

## Make Integration Testing easy for Developers & Agile Teams

Discover proven strategies to eliminate integration failures in your apps & services



WHITEPAPER



### **TABLE OF CONTENTS**

### WHAT MAKES INTEGRATION TESTING DIFFICULT FOR DEVELOPERS?

3-5

Downstream changes ► Upstream failures Writing integration tests harder than development Collaboration in multi-service env is not easy Debugging nightmare

#### WHY UNIT TESTS ARE NOT ENOUGH? 6-7

Unit Tests have limited scope High coverage Dow quality Incomplete Testing No coverage for validations

2

3

INTEGRATION TESTING MADE EASY WITH 8-9 HYPERTEST

## What Makes Integration Testing Difficult for Developers?

The hidden cost of microservices is managing integration failures All modern applications are distributed. This design helps accelerate development and maintain performance, reliability and fault tolerance at scale.

But this exposes modern applications to the risk of integration failures i.e. issues and downtimes of failing because interactions between your code and its dependencies.

Consider incorrect database calls, failing inter-service or 3rd party API contracts and erroneous async operations.

Achilles Heel for all modern distributed backends



# 01

### DOWNSTREAM CHANGES ➡ UPSTREAM FAILURES

Frequent service changes lead to unexpected upstream failures. This becomes even harder to predict with multiple dependencies or unclear dependency chains, causing chaos for developers



### WRITING INTEGRATION TESTS HARDER THAN DEVELOPMENT

2 problems: Devs mostly write unit tests, which are never enough. Second, testing integrations means simulating or mocking dependencies which is never easy to get right



### COLLABORATION IN MULTI-SERVICE SETUP NOT EASY

Very hard for devs to collaborate in a large team over changes that involve multiple services. What it gives? Larger teams and too many services



NIGHTMARE

Debugging these issues is difficult for developers. Tracing the failing request through multiple services lacks any easy way. Most devs struggle to pinpoint the origin of the problem



Unit tests are useful for checking the logic within a service but fail to test the dependencies between services.

Too much reliance on unit tests has its risks:

### UNIT TESTS HAVE LIMITED SCOPE

Microservices interact with databases, services, queues, and APIs. Unit tests focus on isolated code fragments but not these interactions. For services, a broad testing scope is crucial.

### HIGH COVERAGE → LOW QUALITY

Dev writes a test for adding numbers, but only checks if the output is a number, not the correct sum. This lowquality test covers the function but misses a bug.

### **INCOMPLETE TESTING**

Unit tests miss interaction failures. These, often the trickiest issues, can cause data loss, slowdowns, or production crashes.

### NO COVERAGE FOR VALIDATIONS

Validations check logical boundaries but tests that cover all the lines but poorly validate these boundaries still end up reporting high coverage.

### Integration Testing made easy for Developers with HyperTest

## 99

HyperTest is a game changer for us in integration testing, It has significantly saved time and effort by green-lighting changes before they go live with our weekly releases.

**Vinay Jaasti** CTO, Airmeet

Explore 🗹 🔥 HYPERTEST

HyperTest captures real interactions between code and external components using actual application traffic, then converted into integration tests

## C TESTS INTEGRATION SCENARIOS

It verifies data and contracts across all database, 3rd party API calls and events



### **SMART MOCKING**

HyperTest mocks external components and autorefreshes mocks when dependencies change behaviour



### RUN WITH CI OR LOCAL

These tests can be run locally or with CI pipeline much like unit tests



HyperTest is initialized on every microservice with its SDK.

- It then generates the trace of every incoming call i.e. request, response, outgoing call and outbound response.
- When done for all services, it generates an observability chart that reports all upstream-downstream pairs i.e. relationship between all services.



HyperTest **context propagation** provides traces that spans multiple microservices and helps developers debug the root cause of any failure in a single view

[User Interface Service]
1
> [Order Service]
1
-> [Inventory Service]
> Check Inventory
1
-> [Payment Service]
> Process Payment
l l
-> [Shipping Service]
l l
> Schedule Delivery
l -
> [Order Service]
> Update Order Status
l i
> [User Interface Service]
> Display Confirmation

### 3 Support 3 Multiple Protocols

HyperTest is capable of supporting all the commonly used web protocols like HTTP HTTP 1.1 , HTTP 2 LIKE GraphQL, gRPC etc.

