

# Makefile API testing



I am testing an api I am using prism, have defined a postman mock server, and am using newman. How do I do this with makefile rules

To automate API testing using Prism (OpenAPI mock server), a Postman mock server, and Newman (Postman CLI runner) in a **Makefile**, you can define a structured set of targets to:

1. Start Prism with your OpenAPI spec
2. Run Newman tests against the Prism mock server or Postman mock server
3. Upload or configure environments/collections if needed

Here's a complete example of **Makefile** rules with comments for clarity:

## Makefile Example:

```
makefile

# ----- CONFIGURATION -----

PRISM      := npx @stoplight/prism-cli
NEWMAN     := npx newman
SPEC       := openapi/c2m_openapi_spec_final.yaml
PRISM_PORT := 4010
MOCK_URL_FILE := mock-url.txt
COLL_FINAL  := postman/Click2Mail.collection.merged.json
OVERRIDE_FILE := postman/mock-env.json
REPORT_HTML := newman-report.html

# ----- TARGETS -----

.PHONY: prism prism-stop postman-mock mock-test test

## Start Prism mock server
prism:
    @echo "🚀 Starting Prism mock server on port $(PRISM_PORT)..."
    $(PRISM) mock $(SPEC) -p $(PRISM_PORT)

## Kill Prism server if needed (assumes Prism is running in foreground)
prism-stop:
    @echo "🛑 Killing Prism on port $(PRISM_PORT)..."
    @lsof -ti tcp:$(PRISM_PORT) | xargs kill -9 || echo "No Prism process found"

## Run tests against Postman mock server
postman-mock:
    @echo "📄 Getting Postman mock URL..."
    $(eval MOCK := $(shell cat $(MOCK_URL_FILE)))
    @echo "🔗 Using mock URL: $(MOCK)"
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=$(MOCK) \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(OVERRIDE_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

## Run tests against Prism mock server
mock-test:
```

```
@echo "🔥 Running Newman tests against Prism mock..."
$(NEWMAN) run $(COLL_FINAL) \
  --env-var baseUrl=http://localhost:$(PRISM_PORT) \
  --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(OVERRIDE_FILE)) \
  --reporters cli,html \
  --reporter-html-export $(REPORT_HTML)

## Alias: default test target
test: mock-test
```

## Example Usage

```
bash

make prism          # Starts Prism mock server
make mock-test      # Runs Newman tests against Prism mock
make postman-mock   # Runs Newman tests against Postman mock (URL stored in mock-url.txt)
make prism-stop     # Force-kills Prism server on default port
```

## Notes:

- `$(MOCK_URL_FILE)` should contain the Postman mock URL (e.g., `https://<mock-id>.mock.pstmn.io`).
- Your Postman environment override (`mock-env.json`) must include a token variable.
- You can automate writing the environment or mock URL file using a `postman-env-create` or `postman-mock-url-save` target if needed.

Would you like me to help generate those too?



what dir should the mockurlfile go

Great question.

The `mock-url.txt` file (or whatever filename you use) should ideally go in your `postman/` directory alongside your collections, environments, and override files, since:

- It's **Postman-specific**
- It's used by the `postman-mock` target
- It helps group all mock/test assets together

## Recommended Location

```
vbnet

project-root/
├── openapi/
│   └── c2m_openapi_spec_final.yaml
├── postman/
│   ├── Click2Mail.collection.merged.json
│   ├── mock-env.json
│   └── mock-url.txt      🖱️ Put it here
```

```
├── overrides.json
├── Makefile
└── ...
```

## 🔧 Update Makefile Variable

In your **Makefile**, make sure this line reflects the correct path:

```
makefile

MOCK_URL_FILE := postman/mock-url.txt
```

Let me know if you'd like a rule to **automatically generate or update that file** from the Postman API.



where does collection file go

The Postman **collection file** (e.g. `Click2Mail.collection.merged.json`) should go in the same **postman/** directory as your environments and mocks.

