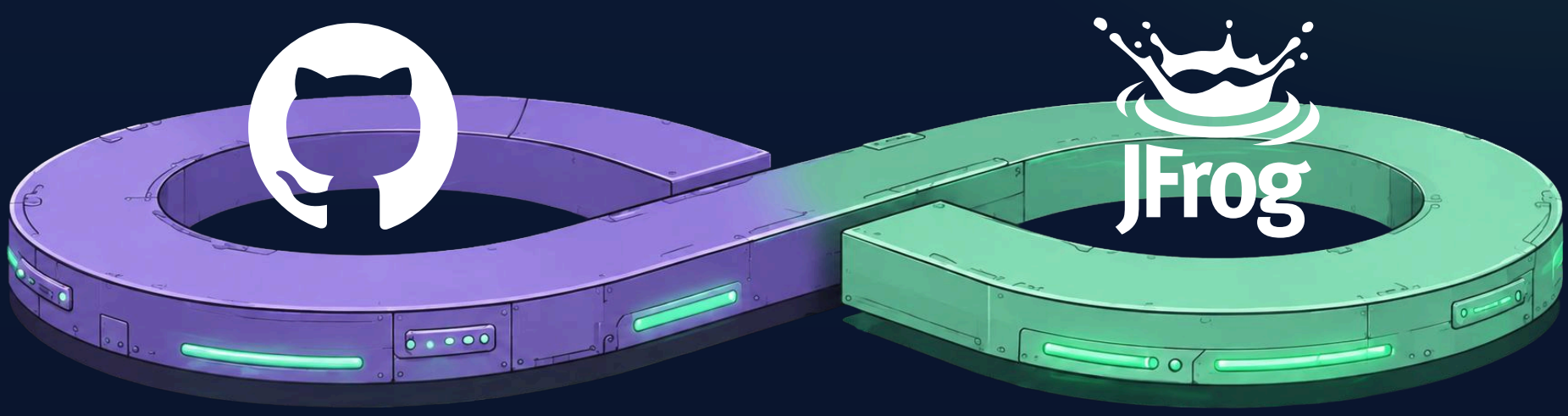
JFrog × GitHub



Best Practices for the Agentic Software Supply Chain

Securing AI-assisted development from IDE to production with JFrog and GitHub.

- Reference architecture
- Guardrails & governance
- Staged rollout plan

A hands-on reference for platform, Devops, and Appsec teams

Table of Contents

What you'll take away

A reference architecture

Seven components that every agentic development environment needs, and how they fit together.

Concrete guardrails

How to constrain AI agents with JFrog Curation policies, MCP governance, and contextual analysis.

A staged rollout plan

A reproducible path from one repo to enterprise scale.

Executive Summary

Agentic development has arrived. Your supply chain needs to catch up.

AI coding agents are now shipping production code. Tools like **GitHub Copilot**, **Claude Code**, **OpenAI Codex**, and **Cursor** write significant portions of modern codebases, select dependencies, and remediate vulnerabilities autonomously. That speed creates a new class of risk: code that ships faster than your security processes can evaluate it.

This guide is a practitioner's reference for teams adopting agentic workflows with the JFrog Platform and GitHub. It's structured as a hands-on reference with checklists, best practices, and lessons learned. You'll find concrete patterns for setting up your environment, securing AI-generated code, automating remediation, and maintaining governance across the supply chain.

WHY IT'S IMPORTANT

Your AI agents are only as trustworthy as the supply chain they operate in. Relying on traditional security gates for machine-speed code generation leaves your organization exposed to entirely new attack surfaces. Discover how to fundamentally shift your security paradigms, secure your automated workflows, and give agents the context they need to make the right choices the first time.

Ready to jump in? Install the JFrog App for GitHub, or request a personalized demo of the JFrog Platform today.

> *We no longer spend days chasing down vulnerabilities. The platform gives us the visibility and automation to act within hours, not days.*
>
> **SOFTWARE SUPPLY CHAIN MANAGER, TELECOM ENTERPRISE**

The Agentic Development Landscape

Agentic development represents a fundamental shift in how code is produced. Rather than writing every line manually, developers increasingly collaborate with AI agents that can plan multi-step tasks, select dependencies, generate implementations, run tests, and iterate on failures with minimal human intervention.

