

# 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).



so newman cannot do anything unless it is pointing to a mock server or the collection file requests are pointing to real urls

Exactly — you've got it.

## ✓ How Newman Works

Newman executes requests **exactly as they appear in your collection**, meaning:

- Each request must have a valid, reachable URL — real or mock.
- It doesn't "simulate" responses — it **sends real HTTP requests**.

## ✗ What Newman doesn't do:

- Newman does **not mock** requests itself.
- It won't "stub" or "simulate" responses if your server isn't running.
- If your `baseUrl` is something like `http://localhost:4010` and **nothing is listening on that port**, requests will fail.

## ✓ What You Can Do:

Scenario	What to Point baseUrl To
API not implemented yet	Prism mock server ( <code>http://localhost:4010</code> )
Hosted mock needed	Postman mock server ( <code>https://xxxx.mock.pstmn.io</code> )
Testing live service	Your actual staging/prod URL
Static example testing only	Use Prism + OpenAPI <b>example:</b> blocks

## ↺ If You Don't Want to Spin Up a Server

If you want **mock behavior without spinning up Prism**, use:

- **Postman mock server** (easy, hosted)
- Or Prism via `make prism` for fast local mocking

Newman + Postman mocks or Prism gives you a realistic test environment even before the backend is built.

Let me know if you want help generating a minimal OpenAPI spec + example block to feed into Prism or Postman for simulated responses!



Static example testing only      Use Prism + OpenAPI example: blocks

Perfect — this is a great approach when your backend isn't live yet. Let's walk through how to do **static example-based testing** using:

- ✓ **Prism** (mock server that returns **example:** values from your OpenAPI spec)
- ✓ **Newman** (running your Postman collection against the Prism server)



## ✓ How Static Example Testing Works

### 🔄 Flow:

1. You define **example:** or **examples:** in your OpenAPI spec
2. Prism reads your OpenAPI file and **serves responses from those examples**
3. Newman sends real HTTP requests to Prism (e.g., `localhost:4010`)
4. Newman tests validate the responses against expectations

## ✓ 1. Add Examples to Your OpenAPI Spec

Here's an example for a **POST** `/jobs/submit/single/doc` endpoint:

```
yaml

paths:
  /jobs/submit/single/doc:
    post:
      summary: Submit single doc
      tags: [Job Submission]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Job'
            example:
              jobId: "abc123"
              printOption: "color"
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Job'
              example:
                jobId: "abc123"
                status: "submitted"
```

✓ Prism will now return:

- The request example as a default body when you generate Postman requests (if you're using `@apideck/postman-generator`)
- The response example (`status: submitted`) as its response payload

## ✓ 2. Start Prism Mock Server

```
bash

make prism
```

This runs:

```
bash

npx @stoplight/prism-cli mock openapi/c2m_openapi_spec_final.yaml -p 4010
```

### ✓ 3. Point Your Postman Environment at Prism

In `mock-env.json`:

```
json

{
  "values": [
    { "key": "baseUrl", "value": "http://localhost:4010", "enabled": true }
  ]
}
```

Each Postman request should use `{{baseUrl}}` in the URL.

### ✓ 4. Run Newman Against Prism

```
bash

make test-prism
```

Or manually:

```
bash

npx newman run postman/Click2Mail.collection.merged.json \
  --environment postman/mock-env.json \
  --reporters cli
```

You'll get **real HTTP responses, generated from static OpenAPI examples** — perfect for contract-first testing and debugging without any backend code running.

### ✓ Summary

Tool	Purpose
OpenAPI	Define request/response schemas + examples
Prism	Serve the example responses as a local mock server
Postman	Define and organize API requests/tests
Newman	Run the Postman tests via CLI

Let me know if you want me to scan your OpenAPI and help auto-insert good **example:** blocks!



what is the difference between these two:

```
GENERATOR    := npx @apideck/postman-generator
# GENERATOR  := npx openapi-to-postmanv2
```

Great question — both tools convert OpenAPI specs into Postman collections, but they differ in **features, output quality, structure, and use cases**.

