Monolitne rešitve za mikrostoritvene izzive

Miroslav Beranič

Bintegra d.o.o., Maribor, Slovenija
miroslav@beranic.si

V članku bom raziskal Quarkus, ogrodje za razvoj Java aplikacij. Z optimiziranimi lastnostmi delovanja ogrodje primarno cilja na razvoj Java aplikacij v oblaku. Quarkus se osredotoča na hitrost, majhen odtis pomnilnika in nizko latenco; sebe opisuje kot "supersonic subatomic Java". Ena izmed ključnih sestavin Quarkusa je GraalVM, ki omogoča hitrejši čas zagona aplikacij in učinkovito porabo virov s pretvorbo Java kode v native obliko, brez vmesnega Java VM. To je še posebej koristno za mikrostoritvene arhitekture. V članku bom pregledal tudi konkurenčne rešitve ter primerjal prednosti Quarkusa v primerjavi z drugimi razvojnimi ogrodji za oblak. Predstavil bom preprost primer uporabe Quarkusa, da bo bralec dobil vpogled v način razvoja in delovanje. Kot praktičen primer bo opisan tudi način objave aplikacije v sistemu za upravljanje z mikrostoritvam. S celovito analizo Quarkusa in tehnologije GraalVM bo članek bralcu omogočil boljše razumevanje in uporabno ter izkoristek teh inovativnih rešitev za razvoj Java aplikacij v oblaku. Članek se lahko uporabi kot vodnik: kako lahko Quarkus izboljša izkušnje pri razvoju sodobnih oblačnih storitev in aplikacij.

Ključne besede:
Quarkus
Java
GraalVM
mikrostoritve
Kubernetes

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.7

1 Uvod

Dobrodošel v svetu Quarkusa - inovativnega ogrodja za razvoj Java aplikacij v oblaku [1]. Quarkus je bil zasnovan, da odpravi izzive tradicionalnih pristopov razvoja in zagona aplikacij in ponudi razvijalcem visoko zmogljive rešitve. S svojo osredotočenostjo na hitrost, majhen odtis pomnilnika in nizko latentnost, Quarkus omogoča razvoj izjemno odzivnih aplikacij.

V tem članku bom raziskal bistvo Quarkusa, kako deluje in zakaj je postal priljubljen med razvijalci. Osvetlil bom tehnologijo GraalVM, ki prispeva k izjemno hitremu zagonu aplikacij. Poleg tega bom analiziral konkurenčne rešitve na trgu, da boš razumel, kako se Quarkus primerja z njimi in kakšne so prednosti ter slabosti. Pogledal bom tudi, kaj je novega v verziji 3 in v čem se razlikuje s predhodno iteracijo.

Za lažje razumevanje bom predstavil praktičen primer uporabe, ki bo razkril, kako Quarkus izboljša učinkovitost in odzivnost aplikacij. S tem boš dobil vpogled v to, kako enostavno je razvijati aplikacije z Quarkusom in kako ta tehnologija prinaša konkurenčno prednost. Predstavljeno aplikacijo bom zagnal preko aplikacije za upravljanje z mikrostoritvami.

Brez komercialnih buzzwordov in v tehničnem kontekstu bom raziskal vse vidike Quarkusa, kar ti bo pomagalo bolje razumeti njegove koristi in možnosti za razvoj Java aplikacij v oblaku.

2 Quarkus

2.1 Zgodovina

Ogrodje Quarkus ima objavljeno prvo verzijo, 0.0.1, dne 12.12.2018 [3], prvi zapis v zgodovini izvorne kode je bil narejen dne 22.6.2018 [4]. Časovno to pomeni, da je ogrodje relativno mlado in hkrati hitro rastoče in razvijajoče, če vemo, da je trenutna verzija 3.3.2 z dne 21.7.2023 [5]. Verzijo 1.0 (1.0.0.Final) je doseglo dne 25.11.2019, verzijo 2.0 (2.0.0.Final) dne 30.6.2021 in verzijo 3.0 (3.0.0.Final) dne 26.4.2023.

Tabela 1: Krajši pregled zgodovine verzij.

Verzija	Datum	Glavne lastnosti
0.0.1	12.12.2018	Prva verzija, praktično skelet projekta in osnovni koncepti.
0.2.0	19.12.2018	Izboljšana podpora za *Apache Maven*, uporablja se *fedora-minimal* za zabojnik, podpora za referenciranje Java razredov preko imen.
0.18.0	27.6.2019	GraalVM 19, native image je ločen paket izven GraalVM
0.20.0	31.7.2019	Dodan *quarkus-jackson*, REST client privzeto uporablj *RESTEasy*, premik razširitev za *Apache Camel* v ločen projekt.
1.0.0	25.11.2019	Stabilizacija in uporaba GraalVM 19.2.1, uporaba Quarkus Security 1.0.0.
1.1.0	17.12.2019	Ločen build za native-image, uvedba YAML za nastavitve, uporaba GraalVM 19.3, zagon samostojne RESTeasy aplikacije v glavni niti.
1.2.0	23.1.2020	Razširitev za zalogo (cache), Hibernate ORM
1.3.0	13.3.2020	Nov classloader in podpora za GraalVM 20.0.0, ena Vert.x instanca, dopolnjena podpora za deploy v OpenShift implementacija Eclipse MicroProfile specifikacije 3.3.
2.0.0	30.6.2021	Minimalno Java 11, uporaba GraalVM 21.1, Vert.x 4 ter implementacija MicroProfile 4, Hibernate 5.5, MongoDB in Panache s podporo za transakcije.
2.1.0	29.7.2021	Podpora Dev Services za Keycloak, Reactive SQL Server, Kotlin 1.5
2.2.1	31.8.2021	Podpora za GraalVM 21.2
2.3.0	6.10.2021	Podpora za Jakarta EL 3

Verzija	Datum	Glavne lastnosti
3.0.0	26.4.2023	Podpora za Hibernate ORM 6, Hibernate Reactive 2, izboljšan Dev UI, Jakarta EE 10, uporaba OpenTelemetry
3.1.0	31.5.2023	Podpora za GraalVM 22.3.2, Kotlin 1.8.21
3.2.0	5.7.2023	Dodana analitika v korake izgradnje.

