The Trusted Release Model:
A Strategic Blueprint for Enterprise Software Delivery

Table of Contents

Executive Summary

For CISOs, CIOs, and GRC (Governance, Risk, and Compliance) leaders, there is an inherent conflict balancing speed and trust. Businesses need to deliver new features quickly to stay competitive, but this can't happen at the expense of security, compliance, or quality. To increase development speed while preserving application integrity, leaders must move beyond incremental tool improvements, and adopt a foundationally suitable architectural framework.

JFrog refers to this framework as the Trusted Release Model, an approach that embeds **trust**, **governance**, and **security** directly into high-velocity software delivery pipelines.

The model's three foundational pillars systematically address the critical governance gaps left by previous software delivery approaches:

- **System of Record for Applications:** The model establishes a centralized, business-aware registry for all software assets, transforming abstract concepts like ownership, business risk, and maturity into programmable objects, and creating a single source of truth for governance and automation.
- **Automated Preventative Compliance:** The model shifts compliance from reactive, manual audits to proactive, automated governance, leveraging an integrated policy engine that ensures compliance is a continuous, preventative measure, and not a post-release discovery.
- **Immutable Audit Trail:** The model provides an unbreakable, API-accessible chain of evidence for every release, so you can instantly verify compliance with internal quality standards and external industry regulations.

The Trusted Release Model can be broken down into 4 implementation phases:

- **Phase 1:** Establish the Foundation - System of Record
- **Phase 2:** Implement Governed Lifecycles - Control Plane
- **Phase 3:** Automate and Enforce - Operational Blueprint
- **Phase 4:** Scale and Observe - Visibility & Governance at Scale

This blueprint for the Trusted Release Model gives DevOps, Security and GRC leaders a structured, actionable approach to redesign their software delivery processes, with the explicit goal of enabling their organizations to release quality software with both speed and trust.

Section I
The Enterprise Trust Dilemma: Moving Beyond Orchestration

CI/CD pipelines transformed modern software development by automating individual component builds and deployments. But teams now face a new, more complex challenge: managing entire applications - often composed of hundreds of interdependent microservices - as single, cohesive, and trusted products.

- **The industry provided an Application Release Orchestration (ARO) solution.** ARO automates and sequences the steps to release entire applications across environments, solving the technical coordination problem. For example, JFrog's Release Lifecycle Management (RLM) manages a collection of artifacts, packaging them into an immutable release candidate called a Release Bundle. It then promotes this bundle through a series of environments from DEV to PRODUCTION.

- **ARO has one fundamental limitation: it lacks business context.** ARO doesn't understand who owns the application, how critical it is, or what compliance rules apply. This forces teams to track ownership, manage risk, and check compliance outside the platform, creating friction, delays, and audit challenges. GRC teams are forced to rely on a patchwork of ticketing systems and manual checklists to answer basic questions about their applications.

- **The Trusted Release Model goes beyond orchestration.** This new approach closes the governance gap by integrating business context, ownership, and a native policy engine directly into the software supply chain.

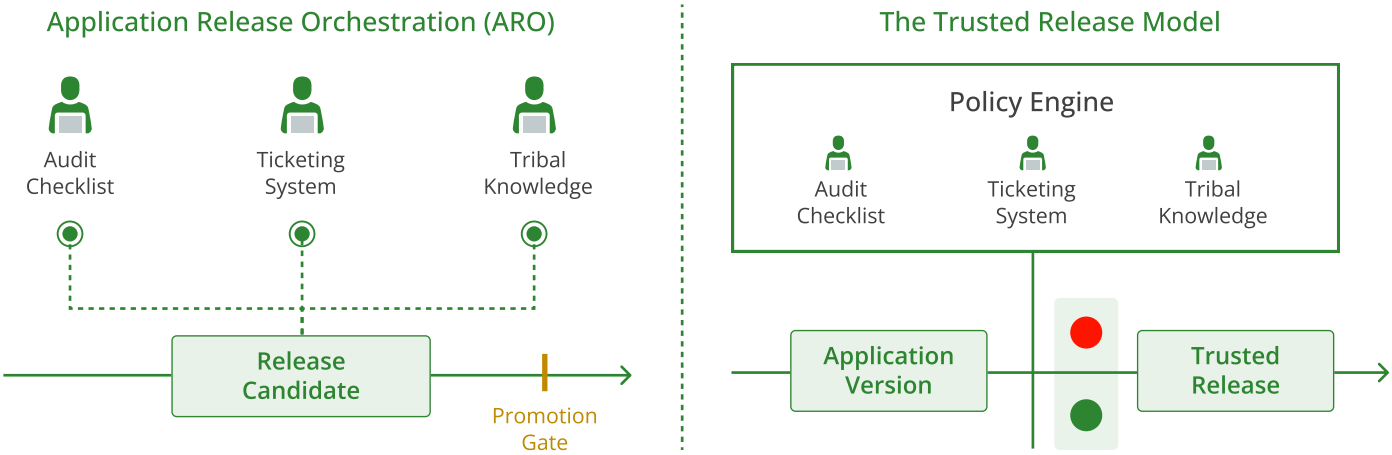

Figure 1: ARO vs. The Trusted Release Model

An immutable release candidate provides a technical snapshot of the software at a specific point in time. Governance with ARO requires a gating process where users rely on manual workflows.

