THE DEVELOPER'S CONFERENCE

Serverless com Java, dá certo?

Leandro Del Sole

Eu

- Leandro Del Sole
- Desenvolvedor há 10 anos - principalmente Java
- Atualmente Líder de time na Navita

Apresentação

1. Conceito
2. Contexto
3. Desafios utilizando Java
4. Conclusão

Conceito

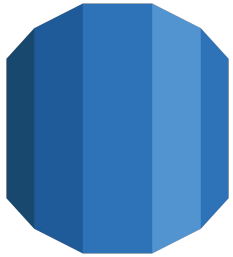
Serverless

“[...] refere-se ao conceito de construção e execução de aplicações que **não requerem gerenciamento de um servidor**.

Descreve um modelo de implantação mais refinado no qual os aplicativos, agrupados como uma ou mais funções, **são carregados em uma plataforma** e, em seguida, executados, dimensionados e cobrados em resposta à demanda exata necessária no momento. ”

Backend-as-a-Service (BaaS)

Firebase



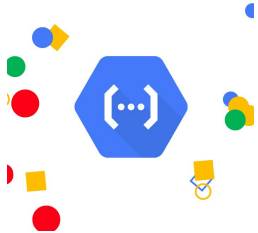
AWS RDS

AWS API Gateway

Function-as-a-Service (SaaS)

