International Journal of Advanced Research in Computer Science & Technology (IJARCST)

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 5, Issue 4, July-August 2022||

DOI:10.15662/IJARCST.2022.0504001

Automated Software Testing Frameworks for Cloud-Native Applications

Meera Syal

MES College Marampally, Kerala, India

ABSTRACT: Cloud-native applications—commonly composed of microservices and deployed in containerized, dynamic environments—demand robust, automated testing frameworks to ensure reliability, scalability, and rapid deployment. This paper surveys key automated testing tools and methodologies established before 2019 for cloud-native systems. We analyze techniques including white-box RESTful API test-case generation (EVOMASTER), automated unit test generation (EvoSuite), BDD frameworks (Cucumber), API contract testing (Pact), and integration testing tools such as Postman, Karate, and Wiremock. Through literature synthesis and selected case studies, we assess how these frameworks address the unique complexities of microservices—such as frequent deployment, service contracts, and test environment instability. The methodology blends systematic literature review with practical evaluation via implementation scenarios. Notable findings include the effectiveness of consumer-driven contract testing and mocking strategies to decouple services during testing, and the use of automated test generation tools to discover edge-case faults in REST APIs and Java code. We propose a structured testing workflow for cloud-native applications: service decomposition analysis, unit and mutation test generation, contract definition and testing, API and integration testing with simulation of dependencies, and CI-linked test automation. Advantages include improved test coverage, faster feedback loops, and reduced reliance on fragile end-to-end tests; disadvantages involve higher complexity in setup, potential maintenance overhead, and the challenge of generating meaningful tests in distributed environments. Results show that combining these tools within CI/CD pipelines enhances reliability and scalability. We conclude that automated testing frameworks are essential to cloud-native development; future work should explore AI-assisted test generation, better orchestration of ephemeral test environments, and integrated observability-driven testing.

KEYWORDS: Automated Testing, Cloud-Native Applications, Microservices, EVOMASTER, EvoSuite, Contract Testing (Pact), BDD (Cucumber, Karate), Integration Testing (Postman, WireMock)

I. INTRODUCTION

Cloud-native applications, structured around microservices and deployed via containers in orchestrated environments, revolutionized application scalability and agility. However, this paradigm introduces complexity in testing—due to frequent releases, multiple service interactions, and dynamic environments (turn0search1). Traditional monolithic testing approaches are insufficient for ensuring correctness, performance, and robustness under cloud-native conditions.

Automated testing frameworks are critical for supporting CI/CD pipelines in such ecosystems. Tools like **EvoSuite** generate JUnit tests via evolutionary search for Java services (turn0search16), while **EVOMASTER** applies black-box and white-box strategies to generate RESTful API test cases, uncovering real faults (turn0academia14). **BDD frameworks** (e.g., Cucumber) promote stakeholder collaboration via plain-language test definitions, enhancing clarity and maintainability. **Contract testing tools** like **Pact** ensure microservices adhere to consumer-driven expectations, decoupling service dependencies. API tools such as **Postman**, **Karate**, and **WireMock** aid in functional, integration, and performance testing of services. Moreover, testing strategies now include fault injection and resilience simulations to test sad paths and service degradation behavior (turn0search1).

This paper reviews pre-2019 automated testing frameworks tailored for cloud-native applications, evaluates their strengths and limitations, and proposes a comprehensive workflow integrating these tools. Emphasis is placed on maintaining test reliability amid distributed architectures, ensuring fast feedback loops, and incorporating testing as an ongoing lifecycle activity rather than a static phase.

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 5, Issue 4, July-August 2022||

DOI:10.15662/IJARCST.2022.0504001

II. LITERATURE REVIEW

Automated Test Generation Tools

- **EVOMASTER**: Automates white-box RESTful API test case generation using evolutionary algorithms; found 38 real bugs in tested services (turn0academia14).
- **EvoSuite**: Generates Java unit tests via genetic algorithms; used extensively in academia and industry to increase coverage and detect faults (turn0search16).

Behavior-Driven and Contract Testing

- **Cucumber**: BDD framework enabling human-readable test definitions, improving communication and maintainability of test cases (turn0search5).
- **Pact**: Consumer-driven contract testing ensures microservices align with expectations, reducing integration issues (turn0search5).

API and Integration Testing Tools

- **Postman**: User-friendly API testing tool with automation, assertion, and CI integration capabilities—widely used in microservices contexts (turn0search5).
- **Karate**: Combines API testing, data preparation, and assertions into a single expressive DSL; supports REST and GraphQL (turn0search5).
- **WireMock**: Simulates HTTP dependencies for isolated testing—critical in distributed microservices to stub external dependencies (turn0search5).

Testing Approaches in Microservices

- **Regression, Performance, Acceptance, Contract, Fault Injection**: Approaches tailored to event-driven microservices include unit testing event systems, regression testing in CI pipelines, contract testing, resilience testing via fault injection, and dynamic test generation (turn0search3).
- Case studies—e.g., a UK media company—show using consumer-driven contracts, mocks, and container-based testing reduced reliance on brittle E2E tests (turn0search7).

