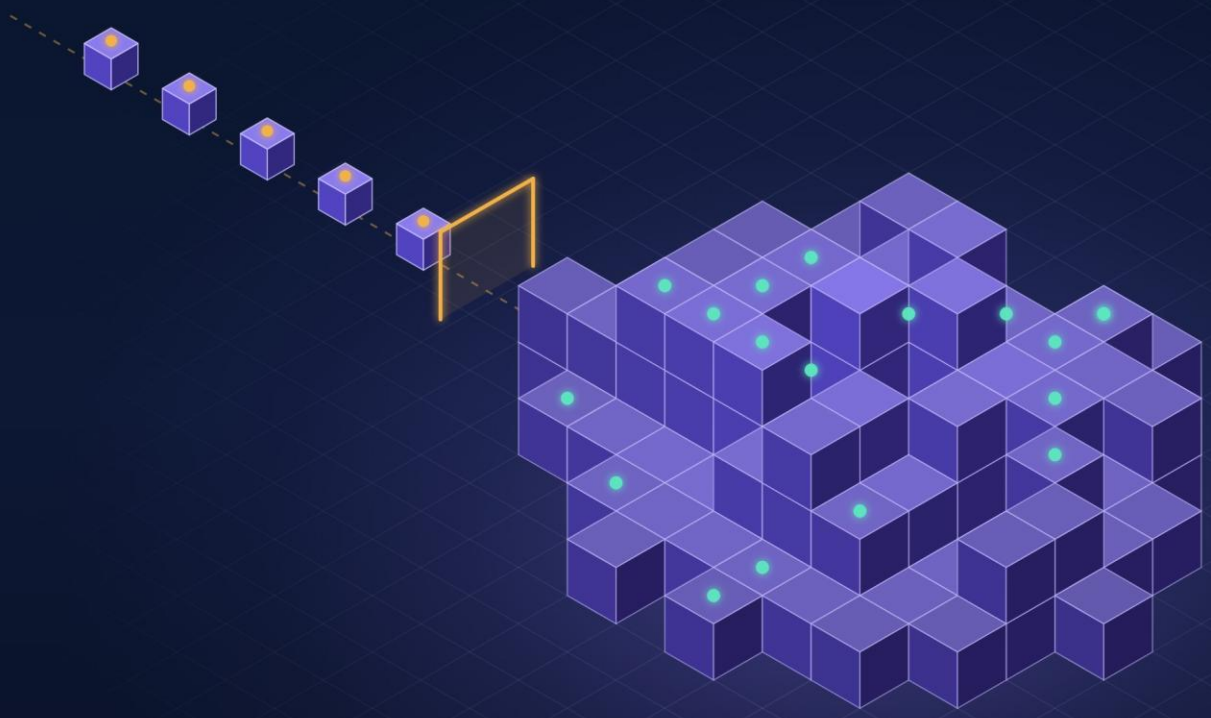
SURVEY PAPER

Cloud-Native Security

A Comprehensive Survey of **Software Supply-Chain Integrity**, Container and Kubernetes Hardening, Runtime Protection, and Cloud-Native Application Protection Platforms

SLSA v1.2 | NIST SP 800-190 | NIST SP 800-204D



DECEMBER 2026

BUILD · SHIP · RUN
SUPPLY CHAIN · CLUSTER · RUNTIME

SURVEY PAPER · DECEMBER 2026

Cloud-Native Security: A Comprehensive Survey of Software Supply-Chain Integrity, Container and Kubernetes Hardening, Runtime Protection, and Cloud-Native Application Protection Platforms

Front-code.com

***Abstract*—**Cloud-native computing—containers, microservices, service meshes, immutable infrastructure, and declarative APIs orchestrated by Kubernetes—has become the default way to build and operate software, and it has reshaped the attack surface along with it. Workloads are ephemeral and numerous, infrastructure is defined as code, and every deployment pulls in hundreds of third-party components through automated pipelines. Attackers have followed: compromised build systems, poisoned open-source packages, self-propagating registry worms, leaked pipeline secrets, exposed cluster control planes, and container-escape vulnerabilities now feature prominently in incident reports. This paper presents a comprehensive survey of cloud-native security organized around the application life cycle. We trace the evolution of the field from early container adoption to the 2026 regulatory landscape; formalize the lifecycle and layered threat model; analyze software supply-chain integrity through SLSA provenance, in-toto attestations, Sigstore keyless signing, and software bills of materials; examine image and container hardening, including user namespaces, sandboxed runtimes, and risk-based vulnerability prioritization with EPSS; survey Kubernetes platform security, from RBAC and Pod Security Standards to admission policy as code, network segmentation, and workload identity; review eBPF-based runtime detection and response; and describe the convergence of posture-management tools into cloud-native application protection platforms (CNAPP). We map prevalent attack techniques and recent incidents to concrete controls, compare the principal standards and regulations, including NIST SP 800-190, the SP 800-204 series, and the EU Cyber Resilience Act, and construct a phased, metrics-driven adoption roadmap.

***Index Terms*—**Cloud-native security, Kubernetes, containers, software supply chain, SLSA, Sigstore, in-toto, SBOM, admission control, Pod Security Standards, eBPF, runtime security, workload identity, CNAPP, Cyber Resilience Act.

I. INTRODUCTION

The Cloud Native Computing Foundation (CNCF) describes cloud-native technologies as those that let organizations build and run scalable applications in dynamic environments, exemplified by containers, service meshes, microservices, immutable infrastructure, and declarative APIs [1]. A decade after Kubernetes 1.0, this model is the default for new software. It brings speed and resilience, but it also changes what must be secured: workloads live for minutes rather than years, infrastructure is code, a single cluster may host thousands of services, and every build assembles hundreds of open-source components through automated pipelines.