What has changed

The toolchain matured fast. GitHub Copilot's agent mode independently translates high-level goals into multi-file changes and self-corrects. Claude Code operates directly in your terminal, executing shell commands and editing across the repo. Cursor and similar IDE-integrated agents provide inline generation with deep project context. All of them now connect to external services via the **Model Context Protocol (MCP)**, giving agents access to registries, security databases, and CI/CD systems.

Agents are no longer suggesting completions. They are making architectural decisions, picking open-source packages, and shipping artifacts. **Every one of those decisions has supply-chain implications.**

Why your supply chain must adapt

Dependency selection at machine speed

Agents pick packages based on training data and context, not your organization's security policy. Without guardrails, a vulnerable or malicious package lands before anyone reviews it.

Volume of generated code

When agents produce thousands of lines per session, manual code review becomes a bottleneck. You need automated shift-left scanning that runs at the speed of generation.

MCP server proliferation

Each new MCP connection is a potential attack surface. Prompt injection, over-privileged access, and credential exposure are real. Shadow AI is the new shadow IT.

Regulatory pressure

EU Cyber Resilience Act, NIST SSDF, and executive orders require demonstrable supply-chain integrity. Agentic workflows must produce auditable evidence.

Setting Up Your Agentic Development Environment

A well-configured environment is the foundation for safe agentic development. The stack isn't any one thing; it's how the components fit together.

