



THE LINUX FOUNDATION



NORTH AMERICA

# Beyond Vector Search

Building Knowledge Graphs for  
Autonomous Infrastructure



# AGENTIC INFRASTRUCTURE

While Infrastructure as Code (IaC) got us to the cloud faster; Agentic Infrastructure Operations ensures we can actually run a complex footprints without drowning in operational complexity. It's about automating Day-2 tasks, and helps to reallocate up to 60% of engineering time from routine maintenance back to core business innovation.



# What does an AI agent need to operate IT infrastructure reliably?

## Context

Understand what a configuration item is and what it means

## Relationships

Know what depends on what — blast radius, impact chains

## Governance

Enforce policies, compliance rules, and change controls



# APPROACH 1 — VECTOR SEARCH

Infrastructure docs, runbooks, and config files are embedded as vectors. Queries retrieve semantically similar documents.

## ✓ Strengths

- Fast semantic search across large collection of texts
- Easy to implement with off-the-shelf embeddings
- Good for free-text runbook lookup
- Scales well for document retrieval

## ✗ Limitations

- No relationship graph — can't trace dependencies
- Cannot answer 'what breaks if X fails?'
- Blind to topology and blast radius
- Config drift invisible — embeddings go stale

*Verdict: Great for knowledge retrieval. Insufficient for autonomous decision-making.*

# Context Aggregation

## Unlocking Aggregation Reasoning

On complex cross-document queries requiring structural grouping and counting (e.g., "Which departments have outdated compliance documents?"), standard semantic search drops under 20% accuracy because it cannot synthesize unstructured text. GraphRAG utilizes pre-computed community summaries to deliver highly accurate context windows.

# <70%

MULTI-DOCUMENT REASONING ACCURACY

# APPROACH 2 — GraphRAG

Infrastructure entities and their relationships are modeled as a knowledge graph. AI traverses edges to reason across dependencies.

## ✓ Strengths

- Full dependency traversal — blast radius in seconds
- Context-rich: apps, infra, teams, policies in one graph
- Powers autonomous impact analysis and change planning
- Compliance posture queryable at any time

## ✗ Limitations

- Complex to build & maintain the graph schema
- Requires continuous graph population from live infrastructure
- High operational burden: GraphDB + ETL pipelines
- Keeping graph current with infrastructure drift is hard

*Verdict: Architecturally superior — but the graph is hard to build, populate, and keep accurate.*

# Cost Analysis

Vector RAG (Ingestion)

10%

GraphRAG (Ingestion)

95%

Vector RAG (Query)

5%

LazyGraphRAG (Query)

12%

*GraphRAG incurs high up-front entity extraction costs, but search patterns like Microsoft's LazyGraphRAG reduced active runtime token inferencing costs by over 700x.*

Source: <https://www.microsoft.com/en-us/research/blog/lazygraphrag-setting-a-new-standard-for-quality-and-cost/>



# APPROACH 3 — Resource Graph

Rescile's Unified Configuration Server (UCS) constructs a entity-relationship diagram based on configuration files and enables multi-hop logic traversals that explicitly trace inherited policies, ownership structures, and data flows.

## Precision

Deterministic (Follows hard code/graph links; mathematically precise topology).

## State Awareness

Real-time (The graph maps the *structure*, REST fetches the *live state* on-demand).

## Hallucination Risk

Extremely Low (The agent is restricted to verified graph structures and API payloads).

## Multi-Hop Reasoning

Native (The graph is built specifically to traverse deep dependency chains).

# APPROACH 3 — Resource Graph

Rescile's UCS transforms static configuration files into a dynamic knowledge graph that maps the entire infrastructure as an interconnected entity-relationship network, instantly tracing inherited policies, ownership structures, and data flows.

01

Ingestion

Declarative TOML/JSON/CSV blueprints · No LLM tokens required · State Sync · Git-Versioned

02

Storage &  
Indexing

In-memory JSON graph · Transient properties · GraphML export · GraphQL + MCP server

03

Query Flow

Typed MCP tool calls · Full dependency traversal · Always-live data · Blast-radius in ms

