AdoptOpenJDK Quality Assurance (AQA)

1.0

1 Introduction

At the AdoptOpenJDK project, we are committed to producing high-quality OpenJDK binaries for consumption by the community. After speaking with different implementers, and listening to the needs of our community, developers and enterprise consumers alike, we have heard very clearly the desire to guarantee a certain level of quality.

Quality Assurance means, *"Make quality certain to happen"*. Our goal is to make it certain that the project delivers high-quality binaries, that satisfy enterprise-grade requirements. Enterprises and developers alike require that the binaries be:

- **Secure** – ensure binaries are not susceptible to known security vulnerabilities
- **Functional** – runs the apps they care about, on platforms they care about
- **Performant** – meets or exceeds a set performance bar for various types of operations
- **Scalable** – can span and scale to enterprise workloads

2 Guidelines for AQA

2.1 Open & Transparent

We believe open languages deserve open tests. This means test source should be open and tests should be executed in the open with their results openly available.

Transparent processes strengthen confidence. Consumers get to see test results directly and know a certain quality bar was achieved, rather than just be told some testing was done. Open tests get scrutinized, get fixed, get loved.

Open testing engages more people and helps to drive innovation and build community. Being able to see and compare test results across implementations also creates a healthy and competitive ecosystem that ultimately benefits all.

2.2 Diverse & Robust Set of Test Suites

We want a diverse and robust set of test suites to fulfill enterprise and developer requirements.

These tests should cover different categories including functional/regression, security, performance, scalability, load/stress. The tests also need to span different JDK versions, implementations, platforms, and applications.

2.2.1 Functional and Regression Tests

We currently utilize both the OpenJDK regression test suite and the Eclipse OpenJ9 functional test suite. While there remains some effort to segregate the portions of these test suites that are VM-implementation specific (OpenJDK functional originally written to target Hotspot VM, Openj9 functional originally written to target Openj9 VM), we see there is a great number of tests that are implementation agnostic and can be used to verify different implementations.

Both of these test groups, which we call “openjdk” and “functional” are typically unit tests designed to verify the APIs and features of the JDK. By thorough coverage of the APIs and coverage of the JEPs/JSRs, we identify interoperability issues and consistency results. Both test suites are continuously augmented at their source projects/repositories, OpenJDK and Eclipse OpenJ9 respectively. For this reason, we have chosen to include portions of these as part of AQA, since the tests are being kept current and relevant to the changes to the binaries we test.

While there are different JDK implementations (ranging from different garbage collection policies to different code generation options to different JVM technologies), they all draw from OpenJDK. Functional correctness and consistency can be measured through a common set of quality metrics.

2.2.2 System and Load Tests

This category of testing includes tests designed and written from a system-wide perspective. Quality engineers have written these tests from a consumer perspective, designing common customer scenarios based on feedback and observation from service professionals and consumer feedback. Importantly, this test group also includes load and stress testing which is important to enterprise consumers. These tests ask the question, *“what level of load can this binary take before it breaks?”*.

Some of these load and stress tests fire up thousands of threads and iterate 10’s of thousands of times. The tests can also be tuned, so that as binaries become more resilient, we can increase the load to further stress and push the load bar higher.

2.2.3 External Application Tests

This test category includes various suites, at present, functional and smoke tests from a set of third-party applications and Microprofile TCKs (run on different implementations of application server).

Current External Applications Suites being run at AdoptOpenJDK

Third-party applications / running Functional and Smoke tests	Application Servers / running Microprofile TCKs
Scala	OpenLiberty
Jenkins	Payara
Lucene-Solr	Thorntail
Tomcat	Tomee
Wildfly	
Kafka	

Derby	
Elasticsearch	

These tests are run in containers based from AdoptOpenJDK docker images in order to both exercise those images from docker hub, and to verify these third-party applications work well against them. Application suites are selected for both their real-world popularity and because they offer interesting workloads that best exercise an OpenJDK binary. This suite is expandable and there are more applications in the plan captured in AdoptOpenJDK/openjdk-tests issue #172.

We are interested in ensuring these suites run well. We want to engage and share with application communities, but more importantly, we aim to demonstrate to enterprise consumers the correct execution of key applications.

2.2.4 Performance Tests

Performance benchmarks are important verification tools to compare binaries against an acceptable baseline. Consumers of our binaries require them to be performant. This category of tests includes microbenchmarks and large open-source benchmark suites.

In preparing AQA, we ask how can we run performance benchmarks against AdoptOpenJDK binaries and analyze and compare results? What information is coming out of some common open-source benchmarks and why might it be interesting?

The challenge of selecting a set of benchmarks is the varied performance metrics are often in opposition with each other, throughput versus footprint being the classic example. When adding benchmarks, it will be important to clarify what the benchmark measures and how we run the benchmark (how many repetitions, with what command line options was it run, what gc policies were used, etc). We will favour benchmarks that are not implementation-specific, but rather gather a broad set of metrics that enterprise customers may expect in various deployment scenarios.

