OpenJ9

The Next Frontier in Open Source Java Compilers:

Just-In-Time Compilation as a Service

OpenJ9

Rich Hagarty

IBM Developer Advocate

rich.hagarty@ibm.com
@rhagarty8
https://www.linkedin.com/in/rhagarty/

Java™

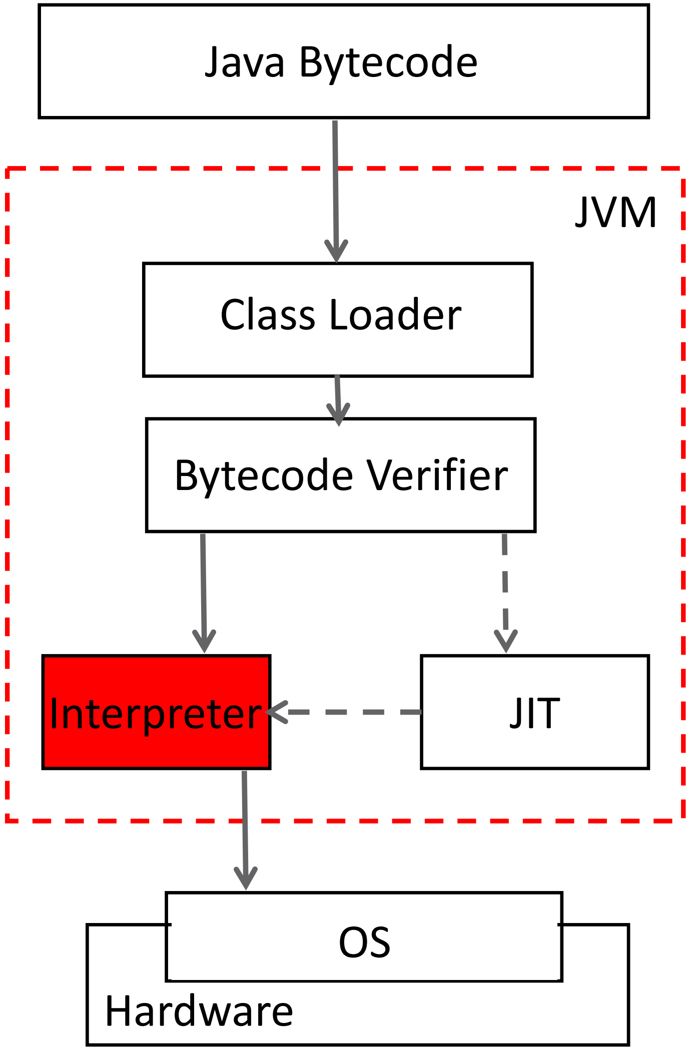
Agenda

- JVM and JIT
- JIT-as-a-Service
- JITServer from Eclipse OpenJ9
- Experiment results
- Demo
- Usage and recommendations

JVM primer

JVM Interpreter

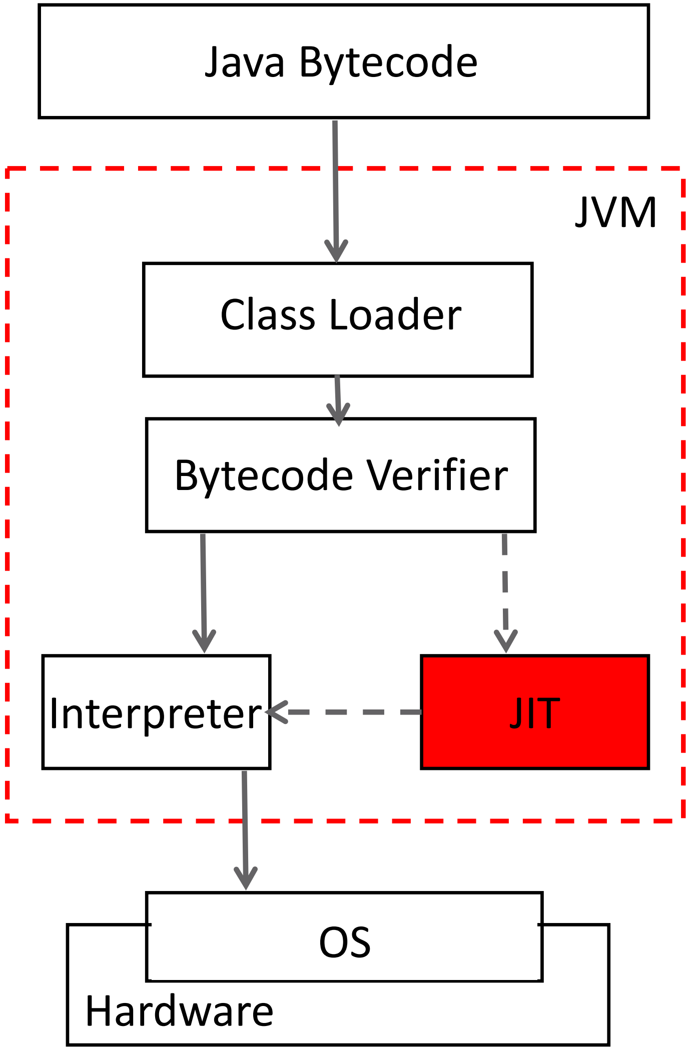
- Java programs are converted into bytecode by the javac compiler
- Machine independent bytecodes are interpreted by the JVM at runtime
- This ensures portability of Java programs across different architectures
- But it affects performance because interpretation is relatively slow



Just-in-Time Compiler

OpenJ9

- Performance is helped by the JIT compiler, which transforms sequences of bytecodes into optimized machine code
- Unit of compilation is a typically a method. To save overhead, only “hot” methods are compiled
- Compiled native machine code executes ~10x faster than a bytecode-by-bytecode interpreter
- Generated code is saved in a "code cache" for future use for lifetime of JVM



JIT advantages over static compilers

JIT compilation is performed at runtime, while the Java application is running. The has advantages over static compilers:

- JIT can optimize generated code for the machine they are running on
- JIT can tailor-fit code to the application that is executed and to the input that is provided. This is done through runtime profiling

