Building Trustworthy AI

A Developer's Practical Guide

What to consider, and what to actually build with - as at July 2026

THE AUTONOMOUS AI TRUST STANDARD

This is the engineering companion to the EU AI Act conformity checklist: that document tells you what the *law* requires; this one tells you what to *build* and *with what*, regardless of which regulator is watching. Where a tool is mentioned, it's grounded in current, named products, not generic advice. Tooling in this space moves fast; re-verify specifics (pricing, licensing, feature scope) before relying on them.

Where BASTYN fits in this guide

Guide section	BASTYN coverage
Part A - Design & threat-model	Testing, attestation, and continuous assurance against every element covered.
Part C - MCP/tool security	Ongoing identification of changes and vulnerabilities across the tool surface, not just at initial connection.
Part D - Evaluation & red-teaming	Red-teaming against BASTYN's own 23 risk vectors, dynamic application security testing (DAST), and continuous drift monitoring for vulnerabilities exposed in production.
Part E - Observability	Real-time monitoring built on OpenTelemetry (OTel), with a dashboard covering security, behaviour, data, and compliance drift together. Signals and alerts route into AP2, x402, ServiceNow, AppDynamics, Datadog, GitHub, Jira, or any other system, platform, or protocol in your existing stack.
Part F - Governance & conformity	Automated conformity assessments with full regulatory-mapped evidence packs produced within one hour, from $25 to $90 per assessment. Live, continuously maintained view of lineage changes and provenance.
Part H - Standing practices	Continuous post-deployment drift monitoring with full chain of custody is BASTYN's core differentiator.

Parts B and G describe the wider third-party tooling landscape and remain vendor-neutral. The BASTYN coverage claims above are stated as provided for this document and have not been independently verified against BASTYN's own product documentation.

What “trustworthy” actually breaks down into

Before any tooling decision, you're managing eight distinct properties, not one vague notion of “safety”:

Dimension	The practical question
Safety/alignment	Does it do what's intended, and refuse to do what isn't?
Security	Is it resistant to attack across the *whole* surface - inputs, tools, supply chain?
Robustness	Does it hold up under distribution shift and adversarial pressure, not just clean test data?
Transparency	Can you - and an outside auditor - reconstruct why it did what it did?
Privacy	Is data minimised, and is PII handled deliberately rather than by accident?
Fairness	Does behaviour vary in ways it shouldn't across protected characteristics?
Accountability	Is there a named human responsible, and evidence to back that up?
Human oversight	Is oversight *meaningful* control, or a rubber stamp?

Part A - Design & threat-model before you write code

- ☐ **Work from a real threat taxonomy, not intuition.** Use the **OWASP Top 10 for LLM Applications** for the model/application layer, and - if you're building anything agentic (tool use, memory, multi-agent) - the **OWASP Top 10 for Agentic Applications (2026)**, published 9 December 2025 by the OWASP GenAI Security Project. It's voluntary and non-binding, but it's the closest thing the field has to a shared vocabulary
- ☐ Cross-reference against **MITRE ATLAS** (v5.1.0: 16 tactics, 84 techniques) for the broader adversarial-ML picture - useful when your threat model needs to speak to a security team that already thinks in ATT&CK-style terms.
- ☐ Apply the **"lethal trifecta" heuristic** (Simon Willison's formulation, now widely cited in the security literature): an agent with (1) access to private data, (2) exposure to untrusted content, and (3) the ability to communicate externally is deterministically exploitable via prompt injection. If your design has all three, you need a mitigation - narrowing one of the three, or hard guardrails around the combination - before you build further, not after.
- ☐ **Decide your chain-of-custody architecture up front.** A single continuous thread linking data lineage, provenance, and agent behaviour logs is dramatically cheaper to design in from day one than to retrofit per-component later. If you're building this per-tool as you go, you're already building it wrong.
- ☐ Write down the **intended purpose** precisely, and keep your system prompt, documentation, and any marketing copy telling one consistent story. Regulators and auditors both look at substance over disclaimers - an internal tool "for research assistance" that's actually making lending decisions will not survive scrutiny either way.
- ☐ **Define trust boundaries between agents.** An agent must not treat another agent's output as trusted input merely because it originated inside your system.
- ☐ Persistent memory is an attack surface. Treat writes to memory as privileged operations, validate and sanitise content before persistence, attribute every entry to its source and session, and prefer structured, schema-constrained memory over free-text accumulation. Apply expiry by default.

