PROWESS

Behind the Report—Testing Addendum:

Intel® Data Center and AI (DCAI) Java Benchmarking Methodology

Methodology

This document describes Prowess Consulting's approach to measuring the real-world impact of Java Development Kit (JDK®) version selection on data-center workload performance, as demonstrated by performance benefits for Elasticsearch®, Apache Cassandra®, and Apache Spark™ workloads running in Amazon® Elastic Compute Cloud™ (Amazon EC2®) M6 instances. We tested cloud instances powered by 3rd Gen Intel® Xeon® Scalable processors, 3rd Gen AMD EPYC™ processors, and Amazon Web Services® (AWS®) Graviton2 processors. For each CPU and application, we compared the latest JDK long-term support (LTS) release against the prior JDK LTS release. To emulate live-production conditions, we secured test runs using authentication and Transport Layer Security (TLS)/Secure Sockets Layer (SSL) encryption.

We proposed the following hypotheses:

- Intel Xeon Scalable processor optimizations available in newer JDK versions will translate into improved application performance for:
  - Elasticsearch, after upgrading to JDK 17 from JDK 11
  - Apache Cassandra, after upgrading to JDK 11 from JDK 8
  - Apache Spark, after upgrading to JDK 11 from JDK 8
- OpenJDK® optimizations for Intel Xeon Scalable processors will help Intel Xeon Scalable processor–based instances outperform AWS Graviton2 processor–based instances when running Elasticsearch, Apache Cassandra, and Apache Spark applications.
- OpenJDK optimizations for Intel Xeon Scalable processors will help Intel Xeon Scalable processor–based instances outperform AMD EPYC processor–based instances when running Elasticsearch, Apache Cassandra, and Apache Spark applications.

Test Plan

At a high level, the test plan encompassed setting up the Amazon EC2 test environment and subsequently deploying and running the benchmarks in that environment. Table 1 outlines the Amazon EC2 instances, JDK LTS versions, applications, and benchmarks Prowess used for this test project.

Table 1. Hardware (platform architecture), Java Development Kit (JDK®) versions, data workloads, and Amazon® Elastic Compute Cloud™ (Amazon EC2®) instance sizes used in testing

Hardware (architecture)	JDK® version	Workload (benchmark)	Instance size
3rd Gen Intel® Xeon® Scalable processor (x86)	JDK 11, JDK 17	Elasticsearch® (Rally)	m6i.xlarge m6i.4xlarge m6i.16xlarge
	JDK 8, JDK 11	Apache Cassandra® (Cassandra)* Apache Spark™ (HiBench)*	
Amazon Web Services® (AWS®) Graviton2 processor (ARM®)	JDK 17	Elasticsearch (Rally)	m6g.xlarge m6g.4xlarge m6g.16xlarge
	JDK 11	Apache Cassandra (Cassandra)* Apache Spark (HiBench)*	
AMD EPYC™ processor (x86)	JDK 17	Elasticsearch (Rally)	m6a.xlarge m6a.4xlarge m6a.16xlarge
	JDK 11	Apache Cassandra (Cassandra)* Apache Spark (HiBench)*	

* Apache Cassandra and HiBench do not currently support JDK 17.

Testing Process

This section describes the testing process for each benchmark. During the testing process, the Prowess engineers captured CPU, memory, and disk metrics. We ran each test three times, taking the median of the results for the final result.

The Amazon EC2 instances with 3rd Gen Intel Xeon Scalable processors started with the older versions of JDK for testing—JDK 8 or JDK 11—depending on the workload. Once we completed the initial testing on JDK 8 or JDK 11, we upgraded the Amazon EC2 instance on the same Java virtual machine (JVM) to JDK 11 or JDK 17, depending on the workload. Details can be found in the "Pre-Test Setup" section later in this report.

We ran the AWS Graviton2 processor–based and AMD EPYC processor–based Amazon EC2 instances with the latest LTS JDK: either the JDK 11 or JDK 17 version, again depending on the workload.

