AI-Powered DevSecOps Platform

Y Siva Narayana[1], M Sasidhar[2], M G Jagadeesh[3]
[1,2,3] *Rmk Engineering College*

***Abstract*—Getting software out the door quickly without cutting corners on security is something every development team wrestles with, and most DevSecOps setups today make it harder than it needs to be — scattered tools, steep costs, and security checks that show up too late to actually help. This paper describes an AI-Powered DevSecOps SaaS Platform built to pull those pieces together — CI/CD automation, security checks from the start, and smarter vulnerability ranking, all in one place. Rather than flagging everything equally, the platform uses AI to evaluate each issue in context and sort it by actual risk.**

Running on Python/Go backends, a React frontend, and Kubernetes, the system is built to scale without requiring a dedicated security team to operate it. Startups and smaller companies get access to the same level of DevSecOps maturity that enterprises typically rely on expensive specialists to maintain.

Testing shows teams can fix issues faster, see their risk exposure more clearly, and spend less effort managing security operations overall. The framework shows it's possible to move quickly and stay secure without treating those two goals as opposites.

***Index Terms*—Supply Chain Management, Artificial Intelligence, Machine Learning, Internet of Things, Predictive Analytics, Real-Time Monitoring, Reinforcement Learning, Data Fusion, Risk Mitigation, Operational Resilience.**

I. INTRODUCTION

Software development has shifted dramatically over the past decade. Cloud infrastructure, microservices, and continuous delivery have become the norm, and teams are now expected to ship faster than ever while keeping everything secure and stable. CI/CD pipelines are what make that pace possible — but as release frequency climbs, so does the number of ways an application can be attacked.

DevOps has gone mainstream, but security integration hasn't kept pace. DevSecOps was meant to fix this by weaving security into the pipeline from the start. In practice, though, security still tends to get pushed to the end — scans run late, findings pile up, and remediation gets delayed because someone has to manually sort through results without always knowing what to prioritize.

Smaller teams have it even harder. Startups and SMEs are usually working with limited infrastructure, tight budgets, and no dedicated security person. Their pipelines end up as a patchwork of separate tools — one for version control, one for builds, another for scanning — and keeping all of that in sync creates more overhead than most small teams can afford.

On top of that, most tools dump a long list of vulnerabilities without helping teams figure out which ones actually matter. CVSS scores give a rough sense of severity, but they don't tell you whether something is exploitable in your specific environment, how often the affected code runs, or what the real business consequences would be. Teams end up chasing low-priority issues while the more dangerous ones sit unresolved.

This combination — pressure to ship fast, limited resources, and security that never quite fits into the workflow — leaves a real gap. What's needed is something that brings security in early, makes risk assessment automatic, and keeps deployment decisions manageable without becoming a bottleneck.

In designing a solution to these challenges, this project proposes an AI-Powered DevSecOps SaaS Platform designed to unify CI/CD automation, intelligent security analysis, and risk-based deployment control within a centralized framework. The platform embeds security checks throughout the SDLC, leverages artificial intelligence to prioritize vulnerabilities based on contextual risk, and orchestrates deployments using containerized microservices and Kubernetes infrastructure.

Automated pipelines, smart risk scoring, policy-driven gates, and live compliance tracking work together to reduce how much manual effort security requires while actually improving outcomes. Because it's

delivered as a SaaS product, teams without large budgets or specialized staff can still access capabilities that would normally require enterprise-scale investment.

The goal, ultimately, is to make speed and security work together rather than against each other — for organizations of any size, without the cost or complexity that has historically made serious DevSecOps out of reach.

A. Background and Motivation

The way software gets built and delivered has changed more in the past five years than in the two decades before that. Cloud infrastructure, container-based architectures, and continuous delivery pipelines have reset expectations — teams ship faster now, and the systems they build are expected to stay secure and reliable while doing it. CI/CD pipelines are what enable that pace, but as release frequency goes up, so does the number of potential vulnerabilities introduced into production.

DevOps practices have spread widely, but security hasn't always made it into the pipeline alongside everything else. DevSecOps was introduced specifically to close that gap — embedding security scanning and validation directly into CI/CD workflows. In practice, though, security still tends to be treated as a checkpoint near the end of the process rather than something built into every stage. Scans happen late, findings land without enough context to act on, and the people responsible for remediation often lack the security background to interpret results quickly.

Smaller companies feel these problems more acutely. Startups and SMEs are typically running lean — limited infrastructure, stretched budgets, no in-house security team to speak of. Their CI/CD setups grow organically, pulling in whatever tool solves the immediate problem, and over time the toolchain itself becomes a management burden. Visibility suffers, integrations break, and security ends up as nobody's primary job.

There's also a quality problem with the data that existing tools produce. Standard severity ratings like CVSS give a broad sense of how dangerous something is, but they don't factor in whether the vulnerability is actually reachable in your environment, how critical the affected component is to your business, or how likely exploitation really is given your setup. The result is that teams spend time on relatively harmless issues while genuinely dangerous ones sit in the backlog.

The combination of release pressure, resource constraints, and security expectations that don't match available tools has created a gap that most organizations are struggling to close. A workable solution needs to bring security into the process from the beginning, make risk assessment automatic and intelligent, and keep the whole thing lightweight enough not to slow teams down.

