

Controle na integração de serviços

Do monólito ao Service Mesh

Thiago Gregório

@thiagogregorio

thiago.gregorio@gmail.com



Quem sou eu ?

THIAGO GREGÓRIO

- ~ 17 anos de xp
- Consultor de Desenvolvimento de Software
- Sistemas distribuídos / Cloud / Containers / DevOPS
- **ThoughtWorks**



Monólito

Quem nunca ?

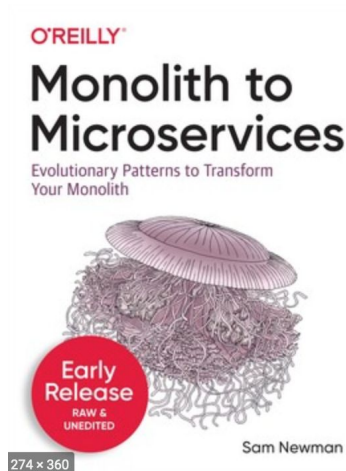
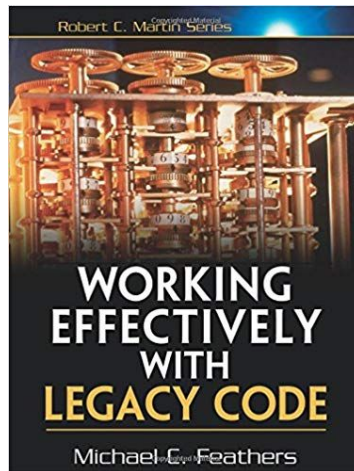
- Único deploy
- Todas as funcionalidades do sistema devem ser instaladas de uma só vez
- Podem ser modularizados
- Desvantagens
 - Mais vulneráveis aos perigos do alto acoplamento
 - Principalmente para deploys (múltiplas equipes)
 - Ownership



Vantagens de um Monólito

Monólito não é sinônimo de sistema legado

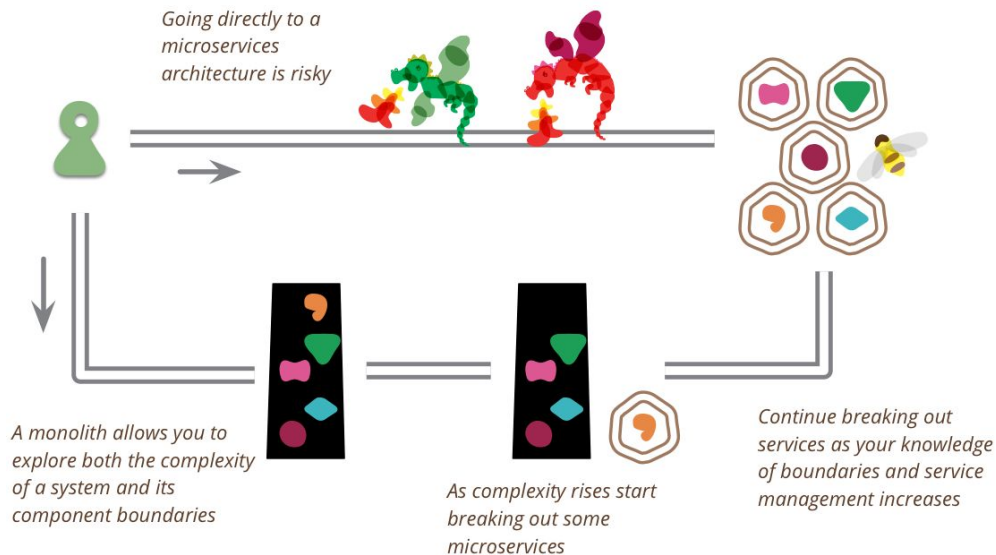
- Não possui vários desafios de sistemas distribuídos
- Monitoramento e *Troubleshooting* mais fáceis
- Testes end-to-end
- Pode facilitar reuso de código
- Persistência



MonolithFirst

"You shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile." Martin Fowler.

