



NetFortress API

Integration and Test Reference

API v1 - Version 3.1

1. Connection and conventions

Item	Value
Base URL	http://APPLIANCE-IP:8080/api/v1 (development); HTTPS reverse proxy required for production
Format	application/json; UTF-8
Authentication	Bearer session token or scoped API token
API token prefix	fw_
Timestamps	ISO 8601 UTC
Versioning	URI version /api/v1
Interactive specification	http://APPLIANCE-IP:8080/docs
OpenAPI JSON	http://APPLIANCE-IP:8080/openapi.json

Never send credentials or activation keys over plain HTTP outside an isolated test network.

Authentication header

Authorization: Bearer YOUR_ACCESS_TOKEN

Common response codes

Code	Meaning
200	Successful read or action
201	Resource created
204	Successful deletion with no response body
401	Missing, invalid, revoked, or expired authentication
402	Feature requires trial or purchased licence
403	Authenticated but required API scope is missing
404	Resource or transaction not found
409	Conflict, unsafe transition, or transaction failure
422	Request validation or deployment validation failed
423	Live activation safety switch is disabled
503	External licensing service unavailable

2. Authentication and tokens

GET	/auth/status	No	-
Returns whether first-administrator bootstrap is required.			
POST	/auth/bootstrap	No	One time
Creates the first administrator. Fails with 409 after bootstrap.			
POST	/auth/login	No	-
Returns an 8-hour signed session token.			
GET	/auth/me	Yes	Any
Returns the authenticated user or API-token principal.			
POST	/auth/tokens	Yes	tokens:write
Creates a scoped API token. The raw token is shown once.			
DELETE	/auth/tokens/{token_id}	Yes	tokens:write
Revokes an API token owned by the administrator.			

Bootstrap/login request

```
{
  "username": "admin",
  "password": "StrongPassword!123"
}
```

Authentication response

```
{
  "access_token": "SIGNED_SESSION_TOKEN",
  "token_type": "bearer",
  "expires_in": 28800,
  "user": {"id": 1, "username": "admin", "role": "admin"}
}
```

Create API token request

```
{
  "name": "automation",
  "scopes": ["deployment:read", "policies:read", "policies:write"],
  "expires_at": "2027-09-21T00:00:00Z"
}
```

3. System, devices, and audit

GET	/health	No	-
Liveness, software version, firewall apply mode, and network apply safety state.			
GET	/system/overview	Yes	deployment:read
Hostname, kernel, uptime, load averages, and memory usage.			
GET	/system/interfaces	No	-
Detected physical and virtual interfaces, MAC, state, MTU, speed, and addresses.			
GET	/devices	Yes	deployment:read
Refreshes ARP/NDP discovery and returns persisted device identity history.			
GET	/audit?limit=100	Yes	audit:read
Returns newest audit events. Limit is clamped to 1-500.			

Device response item

```
{
  "ip_address": "192.168.10.22",
  "mac_address": "aa:bb:cc:dd:ee:ff",
  "hostname": "office-laptop",
  "interface": "lan0",
  "state": "reachable",
  "source": "neighbour",
  "first_seen_at": "2026-09-21T10:00:00Z",
  "last_seen_at": "2026-09-21T10:05:00Z"
}
```

4. Device groups and blocking policies

GET	/device-groups	Yes	policies:read	Lists groups with member counts.
POST	/device-groups	Yes	policies:write	Creates a group.
POST	/device-groups/{group_id}/members	Yes	policies:write	Adds a MAC, IP, or hostname identity to a group.
DELETE	/device-groups/{group_id}	Yes	policies:write	Deletes a group and its memberships.
GET	/policies	Yes	policies:read	Lists blocking rules.
POST	/policies	Yes	policies:write	Creates a MAC, IP, hostname, or group blocking rule.
DELETE	/policies/{policy_id}	Yes	policies:write	Deletes a blocking rule.
GET	/policy-compiler	Yes	policies:read	Shows resolved IPv4, IPv6, MAC, and unresolved-hostname targets.