B. Proposed Solution: AI-Powered DevSecOps SaaS Platform

To tackle these problems head-on, this project introduces an AI-Powered DevSecOps SaaS Platform. This platform is designed to unify CI/CD automation, intelligent security analysis, and risk-based deployment control within a centralised framework. Security checks are embedded throughout the Software Development Lifecycle, leveraging artificial intelligence to prioritise vulnerabilities based on contextual risk. Deployments are orchestrated using containerised microservices and Kubernetes infrastructure.

Automated pipeline management, context-aware risk scoring, policy-driven deployment gates, and live compliance tracking combine to cut down how much manual effort security requires — while actually improving the outcomes. Delivering the platform as SaaS means teams without large infrastructure budgets or dedicated security specialists can still access capabilities that have historically been out of reach.

The underlying goal is straightforward: make it possible for organizations of any size to ship software that's genuinely secure without sacrificing the speed they need to stay competitive.

II. PROBLEM STATEMENT

Secure software delivery has become a priority across the industry, but the tools available to achieve it haven't caught up. Most development environments require teams to manage a separate tool for every function — source control, CI/CD, vulnerability scanning, container security, compliance, monitoring — and none of them talk to each other well. The result is poor visibility, duplicated effort, and pipelines that are harder to operate than they need to be.

Security validation still shows up late in too many workflows — sometimes not until staging, sometimes not until after deployment. By that point, the cost to fix problems has multiplied and the window for exploitation has already opened. Running security checks continuously from the first commit isn't an advanced practice; it's what DevSecOps is supposed to mean.

Most tools stop at detection. They hand developers a report full of findings and leave them to figure out what to do next. Without any sense of which issues are truly urgent or what fixing them actually looks like, teams spend hours interpreting results that could have been triaged automatically. Mean Time To Remediate (MTTR) climbs, and developer time gets spent on security work instead of building.

The platforms that do offer sophisticated DevSecOps capabilities are almost uniformly designed with large enterprises in mind. The configuration complexity alone is enough to deter smaller teams, and the pricing structures assume a security budget that most startups simply don't have. Good security tooling shouldn't require a specialist and a significant financial commitment just to get started.

There's also a communication gap between security data and business decisions. Founders and product managers rarely have the background to interpret technical vulnerability reports, and existing tools don't bridge that gap. Risk information stays inside the security team, disconnected from the people who need to factor it into product and release decisions.

Taken together, these shortcomings point to a clear need: a platform that handles CI/CD automation and continuous security validation in one place, uses AI to make sense of what it finds, and presents the results in a way that's useful to engineers and decision-makers alike — without requiring enterprise resources to run.

.

A**.** Formal Problem Definition

Development teams today are expected to ship features quickly while keeping their applications secure, scalable, and compliant. DevSecOps exists to make that possible by embedding security into CI/CD workflows, but in practice, the available solutions are fragmented, difficult to operate, and built for organizations with far more resources than most startups and growing teams possess. High costs, steep learning curves, and a toolchain that requires constant integration work all combine to make serious DevSecOps feel out of reach.

The scanning tools that do exist tend to generate noise rather than insight — large lists of findings sorted by static severity scores, with no consideration of whether a given vulnerability is actually exploitable in a particular environment, how much it matters to the business, or what the deployment context looks like. Security checks still show up late in most SDLCs, which means problems are found after they're already expensive to fix.

This results in three fundamental gaps:

1. Operational Fragmentation – Lack of unified integration across CI/CD, security scanning, and deployment orchestration.
2. Intelligence Deficiency – Absence of automated, AI-based vulnerability prioritization and remediation recommendations.
3. Accessibility Barrier – Limited affordability and usability for startups and resource-constrained teams.

Therefore, the core problem is the absence of a unified, intelligent, and startup-accessible DevSecOps platform that embeds continuous security throughout the SDLC, automates risk-based deployment decisions, and provides actionable insights aligned with business impact.

Addressing this problem requires designing an integrated AI-powered DevSecOps SaaS architecture capable of reducing remediation latency, simplifying pipeline management, and democratizing enterprise-grade security automation for organizations of all sizes.

B. Limitations of Existing Systems

Despite the growing adoption of DevSecOps practices, existing systems exhibit several structural and operational limitations that hinder effective and scalable secure software delivery.

1. Fragmented Tool Ecosystem

The typical DevSecOps stack today is a collection of individually selected tools that were never designed to work together. Source control, CI/CD, scanning, container security, compliance, and monitoring each live in their own system, requiring custom integrations that break, configurations that drift, and manual effort to correlate results across tools.

2. High Learning Curve and Operational Complexity

The more capable DevSecOps platforms tend to be genuinely difficult to set up and keep running. Managing pipelines, security policies, container orchestration, and compliance requirements simultaneously requires expertise that most small teams don't have on staff, and the learning curve can eat weeks of time that growing companies can't spare.

3. Expensive Licensing and Infrastructure Costs
Pricing is another barrier that rarely gets talked about directly. Serious DevSecOps platforms are designed and priced for enterprise contracts, with licensing structures that assume a dedicated security team and a budget to match. For a startup or a lean engineering team, those costs are simply prohibitive.

4. Late Security Integration (Reactive Approach)
Security scanning that only kicks in during pre-production is a common pattern and a real problem. By the time a vulnerability is found that late in the process, fixing it usually means rework, schedule delays, and in the worst cases, shipping known vulnerabilities and hoping nothing exploits them before a patch is ready.