Elasticsearch® Testing

1. Run Rally benchmarks for the nyc_taxis and http_logs datasets:

   For JDK 11:

```
esrally race --distribution-version=7.13.0 --track=nyc_taxis --challenge=append-fast-with-conflicts
—-runtime-jdk=11 --report-file=/path/to/your/report.md --report-format=csv
esrally race --distribution-version=7.13.0 --track=http_logs --challenge=append-fast-with-conflicts
—-runtime-jdk=11 --report-file=/path/to/your/report.md --report-format=csv
```

   For JDK 17:

```
esrally race --track=nyc_taxis --distribution-version=8.2.2 --challenge=append-fast-with-conflicts
—-runtime-jdk=17 --report-file=/path/to/your/report.md --report-format=csv
esrally race --track=http_logs --distribution-version=8.2.2 --challenge=append-fast-with-conflicts
—-runtime-jdk=17 --report-file=/path/to/your/report.md --report-format=csv
```

Apache Cassandra® Testing

1. Run a **write** workload with the following command, replacing <*user*> and <*password*> accordingly:

```
cassandra-stress write n=100k cl=ONE no-warmup -mode native cql3 user=<user> password=<password>
-transport "keystore=/home/ubuntu/keystore.jks keystore-password=cassandra" -log file=run1.log
```

2. Run a **read** workload with the following command, replacing <user> and <password> accordingly:

```
cassandra-stress read n=100k cl=ONE no-warmup -mode native cql3 user=<user> password=<password>
-transport "keystore=/home/ubuntu/keystore.jks keystore-password=cassandra" -log file=run1.log
```

Apache Spark™ Testing

1. Move to HiBench-HiBench-7.0 and test k-means using the following commands:

```
bin/workloads/ml/kmeans/prepare/prepare.sh
bin/workloads/ml/kmeans/spark/run.sh
```

2. Move to HiBench-HiBench-7.0 and test linear regression using the following commands:

```
bin/workloads/ml/linear/prepare/prepare.sh
bin/workloads/ml/linear/spark/run.sh
```

Amazon® Elastic Compute Cloud™ (Amazon EC2®) Setup

All Amazon EC2 instances used Ubuntu® 20.04 as the operating system (OS) with 200 GB of disk space.

JDK® Versions

We ran the Amazon EC2 instances with 3rd Gen Intel Xeon Scalable processors with older versions of JDK to establish baseline performance. We ran the Elasticsearch baseline and testing workloads on JDK 11 and JDK 17, respectively. The Apache Cassandra application and HiBench benchmark do not currently support JDK 17, so we ran baseline and testing workloads on JDK 8 and JDK 11.

We ran Amazon EC2 instances with AWS Graviton2 processors and AMD EPYC processors using the latest LTS JDK releases: JDK 17 for Elasticsearch workloads and JDK 11 for Apache Cassandra and Apache Spark workloads.

Workloads and Benchmarks

Our engineers conducted testing using the following benchmarks:

- Elasticsearch workload testing with Rally 2.6.0
- Apache Cassandra workload testing with Cassandra 3.11.13
- Apache Spark workload testing with HiBench 7.1.1

M6 Instance Sizes

We tested each workload for three M6 instance sizes: xlarge, 4xlarge, and 16xlarge. Testing started with the smallest instance size, followed by testing on incrementally larger instance sizes.

Pre-Test Setup

We installed the following benchmarks prior to testing: the Rally benchmark for Elasticsearch, the Cassandra benchmark for Apache Cassandra, and the HiBench benchmark for Apache Spark.

Rally Benchmark for Elasticsearch Installation

1. Install Ubuntu 20.04.
2. Install JDK 11 for the 3rd Gen Intel Xeon Scalable processors. Install JDK 17 for the other instances. After the initial Rally benchmark runs in JDK 11 on the 3rd Gen Intel Xeon Scalable processor, upgrade to JDK 17 and run the Rally benchmark tests again.
3. Set **JAVA_HOME** with the following commands. Update <*jdk-install-dir*> to reflect the actual directory used in your implementation:

```
export JAVA_HOME=<jdk-install-dir>
export PATH=$JAVA_HOME/bin:$PATH >> ~/.bashrc
```

4. Install Python® 3.10.4 and Git 1.9+ with the following commands:

```
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.10
sudo apt install python3-pip
```

5. Install the Rally benchmark with the following command:

```
sudo pip3 install esrally
```

6. Run the Rally benchmark for the nyc_taxis and http_logs datasets. Do not use the sudo command for this benchmark, per the Rally benchmark instructions.
7. For Intel® processor–based Amazon EC2 instances running JDK 11, use the following commands to run NYC Taxi and HTTP Logs tests:

```
esrally race --distribution-version=7.13.0 --track=nyc_taxis --challenge=append-sorted-no-con-
flicts-index-only —-runtime-jdk=11 --report-file=/path/to/your/report.csv --report-format=csv
esrally race --distribution-version=7.13.0 --track=http_logs --challenge=append-sorted-no-conflicts
—-runtime-jdk=11 --report-file=/path/to/your/report.csv --report-format=csv
```