THE REFERENCE ARCHITECTURE

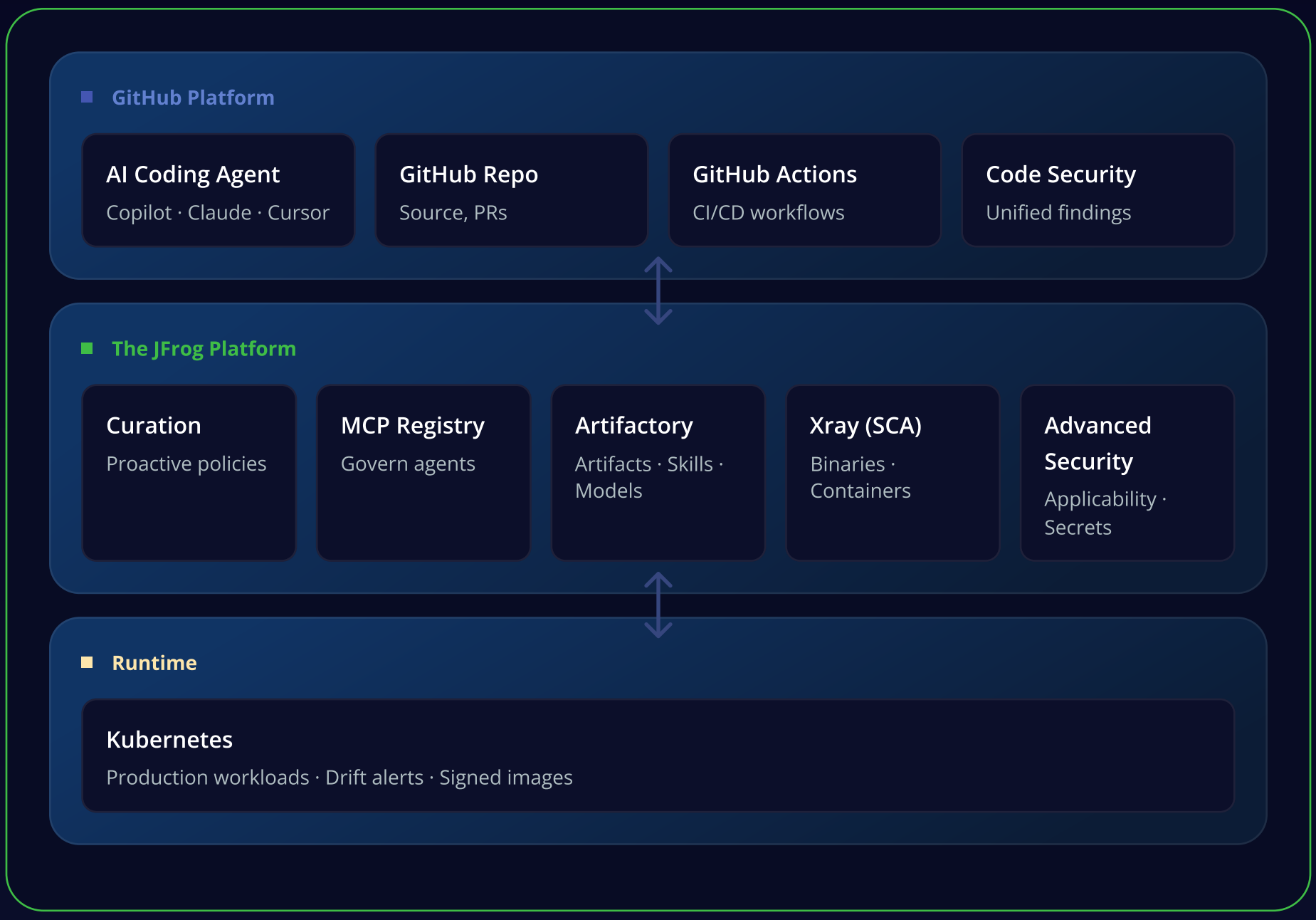


Fig. 2.1 — A unified agentic software development stack.

Essential components

Component	Role in agentic workflows
AI Coding Agent	GitHub Copilot, Claude Code, Codex, Cursor, or similar. Generates code, selects dependencies, and iterates on test failures.
Source Platform	GitHub repositories, Actions workflows, and Copilot coding agent for async task delegation.
Artifact Registry	JFrog is the single source of truth for binaries, containers, Helm charts, ML models, and agent skills.
Security Scanning	JFrog Xray + Advanced Security for binary SCA, SAST, secrets, and Contextual Analysis surfaced in GitHub Code Security.
PR Security Scanning (Frogbot)	JFrog's Git integration that automatically scans AI-generated pull requests for vulnerabilities, surfacing Xray and Advanced Security findings as actionable PR comments before the code is merged.
Package Curation	By blocking malicious or risky open-source packages at the perimeter, JFrog Curation defends your software supply chain and keeps both development teams and CI workflows secure.
MCP and Skills Registry	JFrog's MCP & Skills Registries govern which MCP servers and agent skills are approved for enterprise use.
CI/CD Pipeline	GitHub Actions integrated with JFrog for automated build, test, scan, and promotion workflows.

Integration Checklist Before Your First Agentic Sprint

Verify that the following are in place. Each row maps to a one-time setup step; once configured, agents and pipelines inherit the guardrails.

Check	What it does	Owner
JFrog App for GitHub installed	Automates integration via OIDC; no long-lived credentials. Installs in minutes from the GitHub Marketplace.	Platform engineer
Bidirectional traceability	Developers can navigate from source in GitHub to binaries in Artifactory and back, without context-switching.	Platform engineer
Xray results in GitHub Code Security	Binary scan findings appear alongside source-level findings. One dashboard, one triage queue.	DevSecOps
Curation policies active	Define allowed, blocked, and review-required packages. Agents are bound by these policies via the JFrog MCP server.	AppSec
MCP servers registered	Every MCP server catalogued in the JFrog MCP Registry has scoped permissions and monitoring.	AppSec
GitHub Actions linked to JFrog	CI workflows publish to Artifactory, trigger Xray, and enforce promotion gates automatically.	Platform engineer
AGENTS.md committed	Per-repo instructions telling coding agents which MCP servers to use, what to scan, and how to handle findings.	Dev leads

Agent-Specific Notes

Github Copilot · native integration

Copilot Autofix consumes JFrog SAST findings and generates PR-ready fixes. Agent mode delegates scoped tasks asynchronously.

Claude Code · terminal-native

Runs shell commands and edits files across the repo. Add the JFrog MCP server in `.mcp.json` so Curation is in the loop before dependency edits.

Cursor · IDE integrated

Inline generation with project context. Configure the JFrog MCP server in Cursor settings; surface Xray findings inline.

Portability

Write your agent configuration once in `AGENTS.md` or `.mcp.json` Developers can then switch between Copilot, Claude Code, and Cursor without rewiring policy. The MCP contract is stable across agents.

BEST PRACTICE

While GitHub Code Security provides a unified pane of glass for security findings, deploying JFrog's Frogbot brings JFrog's contextual and binary analysis directly into the developer's PR workflow. Use Frogbot as your primary gatekeeper for AI-generated pull requests to catch hallucinated or vulnerable dependencies before they trigger a CI build, saving compute time and developer context-switching.