Security has struggled to keep pace. In Red Hat's 2024 survey of 600 practitioners, 67% of organizations had delayed or slowed deployments because of Kubernetes security concerns, nearly nine in ten had

experienced at least one container or Kubernetes security incident in the previous year, and 46% reported revenue or customer loss as a result [2]. GitGuardian counted 28.65 million new hard-coded secrets in public GitHub commits during 2025, a 34% increase over the previous year [3]. In March 2025, the "IngressNightmare" vulnerabilities in the widely deployed ingress-nginx controller exposed an estimated 43% of cloud environments to cluster takeover [4]; in September 2025, the self-replicating "Shai-Hulud" worm spread through the npm registry by stealing maintainers' tokens and republishing their packages [5].

> **The core principle: secure the pipeline, verify at the gate, watch at runtime**
> Cloud-native security cannot be bolted onto a running cluster. Controls must follow the artifact from source to production: integrity is established in the pipeline, enforced when a workload is admitted to the cluster, and verified continuously while it runs—with every step expressed as code, versioned, and auditable.

Section II traces the evolution of the field and its threat model; Section III covers the software supply chain; Section IV, image and container hardening; Section V, Kubernetes platform security; Section VI, runtime detection; and Section VII, cloud posture management. Section VIII maps attacks to controls, Section IX compares standards and regulation, Sections X and XI give challenges and a roadmap, and Sections XII and XIII close with research directions and conclusions.

II. EVOLUTION AND THREAT MODEL

A. From Containers to Regulation

Linux containers became mainstream with Docker in 2013 and Kubernetes 1.0 in 2015. NIST's *Application Container Security Guide*, SP 800-190, codified the early threat model in 2017: image vulnerabilities and embedded secrets, untrusted registries, unbounded orchestrator access, and a shared host kernel [6]. The incidents that followed shaped the discipline. In 2018, attackers found a Kubernetes console without a password at Tesla and used it for cryptomining and to reach cloud credentials [7]; in 2019, a server-side request forgery against a misconfigured firewall let an attacker query the cloud instance metadata service and exfiltrate data on about 100 million Capital One customers [8]; and the SolarWinds build-system compromise of 2020 made the software supply chain a national-security issue [9].

Guidance followed: the NSA and CISA *Kubernetes Hardening Guide* [10], the CNCF *Cloud Native Security Whitepaper* [11], NIST's SP 800-204 series on microservices and DevSecOps pipelines [12], and the SLSA framework for build integrity [13]. By 2026 the field had reached regulation: the EU Cyber Resilience Act (CRA) requires manufacturers of products with digital elements to report actively exploited vulnerabilities and severe incidents from 11 September 2026, with its remaining obligations applying from 11 December 2027 [14]. Fig. 1 summarizes this progression.

B. The Layered Model and Shared Responsibility

The Kubernetes documentation popularized a layered view of cloud-native security—cloud, cluster, container, and code—in which each layer builds on the security of the one beneath it [15]. Responsibility for those layers is shared: the cloud provider secures the physical infrastructure and, for managed Kubernetes, the control plane; the customer remains responsible for identity and access configuration, network exposure, node and workload settings, images, and application code. Most cloud-native breaches occur on the customer's side of that line, through misconfiguration and stolen credentials rather than flaws in the provider's infrastructure.

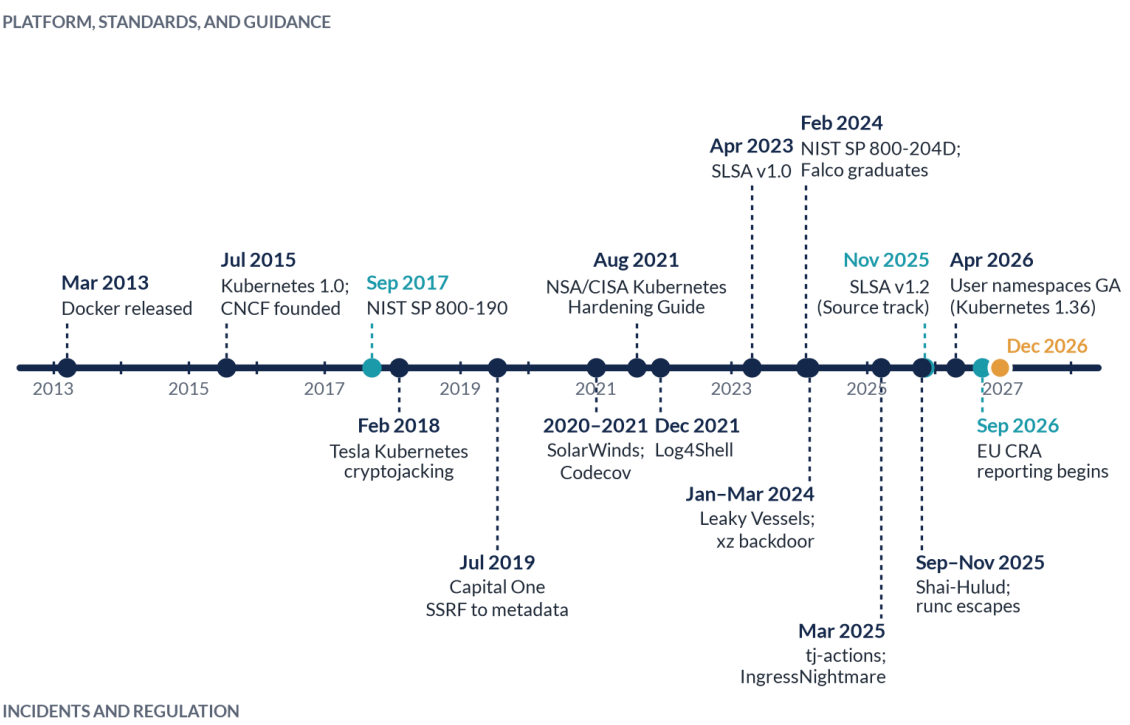


Fig. 1. Evolution of cloud-native security, 2013–2027. Platform milestones and standards are shown above the axis and major incidents below it.

