AI Infrastructure Security Playbook

AWS Implementation Guide

Cloud-Specific Controls Mapped to ETSI EN 304 223 Risk Gradient L1–L8

V1.0 — February 2026

Risk-tiered · Capability-based · Exposure-driven

https://deepcyber.ai

Contents

1. Introduction

This guide provides AWS-specific implementation instructions for the AI Infrastructure Security Playbook. It maps each control from the playbook's seven dimensions (Governance, Identity, Data, Processing, Network, Supply Chain, Monitoring and Operations) to concrete AWS services, configurations, and recommended settings. All controls are cumulative: higher tiers inherit and extend lower-tier controls. Controls must be validated via automated policy enforcement where technically feasible.

This guide should be read alongside the main AI Infrastructure Security Playbook, which defines the Risk Gradient (L1–L8), cross-cutting risk modifiers, and the ETSI EN 304 223 alignment rationale. This document focuses on the how rather than the what and why.

This guide assumes a multi-account AWS Organizations structure with Control Tower as the landing zone. Controls reference AWS-native services by default, with third-party alternatives noted where relevant.

For Security Hub, enable both CIS AWS Foundations Benchmark and AWS Foundational Security Best Practices standards. For SageMaker workloads, enable SageMaker-specific AWS Config rules.

2. Getting Started

Before working through the detailed control checklists, follow these five steps to establish your baseline and create a prioritised action plan. This sequence moves from discovery through assessment to actionable remediation, ensuring you focus effort where risk is highest.

Step 1: Build Your AI Inventory

You cannot secure what you do not know exists. Begin by discovering and documenting every AI workload, tool, and service in use across your organisation. Shadow AI is often the largest unmanaged risk.

- ☐ Conduct a discovery exercise across all business units to identify every AI tool, service, and automation in use — including embedded features (Copilot, Gemini, Atlassian Intelligence), citizen developer automations, API integrations, and ML workloads
- ☐ Classify each AI workload against the Risk Gradient (L1–L8) using the tier definitions in the main playbook
- ☐ For each workload, assess the five cross-cutting risk modifiers (Exposure, Data Sensitivity, Autonomy, Integration Privilege, Physical-World Effect) to determine the effective risk level
- ☐ Document the data sensitivity classification for each workload: what data does it access, process, store, or generate? Include both designed access and potential access given current permissions
- ☐ Assign an owner to each AI workload who is accountable for its security posture
- ☐ Record all entries in a centralised AI Inventory (spreadsheet, CMDB, or dedicated AI governance tool) that is maintained as a living document

Step 2: Assess Your Current Security Baseline with CSPM

For existing AI workloads, your Cloud Security Posture Management (CSPM) tooling provides an immediate, evidence-based view of your current gaps. Enable AWS Security Hub (enable CIS and AWS Foundational Security Best Practices standards), Amazon GuardDuty (all regions), and Amazon Inspector (for container and Lambda scanning). Use Security Hub custom insights to filter findings for AI-related resources by tag (e.g. ai-tier:L1–L8) and review findings across all environments hosting AI workloads.

- ☐ Review findings and ensure the following baseline controls are in place for every AI workload:
  - (a) Encryption at rest and in transit is enforced on all AI data stores, model artifacts, and communication channels
  - (b) Identity and access policies follow least privilege — no overly permissive roles on AI resources
  - (c) Network exposure is minimised — no unintended public endpoints on AI services
  - (d) Logging and monitoring are enabled for all AI workloads
  - (e) Vulnerability scanning is active on all compute, containers, and dependencies
- ☐ Record your CSPM secure score or compliance percentage as your starting baseline for AI workloads
- ☐ Export the CSPM findings for your AI workloads for use in Step 3

AWS-specific CSPM setup:

- ☐ Enable Security Hub across all AWS accounts in your Organisation via AWS Organizations delegated administrator
- ☐ Enable GuardDuty in all regions for all accounts (use GuardDuty delegated administrator)
- ☐ Enable Amazon Inspector for EC2, Lambda, and ECR container scanning across all AI workload accounts
- ☐ Enable Amazon Macie on S3 buckets containing AI training data, model artifacts, and agent memory stores
- ☐ Use AWS Config Aggregator for organisation-wide visibility of AI resource compliance
- ☐ Export Security Hub findings to S3 (via EventBridge) for ingestion by the DeepCyber ETSI Agent in Step 3

Step 3: Map Findings to ETSI EN 304 223 with the DeepCyber ETSI Agent

Raw CSPM findings tell you what is misconfigured, but not how it maps to AI-specific security requirements. The DeepCyber ETSI Agent bridges this gap by ingesting your CSPM findings, mapping

them against ETSI EN 304 223 provisions, and generating a tailored, prioritised remediation checklist aligned to your assessed risk gradient level.

- ☐ Feed your exported CSPM findings into the DeepCyber ETSI Agent along with your AI Inventory and risk gradient assessments from Step 1
- ☐ The agent maps each finding to the relevant ETSI EN 304 223 principle and provision, identifies which risk gradient levels are affected, and prioritises remediation by risk severity and blast radius
- ☐ Review the generated checklist: it will highlight quick wins (controls that close multiple ETSI provisions simultaneously) and critical gaps (controls missing at your highest risk gradient levels)
- ☐ Use the prioritised output to create your remediation backlog, assigning owners and target dates to each action item

Step 4: Remediate by Priority

Work through the prioritised remediation checklist, starting with critical findings at your highest effective risk gradient levels. Focus on controls that deliver the greatest risk reduction per unit of effort.

- ☐ Address critical and high-severity findings first, prioritising workloads at the highest effective risk gradient level
- ☐ Target quick wins that close multiple ETSI provisions simultaneously: enforcing encryption (P6), enabling MFA (P6), activating logging (P12), and applying least-privilege access (P5, P6)
- ☐ Use the detailed control checklists in this guide as the reference for implementing each control
- ☐ Track remediation progress against your CSPM secure score and ETSI provision coverage percentage

Step 5: Establish Continuous Governance

AI security is not a one-time exercise. New AI workloads are deployed continuously, cloud configurations drift, and new threats emerge. Establish recurring processes to maintain your security posture over time.