Securing AI-Generated Code in Your Supply Chain

While AI-generated code is not inherently less secure than human-written code, it arrives faster, in larger volumes, and with entirely different failure modes: **hallucinated imports**, plausible-but-wrong cryptography, and over-broad permissions. Your defenses need to run at the same tempo.

Scan at every stage

Build checks into each point where an agent can introduce risk:

The five checkpoints

01 **IDE / coding time**	JFrog SAST flags vulnerable patterns as agents write them. Findings appear in the IDE with AI-suggested fixes.
02 **Dependency selection**	Curation enforces enterprise policies that determine which packages are allowed, based on CVE feeds, license policy, and malicious-package intelligence, before they enter. Via MCP, the agent gets this verdict inline and picks a compliant alternative.
03 **Build / CI**	GitHub Actions triggers Xray on every build. Contextual analysis automatically detects non-exploitable vulnerabilities. JFrog Research finds 66% of High and Critical CVEs have an applicability rate under 20%, keeping the pipeline frictionless.
04 **Binary and container**	Xray recursively analyzes all layers, including OS packages, libraries, and transitive dependencies, that source-only scanners miss.
05 **Runtime**	JFrog Runtime verifies that running images match their stored counterparts in Artifactory, and alerts on drift.

BEST PRACTICE

Establish a multi-layered defense: **JFrog Curation** as your entry gate to block risky dependencies and models at the perimeter, and **JFrog Xray** as your production gate for CD. Enforcing Xray build and deployment blocks guarantees that vulnerable container images never reach your production clusters, regardless of how fast agents generate them.

Source + Binary:
One Picture, One Triage Queue

A persistent gap in most security programs is the split between source scanning and binary analysis. GitHub sees the code; JFrog sees the shipped artifact. They describe the same application but rarely the same risk. The integration closes that gap.

