

**JAMIU
SALIMON**

CONTACT:
[LINKEDIN](#)
[MEDIUM](#)

SYNTHETIC TESTING: THE ART OF FAUX TESTING

CONTEXT



Synthetic testing is also known as;

Semantic Testing
Synthetic Monitoring Testing
Active/Proactive Monitoring Testing



**Popular monitoring approaches like
RUM & APM are reactive/passive in
nature**

Issues are only raised after real users have
encountered them



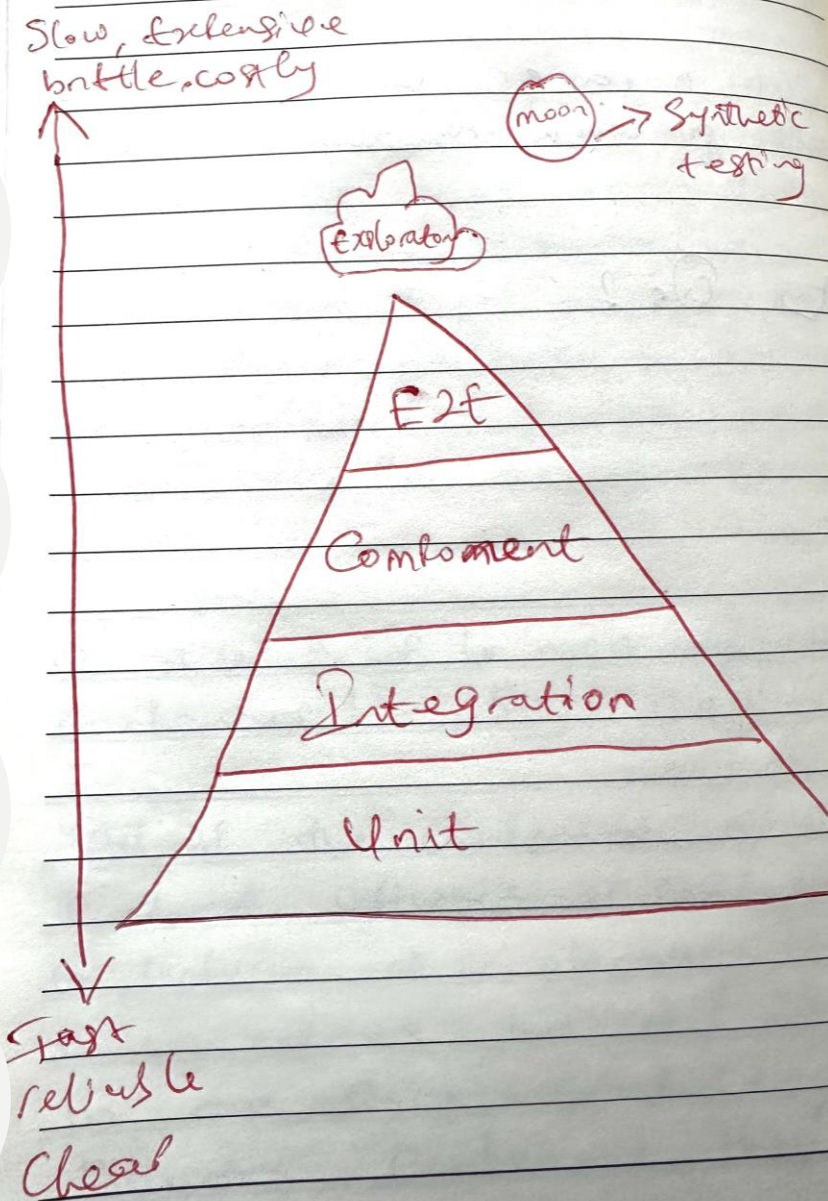
**Synthetic testing sought to alleviate
this shortcoming by proactively
testing & monitoring**

CONTEXT

- Definition of synthetic testing (Datadog, Microsoft, & MartinFowler.com)
 - Focus Area:
 - Absolute critical paths – Key user journeys & business functionalities
 - Performance
 - Input Data – Simulated real user data
 - Where (Live):
 - **Production**
 - *Production-like*
 - Outcome:
 - Detecting failing business requirements
 - Identify functional & performance issues
 - Continuously reiterate product's good health & resilience
 - Notify app owners when an issue is encountered (relies on app's observability)

CONTEXT

- Intersection of testing & observability
 - Agile testing manifesto
- Where does this fall in our testing pyramid?
 - Small subset of E2E
 - Envisioned as “the moon” – shines light on any gap in testing done earlier during development
 - Most expensive in the testing pyramid
 - Less of synthetic testing, more of the rest
 - Feeds from the rest



CLASSES OF SYNTHETIC TESTING

Based on app layer under test

- Browser testing
- API testing - Different flavours depending on info exchange protocol used by app e.g., HTTP, SSL, TCP, DNS, ...

