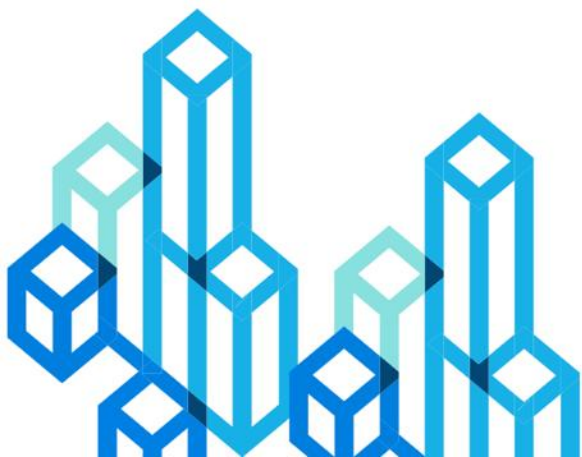


中间件视角看 Java 云原生趋势



张建锋

金蝶天燕 CTO

原红帽公司 JBoss 应用服务器核心开发组成员。毕业于北京邮电大学和清华大学，曾供职于金山软件，IONA 科技公司，红帽软件，初创永源中间件。

对于 JavaEE / JakartaEE 的各项规范比较熟悉；开源技术爱好者，喜欢接触各类开源项目，学习优秀之处并加以借鉴，认为阅读好的源码就和阅读一本好书一样让人感到愉悦

从一线中间件开发者的角度，剖析探讨现有中间件产品的发展方向。

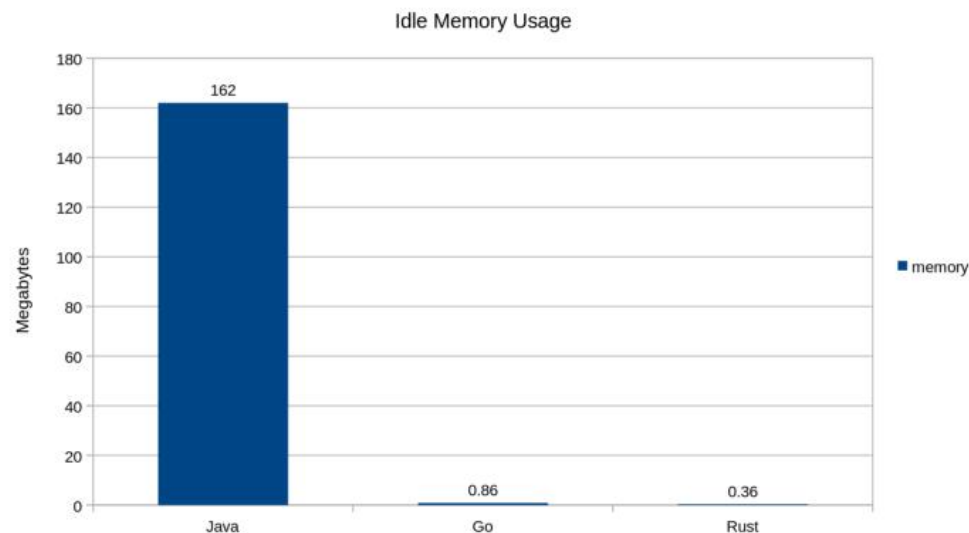
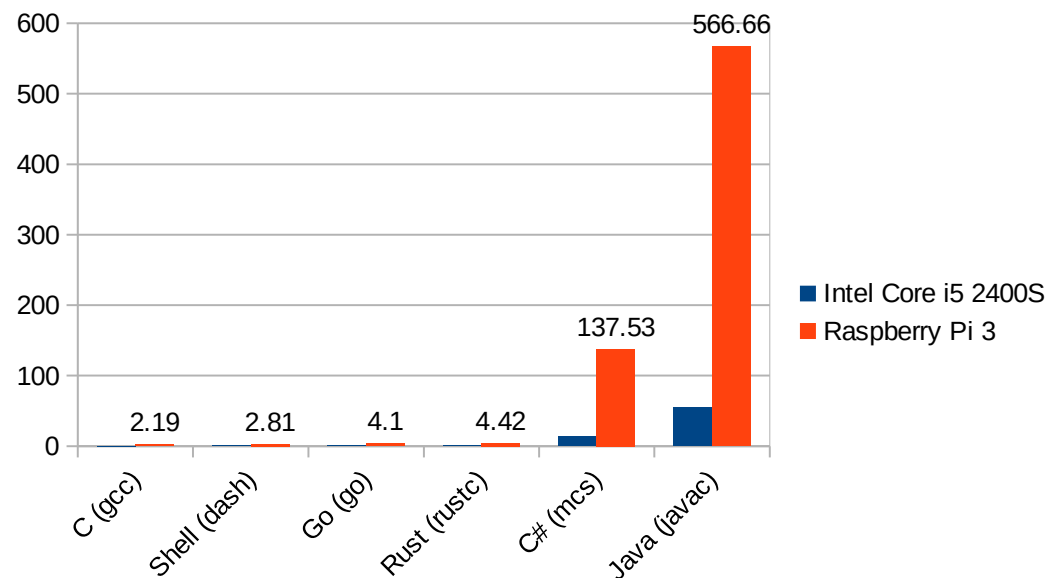
1. Java 语言如何应对的云原生带来的若干挑战
2. 经典 Java 开发框架的发展，如 JavaEE/JakartaEE 等对微服务 / 云原生的支持
3. 当前主流云原生框架优劣势分析
4. 中间件产品技术演进分析