OpenJ9

JIT Compilation disadvantages

Java™

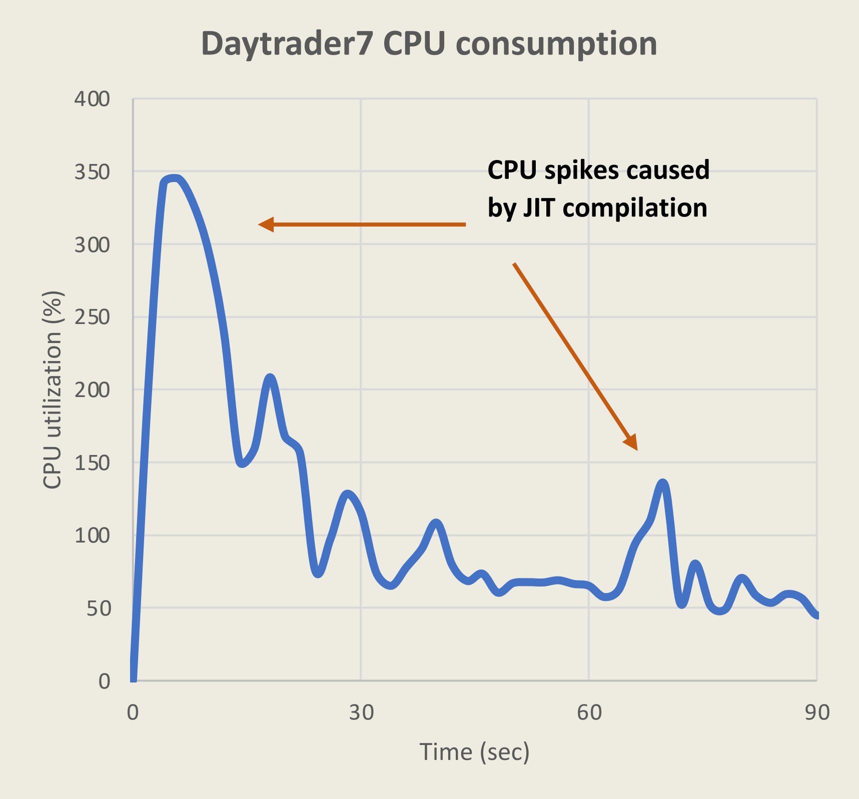
JIT disadvantages

- JIT compilers requires CPU and memory at runtime, which interferes with the running Java application
- Affects application startup/ramp-up and QoS
  - Most JIT compilations occur during this phase
  - Worse on small containers/VMs where resources are limited

JIT – CPU usage

JIT compilation creates spikes in CPU usage.

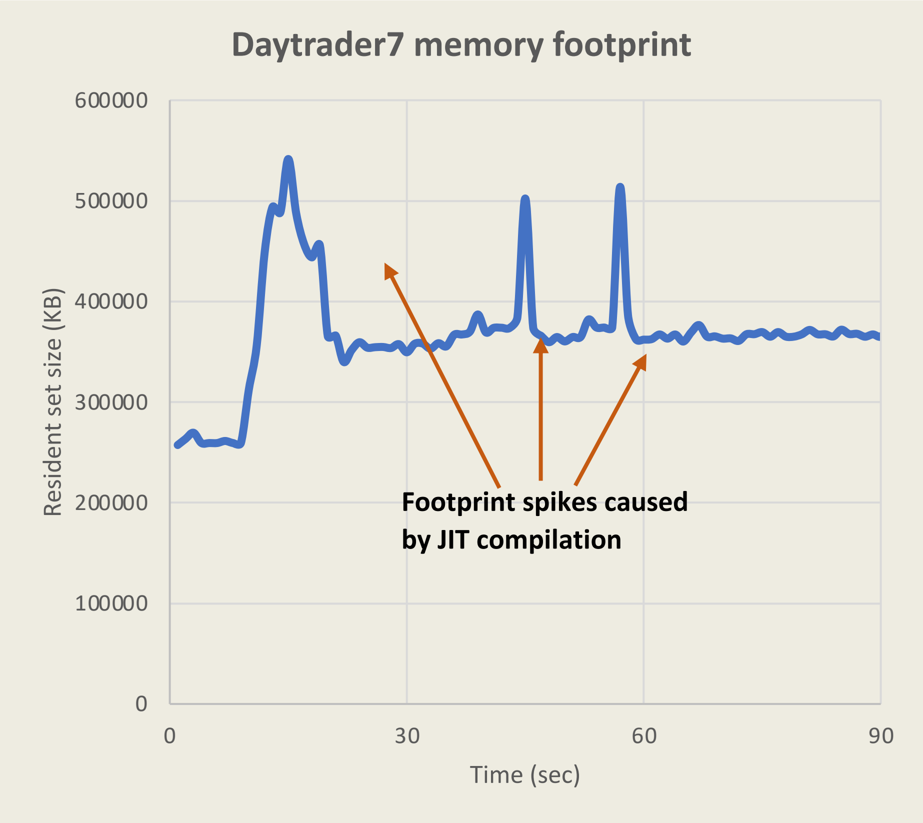
- Can slow application start-up/ramp-up
- Can create hiccups in Java applications and lower QoS



JIT – memory footprint

JIT compilation creates spikes in memory usage

- These can create OOM events resulting in crashes, lower availability or lower QoS
- A way to avoid OOM is to overprovision for the peak memory consumption – resulting in higher costs
- Determining the amount to overprovision is hard – JVMs have a non-deterministic behavior



OpenJ9

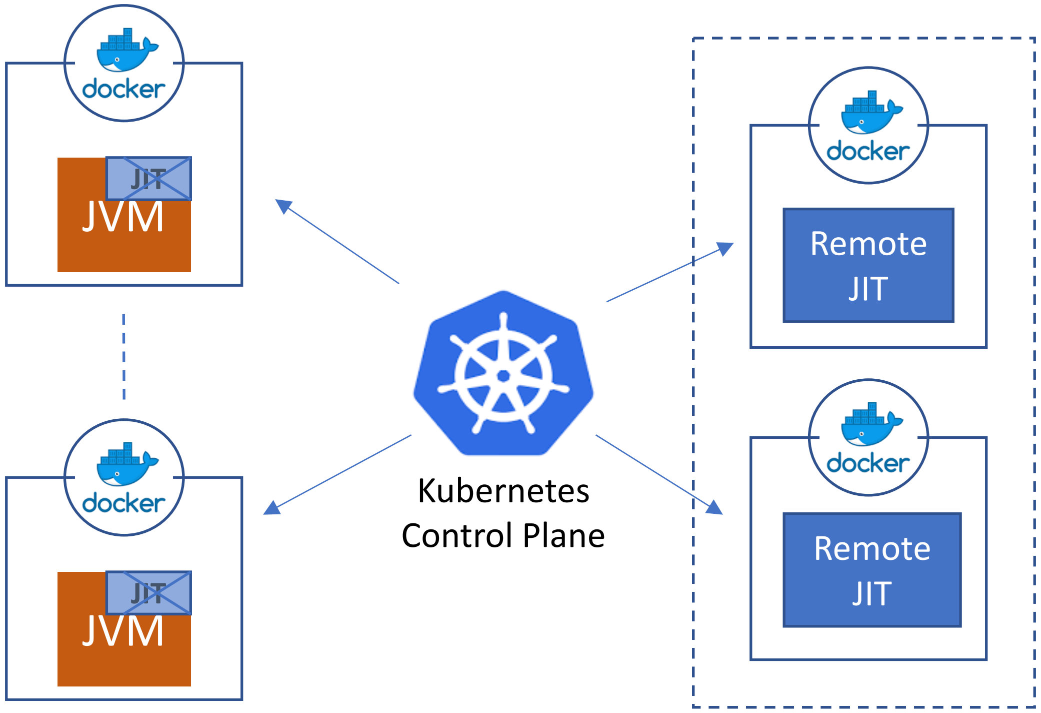
A Solution – JIT-as-a-Service

Java™

JIT-as-a-Service

Decouple the JIT compiler from the JVM and let it run as an independent process

Offload JIT compilation to remote process



Treat JIT compilation as a cloud service

- Auto-managed by orchestrator
- A mono-to-micro solution
- Local JIT still available

