Migracja

Z RH SSO do RHBK

OSEC

FORUM 2024

Warsaw 11.06.2024

Źródło zdjęcia: unsplash.com

Agenda

- Nudne rzeczy:
  - Agenda ✔
- Rzeczy teoretyczne:
  - Ekosystem JVM
  - Architektura RH SSO / Keycloak <= 18
  - Architektura RH BK / Keycloak >= 22
- Rzeczy ciekawe:
  - Konfiguracja
  - Implementacja Identity Provider
- Wymagane rzeczy
  - Pytania i odpowiedzi

Cel prezentacji

1. Określenie przyczyn zmian w Keycloak
2. Weryfikacja wprowadzonych zmian
3. Ocena wyników

1. Co się zmieniło?
2. Jak się zmieniło?
3. Jak korzystać?

Wprowadzenie do protokołu - prezentacja z 2023 r.
https://www.youtube.com/watch?v=MiKHPr50sIU
^ Keycloak w mniej typowych scenariuszach ^

O mnie

Łukasz Dywicki

W IT od 2005 roku. Pracuję z Keycloak od 2018 roku / v3.4.3, głównie na bazie kodu źródłowego. Współpracuję z OSEC od 2020 r.

Główne osiągnięcia do tej pory:

- [2018] Integracja logowania dla aplikacji mobilnej.
- [2019] Dostosowanie KC dla dużej sieci sklepów.
- [2020] Integracja OAuth Device Grant (KEYCLOAK-7675).
- [2020] Uruchomienie "Cross DC" dla jednego z banków w PL.
- [2021] Migracja powyższego do Red-Hat SSO.
- [2022] Uruchomienie multi-tenant KC dla projektu w chmurze.
- [2023] Integracja w infrastrukturze MObywatel.

cd. O mnie

Łukasz Dywicki

... Jestem również autorem szkolenia, które uczy uczestników jak tworzyć rozszerzenia dla Keycloak w *bardziej typowych scenariuszach*.