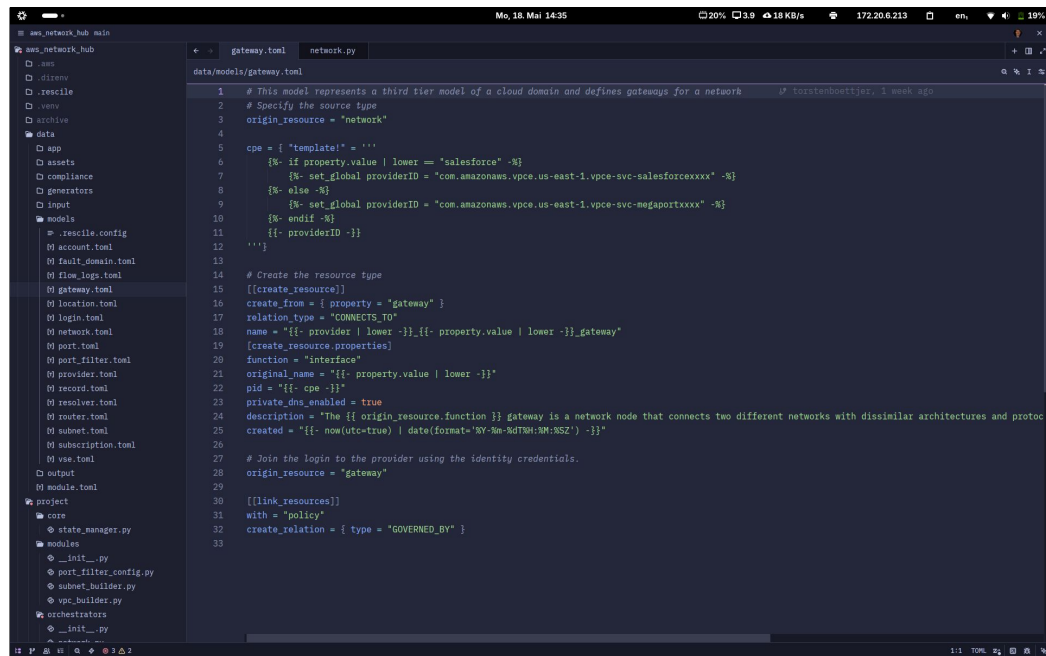
04

Generation &  
Governance

Native IaC generation · Separated compliance layer · OSCAL auto-export · Deterministic plans

# Ingestion

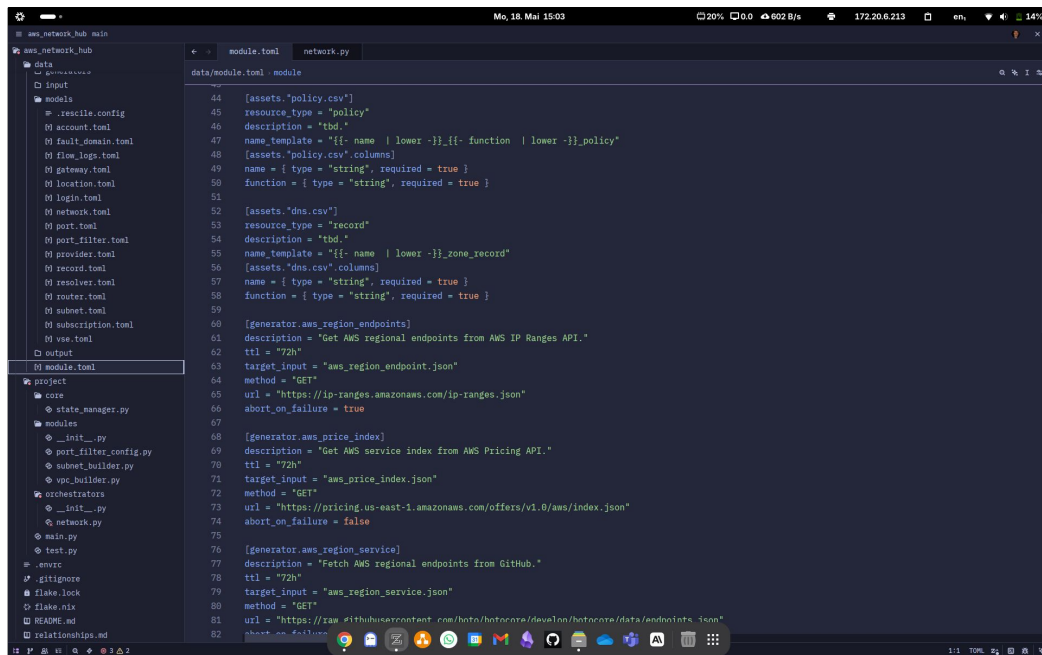
The framework ingests flat files. Instead of calculating cosine similarities, it parses the explicit declarations (e.g., a CSV mapping networks, a TOML file defining container specs). It instantiates them as nodes within a lean, in-memory JSON object graph.



```
1 # This model represents a third tier model of a cloud domain and defines gateways for a network
2 # Specify the source type
3 origin_resource = "network"
4
5 cpe = [ "template" = '''
6
7 [%- if property.value | lower = "salesforce" -%]
8 [%- set_global providerID = "com.amazonaws.vpc.us-east-1.vpc-svc-salesforcexxxx" -%]
9 [%- else -%]
10 [%- set_global providerID = "com.amazonaws.vpc.us-east-1.vpc-svc-megaportxxxx" -%]
11 [%- endif -%]
12 {{- providerID -}}
13 '''
14
15 # Create the resource type
16 [[create_resource]]
17 create_from = { property = "Gateway" }
18 relation_type = "CONNECTS_TO"
19 name = "{{- provider | lower -}}_{{- property.value | lower -}}_gateway"
20 [create_resource.properties]
21 function = "interface"
22 original_name = "{{- property.value | lower -}}"
23 pid = "{{- cpe -}}"
24 private_dns_enabled = true
25 description = "The {{ origin_resource.function }} gateway is a network node that connects two different networks with dissimilar architectures and protocols."
26 created = "{{- now(utc=true) | date(format='%Y-%m-%dTHH:MM:SSZ') -}}"
27
28 # Join the login to the provider using the identity credentials.
29 origin_resource = "Gateway"
30
31 [[link_resources]]
32 with = "policy"
33 create_relation = { type = "GOVERNED_BY" }
```

