

Using Kiro to Implement the AWS Well-Architected Agentic AI Lens

Best-practice documents tell you what good looks like — they don't enforce it. This leave-behind shows how to encode the Agentic AI Lens into Kiro's control surface so compliance becomes a **side effect of normal development**, not a separate review step.

THE PROBLEM

Why best practices don't stick

Runtime guardrails alone Bedrock Guardrails, IAM, CloudWatch catch problems after the agent is built — detecting the bad pattern, not preventing it.	Author-time controls (Kiro) Steering, specs, hooks and permissions shape code as it's written , so the bad pattern is never authored. The layer the Lens assumes but doesn't tool.
---	---

Key insight: agentic systems need controls at **authoring time**, not just at runtime. Kiro is where you install them.

KIRO'S CONTROL SURFACE

Four mechanisms that enforce the Lens

1 • Steering files	.kiro/steering/*.md — persistent standards. Four inclusion modes: always, fileMatch, manual, auto (description-matched, ideal for guardrails). Custom agents need the path in resources.
2 • Spec-driven dev	requirements.md (EARS) → design.md (architecture, model rationale, cost, rollback) → tasks.md. A review checkpoint at each gate, before code exists.
3 • Agent hooks	.kiro/hooks/*.json — trigger + matcher + action. PreToolUse can BLOCK; PostToolUse sees results; plus SessionStart, Stop, file-change (CLI V3).
4 • Permissions + MCP	Capability rules with three effects: deny > ask > allow (deny-overrides). 'ask' is the native human-in-the-loop gate. MCP gives scoped tool access.

MAKE IT AUDITABLE

Pillar → Lens best practice → Kiro control → runtime

Leaf best-practice IDs from the Lens's Appendix A (150 practices). Priority = the Lens's own risk rating.

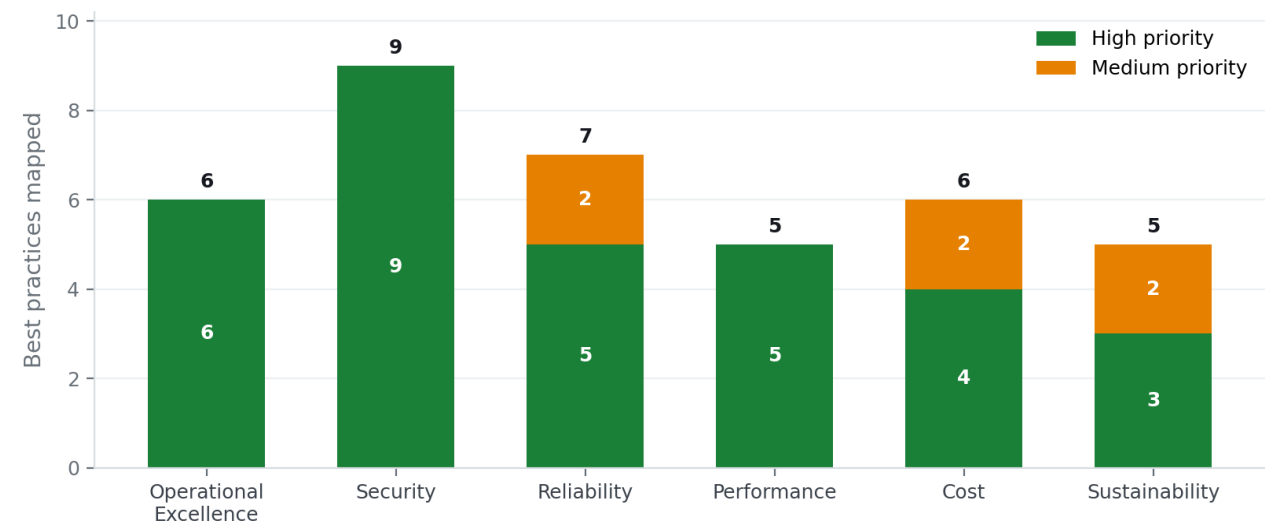
Pillar	Best practices	Kiro control	Runtime	Priority
Operational Excellence	OPS01-BP01/02; OPS05-BP01/03; OPS06-BP01/02	Specs; structured-log + X-Ray steering; PostToolUse observability & eval hooks	AgentCore Observability & Evaluations	All High
Security	SEC02-BP01; SEC03-BP02/03; SEC04-BP02; SEC08-BP01/02; SEC01	Permissions (ask on mutations); scoped MCP; PreToolUse credential scan (blocks)	AgentCore Identity & Policy; Bedrock Guardrails	All High
Reliability	REL02-BP01/02/04/05; REL06-BP02/04; REL07-BP02	EARS criteria: circuit breaker, fallback model, retries, idempotency; test hooks	AgentCore Runtime; model fallback	High · 2 Med
Performance	PERF02-BP02/04; PERF01-BP03; PERF03-BP02; PERF05-BP02	Model-selection & latency steering; design.md budgets; PostToolUse perf hook	AgentCore Runtime & Memory	All High