A Trusted Release is the final, approved state of a software application. An application earns this distinction after passing an automated evidence-based policy. With the Trusted Release Model, governance automates the process of checking a release against its business context to ensure it meets policy.

The following table illustrates the differences between **ARO** and the **Trusted Release Model:**

Capability	**Application Release Orchestration (ARO)**	**The Trusted Release Model**
Unit of Management	Release Candidate (A technical collection of artifacts)	Application (A business-aware system of record)
Ownership	Implicit (via naming conventions or external systems)	Explicit & Enforceable (defined metadata field)
Governance	External Gates (Reactive "proof-of-work")	Integrated Unified Policy (Proactive "proof-of-compliance")
Release Status	"Released" (A promotion to a final stage)	”Trusted Release” (A cryptographically provable status)
Audit Trail	Dispersed promotion logs	Centralized, immutable Activity Log
Business Context	Absent (Requires external correlation)	Native (criticality, maturity, labels)

Table 1: ARO vs. The Trusted Release Model

Section II:
The Trusted Release Model: How It Works

The Trusted Release Model's core is a definitive, centralized, and API-first system of record. It turns anonymous artifacts into well-defined, business-aware entities. This section deconstructs the core, programmable objects that make up this system of record, transforming abstract governance concepts into manageable data structures.

The Application Object: From Artifacts to Applications

The model's core innovation is the Application Object, which turns a collection of technical files into a logical, business-aware entity, creating the foundation for intelligent, risk-aware governance.

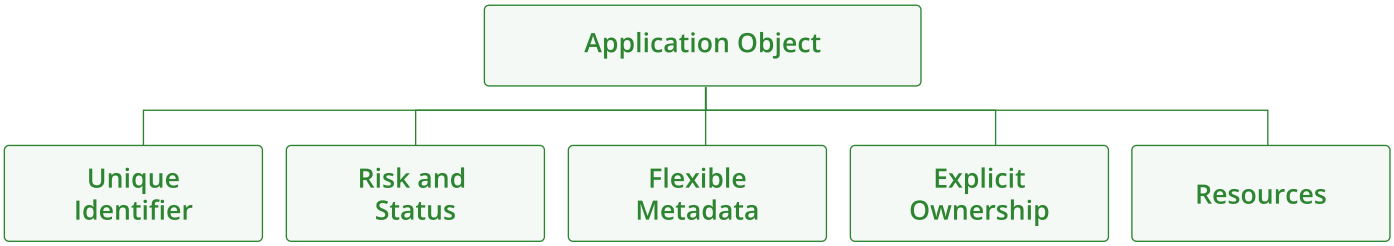

Figure 2: Application Object

The Application Object tracks these essential attributes:

- **Unique Identifier:** An immutable `application_key` providing a permanent, unique ID.
- **Risk and Status:** Fields for business criticality (e.g., low, high) and maturity (e.g., production) allow classification of risk and lifecycle status.
- **Metadata:** Extensible labels (e.g., `pci-scope: true`) provide a flexible way to apply custom metadata for granular policy targeting.
- **Explicit Ownership:** The `owners` field provides a direct solution to manage accountability. This eliminates manual effort in tracking down responsible teams during an incident or audit.
- **Resources:** Directly links the application to its technical components such as artifacts, packages, and builds for verifiable composition.

How It Works

You can create and manage the Application Object via a well-defined REST API. This API-driven approach ensures the system of record is a dynamic, integrated part of the software supply chain, not a static database.

Binding the Software Supply Chain

With the **Trusted Release Model**, an application establishes ownership over the packages it creates through a process called binding. This action creates a clear, auditable link between the business entity of the application and the software components it maintains. When an issue arises in a package, the application's owners are directly accountable for the fix.

While an application establishes ownership over the packages it creates, a single version of a specific application - the actual unit of release - is actually a composite of packages, all with different owners. It might include packages the application itself owns, public open-source components, or shared internal packages from other applications. This federated ownership model creates a clear, scalable line of responsibility regardless of how many owners are involved.