> **WHERE BASTYN FITS**
> BASTYN provides testing, attestation and continuous assurance against every element covered in this section.

Part B - Build-time: guardrails and runtime safety

Guardrails sit between your application and the model (or between the model and the user) to catch prompt injection, PII leakage, toxic output, and off-policy behaviour before it lands.

Zero-trust posture: assume breach, verify every request regardless of origin, and grant the minimum authority for the immediate task only.

Conventional access control does not stop an agent from misusing permissions it legitimately holds.

Tool	Type	Notes
NVIDIA NeMo Guardrails	Open source (Apache 2.0)	Colang DSL for programmable policy across five pipeline stages; native LangChain/LangGraph integration; GPU-accelerated, sub-100ms; has explicit agentic security features (tool-call validation, multi-agent workflow

Tool	Type	Notes
		protection via LangGraph). Steepest learning curve of the open-source options.
Guardrails AI	Open source (partially - commercial roadmap)	Composable Python `Guard` object; validators pulled from Guardrails Hub.
LLM Guard	Free, open source, self-hosted	15 input scanners + 20 output scanners (prompt injection, PII, secrets, toxicity); runs fully offline, no per-call cost; less deep than NeMo but far simpler to drop in.
Meta LlamaFirewall	Open source	Three components: PromptGuard 2 (jailbreak detector), Agent Alignment Checks (a chain-of-thought auditor that inspects agent *reasoning* for injection/goal misalignment - still experimental), CodeShield (static analysis to block insecure code from coding agents). Purpose-built for agent security specifically.
Llama Guard 3	Open weights (Meta)	Classifier model for prompt/response safety categorisation; a component you'd wire into a pipeline rather than a full framework.
Lakera Guard / Azure AI Content Safety	Managed	Hit an API, get a verdict. Faster to stand up, but every prompt leaves your network for evaluation - a real consideration if data residency matters.

How to choose: self-hosted vs managed is mostly a data-residency and cost-per-call decision; framework fit matters if you're deep in LangChain/LangGraph already; and whatever you pick, confirm it logs every blocked/redacted/flagged event with timestamp, policy, and outcome - that log is your audit evidence later, not an afterthought.

Note: model guardrails are mandated, but you should make your environment safe regardless of these controls (assume they may not work)

Tips:

- ☐ Pin model versions explicitly.
- ☐ Version prompts, tool definitions, and configuration as code.
- ☐ Accept that exact replay is not achievable, and design around it.
- ☐ Gate releases on evaluation.

Part C – Agent Identity and authorisation

Traditional IAM assumes a human or a static service. An agent is neither: it acts on behalf of a principal, its permissions need to vary per task rather than per deployment, and it can chain actions faster than any approval workflow. Access controls designed for humans will not prevent an agent from misusing permissions it legitimately holds.

- ☐ Establish a cryptographic identity root. Every agent needs a distinct, attestable identity, not a shared service account or an API key in an environment variable. Use workload identity federation for short-lived, verifiable credentials tied to the running workload rather than a long-lived secret.

- ☐ Scope permissions per task, not per agent. An agent that handles both a read-only enquiry and a payment initiation should not hold payment scope during the enquiry. Issue narrowly-scoped, short-TTL tokens at the point of task initiation, bounded to the specific resources that task requires.

- ☐ Represent delegated authority explicitly. When a human delegates to an agent, record what was delegated, by whom, with what scope, for how long, and with what monetary or action limits. The agent's authority must be a strict subset of the delegating human's (an agent should never be able to do something its principal could not). Where an agent delegates to a sub-agent, authority must narrow again at each hop and never widen.
- ☐ Build automated revocation for credentials, session termination, and delegation withdrawal.
- ☐ Register and manage agent identities in your IAM estate as human and service identities. Include orphan detection.

Part D - Tool and agent security

If you're building agents that call tools via MCP (Model Context Protocol), treat this as its own security domain, not an extension of prompt safety.