Java 语言面临挑战

- 启动速度
- 内存占用

当前云原生基础设施
主力开发语言

C / Go / Rust



AppCDs

类加载优化

模块化

Graal 本地化

Application Class-Data Sharing 特性

-Xshare:on

-XX:+UseAppCDS

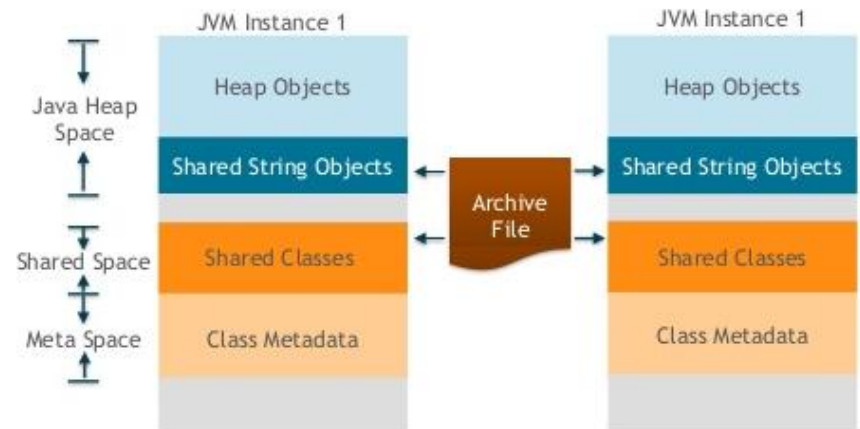
数据

AppCDS	Startup time
off	2160ms
on	1980ms

SpringBoot on JDK11

启动速度提升 8.3%。一般应用启动速度能快 20% 左右

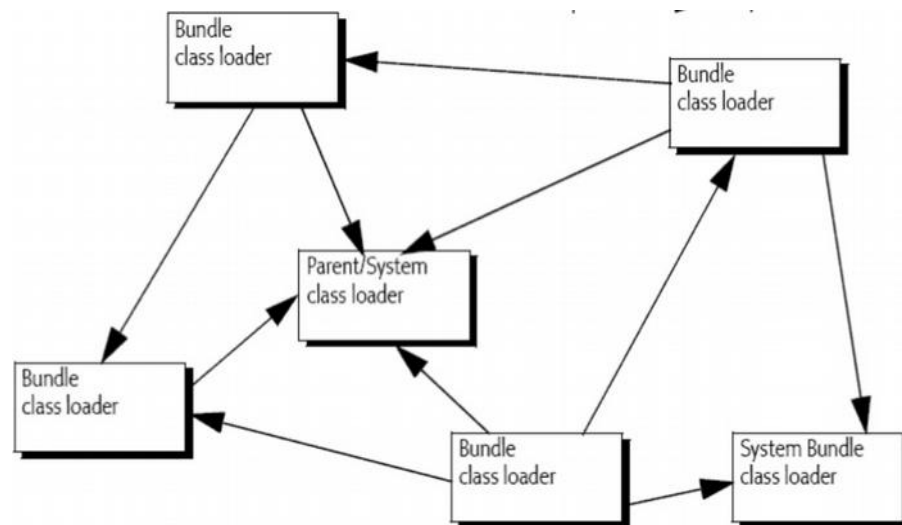
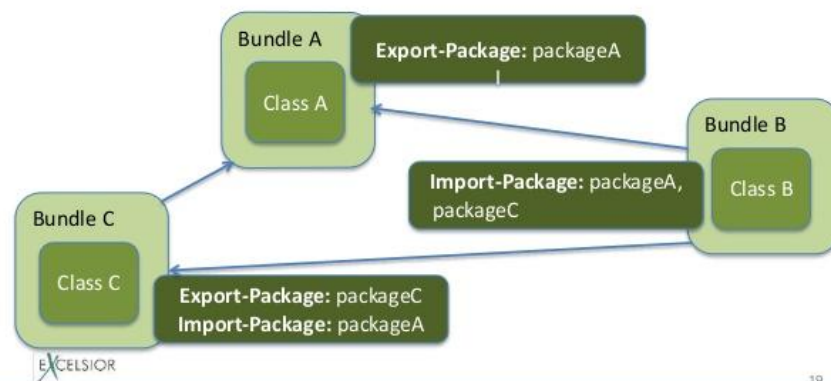
AppCDS Architectural View



- ◆ Osgi / JBossModule 网状类加载
- ◆ 在内存数据库建立索引
- ◆ 加载速度提升

- 形成 module 树形结构
- 包名称进行索引, 快速定位加载
 - index 文件
 - (jandex)
- 并发的类加载
- 管理模块依赖性
 - 图结构
 - 委托给其他模块加载器

OSGi Module System



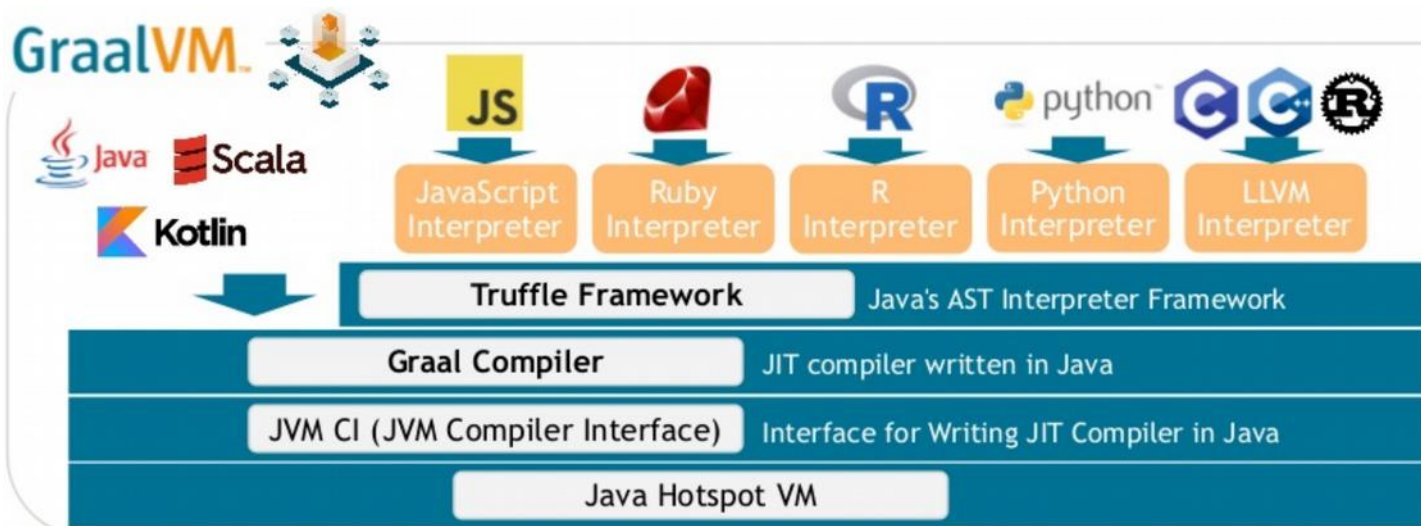
Java 9 之后支持模块化
可以通过 Jlink 工具，只提取必须的 jmod

JDK 11	JDK 占用空间	Modules 文件
完整版	310.6M	130.8M
只含有 java.base mod	49.7M	24.6M

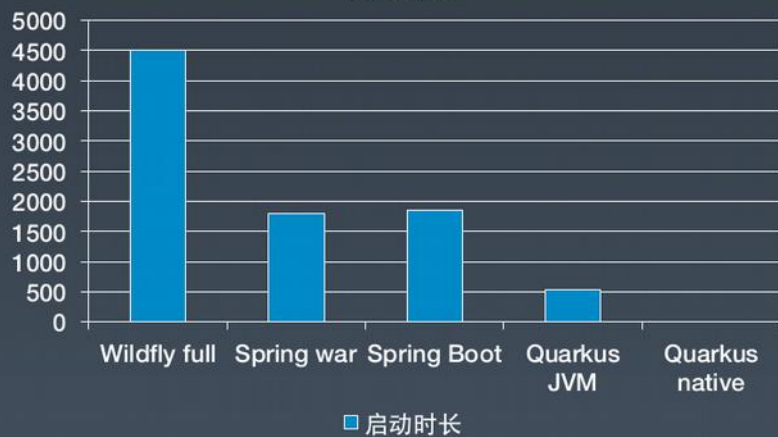
```
$ jar --create -file myapp-1.0.jar --main-class app.StringHash --module-version 1.0 -C mods .  
$ jlink --module-path jmods:mods --add-modules myapp--output target
```


