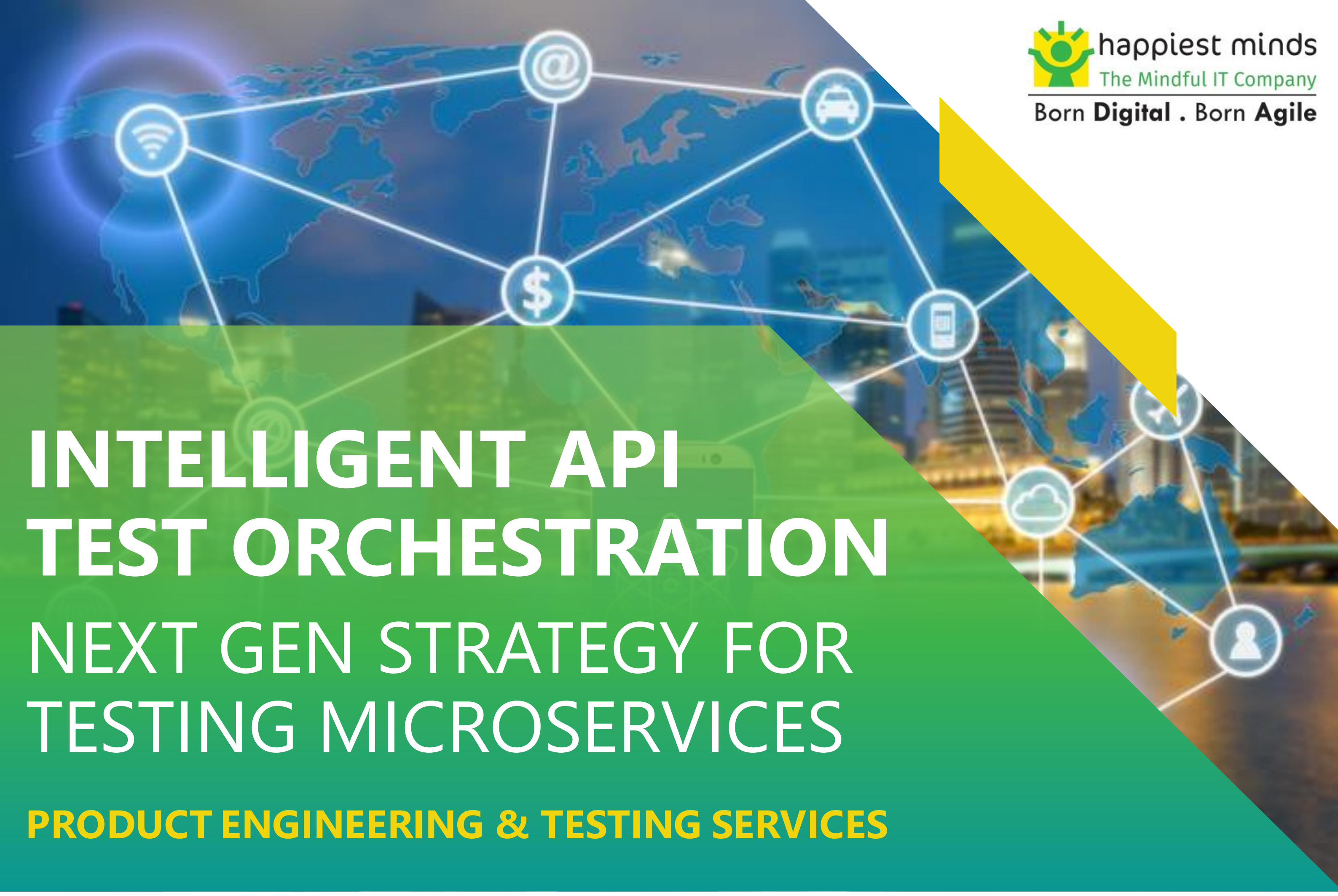


INTELLIGENT API TEST ORCHESTRATION

NEXT GEN STRATEGY FOR TESTING MICROSERVICES

PRODUCT ENGINEERING & TESTING SERVICES

ABSTRACT

Traditionally, businesses have spent a significant amount of time and effort on coding, testing and maintaining the backend APIs for a software application. However, we at Happiest Minds decided to pivot towards automating the end-to-end testing of microservices with next-gen intelligent test design techniques. Our Script-less API Test orchestrator engine marks a huge shift towards eliminating the necessity of writing code and helping in the validation of inter-services communication and the intended business logic across multiple sub-systems and component interactions.

MICROSERVICES VS API

Monolithic programs usually have a wide code base and lack modularity. Also, developing a monolithic framework poses several challenges associated with managing a large codebase, adopting new technology, scaling, launching, implementing new changes and more.