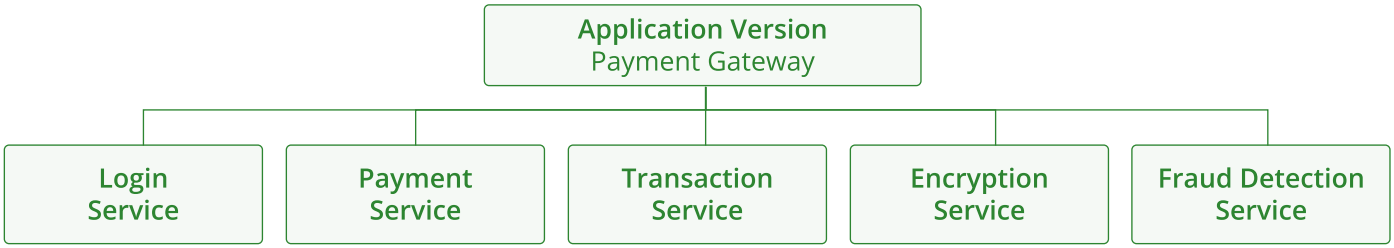

Figure 3: Example of Different Microservices in an Application Version for a Payments Gateway

For example, a payments gateway might use a login package owned by a separate common-utils application. If a vulnerability appears in the login functionality within the payments gateway, the system of record immediately clarifies that the common-utils application owners are responsible for addressing the issue. This eliminates ambiguity and ensures problems go to the right team, a critical capability for managing risks and incidents at enterprise scale.

How It Works

This demonstrates how a Bind Package API creates the ownership link. You can see the ownership model within a release, for example, in JFrog AppTrust, through the Get Application Version Content API. The response for this endpoint includes an `owning_application_key` and a connection_level field (e.g., `1st_party, 2nd_party, 3rd_party`) for each component, identifying which application is the owner.

The Application Version: Moving Applications to a Single Unit of Trust

The **Application Version** is the immutable, verifiable, and atomic unit of release in the **Trusted Release Model**. It represents a specific, versioned instance of an Application, with a precise bill of materials at a single point in time.

Strategically, the Application Version is a step up from the immutable release candidate in the ARO approach. While it can use the same underlying technology for immutability and signing, its true significance comes from its direct link to the business-aware Application Object. An Application Version inherits its parent Application's owners, criticality, and compliance labels, elevating it from a simple collection of artifacts to a "Unit of Trust" that is ready for governance and policy enforcement. This context enables a more efficient, intelligent form of governance. Policies are no longer blunt, one-size-fits-all instruments.

For example, a Unified Policy Engine applies risk-calibrated logic by using an application's criticality attribute. You can write a policy to enforce the most stringent security checks on applications where `application.criticality == 'critical'` while applying a baseline standard to less critical systems. This dynamic enforcement allows organizations to focus their security efforts where business risk is greatest.

Section III: The Control Plane: Governed Application Lifecycles

With a robust system of record in place, the Trusted Release Model adds an active, intelligent Control Plane to enforce governance. This is the model's core, where declarative rules become automated actions. This section details the Control Plane's architecture, which empowers GRC and security teams by transforming them from passive auditors into proactive strategic decision makers.

From "Proof-of-Work" to "Proof-of-Compliance"

ARO systems rely on a reactive "proof-of-work" model. They collect evidence for validation in quality and security gates, but the development platform remains passive. A human auditor or external script must later inspect this evidence to check for compliance. This after-the-fact validation is inefficient, error-prone, and creates a significant delay between a policy violation and when it gets discovered.

The Trusted Release Model fundamentally shifts to a proactive, "proof-of-compliance" system. It acts as an automated auditor at every critical stage, enforcing compliance before a release can proceed. Policy is no longer a checklist - it's built into the release process.

Policy as Code: Unifying Governance Workstreams

The Unified Policy Engine simplifies workflows for everyone involved in enterprise governance.

- **GRC Teams:** An intuitive wizard lets GRC teams create rules without writing a single line of code.
- **Platform Engineers:** The Unified Policy Engine provides more flexibility for platform and security engineers to author complex policies using the industry-standard Rego language.

This dual-mode approach creates a common ground for collaboration. GRC leaders can define high-level requirements, and platform teams can implement them as automated "policy as code." These policies are dynamic and scoped using Application Object labels for granular control. For example, you can author a single policy and automatically apply it to all applications labeled `data-classification: confidential`, ensuring consistent data protection across the enterprise as an automated auditor at every critical stage, enforcing compliance before a release can proceed. Policy is no longer a checklist - it's built into the release process.

The Trusted Release Designation: Verifiable Trust

ARO systems rely on a reactive "proof-of-work" model. They collect evidence for validation in quality and security gates, but the platform remains passive. A human auditor or external script must later inspect this evidence to check for compliance. This after-the-fact validation is inefficient, error-prone, and creates a significant delay between a policy violation and when it gets discovered.

The Trusted Release Model fundamentally shifts to a proactive, "proof-of-compliance" system. It acts as an automated auditor at every critical stage, enforcing compliance before a release can proceed. Policy is no longer a checklist - it's built into the release process.