5. Lack of Intelligent Risk Prioritization
CVSS scores tell you something about a vulnerability in the abstract, but not much about whether it matters in your specific environment. Tools that rank findings purely by static severity end up pushing teams toward spending their time on issues that are theoretically serious but practically low-risk, while potentially dangerous issues in business-critical systems stay buried in the list.

6. Absence of AI-Driven Remediation Guidance
Detection without guidance puts the burden on developers who often don't have deep security expertise. When a tool reports a vulnerability and stops there, whoever owns the fix has to research the issue, figure out the right approach, and implement it — all of which takes time that could be compressed with better tooling.

7. Limited Business-Level Visibility
Technical dashboards built for engineers don't serve the rest of the organization. Founders, product managers, and other decision-makers need to understand security risk in terms of business exposure — and today's tools rarely offer that translation.

8. Poor Startup Accessibility
When platforms are architected for Fortune 500 security teams, smaller organizations are left adapting tools that don't fit their scale, culture, or resource levels. The result is either compromised security practices or teams that simply opt out of adopting DevSecOps at all.

C. Research Challenges
Building a platform like this isn't just an engineering problem — it involves research questions that don't have clean answers yet. Several are worth examining directly.

1. Unified Integration of Heterogeneous Tools
Pulling together diverse CI/CD tools, security scanners, container orchestration systems, and monitoring solutions into one coherent architecture is harder than it looks. Each operates with its own APIs, configurations, and output formats. Designing a standardized integration layer that ensures interoperability without introducing performance bottlenecks is a significant architectural challenge.
2. Context-Aware Risk Quantification
Moving beyond static CVSS scoring requires building a model that accounts for exploit probability, runtime behavior, business impact, and compliance requirements — and that adapts as those factors change. Getting the feature engineering right, keeping the model explainable, and making its outputs trustworthy enough for automated deployment decisions is a genuine research challenge, not just an implementation task.
3. Balancing Automation with Governance
Automation in deployment decisions is powerful, but it needs guardrails. A Policy Decision Engine that can block or approve deployments based on risk scores has to maintain full audit trails, support human override when context demands it, and remain transparent enough that people trust it. Getting that balance right is a design challenge as much as a technical one.
4. Reducing False Positives and Alert Fatigue
Alert fatigue is a real problem in security, and AI systems can make it worse if they're not carefully designed. Building filtering and prioritization that reduces noise without accidentally burying critical

findings requires extensive validation — and getting that wrong in either direction has real consequences.

5. Scalability in Cloud-Native Environments

Supporting teams that run microservices across multiple cloud environments means the platform's own infrastructure has to be genuinely elastic. Risk computation needs to keep up with high-concurrency pipelines without becoming a bottleneck, which creates real infrastructure and orchestration demands.

6. Real-Time Performance Constraints

AI inference that takes too long kills the value proposition. Risk scoring and policy evaluation have to complete within the time budget of a typical pipeline run — which means the models need to be both accurate and fast. Achieving that tradeoff without compromising either is a real engineering and research problem.

7. Data Availability and Model Training

Good training data for vulnerability models is surprisingly hard to come by. Historical security data is often proprietary, incomplete, or heavily biased toward certain types of projects. Critical vulnerabilities are rare by definition, which creates class imbalance that needs careful handling to avoid models that systematically underrate serious issues.

8. Security of the DevSecOps Platform Itself

A security platform that can itself be compromised undermines everything it's meant to protect. Pipeline manipulation, malicious dependency injection, and privilege escalation are real threats to this kind of system, which means secure API design, strict RBAC enforcement, and zero-trust communication between services aren't optional — they're fundamental requirements.

9. Explainability and Trust in AI Decisions

AI-generated deployment decisions need to be explainable before teams will trust them. Stakeholders at every level — developers, security leads, executives — need to be able to understand why a build was blocked or approved. Building transparency into the risk model without sacrificing its sophistication is an active area of research.

10. Adoption by Resource-Constrained Teams

Making a system powerful enough for serious security work while keeping it simple enough for a five-person startup to adopt requires hard tradeoffs in design. A one-click setup and an interface that doesn't require a security background to navigate are real goals — achieving them without hollowing out the platform's capabilities is the challenge.