## 1. @apideck/postman-generator

```
makefile

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

### Pros:

- **Cleaner, production-grade collections** with:
  - Organized folders (e.g., by tag or path)
  - Smart request names (from **summary** or **operationId**)
- Supports **folderStrategy**, **auth**, and override merging
- Can output pretty-formatted JSON
- Integrates well into CI pipelines
- Designed specifically for developer experience and workflow automation

### Best For:

- Contract-first development
- Internal dev teams
- Clean, readable Postman UX

### Example:

```
bash

npx @apideck/postman-generator -s openapi.yaml -o postman.json -O folderStrategy=Tags
```

## 2. openapi-to-postmanv2

```
makefile

GENERATOR := npx openapi-to-postmanv2
```

## ✓ Pros:

- Official conversion library used by Postman itself
- Highly configurable (lots of flags for backward compatibility)
- Good for very complex specs or debugging edge cases
- Supports OpenAPI v2 & v3

## ⚠ Cons:

- Output is more **mechanical** — names like **POST /users/:id**, fewer UX niceties
- May need **manual cleanup** in Postman UI (folder names, request names, etc.)
- Doesn't offer the same folder grouping or override merging

## ✓ Best For:

- Compatibility testing
- Converting legacy specs
- Using Postman's internal tooling

## 🔧 Example:

```
bash

npx openapi-to-postmanv2 -s openapi.yaml -o postman.json -p -0 folderStrategy=Tags
```

## 🧠 Summary

Feature	@apideck/postman-generator	openapi-to-postmanv2
Output structure (folders, names)	✓ Clean & friendly	⚠ More raw or generic
Supports overrides	✓ Yes	✗ No
Folder strategies (by tag/path)	✓ Yes	⚠ Experimental
Used by Postman CLI	✗ No	✓ Yes (internal tool)
Use case	Dev-friendly automation	Compatibility & low-level

## ✓ Recommendation for You:

Since you're doing:

- Contract-first dev (API not live)
- Static + mock testing
- Using Makefiles and automation

👉 **Stick with @apideck/postman-generator** — it's designed for exactly your workflow.

Let me know if you want both installed to compare outputs side-by-side.



so to get any use out of prism or the postman mock server I need newman to send requests?

Not necessarily — let's clarify the roles of **Prism**, **Postman mock server**, and **Newman**, and how they can be used **together or independently**:



## Prism and Postman Mock Servers: What They Do

Both **Prism** and the **Postman mock server** can:

- ✓ **Simulate a running backend API**
- ✓ Respond to HTTP requests using OpenAPI or Postman examples
- ✓ Be used manually (via browser, curl, Postman UI) or automatically (e.g. Newman)



### You can use Prism or Postman mock servers:

#### Manually:

- In Postman app or curl:
  - GET `http://localhost:4010/jobs/submit/single/doc`
  - GET `https://mock.pstmn.io/yourpath`



#### Automatically:

- In **Newman** test automation

So they're **not dependent on Newman**. You can use them standalone.



## Newman: What It Does

Newman runs tests from Postman collections. It's **not required** to use Prism or Postman mocks — but it's the tool that **lets you automate** testing against them.



## Summary

Tool	Needs Newman?	Use Cases
Prism	✗ No	Run local mock server based on OpenAPI
Postman mock	✗ No	Hosted mock server using Postman collection
Newman	✓ Needs a URL to hit	Automate tests against real or mock APIs

## ✓ Example Workflows:

### 1. Manual testing (no Newman):

- Spin up Prism → test in Postman
- Create Postman mock → test in browser or Postman

### 2. Automated testing:

- Use Newman to hit:
  - Prism: `http://localhost:4010`
  - Postman mock: `https://abc.mock.pstmn.io`

Let me know and I can give you a quick shell script or Postman collection example that tests Prism manually without using Newman.



OK - can you now list all the tasks I need to complete to stand up new, the postman mock server, prism and all the collections and everything else I need and what reports I get

Absolutely. Here's a **complete checklist** for standing up your full API testing workflow using:

- ✓ **OpenAPI spec**
- ✓ **Postman mock server**
- ✓ **Prism (local mock server)**
- ✓ **Postman collections**
- ✓ **Newman (automated tests & reports)**

## ✓ 1. OpenAPI Spec

### Task

- Write a valid OpenAPI 3.x spec (`.yaml` or `.json`)
- Include:
  - `paths` with all operations (`GET`, `POST`, etc.)
  - `components/schemas` for request/response bodies
  - `example:` or `examples:` in responses (for Prism/Postman mocks)

### Output

- `openapi/c2m_openapi_spec_final.yaml`

## ✓ 2. Generate Postman Collection

### Task

- Use `@apideck/postman-generator` to convert OpenAPI → Postman collection

### Makefile Target

```
bash

make generate
```

### Output

- `postman/Click2Mail.collection.json`

## ✓ 3. Merge Overrides (Optional)

### Task

- Define `overrides.json` if you want to rename requests, add auth, etc.
- Use `merge-postman.js` or custom script to merge

### Makefile Target