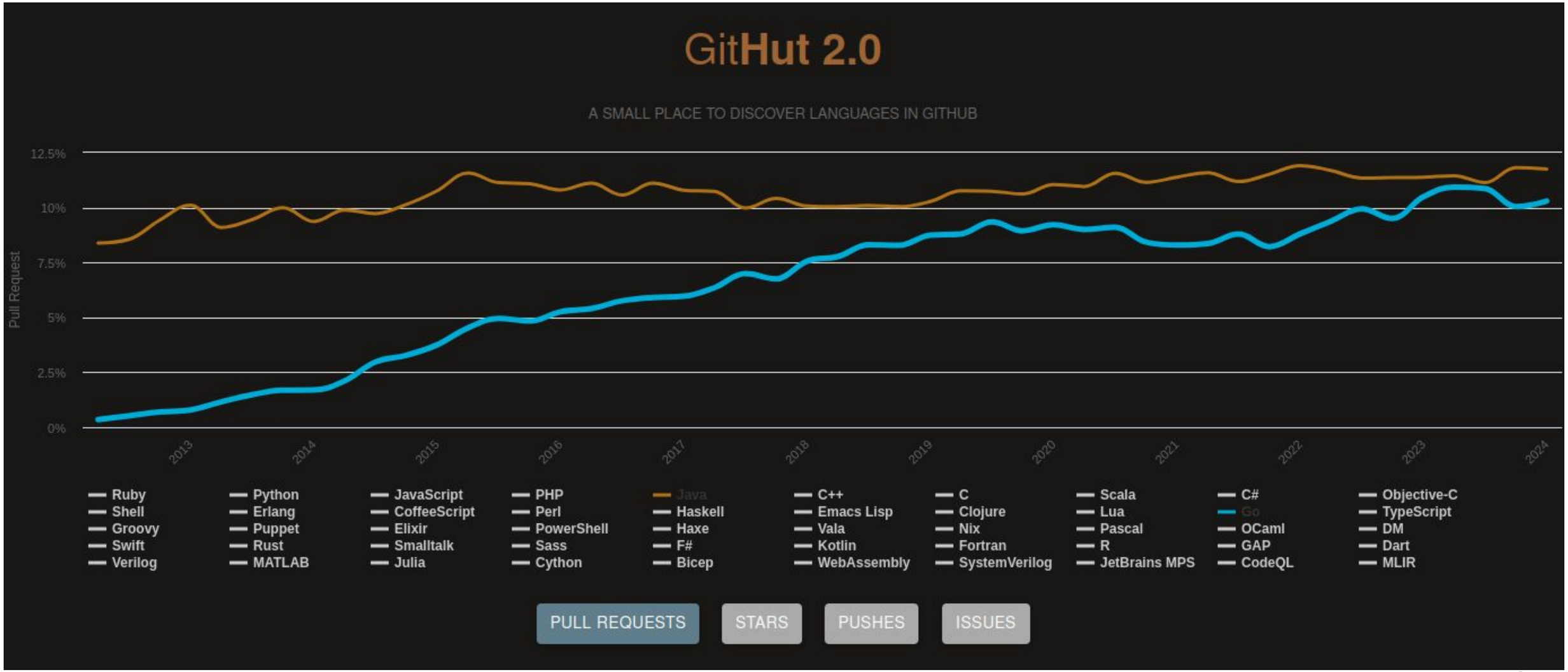
Rzeczy teoretyczne

FORUM 2024

Ekosystem JVM

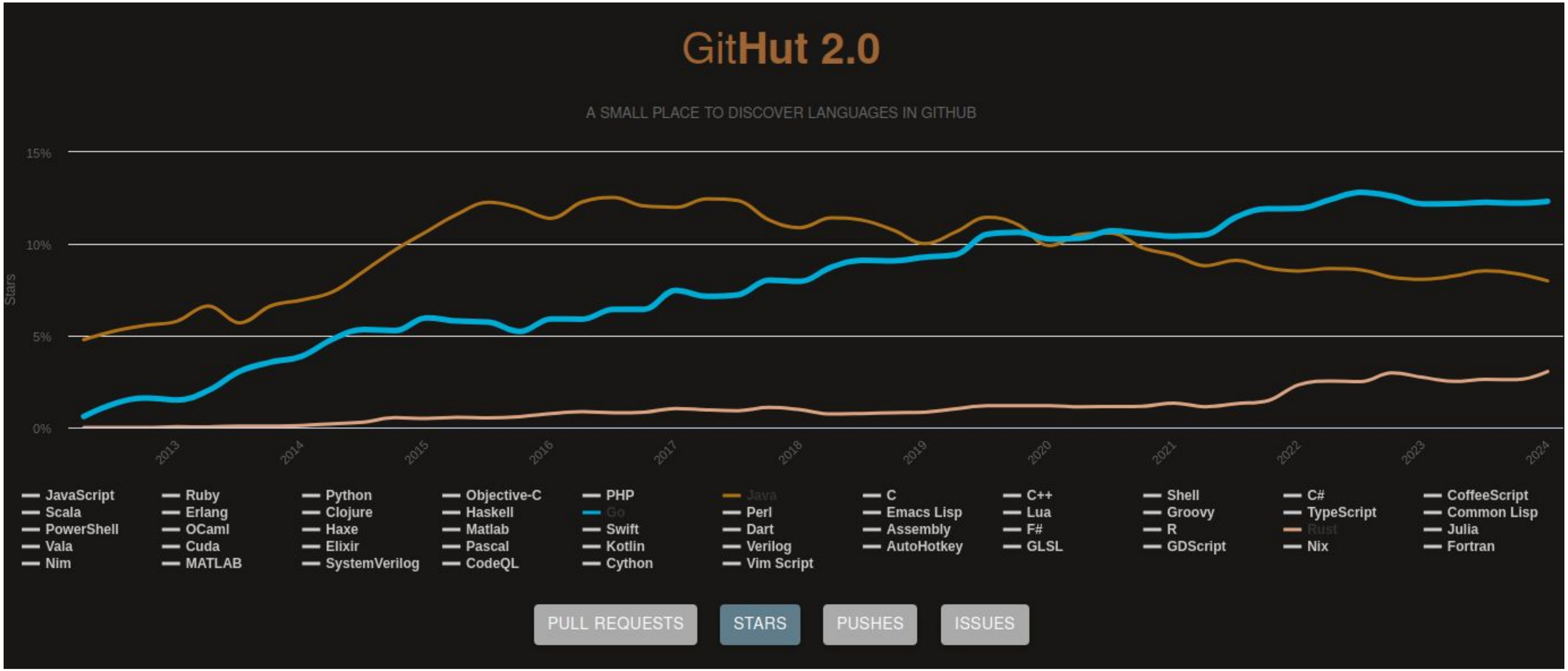
- Konkurencja
- Wersje Java
- Ekosystem

Konkurencja



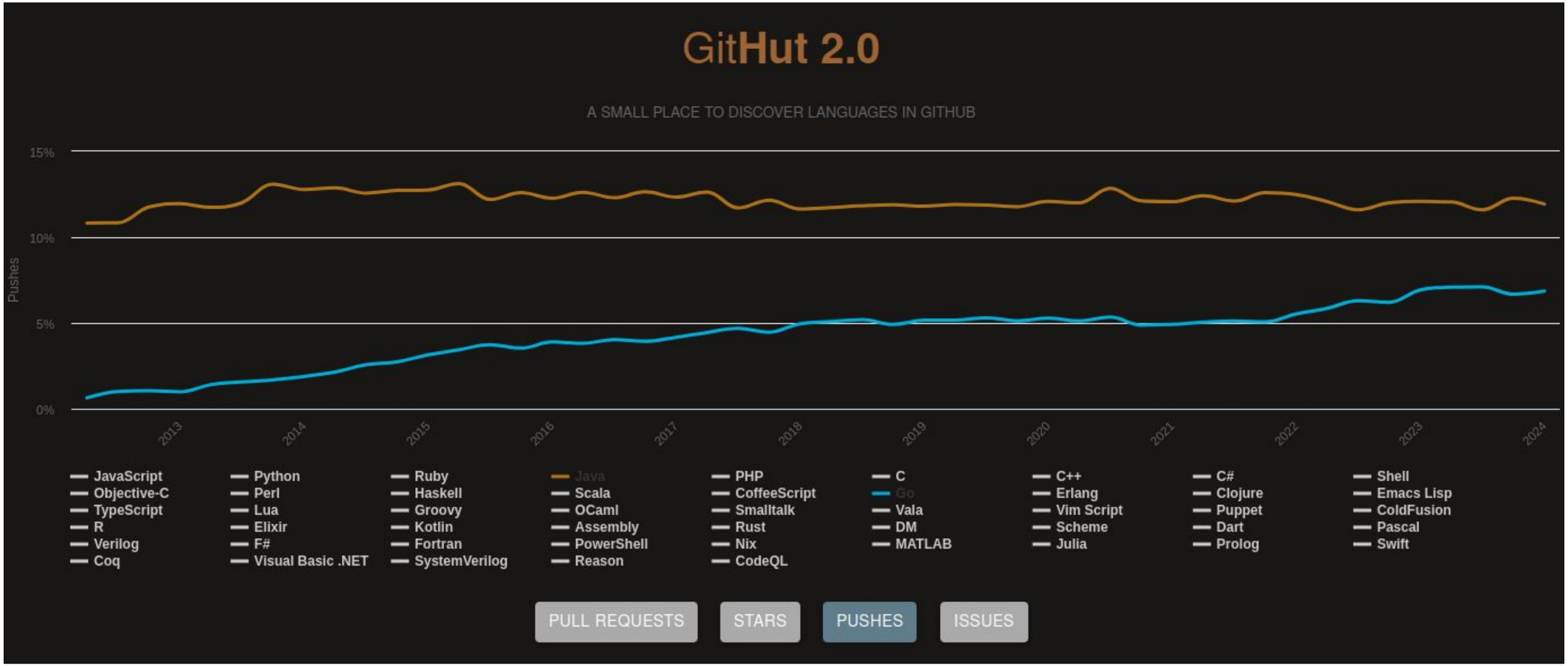
Źródło:
https://madnight.github.io/githut/

