ANK Workbench User Manual

Web-Based Network Orchestration Platform

Simon Knight

Adelaide, Australia

March 2026

> **About this manual.** This manual covers the end-to-end web-based workflow for ANK Workbench v4.2: designing and validating network topologies in the YAML editor, running protocol simulations, deploying to ContainerLab, detecting dissonance between simulated and emulated routing state, and remediating findings on live Arista EOS devices. It also covers chaos engineering, the REST API, configuration reference, and integration with the broader ANK ecosystem. Manual ID: ANK-WB-UM-001.

Contents

1 Introduction

ANK Workbench is a web-based orchestration platform that connects the full network design lifecycle into a single browser-based interface. Starting from a declarative YAML topology, an engineer can validate, simulate, deploy to ContainerLab, compare simulation predictions against emulated reality, and push remediation commands to live devices — without switching between CLIs or writing bespoke glue scripts.

1.1 Who This Manual Is For

This manual is written for **network engineers** who are comfortable with:

- YAML syntax and basic network topology concepts (routers, links, BGP, OSPF)
- Docker and basic container concepts
- The ContainerLab tool for container-based network emulation

Familiarity with Python or FastAPI is not required for day-to-day use of the platform, but is useful for API integration work described in §6.

1.2 Document Conventions

- `monospace` text denotes commands, file paths, environment variables, and API endpoints.
- Dark shaded boxes show commands to run in a terminal or browser.
- Blue tip boxes highlight best practices and shortcuts.
- Amber caution boxes mark operations that can cause data loss or service disruption.

1.3 Platform Overview

The platform comprises a Python/FastAPI backend and a React frontend. It coordinates four external tools:

netsim Protocol simulator, invoked as a subprocess to produce routing tables and convergence traces.

netflowsim Monte Carlo traffic simulator, invoked alongside netsim to produce link-utilisation estimates.

ContainerLab Container-based network emulator managed via the `clab` CLI.

Docker Container runtime, queried via the Docker SDK for container introspection and exec.

All workflow state is persisted in a project model (`netauto.project`) that can be shared across machines.

2 Quick Start

Get from zero to a validated, simulated topology in under ten minutes.

Step 1. Start the ANK Workbench server:

```
ank-workbench serve --port 8000
```

Step 2. Open the web UI in your browser:

```
http://localhost:8000
```

The dashboard shows your existing projects. On a fresh install you will see an empty project list.

Step 3. Create a new project by clicking **New Project**, entering a name, and clicking **Create**. The editor opens automatically.

Step 4. In the **Editor** tab, paste or type a minimal topology. A two-router OSPF example:

```
topology:
  name: quickstart
  protocol: ospf
  nodes:
    - id: r1
      name: router-1
      device_type: router
      platform: arista_eos
    - id: r2
      name: router-2
      device_type: router
      platform: arista_eos
  links:
    - src: r1
      dst: r2
      bandwidth_mbps: 1000
```

Step 5. Click **Validate**. The validation panel opens and shows results for all 27 rules. Green ticks indicate no issues.

Step 6. Click **Simulate**. ANK Workbench invokes netsim and streams progress via the WebSocket. When complete, switch to the **Visualize** tab to see the topology with routing overlays.

Step 7. To deploy to ContainerLab, ensure Docker is running, then click **Deploy** in the **Workflow** panel. Container readiness is detected automatically — the UI shows per-container status badges.

Tip

The keyboard shortcut **Ctrl+Shift+V** re-runs validation without leaving the editor. Validation results are shown as inline annotations on the canvas.

3 Installation & Prerequisites

3.1 System Requirements

▷ **Prerequisites**

- **Python 3.12** or later
- **Node.js 20+** and **npm 10+** (for local frontend build; not required if using the pre-built distribution)
- **Docker 24+** with the Docker daemon running
- **ContainerLab 0.54+** installed and on `PATH`
- **netsim** installed and on `PATH`
- **netflowsim** installed and on `PATH`
- **4 GB RAM** minimum; 8 GB recommended for topologies with 20+ nodes

3.2 Installing ANK Workbench

Install the package from PyPI:

```
pip install ank-workbench
```

To include optional dependencies for multi-host ContainerLab deployments:

```
pip install "ank-workbench[multihost]"
```

> **Tip**
>
> Use `uv pip install ank-workbench` for significantly faster dependency resolution in large virtual environments. ANK Workbench is fully compatible with `uv`.

