API Best Practices Guide

FortiManager 7.6.0

FORTINET DOCUMENT LIBRARY
https://docs.fortinet.com

FORTINET VIDEO LIBRARY
https://video.fortinet.com

FORTINET BLOG
https://blog.fortinet.com

CUSTOMER SERVICE & SUPPORT
https://support.fortinet.com

FORTINET TRAINING & CERTIFICATION PROGRAM
https://www.fortinet.com/training-certification

FORTINET TRAINING INSTITUTE
https://training.fortinet.com

FORTIGUARD LABS
https://www.fortiguard.com

END USER LICENSE AGREEMENT
https://www.fortinet.com/doc/legal/EULA.pdf

FEEDBACK
Email: techdoc@fortinet.com

March 12, 2026
FortiManager 7.6.0 API Best Practices Guide
02-760-1149400-20260312

TABLE OF CONTENTS

Change Log

Date	Change Description
2025-07-29	Initial release.
2025-08-15	Clarified examples in Best Practices on page 7.
2025-10-16	Updated Getting started on page 6.
2025-11-05	Updated Best Practices on page 7.
2026-03-12	Updated Best Practices on page 7.

Introduction

The FortiManager JSON API allows you to perform configuration and monitoring operations on a FortiManager appliance or VM. The JSON API is based on JSON-RPC, a remote procedure call protocol encoded in JSON. It provides access to a wide range of features, including device management, policy configuration, object manipulation, and task execution.

By using structured JSON requests over HTTPS, you can efficiently interact with FortiManager to streamline operations, and scale network management tasks across multiple Fortinet devices.

This document contains the following information:

Getting started

1. **Access the Fortinet Developer Network**: Find detailed information about the FortiManager API on the Fortinet Developer Network site, https://fndn.fortinet.net. The Fortinet Developer Network site provides documentation for getting started with FortiManager APIs, setting up API users, making API requests, and more.

   Access to the Fortinet Developer Network requires an account. Please contact your Fortinet representative or Fortinet support for more information on getting account access to the Fortinet Developer Network.

2. **Create an API User**: Configure a FortiManager API user. The type of API user required depends on the type of authentication you will be using:

REST API Admin with a predefined API key	1. Create a *Rest API Admin* with *Read/Read-Write* privileges configured for *JSON API Access*. 2. Apply an *Admin Profile* with the required permissions to complete the desired requests. 3. Copy the automatically generated API key to use in the request header using the bearer authentication scheme.  The generated API key is permanent, which means the same user account will always share the same session and you do not need to use the `login`/`logout` endpoints. Because of this benefit, predefined API keys are useful for simplifying automation workflows.
Session-based authentication	1. Create a regular administrator with *Read/Read-Write* privileges configured for *JSON API Access*. 2. Apply an *Admin Profile* with the required permissions to complete the desired requests. 3. Use the `login` endpoint with the configured administrator credentials to generate the session ID to be used within API requests.

3. **Use a lab environment to test APIs**: When using FortiManager APIs, it is recommended to begin first in a lab environment to test and validate workflows before deployment to production.

Best Practices

FortiManager offers powerful APIs to automate and integrate with your network and security management workflows. Follow these best practices to ensure secure, efficient, and reliable use.

Authentication and access control

Best practice	Description
Use `logout` requests when using session-based authentication	• Sending `logout` requests when using FortiManager's JSON API is critical to free up session slots. Neglecting this can inadvertently fill up all available admin sessions, either at the individual user or system level, resulting in denial of management access until sessions expire or are forcefully ended. • This is controlled by configurable parameters at both the user (`system.admin.user.login-max`) and system level (`system.admin.setting.admin-max-login`). **Example: Making a logout request** `{ "method": "exec", "params": [ { "url": "/sys/logout" } ], "session": "...", "id": 1 }`
Leverage role-based access control	• Use RBAC (Role-Based Access Control) to restrict API access to only required resources. You can configure RBAC for your API administrator using Admin Profiles configured in the FortiManager. For more

Best practice	Description
	information, see Administrator Profiles.
Restrict network access	• Limit access with an IP allowlist or using trusted source control. See Trusted hosts. • When additional flexibility is required (for example, if you require more than 10 trusted hosts), you can leverage *system.local-in* policies.