Vir: [2].

2.1.1 GraalVM

GraalVM je visoko zmogljiva in univerzalna virtualna strojna oprema (VM) ter razvojno orodje, ki ga je razvilo podjetje Oracle [5]. Omogoča izvajanje različnih programskih jezikov, kot so Java, JavaScript, Python, Ruby, R, in drugi, na istem virtualnem stroju. To je edinstvena značilnost GraalVM, saj večina drugih VM-jev podpira izvajanje samo enega programskega jezika.

GraalVM se ponaša z izjemno visoko zmogljivostjo in učinkovitostjo, kar je doseženo s tehnologijo Just-In-Time (JIT) prevajanja, ki pretvori visokonivojski kode v izvorno kodo, ki jo lahko izvaja procesor. Ta tehnologija omogoča hitrejše zaganjanje in izvajanje aplikacij, saj se izvorna koda prilagaja konkretnemu okolju in arhitekturi sistema.

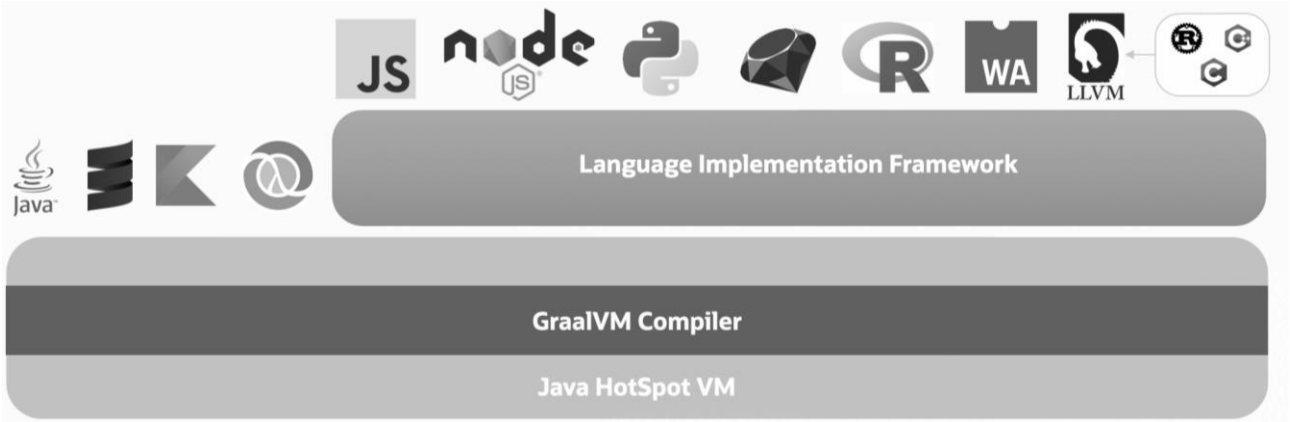
Ena izmed posebnosti GraalVM je tudi možnost prevajanja Java bytecode-a v native kodo. To omogoča ustvarjanje native izvedljivih aplikacij, ki ne potrebujejo ločenega nameščanja virtualnega stroja za njihovo delovanje. S tem se doseže še večja učinkovitost in zmanjša odtis pomnilnika, kar je ključnega pomena za izvajanje aplikacij v oblaku in v mikrostoritvenih arhitekturah.

GraalVM se uporablja v različnih scenarijih, kot so optimizacija izvajanja Java aplikacij, izvajanje več programskih jezikov na istem VM-ju, izdelava native izvedljivih aplikacij in celo kot razvojno orodje za ustvarjanje novih programskih jezikov in prevajalnikov programske kode.

Zaradi svoje zmogljivosti, vsestranskosti in inovativnih trikov je GraalVM postala priljubljena izbira za razvijalce, ki želijo izboljšati odzivnost svojih aplikacij in optimizirati njihovo delovanje v različnih okoljih [6].

Glavne razlike GraalVM v primerjavi z klasičnim JDK so:

1. GraalVM Compiler, JIT prevajalnik za Java
2. GraalVM Native Image, ki omogoča prevajanje Java aplikacij v izvorno kodo pred časom (AOT - ahead-of-time)
3. Ogrodje za implementacijo jezikov Truffle in GraalVM SDK, za dodajanje izvajanja programskih jezikov
4. LLVM Runtime in JavaScript Runtime



Slika 1: Shema arhitekture GraalVM

Vir: [7].

2.1.2 Mandrel

Mandrel je različica GraalVM skupnostne izdaje - GraalVM CE. Glavni cilj Mandrela je zagotoviti izdelave *native-image*-a, posebej prilagojenega za podporo Quarkus-a. Namen je uskladiti podporo za *native-image* med GraalVM z OpenJDK in knjižnicami Red Hat Enterprise Linux, da se izboljša podpora za native Quarkus aplikacije. Mandrel se lahko najbolje opiše kot različica običajnega OpenJDK s posebej zapakiranim graditeljem GraalVM Native Image (native-image).