Microservices have become a popular architectural style for today's cloud-driven and native applications that are self-contained, independently deployable, resilient and evolve quickly. It is high time to research and implement innovative test design techniques to orchestrate and automate the interactions involving all sub-systems.

MICROSERVICES TESTING

"Quality means doing it right when no one is watching." - Henry Ford

Microservices communicate over networks with each other and use external data-driven interactions. Microservices handle requests by passing messages between the respective modules to form a response. A specific request can involve interaction with databases and gateways of repositories to access data. So, Automated Test assessments should provide wider coverage with the finest possible granularity for each of those interactions.

happiest minds
The Mindful IT Company
Born Digital . Born Agile

TEST LEVELS

With the help of unit and integration testing, we can validate the functional correctness and the rationale present in the individual modules that make up each microservice. In any case, without progressively fine-grained test suites, we can't be certain that the microservices interact to fulfill business prerequisites. While this can be accomplished with completely integrated end-to-end test execution, precise test feedback and shorter test runtimes can be acquired by testing the microservices detached from its outside dependencies.

Automated Regression & Explanatory Testing
- Executing all levels of test cases in the pre-production environment
- Explore the system manually using only the exposed end points by considering the Business Requirements in a Time Boxed Manner

Sprint Level

Unit Testing
- Unit Testing of Individual Domain Logic at black box
- Unit Testing of objects & methods by mocking external behavior
- Unit Testing of Individual Services

Integration Testing
- Integration Testing of every service with other Micro Services, Data Stores, Caches etc.
- Service Gateway Integration Testing for Protocol Compliance & Errors (HTTP, SSL etc.)
- State Transition Testing for any external events / triggers
- Persistence Integration Testing for testing of any persistent data objects, Object Relational Mappers etc.
- Testing of different data exchange formats between Micro Services - JSONs, AVRO, Atom Syndication Format, Atom Publishing Protocol etc.

Post Sprint Pre-Production

Component Testing
- This involves testing of all the End Points with all the Internal and External Integration Points. This can involve High Level Endpoints, Triggers from Batch Jobs etc.
- Allows more layers and integration points to be exercised
- Verify that the micro service has the correct network configuration, correct URL and is capable of handling network requests.
- Ensure the contract expected by the consuming service is met in all cases and the reliability and robustness aspects of the contract are met.

Pre-Production & Production

End-to-End Testing / Contract Testing
- End to End Testing of the Business Logic with Complete Service Orchestration and Choreography.

TARGETED TEST TYPES

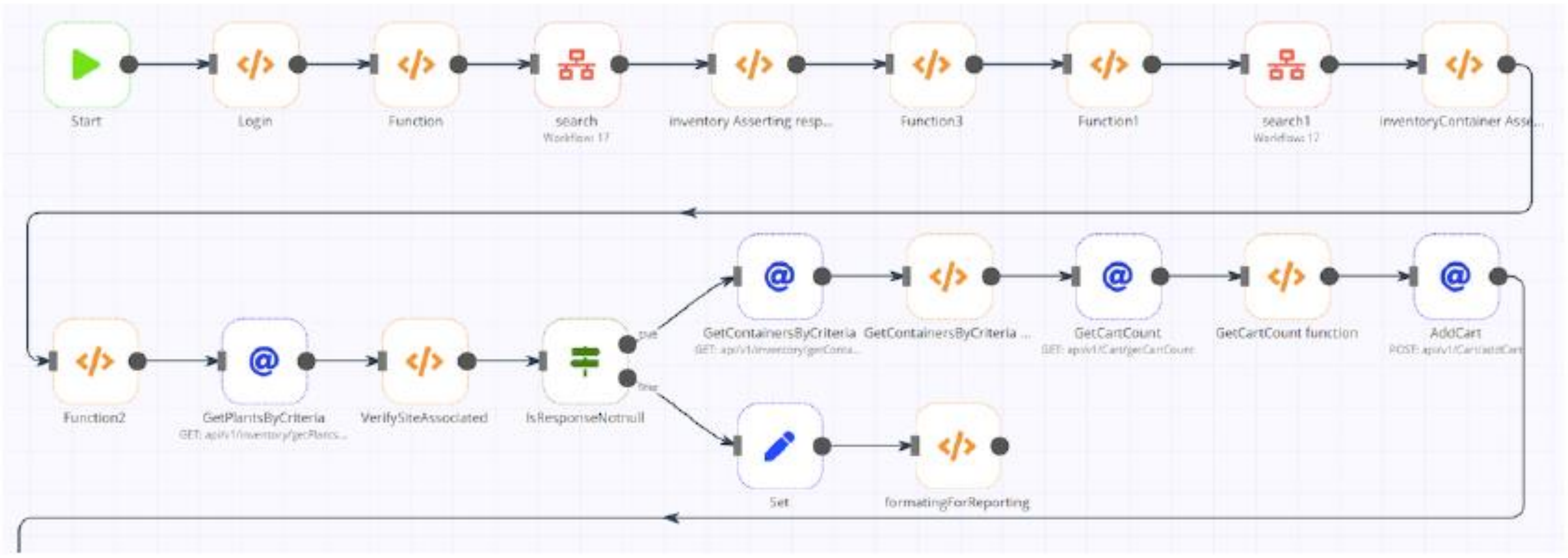
- Installation & Upgrade Testing
- Configuration Testing
- Protocol testing, Network, Data Formats, Schema etc.
- Interface Testing - Validate Data and Control Coupling between Micro Services
- Business / Domain / Orchestration Testing
- Compliance Testing for Standards & Regulations