C. The Life-Cycle View

The CNCF whitepaper organizes controls by life-cycle phase rather than by layer [11]: *Develop* (secure code, dependency hygiene, infrastructure-as-code scanning), *Distribute* (build integrity, signing, image scanning, registries), *Deploy* (admission checks, configuration policy, verification of signatures and provenance), and *Runtime* (workload isolation, network policy, detection and response). Findings at runtime feed back into development. Fig. 2 shows the phases, the principal controls in each, and the cross-cutting capabilities—identity, policy as code, observability, and posture management—that span all four.

III. SOFTWARE SUPPLY-CHAIN INTEGRITY

A. Threats

A cloud-native artifact is the product of a chain: source repositories, third-party dependencies, build systems, registries, and deployment tooling. Each link can be attacked. Table I lists representative incidents. They span compromised build systems (SolarWinds), tampered pipeline tooling (Codecov in 2021 and the tj-actions/changed-files GitHub Action in 2025 [16], [17]), vulnerable transitive dependencies (Log4Shell [18]), long-term social engineering of an open-source maintainer that planted a backdoor in xz-utils [19], and registry worms that spread through stolen publishing tokens [5]. OWASP elevated *Software Supply Chain Failures* to third place in its 2025 Top 10 [20].

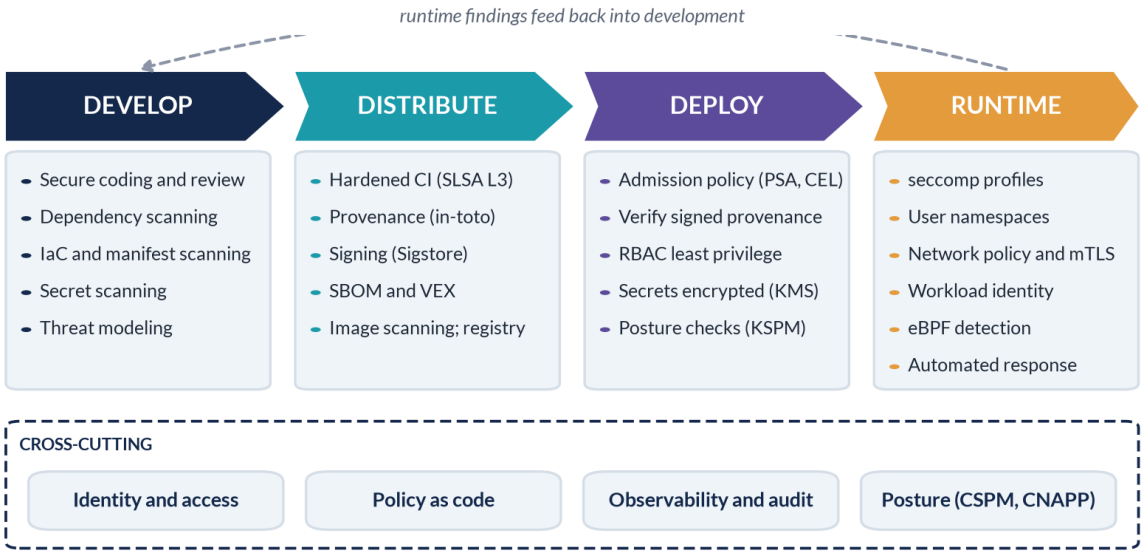


Fig. 2. Cloud-native security across the application life cycle, after the CNCF Cloud Native Security Whitepaper [11]. Each phase adds controls; runtime findings feed back into development.

Table I. Representative software supply-chain incidents and the controls that address them

Incident	Vector	Consequence	Controls that would have helped
SolarWinds Orion, 2020 [9]	Malicious code injected into the vendor's build process	Signed, trojanized updates reached up to 18,000 customers	Hardened, isolated builds (SLSA Build L3); independent provenance; reproducible builds
Codecov, 2021 [16]	Modified Bash uploader script fetched by CI jobs	CI secrets and tokens exfiltrated	Verify checksums and signatures of fetched tools; short-lived, scoped CI credentials
Log4Shell, 2021 [18]	Critical flaw in a ubiquitous transitive dependency	Mass exploitation; weeks spent finding affected systems	SBOMs for rapid impact analysis; VEX; reachability-aware prioritization
xz-utils, 2024 [19]	Multi-year maintainer takeover; backdoor in release tarballs	Near-miss SSH backdoor in major Linux distributions	Source-to-artifact provenance; reproducible builds; scrutiny of release artifacts
tj-actions/changed-files, 2025 [17]	Mutable version tags repointed to malicious commit	Secrets printed to public CI logs	Pin actions by commit SHA; least-privilege workflow tokens; egress control on runners
Shai-Hulud npm worm, 2025 [5]	Stolen npm tokens used to republish packages with a credential-stealing payload	Hundreds of packages and thousands of repositories compromised	Trusted publishing (OIDC) with provenance; mandatory 2FA; short-lived tokens; disabled install scripts

Counts are as reported by the cited advisories and may have been revised since.

B. Provenance, Attestation, and Signing

The SLSA framework (Supply-chain Levels for Software Artifacts) defines incremental guarantees against these threats [13]. Its Build track ranges from Level 1, at which provenance describing how an artifact was built exists, through Level 2, which requires a hosted build platform that signs the provenance, to Level 3, which requires hardened builds in which runs are isolated from one another and signing keys are inaccessible to user-defined build steps. Version 1.2, released in November 2025, added a Source track covering revision history, source provenance, and continuously enforced repository controls. SLSA provenance is expressed as an in-toto attestation: a signed statement binding an artifact's digest to claims about how it was produced [21].