Along with the output and ‘score’ from the benchmark itself, the metrics gathered from a performance test run should also include the calculated confidence interval along with information about the machine upon which the test was run. We need to be confident that we have a repeatable measure, or at least understand in what ways a particular benchmark score can vary.

We will set baseline scores for the performance benchmarks included in AQA that binaries must meet or exceed in order to pass the performance bar.

2.2.5 Security Tests

While the regression and functional suites contain many security tests, we intend to increase the level of security tests run against the AdoptOpenJDK binaries. We will include open security test suites that test for known vulnerabilities. We also intend to use fuzzers to search for security vulnerabilities.

2.3 Evolution Alongside Implementations

2.3.1 Continual Investment

Tests (like the products they verify) need to continuously evolve & change. This is not a small effort, so is best achieved when we coordinate our efforts with many like-minded quality professionals and developers on a common goal of quality assurance.

Tests require maintenance and care. We want to continuously improve and become more effective and efficient. This includes:

- Refining automation and tools
- Automate re-inclusions upon fixes
- Remove friction, make testing easier
- Reduce process, make tools simpler

2.3.2 Process to Modify

As described in more detail in Section 2.5, the AQA will be a versioned, reproducible set of suites. As new OpenJDK versions with their new features come along, so do new or updated test suites. We may also gather metrics and additional information that will inform us that we should revise the AQA. For these reasons, the test suites that form the AQA need to be reviewed and modified on occasion. This process will be lead by the TSC test leads and should include:

- Review/assess existing & new tests on a regular basis (initially as a quarterly review)
- Community awareness of major additions/modifications/deletions to the AQA
- Update the current active set of tests (fixes, levels)

The project should also produce guides for plugging in new suites, the test source plus build file and playlist files, essential instructions for how to build and execute new test material. In this way, interested parties may introduce new test suites for trial and review and possible inclusion into an AQA update.

To summarize this section on the process to modify, the selection criteria would favour test suites that address the secure, functional, performant and scalable requirements best. We aim to improve code coverage in prioritized areas of the heat map (to best reflect real-world API usage), broaden execution of edge cases, test for newly identified vulnerabilities and new features in later releases.

2.3.3 Codecov & Other Metrics

We should continually review the value of the tests and processes that we employ. The project should gather data to measure the effectiveness of tests. This data helps inform our process of improvement and change.

Metrics of interest are:

- Heat maps
    - List of most used APIs to evaluate gaps and risk and prioritize testing
- Code coverage
    - Set bar to stay above a certain score (especially for priority areas in the heat map)
    - Shows gaps, limited, code coverage does not equate to functional coverage
- Bug prediction
    - List of most changed files, bugs often to occur in more changed areas
- Bug injection, mutation testing (in the plan, to measure the quality of our testing)
- Comparative analysis
    - Test-the-tests (Section 2.3.4)

In summary, we will gather metrics for the purpose of improving test material and our ability to assure quality.

2.3.4 Comparative Analysis

We can employ a “test-the-tests” mechanism, running tests and seeing how they perform across implementations/versions etc. This allows for a repeatable pattern to follow when triaging test results, look first at the failure and look to see if it fails across versions, platforms and implementations to hone in on root cause. We can also employ tools for a ‘diff’ of test results, to compare across the variations that we encounter at the project.

One of the greatest benefits we offer is that we are testing many different implementations, versions and platforms in one spot, making it easy to compare. This comparison informs stakeholders, enterprises, open-source communities, and developers on the qualities of a particular binary as it compares to others. Stakeholders have answers to questions like:

- how did each implementation fair against particular test criteria?
- how stable/fast/secure is the new release?

2.4 Portable

Test portability is related to an open and transparent statement. The tests that form the AQA need to be easily run:

- on newly-added platforms
- against new JDK versions
- by any developer on a laptop via command line
- in any implementers continuous integration (CI) server

Portable tests allow for both easier upkeep and maintenance but also faster evolution of the AQA. This also allows developers to reproduce test failures, make changes and retest, reducing turnaround time when incorporating fixes and features.

2.5 Tag & Publish

We want consumers of our binaries to be able to see what tests were run against them, and how the binaries scored against a rigorous onslaught of tests. Consumers should be able to get a Bill of Materials (BoM), essentially a listing of the test materials used to verify an OpenJDK binary.

Per test run:

- Set the bar by which binaries are judged (target pass rate)
- Determining which failures/exclude ‘block’ releases
- Ability to associate binaries to:
    - Test summary (pass/fail/excluded totals)
    - BoM - list of SHAs of every test repo used
    - Downloadable relevant test artifacts
    - Badge/indicator on website marking binaries that pass the AQA

The goal would be the ability for 100% reproducible test results per release. Anyone should be able to grab a binary and the test artifacts including the BoM and reproduce the same test set that was originally run at the project.

3 Summary

Our goals and intention for AQA is to provide the community with a valuable quality assurance toolkit and set a high bar for OpenJDK binaries being produced and distributed. We believe that working together on this toolkit in an open and collaborative environment will be beneficial to all implementers, developers, consumers, and stakeholders within the open community.