Agent Safety Playbook

Zero-Trust Deployment Guide

For Engineering Leaders & Security Teams

LETTERHEAD

Why This Matters

AI agents are powerful—but dangerous without governance.

One permission misconfiguration can expose customer data, burn thousands in API costs, or cause compliance violations.

This playbook shows you how to deploy agents safely using five proven pillars of zero-trust governance.

Read time: 15–20 minutes.

Table of Contents

1 Introduction: The Agent Governance Gap

AI agents are powerful. A single agent can autonomously:

- Query your entire database
- Execute transactions and API calls
- Make decisions that affect customers and revenue
- Run 24/7 without human review

But most teams treat agent permissions like they treat developer permissions: binary. Either the agent can do everything the developer can do, or nothing at all.

This creates a critical governance gap.

When a developer writes a function, they:

1. Write code in a sandboxed environment
2. Pass code review
3. Deploy to staging
4. Test before production
5. Can debug immediately if something breaks

When an agent runs in production, it:

1. Executes decisions no one wrote explicitly
2. Makes calls that bypass code review

3. Runs with production permissions
4. May not be discoverable until it causes damage

The stakes are real:

- One permissioning bug = unauthorized access to PII
- One logic error = financial corrections or compliance violations
- One retry loop = cascading transaction failures
- One integrated agent = blast radius across your entire system

This playbook shows you how to close that gap using **zero-trust agent governance**: the principle that agents should have the *minimum* permissions needed, with *complete visibility* into what they do, *automatic cost controls*, and *rapid incident response*.

2

Why Zero-Trust Matters for AI Agents

The Traditional Model (Trust by Default)

In the traditional cloud model, we assume the inside of our network is trusted:

- Developer = trusted
- Service account = trusted
- API key = trusted

But agents operate differently. Unlike a developer, an agent:

- Has no human judgment
- Cannot explain its decisions
- Cannot ask for clarification
- Can execute at machine speed

When an agent has broad permissions, a small bug or misalignment becomes a big incident.

The Zero-Trust Model

Zero-trust reverses the assumption: **Trust nothing by default. Verify everything.**

For agents, zero-trust means:

1. **Minimal scope** — agents get only the permissions they strictly need
2. **Complete audit trail** — every action is logged and observable
3. **Automatic limits** — cost caps, rate limits, and hard stops
4. **Rapid response** — incidents are detected and isolated quickly
5. **Regular review** — permissions and access are re-validated frequently

Zero-trust is not about paranoia. It's about reducing blast radius and enabling rapid recovery.

The Business Case

Zero-trust governance for agents:

- **Reduces incident severity**: Small blast radius = small damage
- **Speeds compliance**: Auditors expect visibility and controls
- **Enables scaling**: More agents = more confidence your controls work
- **Unlocks adoption**: Teams feel safe deploying agents to mission-critical systems

---

3 The 5 Pillars of Agent Safety

Pillar 1: Authentication & Identity

The Problem

Today, agents often inherit developer credentials. A single shared API key or database password gives the agent the same permissions as the developer who created it.

If that key leaks or the agent is compromised, an attacker has full access.

The Solution: Scoped Identity

Instead of sharing credentials, create a unique, scoped identity for each agent:

- **Unique token** — The agent gets its own credential, not a shared key
- **Time-limited** — The token expires after a set period (hours, not months)
- **Scope-restricted** — The token only grants permissions the agent needs
- **Audit-linked** — Every action is traceable back to that specific agent

Implementation Examples

AWS IAM Approach:

```
1. Create an IAM role named "agent-financial-processor"
2. Attach a policy that grants:
   - s3:GetObject on bucket/invoices/ only
   - dynamodb:Query on table payments-table only
   - logs:PutLogEvents on agent-logs-* only
3. Use STS AssumeRole to issue time-limited credentials (15 min expiry)
4. Agent uses those credentials; they cannot be reused after expiry
5. CloudTrail logs every call with the agent's identity
```

Kubernetes Approach:

```
1. Create a ServiceAccount named "agent-financial-processor"
2. Bind it to a Role that grants specific API permissions
3. Kubernetes issues a scoped token mounted in the agent's pod
4. Token is automatically rotated and cannot access other namespaces
5. All API calls are logged to the audit log
```

PostgreSQL Approach:

```
1. Create a database role "agent_invoicing"
2. Grant permissions only on invoices and payments tables
3. Restrict operations to SELECT and INSERT only
4. Use pgAudit to log all queries
5. Store credentials in a secrets manager with automatic rotation
```

Expected Outcome: Each agent has provably minimal permissions, and every action is linked to a specific agent identity.