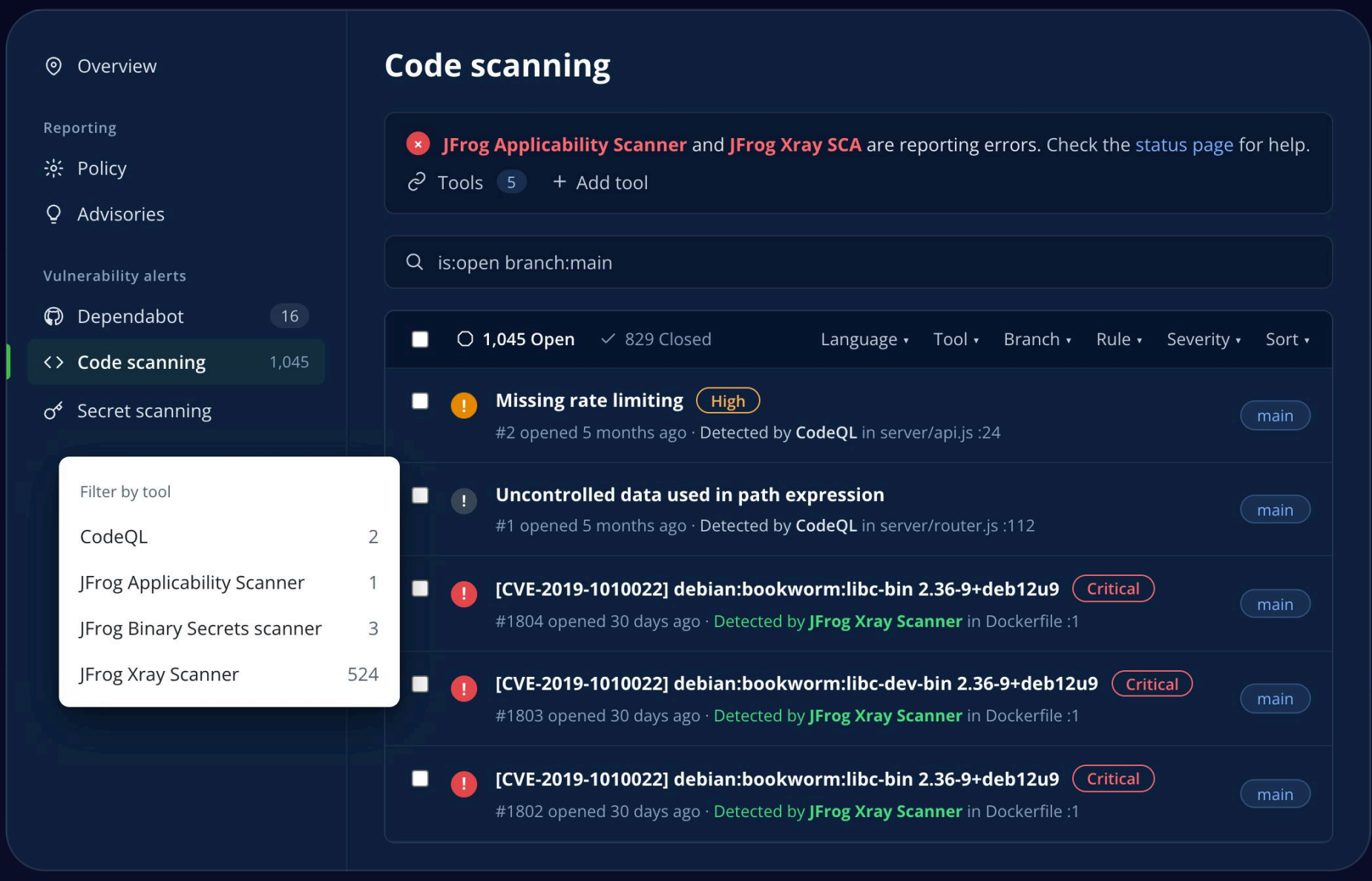


Fig. 3.2 — JFrog Xray, Advanced Security, Curation, and CodeQL findings unified in a single Code Scanning view.

---

[1] JFrog Security Research, "Applicability types of high-profile CVEs," *2026 Software Supply Chain Security State of the Union*, p. 32, available at jfrog.com/software-supply-chain-state-of-union/.

What Unification Gives You

Holistic SBOMs

GitHub's source SBOM merges with JFrog's binary SBOM in the GitHub Dependency Graph: one end-to-end bill of materials across CycloneDX and SPDX.

One triage queue

Xray and Advanced Security findings push directly into GitHub Code Security alongside CodeQL results, in the same UI your developers already live in.

Build attestations

GitHub build attestations integrate with JFrog AppTrust for evidence-backed, policy-driven promotions.

Shared context

A vulnerability in a transitive dependency surfaced by Xray is linked to the exact PR and code path that introduced it.

BEST PRACTICE

Pin your triage SLA to **contextual severity**, not raw CVSS. Reachability analysis in **JFrog Advanced Security** identifies vulnerabilities that aren't actually exploitable in your code. Teams using this feature typically see **alert volume drop by 50%** within a quarter.

Automating Remediation With Agentic Workflows

Finding vulnerabilities is only half the problem. The real productivity gain comes from automating the fix. JFrog and GitHub together enable true **agentic remediation**, where AI agents identify, prioritize, and resolve security issues with minimal human intervention.

How the loop works

01 **Agent writes code:** Copilot, Claude Code, or Cursor drafts changes with project context.

02 **Packages checked:** JFrog Curation evaluates each dependency against policy in real time.

03 **Enriched research and context:** JFrog feeds the agent enhanced CVE data, reachability analysis, and precise remediation strategies curated by JFrog's Security Research team.

04 **Smart strategy selection:** Instead of blindly defaulting to a package upgrade (which may not be available), the agent uses that context to strategize the optimal fix:

Development Remediation:

Modify code usage patterns to make the CVE completely non-applicable, without changing the package version.

Configuration Changes:

Tweak environment settings to neutralize the vulnerability.

Runtime Mitigations:

Implement protective guardrails at the execution layer.

04 **PR with evidence:** Copilot Autofix attaches the SAST fix, SBOM, and attestation to the pull request.