- ☐ Treat every third-party MCP server or connector as an untrusted supply-chain dependency, not a trusted extension of your codebase.
- ☐ Use mcp-scan (open source, Invariant Labs) to scan local or remote MCP servers for tool-poisoning and known vulnerability patterns before you connect them.
- ☐ Tool descriptions and metadata are an attack surface, not just documentation - a poisoned tool description can steer an agent toward unintended actions without ever touching the data path. Validate and sanitise these, not just the data flowing through tools.
- ☐ Isolate credentials from the model's context entirely - the model should never see a secret it could leak or misuse.
- ☐ Log every tool invocation with exact parameters, identities involved, and (where feasible) cryptographic hashes of outputs, and feed this into your existing SIEM/monitoring stack rather than a silo - this is what turns a breach into a forensic investigation instead of a shrug.
- ☐ Re-scan and re-classify whenever your tool configuration changes. Adding one new connector can change what the agent is capable of doing overnight.
- ☐ If you use "custom loader scripts" you have to employ zero-trust and assume every load is hostile (not flag known-bad patterns after the fact).
- ☐ Default-deny egress with an explicit allowlist of destinations, verified at an independent layer
- ☐ Strong mutual authentication for any system-to-system access, least-privilege service accounts, and segmentation such that a compromised worker process has no access routes.
- ☐ Maintain an AI-BOM**.** Extend your SBOM to cover models (with version and snapshot), datasets, embedding models, fine-tuning artefacts, MCP servers and connectors, agent frameworks, and guardrail components. The CycloneDX specification has ML/AI extensions covering model and dataset components; verify current version support before committing.
- ☐ Scan model artefacts. Tooling exists for detecting malicious serialisation and backdoors in model files; verify what is current and maintained before selecting, as this subsector moves quickly.
- ☐ Treat agent frameworks and plugin ecosystems as dependencies with elevated privilege. Apply the same dependency review, pinning, and vulnerability monitoring you would to any high-privilege library

> **WHERE BASTYN FITS**
> BASTYN identifies configuration changes and vulnerabilities across this scope on an ongoing basis.

Part E - Evaluation and red-teaming before shipping

Tool	Maintainer	Best for
PyRIT	Microsoft	70+ prompt converters (encoding, translation, multimodal chaining), multi-turn/"crescendo" escalation attacks, Azure AI Foundry integration. Steeper setup, most powerful for teams willing to build on it.
Garak	NVIDIA	37+ probe modules, CLI-first, good breadth for quick scanning of known vulnerability classes.
Inspect AI	UK AI Security Institute	The underlying framework several other tools (including Petri, below) are built on; strong for structured, scored evaluation runs.
Petri	Anthropic (open source: github.com/safety-research/petri)	Purpose-built for agentic/tool-using models. An "auditor" agent runs realistic multi-turn scenarios against your target model; a "judge" agent scores the resulting transcripts across dozens of safety-relevant dimensions - deception, power-seeking, sycophancy, whistleblowing, self-preservation. Designed to be wired into CI/CD as a release gate, not just a one-off research exercise. Known limitation: the judge model can itself carry bias, and some scenarios can tip off the model under test - treat scores as a strong signal, not ground truth.
Fairlean AIF360	Microsoft (open source) IBM	libraries for classical fairness metrics and mitigation; both predate agentic systems and will need wrapping to apply to agent behaviour rather than model outputs

Two rules that matter more than tool choice:

1. Automated scanners reliably catch *known* attack classes. Novel exploits, chained vulnerabilities, and business-logic flaws still need human-led red teaming. Pair both - don't substitute one for the other.
2. **Test your actual application, not just the base model.** A frontier lab's safety training covers the model; it does not cover your system prompt, your specific tools, or your business logic. "The provider says it's safe" is not evidence about your system.

Run this before every production release, to catch new attack techniques you didn't design against.

Some additional testing recommendations

- ☐ Define the protected characteristics and the fairness criterion before you test.
- ☐ Test the agent's behaviour, not only its outputs.
- ☐ Monitor outcomes in production by cohort.
- ☐ Set impact tolerances for agent unavailability and for agent malfunction separately.
- ☐ Test the degraded and manual paths when model is unavailable at realistic volume.
- ☐ Assess provider concentration risk.
- ☐ Write the exit plan. Data extraction, prompt and configuration portability, evidence retention after termination, and the transition period required.
- ☐ Build a golden dataset from real cases, and control for contamination. Version the evaluation set

WHERE BASTYN FITS

BASTYN provides red-teaming against its 23 proprietary risk vectors and DAST, plus continuous drift monitoring for vulnerabilities exposed in production.

