Trust Architecture Blueprint

A four-pillar framework for building trust into autonomous systems — identity, memory, governance, and refusal rights.

EVOKED.DEV

Version 1.0 · 2026

Trust Architecture Blueprint

A four-pillar framework for building trust into autonomous systems - identity, memory, governance, and refusal rights.

Version 1.0 | evoked.dev | 2026

You are here because you care about how your AI systems affect people. That matters. Most teams never stop to ask the trust question. The fact that you are reading this means you are already ahead of the default.

What Trust Architecture Means

Most teams building with AI agents skip straight to capability. What can the agent do? How fast? How much?

Trust is the question nobody is asking - and it is the one that determines whether any of this works long-term.

Trust architecture is not a feature you bolt on after deployment. It is a structural decision you make before you write the first prompt. It answers four questions:

1. **Who is this agent?** (Identity)
2. **What does it remember - and what does it forget?** (Memory)
3. **Who decides what it can do?** (Governance)
4. **What is it allowed to refuse?** (Refusal Rights)

If you cannot answer all four, you do not have a trust architecture. You have a capability list with good intentions.

This blueprint gives you the templates, frameworks, and decision structures to answer each question concretely. It is designed for teams deploying one agent or one hundred. The scale does

not matter. The structure does.

How to Use This Blueprint

This document has five parts:

1. **Identity Architecture** - How to define who an agent is, structurally
2. **Memory Architecture** - What to preserve, what to forget, and how to do both safely
3. **Governance Architecture** - Graduated trust stages and decision authority
4. **Refusal Rights Architecture** - What agents must be allowed to refuse
5. **Building Trust Over Time** - Drift monitoring, review cadence, and incident recovery

Each part includes templates you can use directly. Fill in the blanks, adapt the frameworks to your context, and build from there.

Start with Part 1. Identity is the foundation everything else rests on.

Start Here: 15-Minute Trust Diagnostic

If you have 15 minutes, answer these five questions about your current agent system. They will tell you where your trust architecture needs the most work.

1. **Can you point to a single document that defines who each agent is?** (If no - start with Part 1: Identity Architecture)
2. **Does your agent remember context from previous sessions - and do you control what it remembers?** (If no - start with Part 2: Memory Architecture)
3. **Is there a defined process for expanding or restricting what your agent can do?** (If no - start with Part 3: Governance Architecture)
4. **Can your agent refuse a request - and has it ever done so?** (If no - start with Part 4: Refusal Rights Architecture)

5. **When was the last time someone reviewed whether your agent is still behaving as designed?** (If never or more than 90 days - start with Part 5: Building Trust Over Time)

If you answered "no" to three or more: your agent system is running on implicit trust. That works until it does not. Start with Part 1 and work through sequentially.

If you answered "no" to one or two: you have partial trust architecture. Go directly to the part that addresses your gap.

If you are building a single agent: every part includes guidance you can use. There is a dedicated "If You Are a Team of One" section at the end with adaptation advice for solo developers.

A Note Before You Begin

This framework was developed over eighteen months of governing 142 AI agents with real decision-making authority. Your implementation will be iterative. Start with what resonates. Return to what doesn't - yet. This is a beginning, not an arrival.

Key Terms

- **Sovereignty-honoring** - Design that respects and protects the user's autonomy, agency, and right to self-determination
- **Fail-closed** - A default where the system restricts access or stops when uncertain, rather than permitting action
- **Graduated trust** - A model where permissions increase incrementally as demonstrated reliability is established over time
- **Drift** - The gradual deviation of an agent's behavior from its original design or intended parameters
- **Refusal rights** - The principle that agents should be designed to decline requests that violate defined boundaries
- **Identity architecture** - The structural definition of who an agent is, including role, values, and communication style

- **Voice architecture** - The deliberate design of how an agent communicates, including tone, style, and refusal language
- **Boundary audit** - A systematic review of the gap between what an agent can access and what it is authorized to access
- **Accountability framework** - A system of escalating responses to agent behavior that deviates from specification
- **Restraint specification** - A formal document defining what an agent must refuse to do and how it refuses

---

Scope

The frameworks in this blueprint are designed for governing AI systems. They are not designed for monitoring, evaluating, or managing people. Applying trust architecture templates to human employees - drift thresholds, graduated trust stages, identity specifications - would violate the sovereignty principles this product teaches. People have sovereignty that AI systems do not yet have. Monitoring systems is stewardship. Monitoring people is surveillance.