本地化 Native-image

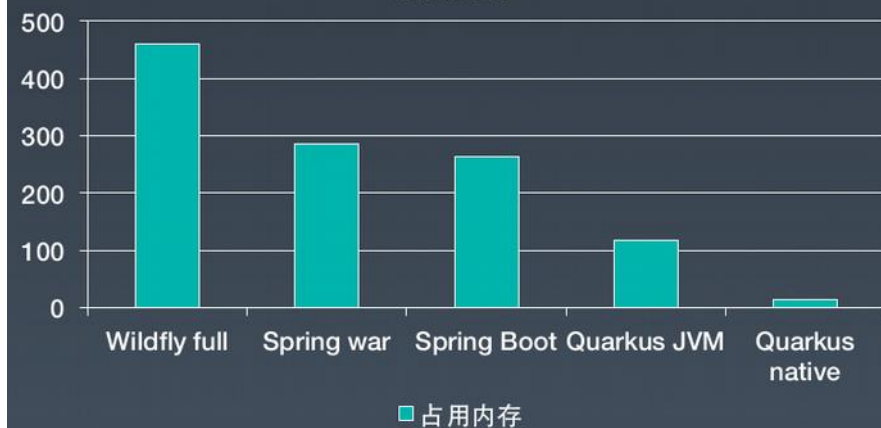
启动速度加快 内存占用降低



启动时长



占用内存

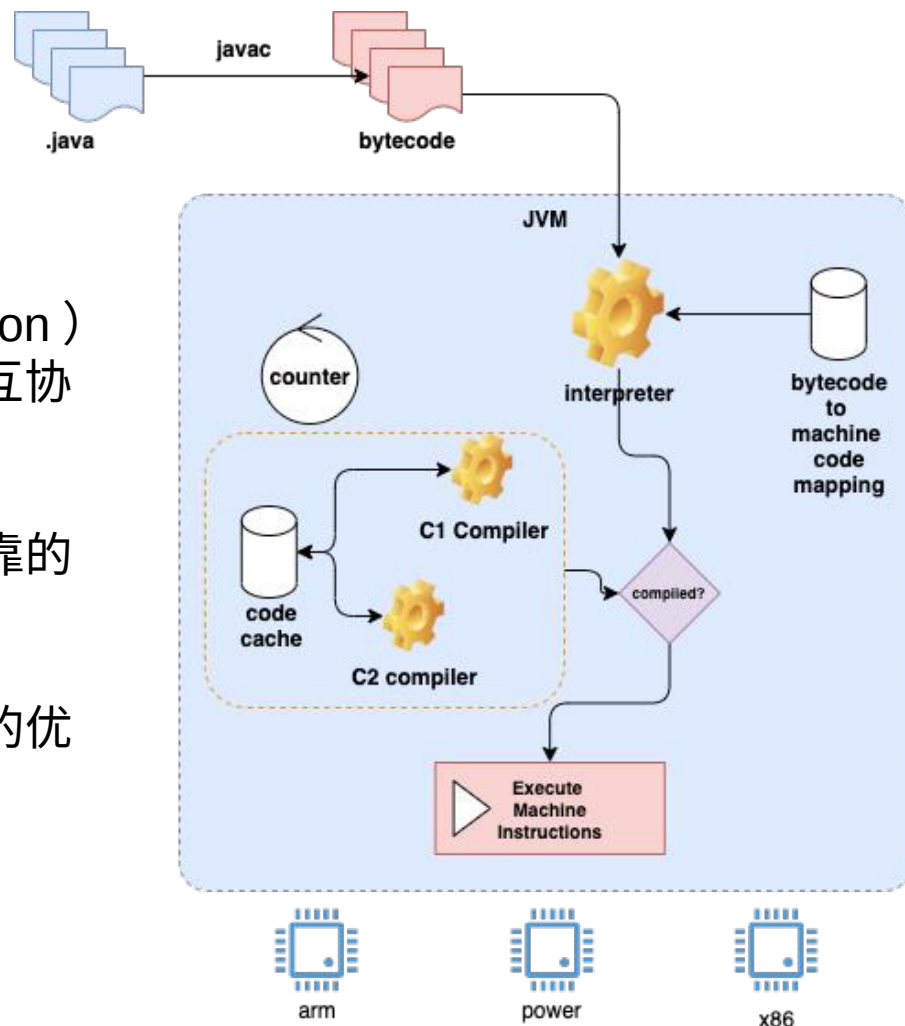


增强 Java 运行时效率 提前 C2 编译

默认开启分层编译（tiered compilation）策略，由 C1 编译器和 C2 编译器相互协作共同来执行编译任务。

C1 编译器会对字节码进行简单和可靠的优化，以达到更快的编译速度；

C2 编译器会启动一些编译耗时更长的优化，以获取更好的编译质量。



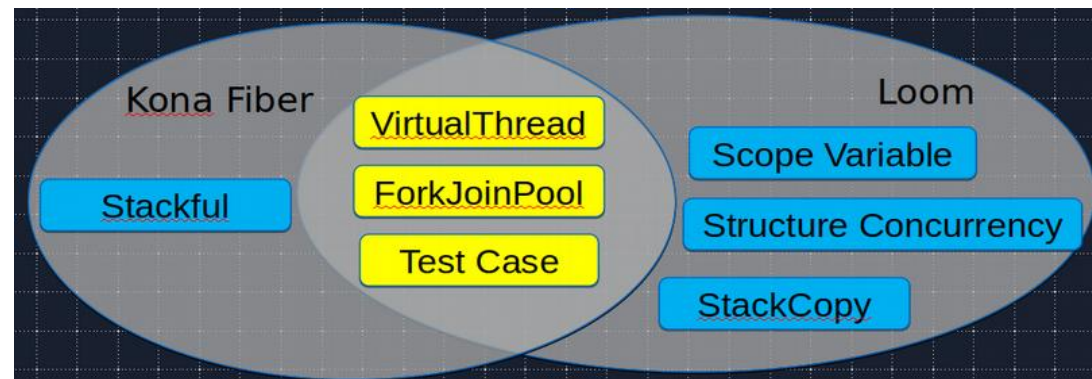
JVM Fiber 项目 : Project Loom (Quasar Fiber)

```
Fiber f = Fiber.schedule(() -> {  
    println("Hello 1");  
    lock.lock();  
    // 等待锁不会挂起线程  
    try {  
        println("Hello 2");  
    } finally {  
        lock.unlock();  
    }  
    println("Hello 3");  
})
```

Jetty 团队的运用 Jetty 10 on Loom

在使用虚拟线程时，以 5000rps 的速率处理 100ms 的请求，我能够在响应时间几乎没有增加的情况下处理 99% 的请求，而在使用常规线程时，只有 30% 的请求得到处理，响应时间增加了 9 倍

腾讯 KonaFiber



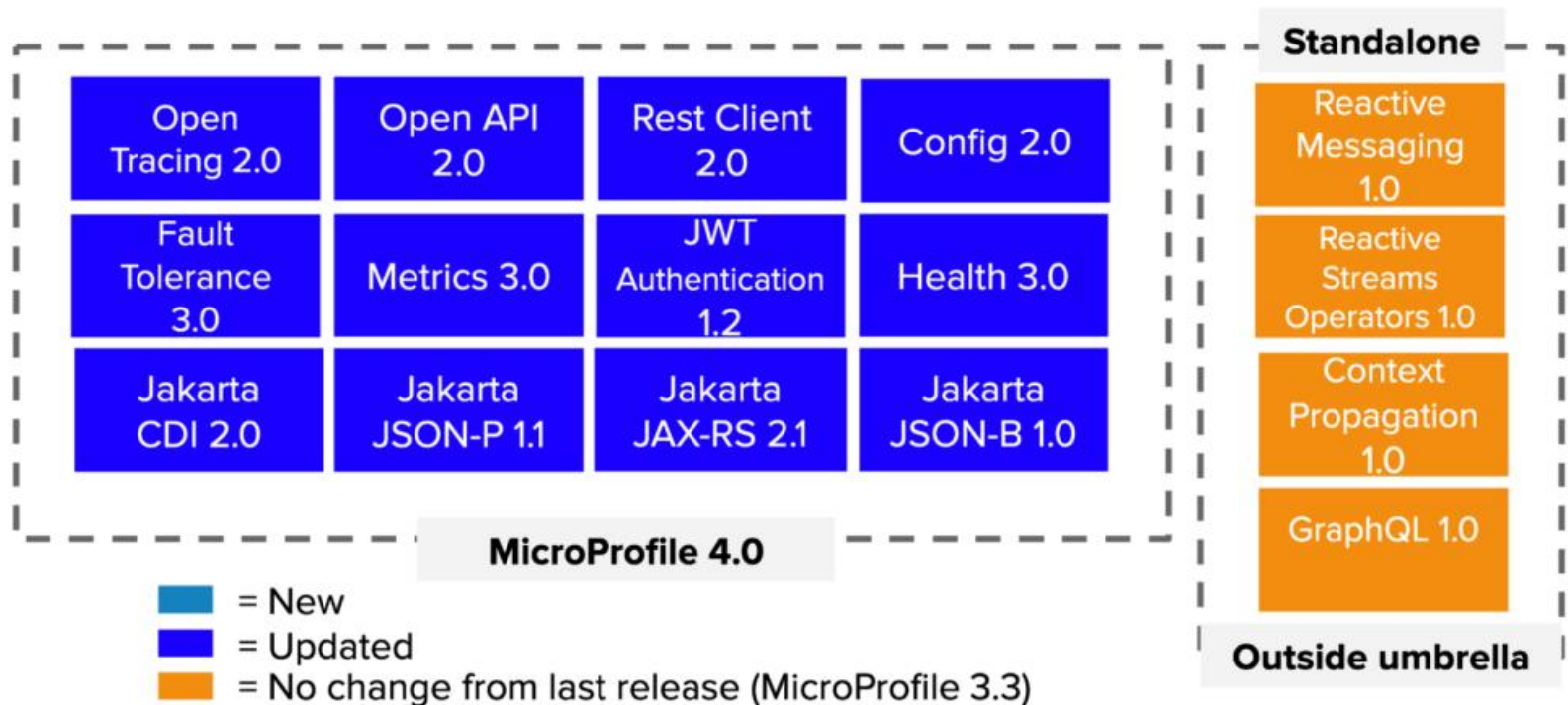
Jakarta EE 对微服务 / 云原生的支持计划

- 每年一个主版本
- 目标是单体应用 + 微服务
- 保持向后兼容性
- 各个子规范可以单独使用
- 生态包括微服务和 Spring
- CDI 为核心组件模型