- ☐ Schedule recurring CSPM reviews (minimum monthly) across all AI workload environments
- ☐ Re-run the DeepCyber ETSI Agent periodically (quarterly recommended) to reassess posture against ETSI provisions as your environment evolves
- ☐ Update the AI Inventory whenever new AI workloads are deployed, existing workloads change scope, or workloads are decommissioned
- ☐ Integrate AI workload onboarding into existing change management processes: require risk gradient assessment and security review before any new AI workload enters production
- ☐ Report AI security posture to leadership quarterly using CSPM scores, ETSI provision coverage, and remediation velocity as key metrics

2A. How to Use This Playbook

This playbook is designed for multiple audiences. Use the guide below to find the fastest path to the content most relevant to your role.

Your Role	What to Do	Start Here
CISO / Security Leader	Review the service mapping table and minimum baseline to understand your cloud posture requirements.	Section 3 (Service Mapping), then Minimum Baseline table in Section 3A
Platform / Cloud Engineer	Use the tier checklists as implementation runlists. Each control names the specific cloud service and configuration.	Section 2 (Getting Started), then Section 4 (Implementation Checklists)
Security Architect	Map the service table to your existing cloud architecture and identify gaps per tier.	Section 3 (Service Mapping), then Section 4 (Checklists)
GRC / Compliance	Use the checklists as evidence of control implementation against ETSI EN 304 223 provisions.	Section 2 (Getting Started, Step 3: DeepCyber ETSI Agent), then Section 4 (Checklists)

Tip: All readers should begin with the Getting Started workflow (five steps) to build an AI inventory and establish a baseline before diving into detailed controls.

3. AWS Service Mapping

The following table maps each control dimension to the primary AWS services used to implement the playbook controls.

Control Dimension	Primary Services	Key Configuration
Governance	AWS Config, Audit Manager, Organizations, SCPs, Control Tower	Enable Config rules for AI resources; deploy SCPs restricting AI service usage; use Audit Manager frameworks for ETSI mapping
Identity	IAM, IAM Identity Center, Secrets Manager, KMS, STS	Enforce least-privilege IAM policies; use IAM roles (not keys); store secrets in Secrets Manager with auto-rotation; use KMS CMKs
Data	Macie, Lake Formation, Glue Data Catalog, S3 encryption, DynamoDB encryption	Enable Macie for sensitive data discovery; enforce S3 SSE-KMS; use Lake Formation for fine-grained data access; tag data classifications
Processing	Lambda, ECS/EKS, API Gateway, Step Functions, CodePipeline, CodeGuru, SageMaker	Deploy workloads in ECS/EKS with Fargate; use API Gateway for all AI API calls; enforce CodePipeline gates
Network	VPC, Security Groups, NACLs, PrivateLink, WAF, Shield, Network Firewall, Route 53 Resolver	Isolate AI workloads in dedicated VPCs; use PrivateLink for SageMaker/Bedrock; deploy WAF on API Gateway; enable Shield Advanced
Supply Chain	Inspector, ECR image scanning, CodeArtifact, SBOM generation (Inspector SBOM)	Enable ECR scan-on-push; use Inspector for container and Lambda scanning; maintain CodeArtifact for approved packages
Monitoring & Ops	Security Hub, GuardDuty, CloudTrail, CloudWatch, Detective, EventBridge	Enable Security Hub with CIS/NIST standards; enable GuardDuty for all accounts; aggregate CloudTrail to centralised S3; set CloudWatch alarms

3A. Minimum Baseline Controls by Tier

The following table summarises the minimum required infrastructure controls at each risk gradient level. Use this as a quick-reference before working through the detailed checklists. Controls are cumulative: each tier inherits all controls from lower tiers.

Tier	Pattern	Minimum Required Controls (cumulative)
L1	**Embedded AI**	MFA on all AI-enabled accounts, DLP integration with AI endpoints, AI acceptable use policy, CSPM enabled, prompt/response logging, shadow AI discovery, quarterly access reviews
L2	**AI-Assisted Dev**	+ Mandatory code review for AI-generated code, SAST/DAST in CI/CD, AI code provenance tracking, developer training on AI code risks, dependency scanning on AI-suggested packages
L3	**Citizen Agents**	+ Environment separation (dev/prod), connector permission reviews, DLP on low-code connectors, workflow approval gates, JIT access for agent service accounts, agent inventory register
L4	**API Orchestration**	+ API gateway with rate limiting, credential vault (no hardcoded keys), network segmentation for AI services, input validation on all API endpoints, egress filtering, API key rotation policy
L5	**Autonomous Agents**	+ Tool call allowlisting, agent sandboxing, memory TTL enforcement, human-in-the-loop for high-risk actions, permission boundaries preventing self-escalation, kill switches
L6	**ML Pipelines**	+ Data provenance and lineage tracking, artifact signing and integrity verification, training environment isolation, model registry access controls, pipeline audit logging, SBOM for model dependencies
L7	**Model Hosting**	+ Inference endpoint rate limiting, model extraction detection, adversarial input filtering, private endpoints (no public exposure), model versioning with rollback, A/B deployment gates
L8	**Multi-Agent**	+ mTLS between agents, capability-scoped identity per agent, trust boundary enforcement, circuit breakers, cascade failure detection, real-time agent behaviour monitoring, autonomous privilege escalation prevention

Note: "+" indicates controls added at this tier, in addition to all controls inherited from lower tiers. The detailed checklists in the following section provide the full implementation specification.

Control Dimensions at a Glance

Each dimension applies at every tier. The table shows what changes as you move up the risk gradient.

Dimension	At L1 (Foundational)	At L8 (High Assurance)	ETSI
Governance	AI acceptable use policy, shadow AI audits	AI safety board, autonomous system risk review, kill-switch governance	P1, P3, P4
Identity	MFA, Conditional Access on licences	mTLS, capability-scoped agent identity, JIT with session tokens	P4, P5, P6
Data	DLP on prompts, data classification	Provenance tracking, lineage audit, cross-agent data flow controls	P5, P8, P13
Processing	Tenant config review, feature toggles	Agent sandboxing, capability tokens, tool-call enforcement	P2, P6, P9
Network	Existing segmentation sufficient	Service mesh, private endpoints, per-agent egress rules, circuit breakers	P2, P6
Supply Chain	Vendor DPA review	Model SBOM, artifact signing, training data provenance, dependency pinning	P7
Monitoring	CSPM, basic alerting	Real-time agent behaviour monitoring, cascade detection, anomaly ML	P11, P12

Same seven dimensions at every tier – radically different infrastructure controls. The detailed checklists specify every control.