Create group

```
{"name":"Children","description":"Restricted devices"}
```

Add group member

```
{"target_type":"mac","target_value":"aa:bb:cc:dd:ee:ff"}
```

Create blocking policy

```
{
  "name": "Block children",
  "target_type": "group",
  "target_value": "1",
  "enabled": true,
  "reason": "Restricted by administrator"
}
```

5. Licensing

GET	/licensing/status	Yes	Any	Current licence state, edition, enabled features, device/account usage and limits, expiry and messages.
POST	/licensing/trial	Yes	licensing:write	Starts the one-time 60-day trial.
POST	/licensing/activate	Yes	licensing:write	Performs online activation using a licence key (FA-XXXXX-...).
POST	/licensing/activate-offline	Yes	licensing:write	Offline activation: upload the signed licence envelope downloaded from the licensing host. Works without internet.
POST	/licensing/deactivate	Yes	licensing:write	Removes the installed licence.
GET	/licensing/history?limit=100	Yes	licensing:read	Returns licensing events.

Activation request

```
{"license_key": "XXXX-XXXX-XXXX-XXXX"}
```

Important licence states

State	Expected behaviour
UNLICENSED	Basic monitoring and recovery only; trial may be available
TRIAL_ACTIVE	Full features for 60 days
TRIAL_EXPIRED	Immediate restricted/basic mode
SUBSCRIPTION_ACTIVE	Licensed features active
SUBSCRIPTION_GRACE	30-day renewal grace
PERPETUAL_ACTIVE	Purchased features remain active
SUSPENDED	Restricted/basic mode; core connectivity remains safe

6. Deployment validation and candidates

POST	/deployment/validate	No	-
Validates topology, addresses, interfaces, acknowledgements, and safety constraints.			
POST	/deployment/plan	No	-
Returns topology explanation, generated actions, limitations, and validation.			
POST	/deployment/render	Yes	deployment:write
Renders nftables and network candidates plus resolved policy targets.			
POST	/deployment/apply	Yes	deployment:write
Prepares a versioned transaction; it does not directly activate it.			
GET	/deployment/transactions/{transaction_id}	Yes	deployment:read
Returns transaction metadata and state.			

Routed gateway request

```
{
  "mode": "routed_gateway",
  "wan_interface": "eth0",
  "lan_interface": "eth1",
  "management_interface": "eth1",
  "management_address": "192.168.10.1/24",
  "upstream_gateway": "192.168.1.1",
  "vlans": [],
  "enable_dhcp": true,
  "architecture_acknowledged": true,
  "cabling_acknowledged": true,
  "recovery_acknowledged": true
}
```

Supported mode values

- routed_gateway - two-NIC routed gateway
- transparent_bridge - two-NIC inline bridge
- vlan_trunk - one-NIC tagged WAN/LAN
- dns_proxy - one-NIC limited DNS/proxy mode

7. Guarded activation transaction

POST	/deployment/transactions/{transaction_id}/activate	Yes	deployment:write
Snapshots current state, arms 120-second rollback, applies network and nftables candidates, then runs health checks.			
POST	/deployment/transactions/{transaction_id}/confirm	Yes	deployment:write
Commits a healthy activation and cancels automatic rollback.			
POST	/deployment/transactions/{transaction_id}/rollback	Yes	deployment:write
Immediately restores the saved network and nftables state.			

Activation remains locked unless `FIREWALL_ENABLE_LIVE_APPLY=1` and `FIREWALL_ENABLE_NETWORK_APPLY=1` are deliberately configured. Verify console recovery access before enabling either switch.

Transaction states

State	Meaning
prepared	Candidate exists and is not active
awaiting_confirmation	Active with rollback timer running
committed	Administrator confirmed connectivity
rolled_back	Previous state restored

8. Practical curl integration test

1. Set address

```
BASE=http://FIREWALL-IP:8080/api/v1
```

2. Log in

```
curl -sS -X POST "$BASE/auth/login" -H 'Content-Type: application/json' -d '{"username":"admin","password":"YOUR_PASSWORD"}'
```

3. Export the returned token

```
TOKEN='PASTE_ACCESS_TOKEN_HERE'
```

4. Check identity and system

```
curl -sS "$BASE/auth/me" -H "Authorization: Bearer $TOKEN"
curl -sS "$BASE/system/overview" -H "Authorization: Bearer $TOKEN"
curl -sS "$BASE/devices" -H "Authorization: Bearer $TOKEN"
```