OpenJ9

JITServer from the Eclipse OpenJ9 JVM

Java™

Eclipse OpenJ9

- JITServer feature is available in the Eclipse OpenJ9 JVM

- Branded name is "Semeru Cloud Compiler"

- OpenJ9 combines with OpenJDK to form a full JDK

Link to GitHub repo: https://github.com/eclipse-openj9/openj9

Overview of Eclipse OpenJ9

Designed from the start to span all the operating systems needed by IBM products

This JVM can go from small to large

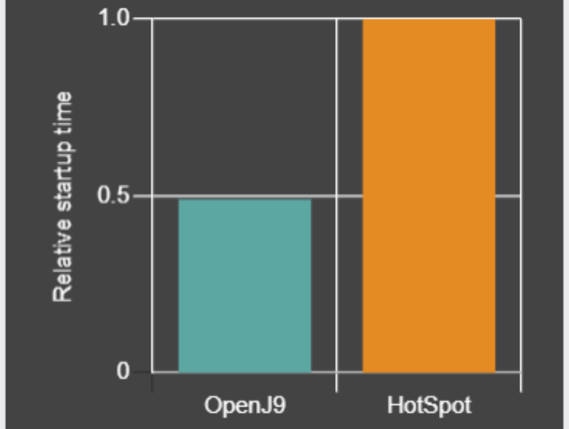
Can handle constrained environments or memory rich ones

Renowned for its small footprint, fast start-up and ramp-up time

Is used by the largest enterprises on the planet

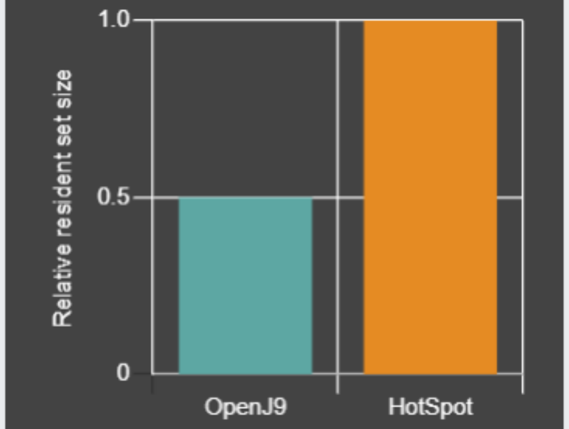
Eclipse OpenJ9 Performance

51% faster startup time



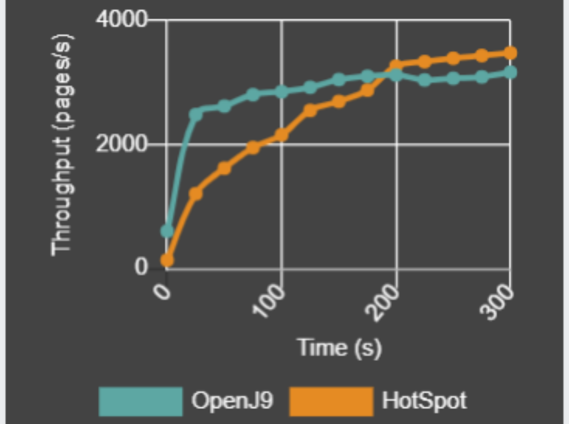
By using shared classes cache and AOT technology, OpenJ9 starts in roughly half the time it takes HotSpot.

50% smaller footprint after startup



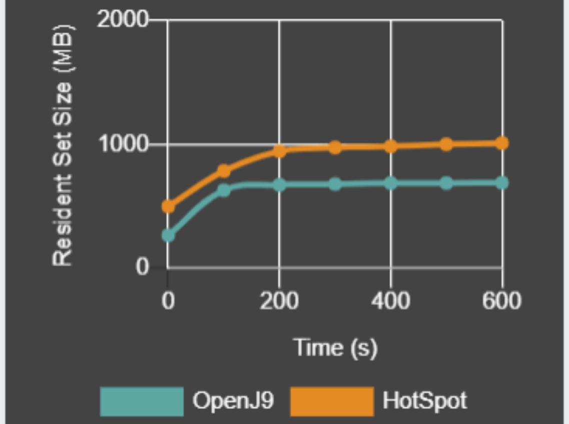
After startup, the OpenJ9 footprint is half the size of HotSpot, which makes it ideal for cloud workloads.

Faster ramp-up time in the cloud



OpenJ9 reaches peak throughput much faster than HotSpot making it especially suitable for running short-lived applications.

33% smaller footprint during load



Consistent with the footprint results after startup, the OpenJ9 footprint remains much smaller than HotSpot when load is applied.

OpenJ9

IBM Semeru Runtimes

Java™

IBM Semeru Runtimes

“The part of Java that’s really in the clouds”

IBM-built OpenJDK runtimes powered by the Eclipse OpenJ9 JVM

No cost, stable, secure, high performance, cloud optimized, multi-platform, ready for development and production use

Open Edition

- Open source license (GPLv2+CE)
- Available for Java 8, 11, 17, 18 (soon 19)

Certified Edition

- IBM license
- Java SE TCK certified.
- Available for Java 11, 17

IBM Semeru Runtimes

All supported architectures for both Open and Certified Editions are available at:
https://ibm.biz/GetSemeru

