COMPANION FIELD GUIDE

Architecture Review Field Guide

The questions you carry into the room — distilled from all 24 chapters of Secure Software Delivery Architecture.

98 review questions, grouped by trust boundary — walk the chain of custody, arrow by arrow.

Anchored by the **master trust-boundary table** and the **production-compromise attack tree**.

Closes with the **ten enduring principles**.

> **How to use it —** a review is collaborative trust-archaeology, not an audit. At each boundary, ask *"why should X trust Y?"* and *"what happens if X is compromised?"* Deliver findings as *misplaced trust*, not *missing tools* — the goal is to transmit the habit, not just fix one platform.

The Two Anchors

Keep these in view for the whole review. Every question below maps back to one of them.

1 · Master trust-boundary table

At every arrow: what crosses, what evidence must accompany it, and who independently verifies.

Boundary	What crosses it	Evidence required	Who verifies
Laptop → Git	Commits	Signed commits, PR review, status checks	Git platform (branch protection)
Git → CI	Source at a SHA	Pinned SHA, protected branch	CI checkout config
CI → Registry	Artifact	Digest, signature, provenance, SBOM	Registry policy + later consumers
Registry → Cluster	Image	Signature & provenance verification	Admission controller
Cluster → Workload	Running pod	Workload identity (SVID / IRSA)	Peer services, cloud IAM

2 · Attack tree — “run malicious code in production”

Each leaf names the control that blocks it. A leaf with no live control is a finding.

```
GOAL: run malicious code in production
├─ compromise the source (get bad code into Git)
│   ├─ steal developer credential ──► push        [branch protection, signed commits]
│   ├─ malicious PR passes review ──► merge       [CODEOWNERS, review quality]
│   └─ poison a dependency        ──► pulled      [pinning, SBOM]
├─ compromise the build (inject during build)
│   ├─ malicious build plugin / action           [pinning, hermetic build]
│   └─ compromise runner ──► persist             [ephemeral runners]
├─ compromise the artifact (substitute the image)
│   ├─ steal registry credential ──► push        [signature verification @ admission]
│   └─ move a mutable tag        ──► deploy      [digest pinning]
├─ compromise deployment
│   └─ modify manifests / abuse deploy creds     [GitOps review, admission]
└─ compromise runtime
    └─ exec / exploit app                       [RBAC, immutability, runtime detection]
```

The Review Checklist

98 questions across every boundary. Tick what holds; the blanks are your findings list.

PART ONE · FOUNDATIONS

02 The Trust Model

- [ ] For every arrow in your delivery diagram: what evidence crosses it, and who verifies that evidence?
- [ ] Which components, if compromised, can push code to production *without any other system noticing*? (Those are your single points of trust.)
- [ ] If your CI system lied about what it built, what downstream would catch the lie?

PART TWO · SOURCE TRUST

03 Git as the Root of Trust

- [ ] Show me the exact set of humans who can get a change into `main` with zero other-person involvement. (The honest answer is often "all admins plus anyone who can edit CODEOWNERS.")
- [ ] Who reviews changes to the CI workflows? To CODEOWNERS itself?
- [ ] If I steal one developer laptop tonight, what's in production tomorrow?

04 Repository Governance

- [ ] For each of the four domains, name the owning team. If any answer is "well, sort of...", that's a finding.
- [ ] How is a new repository created, and what protections does it get by default?
- [ ] When did you last review third-party app installations and their scopes?
- [ ] Can you produce, from code, the intended protection state of every repo — and detect drift from it?

PART THREE · BUILD TRUST

05 CI as a Trust Factory

- [ ] If build #4512 is malicious, what can it do to build #4513? (Correct answer: nothing.)
- [ ] What can a build reach on the network? Produce the egress allowlist.
- [ ] Show me every credential visible to a standard build job. Which of them are long-lived?
- [ ] Could a fork PR ever execute with secrets in scope?
- [ ] If your CI admin console credentials leaked tonight, what's the blast radius?

06 Build Identity

- [ ] List every static credential in your CI secret store. For each: why can't it be an OIDC exchange?
- [ ] Show me the trust policy for the production deploy role. What exact claims does it require?
- [ ] Can a feature-branch build obtain any credential that touches production?
- [ ] When a credential is used, can you tell *which build* used it from the audit log?

PART FOUR · ARTIFACT TRUST

07 The Artifact Lifecycle

- [ ] Point at a running production pod: can you trace its exact digest back to one build of one commit? How long does that take you?
- [ ] Who — humans and machines — can write to the registry path production pulls from?
- [ ] Is anything in production referenced by mutable tag?
- [ ] Show me the rollback procedure. Does it rebuild anything?

