AUTOCON 4

aviz

AI for Networks
Networks for AI

WS D1: Build Your Own AI Ops Workflows with an Agentic Private AI Platform

Why Do You Need AI for Networks Operations?

What NetOps and SREs do today

Constant Context Switching

- Constantly jump across tools, logs, flows, configs, tickets, controllers, and docs to keep things healthy and secure
- Ad-hoc/ fast (troubleshooting) and repetitive/ boring (audits, documentation)

Address Multiple Stakeholders

- Infra operators, users, architects, security/IT leadership, capacity/procurement
- On demand collaboration accessing the required set of tools and data

What AI Should Enable

Common Language Interface

Embed SRE knowledge

Rationalize your Data

Bottom line: AI is an enabling technology that is rapidly evolving. The AI platform must be tool-agnostic (multi-vendor, multi-source), model-flexible (LLM choice), and deployment-flexible (private/SaaS) so it grows with you—not lock you in.

Network Copilot™ - Modular, Agentic AI Platform

Bring AI to Your Network — On Your Terms

Private AI for NetOps – Runs fully on-prem or in your private cloud

Multi-Vendor/ Multi-Tool – Works across switches, firewalls, SD-WAN, packet brokers, and monitoring tools

Actionable in Plain Language – Ask, "Generate Security Audit Report" and get instant, correlated answers

Agent/Scripts MCP/API/SDK; ships with popular connectors

LLMs model-agnostic (local/open/commercial)

Data Connection Layer Tools, Devices, Files; ships with popular connectors

Open & future-proof (swap connectors/models)

One reasoning plane across multiple infrastructure tools & Knowledgebase

Upskilling your team for AI first NetOps

How Network Copilot™ Works

Administrator

Manage Resource Access

Connect or Activate
* Data sources, Devices, Knowledge Bases
* Custom Agents
* LLMs

Manage User and Roles/ Permission

Onboard Users

Map Resources to Roles

Map Users to Roles

User(s)

Create a Project

Select required data sources
Add project specific context/ files
Choose an LLM
Invite collaborators to participate

Solve the Problem

Start a new session/ chat or continue previous session
Use natural language prompts to activate copilot

What are LLM Agents?

🤖 Definition

An **LLM Agent** is an autonomous AI system that combines Large Language Models with:

- 🧠 **Reasoning**: Understanding complex problems and context
- 💾 **Memory**: Storing context, history, and learned experiences
- 🛠️ **Tools**: Interacting with external systems and APIs
- 📋 **Planning**: Breaking down tasks and executing them

Agent Architecture Components

🧠

Language Model Core

GPT, Claude, Llama

Reasoning & Understanding

🛠️

Tool Interface

APIs, Databases

External Systems

💾

Memory System

Context, History

Working Memory

📋

Planning Engine

Task Breakdown

Execution Strategy

How Agents Think: ReAct Pattern

🔄 What is ReAct?

- **Reasoning + Acting** - Combines thinking with doing
- **LLM Reasoning** decides what action to take next
- **Tools** provide real-world observations
- **Iterative cycles** until problem is solved

The ReAct Cycle

🤔

Thought

Agent reasons about what to do next

⚡

Action

Execute tool or query

👁️

Observation

Receive results and learn

ReAct in Action: Network Troubleshooting

👤 **User** → "Check for network issues"

- 🤔 **Thought** "I need to check device connectivity"
- ⚡ **Action** `query_device_inventory(reachable=false)`
- 👁 **Observation** "Found 3 unreachable devices"
- 🤔 **Thought** "Need interface details for these devices"
- ⚡ **Action** `get_interface_status(device_list)`
- 👁 **Observation** "Interfaces show 'down' status"

💡 **Result** → "Found unreachable devices with down interfaces - investigating further..."