```
bash

make merge
```

### Output

- `postman/Click2Mail.collection.merged.json`

## ✓ 4. Postman Mock Server Setup

### Task

- In Postman:
  1. Import your collection
  2. Click "Mock Server" → Create Mock
  3. Save the URL in `postman/mock-url.txt`

### Output

- Hosted mock server at `https://xxxx.mock.pstmn.io`

- File: `postman/mock-url.txt`

## ✓ 5. Prism Mock Server (Local)

### Task

- Start Prism using OpenAPI spec

### Makefile Target

```
bash

make prism
```

### Output

- Local mock server at `http://localhost:4010`

## ✓ 6. Create Postman Environment

### Task

- Create `mock-env.json` with `baseUrl` and any tokens

### Example

```
json

{
  "values": [
    { "key": "baseUrl", "value": "http://localhost:4010", "enabled": true },
    { "key": "token", "value": "mock-token", "enabled": true }
  ]
}
```

### Output

- `postman/mock-env.json`

## ✓ 7. Run Newman Tests

### Task

- Use Newman to run tests against mock servers

### Makefile Targets



```
bash

make test-prism      # Runs against Prism
make test-postman    # Runs against Postman mock
make test-local      # Uses environment JSON
make test-all        # Runs both Prism and Postman tests
```

## ✓ 8. Reports and Output

### Outputs from Newman

- CLI output: assertion success/failure
- HTML report:
  - `newman-report.html` (view in browser)
- Optional: JUnit/XML, JSON, or CI-compatible formats

## ✓ Bonus: Add these for CI/CD integration

- GitHub Actions or GitLab pipeline to:
  - Validate OpenAPI spec
  - Auto-generate Postman collection
  - Run Newman tests and fail build if assertions fail
  - Upload `newman-report.html` as a build artifact

## ✓ Summary of What You'll Have

Component	File / Result
OpenAPI spec	<code>openapi/c2m_openapi_spec_final.yaml</code>
Postman collection	<code>postman/Click2Mail.collection.merged.json</code>
Postman mock server	<code>postman/mock-url.txt</code>
Prism mock server	<code>http://localhost:4010</code>
Postman environment	<code>postman/mock-env.json</code>
Test runner	<code>make test-*</code> targets
Reports	<code>newman-report.html</code>

Would you like me to generate a README with all these steps documented for your project?



yes please






Here's a project-friendly **README.md** file that documents **all the setup and usage instructions** for your OpenAPI-to-Postman-to-Prism/Newman testing stack.

## README.md — API Testing Workflow

markdown

### # 🚀 Click2Mail API Testing & Mocking Framework

This project supports a contract-first API development workflow using:

-  OpenAPI 3.x spec
-  Postman collections & mock server
-  Prism local mock server
-  Newman CLI test runner
-  Makefile automation

---

### ## 🏠 Project Structure

project-root/

```
├── openapi/
│   └── c2m_openapi_spec_final.yaml # OpenAPI spec
├── postman/
│   ├── Click2Mail.collection.json # Generated raw collection
│   ├── Click2Mail.collection.merged.json # (Optional) collection with overrides
│   ├── mock-env.json # Environment (baseUrl, token)
│   ├── mock-url.txt # Postman mock URL
│   └── overrides.json # (Optional) overrides
├── newman-report.html # Newman test report
├── Makefile
└── ...
```

yaml

---

### ## 🚀 Setup Instructions

#### ### 1. Generate Postman Collection from OpenAPI

```
```bash
make generate
```

Generates `postman/Click2Mail.collection.json`

## 2. (Optional) Merge Overrides into Collection

If you're customizing names, auth headers, etc.:

```
bash

make merge
```

Generates `postman/Click2Mail.collection.merged.json`

### 3. Set Up Postman Mock Server

In the Postman app:

1. Import the collection
2. Click **Mock Server** → **Create mock server**
3. Save the URL to `postman/mock-url.txt`

### 4. Start Local Prism Server

```
bash

make prism
```

Starts local mock server at `http://localhost:4010`

To stop it:

```
bash

make prism-stop
```

### 5. Configure Postman Environment

Create `postman/mock-env.json`:

```
json

{
  "values": [
    { "key": "baseUrl", "value": "http://localhost:4010", "enabled": true },
    { "key": "token", "value": "mock-token", "enabled": true }
  ]
}
```



## Running Tests with Newman

### Against Prism (local mock)

```
bash

make test-prism
```

---

## Against Postman mock server

```
bash

make test-postman
```

---

## Against current collection + env (local)

```
bash

make test-local
```

---

## Run all mocks


```
bash

make test-all
```

---

## Test Reports

After each run, Newman generates:

-  `newman-report.html` — open in your browser for full results




---

## Tools Used

- [@apideck/postman-generator](#)
- [Prism](#) – OpenAPI mock server
- [Newman](#) – CLI runner
- [Postman](#) – API client & mock hosting

---

## Coming Soon

-  GitHub Actions for CI test runs
-  OpenAPI linting & validation
-  Swagger & Redoc documentation previews

yaml

---

Would you like me to also generate a `Makefile` scaffold + `mock-env.json` + example OpenAPI snippet so you can bootstrap this from scratch?



yes please

Here you go — a complete starter kit to **bootstrap your OpenAPI** → **Postman** → **Prism** → **Newman workflow** from scratch.