8. For Intel, AMD, and Graviton2 processor–based Amazon EC2 instances running JDK 17, use the following commands to run NYC Taxis and HTTP Logs tests:

```
esrally race --distribution-version=8.2.2 --track=nyc_taxis --challenge=append-sorted-no-conflicts-in-dex-only —-runtime-jdk=17 --report-file=/path/to/your/report.csv --report-format=csv
esrally race --distribution-version=8.2.2 --track=http_logs --challenge=append-sorted-no-conflicts —-runtime-jdk=17 --report-file=/path/to/your/report.csv --report-format=csv
```

Apache Cassandra Installation

1. Install Ubuntu 20.04.
2. Install JDK 8 for the 3rd Gen Intel Xeon Scalable processors. Install JDK 11 for the other instances. After initial runs of the Apache Cassandra benchmark in JDK 8 on the 3rd Gen Intel Xeon Scalable processor, upgrade to JDK 11 and run the Apache Cassandra benchmark again. Use the following commands:

```
sudo apt install openjdk-8-jdk -y
sudo apt install openjdk-11-jdk -y
```

3. Set **JAVA_HOME** with the following commands:

```
export JAVA_HOME=jdk-install-dir
export PATH=$JAVA_HOME/bin:$PATH
```

4. Set the Apache Cassandra repository with the following commands:

```
sudo sh -c 'echo "deb http://www.apache.org/dist/cassandra/debian 40x main" > /etc/apt/sources.list.d/cassandra.list'
wget -q -O -- https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add —
```

5. If the command in step 4 does not work, use the following commands instead:

```
wget -q -O KEYS https://www.apache.org/dist/cassandra/KEYS
sudo apt-key add KEYS
sudo apt update
sudo apt install cassandra
```

6. Set up SSL for Apache Cassandra. Create a Certificate Authority (CA) root certificate and save it as **gen_ca_cert.conf.** Use the following commands to perform these actions:

```
[ req ]
distinguished_name     = req_distinguished_name
prompt                 = no
output_password        = cassandra
default_bits           = 2048
[ req_distinguished_name ]
C                      = US
ST                     = WA
L                      = Bellevue
O                      = Engineer
OU                     = Engineer
CN                     = CA
emailAddress           = engineer@email.com
openssl req -config gen_ca_cert.conf -new -x509 -keyout ca-key -out ca-cert -days 365
openssl x509 -in ca-cert -text -noout
keytool -genkeypair -keyalg RSA -alias localhost -keystore keystore.jks -storepass cassandra -keypass cassandra -validity 365 -keysize 2048 -dname "CN= localhost, OU=Engineering, O=Engineering, C=US"
keytool -keystore keystore.jks -alias localhost -certreq -file localhost_cert_sr -keypass cassandra -storepass cassandra
openssl x509 -req -CA ca-cert -Cakey ca-key -in localhost_cert_sr -out localhost_cert_signed -days 365 -Cacreateserial -passin pass:cassandra
```

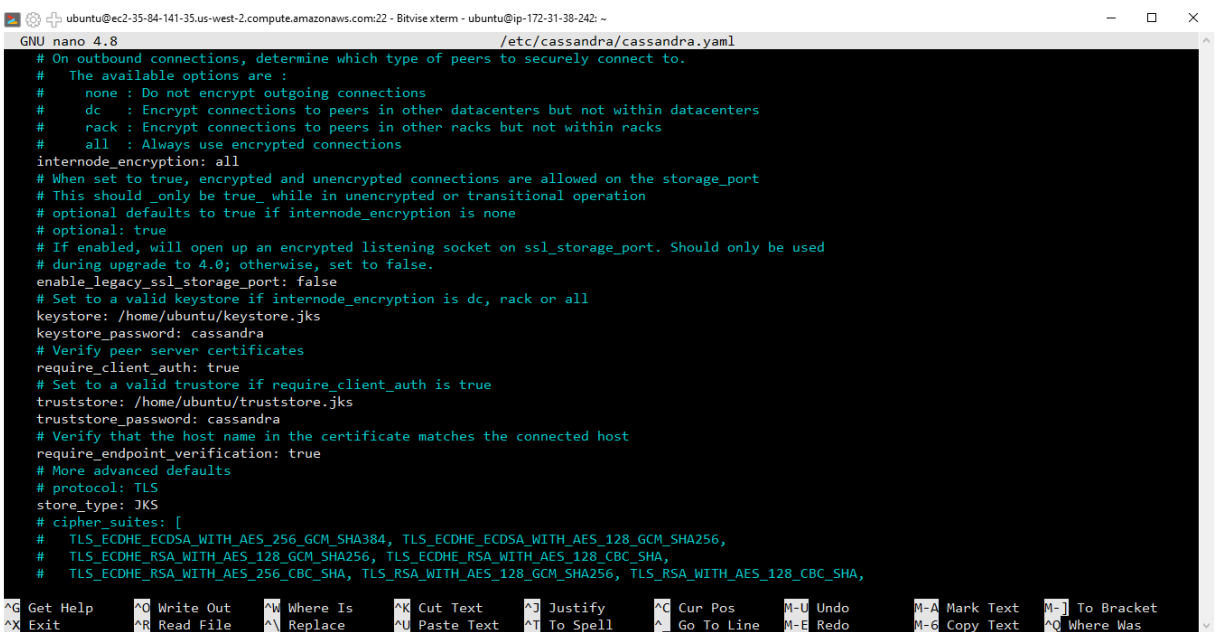
```
keytool -keystore keystore.jks -alias CARoot -import -file ca-cert -noprompt -keypass cassandra
-storepass cassandra
keytool -keystore keystore.jks -alias localhost -import -file localhost_cert_signed -keypass cassandra
-storepass cassandra
keytool -keystore truststore.jks -alias CARoot -importcert -file ca-cert -keypass cassandra -storepass
cassandra -noprompt
```