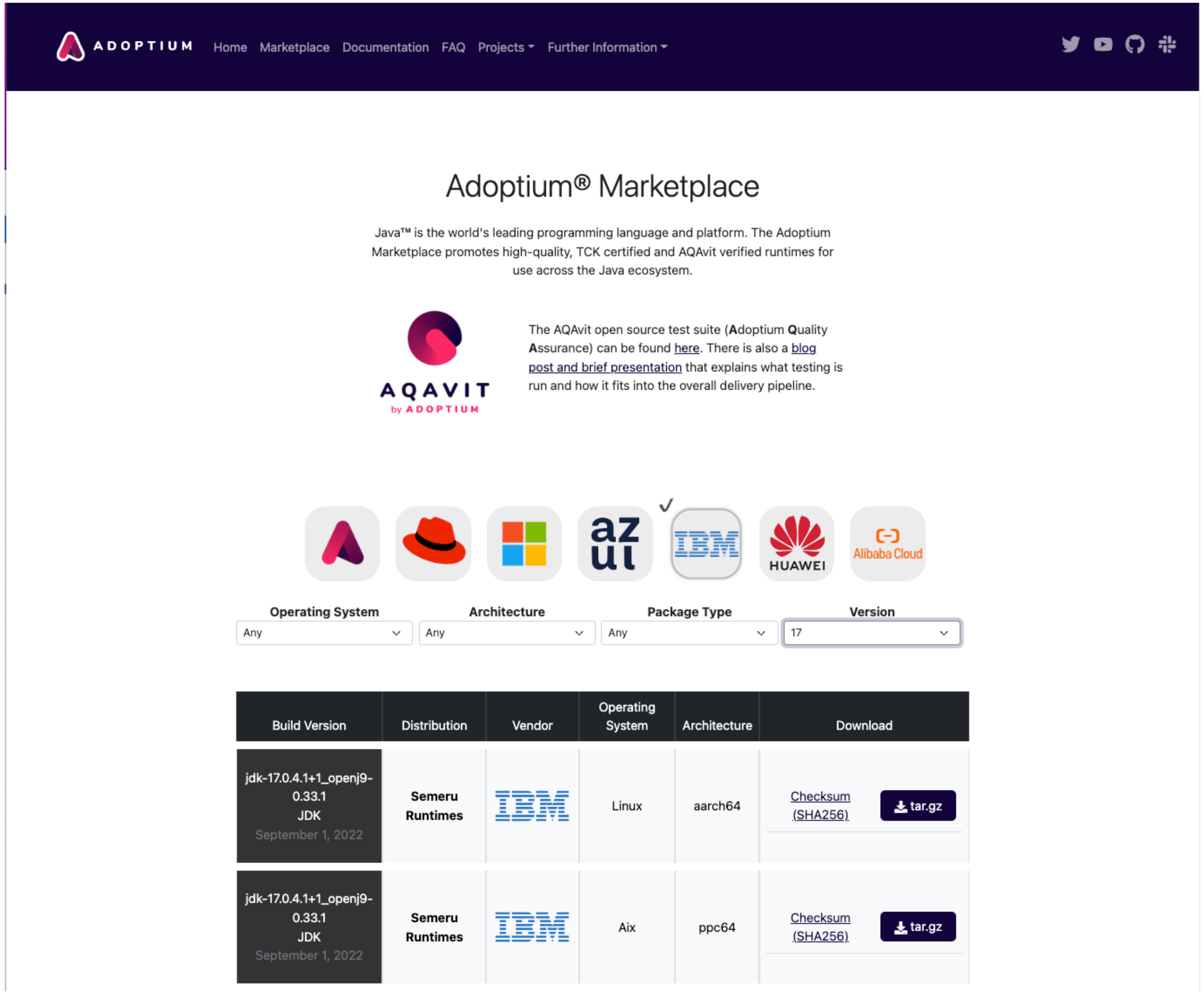
DockerHub official images (Open Edition only):
https://hub.docker.com/_/ibm-semeru-runtimes

IBM container registry:
icr.io/appcafe/ibm-semeru-runtimes:{open/certified}-{8/11/17/18}-{jdk/jre}-{ubi/ubi-minimal}

Red Hat Registry:
https://catalog.redhat.com/software/containers/search?q=semeru

Adoptium Marketplace (Certified Edition)

adoptium.net/marketplace



Adoptium® Marketplace

Java™ is the world's leading programming language and platform. The Adoptium Marketplace promotes high-quality, TCK certified and AQAvit verified runtimes for use across the Java ecosystem.

The AQAvit open source test suite (**A**doptium **Q**uality **A**ssurance) can be found here. There is also a blog post and brief presentation that explains what testing is run and how it fits into the overall delivery pipeline.

Operating System	Architecture	Package Type	Version
Any	Any	Any	17

Build Version	Distribution	Vendor	Operating System	Architecture	Download
jdk-17.0.4.1+1_openj9-0.33.1 JDK September 1, 2022	**Semeru Runtimes**	IBM	Linux	aarch64	Checksum (SHA256) tar.gz
jdk-17.0.4.1+1_openj9-0.33.1 JDK September 1, 2022	**Semeru Runtimes**	IBM	Aix	ppc64	Checksum (SHA256) tar.gz

OpenJ9

OpenJ9 JITServer Technology

Java™

JITServer Advantages for JVM Clients

PROVISIONING

- Less memory required – no local JIT compilation spikes
- Easier to size – only consider needs of application

PERFORMANCE

- More predictable – JIT no longer steals CPU cycles from the app
- Improved ramp-up time due to JITServer supplying extra CPU power when the JVM needs it the most
- Improved ramp-up most notable in performance of short-lived apps

RESILIENCY

If the JITServer crashes, the JVM can continue to run and compile with its local JIT

JITServer – natural fit for the cloud

- JITServer performs better in constrained environments
- Smaller containers increase application density and thus, reduce operational costs
- JITServer can be easily containerized and deployed to Kubernetes, OpenShift, etc., which makes it easier to run Java applications in densely packed cloud environments
- Use of server-side caching can lead to better cluster-wide CPU utilization
- Improved ramp-up time improves auto-scaling behavior
- JITServer can be scaled to match demand

JITServer Technology availability

- Available on
  - Linux on x86-64 (GA’d with OpenJ9 release 0.29.0, Oct. 2021)
  - Linux on Power (GA’d with OpenJ9 release 0.29.0, Oct. 2021)
  - Linux on zSystems (GA’d with OpenJ9 release 0.32.0, Apr. 2022)
- Supported Java: Java8, Java11, Java17
- Works on bare metal, in containers, on virtual machines, and in the cloud
- Dependencies: openssl dll, but only if using encryption

Performance graphs

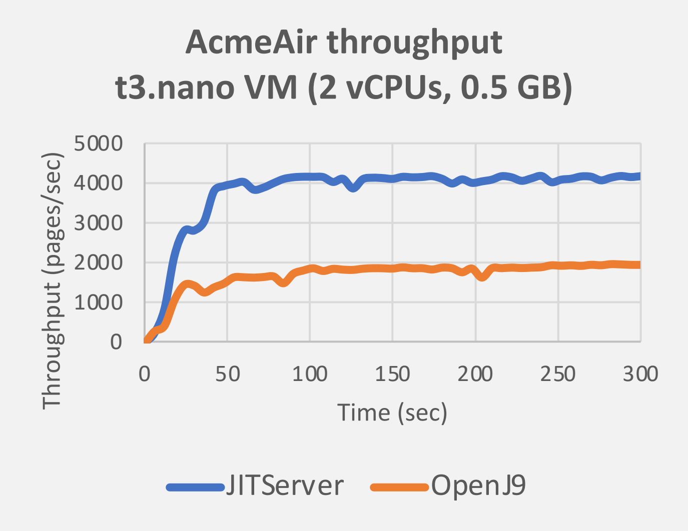
Improve VM costs in Amazon EC2 with JITServer