3C. Common AI Infrastructure Failure Patterns

AI infrastructure failures are rarely model failures – they are identity, network, and supply chain failures amplified by AI capability. The following patterns represent the most common infrastructure-level failures observed in AI deployments. Each maps directly to controls in the tier checklists that follow.

Failure Pattern	What Happens	Controls That Prevent It
Over-permissioned agent with production write access	A citizen-built Power Automate agent granted broad connector permissions writes directly to production HR or finance systems. No approval gate, no audit trail.	L3+: Governance, Identity
LLM API key committed to a public repository	An API key for a paid LLM service is hardcoded in source and pushed to GitHub. Automated scanners find it within minutes. The key has no spending cap or IP restriction.	L4+: Identity, Supply Chain
Fine-tuned model promoted without integrity verification	A model trained on sensitive data is promoted from staging to production with no cryptographic hash check. A tampered artifact enters the serving pipeline undetected.	L6+: Processing, Supply Chain
Inference endpoint exposed without rate limiting	A self-hosted LLM endpoint is deployed on a public subnet with no WAF, no rate limit, and no query pattern monitoring. Systematic extraction queries exfiltrate the model weights.	L7+: Network, Processing, Monitoring
Static service account shared across agent mesh	Multiple agents in a multi-agent system share a single service account with broad IAM permissions. One compromised agent inherits the access of all others.	L8: Identity, Governance
Cross-agent implicit trust enabling lateral compromise	Agents in a multi-agent system accept task delegations from any peer without verifying identity or capability. An injected prompt in one agent propagates through the mesh.	L8: Processing, Network, Identity
Shadow AI tool adopted without security review	A team enables an AI meeting transcription service and grants it access to calendar and email. No DPIA, no DLP, no vendor security review. Sensitive data flows to an unvetted third party.	L1+: Governance, Data
AI-generated code deployed without review	A developer uses AI code completion to generate database queries. The AI hallucinates a dependency and introduces an SQL injection vulnerability. No human review, no SAST scan.	L2+: Processing, Supply Chain

Every pattern above was preventable with controls already in this playbook. The tier checklists that follow provide the specific implementation steps.

4. Implementation Checklists by Risk Gradient Level

Each tier's controls are mapped to specific AWS services with actionable configuration items. Items marked with □ are implementation checkpoints.

4.1 L1: Embedded AI

Example: Amazon Q in AWS Console, Amazon Q Business, Copilot features

Governance

- □ Deploy AWS Organizations SCP to restrict which accounts/OUs can enable AI services (Bedrock, Q, etc.)
- □ Use AWS Config rules to detect non-compliant AI service configurations across the organisation
- □ Enable AWS Audit Manager with custom framework mapping ETSI EN 304 223 awareness requirements
- □ Document AI acceptable use policy in AWS SSO landing page and onboarding materials
- □ Use AWS Config Aggregator to maintain organisation-wide visibility of AI service enablement
- □ Tag all AI-related resources with classification tags (ai:tier=L1, data:classification=internal)

Identity

- □ Use IAM Identity Center (SSO) with MFA enforced for all users accessing AI features
- □ Create dedicated IAM permission sets for AI feature access scoped by user role
- □ Apply IAM condition keys to restrict AI service access by IP range, time, or MFA status
- □ Use SCPs to prevent IAM users from creating long-lived access keys for AI service accounts
- □ Audit AI feature access via IAM Access Analyzer: identify overly permissive policies

Data

- □ Enable Amazon Macie on all S3 buckets accessible to AI services to discover and classify sensitive data
- □ Enforce S3 SSE-KMS encryption on all buckets that AI services read from or write to
- □ Apply S3 bucket policies restricting AI service principals to specific bucket prefixes
- □ Enable CloudTrail data events for S3 buckets accessed by AI services
- □ Use AWS Config rule s3-bucket-ssl-requests-only to enforce TLS on all AI data access
- □ Review Amazon Q / Bedrock data retention settings and disable training data usage where applicable

Processing (Apps, APIs, Integration)

- □ Review Amazon Q Business application settings: verify data source connectors are scoped correctly
- □ Disable AI plugins or extensions not explicitly approved via SCP or IAM policy
- □ Verify AI features respect existing AWS Lake Formation permissions and S3 access policies
- □ Test that AI-generated outputs in Amazon Q do not surface data from restricted data sources

Network

- □ Use VPC endpoints (PrivateLink) for Bedrock and other AI service API calls to keep traffic off public internet
- □ Apply Security Group rules restricting which compute resources can reach AI service endpoints
- □ Deploy Route 53 Resolver DNS Firewall rules to block access to unapproved third-party AI services
- □ Enable VPC Flow Logs on subnets where AI-enabled workloads run
- □ Verify TLS 1.2+ enforcement on all AI service connections (default for AWS SDK)

Supply Chain

- □ Review AWS Shared Responsibility Model for each AI service: document what AWS manages vs customer responsibility
- □ Verify AWS AI service compliance certifications (SOC 2, ISO 27001) via AWS Artifact
- □ Monitor AWS Security Bulletins and AI service changelogs for security-relevant updates

- ☐ Review sub-processor lists for AI services processing customer data

Monitoring and Operations

- ☐ Enable AWS Security Hub across all accounts with CIS AWS Foundations and AWS Foundational Security Best Practices standards
- ☐ Enable GuardDuty in all regions and accounts for anomaly detection
- ☐ Stream CloudTrail logs to centralised S3 bucket and CloudWatch Logs for SIEM ingestion
- ☐ Create CloudWatch alarms for unusual AI API call volumes (Bedrock InvokeModel, Q query spikes)
- ☐ Enable AWS Config conformance packs for continuous compliance monitoring
- ☐ Use Security Hub custom insights to track AI-specific security posture
- ☐ Set up EventBridge rules to alert on AI service configuration changes

