Revolutionise Your Service Testing

Achieve Fast, Isolated Integration Tests for Services Dependent on `MGraph-AI Service GitHub`

Test Your Full Service Logic, No Network Required

We've developed a new testing architecture that allows services like the Deploy Service to be tested without network access, GitHub credentials, or live deployments.

1

Run Your Service

Your production code runs unmodified.

2

Local GitHub Service

The `mgraph-ai-service-github` runs as a local, in-memory FastAPI instance.

3

Surrogate GitHub API

All calls to the actual GitHub API are intercepted by an in-memory surrogate.

The Result: Fully isolated, fast, and deterministic tests that exercise your real production code paths.

The Current Testing Challenge: Slow and Brittle

Each test is a slow, multi-hop journey across the network, introducing multiple points of failure.

Why Traditional Integration Testing Fails Us

Network Dependency: Tests fail offline or with poor connectivity.

Slow Feedback: Each HTTP hop adds 100-500ms of latency, slowing down test suites.

Credential Management: Requires valid GitHub Personal Access Tokens (PATs) in the test environment.

Unintended Side Effects: Tests can accidentally create or modify real GitHub secrets or repositories.

Rate Limits: CI pipelines can be blocked by the GitHub API limit of 5,000 requests/hour.

Non-Deterministic: Results can vary based on the state of external services, leading to flaky tests.

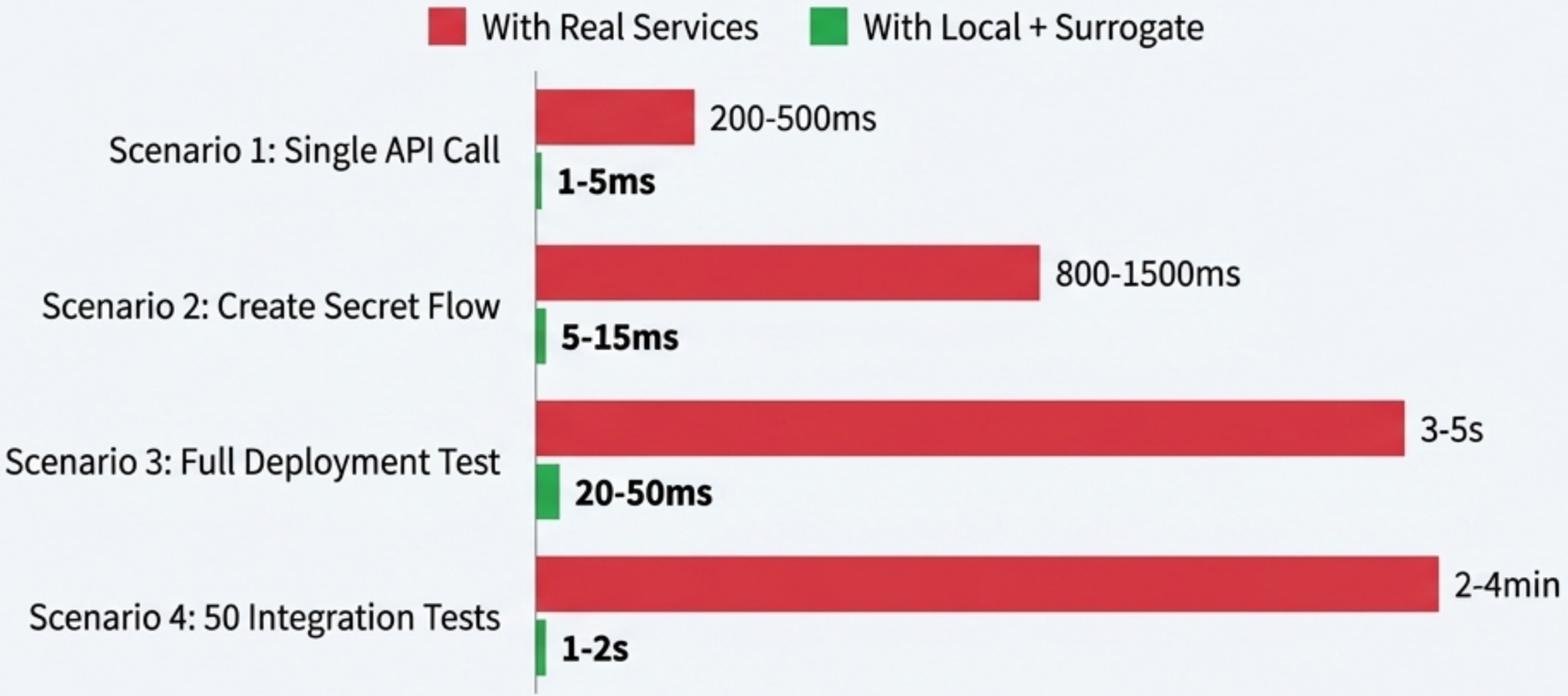
The Solution: A Fully Local, In-Memory Stack

The entire integration, from your client to the mocked API, runs in-process.

Understanding the Key Components

Component	Source Package	Purpose
`GitHub__Service__Fast_API`	`mgraph-ai-service-github`	A local, in-memory instance of the complete GitHub Service application.
`GitHub__API__Surrogate`	`mgraph-ai-service-github`	Intercepts all outgoing calls to the GitHub API, providing a mock with in-memory state.
`GitHub__API__Surrogate__Test_Context`	`mgraph-ai-service-github`	A powerful test helper for managing the setup, teardown, and state of the surrogate.
`Client__GitHub__Service`	Your Service's Codebase	Your existing client, now pointed at the local FastAPI instance instead of a remote URL.

The Performance Impact: From Minutes to Seconds



Run 50 integration tests in under 2 seconds, not 4 minutes.