# Ingestion

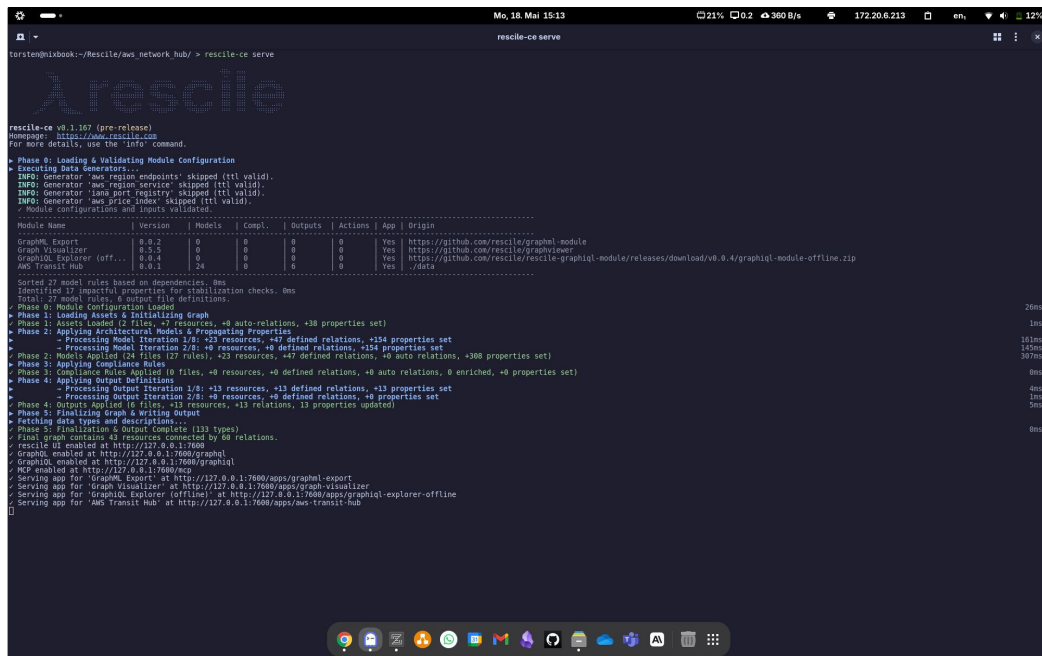
The UCS bridges the gap between static engineering files and live enterprise realities providing a true operational context. This is achieved, with the server acting as a universal ingestion layer. It collects and harmonizes resource information from across the organization's.



```
44 [assets."policy.csv"]
45 resource_type = "policy"
46 description = "tbd."
47 name_template = "{{- name | lower -}}_{{- function | lower -}}_policy"
48 [assets."policy.csv".columns]
49 name = { type = "string", required = true }
50 function = { type = "string", required = true }
51
52 [assets."dns.csv"]
53 resource_type = "record"
54 description = "tbd."
55 name_template = "{{- name | lower -}}_zone_record"
56 [assets."dns.csv".columns]
57 name = { type = "string", required = true }
58 function = { type = "string", required = true }
59
60 [generator.aws_region_endpoints]
61 description = "Get AWS regional endpoints from AWS IP Ranges API."
62 ttl = "72h"
63 target_input = "aws_region_endpoint.json"
64 method = "GET"
65 url = "https://ip-ranges.amazonaws.com/ip-ranges.json"
66 abort_on_failure = true
67
68 [generator.aws_price_index]
69 description = "Get AWS service index from AWS Pricing API."
70 ttl = "72h"
71 target_input = "aws_price_index.json"
72 method = "GET"
73 url = "https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/index.json"
74 abort_on_failure = false
75
76 [generator.aws_region_service]
77 description = "Fetch AWS regional endpoints from GitHub."
78 ttl = "72h"
79 target_input = "aws_region_service.json"
80 method = "GET"
81 url = "https://raw.githubusercontent.com/boto/botocore/dev/botocore/data/andnoints.json"
82 abort_on_failure = true
```