Konkurencja



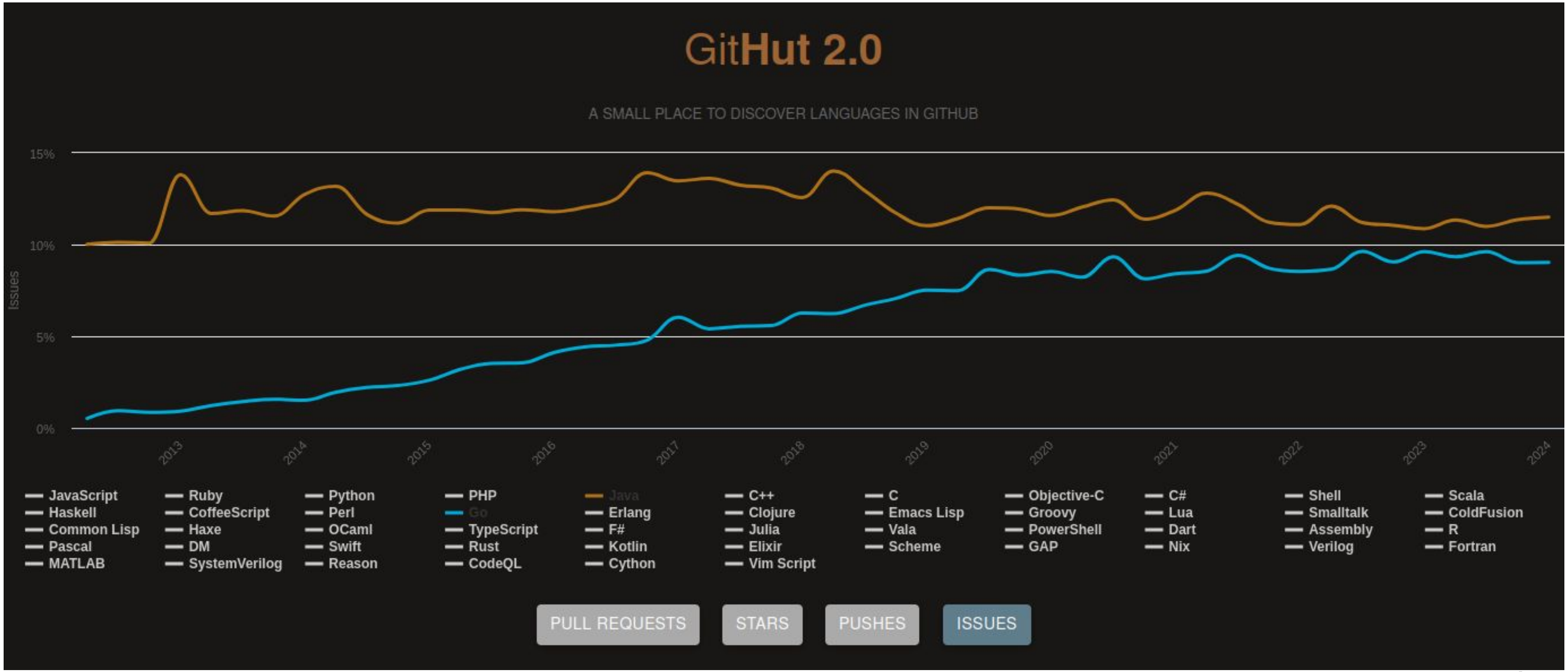
Źródło:
https://madnight.github.io/githut/

Konkurencja



Źródło:
https://madnight.github.io/githut/

Konkurencja



Źródło:
https://madnight.github.io/githut/

Konkurencja cd.

Pretendenci:

- Python - łatwość tworzenia (j. skryptowy)
- Go:
  - szybki start
  - pauseless gc
- Rust
  - ochrona pamięci (gc less)
  - bezpieczna wielowątkowość

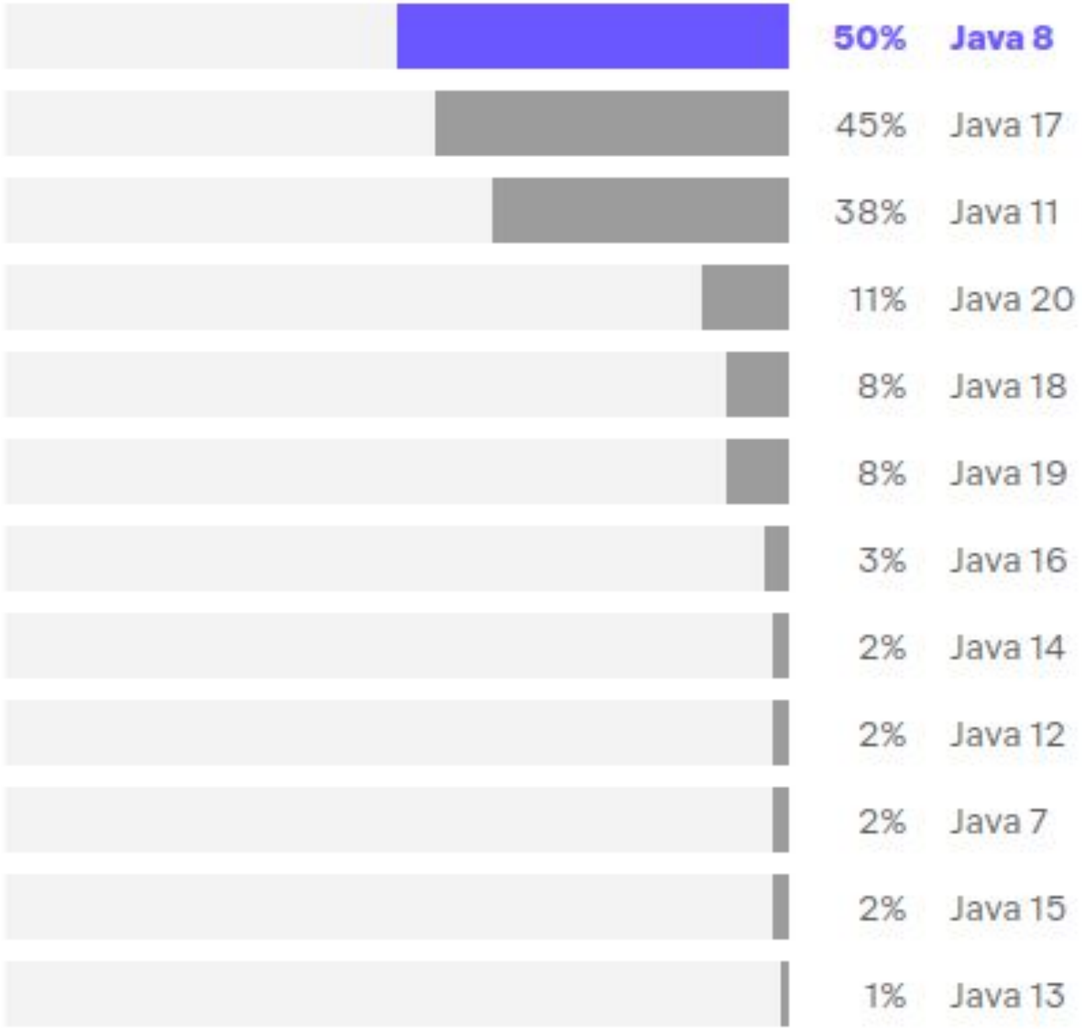
Przewagi JVM:

- JIT - optymalizacja do kodu natywnego
- (?) AOT - kompilacja do kodu natywnego / GraalVM

Problemy z JVM

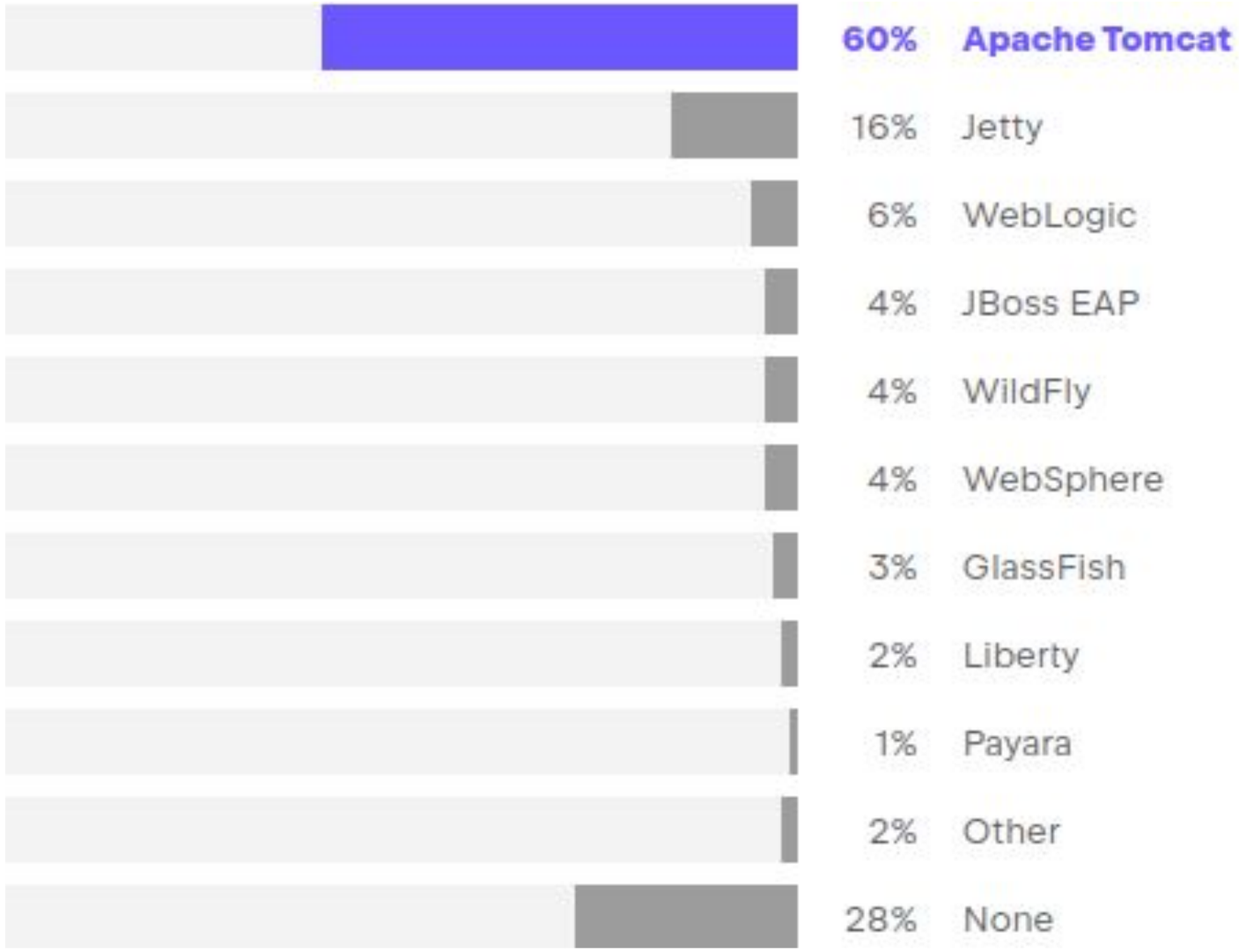
- Classpath: czas wykonania a czas budowania
  - Classpath scanning
- Zaszłości w samym JVM / języku
  - Przyspieszenie cyklu wydań
- Przerost ekosystemu
- Fragmentaryzacja ekosystemu