5. Create and inspect a blocking rule

```
curl -sS -X POST "$BASE/policies" \
-H "Authorization: Bearer $TOKEN" \
-H 'Content-Type: application/json' \
-d '{"name":"Test IP block","target_type":"ip","target_value":"192.168.10.99","enabled":true,"reason":"Integration test"}'

curl -sS "$BASE/policy-compiler" -H "Authorization: Bearer $TOKEN"
```

6. Review audit

```
curl -sS "$BASE/audit?limit=20" -H "Authorization: Bearer $TOKEN"
```

9. Site operations — pools, portal, IDS/IPS, web-control groups, block page

GET	/dhcp/pools	Yes	deployment:read	Lists the extra client IP pools (each serves DHCP on its own virtual VLAN adapter).
POST	/dhcp/pools	Yes	deployment:write	Creates a pool: {name, vlan_id, subnet, pool_start, pool_end}. The appliance adds the virtual adapter, Kea subnet and kernel rules immediately.
DELETE	/dhcp/pools/{id}	Yes	deployment:write	Removes the pool, its virtual adapter and DHCP subnet.
GET	/portal/settings	Yes	policies:read	Policy-acceptance captive portal settings.
PUT	/portal/settings	Yes	policies:write	Saves and applies the portal instantly (page, nginx site, kernel set and rules).
GET	/portal/accepted	Yes	deployment:read	Devices that accepted the policy (MAC, IP, time).
DELETE	/portal/accepted/{id}	Yes	policies:write	Revokes an acceptance — the device must accept again.
POST	/portal/accept	No	-	PUBLIC. A device accepts the policy; its MAC is resolved and freed instantly.
GET	/firewall/modules	Yes	deployment:read	All security modules with plain-language detail and readiness (enforced / configured / software / planned).
PUT	/firewall/modules/{key}	Yes	deployment:write	

Enables or disables a module (desired state; the reconciler enforces).			
GET	/firewall/modules/ids_ips/status	Yes	deployment:read
Suricata install/run/kernel-rule state for the IDS/IPS module.			
POST	/firewall/modules/ids_ips/install	Yes	deployment:write
Installs Suricata on demand, writes the managed NFQUEUE config, starts the service.			
POST	/firewall/modules/ids_ips/uninstall	Yes	deployment:write
Stops and removes Suricata; the module is switched off.			
POST	/categories	Yes	policies:write
Creates a custom web-control group card.			
DELETE	/categories/{name}	Yes	policies:write
Deletes a custom group (built-in categories cannot be deleted).			
POST	/categories/{name}/members	Yes	policies:write
Adds a service to a custom group.			
DELETE	/categories/{name}/members/{application}	Yes	policies:write
Removes a service from a custom group.			
POST	/applications	Yes	policies:write
Creates a custom service {name, category, domains[]} — behaves exactly like curated services and is enforced via DNS when on.			
DELETE	/applications/{name}	Yes	policies:write
Deletes a custom service (curated services are switched off, not deleted).			
GET	/block-page/settings	Yes	policies:read

Block page configuration (branding, logo, custom page, external redirect, access requests).

PUT	/block-page/settings	Yes	policies:write	Saves branding and mode settings (custom_html_active, redirect_url).
POST	/block-page/logo	Yes	policies:write	Uploads a logo (PNG/JPEG/SVG under 256 KB); served on the live block page.
DELETE	/block-page/logo	Yes	policies:write	Removes the logo.
POST	/block-page/custom	Yes	policies:write	Uploads your own HTML page (under 200 KB) as the block page; placeholders {{REQUESTED_HOST}} {{CLIENT_IP}} {{HOSTNAME}} {{POLICY}} are filled live.
DELETE	/block-page/custom	Yes	policies:write	Removes the uploaded page and returns to the generated page.
GET	/block-page/preview	No	-	PUBLIC. The exact page blocked clients see in the active mode (generated, custom, or redirect stub).
GET	/block-page/ca	No	-	PUBLIC. The appliance CA certificate for client installation.
POST	/access-requests	No	-	PUBLIC. Submitted from the block page request-access form; the appliance records the client IP itself.
POST	/operations/backup	Yes	deployment:write	One click: saves a timestamped backup on the appliance and reports pending signed updates.