## ✅ Recommended Directory Structure

```
bash

project-root/
├── openapi/
│   └── c2m_openapi_spec_final.yaml    # Your OpenAPI spec
├── postman/
│   ├── Click2Mail.collection.merged.json # ← Collection file goes here
│   ├── mock-env.json                  # Environment for Newman tests
│   ├── overrides.json                  # Overrides if you're merging collections
│   └── mock-url.txt                    # URL for Postman mock server
├── Makefile
└── ...
```

## Why this is best:

- Keeps all Postman-related files together
- Works well with automation scripts (e.g., merge, lint, test)
- Keeps your project modular and clean

## 🔄 If you're auto-generating collections

Make sure your Makefile rules like `COLL_FINAL := postman/Click2Mail.collection.merged.json` match the actual output path.

Let me know if you want a **generate-collection** rule to go from OpenAPI → Postman Collection automatically.



```
how does this work? # --- generate ---
.PHONY: generate merge
generate: $(COLL_RAW) ## Generate Postman collection

$(COLL_RAW): $(SPEC)
    mkdir -p $(dir $@)
    $(GENERATOR) -s $(SPEC) -o $@ -p -O folderStrategy=Tags
```

Great — let's break down how this Makefile rule works:

## Purpose:

To **automate generating a Postman Collection** (`COLL_RAW`) from an OpenAPI spec (`SPEC`) using a CLI tool — typically [@apideck/postman-generator](#).

## Line-by-line Explanation:

```
makefile

# Declare that generate and merge are PHONY (i.e., not actual files)
.PHONY: generate merge
```

```
makefile

generate: $(COLL_RAW) ## Generate Postman collection
```

- This sets the **generate** target to depend on the `$(COLL_RAW)` file.
- When you run `make generate`, it will try to **build the raw collection file** if it doesn't exist or is outdated.

```
makefile

$(COLL_RAW): $(SPEC)
```

- This is an implicit rule that says:  
*"To build the `COLL_RAW` file, you need the `SPEC` file."*
- If `$(SPEC)` (your OpenAPI YAML) changes, it will **regenerate** the collection.

```
makefile

    mkdir -p $(dir $@)
```

- Ensures that the directory for the output file exists (i.e., `postman/`).

```
makefile
```

```
$(GENERATOR) -s $(SPEC) -o $$@ -p -0 folderStrategy=Tags
```

Assuming:

```
makefile
```

```
GENERATOR := npx @apideck/postman-generator
```

This runs the generator with the following options:

- **-s \$(SPEC)** → input OpenAPI spec file (e.g., `openapi/c2m_openapi_spec_final.yaml`)
- **-o \$\$@** → output file (\$@ is the target = `$(COLL_RAW)`)
- **-p** → prettify JSON output
- **-0 folderStrategy=Tags** → organizes collection folders by OpenAPI **tags** (neatly groups endpoints)

## ✓ Result:

This will generate a raw Postman collection (e.g. `Click2Mail.collection.json`) in your `postman/` directory, organized by tag folders.

## 🔄 Follow-up Suggestion:

If you also use `merge` to apply overrides, you can chain them like:

```
makefile
```

```
all: generate merge test
```

Let me know if you want help setting up the `merge` target as well.



so it builds a call for every endpoint in the spec?

Yes — exactly. When you run:

```
bash
```

```
make generate
```

...and your **Makefile** uses:

```
makefile
```

```
GENERATOR := npx @apideck/postman-generator
```

...it **parses your OpenAPI spec** and **generates one Postman request per defined operation**, like:

```
yaml

paths:
  /jobs/submit/single/doc:
    post:
      summary: Submit a single doc
```

➡ turns into a **POST request** named "Submit a single doc" in the generated Postman Collection.