Based on aspects of the app under test

- Availability testing - e.g., health checks
- Performance testing – Different flavours e.g, load testing, stress testing ...
- Transaction flow testing - ensure reliability of a feature flow which can involve multiple components
- User journey testing – ensure the reliability of specific user journeys which can involve a single component or multiple components or transactions
- Smoke testing – Regression or backwards compatibility testing for stability

3 W's - WHERE, WHAT & WHICH

Prod environment – READ
ops only

- Simple health checks
- Monitor uptime of critical functionality

Prod-like environment -
READ & WRITE ops

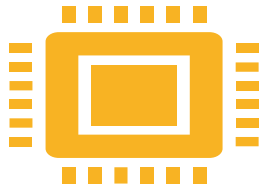
- Same as prod env w/ more flexibility
- Simulating entering new markets – e.g., Opening up functionality to new line of business
- Monitor 3rd party APIs
- Integrate into deployment pipeline for quicker & consistent feedback

*See table in next slide showing
some recommendations*

3 W'S - WHERE, WHAT & WHICH

	API testing	Browser testing	Availability testing	Performance testing	Transaction flow testing	User journey testing	Smoke testing
Prod	R	R	R	-	R	R	R
Prod-like	RW	RW	R	RW	RW	RW	RW

HOW TO



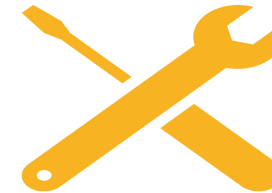
Tools that can be used to set up synthetic testing with ease are as follows;

API

- Datadog synthetic API testing
- Postman (SDK for securely integrating into deployment pipeline)
- AWS CloudWatch Synthetics Canary Blueprints
- Spring Boot Actuators (Need to be used in conjunction w/ some of the other tools)
- JMeter
- Wrk

UI

- AWS CloudWatch Synthetics Canary Blueprints
- Cypress
- Selenium



Review some of these tools - TBC

DEMO



DD Synthetic testing....



AWS CloudWatch Synthetics
Canary Blueprints...

DEMO – DD SYNTHETIC TESTING

Define request

1. In the Datadog site, hover over **Digital Experience** and select **Tests** (under **Synthetic Monitoring & Testing**).
2. Click **New Test** > **New API test**.
3. Select the **HTTP** request type.
4. Define your request:
 - Add the URL of the endpoint you want to monitor. If you don't know what to start with, you can use `https://www.shopist.io/`, a test e-commerce web application. Defining the endpoint to test automatically populates the name of your test to `Test on www.shopist.io`.
 - You can select **Advanced Options** to set custom request options, certificates, authentication credentials, and more.
Note: You can create secure global variables to store credentials and create local variables to generate dynamic timestamps to use in your request payload. After creating these variables, type `{{` in any relevant field and select the variable to inject its value in your test options. In this example, no specific advanced option is needed.
 - You can set tags such as `env:prod` and `app:shopist` on your test. Tags allow you to keep your test suite organized and quickly find tests you're interested in on the homepage.

5. Click **Test URL** to trigger a sample test run.

Synthetic > New API Test

1 Choose request type

2 Define request

URL

GET https://www.shopist.io

Advanced Options

Name

Test on www.shopist.io

Environment (env)

env:prod

Additional Tags

app:shopist

Test URL

Define assertions

Clicking **Test URL** automatically populates basic assertions about your endpoint's response. Assertions define what a successful test run is.

In this example, three default assertions populate after triggering the sample test run:

automatically populates the name of your test to `Test on www.shopist.io`.