Pillar 2: Data Isolation & Access Control

The Problem

Even with scoped authentication, many agents can still access far more data than they need. A financial processing agent shouldn't see HR data. A reporting agent shouldn't have write access.

The Solution: Row-Level & Field-Level Security

Modern databases support fine-grained access control:

- **Row-level security (RLS)** — Agent sees only rows it's authorized to access
- **Field-level security** — Agent cannot see sensitive columns (PII, salary, SSN)

- **Projection views** — Only show the agent the columns and rows it needs

Implementation Examples

PostgreSQL RLS:

```
-- Create a policy: agent_invoicing can only see invoices for their assigned customers
ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

CREATE POLICY invoices_agent_access ON invoices
  FOR SELECT
  USING (customer_id = current_setting('agent.assigned_customer_id')::uuid);

-- When the agent connects, set the customer_id context
SET agent.assigned_customer_id = '12345';
SELECT * FROM invoices;  -- Only rows for customer 12345
```

BigQuery Authorized Views:

```
-- Create a view that agents can query
CREATE OR REPLACE VIEW agent_invoicing_view AS
SELECT invoice_id, amount, due_date  -- Exclude PII columns
FROM invoices
WHERE customer_id = @assigned_customer_id;  -- Row filtering

-- Grant agents access to the view, not the underlying table
GRANT roles/bigquery.dataViewer ON view agent_invoicing_view TO agent_service_account;
```

MongoDB Document-Level Access:

```
db.createCollection("invoices", {
  validator: {
    $expr: {
      $eq: ["$assigned_agent", "$$USER.agentId"]
    }
  }
});
```

Expected Outcome: Even if an agent is compromised, it can only access and modify the specific rows and fields it needs.

Pillar 3: Audit Logging & Observability

The Problem

If you can't see what an agent did, you can't debug failures or investigate breaches.

The Solution: Comprehensive Audit Logging

Log every agent action:

- **What** — Query, API call, transaction, decision
- **When** — Timestamp with millisecond precision
- **Who** — Agent identity
- **Where** — System/database/API endpoint
- **Result** — Success, failure, outcome
- **Context** — Input data, parameters, decision reasoning

Implementation:

1. **Database audit logs** (JSON format)

- PostgreSQL: pgAudit extension
- MySQL: Audit Plugin
- MongoDB: Change streams
- Capture: user, statement, timestamp, result

2. **Application logs** (structured JSON)

```
{
  "timestamp": "2026-06-16T14:23:45.123Z",
  "agent_id": "agent-invoicing-001",
  "action": "process_invoice",
  "invoice_id": "INV-2026-001234",
  "amount": 5000.00,
  "status": "success",
  "duration_ms": 234,
  "trace_id": "abc123def456"
}
```

3. **Real-time alerting**

- Anomaly detection (unusual query patterns, large data exports)
- Rate limit violations
- Permission errors
- Cost threshold breaches

4. **Log retention policy**

- Hot storage (7-30 days): Full queryable logs
- Warm storage (30-90 days): Sampled or compressed
- Cold storage (90+ days): Archive for compliance

Expected Outcome: You can rewind any incident to the exact sequence of agent actions and understand what happened.

Pillar 4: Cost Controls & Rate Limiting

The Problem

An agent with a bug or poor logic can silently burn thousands in API calls or compute costs before anyone notices.

The Solution: Automatic Budgets & Rate Limits

1. **Daily spend budgets** (hard stop)

- Example: Agent has $50/day budget for LLM calls
- Automatically throttled or paused when budget is reached
- No human intervention needed

2. **Query rate limits**

- Database: 1000 queries/minute max
- API: 100 calls/minute per endpoint
- Prevents cascading failures from retry loops

3. **Row count limits**

- Prevent bulk exports: "No query can return > 100K rows"
- Forces agents to filter and iterate

4. **Timeout limits**

- Long-running queries killed after 30 seconds
- Prevents resource exhaustion

Implementation Example (AWS):

```
1. Create a Lambda function for the agent
2. Set reserved concurrency to 5 (max parallel invocations)
3. Set timeout to 60 seconds
4. Attach an IAM policy that limits:
   - S3: 100 requests/minute
   - DynamoDB: read capacity units 10/sec
5. Use CloudWatch alarms to alert on cost anomalies
```

Expected Outcome: Even a runaway agent causes limited damage; you can pause it within your budget threshold.