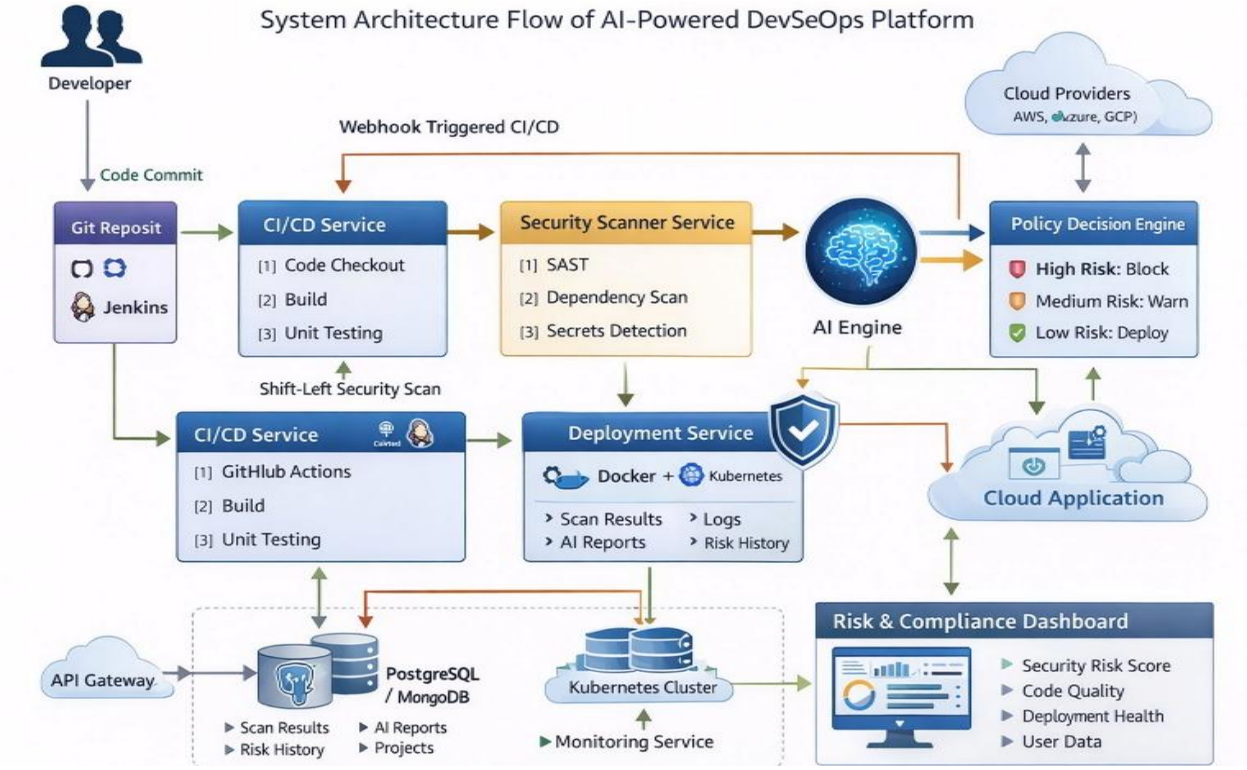

Figure 1: System Architecture Flow of AI-Powered DevSecOps Platform

III. PROPOSED SYSTEM

To address the limitations of existing DevSecOps solutions, this project proposes an AI-Powered Unified DevSecOps SaaS Platform designed to integrate CI/CD automation, intelligent security analysis, risk-based deployment control, and compliance monitoring within a centralized, scalable framework.

The proposed system follows a cloud-native, microservices-based architecture that embeds security throughout the Software Development Lifecycle (SDLC) while maintaining development velocity.

.

A. System Overview

Rather than requiring teams to manage a collection of disconnected tools, the platform brings everything into one dashboard-driven environment. It covers:

- Automated CI/CD pipeline orchestration
- Continuous security validation
- AI-driven vulnerability prioritization
- Policy-based deployment decisions
- Compliance visibility and risk analytics

Security doesn't wait until the end of the pipeline. It runs from the first commit, catching issues when they're cheap to fix rather than after they've made it into production.

B. Architecture Overview

1. Smart CI/CD Pipeline Module

This module takes care of code checkout, building, testing, containerization, and deployment automatically, integrating with Jenkins and GitHub Actions. Every code push kicks off a pipeline run via webhook — no manual steps required.

Key features:

- Automated build and test execution
- Docker image generation
- Kubernetes deployment orchestration
- Real-time pipeline monitoring

2. AI Code & Security Analysis Module

Security scanning runs as part of every pipeline execution, pulling together results from multiple specialized tools:

Static Application Security Testing (SAST)

Dependency vulnerability analysis

Secret detection

- Container security scanning

Results from all scanners are normalized into a consistent format before anything reaches the AI Risk Engine, so downstream analysis doesn't have to deal with the inconsistencies between tools.

3. AI Risk Engine

The AI Risk Engine is the heart of the platform. For each detected vulnerability, it calculates a composite risk score for each detected vulnerability using multiple contextual parameters:

- CVSS severity score
- Exploit probability
- Runtime usage frequency
- Business criticality
- Compliance sensitivity

The composite risk model can be represented as:

$$R = \alpha S + \beta E + \gamma B + \delta F$$

Where:

- S = Severity
- E = Exploit probability
- B = Business impact
- F = Runtime frequency

The weights $\alpha, \beta, \gamma, \delta$ are dynamically configurable. This enables intelligent prioritization rather than static vulnerability listing.

4. Policy Decision Engine

Once the Risk Engine has scored a build's vulnerabilities, the Policy Engine translates those scores into deployment decisions:

- High Risk → Block Deployment
- Medium Risk → Warn with Recommendation
- Low Risk → Allow Deployment

Every decision gets logged with its rationale, giving teams a full audit trail and keeping compliance requirements satisfied automatically.

5. Deployment & Orchestration Layer

The system leverages:

- Docker for containerization
- Kubernetes for orchestration
- Cloud providers (AWS, Azure, GCP) for scalable infrastructure

The deployment layer ensures high availability, fault tolerance, and elastic scaling.

6. Risk & Compliance Dashboard

The React-based dashboard gives different audiences what they need:

- Real-time risk scoring visualization
- Compliance readiness indicators
- Founder-friendly business risk insights
- Historical vulnerability trends
- Deployment health metrics

The interface is deliberately designed for both engineers who want technical depth and business stakeholders who need to understand risk at a higher level.