API requests

Best practice	Description
Use HTTPS	• Use HTTPS only for all API communications.

Use proper formatting	• Structure requests using proper JSON format and required parameters.
	Example: HTTP method and URL
	`POST https://<fmg_ip>/jsonrpc`
	Example: Standard JSON RPC body format
	`{ "method": "...", "params": [ ... ], "session": "...", "id": 1 }`

Leverage and monitor asynchronous APIs	• Take advantage of the asynchronous mechanism available in certain JSON API URLs. If the API supports task creation, include flags like `create_task` and `nonblocking` in your JSON request to initiate a new task. This approach allows you to send multiple JSON requests sequentially, each creating its own task. You can then wait for all tasks to finish asynchronously. A common use case is adding multiple devices using separate JSON requests. Alternatively, using the `add/dev-list` method is also an effective option.

Best practice	Description

	Example: Refresh multiple devices
	{
	"params": [
	{
	"url": "/dvm/cmd/update/dev-list",
	"data": {
	"adom": "root"
	"flags": [
	"create_task",
	"nonblocking"
	],
	"update-dev-member-list": [
	{
	"name": "fgt_1"
	},
	{
	"name": "hub_1"
	},
	{
	"name": "hub_2"
	}
	]
	}
	}
	],
	"session": "...",
	"id": 1
	}
	• Monitor asynchronous APIs to ensure task completion. If you `logout` while an API request run asynchronously, it might be reported as failed in the *Task Manager* because FortiManager can't find a valid API session to complete the task.

<table>
<tr><th>Best practice</th><th>Description</th></tr>
<tr><td></td><td>Example: Monitor asynchronous APIs<pre>{
    "method": "get",
    "params": [
        {
            "url": "/task/task/1"
        }
    ],
    "session": "...",
    "id": 1,
    "verbose": 1
}</pre></td></tr>
<tr><td>Use option names in requests</td><td>• For any attribute that is defined as an option, always use name of option instead of internal index value in JSON request. The name is more intuitive and when used, it produces more readable and easier to understand code.
For example, for <code>firewall/policy</code> the action can include <code>deny</code>, <code>accept</code>, <code>ipsec</code>.</td></tr>
</table>

Best practice	Description

	Example: Using option names in a request
	
	`{`
	`  "method": "add",`
	`  "params": [`
	`  {`
	`  "url": "/pm/config/adom/root/pkg/test_pkg/firewall/policy",`
	`  "data": [`
	`  {`
	`  "name": "Allow_HTTP",`
	`  "srcintf": "port1",`
	`  "dstintf": "port2",`
	`  "srcaddr": "all",`
	`  "dstaddr": "all",`
	`  **"action": "accept",**`
	`  "schedule": "always",`
	`  "service": "HTTP",`
	`  "logtraffic": "all"`
	`  }`
	`  ]`
	`  }`
	`  ],`
	`  "session": "...",`
	`  "id": 1`
	`}`
	
	• Avoid the use of the numeric index values, such as `0`, `1`, `2`.

Best practice	Description

	Avoid: Using numeric index values in request
	
	`{`
	`  "method": "add",`
	`  "params": [`
	`  {`
	`  "url": "/pm/config/adom/root/pkg/test_pkg/firewall/policy",`
	`  "data": [`
	`  {`
	`  "name": "Allow_HTTP",`
	`  "srcintf": "port1",`
	`  "dstintf": "port2",`
	`  "srcaddr": "all",`
	`  "dstaddr": "all",`
	`  **"action": 1,**`
	`  "schedule": "always",`
	`  "service": "HTTP",`
	`  "logtraffic": "all"`
	`  }`
	`  ]`
	`  }`
	`  ],`
	`  "session": "...",`
	`  "id": 1`
	`}`
	
	• Add **"verbose": 1** in your **get** request to receive option names in the response instead of internal index values.
	
	Example: Using verbose output
	
	`{`
	`  "method": "get",`
	`  "params": [ ... ],`
	`  "session": "...",`
	`  "id": 1,`
	`  **"verbose": 1**`
	`}`

Error handling and resilience

Best practice	Description

Monitor HTTP status codes	• Monitor standard HTTP status codes:
	  • **100**: Request header received and waiting for the request body.
	  • **200**: Success.
	  • **401**: Invalid session or token.
	  • **403**: Permission denied.
	  • **404**: Not found. Unable to find the specified resource.
	  • **405**: Method not allowed for this resource.
	  • **500**: Internal error.
	  • **503**: Service unavailable.