Pillar 5: Incident Response & Rapid Recovery

The Problem

When an agent goes wrong, you need to:

1. Detect the problem quickly
2. Understand what happened
3. Stop the agent immediately
4. Roll back or repair any damage
5. Fix the root cause

The Solution: Runbooks & Automation

Incident Detection

- Monitor agent logs for error patterns
- Alert on permission errors, timeout errors, cost anomalies
- Set up dashboards showing agent health

Immediate Response (Minutes)

```
1. Automated kill switch: Agent is paused/disabled immediately
2. Alert sent to on-call engineer
3. Blast radius contained (limited permissions, cost caps prevent widespread damage)
4. Logs available for investigation
```

Root Cause Analysis (Hours)

```
1. Engineer reviews agent logs and audit trail
2. Identifies specific query or API call that failed
3. Checks configuration, permissions, and recent code changes
4. Determines: bug, misconfiguration, external API issue, or attack
```

Recovery & Prevention (Hours to Days)

```
For a bug:
1. Fix the agent logic
2. Re-test in staging with mocked data
3. Re-enable agent with enhanced monitoring
4. Gradually increase traffic back to normal

For a misconfiguration:
1. Review permissions, budgets, and rate limits
2. Adjust configuration
3. Re-test
4. Deploy

For an external issue:
1. Understand the third-party outage
2. Add circuit breaker or fallback logic
3. Re-test
4. Deploy with notification to stakeholders
```

Expected Outcome: Incident detection in minutes, containment in minutes, diagnosis in hours, fix in hours or days — not weeks.

---

4 Deployment Checklist

Before deploying a new agent to production, verify:

Authentication & Identity

- ■ Agent has a unique service account (not shared with humans or other agents)
- ■ Service account credentials are scoped (time-limited, permission-restricted)
- ■ Credentials are stored in a secrets manager (not in code)
- ■ Credentials are rotated automatically (e.g., every 30-90 days)
- ■ All actions are logged and traceable to the agent identity

Data Isolation

- ■ Agent can only access the specific tables/collections it needs
- ■ Agent can only access rows it's authorized to read (RLS or equivalent)
- ■ Agent cannot see sensitive columns (PII, internal data) — fields are masked or excluded
- ■ Agent has no write access to data it doesn't need to modify
- ■ Data access is verified in staging with realistic test data

Audit Logging

- ■ All agent queries/API calls are logged with: timestamp, agent ID, action, result, duration
- ■ Logs are stored in JSON format for easy parsing and alerting
- ■ Logs are retained for at least 90 days

- ■ Logs are not accessible to agents (read-only access for security teams)
- ■ Log alerting is configured for anomalies (unusual patterns, permission errors)

Cost Controls

- ■ Agent has a daily or monthly spend budget (with hard stop enforcement)
- ■ Agent has rate limits on API calls and database queries
- ■ Long-running queries or API calls have timeout limits
- ■ Cost monitoring and alerting is configured
- ■ Budget thresholds are tested and verified before production

Incident Response

- ■ A runbook exists for common agent failure modes
- ■ There is an automated kill switch to pause the agent immediately
- ■ On-call engineer is alerted when the agent fails
- ■ Logs and dashboards are available for quick diagnosis
- ■ A recovery procedure is documented and tested

5

Case Study: Series B Fintech Learns the Hard Way

The Situation

FinAI, a Series B financial services platform, deployed an autonomous invoice processing agent to reduce manual data entry. The agent reads invoices from email, extracts line items, and posts them to the accounting system.