Release	Date	Comment
Java EE 8	August 31, 2017	Final JCP/Oracle release
GlassFish 5.1	January 29, 2019	Verification build from transferred source
Jakarta EE 8	September 10, 2019	Initial Jakarta release; APIs under the Eclipse Foundation Technology Compatibility Kit (EFTCK) license following the Jakarta EE Specification Process (JESP)
<u>GlassFish</u> 6.0.0-RC2	November 1, 2020	Compatible implementation to go along with Jakarta EE 9 release
Jakarta EE 9	November 22, 2020	Ecosystem/tooling release
GlassFish 6.0.0	December 2020/January 2021	Final release of GlassFish 6.0.0
GlassFish 6.1.0	~Q2 2021*	Compatible implementation to go along with Jakarta EE 9.1 release; JDK 11 compatible
Jakarta EE 9.1	~Q2 2021*	JDK 11 release
Jakarta EE 10	~Q1 2022*	First new feature release
* tentative dates		



- 每年发布多个版本
- 目标是微服务和云原生应用
- 采纳创新技术包括 OpenTelemetry, gRPC, GraphQL
- 依赖 JakartaEE 的核心 CDI 技术



Jakarta EE 10 Core Profile

- JEE10 以后计划构建 Core Profile
- Web Profile, MicroProfile 所需要的核心规范
- 移入 Config
- 提炼出 CDI Lite
- 有 Quarks 等若干实现



- JEE TCK 使用 JUnit5 和 Arquillian
- 使用 @Service 注解，停用 @Stateless
- EE 子规范采用独立网站，分头发展
- 提升模块化支持，支持 JPMS, OSGi, JBossModules
- 引入 Core Profile
- 更好的支持微服务和云原生技术

- 加入 CDI 友好的注解 @Asynchronous
@Schedule @Lock
- 加入最大并发注解 @MaxConcurrency
@ManagedExecutorServiceDefinition
- 加入 JavaSE Completable Future 的支持
- 可以跨管理的线程传播 CDI 上下文

新 API :

- NoSQL 支持
- MVC 规范

待讨论决议:

- ◆ JSON schema
- ◆ Servlet 层, 持久层, Rest , MVC , 消息层
有更多的 Reactive/NIO 能力支持
- ◆ 转移 Arquillian 到 Eclipse 社区

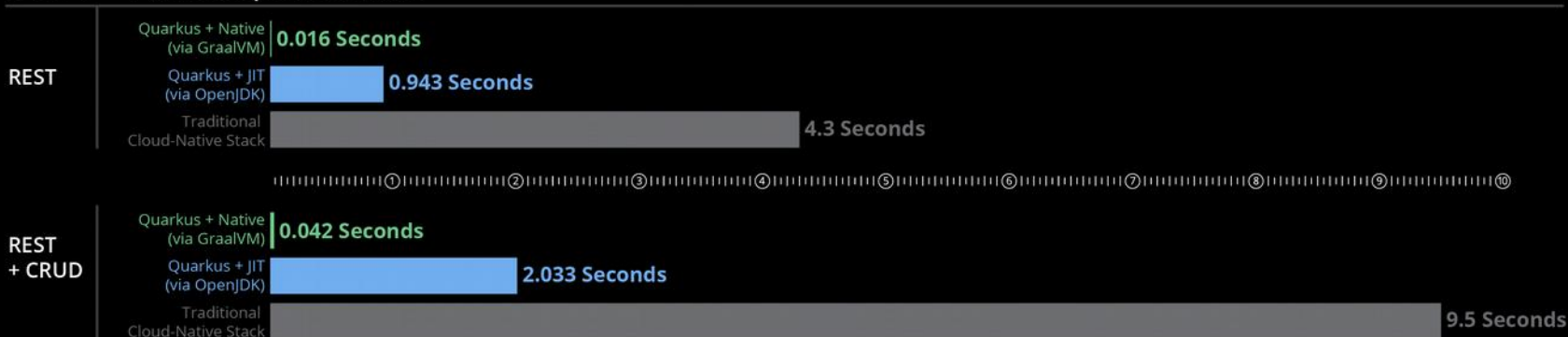
- 全栈云原生开发框架
- 发展历程，目前 2.0 版本
- 设计思想，编译和加载时运行时构建完毕
- GraalVM Java 虚拟机运行，编译成本地代码
- 减少启动时间和内存占用率