- You can select **Advanced Options** to set custom request options, certificates, authentication credentials, and more.
Note: You can create secure global variables to store credentials and create local variables to generate dynamic timestamps to use in your request payload. After creating these variables, type `{{` in any relevant field and select the variable to inject its value in your test options. In this example, no specific advanced option is needed.
- You can set tags such as `env:prod` and `app:shopist` on your test. Tags allow you to keep your test suite organized and quickly find tests you're interested in on the homepage.

5. Click **Test URL** to trigger a sample test run.

Synthetic > New API Test

1 Choose request type

2 Define request

URL

GET https://www.shopist.io

Advanced Options

Name

Test on www.shopist.io

Environment (env)

env:prod

Additional Tags

app:shopist

Test URL

Define assertions

Clicking **Test URL** automatically populates basic assertions about your endpoint's response. Assertions define what a successful test run is.

In this example, three default assertions populate after triggering the sample test run:

3 Define assertions

Your test is successful:

When status code is 200 PASSED

And header content-type is text/html PASSED

And response time is less than 2000 ms PASSED

New Assertion

Assertions are fully customizable. To add a custom assertion, click on elements of the response preview such as the headers or click **New Assertion** to define a new assertion from scratch.

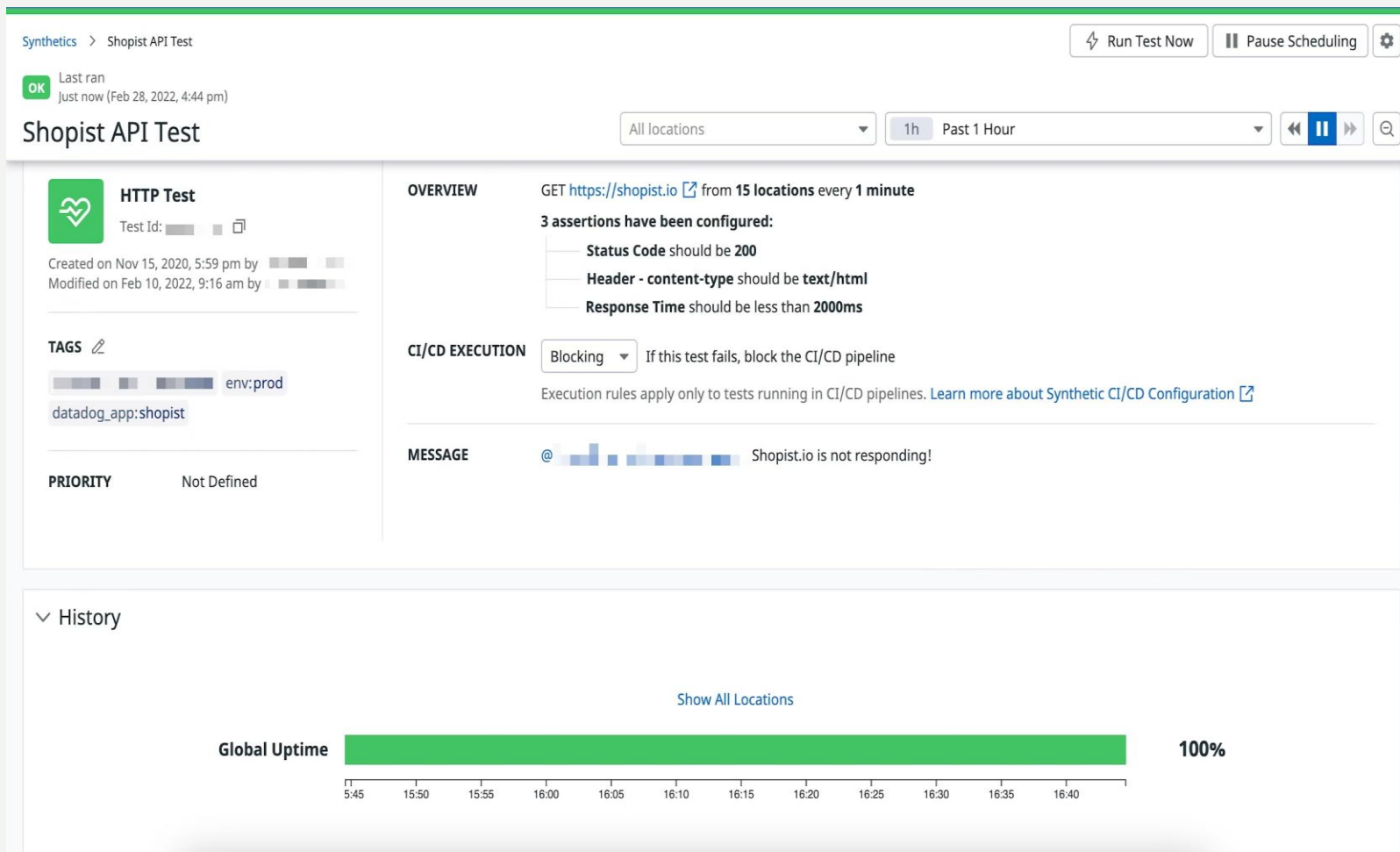
Define assertions

Response Preview

Documentation

Variables

DEMO – DD SYNTHETIC TESTING



DEMO - AWS CLOUDWATCH SYNTHETICS CANARY BLUEPRINTS

CloudWatch > Synthetics Canaries > Create a canary

Create canary [Info](#)

**Use a blueprint**
Work from a template script

**Inline Editor**
Edit inline or upload your own scripts

**Import from S3**
Use existing scripts from S3

Blueprints

☐ **Heartbeat monitoring**
Run a basic page load on a single URL.

☒ **API canary**
Monitor your APIs as HTTP steps.

☐ **Broken link checker**
Run a basic web crawler on designated URL.

☐ **Canary Recorder**
Use the AWS Canary Recorder plugin.

☐ **GUI workflow builder**
Create a GUI workflow with actions to perform.

☐ **Visual monitoring**
Monitor visual changes for every run

Canary builder

Name

test_canary

A name consists of up to 255 lowercase letters, numbers, hyphens or underscores with no spaces.