## What Gets Generated

For each endpoint in the spec, the generator will:

Spec Element	Postman Output
<b>paths + method</b>	Request (GET, POST, PUT, etc.)
<b>summary / operationId</b>	Name of the request
<b>tags</b>	Folder (if using <b>folderStrategy=Tags</b> )
<b>requestBody / parameters</b>	Included in the request (with examples, if present)
<b>responses</b>	Used in test stubs (if supported/tested manually)

## Example Structure in Postman

If your spec has:

```
yaml

paths:
  /jobs/submit/single/doc:
    post:
      summary: Submit single doc
      tags: [Job Submission]

  /jobs/submit/multi/doc:
    post:
      summary: Submit multiple docs
      tags: [Job Submission]

  /status/{jobId}:
    get:
      summary: Check job status
      tags: [Status]
```

Your Postman Collection (organized by **Tags**) will look like:

```
lua

Click2Mail Collection
├── Job Submission
│   ├── Submit single doc
│   └── Submit multiple docs
└── Status
    └── Check job status
```

## Bonus: Examples Populate Body

If your OpenAPI spec includes example request bodies like:

```
yaml
requestBody:
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/JobRequest'
      example:
        jobId: "abc123"
        ...
```




The generated Postman request will **automatically include** that JSON in the body — saving you time.

Let me know if you'd like a rule to *auto-merge overrides* or *auto-generate example data*!



I want to generate a collection from a spec and then do some testing using prism, newman, and the postman mock server. HOW do i do all of this?

To fully automate your workflow — generating a Postman collection from an OpenAPI spec and running tests against:

1.  **Prism** (OpenAPI-based local mock server)
2.  **Postman mock server**
3.  **Newman** (test runner)

Here's a clean, working **Makefile-driven workflow** from start to finish.

## Directory Layout Assumptions

```
pgsql

project-root/
├── openapi/
│   └── c2m_openapi_spec_final.yaml
├── postman/
│   ├── Click2Mail.collection.json           # Generated from spec
│   ├── Click2Mail.collection.merged.json   # (optional) with overrides
│   ├── mock-env.json                       # Newman env with token, baseUrl
│   ├── mock-url.txt                        # Postman mock URL
│   └── overrides.json                      # Optional override template
└── Makefile
```

## Complete Makefile

```

makefile

# === CONFIG ===
SPEC          := openapi/c2m_openapi_spec_final.yaml
COLL_RAW      := postman/Click2Mail.collection.json
COLL_FINAL    := postman/Click2Mail.collection.merged.json
OVERRIDE_FILE := postman/overrides.json
ENV_FILE      := postman/mock-env.json
MOCK_URL_FILE := postman/mock-url.txt
PRISM_PORT    := 4010
REPORT_HTML   := newman-report.html

GENERATOR      := npx @apideck/postman-generator
MERGER         := node scripts/merge-postman.js
PRISM          := npx @stoplight/prism-cli
NEWMAN         := npx newman

# === TARGETS ===

.PHONY: all generate merge prism prism-stop mock-test postman-mock test

## Full test suite
all: generate merge test

## Generate Postman collection from OpenAPI
generate: $(COLL_RAW)

$(COLL_RAW): $(SPEC)
    mkdir -p $(dir $@)
    $(GENERATOR) -s $(SPEC) -o $@ -p -O folderStrategy=Tags
    @echo "✅ Generated Postman collection: $@"

## Merge collection with overrides (optional)
merge: $(COLL_FINAL)

$(COLL_FINAL): $(COLL_RAW) $(OVERRIDE_FILE)
    $(MERGER) -c $(COLL_RAW) -o $(OVERRIDE_FILE) -r $@
    @echo "✅ Merged collection saved: $@"

## Start Prism mock server
prism:
    @echo "🚀 Starting Prism on port $(PRISM_PORT)..."
    $(PRISM) mock $(SPEC) -p $(PRISM_PORT)

## Stop Prism if needed
prism-stop:
    @echo "🛑 Stopping Prism (if running)..."
    @lsof -ti tcp:$(PRISM_PORT) | xargs kill -9 || echo "No Prism running"

## Run Newman tests against Prism
mock-test:
    @echo "🧪 Testing against Prism @ localhost:$(PRISM_PORT)"
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=http://localhost:$(PRISM_PORT) \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

## Run Newman tests against Postman mock server
postman-mock:
    @echo "🧪 Testing against Postman Mock Server"
    $(eval MOCK := $(shell cat $(MOCK_URL_FILE)))
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=$(MOCK) \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

## Default test: runs against Prism
test: mock-test