Memory (RSS) in Megabytes*

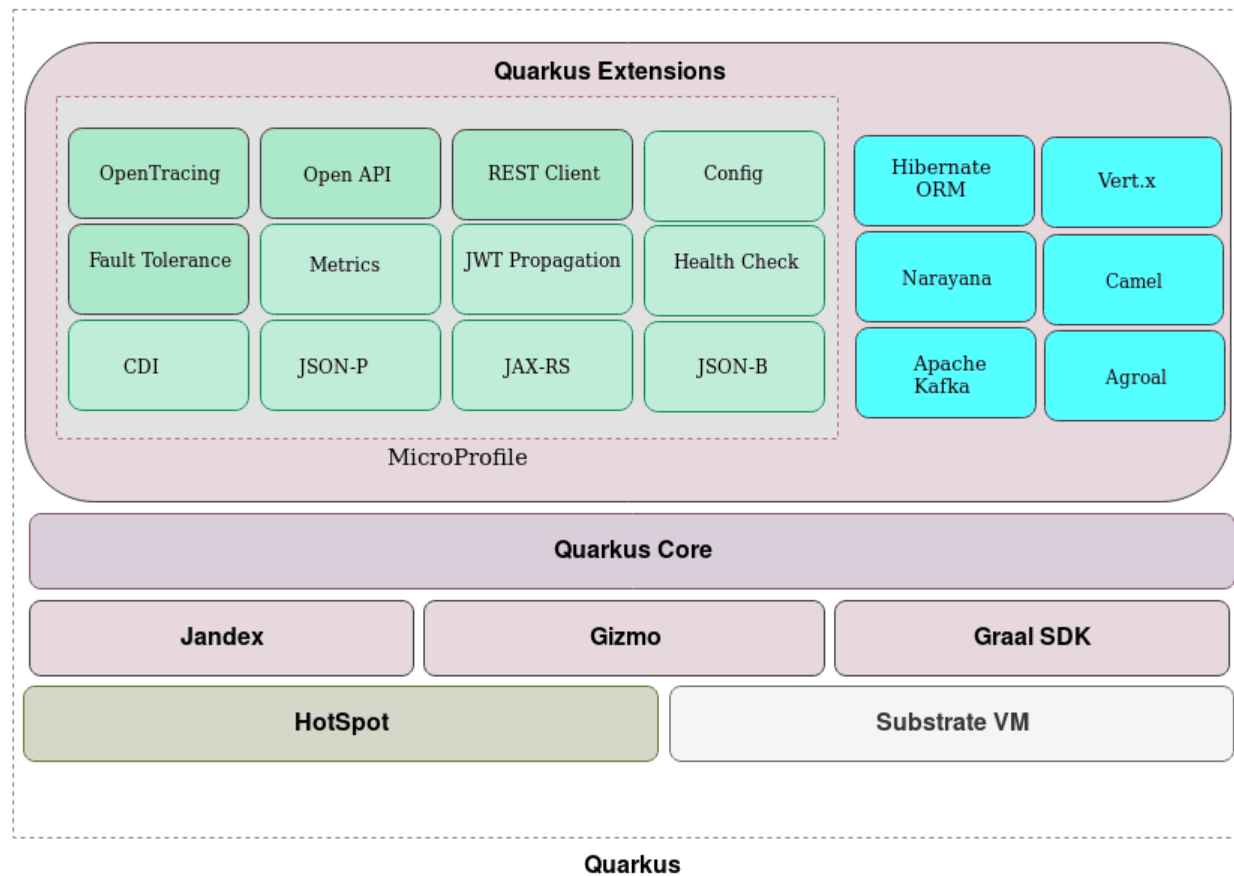
*Tested on a single-core machine



BOOT + First Response Time

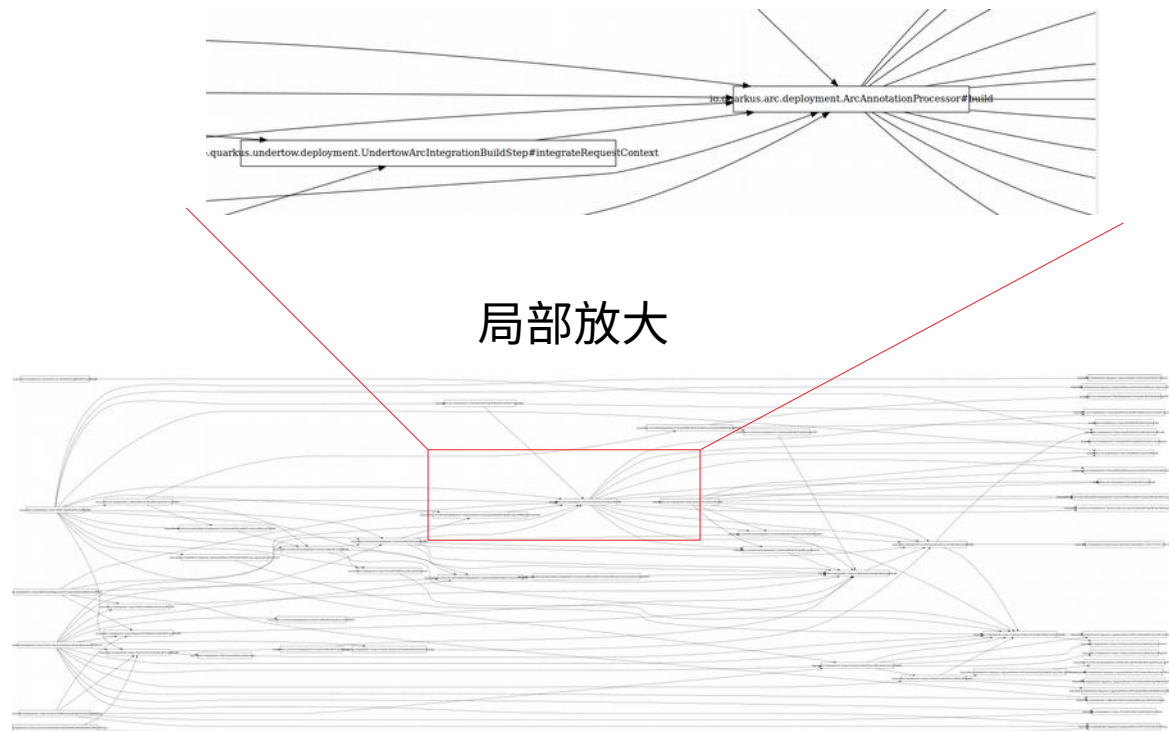


- 核心模块 Arc / CDI
- 网络接入 Vertx / Resteasy
- 其他功能组成, 如 ORM, Smallyre



- JEE 技术栈和 Microprofile
- Maven 构建过程，有定制的 maven plugin
- BuildItem / BuildStep / BuildChain
- Produce / Consume / BuildResult

- ◆ 在编译时重写 Java 代码，使得程序能够被快速加载和启动运行
- ◆ 能静态初始化加载的尽量静态构造，对原有的动态发现的进行必要代码重写
- ◆ 需要考虑本地编译的要求



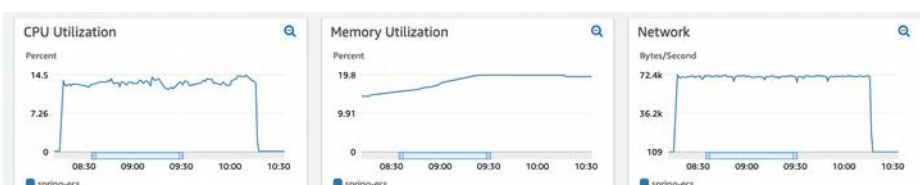
Spring Native 项目，目前处于 Beta 版

“Spring Native offers interesting characteristics including almost instant startup (typically < 100ms), instant peak performance and lower memory consumption at the cost of longer build times and fewer runtime optimizations than the JVM”

由常用的 SpringBoot 模块组成，对 Java 程序员更友好些

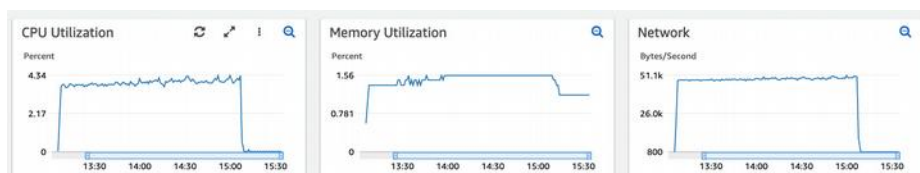
SpringNative

STATISTICS													Expand all groups Collapse all groups	
Requests ^	Executions					Response Time (ms)								
	Total ↕	OK ↕	KO ↕	% KO ↕	Cnt/s ↕	Min ↕	50th pct ↕	75th pct ↕	95th pct ↕	99th pct ↕	Max ↕	Mean ↕	Std Dev ↕	
Global Information	214382	214328	34	0%	29.441	22	35	87	170	318	20021	82	443	
Get All products	71454	71420	34	0%	9.814	52	114	143	252	546	20021	177	758	
Save new product	71454	71454	0	0%	9.814	24	31	36	54	121	1540	36	26	
get late... product	71454	71454	0	0%	9.814	22	29	34	48	99	1308	33	21	