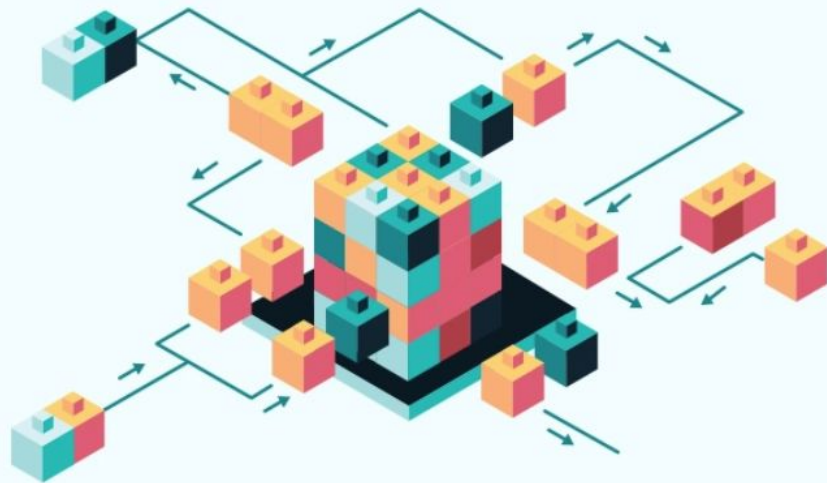
- Utilidade vs Escalabilidade
- Contexto bem definido
 - Refactoring de funcionalidade
- Projetar o monólito com cuidado
 - Modularidade API
 - Persistência de dados



Microsserviços

Transformar módulos em pequenos serviços

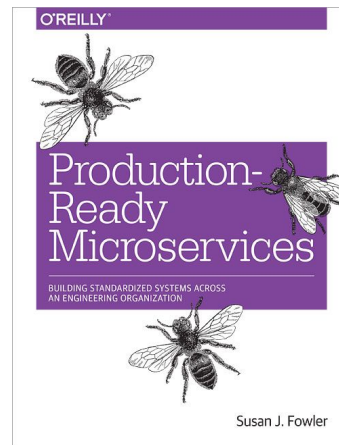
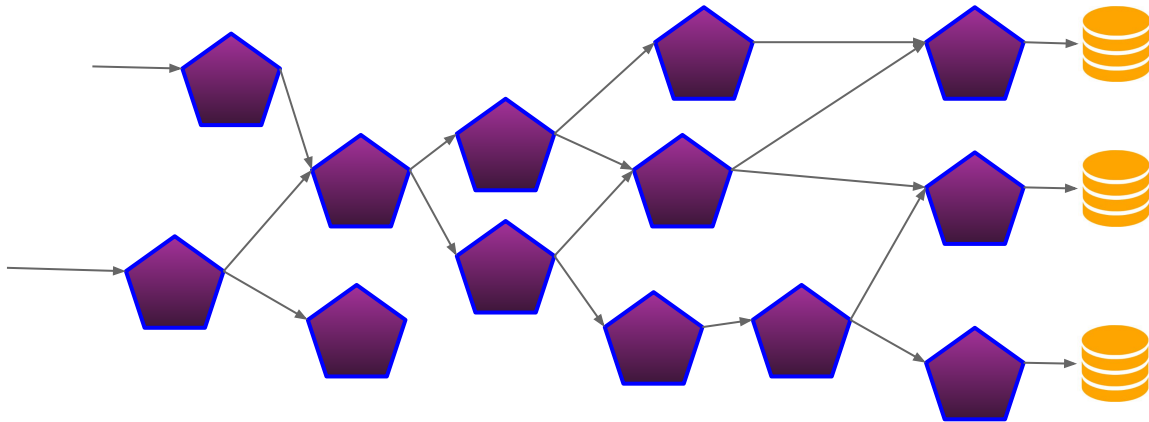
- Deploy independente
- Velocidade para chegar em produção
- Independente de linguagem (**API**)
- *Two-Pizza team* 🍕
- Desacoplamento
- Isolamento de falhas
- Persistência isolada
- Escalabilidade granular
- ThoughtWorks Tech Radar 2012 - Assess