Practical Tips

- **Start with Curation policies.** Define clear rules for acceptable versions, licenses, and security scores: the guardrails every agent respects.
- **Use Contextual Analysis to prioritize.** Contextual and reachability analysis from JFrog Advanced Security determines applicability, reducing noise so agents (and humans) fix what matters.
- **Enable "Ask Copilot to fix" on CVEs.** When Xray surfaces a vulnerability in the IDE, developers trigger an AI-assisted fix with one click.
- **Leverage the JFrog Claude plugin and Skills Registry.** Equip your coding agents with native platform context, verified skills, CLI execution patterns, and API guardrails.
- **Leverage `AGENTS.md` and custom instructions.** Tell agents how to handle security findings, which MCP servers to use, and which standards apply.
- **Review agent PRs like junior-dev PRs.** Treat well-scoped remediation tasks as autonomous PRs. Read, iterate, merge.

Picking an agent

The right agent depends on where your developers already work. All three integrate with the JFrog MCP server; your policy is portable.

When Copilot fits

Deep GitHub UI integration, agent mode for async delegation, first-class Autofix. Strongest option if your org is GitHub-centric.

When Claude Code fits

Heavy terminal work, multi-repo refactors, CI scripting. Runs where your developers run; strong at long-horizon tasks.

When Cursor fits

IDE-native inline generation with project-wide context. A good default for day-to-day feature work and quick edits.

Governing Your Agentic Supply Chain

As AI agents become autonomous participants in your supply chain, governance shifts from gating human actions to governing agent behavior. Credentials, permissions, audit trails: all of them need to cover the non-human identity on the other end of the commit.

MCP server governance

MCP servers are the integration points that give agents access to your systems. Left unmanaged, they create the same risk profile as shadow SaaS in 2018: ungoverned, over-privileged, and invisible to security. The **JFrog MCP Registry** provides centralized control.

Catalog and approve every server

Each MCP server passes the same policy controls applied to software artifacts: provenance, signatures, and review workflow.

Monitor connections live

Record which agents connect to which servers, what actions they perform, and flag anomalies in real time.

Scope every permission for least privilege

Grant only what's needed. Disable write access to Kubernetes APIs unless the specific use case justifies it.

Discover shadow AI and unauthorized tools

JFrog AI Catalog detects unauthorized AI tools and MCP servers operating outside approved channels.

Compliance & audit readiness

- **Automated SBOM generation.** Source and binary SBOMs combine in GitHub's Dependency Graph, in CycloneDX and SPDX formats.
- **Application management.** Use GitHub Attestation to create build provenance that automatically converts into JFrog evidence for AppTrust control and SLSA v2 compliance. AppTrust workflows leverage these cryptographic attestations to enforce policy gates at every stage: dev, staging, and production.
- **Regulatory alignment.** These practices map directly to EU Cyber Resilience Act (CRA) requirements, NIST SSDF guidelines, and SOC 2 controls for supply chain integrity.

A Field-Tested Pattern

Separate your MCP registry into three tiers.

Tier 01

Production

Approved and monitored. Full scopes, live connection telemetry, anomaly alerting.

Tier 02

Evaluation

Sandboxed with limited scopes. Under review, no production credentials.

Tier 03

Blocked

Denied at the registry. Agents cannot connect, and attempts are logged.

Use the same tiering you already use for package registries, developers already understand the model.

BEST PRACTICE

Treat MCP server approvals like third-party library approvals. Every new server goes through a security review before being added to the registry. The JFrog MCP Registry automates most of the evaluation (license, provenance, permission footprint), so review stays lightweight.

Measuring Impact

Adopting these practices delivers measurable outcomes:

282%

3-year ROI on the JFrog Platform, with payback in under six months for the composite enterprise.

65%

Fewer critical vulnerabilities reaching production, driven by contextual analysis, Curation, and shift-left scanning.

80%

Faster remediation. Issues resolved in hours rather than days through automated prioritization and agentic fix suggestions.

71%

Lower tool spend. Consolidation onto the unified platform eliminated redundant scanning, artifact management, and compliance tools.

38 hrs

Saved per developer onboarded. Preconfigured environments, SSO, and standardized DevSecOps get new engineers shipping secure code by day two.

$5.4M

Total 3-year benefits (risk-adjusted) for the composite organization modeled by Forrester.

About the data: These financial and operational metrics are derived from a January 2026 *Forrester Consulting Total Economic Impact™ (TEI) study* commissioned by JFrog. Forrester interviewed existing enterprise customers to construct a multi-billion-dollar composite organization and model these risk-adjusted benefits.