4.2 L2: AI-Assisted Development

Example: Amazon CodeWhisperer, Amazon Q Developer, AI code gen tools in IDE

Governance

- ☐ Define approved AI coding tools via SCP restricting CodeWhisperer/Q Developer enablement to approved accounts
- ☐ Publish AI code generation policy in developer onboarding and enforce via CodePipeline gates
- ☐ Use AWS Config to ensure CodeCommit/CodePipeline repos have branch protection rules enabled
- ☐ Require AI-generated code flagging in commit messages (enforced via CodeCommit hooks or CodePipeline Lambda)

Identity

- ☐ Provision CodeWhisperer/Q Developer access via IAM Identity Center with role-based permission sets
- ☐ Enforce MFA on all developer accounts with AI tool access
- ☐ Restrict CodeWhisperer admin settings to platform engineering roles via IAM policies
- ☐ Use IAM Identity Center attribute-based access control (ABAC) to scope AI tool features by team

Data

- ☐ Configure CodeWhisperer reference tracking to identify when suggestions match training data
- ☐ Review Amazon Q Developer data handling: verify code snippets are not retained for model training
- ☐ Use Amazon CodeGuru Secrets Detector in CI/CD to catch hardcoded secrets in AI-generated code
- ☐ Enable Macie to scan any S3 buckets used for code artifacts

Processing (Apps, APIs, Integration)

- ☐ Deploy Amazon CodeGuru Reviewer in CodePipeline for automated security review of all code including AI-generated
- ☐ Integrate Amazon Inspector for Lambda and container vulnerability scanning in CI/CD
- ☐ Enforce CodePipeline approval gates before deployment to production stages
- ☐ Use AWS CodeArtifact to proxy package registries: block AI-suggested packages not in approved upstream
- ☐ Enable CodeBuild SAST integration (e.g. Bandit for Python, Semgrep) on all builds
- ☐ Enforce branch protection: require PR reviews with minimum two approvals on CodeCommit repos

Network

- ☐ Route CodeWhisperer/Q Developer traffic through VPN or Direct Connect where enterprise proxy required
- ☐ Block AI coding tool endpoints from production and CI/CD build environments that do not need them
- ☐ Use VPC endpoints for CodeArtifact to keep dependency downloads on private network

Supply Chain

- ☐ Use CodeArtifact upstream repositories to curate and approve package sources
- ☐ Enable Amazon Inspector SBOM generation for all build artifacts
- ☐ Pin all dependency versions in requirements files; reject unpinned references from AI suggestions
- ☐ Monitor CodeArtifact for dependency confusion attacks targeting AI-suggested package names

Monitoring and Operations

- ☐ Track CodeGuru findings specifically from AI-flagged code to identify recurring vulnerability patterns
- ☐ Alert on new packages introduced to CodeArtifact that are not in the approved catalogue
- ☐ Monitor CodeWhisperer usage metrics via CloudWatch to detect misuse or excessive context sharing
- ☐ Integrate Inspector findings into Security Hub for centralised vulnerability tracking
- ☐ Set up CloudWatch dashboards showing SAST/DAST pass rates per pipeline stage

4.3 L3: Citizen Developer Agents

Example: Amazon Q Apps, AppFlow automations, Step Functions with AI, third-party low-code on AWS

Governance

- ☐ Deploy SCPs restricting which OUs/accounts can create automation resources (Step Functions, AppFlow, Lambda)
- ☐ Maintain a DynamoDB or ServiceCatalog registry of all citizen-built automations with owner and data scope
- ☐ Require AWS Service Catalog provisioned products for citizen developer environments (pre-approved templates)
- ☐ Enforce mandatory approval steps in Step Functions workflows that write to production resources
- ☐ Use AWS Config rules to detect automation resources without required tags (owner, classification, approval-status)
- ☐ Mandate sandbox and production account separation via AWS Organizations OU structure

Identity

- ☐ Use IAM roles with session tags for citizen developer automation execution (not long-lived credentials)
- ☐ Store connector credentials in AWS Secrets Manager with automatic rotation enabled
- ☐ Apply IAM permission boundaries on citizen developer roles to cap maximum permissions
- ☐ Use STS AssumeRole with ExternalId for cross-account connector access
- ☐ Enable IAM Access Analyzer to detect overly permissive automation role policies
- ☐ Implement Step Functions callback patterns for human approval of high-privilege actions

Data

- ☐ Map all AppFlow/Step Functions data flows: tag source and destination with data classification
- ☐ Apply Lake Formation permissions to restrict which data automation pipelines can access
- ☐ Enforce S3 SSE-KMS encryption on all automation data stores
- ☐ Enable CloudTrail data events for all S3, DynamoDB, and RDS resources accessed by automations
- ☐ Use Macie to scan automation output buckets for unintended sensitive data exposure

Processing (Apps, APIs, Integration)

- ☐ Deploy citizen developer workloads in isolated sandbox accounts (AWS Organizations)
- ☐ Use API Gateway between citizen automations and backend services with rate limiting and schema validation
- ☐ Apply Lambda concurrency limits and Step Functions execution quotas to prevent runaway automations
- ☐ Enforce Service Catalog constraints: citizen developers can only deploy approved resource types
- ☐ Test automations against over-privilege scenarios in sandbox before production promotion

Network

- ☐ Deploy citizen developer Lambda and Step Functions in dedicated VPCs with restricted Security Groups
- ☐ Route automation-to-backend traffic through API Gateway or AppMesh, not direct service calls
- ☐ Apply VPC endpoint policies restricting which services citizen automations can reach
- ☐ Block outbound internet from citizen developer VPCs unless explicitly allowlisted via Network Firewall

Supply Chain

- ☐ Use Service Catalog to curate approved AppFlow connectors and Lambda layers
- ☐ Scan all Lambda layers and custom runtimes for vulnerabilities with Inspector
- ☐ Maintain approved connector catalogue in DynamoDB with security ratings
- ☐ Verify third-party connector publisher identity before enabling in citizen developer accounts

Monitoring and Operations

- ☐ Enable Security Hub across citizen developer accounts with automated remediation for critical findings
- ☐ Stream Step Functions execution logs and Lambda logs to centralised CloudWatch Logs / SIEM
- ☐ Create CloudWatch alarms for automation execution frequency spikes and error rate increases
- ☐ Monitor AppFlow transfer volumes and alert on anomalies
- ☐ Use GuardDuty to detect unusual API activity from automation IAM roles
- ☐ Conduct quarterly dormant automation audits: identify unused Step Functions and Lambda with active roles