AWS Lambda

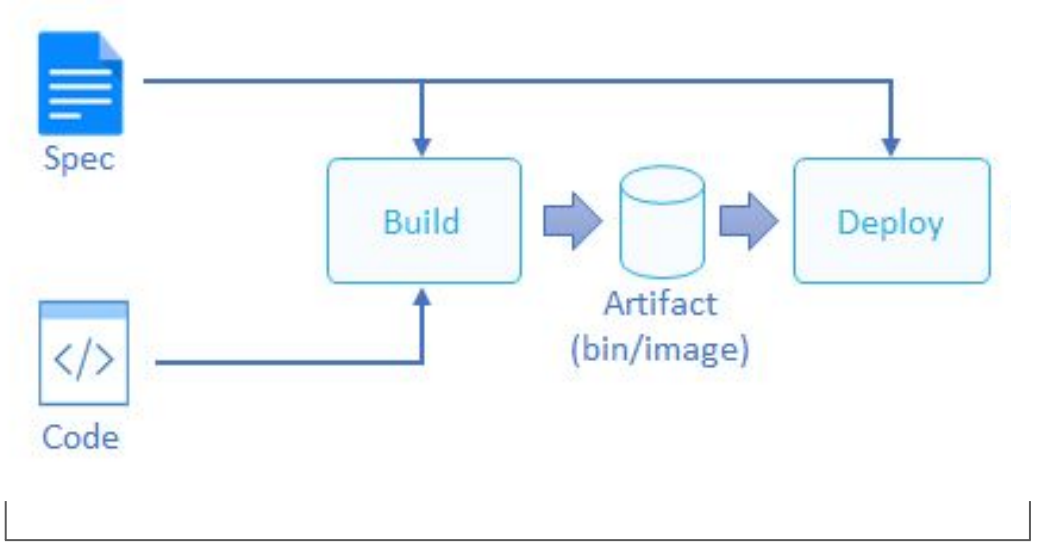
Azure Function



Google Cloud Functions

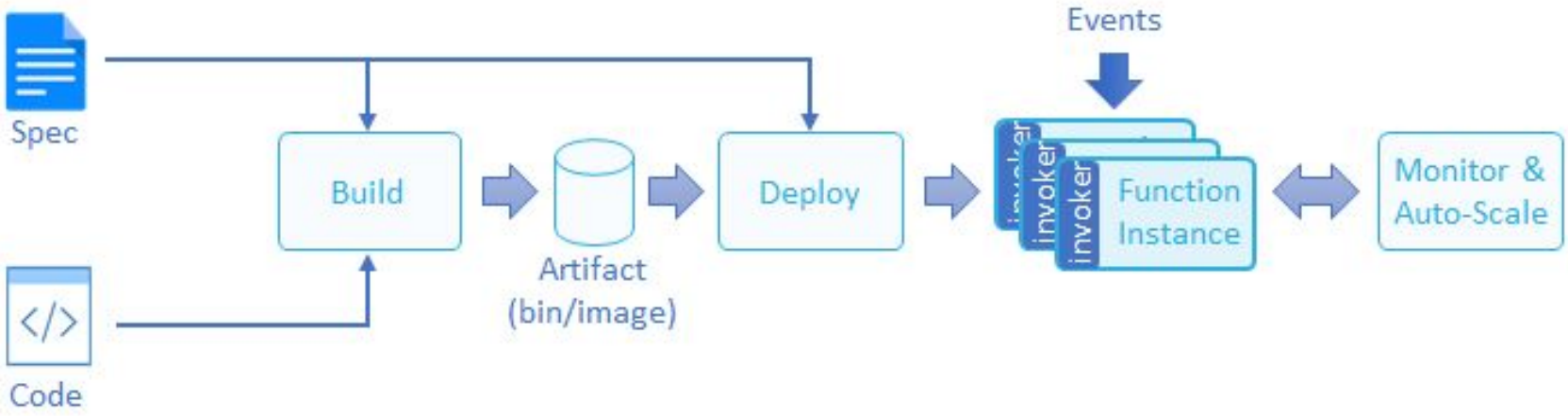
IBM Cloud Functions

Função



você controla e automatiza!

Função



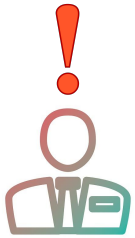
você controla e automatiza!

o provedor controla

Modelo de programação FaaS

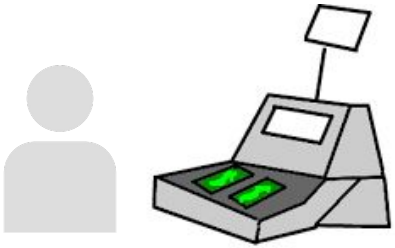
Seu código deve ser escrito em um estilo **stateless e não ter nenhuma afinidade com a infraestrutura computacional** que está executando-o. Seu código deve esperar que o acesso ao filesystem, processos filhos, e artefatos similares são limitados ao tempo de vida da requisição.