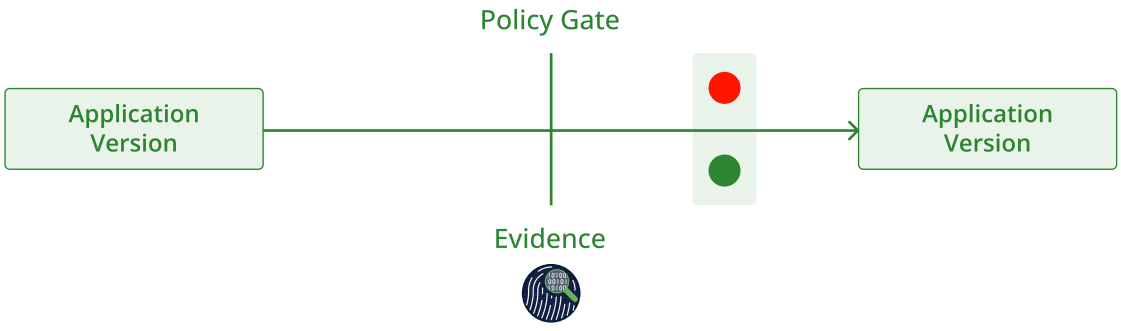

Figure 4: How an Application Version Earns the Trusted Release Designation

The Trusted Release Model only deploys software it can verify and trust. An Application Version earns Trusted Release status only when it successfully passes a designated release gate policy during its promotion to production. “Trusted Release” is not a subjective label; it’s a formal, provable designation.

This event marks the culmination of the entire governance process, providing a definitive, auditable “go/no-go” decision. If no policy is configured on the release gate, the version is simply “released”, creating an intentional distinction between a standard deployment and one cryptographically proven to meet the organization’s highest standards.

This architectural shift fundamentally changes the operational role of GRC teams, from manually chasing down evidence after the fact, to becoming proactive rule-makers, codifying trust and compliance standards directly into the platform. Automated enforcement via the platform transforms GRC from a bottleneck to a strategic enabler of safe, fast software delivery.

How It Works

The **Trusted Release** is a technically provable state recorded by the platform. When a user calls the `Release Application Version API`, the policy engine evaluates the release candidate. The `evaluations` block in the API response provides immutable proof of this event. Specifically, a `release_gate` object with a decision: `"pass"` and a unique `eval_id` constitutes the cryptographic evidence that the release is trusted.

Section IV: Moving Trusted Applications to Production

Now that we’ve defined the model’s foundational objects and Control Plane, this section provides a blueprint for how they operate in a real-world software delivery lifecycle.

It also details how the Trusted Release Model translates governance into an automated practice, from initial development through production.

The Governed Path to Promotion and Deployment

The model follows a standard enterprise promotion path below, but upgrades the process by governing each transition as a recorded event rather than a deployment:

At each boundary, the Unified Policy Engine acts as an automated gatekeeper. It defines policies for both stage entry (`entry_gate`) and stage exit (`exit_gate`), which creates a series of checkpoints that ensure an Application Version meets progressively stricter criteria as it matures.

For example, an `entry_gate` policy for QA might require a completed security scan. The `entry_gate` for STAGING could require proof of successful end-to-end testing from the QA stage. The release gate for PROD can enforce the most stringent requirements, like zero critical CVEs and verifiable evidence of UAT (User Acceptance Testing) sign-off.

This gating structure is how the model actively manages the trade-off between speed and trust. The **Trusted Release Model** provides the flexibility to match policy rigor to the specific risk tolerance of the stage.

Early in the pipeline, such as in the Development and QA stages, teams can deploy **warn** policies (.e.g., “Warn if test coverage is low”) to collect compliance data without disrupting velocity. As the application moves closer to production, policies transition to **fail** conditions (e.g., “Block if critical CVEs exist”), making trust enforceable. This strategic deployment of policy types ensures teams move fast where risk is lower, and slow down only when risk and compliance requirements are absolute.

Lifecycle Stages & Gates (Policy Based)

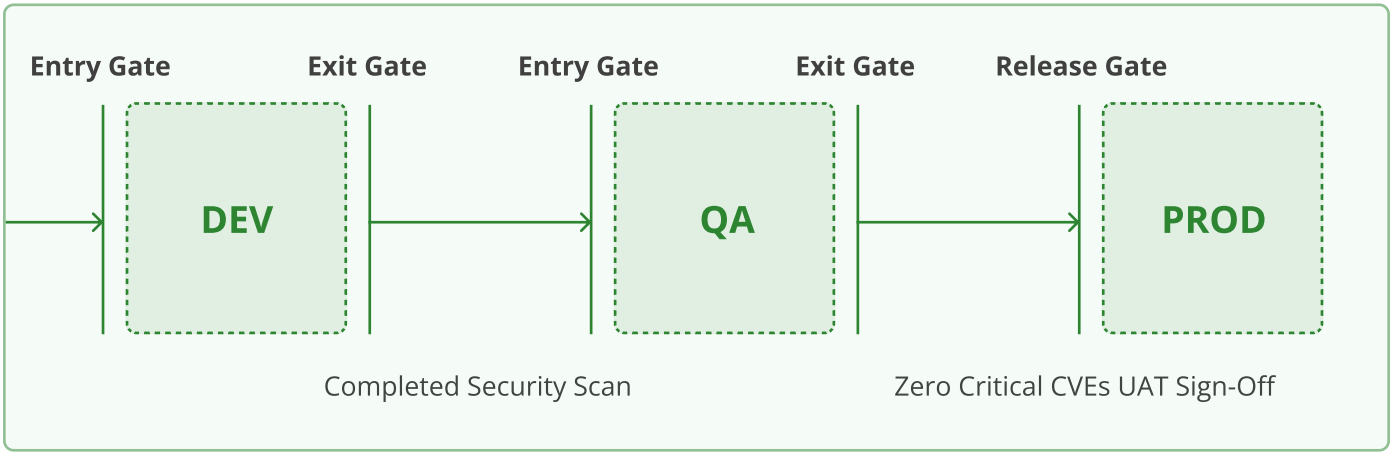

Figure 5: Policy-Based Gates

The Trusted Release Model creates a governed, evidence-based process that builds and proves trust at every step.