Mandrel izdaje so zgrajene iz kode, ki izhaja iz izvorne kode GraalVM, pri čemer so narejene le manjše spremembe, vendar nekatere pomembne izključitve. Polna distribucija GraalVM presega *native-image*: vključuje podporo za več programskih jezikov (polyglot), ogrodje Truffle, ki omogoča učinkovito izvajanje interpreterjev, prevajalnik LLVM za native image, JIT prevajalnik libgraal kot zamenjavo za HotSpot-ov C2 strežni prevajalnik in še veliko več. Mandrel predstavlja majhen podsklop funkcionalnosti, ki se potrebujejo za uporabo z native-image [8].

Mandrel-ova izdaja *native-image* tudi ne vključuje naslednjih funkcij:

- Eksperimentalni strežnik za gradnjo slik, tj. možnost *--experimental-build-server*
- Prevajalnik LLVM, tj. možnost *-H:CompilerBackend=llvm*
- Implementacija musl libc, tj. možnost *--libc=musl*
- Podpora za ustvarjanje statičnih native slik, tj. možnost *--static*
- Podpora za jezike, ki niso osnovani na JVM, in polyglot, tj. možnost *--language:<languageId>*

Mandrel je zgrajen nekoliko drugače od GraalVM, ki uporablja standardno izdajo projekta OpenJDK verzije JDK17. To pomeni, da ne izkoristi nekaj majhnih izboljšav, ki jih je Oracle dodal v različico OpenJDK, ki se uporablja za gradnjo njihovih lastnih GraalVM distribucij. Večina teh izboljšav je v modulu JVMCI, ki omogoča zagon Graal prevajalnika znotraj OpenJDK. Ostale so manjše kozmetične spremembe v delovanju. Te izboljšave lahko v nekaterih primerih povzročijo manjše razlike v postopku generiranja native slik. Sama izvedba slik naj ne bi povzročila opaznih razlik v načinu izvajanja.

2.1.3 Microprofile

Eclipse MicroProfile je odprtokodni projekt, ki je namenjen razvoju mikrostoritvenih aplikacij na platformi Java. Projekt je bil ustanovljen leta 2016 kot skupna pobuda več tehnoloških podjetij in organizacij, vključno z Eclipse Foundation, Red Hat, IBM, Payara, Tomitribe in drugimi [9].

Glavni cilj Eclipse MicroProfile je poenostaviti razvoj mikrostoritvenih aplikacij in zagotoviti standardizirano platformo za izgradnjo in upravljanje mikrostoritvenih arhitektur v Java okolju. MicroProfile temelji na obstoječih Java EE (Java Platform, Enterprise Edition) tehnologijah in specifikacijah, ter dodaja nove funkcionalnosti in lastnosti, ki so posebej zasnovane za potrebe mikrostoritvenih aplikacij.

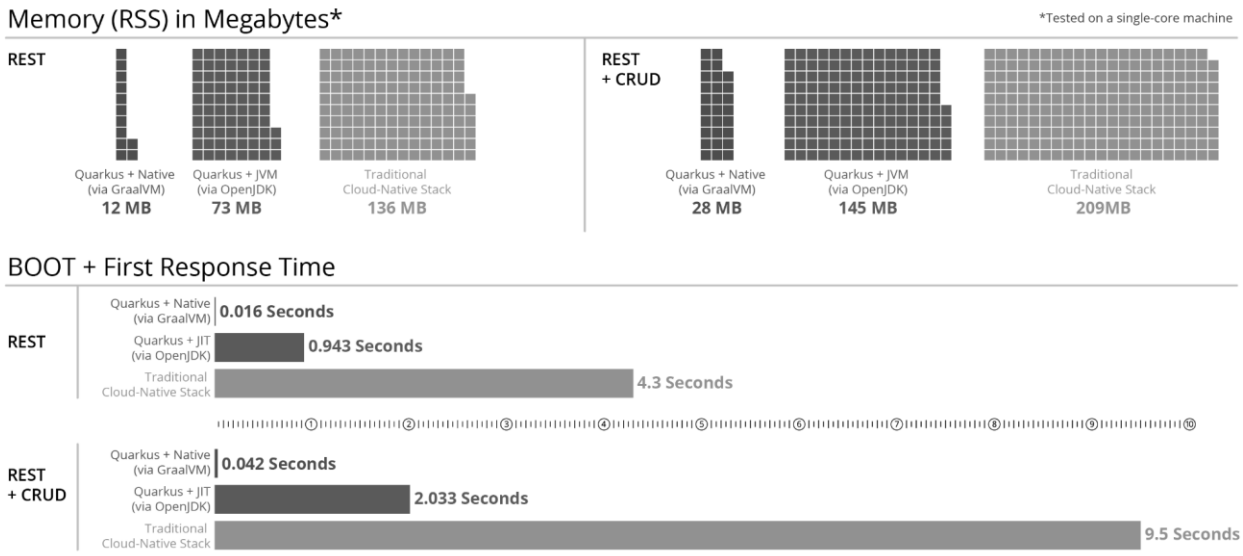
Med glavnimi lastnostmi in specifikacijami, ki jih prinaša Eclipse MicroProfile, so podpora za zaganjanje mikrostoritvenih aplikacij v oblaku, upravljanje konfiguracije, sledenje (tracing), upravljanje z napakami, varnost in še več. Vse te lastnosti so zasnovane z mislijo na prilagodljivost, lahkotnost in učinkovitost, ki so ključnega pomena za razvoj in izvajanje mikrostoritvenih aplikacij.