Using an Amazon API Gateway API [Info](#)

» Check this box if you are using an Amazon API Gateway API. Canaries have additional options specific to API Gateway, like uploading your Swagger template.

☐ I'm using an Amazon API Gateway API

HTTP requests (1)

Add your HTTP requests as steps. Drag and drop to change order.

Edit

Delete

Add HTTP request

	Call Order	Step name	Endpoint	Method
<input type="radio"/>	1	Verify us-east-...	https://test.com/content/1	GET

Script editor

Undo

Clear editor



Runtime version [Info](#)

syn-nodejs-puppeteer-9.0

Choose a synthetics runtime version to execute this canary.

```
1 const synthetics = require('Synthetics');
2 const log = require('SyntheticsLogger');
```

BENEFITS & DRAWBACKS

Benefits

- Shifts focus from MTBF to MTTR
- Minimizes MTTR
- Confidently expand to new markets & regions
- Achieve performance objective –
 - Makes it easier to adhere & maintain system OKRs, SLOs, & SLAs.
- Informs performance objectives –
 - AWS Observability motto – “Remember to build, measure, learn, repeat!”
 - Set sensible expected baseline for app’s current state
 - Good benchmark metrics isn’t set in stone – Metrics need to evolve!
- More regular releases – A must have to drive towards daily deploys

Drawbacks

- **Effort vs Value:** UI/Browser tests can be quite brittle. Upfront effort to set up test might not necessarily yield long term benefits if the UI changes ever so often.
- Cost vs Value

CONCLUSION

From AWS Serverless badge –
“When you’re operating your serverless applications at scale, you can’t afford to fly blind. You need to be able to answer important operational and business questions”

It shouldn’t be a case of passive
VS active monitoring but
combination of both gain more
insight into these questions

Reaffirms app-level operational
goals

Most dev teams already abit of
synthetic testing implemented for
their apps. This can be taken further.

Healthy \neq Functional

Why wait for the user to
uncover this for you? Be
proactive!

ANY QUESTIONS ?

RESOURCES

- My blog post on synthetic testing - <https://dkmdebugin.medium.com/synthetic-testing-the-art-of-faux-testing-429379b3bee2>
- Datadog docs on synthetic testing - <https://docs.datadoghq.com/synthetics/>
- AWS docs on AWS Cloudwatch canary blueprint - https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Synthetics_Canaries_Blueprints.html#CloudWatch_Synthetics_Canaries_Blueprints_GUI_Workflow