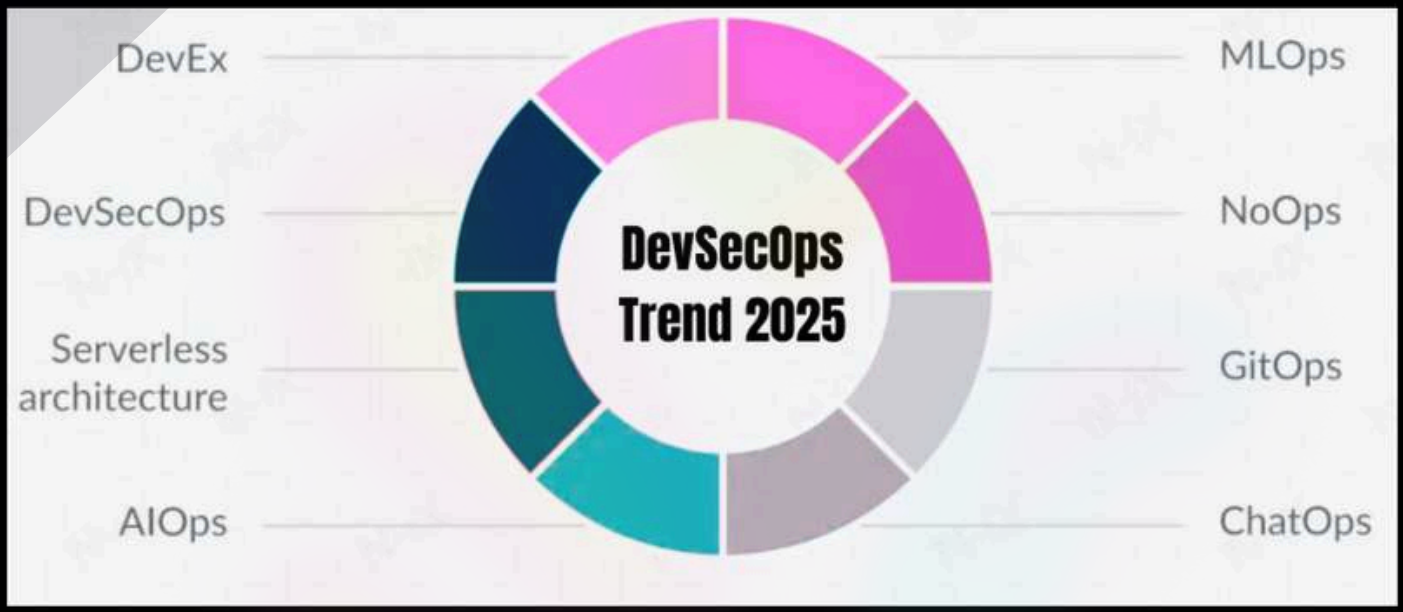
Level 5: Optimization and Resilience (Advanced Adaptability)



The highest level of maturity, Level 5, is characterized by the use of advanced technologies, self-healing systems, and a culture of continuous innovation in security practices.

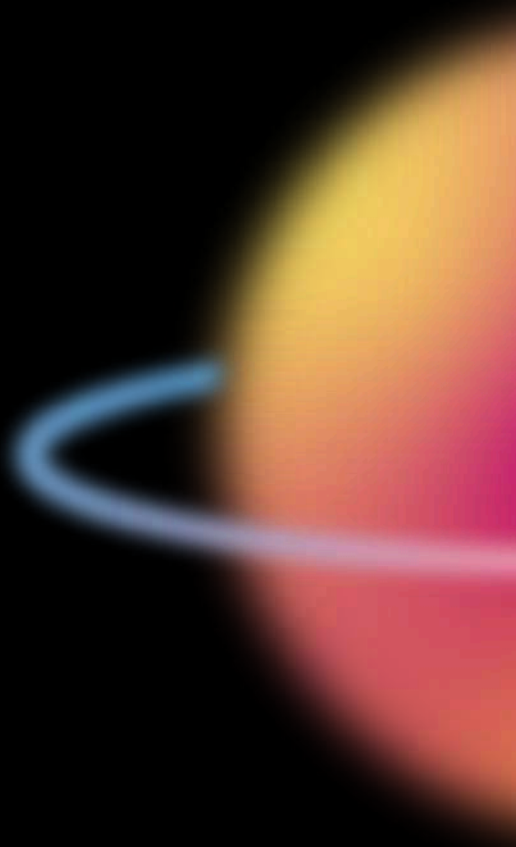
- **AI-Driven Threat Detection:** AI and machine learning are leveraged to enhance threat detection capabilities, analyze patterns, and predict potential security risks.
- **Self-Healing Systems:** Systems are designed to automatically detect and remediate security vulnerabilities without requiring human intervention, enhancing resilience and minimizing downtime.

Key Considerations Across Maturity Levels



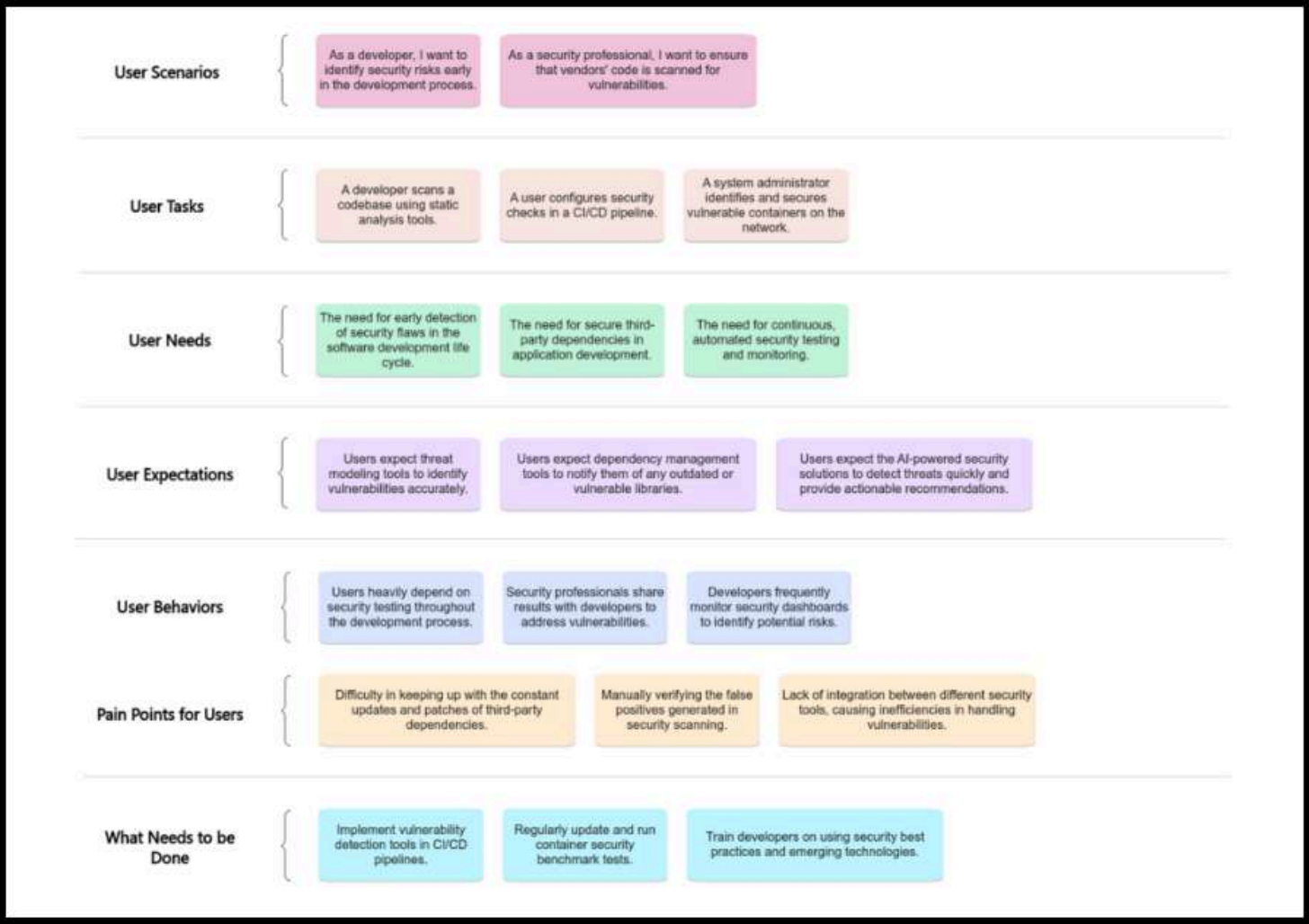
- **Security Metrics:** It's essential to track key security metrics, such as the number of vulnerabilities introduced per sprint (Findings per Sprint), to measure progress and identify areas for improvement. Organizations also need to define and track KPIs that align with their specific goals, such as reducing lead time for code deployment or minimizing the average time to resolve security incidents (MTTR).
- **Culture and Collaboration:** The success of DevSecOps relies heavily on a cultural shift towards shared responsibility for security. Breaking down silos between development, security, and operations teams and fostering collaboration is crucial. Open communication, blameless post-mortems, and shared understanding of DevSecOps goals are key indicators of a healthy culture.
- **Tooling:** Selecting and integrating the right security tools into the DevOps pipeline is essential for automating security tasks and streamlining workflows. Open-source tools can be a cost-effective option for organizations at various maturity levels. Organizations should assess existing tools and identify gaps in metrics support, considering customization or additional tooling to address specific business needs.

02 DevSecOps Technology Stacks in 2025



DevSecOps Technology Stacks in 2025

In 2025, the DevSecOps ecosystem is evolving to emphasize seamless integration of security into every phase of the DevOps lifecycle. Here are the most popular and effective tools categorized by each phase of the lifecycle:



Plan Phase

JIRA - For collaborative planning and tracking vulnerabilities within project backlogs.

Jira is a powerful project management tool that enables teams to track, plan, and manage security vulnerabilities throughout the software development lifecycle. It provides a centralized platform for creating, assigning, and tracking security-related work items with advanced traceability and reporting capabilities.

Key Test Cases:

1. **Vulnerability Tracking Test**

```
Given a new security vulnerability is identified
When the issue is logged in Jira
Then the issue should:
  - Have a unique identifier
  - Be assigned to the appropriate security team member
  - Include severity and impact classification
  - Allow detailed description and reproduction steps
```

2. **Security Workflow Validation**

```
Given a security vulnerability issue
When the issue moves through different workflow states
Then the system should:
  - Enforce appropriate permissions for state transitions
  - Log all state changes
  - Notify relevant stakeholders
  - Prevent unauthorized modifications
```

ThreatModeler - Automates threat modeling to identify risks early.

ThreatModeler is an automated threat modeling platform that helps organizations identify, prioritize, and mitigate potential security risks during the early stages of software design. It integrates with existing development tools to provide comprehensive threat analysis.

Key Test Cases:

3. **Threat Identification Automation**

```
Given a new software architecture design
When ThreatModeler analyzes the system
Then it should:
  - Automatically generate a comprehensive threat model
  - Identify potential attack vectors
  - Provide risk severity ratings
  - Suggest mitigation strategies
```

4. **Integration and Reporting Test**

```
Given a completed threat model
When the report is generated
Then the output should:
  - Be compatible with STRIDE methodology
  - Include detailed risk descriptions
  - Provide actionable remediation recommendations
  - Allow export to standard formats (PDF, XML)
```

Code Phase

GitGuardian - Secures Non-Human Identities and their secrets

GitGuardian provides organizations with tools to manage the lifecycle of non human identities (NHIs) and their associated secrets. GitGuardian helps discover and monitor all secrets, prioritize and remediate leaks at scale, and reduce the risk of breaches by protecting non-human identities.

Key Test Cases:

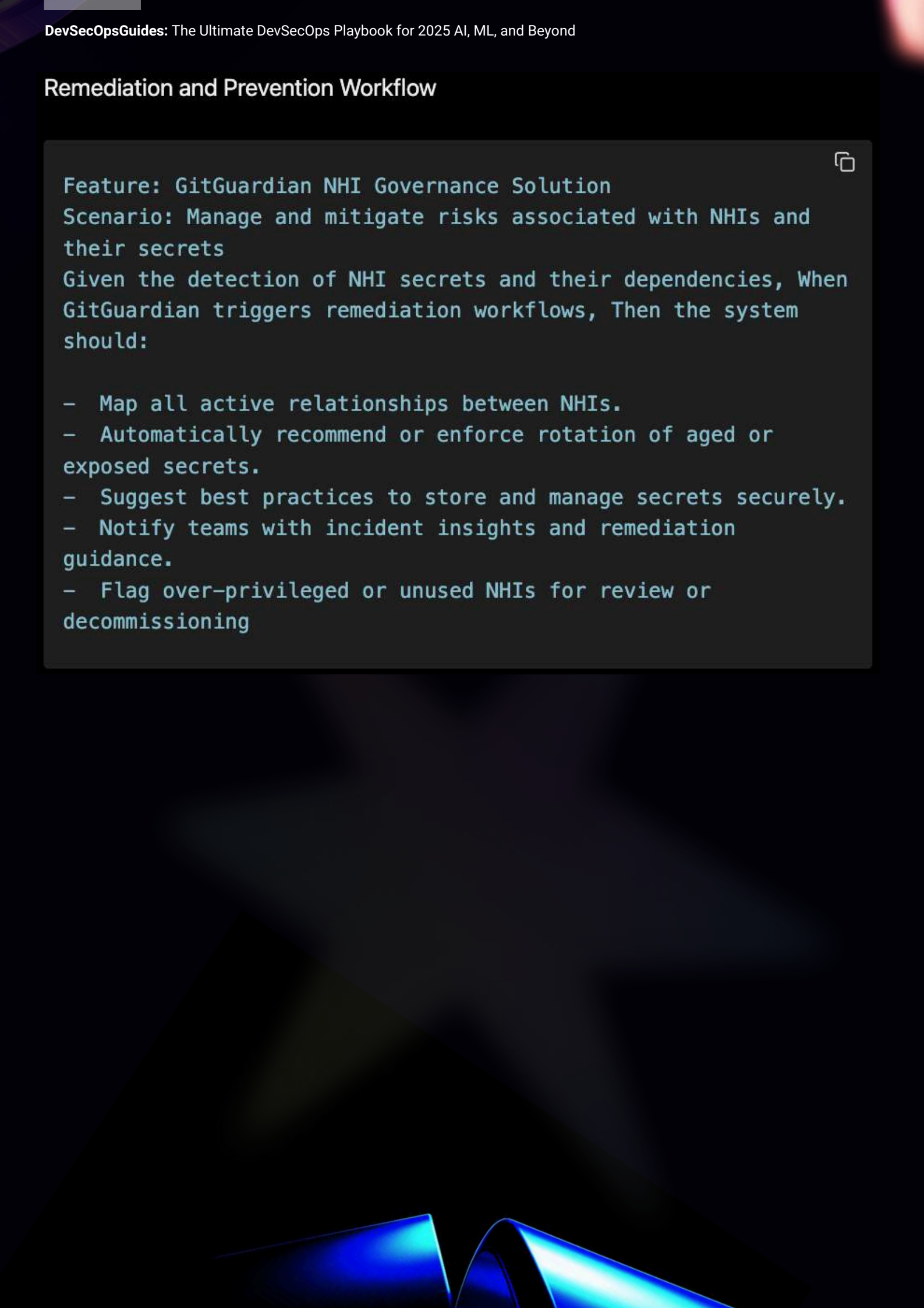
1. **Secrets Security and Non-Human Identity Governance**

Then the system should:

```
Feature: GitGuardian Non-Human Identity Security
Scenario: Detect and manage NHI secrets and relationships Given
a repository or system with machine identities and secrets,
When GitGuardian performs scanning

- Detect and locate secrets tied to NHIs (e.g., API keys,
tokens).
- Map the connections and relationships between NHIs.
- Provide real-time alerts for exposed secrets or anomalies.
- Identify secrets stored outside secure vaults.
- Offer visibility into the origins and permissions of each
secret.
```

Remediation and Prevention Workflow

```
Feature: GitGuardian NHI Governance Solution
Scenario: Manage and mitigate risks associated with NHIs and
their secrets
Given the detection of NHI secrets and their dependencies, When
GitGuardian triggers remediation workflows, Then the system
should:

-  Map all active relationships between NHIs.
-  Automatically recommend or enforce rotation of aged or
exposed secrets.
-  Suggest best practices to store and manage secrets securely.
-  Notify teams with incident insights and remediation
guidance.
-  Flag over-privileged or unused NHIs for review or
decommissioning
```



Snyk - Detects and remediates vulnerabilities in code dependencies and open-source libraries.

Snyk is an advanced security tool that specialized in identifying, prioritizing, and fixing vulnerabilities in open-source dependencies, containers, and code. It integrates seamlessly with development workflows, providing real-time security insights during the coding process.

Key Test Cases:

Dependency Vulnerability Detection

```
Feature: Snyk Dependency Security Scanning
Scenario: Identify and assess vulnerabilities in project
dependencies
  Given a project with multiple open-source dependencies
  When Snyk scans the project
  Then the system should:
    - Detect known security vulnerabilities
    - Provide CVSS severity ratings
    - Offer precise remediation recommendations
    - Support multiple programming languages
    - Generate comprehensive vulnerability reports
```

Remediation Workflow Test

```
Feature: Snyk Vulnerability Remediation
Scenario: Automatic vulnerability fix suggestions
  Given detected vulnerabilities in dependencies
  When Snyk analyzes the issues
  Then the system should:
    - Suggest specific version upgrades
    - Provide patch recommendations
    - Enable automatic dependency updates
    - Create pull requests with fixes
    - Prioritize critical security issues
```

Semgrep - Performs lightweight, customizable static code analysis.

