Docker MCP Gateway: Enabling MCP-as-a-Service for Enterprise AI Integration

A Technical Guide to Multi-Client Model Context Protocol Deployment

NOAA EMC Global Workflow MCP Team
Environmental Modeling Center
National Oceanic and Atmospheric Administration
Terry.McGuinness@noaa.gov

December 15, 2025

Abstract

The Model Context Protocol (MCP) has emerged as a critical standard for enabling AI assistants to interact with external tools and data sources. However, the default stdio-based transport mechanism limits MCP servers to single-client, single-session usage patterns. This paper presents the **Docker MCP Gateway**—an open-source solution that transforms containerized MCP servers into multi-client services accessible via HTTP/SSE transports. We describe the architectural challenges of MCP-as-a-Service deployment, analyze the gateway's security model, and demonstrate practical implementation within the NOAA Global Workflow AI assistance infrastructure. Our deployment enables simultaneous access from VS Code Copilot, Claude Desktop, LangFlow, and custom applications to a unified 32-tool MCP server backed by vector and graph databases. This work addresses the critical gap between MCP's powerful tool integration capabilities and enterprise requirements for scalable, multi-tenant AI service deployment.

Keywords: Model Context Protocol, Docker, Gateway Pattern, AI Infrastructure, Multi-Client Services, Container Orchestration, RAG Systems

1 Introduction

1.1 The MCP Revolution

The Model Context Protocol (MCP), introduced by Anthropic in late 2024, represents a paradigm shift in how AI assistants interact with external systems. Unlike traditional approaches where each AI application implements custom integrations, MCP provides a standardized protocol enabling:

- **Tool Discovery**: AI clients can enumerate available tools dynamically
- **Structured Invocation**: Type-safe parameter passing with JSON Schema validation
- **Resource Access**: Standardized file, database, and API access patterns
- **Prompt Templates**: Reusable prompt components shared across applications

The MCP specification defines a JSON-RPC 2.0 based protocol that enables AI clients (Claude, VS Code Copilot, etc.) to communicate with *MCP servers*—specialized processes that expose tools, resources, and prompts to the AI.

1.2 The Single-Client Limitation

Despite MCP's power, its default transport mechanism—standard input/output (stdio)—creates a fundamental limitation: **one client, one session, one server instance**. This design, while elegant for local development, prevents:

1. Multiple developers sharing a common MCP server
2. Integration with web-based tools (LangFlow, n8n, custom dashboards)
3. Centralized logging and monitoring of tool invocations
4. Load balancing across multiple server instances
5. Separation of MCP server deployment from client workstations

For enterprise deployments requiring centralized AI tool infrastructure, this limitation is critical.

1.3 Our Contribution

This paper presents:

1. Analysis of the **Docker MCP Gateway** architecture and its role in enabling MCP-as-a-Service
2. Practical deployment patterns for **containerized MCP servers** with database dependencies
3. Security considerations for **multi-client MCP access**
4. Implementation details from our **NOAA Global Workflow** deployment with 32 tools serving multiple AI clients simultaneously

2 Background: MCP Transport Mechanisms

2.1 Standard I/O Transport (Default)

The MCP specification's primary transport uses Unix standard streams:

```
# Client spawns server process
node mcp-server.js

# JSON-RPC messages flow through stdin/stdout
< {"jsonrpc":"2.0","id":1,"method":"initialize",...}
> {"jsonrpc":"2.0","id":1,"result":{...}}
```

Listing 1: Stdio MCP Communication

Advantages:

- Simple process management
- No network configuration required
- Inherent security (local process only)

Limitations:

- Single client per server instance
- Client must manage server lifecycle
- No remote access capability

2.2 Server-Sent Events (SSE) Transport

MCP also supports SSE for network-based communication:

```
# Server listens on HTTP port
POST /mcp HTTP/1.1
Content-Type: application/json

{"jsonrpc":"2.0","id":1,"method":"tools/list"}

# Server responds with SSE stream
event: message
data: {"jsonrpc":"2.0","id":1,"result":{"tools":[...]}}
```

Listing 2: SSE MCP Communication

Advantages:

- Network-accessible
- Multiple clients possible (with session management)
- Standard HTTP infrastructure compatibility

Limitations:

- Server must implement HTTP handling
- Session state management complexity
- Security considerations (authentication, CORS)

2.3 The Gateway Pattern

The **Docker MCP Gateway** bridges these transports: it accepts HTTP/SSE connections from multiple clients and translates them to stdio connections with containerized MCP servers. This enables MCP-as-a-Service without modifying existing stdio-based servers.

