

# Le Framework Quarkus et GraalVM AOT

Florian Nari

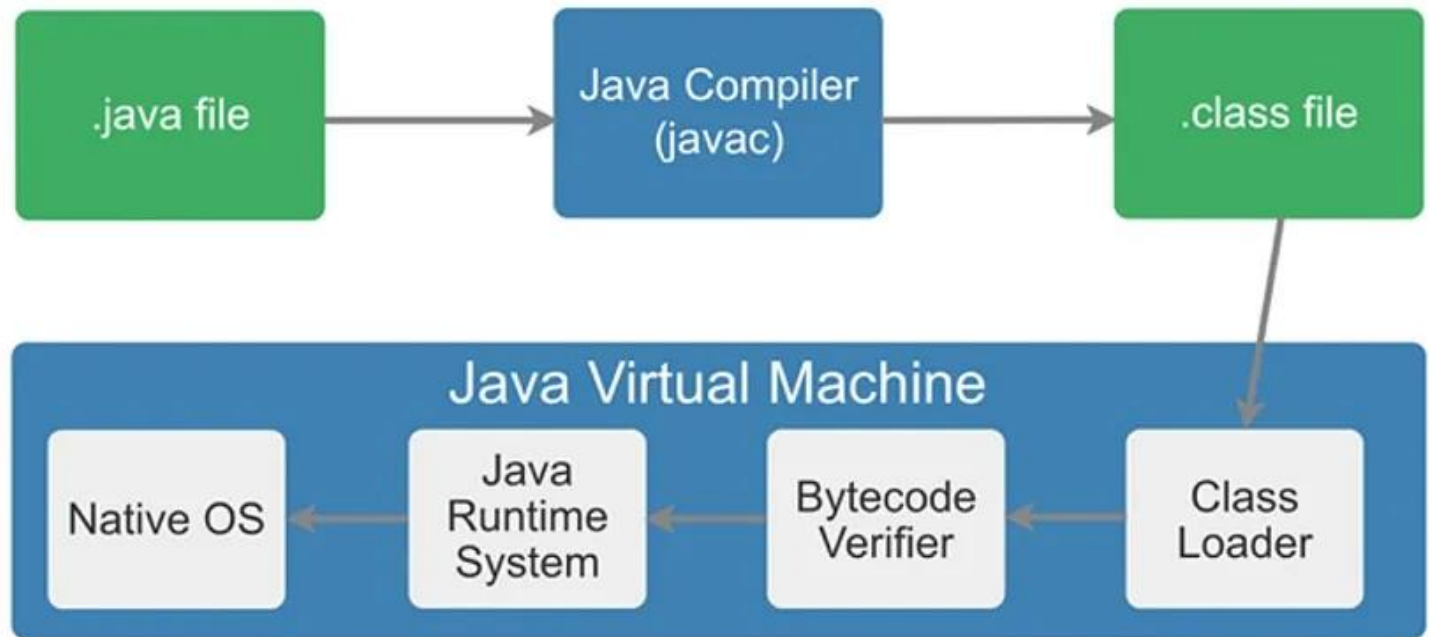


QUARKUS

GraalVM™

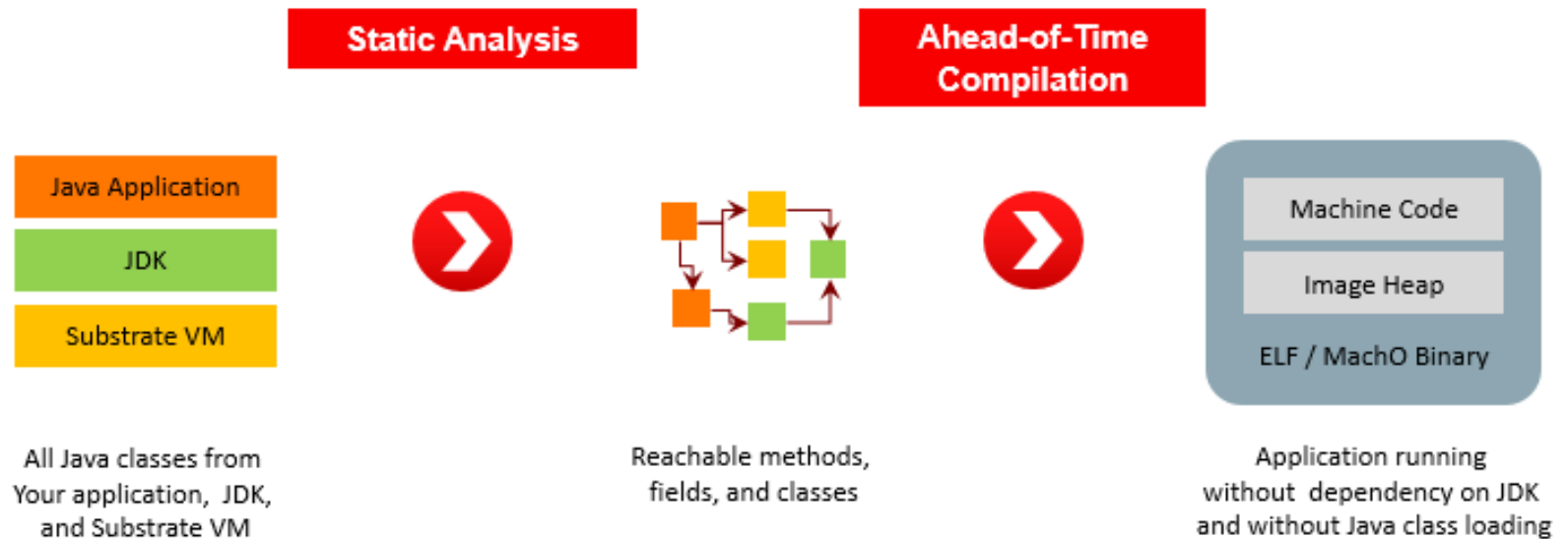
## Compilation et exécution d'une application java classique