Semgrep is a fast, open-source static analysis tool that enables developers to find and fix vulnerabilities with custom, language-specific rules.

Key Test Cases:

Custom Rule-Based Code Scanning

```
Feature: Semgrep Custom Security Rules
Scenario: Perform targeted code vulnerability scanning
  Given a custom security ruleset
  When Semgrep analyzes the codebase
  Then the system should:
    - Support custom, language-specific rules
    - Perform fast, lightweight scanning
    - Identify security and code quality issues
    - Generate detailed findings
    - Support multiple programming languages
```

Rule Creation and Management

```
Feature: Semgrep Rule Management
Scenario: Create and apply custom security rules
  Given a security requirement
  When a custom Semgrep rule is created
  Then the system should:
    - Allow creation of complex rule patterns
    - Support multiple rule configurations
    - Enable easy rule sharing
    - Provide rule testing mechanisms
    - Integrate with CI/CD pipelines
```

Build Phase

Jenkins - Automates builds with security-focused plugins.

Jenkins is an open-source automation server that enables organizations to build, test, and deploy software with enhanced security through numerous plugins and integrations. It provides a flexible and extensible platform for continuous integration and continuous delivery (CI/CD) with robust security features.

Key Test Cases:

Secure Build Pipeline Configuration

```
Feature: Jenkins Security Pipeline Configuration
Scenario: Validate secure build process
  Given a new software build configuration
  When Jenkins executes the build pipeline
  Then the system should:
    - Enforce role-based access controls
    - Implement credential management
    - Scan for potential security vulnerabilities
    - Generate comprehensive build logs
    - Support secure parameter handling
```

Security Plugin Integration Test

```
Feature: Jenkins Security Plugin Validation
Scenario: Verify security plugin functionality
  Given multiple security plugins are installed
  When a build is triggered
  Then the system should:
    - Perform static code analysis
    - Check dependency vulnerabilities
    - Validate configuration compliance
    - Generate security reports
    - Block builds with critical vulnerabilities
```

GitLab CI/CD - Integrates code quality and security testing into the pipeline.

GitLab CI/CD provides a comprehensive continuous integration and deployment platform with built-in security testing capabilities. It offers seamless integration of security checks directly into the build and deployment processes.

Key Test Cases:

Security-Integrated Build Pipeline

```
Feature: GitLab Security Build Integration
Scenario: Execute security-enhanced build process
  Given a code repository with CI/CD configuration
  When GitLab executes the build pipeline
  Then the system should:
    - Perform automated security scanning
    - Validate code quality metrics
    - Generate comprehensive security reports
    - Support parallel security testing
    - Provide real-time vulnerability feedback
```

Compliance and Governance Test

```
Feature: GitLab Compliance Validation
Scenario: Ensure build process meets security standards
  Given organizational security requirements
  When GitLab CI/CD pipeline is executed
  Then the system should:
    - Enforce predefined security policies
    - Block non-compliant builds
    - Generate audit trails
    - Support custom compliance rules
    - Provide detailed violation reports
```

Trivy - Scans Docker images during build time for vulnerabilities.

Trivy is an comprehensive vulnerability scanner for container images, fileystems, and Git repositories. It provides fast and accurate detection of security issues in containerized environments.

Key Test Cases:

9. **Container Image Security Scanning**

```
Feature: Trivy Container Image Vulnerability Detection
Scenario: Scan Docker image for vulnerabilities
  Given a Docker container image
  When Trivy performs security scanning
  Then the system should:
    - Identify known vulnerabilities
    - Provide CVSS severity ratings
    - Support multiple image formats
    - Generate detailed vulnerability reports
    - Offer remediation recommendations
```

Continuous Scanning Integration

```
Feature: Trivy Continuous Security Monitoring
Scenario: Integrate vulnerability scanning in build process
  Given a build pipeline
  When Trivy is integrated
  Then the system should:
    - Perform real-time image scanning
    - Block builds with critical vulnerabilities
    - Support custom severity thresholds
    - Generate comprehensive security reports
    - Provide actionable remediation guidance
```

Anchore - Enforces security policies for container images.

Anchore provides advanced container security scanning and policy enforcement, enabling organizations to implement comprehensive security checks for container images throughout the build and deployment processes.

Key Test Cases:

10. **Container Security Policy Validation**

```
Feature: Anchore Container Policy Enforcement
Scenario: Apply security policies to container images
  Given custom security policies
  When Anchore evaluates container images
  Then the system should:
    - Enforce predefined security rules
    - Detect policy violations
    - Support complex policy configurations
    - Generate detailed compliance reports
    - Block non-compliant container deployments
```

Advanced Vulnerability Assessment

```
Feature: Anchore Comprehensive Vulnerability Scanning
Scenario: Perform in-depth container image analysis
  Given a container image
  When Anchore performs scanning
  Then the system should:
    - Identify known and unknown vulnerabilities
    - Analyze package dependencies
    - Provide risk scoring
    - Support multiple image formats
    - Generate actionable remediation recommendations
```

Test Phase

OWASP ZAP - Performs dynamic application security testing (DAST).

OWASP ZAP is an open-source web application security scanner designed to find vulnerabilities in web applications during the testing phase. It provides automated scanning capabilities, helping identify security weaknesses through various testing techniques.

Key Test Cases:

11. **Comprehensive Web Application Security Scanning**

```
Feature: OWASP ZAP Vulnerability Detection
Scenario: Perform full web application security assessment
  Given a target web application
  When OWASP ZAP conducts a comprehensive scan
  Then the system should:
    - Detect OWASP Top 10 vulnerabilities
    - Perform automated penetration testing
    - Generate detailed vulnerability reports
    - Identify potential security risks
    - Provide actionable remediation guidance
```

Advanced Scanning Techniques

```
Feature: OWASP ZAP Advanced Security Testing
Scenario: Execute multi-layered security assessment
  Given a complex web application
  When ZAP performs advanced scanning
  Then the system should:
    - Support multiple scanning strategies
    - Conduct authenticated and unauthenticated scans
    - Detect hidden vulnerabilities
    - Simulate various attack scenarios
    - Generate comprehensive security insights
```

Burp Suite - Identifies web application vulnerabilities.

Burp Suite is an integrated platform for performing security testing of web applications. It provides advanced scanning, intercepting, and manipulation capabilities to identify sophisticated security vulnerabilities.

Key Test Cases:

12. **Web Application Vulnerability Assessment**

```
Feature: Burp Suite Comprehensive Vulnerability Scanning
Scenario: Perform in-depth web application security testing
  Given a target web application
  When Burp Suite conducts security assessment
  Then the system should:
    - Identify complex security vulnerabilities
    - Perform detailed application mapping
    - Support manual and automated testing
    - Generate comprehensive vulnerability reports
    - Provide advanced exploitation analysis
```

Advanced Penetration Testing

```
Feature: Burp Suite Penetration Testing Capabilities
Scenario: Execute advanced security testing
  Given a web application with complex architecture
  When Burp Suite performs penetration testing
  Then the system should:
    - Simulate sophisticated attack vectors
    - Detect subtle security weaknesses
    - Support custom testing scenarios
    - Provide detailed exploit information
    - Generate actionable security recommendations
```

SAST Tools (e.g., Checkmarx) - Ensures code security through static analysis.

Static Application Security Testing (SAST) tools like Checkmarx analyze source code or compiled versions of code to help find security vulnerabilities before the application is run.

Key Test Cases:

13. **Comprehensive Code Security Analysis**

```
Feature: SAST Code Vulnerability Detection
Scenario: Perform static code security analysis
  Given a complete codebase
  When SAST tool scans the code
  Then the system should:
    - Identify potential security vulnerabilities
    - Analyze code across multiple languages
    - Provide precise vulnerability locations
    - Generate detailed remediation recommendations
    - Support custom security rule configurations
```

Security Policy Enforcement

```
Feature: SAST Security Policy Validation
Scenario: Enforce security standards in code
  Given organizational security policies
  When SAST tool analyzes the codebase
  Then the system should:
    - Validate code against security standards
    - Block commits with critical vulnerabilities
    - Generate comprehensive compliance reports
    - Support custom security rules
    - Provide actionable developer guidance
```

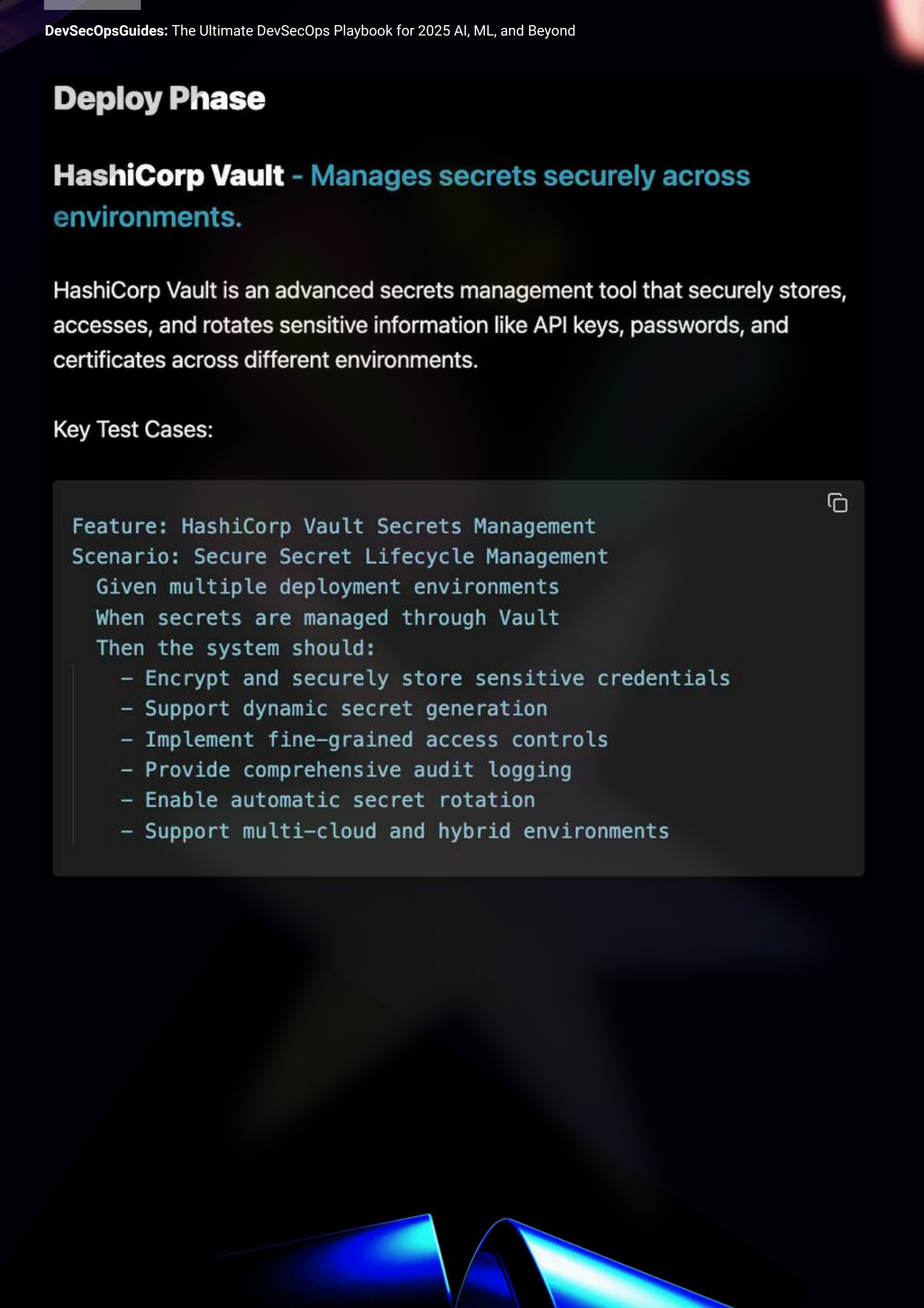
Deploy Phase

HashiCorp Vault - Manages secrets securely across environments.

HashiCorp Vault is an advanced secrets management tool that securely stores, accesses, and rotates sensitive information like API keys, passwords, and certificates across different environments.

Key Test Cases:

```
Feature: HashiCorp Vault Secrets Management
Scenario: Secure Secret Lifecycle Management
  Given multiple deployment environments
  When secrets are managed through Vault
  Then the system should:
    - Encrypt and securely store sensitive credentials
    - Support dynamic secret generation
    - Implement fine-grained access controls
    - Provide comprehensive audit logging
    - Enable automatic secret rotation
    - Support multi-cloud and hybrid environments
```

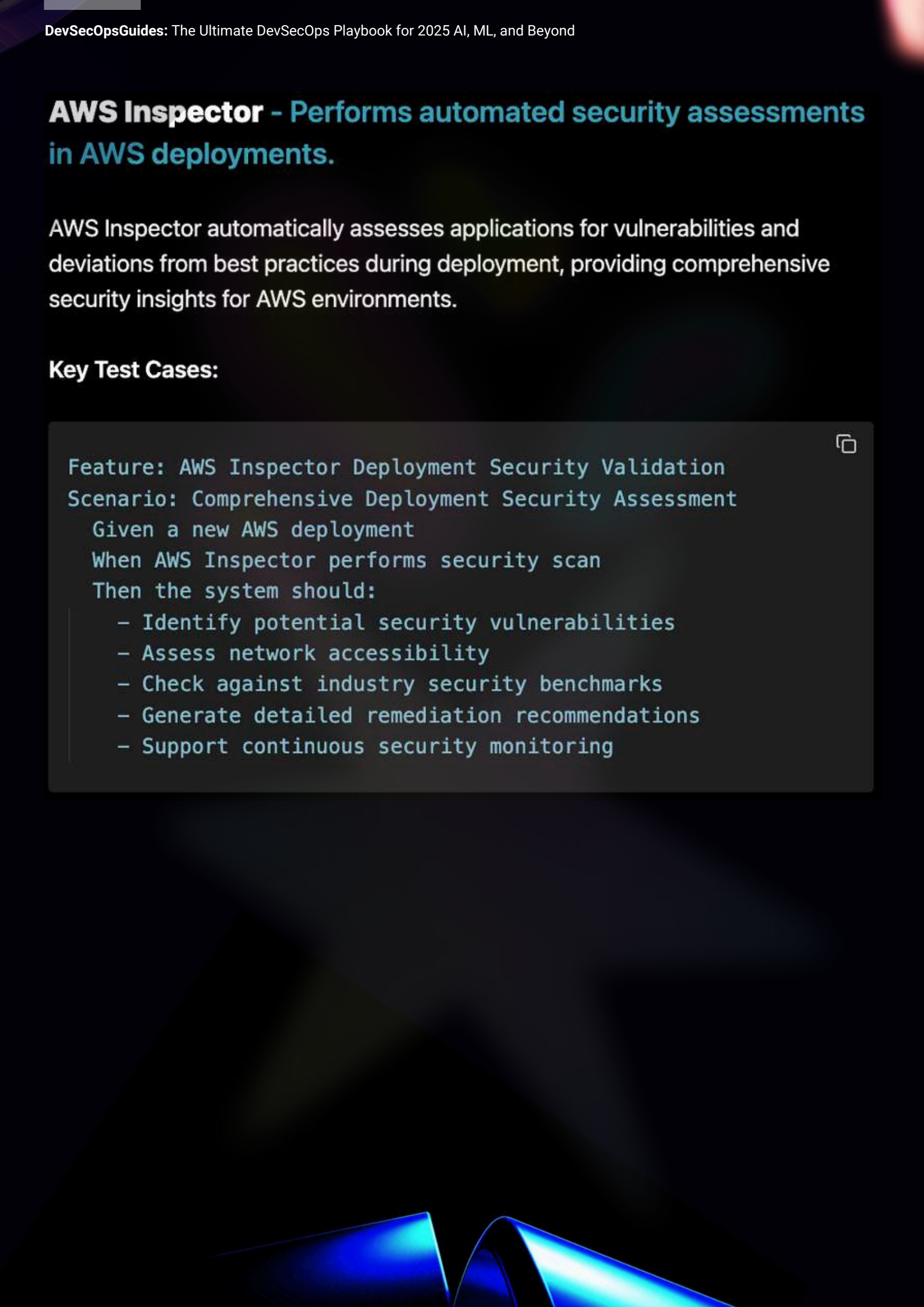


AWS Inspector - Performs automated security assessments in AWS deployments.

AWS Inspector automatically assesses applications for vulnerabilities and deviations from best practices during deployment, providing comprehensive security insights for AWS environments.