Pillar	Best practices	Kiro control	Runtime	Priority
Cost Optimization	COST02-BP01/02/03 ; COST03-BP02 ; COST05-BP01/02	Token budgets + max_tokens; design.md cost section; agent_id tagging	AgentCore Observability; cost tags	High · 2 Med
Sustainability	SUS01-BP05 ; SUS01-BP01/02 ; SUS03-BP03/04	The spec workflow (modular boundaries); reuse steering	Shared libs / base class; portfolio gov.	High · 2 Med

A quiet endorsement: AGENTSUS01-BP05's title literally recommends *"specification-driven tasks for frontier agents and long-running workflows"* — the Lens names a best practice for exactly this approach. And nothing mapped here is rated Low: every control is High or Medium.

AT A GLANCE

Best practices encoded, by pillar



Counts reflect the leaf best practices referenced in the matrix, rated per Lens Appendix A — not the full 150-practice set.

THE FULL PICTURE

Kiro is one layer of a closed loop

Author-time · Kiro Steering, specs, hooks, property tests. Prevents bad patterns being written.	Build-time · CI/CD Hook scripts run as merge gates. Headless: every 'ask' becomes 'deny'.	Inference · Guardrails Content filters, denied topics, grounding. Catches runtime violations.	Operations Incident investigation feeds lessons back into steering & specs.
---	---	---	---

■ Build in Kiro → enforce at runtime → detect in ops → feed fixes back into Kiro.

Each Kiro standard has a runtime home in AgentCore

Observability steering (logs, X-Ray, audit)	→ AgentCore Observability
Identity propagation; agent/human perms	→ AgentCore Identity
Least-privilege rules; tool authz; scoped MCP	→ AgentCore Policy + Gateway
Memory governance (classification, pruning)	→ AgentCore Memory
Evaluation gate from reliability/ops criteria	→ AgentCore Evaluations
Input validation, output filtering, injection	→ Bedrock Guardrails

GETTING STARTED

Build a Lens-compliant agent — and where to begin

1	Set up steering files — Drop Lens-aligned templates into .kiro/steering/ and trust the workspace so they load.
2	Create the spec — Kiro generates requirements/design/tasks shaped by the steering — security, reliability, cost all baked in.
3	Implement with hooks active — PreToolUse credential scanner blocks on secrets; PostToolUse flags missing structured logging.
4	Property tests + evaluation — Generate edge-case tests from reliability criteria; add an eval gate for stochastic behavior.

Shift controls left Guardrails detect; author-time controls prevent.	'ask' is the HITL gate Turn "needs approval" from prose into a permissions rule.	Start with one of each security-guardrails.md + the credential hook. Add a pillar per sprint.
--	--	---

References: AWS Well-Architected Agentic AI Lens & Appendix A · Kiro docs (steering, specs, hooks, permissions, MCP) · Amazon Bedrock Guardrails.

Examples target Kiro CLI 3.0 / IDE 1.0; syntax evolves — check current Kiro docs. AgentCore mapping is qualitative; confirm service quotas against current AgentCore documentation.