Eclipse MicroProfile je postal zelo priljubljen med razvijalci, ki se ukvarjajo z razvojem oblačnih in mikrostoritvenih aplikacij. Projekt je v aktivnem razvoju in se hitro razvija, saj skupnost neprestano prispeva z novimi funkcionalnostmi in izboljšavami. Standardizacija, ki jo prinaša Eclipse MicroProfile, omogoča razvijalcem, da svoje mikrostoritvene aplikacije izdelujejo na enoten način, kar olajša prenosljivost aplikacij med različnimi platformami in omogoča boljšo vzdrževanost ter skalabilnost.

2.1.4 Quarkus

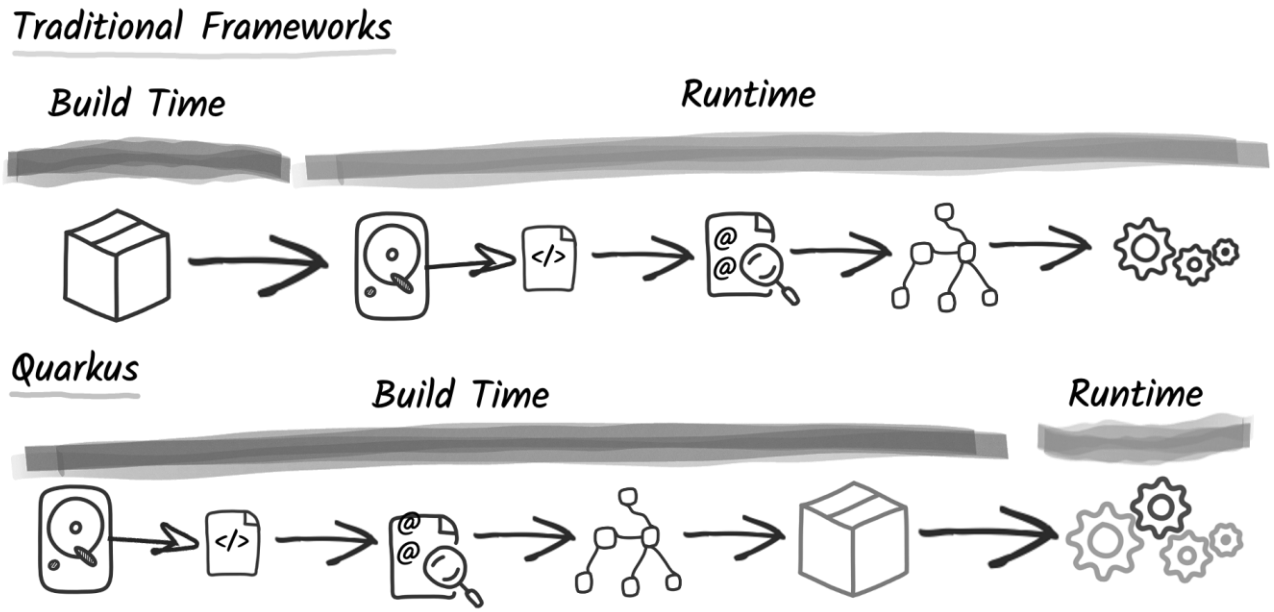
Quarkus je ogrodje, ki temelji na številnih povezanih in ločenih projektih ter jih povezuje v uporabno celoto. Primaren fokus ogrodja je razvoj oblačnih rešitev, a to ne pomeni, da ni mogoče razvijati "klasičnih" aplikacij. Zadnja večja iteracija ogrodja je verzija 3, ki prinaša posodobitve z zunanjimi knjižnicami in ogrodji ter prvič dolgoročno podprto verzijo, ki zagotavlja stabilizacijo in podporo za daljše časovno obdobje.

Quarkus omogoča izgradnjo aplikacije, ki je v primerjavi z klasičnim načinom razvoja hitrejša v času izvajanja.



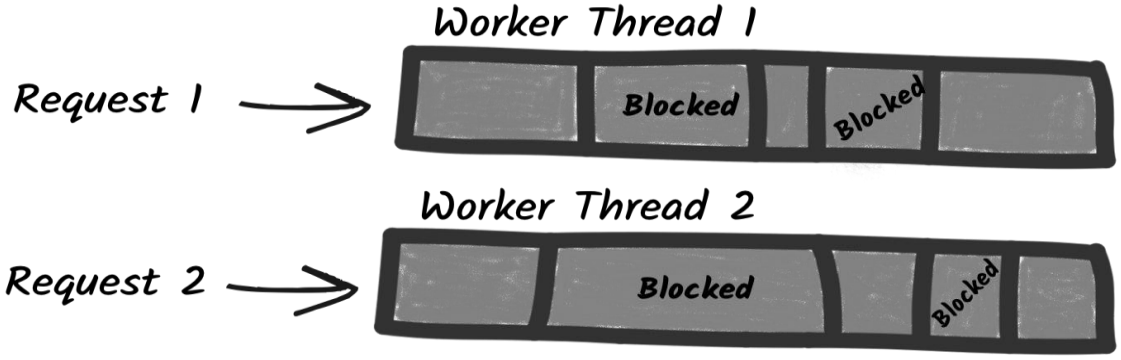
Slika 2: Razlike v načinu zagona aplikacij
Vir: [10].