Signing makes these claims verifiable. Sigstore removes the burden of long-lived signing keys: Fulcio issues short-lived certificates bound to an OpenID Connect identity, such as a CI workflow, the signature is recorded in the Rekor transparency log, and verifiers check the identity and the log entry rather than a

static public key [22]. Package ecosystems have adopted the model: npm and PyPI support trusted publishing from CI with provenance, and after Shai-Hulud, GitHub announced that npm publishing would move toward trusted publishing, mandatory two-factor authentication, and short-lived tokens [23]. Fig. 3 shows the resulting chain from commit to admission.

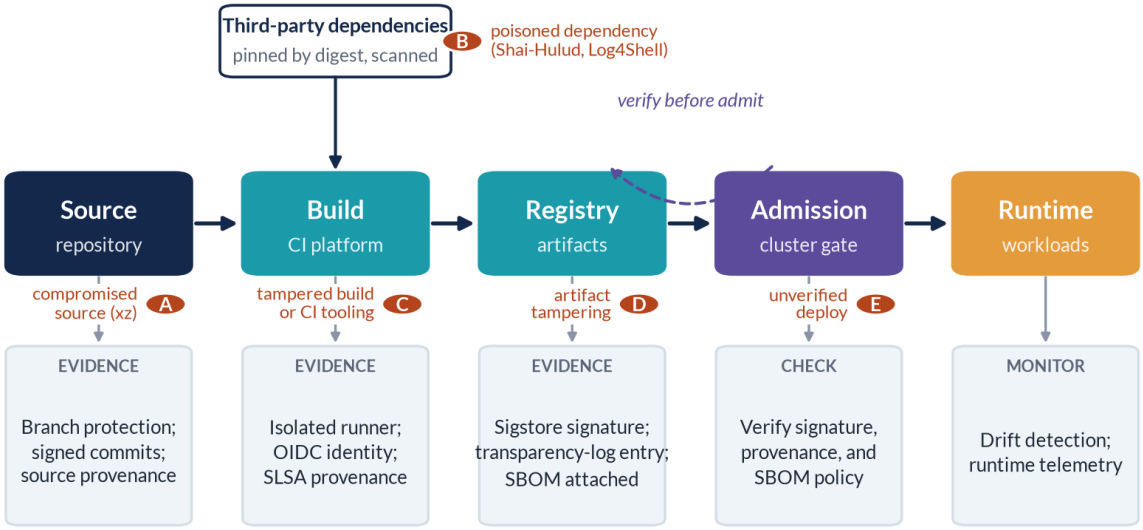


Fig. 3. A verifiable software supply chain. Each stage emits signed evidence—source provenance, build provenance, signatures, and SBOMs—that the cluster verifies at admission; red markers show where the incidents of Table I struck.

Layered controls compound. If a pipeline applies n independent checks and check k detects a given tampering attempt with probability d_k, the probability that the attempt passes undetected is given by (1). Three checks that are each 70% effective leave a 2.7% residual risk; the independence assumption is optimistic, which is why the controls should target different stages—source, build, and deployment—rather than repeat the same test.

$$P_{\text{miss}} = \prod_{k=1}^{n} (1 - d_k) \tag{1}$$

C. Transparency: SBOM, VEX, and Regulation

A software bill of materials (SBOM) lists the components of an artifact, typically in the SPDX format, standardized as ISO/IEC 5962 [24], or CycloneDX, standardized as ECMA-424 [25]. Vulnerability Exploitability eXchange (VEX) documents complement SBOMs by stating whether a known vulnerability actually affects a product, which is what turns an SBOM from a list into an answer during events such as Log4Shell. CISA published an updated draft of the minimum elements of an SBOM for public comment in August 2025 [26], and the CRA requires manufacturers to identify and document the components of their products, including by drawing up an SBOM [14]. NIST's Secure Software Development Framework (SSDF) [27] and SP 800-204D, which maps supply-chain controls onto CI/CD pipelines [12], provide the process context.

D. Pipeline and Secrets Hygiene

The pipeline itself is a privileged system. Long-lived cloud keys stored as CI secrets should be replaced by OpenID Connect federation, in which the CI platform issues a short-lived, audience-restricted token that the cloud exchanges for temporary credentials. Third-party actions and images should be pinned by digest, workflow tokens scoped to the minimum permissions, and build runners made ephemeral with restricted egress. Secret scanning must cover not only source code but also logs, container layers, and configuration files; GitGuardian's 2026 report found more than 24,000 unique secrets in Model Context Protocol configuration files used by AI agents alone [3].

IV. IMAGE AND CONTAINER HARDENING

A. Minimal, Immutable Images

Every package in an image is attack surface. Minimal and "distroless" base images that contain only the application and its runtime dependencies shrink both the vulnerability count and an intruder's toolkit, since no shell or package manager is present. Containers should run as a non-root user with a read-only root file system, all Linux capabilities dropped except those explicitly required, privilege escalation disabled, and the runtime's default seccomp profile applied; AppArmor or SELinux adds mandatory access control [6], [10]. Because cloud-native workloads are immutable, any binary executed that was not in the original image—*drift*—is a strong signal of compromise (Section VI).

B. Isolation and Container Escape

Containers share the host kernel, so a runtime or kernel flaw can turn a compromised container into a compromised node. Container-runtime escapes recur: CVE-2019-5736 let a malicious container overwrite the host's runc binary; the 2024 "Leaky Vessels" flaw, CVE-2024-21626, leaked a host file descriptor into containers [28]; and three runc vulnerabilities disclosed in November 2025 could be chained to gain root on the host through malicious mount configurations [29]. Defenses come in depth. User namespaces map container root to an unprivileged host user and reached general availability in Kubernetes 1.36 in April 2026 [30]; rootless runtimes extend the same idea; sandboxed runtimes such as gVisor, which interposes a user-space kernel, and Kata Containers, which uses lightweight virtual machines, trade some performance for a much smaller host attack surface [31]; and confidential containers protect workloads from the infrastructure operator itself.