## ✓ Files Included:

File	Purpose
<b>Makefile</b>	Automates generation, mocking, and testing
<b>mock-env.json</b>	Postman environment with <b>baseUrl</b> , <b>token</b>
<b>c2m_openapi_spec_final.yaml</b>	Sample OpenAPI 3.0.3 spec
<b>overrides.json</b>	Sample override template
<b>README.md</b>	Full documentation (already provided)

## 1. 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-prism test-postman test-local test test-all

**all:** generate merge test

**generate:** \$(COLL\_RAW)

\$(COLL\_RAW): \$(SPEC)

mkdir -p \$(dir \$@)

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

@echo "✓ Generated: \$@"

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

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

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

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

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

test-local:
    @echo "📁 Running tests using local env file..."
    $(NEWMAN) run $(COLL_FINAL) \
        --environment $(ENV_FILE) \
        --reporters cli,html \
        --reporter-html-export $(REPORT_HTML)

test: test-prism
test-all: test-prism test-postman
```

## 2. postman/mock-env.json

```
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"
}
```

## 3. postman/overrides.json

```
json

{
  "auth": {
```

```

    "type": "bearer",
    "bearer": [
      { "key": "token", "value": "{{token}}", "type": "string" }
    ]
  },
  "info": {
    "name": "Click2Mail API (Merged)"
  }
}

```

## 4. openapi/c2m\_openapi\_spec\_final.yaml

```

yaml

openapi: 3.0.3
info:
  title: Click2Mail Submission API
  version: 1.0.0
paths:
  /jobs/submit/single/doc:
    post:
      summary: Submit a single document
      tags: [Job Submission]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/JobRequest'
            example:
              documentId: "abc123"
              recipient: "Jane Doe"
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/JobResponse'
              example:
                jobId: "abc123"
                status: "submitted"

components:
  schemas:
    JobRequest:
      type: object
      properties:
        documentId:
          type: string
        recipient:
          type: string
    JobResponse:
      type: object
      properties:
        jobId:
          type: string
        status:
          type: string

```

## Next Steps

1. `npm install -g @apideck/postman-generator @stoplight/prism-cli newman`

2. `make all`

3. Run:

- `make prism`
- `make test-prism`
- `make test-postman` (after creating a Postman mock)

Let me know if you want this zipped or turned into a GitHub repo template.



when I run prism from makefile it takes over the terminal and makefile cannot contijued

You're absolutely right — by default, `prism mock` runs in the **foreground**, which blocks further execution in your **Makefile**.

To fix this, you have two good options:

## ✓ Option 1: Run Prism in the Background (Recommended for Local Dev)

Update your **Makefile** `prism` target to run Prism as a background process:

```
makefile

prism:
  @echo "🚀 Starting Prism in background on port $(PRISM_PORT)..."
  $(PRISM) mock $(SPEC) -p $(PRISM_PORT) > prism.log 2>&1 &
  @sleep 1
  @echo "🔗 Prism running at http://localhost:$(PRISM_PORT)"
```

### What this does:

- `&` → runs Prism in the background
- `> prism.log 2>&1` → sends stdout + stderr to a log file
- `sleep 1` → gives it a second to start before you continue
- `make` continues with the next command

🔄 You can now chain `make prism && make test-prism` or even automate `test-prism` to start Prism first.

## ✓ Option 2: Run Prism in a Separate Terminal or Pane

Just run Prism manually in a separate terminal window:

```
bash

make prism
```



Then in another terminal:

```
bash

make test-prism
```

Use this if you want to inspect logs in real time or don't need full automation yet.

## **Optional: Stop Prism Automatically**

You can stop any Prism instance on the default port like this:

```
makefile

prism-stop:
    @echo "● Stopping Prism..."
    @lsof -ti tcp:${PRISM_PORT} | xargs kill -9 || echo "No Prism running"
```

## **Bonus: One-Step Mock + Test**

If you want a single `make run-test-prism` that:

1. Starts Prism
2. Waits a moment
3. Runs Newman tests
4. Stops Prism

Let me know — I can generate that too.