Praviloma velja, da se prihranek na eni strani pozna na drugi strani. Pri Quarkusu to pomeni, da potrebuje več časa za prevajanje in pripravo zagonske slike aplikacije. Daljše prevanje pa je posledica, ker Quarkus prenese proces razrešitve odvisnosti in preračun izvajanja programske kode v čas prevajanja.



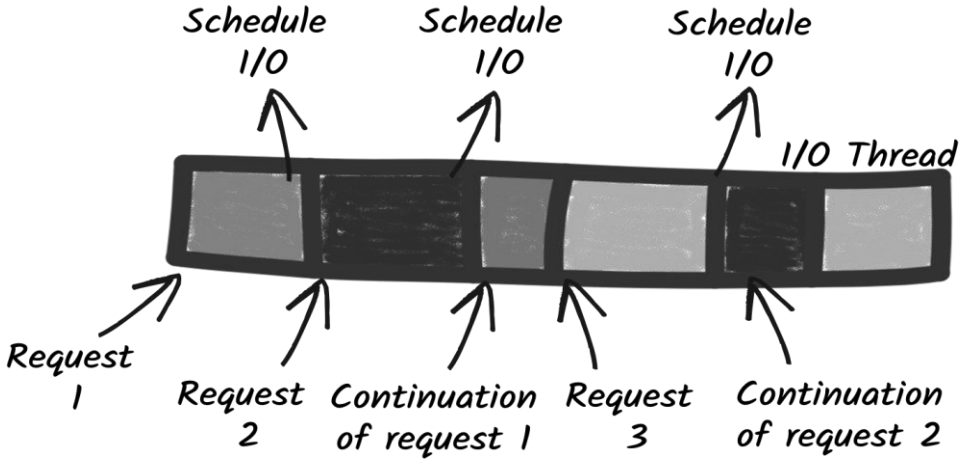
Slika 3: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [11].

Tak način priprave zagonske aplikacije pomeni, da se pred zagonom pripravi praktično vse, kar se v klasični Java aplikaciji zgodi ob zagonu. Prenos teh korakov v čas gradnje in uporaba t.i. reaktivnih načinov izvajanja programske kode, so glavni stebri razlik in prednosti proti trenutnim klasičnim načinom razvoja.
V tradicionalnem oz. imperativnem pristopu ogrodja dodelijo nit za obdelavo zahtevka. Tako celoten postopek zahtevka poteka na tej delovni niti. Ta model se ne skalira zelo dobro. Dejansko za obdelavo več sočasnih zahtevkov potrebujete več niti; in tako je vzporednost vaše aplikacije omejena s številom niti. Poleg tega so te niti blokirane, takoj ko vaša koda komunicira z oddaljenimi storitvami. To vodi v neučinkovito uporabo virov, saj bo morda potreboval več niti, in vsaka nit, saj so preslikane na niti operacijskega sistema, ima strošek v smislu pomnilnika in procesorske moči.



Slika 4: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [12].

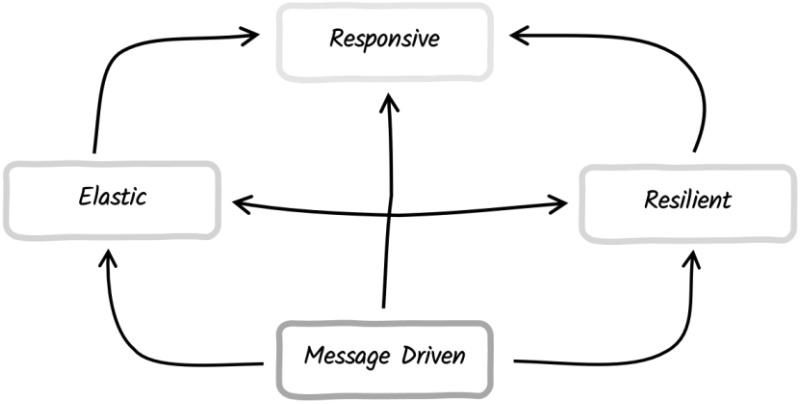
Na drugi strani pa reaktivni model temelji na neblokirajočih I/O operacijah in drugačnem modelu izvajanja. Neblokirajoči I/O zagotavlja učinkovit način za ravnanje s sočasnim I/O. Minimalna količina niti, imenovanih I/O niti, lahko obdela veliko sočasnih I/O operacij. S takšnim modelom se obdelava zahtevka ne prenaša na delovno nit, ampak uporablja te I/O niti neposredno. S tem se prihrani pomnilnik in procesorska moč, saj ni potrebe po ustvarjanju delovnih niti za obdelavo zahtevkov. Prav tako izboljša vzporednost, saj odpravlja omejitev glede števila niti. Nazadnje, izboljša tudi čas odziva, saj zmanjša število preklopov niti.



Slika 5: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [13].

Quarkus reaktivnost sledi "Reactive Manifesto", ki opisuje Reaktiven sistem kot distribuiran sistem, ki ima štiri lastnosti:

1. Odzivni (Responsive) - morajo se odzivati pravočasno.
2. Elastični (Elastic) - prilagajajo se nihajoči obremenitvi.
3. Vzdržljivi (Resilient) - omogočajo grajen prehod ob napakah.
4. Asinhrono posredovanje sporočil (Asynchronous message passing) - komponente reaktivnega sistema medsebojno delujejo preko sporočil.