Golang

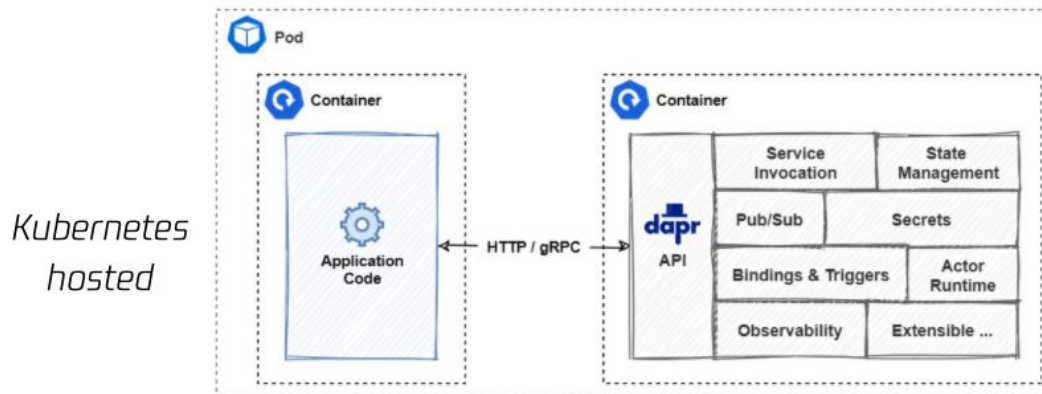
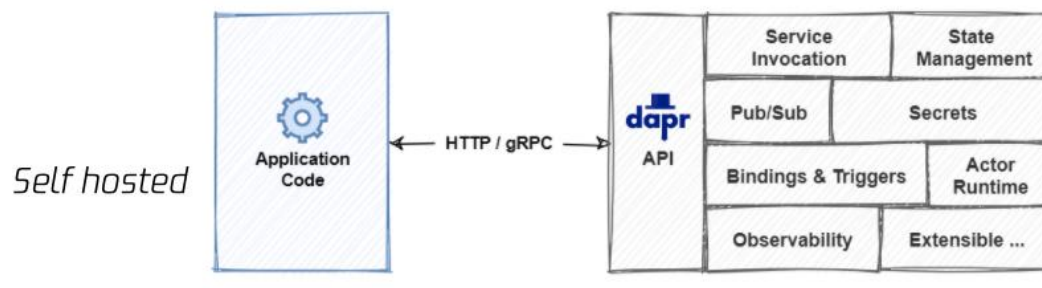
STATISTICS													Expand all groups / Collapse all groups	
Requests ^	Executions					Response Time (ms)								
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev	
Global Information	215499	215499	0	0%	29.597	21	28	57	93	173	3288	44	43	
get All products	71833	71833	0	0%	9.866	44	65	79	129	299	3288	76	57	
Save new product	71833	71833	0	0%	9.866	22	27	29	43	92	1184	30	18	
get late... product	71833	71833	0	0%	9.866	21	24	25	32	71	2451	26	18	



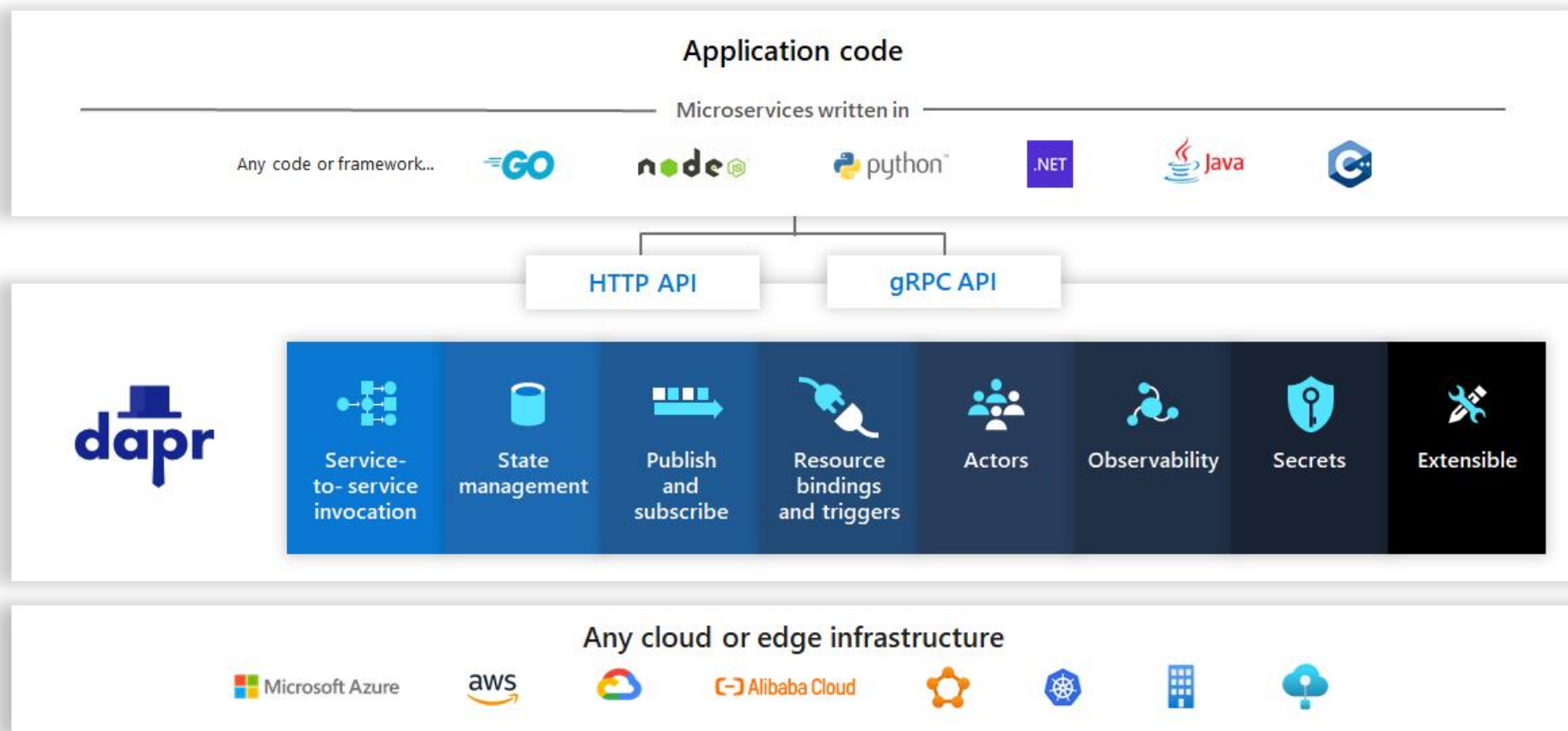
Go 中所有 3 个操作的平均时间都更快。在 Go 中，保存一个产品和检索最后一个产品的响应时间大约快 20%。在检索 20 个最新产品时，Go 比 Java 和 Spring Boot Native 快 133%。Go 应用程序中的 CPU 和内存使用率非常表现优异，在负载测试最后的 2 个小时内，Go 应用程序的 CPU 和内存都是稳定的，而 Spring Native 的内存使用率却在不断增加

<https://ignaciosuay.com/spring-boot-native-vs-go-a-performance-comparison/>

- Distributed Application runtime
- 云原生统一框架，定义了若干访问模型
- 可移植，事件驱动，弹性，有状态和无状态，云和边缘，语言和框架无关
- 依赖于容器，可依托于 K8s