Other Agent Reasoning Patterns

- 🔗 **Chain of Thought** - Step-by-step reasoning without intermediate actions
- 🌳 **Tree of Thoughts** - Explores multiple reasoning paths simultaneously
- 🔄 **Self-Reflection** - Reviews and improves its own reasoning process
- 🎯 **Plan and Execute** - Creates detailed plan first, then executes systematically

Further Reading:

- ReAct: Reasoning and Acting with LLMs
- Chain-of-Thought Prompting
- Tree of Thoughts

Multi-Agent Systems

Collaborative Intelligence Patterns

Single Agent vs Multi-Agent Systems

🤖 Single Agent

- **One agent** handles entire task
- **Sequential reasoning** within single context
- **Limited specialization** - generalist approach
- **Simple coordination** - self-contained

Good for: Simple tasks, single domain problems

🏢 Multi-Agent Systems

- **Multiple agents** collaborate on complex tasks
- **Parallel processing** across specialized agents
- **Deep specialization** - expert agents per domain
- **Advanced coordination** - orchestrated workflows

Good for: Complex workflows, multi-domain problems

"The whole is greater than the sum of its parts"

Core Multi-Agent Patterns

🔗

SequentialAgent

Linear chain of specialized agents processing in sequence

Input → Agent 1 → Agent 2 → Agent 3 → Output

🔀

RouterAgent

Intelligent routing to the most appropriate specialized agent

Input → Analysis → Route → Selected Agent → Output

👥

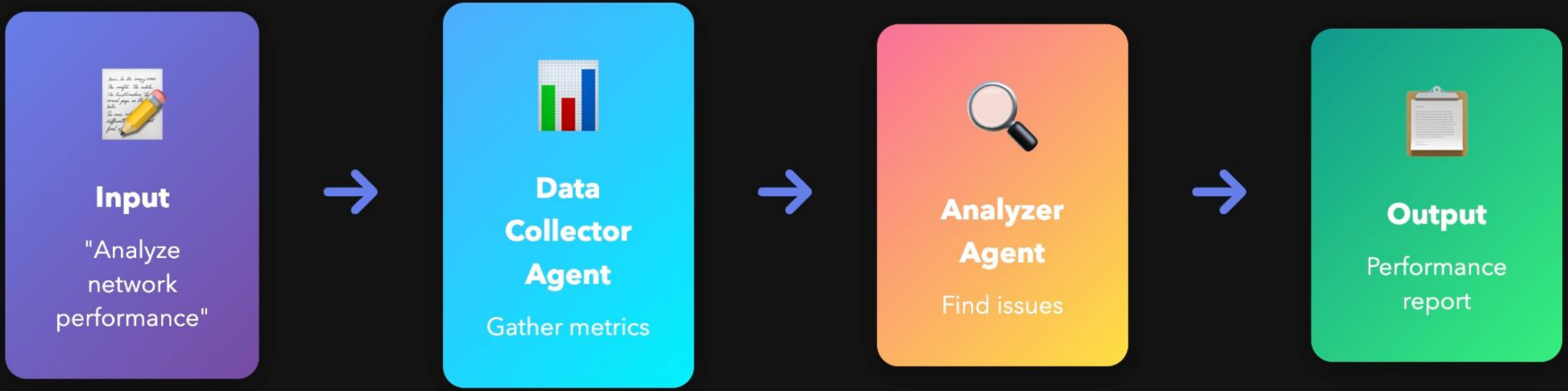
ManagerAgent

Manager coordinates multiple worker agents collaboratively

Manager → Task Distribution → Workers → Synthesis

All patterns can be nested and combined to create complex hybrid architectures

SequentialAgent Visual Flow



📝 **Input**
"Analyze network performance"