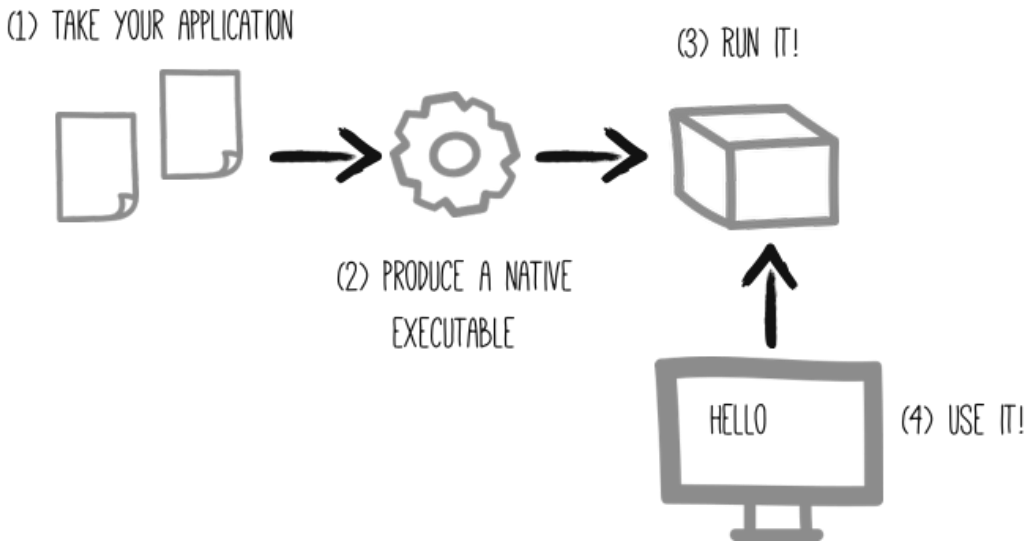


Slika 6: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [14].

Poleg tega dokument o reaktivnih načelih (Reactive Principles white paper) našteva niz pravil in vzorcev, ki pomagajo pri gradnji reaktivnih sistemov.
Reaktivna načela se nahajajo na spletni strani https://www.reactiveprinciples.org/
Quarkus podpora zagon aplikacije v dveh načinih:

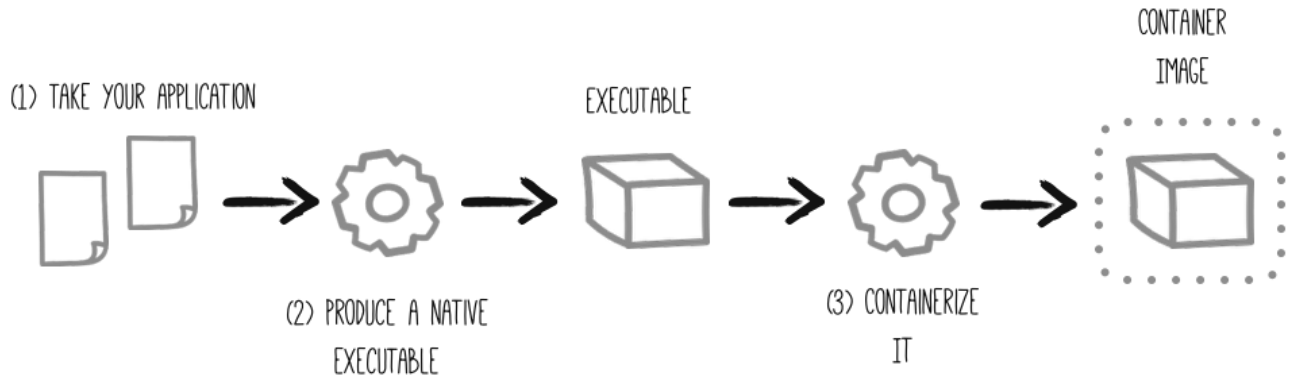
1. tradicionalen preko uporabe Java JDK
2. izdelavo binarne oblike za gostiteljevo arhitekturo
    a. specializirana oblika je "native zabojnik", ki naredi linux x64 native image (tudi na ne-linux x64 sistemih)

Native zagonska datoteka zajema celotno aplikacijo in vse odvisnosti. Nabor odvisnosti se je izdelal v času gradnje.



Slika 7: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [15].

Zabojnik je specializirana oblika, kjer se vse potrebne datoteke za zagon zapakirajo v sliko zabojnika.



Slika 8: Zagon običajnih Java aplikacij in Quarkus Java aplikacije
Vir: [16].

Zagon preko načina native-image ima specifične lastnosti, ki lahko določenih scenarijih uporabe privede do neželenih stranskih učinkov, zato tak način zagona ni najboljša rešitev za vse. Glede na lastnosti in potrebe delovanja aplikacije se izvede odločitev, kdaj uporabiti kateri način zagona.

JIT - OpenJDK HotSpot

- Učinkovita poraba CPU-ja.
- Najboljši Garbage Collector.
- Večja poraba internega spomina.
- Obstoječa monitoring in instrumentalna orodja in ogrodja.
- Knjižnice podpirajo samo klasičen JDK.

AOT - GraalVM native image

- Zahteva po veliki količini spomina in obdelava velike količine podatkov - recimo obdelava slik ali velikih datotek.
- Veliko število zahtev - zahtev/sekundo.
- Hitrejši prvi zagon.