Analogia



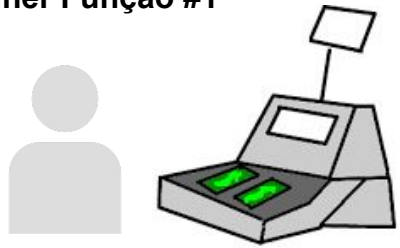
~~Gerente~~
FaaS Controller

Invocações da função



~~Operador #1~~
Container Função #1

~~#1~~ #2 #3 #4 #9 #10 #11 #12

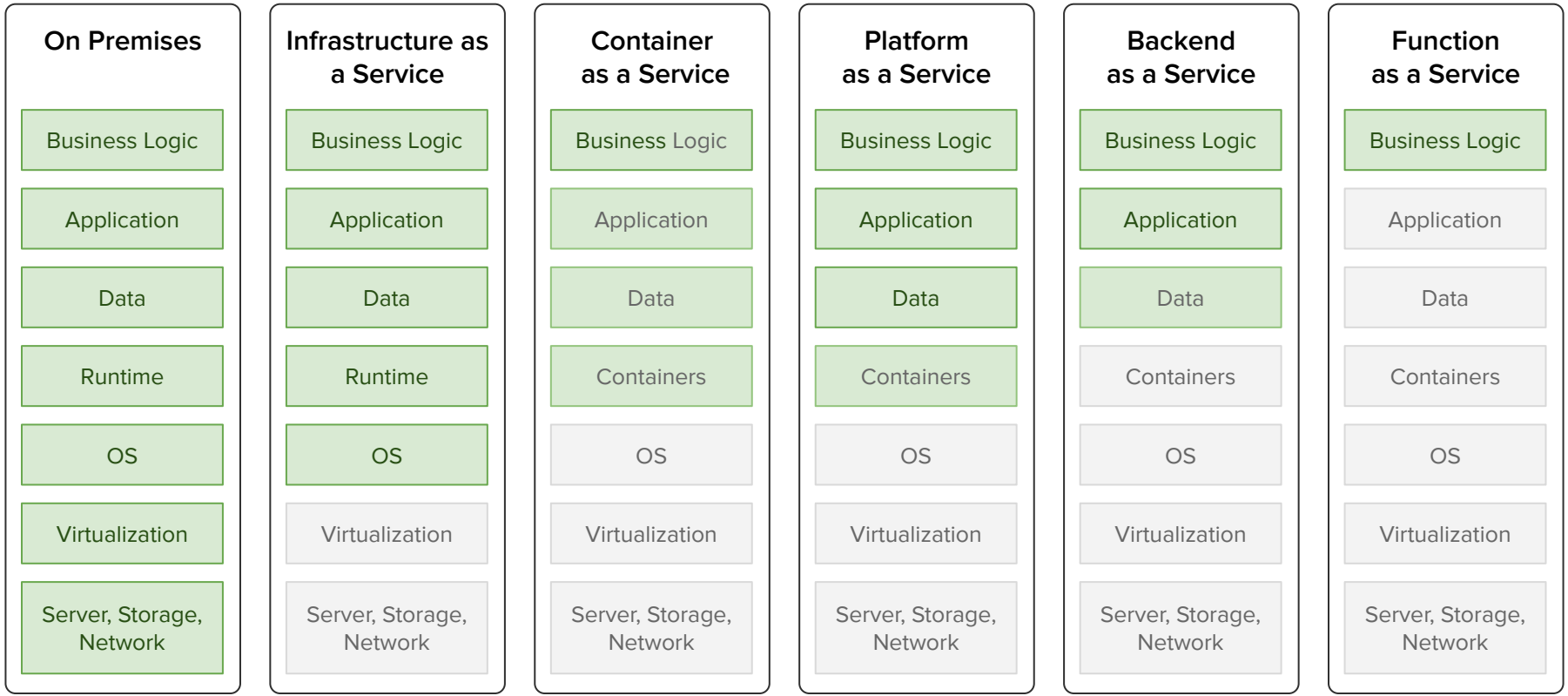


~~Operador #2~~
Container Função #2

#8

Contexto

*-as-a-Service


On Premises
Business Logic
Application
Data
Runtime
OS
Virtualization
Server, Storage, Network
Infrastructure as a Service
Business Logic
Application
Data
Runtime
OS
Virtualization
Server, Storage, Network
Container as a Service
Business Logic
Application
Data
Containers
OS
Virtualization
Server, Storage, Network
Platform as a Service
Business Logic
Application
Data
Containers
OS
Virtualization
Server, Storage, Network
Backend as a Service
Business Logic
Application
Data
Containers
OS
Virtualization
Server, Storage, Network
Function as a Service
Business Logic
Application
Data
Containers
OS
Virtualization
Server, Storage, Network