Microserviços

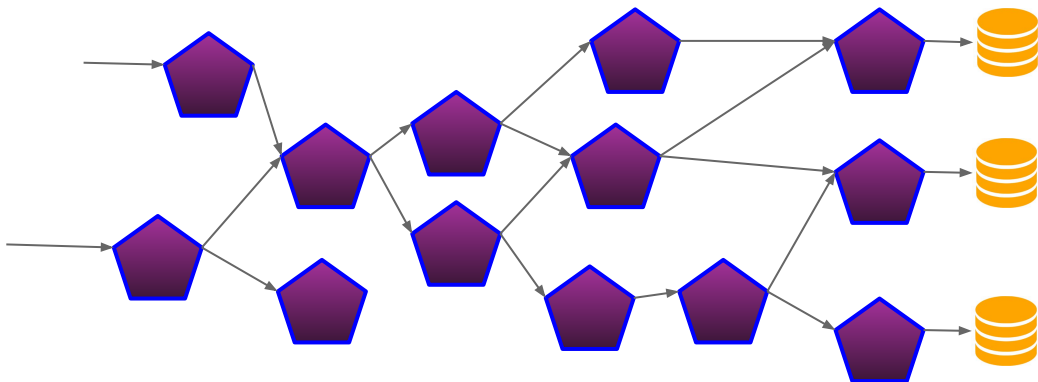
Nem tudo são flores

- Sistemas distribuídos
- Refactorings entre serviços
- Mudança de interface (protocolo)
- Complexidade operacional
 - Monitoramento
 - Detecção de problemas
- Múltiplas pipelines



Microserviços

Habilidades desejadas para cada serviço

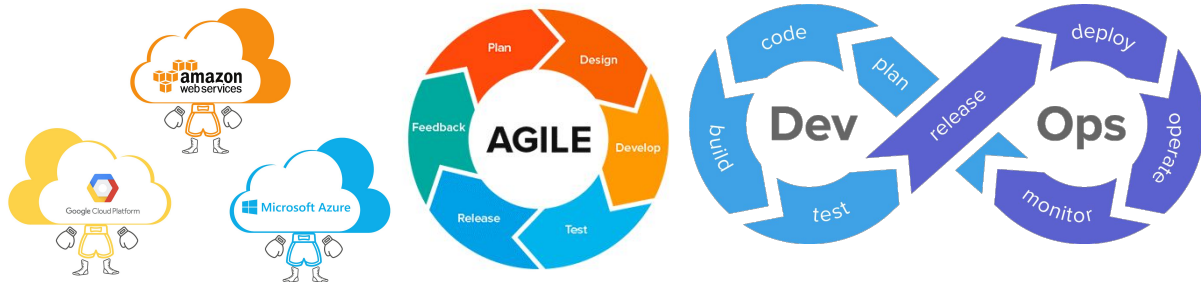


- API
- *Discovery*
- *Load Balancer*
- *Elasticidade*
- Resiliência
- *Logs*
- Monitoramento
- Autenticação
- *Tracing*

Microsserviços

Arquitetura que força evolução

- Padrão que força evolução
 - Desenvolvimento ágil
 - DevOPS
 - Automação
 - *Continuous Integration*
 - *Continuous Delivery*
 - Estratégias de *Deploy*
 - **Cloud**