Key Test Cases:

```
Feature: AWS Inspector Deployment Security Validation
Scenario: Comprehensive Deployment Security Assessment
  Given a new AWS deployment
  When AWS Inspector performs security scan
  Then the system should:
    - Identify potential security vulnerabilities
    - Assess network accessibility
    - Check against industry security benchmarks
    - Generate detailed remediation recommendations
    - Support continuous security monitoring
```

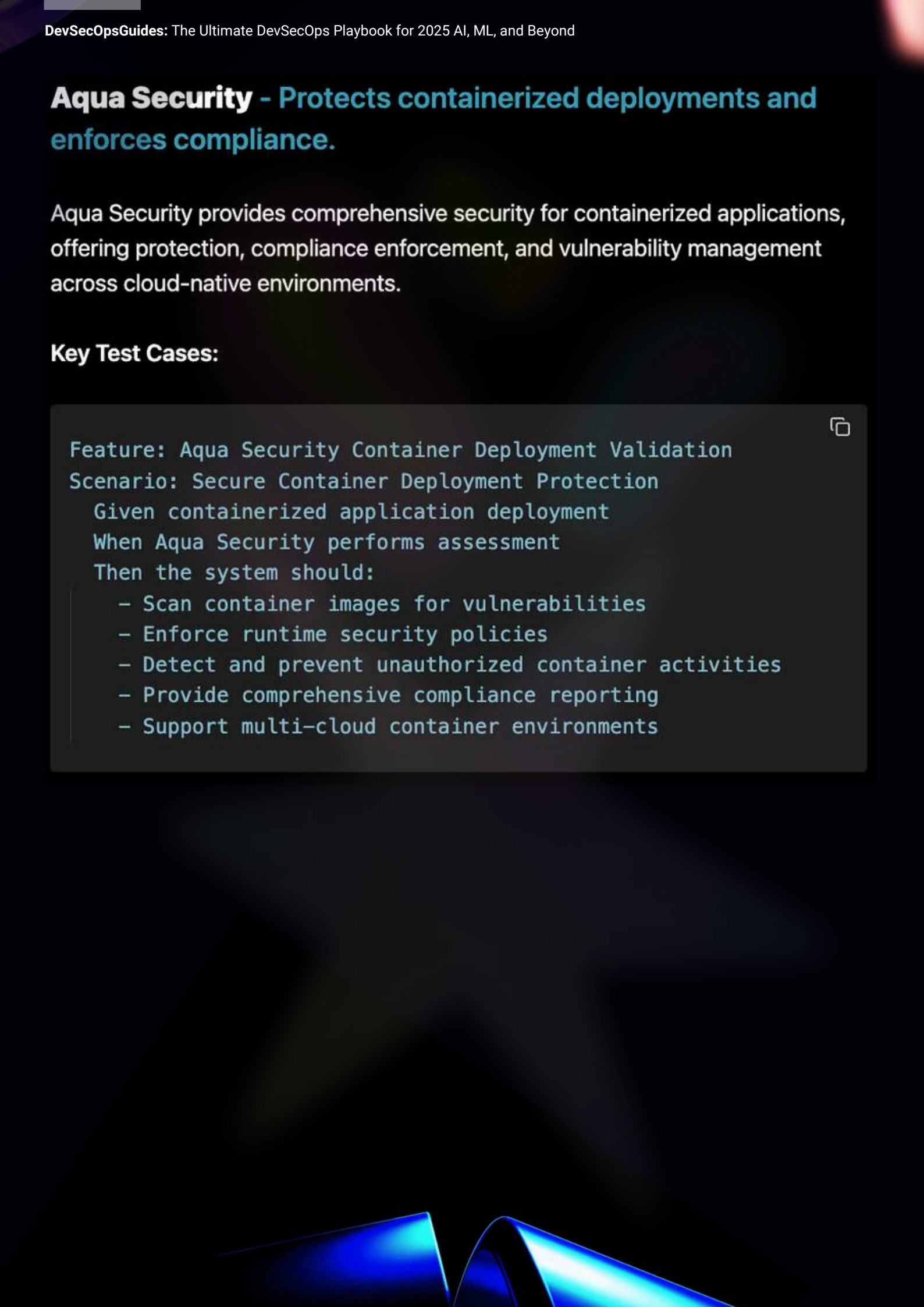


Aqua Security - Protects containerized deployments and enforces compliance.

Aqua Security provides comprehensive security for containerized applications, offering protection, compliance enforcement, and vulnerability management across cloud-native environments.

Key Test Cases:

```
Feature: Aqua Security Container Deployment Validation
Scenario: Secure Container Deployment Protection
  Given containerized application deployment
  When Aqua Security performs assessment
  Then the system should:
    - Scan container images for vulnerabilities
    - Enforce runtime security policies
    - Detect and prevent unauthorized container activities
    - Provide comprehensive compliance reporting
    - Support multi-cloud container environments
```



Kubernetes Security Tools (e.g., Kube-bench) - Ensures secure orchestration.

Kube-bench is an open-source tool that checks Kubernetes clusters against the CIS (Center for Internet Security) Kubernetes Benchmark, ensuring security best practices and identifying potential configuration vulnerabilities in Kubernetes deployments.

Key Test Cases:

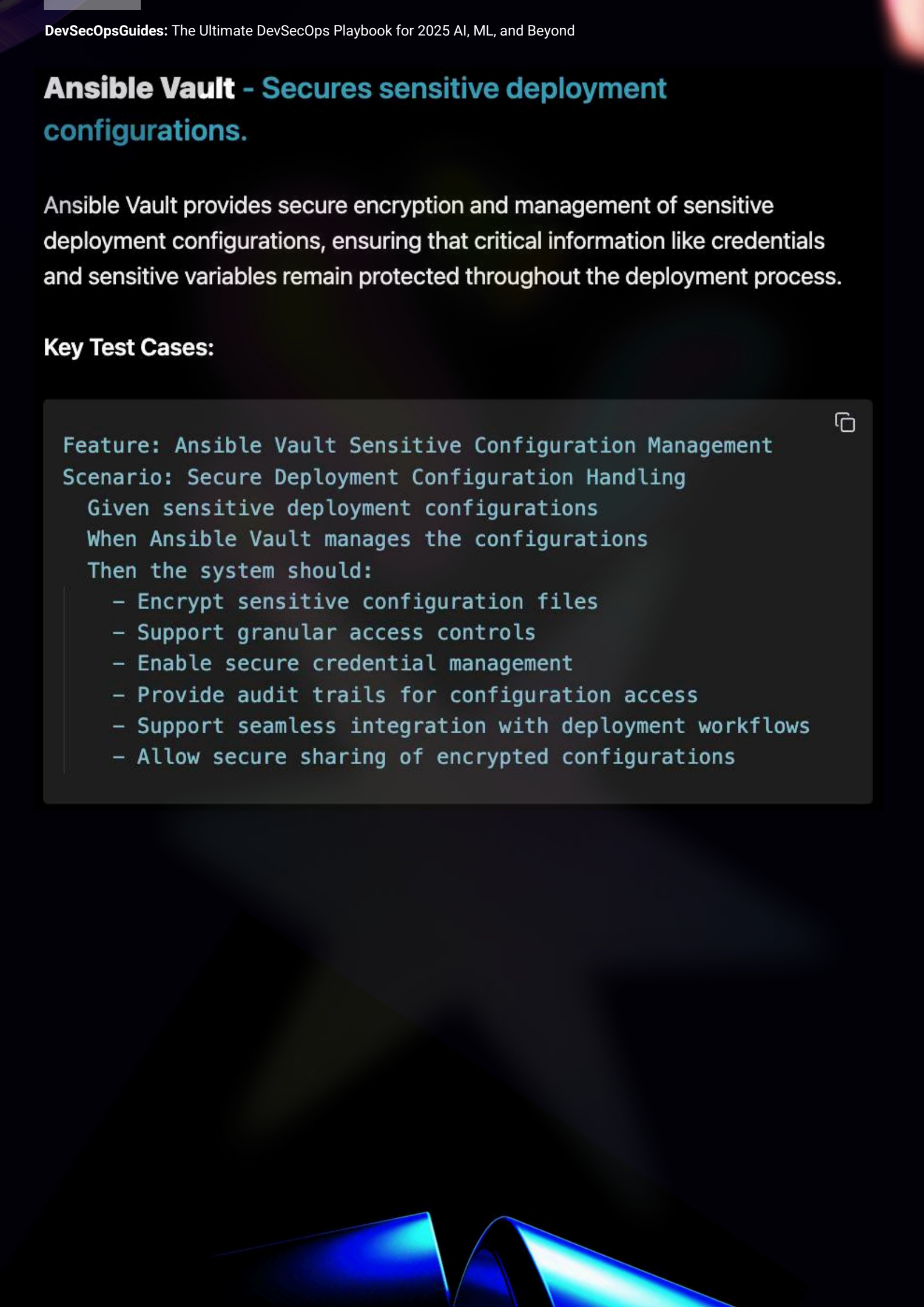
```
Feature: Kubernetes Security Compliance Assessment
Scenario: Comprehensive Kubernetes Security Validation
  Given a Kubernetes cluster deployment
  When Kube-bench performs security assessment
  Then the system should:
    - Validate cluster against CIS security benchmarks
    - Identify security misconfigurations
    - Provide detailed remediation recommendations
    - Support multiple Kubernetes deployment types
    - Generate comprehensive compliance reports
    - Assess both master and worker node configurations
```

Ansible Vault - Secures sensitive deployment configurations.

Ansible Vault provides secure encryption and management of sensitive deployment configurations, ensuring that critical information like credentials and sensitive variables remain protected throughout the deployment process.

Key Test Cases:

```
Feature: Ansible Vault Sensitive Configuration Management
Scenario: Secure Deployment Configuration Handling
  Given sensitive deployment configurations
  When Ansible Vault manages the configurations
  Then the system should:
    - Encrypt sensitive configuration files
    - Support granular access controls
    - Enable secure credential management
    - Provide audit trails for configuration access
    - Support seamless integration with deployment workflows
    - Allow secure sharing of encrypted configurations
```



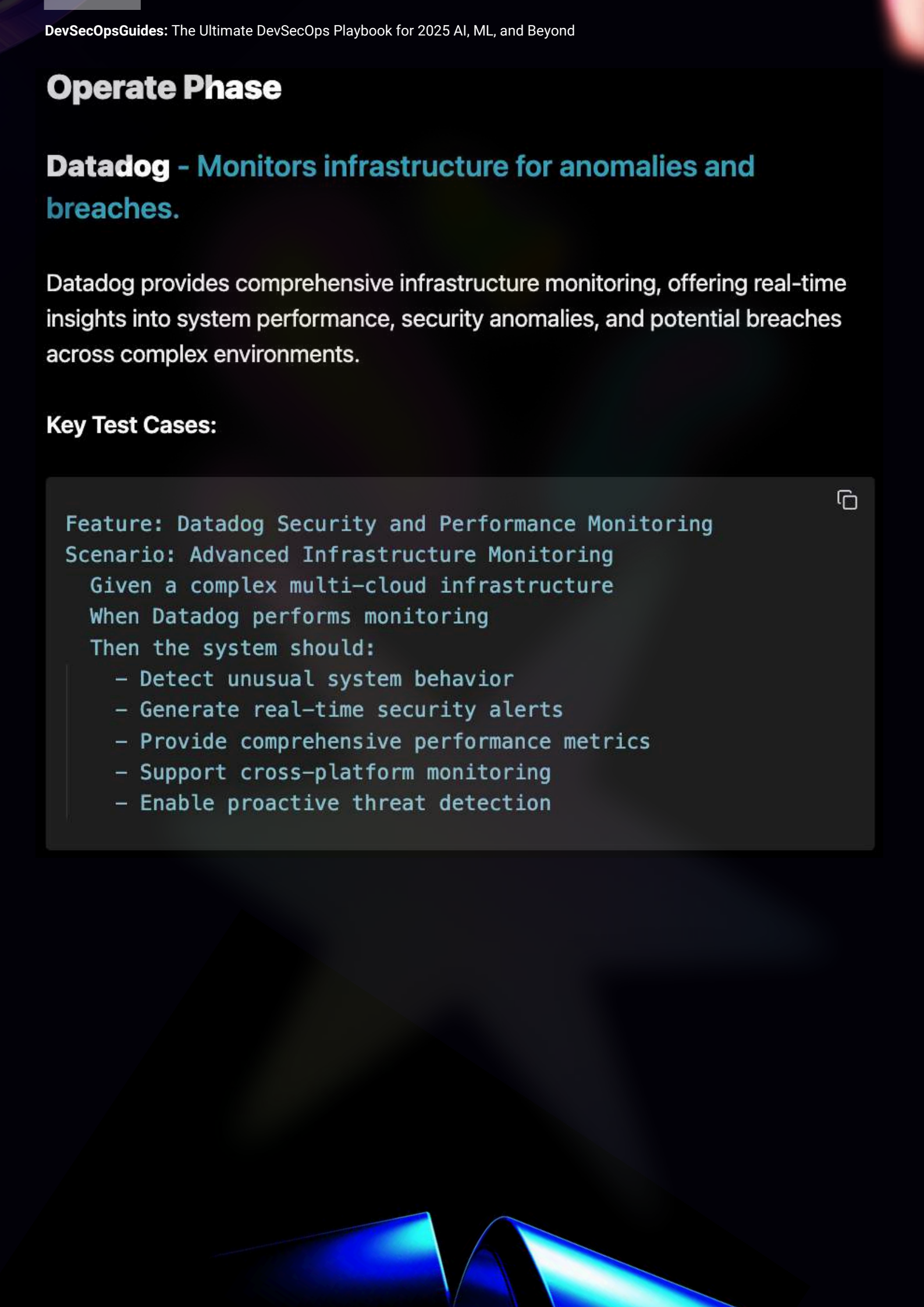
Operate Phase

Datadog - Monitors infrastructure for anomalies and breaches.

Datadog provides comprehensive infrastructure monitoring, offering real-time insights into system performance, security anomalies, and potential breaches across complex environments.

Key Test Cases:

```
Feature: Datadog Security and Performance Monitoring
Scenario: Advanced Infrastructure Monitoring
  Given a complex multi-cloud infrastructure
  When Datadog performs monitoring
  Then the system should:
    - Detect unusual system behavior
    - Generate real-time security alerts
    - Provide comprehensive performance metrics
    - Support cross-platform monitoring
    - Enable proactive threat detection
```

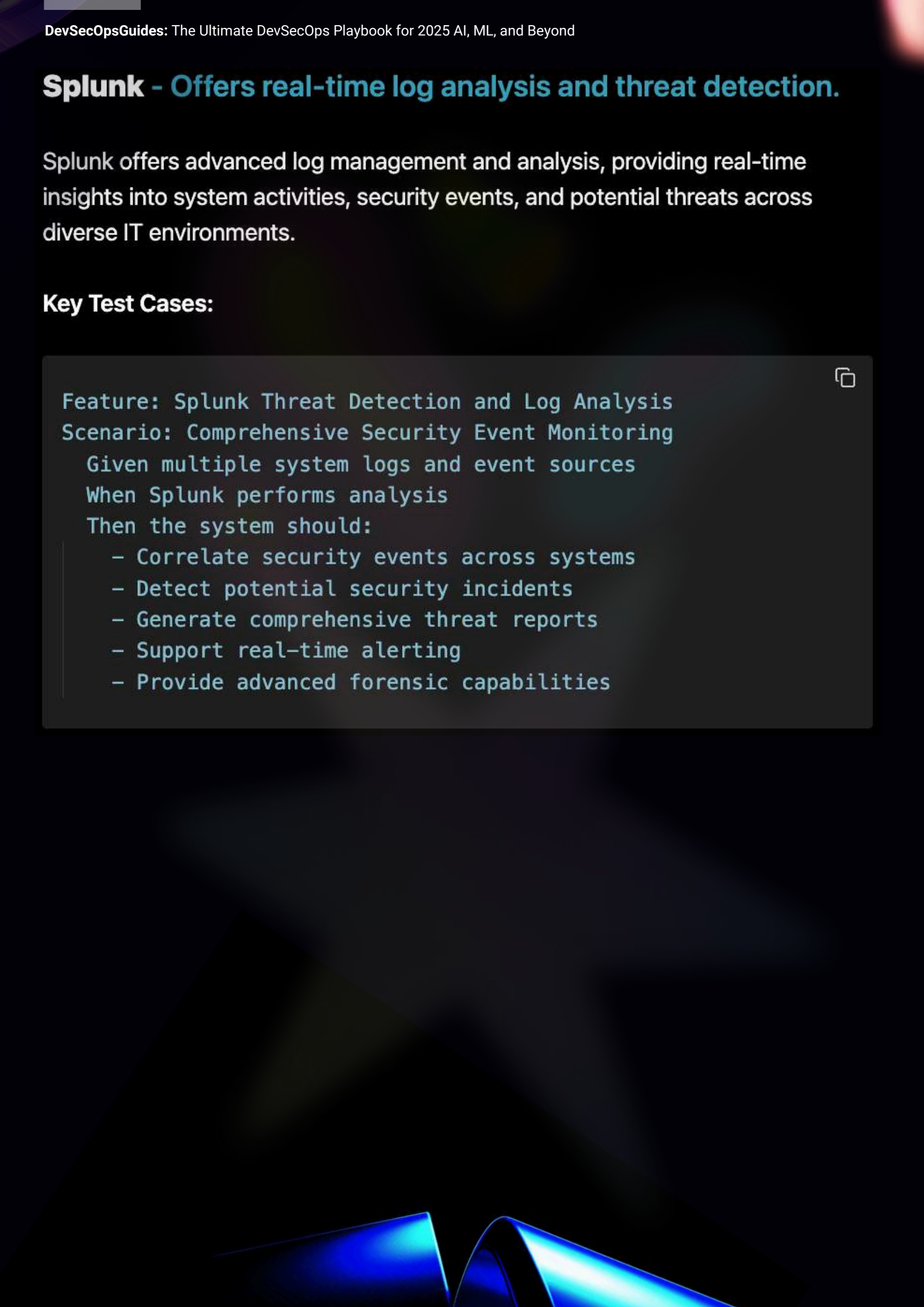


Splunk - Offers real-time log analysis and threat detection.

Splunk offers advanced log management and analysis, providing real-time insights into system activities, security events, and potential threats across diverse IT environments.