7. Check the status of the Apache Cassandra installation with the following command:

```
nodetool status
```

8. Update cassandra.yaml with the following command:

```
Search for authenticator: AllowAllAuthenticator.
Update to authenticator: PasswordAuthenticator.
Search for server_encryption_options, update the following settings:
internode_encryption: all
keystore: /home/ubuntu/keystore.jks
keystore_password: cassandra
require_client_auth: true
require_endpoint_verification: true
truststore: /home/ubuntu/truststore.jks
store_type: JKS
```



9. Enable authentication in Apache Cassandra.
10. Connect to Apache Cassandra with the following command:

```
cqlsh -u cassandra -p cassandra
```

11. Create a new user with the following command, replacing <*user*> and <*password*> accordingly:

```
CREATE ROLE <user> WITH PASSWORD = '<password>'
    AND SUPERUSER = true
    AND LOGIN = true;
```

12. Restart the Apache Cassandra service with the following command:

```
sudo service cassandra restart
```

13. Build out the cassandra-stress basic schema with the following command, replacing <*user*> and <*password*> accordingly:

```
cassandra-stress write n=1000000 cl=one -mode native cql3 user=<user> password=<password> -schema
keyspace="keyspace1" -log file=load_1M_rows.log
```

14. Run a write workload with the following command, replacing <*user*> and <*password*> accordingly:

```
cassandra-stress write n=100k cl=ONE no-warmup -mode native cql3 user=<user> password=<password>
-transport "keystore=/home/ubuntu/keystore.jks keystore-password=cassandra" -log file=run1.log
```

15. Run a read workload with the following command, replacing <*user*> and <*password*> accordingly:

```
cassandra-stress read n=100k cl=ONE no-warmup -mode native cql3 user=<user> password=<password>
-transport "keystore=/home/ubuntu/keystore.jks keystore-password=cassandra" -log file=run1.log
```

Apache Spark Installation

1. Install Ubuntu 20.04.
2. Install JDK 8 for the 3rd Gen Intel Xeon Scalable processors. Install JDK 11 for the other instances. After initial runs of the HiBench benchmark in JDK 8 on the 3rd Gen Intel Xeon Scalable processor, upgrade to JDK 11 and run the HiBench benchmark again.
3. Install prerequisites with the following command:

```
sudo apt install bc git python2 openjdk-8-jdk maven -y
sudo apt-get remove scala-library scala
sudo wget https://downloads.lightbend.com/scala/2.12.3/scala-2.12.3.deb
sudo dpkg -I scala-2.12.3.deb
```

4. Set **JAVA_HOME** with the following command:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-arm64
```

5. Install and configure Apache Hadoop® and Apache Spark.
6. Start Apache Hadoop and Apache Spark.
7. Download and configure the Intel HiBench benchmark:

```
nano conf/spark.conf
```

   - Update the following setting accordingly:

```
hibench.spark.com /opt/spark
hibench.spark.master spark://127.0.0.1:7077
```

8. Run the HiBench benchmark.
   - Move to HiBench-HiBench-7.0 and test k-means using the following commands:

```
bin/workloads/ml/kmeans/prepare/prepare.sh
bin/workloads/ml/kmeans/spark/run.sh
```

   - Move to HiBench-HiBench-7.0 and test linear regression using the following commands:

```
bin/workloads/ml/linear/prepare/prepare.sh
bin/workloads/ml/linear/spark/run.sh
```