### What is Java bytecode?



Source: <http://www.techlila.com/write-programs-linux/>

## Compilation anticipée (AOT) d'une application Java grâce à GraalVM



# Spring

- Framework java le plus utilisé
- L'injection de dépendances
- La persistance des données
- La construction d'API REST
- Et bien d'autres (Spring MVC, Spring Security...)
- Démarrage souvent long (> 1 minute)
- Incompatible avec GraalVM AOT
  - (c'est en phase expérimentale)

# Quarkus

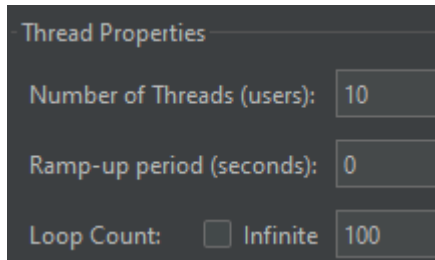
- Framework récent (2019)
- Encore très peu utilisé  
(30 à 50 fois moins d'offres d'emploi que Spring en France)
- L'injection de dépendances
- La persistance des données
- La construction d'API REST
- Et d'autres (mais beaucoup moins que Spring)
- Démarrage rapide
- Compatible avec GraalVM AOT

# Une première application pour tester

- Model
  - User(id, username)
  - Message(id, user, message, date)
- Services et APIs
  - Users
    - Ajouter un utilisateur
    - Rechercher un utilisateur via son username
    - Lister les utilisateurs
  - Messages
    - Ajouter un message
    - Lister les messages

# Benchmark - JMeter

- Thread Group



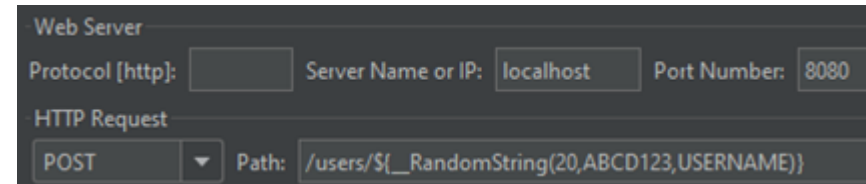
Thread Properties

Number of Threads (users): 10

Ramp-up period (seconds): 0

Loop Count: ☐ Infinite 100

- Ajouter utilisateur



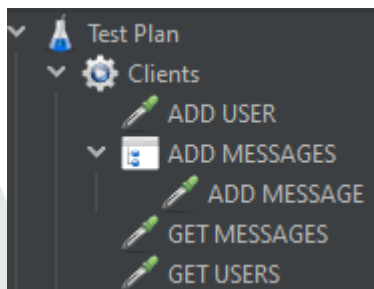
Web Server

Protocol [http]: Server Name or IP: localhost Port Number: 8080

HTTP Request

POST Path: /users/\${\_\_RandomString(20,ABCD123,USERNAME)}

- Ajouter 20 messages
    - Lister les messages
    - Lister les groupes



# Benchmark - Résultats

- Plan de test présenté précédemment

Plateforme	Framework	Démarrage	Plan de test
JVM HotSpot	Spring	4.9 secondes	305 secondes
	Quarkus	1.7 secondes	79 secondes
GraalVM EE 22.1 AOT	Quarkus	0.0 secondes	74 secondes
GraalVM CE 22.2 AOT Pré-release	Spring Native	0.2 secondes	327 secondes

- Analyse des résultats :
  - Démarrage
    - Native Image >>> JVM
    - Quarkus > Spring
  - Performances
    - Quarkus > Spring
    - Native Image = JVM



## Deuxième Benchmark

- Deuxième Benchmark
  - Distance de Levenshtein entre 2 mots de 100 000 caractères
  - $O(n * m)$  avec  $n$  et  $m$  la longueur des mots à calculer
  - 6 Threads x 1 000 appels
  - Avec Quarkus uniquement

JVM HotSpot	57 secondes
GraalVM EE 22.1 AOT	61 secondes

- Pas de gain de performance avec AOT
- Grâce à la compilation à la volée (JIT)
  - Optimisé pour la machine
  - Analyse du code en conditions réelles (vs analyse statique)

## Limites de ces Benchmark

- Windows
- Pas de prise en compte de la mémoire utilisée
- Tester avec un autre CPU/PC
- Tester avec d'autres algorithmes
- Tester avec des drapeaux (flag) différents
- Effets de sonde

# Conclusion

- GraalVM AOT
  - Actuellement ne me semble pas pertinent
    - Surtout avec Spring non compatible
  - Utile dans des cas spécifiques
    - Microservices où la vitesse de démarrage est cruciale
- Quarkus
  - Pas encore assez mature
  - Sera un concurrent à Spring dans quelques années
  - Utile pour les microservices