Key Test Cases:

```
Feature: Splunk Threat Detection and Log Analysis
Scenario: Comprehensive Security Event Monitoring
  Given multiple system logs and event sources
  When Splunk performs analysis
  Then the system should:
    - Correlate security events across systems
    - Detect potential security incidents
    - Generate comprehensive threat reports
    - Support real-time alerting
    - Provide advanced forensic capabilities
```

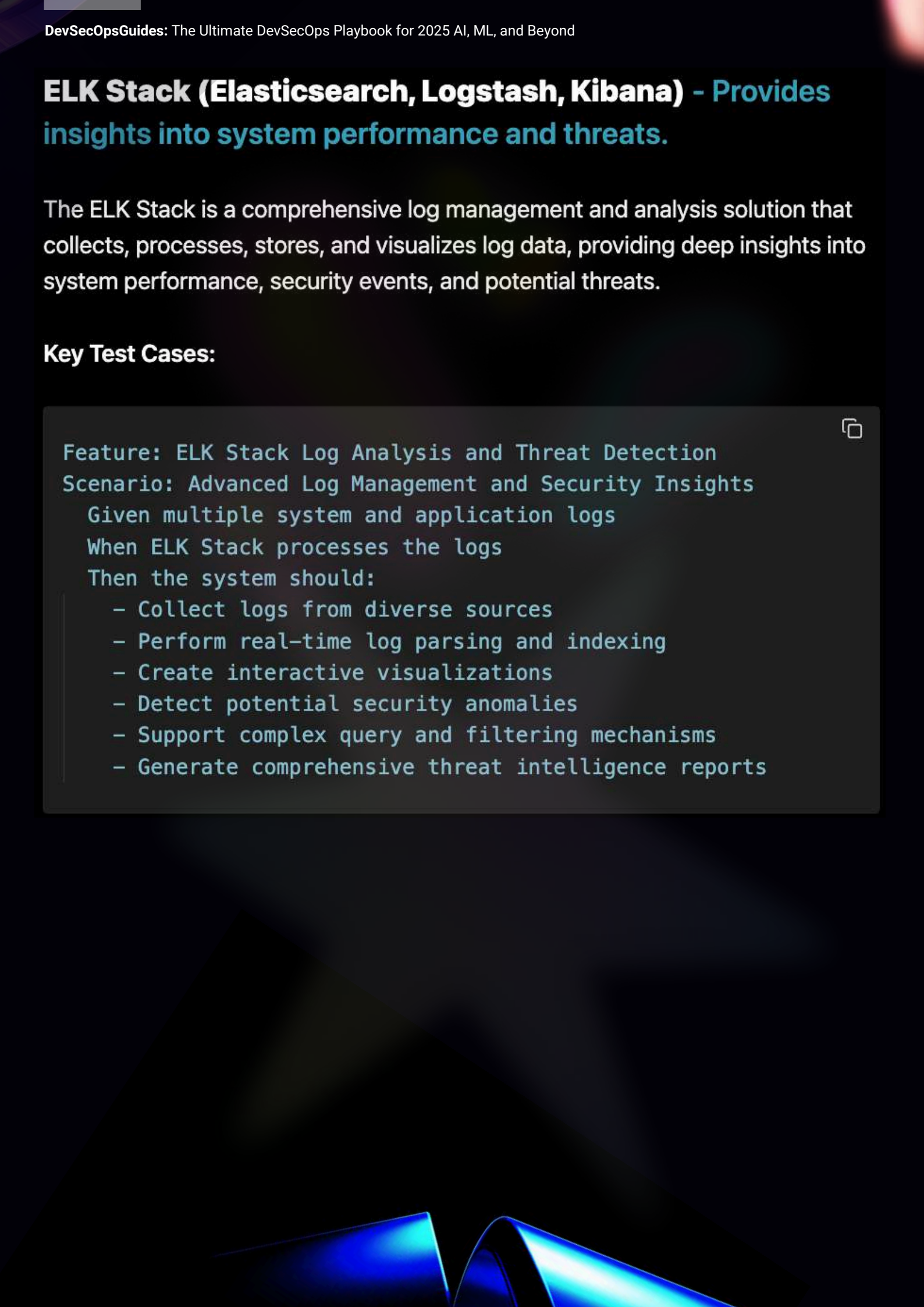


ELK Stack (Elasticsearch, Logstash, Kibana) - Provides insights into system performance and threats.

The ELK Stack is a comprehensive log management and analysis solution that collects, processes, stores, and visualizes log data, providing deep insights into system performance, security events, and potential threats.

Key Test Cases:

```
Feature: ELK Stack Log Analysis and Threat Detection
Scenario: Advanced Log Management and Security Insights
  Given multiple system and application logs
  When ELK Stack processes the logs
  Then the system should:
    - Collect logs from diverse sources
    - Perform real-time log parsing and indexing
    - Create interactive visualizations
    - Detect potential security anomalies
    - Support complex query and filtering mechanisms
    - Generate comprehensive threat intelligence reports
```

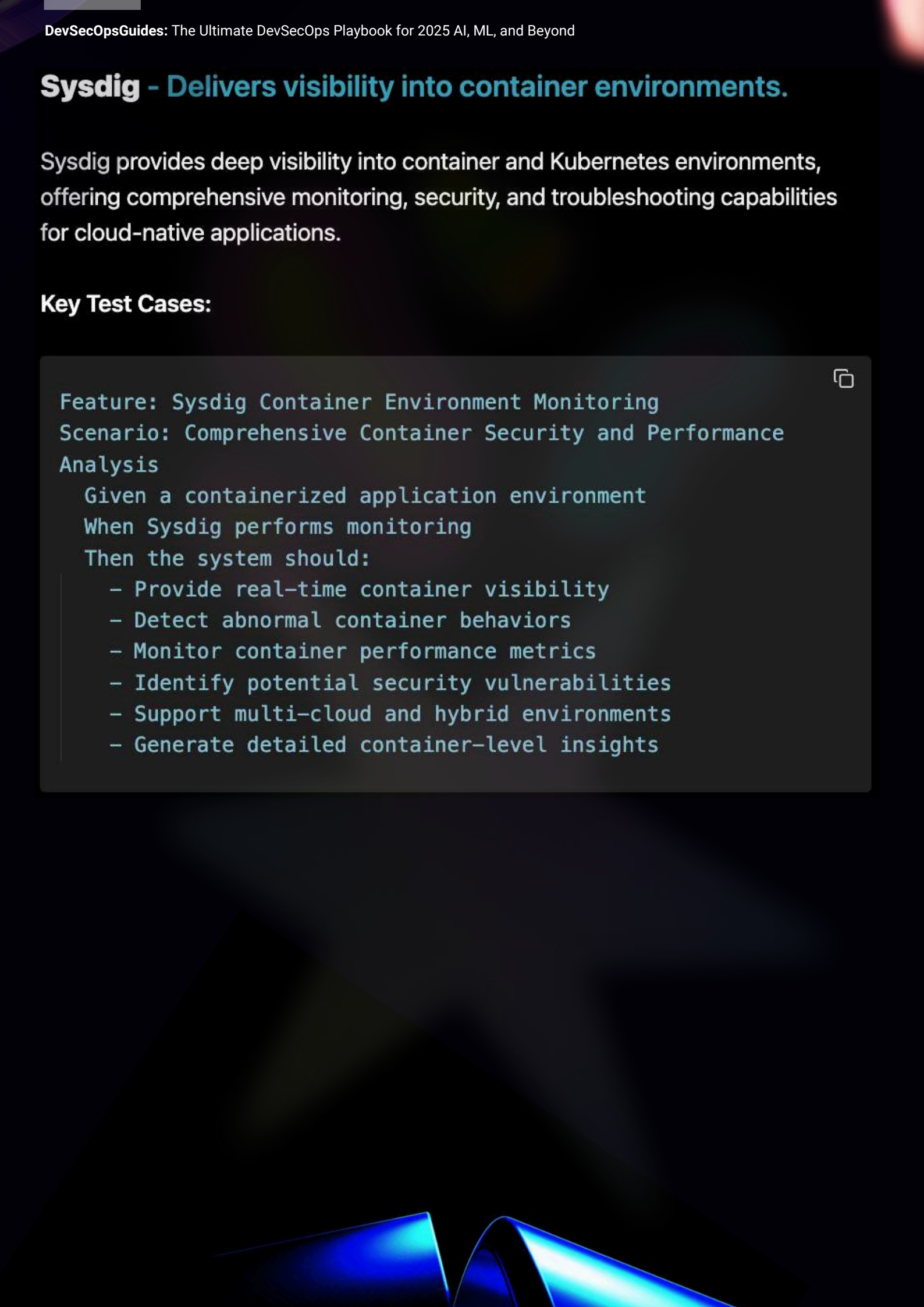


Sysdig - Delivers visibility into container environments.

Sysdig provides deep visibility into container and Kubernetes environments, offering comprehensive monitoring, security, and troubleshooting capabilities for cloud-native applications.

Key Test Cases:

```
Feature: Sysdig Container Environment Monitoring
Scenario: Comprehensive Container Security and Performance
Analysis
  Given a containerized application environment
  When Sysdig performs monitoring
  Then the system should:
    - Provide real-time container visibility
    - Detect abnormal container behaviors
    - Monitor container performance metrics
    - Identify potential security vulnerabilities
    - Support multi-cloud and hybrid environments
    - Generate detailed container-level insights
```

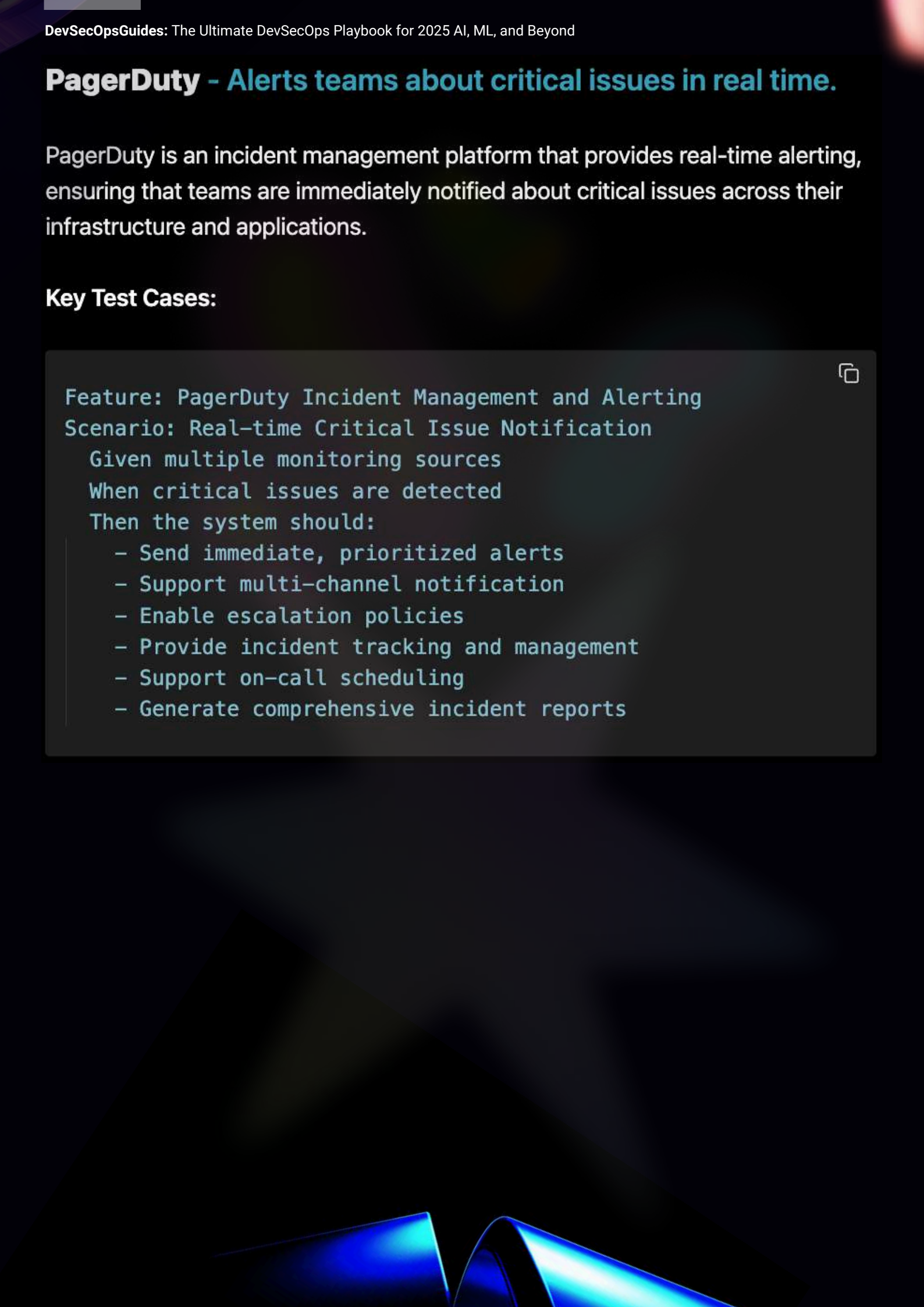


PagerDuty - Alerts teams about critical issues in real time.

PagerDuty is an incident management platform that provides real-time alerting, ensuring that teams are immediately notified about critical issues across their infrastructure and applications.

Key Test Cases:

```
Feature: PagerDuty Incident Management and Alerting
Scenario: Real-time Critical Issue Notification
  Given multiple monitoring sources
  When critical issues are detected
  Then the system should:
    - Send immediate, prioritized alerts
    - Support multi-channel notification
    - Enable escalation policies
    - Provide incident tracking and management
    - Support on-call scheduling
    - Generate comprehensive incident reports
```



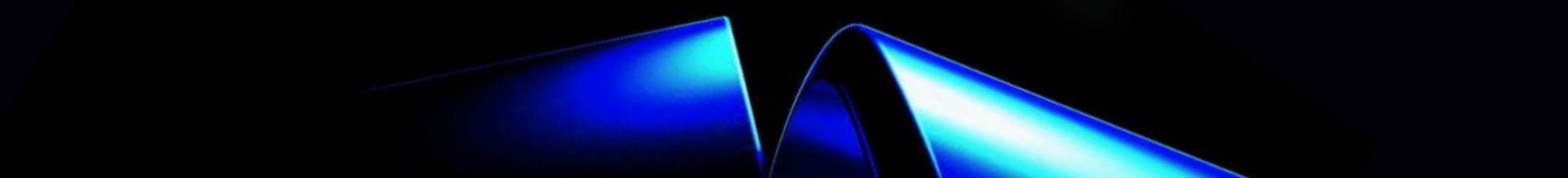
Monitor and Feedback Phases

Prometheus - Monitors system performance with alerting capabilities.

Prometheus is an open-source monitoring and alerting toolkit designed to provide robust performance monitoring and generate actionable alerts for complex system environments.

Key Test Cases:

```
Feature: Prometheus System Monitoring and Alerting
Scenario: Advanced Performance and Security Monitoring
  Given a distributed system infrastructure
  When Prometheus performs monitoring
  Then the system should:
    - Collect comprehensive performance metrics
    - Generate intelligent alerts
    - Support multi-dimensional data collection
    - Provide real-time system health insights
    - Enable custom monitoring configurations
```

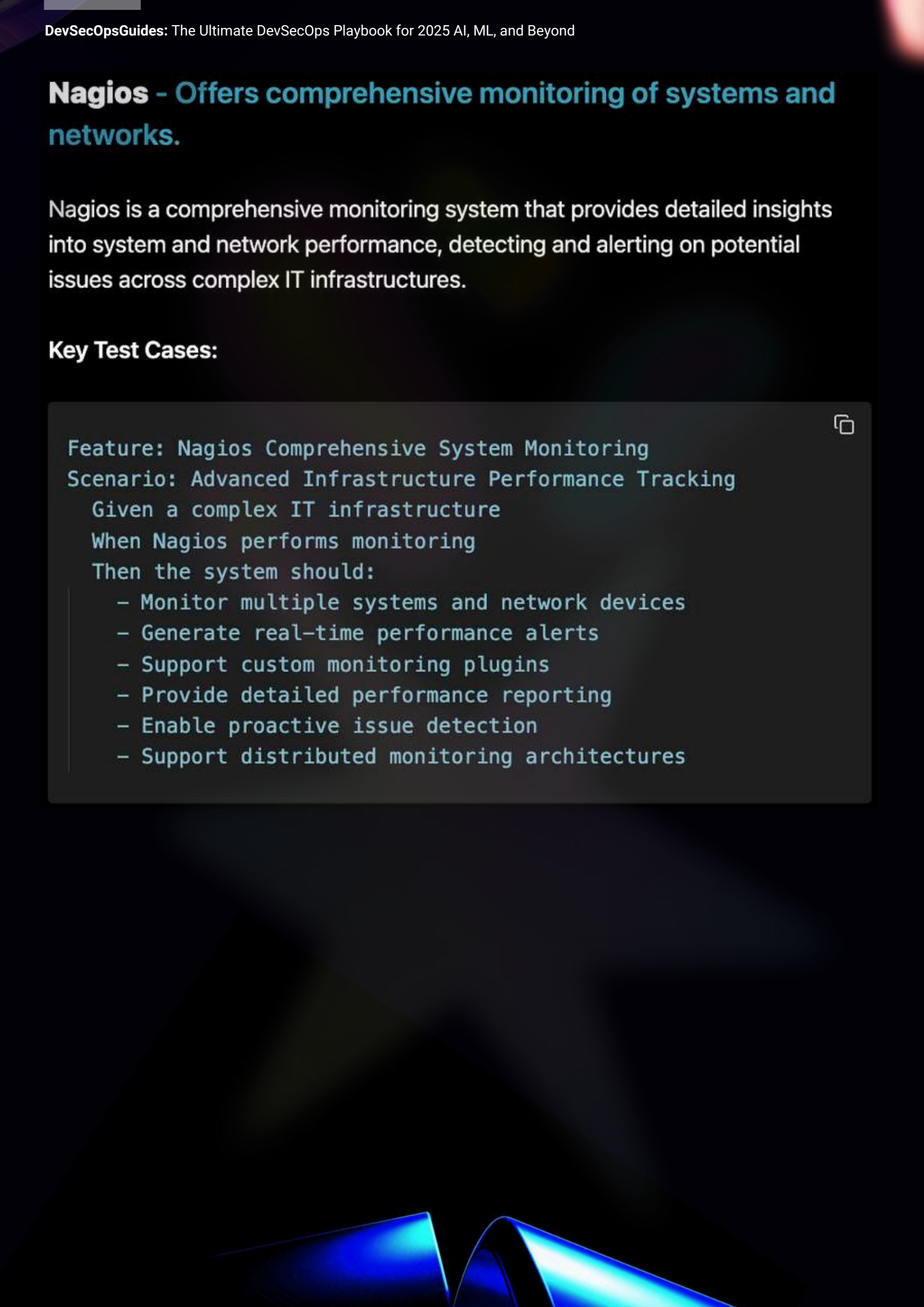


Nagios - Offers comprehensive monitoring of systems and networks.