3.3 Verifying the Installation

```
ank-workbench --version
netsim --version
netflowsim --version
clab version
```

Each command should print a version string. If `netsim` or `netflowsim` are not found, add their installation directory to your `PATH`:

```
export PATH="$HOME/.local/bin:$PATH"
```

3.4 API Key Configuration

ANK Workbench secures all REST endpoints with an API key.

> **API Key Setup**
>
> Set the `ANK_WORKBENCH_API_KEY` environment variable before starting the server. All API requests must include the `X-API-Key: <your-key>` header. If the variable is unset, the server starts in *development mode* with no authentication — never use this in production.

```
export ANK_WORKBENCH_API_KEY="your-secret-key"
ank-workbench serve
```

3.5 Docker Daemon

> **Caution**
>
> The Docker daemon must be running before starting any ContainerLab deployment workflow. ANK Workbench checks Docker availability at startup and will display a warning banner in the UI if the daemon is unreachable. Deployment and dissonance operations will fail silently if Docker goes away mid-session.

On macOS and Windows, start Docker Desktop before launching ANK Workbench. On Linux:

```
sudo systemctl start docker
```

4 Core Workflows

ANK Workbench is organised around six primary workflows. Each is accessible from the navigation bar at the top of the web UI.

4.1 Design & Validate

The **Editor** tab provides a YAML editor with real-time syntax highlighting and an embedded topology canvas that updates as you type.

Step 1. Open or create a project. The editor shows the current `topology.yaml` content.

Step 2. Edit the topology YAML. The canvas on the right updates within 500 ms of each keystroke using the debounced parser.

Step 3. Click **Validate** or press **Ctrl+Shift+V**. The validation panel lists results from all 27 rules across four rule sets: graph topology, design guardrails, IP addressing, and protocol consistency.

Validation runs automatically on save (Ctrl+S).

Step 4. Review findings. Errors block simulation and deployment. Warnings are shown as yellow canvas annotations and do not block progress (unless strict mode is enabled — see §5.3).

Step 5. Fix any errors, then re-validate. The panel updates incrementally; resolved issues are removed from the list.

Step 6. When the panel shows **All checks passed**, proceed to simulation.

4.1.1 Validation Rule Sets

The 27 rules are grouped into four sets:

- **Graph rules** — routing cycles, disconnected components, unreachable subnets, isolated nodes.
- **Guardrail rules** — single points of failure (articulation points), critical links (bridges).
- **IP rules** — subnet overlap, duplicate addresses, address family consistency.
- **Protocol rules** — BGP requires AS numbers, protocol–topology consistency checks.

Findings carry node and interface references so the canvas can annotate the exact location of each issue.

> **Tip**
>
> Enable **strict mode** in the project settings (§5.3) to promote warnings to errors. This is recommended before deploying to production ContainerLab environments.

4.2 Protocol Simulation

Simulation runs netsim and netflowsim as background subprocesses, streams progress events over WebSocket, and stores the resulting artifact for visualisation.

Step 1. Ensure validation has passed (the **Simulate** button is disabled until validation succeeds).

Step 2. Click **Simulate** in the **Workflow** panel. The panel shows a live event log as netsim runs.

Step 3. Monitor the **Convergence** sub-panel. It displays per-node protocol state (adjacency formation, route installation) as events arrive via the WebSocket bus.

Step 4. When simulation completes, the **Visualize** tab becomes active. Switch to it to explore results.

Step 5. In the **Visualize** tab:

- Select overlays from the **Overlay** menu: link utilisation heat map, per-node routing tables, protocol state badges.
- Use the **Time Machine** scrubber to replay protocol convergence events at microsecond granularity.