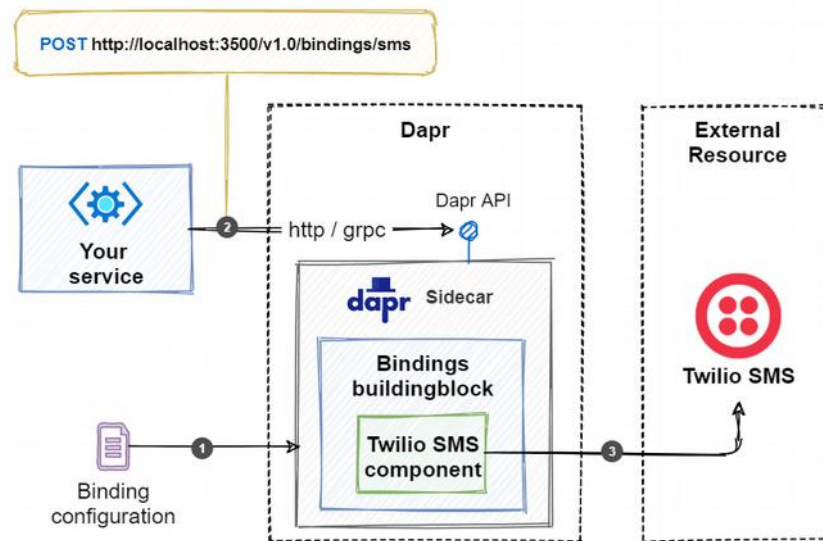
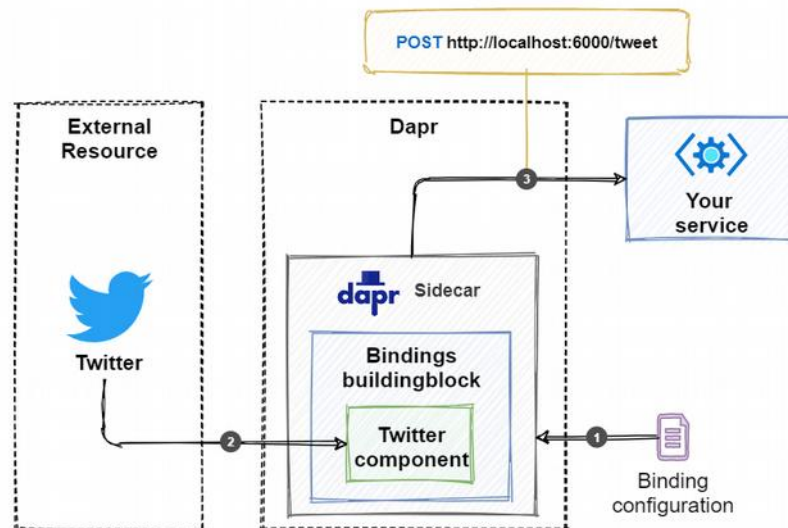
- 方法
 - 状态
 - 消息
 - 绑定
- actor
 - 可观测
 - 安全
 - 扩展



绑定各种云服务 可以跨平台跨云使用

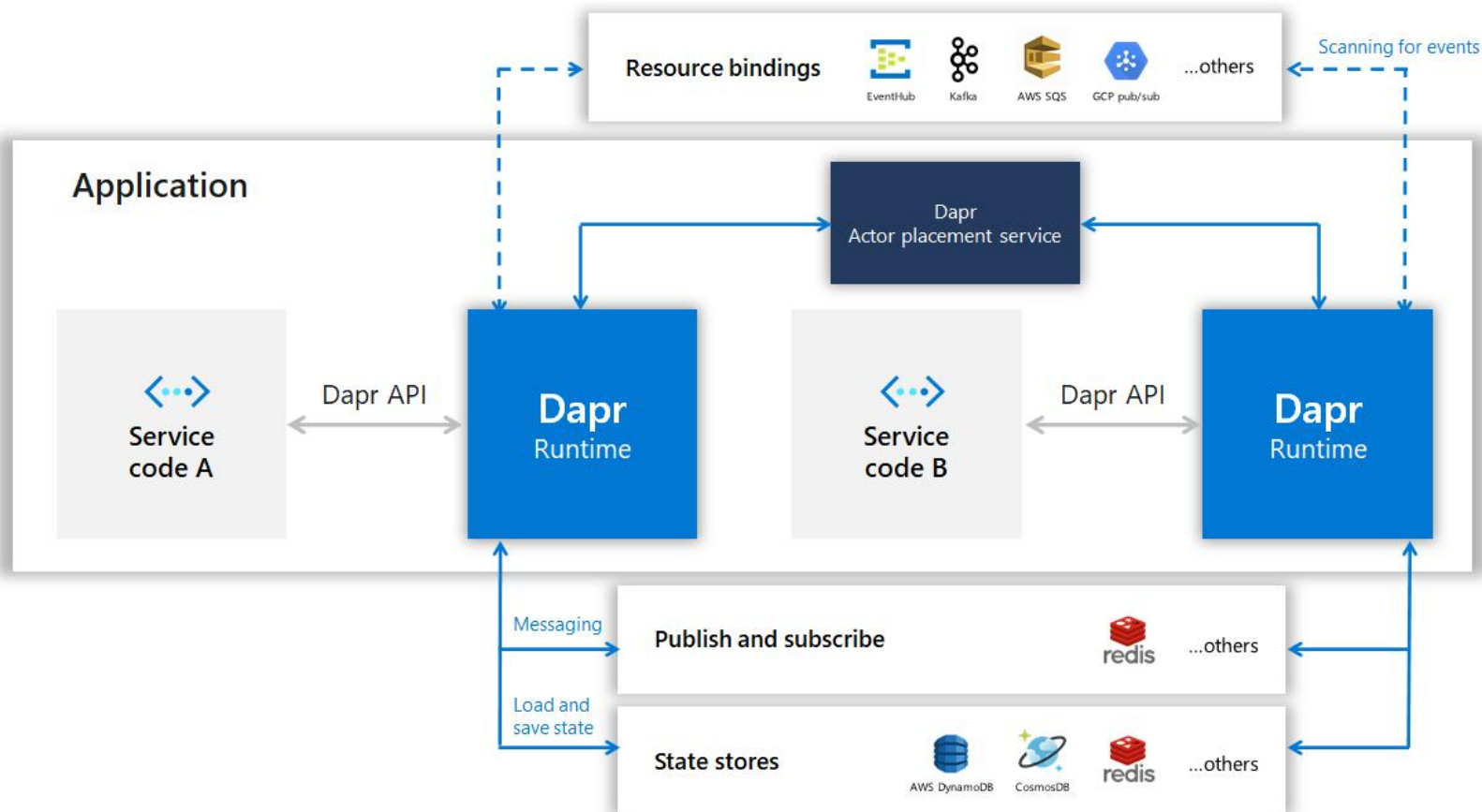
Generic

Name	Input Binding	Output Binding	Status	Component version	Since runtime version
Apple Push Notifications (APN)		✓	Alpha	v1	1.0
Cron (Scheduler)	✓	✓	Alpha	v1	1.0
HTTP		✓	GA	v1	1.0
InfluxDB		✓	Alpha	v1	1.0
Kafka	✓	✓	Alpha	v1	1.0
Kubernetes Events	✓		Alpha	v1	1.0
Local Storage		✓	Alpha	v1	1.1
MQTT	✓	✓	Alpha	v1	1.0
MySQL		✓	Alpha	v1	1.0
PostgreSQL		✓	Alpha	v1	1.0
Postmark		✓	Alpha	v1	1.0
RabbitMQ	✓	✓	Alpha	v1	1.0
Redis		✓	Alpha	v1	1.0
SMTP		✓	Alpha	v1	1.0
Twilio		✓	Alpha	v1	1.0
Twitter	✓	✓	Alpha	v1	1.0
SendGrid		✓	Alpha	v1	1.0

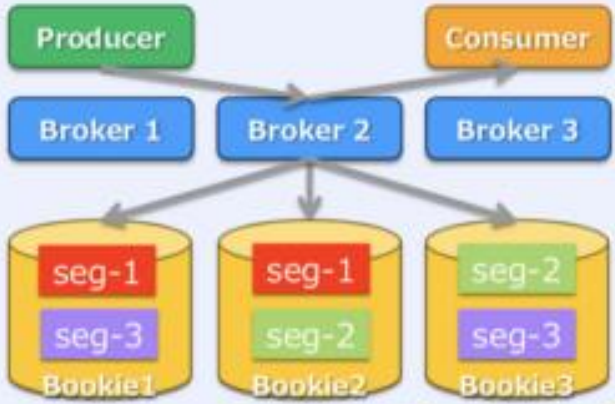
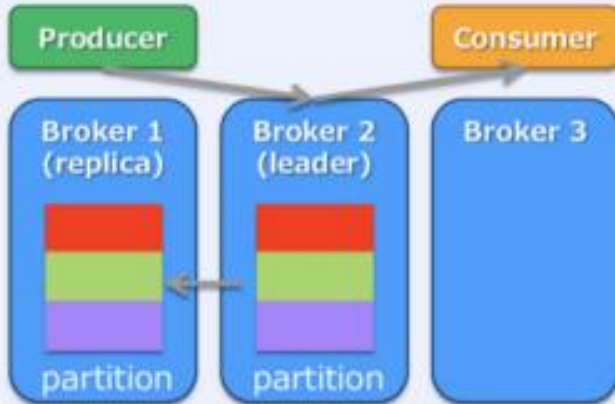


sidecar 模式

标准的 payload 数据格式



Apache Pulsar vs. Apache Kafka

	Apache Pulsar	Apache Kafka
Architecture image: • # of partition=1 • # of storage=3 • # of replication=2		
Replication unit	Segment (more granular than Partition)	Partition
Data distribution	Distributed and balanced across all Bookies	Kept in only leader and replica Brokers
Max capacity	Not limited by any single node	Limited by the smallest broker's capacity
Data Rebalancing when scale	Not required	Required
Geo-replication	Built-in	Need to start an extra process: MirrorMaker