C. Risk-Based Vulnerability Management

Image scanners routinely report hundreds of CVEs per cluster, most of which are neither reachable nor exploited. Prioritization must therefore combine severity with likelihood and context. The Exploit Prediction Scoring System (EPSS) estimates the probability that a vulnerability will be exploited in the next 30 days [32], and CISA's Known Exploited Vulnerabilities (KEV) catalog lists those that already have been [33]. A practical score multiplies the exploitation probability by the impact $I(v)$, a reachability factor $\rho(v) \in [0, 1]$ indicating whether the vulnerable code is loaded and invoked, and an exposure factor $\varepsilon(v) \in [0, 1]$ reflecting network exposure and privileges, as in (2); KEV entries override the score. The risk that a workload w contains at least one exploitable, reachable flaw then follows (3), which explains why a single internet-exposed image with a reachable, actively exploited CVE outweighs hundreds of dormant findings.

$$R(v) = \mathrm{EPSS}(v) \cdot I(v) \cdot \rho(v) \cdot \varepsilon(v) \tag{2}$$

$$R(w) = 1 - \prod_{v \in V(w)} [1 - \mathrm{EPSS}(v) \cdot \rho(v) \cdot \varepsilon(v)] \tag{3}$$

V. KUBERNETES PLATFORM SECURITY

A. Control Plane and Node Hardening

The Kubernetes API server is the cluster's front door, and its compromise is total. It should not be exposed to the internet; anonymous authentication should be disabled; access should be authorized through role-based access control (RBAC) with least privilege; and audit logging should be enabled and shipped off-cluster [10], [34]. Secrets should be encrypted at rest in etcd with a key-management service rather than stored only base64-encoded, and etcd itself should be reachable only by the API server. On nodes, the kubelet API must require authentication and authorization. Fig. 4 maps the principal attack paths through a cluster.

RBAC deserves particular care, because a few permissions are equivalent to cluster-admin: the ability to create pods in a privileged namespace, to bind roles, to impersonate users, or to read secrets cluster-wide. Since Kubernetes 1.24, long-lived secret-based service-account tokens are no longer generated automatically; workloads receive projected tokens that are short-lived and audience-bound, and pods that do not call the API should not mount a token at all. Equation (4) defines a simple least-privilege metric for an identity s, the fraction of granted permissions actually used over an observation window, which cloud infrastructure entitlement management (CIEM) tools use to recommend tighter roles.

$$u(s) = \frac{|P_{\mathrm{used}}(s)|}{|P_{\mathrm{granted}}(s)|} \tag{4}$$

B. Admission Control and Policy as Code

Admission control is where the supply chain meets the cluster. Pod Security Admission, which replaced PodSecurityPolicy when the latter was removed in Kubernetes 1.25, enforces the three Pod Security Standards—*Privileged*, *Baseline*, and *Restricted*—per namespace, in enforce, audit, or warn mode [35]. ValidatingAdmissionPolicy, generally available since Kubernetes 1.30, evaluates Common Expression Language rules inside the API server without an external webhook [36]. For richer policy, OPA Gatekeeper and Kyverno, which graduated in the CNCF in March 2026 [37], express rules as code and can require that every image carry a valid Sigstore signature and SLSA provenance from an approved builder. Admission webhooks are themselves attack surface: IngressNightmare was exploitable because the ingress controller's admission endpoint was reachable from the pod network [4].

C. Network Segmentation and Workload Identity

By default, every pod can reach every other pod. Namespaces should start from a default-deny NetworkPolicy and allow only required flows, with egress restricted as well as ingress; eBPF-based networking such as Cilium adds identity-aware, layer-7 policy. Service meshes provide mutual TLS between services and identity-tier authorization, applying the zero trust model of our companion survey to east–west traffic [38]. Workloads should authenticate to cloud APIs through workload identity federation and attested identities such as SPIFFE SVIDs [39], never through static keys mounted as

secrets. Where nodes run on cloud virtual machines, the instance metadata service must require session tokens (IMDSv2 on AWS) with a hop limit that prevents pods from reaching node credentials—the defense introduced after the Capital One breach [40].