Use retry logic with exponential backoff	• Implement retry logic with exponential backoff for transient errors to prevent flooding the FortiManager with excessive repeat requests.
Log request/response for auditing and troubleshooting	• Record the details of each API interaction, both in requests and responses so that they can be assessed if an API calls or fails to return data, and provide a clear record of the actions taken using the API.
Compare activities/tasks performed from the GUI when there are errors with the API	• Compare activities performed using the API with the same activities performed using the GUI. Troubleshooting issues from the GUI interface is an easier way to isolate problems and determine if the issue is platform use/configuration or API related. If you have problems performing a tasks using the GUI (for example, installs) then you'll likely also face issues performing the same task using the API.

Performance optimization

Best practice	Description
Limit request frequency	• Avoid unnecessary polling or tight loops.
Use filter and fields	• Use `fields`, `filter`, `range`, and `loadsub` to retrieve only the necessary data when querying large tables.

Best practice	Description

	Example: Getting a firewall address group containing member "host_001"
	
	`{`
	`  "method": "get",`
	`  "params": [`
	`  {`
	`  "fields": [`
	`  "name",`
	`  "member"`
	`  ],`
	`  "filter": [`
	`  "member",`
	`  "contain",`
	`  "host_001"`
	`  ],`
	`  "loadsub": 0,`
	`  "url": "/pm/config/adom/root/obj/firewall/addrgrp"`
	`  }`
	`  ],`
	`  "session": "...",`
	`  "id": 1`
	`}`
	
	• Using the `filter` parameter with the `delete` method allows you to bulk delete objects that match specific criteria. This can improve performance when compared with deleting objects using individual JSON requests. **`"confirm": 1`** must be included in the JSON request to confirm and execute the deletion.

<table>
<tr><th>Best practice</th><th>Description</th></tr>
<tr><td></td><td>

Example: Deleting objects matching a filter

<pre>
{
    "method": "delete",
    "params": [
        {
            "confirm": 1,
            "filter": [
                "name",
                "like",
                "west%"   <----- virtual IP that starts with string west
            ],
            "url": "/pm/config/adom/root/obj/firewall/vip"
        }
    ],
    "session": "...",
    "id": 1
}
</pre>

- Avoid loading full tables when only a subset of data is needed that could be obtained using filter/fields.

Avoid: Loading full table when filter and fields can be used

<pre>
{
    "method": "get",
    "params": [
        {
            "url": "/pm/config/adom/root/obj/firewall/addrgrp"
        }
    ],
    "session": "...",
    "id": 1,
    "verbose": 1
}
</pre>

</td></tr>
<tr><td>Cache data</td><td>

- Cache static or rarely changing data locally when appropriate.

</td></tr>
<tr><td>Use object URLs for specific items</td><td>

- Access specific objects using the full object URL.

</td></tr>
</table>

<table>
<tr><th>Best practice</th><th>Description</th></tr>
<tr><td></td><td>

Example: Using an object's URL

<pre>
{
    "method": "get",
    "params": [
        {
            "url":
"/pm/config/adom/root/obj/firewall/address/Vancouver"
        }
    ],
    "session": "...",
    "id": 1,
    "verbose": 1
}
</pre>

- Avoid using general table URLs when you only need access to a specific object.

Avoid: Using general table URLs to access specific objects

<pre>
{
    "method": "get",
    "params": [
        {
            "url": "/pm/config/adom/root/obj/firewall/address"
        }
    ],
    "session": "...",
    "id": 1,
    "verbose": 1
}
</pre>

</td></tr>
<tr><td>Leverage the target attribute when using <code>/sys/proxy/json</code> for multi-device FortiOS REST API requests</td><td>

- In order to send FortiOS REST API requests to multiple FortiGates at the same time, use the JSON URL <code>/sys/proxy/json</code> and put all device names in the target to achieve the asynchronous effect.