TEST ORCHESTRATION

Development teams probably aren't building the APIs or microservices in a vacuum. Although these microservices exist as independent functional entities, they are also designed to communicate with all services & sub-systems.

By consolidating the unit, integration tests and component testing, we can accomplish higher modular test coverage that makes up every microservice. The implementation correctness is validated against the intended business logic.

However, in everything except the most straightforward use cases, clear value addition to the business is not accomplished except if numerous set of microservices cooperate to satisfy bigger business processes. Hence, there is a need to have a well orchestrated test execution to validate the end-to-end system interactions. To start off, the microservices architecture is reviewed with architects and subject matter experts for correctness and completeness from a business perspective.



'**Orchestration**' here, refers to the automated configuration, coordination, and management of all internal and external systems. It is the process in which a single consumer of the API collects information from different endpoints of the API, using the data received from the initial calls to make further calls to other endpoints.

The system APIs and microservices, through their definition and architecture, decouple the complexities of underlying structures. Based on the contractual definition of API calls, relevant CRUD operations are arranged together based on the use cases, business logic and are chained together based on schema definition to ensure continuity in test execution.

An orchestration layer (reference image above) is formed by sequencing the API calls and chaining them to support the inter-services communication. In addition, it also enables parallel execution triggers to suit the end-to-end test logic and application's user workflows.

"If you're afraid to change something it is clearly poorly designed" - Martin Fowler

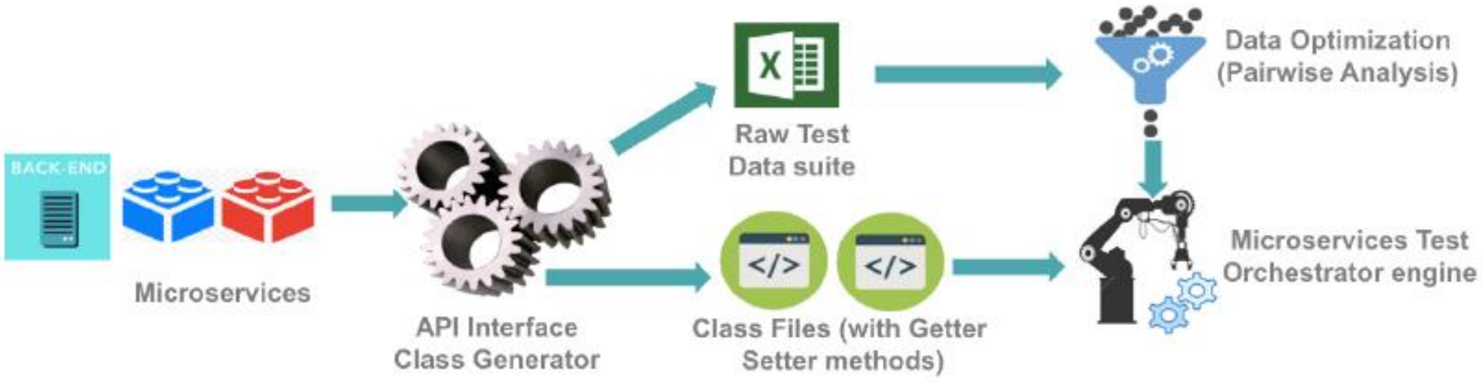
SCRIPT-LESS AUTOMATION

END-TO-END TESTING

Test infrastructures with execution pipelines are a must to enable this form of end-to-end testing. Here the relevant test suites of every service are executed multiple times across environments (Dev, QA, Pre-Prod, Prod) to validate that there aren't any regression defects or breaking code changes being introduced.

FAIL EARLY, FAIL OFTEN

It becomes evident that when the back-end test workflows are run against multiple environments, there is a greater probability of bugs surfacing. This is especially valid for complex systems running on multiple virtual machines where there might be minute contrasts between various libraries and where the design may vary, regardless of the virtualization layer.


BACK-END
Microservices
API Interface Class Generator
Raw Test Data suite
Class Files (with Getter Setter methods)
Data Optimization (Pairwise Analysis)
Microservices Test Orchestrator engine

With the help of mind maps and visual modeling techniques and discussion with architects or SMEs, the end-to-end system & component interactions are evaluated with data and control coupling analysis design techniques.