# Storage & Indexing

An in-memory engine resolves the templates and blueprints, linking the resources deterministically. If a TOML specifies that App-Server-1 requires Subnet-A, the engine draws a hard dependency edge in the knowledge graph as an in-memory, structural representation of the target infrastructure environment.



```
rescile-ce serve

rescile-ce v0.1.107 (pre-release)
Homepage: https://aws.amazon.com/rescile/
For more details, use the 'info' command.

Phase 0: Loading & Validating Module Configuration
Executing Data Generators...
INFO: Generator 'aws_region_endpoints' skipped (ttl valid).
INFO: Generator 'aws_region_service' skipped (ttl valid).
INFO: Generator 'aws_port_registry' skipped (ttl valid).
INFO: Generator 'aws_price_index' skipped (ttl valid).
Module configurations and inputs validated.

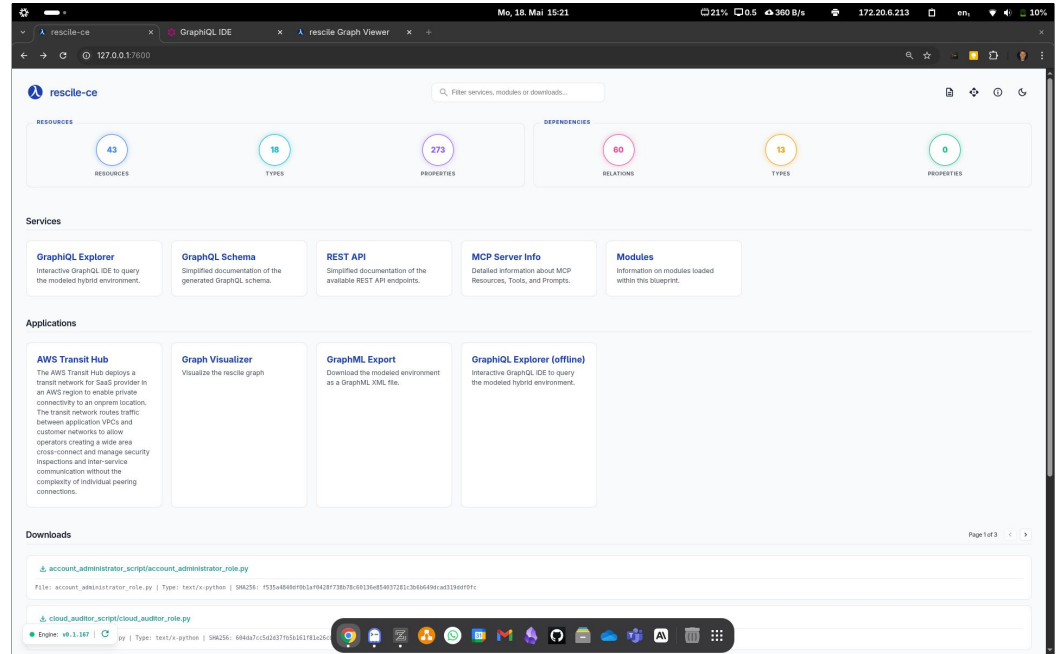
Module Name | Version | Models | Compl. | Outputs | Actions | App | Origin
-----
Graph Export | 0.0.2 | 0 | 0 | 0 | 0 | Yes | https://github.com/rescile/graph-export
Graph Visualizer | 0.5.5 | 0 | 0 | 0 | 0 | Yes | https://github.com/rescile/graphviewer
GraphQL Explorer (off) | 0.0.4 | 0 | 0 | 0 | 0 | Yes | https://github.com/rescile/rescile-graphql-module/releases/download/v0.0.4/graphql-module-offline.zip
AWS Transit Hub | 0.0.1 | 24 | 0 | 0 | 0 | Yes | ./data

Sorted 27 model rules based on dependencies: 8ms
Identified 17 impactful properties for stabilization checks: 8ms
Total 27 model rules, 6 output file definitions.

Phase 0: Module Configuration loaded
Phase 1: Loading Assets & Initializing Graph
Phase 2: Applying Architectural Models & Propagating Properties
  -> Processing Model Iteration 1/8: +23 resources, +47 defined relations, +154 properties set
  -> Processing Model Iteration 2/8: +8 resources, +8 defined relations, +154 properties set
Phase 3: Models Applied (24 files, 107 rules), +23 resources, +47 defined relations, +8 auto relations, +388 properties set
Phase 3: Applying Compliance Rules
Phase 3: Compliance Rules Applied (0 files, +8 resources, +8 defined relations, +8 auto relations, 0 enriched, +8 properties set)
Phase 4: Applying Output Definitions
  -> Processing Output Iteration 1/3: +13 resources, +13 defined relations, +13 properties set
  -> Processing Output Iteration 2/3: +8 resources, +8 defined relations, +8 properties set
Phase 4: Output Applied (13 files, +13 resources, +13 relations, 13 properties updated)
Phase 5: Finalizing Graph & Writing Output
Retrieving data types and dependencies...
Phase 5: Finalization & Output Complete (133 types)
Final graph contains 43 resources connected by 68 relations.
rescile cli enabled at http://127.0.0.1:17600/
rescile cli enabled at http://127.0.0.1:17600/graphql
MCP enabled at http://127.0.0.1:17600/mcp/
Serving app for 'Graph Export' at http://127.0.0.1:17600/apps/graph-export
Serving app for 'Graph Visualizer' at http://127.0.0.1:17600/apps/graph-visualizer
Serving app for 'GraphQL Explorer (offline)' at http://127.0.0.1:17600/apps/graphql-explorer-offline
Serving app for 'AWS Transit Hub' at http://127.0.0.1:17600/apps/aws-transit-hub
```