Goal to minimize cost ➔ Use t3.nano VM with 0.5 GB

- ~200 MB needed by OS ➔ 300 MB left for AcmeAir container

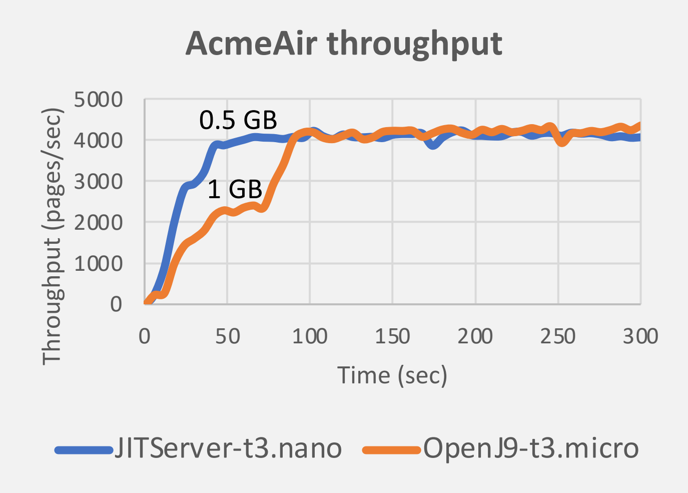
➔JITServer can double the throughput of vanilla OpenJ9

Instance type	vCPU	Memory (GiB)	Price (Linux)
t3.nano	2	0.5	$0.0052/hour
t3.micro	2	1.0	$0.0104/hour



Goal to improve throughput ➔ Vanilla OpenJ9 must move up to a t3.micro VM

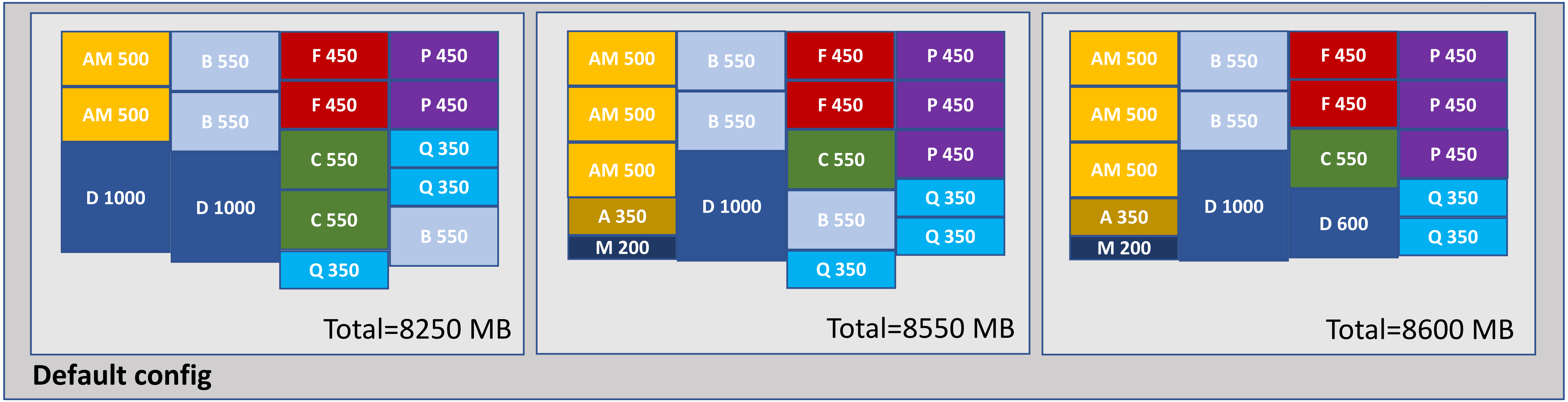
➔ JITServer achieves same throughput for half the cost



JITServer value in Kubernetes

- https://blog.openj9.org/2021/10/20/save-money-with-jitserver-on-the-cloud-an-aws-experiment/
- Experimental test bed
  - ROSA (RedHat OpenShift Service on AWS)
    - Demonstrate that JITServer is not tied to IBM HW or SW
  - OCP cluster: 3 master nodes, 2 infra nodes, 3 worker nodes
    - Worker nodes have 8 vCPUs and 16 GB RAM (only ~12.3 GB available)
  - Four different applications
    - AcmeAir Microservices
    - AcmeAir Monolithic
    - Petclinic (Springboot framework)
    - Quarkus
  - Low amount of load to simulate conditions seen in practice
  - OpenShift Scheduler to manage pod and node deployments/placement