→

📊 **Data Collector Agent**
Gather metrics

→

🔍 **Analyzer Agent**
Find issues

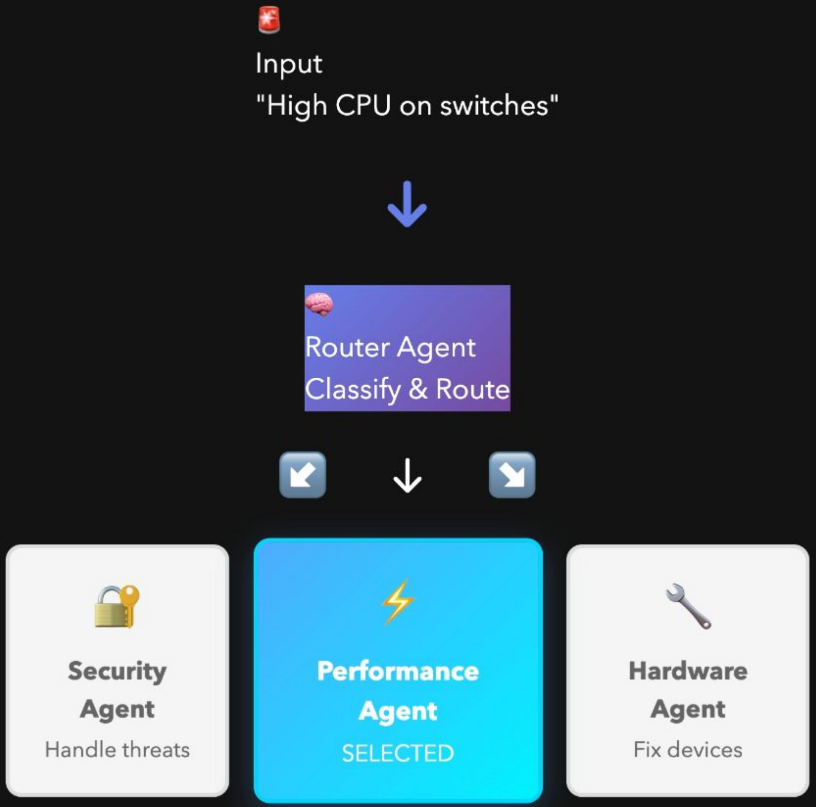
→

📋 **Output**
Performance report

📋 Characteristics

- ✅ Simple to understand and debug
- ✅ Clear dependencies and deterministic flow
- ⚠ Latency accumulates with each step
- ❌ Single point of failure at each stage

RouterAgent Visual Flow



🚨
Input
"High CPU on switches"