C. System Workflow

- Developer commits code
- Webhook triggers CI/CD pipeline
- Build and security scans execute
- Vulnerability findings collected
- AI Risk Engine computes contextual risk score
- Policy Decision Engine evaluates risk threshold
- Deployment decision executed
- Dashboard updated with compliance and risk metrics

D. Key Innovations

- Unified DevSecOps architecture
- AI-driven contextual vulnerability prioritization
- Risk-based automated deployment gating
- Startup-friendly SaaS accessibility
- Business-level visibility integration

E. Expected Outcomes

- The proposed system aims to:
- Reduce Mean Time To Remediate (MTTR)
- Minimize alert fatigue
- Lower operational DevSecOps costs
- Improve vulnerability prioritization accuracy
- Democratize enterprise-grade DevSecOps for startup.Top of Form

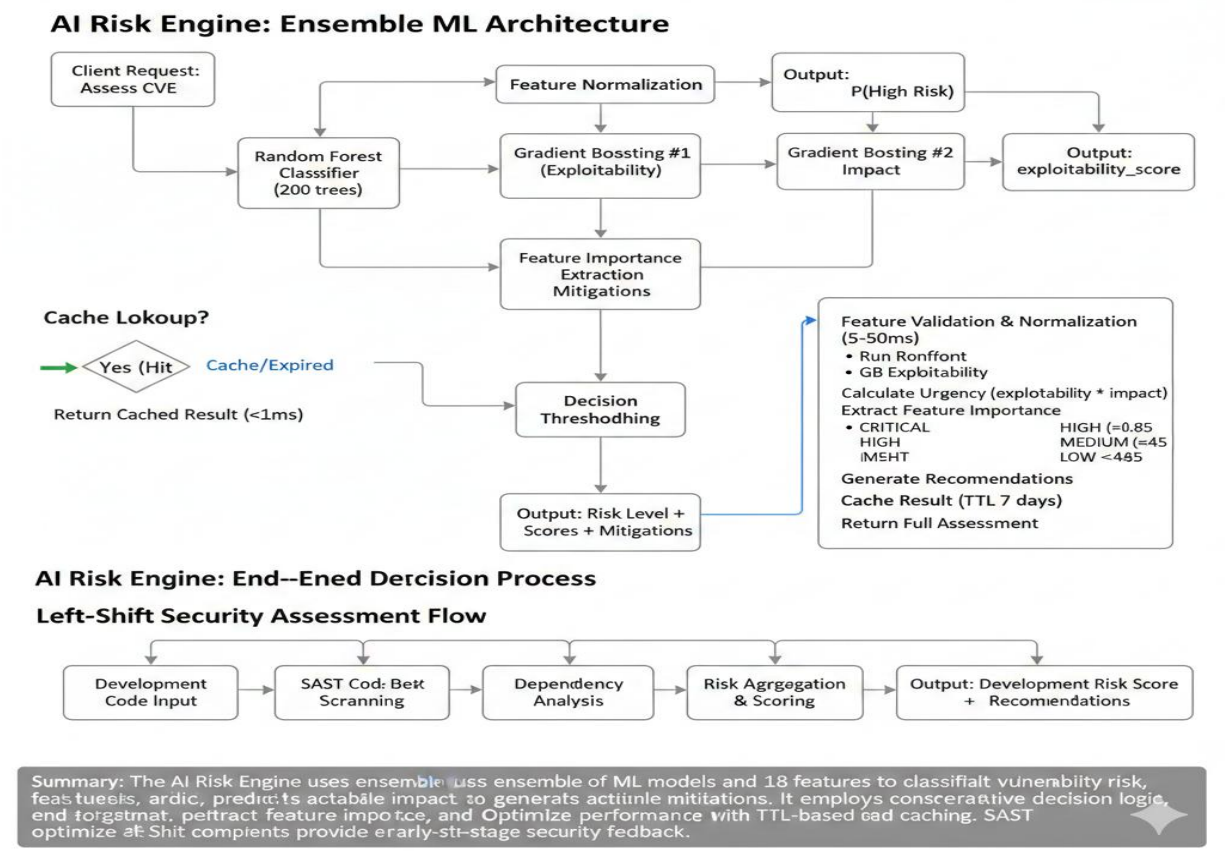

Figure 2: AI Risk Engine – Ensemble ML Architecture and End-to-End Decision Process

IV. AI RISK ENGINE: ALGORITHMS AND DECISION LOGIC

The platform's intelligence is powered by an ensemble of machine learning models working together to assess vulnerability risk. Rather than producing a single static severity score, the engine combines three specialized models — each trained for a different aspect of risk — to generate decisions that reflect real-world context.

4.1 Ensemble ML Architecture

The Risk Engine uses three independent models in sequence. A Random Forest Classifier (200 trees, max_depth=15) handles binary risk classification, outputting P(High Risk) as a probability between 0 and 1. Two separate Gradient Boosting regressors run in parallel: one predicts exploitability (how likely a vulnerability is to be successfully exploited), and the other predicts business impact (the severity of consequences if compromise occurs). Each model is trained on 18 vulnerability features covering severity scores, exploit maturity, patch availability, environmental context, and attack capabilities.

The Random Forest aggregates predictions across all 200 trees: $P(\text{high\_risk}) = (1/200) \times \Sigma(\text{tree\_prediction\_i})$. The Gradient Boosting models use sequential additive learning: $F(x) = \Sigma(\eta \times h\_t(x))$, where $\eta = 0.1$ (learning rate) and $h\_t(x)$ is the prediction from tree t. This sequential correction mechanism allows each new tree to focus on the mistakes of previous ones, improving accuracy on difficult edge cases.