Create custom service

```
{
  "name": "showmax-extra",
  "category": "streaming-media",
  "domains": ["showmax-extra.com", "cdn.showmax-extra.com"]
}
```

Custom services are enforced through the same DNS compilation as curated ones: when the service is switched on its domains are blocked, and the reconciler re-applies dnsmasq within the minute. An external `redirect_url` on the block page takes priority over an uploaded custom page. Management networks on the firewall engine accept a comma-separated list.

10. Users, groups and permissions (Build 5)

User groups are the source of API scopes. Every account belongs to one group; the group's permissions map to scopes (full_access grants all scopes). A session's scopes are recomputed from the group on every request, so a permission change takes effect on the member's very next call. API tokens stay bound to the scopes their creator had when the token was minted.

GET	/permissions	Yes	users:read	The permission catalog the Users page renders as checkboxes (11 keys).
GET	/user-groups	Yes	users:read	All groups with member counts, permissions and builtin flags.
POST	/user-groups	Yes	users:write	Creates a group: name, description and permissions from the catalog.
PUT	/user-groups/{id}	Yes	users:write	Replaces name/description/permissions. The Administrators group cannot lose full access.
DELETE	/user-groups/{id}	Yes	users:write	Deletes only an empty custom group; built-in groups are refused.
GET	/users	Yes	users:read	Accounts with group membership and enablement.
POST	/users	Yes	users:write	Creates an account in a group: username, password, group_id.
PUT	/users/{id}	Yes	users:write	Changes enabled, role or group_id; an absent group_id keeps the current group. The last enabled full-access account cannot be disabled or deleted.
DELETE	/users/{id}	Yes	users:write	

Deletes an account; last-administrator protection applies.

GET	/system/admin-port	Yes	deployment:read
The TCP port the admin dashboard answers on (default 80).			
PUT	/system/admin-port	Yes	deployment:write
Moves the dashboard: port 1-65535. Re-applies the firewall rules first, switches the web server last; rolls the setting back if the switch fails.			
GET	/system/docs/{filename}	No	-
Downloadable manuals (PDF) served from the appliance.			

Create group

```
{
  "name": "Helpdesk",
  "description": "First line support",
  "permissions": ["devices", "access_requests"]
}
```

Built-in groups

Group	Permissions
Administrators	full_access
Operators	dashboard, devices, web_control, access_requests, reports, modules, operations
Viewers	dashboard, reports

11. API scopes

Scope	Permits
deployment:read/write	Network, DHCP, recovery, backup and update transactions
policies:read/write	Groups, zones, NAT, schedules, applications and access requests
dns:read/write	DNS records, domain rules, blocklists and scheduled refresh
proxy:read/write	Proxy/cache configuration and purge
events:write	Ingest normalized enforcement events
audit:read/write	Audit, reports, retention and support resources
licensing:read/write	Status, trial and activation lifecycle
users:read/write	Group-managed accounts, user groups and permissions
*	All scopes; administrator sessions inherently have full access

12. Integration rules

- Discover capabilities from /openapi.json and pin the API major version in clients.
- Treat 401 differently from 403: refresh/re-authenticate for 401; request scope changes for 403.
- Do not retry POST blindly after timeouts; first query resource or transaction status.
- Never automatically confirm a deployment transaction without independently testing management and gateway connectivity.
- Store API tokens in a secret manager; never log them.
- Expect additive response fields within API v1; ignore unknown fields.

13. Acceptance baseline

Version 3.1 adds edition-based licensing (Home/Essential/Professional/Corporate), device and account limits, offline activation, the enforce-but-freeze lapse behaviour, user groups with per-group permissions, automated last-known-good blocklist updates, normalized security events, reports and CSV export, access-request decisions, retention, zones, NAT, DHCP reservations, local DNS, signed updates and atomic restore. The live OpenAPI contract at /openapi.json is authoritative.