Hospedado

Logos com fundo cinza não são open source

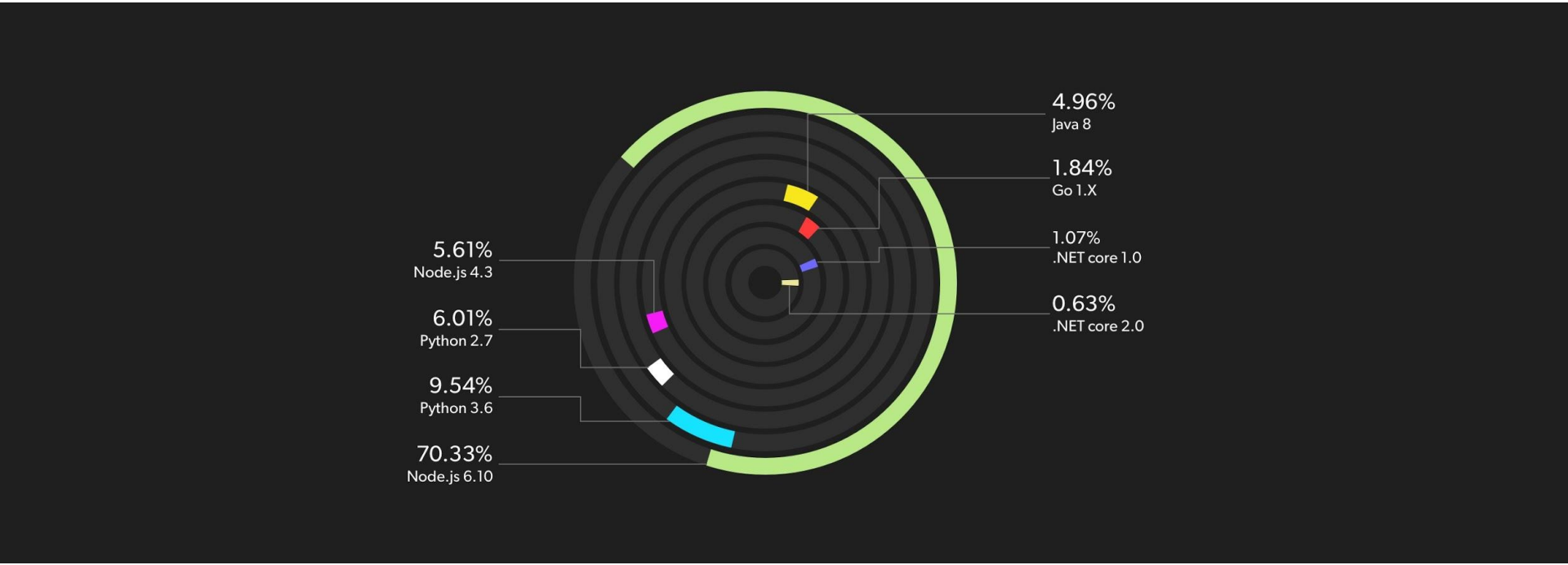
Instalável

Hospedado vs Instalável

	OpenFaaS no k8s	AWS Lambda
Memória	12 TiB*	3 GiB
CPU	96 cores* / GPU / FGPA	2 cores?
Timeout	290 anos	15min / 30s API GW
Limite de tam. do código	∞	50 MB
Espaço em disco	∞	512 MB
Qtde. mínima de réplicas	sim	não
Zero replicas	sim	sim

* Maior máquina encontrada disponível em Cloud

Uso por linguagem



Pesquisa do Serverless Framework feita em 2018

Desafios utilizando Java

Desafios

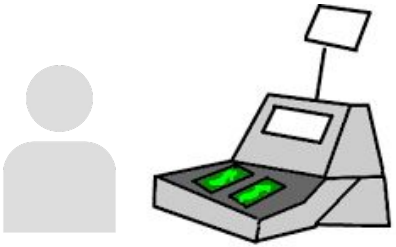
1. Cold Start time elevado
2. Consome muita memória

Fato ou Fake?

NodeJS 10x - o queridinho dos serverlesseiros

Create Movie com **256 MB**

Cold (579,30 ms e 93 MB)
Warm (50,92 ms e 93 MB)

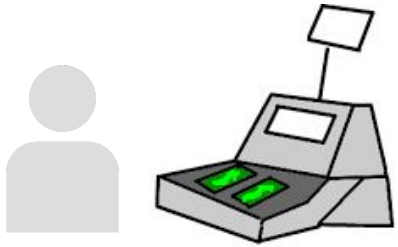


Pronto!

Java8 - o nosso queridinho

Insert User com **256 MB**

Cold (16.271,44 ms / ~16s e 138 MB)
Warm (16,60 ms e 138 MB)



Pronto!

Infelizmente, fato.

Dá pra melhorar?

Abordagem #1

Aumentar memória da função, para aumentar CPU

Insert User com **256 MB** - **Cold (16.271,44 ms / ~16s e 138 MB)** e **Warm (16,60 ms e 138 MB)**

Insert User com **3008 MB** - **Cold (1.366,55 ms / ~1,3s e 157 MB)** e **Warm (16,89 ms e 157 MB)**

O aumento de quase 12x na memória é praticamente um aumento de quase 12x no custo, já que o tempo Warm se mantém o mesmo.