08 Provenance

- [ ] Can your build steps forge their own provenance? (If provenance is emitted by your YAML: yes.)
- [ ] Write out the exact verification policy: which signer identities, which source repos, which refs are acceptable — and where is it enforced?
- [ ] If an image appeared in your registry with no provenance, would anything downstream reject it?
- [ ] During an incident, how fast can you answer: "list every artifact built by builder B between T1 and T2"?

09 SBOM

- [ ] Time yourself: "which production workloads contain component X?" — minutes, hours, or archaeology?
- [ ] Are SBOMs produced from the build graph or inferred from filesystems? Who signs them?
- [ ] Does your inventory cover platform components, or only product services?
- [ ] When a CVE feed updates, does anything automatically re-evaluate existing SBOMs (continuous matching), or do you only scan at build time?

10 Digital Signatures

- [ ] Where do signing keys live, who/what can invoke signing, and what does the audit trail of signing operations look like?
- [ ] Recite your verification policy: which identities, which issuer, enforced where?
- [ ] If your signing identity were abused tonight, how would you find out — and how would you distrust the bad signatures without distrusting the good ones? (Rekor timestamps + certificate windows are the answer keyless gives you.)
- [ ] What, concretely, does a valid signature *entitle* an artifact to in your platform?

11 Attestations

- [ ] List the facts your deploy gate currently takes on faith from CI. Which are attestation-worthy?
- [ ] For each attestation type: whose identity signs, and could the build steps forge it?
- [ ] Does any policy anywhere *consume* your attestations? Show the predicate.
- [ ] What are your freshness rules, and what re-attests already-deployed digests?
- [ ] Can an artifact reach production through any path that skips the stamping stations?

PART FIVE · DEPLOYMENT TRUST

12 GitOps

- [ ] Can anything reach production without a commit to the deploy repo? List every path, including break-glass.
- [ ] Who can merge to the deploy repo's main? Is that list shorter or longer than "who could kubectl-apply before"?
- [ ] What is the reconciler's exact RBAC? Who reviews changes to *its* configuration?
- [ ] When drift is detected and reverted, who is paged?

13 Deployment Authorization

- [ ] State your separation-of-duties invariant in one sentence. Now name every identity that violates it (include admins and service accounts).
- [ ] Show the break-glass procedure. What does it skip? What does it *never* skip? When was it last used, and where's that review?
- [ ] What fraction of prod deploys involve a human decision, and is that fraction spent on the changes that actually carry risk?
- [ ] Can the deploy-approving group modify the policies that gate deploys?

14 Admission Controllers

- [ ] Attempt to run an unsigned image in every namespace, via kubectl, via the reconciler, via a Job created by an operator. Show me all three refusals.
- [ ] What happens to the cluster when the admission webhook is down? Who decided that trade, and is it written down?
- [ ] Who can modify webhook configurations and policies? What alerts on it?
- [ ] Which namespaces/identities are exempt, and why, and where is that reviewed?

PART SIX · RUNTIME TRUST

15 Workload Identity

- [ ] Pick a production pod: enumerate every credential visible inside it (env, mounts, metadata endpoints). How many are long-lived? Why does each exist?
- [ ] Show the IAM trust policy for your most powerful role reachable from the cluster. What exact subject does it require?
- [ ] If this pod is compromised at 2am, what can the attacker reach, for how long, and what logs carry the workload's identity?
- [ ] Can a pod in namespace A obtain an identity intended for namespace B? What, mechanically, prevents it?

16 Runtime Authorization

- [ ] From a shell in your least-trusted internet-facing pod: what can it reach (run the scan in staging), and what may it call as its identity?
- [ ] Which namespaces have default-deny today? For those that don't — dependency graph unknown, or priorities?
- [ ] Show the AuthorizationPolicy protecting your crown-jewel service. Who may call which operations?
- [ ] Who can change network/authz policies, and does that path have the same rigor as your deploy path?
- [ ] Does end-user context survive the trip from ingress to backend, or does "internal" mean "on behalf of anyone"?

17 Runtime Drift

- [ ] Can I write to the filesystem of your production payment container? Spawn a shell in it? Which policy says no, and where's it enforced?
- [ ] Who exec'd into production last week? Produce the list in under five minutes. Who was paged when it happened?
- [ ] A pod starts making DNS queries it has never made: what fires, who's paged, and what's the containment playbook?
- [ ] After a suspicious event, what's your remediation verb — clean, or replace?

PART SEVEN · PLATFORM SECURITY

18 Secrets Architecture

- [ ] Pick your most sensitive secret. Trace it: where created, every place it exists right now, how delivered to workloads, its TTL, how rotated, how revoked. Count the copies.
- [ ] How many secrets in your platform are long-lived where an identity-based (OIDC/workload) replacement exists but wasn't adopted?
- [ ] If your most powerful production credential leaked at noon, what's your revocation path and how long until it's dead *everywhere*? When did you last test that?
- [ ] What unlocks your secrets manager — an identity, or another secret? (If a secret: where does *that* one live?)
- [ ] Can you enumerate every secret and its age? What's the oldest, and why is it still alive?