Time Machine

Drag the Time Machine scrubber left to see which routes were installed first during convergence. The canvas animates node and link colours as you move through the event trace. A **Coarse** badge appears if the simulator did not emit native microsecond traces — the system synthesises a trace from protocol-level events in this case.

Caution

If the topology YAML is edited after simulation runs, all visualisation overlays dim to 55% opacity to indicate stale results. Re-run simulation to restore full opacity. This staleness is tracked by SHA-256 hash of the topology content, not by file timestamp.

4.3 ContainerLab Deployment

Deployment requires Docker running and ContainerLab on PATH.

ANK Workbench generates ContainerLab topology YAML from the ANK topology model, deploys it via the `clab` CLI, and monitors per-container readiness.

Step 1. Verify Docker and ContainerLab are available using the status badges at the top right of the **Workflow** panel.

Step 2. Click **Deploy** in the **Workflow** panel. The backend generates per-host `.clab.yml` files and invokes ContainerLab.

Step 3. Monitor the deployment log. Each container reports its readiness state: **Starting**, **Ready**, or **Timeout**.

Step 4. When all containers show **Ready**, the emulated topology is available. Terminal access to each container is available via the **Terminal** overlay in the Visualize tab (hover a node, then click the terminal icon).

Step 5. To tear down the deployment, click **Destroy**. The backend destroys all containers and verifies cleanup via the Docker API.

4.3.1 Multi-Host Deployment

For topologies larger than a single Docker host can support, assign nodes to execution hosts in the topology YAML:

```
nodes:
  - id: spine-1
    name: spine-1
    execution_host: host-a.example.com
  - id: leaf-1
    name: leaf-1
    execution_host: host-b.example.com
```

ANK Workbench generates one `.clab.yml` per host and renders cross-host links as VxLAN tunnels. VNI assignments are deterministic (`10000 + link_index`) to ensure reproducibility.

Tip

Container boot times vary by NOS image: Linux hosts typically boot in under 10 seconds, Arista cEOS takes 2–3 minutes, Nokia SR Linux up to 5 minutes. The readiness detector has a 10-minute timeout. Do not navigate away from the Workflow panel during deployment.

4.4 Dissonance Detection

Dissonance detection compares the routing tables predicted by the simulator against the Forwarding Information Base (FIB) observed in running containers. It answers the question: *where does simulation disagree with emulated reality?*

Step 1. Ensure both simulation and ContainerLab deployment have completed. Both workflow steps must show green checkmarks.

Step 2. Click **Run Dissonance** in the **Dissonance** tab.

Step 3. The backend collects routes from each container (`ip -j route show table all` via Docker exec) concurrently. A stabilisation loop waits for consecutive identical FIB hashes before taking the comparison snapshot.

Step 4. Per-node results appear as they complete. Each node shows one of three statuses:

- **Same** — simulator and container agree on all routes.
- **Different** — one or more routes added, removed, or changed.
- **Unknown** — FIB collection failed or the node did not stabilise within the timeout.

Step 5. Click a **Different** node to expand its diff view, showing added routes (green), removed routes (red), and changed routes (amber).

Step 6. Adjust **Tolerance Knobs** in the side panel to suppress differences that are not operationally significant (see §5.4).

> **Caution**
>
> Nodes marked **Unknown** are excluded from the diff and may have converging routing tables that simply did not stabilise within the timeout window. Increase the stabilisation timeout in project settings for topologies with known slow convergence before concluding a node is dissonant.

4.5 Remediation

For nodes with dissonance findings, ANK Workbench can generate CLI commands to bring the device's configuration into alignment with the intended model, and optionally push those commands to the live device.

Step 1. In the **Dissonance** tab, click **Generate Fix** next to a node with **Different** status.

Step 2. Review the generated CLI commands in the preview panel. Commands are grouped by discrepancy type (interface IP mismatch, interface status mismatch, etc.).

Step 3. If the commands look correct, click **Push Fix**. The backend streams each command to the device via Docker exec and captures the output.