4.2 Feature Engineering: 18-Dimensional Analysis

Each vulnerability is analyzed across 18 dimensions organized into four categories. Severity and exploit status (CVSS score normalized to 0–1, exploit maturity from unproven to widely available, and public exploit availability) provide the baseline risk signal. Patching information captures patch completeness, days until a patch is available, and how long the CVE has been public. Tracking and awareness features measure database mentions, active exploitation campaigns, number of affected components, and whether a public proof-of-concept exists. Finally, attack capability features flag worm propagation potential, privilege escalation possibility, lateral movement capability, and data breach potential — each of which carries a significant risk multiplier.

Environmental context adds further precision: internet exposure (0.0 for internal to 1.0 for public-facing), environment criticality (0.1 for development to 0.9 for production), third-party and supply chain dependency risk. These contextual features ensure the risk score reflects actual deployment conditions rather than abstract severity in isolation.

4.3 Decision Logic and Risk Thresholds

The three model outputs combine to produce an urgency score: urgency = exploitability_score × impact_score. This multiplicative formula reflects a deliberate design choice — a vulnerability is only genuinely urgent when it is both easy to exploit and highly impactful. A vulnerability that is trivially exploitable but has negligible business consequences carries the same low urgency as one that would be catastrophic if exploited but is practically unreachable.

Risk level assignment uses conservative thresholds: CRITICAL at $P \geq 0.85$ (deploy block, patch within 24 hours); HIGH at $P \geq 0.65$ (restricted access, patch within 7 days); MEDIUM at $P \geq 0.45$ (monitor, patch within 30 days); LOW below 0.45 (standard maintenance cycle). The system is designed to err on the side of caution — a false positive blocked deployment is always preferable to a missed threat reaching production.

4.4 SAST Pattern Detection

Static Application Security Testing runs in parallel with the ML risk assessment, scanning source code without execution using multi-pass regex pattern matching. The scanner strips comments and docstrings first, then runs patterns across four vulnerability categories. Hardcoded secrets (passwords, API keys, AWS credentials, private keys) are flagged as CRITICAL under CWE-798. Insecure cryptography — DES ciphers, MD5/SHA1 hashing, weak random number generation — is flagged as HIGH under CWE-327. Code injection patterns covering SQL injection, command injection, and XSS are flagged as HIGH under CWE-89/78/79 respectively. Authentication issues are flagged as MEDIUM under CWE-287.

The left-shift security assessment aggregates SAST findings, dependency vulnerability analysis, and configuration review into a single development risk score: risk = min((critical × 0.10) + (high × 0.05) + (medium × 0.02) + (dependency_vulns × 0.02) +

(config_issues × 0.01), 1.0). Scores below 0.2 pass automatically; scores above 0.7 block the build until critical issues are resolved.

4.5 Performance and Caching

A single ML assessment completes in 5–50ms, SAST file scanning in 15–30ms, and a full left-shift assessment in 100–500ms. To avoid recomputing assessments for the same vulnerability across multiple pipeline runs, the engine uses TTL-based caching keyed on CVE ID, CVSS score, and exploit availability. Cache hits return results in under 1ms. Cache entries expire after 7 days to ensure the risk picture stays current as the threat landscape evolves. With a typical 70% hit rate, the caching layer significantly reduces both latency and compute overhead in high-frequency pipeline environments. Model storage across all three models and their preprocessing pipelines totals approximately 15–25 MB per instance.

System Design & Implementation

The AI-Powered Unified DevSecOps SaaS Platform is built on a cloud-sssssssssssss, microservices-based foundation that ties together CI/CD automation, security analysis, deployment control, and compliance monitoring. The design prioritizes scalability and modularity so that individual components can evolve independently, and the whole system stays accessible to teams without enterprise-level resources

A. Architectural Overview

The layered architecture separates concerns cleanly across six distinct layers:

- Presentation Layer (User Interface)
- Application Layer (Microservices and API Gateway)
- Intelligence Layer (AI Risk Engine)
- Orchestration Layer (CI/CD and Kubernetes)
- Data Layer (Persistent Storage and Logs)
- Cloud Infrastructure Layer

Each layer can scale independently, which keeps the overall system manageable as organizations grow. The platform supports multiple projects and multiple tenants through a standard SaaS delivery model.

B. Microservices Architecture

Every major functional area runs as its own microservice, communicating over REST APIs. This keeps the system modular — individual services can be updated, scaled, or replaced without disrupting the rest of the platform.

1) CI/CD Service

The CI/CD service manages the full build and test lifecycle:

- Triggering pipelines via Git webhooks
- Executing build and unit tests
- Managing artifacts
- Integrating with Jenkins or GitHub Actions

Together, these capabilities keep integration continuous and release workflows consistent across projects.

2) Security Scanner Service

The security scanning service pulls together results from multiple specialized tools:

- Static Application Security Testing (SAST)
- Dependency vulnerability analysis
- Secret detection
- Container image scanning

Findings from all tools get normalized into a common format, so the AI Risk Engine can process them consistently regardless of which scanner produced them.