4.4 L4: Data Analytics & API Orchestration

Example: Python on Lambda/ECS calling Bedrock API, RAG with OpenSearch, Glue ETL with LLM enrichment

Governance

- ☐ Require threat modelling for pipelines processing business confidential data via external AI APIs
- ☐ Document all AI API dependencies and data flows in architecture decision records (ADRs) in CodeCommit
- ☐ Set Bedrock/third-party API cost budgets via AWS Budgets with alerts at 50%, 80%, 100%
- ☐ Use AWS Config to enforce tagging compliance on all pipeline resources

Identity

- ☐ Store all AI API keys in Secrets Manager with automatic rotation (use Secrets Manager Lambda rotation function)
- ☐ Use IAM roles with least-privilege for all pipeline Lambda/ECS tasks: scope Bedrock permissions to specific models
- ☐ Apply Bedrock model access policies to restrict which models each pipeline role can invoke
- ☐ Never embed API keys in Lambda environment variables: use Secrets Manager runtime retrieval
- ☐ Use IAM condition keys to restrict Bedrock API calls by source VPC or IP range

Data

- ☐ Enforce S3 SSE-KMS with customer-managed keys (CMK) for all pipeline data stores
- ☐ Apply data minimisation: use Bedrock guardrails to filter PII before sending to external APIs
- ☐ Enable CloudTrail data events for all S3 and DynamoDB pipeline resources
- ☐ Log Bedrock InvokeModel requests via CloudTrail for audit trail of all AI API interactions
- ☐ Use Glue Data Catalog with Lake Formation to enforce column-level access control on pipeline data
- ☐ Implement data transformation validation: Lambda functions validate output schemas before writing

Processing (Apps, APIs, Integration)

- ☐ Deploy API Gateway as the single entry point for all AI API calls with rate limiting, throttling, and API keys
- ☐ Use Lambda Powertools for structured logging, tracing (X-Ray), and metrics on all pipeline functions
- ☐ Implement Step Functions error handling with retry, exponential backoff, and circuit breaker patterns
- ☐ Separate dev, staging, and prod pipeline environments in distinct AWS accounts (Organizations)
- ☐ Use CodePipeline with CodeBuild for pipeline CI/CD with mandatory security scanning stages
- ☐ Pin all Python/Node.js dependency versions in Lambda layers; scan with Inspector

Network

- ☐ Deploy pipeline Lambda/ECS in VPCs with VPC endpoints for Bedrock, S3, Secrets Manager, and DynamoDB
- ☐ Apply Security Groups restricting pipeline compute to approved API endpoints only
- ☐ Use PrivateLink for Bedrock API calls to keep inference traffic off public internet
- ☐ Implement NAT Gateway egress filtering for any required external API calls (non-AWS)
- ☐ Apply WAF rate limiting rules on API Gateway to throttle excessive requests and prevent automated abuse of AI endpoints
- ☐ Enable API Gateway request validation models to enforce schema validation on all AI API inputs before reaching backend
- ☐ Enforce VPC endpoint-only access for Bedrock and SageMaker APIs; deny public endpoint access via IAM condition keys (aws:sourceVpc)
- ☐ Enable VPC Flow Logs for all pipeline subnets

Supply Chain

- ☐ Scan all Lambda layers and ECS container images for vulnerabilities with Inspector

- ☐ Use CodeArtifact to proxy PyPI/npm and block unapproved packages
- ☐ Pin SDK versions (boto3, anthropic, openai) and monitor for security advisories
- ☐ Generate SBOMs for all pipeline artifacts with Inspector SBOM export

Monitoring and Operations

- ☐ Monitor Bedrock API call patterns via CloudWatch Metrics: alert on volume spikes and error rate increases
- ☐ Track API costs via AWS Cost Explorer and Budgets with automated SNS alerting
- ☐ Stream all pipeline logs to CloudWatch Logs Insights for ad-hoc security investigation
- ☐ Enable Security Hub for pipeline accounts; integrate Inspector findings
- ☐ Use CloudWatch Contributor Insights to identify top callers and data access patterns
- ☐ Implement X-Ray tracing across pipeline components for end-to-end visibility

4.5 L5: Custom Autonomous Agents

Example: LangChain agents on ECS/Lambda, Bedrock Agents, custom tool-calling agents on EKS

Governance

- ☐ Require formal threat model (STRIDE for AI) documented in CodeCommit before agent production deployment
- ☐ Define graduated autonomy tiers in IAM policy: low-privilege tools auto-approved, high-privilege require Step Functions human callback
- ☐ Publish Agent Security Standard defining tool-call boundaries, memory policies, and escalation paths
- ☐ Conduct agent red-team exercises using dedicated adversarial test environments in isolated accounts
- ☐ Define agent decommissioning runbook: IAM role deletion, Secrets Manager cleanup, DynamoDB memory purge

Identity

- ☐ Create dedicated IAM role per agent with least-privilege policies scoped to approved tools only
- ☐ Use Bedrock Agent resource policies to enforce which tools each agent can invoke
- ☐ Implement Step Functions human approval callback for high-privilege agent actions (e.g. S3 delete, SES send)
- ☐ Store agent credentials in Secrets Manager with session-scoped temporary credentials via STS
- ☐ Apply IAM permission boundaries preventing agents from escalating their own permissions
- ☐ Set IAM role MaxSessionDuration to minimum viable value (1 hour default) for all agent execution roles; enforce short-lived STS credentials that require frequent re-authentication
- ☐ Deploy EventBridge-triggered security isolation runbook: on GuardDuty high-severity finding or anomaly alarm, automatically (a) modify the agent IAM role policy to deny all actions, (b) update Security Groups and Network Firewall rules to block all traffic, and (c) revoke future STS role assumptions via IAM policy updates. Note: AWS does not directly revoke issued STS credentials; containment is achieved by removing effective permissions and network reachability. Notify SOC via SNS on all containment actions
- ☐ Audit all agent IAM role assumption and tool invocation events via CloudTrail