↓

🧠
Router Agent
Classify & Route

↙️ ↓ ↘️

🔐
Security Agent
Handle threats

⚡
Performance Agent
SELECTED

🔧
Hardware Agent
Fix devices

📋 Characteristics

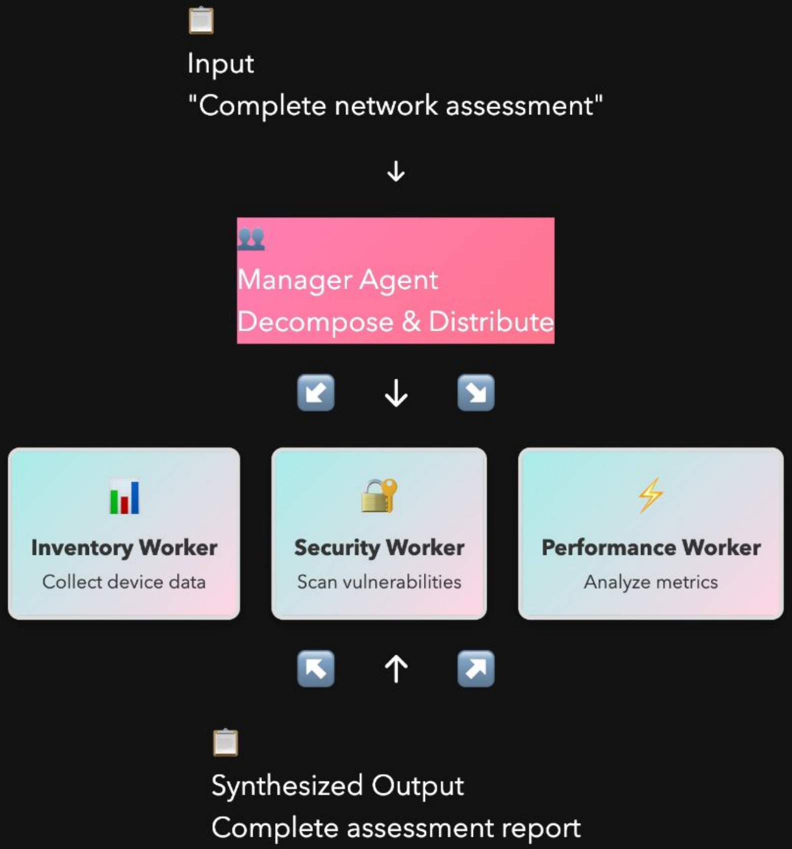
✅ Intelligent routing based on content

✅ Runtime decision making

✅ Specialized handling for scenarios

⚠️ Complex routing logic management

ManagerAgent Visual Flow



📋
Input
"Complete network assessment"

↓

👥
Manager Agent
Decompose & Distribute

↙ ↓ ↘

📊
Inventory Worker
Collect device data

🔐
Security Worker
Scan vulnerabilities

⚡
Performance Worker
Analyze metrics

↖ ↑ ↗

📋
Synthesized Output
Complete assessment report

📋 Characteristics

✅ Central coordinator distributes tasks
✅ Parallel execution for independent tasks
✅ Dynamic worker allocation based on load
✅ Scalable by adding more workers
⚠️ Manager can become a bottleneck

Background Agents

Autonomous Operations

Interactive vs Background Agents

Interactive Agents

- Request-response model
- Human interaction required
- Short-lived execution
- Burst resource usage

Background Agents

- Continuous/scheduled execution
- Fully autonomous
- Long-running lifecycle
- Steady-state resources

Network Use Cases

🔍 System Monitoring Agent

Task: Monitor device health, interface utilization, and service availability
Schedule: Every 5 minutes
Tools: ping_devices, check_interfaces, get_cpu_memory, send_alerts

📊 Performance Reporting Agent

Task: Generate weekly network performance and capacity reports
Schedule: "0 8 * * 1" (Monday 8 AM)
Tools: query_metrics, generate_charts, create_reports, send_email

🧹 Configuration Backup Agent

Task: Backup device configurations and detect changes
Schedule: "0 2 * * *" (Daily 2 AM)
Tools: get_device_configs, compare_configs, store_backup, alert_changes

Network Copilot Platform

Complete Infrastructure for Network AI

Platform Architecture & Capabilities

🔌 Data Integration Layer

- **Network Management:** ONES, Nexus Dashboard, NetBox, Catalyst Center
- **Observability:** Splunk, ELK, Loki, InfluxDB
- **Cloud & Enterprise:** AWS, Azure, ServiceNow, Snowflake
- **15+ Ready Integrations** with unified data model

🤖 AI Infrastructure

- **LLMs Included:** No GPU procurement or deployment needed
- **Vector Store:** Ready for knowledge base and RAG
- **Model Context Protocol:** Standardized AI communication
- **Python SDK:** Simple decorators and auto-documentation

⚡ Agent Runtime

- **Container Infrastructure:** Scalable, secure execution
- **State Management:** Persistent agent memory
- **Auto-scaling:** Handles any workload automatically
- **Multi-agent Coordination:** Workflow orchestration

👥 Enterprise Features

- **Multi-tenancy:** Project isolation & access control
- **Web Interface:** Chat UI, dashboards, monitoring
- **API Gateway:** RESTful APIs for integration
- **Production Ready:** Security & monitoring

NCP SDK

Development Kit for Custom Agents

NCP SDK Overview

🎯 What is NCP SDK?