JITServer improves container density and cost



AM 500
AM 500
D 1000

B 550
B 550
D 1000

F 450
F 450
C 550
C 550
Q 350

P 450
P 450
Q 350
Q 350
B 550

Total=8250 MB

AM 500
AM 500
AM 500
A 350
M 200

B 550
B 550
D 1000

F 450
F 450
C 550
B 550
Q 350

P 450
P 450
P 450
Q 350
Q 350

Total=8550 MB

AM 500
AM 500
AM 500
A 350
M 200

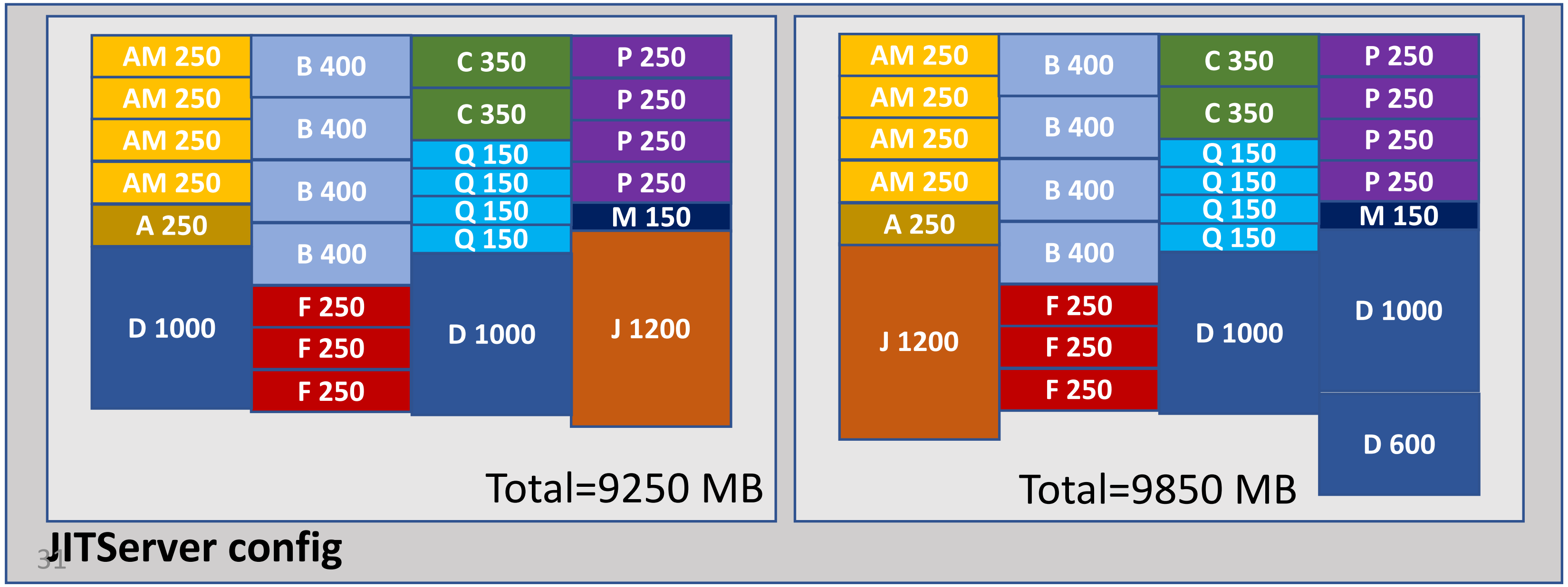
B 550
B 550
D 1000

F 450
F 450
C 550
D 600

P 450
P 450
P 450
Q 350
Q 350

Total=8600 MB

Default config



AM 250
AM 250
AM 250
AM 250
A 250
D 1000

B 400
B 400
B 400
B 400
F 250
F 250
F 250

C 350
C 350
Q 150
Q 150
Q 150
Q 150
D 1000

P 250
P 250
P 250
P 250
M 150
J 1200

Total=9250 MB

AM 250
AM 250
AM 250
AM 250
A 250
J 1200

B 400
B 400
B 400
B 400
F 250
F 250
F 250

C 350
C 350
Q 150
Q 150
Q 150
Q 150
D 1000

P 250
P 250
P 250
P 250
M 150
D 1000
D 600

Total=9850 MB

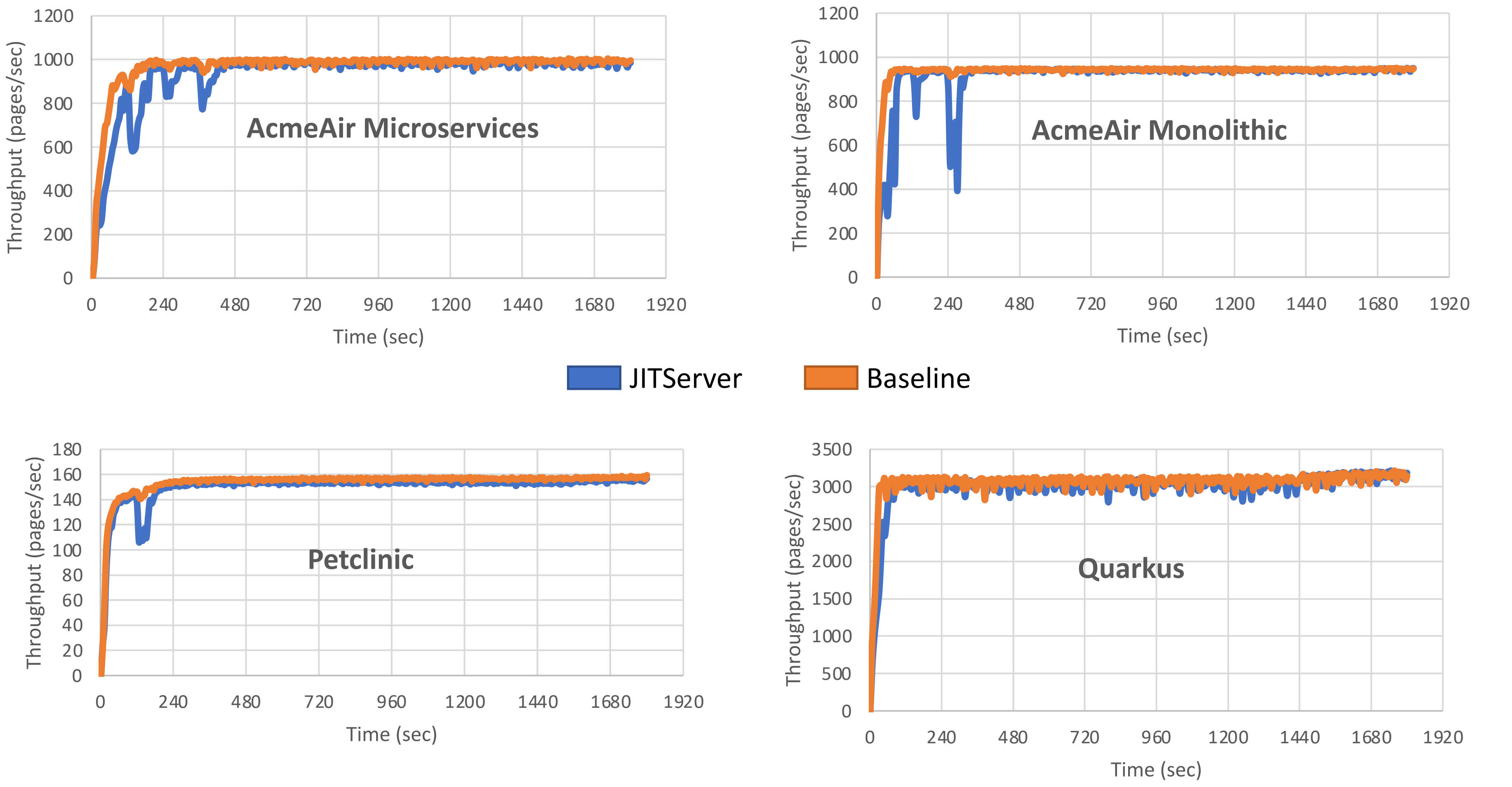
JITServer config

Legend:

AM: AcmeAir monolithic
A: Auth service
B: Booking service
C: Customer service
D: Database (mongo/postgres)
F: Flight service
J: JITServer
M: Main service
P: Petclinic
Q: Quarkus