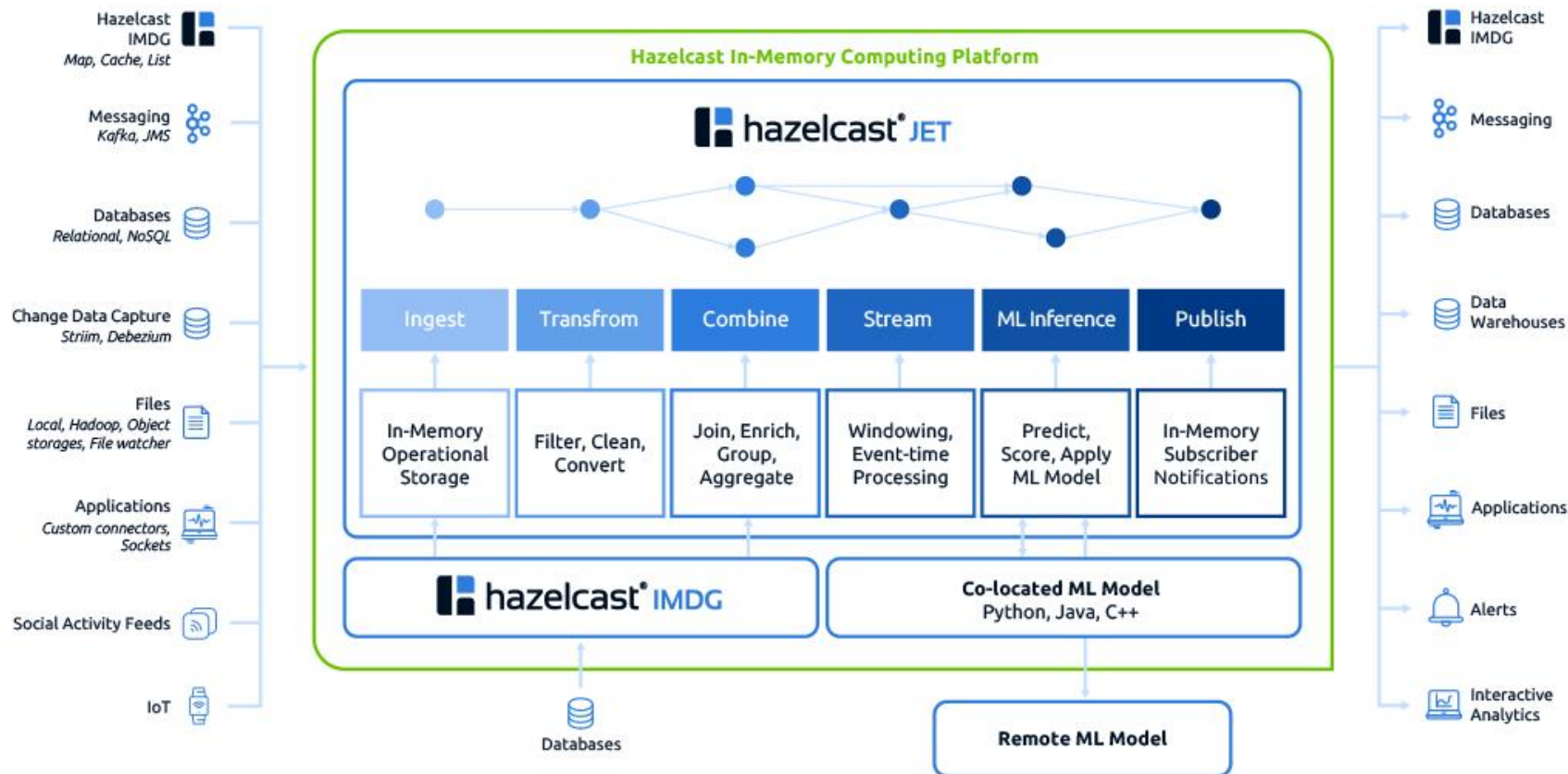
Pursar vs Kafka

存储分离，支持多租户，远程复制能力

- ◆ V2.8.0 采用状态同步，去除 ZK 依赖
 - ◆ Kafka 使用内嵌的 KRaft 替代了 ZooKeeper，类似 ES 分布式系统
 - ◆ Zookeeper 笨重，要求奇数个节点的集群配置，扩缩容不便
-
- 部署更简单，只需一个进程
 - 监控更便捷，直接从 Kafka 获取监控信息
 - 速度更快，Raft 比 ZK 的 ZAB 协议更加易懂，也更加高效
- 需要一段时间的成熟

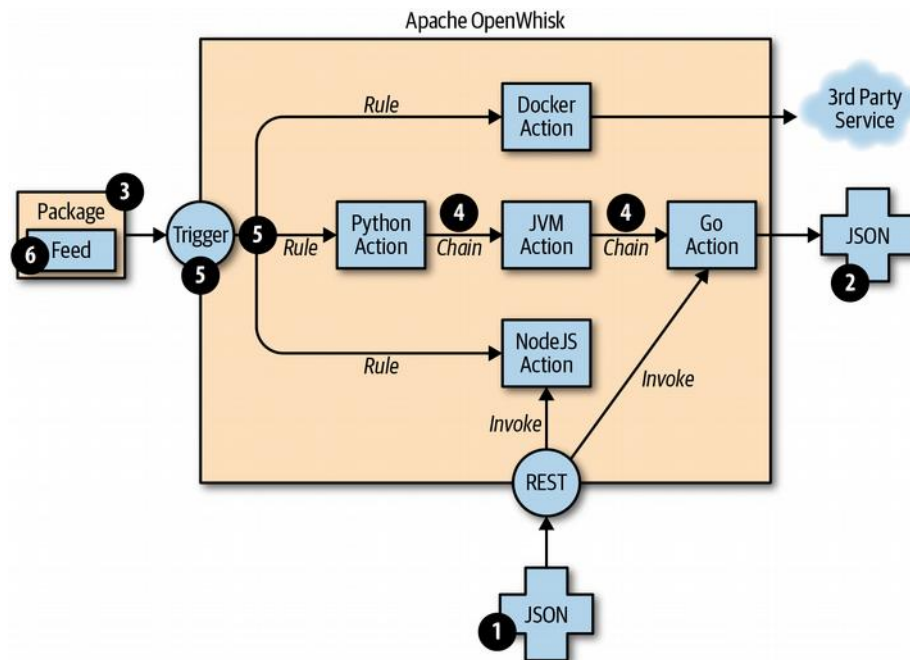
Spark 批量 / Flink 流式计算框架

Halzelcast 分布式缓存 / 内存计算框架



Serverless 框架，部分公有云支持

Java 的无服务器实现框架和编程框架
Apache OpenWhisk



OpenWhisk 体系图



麦思博 (msup) 有限公司是一家面向技术型企业的培训咨询机构，携手 2000 余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过 3000 余家企业续约学习，是科技领域占有率第 1 的客座导师品牌，msup 以整合全球领先经验实践为己任，为中国产业快速发展提供智库。



高可用架构公众号主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。

高可用架构系列社群是一个社区组织，其精神是“分享 + 交流”，提倡社区的人人参与，同时从社区获得高质量的内容。