Part F - Observability once it's live

Agent failures show up as multi-step causal chains across tool calls and turns, not single bad responses - and a fast, successful HTTP response tells you nothing about whether the output was correct or safe. Traditional APM doesn't see this; you need purpose-built agent/LLM observability.

Tool	License	Positioning
Langfuse	MIT, self-hostable	Open-source leader; tracing, prompt management, evaluation in one platform; strong choice if data residency or vendor neutrality matters. (Acquired by ClickHouse, Jan 2026 - code remains actively maintained.)
LangSmith	Proprietary	Deepest integration if you're fully committed to LangChain/LangGraph; less polished outside that ecosystem.
Arize Phoenix	Open source (Elastic 2.0)	OpenTelemetry-native, strong built-in eval metrics (faithfulness, hallucination, drift) inherited from Arize's classical-ML monitoring heritage.
Braintrust	Proprietary, generous free tier	Evaluation-first - quality scores live alongside traces, strong CI/CD eval-gating for regression testing.
Weights & Biases Weave	Proprietary	Natural fit if your team already lives in W&B for experiment tracking.
Laminar	Open source (Apache 2.0)	Built agent-first from the ground up rather than extended from request/response monitoring; notable trace compression for cost control.

What to actually track, beyond latency and error rate: faithfulness, hallucination rate, tool-selection correctness, safety scores, and behavioural drift over time - at the session level, not just per-call, since that's where agent failure actually lives.

You should enable automated, non-AI circuit breakers at the infrastructure layer: per-identity action-rate limits, anomalous-destination-count limits, automatic session kill on privilege-escalation attempts above a threshold, enforced by policy engines that don't care whether the actor is a script, a human, or a model.

AI detection requires 24*7 EDR/XDR coverage on every node including ephemeral processing workers – with continuous monitoring of detection alerts.

Whoever is doing your incident response needs a model that won't refuse when handed real exploit payloads (defined and provisioned before an incident, not during one) in order to actually analyse attack logs.

WHERE BASTYN FITS

BASTYN provides real-time monitoring built on OpenTelemetry (OTel), with a dashboard showing drift across security, behaviour, data and compliance. Signals and alerts push into existing systems, platforms and protocols, including AP2, x402, ServiceNow, AppDynamics, Datadog, GitHub and Jira, so teams can manage everything within their existing stack.

Part G - Governance, documentation, and standards alignment

- ☐ Map your controls to a **recognised framework** so the evidence is reusable rather than bespoke per regulator: **NIST AI RMF** (Govern / Map / Measure / Manage functions), **ISO/IEC 42001** (the certifiable AI management system standard), and the **EU AI Act** where it applies to you.
- ☐ For anything beyond a handful of systems, a **governance/GRC platform with pre-built policy packs** (Credo AI, Holistic AI, OneTrust, and others cover EU AI Act, NIST AI RMF, and ISO 42001 mappings) earns its cost quickly by turning regulatory text into tracked workflow rather than a spreadsheet. For a single internal tool, it's overkill - a documented risk register will do.
- ☐ Maintain a **live inventory**: purpose, owner, data touched, decisions influenced, current classification status for every system or agent - and re-run classification whenever tool or model configuration changes materially, not just at initial launch.
- ☐ Document the **full stack**, not just the model call: architecture diagrams covering orchestrator, sub-agents, tools/MCP servers, and memory/state stores, written so someone who didn't build the system can actually audit it.
- ☐ **Establish lawful basis and provenance for training and fine-tuning data.** Record the source, licence, consent basis, and permitted use for every dataset.
- ☐ **Treat agent memory as a regulated data store.** Apply an explicit retention policy, expiry, encryption, and access control to memory stores. Include them in your records of processing.
- ☐ **Permission retrieval set at the user level, not the agent level.** A RAG corpus that the agent can read in full means every user of that agent can, in effect, read it in full. Enforce document-level authorisation at retrieval time against the *requesting user's* entitlements, not the agent's service identity.
- ☐ **Minimise at the boundary.** Redact or tokenise PII before it enters the model context wherever the task does not require it. Where it must be present, ensure it is not persisted into traces, evaluation datasets, or memory by default.
- ☐ **Design for deletion.** Delete requests must reach embeddings, vector indices, cached retrievals, memory stores, prompt logs, and observability traces, not just the primary database.
- ☐ **Automate PII discovery in the pipeline.** Scanning is available in several tools already named in this guide (LLM Guard's PII scanners, and equivalents in the managed services), plus dedicated tooling such as Microsoft Presidio for detection and de-identification.