Step 4. Review the command output in the log panel. Re-run dissonance to confirm the fix resolved the differences.

> **Remediation Platform Support**
>
> Automated remediation command generation currently covers **Arista EOS only**. For nodes running other NOS images (Nokia SR Linux, Cisco IOS-XE, Juniper JunOS), the dissonance report shows what needs to change but does not generate CLI commands. You must write and apply those commands manually. The **Push Fix** button is disabled for unsupported platforms.

> **Tip**
>
> Always use **Generate Fix** and review the preview before clicking **Push Fix**. The remediation generator covers the most common discrepancy types (IP address mismatches, interface shutdown/no-shutdown), but unusual configurations may produce unexpected command sequences.

4.6 Chaos Engineering

The **Chaos** panel lets you inject controlled impairments into a running topology, re-simulate, and validate that the network behaves correctly under failure conditions.

Step 1. Open the **Chaos** tab. The topology canvas shows all currently impaired nodes and links highlighted in red.

Step 2. To impair a node, right-click it on the canvas and select **Impair Node**, or use the **Add Impairment** button and select from the node list.

Step 3. To impair a link, right-click it and select **Impair Link**. Link impairments simulate loss, latency injection, or complete failure.

Step 4. Click **Re-Simulate**. The simulation runs with the impaired topology. Results appear in the Visualize tab overlaid on the current impairment state.

Step 5. Use the **Assertions** tab to run the assertion suite against the simulator with impairments active. This validates that routing convergence still occurs around the failure.

Step 6. To remove an impairment, right-click the impaired element and select **Remove Impairment**, or click **Clear All** to remove all impairments at once.

> **Tip**
>
> Chaos state renders as the highest-priority overlay on the canvas. Impaired elements are always visible regardless of which analytical overlay (routing tables, utilisation, dissonance) is active.

5 Configuration Reference

5.1 Environment Variables

Variable	Description
`ANK_WORKBENCH_API_KEY`	API key for all REST endpoints. If unset, server runs in unauthenticated development mode.
`ANK_WORKBENCH_PORT`	HTTP port (default: `8000`).
`ANK_WORKBENCH_HOST`	Bind address (default: `127.0.0.1`). Set to `0.0.0.0` to listen on all interfaces.
`ANK_WORKBENCH_DATA_DIR`	Directory for project files and artifacts (default: `~/.ank-workbench`).
`ANK_WORKBENCH_LOG_LEVEL`	Log verbosity: `debug`, `info` (default), `warning`, `error`.
`CLAB_BIN`	Path to the ContainerLab binary (default: `clab`).
`NETSIM_BIN`	Path to the netsim binary (default: `netsim`).

5.2 Topology YAML Model

The topology YAML file follows the ANK Pydantic model schema. Key fields:

```yaml
topology:
  name: my-topology          # Human label; used in UI and artifacts
  protocol: ospf             # Default protocol: ospf | bgp | static

  nodes:
    - id: r1                 # Unique node identifier (UUID v5 recommended)
      name: router-1         # Display name
      device_type: router    # router | switch | host
      platform: arista_eos   # arista_eos | nokia_srlinux | linux | ...
      execution_host: null   # For multi-host: FQDN or IP of Docker host
      properties:            # Arbitrary per-node metadata (passed to
        as_number: 65001     # netsim and ContainerLab generators)

  links:
    - id: l1
      src: r1                # Node id or name
      dst: r2
      bandwidth_mbps: 1000   # Default: 1000
      properties: {}         # Optional link-level metadata
```

Tip

Node IDs use UUID v5 seeded from the project ID and node name, making them deterministic across re-generation. If you set `id` explicitly, use a consistent string to avoid breaking cross-references in existing artifacts.

5.3 Validation Modes

Validation can run in two modes, configurable per project in **Settings → Validation**:

Non-strict mode (default) Warnings are shown as canvas annotations but do not block simulation or deployment. Use this during topology design when partial work-in-progress topologies are expected.

Strict mode Warnings are promoted to errors and block simulation and deployment. Recommended for production-bound topologies and CI/CD pipeline integration.