Wersje JVM



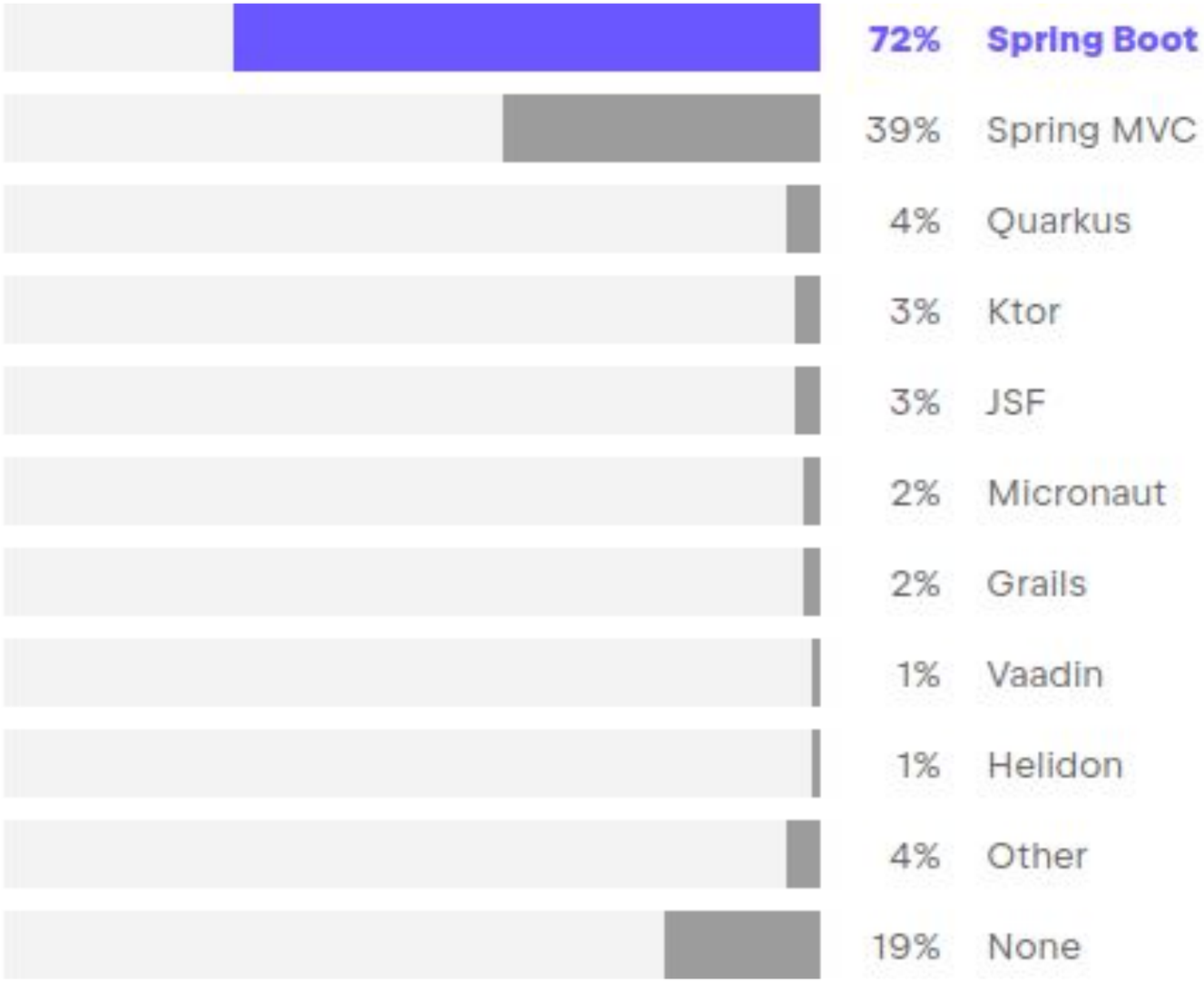
Źródło:
https://www.jetbrains.com/lp/devecosystem-2023/java/

Ekosystem



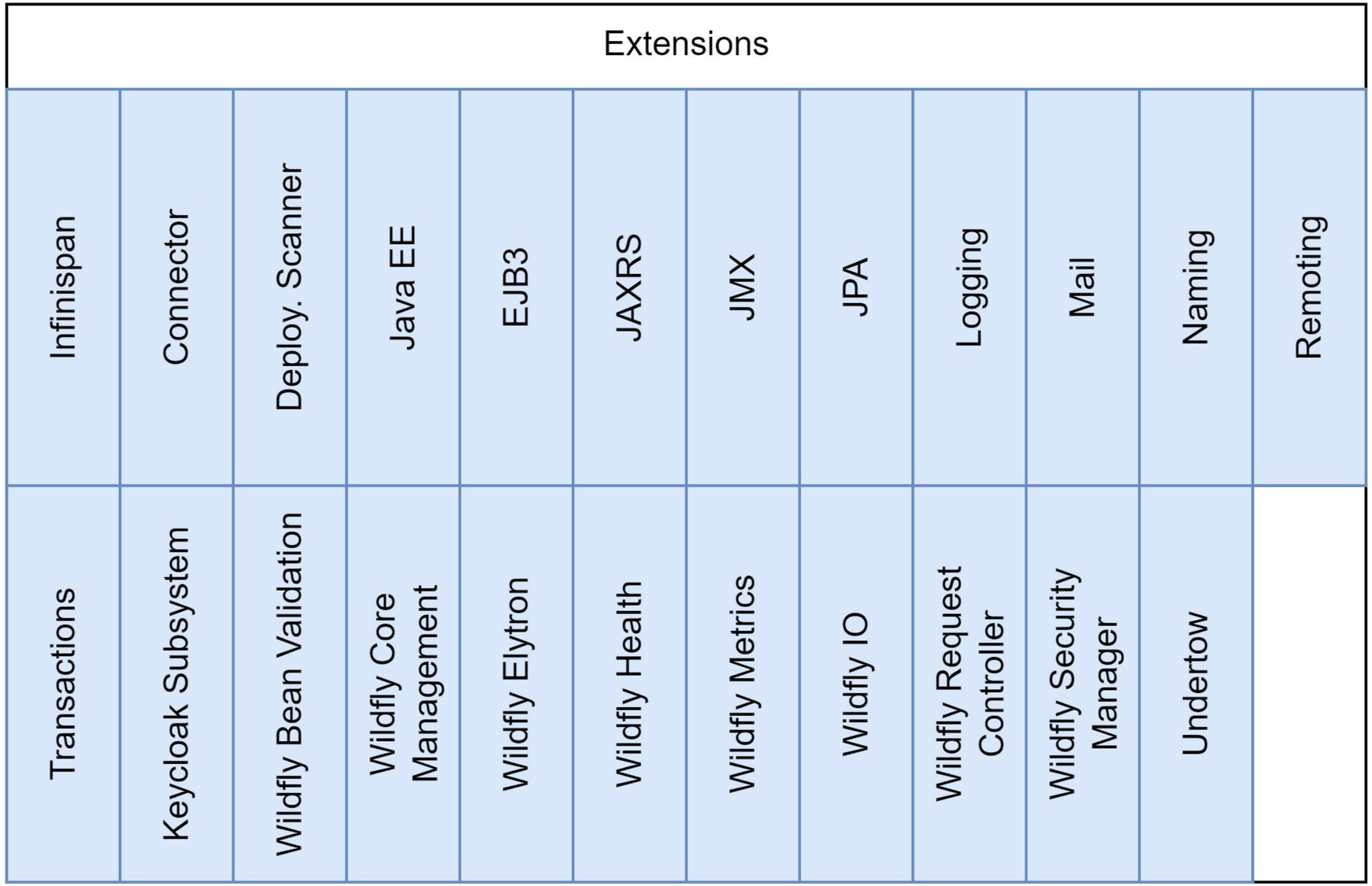
Źródło:
https://www.jetbrains.com/lp/devecosystem-2023/java/

