

# QCon Rio 2015

## Matando um Sistema Monolítico Rumo Aos Microservices

**Bernardo Fontes**

Rio de Janeiro/RJ

27 de Agosto de 2015

**Olar!**

[twitter.com/bbfontes](https://twitter.com/bbfontes) (<https://twitter.com/bbfontes>)

[github.com/berinhard](https://github.com/berinhard) (<https://github.com/berinhard>)

[garimpo.fm](http://garimpo.fm/) (<http://garimpo.fm/>)

[pessoas.cc](http://pessoas.cc/) (<http://pessoas.cc/>)

[bernardoxhc@gmail.com](mailto:bernardoxhc@gmail.com) (<mailto:bernardoxhc@gmail.com>)

[\(http://elgroup.com.br/\)](http://elgroup.com.br/)

[berinhard.github.io/talks/](https://berinhard.github.io/talks/) (<https://berinhard.github.io/talks/>)

[slideshare.net/bernardofontes](http://slideshare.net/bernardofontes) (<http://slideshare.net/bernardofontes/>)

## Quem são vocês?

## Roteiro - ### Nosso Cenário - ### Microservices - ### Domain Driven Design - ### Implantação - ### Dúvidas

□

## Nosso Cenário ### Sistema para um empresa de \*\*medicina do trabalho\*\* realizar atendimentos de \*\*medicina ocupacional\*\* por todo o Brasil e fornecer uma \*\*análise inteligente\*\* sobre o perfil dos colaboradores de uma empresa.

**Maaaaaaas...**

**Tudo começou só com uma filinha:**

□

# Hoje em dia...

Até [envia email](#)

[https://en.wikipedia.org/wiki/Jamie\\_Zawinski#Zawinski.27s\\_law\\_of\\_software\\_envelopment](https://en.wikipedia.org/wiki/Jamie_Zawinski#Zawinski.27s_law_of_software_envelopment)

□

## Histórico do Projeto - ### + de \*\*5 anos\*\* - ### + de \*\*10 devs\*\* passaram - ### Python e \*\*Django 1.4\*\* - ### Hoje: 4 devs e \*\*únicos Pythonistas\*\* - ### Empresa com \*\*12 devs\*\*

## Histórico do Projeto - ### Em \*\*Outubro de 2014\*\* - ### 3 devs - ### 1 saindo em Dezembro - ### Django com \*\*uma única aplicação\*\* - ### 3k LOC de \*\*models.py\*\* (Banco de Dados) - ### ~3k LOC de \*\*views.py\*\* (Controllers) - ### ~2,5k LOC de \*\*services.py\*\*

## Menos de 50% de coverage ![No tests](images/no\_test.jpg)

## Infra do Projeto - ### 9 filiais == 9 máquinas - ### 1 ambiente de clientes - ### 1 única máquina do banco - ### \*\*11 máquinas\*\*

## \$ fab all\_hosts deploy ![XKCD](images/deploying.png)

## Cliente - ### \*\*Novos módulos\*\* no sistema - ### \*\*Melhorias\*\* nos antigos - ### Correção de \*\*bugs\*\*

![Work](images/working.png)

### Estudamos ![Library](images/library.jpg)

## Estudamos

□

## Microservices Um estilo arquitetural para o desenvolvimento de \*\*serviços enxutos\*\* em que cada um possa ser \*\*executado em um processo próprio\*\* e se comunicando através de mecanismos de fácil implementação como o protocolo HTTP. Esses serviços são \*\*construídos focando capacidades de negócio\*\* e devem poder ser \*\*deployados independentemente\*\* através de um processo automatizado. A \*\*necessidade de gestão centralizada deve ser mínima\*\* para os serviços visando viabilizar a independência entre eles.

## Componentes por Serviços - ### Componentes: unidades de software \*\*independentes\*\* - ### Serviços: \*\*componentes externos\*\* com comunicação por API

## Foco em Capacidades do Negócio - ### Serviços limitados ao \*\*contexto do problema\*\* - ### Equipe precisa saber só do \*\*contexto específico\*\* e não mais do todo

## Pensamento em ~~Projeto~~ Produto - ### Fim da \*\*Lei de Conway\*\* - ### Domínio menor == entendimento mais simples - ### Entendimento mais simples == equipe responsável por \*\*todo o serviço\*\* - ### Deploy independentemente

## Smart Endpoints & Dumb Pipes - ### Foco em \*\*coesão\*\* e \*\*desacoplamento\*\* - ### Chamadas por métodos em memória viram \*\*chamadas ao serviço\*\* - ### HTTP >> \*\*síncrono\*\* - ### Mensageria >> \*\*assíncrono\*\*

## Governança Descentralizada - ### \*\*Decisões específicas\*\* para o microservice - ### \*\*Right tool\*\* for the job - ### \*\*Independência\*\* no processo de desenvolvimento da equipe

## Armazenamento de Dados Descentralizados - ### Dados sendo visualizados de \*\*acordo com o contexto\*\* - ### \*\*Perde-se\*\* a gestão de transações automática

## Design orientado a Falhas - ### \*\*Falhas de comunicação\*\* são sempre reais - ### A comunicação deve ser sempre \*\*desenvolvida pensando o cenário de falha\*\* - ### Código \*\*mais estável\*\*

## Conclusões do que poderíamos - ### Desenvolver em \*\*outras tecnologias\*\* - ### Ter rotinas de \*\*deploy mais simples\*\* - ### Envolver \*\*mais devs\*\* no projeto - ### Entregar \*\*mais rápido\*\* - ### Gerar software de \*\*maior qualidade\*\* - ### Focar em entregar \*\*mais valor\*\* para o cliente