6.3 GB less

Throughput comparison



AcmeAir Microservices

AcmeAir Monolithic

Petclinic

Quarkus

Machine load:
17.5% from apps
7% from OpenShift

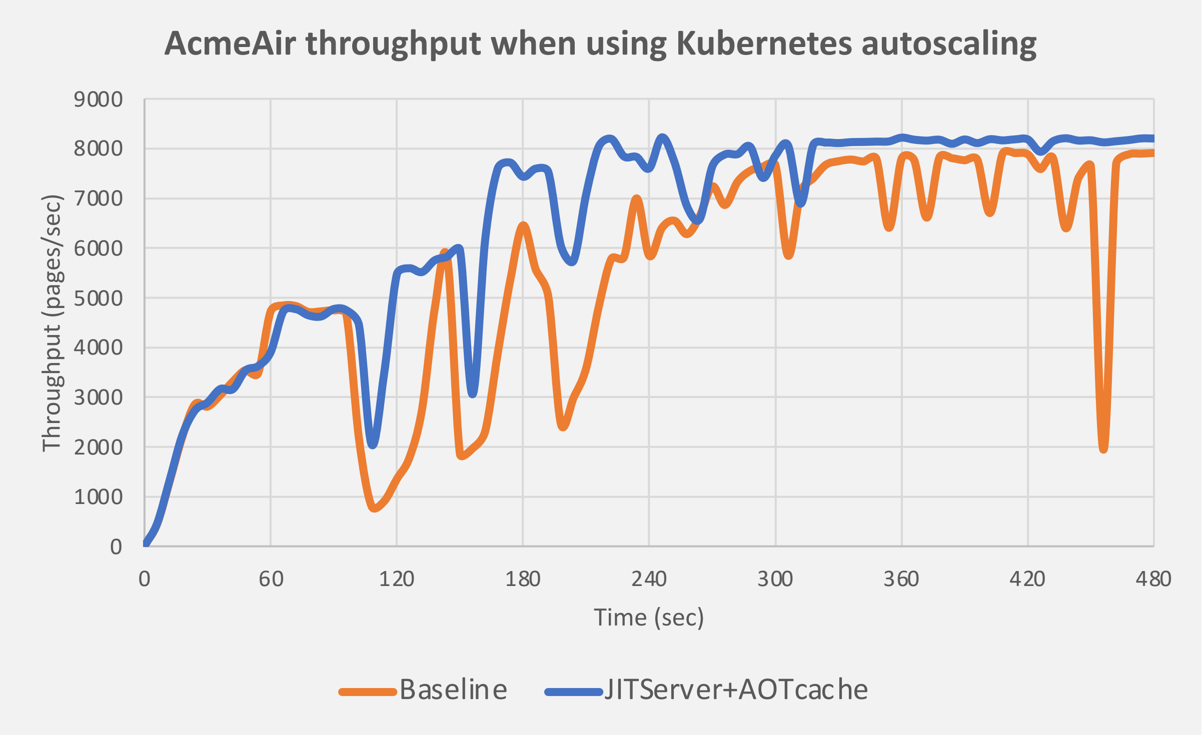
➔ **JITServer and default configuration achieve the same level of throughput at steady-state**

Conclusions from high density experiments

- JITServer can improve container density and reduce operational costs of Java applications running in the cloud by 20-30%

- Steady-state throughput is the same despite using fewer nodes

Autoscaling in Kubernetes



Setup:
Single node Microk8s cluster (16 vCPUs, 16 GB RAM)
JVMs limited to 1 CPU, 500MB
JITServer limited to 8 CPUs and has AOT cache enabled
Load applied with JMeter, 100 threads, 10 ms think-time, 60s ramp-up time

Autoscaler: scales up when average CPU utilization exceeds 0.5P. Up to 15 AcmeAir instances

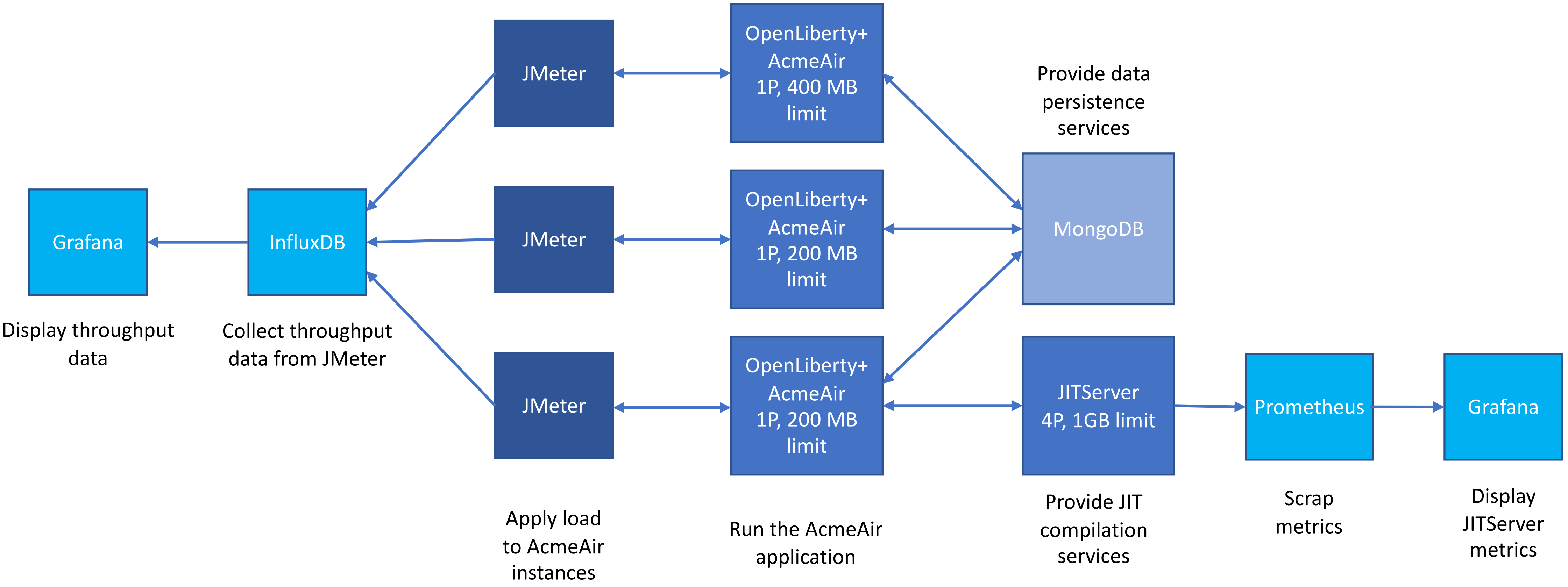
- Better autoscaling behavior with JITServer due to faster ramp-up
- Less risk to fool the HPA due to transient JIT compilation overhead

OpenJ9

Demo

Demo – Improve ramp-up time with JITServer

- Experiment in docker containers
  - Show that JITServer improves ramp-up
  - Show that JITServer allows a lower memory limit for JVM containers