Ekosystem

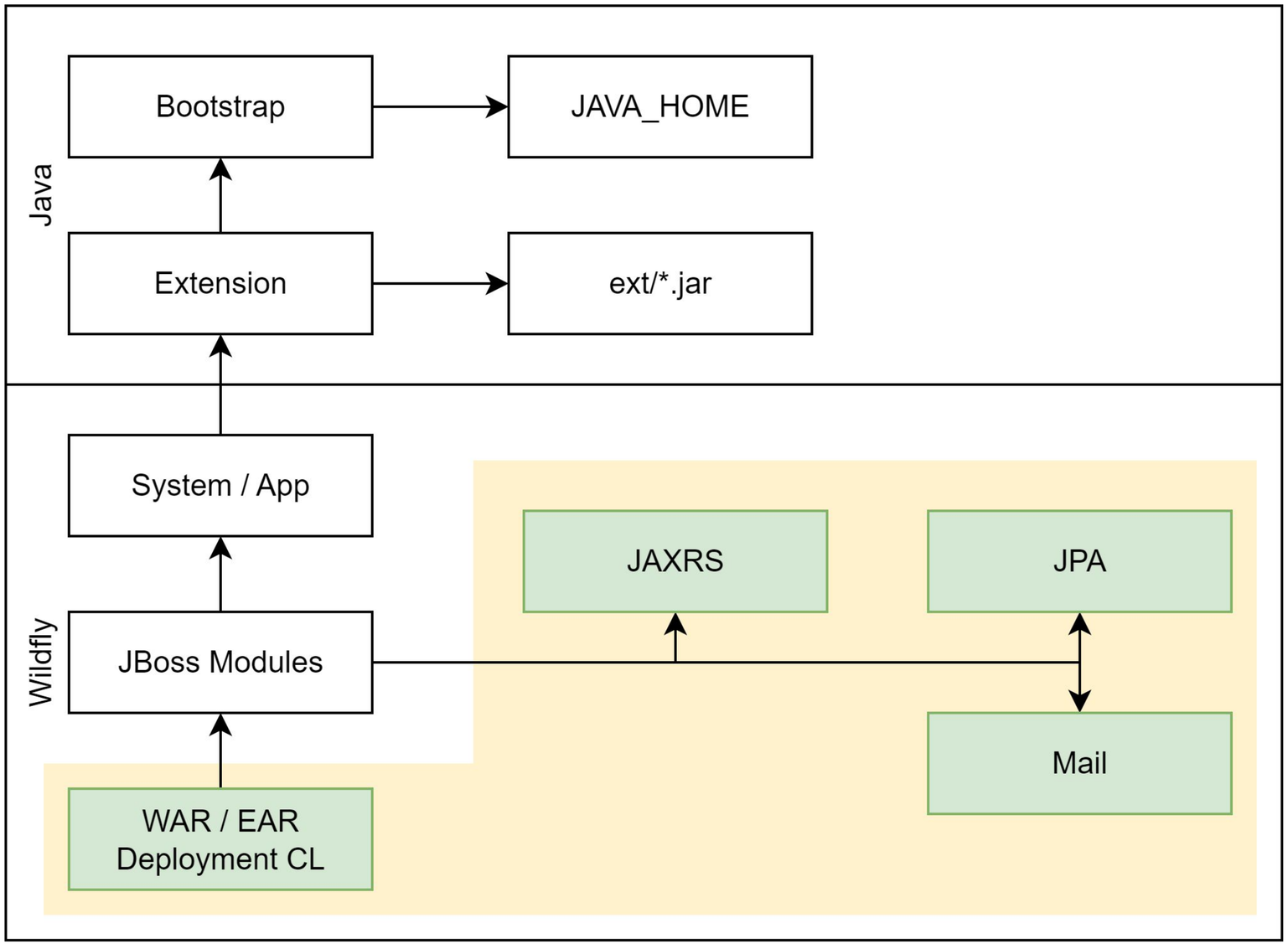


Źródło:
https://www.jetbrains.com/lp/devecosystem-2023/java/

Keycloak + Wildfly => RH SSO

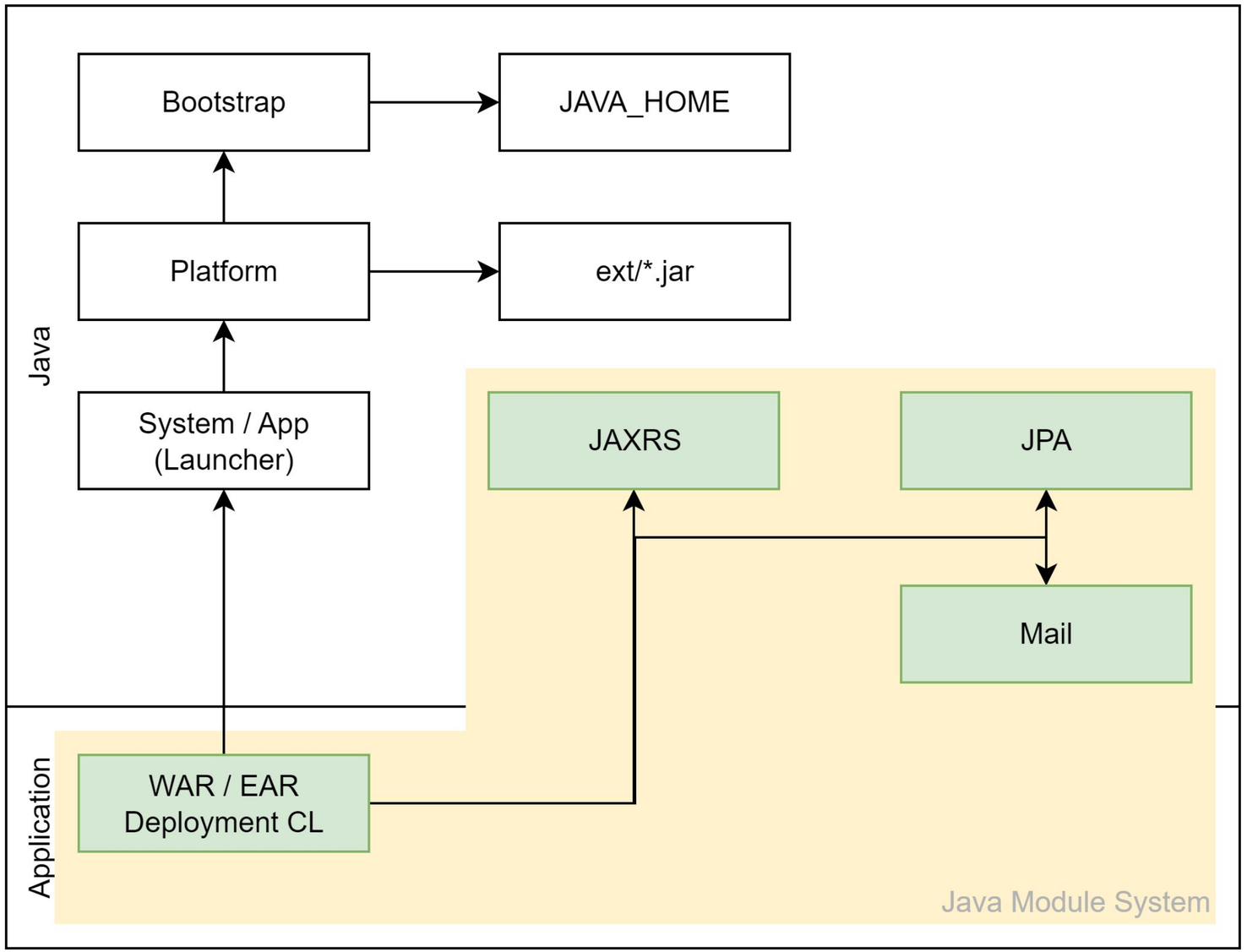


Classloaders: Wildfly + JBoss Modules

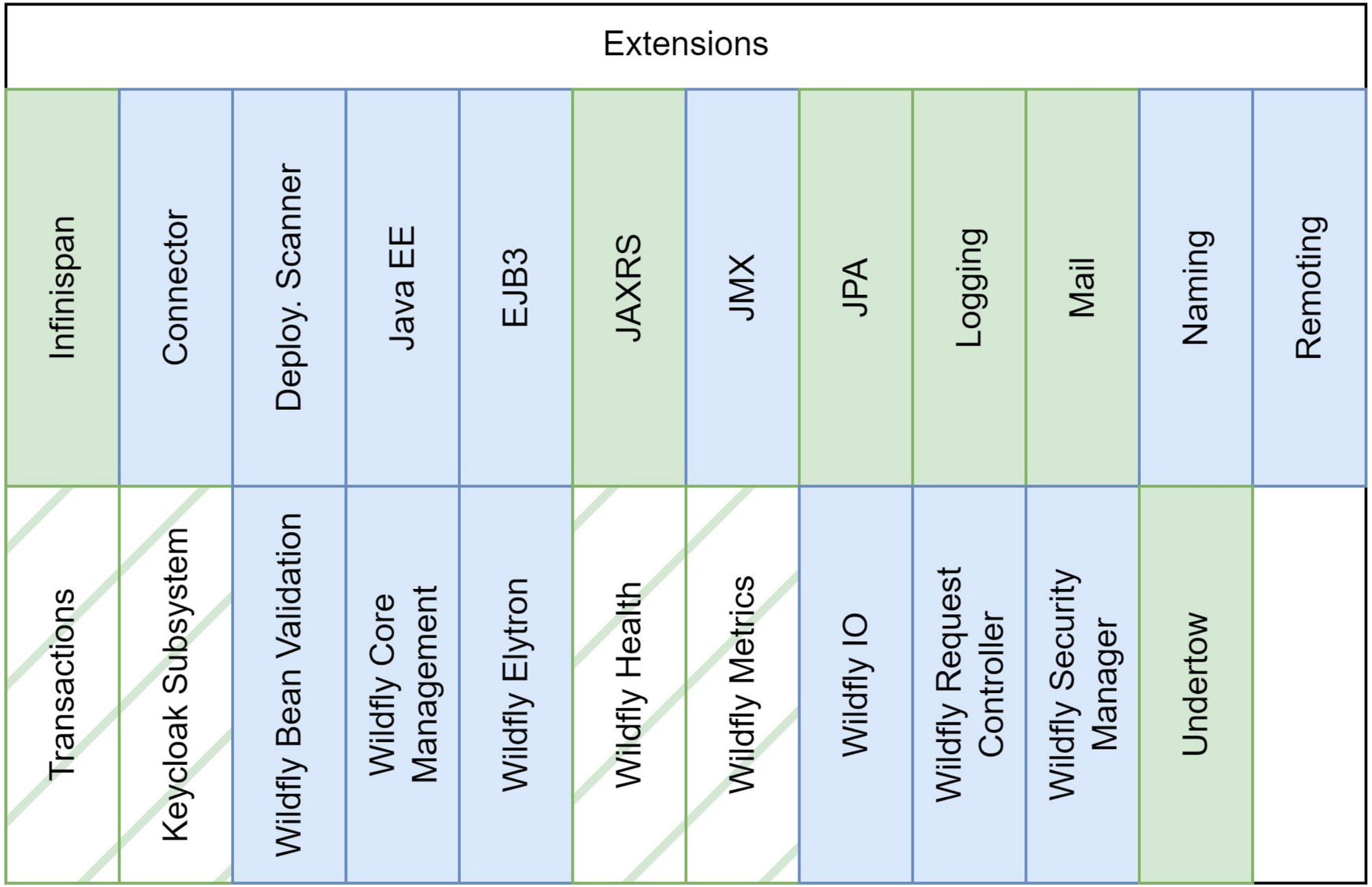


up to Java 8

Classloaders: Java 9+



Keycloak - Wildfly => RHBK



Porównanie dla Keycloak 18.0.2

- Quarkus:
  - 175 MB
- find lib/ -name '*.jar'|wc -l
  - 472
  - -33%

- Wildfly:
  - 252 MB
- find modules/ -name '*.jar'|wc -l
  - 712
  - +50%

- **Keycloak 21.1.1**:
  - 167 MB
- find lib/ -name '*.jar'|wc -l
  - 492

- **Keycloak 24.0.5**:
  - 168 MB
- find lib/ -name '*.jar'|wc -l
  - 375

Zależności w wersji 24.0.5

Grupa	Ilość plików
114	io.quarkus
39	io.smallrye
30	org.wildfly
27	org.keycloak
17	io.netty

Zależności w wersji 24.0.5 cd.