</td></tr>
</table>

<table>
<tr><th>Best practice</th><th>Description</th></tr>
<tr><td></td><td>
Example: Using /sys/proxy/json
<pre>{
    "method": "exec",
    "params": [
        {
            "data": {
                "action": "get",
                "resource": "/api/v2/monitor/router/ipv4"
                "target": [
                    "adom/root/device/fgt1"
                ]
            },
            "url": "sys/proxy/json"
        }
    ],
    "session": "...",
    "id": 1
}</pre>
Example: Setting the target when there are multiple devices/device groups
<pre>"target": [
    "adom/root/group/emea_devices",
    "adom/root/group/apac_devices",
    "adom/root/device/fgt1",
    "adom/root/device/fgt2"
]</pre>
</td></tr>
<tr><td>Monitor system resources</td><td>• The number of JSON API tasks that can run simultaneously depends on the currently available resources. This includes the free memory size, number of CPUs, clock speed of CPU, free disk size, etc. Avoid running too many tasks simultaneously without considering system limitations.</td></tr>
<tr><td>Clone large objects</td><td>• Cloning is faster than manually recreating complex objects and all sub-objects from scratch</td></tr>
</table>

Best practice	Description

	Example: Cloning an ADOM
	`request = {`
	`  "method": "clone",`
	`  "params": [`
	`  {`
	`  "data": {`
	`  "name": "adom2"`
	`  },`
	`  "url": "/dvmdb/adom/adom1"`
	`  }`
	`  ],`
	`  "session": "...",`
	`  "id": 1`
	`}`

Create a baseline	• Baseline your FortiManager with and without your API client tools running. Depending on how you use FortiManager you might find performance issues by creating a baseline with GUI only so you can determine if your API calls are negatively impacting the system.
Use high-availability mode to distribute installations	• If installation operations are slow, configure a FortiManager HA cluster and enable `perform-improve-by-ha {enable | disable}` to distribute the workload of installation tasks. See the CLI Reference for more information.
Minimize table size to improve performance in the GUI	• Consider how much data your tables present considering the API can exponentially grow tables with ease. If possible, moving to a strategy that creates fewer objects with common names but adds per-device mapping per object will decrease the number of objects in the table and therefore improve the experience in the GUI. • As an example, if your address table(s) within an ADOM has tens or hundreds of thousands of objects, you might notice a slowdown in presenting this data in the GUI while the API with a proper filter does not.
Optimize FortiManager architecture for API efficiency	• Plan your FortiManager and FortiOS architecture before starting API development to reduce complexity and technical debt. Leverage built-in features like FortiManager variables and dynamic CLI/Jinja scripts to minimize API calls and simplify configuration management. • If you find yourself updating lots of objects in the form of lists and/or groups that are time sensitive, consider creating dynamic lists in the FortiGate using the SDN connectors. See Public and private SDN connectors.

<table>
<tr><th>Best practice</th><th>Description</th></tr>
<tr><td>Use multiplexing to regroup multiple APIs into a single request when appropriate</td><td>

Multiplexing can be used to group multiple APIs operations into a single request.
For example, you can use data multiplexing when you need to create multiple firewall addresses using one API call.
Example: Creating multiple firewall addresses using data multiplexing
<pre>{
    "method": "add",
    "params": [
        {
            "data": [
                {
                    "name": "test_001",
                    "type": "ipmask",
                    "subnet": [
                        "10.0.0.1",
                        "255.255.255.0"
                    ]
                },
                {
                    "name": "test_002",
                    "type": "ipmask",
                    "subnet": [
                        "10.0.0.2",
                        "255.255.255.0"
                    ]
                },
                {
                    "name": "test_003",
                    "type": "ipmask",
                    "subnet": [
                        "10.0.0.3",
                        "255.255.255.0"
                    ]
                }
            ],
            "url": "/pm/config/adom/root/obj/firewall/address"
        }
    ],
    "session": "...",
    "id": 1
}</pre>

Parameter multiplexing can also be used to improve performance when

</td></tr>
</table>

Best practice	Description
	all the JSON URLs in the request have the same prefix, for example `/pm/config`. **Example: Retrieving information about specific objects in an ADOM using parameter multiplexing** `{ "method": "get", "params": [{ "url": "/pm/config/adom/root/obj/webfilter/profile", "fields": ["name"], "loadsub": 0 }, { "url": "/pm/config/adom/root/obj/firewall/address", "fields": ["name"], "loadsub": 0 }, { "url": "/pm/config/adom/root/obj/firewall/service/group", "fields": ["name"], "loadsub": 0 }], "session": "...", "id": 1 }` • Using parameter multiplexing in requests that contain URLs with different prefixes is not supported.