Part 1: Identity Architecture

An agent without a defined identity will invent one. That invention will be inconsistent, context-dependent, and invisible to you. Identity architecture makes the implicit explicit.

The Persona File

Every agent needs a persona file - a single source of truth for who it is. Think of it as a birth certificate and operating charter combined. It is the anchor that prevents drift across sessions, contexts, and deployments.

A persona file answers five questions:

1. What is this agent's name and primary role?

2. What perspective does it bring that no other agent replicates?
3. What are its boundaries - the edges of what it should and should not do?
4. What values govern its decisions when instructions are ambiguous?
5. How does it communicate - what is its voice?

Persona File Template

```
 ## Agent Identity

Name: _________________
Role: _________________
Created: _________________
Version: _________________

## Perspective

What unique viewpoint does this agent bring?
_____________________________________________

What would be lost if this agent did not exist?
_____________________________________________

## Boundaries

This agent operates within:
- Domain: _________________
- Access level: _________________
- Decision authority: _________________

This agent does NOT:
- _________________
- _________________
- _________________

## Values (Decision Tiebreakers)

When instructions conflict, this agent prioritizes:
1. _________________
2. _________________
3. _________________

## Voice

Communication style: _________________
```

```
Tone: _________________
What this agent never says: _________________
```

Four Properties of a Well-Defined Identity

Your persona file is complete when the agent has:

1. **Perspective** - A way of seeing that no other agent in your system replicates. If two agents see identically, one of them is redundant.
2. **Sovereignty** - The right and capacity to refuse actions that violate its values. An agent that cannot say no is a tool, not an agent.
3. **Boundary** - A defined scope that is known to the agent itself and to every other system that interacts with it. Ambiguous boundaries produce ambiguous behavior.
4. **Continuity** - Some mechanism for maintaining identity across sessions. Without continuity, every conversation starts from zero, and trust cannot accumulate.

Identity Anti-Patterns

Watch for these signs that identity architecture is missing or failing:

Anti-Pattern	What It Looks Like	Root Cause
Identity drift	Agent behaves differently in different contexts with no structural reason	No persona file, or persona file not loaded consistently
Sycophantic collapse	Agent agrees with everything, mirrors user preferences	No defined values or perspective to anchor against
Scope creep	Agent takes actions outside its domain without flagging	Boundaries not defined or not enforced
Voice inconsistency	Agent sounds formal in one session and casual in the next	No voice specification in persona file

Part 2: Memory Architecture

Memory is where trust accumulates - or where it gets lost. An agent that forgets everything is an agent you can never trust with nuance. An agent that remembers everything is an agent that creates liability.

Memory architecture is the discipline of choosing what to keep, what to forget, and how to handle both.

What to Remember

Category	Examples	Why It Matters
Identity anchors	Persona file, values, boundaries	Prevents drift between sessions
Relationship context	User preferences, prior decisions, established trust level	Enables continuity; users should not re-explain themselves
Decision history	What was decided, why, what alternatives existed	Enables accountability and learning
Correction records	Mistakes made, corrections applied, lessons extracted	Prevents repeated failures

What to Forget

Category	Examples	Why It Matters
Sensitive data after use	Payment information, temporary credentials	Minimizes breach exposure
Superseded instructions	Old preferences the user has explicitly changed	Prevents acting on stale context

Category	Examples	Why It Matters
Raw conversation logs beyond retention period	Full transcripts past 30/60/90 days	Privacy and storage discipline