A Unified Workflow with “Promotion-to-Commit” GitOps

A governed release process must integrate with automated deployment workflows. The Trusted Release Model bridges governance and deployment with an event-driven “Promotion-to-Commit” architecture. This workflow goes beyond ARO by making a policy-verified event the trigger, not just a promotion step.

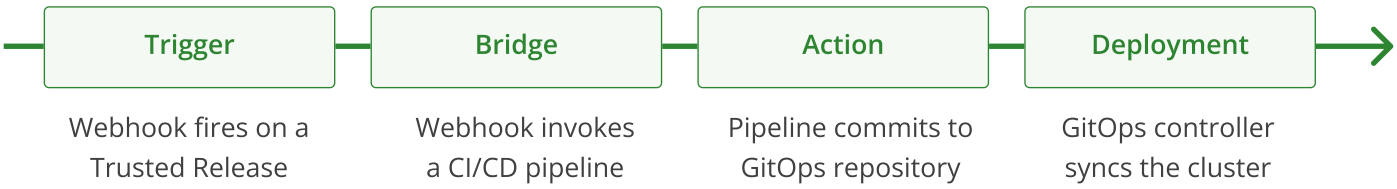

Figure 5: The “Promotion-to-Commit” GitOps Workflow

The workflow includes these key stages:

- **Trigger:** A successful release of an Application Version to the PROD stage, which earns the Trusted Release designation by passing all release gate policies, fires a webhook (e.g., `release_completed`).
- **Bridge:** This webhook invokes a CI/CD pipeline, passing the full context of the trusted release.
- **Action:** The pipeline automatically performs a commit to a designated GitOps repository, updating a configuration file to point to the new, verified Application Version.
- **Deployment:** A GitOps controller (like ArgoCD) detects the change in the Git repository and automatically synchronizes the production Kubernetes cluster to the new state.

This workflow creates an auditable, immutable link between the governance decision and the deployment action within the cluster. It also enriches the Git history with valuable business and governance context.

For example, a commit message like “`Deploying Application 'billing-service' (criticality: high) version 2.5.1, Trusted Release ID: eval-release-e5f6g7h8`” transforms the Git log from a simple record of code changes into a high-level, business-aware audit log of trusted production deployments.

Section V:
The Visibility Layer: Turning Trust into an Observable, Verifiable Entity

The Trusted Release Model makes the abstract concept of “trust” a tangible, observable, and verifiable attribute in the software supply chain. Since governance is ineffective without visibility, this section details how the Model's architecture provides a critical visibility layer for auditors, GRC leaders, and technical teams.

The Activity Log: Audit-Friendly Releases

The Trusted Release Model directly addresses the challenge of evidence collection by providing an Activity Log which acts as an immutable source of truth. It eliminates inefficient manual evidence gathering and fragmented system logs, replacing them with a central, queryable, real-time feed of every significant lifecycle event within the system.

Key log entries include:

- Application creation and modification.
- Detect and record ownership changes.
- The binding and unbinding of packages.
- The creation of every Application Version.
- All successful and failed promotion attempts.
- The detailed results of every Unified Policy evaluation.

This comprehensive log creates an unbreakable chain of evidence for the entire release lifecycle.