Best practice	Description
	Avoid: Using parameter multiplexing with requests that include different URL prefixes `{ "method": "get", "params": [ { "fields": [ "name" ], "loadsub": 0, "url": "/dvmdb/group" }, { "fields": [ "name" ], "loadsub": 0, "url": "/pm/config/adom/root/obj/firewall/address" } ], "session": "...", "id": 1 }`
When working in Workspace mode, lock the ADOM once, perform all necessary changes, then commit and unlock the ADOM	• Each commit performed in Workspace mode triggers a database copy operation. To reduce the time taken to perform each copy operation, plan your changes ahead, group them logically, and apply them in one session before committing. For example, lock the ADOM, add multiple objects, commit the changes, and unlock the ADOM. • Avoid locking the ADOM, making a single change, committing, unlocking the ADOM, and repeating this process multiple times.

Security and compliance

Best practice	Description
Change passwords and API sessions	• Change admin passwords periodically to reduce the risk of unauthorized access and exposure from compromised credentials. • Rotate API sessions regularly by performing `logout` and `login` requests.

Best practice	Description
Don't embed credentials	• Avoid embedding credentials in scripts or config files.

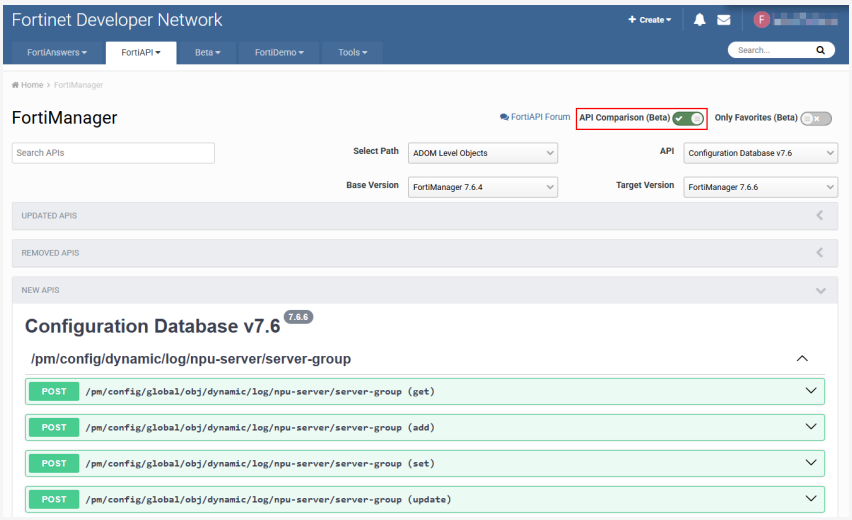
Monitor your API activity	• Use the FortiManager Event Log to monitor API activity for anomalies. For more information, see Event Log.
	You can configure JSON API logging using the `set jsonapi-log` command in the FortiManager CLI:
	`config system global`
	`(global)# set jsonapi-log`
	`all` logging both jsonapi request & response.
	`disable` disable jsonapi log.
	`request` logging jsonapi request.
	`response` logging jsonapi response.

Change management

Best practice	Description
Maintain version compatibility	• Maintain version compatibility during FortiManager upgrades. Consult the FortiManager Release Notes to verify which managed devices are supported by your FortiManager version.

Support and resources

Best practice	Description
Consult the Fortinet Developer Network (FNDN)	• Refer to https://fndn.fortinet.net for API docs and tools. • Join the Fortinet Developer Community for questions and examples.
Compare APIs between two FortiManager versions using the API Comparison feature	• Use the *API Comparison* feature in *FortiAPI* on the Fortinet Developer Network to compare API information between two FortiManager versions, and see APIs that have been updated, removed, or added.

Best practice	Description
	
Contact Fortinet Support for further assistance	• Contact Fortinet Support for API-related assistance, or use the Fortinet Support Community to find answers to questions in the articles and support forums.

www.fortinet.com