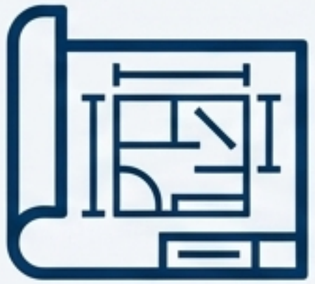
Why It's So Fast

1. No network I/O
2. No TLS handshakes
3. No data serialization (TestClient uses direct calls)
4. No rate limiting
5. No authentication delays

How to Get Started in 3 Steps

1. Install

Add `mgraph-ai-service-github` to your project dependencies.



2. Create Fixtures

Use our test helpers to spin up the local service and surrogate in your test setup.

3. Write Tests

Point your existing
Point your existing service client at the local instance and write your tests against your real code.

Step-by-Step: Code Implementation

Section 1: Install the Package

```
# pyproject.toml
[tool.poetry.group.test.dependencies]
mgraph-ai-service-github = "^0.8.0"
```

Section 2: Create Test Fixtures

```
# tests/helpers.py
from mgraph_ai_service_github.testing import get_github_service_context

def setup_test_class(cls):
    cls.gh_context = get_github_service_context()
    cls.gh_service_client.base_url = cls.gh_context.base_url
```

Section 3: Write Your Test

```
def test_create_secret_successfully(self):
    # Test setup...
    response = self.gh_service_client.create_secret(...)
    self.assertEqual(response.status_code, 201)
```

Full Control Over Your Test Environment

Pre-loading Test Data

Before your test runs, you can configure the surrogate's state to match your exact scenario.

```
# In your test's setUp method
self.gh_context.surrogate.add_repo("my-org/my-repo")
self.gh_context.surrogate.add_environment(
    "my-org/my-repo", "production"
)
```

Inspecting State After Operations

After your code runs, you can directly inspect the surrogate's in-memory state to verify the outcome.

```
# In your test method, after calling the client
secret = self.gh_context.surrogate.get_secret(
    "my-org/my-repo", "production", "MY_SECRET"
)
self.assertEqual(secret.value, "super_secret_value")
```

Testing for Permissions and Error Handling

The surrogate provides pre-configured Personal Access Tokens (PATs) with different permission levels, allowing you to test your service's error handling logic effectively.

PAT Constant	Effective Permissions	Use Case
`FULL_ACCESS_PAT`	`repo`, `admin:org`	Test successful 'happy path' operations.
`READ_ONLY_PAT`	`repo:read`	Test failures when attempting to write data (e.g., creating a secret).
`NO_SCOPE_PAT`	No permissions	Test authentication failures or insufficient scope errors.

```
def test_create_secret_fails_with_read_only_pat(self):
    # Use the read-only PAT for this request
    response = self.gh_service_client.create_secret( ..., token=READ_ONLY_PAT )
    # Assert that your service correctly handles the 403 Forbidden error
    self.assertEqual(response.status_code, 403)
```

A Faster, More Reliable CI/CD Pipeline

Key Benefits for Continuous Integration

- **Zero Secret Management**: No need to store or rotate GitHub PATs in your CI environment.
- **Eliminate Rate Limit Failures:** Run as many test jobs as you want, as often as you want.
- **Inherently Parallel-Safe:** Each test job gets its own perfectly isolated, in-memory state.
- **Dramatically Faster Pipelines:** Test suites complete in seconds, providing near-instant feedback to developers.
- **Increased Reliability:** Eradicate flaky test failures caused by network blips or external service downtime.

The New Standard for Service Development

Core Improvements to Workflow

- **Increased Developer Velocity**: Faster feedback loops mean less waiting and more productive coding time.
- **Higher Code Quality**: Easy, fast testing encourages more comprehensive test coverage, including edge cases.
- **Greater Deployment Confidence**: When tests are reliable and fast, you trust your green builds.
- **Reduced Operational Friction**: No more debugging flaky tests or managing CI secrets.

Troubleshooting Common Issues

Problem

ImportError: cannot import from mgraph_ai_service_github

Source Sans Pro cannot import from mgraph_ai_service_github.

```
ImportError: cannot import from mgraph_ai_service_github
```

Surrogate returns 404 Not Found for a repository.

Source Sans Pro when a Surrogate returns 404 Not Found for a repository.

Environment-level secret operations are failing.

Source Sans Pro as Environment-level secret operations are failing to ai prorotim are failing.

Tests are interfering with each other's state.

Source Sans Pro tresrom eseur uninning with each others as state test. common cons are reocked.

Solution

Ensure the package is installed in your test dependency group

Source Sans Pro is installed in your test dependency group.

```
poetry install --with test
```

You must add the repository to the surrogate's state before the test runs

Surrogate add the repository to ms state before the test runs.

```
surrogate.add_repo("org/repo")
```

Ensure the environment is created on the repository first

Snut the environment is created on the repository first.

```
surrogate.add_environment("org/repo", "production")
```

Ensure each test class gets its own unique context.

Do not share context across classes. Instantiate it in `setUpClass` or an equivalent fixture.

Your Path to Fast, Reliable, and Isolated Tests

1. **Install** the `mgraph-ai-service-github` package.
2. **Create** a fixture that instantiates the local GitHub Service and Surrogate.
3. **Point** your service's client at the local FastAPI application's URL.
4. **Configure** the surrogate with the necessary test repos, environments, and secrets.
5. **Write** your tests using your real client code, just as you would in production.
6. **Verify** the outcome by inspecting the surrogate's state directly.
7. **Tear down** the context after your tests complete.

This is the new standard for testing dependent services: comprehensive, deterministic, and completely free from network dependencies.