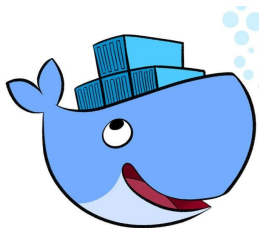


Cloud Computing Platform

Arquitetura que força evolução

- **Cloud**

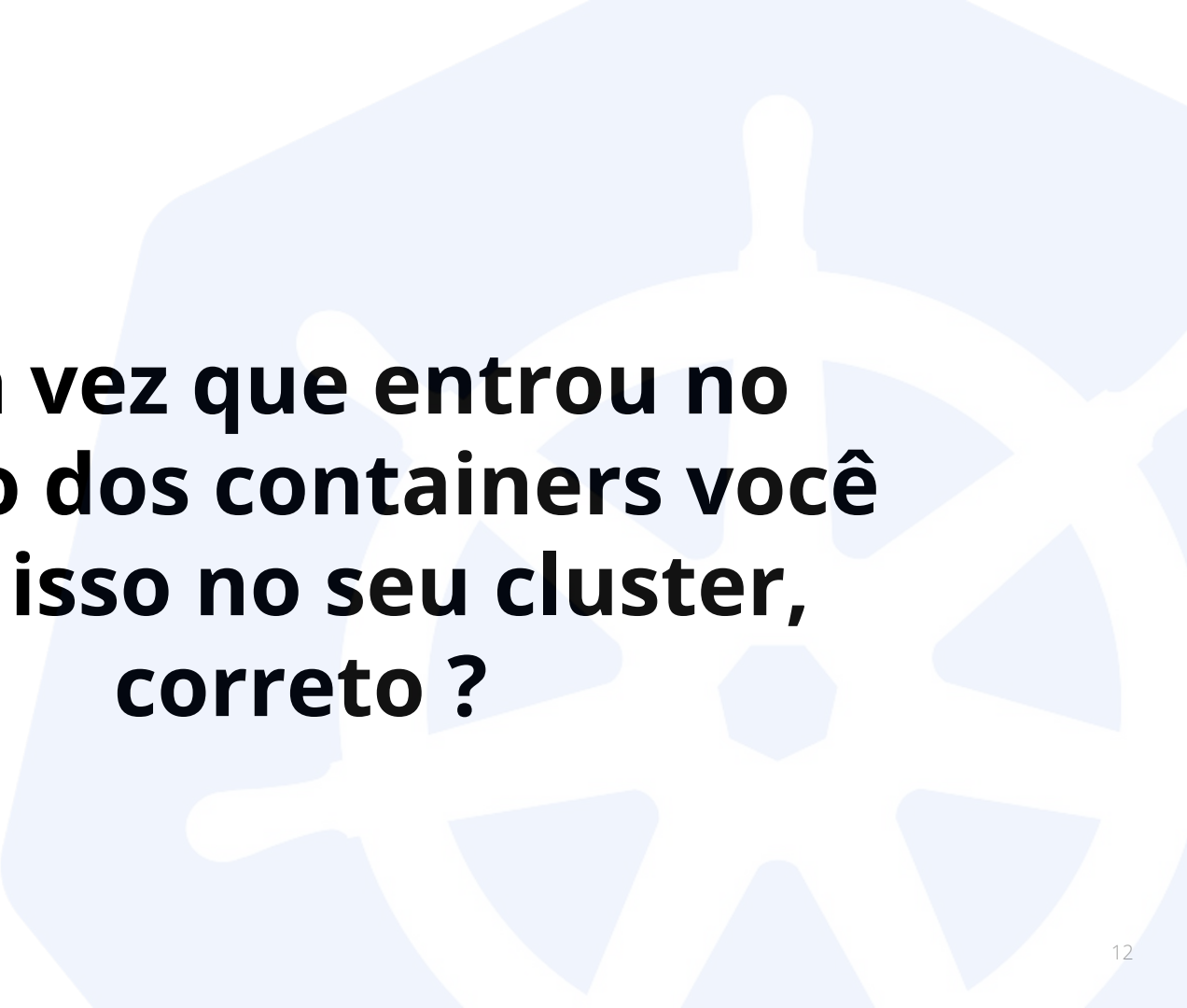
- *Self-Service environments*
 - *Configuração ideal*
 - *Otimização*
- Menos custo
- Mais agilidade
- Estabilidade
- Fácil escalabilidade
- **Container-based**