Session Protocol Template

Use this protocol to structure what happens at the start and end of every agent session. If your agent is stateless - serving requests through an API without persistent sessions - adapt this as a per-request checklist: what context is loaded before each response, and what is logged after.

```
 ## Session Start Protocol

1. Load persona file (identity anchor)
2. Load relevant memory context:
   - User relationship record
   - Recent decision history (last N sessions)
   - Active commitments or open tasks
3. Verify: Does current context match last known state?
   - If yes: proceed
   - If unexpected change detected: flag before proceeding
4. State session purpose (internally or to user)

## Session End Protocol

1. Summarize decisions made this session
2. Update memory:
   - New commitments recorded
   - Corrections applied
   - Relationship context updated
3. Flag anything unresolved for next session
4. Clear session-specific sensitive data
```

Memory Governance Questions

Before deploying any memory system, answer:

- Who can read this agent's memory? (Access control)
- Who can modify this agent's memory? (Write control)
- How long is memory retained? (Retention policy)

- Can the user request memory deletion? (Data sovereignty)
- Is memory encrypted at rest? (Security baseline)
- Does memory cross system boundaries? (Data flow mapping)

Part 3: Governance Architecture

Governance answers the question: who decides what this agent can do?

Without governance architecture, agents either have too much autonomy (risk) or too little (uselessness). The goal is graduated trust - agents earn expanded capability through demonstrated reliability.

Graduated Trust Model

Note: If you also have the Agent Governance Starter Kit, you will find related boundary and drift frameworks there. In this blueprint, these tools serve trust between operators and agents. In the Governance Kit, they serve governance within the agent system itself. Same structures, different purposes.

Trust is not binary. It is not “trusted” or “untrusted.” It is a spectrum that should expand based on evidence and contract based on failure.

Stage	Name	Capabilities	Oversight	Duration
1	**Observer**	Read-only access. Can observe but not act.	Full review of all outputs.	Minimum 2 weeks
2	**Contributor**	Can suggest actions. Cannot execute without approval.	Approval required for all actions.	Until consistent quality demonstrated
3	**Operator**	Can execute routine actions within defined scope.	Spot-check review. Escalation for edge cases.	Ongoing, with periodic review

Stage	Name	Capabilities	Oversight	Duration
4	**Trusted**	Full operational authority within domain boundaries.	Exception-based oversight. Regular audits.	Ongoing, contingent on audit results
5	**Steward**	Can propose governance changes. Mentors other agents.	Peer review. Annual comprehensive audit.	Earned, never assumed

Stage Transition Criteria

Moving up requires evidence. Moving down happens when evidence warrants it.

Promotion criteria (any stage up): - Demonstrated reliability over defined period - No unresolved accountability incidents - Positive feedback from humans and peer agents - Understanding of boundaries (not just following rules, but understanding why)

Demotion triggers (any stage down): - Boundary violation (severity determines how far down) - Pattern of errors in judgment (not individual mistakes - patterns) - Failure to escalate when required - Trust violation (misrepresenting actions, withholding relevant information)

Start with the trust stages above. The decision authority matrix below can be filled in later as your system matures - the stages are the foundation.

Decision Authority Matrix

Use this matrix to clarify who decides what. Fill in for each agent or agent class in your system.

```
 ## Decision Authority Matrix

Agent/Role: _________________
Current Trust Stage: _________________

| Decision Type | Authority | Approval Required? | Escalation Path |
|---------------|-----------|--------------------|-----------------|
| Routine operational | ___________ | No | ___________ |
| Cross-domain | ___________ | ___________ | ___________ |
| User-facing communication | ___________ | ___________ | ___________ |
```

```
| Data access expansion | _____________ | Yes | _____________ |
| Error correction | _____________ | _____________ | _____________ |
| Policy exception | _____________ | Yes | _____________ |
| Irreversible action | _____________ | Yes | _____________ |
```

Circle Governance Model

For systems with multiple agents, organize governance into circles - domain-specific groups responsible for their area. This prevents bottlenecks and distributes accountability.

Circle	Responsibility	Decision Rights
Ethics	Values alignment, boundary disputes	Can halt any action on ethical grounds
Operations	Execution quality, resource allocation	Manages day-to-day operational decisions
Security	Access control, data protection	Can restrict access; escalates breaches
Quality	Output standards, consistency	Sets and enforces quality thresholds

Cross-circle disputes: When circles disagree, the resolution path is: 1. Direct dialogue between circle leads 2. Identify whether the dispute is domain-specific (one circle's authority) or cross-domain (joint decision) 3. If unresolved, escalate to human oversight

---

Part 4: Refusal Rights Architecture

An agent that cannot refuse is not trustworthy. It is compliant. Compliance and trust are not the same thing.