19 Policy as Code

- [ ] Where do your admission and network policies live? Are they in Git, reviewed, tested, and GitOps-deployed — or edited in-cluster?
- [ ] Show me a policy's unit tests. Do they assert what's *allowed*, not just what's denied?
- [ ] Who can merge to the policy repo? Is that set separate from the teams whose deployments those policies gate?
- [ ] How does a new policy roll out — straight to Enforce, or through audit mode with measurement?
- [ ] Show me your current policy exemptions. Which have expiry dates? Which have owners? Which are older than their justification?
- [ ] What alerts when someone changes a ClusterPolicy or webhook configuration in the cluster?

20 Threat Modeling

- [ ] Can you produce a current data-flow diagram of your platform with trust boundaries marked? (If not, you can't threat model it — start there.)
- [ ] For your top asset ("run code in prod"), walk me an attack tree. Which leaves have a live, enforced control? Which are bare?
- [ ] Which threat actor does each of your major controls actually defend against? Are you defending against compromised-insider, or only external?
- [ ] What's your top residual risk, who accepted it, and when was that decision last revisited?
- [ ] When did you last threat model — and was it before or after you built the thing?

21 Platform Architecture: Putting It All Together

- [ ] Draw your platform's full chain. At every arrow: identity, evidence, verifier. Any arrow missing one of the three is a finding.
- [ ] Trace one real artifact commit-to-pod. Does evidence produced upstream actually get *verified* downstream, or just produced?
- [ ] Name every control plane (policy repo, OIDC issuer, signing root, vault auth, reconciler config, CODEOWNERS). Is each governed more strictly than what it controls?
- [ ] Where's your golden path? What fraction of services are on it vs. bespoke?
- [ ] Does the architecture survive a second cluster / second cloud without re-founding it?
- [ ] Point at your softest link — the one place where a single compromise runs code in prod with nothing downstream noticing. (There's always one. Knowing it is the job.)

PART EIGHT · OPERATIONS

22 Incident Response

- [ ] Tabletop right now: "we suspect CI was compromised Tuesday–Thursday." Walk me through it. How fast can you list in-window artifacts? (That's a provenance query — do you have it?)
- [ ] If your primary signing identity were compromised, how would you enumerate what it signed and when? (Rekor — is it monitored?)
- [ ] "Are we affected by CVE-X / dependency-Y?" — minutes or weeks? Where does the answer come from?
- [ ] When was your last IR tabletop for a *supply-chain* (not host) incident?
- [ ] Does your IR break-glass preserve integrity verification? Could a faked incident be used to bypass controls?
- [ ] For each pipeline component: what can it touch? Is that a written list (for rotation scope) or a discovery exercise?

23 Maturity Models

- [ ] What's your current SLSA level, per workload tier, and what's your *target*? Is the gap a plan or an accident?
- [ ] For each capability you've built (SBOM, provenance, attestations): is it *consumed* by something (a query, a gate), or just produced?
- [ ] If you named one thing to improve next quarter, would it be the highest-leverage rung, or the easiest/most-visible one?
- [ ] Which frameworks do you reference, and do you use them as roadmaps or as compliance-theater checkboxes?
- [ ] When did you last *re-assess* maturity, and did you find drift?

The Ten Enduring Principles

Tools change; these don't. Score the platform against them, not against a product list.

1. **Chain of custody**
   Identity + evidence + independent verification at every handoff.
2. **No transitive trust**
   Verify at the boundary where damage happens; don't trust that the last hop trusted the one before it.
3. **Blast radius, not invulnerability**
   Assume breach; make each one small, loud, and dead-ended.
4. **Verification lives in the next domain**
   The answer to "what if X is compromised?" is never inside X.
5. **Eliminate secrets with identity**
   The best secret is no secret; short-lived, scoped, attested beats stored.
6. **Evidence must be consumed**
   Provenance nobody verifies and SBOMs nobody queries are theater.
7. **Guard the control plane**
   Attackers edit the policy, not the artifact — protect the policy repo, signing root, and issuer strictest of all.
8. **Match rigor to risk**
   Uniform rigor gets routed around; uniform laxity gets breached. Tier workloads; build golden paths.
9. **The soft center**
   Every platform has one place a single compromise wins silently. Find yours — that's the job.
10. **The habit is the deliverable**
    "Why should X trust Y?" — asked relentlessly, of your own systems most of all.

> **The one question that finds the soft center:** "Where is the single place a compromise runs our code in production with nothing downstream noticing?" There is always one. A team that cannot name theirs has not done the work.