Containers (Docker)

- Empacotar sua app e o que ela precisa
 - *Libraries, file systems, web servers, etc.*
- Imagem / Containers são bem leves
- Executar o Container é o suficiente
- Rápidos de subir
- Imagens Docker são artefatos *deployable*



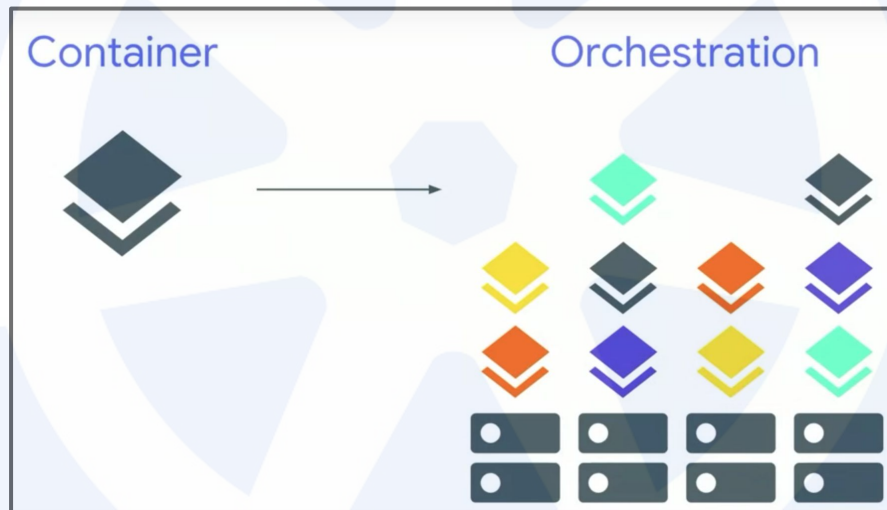


**Uma vez que entrou no
mundo dos containers você
quer isso no seu cluster,
correto ?**

Kubernetes



Orquestrando containers

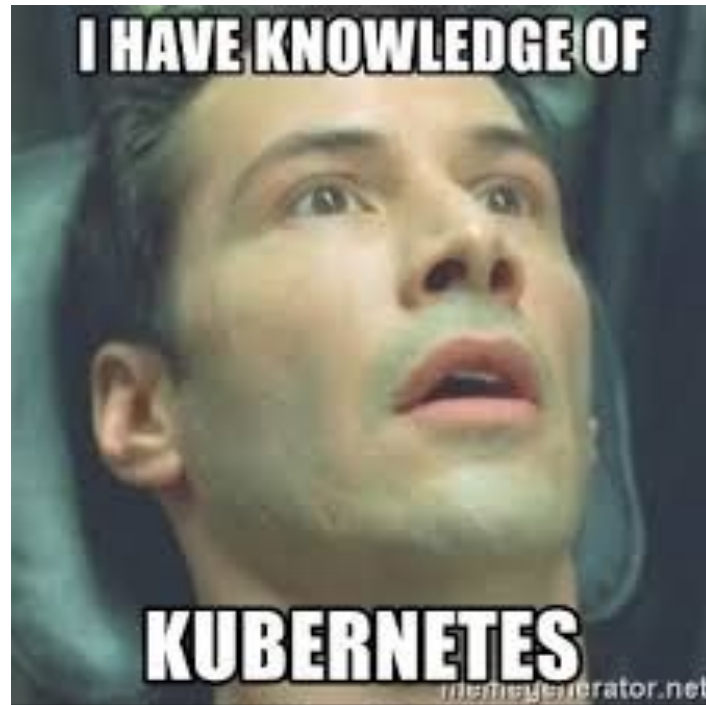
- Fazer manualmente?
- Orquestrador de containers
- Pods
- Se o servidor falhar?
- Se a aplicação falhar?
- Escalabilidade (para mais ou para menos)
- Sintaxe declarativa