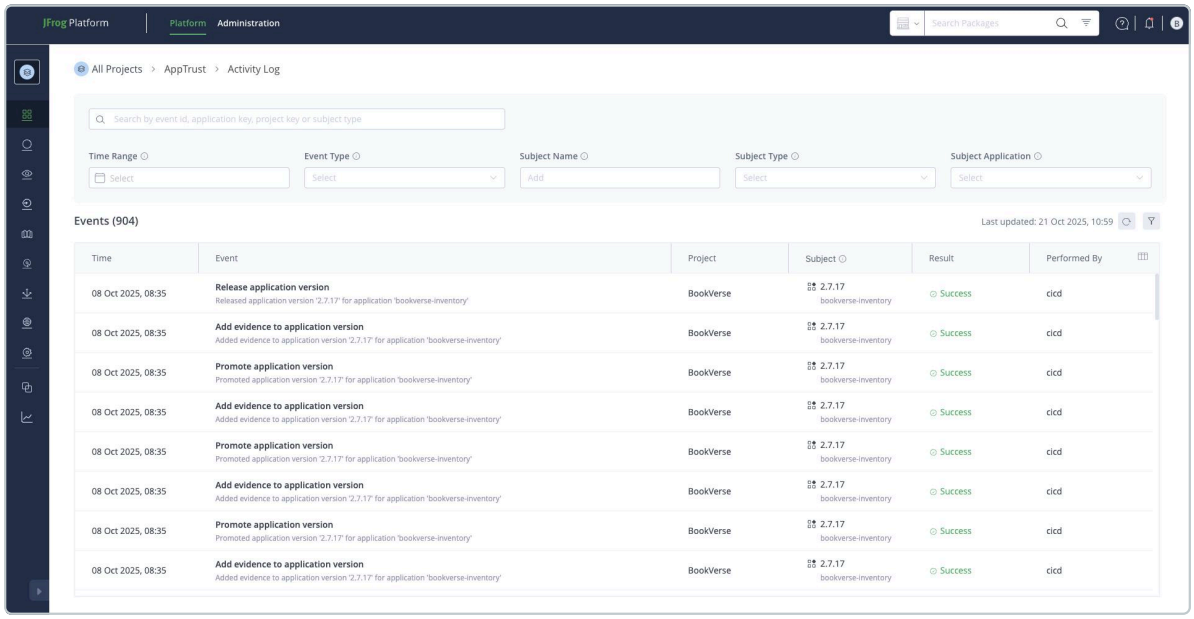

Figure 6: Example of an Activity Log in JFrog AppTrust

How It Works

The Activity Log is an accessible API endpoint, not just a UI feature. Its robust filtering capabilities let auditors and automated compliance systems get a complete, verifiable history of any release with a single API call.

Strategic Insights: Persona-Driven Dashboards for Leadership

The Trusted Release Model translates granular governance data—captured in the Activity Log and Application Objects—into persona-driven views. This critical layer provides leaders with the executive-level insight they need to manage risk, measure performance, and create a continuous strategic feedback loop across the organization.

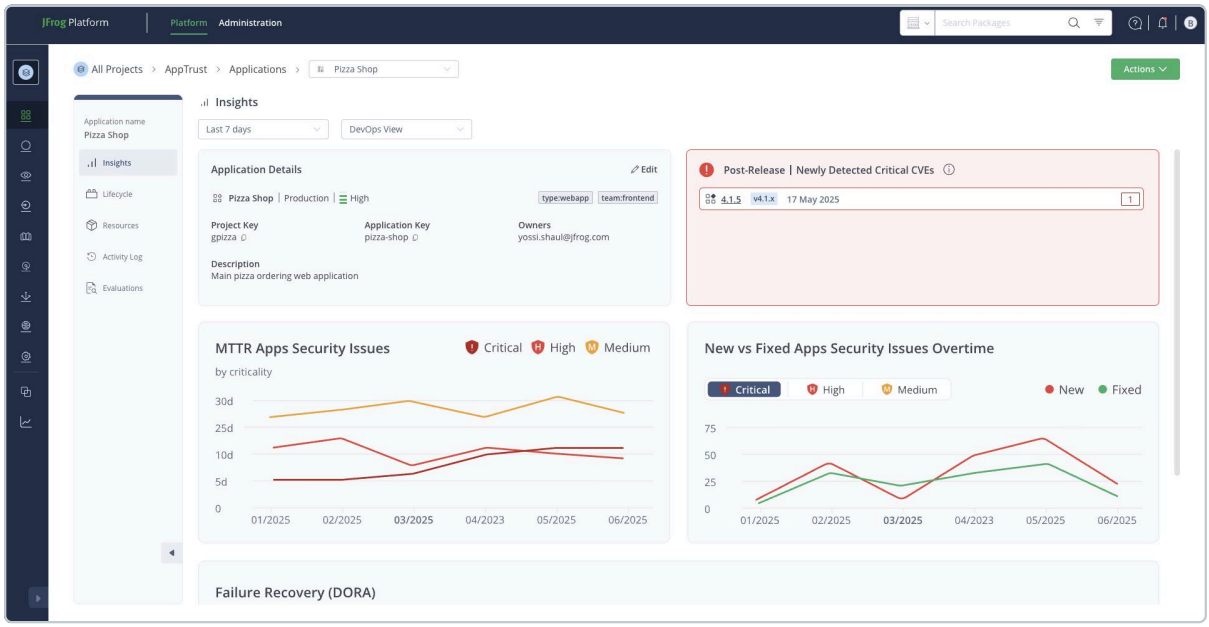

Figure 7: Example of a Dashboard in JFrog AppTrust

DevOps and Platform Engineering Leaders: The dashboard should have DORA metrics, giving them quantitative feedback on how governance policies impact delivery performance. This helps teams find the optimal balance between control and velocity.

Example: A DevOps leader can correlate stricter policies with DORA metrics to make data-driven decisions about their governance strategy.

Security Leaders: Security leaders need aggregated vulnerability summaries and risk trends, helping them move from a ticket-based view of risk to a strategic understanding of their organization's overall security posture.