INTEGRATION WITH CI/CD PIPELINE

End-to-end tests may also have to cater to the asynchronous aspects of a system due to the backend processes between microservices or via the GUI. This can result in flakiness of the tests, consume excessive test run times and impose extra cost of maintenance of the test suites.

The APIs and microservices, through their definition and architecture, decouple the complexities of underlying structures. The end-to-end services test interact at the core granularity of the systems.

An orchestrated workflow is exposed as a service that can be invoked through an API call. The orchestration engine is then integrated to the **CI/CD** pipeline and helps in triggering the script less automated workflows at runtime.

Raw datasets are optimized with combinatorial pairwise coverage analysis. The resultant optimal data sets are then fed as parameterized input datasets for the end-to-end API regression test scenarios.

TEST OUTCOMES

FASTER BUILDS, RAPID FEEDBACK

From an architectural perspective, every microservice is treated as a black box on its own and interacts with other services or sub-systems to achieve the desired goals. So, integration and end-to-end tests are authored in this style to provide quicker feedback on the build stability while refactoring or expanding the rationale contained inside the modules.

LOWEST MAINTENANCE

Visual automation of backend test workflows, consistent with current business rules and best practices, increasing the test effectiveness and making the approach much easier to scale and robust to withstand any amount of code changes.

VISUAL TEST COVERAGE

Visual approach to create end-to-end API test scenarios for system interactions and fine tuning them to generate optimized tests for complex chaining of API workflows.

IMPROVED TEST DESIGN

End-to-end test scenarios formulated as visual workflows from all requirement sources, use cases, architectural reviews & component interactions.

MOCKING API ENDPOINTS

The visual workflows will map all possible routes that data can flow through an API. Missing logic or APIs is quickly spotted and added. This can be mocked as stubs to ensure test execution is not hindered with missing or not yet developed microservices.

INCREASED PRODUCTIVITY

Testers no longer have to spend all their time attempting to optimize their automated test scenarios by writing code. They have more time for exploratory testing of applications under test.

UNEARTH CRITICAL ISSUES EARLY

The inclusion of error workflows and exception handling at the test scenario layer helps in discovering potential showstoppers much early in the game, which otherwise would have resulted in huge production down-times. In addition, mimicking the production infrastructure in the lower environments and running the tests help in simulating real time situations and unearthing defects.

ABSTRACTED & OPTIMIZED TEST DATA

Data is abstracted from the test logic and parameterized at the test scenario layer than at the individual API entity layer. The test data is also decoupled from test logic and optimized by pairwise analysis design techniques with the help of integrated data engineering utilities.

AUTOMATION AT SCALE

Visually structured and measurable end-to-end API test workflows with expected results clearly defined, enable us to automate at a rapid pace and accommodate changes quickly. As a result, scalability to accommodate complex interactions with multiple systems is no longer a challenge.

CONTINUOUS TESTING

SYSTEM HEALTH MONITORING

The end-to-end tests provide quick inputs while the development team is refactoring the code or expanding the rationale contained inside integration modules. In addition, if the rationale contained in the integration module relapses or if the external component becomes inaccessible or breaks its contract the relevant test failures must be identified quickly during execution.

To cater to these challenges, QE teams compose integration test workflows with just the right combination of tests to give quicker feedback of the system's overall health along with the application stability. The system integration tests are then connected to the CI/CD pipeline to be triggered for continuous monitoring of availability, performance and functionality of all microservices upon every code commit.

SIMPLIFIED TEST PROCEDURE

(1) End-to-end test scenarios formulated as visual workflows from all requirement sources, use cases, architectural reviews & component interactions.

(2) The API calls and relevant CRUD operations are arranged and chained together for establishing continuity in the business logic.

(3) API test framework is enabled with auto generation of interface and model classes based on the services configuration for seamless orchestration.

(4) Data is abstracted from the test logic and parameterized at the test scenario layer than at the individual API entity layer.

The most powerful automated test approach is to use the right set of tools and technologies that can directly certify the microservices with simulated actions of a real user. At Happiest Minds, we embraced intelligent test orchestration design techniques for today's organizations to have the edge over seamless adoption of microservices, and to master the inherent challenges of testing in a highly distributed application environment. We firmly believe that Visual Test Orchestration by modeling business logic with script less microservices test automation is the way forward.

SET UP A CONSULTATION

If you are interested in Intelligent API Test Orchestration, please book a 30-minute consultation with our Head of Testing Services, Ananda Kashyap, so we can understand your priorities and identify how we can add value.

Ananda Kashyap
Sr Director and Head of Testing and DevOps Practices
business@happiestminds.com