OpenLiberty+ AcmeAir 1P, 400 MB limit
Provide data persistence services
JMeter
Grafana
InfluxDB
JMeter
OpenLiberty+ AcmeAir 1P, 200 MB limit
MongoDB
Display throughput data
Collect throughput data from JMeter
JMeter
OpenLiberty+ AcmeAir 1P, 200 MB limit
JITServer 4P, 1GB limit
Prometheus
Grafana
Apply load to AcmeAir instances
Run the AcmeAir application
Provide JIT compilation services
Scrap metrics
Display JITServer metrics

OpenJ9

How to use it

Java™

JITServer usage basics

- One JDK, three different personas
  - Normal JVM: $JAVA_HOME/bin/java MyApp
  - JITServer: $JAVA_HOME/bin/jitserver
  - Client JVM: $JAVA_HOME/bin/java -XX:+UseJITServer MyApp

- Optional further configuration through JVM command line options
  - At the server:

    -XX:JITServerPort=… default: 38400
  - At the client:

    -XX:JITServerAddress=… default: 'localhost'

    -XX:JITServerPort=… default: 38400
  - Full list of options: https://www.eclipse.org/openj9/docs/jitserver/

- Note: Java version and OpenJ9 release at client and server must match

JITServer usage in Kubernetes

- Typically we create/configure
  - JITServer deployment
  - JITServer service (clients interact with service)

- Use
  - Yaml files
  - Helm charts
  - Operators (under development)

- Tutorial: https://developer.ibm.com/tutorials/using-openj9-jitserver-in-kubernetes/

JITServer Helm Chart

- How-to
  - Install repo
    - helm repo add openj9 https://raw.githubusercontent.com/eclipse/openj9-utils/master/helm-chart/
  - Deploy JITServer chart
    - helm install SomeName openj9/openj9-jitserver-chart
  - This will instantiate a “deployment” and a “service”
  - Further configuration can be done with arguments given at ‘helm install’ time

    --set image.tag=“MyTag”
    --set image.repository= “MyRepo”
    --set service.port=“MyPort“
  - Passing additional options to JITServer

    E.g.: --set env[0].name="OPENJ9_JAVA_OPTIONS" --set env[0].value="-XX:+JITServerLogConnections“

- Blog post: https://blog.openj9.org/2021/03/20/introducing-the-eclipse-openj9-jitserver-helm-chart/

JITServer traffic encryption through TLS

- Needs additional JVM options
  - Server: -XX:JITServerSSLKey=key.pem -XX:JITServerSSLCert=cert.pem
  - Client: -XX:JITServerSSLRootCerts=cert.pem
- Certificate and keys can be provided using Kubernetes TLS Secrets
  - Create TLS secret:
    - kubectl create secret tls my-tls-secret --key <private-key-filename> --cert <certificate-filename>
  - Use a volume to map “pem” files

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container-name
      image: my-image
      volumeMounts:
        - name: secret-volume
          mountPath: /etc/secret-volume
volumes:
    - name: secret-volume
      secret:
        secretName: my-tls-secret
```

Monitoring

- Support for custom metrics for Prometheus
  - Metrics scrapping: GET request to http://<jitserveraddress>:<port>/metrics
  - Command line options:

    -XX:+JITServerMetrics -XX:JITServerMetricsPort=<port>
  - Metrics available
    - jitserver_cpu_utilization
    - jitserver_available_memory
    - jitserver_connected_clients
    - jitserver_active_threads

- Verbose logging
  - Print client/server connections

    -XX:+JITServerLogConnections
  - Heart-beat: periodically print to verbose log some JITServer stats
    - -Xjit:statisticsFrequency=<period-in-ms>
  - Print detailed information about client/server behavior

    -Xjit:verbose={JITServer},verbose={compilePerformance},vlog=…

JITServer usage recommendations

- When to use it
  - JVM needs to compile many methods in a relatively short time
  - JVM is running in a CPU/memory constrained environment, which can worsen interference from the JIT compiler
  - The network latency between JITServer and client VM is relatively low (<1ms)
    - To keep network latency low, use “latency-performance” profile for `tuned` and configure your VM with SR-IOV

- Recommendations
  - 10-20 client JVMs connected to a single JITServer instance
  - JITServer needs 1-2 GB of RAM
  - In K8s set vCPU “limits” much larger than “requests” to allow for CPU usage spikes
  - Better performance if the compilation phases from different JVM clients do not overlap (stagger)
  - Encryption adds to the communication overhead; avoid if possible
  - In K8s use “sessionAffinity” to ensure a client always connects to the same server
  - Enable JITServer AOT cache: -XX:+JITServerUseAOTCache (client needs to have shared class cache enabled)

OpenJ9

Conclusions

Java™

Final thoughts

- JIT provides advantage, but compilation adds overhead
- Disaggregate JIT from JVM ➔ JIT compilation as a service

- Eclipse OpenJ9 JITServer (a.k.a Semeru Cloud Compiler)
  - Available now on Linux for Java 8, Java 11 and Java 17 (IBM Semeru Runtimes)
  - Retain benefit of JIT optimization. Advantage increases with life of app
  - Especially good for constrained environments (micro-containers)
  - Kubernetes ready (Helm chart available, Prometheus integration)
  - Can improve ramp-up, autoscaling and performance of short lived applications
  - Can reduce peak memory footprint, increase app density and reduce operational costs

Resources

- Blogs
  - JITServer - Optimize your Java cloud-native applications
  - Using OpenJ9 JITServer in Kubernetes
  - Connect a Kubernetes Open Liberty app to OpenJ9 JITServer
  - Exploring JITServer on the new Linux on IBM z16 Platform
  - Save Money with JITServer on the Cloud – an AWS Experiment
  - Introducing the Eclipse OpenJ9 JITServer Helm Chart
  - A glimpse into performance of JITServer technology
  - Free your JVM from the JIT with JITServer technology

- Presentations
  - Live Demo at Oracle Code One 2018: https://youtu.be/GmP7HBzog9Q?t=1169
    - Demo setup: https://github.com/mstoodle/openj9-jitaas-demo
  - CASCON 2018: https://www.youtube.com/watch?v=3gKplQqy3zo
  - SPLASH 2018: https://www.slideshare.net/MarkStoodley/turbo2018-workshop-jit-as-a-service
  - FOSDEM 2019: http://bofh.nikhef.nl/events/FOSDEM/2019/H.1302/jit_cloud.mp4

- USENIX ATC 2022 paper: “JITServer: Disaggregated Caching JIT Compiler for the JVM in the Cloud”
  - https://www.usenix.org/system/files/atc22-khrabrov.pdf

- Documentation: https://www.eclipse.org/openj9/docs/jitserver/

OpenJ9

Thank You!

Questions?

rich.hagarty@ibm.com
@rhagarty8
https://www.linkedin.com/in/rhagarty/