3 Docker MCP Gateway Architecture

3.1 System Overview

The gateway operates as a protocol bridge:

1. **Client Layer**: Multiple AI clients connect via HTTP/SSE
2. **Gateway Layer**: Manages sessions, authentication, and routing
3. **Server Layer**: Containerized MCP servers accessed via Docker stdio
4. **Data Layer**: Backend databases (vector, graph, relational)

3.2 Key Components

3.2.1 Container Discovery

The gateway discovers MCP servers through Docker image labels:

```
LABEL io.docker.server.metadata='{
  "name": "eib-mcp-rag",
  "title": "EIB MCP RAG Server",
  "description": "AI-powered MCP server with RAG",
```

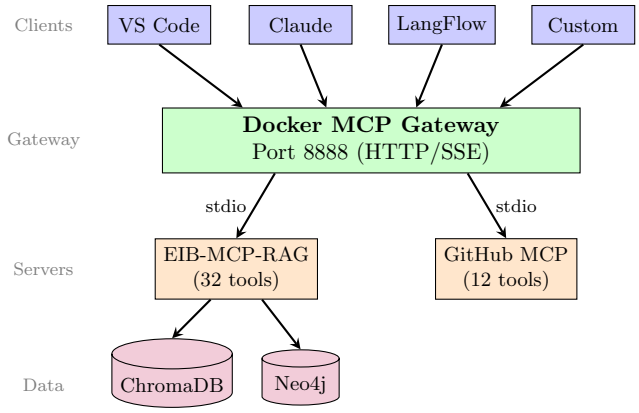

Figure 1: Docker MCP Gateway Architecture: Multiple AI clients connect via HTTP/SSE to the gateway, which translates requests to stdio-based MCP servers running in Docker containers.

```
  "type": "image",
  "image": "eib-mcp-rag:latest"
}'
```

Listing 3: MCP Server Discovery Label

This label enables automatic server registration when images are pulled or built.

3.2.2 Session Management

Each client connection receives a unique session:

```
POST /mcp HTTP/1.1
Authorization: Bearer <token>
Content-Type: application/json

{"jsonrpc":"2.0","id":1,"method":"initialize",...}

# Response includes session ID
event: message
id: YMXXJUBCRW5VJHR5XJY74MR3VP_0
data: {"jsonrpc":"2.0","id":1,"result":{...}}
```

Listing 4: Session Initialization

Sessions maintain state for multi-turn tool interactions.

3.2.3 Container Lifecycle

The gateway manages container lifecycle automatically:

- **Startup**: Container spawned on first tool invocation
- **Communication**: JSON-RPC via Docker's stdio attachment
- **Cleanup**: Container removed after session ends (unless `--long-lived`)

4 Security Model

4.1 Network Isolation

A critical security feature: the gateway runs MCP server containers in **network isolation** by default. Containers cannot access:

- Host network interfaces
- Other containers
- External services

This prevents malicious tools from exfiltrating data or attacking internal services.

4.2 Authentication

The gateway supports bearer token authentication:

```
# Set token via environment
export MCP_GATEWAY_AUTH_TOKEN="secret-token-value"

# Start gateway
docker mcp gateway run --port 8888 --transport streamable-http

# Clients must include token
curl -H "Authorization: Bearer secret-token-value" \
     -X POST http://localhost:8888/mcp
```

Listing 5: Gateway Authentication

4.3 Resource Limits

Per-container resource constraints prevent denial-of-service:

```
docker mcp gateway run \
  --cpus 1 \
  --memory 2Gb \
  --servers docker://eib-mcp-rag:latest
```

Listing 6: Resource Constraints

4.4 Challenge: Database Connectivity

Network isolation creates a challenge for MCP servers requiring database access. Our RAG-enabled server needs ChromaDB and Neo4j connections. Solutions:

1. **Gateway Bypass**: Run container directly on shared network (our approach)
2. **Remote Endpoints**: Configure databases with public endpoints
3. **Sidecar Pattern**: Deploy databases alongside MCP container

5 Implementation: NOAA Global Workflow Deployment

5.1 System Context

Our deployment supports AI-assisted development of the NOAA Global Forecast System (GFS)—operational weather prediction infrastructure processing petabytes of atmospheric data daily. The MCP server provides:

- **32 Tools**: Code analysis, documentation search, compliance checking
- **14,854 Documents**: Indexed in ChromaDB vector database
- **85,894 Relationships**: Code dependency graph in Neo4j
- **Multiple Clients**: VS Code Copilot, Claude Desktop, LangFlow

5.2 Container Configuration

Our Dockerfile includes gateway metadata:

```
FROM node:20-slim

# ... build steps ...

# Gateway discovery label (JSON format)
LABEL io.docker.server.metadata='{␣\
␣␣"name":"eib-mcp-rag",␣\
␣␣"title":"EIB␣MCP␣RAG␣Server",␣\
␣␣"description":"AI-powered␣MCP␣server␣with␣RAG",␣\
␣␣"type":"image",␣\
␣␣"image":"eib-mcp-rag:latest"␣\
}'

CMD ["node", "src/UnifiedMCPServer.js", "full"]
```

Listing 7: Dockerfile with Gateway Support

5.3 Network Architecture

Due to database requirements, we deploy with explicit network configuration:

```
# Create shared network
docker network create mcp-network

# Start databases
docker run -d --network mcp-network \
  --name chromadb chromadb/chroma:latest

docker run -d --network mcp-network \
  --name neo4j neo4j:5-community

# Start MCP server on same network
docker run -d --network mcp-network \
  --name eib-mcp-rag \
  -e CHROMADB_HOST=chromadb \
  -e CHROMADB_PORT=8000 \
  -e NEO4J_URI=bolt://neo4j:7687 \
  eib-mcp-rag:latest
```

Listing 8: Production Container Deployment

5.4 Gateway Deployment

For multi-client access, we run the gateway in streaming mode:

```
export MCP_GATEWAY_AUTH_TOKEN="$(openssl␣rand␣-hex␣32)"

docker mcp gateway run \
  --servers docker://eib-mcp-rag:latest \
  --port 8888 \
  --transport streamable-http \
  --verbose
```

Listing 9: Gateway Startup

Gateway output confirms tool discovery:

```
- Those servers are enabled: eib-mcp-rag
- Listing MCP tools...
  > eib-mcp-rag: (32 tools)
> 32 tools listed in 2.65s
> Start streaming server on port 8888
> Gateway URL: http://localhost:8888/mcp
```

Listing 10: Gateway Initialization Output

5.5 Client Configuration

5.5.1 VS Code MCP Extension

```
{
  "servers": {
    "eib-mcp-rag-gateway": {
      "type": "sse",
      "url": "http://localhost:8888/mcp",
      "headers": {
        "Authorization": "Bearer ${MCP_GATEWAY_AUTH_TOKEN}"
      }
    }
  }
}
```

Listing 11: VS Code mcp.json Configuration

5.5.2 LangFlow Integration

LangFlow connects via its MCP component, enabling visual workflow construction with MCP tools as nodes.

6 Operational Considerations

6.1 Tool Verification

We verify tool availability through the gateway:

```
curl -X POST http://localhost:8888/mcp \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"jsonrpc":"2.0","id":1,"method":"tools/list"}'

# Response includes all 32 tools
```

Listing 12: Tool List Verification

6.2 Health Monitoring

Our MCP server includes health check tools:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "get_knowledge_base_status",
    "arguments": {"include_graph": true}
  }
}
```

Listing 13: Health Check Tool Call

Response confirms database connectivity:

```
Vector Database (ChromaDB):
  - Collections: 12
  - Total Documents: 14,854
  - Status: Healthy

Graph Database (Neo4j):
  - Total Relationships: 85,894
  - Status: Healthy
```

Listing 14: Health Check Response

6.3 Provisioning Automation

We automated gateway deployment in our provisioning system:

```
# 11-docker-mcp-gateway.sh

# Install Go compiler
curl -sLO "https://go.dev/dl/go1.23.4.linux-amd64.tar.gz"
tar -C /usr/local -xzf go1.23.4.linux-amd64.tar.gz

# Build docker-mcp plugin
git clone https://github.com/docker/mcp-gateway.git
cd mcp-gateway
go build -o docker-mcp ./cmd/docker-mcp

# Install plugin
cp docker-mcp ~/.docker/cli-plugins/

# Build MCP server image
```

```
16 docker compose -f docker-compose.mcp-standalone.yaml build
```

Listing 15: Provisioning Script (excerpt)

7 Lessons Learned

7.1 Docker CE vs Docker Desktop

A critical finding: the `docker-mcp` plugin initially only worked with Docker Desktop due to API differences. Pull Request #301 (merged December 12, 2025) added Docker CE support. **Building from source after this merge is required for Docker CE deployments.**

7.2 Label Format

The `io.docker.server.metadata` label must use **JSON format**, not YAML. Multi-line YAML with escaped newlines causes parsing failures:

```
# WRONG - causes parsing errors
LABEL io.docker.server.metadata="name: eib\ntitle: EIB"

# CORRECT - JSON format
LABEL io.docker.server.metadata='{"name":"eib","title":"EIB"}'
```

Listing 16: Label Format Comparison

7.3 Network Isolation Trade-offs

Gateway security isolation prevents database access. For stateful MCP servers requiring backend services, direct network deployment (bypassing gateway container management) provides the necessary connectivity while maintaining protocol translation benefits.

8 Future Work

1. **Gateway Network Configuration**: Propose upstream feature for gateway to attach containers to specified networks
2. **Multi-Tenant Authentication**: Integrate with institutional identity providers (OAuth, SAML)
3. **Tool-Level Authorization**: Fine-grained access control per tool per user
4. **Observability**: Distributed tracing for tool invocations across sessions
5. **High Availability**: Gateway clustering with session affinity

9 Conclusion

The Docker MCP Gateway transforms the Model Context Protocol from a single-client development tool into enterprise-ready infrastructure. By bridging stdio and HTTP/SSE transports, it enables:

- **Multi-Client Access**: Multiple AI assistants sharing common tools
- **Centralized Deployment**: MCP servers as managed services
- **Security Controls**: Authentication, resource limits, network isolation

- **Container Benefits**: Reproducible builds, version management, scaling

Our NOAA Global Workflow deployment demonstrates production viability: 32 tools serving multiple clients with sub-3-second initialization. The combination of vector database (14,854 documents) and graph database (85,894 relationships) provides AI assistants with comprehensive codebase understanding through a unified MCP interface.

As MCP adoption accelerates, gateway patterns will become essential infrastructure for organizations seeking to operationalize AI-assisted development at scale. The open-source Docker MCP Gateway provides a foundation for this transformation.

Acknowledgments

This work was conducted at NOAA's Environmental Modeling Center as part of the Global Workflow modernization initiative. We thank the Docker team for open-sourcing the MCP Gateway and the Anthropic team for the Model Context Protocol specification.

References

1. Anthropic. "Model Context Protocol Specification." https://spec.modelcontextprotocol.io/, 2024.
2. Docker, Inc. "Docker MCP Gateway." https://github.com/docker/mcp-gateway, 2025.
3. NOAA EMC. "Global Workflow Documentation." https://global-workflow.readthedocs.io/, 2025.
4. JSON-RPC Working Group. "JSON-RPC 2.0 Specification." https://www.jsonrpc.org/specification, 2013.

A Tool Inventory

Our EIB-MCP-RAG server exposes 32 tools across six categories:

Category	Tool	Description
Workflow Info	`get_workflow_structure`	System architecture overview
Workflow Info	`get_system_configs`	HPC platform configurations
Workflow Info	`describe_component`	Component file descriptions
Code Analysis	`analyze_code_structure`	File dependency analysis
Code Analysis	`find_dependencies`	Import/export mapping
Code Analysis	`trace_execution_path`	Call chain tracing
Code Analysis	`find_callers_callees`	Function relationship discovery
Semantic Search	`search_documentation`	Hybrid vector+graph search
Semantic Search	`find_related_files`	Similarity-based file discovery
Semantic Search	`explain_with_context`	RAG-enhanced explanations
Semantic Search	`get_knowledge_base_status`	Database health check
EE2 Compliance	`search_ee2_standards`	Standards documentation search
EE2 Compliance	`analyze_ee2_compliance`	Code compliance analysis
EE2 Compliance	`generate_compliance_report`	Automated audit reports
EE2 Compliance	`scan_repository_compliance`	Full repository scanning
Operational	`get_operational_guidance`	HPC procedure guidance
Operational	`list_job_scripts`	Job script inventory
Operational	`explain_workflow_component`	Deep component analysis
SDD Framework	`list_sdd_workflows`	Available workflow listing
SDD Framework	`get_sdd_workflow`	Workflow details
SDD Framework	`execute_sdd_workflow`	Workflow execution
SDD Framework	`execute_sdd_workflow_supervised`	Human-in-loop execution
SDD Framework	`validate_sdd_compliance`	SDD standard validation
SDD Framework	`get_sdd_framework_status`	Framework health check

Table 1: EIB-MCP-RAG Tool Inventory (32 tools)