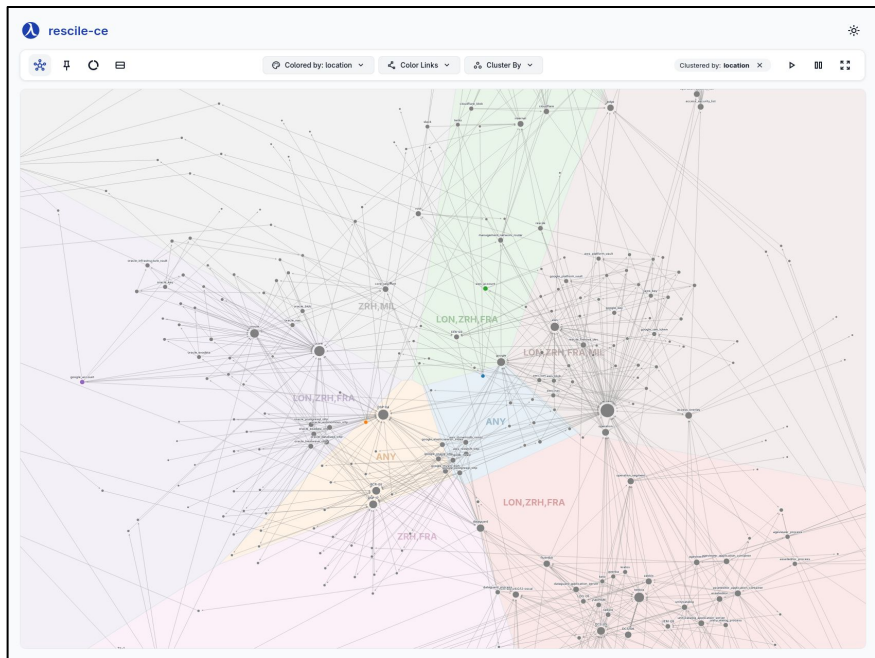
# Storage & Indexing

The core engine enables a highly modular ecosystem. Infrastructure operators can dynamically extend the system's core capabilities by deploying plug-and-play applications. This allows organizations to build out customized, self-service portals and backend engines for specific infrastructure domains