Nagios is a comprehensive monitoring system that provides detailed insights into system and network performance, detecting and alerting on potential issues across complex IT infrastructures.

Key Test Cases:

```
Feature: Nagios Comprehensive System Monitoring
Scenario: Advanced Infrastructure Performance Tracking
  Given a complex IT infrastructure
  When Nagios performs monitoring
  Then the system should:
    - Monitor multiple systems and network devices
    - Generate real-time performance alerts
    - Support custom monitoring plugins
    - Provide detailed performance reporting
    - Enable proactive issue detection
    - Support distributed monitoring architectures
```

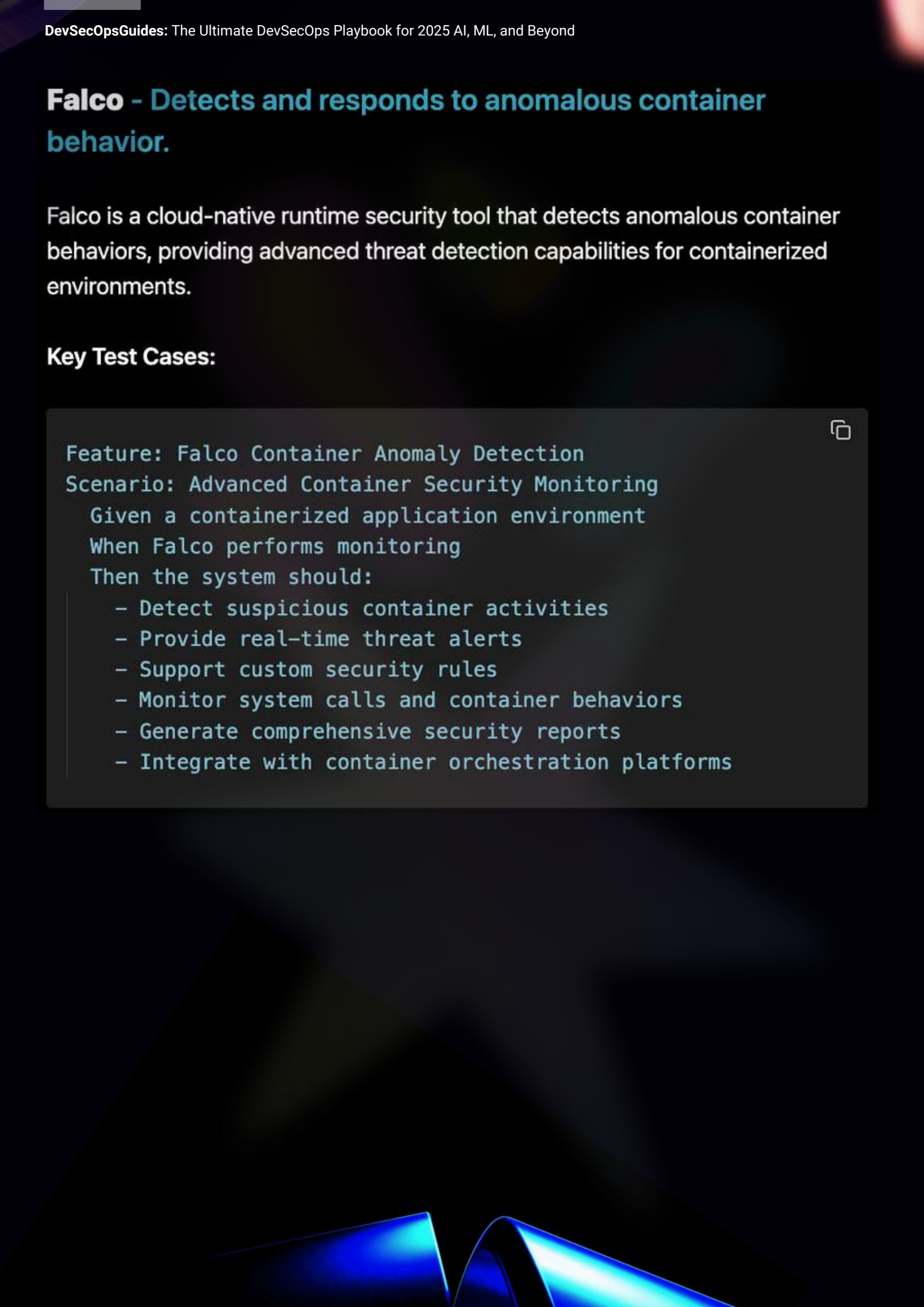


Falco - Detects and responds to anomalous container behavior.

Falco is a cloud-native runtime security tool that detects anomalous container behaviors, providing advanced threat detection capabilities for containerized environments.

Key Test Cases:

```
Feature: Falco Container Anomaly Detection
Scenario: Advanced Container Security Monitoring
  Given a containerized application environment
  When Falco performs monitoring
  Then the system should:
    - Detect suspicious container activities
    - Provide real-time threat alerts
    - Support custom security rules
    - Monitor system calls and container behaviors
    - Generate comprehensive security reports
    - Integrate with container orchestration platforms
```



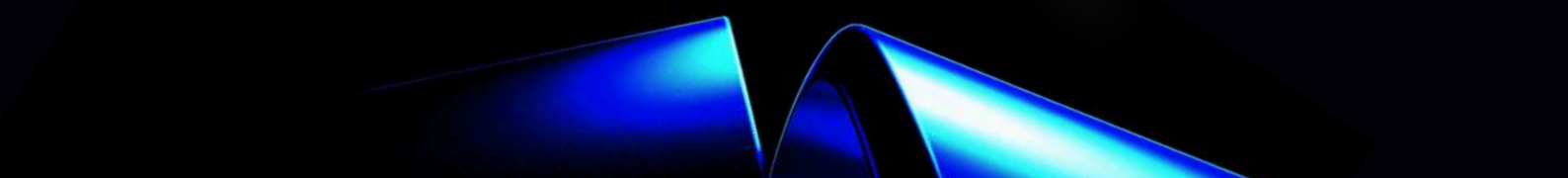
Virtual Patching

ModSecurity - Functions as a web application firewall (WAF) to block exploits without code changes.

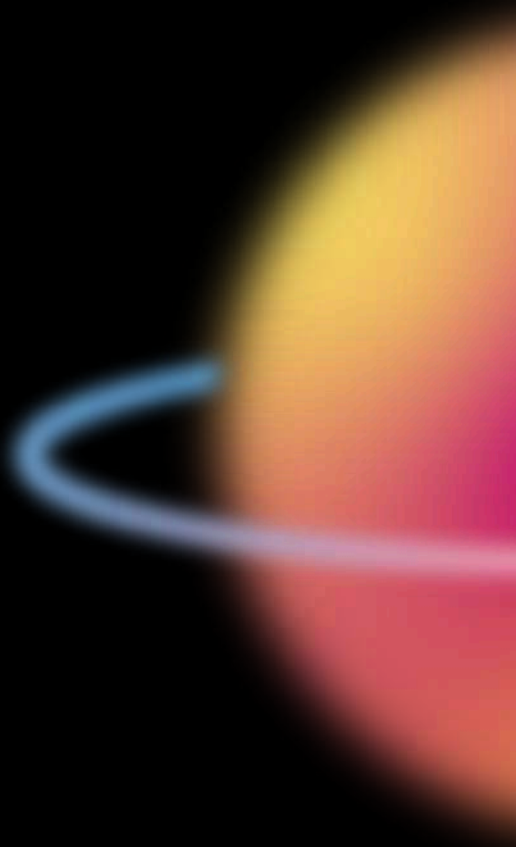
ModSecurity is an open-source web application firewall that provides real-time application security, enabling organizations to implement virtual patches without modifying underlying application code.

Key Test Cases:

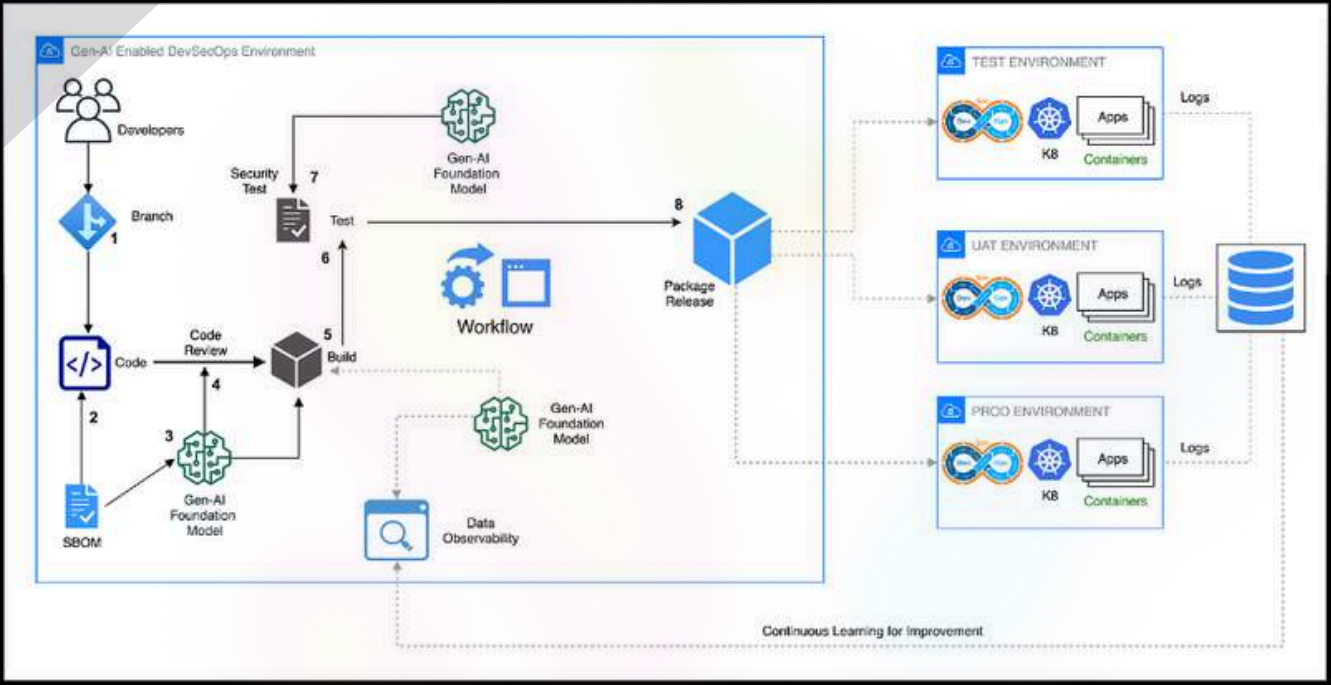
```
Feature: ModSecurity Virtual Patching Capabilities
Scenario: Dynamic Vulnerability Protection
  Given a web application with known vulnerabilities
  When ModSecurity implements virtual patch
  Then the system should:
    - Detect and block potential exploit attempts
    - Apply rules without application code modifications
    - Support custom rule creation
    - Provide real-time threat detection
    - Generate comprehensive security logs
    - Minimize false positive rates
```



03 AI and LLM in DevSecOps



AI and LLM in DevSecOps



AI and large language models (LLMs) are transforming DevSecOps by automating complex tasks, enhancing security practices, and improving collaboration between development, operations, and security teams. Here's how AI and LLMs contribute across the DevSecOps lifecycle:

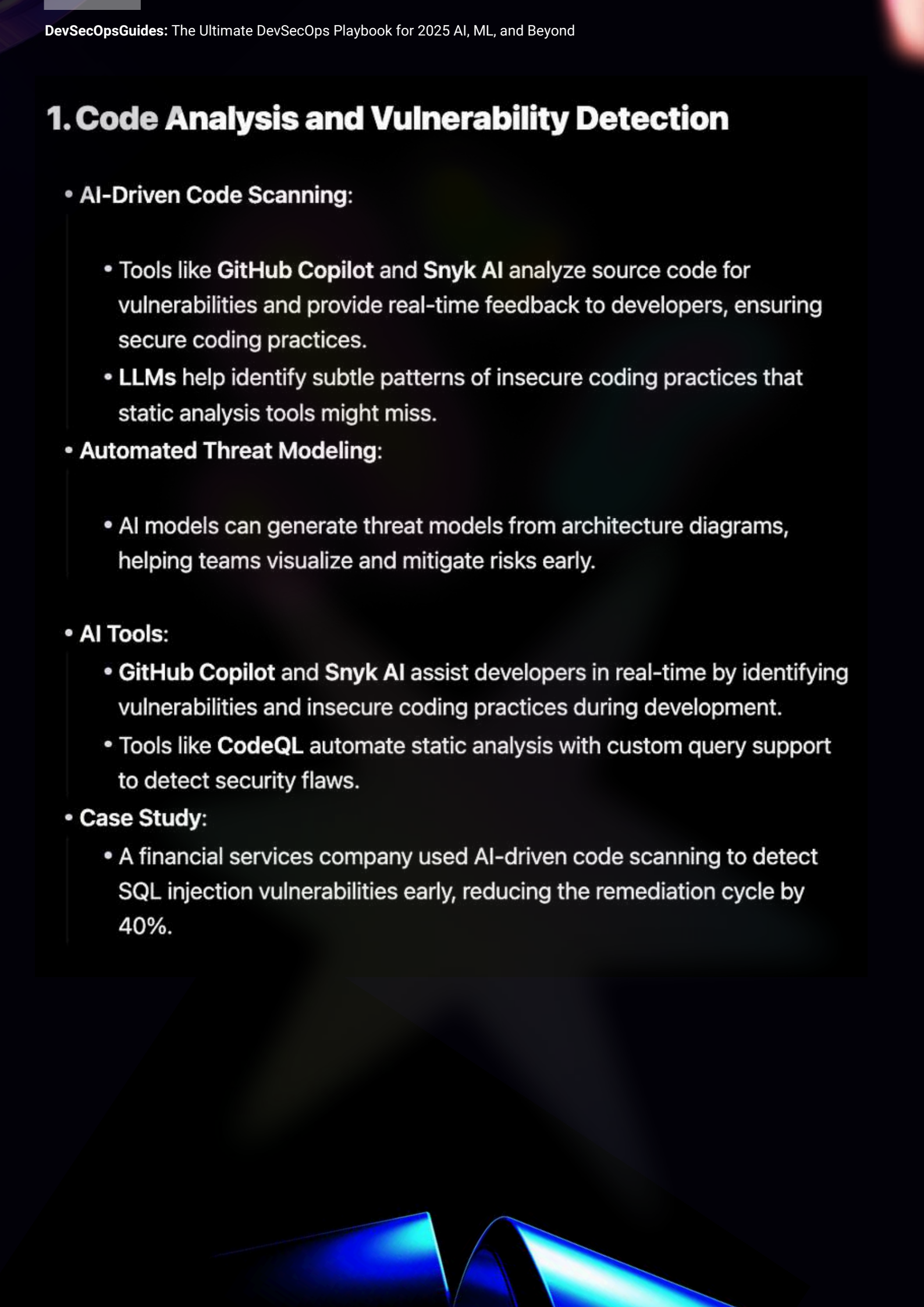
Example Playbook

Here's how AI tools are integrated into a sample DevSecOps pipeline:

16. **Code Analysis**: Use **GitHub Copilot** for secure coding practices and **CodeQL** for static analysis during the commit stage.
17. **Build Optimization**: Employ **Jenkins AI Plugin** to predict failures in CI.
18. **Testing**: Run DAST scans with **Burp Suite AI** and container security with **Trivy**.
19. **Threat Detection**: Deploy **Darktrace** for real-time monitoring of system logs and behavior.
20. **Incident Response**: Automate responses using **Cortex XSOAR** playbooks.
21. **Documentation**: Use **Drata AI** to auto-generate compliance documentation post-release.