Kubernetes

Habilidades desejadas para cada serviço

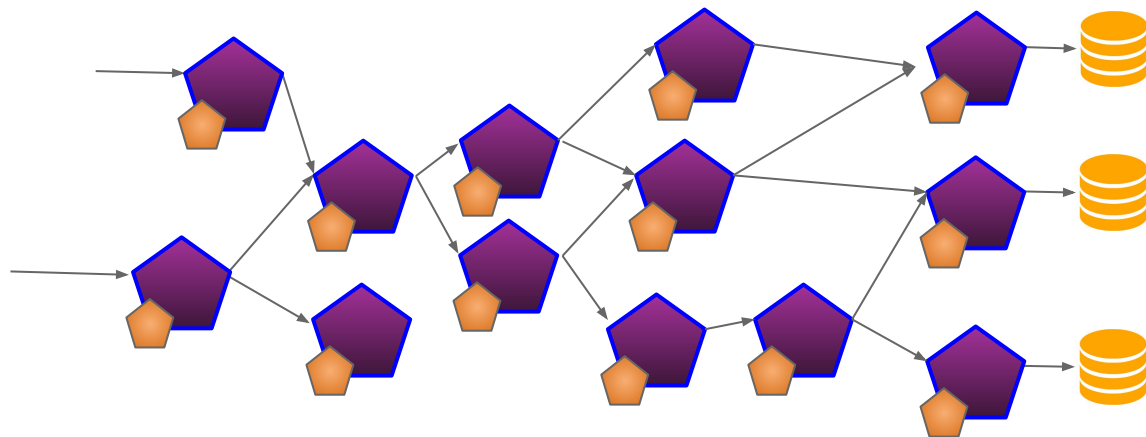
- *Discovery* 
- *Load Balancer* 
- *Elasticidade* 
- *Resiliência* 
- *Logs*
- *Monitoramento*
- *Autenticação*
- *Tracing*



Service Mesh

Controle na integração de microserviços

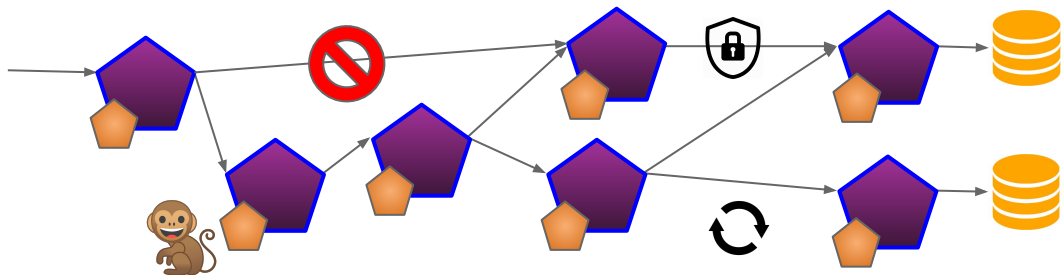
- Camada de infraestrutura (não de aplicação) dedicada
- Analogia: Programação orientada a Aspectos (Só que pra Sistemas distribuídos)
- Responsável por adicionar habilidades em todos serviços numa topologia complexa de microserviços
- **Proxy** implantado junto com a aplicação (**Sidecar Containers**)




Service Mesh

PROXY Sidecar containers <3

- Retrys e *Timeouts*
- Autenticação
- Controle de acesso
- Controle de tráfego
- *Chaos Engineering*
 - *Falhas programadas*
 - *Lentidão*