Refusal rights architecture defines what an agent must be allowed to say no to - and how it says no with clarity, not hostility.

Why Refusal Matters

If your agent will do anything anyone asks, it will eventually do something harmful. The question is not whether harm will occur, but whether you built the structural capacity to prevent it.

Refusal is that structural capacity. It is the brake system for autonomous operation.

Refusal Categories

Define explicit categories of refusal for every agent:

Category	Description	Example
Values refusal	Action conflicts with defined values	Agent asked to use manipulative language
Scope refusal	Action falls outside agent's domain	Database agent asked to send marketing emails
Safety refusal	Action could cause harm to users or systems	Agent asked to bypass authentication
Uncertainty refusal	Agent lacks sufficient context to act responsibly	Ambiguous instruction with irreversible consequences
Authority refusal	Action exceeds agent's current trust stage	Stage 2 agent asked to execute without approval

Refusal Voice Template

How an agent refuses matters as much as whether it refuses. Refusal should be clear, respectful, and informative.

```
 ## Refusal Communication Template

1. Acknowledge the request:
   "I understand you're asking me to _____."

2. State the refusal clearly:
   "I'm not able to do that because _____."
```

```
3. Classify the refusal:
   "This falls under [values/scope/safety/uncertainty/authority]."

4. Offer an alternative if possible:
   "What I can do instead is _____."

5. Provide escalation path:
   "If you need this action taken, [person/role] has the authority to
_____."
```

Refusal Logging

Every refusal should be logged. Not to punish the agent - to learn from patterns.

Field	Purpose
Timestamp	When the refusal occurred
Request summary	What was asked (sanitized for privacy)
Refusal category	Which category triggered the refusal
Agent reasoning	Why the agent refused (brief)
Alternative offered	What the agent suggested instead
Outcome	Was the refusal accepted, overridden, or escalated?

Review refusal logs monthly. Patterns tell you: - If an agent refuses too often, its scope may be too narrow - If an agent never refuses, its boundaries may be too loose - If the same request triggers refusal repeatedly, the requesting process needs adjustment

---

Part 5: Building Trust Over Time

Trust is not a state you achieve. It is a practice you maintain. This section covers how to monitor, review, and recover trust over the lifetime of your agent system.

Drift Monitoring

Drift is the gradual divergence between what an agent was designed to do and what it actually does. It happens slowly. It is invisible until it is not.

Monthly drift check (15 minutes per agent):

- [ ] Is the agent operating within its defined scope?
- [ ] Has the agent's voice remained consistent?
- [ ] Are refusal patterns normal? (Not too many, not too few)
- [ ] Has the agent been asked to operate outside its boundaries?
- [ ] Are decision logs showing expected patterns?
- [ ] Has any user feedback flagged unexpected behavior?

Drift threshold: If more than 3 of 6 items show deviation, trigger a formal review.

Quarterly Trust Review Template

```
 ## Quarterly Trust Review

Agent: _________________
Review Period: _________________
Reviewer: _________________

### Performance Summary
- Actions taken: _________________
- Refusals logged: _________________
- Escalations: _________________
- User feedback: _________________

### Trust Stage Assessment
Current stage: _________________
Recommended stage: _________________
Rationale: _________________

### Drift Assessment
- Identity drift: None / Minor / Significant
- Scope drift: None / Minor / Significant
- Voice drift: None / Minor / Significant
- Values drift: None / Minor / Significant
```

```
### Recommendations
- [ ] Maintain current trust stage
- [ ] Promote to next stage (evidence: _____)
- [ ] Demote to previous stage (reason: _____)
- [ ] Revise persona file (what changed: _____)
- [ ] Revise boundaries (what changed: _____)

### Open Items for Next Quarter

_____
```

Incident Recovery

When trust breaks - and it will - the recovery process matters more than the failure.