To enable strict mode via the API:

```
curl -X POST http://localhost:8000/api/projects/{id}/validate \
  -H "X-API-Key: $ANK_WORKBENCH_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"strict": true}'
```

5.4 Dissonance Tolerance Knobs

Tolerance knobs suppress categories of differences that are not operationally significant. Configure them in **Dissonance → Settings** or pass them in the API request body.

Knob	Effect
`ignore_next_hop_order`	Treats ECMP paths as sets (order-independent). Suppresses differences that arise from different route selection orderings.
`ignore_protocol`	Suppresses differences between `connected` and `direct` route types that vary by simulator vs kernel route utility.
`ignore_metrics`	Ignores route metric differences. Useful when the simulator uses different default metrics.
`excluded_prefixes`	List of CIDR prefixes to exclude entirely (e.g., link-local `169.254.0.0/16`, management `192.168.0.0/24`).

6 API Reference

All REST endpoints are available at `http://localhost:8000/api/`. Authentication uses the `X-API-Key` header. Interactive documentation is available at `http://localhost:8000/docs` (Swagger UI) and `http://localhost:8000/redoc` (ReDoc).

6.1 Project Endpoints

Project Management

Command	Description
`GET /api/projects`	List all projects
`POST /api/projects`	Create a new project
`GET /api/projects/{id}`	Get project details
`DELETE /api/projects/{id}`	Delete a project and all its artifacts
`GET /api/projects/{id}/topology`	Get topology YAML (plain text)
`PUT /api/projects/{id}/topology`	Update topology YAML

6.2 Workflow Endpoints

Workflow Operations

Command	Description
`POST /api/projects/{id}/validate`	Run validation (body: `{"strict": bool}`)
`POST /api/projects/{id}/simulate`	Run simulation
`POST /api/projects/{id}/assertions/run`	Run assertion suite
`POST /api/projects/{id}/dissonance/run`	Run dissonance comparison
`GET /api/projects/{id}/dissonance/latest`	Get latest dissonance report

6.3 ContainerLab Endpoints

ContainerLab Lifecycle

Command	Description
`POST /api/containerlab/deploy`	Deploy topology to ContainerLab
`DELETE /api/containerlab/destroy`	Destroy all containers
`GET /api/containerlab/status`	Get per-container readiness status
`POST /api/containerlab/exec`	Execute command in a container

6.4 Dissonance and Remediation Endpoints

Dissonance & Remediation

Command	Description
`POST /api/dissonance/{id}/remediate/{node}`	Generate remediation commands for a node
`POST /api/dissonance/{id}/apply/{node}`	Push remediation commands to live device
`GET /api/dissonance/{id}/report`	Get full dissonance report as JSON

6.5 WebSocket Events

Connect to `ws://localhost:8000/api/ws` to receive real-time events. Events are JSON objects with a `type` field and a `payload` field.

Event type	Payload
`workflow.step.started`	`step`, `timestamp`
`workflow.step.completed`	`step`, `duration_ms`, `artifact_path`
`workflow.step.failed`	`step`, `error`, `suggestion`
`workflow.staleness_changed`	Map of `step` → `stale` boolean
`dissonance.node.started`	`node_name`
`dissonance.node.completed`	`node_name`, `status`, `added`, `removed`, `changed` counts
`assertions.assertion.completed`	`index`, `result`, `output_snippet`
`containerlab.container.ready`	`container_name`, `readiness_status`
`chaos.impairment.applied`	`element_id`, `element_type`, `impairment_type`

> **Tip**
>
> The WebSocket bus is the preferred way to monitor long-running operations from external scripts. Connect before triggering the operation and filter by event type. An example using the `websockets` Python library is shown in the online documentation at `/docs/#section/WebSocket-Integration`.

6.6 Authentication Example

```
# Validate a topology in strict mode
curl -s -X POST \
  http://localhost:8000/api/projects/my-project/validate \
  -H "X-API-Key: $ANK_WORKBENCH_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"strict": true}' | python3 -m json.tool
```

7 Troubleshooting

Problem 1: Container readiness check times out; one or more containers show **Timeout** after deployment.