What to measure

Pick four leading indicators and track them weekly:

- **Mean time to remediate critical vulnerabilities**
- **Percentage of PRs with automated fix suggestions**
- **Packages blocked by Curation per week**
- **MCP servers in active use vs. registered**

If those trend in the right direction, the lagging outcomes above tend to follow.

Your Journey to Agentic DevSecOps

Getting started doesn't require a complete overhaul. The best approach is incremental: **start small, validate, then expand.**

WEEK 1

Start Small

One repo, one agent

- Install the JFrog App for GitHub
- Connect one coding agent via MCP
- Enable Curation on one ecosystem (npm or PyPI)

MONTH 1

Expand

Team-wide coverage

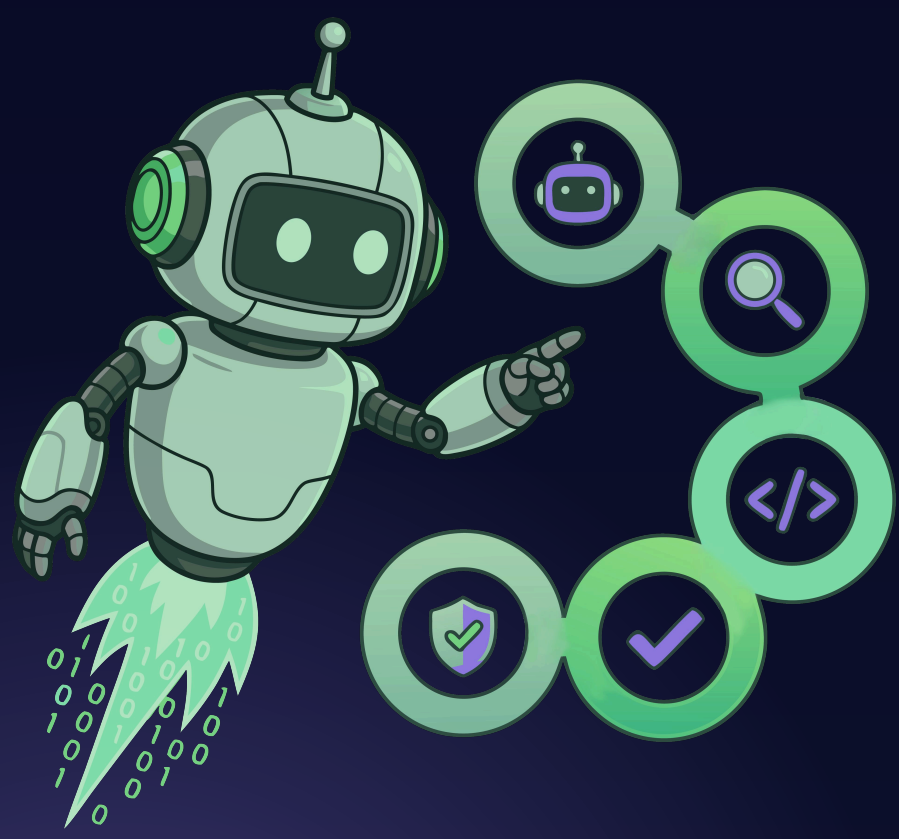
- Xray scanning in GitHub Actions
- MCP governance: register approved servers
- Enable Copilot Autofix + JFrog SAST on PRs

QUARTER 1

Scale

Enterprise-wide production

- Deploy Artifactory HA
- Release lifecycle management across environments
- JFrog Runtime in production clusters



Conclusion

AI coding agents are not a future trend; they are shipping production code today. The organizations that thrive will be those that treat agentic development not as a productivity hack, but as a **supply-chain transformation** that demands new security, governance, and operational practices.

The integration of JFrog and GitHub provides the infrastructure for this transformation: a unified platform where AI agents operate within governed boundaries, security scanning runs at the speed of generation, and every artifact is tracked, verified, and trusted.

Start with the basics, measure your impact, and scale securely. Because **ultimately, your agents are only as trustworthy as the supply chain you build for them.**

Next Steps

01

Take a tour of the JFrog Platform

See the unified supply chain end to end.

02

Install the JFrog App for GitHub

Automate OIDC setup in minutes.

03

Start a free JFrog trial

Connect an Artifactory instance to your GitHub org.

04

Download the TEI study

Read the full 282% ROI analysis.