🛠️ NetForge Technologies & Architecture

A comprehensive breakdown of the technologies used in the **NetForge** platform, organized around the consistent theme of an **Autonomous Reliability and AI-driven Ops Architecture**.

🧠 Core AI & Agent Orchestration Layer

The intelligence engine that powers NetForge's diagnostic reasoning, tool execution, and continuous learning.

- **AWS Strands Agents (`strands-agents`)**: The core agentic framework that powers the orchestrator agent. It manages the execution of 20+ specialized tools across cloud and monitoring boundaries.
- **MiniMax Model (`MiniMax M2.5`)**: The primary LLM used in a dual-model architecture:
  - *Main Orchestrator*: Synchronously handles tool calls, log contextualization, and user-facing remediation summaries.
  - *Background Sub-Model*: Triggered asynchronously for deep pattern analysis and latency insight generation without blocking the user.
- **LiteLLM**: The interface layer utilized to connect the agent execution framework to the MiniMax provider seamlessly.
- **CopilotKit (Backend `AgentCore`)**: Implements the AG-UI Server-Sent Events (SSE) streaming protocol, allowing the agent to stream its thought process directly to the frontend.

🔭 Observability & Validation Layer

The sensory inputs that allow the agent to monitor system health and validate its own remediations.

- **Datadog MCP (Model Context Protocol)**: Integrated as a `stdio` subprocess, granting the agent native access to Datadog's search capabilities, events, and monitors.
- **Datadog REST APIs**: 6 direct Python REST tools configured in the backend to pull high-volume metrics, container resources, and system alerts without subprocess overhead.
- **TestSprite**: Automated testing oracle utilized strictly to validate network stability and service recovery pre- and post-remediation (e.g., verifying safe scaling events).

🕸️ Graph Intelligence & Memory Layer

The topology and knowledge base that allows NetForge to calculate causation and "remember."

- **Neo4j (Graph Database)**: Maintains the core service topology. The agent navigates this via the Neo4j MCP Server to perform blast-radius calculation, trace cascading latency bottlenecks, and execute Text2Cypher queries.
- **Custom JSON-Backed Memory (`store.py`)**: Persistent intelligent memory where the background MiniMax model stores optimizations, historical baselines, and architectural edge cases tagged with `[MiniMax]`.

⚙️ Backend API & Control Loop Layer

The control plane built to handle REST pathways, streaming, and the autonomic loop.

- **Python 3.11**: Primary runtime for the backend intelligence stack.

- **FastAPI**: Used for its high-performance asynchronous execution. Handles the webhooks, cluster simulation, and standard REST requests (`/api/hooks/` , `/api/agent/`).
- **Uvicorn**: The ASGI web server implementation serving FastAPI.
- **SSE-Starlette**: Powers the Server-Sent Events mechanism crucial for streaming token-by-token LLM responses directly to the frontend.
- **Pydantic**: Provides rapid data validation and strictly-typed serialization between agent JSON configurations, API requests, and environment settings.

🖥️ Frontend AI-Native Interface Layer

The real-time interactive dashboard providing visibility into the agent's operations.

- **React 18 & TypeScript**: The core SPA framework powering the real-time operational dashboard component behaviors.
- **D3.js**: Selected specifically for rendering the Force-Directed topology graph. It handles physics simulations, glowing anomaly visualizations, and animated trace routes.
- **CopilotKit React Core / React UI**: Embeds the conversational co-pilot sidebar into the React app. Leverages hooks like `useCopilotReadable` to feed live screen context (what service the user clicked on) securely to the LLM context window.
- **Lucide React**: A lightweight icon suite that aligns UI elements cleanly across insights, cluster control logic, and agent reasoning.

☁️ Infrastructure & Execution Layer

The actionable boundary where NetForge actually changes cloud environments (The "Execute" phase of the MAPE-K loop).

- **AWS SDK (`boto3`)**: Powers the direct remediation tools (`aws_tools.py`) that perform ECS Service scaling, CodeDeploy target rollbacks, and AWS Systems Manager (SSM) parameter updates.
- **Docker & Docker Compose**: Utilized for standardizing the local development footprint, particularly in spinning up the backend services and local Neo4j database containers effortlessly.