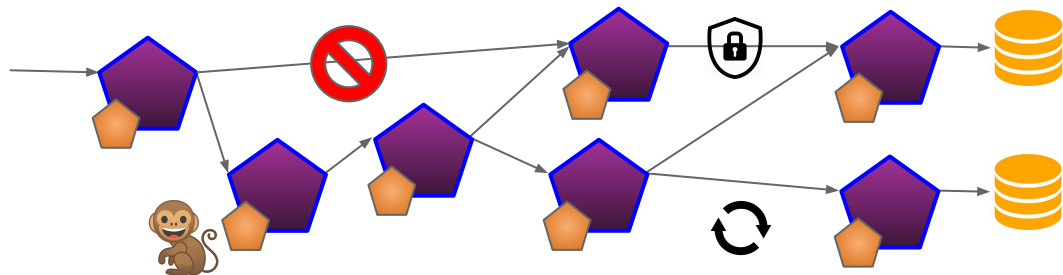
- *Discovery* 
- *Load Balancer* 
- *Elasticidade* 
- *Resiliência* 
- Logs
- Monitoramento
- Autenticação
- Tracing




Service Mesh

PROXY Sidecar containers <3

- Monitoramento
- *Tracing*
- Logs
- *Blue/Green e Canary deploys* mais fáceis
- *Dark Launching*
- *Smart Pipes and Smart Endpoints*

Por favor, tudo isso na pipeline.



- *Discovery* 
- *Load Balancer* 
- *Elasticidade* 
- *Resiliência* 
- Logs
- Monitoramento
- Autenticação
- Tracing

Principais soluções de Service Mesh



LINKERD

Buoyant



Istio

Lyft, IBM, Google



HashiCorp

Consul

HashiCorp



envoy

Lyft

Open Source



JAEGER



Grafana



Prometheus



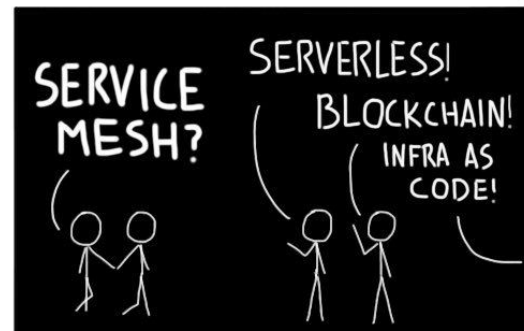
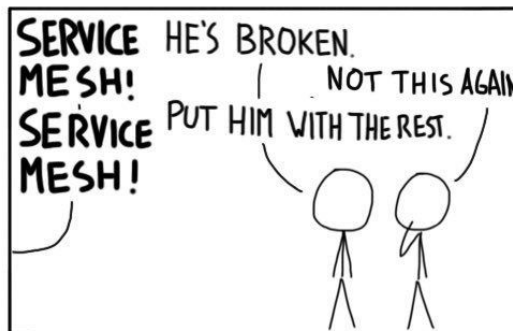
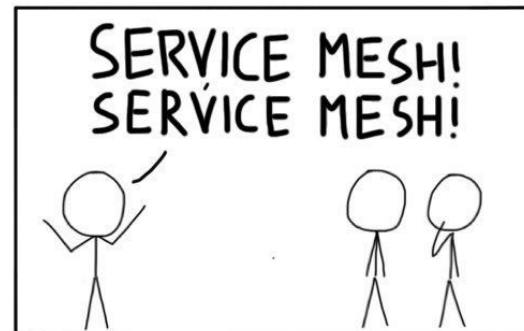
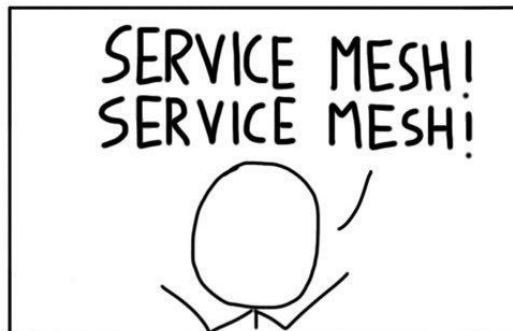
kiali



Flagger

Futuro do Service Mesh?

- Service Mesh Interface (SMI)
- Serverless



@sebiwicb

Obrigado

Thiago Gregório

@thiagogregorio

thiago.gregorio@gmail.com