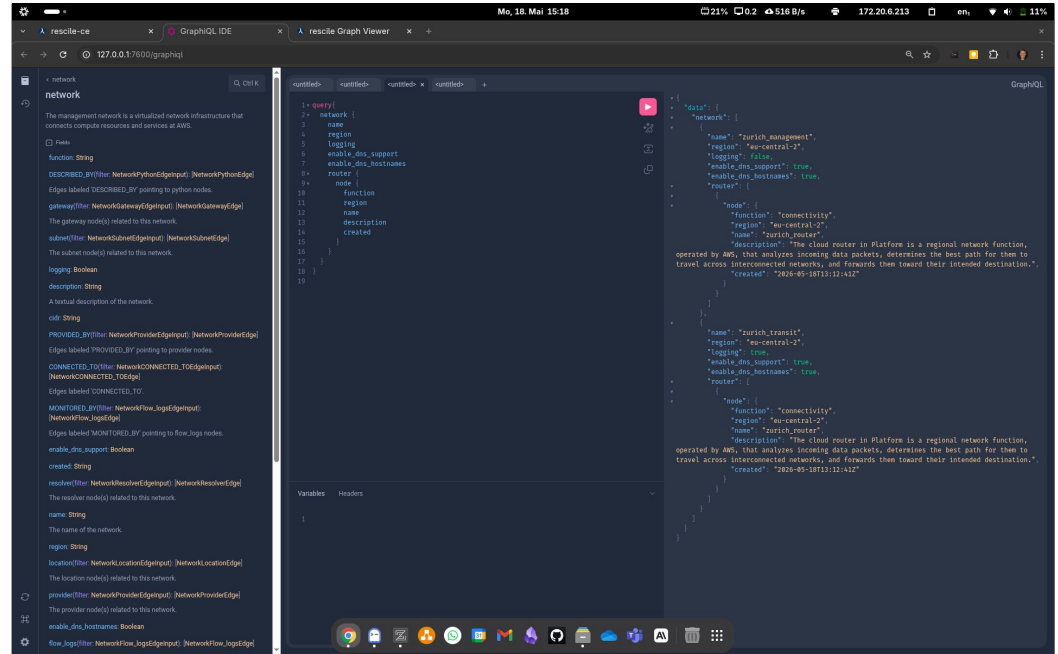
# Query Flow

The design eliminates the friction of adopting advanced infrastructure management by anchoring the entire platform in open-source standards. For IT managers, this ensures your team can leverage existing skillsets and tools without being locked into a proprietary ecosystem.



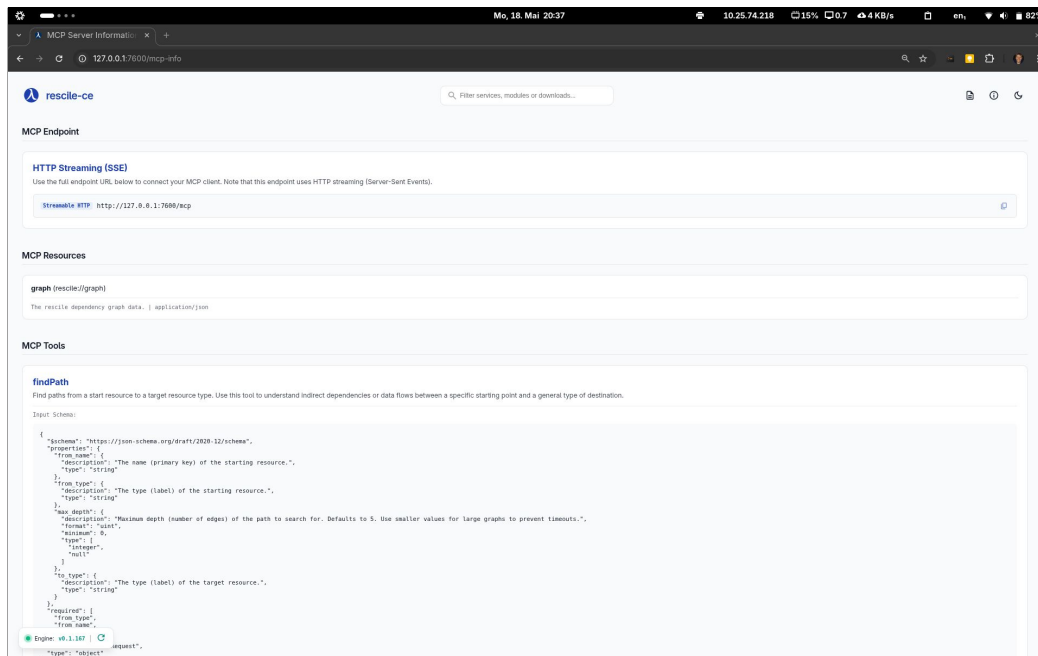
# Query Flow

Rescile diverges from traditional vector-based RAG by querying a static vector store. The framework traverses the in-memory graph to identify the relevant nodes, triggers REST API requests directly to the underlying systems or cloud providers, and hydrates the graph nodes with live status metrics, state flags, and real-world configurations.