Grupa	Ilość plików
13	org.jboss
9	com.fasterxml
8	org.eclipse
7	org.infinispan
6	org.apache

Kompilacja AOT

- **Ahead of Time**
- Generuje kod na daną platformę (np. linux x86_64 / AArch64)
  - Mniejszy rozmiar wynikowy (tj. binarka)
- "Wycina" classpath scanning
  - Bardzo szybki start
- Brak dynamicznej alokacji na bazie konfiguracji ("Reflection API")
  - Typ musi być znany w czasie kompilacji
  - Im mniej dynamicznie - tym łatwiej wygenerować obraz.

- Czasochłonna i pracochłonna
  - Im większa aplikacja tym dłużej trwa
  - Znalezienie wszystkich dynamicznych alokacji
- Konieczna rekompilacji po zmianie rozszerzeń

Optymalizacja startu

- Wykorzystuje JVM
- "Wycina" classpath scanning
  - Przyspiesza start
- Ograniczenie dynamicznej alokacji
  - Typ nie musi być znany w czasie kompilacji (możliwe rozszerzenia)

- Szybka i łatwa
  - Indeksowanie rozszerzeń przy pomocy dostępnych narzędzi
  - Generowanie kodu startowego (cache)
- Brak rekompilacji
- Wciąż korzysta z kompilacji **Just-in-Time**

Start Keycloak 18

- Czas rozruchu "gołego" serwera:
  - Quarkus: ~4s (do otwarcia portu 8080)
    - ./bin/kc.sh start-dev
  - Wildfly: ~8-11s
    - ./bin/standalone.sh

Demo #1, Start Keycloak 22+

Rzeczy ciekawe

Zmiany Quarkus - Wildfly

Wildfly

- bin
- domain
- modules
- standalone
  - configuration
  - deployments
- themes
- welcome-content

Quarkus

- bin
- lib
- conf
- providers
- themes

Zmiany Quarkus - Wildfly

- Quarkus to efektywnie konfiguracja standalone
  - Brak konfiguracji 'domain'
- Zarządzanie "klastrem"
  - Za pośrednictwem operatora Kubernetes
  - Konfiguracja zarządzana przez Ansible/terraform itp.

Zmiany Quarkus - Wildfly

- Bez Wildfly nie mamy XML, w którym była:
  - Konfiguracja logów
  - Konfiguracja puli połączeń
  - Konfiguracja portów publicznych/prywatnych (management)
  - Konfiguracja context path (/auth)
  - Konfiguracja rozszerzeń (preferowanych SPI)
  - Konfiguracja buforów infinispan
  - Konfiguracja keystore (tls)

Konfiguracja Quarkus

- Konfiguracja w oparciu o plik conf/keycloak.conf
  - plik tekstowy (składnia properties)
  - format klucz=wartość
- Zmienne środowiskowe
- Rozdzielenie konfiguracji "build" oraz "runtime"
  - Lista opcji: https://www.keycloak.org/server/all-config

Konfiguracja logów

--log=<target1[, target2]>
KC_LOG=...

--log-level=<ROOT>,[category1:level[, category2:level]]
--log-console-output
--log-console-format

Konfiguracja puli połączeń

--db-url=jdbc:..
KC_DB_URL=jdbc:...

--db-username
--db-password
--db-schema
--db-pool-initial-size
--db-pool-min-size
--db-pool-max-size

Konfiguracja SPI

Konfiguracja parametrów:
--spi-<spi-id>-<provider-id>-<property>=<value>

--spi-<spi-id>-provider-default=<identifier> (build time)
--spi-<spi-id>-enabled=false (build time)

Zewnętrzne rozszerzenia muszą zacząć zarządzać zależnościami, które nie są częścią Keycloak (dodatkowe biblioteki).

Konfiguracja Infinispan

--cache=ispn|local
KC_CACHE=...

--cache-config-file=<file-relative-to-conf-dir.xml>
KC_CACHE_CONFIG_FILE=...

--cache-remote-host
--cache-remote-username
--cache-remote-password
--cache-stack=udp|tcp|ispn|kubernetes|ec2|azure|google

Konfiguracja keystore/truststore

--config-keystore=file/test.p12
--config-keystore-password=...
--config-keystore-type=pkcs12

--https-trust-store-file=..
KC_HTTPS_TRUST_STORE_FILE=...

Zalecana migracja do parametrów truststore standardowych:
-Djavax.net.ssl.trustStore
-Djavax.net.ssl.trustStorePassword
-Djavax.net.ssl.trustStoreType

Rzeczy Ciekawe,
Migracja Identity Broker

OSEC

FORUM 2024

Pytanie na śniadanie

Czy kod z Keycloak 20 zadziała pod Keycloak 22?

Odpowiedź

- Od strony kompilacji - TAK
- Od strony konfiguracji - NIE
  - Należy zadeklarować parametry konfiguracyjne providera (factory).
  - Pierwotnie parametry konfiguracyjne były definiowane przez
    - strukturę konfiguracyjną (IdentityProviderModel)
    - szablon administracyjny (plik html do starej konsoli adm.)

- Zmiany w kodzie wymagane dopiero przy migracji do wersji 25.0

Demo #2, Identity Broker

FORUM 2024

Wymagane rzeczy
Ocena prezentacji
Pytania i odpowiedzi

OSEC

FORUM 2024

Cel prezentacji

1. Określenie przyczyn zmian w Keycloak
2. Weryfikacja wprowadzonych zmian
3. Ocena wyników

1. Co się zmieniło?
2. Jak się zmieniło?
3. Jak korzystać?

Kod źródłowy:

http://github.com
    /Code-House
     /keycloak-deviceflow (wciąż aktualny!)
     /keycloak-discourse   (zaktualizowany do 22.0)