Example: A Security leader can track application security maturity over time, using those insights to modify policies and proactively address growing security risks.

GRC Leaders and CISOs: They need a clear, enterprise-wide view of the teams and applications global policies cover. This information allows leaders to instantly identify and fix compliance gaps.

Example: A CISO can identify a business unit with low policy adoption and justify targeted training.

This visibility layer unlocks a strategic feedback loop, ensuring the platform becomes a single system for continuous improvement and strategic decision-making by DevOps, Security, and GRC leaders.

Section VI: Future Considerations

Extending the Trusted Release Model

The Trusted Release Model provides a complete framework for release governance, but its architectural vision extends across the entire software development lifecycle.

In the future, the model can extend to unify governance into a single plane, spanning from the developer's first line of code to the application running in production.

Here is the plan for further integration of these stages:

Code Stage

This stage should integrate governance directly into the developer's workflow. By integrating with developer tools, policies can be enforced on code commits and builds, providing immediate feedback on security vulnerabilities or license compliance issues before they ever become part of a formal release candidate. This shifts trust and security even further left.

Runtime Stage

This stage should close the feedback loop between development and operations. By integrating with runtime security and observability solutions, the Trusted Release Model will provide continuous visibility into which specific Application Versions are deployed on which clusters. This enables powerful capabilities, such as automatically correlating a production incident with the exact bill of materials of the running software or identifying all running instances of an application affected by a newly discovered zero-day vulnerability.

This comprehensive vision, enabled by the Trusted Release Model, provides a central governance platform for the entire enterprise software supply chain.