3) AI Risk Engine

The AI Risk Engine is where raw findings become actionable intelligence. It calculates a risk score for each vulnerability using a weighted model that accounts for:

$$R = \alpha S + \beta E + \gamma B + \delta F + \lambda C$$

Where:

S = CVSS severity score

E = Exploit probability

B = Business impact weight

F = Runtime frequency

C = Compliance sensitivity

$$\alpha + \beta + \gamma + \delta + \lambda = 1$$

The scoring is driven by supervised learning models — Random Forest and Gradient Boosting — trained on historical vulnerability data to produce classifications that hold up across a range of project types.

Outputs include:
Risk score (0–100)
Risk category (High, Medium, Low)
Deployment recommendation
The outcome is a prioritization that actually reflects what teams should fix first, rather than what happens to have the highest CVSS score.
4) Policy Decision Engine
The Policy Engine turns risk scores into deployment decisions automatically, applying configurable thresholds:
High Risk → Block deployment
Medium Risk → Require warning or manual approval
Low Risk → Allow automated deployment
All decisions are logged for auditability and compliance verification.
5) Deployment and Orchestration Service
The deployment service manages containerized applications using:

- Docker for containerization
- Kubernetes for orchestration
- Rolling updates and auto-scaling
- Rollback mechanisms

Rolling updates, auto-scaling, and rollback mechanisms keep deployments reliable and cloud resources used efficiently.
C. Database Design
Persistent storage uses PostgreSQL and MongoDB depending on the data type involved.
Core entities include:

- Users
- Projects
- Pipeline executions
- Vulnerability reports
- Risk scores
- Deployment records
- Compliance logs

The schema is designed for multi-tenancy from the start, keeping each organization's project data logically separate while sharing underlying infrastructure.
D. System Workflow
A typical run through the system follows this sequence:

- Developer commits code to the repository.
- Git webhook triggers the CI/CD pipeline.
- Build and testing stages execute.
- Security scans analyze source code and dependencies.
- Vulnerability results are forwarded to the AI Risk Engine.
- Risk scores are computed and categorized.
- The Policy Engine evaluates deployment eligibility.
- Approved builds are containerized and deployed to Kubernetes.
- Monitoring metrics and compliance reports are updated in real time.

Every code change goes through the same validated path, keeping security continuous rather than periodic.

V. SYSTEM IMPLEMENTATION

A. Technology Stack
The system implementation leverages modern cloud-native technologies as summarized below:

Layer	Technology
Backend Services	Java Spring Boot
AI Engine	Python (Scikit-learn)
Frontend	React.js
CI/CD	Jenkins, GitHub Actions
Containerization	Docker
Orchestration	Kubernetes
Database	PostgreSQL, MongoDB
Monitoring	Prometheus, Grafana
Cloud Infrastructure	AWS, Azure, GCP

B. Backend Implementation
The backend microservices are implemented using Java Spring Boot. Key features include:

- RESTful API architecture
- JWT-based authentication
- Role-Based Access Control (RBAC)
- Asynchronous processing for scan results
- Centralized logging and error handling

Each service is independently containerized and deployed within Kubernetes clusters.

C. AI Model Implementation
The AI module is implemented in Python using Scikit-learn.
Implementation steps include:

- Data preprocessing and feature engineering

- Feature normalization
- Model training using Random Forest classifier
- Cross-validation and performance evaluation
- REST API exposure using Flask or FastAPI

The model is optimized to operate within CI/CD pipeline time constraints to avoid delaying deployments.

D. Deployment Strategy

Each microservice is packaged as a Docker container. Kubernetes manages:

- Pod scheduling
- Service discovery
- Horizontal scaling
- Rolling updates

Configuration management using ConfigMaps and Secrets

This deployment model ensures scalability and resilience.

E. Security and Compliance Implementation

Security controls implemented include:

- HTTPS encryption for API communication
- Kubernetes RBAC policies
- Secure storage of secrets
- Immutable audit logs
- Automated SBOM generation

These mechanisms support compliance with modern software security standards.

F. SaaS Multi-Tenancy

Multi-tenancy is achieved through:

- Namespace isolation in Kubernetes
- Project-based data segregation
- Tenant-aware pipeline configuration

This ensures secure and scalable support for multiple organizations within the SaaS platform.

VI. CONCLUSION & FUTURE WORK

This paper has described an AI-Powered Unified DevSecOps SaaS Platform built to address the fragmentation, complexity, and inaccessibility that define most current DevSecOps setups. By bringing CI/CD automation, continuous security validation, intelligent risk scoring, and policy-driven deployment control into a single cloud-native framework, the platform moves DevSecOps from reactive detection to proactive risk management.

Where conventional tools stop at static severity ratings, this platform builds a fuller picture of each vulnerability's actual risk — factoring in exploit probability, runtime context, business impact, and compliance requirements to produce scores that teams can act on. The result is shorter remediation cycles, less alert fatigue, and security decisions that connect to business outcomes rather than living in a technical silo.

The microservices architecture and Kubernetes backbone give the platform genuine scalability and fault tolerance, while the SaaS delivery model removes the cost and infrastructure barriers that have historically kept serious DevSecOps out of reach for smaller organizations. Automating the most demanding security workflows cuts down on the specialist knowledge required to pipeline management, the platform reduces operational overhead and dependency on specialized security expertise.

development velocity and application security, giving teams a real path to delivering software that is secure, reliable, and scalable software without compromising agility.