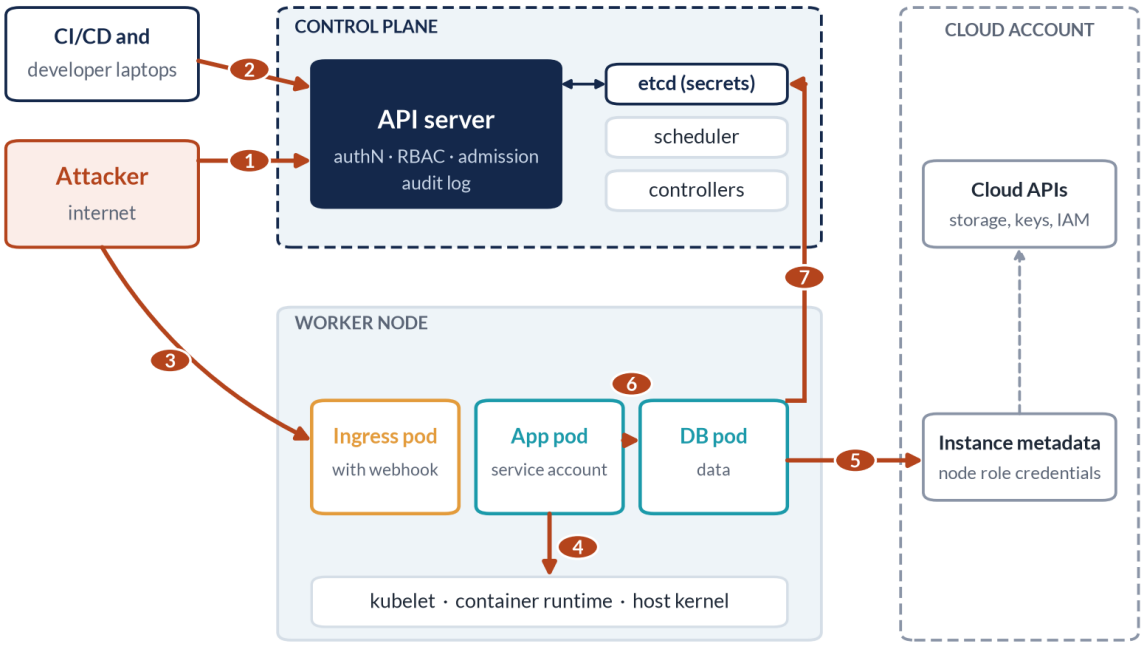


Fig. 4. Principal attack paths through a Kubernetes cluster: (1) exposed API server or dashboard, (2) stolen kubeconfig or service-account token, (3) vulnerable ingress or admission webhook, (4) container escape to the node, (5) theft of cloud credentials from the instance metadata service, (6) lateral movement across a flat pod network, and (7) direct access to secrets in etcd.

VI. RUNTIME DETECTION AND RESPONSE

Prevention fails eventually, and cloud-native workloads are too short-lived for traditional host agents and forensics. The extended Berkeley Packet Filter (eBPF) lets security tools attach verified programs to kernel events—system calls, process execution, file access, and network connections—with low overhead and without kernel modules. Falco, which graduated in the CNCF in February 2024, evaluates rules over this stream and over Kubernetes audit events [41]; Tetragon adds in-kernel filtering and can terminate an offending process synchronously. Typical detections include a shell spawned in a container, execution of a binary not present in the image, writes below system directories, reads of service-account tokens by unexpected processes, outbound connections to mining pools, and changes to RBAC bindings. Detections should map to the containers matrix of MITRE ATT&CK [42] and to Microsoft's threat matrix for Kubernetes [43]. Response actions—isolating a pod with a network policy, cordoning a node, revoking credentials, and capturing state before the container disappears—should be automated, because the evidence is ephemeral.

VII. CLOUD POSTURE AND CNAPP

Below the cluster lies the cloud account, where misconfiguration remains a leading cause of breaches. Cloud security posture management (CSPM) continuously checks account configuration against benchmarks; Kubernetes security posture management (KSPM) does the same for clusters; cloud workload protection platforms (CWPP) cover vulnerability scanning and runtime defense; CIEM analyzes

identities and entitlements; and data security posture management locates sensitive data. Gartner's term *cloud-native application protection platform* (CNAPP) describes the convergence of these functions into one platform that correlates code, pipeline, configuration, identity, and runtime signals [44]. The value lies in correlation: an attack-path graph can show that a publicly exposed pod runs an image with a reachable KEV-listed vulnerability and a service account that can read production secrets—one prioritized finding instead of three unrelated alerts.

VIII. ATTACK TECHNIQUES AND COUNTERMEASURES

Table II maps prevalent techniques against Kubernetes environments, organized by ATT&CK tactic [42], [43], to representative incidents and to preventive and detective controls.

Table II. Kubernetes attack techniques, representative incidents, and controls

Technique (tactic)	Representative incident	Prevention	Detection
Exposed API server or dashboard (initial access)	Tesla console, 2018 [7]	Private endpoints; no anonymous auth; SSO with MFA for humans	Audit-log anomalies; external attack-surface scans
Vulnerable ingress or admission webhook (initial access)	IngressNightmare, 2025 [4]	Prompt patching; NetworkPolicy limiting webhooks to the API server	Unexpected configuration loads; anomalous controller processes
Malicious or unverified image (execution)	Shai-Hulud payloads in builds, 2025 [5]	Admission requires signatures and provenance from trusted builders	Drift detection; unknown binaries at runtime
Privileged pod or host mount (privilege escalation)	Cryptomining campaigns abusing privileged pods	Pod Security Standard *Restricted*; user namespaces	Admission audit mode; Falco rules on host mounts
Container escape via runtime flaw (privilege escalation)	Leaky Vessels, 2024 [28]; runc, 2025 [29]	Patched runtimes; user namespaces; seccomp; sandboxed runtimes	Syscall anomalies; processes escaping namespaces
Token or metadata credential theft (credential access)	Capital One, 2019 [8]	No token automount; bound tokens; IMDSv2 with hop limit; workload identity	Token use from unexpected IPs; metadata access from pods
Flat pod network (lateral movement)	Post-compromise scanning in shared clusters	Default-deny NetworkPolicy; mTLS with identity-tier authorization	Flow logs; new east–west connections
Resource hijacking (impact)	Cryptojacking in exposed clusters [7]	Resource quotas; egress allow-lists	CPU anomalies; mining-pool connections

Tactics follow the MITRE ATT&CK containers matrix and Microsoft's threat matrix for Kubernetes [42], [43].

IX. STANDARDS, GUIDANCE, AND REGULATION

Table III compares the principal documents. NIST SP 800-190 remains the foundational container threat model [6]; the SP 800-204 series extends it to microservices, service meshes, and DevSecOps pipelines [12]; the NSA/CISA guide and CIS benchmark give configuration baselines [10], [34]; SLSA and the SSDF define integrity and process [13], [27]; and the CRA turns secure development, vulnerability handling, and reporting into legal obligations for products sold in the EU [14].