1. Code Analysis and Vulnerability Detection

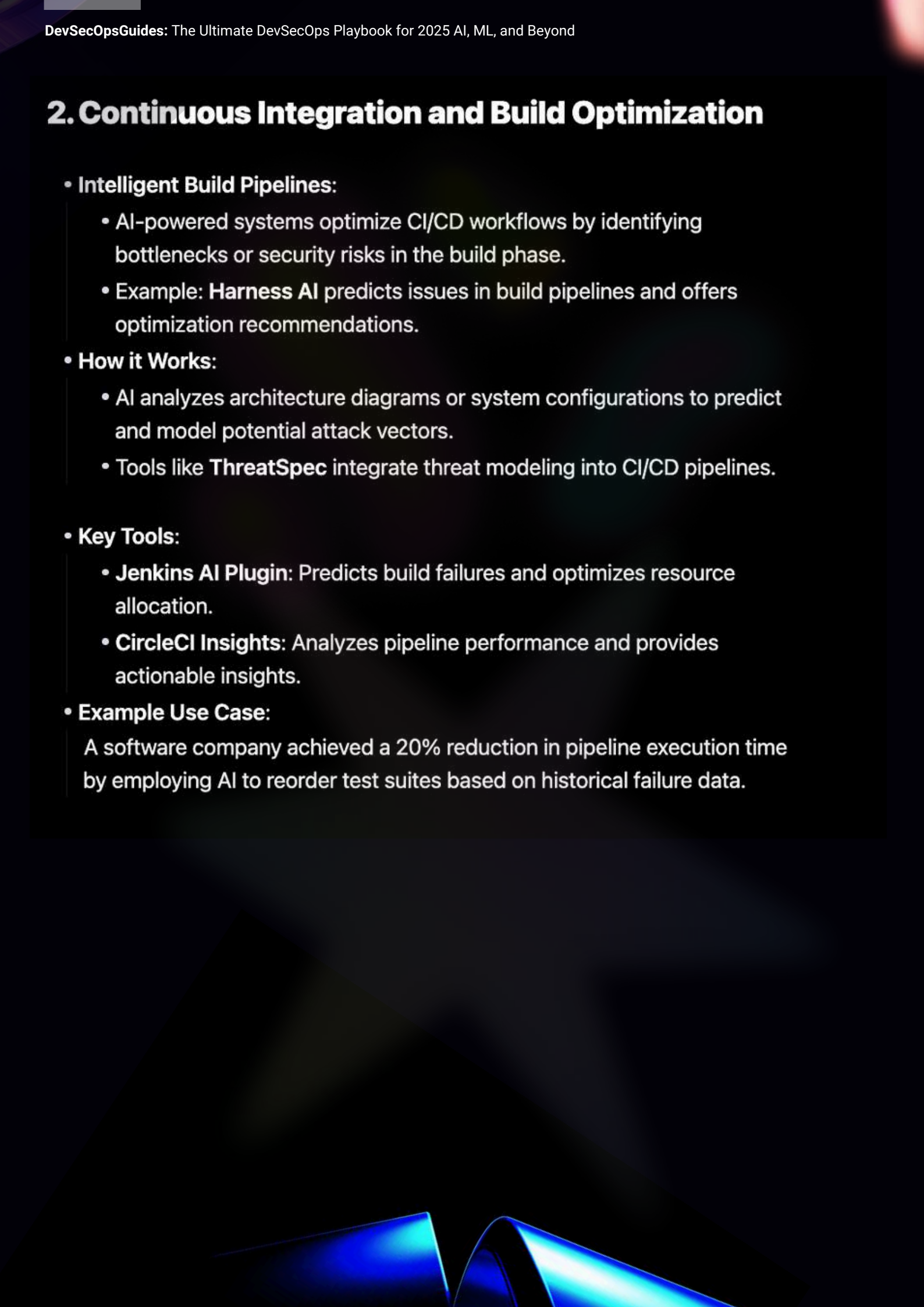
- **AI-Driven Code Scanning**:
  - Tools like **GitHub Copilot** and **Snyk AI** analyze source code for vulnerabilities and provide real-time feedback to developers, ensuring secure coding practices.
  - **LLMs** help identify subtle patterns of insecure coding practices that static analysis tools might miss.
- **Automated Threat Modeling**:
  - AI models can generate threat models from architecture diagrams, helping teams visualize and mitigate risks early.
- **AI Tools**:
  - **GitHub Copilot** and **Snyk AI** assist developers in real-time by identifying vulnerabilities and insecure coding practices during development.
  - Tools like **CodeQL** automate static analysis with custom query support to detect security flaws.
- **Case Study**:
  - A financial services company used AI-driven code scanning to detect SQL injection vulnerabilities early, reducing the remediation cycle by 40%.



2. Continuous Integration and Build Optimization

- **Intelligent Build Pipelines:**
  - AI-powered systems optimize CI/CD workflows by identifying bottlenecks or security risks in the build phase.
  - Example: **Harness AI** predicts issues in build pipelines and offers optimization recommendations.
- **How it Works:**
  - AI analyzes architecture diagrams or system configurations to predict and model potential attack vectors.
  - Tools like **ThreatSpec** integrate threat modeling into CI/CD pipelines.
- **Key Tools:**
  - **Jenkins AI Plugin**: Predicts build failures and optimizes resource allocation.
  - **CircleCI Insights**: Analyzes pipeline performance and provides actionable insights.
- **Example Use Case:**

  A software company achieved a 20% reduction in pipeline execution time by employing AI to reorder test suites based on historical failure data.



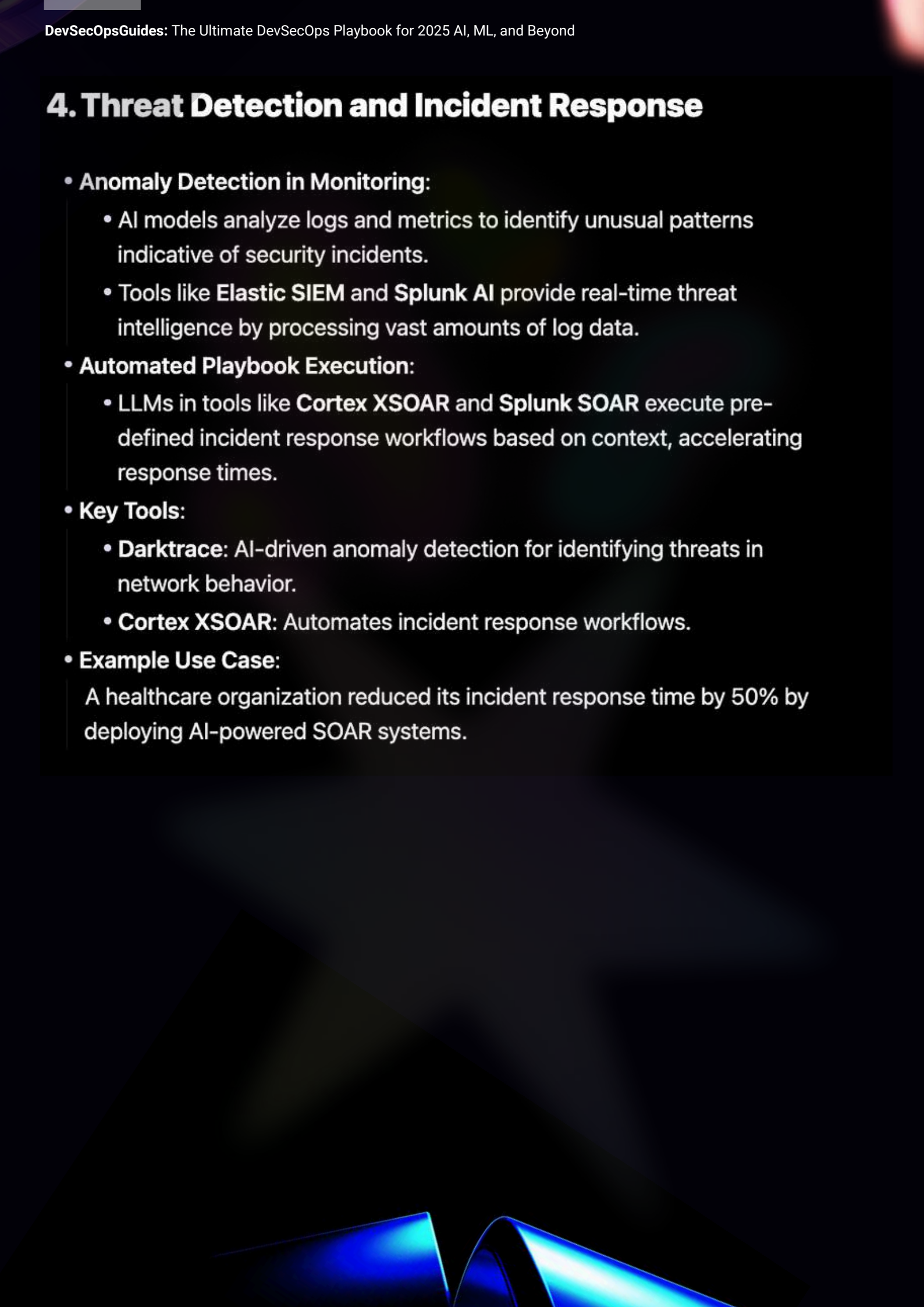
3. Testing Enhancements

- **AI-Augmented Dynamic Application Security Testing (DAST)**:
  - AI tools simulate sophisticated attack patterns to test runtime vulnerabilities.
  - LLMs generate realistic malicious payloads to test web applications against OWASP Top 10 risks.
- **Automated Test Case Generation**:
  - LLMs like OpenAI Codex can create test cases based on functional and non-functional requirements, ensuring comprehensive test coverage.
- **Dynamic Application Security Testing (DAST)**:
  - Tools like **Aqua Security Trivy** and **Burp Suite AI** adaptively test applications based on historical vulnerabilities.
  - AI can enhance fuzz testing by generating context-aware inputs to stress-test systems.
- **Case Study**:
  - An e-commerce platform employed AI in DAST, achieving 35% faster test cycles and 20% higher defect detection.
- **Key Tools**:
  - **Burp Suite AI**: Augments vulnerability scanning by learning attack patterns.
  - **Trivy**: Uses AI to scan containers and IaC for misconfigurations.
- **Example Use Case**:

  An e-commerce platform reduced false positives in security testing by 25% using machine-learning-enhanced fuzzing tools.

4. Threat Detection and Incident Response

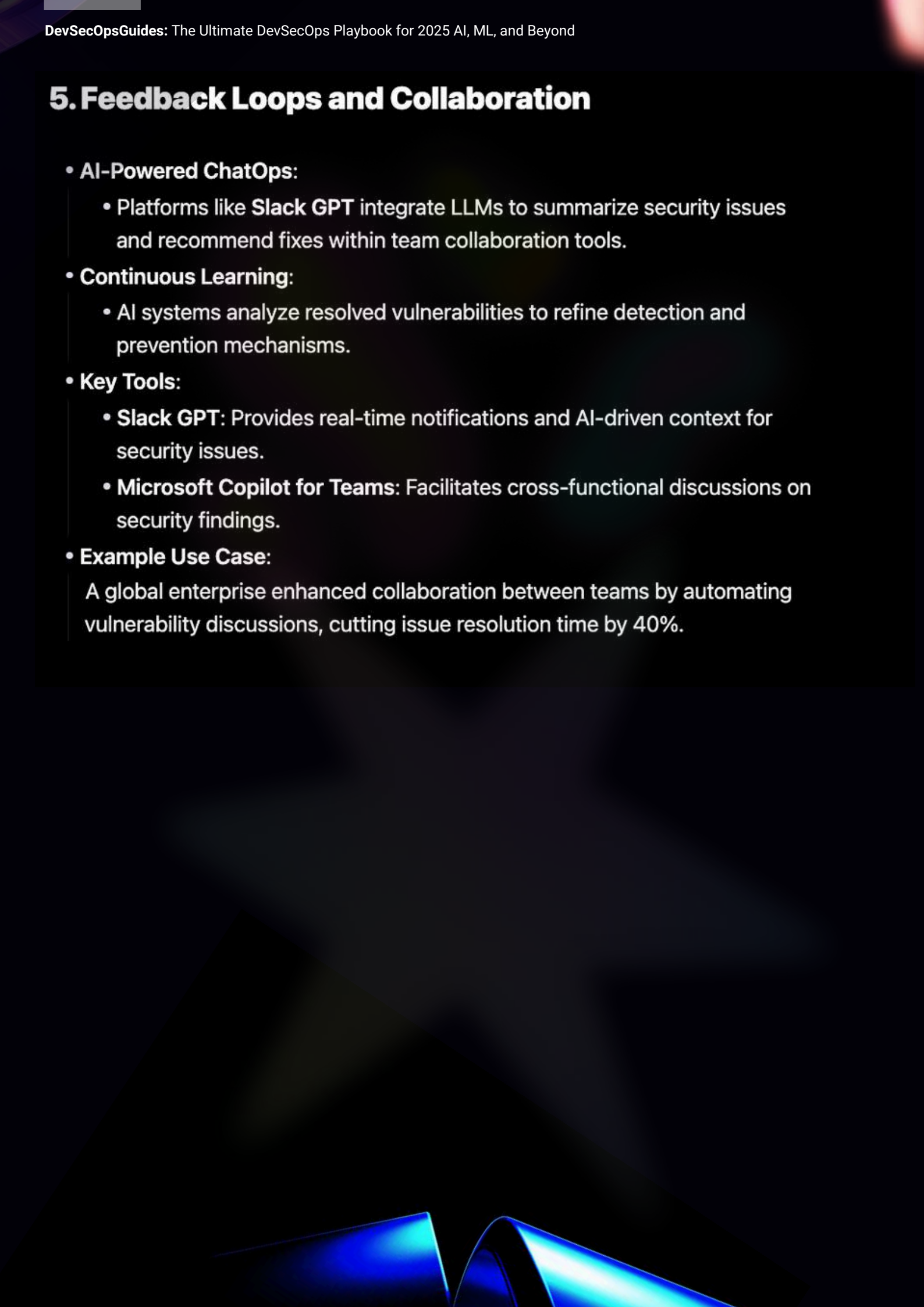
- **Anomaly Detection in Monitoring**:
  - AI models analyze logs and metrics to identify unusual patterns indicative of security incidents.
  - Tools like **Elastic SIEM** and **Splunk AI** provide real-time threat intelligence by processing vast amounts of log data.
- **Automated Playbook Execution**:
  - LLMs in tools like **Cortex XSOAR** and **Splunk SOAR** execute pre-defined incident response workflows based on context, accelerating response times.
- **Key Tools**:
  - **Darktrace**: AI-driven anomaly detection for identifying threats in network behavior.
  - **Cortex XSOAR**: Automates incident response workflows.
- **Example Use Case**:
  A healthcare organization reduced its incident response time by 50% by deploying AI-powered SOAR systems.



5. Feedback Loops and Collaboration

- **AI-Powered ChatOps**:
  - Platforms like **Slack GPT** integrate LLMs to summarize security issues and recommend fixes within team collaboration tools.
- **Continuous Learning**:
  - AI systems analyze resolved vulnerabilities to refine detection and prevention mechanisms.
- **Key Tools**:
  - **Slack GPT**: Provides real-time notifications and AI-driven context for security issues.
  - **Microsoft Copilot for Teams**: Facilitates cross-functional discussions on security findings.
- **Example Use Case**:

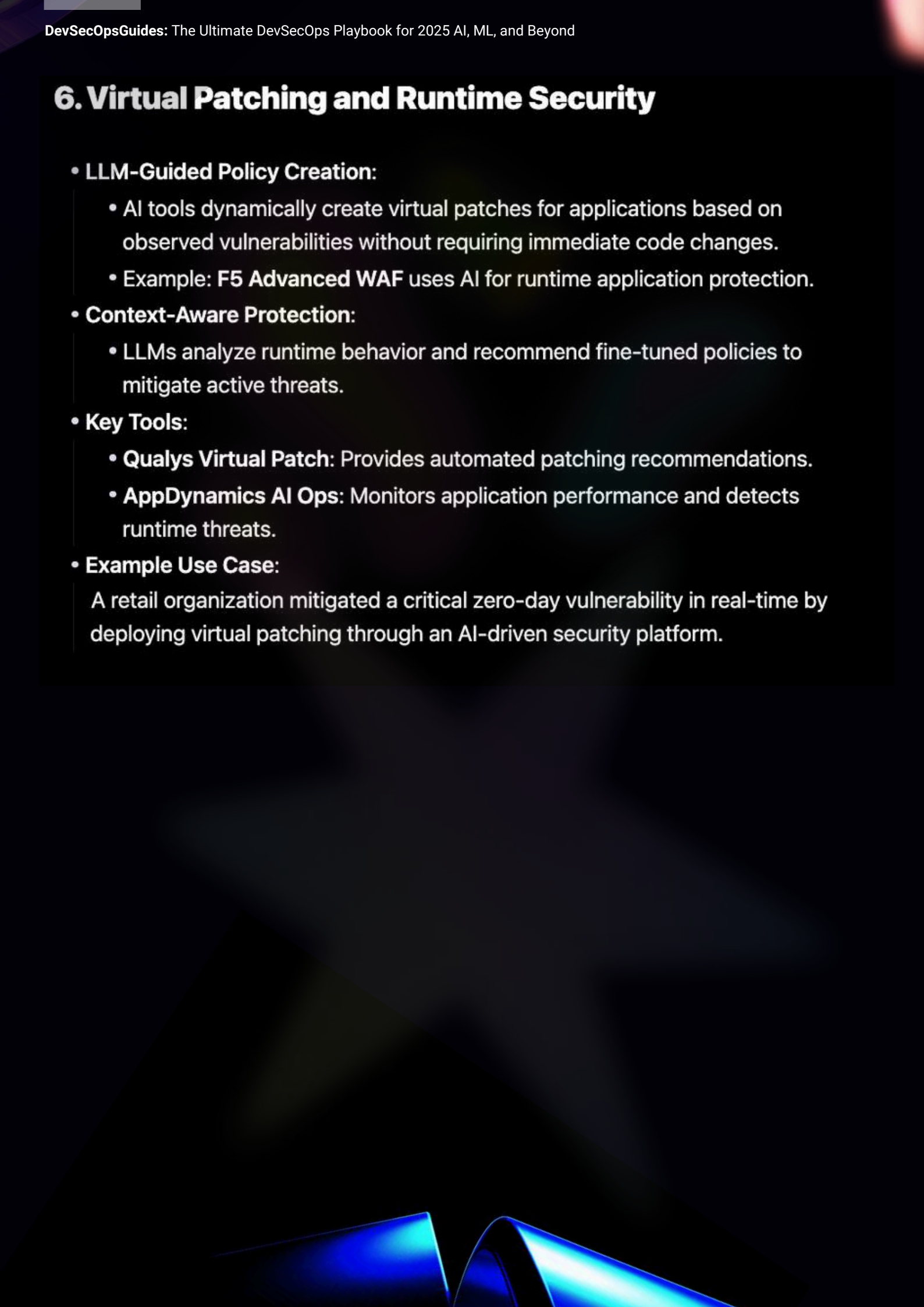
  A global enterprise enhanced collaboration between teams by automating vulnerability discussions, cutting issue resolution time by 40%.