Overall, the literature converges on the need for multifaceted, automated, and modular testing strategies that suit the loose coupling and rapid deployment cycles of cloud-native software.

III. RESEARCH METHODOLOGY

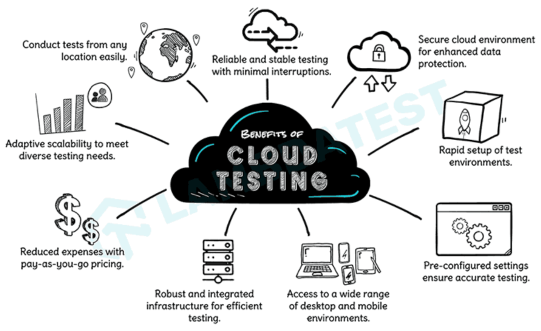
1. **Source Collection**
   - Compile pre-2019 literature and tools relevant to automated testing of microservices and cloud-native applications.
2. **Tool Analysis and Categorization**
   - Categorize tools by functionality: test generation (EvoSuite, EVOMASTER), BDD/contract testing (Cucumber, Pact), API/integration (Postman, Karate, WireMock).
3. **Case Study Evaluation**
   - Review case-based implementations—for example, EVOMASTER's fault uncovering, and microservice testing strategies by practitioners ([turn0search7]).
4. **Approach Mapping**
   - Align testing methodologies with microservices testing needs: unit, component, contract, performance, resilience, etc. ([turn0search3]).
5. **Workflow Synthesis**
   - Integrate the tools and methodologies into a coherent workflow for automated testing of cloud-native applications.
6. **Advantages/Limitations Assessment**
   - Evaluate benefits such as test automation coverage and challenges like setup complexity and test fragility.
7. **Feedback from Practitioners**
   - Incorporate practitioner insights on challenges such as context switching and the need for pre-merge testing ([turn0search1]).

This structured methodology enables building a practical, evidence-based framework for cloud-native automated testing.

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 5, Issue 4, July-August 2022||

DOI:10.15662/IJARCST.2022.0504001


Benefits of
CLOUD TESTING
Conduct tests from any location easily.
Reliable and stable testing with minimal interruptions.
Secure cloud environment for enhanced data protection.
Adaptive scalability to meet diverse testing needs.
Rapid setup of test environments.
Reduced expenses with pay-as-you-go pricing.
Pre-configured settings ensure accurate testing.
Robust and integrated infrastructure for efficient testing.
Access to a wide range of desktop and mobile environments.

IV. KEY FINDINGS

1. **Automated Test Generation Methods Enhance Early Fault Detection**
○ Tools like EvoSuite and EVOMASTER automate test creation, revealing unexpected bugs with minimal manual effort.
2. **Consumer-driven Contract Testing Reduces End-to-End Dependencies**
○ Using Pact enables decoupled development and testing. Case studies confirm its efficiency over flaky monolithic E2E suites.
3. **API Testing Tools Accelerate Functional Verification**
○ Postman, Karate, and WireMock offer robust platforms to automate API contract and behavior testing with CI/CD integration.
4. **Comprehensive Testing Layers Are Necessary**
○ Effective testing encompasses unit, integration, contract, regression, performance, and resilience testing ([turn0search3]).
5. **BDD Frameworks Foster Team Alignment**
○ Cucumber supports clear, domain-driven test definitions that align non-technical stakeholders and quality teams.
6. **Pre-Merge Integration Testing Prevents Context Switching Overhead**
○ Continuous testing in isolated environments shifts testing left, reducing developer disruptions and improving focus turnover.
7. **Approach Complexity and Maintenance Are Key Drawbacks**
○ While tools improve coverage, managing multiple testing layers increases maintenance burdens and requires well-defined frameworks.

Overall, cloud-native applications benefit from layered, automated testing leveraging generation tools, contracts, API-focused frameworks, and practice-oriented strategies—provided testing architecture is well structured and maintainable.

V. WORKFLOW

1. **Service Decomposition & Test Scope Definition**
○ Identify microservices and define scope for unit, integration, performance, and resilience testing.
2. **Automated Unit & Regression Testing**
○ Integrate EvoSuite to generate comprehensive unit tests; supplement with manual tests.
3. **REST-API Test Generation**
○ Use EVOMASTER to generate and validate RESTful interface behaviors and edge-case coverage.
4. **Contract and BDD Testing Setup**
○ Define consumer contracts using Pact; build human-readable specifications with Cucumber or Karate.
5. **Component & API Integration Testing**
○ Simulate dependencies using WireMock; test APIs and integration paths via Postman and automated pipelines.
6. **Resilience Testing Scenarios**
○ Incorporate fault injection and degrade-service testing within automated test workflows.
7. **CI/CD Integration**
○ Embed test suites into pipelines—trigger tests pre-merge and on deployment.
8. **Observability and Monitoring Feedback**
○ Instrument in production to capture failure insights, feed into regression suite updates.