Data

- ☐ Encrypt agent memory stores (DynamoDB, OpenSearch, ElastiCache) with KMS CMKs
- ☐ Apply DynamoDB fine-grained access control: agents can only read/write their own memory partition
- ☐ Implement DynamoDB TTL on memory entries to enforce automatic data expiry
- ☐ Use Bedrock Guardrails to filter PII and harmful content from agent inputs and outputs
- ☐ Log all data accessed and generated by agents to CloudWatch Logs for audit trail
- ☐ Apply VPC endpoint policies restricting which data stores agents can reach

Processing (Apps, APIs, Integration)

- ☐ Execute agents in ECS Fargate tasks or EKS pods with restricted IAM roles and no host access
- ☐ Enforce tool-call allowlists via Bedrock Agent action groups or custom Lambda authorizers
- ☐ Apply ECS task-level resource limits (CPU, memory) and Lambda concurrency limits to prevent runaway execution
- ☐ Deploy circuit breakers via Step Functions Choice states that halt execution on error rate thresholds
- ☐ Version all agent configurations, system prompts, and tool definitions in CodeCommit with PR review
- ☐ Implement Bedrock Guardrails as infrastructure-level input/output filtering, not just prompt instructions
- ☐ Test agents against prompt injection and tool abuse in isolated sandbox accounts

Network

- ☐ Deploy agents in dedicated VPC subnets with Security Groups restricting egress to approved tool endpoints only
- ☐ Block all outbound internet from agent subnets; use VPC endpoints for AWS services and PrivateLink for Bedrock
- ☐ Implement Network Firewall rules enabling rapid agent isolation (kill switch)

- ☐ Route agent-to-tool traffic through API Gateway for authentication and rate limiting
- ☐ Enable VPC Flow Logs with CloudWatch alerting on unexpected destination IPs
- ☐ Apply EKS Kubernetes Network Policies (Calico or Cilium CNI) to enforce pod-level microsegmentation between agents and supporting services

Supply Chain

- ☐ Pin LangChain/agent framework versions in container images; scan with Inspector before deployment
- ☐ Use CodeArtifact to proxy all agent framework dependencies
- ☐ Audit prompt templates sourced from external repositories before use
- ☐ Verify integrity of external tool APIs consumed by agents (TLS certificate pinning where feasible)
- ☐ Generate SBOM for all agent containers with Inspector

Monitoring and Operations

- ☐ Log all agent tool calls with parameters, responses, and timestamps to CloudWatch Logs in structured JSON format
- ☐ Establish behavioural baselines per agent in CloudWatch Metrics; create anomaly detection alarms
- ☐ Monitor DynamoDB/OpenSearch memory store growth with CloudWatch alarms
- ☐ Integrate agent logs into SIEM (Sentinel, Splunk) with correlation rules for prompt injection and tool abuse
- ☐ Enable Security Hub and GuardDuty for all agent accounts
- ☐ Deploy canary inputs via EventBridge scheduled rules to verify agent behaviour periodically
- ☐ Implement CloudWatch alarms on high-privilege tool calls for SOC visibility even when approved

4.6 L6: Data Science & ML Pipelines

Example: SageMaker training, fine-tuning on Bedrock, Step Functions ML pipelines, EMR feature engineering, MLflow on ECS

Governance

- ☐ Require formal threat model covering training-time attacks for all SageMaker/Bedrock fine-tuning projects
- ☐ Mandate model risk assessment and human sign-off before SageMaker endpoint promotion to production
- ☐ Use SageMaker Model Cards for all production models documenting purpose, data, limitations, and biases
- ☐ Implement SageMaker Model Registry approval workflows with mandatory security review stage
- ☐ Deploy SageMaker ML Governance features: Role Manager, Model Dashboard, Model Monitor
- ☐ Use AWS Config to enforce tagging and encryption compliance on all SageMaker resources

Identity

- ☐ Apply SageMaker execution roles with least-privilege: separate roles for notebook, training, and deployment
- ☐ Use Lake Formation to enforce fine-grained access to training data (column/row-level security)
- ☐ Restrict SageMaker Model Registry write access to CodePipeline service role only (no manual model uploads)
- ☐ Implement IAM permission boundaries on data scientist roles capping maximum permissions
- ☐ Use Secrets Manager for all model API keys and external service credentials with auto-rotation
- ☐ Enforce MFA on all SageMaker Studio user profiles
- ☐ Sign model artifacts with AWS Signer or store SHA-256 hashes in Model Registry metadata for integrity verification

Data

- ☐ Encrypt SageMaker training volumes, S3 training data, and model artifacts with KMS CMKs
- ☐ Use Lake Formation to isolate training data from production data stores
- ☐ Enable S3 Object Lock (WORM) for critical training datasets to prevent modification
- ☐ Implement SageMaker Data Wrangler or Glue for documented, reproducible data transformations
- ☐ Use Macie to scan training data S3 buckets for unintended PII or sensitive data
- ☐ Apply S3 Lifecycle policies for training data retention and automatic deletion
- ☐ Version training datasets using S3 versioning or DVC with S3 backend
- ☐ Enable CloudTrail data events on all S3 buckets containing training data
- ☐ Use SageMaker Feature Store with offline store encryption and access auditing

Processing (Apps, APIs, Integration)

- ☐ Deploy SageMaker training jobs in dedicated VPCs with isolated subnets and restricted Security Groups
- ☐ Enable SageMaker Network Isolation mode (EnableNetworkIsolation=true) on all training and processing jobs to prevent containers from making outbound network calls during execution
- ☐ Use SageMaker Pipelines (or Step Functions) for immutable, versioned pipeline definitions
- ☐ Apply SageMaker training job resource limits: max runtime, instance count, and spot instance allocation
- ☐ Enforce SageMaker notebook lifecycle configurations for security hardening (disable root, install security agents)
- ☐ Scan all SageMaker container images with ECR image scanning and Inspector before use
- ☐ Deploy SageMaker Clarify for bias detection as a mandatory pipeline stage
- ☐ Implement model validation gates: accuracy, bias, robustness tests must pass before Model Registry approval
- ☐ Use SageMaker Debugger to detect training anomalies (exploding gradients, unusual loss patterns)