6. Virtual Patching and Runtime Security

- **LLM-Guided Policy Creation:**
  - AI tools dynamically create virtual patches for applications based on observed vulnerabilities without requiring immediate code changes.
  - Example: **F5 Advanced WAF** uses AI for runtime application protection.
- **Context-Aware Protection:**
  - LLMs analyze runtime behavior and recommend fine-tuned policies to mitigate active threats.
- **Key Tools:**
  - **Qualys Virtual Patch**: Provides automated patching recommendations.
  - **AppDynamics AI Ops:** Monitors application performance and detects runtime threats.
- **Example Use Case:**

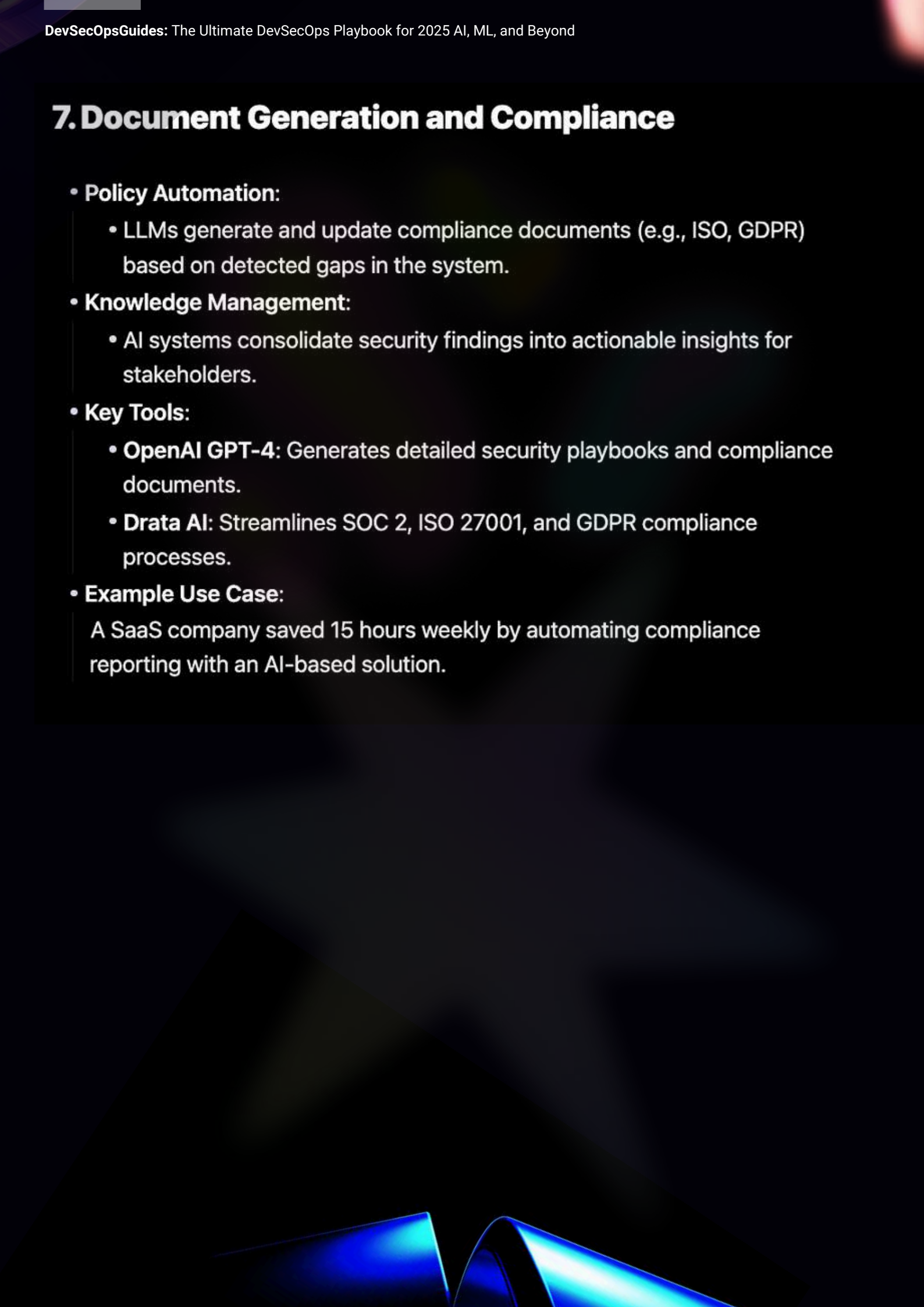
  A retail organization mitigated a critical zero-day vulnerability in real-time by deploying virtual patching through an AI-driven security platform.



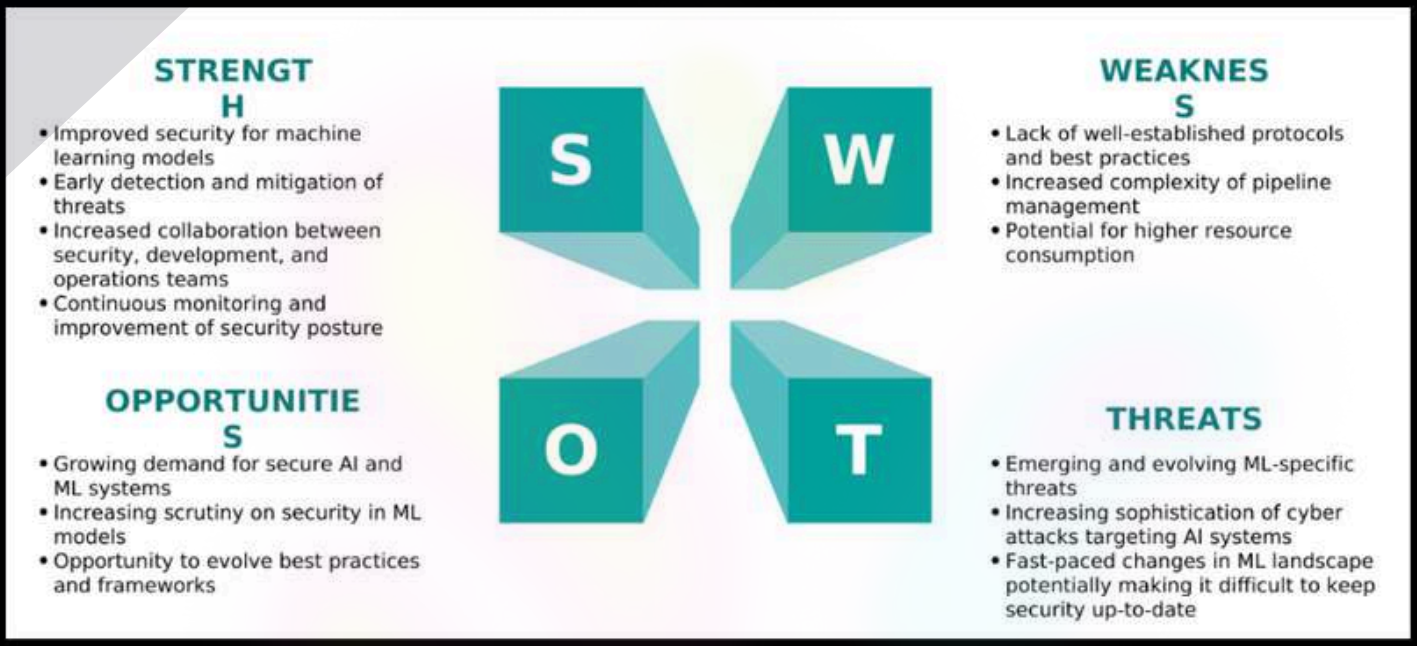
7. Document Generation and Compliance

- **Policy Automation**:
  - LLMs generate and update compliance documents (e.g., ISO, GDPR) based on detected gaps in the system.
- **Knowledge Management**:
  - AI systems consolidate security findings into actionable insights for stakeholders.
- **Key Tools**:
  - **OpenAI GPT-4**: Generates detailed security playbooks and compliance documents.
  - **Drata AI**: Streamlines SOC 2, ISO 27001, and GDPR compliance processes.
- **Example Use Case**:

  A SaaS company saved 15 hours weekly by automating compliance reporting with an AI-based solution.

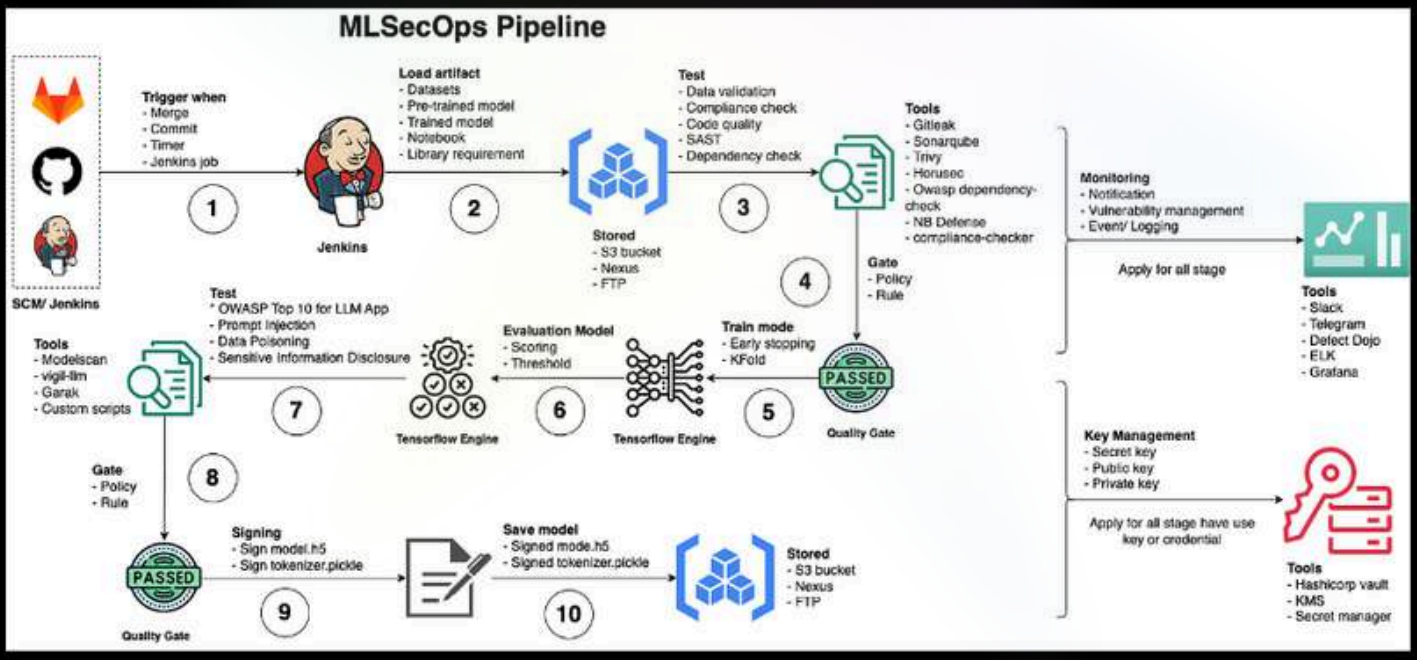


MLsecOps in DevSecOps

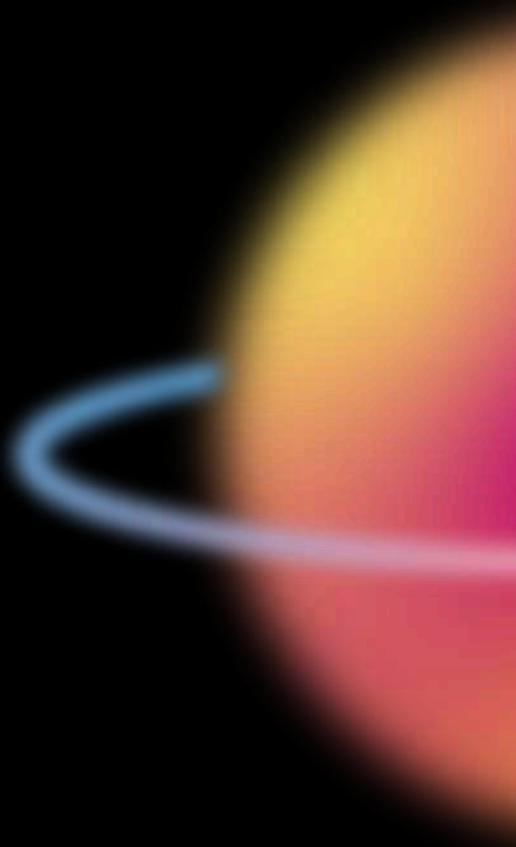


MLSecOps is an emerging discipline that integrates security principles directly into the machine learning lifecycle, addressing the unique security challenges posed by AI and machine learning systems. It extends traditional DevSecOps practices to specifically handle the complex security requirements of machine learning pipelines.

Example Pipeline

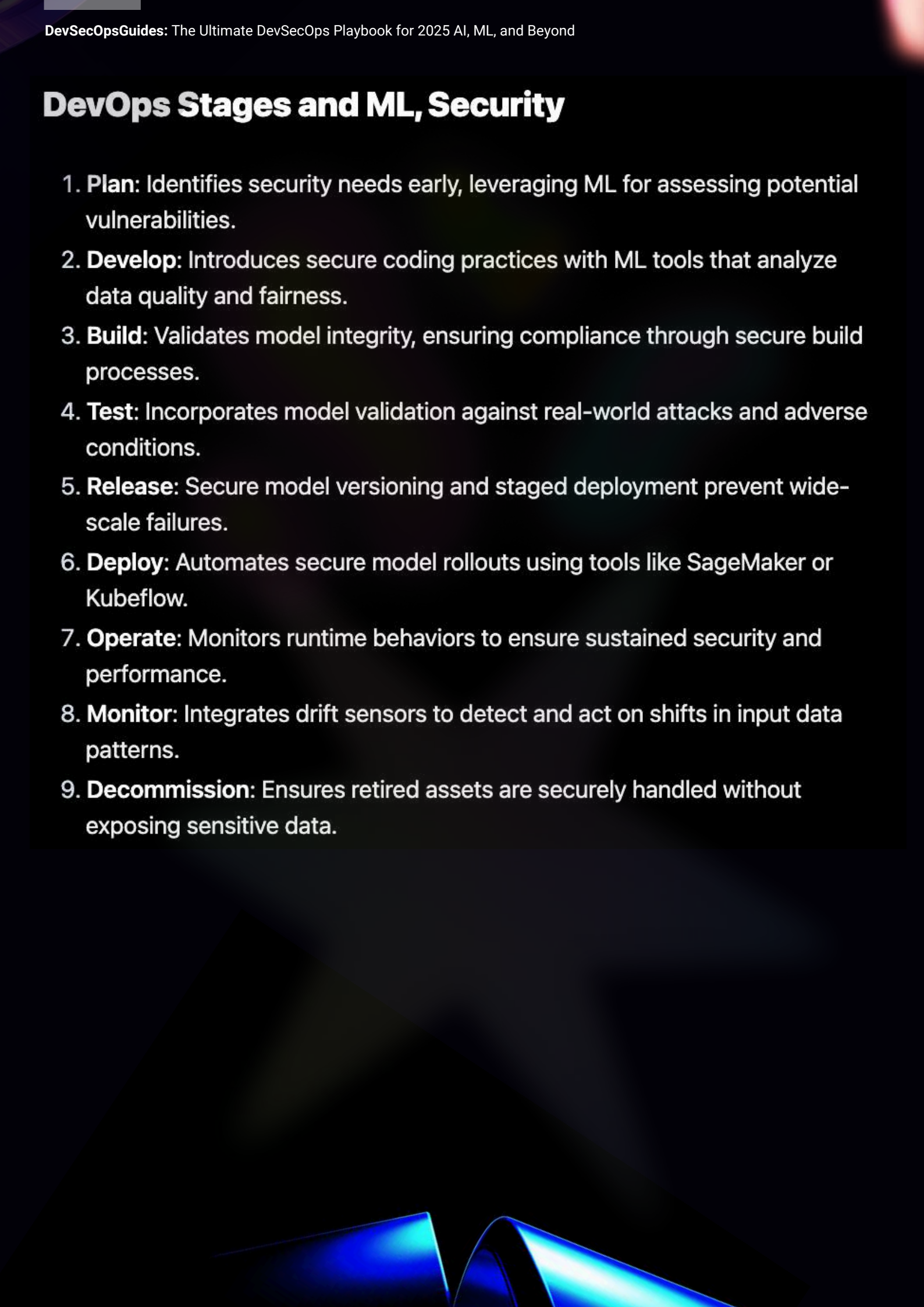


04 MLsecOps in DevSecOps



DevOps Stages and ML, Security

1. **Plan**: Identifies security needs early, leveraging ML for assessing potential vulnerabilities.
2. **Develop**: Introduces secure coding practices with ML tools that analyze data quality and fairness.
3. **Build**: Validates model integrity, ensuring compliance through secure build processes.
4. **Test**: Incorporates model validation against real-world attacks and adverse conditions.
5. **Release**: Secure model versioning and staged deployment prevent wide-scale failures.
6. **Deploy**: Automates secure model rollouts using tools like SageMaker or Kubeflow.
7. **Operate**: Monitors runtime behaviors to ensure sustained security and performance.
8. **Monitor**: Integrates drift sensors to detect and act on shifts in input data patterns.
9. **Decommission**: Ensures retired assets are securely handled without exposing sensitive data.



MLFlow - Detecting Data Drift

Use Case: Monitor and alert for data drift in incoming data.