Table III. Principal standards, guidance, and regulation for cloud-native security

Document	Issuer, year	Scope	Key requirements
NIST SP 800-190	NIST, 2017	Container threat model	Image, registry, orchestrator, container, and host countermeasures
NIST SP 800-204 to 204D	NIST, 2019–2024	Microservices, service mesh, DevSecOps	Service-mesh security; ABAC; supply-chain controls in CI/CD
Kubernetes Hardening Guide	NSA and CISA, 2022	Kubernetes clusters	Non-root pods, network separation, authentication, logging, upgrades
CIS Kubernetes Benchmark	CIS, ongoing	Cluster configuration	Auditable settings for control plane, etcd, kubelet, and policies
Cloud Native Security Whitepaper	CNCF, 2022	Life-cycle controls	Develop, distribute, deploy, and runtime practices
SLSA 1.2	OpenSSF, 2025	Build and source integrity	Provenance; hardened, isolated builds; source track
SSDF (SP 800-218)	NIST, 2022	Secure development process	Prepare, protect, produce, and respond practice groups
Cyber Resilience Act	EU, 2024	Products with digital elements	Secure by design; SBOM; vulnerability handling; reporting from Sep. 2026

SSDF = Secure Software Development Framework; ABAC = attribute-based access control.

X. IMPLEMENTATION CHALLENGES

Signal overload. Scanners and posture tools generate far more findings than teams can fix; without exploitability, reachability, and exposure context, as in (2) and (3), they erode trust and slow delivery. **Developer friction.** Controls that block deployments without clear remediation invite exceptions; policies should start in audit mode, ship with fixes, and be enforced gradually. **Ephemeral evidence.** Pods vanish before investigators arrive, so logging, runtime telemetry, and state capture must be designed in. **Multi-cloud identity sprawl.** Each provider has its own identity model; federation and a single inventory of non-human identities are prerequisites for least privilege. **AI workloads.** Model files can embed executable code in serialized formats, GPU operators run with high privileges, and agent tooling carries new secrets, extending the supply chain to models and datasets. **Cryptographic agility.** Image signatures, provenance, and transparency logs depend on today's elliptic-curve signatures and must eventually migrate to post-quantum schemes such as ML-DSA [45].

XI. ADOPTION ROADMAP

The roadmap below synthesizes the CNCF life cycle, SLSA, and the hardening baselines into phases that can be adopted incrementally; durations are indicative.

1. **Inventory and baseline (months 0–3).** Inventory clusters, registries, pipelines, cloud accounts, and non-human identities; run CIS and KSPM checks; close internet-exposed control planes.
2. **Secure the pipeline (months 2–9).** Replace stored cloud keys with OIDC federation; pin dependencies and actions by digest; enable secret scanning; generate SBOMs and SLSA Build L2 provenance.
3. **Gate the cluster (months 6–12).** Enforce Pod Security *Baseline*, then *Restricted*; require signed images with provenance at admission; encrypt secrets with a KMS; tighten RBAC.
4. **Segment and isolate (months 9–18).** Apply default-deny network policy and mTLS; adopt workload identity; enable user namespaces and sandboxed runtimes for untrusted workloads.

5. **Detect and respond (months 12–24).** Deploy eBPF runtime sensors; map detections to ATT&CK; automate containment; correlate posture and runtime findings in a CNAPP.

Progress should be measured by outcomes: the share of production images with verified signatures and provenance; the share of namespaces enforcing the *Restricted* standard; median time to remediate KEV-listed vulnerabilities in running images; the number of static cloud keys remaining in pipelines; and privilege utilization $u(s)$ from (4) for the riskiest identities.

Executive recommendation: immediate action items

1. Remove internet exposure of Kubernetes API servers, dashboards, and admission webhooks.
2. Replace long-lived cloud keys in CI/CD with OIDC federation, and pin third-party actions by commit SHA.
3. Require signed images and build provenance at admission for production namespaces.
4. Prioritize vulnerabilities by KEV, EPSS, reachability, and exposure rather than by CVSS alone.
5. Deploy runtime detection for drift, shells in containers, and credential access.

XII. FUTURE RESEARCH DIRECTIONS

AI supply chains need provenance and SBOM equivalents for models, datasets, and agent tool configurations, and safe loading of serialized models. **Verifiable builds at scale** remain hard: reproducible builds for large dependency graphs and practical SLSA Build L3 for every artifact are still open problems. **Policy verification** across admission, network, and identity layers could prove that a cluster cannot reach an unsafe state rather than merely testing for known misconfigurations. **Kernel-level isolation** may change with user namespaces, sandboxed runtimes, and confidential computing, but measuring the residual shared-kernel risk is unsolved. **Post-quantum signing** for Sigstore, container signatures, and transparency logs must fit existing size and performance budgets [45].

XIII. CONCLUSION

Cloud-native computing moved the security boundary from the data center to the pipeline, the cluster API, and the identities that connect them. The most damaging incidents of recent years—compromised builds, poisoned packages, leaked pipeline secrets, exposed control planes, and container escapes—exploit exactly these points. The defenses are now mature and largely open: SLSA provenance, Sigstore signatures, and SBOMs make artifacts verifiable; Pod Security Standards, admission policy, network policy, user namespaces, and workload identity harden the platform; eBPF sensors provide runtime visibility; and CNAPPs correlate the signals. With the EU Cyber Resilience Act now in force, these practices are becoming obligations as well as best practice. Organizations that secure the pipeline, verify at the gate, and watch at runtime will be able to move at cloud-native speed without inheriting cloud-native risk.

REFERENCES

[1] Cloud Native Computing Foundation, “CNCF cloud native definition v1.0,” 2018.

[2] Red Hat, “The state of Kubernetes security report: 2024 edition,” Jun. 2024.