```



## ✓ Usage

```
bash

make all          # Generates → Merges → Tests via Prism
make prism        # Start Prism manually
make test         # Run tests against Prism
make postman-mock # Run tests against Postman Mock
make prism-stop   # Kill Prism server
```

## 🔧 Notes

- `mock-env.json` should contain values like:

```
json

{
  "values": [
    { "key": "baseUrl", "value": "", "enabled": true },
    { "key": "token", "value": "mock-token", "enabled": true }
  ]
}
```

- `mock-url.txt` must be updated with your Postman mock server URL:

```
cpp

https://<mock-id>.mock.pstmn.io
```

- `scripts/merge-postman.js` is optional and can be replaced with:
  - `postman-overrides`
  - or simply skip if you're not customizing

Would you like a sample `mock-env.json` file and the content of `mock-url.txt` to get you started?






newman can run tests without a mock server, correct?

Yes, absolutely — **Newman can run tests without a mock server.**

## ✓ What Newman Does

Newman runs Postman collections against **any target URL**, including:

-  **Live production or staging APIs**
-  **Local dev servers** (e.g. localhost:3000)
-  **Mock servers** (optional, e.g. Prism or Postman mock)

Newman simply reads:

- the collection JSON (API definitions)
- the environment file (e.g. base URL, token)
- optional tests and assertions defined in the Postman requests

## Example: Run Against Live API

```
bash

newman run postman/Click2Mail.collection.merged.json \
  --env-var baseUrl=https://api.yourservice.com \
  --env-var token=your-token-here
```

✓ This will run all requests and assertions defined in your collection — no mocks required.

## Summary




Use Case	Mock Server Needed?
Testing live API endpoints	✗ No
Validating contract/test coverage during dev	✗ No
Simulating responses without backend running	✓ Yes (Prism/Postman)

Let me know if you'd like help writing or organizing your Newman tests inside your collection.



please generate the makefile the also runs newman tests without a mock server, runs newman tests against prism and runs newman tests against postman mock

Here's a clean, full-featured **Makefile** that:

- ✓ Generates a Postman collection from your OpenAPI spec
- ✓ Optionally merges with override files
- ✓ Runs **Newman tests**:
  -  Against your **live API**
  -  Against a **Prism mock server**
  -  Against a **Postman mock server**