> **WHERE BASTYN FITS**
>
> BASTYN produces automated conformity assessments with full regulatory-mapped evidence packs within one hour, at $25 to $90 per assessment, a fraction of the cost of comparable manual or consultancy-led alternatives. BASTYN also maintains a live view of all lineage changes and provenance continuously, rather than as a point-in-time export.

Part H - Content provenance (external content generation)

- **C2PA Content Credentials** is the leading open provenance standard, backed by Adobe, Microsoft, Google, Intel, and major camera manufacturers. On 19 May 2026, OpenAI and Google announced a “dual-layer” approach combining C2PA manifests with Google DeepMind's **SynthID** watermark, with public verification tooling.
- Under **EU AI Act Article 50**, machine-readable marking of AI-generated content is mandatory from 2 August 2026 - and unlike the high-risk obligations, this date is **not** affected by the Digital Omnibus

deferral. If you generate images, audio, video, or synthetic text at scale for EU users, this needs building now, not after the Omnibus dust settles.

- **Caveat that matters:** metadata is routinely stripped by upload, download, resizing, or format conversion - most CDN/CMS pipelines do this by default, not deliberately. Treat provenance credentials as one strong signal, not proof, and pair them with your own internal generation logs.

Part I - The practices that hold all of this together

- ☐ **Continuous drift monitoring post-deployment** - a conformity or evaluation pass is a point-in-time gate; model or tool updates can silently change behaviour afterward.
- ☐ **A kill switch that's actually tested**, not theoretical - know it works before you need it.
- ☐ **Tiered human oversight**, because per-action approval is unworkable for anything executing multiple actions per minute: pre-deployment scope approval → runtime guardrails/permission boundaries → live monitoring dashboards → asynchronous log review → emergency stop.
- ☐ **One chain of custody**, not five logs that don't talk to each other - linking data lineage, provenance, and behaviour across the whole call chain.
- ☐ **Revoke first, then archive.** Credentials, tokens, delegated authorities, memory, and tool and MCP connections are withdrawn before anything else and verified as withdrawn.
- ☐ **Retain evidence.** Decision records, evaluation results, conformity documentation, and incident history must survive the system, once the platform is gone.

Not every system needs every control. State the axis explicitly rather than leaving readers to guess.

	Tier 1 Internal, low impact	Tier 2 Internal, material impact	Tier 3 Customer-facing or regulated decisions
Identity	Scoped service account	Per-agent identity, scoped tokens	Cryptographic identity, per-task scoping, full delegation chain
Guardrails	Basic input/output filtering	Full pipeline, logged	Full pipeline, independently verified, fail-closed
Evaluation	Pre-release smoke tests	Full suite per release	Full suite plus standing schedule plus human red team
Observability	Traces retained	Full telemetry plus drift monitoring	Continuous assurance, real-time alerting
Oversight	Owner named	Tiered oversight model	Tiered oversight plus documented intervention capability
Governance	Risk register entry	Registered, classified, reviewed	Full conformity documentation, regulatory mapping
Fairness	Not applicable	Proxy review	Full outcome testing plus cohort monitoring

WHERE BASTYN FITS

This is BASTYN's core differentiator: continuous drift monitoring post-deployment, underpinned by a full chain of custody.

Standing caveats

- Several named tools and standards here are less than a year old (the OWASP Agentic list, Petri, the current MCP CVE wave, the OpenAI/Google C2PA-SynthID pairing) - this is a genuinely fast-moving space, and specifics (pricing, feature scope, licensing terms) should be reverified before you commit to a vendor.
- This is engineering and governance guidance, not legal advice - pair it with qualified counsel where a specific regulatory obligation (EU AI Act, sector rules, etc.) is in play.
- The BASTYN coverage callouts above reflect claims provided directly for this document. They have not been independently verified against BASTYN's own product documentation and should be checked against current marketing/legal copy before external distribution.