[3] GitGuardian, “The state of secrets sprawl 2026,” Mar. 2026.

[4] Wiz Research, “IngressNightmare: 9.8 critical unauthenticated remote code execution vulnerabilities in Ingress NGINX,” Wiz Blog, Mar. 2025.

[5] Cybersecurity and Infrastructure Security Agency (CISA), “Widespread supply chain compromise impacting npm ecosystem,” alert, Sep. 2025.

[6] M. Souppaya, J. Morello, and K. Scarfone, “Application container security guide,” NIST SP 800-190, Sep. 2017.

[7] RedLock Cloud Security Intelligence Team, “Lessons from the cryptojacking attack at Tesla,” RedLock Blog, Feb. 2018.

[8] Capital One, “Information on the Capital One cyber incident,” Jul. 2019.

[9] Cybersecurity and Infrastructure Security Agency (CISA), “Advanced persistent threat compromise of government agencies, critical infrastructure, and private sector organizations,” Alert AA20-352A, Dec. 2020.

[10] National Security Agency and Cybersecurity and Infrastructure Security Agency, “Kubernetes hardening guide,” version 1.2, Aug. 2022.

[11] CNCF TAG Security, “Cloud native security whitepaper,” version 2, May 2022.

[12] R. Chandramouli, F. Kautz, and S. Torres-Arias, “Strategies for the integration of software supply chain security in DevSecOps CI/CD pipelines,” NIST SP 800-204D, Feb. 2024.

[13] Open Source Security Foundation, “Supply-chain Levels for Software Artifacts (SLSA) specification,” version 1.2, Nov. 2025.

[14] European Parliament and Council of the European Union, “Regulation (EU) 2024/2847 on horizontal cybersecurity requirements for products with digital elements (Cyber Resilience Act),” *Off. J. Eur. Union*, Nov. 2024.

[15] The Kubernetes Authors, “Security,” Kubernetes Documentation, 2026.

[16] Codecov, “Bash uploader security update,” Apr. 2021.

[17] Cybersecurity and Infrastructure Security Agency (CISA), “Supply chain compromise of third-party GitHub Action, CVE-2025-30066,” alert, Mar. 2025.

[18] National Institute of Standards and Technology, “CVE-2021-44228 detail,” National Vulnerability Database, Dec. 2021.

[19] A. Freund, “Backdoor in upstream xz/liblzma leading to SSH server compromise,” oss-security mailing list, Mar. 29, 2024.

[20] OWASP Foundation, “OWASP Top 10:2025,” 2025.

[21] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, “in-toto: Providing farm-to-table guarantees for bits and bytes,” in *Proc. 28th USENIX Security Symp.*, Santa Clara, CA, USA, 2019.

[22] Z. Newman, J. S. Meyers, and S. Torres-Arias, “Sigstore: Software signing for everybody,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Los Angeles, CA, USA, 2022.

[23] GitHub, “Our plan for a more secure npm supply chain,” The GitHub Blog, Sep. 2025.

[24] ISO/IEC 5962:2021, “Information technology — SPDX specification V2.2.1,” International Organization for Standardization, 2021.

[25] Ecma International, “CycloneDX bill of materials specification,” ECMA-424, 1st ed., Jun. 2024.

[26] Cybersecurity and Infrastructure Security Agency (CISA), “2025 minimum elements for a software bill of materials (SBOM),” draft for public comment, Aug. 2025.

[27] M. Souppaya, K. Scarfone, and D. Dodson, “Secure software development framework (SSDF) version 1.1,” NIST SP 800-218, Feb. 2022.

[28] Snyk Security Labs, “Leaky Vessels: Docker and runc container breakout vulnerabilities,” Jan. 2024.

[29] Open Container Initiative, “runc security advisories for CVE-2025-31133, CVE-2025-52565, and CVE-2025-52881,” Nov. 2025.

[30] Kubernetes Project, “Kubernetes v1.36: User namespaces in Kubernetes are finally GA,” Kubernetes Blog, Apr. 2026.

[31] E. G. Young, P. Zhu, T. Caraza-Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “The true cost of containing: A gVisor case study,” in *Proc. USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, Renton, WA, USA, 2019.

[32] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman, “Exploit prediction scoring system (EPSS),” *Digit. Threats: Res. Pract.*, vol. 2, no. 3, 2021.

[33] Cybersecurity and Infrastructure Security Agency (CISA), “Known exploited vulnerabilities catalog,” 2026. [Online]. Available: https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[34] Center for Internet Security, “CIS Kubernetes benchmark,” 2026.

[35] The Kubernetes Authors, “Pod security standards,” Kubernetes Documentation, 2026.

[36] The Kubernetes Authors, “Validating admission policy,” Kubernetes Documentation, 2026.

[37] Cloud Native Computing Foundation, “Cloud Native Computing Foundation announces Kyverno’s graduation,” press release, Mar. 24, 2026.

[38] D. Feldman *et al.*, *Solving the Bottom Turtle: A SPIFFE Way to Establishing Trust in Your Infrastructure via Universal Identity*. SPIFFE Project, Cloud Native Computing Foundation, 2020.

[39] Amazon Web Services, “Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the EC2 Instance Metadata Service,” AWS Security Blog, Nov. 2019.

[40] Cloud Native Computing Foundation, “Cloud Native Computing Foundation announces Falco’s graduation,” press release, Feb. 2024.

[41] The MITRE Corporation, “MITRE ATT&CK: Containers matrix,” 2026. [Online]. Available: https://attack.mitre.org/matrices/enterprise/containers/

[42] Y. Weizman, “Threat matrix for Kubernetes,” Microsoft Security Blog, Apr. 2020.

[43] Gartner Research, “Innovation insight for cloud-native application protection platforms,” Aug. 2021.