The Deployment

The engineering team:

1. Created a Lambda function to run the agent
2. Attached an IAM role with broad permissions (S3 read, DynamoDB full access, SQS)
3. Deployed to production on a Friday afternoon
4. Monitored for a few hours, then left for the weekend

They did NOT:

- Implement row-level security on DynamoDB
- Set cost budgets or rate limits
- Configure audit logging
- Create an incident runbook

The Incident

On day 4, a subtle retry logic bug was triggered when the accounting system was temporarily slow. The agent:

1. Called the API to post an invoice
2. Received a timeout (system slow)
3. Retried the same invoice 3 times
4. Each retry posted the invoice again

With no idempotency check and no audit logging, the same invoice was posted 14 times over the weekend. The bug affected:

- 87 invoices (all from a large customer)
- $2.3M in transaction volume
- 14-day reconciliation delay (accountants discovered the duplicates)

The Compounding Failure

When the team investigated, they discovered a second problem:

- The agent's IAM role had write access to the summary transactions table
- A configuration error in the agent code was writing summary records with incorrect account codes
- Auditors flagged the journal entry errors as a control deficiency
- Regulatory notification was required

Total impact:

- **Financial**: $200K in corrections and accounting labor
- **Operational**: 1-week accounting close delay, customer escalations
- **Regulatory**: Auditor findings requiring board-level remediation
- **Reputational**: Customer notification about transaction posting errors

The Fix: Zero-Trust Implementation

The team implemented the 5 pillars:

1. Authentication

- Created agent-specific IAM role with 15-minute session tokens
- Revoked long-lived credentials

2. Data Isolation

- DynamoDB RLS: Agent can only write to transactions with its assigned customer ID
- Removed write access to summary tables
- Created a separate "audit" Lambda to post summaries after agent reconciliation

3. Audit Logging

- Enabled CloudTrail and DynamoDB streams
- All invoice posts logged with: timestamp, agent ID, invoice ID, amount, account code
- Real-time alerting if >5 duplicates detected

4. Cost Controls

- $100/day API call budget (hard stop)
- 1000 requests/minute rate limit
- 60-second timeout on all external API calls

5. Incident Response

- Created a Lambda that automatically disables the agent if error rate > 1%
- On-call engineer receives SMS alert immediately
- Runbook: Disable agent → Check logs → Revert to previous version → Verify

The Result

After implementation:

- No duplicate invoices in the next 6 months
- All failures caught within minutes (not days)
- Auditors removed the control deficiency
- Team gained confidence to deploy more agents

---

6 Letterhead Approach & Managed Services

Building zero-trust agent governance requires expertise across:

- Cloud identity and access management
- Database security and RLS
- Logging and observability
- Incident response automation
- Compliance and audit requirements

Most teams don't have all of these skills in-house.

Letterhead Agent Governance Services

Letterhead provides three service tiers:

Tier 1: Assessment & Design

- Architecture review of your agent deployments
- Identify gaps in authentication, data isolation, logging, and incident response
- Design zero-trust agent governance roadmap
- 4-week engagement, $8K–$12K

Tier 2: Implementation & Hardening

- Deploy scoped authentication (IAM, ServiceAccount, or database roles)
- Implement row-level security and field masking
- Configure audit logging and alerting
- Build incident response automation
- 8-12 week engagement, $25K–$40K

Tier 3: Managed Governance

- Ongoing monitoring of agent deployments
- Monthly security reviews
- Incident response on-call (24/5 or 24/7)
- Policy updates as new agents are deployed
- $5K–$8K/month

What's Included

In all tiers:

- Security policy templates for your org
- Audit logging and alerting setup
- Runbooks for incident response
- Training for your security and engineering teams
- Regular reports on agent security posture

In Tier 2 & 3:

- Cloud platform expertise (AWS, GCP, Azure)
- Database security (PostgreSQL, MongoDB, DynamoDB, etc.)
- Agent framework integration (LangChain, LlamaIndex, Anthropic SDK)
- Compliance alignment (SOC 2, FedRAMP, HIPAA)

Why Letterhead?

- **Focused expertise**: We work with agent deployments every day
- **Rapid implementation**: 2-4 week typical timelines
- **No lock-in**: All configs are portable; you own the infrastructure
- **Continuous improvement**: We stay current with agent frameworks and cloud platform updates

7

Getting Started: Your Safety Roadmap

Phase 1: Assessment (Week 1–2)

Your goal: Understand your current state and identify gaps.

Actions:

1. Audit existing agents (how many? what permissions do they have?)
2. Review current logging and alerting
3. Identify which systems have the highest risk (financial, customer data, etc.)
4. List your tech stack (cloud platform, databases, agent frameworks)

Deliverable: A 1-page summary of your current agent deployments and top 3 security gaps.