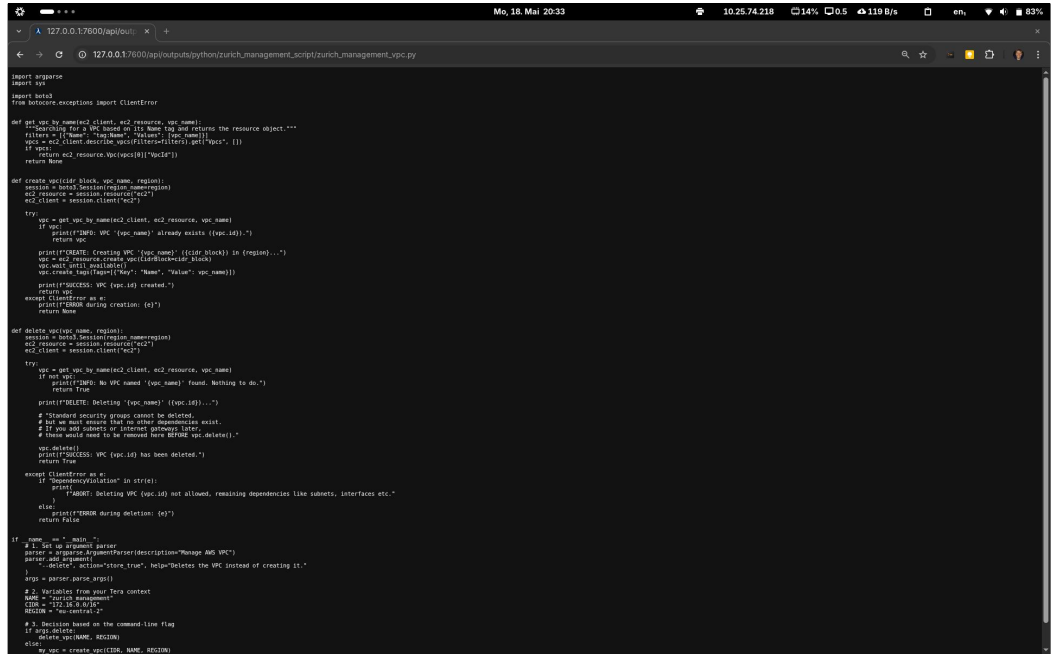
# Query Flow

To enable agentic operation without building proprietary, brittle integrations, Rescile exposes the graph through the Model Context Protocol (MCP). As an open, industry-standard interface designed specifically for AI models, MCP acts as a universal data fabric. It allows both internal native agents and external LLM ecosystems to seamlessly consume and manipulate infrastructure state.