# Makefile

```

makefile

# === CONFIGURATION ===

SPEC           := openapi/c2m_openapi_spec_final.yaml
COLL_RAW       := postman/Click2Mail.collection.json
COLL_FINAL     := postman/Click2Mail.collection.merged.json
OVERRIDE_FILE  := postman/overrides.json
ENV_FILE       := postman/mock-env.json
MOCK_URL_FILE  := postman/mock-url.txt
PRISM_PORT     := 4010
REPORT_HTML    := newman-report.html

GENERATOR      := npx @apideck/postman-generator
MERGER         := node scripts/merge-postman.js
PRISM          := npx @stoplight/prism-cli
NEWMAN         := npx newman

# === TARGETS ===

.PHONY: all generate merge prism prism-stop test-live test-prism test-postman test

## All steps: generate collection, merge overrides, run default test (Prism)
all: generate merge test

## Generate Postman collection from OpenAPI spec
generate: $(COLL_RAW)

$(COLL_RAW): $(SPEC)
    mkdir -p $(dir $@)
    $(GENERATOR) -s $(SPEC) -o $@ -p -O folderStrategy=Tags
    @echo "✅ Generated: $@"

## Merge collection with override (optional)
merge: $(COLL_FINAL)

$(COLL_FINAL): $(COLL_RAW) $(OVERRIDE_FILE)
    $(MERGER) -c $(COLL_RAW) -o $(OVERRIDE_FILE) -r $@
    @echo "✅ Merged with overrides: $@"

## Start Prism mock server
prism:
    @echo "🚀 Starting Prism on port $(PRISM_PORT)..."
    $(PRISM) mock $(SPEC) -p $(PRISM_PORT)

## Stop Prism mock server
prism-stop:
    @echo "🛑 Killing Prism if running..."
    @lsof -ti tcp:$(PRISM_PORT) | xargs kill -9 || echo "No Prism running"

## Run Newman tests against live API (use real baseUrl + token)
test-live:
    @echo "🌐 Running Newman tests against live API..."
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=https://api.yourdomain.com \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

## Run Newman tests against Prism mock server
test-prism:
    @echo "🧪 Running Newman tests against Prism..."
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=http://localhost:$(PRISM_PORT) \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

```

```
## Run Newman tests against Postman mock server
test-postman:
    @echo "📦 Running Newman tests against Postman mock server..."
    $(eval MOCK := $(shell cat $(MOCK_URL_FILE)))
    $(NEWMAN) run $(COLL_FINAL) \
        --env-var baseUrl=$(MOCK) \
        --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

## Default test target (uses Prism)
test: test-prism
```

## ✓ How to Use

Command	Description
<b>make generate</b>	Generate collection from OpenAPI spec
<b>make merge</b>	Apply override file to collection (optional)
<b>make prism</b>	Start Prism mock server
<b>make test-prism</b>	Run tests against Prism
<b>make test-live</b>	Run tests against real live API
<b>make test-postman</b>	Run tests against Postman mock server
<b>make all</b>	End-to-end: generate → merge → test (prism)

## 🔧 Example mock-env.json

```
json
{
  "values": [
    { "key": "baseUrl", "value": "", "enabled": true },
    { "key": "token", "value": "mock-token-123", "enabled": true }
  ]
}
```

## 📄 Example mock-url.txt




```
cpp
https://12345678.mock.pstmn.io
```

Let me know if you'd like a **make test-all** target that runs all 3 environments in sequence!



the api is not live yet

Perfect — since the API isn't live yet, your testing setup should focus on:

-  **Prism mock server** (OpenAPI contract-based local simulation)
-  **Postman mock server** (Postman's hosted mock environment)
-  **Newman test runs** against both mock environments

You can ignore `test-live` for now — or leave it as a placeholder for later.