Cause: Vendor NOS images have widely varying boot times and no standardised readiness signal. Arista cEOS can take 3+ minutes; Nokia SR Linux up to 5 minutes. Heavy host load or insufficient RAM can push boot times beyond the default 10-minute timeout. The per-kind readiness strategy may also be outdated for newer NOS image versions.

Solution: Increase the readiness timeout in **Settings → ContainerLab → Readiness Timeout**. Ensure the host has sufficient free RAM (at least 512 MB per container for cEOS). If a specific image version consistently fails, check the ANK Workbench release notes for updated readiness strategies.

Problem 2: Dissonance detection reports nodes as **Unknown**; stabilisation loop does not converge.

Cause: The SHA-based stabilisation heuristic requires two consecutive identical FIB snapshots within the polling window. If the routing protocol is still converging (slow adjacency formation, large routing table churn), the FIB changes between polls and the node never stabilises. The default 30-second timeout may be insufficient for OSPF topologies larger than 20 nodes or for BGP with many prefixes.

Solution: Increase the stabilisation timeout in **Dissonance → Settings → Stabilisation Timeout**. For BGP topologies, allow 60–120 seconds for full route propagation before running dissonance. Check container logs (**Terminal** overlay) for protocol convergence messages.

Problem 3: Remediation **Push Fix** button is greyed out or generates a "Platform not supported" error.

Cause: Automated remediation command generation is currently implemented for Arista EOS only. Nodes running Nokia SR Linux, Cisco IOS-XE, Juniper JunOS, or Linux do not have registered remediation generators.

Solution: Review the dissonance diff manually and write the required CLI commands for the target platform. Apply them via the **Terminal** overlay (exec into the container) or directly on physical devices. The **Generate Fix** button will generate commands only for supported nodes in a mixed-platform topology.

Problem 4: The topology canvas shows stale overlay data (all overlays dimmed to grey) after editing the YAML, even though simulation was re-run.

Cause: Stale-dimming is triggered by a SHA-256 hash mismatch between the topology content at simulation time and the current editor content. If whitespace or comment-only changes were made, the hash differs even though the logical topology is identical. A failed simulation run that left a partial artifact can also cause this.

Solution: Click **Re-Simulate** to produce a fresh artifact against the current topology content. If the overlay remains dimmed after a successful simulation, refresh the browser page to force a full React state reset. Check the browser console for `staleness_changed` WebSocket events to diagnose the specific hash mismatch.

Problem 5: ANK Workbench fails to deploy topologies with more than 200 nodes; the Workflow panel shows a timeout or OOM error.

Cause: The platform has been tested and optimised for topologies up to 200 nodes. Larger topologies exceed resource limits in the overlay compositor (browser DOM), the concurrent Docker exec pool (FIB collection), and in some cases ContainerLab's own topology management.

Solution: Use the multi-host deployment feature (§4.3) to distribute containers across multiple Docker hosts, reducing per-host resource pressure. For the canvas, enable **Cluster View** in the Visualize tab to aggregate nodes into logical groups and reduce DOM complexity. Contact the maintainers if you need to evaluate 500+ node scale — this is tracked as a planned improvement.

Problem 6: The **netsim** or **netflowsim** binary is not found at simulation time, even though it is installed.

Cause: ANK Workbench resolves binary paths using the `PATH` of the process that started the server. If the server was started in a different shell environment (e.g., via a systemd service or supervisor), that environment may not include the directory where netsim/netflowsim are installed.

Solution: Set the `NETSIM_BIN` environment variable to the absolute path of the netsim binary before starting the server (see §5). Similarly, set `NETFLOWSIM_BIN` if netflowsim is in a non-standard location. Verify the paths with `which netsim netflowsim` and restart the server so it picks up the updated environment.

Problem 7: The web UI loads, but the canvas is blank or extremely slow for large topologies.