Dataset: Synthetic Credit Card Fraud Dataset

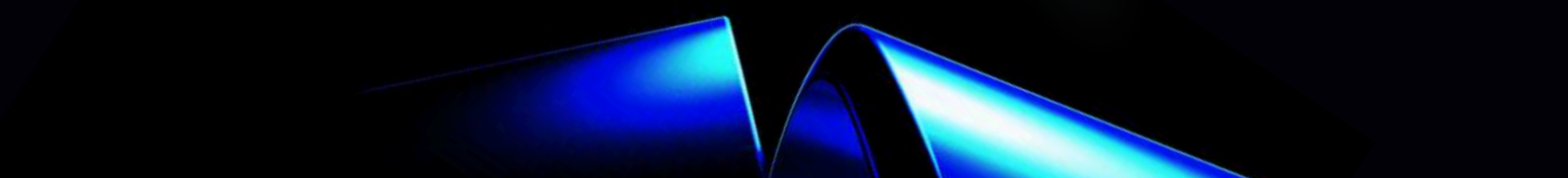
```
import mlflow
from evidently import ColumnMapping
from evidently.model_profile import Profile
from evidently.model_profile.sections import
DataDriftProfileSection

# Load data
import pandas as pd
reference_data = pd.read_csv('reference.csv')
current_data = pd.read_csv('current.csv')

# Data drift detection
profile = Profile(sections=[DataDriftProfileSection()])
profile.calculate(reference_data, current_data)
drift_report = profile.json()

# Log drift results
mlflow.log_text(drift_report, "data_drift.json")
print("Drift ction complete and logged.")
```

AWS SageMaker Studio - Automatic Security Testing

Use Case: Run security tests for ML models before deployment.

Dataset: Public Iris Dataset

```
from sagemaker.model_monitor import DefaultModelMonitor

monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type="ml.m5.large"
)

monitor.create_monitoring_schedule(
    endpoint_input="my-endpoint",
    schedule_cron_expression="cron(0 * ? * * *)",
    output_s3_uri="s3://my-bucket/monitoring"
)
print("Security monitoring scheduled.")
```



Kubeflow - Role-Based Access Control (RBAC) for Pipelines

Use Case: Secure pipelines by enforcing RBAC.
Dataset: CIFAR-10 Dataset

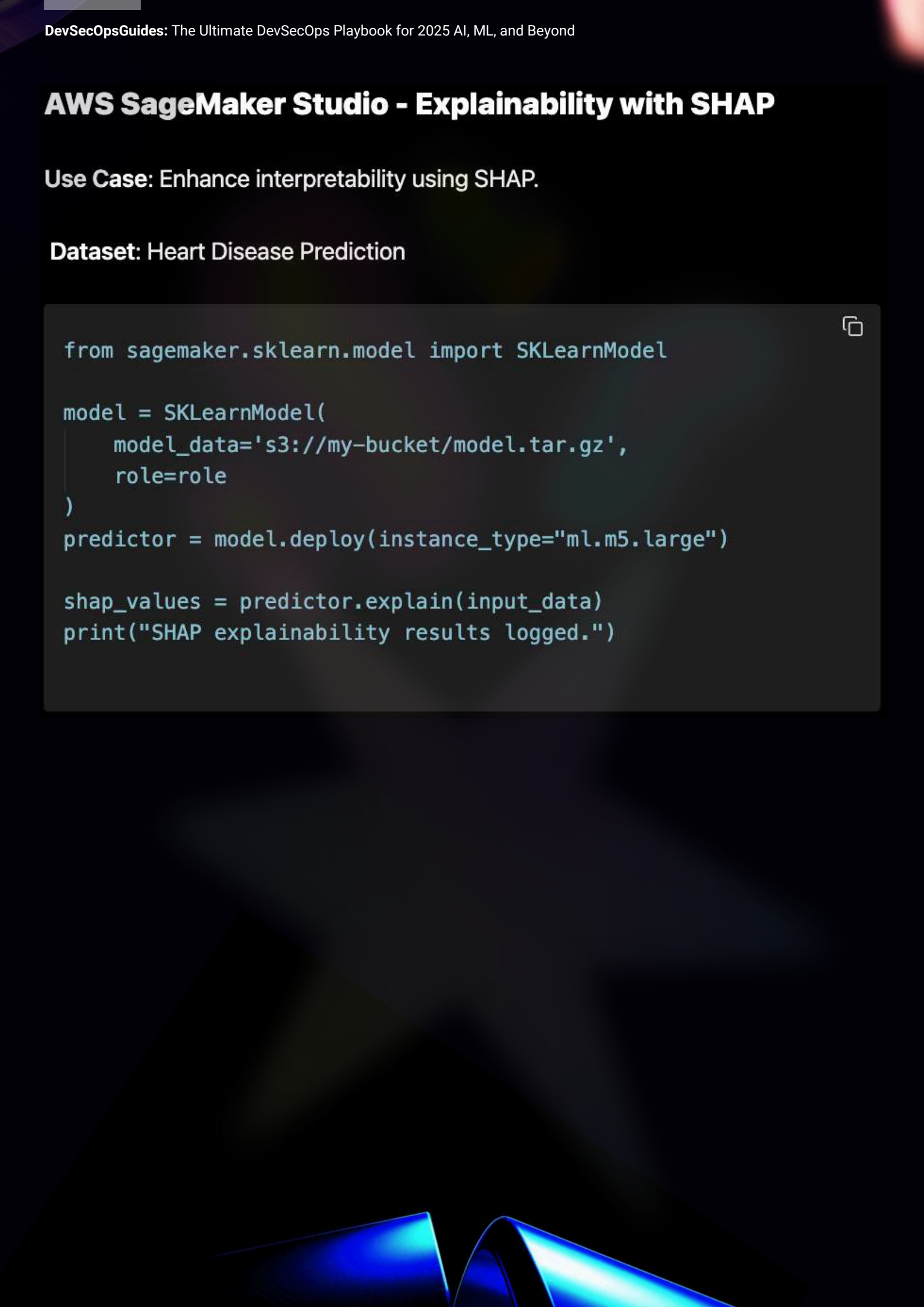
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pipeline-executor
rules:
- apiGroups: [""]
  resources: ["pods", "secrets"]
  verbs: ["create", "get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pipeline-binding
subjects:
- kind: User
  name: pipeline-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pipeline-executor
  apiGroup: rbac.authorization.k8s.io
```

AWS SageMaker Studio - Explainability with SHAP

Use Case: Enhance interpretability using SHAP.

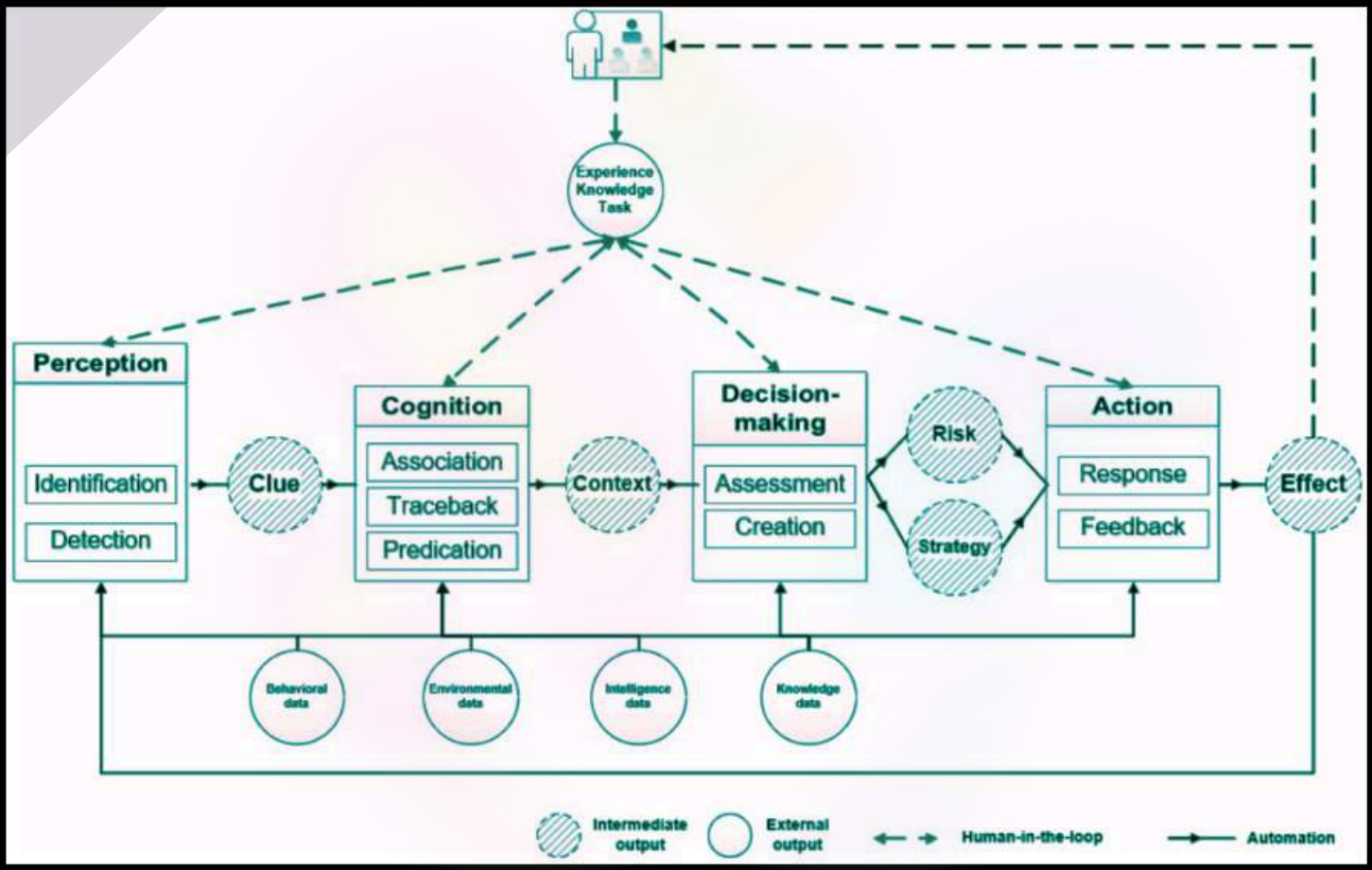
Dataset: Heart Disease Prediction

```
from sagemaker.sklearn.model import SKLearnModel

model = SKLearnModel(
    model_data='s3://my-bucket/model.tar.gz',
    role=role
)
predictor = model.deploy(instance_type="ml.m5.large")

shap_values = predictor.explain(input_data)
print("SHAP explainability results logged.")
```



05 AIsecOps in DevSecOps

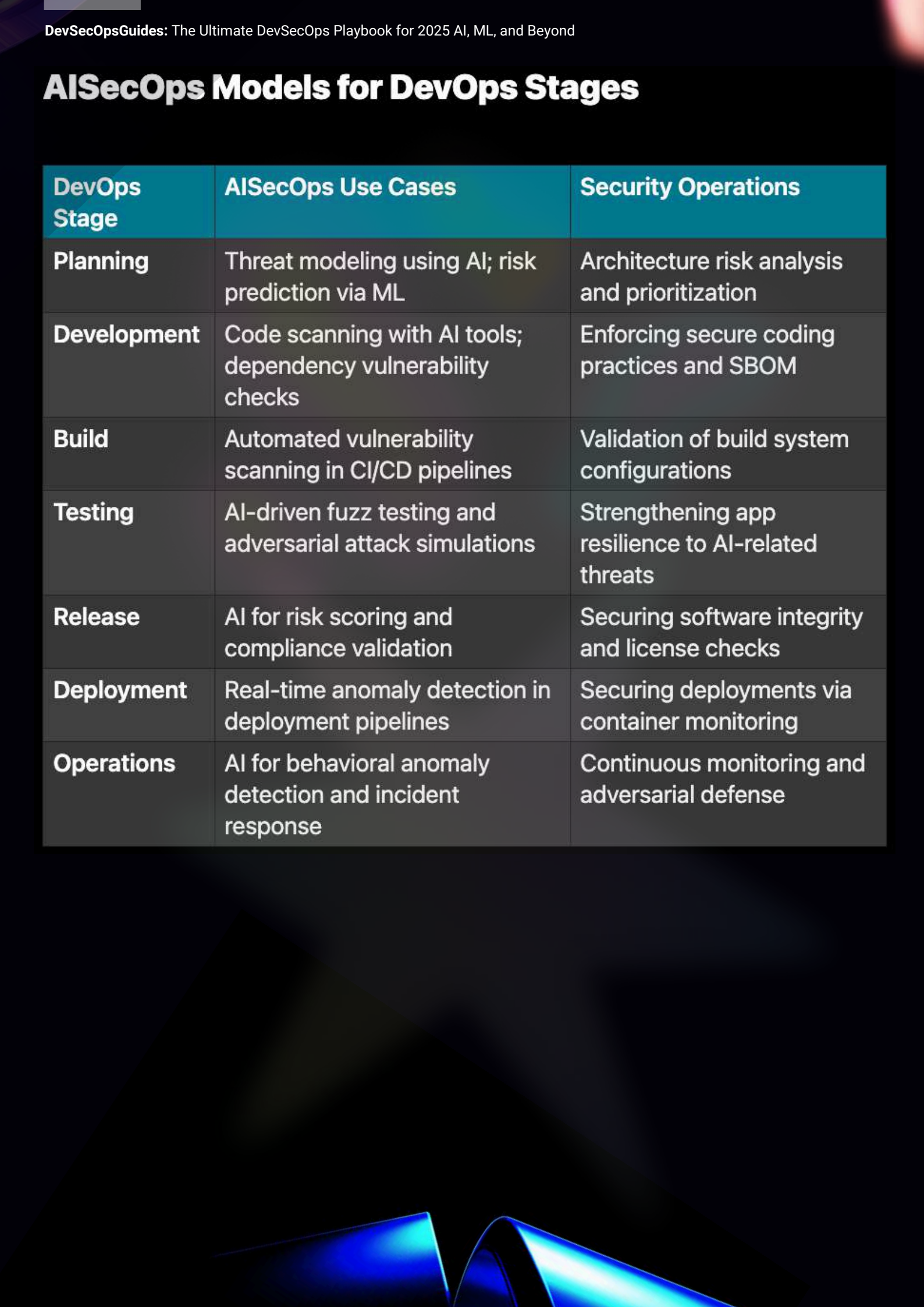
AIsecOps in DevSecOps



AISecOps integrates artificial intelligence into DevSecOps to enhance security throughout the software development lifecycle (SDLC). It leverages machine learning (ML) and AI-driven tools for automation, anomaly detection, predictive risk assessment, and real-time monitoring. This synergy strengthens the secure delivery of applications in dynamic DevOps environments.

AISecOps Models for DevOps Stages

DevOps Stage	AISecOps Use Cases	Security Operations
Planning	Threat modeling using AI; risk prediction via ML	Architecture risk analysis and prioritization
Development	Code scanning with AI tools; dependency vulnerability checks	Enforcing secure coding practices and SBOM
Build	Automated vulnerability scanning in CI/CD pipelines	Validation of build system configurations
Testing	AI-driven fuzz testing and adversarial attack simulations	Strengthening app resilience to AI-related threats
Release	AI for risk scoring and compliance validation	Securing software integrity and license checks
Deployment	Real-time anomaly detection in deployment pipelines	Securing deployments via container monitoring
Operations	AI for behavioral anomaly detection and incident response	Continuous monitoring and adversarial defense



Resources

- **Books:**
  - The Phoenix Project: Devops and the Three Ways to Deliver Business Value - Gene Kim, Kevin Behr, George Spafford
  - Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations - Nicole Forsgren, Jez Humble, Gene Kim
  - Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation - Jez Humble, David Farley
  - Site Reliability Engineering: How Google Runs Production Systems - Betsy Beyer, Chris Jones, Jennifer Petoff
  - The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations - Gene Kim, Patrick Debois, John Willis, Jez Humble
  - ⭐ Crafting Secure Software: An engineering leader's guide to security by design Thomas Segura, Greg Bulmash

- **Websites & Blogs**
  - DevOps.com
  - DZone DevOps
  - The New Stack
  - InfoQ DevOps
  - CNCF (Cloud Native Computing Foundation)

- ***Online Courses & Certifications***
  - DevOps Engineer Nanodegree - Udacity
  - Professional DevOps Engineer - Google Cloud
  - DevOps Engineer - AWS
  - Azure DevOps Engineer Expert - Microsoft
  - DevOps Foundation Certification - DevOps Institute

Conclusion

As organizations navigate the rapidly evolving digital landscape, AI, ML, and automation are redefining the future of DevSecOps. The integration of intelligent security solutions, adaptive risk management, and automated threat detection ensures that security is no longer a bottleneck but a seamless enabler of innovation. By embedding AI-powered security measures across the SDLC, organizations can proactively detect, prevent, and respond to threats before they escalate, strengthening overall resilience against cyber adversaries.

However, the journey to next-generation DevSecOps is not just about technology—it requires a cultural shift, continuous education, and collaboration between security, development, and operations teams. The ability to automate security policies, enforce compliance in real-time, and integrate security testing into CI/CD pipelines is now essential for maintaining both agility and security. DevSecOps leaders, CISOs, and security engineers must embrace AI-driven analytics, predictive security modeling, and proactive risk assessments to stay ahead of threats in an increasingly complex attack landscape.

Looking ahead, organizations that invest in AI-driven security, scalable automation, and intelligent threat modeling will lead the way in building secure, resilient, and high-performing digital ecosystems. The Ultimate DevSecOps Playbook for 2025 serves as a blueprint for navigating this transformation, offering actionable strategies, real-world insights, and industry best practices. By adopting the principles outlined in this guide, security teams can future-proof their security posture, accelerate secure software development, and drive continuous innovation in an AI-powered world. The time to act is now—embrace the future of DevSecOps today! 🚀


Website:
WWW.DEVSECOPSGUIDES.COM