Section VII:
The Trusted Release Model with JFrog AppTrust

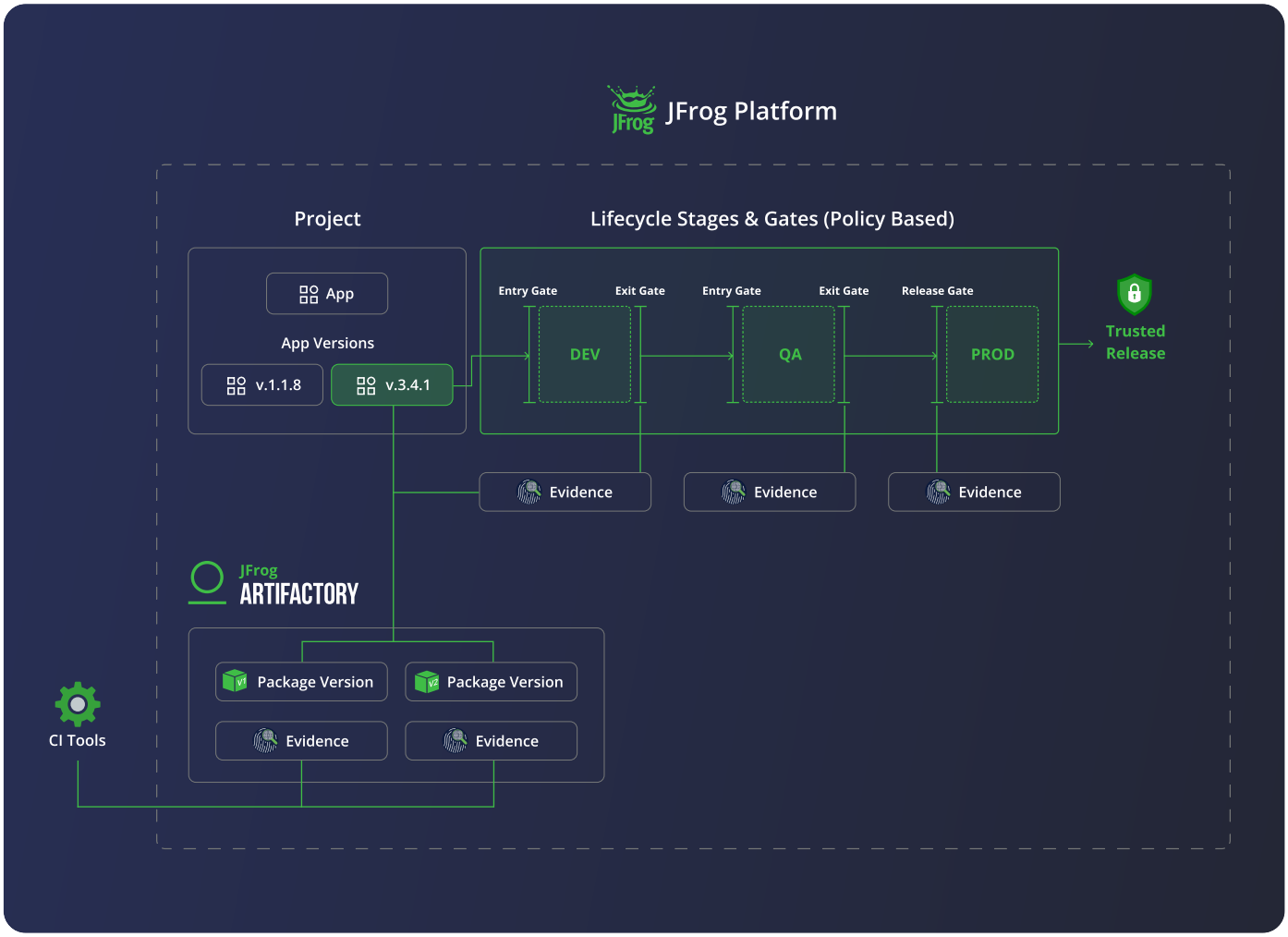

Figure 8: JFrog AppTrust Solution Overview

JFrog AppTrust implements the Trusted Release Model, which solves the dilemma that traditional orchestration solutions can't handle. Application Release Orchestration (ARO) falls short because it treats software as anonymous technical artifacts, forcing risk management and compliance into manual, external workflows.

AppTrust closes the governance gap by fully integrating business context (like ownership and criticality) and a Unified Policy Engine directly into the Software Supply Chain Platform.

Here is how JFrog AppTrust helps you execute every element of the Trusted Release Model:

- **System of Record:** AppTrust establishes a System of Record where every application becomes a business-aware entity.
- **Unit of Release:** The Application Version (the immutable unit of release) inherits this crucial business context.
- **Automated Compliance:** AppTrust's Control Plane enforces a powerful "Proof-of-Compliance" model during promotion across stages of the SDLC.
- **Verifiable Trust:** The model automatically grants Trusted Release status, only after the application passes all evidence-based policy gates. Trust is not just a label - it's verified.
- **Auditability & Visibility:** The platform provides an immutable Activity Log for a complete audit trail, with Persona-Driven Dashboards that instantly translate complex governance data into strategic insights.

AppTrust transforms your software supply chain from a source of risk into a strategic, competitive advantage.

Section VIII: Conclusions & Recommendations

Managing software releases in a large-scale microservices environment presents a fundamental challenge: balancing the speed of autonomy with the need for security and stability. While decentralization helps teams move quickly, the result is often release paralysis and production instability, fueled by siloed data and fragmented workflows. We don't need to abandon autonomy to solve these challenges. Instead, we can balance speed and trust through an automated governance framework.

In this report, we provided a blueprint of the Trusted Release Model, a framework that can be deployed using **JFrog AppTrust** in the **JFrog Software Supply Chain Platform** to give enterprises speed, security, visibility and control over their software releases.

The Trusted Release Model's Core Tenets include:

- **A Synchronized Cadence:** Align technical releases with a business rhythm, like a SAFe Program Increment.
- **Provable Quality:** Define a stable version not by a tag, but by a complete set of signed attestations from automated quality gates.

- **The Atomic Unit of Release:** Bundle entire multi-service applications into a single, immutable Application Version.
- **Automated Governance:** Govern the Application Version's progression with automated Unified Policy gates.
- **Declarative Deployment:** Deploy the approved release candidate with an automated "Promotion-to-Commit" GitOps workflow.

This model transforms release management from a chaotic, high-risk manual effort into a predictable, auditable, and automated strategic function of the business.

Adopting the Trusted Release Model Across the Enterprise: Checklist

This checklist summarizes the key actions to implement the Trusted Release Model in a structured, phased approach, using JFrog AppTrust.

Phase 1: Establish the Foundation (System of Record)

- [ ] Conduct an inventory of all business-critical applications.
- [ ] Onboard these applications into JFrog AppTrust using the Application API, defining their owners, business criticality, and maturity levels.
- [ ] Bind key first-party software packages to their owning applications to build an initial bill of materials.

Phase 2: Implement Governed Lifecycles (Control Plane)

- [ ] Model the organization's SDLC stages within the AppTrust lifecycle configuration (e.g., DEV, QA, STAGING, PROD).
- [ ] Introduce simple, non-blocking (warn) policies for key quality gates (e.g., "Warn if an Xray scan is missing before promoting to QA").
- [ ] Automate the creation of Application Versions as a final step in successful CI pipeline runs.

Phase 3: Automate and Enforce (Operational Blueprint)

- [ ] Mature the governance model by transitioning key policies from warn to fail to actively enforce compliance (e.g., "Block release to PROD if critical CVEs exist").
- [ ] Implement the "Promotion-to-Commit" GitOps workflow for a single pilot application to prove end-to-end automation.

Phase 4: Scale and Observe (Visibility & Governance at Scale)

- [ ] Roll out the standardized GitOps workflow across all application teams.
- [ ] Define and implement global policies to enforce organization-wide security and compliance standards.
- [ ] Integrate the Activity Log API with SIEM and internal auditing tools.

By following this blueprint, enterprises can harness the power of their microservice architecture, resolving the "Speed vs. Trust" dilemma once and for all.

The Trusted Release Model empowers organizations to transform their software supply chain from a source of risk into a strategic competitive advantage, delivering solutions to market faster, with more confidence and verifiable trust than ever before.

Learn how **JFrog AppTrust** can help your organization transition to a Trusted Release Model by **booking a demo** today!