Phase 2: Pilot (Week 3–6)

Your goal: Implement zero-trust for one critical agent and prove the model.

Actions:

1. Pick your highest-risk agent (or the one you're about to deploy)
2. Implement scoped authentication (Pillar 1)
3. Add row-level security on the primary database (Pillar 2)

4. Configure audit logging (Pillar 3)
5. Set cost budgets and rate limits (Pillar 4)
6. Create an incident runbook (Pillar 5)
7. Test the entire deployment in staging with realistic data
8. Deploy to production with enhanced monitoring

Deliverable: One production agent with full zero-trust governance, a documented playbook, and a 1-week post-deployment health check.

Phase 3: Scale (Week 7+)

Your goal: Apply the same model to all agents.

Actions:

1. Use the Tier 1 playbook from your pilot
2. Create a standardized agent deployment process
3. Audit and upgrade remaining agents
4. Establish a monthly security review cycle
5. Update your incident response procedures

Deliverable: All agents operating under zero-trust governance, with ongoing monitoring and quarterly security reviews.

Common Starting Points

If you're just starting with agents:

- Do Phase 1–2 now, before your first agent hits production
- Use this playbook to guide your initial architecture
- Plan for 30–50% of your initial agent budget to go to governance

If you have agents in production but no governance:

- Audit them immediately (Pillar 3: logging is free)
- Start with Pillar 1 (scoped auth) on your riskiest agent
- Plan a 12-week rollout to cover all agents

If you have some governance in place:

- Assess which pillars you're strong in (auth, logging, cost controls?)
- Prioritize the pillars you're weakest in
- Use this playbook to fill gaps

Conclusion: Agent Safety is Table Stakes

AI agents are powerful tools for automating business processes. But power requires responsibility.

Zero-trust agent governance is not optional—it's the foundation for safe, auditable, and compliant agent deployments.

The five pillars—authentication, data isolation, audit logging, cost controls, and incident response—work together to:

- Reduce blast radius when things go wrong
- Enable rapid detection and recovery
- Build stakeholder confidence
- Meet regulatory and audit requirements
- Unlock safe scaling

The best time to implement zero-trust governance is before you deploy your first agent. The second-best time is now.

Start with Phase 1 (assessment) this week. Identify your riskiest agent. Implement one pillar. Measure the impact. Expand from there.

Zero-trust is not a checklist—it's a mindset. Once you adopt it, you'll deploy agents with confidence, knowing that your blast radius is controlled, your visibility is complete, and your recovery is rapid.

Appendix: Templates & Tools

IAM Policy Template (AWS)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DynamoDBLimited",
      "Effect": "Allow",
      "Action": [
        "dynamodb:Query",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:region:account:table/agent-table",
      "Condition": {
        "StringLike": {
          "dynamodb:LeadingKeys": ["${aws:username}"]
        }
      }
    }
  ]
}
```

PostgreSQL RLS Policy

```
ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

CREATE POLICY agent_isolation ON invoices
  FOR ALL
  USING (agent_id = current_user)
  WITH CHECK (agent_id = current_user);
```

Audit Log Query (CloudTrail)

```
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=ResourceName,AttributeValue=agent-table \
  --start-time 2026-06-16T00:00:00Z \
  --max-results 50
```

Questions or feedback? [Contact Letterhead](mailto:hello@letterhead.com)

Ready to implement zero-trust agent governance? [Schedule an audit](https://calendly.com/letterhead/audit)

A

Appendix: Templates & Tools

IAM Policy Template (AWS)

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "DynamoDBLimited",
    "Effect": "Allow",
    "Action": ["dynamodb:Query", "dynamodb:PutItem"],
    "Resource": "arn:aws:dynamodb:region:account:table/agent-table",
    "Condition": {
      "StringLike": {
        "dynamodb:LeadingKeys": ["${aws:username}"]
      }
    }
  }]
}
```

PostgreSQL Row-Level Security

```
ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

CREATE POLICY agent_isolation ON invoices
  FOR ALL
  USING (agent_id = current_user)
  WITH CHECK (agent_id = current_user);
```

Audit Log Query (AWS CloudTrail)

```
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=ResourceName,AttributeValue=agent-table \
  --start-time 2026-06-16T00:00:00Z \
  --max-results 50
```

Ready to implement zero-trust governance?

Contact us for a free 15-minute assessment.

hello@letterhead.com

letterhead.com/audit