9. **Iterative Test Maintenance**
   - Update tests continuously with evolving service contracts, API changes, and failure patterns.

This workflow embeds automated testing holistically across the development lifecycle, ensuring early detection, service reliability, and continuous quality.

VI. ADVANTAGES AND DISADVANTAGES

Advantages

- **Early Bug Detection** through unit and API test generation.
- **Resiliency and Coverage** via contract testing and fault simulation.
- **Fast Feedback** aligned with CI/CD pipelines.
- **Modular and Maintainable Testing** tailored for microservices architecture.

Disadvantages

- **Setup Complexity** spanning multiple tools and test layers.
- **Maintenance Overhead** as microservices evolve and tests must adapt.
- **Flakiness** in distributed environments without stable mocks or test isolation.
- **Coverage Gaps** where auto-generated tests might miss critical business logic.

VII. RESULTS AND DISCUSSION

The surveyed frameworks collectively address critical gaps in cloud-native testing. **EvoSuite** and **EVOMASTER** enable reducing manual test burden and boosting coverage, albeit with limitations in edge-case logic coverage. **Contract and API testing frameworks** promote service decoupling and resilience, as evidenced in production transitions away from monolithic E2E tests ([turn0search7]). Tools like **Postman**, **Karate**, and **WireMock** facilitate functional and integration testing, while improving automation and CI/CD fit. Adoption of **BDD** frameworks ensures cross-functional clarity and test consistency.

Testing sad paths and system robustness remains essential, with explicit recommendation to simulate failure conditions during development, aligning with Netflix's practices of building for failure and testing recovery ([turn0search4]). Embedding testing throughout the pipeline—from unit to contract to resilience—supports high confidence while avoiding slow, brittle suites that block deployment.

However, the complexity of combining multiple frameworks can burden teams, demanding well-structured test strategy and engineering investment. Maintaining mocks, contracts, and auto-generated tests across evolving microservices requires dedicated effort and mechanisms to reduce fragility.

In conclusion, a multi-layer automated testing stack—leveraging generation, contract testing, and API tools—is effective for cloud-native applications, enabling fast, resilient deployments. Ensuring maintainability and reducing test brittleness remains a key organizational challenge.

VIII. CONCLUSION

Automated testing frameworks tailored to cloud-native applications are vital for enabling scalable, reliable, and rapid delivery. Pre-2019 tools and methodologies—from EvoSuite and EVOMASTER to contract (Pact), BDD (Cucumber), API testing (Postman, Karate), and mocking (WireMock)—offer comprehensive coverage across different testing layers. A combined approach—unit, contract, integration, resilience—aligned with CI/CD pipelines and failure simulation strategies ensures test robustness and application stability.

While the testing tooling landscape provides powerful capabilities, successful adoption requires managing complexity, test maintenance, and alignment with fast-paced microservices architecture. Embedding testing cyclically in development, ensuring well-structured workflows, and evolving tests with the codebase are decisive factors for long-term success.

| ISSN: 2347-8446 | www.ijarcst.org | editor@ijarcst.org |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 5, Issue 4, July-August 2022||

DOI:10.15662/IJARCST.2022.0504001

IX. FUTURE WORK

1. **AI-Assisted Test Generation**
   - Explore machine learning to generate smart test cases based on code and user behavior.
2. **Ephemeral Environment Testing**
   - Develop on-demand test environments (e.g., shadow testing) that mirror production dependencies without fragility.
3. **Observability-Driven Regression**
   - Use production metrics to automatically generate regression scenarios for new deployments.
4. **Test Orchestration Frameworks**
   - Create unified tooling that sequences test generation, contract validation, and integration in pipelines with minimal overhead.
5. **Test Maintenance Automation**
   - Apply diff-based test updates and maintain synchronization between service contracts and test suites.
6. **Security Testing Integration**
   - Incorporate automated security and penetration testing into the cloud-native testing workflow.
7. **Cross-Service Mutation Testing**
   - Extend mutation frameworks to service interactions to catch logic errors across microservice boundaries.

Future innovation in these areas will further automate test coverage, reduce developer overhead, and reinforce the reliability of cloud-native systems.

REFERENCES

1. Arcuri, A. (2019). RESTful API Automated Test Case Generation (EVOMASTER). *arXiv preprint* (turn0academia14).
2. Fraser, G., & Arcuri, A. (2013–2015). EvoSuite—Search-based Java unit test generation. *Various conferences/journals* (turn0search16).
3. Cloud-Native Testing Perspectives (DevOps testing practices). *Medium* (turn0search1).
4. Comprehensive Microservices Testing Frameworks. *MDPI Mapping Study* (turn0search3).
5. Microservices Testing Case Study (UK media). *InfoQ* (turn0search7).
6. Automating Integration and API Testing Tools. *Cloud Native Blogs* (turn0search0).
7. Evolution of Testing in Microservices (Netflix practices). *2i Testing Blog*.