The AI-Powered Unified DevSecOps Platform offers a concrete, scalable path toward making intelligent DevSecOps, laying the foundation for future advancements in autonomous security orchestration and adaptive risk-based softwa

REFERENCES

[1] G. Baryannis together with S. Validi, S. Dani, also G. Antoniou explored supply chain risks alongside artificial intelligence - covering current trends plus paths for future study - in the International Journal of Production Research, volume 57, issue 7, pages 2179 through 2200, published in 2019.

[2] B. R. Tukamuhabwa, M. Stevenson, J. Busby - alongside M. Zorzini - “Supply Chain Resilience: Meaning, Overview and Basic Ideas for Future Research,” International Journal of Production Research, volume 53, issue 18, pages 5592 to 5623, published 2015.

[3] W. Xiong, along with D. D. Wu and also J. H. Y. Yeung published a study about semiconductor

supply chains dealing with disruptions - looking into insights, ways to reduce risks, plus what might come next; it appeared in IEEE Transactions on Engineering Management, volume 71, issue 2, pages 341 through 357 during the year 2024.

[4] S. Gupta, along with S. Modgil and R. Meissonier, teamed up with Y. K. Dwivedi to explore how artificial intelligence boosts information system resilience during supply chain disruptions - this study appeared in IEEE Transactions on Engineering Management back in 2023, volume 70, issue 5, pages 1449 through 1462.

[5] A. Zerine, M. Cherif, along with N. Elghazali explored how AI boosts supply chain resilience by combining reinforcement learning with predictive analytics to handle disruptions early - published in the Elsevier Journal of Intelligent Manufacturing Systems, volume 36, pages 142 to 156, released in 2023.

[6] Y. Fang, along with D. Ivanov and also A. Dolgui, published work on spotting disruptions within a smart digital supply chain model - this used mixed deep learning methods - for an article appearing in Computers & Industrial Engineering; there it filled page 109982 across volume 187 during 2024.

[7] M. S. Sodhi along with C. S. Tang explored research chances in supply chain visibility and eco-friendly practices - article titled "Research Opportunities in Supply Chain Transparency and Sustainability," published in Production and Operations Management, volume 30, issue 9, pages 2814 to 2830, year 2021.

[8] S. Dubey, A. Gunasekaran, and S. J. Childe, "Big Data Analytics and Artificial Intelligence Pathway to Operational Performance Under the Effects of Entrepreneurial Orientation and Environmental Dynamism: A Study of Manufacturing Organisations," International Journal of Production Economics, vol. 226, p. 107599, 2020.

[9] D. Ivanov, "Predicting the Impact of Epidemic Outbreaks on Global Supply Chains: A Simulation-Based Analysis on the Coronavirus Outbreak (COVID-19/SARS-CoV-2) Case," Transportation Research Part E: Logistics and Transportation Review, vol. 136, p. 101922, 2020.

[10] L. Hosseini, E. Ivanov, along with M. Dolgui published "Resilient and Sustainable Supply Chain Design: Hybrid Optimization Approach" in the International Journal of Production Research, volume 59, issue 12, pages 3562–3578, released in 2021.

[11] X. Zhang, plus P. He, along with M. Ren - "IoT-Driven Forecast Tools for Handling Supply Chain Breaks," IEEE Internet of Things Journal, volume 8, issue 15, pages 12177 to 12189, published 2021.

[12] J. Chen with T. Huang, "Live supply chain tracking plus optimization using digital twins," IEEE Transactions on Industrial Informatics, volume 17, issue 6, pages 4152–4163, published 2021.

[13] C. Li, alongside L. Wang and also F. Li, explored deep reinforcement learning applied to instant supply chain improvements - published in Expert Systems with Applications, volume 197, article number 116780, released in 2022.

[14] A. Kaur, along with P. Singh but also S. Gupta, wrote a paper titled "AI-Driven Forecasting in Supply Chains: Combining Risk Control with Better Results," published in the Journal of Smart Production, volume 35, pages 1421 to 1438, released in 2023.

[15] N. Christopher, along with M. Dubey, together with V. R. Mani wrote a paper titled "Blockchain meets IoT - Boosting Transparency and Safety in Supply Chains," published in IEEE Access, volume 9, pages 132551 to 132563, back in 2021.

[16] P. Xu, together with J. Liu and also Z. Chen, published a study titled "A Data-Driven Method for Real-Time Risk Evaluation Across Worldwide Supply Networks by Means of Machine Learning" in Omega, volume 102, page 102384, released in 2021.

[17] A. Dolgui, along with D. Ivanov and F. Sokolov, explored reconfigurable supply chains amid uncertain conditions - a review plus outlook on future studies appeared in Omega, volume 90, page 102012, published in 2020.

[18] Y. Yang, along with C. Wang and also X. Lin wrote a study titled "Hybrid Deep Learning for Supply Chain Disruption Prediction: A Comparative Analysis," published in IEEE

Transactions on Artificial Intelligence - volume 4, issue 1, pages 33 to 46 during 2023.

[19] M. Kumar, along with N. Gupta and also R. Jain, wrote a paper titled “AI-Driven Decision Support Systems for Logistics Risk Mitigation: A Real-Time Framework,” published in Computers & Operations Research, volume 156, page 106239, released in 2024.

[20] S. Li, along with W. Xu plus J. Zhao, wrote "Moving toward Smarter Supply Chains: A Full Look at How AI Meets IoT," published in IEEE Access, volume 10, pages 99456 to 99475, released in 2022.