Abordagem #2

Migrar AWS SDK da V1 para V2
utilizar um client HTTP mais simples, porém mais rápido

SDK V1 Insert User com **256 MB** - **Cold (16.271,44 ms / ~16s e 138 MB)** e **Warm (16,60 ms e 138 MB)**

SDK V2 Insert User com **256 MB** - **Cold (12.651,65 ms / ~13s e 132 MB)** e **Warm (40,98 ms e 132 MB)**

Melhorou, mas muito longe do Node ainda.
Mas vamos seguir com essa stack.

Abordagem #3

Runtime customizado Java 11

Java 8 Insert User com **256 MB** - **Cold (12.651,65 ms / ~13s e 132 MB)** e **Warm (40,98 ms e 138 MB)**

Java 11 Insert User com **256 MB** - **Cold (15.662,58 ms / ~16s e 143 MB)** e **Warm (132,84 ms e 143 MB)**

Confesso que deve ser possível melhorar, mas mesmo assim não vai chegar no nível Node

Abordagem #4

Runtime customizado executando imagem nativa criado pela GraalVM

Insert User com **256 MB** - **Cold (12.651,65 ms / ~13s e 132 MB)** e **Warm (40,98 ms e 138 MB)**

Nativo Insert User com **256 MB** - **Cold (1.090,37 ms / ~1s e 73 MB)** e **Warm (14,87 ms e 73 MB)**

Bem mais próximo do Node, com queda na memória ainda por cima!

Comparativo com o Node

Round-trip extraídos utilizando a ferramenta Hey.
2000 requests com 50 executores simultâneos.

Métricas das reqs.	NodeJS 10x	Java GraalVM Nativo
Total	5,09 s 🔴	4,74 s 🟢
Mais lenta	1,6930 s 🟢	1,7510 s 🔴
Mais rápida	0,0542 s 🔴	0,0352 s 🟢
Média	0,1219 s 🔴	0,1138 s 🟢
Requests por segundo	392,8430 🔴	421,5070 🟢
Grupo mais frequente	0,218 s (1949) 🔴	0,207 s (1941) 🟢
Memória*	93 MB 🔴	73 MB 🟢

* não foi coletada pelo Hey, e sim pelo console do Lambda

Conclusão

Conclusão

- Já dá certo, de forma tradicional, para casos de usos mais específicos
- Dá mais certo ainda com as inovações que vêm acontecendo

Referências

- https://github.com/cncf/wg-serverless/blob/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf
- https://docs.google.com/spreadsheets/d/10rSQ8rMhYDgf_ib3n6kfzwEuoE88qr0amUPRxKbwVCk/edit?usp=sharing
- http://bit.ly/serverlessfull
- https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview
- https://landscape.cncf.io/images/serverless.png
- https://s.cncf.io
- https://docs.aws.amazon.com/lambda/latest/dg/limits.html
- https://aws.amazon.com/pt/lambda/faqs/
- https://docs.aws.amazon.com/lambda/latest/dg/programming-model-v2.html
- https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html
- https://read.acloud.guru/simon-wardley-is-a-big-fan-of-containers-despite-what-you-might-think-18c9f5352147

Referências [2]

- https://serverless.com/blog/serverless-by-the-numbers-2018-data-report/
- https://www.openfaas.com/blog/introducing-openfaas-for-lambda/
- https://aws.amazon.com/pt/blogs/opensource/deploy-openfaas-aws-eks/
- https://andthearchitect.com/2019/02/20/running-java-11-on-aws-lambda-using-custom-runtimes/
- https://medium.com/graalvm/instant-netty-startup-using-graalvm-native-image-generation-ed6f14ff7692
- https://medium.com/graalvm/announcing-graalvm-19-4590cf354df8
- https://medium.com/@mathiasdpunkt/fighting-cold-startup-issues-for-your-kotlin-lambda-with-graalvm-39d19b297730
- https://github.com/panga/fn-native-java
- https://github.com/rakyll/hey
- https://github.com/leandrodelsole/aws-lambda-http-api-node10x

Obrigado!


https://github.com/leandrodelsole/aws-lambda-http-api


http://bit.ly/serverless-da-certo-tdc


https://www.linkedin.com/in/leandro-del-sole-da-silva/


leandrodelsole@gmail.com


@leandrodelsole


http://bit.ly/serverless-soujava-yt