3 Alternativne rešitve

V zadnjem času so bile ustvarjene alternativne rešitve, ki konkurirajo direktno ali posredno. Pri tem je potrebno imeti v mislih, da je Quarkus, čeprav je odprtokodno ogrodje, primarno otrok Red Hat-a. Že iz tega je pričakovati, da se pojavijo alternativne rešitve. Glavne alternativne rešitve, ki obstajajo so:

- Micronaut,
- Spring Boot in
- OSGi.

OSGi je Java specifikacija in ogrodje, ki ima jedro v dinamičnem načinu nalaganja storitev in prilagajanja na spremembe v okolju brez vnovičnih zagonov. Vsa ta dinamičnost je v Quarkus-u praktično nemogoča, saj se vse dinamične variacije razdelajo v času gradnje; kar predstavlja konceptualno drugačnost. OSGi nima nekega lastnika, čeprav nekako spada pod Eclipse fundacijo, največ doprinosa izvorne kode pride iz krogov Red Hat-a.

Spring Boot je najstarejše ogrodje in knjižnica, od vseh tukaj opisanih. Razvoj se je začel leta 2012, nato se je v letu 2013 stabiliziralo in uredilo izvorno kodo iz različnih virov ter nato leta 2014 predstavljena verzija 1.0. Spring Boot gradi nad obstoječim in dobro razvitim ter uporabnim ogrodjem Spring Framework. V zadnjem času - letu, tudi Spring Boot podpira in omogoča izdelavo native-image preko GraalVM, podobno kot Quarkus. Spring Boot je preko lastništva pod varstvom VMware-a, ki je del skupine Dell.

Micronaut je praktično edina direktna konkurenca. Glede načina razvoja aplikacije in števila ter verzij integriranih zunanjih knjižnic je razlik zelo malo - obe ogrodji sta v aktivnem razvoju in spremembe se dogajajo iz meseca v mesec; če je še pred meseci veljalo da eno ogrodje ne podpira neke knjižnice, se je naslednji mesec dejstvo obrnilo in tako se ponavlja.
Glavna razlika, kjer je vidna razlika je v celovitosti celotne vertikale, kjer mislim na podporo in sodelovanje razvojnega okolja, Java API-ja (vmesniki, anotacije, analiza in predlogi), dokumentacije - učni material s primeri in na koncu zagon aplikacije - v oblačnem okolju (aka OpenShift), je zaenkrat prednost na Quarkus, a to se lahko hitro spremeni. Koncepti so praktično isti - oboje temelji na GraalVM (tudi Spring Boot, če se primerja po tem kriteriju). Glavna sestavina je GraalVM in pa vzorci ter knjižnice, ki podpirajo vzorce - recimo Vert.x in OpenTelemetry ter podobno.
Micronaut trenutno nima tako podrobno razvitega prevajalnika, ki bi bil sposoben tako podrobno in učinkoviti prevesti in prenesti izvorno kodo v izvajanje. Dodatno k temu Micronaut nima primerljive rešitve za Quarkus Dev Services in pa tako dobre podpore za razvojna okolja - IDE-je. Micronaut je lastniško pod Oracle.

4 Praktični primer

Prikazan je praktičen primer razvoja aplikacije z uporabo ogrodja Quarkus 3. Začne se z kreiranjem Java projekta z uporabo Quarkus CLI. Ustvari se PostgreSQL baza s tabelo za hrambo podatkov. Doda se razred za modeliranje podatkovnega modela in za dostop do podatkov v SQL bazi. Na koncu se doda reaktivna REST storitev, ki omogoča osnovne CRUD operacije. Izvorna koda se prevede in naredi paket za zagon. Zagonska datoteka se izvede in z uporabo programa curl se pokliče REST spletni vmesnik.

Za kreiranje in delo z Quarkus projekti sta na voljo dva načina:

1. Maven CLI - uporablja se Apache Maven in se preko razširitev kreira in upravlja s projektom
2. Quarkus CLI - uporablja se namenski CLI, ki ima jasno sintakso in omogoča "samodejno dopolnjevanje ukazov" (auto-complete). Uporabljam oboje, čeprav se pogosteje obrnem na Quarkus CLI, ki se bo uporabljal tudi v nadaljevanju.

Najlažji način namestitve Quarkus CLI-ja je z uporabo aplikacije SDKMAN! [17]:

```
$ sdk install quarkus
```

Ko imam nameščen Quarkus CLI, lahko ustvarim projekt z ukazom:

```
$ quarkus create app si.beranic:rest-json-quickstart \
    --extension='resteasy-reactive-jackson' \
    --no-code
$ cd rest-json-quickstart
```

Argument --extension='resteasy-reactive-jackson' doda Maven odvisnost v datoteko pom.xml:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-reactive-jackson</artifactId>
</dependency>
```

Začnem z kreiranjem podatkovnega modela. Za komunikacijo z SQL bazo se uporablja knjižnica Hibernate, za oblikovanje programskega vmesnika pa knjižnica Panache. Podatkovni model SQL tabele:

```
@Entity
@Cacheable
public class Fruit extends PanacheEntity {

       @Column(length = 40, unique = true)
       public String name;

}
```

V nastavitveni datoteki src/main/resources/application.properties vpišem parametre za dostop do PostgreSQL baze:

```
quarkus.datasource.db-kind=postgresql
quarkus.hibernate-orm.database.generation=drop-and-create
```

Ti parametri so za Quarkus zadostni, da kreira vse potrebno za uspešno komunikacijo in izmenjavo podatkov. V datoteko import.sql vpišem začetne SQL stavke, ki napolnijo SQL tabelo:

```
INSERT INTO fruit(id, name) VALUES (1, 'Cherry');
INSERT INTO fruit(id, name) VALUES (2, 'Apple');
INSERT INTO fruit(id, name) VALUES (3, 'Banana');
ALTER SEQUENCE fruit_seq RESTART WITH 4;
```

Aplikacijo lahko poženemo v t.i. dev mode načinu:

```
$ quarkus dev
```

Quarkus ustvari SQL strežnik z bazo in tabelo primerno za nadaljnji razvoj. Dodam REST API storitev:

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.ws.rs.Path;

@Path("/fruits")
@ApplicationScoped
public class FruitResource {

  @GET
  public Uni<List<Fruit>> get() {
    return Fruit.listAll(Sort.by("name"));
  }

  @GET
  @Path("/{id}")
  public Uni<Fruit> getSingle(Long id) {
    return Fruit.findById(id);
  }

  @POST
  public Uni<RestResponse<Fruit>> create(Fruit fruit) {
```

```
    return
Panache.withTransaction(fruit::persist).replaceWith(RestResponse.status(CREATED,
fruit));
  }

}
```

S tem je aplikacija primerna za testiranje. Z uporabo aplikacije curl preverimo zapis nove vrednosti, preko REST vmesnika v SQL tabelo.

```
$ curl --header "Content-Type: application/json" \
  --request POST \
  --data '{"name":"Peach"}' \
  http://localhost:8080/fruits
```

Aplikacija je pripravljena na prenos v ciljno okolje. Najprej ga prevedem, to lahko naredim na dva različna načina: a.) klasičen "java" način ali b.) "native-image" način. Za klasične način izvedem:

```
$ quarkus build
```

Za "native-image" način izvedem:

```
$ quarkus build --native
```

Na ciljnem okolju se ustvari SQL strežnik, preko zabojnika:

```
$ podman run -it --rm=true \
    --name postgres-quarkus -e POSTGRES_USER=quarkus \
    -e POSTGRES_PASSWORD=quarkus -e POSTGRES_DB=fruits \
    -p 5432:5432 postgres:15.3
```

Če sem aplikacijo prevedel v klasičen način se izvede:

```
$ java \
   -Dquarkus.datasource.reactive.url=postgresql://localhost/fruits \
   -Dquarkus.datasource.username=quarkus \
   -Dquarkus.datasource.password=quarkus \
   -jar target/quarkus-app/quarkus-run.jar
```

Če pa sem program prevedel v "native-image" način, pa izvedem sledeče:

```
$ ./target/getting-started-with-reactive-runner \
  -Dquarkus.datasource.reactive.url=postgresql://localhost/fruits \
  -Dquarkus.datasource.username=quarkus \
  -Dquarkus.datasource.password=quarkus
```

5 Zaključek

V prispevku smo pogledali začetke in namen ogrodja Quarkus ter načine uporabe. Pogledali smo alternativne rešitve in kako se Quarkus primerja z njimi. Prikazan je način izdelave splošno uporabne native aplikacije, ki se lahko izvaja ročno ali preko uporabe zabojnikov. Predstavitev služi kot priročnik začetniku pri prvih korakih uporabe ogrodja Quarkus od razvoja do zagona.

Literatura

[1] https://quarkus.io/, Quarkus - Supersonic Subatomic Java, dostopano 1.7.2023

[2] https://github.com/quarkusio/quarkus/releases, Releases · quarkusio/quarkus, dostopano 1.7.2023

[3] https://github.com/quarkusio/quarkus/releases/tag/0.0.1, Release 0.0.1 · quarkusio/quarkus, dostopano 1.7.2023

[4] https://github.com/quarkusio/quarkus/commit/161cfa303b4ea366dbd07e54bf4fe5a67ddec497, Initial · quarkusio/quarkus@161cfa3, dostopano 1.7.2023

[5] https://github.com/quarkusio/quarkus/releases/tag/3.2.2.Final, Release 3.2.2.Final · quarkusio/quarkus, dostopano 23.7.2023

[6] https://en.wikipedia.org/wiki/GraalVM, GraalVM - Wikipedia, dostopano 1.7.2023

[7] https://upload.wikimedia.org/wikipedia/commons/1/13/GraalVM_CE_architecture.png, GraalVM_CE_architecture.png, dostopano 1.7.2023

[8] https://github.com/graalvm/mandrel, graalvm/mandrel, dostopano 1.7.2023

[9] https://microprofile.io/, Home - MicroProfile, dostopano 1.7.2023

[10] https://microprofile.io/workinggroup/#mptab-members, Working Group - MicroProfile, dostopano 1.7.2023

[11] https://quarkus.io/assets/images/quarkus_metrics_graphic_bootmem_wide.png, quarkus_metrics_graphic_bootmem_wide.png, dostopano 1.7.2023

[12] https://quarkus.io/guides/images/build-time-principle.png, build-time-principle.png, dostopano 1.7.2023

[13] https://quarkus.io/guides/images/blocking-threads.png, blocking-threads.png, dostopano 1.7.2023

[14] https://quarkus.io/guides/images/reactive-thread.png, reactive-thread.png, dostopano 1.7.2023

[15] https://quarkus.io/guides/images/reactive-systems.png, reactive-systems.png, dostopano 1.7.2023

[16] https://quarkus.io/guides/images/native-executable-process.png, native-executable-process.png, dostopano 1.7.2023

[17] https://quarkus.io/guides/images/containerization-process.png, containerization-process.png, dostopano 1.7.2023

[18] https://sdkman.io/install, Installation - SDKMAN!, dostopano 1.7.2023