## Adjusted Makefile (no live API yet)

Here's the simplified version without `test-live`, and with an optional `test-all` to run both Prism and Postman tests:

```
makefile

# === CONFIGURATION ===

SPEC           := openapi/c2m_openapi_spec_final.yaml
COLL_RAW       := postman/Click2Mail.collection.json
COLL_FINAL     := postman/Click2Mail.collection.merged.json
OVERRIDE_FILE  := postman/overrides.json
ENV_FILE       := postman/mock-env.json
MOCK_URL_FILE  := postman/mock-url.txt
PRISM_PORT     := 4010
REPORT_HTML    := newman-report.html

GENERATOR      := npx @apideck/postman-generator
MERGER         := node scripts/merge-postman.js
PRISM          := npx @stoplight/prism-cli
NEWMAN         := npx newman

# === TARGETS ===

.PHONY: all generate merge prism prism-stop test-prism test-postman test test-all

all: generate merge test

## Generate Postman collection from OpenAPI spec
generate: $(COLL_RAW)

$(COLL_RAW): $(SPEC)
    mkdir -p $(dir $@)
    $(GENERATOR) -s $(SPEC) -o $@ -p -O folderStrategy=Tags
    @echo "✅ Generated: $@"

## Merge collection with override (optional)
merge: $(COLL_FINAL)

$(COLL_FINAL): $(COLL_RAW) $(OVERRIDE_FILE)
    $(MERGER) -c $(COLL_RAW) -o $(OVERRIDE_FILE) -r $@
    @echo "✅ Merged with overrides: $@"

## Start Prism mock server
prism:
    @echo "🚀 Starting Prism on port $(PRISM_PORT)..."
    $(PRISM) mock $(SPEC) -p $(PRISM_PORT)

## Stop Prism mock server
prism-stop:
    @echo "🛑 Killing Prism if running..."
    @lsof -ti tcp:$(PRISM_PORT) | xargs kill -9 || echo "No Prism running"

## Run Newman tests against Prism
test-prism:
    @echo "🧪 Running Newman tests against Prism..."
```

```

$(NEWMAN) run $(COLL_FINAL) \
  --env-var baseUrl=http://localhost:$(PRISM_PORT) \
  --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
  --reporters cli,html \
  --reporter-html-export $(REPORT_HTML)

## Run Newman tests against Postman mock server
test-postman:
@echo "👉 Running Newman tests against Postman mock server..."
$(eval MOCK := $(shell cat $(MOCK_URL_FILE)))
$(NEWMAN) run $(COLL_FINAL) \
  --env-var baseUrl=$(MOCK) \
  --env-var token=$(shell jq -r '.variable[]|select(.key=="token").value'
$(ENV_FILE)) \
  --reporters cli,html \
  --reporter-html-export $(REPORT_HTML)

## Default test: use Prism
test: test-prism

## Run both Prism and Postman mock tests
test-all: test-prism test-postman
@echo "✅ All mock tests completed"

```

## ✅ Example Usage

```

bash

make prism          # Start Prism server
make test           # Test against Prism
make test-postman   # Test against Postman mock
make test-all       # Run both sets of tests

```

Let me know if you'd like a target that waits for Prism to be ready before running the test.



what about running newman against the actual local collection?

Great follow-up!

You **can absolutely run Newman directly against your local Postman collection** (no mock server needed), as long as the requests in that collection point to valid URLs — for example:

- Prism mock server (<http://localhost:4010>)
- Postman mock server (<https://<mock-id>.mock.pstmn.io>)
- Eventually: your real API (once it's live)

## ✅ Target to Run Against Local Collection

Add this target to your **Makefile**:

```

makefile

## Run Newman tests directly using local collection and env
test-local:

```

```
@echo "📁 Running Newman tests on local collection using env values..."
$(NEWMAN) run $(COLL_FINAL) \
  --environment $(ENV_FILE) \
  --reporters cli,html \
  --reporter-html-export $(REPORT_HTML)
```

This uses:

- `$(COLL_FINAL)` → your generated Postman collection file
- `$(ENV_FILE)` → a full Postman environment JSON (can include `baseUrl`, `token`, etc.)

## ✓ Sample mock-env.json (Postman environment format)

```
json

{
  "id": "c2m-env-id",
  "name": "C2M Local Dev",
  "values": [
    { "key": "baseUrl", "value": "http://localhost:4010", "enabled": true },
    { "key": "token", "value": "mock-token", "enabled": true }
  ],
  "_postman_variable_scope": "environment",
  "_postman_exported_at": "2025-07-16T12:00:00Z",
  "_postman_exported_using": "Postman/10.23.0"
}
```

## ✓ Example Command

```
bash

make test-local
```

This will:

- Use your local collection
- Substitute the environment variables (`baseUrl`, `token`)
- Run all request + test scripts in Postman
- Output a CLI and HTML report

Let me know if you'd like a target that dynamically **sets the base URL via --env-var instead** (useful when not using a full Postman environment JSON).