# Generation & Governance

An accurate graph is only as good as its real-world implementation. To ensure 100% reliable execution, Rescile's engine takes the relationships, dependencies, and policy guardrails validated in-memory and compiles them into concrete, deployable artifacts.



```
import argparse
import sys

import boto3
from botocore.exceptions import ClientError

def get_vpc_by_name(ec2_client, ec2_resource, vpc_name):
    """Returns the VPC object, or None if not found and returns the resource object...
    filters = [{"Name": "tag:Name", "Values": [vpc_name]}]
    vpc = ec2_client.describe_vpcs(Filters=[{"Name": "tag:Name", "Values": [vpc_name]}])
    if vpc:
        return ec2_resource.Vpc(vpc["VpcId"])
    return None

def create_vpc(cidr_block, vpc_name, region):
    ec2_resource = boto3.Session(profile_name='rescile').resource('ec2')
    ec2_client = session.client('ec2')

    try:
        vpc = get_vpc_by_name(ec2_client, ec2_resource, vpc_name)
        if vpc:
            print(f"INFO: VPC '{vpc_name}' already exists (vpc-id: '{vpc.VpcId}')")
            return vpc

        print(f"CREATE: Creating VPC '{vpc_name}' (cidr-block: '{cidr_block}') in (region: '{region}')")
        vpc = ec2_resource.create_vpc(CidrBlock=cidr_block)
        vpc.wait_until_available()
        vpc.create_tags(Tag=[{"Key": "Name", "Value": vpc_name}])

        print(f"SUCCESS: VPC (vpc-id) created.")
    except ClientError as e:
        print(f"ERROR during creation: (e)")
        return None

def delete_vpc(vpc_name, region):
    ec2_resource = boto3.Session(profile_name='rescile').resource('ec2')
    ec2_client = session.client('ec2')

    try:
        vpc = get_vpc_by_name(ec2_client, ec2_resource, vpc_name)
        if not vpc:
            print(f"INFO: No VPC named '{vpc_name}' found. Nothing to do.")
            return True

        print(f"DELETE: Deleting '{vpc_name}' (vpc-id: '{vpc.VpcId}')")
        # Standard security group cannot be deleted
        # but we must ensure that no other dependencies exist.
        # If no subnets or interfaces are found,
        # these would need to be removed here BEFORE vpc.delete().
        vpc.delete()
        print(f"SUCCESS: VPC (vpc-id) has been deleted.")
        return True
    except ClientError as e:
        if "DependencyViolation" in str(e):
            print(f"ERROR: Deleting VPC (vpc-id) not allowed, remaining dependencies like subnets, interfaces etc.")
        else:
            print(f"ERROR during deletion: (e)")
        return False

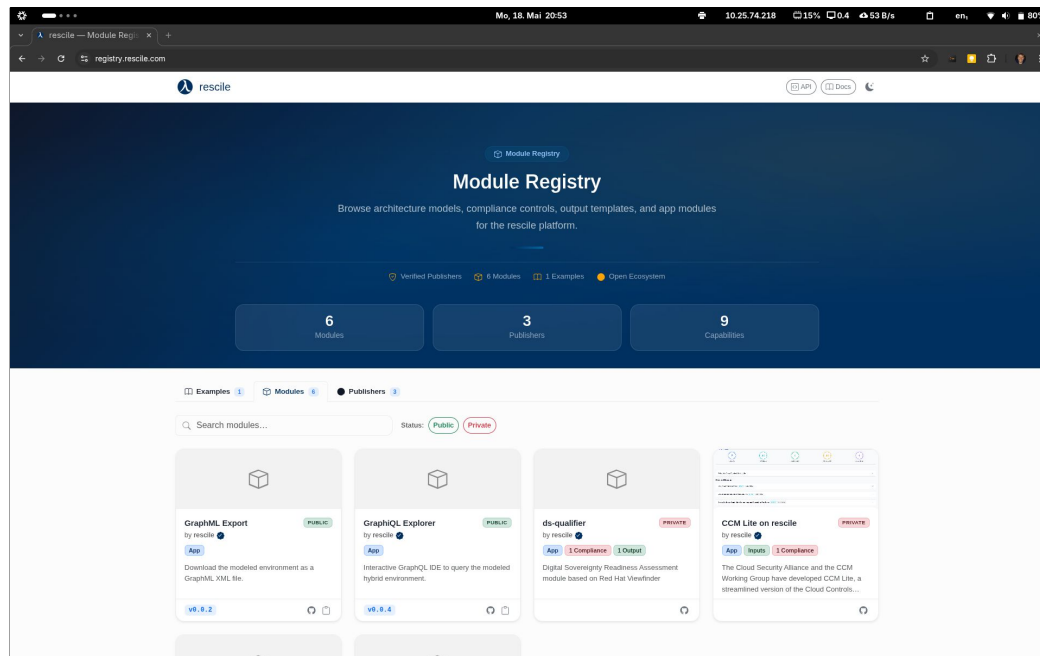
if __name__ == "__main__":
    # 1. Set up argparse parser
    parser = argparse.ArgumentParser(description="Manage AWS VPC")
    parser.add_argument(
        "--create",
        action="store_true",
        help="Creates the VPC instead of creating it."
    )
    args = parser.parse_args()

    # 2. Decision from argparse context
    NAME = "rescile-internal"
    CIDR = "10.0.0.0/24"
    REGION = "us-central-2"

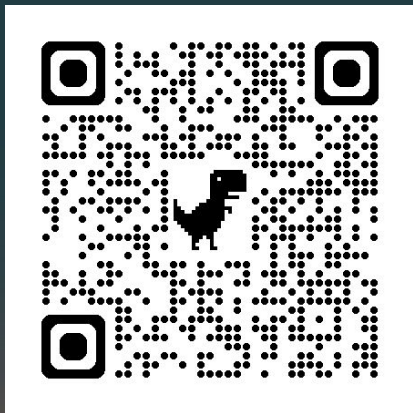
    # 3. Decision based on the command-line flag
    if args.create:
        create_vpc(NAME, CIDR, REGION)
    else:
        delete_vpc(NAME, REGION)
```

# Generation & Governance

To accelerate time-to-value and eliminate redundant engineering effort, Rescile features a centralized Blueprint Registry. This registry acts as a secure, shared repository where operators and community members can publish, discover, and distribute pre-built blueprint modules and architectural templates.



# Thank You



Download

<https://www.rescile.com>