- **Development Kit** for creating custom agents
- **Type-safe Python** with full IDE support
- **Local Development**, remote execution model
- **Tool ecosystem** for network operations

💡 Core Philosophy

- **Develop Locally** → Full IDE experience
- **Deploy Remotely** → Secure platform execution
- **Type Everything** → Catch errors early
- **Tool-First** → Composable functionality

```
# Get started in seconds
pip install ncp
ncp init my-agent-project
```

Creating Custom Tools

🛠️ Tool Features

- 🏷️ **@tool decorator** - Simple annotation
- 🔍 **Type hints** - Auto schema generation
- 📖 **Docstrings** - Built-in documentation
- ✅ **Validation** - Input/output checking

📝 Network Query Tool

```python
from ncp import tool

@tool
def get_device_status(hostname: str) -> dict:
    """Check device reachability.

    Args:
        hostname: Device hostname or IP

    Returns:
        Device status information
    """
    return {
        "hostname": hostname,
        "reachable": True,
        "uptime": "5 days, 2 hours"
    }
```

Creating Agents

🤖 Agent Components

- 🏷️ **Name & Description** - Agent identity
- 📋 **Instructions** - Behavior guidelines
- ⚒️ **Tools** - Available capabilities

📝 Network Monitoring Agent

```python
from ncp import Agent, tool

@tool
def check_device_health(hostname: str) -> dict:
    """Check device health metrics."""
    return {"cpu": 45, "memory": 60, "status": "ok"}

@tool
def get_interface_stats(hostname: str) -> dict:
    """Get interface utilization."""
    return {"eth0": {"util": 30, "errors": 0}}

# Create the agent
monitor_agent = Agent(
    name="NetworkMonitor",
    description="Monitors network health",
    instructions="""You are a network monitoring
    agent. Check device health and interface
    statistics when requested.""",
    tools=[check_device_health, get_interface_stats]
)
```

Functions vs LLM Agents

🔧 Just Functions

- **Fixed workflows** - Predefined call sequences
- **Manual orchestration** - Developer decides when/how
- **No context awareness** - Each call is independent
- **Limited reasoning** - Execute exactly as programmed

❌ User must know which function to call and when

🤖 LLM Agent with Tools

- **Dynamic reasoning** - Decides which tools to use
- **Natural language** - User describes what they want
- **Context aware** - Remembers conversation history
- **Adaptive planning** - Adjusts based on results

✅ Agent understands intent and orchestrates tools

🎯 The Power of LLM Agents

"Check network health" → Agent decides: query devices → analyze results → check interfaces → provide summary

Deployment & Packaging

📦 Project Structure

```
my-agent-project/
├── requirements.txt
├── agents/
│   ├── __init__.py
│   └── network_monitor.py
├── tools/
│   ├── __init__.py
│   └── device_tools.py
│
```

🚀 Deployment Commands

- **Package**: `ncp package . --output my-agent.ncp`
- **Validate**: `ncp validate my-agent.ncp`
- **Deploy**: `ncp deploy my-agent.ncp`

⚙️ What Happens on Deploy

- 📤 **Upload** - Agent code sent to NCP platform
- 🔍 **Validation** - Type checking and dependency resolution
- 🏃 **Runtime** - Agent ready for execution
- 🌐 **Access** - Interactive playground available

✅ Agent execute securely on NCP platform

MCP Integration

🔗 What is MCP?

- **Model Context Protocol** - Standard for AI tool integration
- **Created by Anthropic** - Open protocol for LLM tools
- **External Servers** - Tools run on separate processes
- **Agent Configuration** - Declare servers in agent setup

🌐

Platform handles MCP connections automatically

📝 Configuring MCP Servers

```
from ncp import Agent
from ncp.mcp import MCPServerConfig

# Configure NetBox MCP server
netbox_server = MCPServerConfig(
    source="https://netbox.company.com/mcp",
    auth_token="netbox-api-token"
)

# Network agent with native + MCP tools
network_agent = Agent(
    name="NetworkOperator",
    description="Network inventory management",
    tools=[ping_tool, check_device_status],  # Native tool
    mcp_servers=[netbox_server]              # NetBox MCP
)
```