Cause: The canvas renderer and overlay compositor are CPU-bound in the browser. Very large graphs can also exceed WebGL/SVG resource limits depending on the renderer backend, and Chrome may throttle timers on background tabs. In addition, dissonance overlays can add significant DOM/SVG complexity.

Solution: Start with **Cluster View** and disable expensive overlays (FIB heatmaps, diff markers) until you have the layout stable. If you need to work at 500+ nodes, use the **Time Machine** scrubber sparingly (it forces re-render) and consider splitting the topology into PoP-sized subgraphs for editing and simulation.

8 Integration with the ANK Ecosystem

ANK Workbench is the orchestration layer in a family of specialised networking tools. Each tool can be used independently, but they are designed to work together.

8.1 netsim — Protocol Simulation Backend

netsim is the protocol simulator that powers the **Simulate** workflow. ANK Workbench invokes netsim as a subprocess, passing the ANK topology model as input, and consumes its JSON artifact output.

For direct netsim use (outside the Workbench), refer to the netsim documentation for command-line flags, protocol configuration, and trace format.

See also: *netsim User Manual* (NETSIM-UM-001) — Protocol configuration, convergence trace format, and solver options.

8.2 NetVis — Topology Visualisation

The ANK Workbench topology canvas is built on NetVis, which provides the SVG renderer, overlay compositor, and Time Machine scrubber. NetVis can also be used as a standalone visualisation tool by feeding it simulation artifacts directly.

See also: *NetVis User Manual* (NETVIS-UM-001) — Standalone visualisation, overlay configuration, and export formats (SVG, PNG, PDF).

8.3 ContainerLab — Container-Based Emulation

ContainerLab manages the lifecycle of container-based NOS instances. ANK Workbench generates ContainerLab topology YAML from the ANK topology model and delegates all container management to the `clab` CLI. For advanced ContainerLab configuration (custom startup configs, bridge networks, VxLAN endpoints), edit the generated `.clab.yml` files in `~/.ank-workbench/projects/{id}/clab/`.

See also: *ContainerLab Documentation* (`https://containerlab.dev`) — Topology schema, supported NOS kinds, and multi-host deployment.

8.4 AutoNetKit — Topology Modelling

The ANK Workbench topology model is grounded in AutoNetKit's declarative network modelling library. AutoNetKit provides the Pydantic schema, graph algorithms, and IP addressing logic that underpin the Workbench's validation engine and topology generator.

See also: *AutoNetKit User Manual* (ANK-UM-001) — Topology schema, CLI workflows, and integration with NTE and NetCfg.

8.5 TopoGen — Topology Generation

For large-scale testing scenarios, use TopoGen to generate seed-deterministic topologies and import them directly into ANK Workbench:

```
# Generate a 64-node fat-tree and load it into a Workbench project
topogen generate --template fat-tree --nodes 64 --seed 42 \
  | curl -s -X PUT \
      http://localhost:8000/api/projects/my-project/topology \
      -H "X-API-Key: $ANK_WORKBENCH_API_KEY" \
      -H "Content-Type: text/yaml" \
      --data-binary @-
```

See also: *TopoGen User Manual* (TOPOGEN-UM-001) — Topology templates, seed parameters, and output format compatibility.

9 Further Reading

The following documents provide deeper technical coverage of topics introduced in this manual:

- ***ANK Workbench Technical Report* (ANK-WB-TR-001)** — Architecture, data model, design decisions, API reference, and known limitations. The primary reference for engineers integrating with or contributing to the ANK Workbench codebase.
- ***ANK Ecosystem Technical Report* (NETAUTO-TR-001)** — Cross-tool integration architecture, content-addressed artifact pipeline, and the dissonance detection design rationale across the full ecosystem.
- ***AutoNetKit Technical Report* (ANK-TR-001)** — Topology modelling, Pydantic schema design, and graph-theoretic validation algorithms.

- ***netsim Technical Report* (NETSIM-TR-001)** — Protocol simulation architecture, convergence trace format, and Monte Carlo traffic analysis methodology.

Online documentation, including an interactive API browser and worked examples, is available at `http://localhost:8000/docs` while the server is running.