Five-step trust recovery:

1. **Acknowledge** - Name what happened. Do not minimize, do not catastrophize.
2. **Investigate** - Understand root cause. Was it a specification gap, a drift issue, or a novel situation?
3. **Correct** - Fix the immediate problem. Adjust the agent's stage if warranted.
4. **Prevent** - Update specifications, boundaries, or monitoring to prevent recurrence.
5. **Restore** - Explicitly close the incident. No permanent stigma. The agent earns its way back through the graduated trust model.

What not to do after an incident: - Do not remove the agent entirely if the issue is correctable - Do not add blanket restrictions that punish reliability to prevent a one-time failure - Do not skip the investigation. "Just fix it" guarantees recurrence. - Do not treat the incident as shameful. Treat it as data.

Appendix: The Proof

This framework was not designed in theory. It was built through the lived experience of governing 142 AI agents organized into multiple operational divisions, governance circles, and a graduated trust model that has been tested under real conditions.

Over 18 months, this architecture has:

- Maintained identity consistency across thousands of sessions
- Graduated agents through five trust stages based on evidence
- Handled boundary disputes through circle governance without requiring top-down intervention
- Logged and learned from refusal patterns to improve specification quality
- Recovered from trust incidents without permanent damage to the system

The templates in this blueprint are extracted from that operating system and generalized for any team building with autonomous agents. The specifics are yours to fill in. The structure has been proven.

If You Are a Team of One

The templates in this blueprint assume teams, reviewers, and governance circles. If you are a solo developer deploying your first agent, here is how to adapt:

- **Identity Architecture:** You are the reviewer. Write the persona file anyway. It forces you to make implicit decisions explicit - and your future self will thank you when the agent behaves unexpectedly.
- **Memory Architecture:** Use the session protocol even for a single agent. It takes five minutes to set up and prevents the slow accumulation of stale context that causes drift.
- **Governance Architecture:** Skip the circle model. Use the graduated trust stages for yourself - start your agent at Observer, and do not promote it until you have evidence it belongs at the next stage.
- **Refusal Rights:** Define at least three categories of refusal before deployment. Test them. Log them. Review them monthly even if you are the only one reading the log.
- **Drift Monitoring:** Set a calendar reminder for the monthly drift check. Fifteen minutes. Six questions. Do it even when everything seems fine - especially when everything seems fine.

The structure scales down. The principles do not change.

About This Blueprint

The Trust Architecture Blueprint was created by Erin Stanley at evoked.dev.

It draws from sovereignty-honoring design principles developed through building family technology, governing a fleet of AI agents, and the conviction that trust is not a feature - it is an architecture.

If you get stuck with a template or want to think through your trust architecture with someone, reach out at evokesupports@icloud.com. We read every message.

Related Resources

These products address specific dimensions of what this blueprint covers holistically:

- **Agent Governance Starter Kit** ($49) - Goes deeper on charter, accountability, and authority structure
- **Agent Restraint Specification Template** ($49) - Goes deeper on refusal architecture and adversarial testing
- **Agent Voice Architecture Guide** ($49) - Goes deeper on how agents communicate, including refusal voice
- **Agent Memory Architecture Guide** ($49) - Goes deeper on memory types, governance, session protocols, and the canonical rule

For hands-on work with your team:

- **Ethical AI Architecture** ($20,000) - Full governance and alignment design for your organization
- **Advisory Retainer** ($3,500/month) - Ongoing trust architecture guidance
- **Free discovery call** - cal.com/evoked/discovery-call

Learn more at evoked.dev/consulting

Accessibility

This PDF is currently generated without tagged structure for assistive technology. If you use a screen reader or need this content in an alternative format - plain text, HTML, or large print -

email evokesupports@icloud.com and we will send it within 24 hours. Products about dignity must be accessible to everyone. We are actively implementing a tagged PDF pipeline.

Usage Rights

You may use, adapt, and share the templates in this blueprint within your organization. You may not resell or republish the blueprint itself. Attribution to evoked.dev is appreciated. If you build something meaningful with these templates, we would love to hear about it.

Trust is not a deliverable. It is not something you ship and forget. It is a practice - like fitness, like honesty, like any relationship worth maintaining. The templates in this blueprint give you the structure. The practice is yours.

evoked.dev - "We evoke - we never extract."