Network

- ☐ Deploy SageMaker in VPC-only mode: training, processing, and inference all run in customer VPC
- ☐ Apply Security Groups restricting SageMaker instances to approved data sources and package registries only
- ☐ Use VPC endpoints for S3, ECR, SageMaker API, and CloudWatch from training subnets
- ☐ Block all internet access from training and notebook subnets via Network Firewall or NAT Gateway removal
- ☐ Use PrivateLink for SageMaker Studio and API access
- ☐ Enable VPC Flow Logs on all ML workload subnets

Supply Chain

- ☐ Verify provenance of pre-trained base models from SageMaker JumpStart or HuggingFace Hub
- ☐ Use ECR image scanning for all custom training and inference containers
- ☐ Pin all ML framework versions (PyTorch, TensorFlow, transformers) in container Dockerfiles
- ☐ Use CodeArtifact for Python packages consumed by training pipelines
- ☐ Generate SBOM for all ML containers with Inspector SBOM export
- ☐ Monitor ML framework security advisories and apply patches within defined SLAs
- ☐ Store model checkpoints in S3 with versioning and integrity hashes in Model Registry metadata

Monitoring and Operations

- ☐ Enable SageMaker Model Monitor for data drift, model quality, bias drift, and feature attribution drift
- ☐ Alert on training job resource utilisation anomalies (potential cryptomining or unauthorised workloads)
- ☐ Monitor Model Registry operations via CloudTrail: alert on model promotion, deletion, or metadata changes
- ☐ Enable Security Hub for all ML accounts with SageMaker-specific Config rules
- ☐ Use SageMaker Experiments to track all hyperparameters, metrics, and artifact lineage
- ☐ Integrate SageMaker CloudWatch metrics into centralised monitoring dashboards
- ☐ Run periodic model integrity checks: compare deployed model hash against Model Registry approved hash
- ☐ Enable GuardDuty for ML accounts to detect unusual IAM activity

4.7 L7: Model Hosting & Serving

Example: SageMaker real-time endpoints, SageMaker serverless inference, self-hosted vLLM on EKS, Bedrock custom model endpoints

Governance

- ☐ Publish Model Serving Security Standard covering endpoint hardening, rate limiting, and monitoring requirements
- ☐ Require security review before any SageMaker endpoint or EKS inference service is exposed
- ☐ Implement SageMaker endpoint promotion pipeline with mandatory security gate in CodePipeline
- ☐ Define rate-limiting policies designed to impede model extraction (Provision 5.2.2-2)
- ☐ Maintain inference endpoint inventory in SageMaker Model Dashboard or custom DynamoDB registry
- ☐ Define rollback SLA: ability to revert to previous model version using SageMaker endpoint update

Identity

- ☐ Apply IAM resource policies on SageMaker endpoints restricting which IAM roles can invoke them
- ☐ Use API Gateway with API keys and usage plans for per-consumer rate limiting and quota management
- ☐ Implement AWS WAF API key validation and client certificate authentication for external consumers
- ☐ Store inference API credentials in Secrets Manager with automatic rotation
- ☐ Monitor SageMaker InvokeEndpoint CloudTrail events for consumer identity and volume tracking
- ☐ Apply SageMaker endpoint tag-based access control via IAM condition keys

Data

- ☐ Enforce TLS 1.2+ on all SageMaker endpoints (default) and EKS ingress controllers
- ☐ Apply input validation via SageMaker inference pipeline or API Gateway request validation
- ☐ Implement output filtering to strip PII from model responses using post-processing Lambda
- ☐ Log inference requests and responses (sample or full) to S3 via SageMaker Data Capture
- ☐ Enable SageMaker Data Capture with KMS encryption for audit and forensics
- ☐ Apply content filtering on model outputs via Bedrock Guardrails or custom Lambda post-processing

Processing (Apps, APIs, Integration)

- ☐ Verify model artifact integrity before deployment: compare S3 model hash against Model Registry approved hash
- ☐ Implement canary deployments using SageMaker production variants with traffic splitting
- ☐ Deploy blue-green infrastructure using SageMaker endpoint update with rollback capability
- ☐ Run SageMaker inference containers with minimal base images, non-root execution, read-only file systems
- ☐ Apply EKS pod security standards (restricted profile) for self-hosted inference on Kubernetes
- ☐ Implement API Gateway request size limits, timeout enforcement, and throttling
- ☐ Test endpoints against adversarial inputs and model extraction probes in staging environment

Network

- ☐ Place SageMaker endpoints behind API Gateway or ALB with WAF rules for malicious request filtering
- ☐ Enable AWS Shield Advanced on inference ALBs/API Gateway for DDoS protection
- ☐ Deploy SageMaker endpoints in VPC for private internal access; use PrivateLink for cross-account consumers
- ☐ Implement Security Groups restricting inference container egress (no outbound internet)
- ☐ Enable VPC Flow Logs on inference subnets
- ☐ Use Network Firewall for granular egress filtering if inference containers need external access

Supply Chain

- ☐ Verify model artifact provenance before serving: SageMaker Model Registry hash matches deployed artifact
- ☐ Scan all inference container images with ECR scanning and Inspector before deployment
- ☐ Pin serving framework versions (TGI, vLLM, Triton) and monitor for security advisories
- ☐ Implement ECR image signing with AWS Signer and verify signatures in deployment pipeline

Monitoring and Operations

- ☐ Monitor SageMaker endpoint latency, error rates, and invocation counts via CloudWatch with auto-scaling policies
- ☐ Enable SageMaker Model Monitor for output distribution drift detection
- ☐ Deploy canary queries via EventBridge + Lambda and alert on output deviation from baseline
- ☐ Create CloudWatch anomaly detection alarms on invocation patterns to identify extraction attempts
- ☐ Deploy extraction-specific detection rules: alert on systematic query patterns, unusual token-length distributions, high-frequency enumeration of model outputs, and repetitive boundary-probing inputs. Source detection telemetry from API Gateway and AWS WAF access logs, SageMaker Data Capture logs, CloudWatch Logs Insights queries, and GuardDuty anomaly findings. Correlate request frequency, token distributions, and input variation patterns to identify potential extraction attempts
- ☐ Enable Security Hub and GuardDuty for inference hosting accounts
- ☐ Integrate SageMaker Data Capture logs into SIEM for forensic analysis
- ☐ Monitor SageMaker endpoint deployment events via CloudTrail: alert on unexpected model changes
- ☐ Enable Inspector for continuous vulnerability scanning on EKS inference nodes