Also supports all non-http calls like databases, queues like Kafka, NATS, RabbitMQ and all pub/sub systems

- ⊘ НТТР, НТТР1.1, НТТР 2
- SQL / noSQL databases
- Message Queues and Pub/Sub Systems
  - Web sockets





HyperTest generates a **code coverage report** after every run. This highlights clearly how it tests code paths for both the data and integration layer along with the core logic

All files sample-banking-node				e v of					
62.96% Statements 136/216 61.19% Branches 41/67 43.47% Functions 18/23 63.84% Lines 136/213			code cov	erage					
Press n or j to go to the next uncovered block, b, p or k for the previous block. Filter:									
File •	٥	Statements 0	0	Branches 0	0	Functions 0	0	Lines 0	
.htConf.js		100%	3/3	100%	0/0	100%	0/0	100%	
creds.js		100%	1/1	100%	0/0	100%	0/0	100%	
dbSeed.js		0%	0/29	096	0/6	0%	0/3	0%	
index.js		92.3%	132/143	87.23%	41/47	100%	10/10	92.3%	
simimulateTraffic.js		0%	0/40	0%	0/14	0%	0/10	0%	

```
All files / sample-banking-node index.js
```

 92.3% Statements
 132/143
 87.23% Branches
 41/47
 100% Functions
 10/10
 92.3% Lines
 132/143

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1
 2 1x
       process.env.HT_MODE = process.env.HT_MODE || 'RECORD';
 3
 4 1x const htSdk = require('@hypertestco/node-sdk');
 5 1x const Date = htSdk.HtDate; // if you want to mock system time
 6
 7
        /* -- DELETE befpre pushing to git -- */
 8 1x const localServiceId = 'e700b4bd-7395-4217-988e-8bc4cc3bcfb6';
 9 1x const remoteServiceId = '8e950615-2d5f-4e64-ac10-62d972e82c80'
10 1x const creds = require('./creds');
11 1x const serviceId = 'e700b4bd-7395-4217-988e-8bc4cc3bcfb6' || creds.serviceIdentifer; //process.env
12
       /* istanbul ignore next */
13
       if (!serviceId) {
14
15
          throw new Error('Please set service id');
                                                                                            uncovered lines
16
        3
                                                                                                 of code
17
                                                                                              highlighted
207
           // bug 2 - flip amount to negative -> credit becomes debit and vice-versa
208 2X
           let newBalance = account.current_balance - amount;
209
            I if (newBalance < account.minimum_balance) {</pre>
210 2x
             throw new Error('Transaction would result in balance falling below the minimum required');
211
212
           3
           await pool.query('UPDATE accounts SET current_balance = $1 WHERE 1d = $2', [newBalance, accountid]);
213 2x
           const transactionType = amount >= 0 ? 'credit' : 'debit';
214 1x
215 1x
           await pool.query('INSERT INTO transactions (account_id, amount, transaction_type) VALUES ($1, $2, $3)', |
216 1x
           return { status: 'Transaction successful', oldBalance: account.current_balance, newBalance };
217
         } catch (error) {
218 3x
           reply.status(400).send({ error: error.message });
219
         3
```





HyperTest smartly mocks external systems like **databases**, **queues**, **downstream or 3rd party APIs** that your code interacts with.

It also smartly auto-refreshes these mocks as dependencies change their behavior keeping tests non-flaky, deterministic, trustworthy and consistent



### **TEST GENERATION MODE**

#### **TEST MODE**



## 6 Never worry about creating or managing test data

HyperTest can test stateful flows without needing teams to create or manage test data.



## Summary

The hidden cost of microservices is integration failures.

• With the help of its capabilities like Smart mocking, no test data preparation, tracing etc., HyperTest is trying to make this uncertainty go away by making integration testing easy for developers. Teams should never leak integration failures.

2024

Launched our Java SDK

2023

100 services go live

2022

Node SDK Launch

2023

HyperTest was started



### **Generating Greatness**

Companies **like Porter**, **Paysense**, **Nykaa**, **Mobisy**, **Skuad and Fyers** leverage HyperTest to accelerate time to market, reduce delays and improve code quality without needing to write or maintain automation