4.8 L8: Distributed / Multi-Agent

Example: Multi-agent on EKS with Bedrock Agents, Step Functions orchestrating agent swarms, A2A/MCP on AWS

Governance

- ☐ Require system-level threat modelling covering inter-agent trust, lateral movement, and cascading failure
- ☐ Publish Multi-Agent Security Architecture Standard defining trust boundaries and communication protocols
- ☐ Mandate zero-trust: no implicit trust between agents regardless of VPC or account location. Agents must never be permitted to modify their own IAM roles, permission boundaries, or trust policies under any circumstance
- ☐ Implement infrastructure-level kill switches using Step Functions Choice + Lambda + Security Group rules
- ☐ Conduct multi-agent red team exercises in isolated sandbox accounts
- ☐ Define blast radius boundaries: use AWS account boundaries or EKS namespace isolation per trust domain
- ☐ Establish agent registration authority (DynamoDB registry with approval workflow)

Identity

- ☐ Assign each agent a unique IAM role with cryptographic identity (IAM role session tags + STS)
- ☐ Implement mTLS between agents using AWS Certificate Manager Private CA
- ☐ Apply IAM resource policies on each agent's tools preventing cross-agent access without explicit delegation
- ☐ Implement signed, short-lived JWT capability tokens issued by a central authorisation service or orchestrator. Tokens must be audience-bound (specific to the receiving agent or service), contain explicit action scope (tool or API permission), expire within minutes (no long-lived delegation), and be validated cryptographically by the receiving agent before execution. Agents must never rely solely on prompt instructions to determine authorisation
- ☐ Prevent trust delegation abuse: use STS confused deputy prevention (ExternalId) for cross-agent role assumption
- ☐ Deploy agent identity rotation using short-lived STS credentials (max 1 hour session)
- ☐ Audit all inter-agent role assumption events via CloudTrail
- ☐ Apply IAM deny policies on all agent roles preventing iam:CreateRole, iam:AttachRolePolicy, iam:PutRolePolicy, and sts:AssumeRole to self – agents must never be able to mutate their own permissions or assume their own role recursively
- ☐ Enforce IAM permission boundaries on all agent execution roles capping maximum achievable permissions regardless of policy attachments

Data

- ☐ Encrypt all inter-agent messages with ACM Private CA mTLS and optional payload encryption via KMS
- ☐ Implement signed messages between agents using KMS asymmetric keys for integrity and non-repudiation
- ☐ Prevent data leakage between trust domains using VPC endpoint policies and IAM resource policies
- ☐ Log all inter-agent data exchanges to CloudWatch Logs in structured format for audit trail
- ☐ Apply output validation on agent-generated tasks before delegation to peer agents

Processing (Apps, APIs, Integration)

- ☐ Deploy centralised orchestration plane using Step Functions with human override (callback pattern)
- ☐ Implement circuit breakers in Step Functions that halt inter-agent workflows on error rate thresholds
- ☐ Isolate each agent in its own ECS Fargate task or EKS pod with no shared filesystem or memory
- ☐ Apply EKS network policies preventing inter-pod communication except through defined service mesh routes

- ☐ Enforce EKS Pod Security Admission (PSA) at “restricted” level on all agent namespaces: non-root, read-only rootfs, no privilege escalation, no host networking
- ☐ Deploy OPA Gatekeeper or Kyverno admission controllers on EKS to enforce agent-specific policies: allowed container registries, required labels, resource limits, prohibited volume mounts, and mandatory sidecar injection for mesh telemetry
- ☐ Deploy health checking across agent mesh using ECS Service Discovery or EKS liveness/readiness probes
- ☐ Implement rate limiting on inter-agent SQS/SNS/EventBridge messaging to prevent message flooding
- ☐ Test for emergent behaviour in adversarial simulation environments (fault injection with AWS FIS)

Network

- ☐ Apply EKS NetworkPolicy or Security Groups preventing lateral movement between agent trust domains
- ☐ Enforce mutual TLS (mTLS) between agents using AWS App Mesh with ACM Private CA for certificate issuance and rotation. Where App Mesh is not deployed, implement mTLS via Envoy sidecars or equivalent service-mesh architecture. The requirement is cryptographic peer authentication and encrypted east-west traffic – the implementation mechanism may vary
- ☐ Segment agent communication VPCs from broader infrastructure using Transit Gateway route tables
- ☐ Deploy Network Firewall kill switches that can isolate agent groups by modifying firewall rules via Lambda
- ☐ Enable VPC Flow Logs with CloudWatch anomaly detection on inter-agent traffic patterns
- ☐ Apply AWS Shield Advanced on any internet-facing agent infrastructure

Supply Chain

- ☐ Verify identity and integrity of all agents before mesh admission using agent registration authority
- ☐ Scan all agent container images across the mesh with Inspector and ECR scanning
- ☐ Implement binary attestation: require signed container images (AWS Signer) for mesh participation
- ☐ Monitor for compromised or rogue agents: alert on unregistered agent identity attempting mesh communication
- ☐ Generate and maintain SBOM covering all agents, frameworks, and communication infrastructure

Monitoring and Operations

- ☐ Deploy system-level monitoring: CloudWatch dashboards showing inter-agent message volumes, error rates, and latency per agent pair
- ☐ Implement cascade detection: CloudWatch composite alarms that trigger when correlated failures appear across multiple agents
- ☐ Monitor inter-agent trust delegation chains in CloudTrail and alert on anomalous patterns
- ☐ Enable Security Hub and GuardDuty across all accounts in the multi-agent deployment
- ☐ Enable Amazon Detective to correlate GuardDuty findings across agent accounts: identify lateral movement chains, anomalous role-assumption graphs, and cross-agent privilege escalation patterns
- ☐ Integrate agent mesh monitoring into SIEM with correlation rules for lateral movement patterns
- ☐ Use AWS Fault Injection Simulator (FIS) for periodic chaos engineering to validate resilience
- ☐ Alert on unexpected agent registration or deregistration in the agent registry
- ☐ Retain full interaction history in S3 with Object Lock for post-incident forensic analysis