## Nossa reação ![Mind](images/mind.gif)

## E começamos... - ![Dog](images/no\_idea.png)

## Trade-offs que Encontramos - ### Aumento da \*\*complexidade operacional\*\* - ### Ambiente de dev mais burocrático (resolvemos com o Docker) - ### Diferentes processos de deploy - ### Problemas para garantir a \*\*consistência dos dados\*\* - ### Todos os overheads de \*\*comunicação em sistemas distribuídos\*\*

## Killer Problems - ### Nossas regras de negócios estavam \*\*completamente acopladas\*\* entre os módulos services.py, models.py e views.py - ### A \*\*modelagem acoplada\*\* do nosso banco de dados - ### Impossível de mudar - ### Todos esses problemas foram refletidos pros novos serviços

![DatabaseFull](images/whole\_db.jpg)

![Database](images/database.jpg)

## Solução - ![Legacy](images/giphy.gif)  
## "É preciso tirar o legado da cabeça"

## Então estudamos mais...

## Focos no Mindset - ### Definição dos **Bounded Context** dos serviços - ### Utilização de **Domain Objects** na implementação - ### Criação de **Use Cases** orquestrando a troca de mensagem entre os objetos - ### Implementação de **Anti-Corruption Layer** nos serviços  
## Extra Bonus - ### Indo além dos CRUDs com **CQRS** - ### **Event Sourcing** para reduzir intervenção direta no legado

### ## Implementação em Etapas

## 1º Caso: Isolar lógica de um domínio (sem microservice) - Objetivo: **aprender a isolar a lógica** - Modelos de dados continuaram no legado - Nova aplicação interna somente lidando com um tipo de atendimento - Já existiam testes - Migramos os controllers, services, rotas e testes isolando a app - Ponto de acoplamento: import dos models do legado

## 2º Caso: Nova funcionalidade com baixo acoplamento (sem microservice) - Objetivo: aprender a **organizar o código** para ser **consumido por clientes externos** - Modelos, controllers, services e urls próprios em nova app - Utilização de eventos para não mexer no meio legado - Integração através de import em listeners no legado - Ponto de acoplamento: único import para o model legado de Paciente

## 3º Caso: Autenticação como Microservice - Objetivo: validar **participação de outros devs** no projeto - App SSO em Sinatra (Ruby) - **Todo** o ambiente isolado - No legado mudamos a fina camada autenticação

## 4º Caso: Nova funcionalidade agnóstica (microservice interno) - Objetivo: ser possível **deployar isoladamente** no futuro - App plugável do Django - Criação de UC para encapsular lógica de domínio - Integração através de disparo de mensagens via RabbitMq - Modelos sem semântica alguma do domínio do legado - Banco de dados próprio (máquina própria pro banco)

## 5º Caso: Nova Aplicação em Microservice - Objetivo: implementar **microservices por completo** - App Flask (lightweight web framework Python) de ambiente isolado - Utilização de comunicação por RabbitMq **e** API Rest - Tivemos que desenvolver uma API para expor o legado

```
### Proxy de Saída ```python class ConsumeExamsAllocationsProxy(BaseProxy): task_name =
settings.CONSUME_SCHEDULE_EXAMS_ALLOCATIONS def send(self, exam_request, schedule): scheduling_time
= schedule.data_agendamento.isoformat() exams_ids = exam_request.exames_internos.values_list('id', flat=True)
kwargs = { 'scheduling_time': scheduling_time, 'exams_ids': exams_ids, 'subsidiary_id': schedule.filial_id, }
self.send_task(kwargs=kwargs) ```
### Fronteira de Entrada ```python from scheduling.use_cases import ConsumeScheduleSlots class
ConsumeScheduleSlotsExecutor(object): """ Adapter to Consume Scheduel Slots Use Case """ def __init__(self, *args,
**kwargs): self.__blocked_attrs = ['repository'] self.instance = ConsumeScheduleSlots(*args, **kwargs) def
__getattr__(self, name): if name in self.__blocked_attrs: raise AttributeError(u'Attribute %s does not exist.' % name)
return getattr(self.instance, name) ```
## 6º Caso (atual): Kill the Legacy! - Objetivo: **substituir implementações** do legado - Bibliotecas Python
completamente indepententes - Instalação por pacotes - Utilização no legado como dependência do projeto - Organização
das libs por **contexto** do legado
```

□

## Conclusões - ### Tem \*\*muitos\*\* trade-offs para serem pensados - ### Não usamos microservices para tudo - ###  
Foco em \*\*modularização\*\* - ### Diminuímos o \*\*tempo de entrega\*\* - ### Diminuímos a \*\*barreira de entrada\*\* no  
projeto

## Conclusões - ### Aumentamos o \*\*nível técnico\*\* da equipe - ### \*\*Monitoramento efetivo\*\* é muito importante -  
### \*\*Logs\*\* são nossos melhores amigos - ### Qualquer operação feita 2 vezes, deve ser automatizada

## 11ª PythonBrasil

□ (<https://pythonbrasil.github.io/pythonbrasil11-site/>)

## Obrigado!

**Bernardo Fontes**

[twitter.com/bbfontes](https://twitter.com/bbfontes) (<https://twitter.com/bbfontes>)

[github.com/berinhard](https://github.com/berinhard) (<https://github.com/berinhard>)

[garimpo.fm](http://garimpo.fm/) (<http://garimpo.fm/>)

[pessoas.cc](http://pessoas.cc/) (<http://pessoas.cc/>)

[bernardoxhc@gmail.com](mailto:bernardoxhc@gmail.com) (<mailto:bernardoxhc@gmail.com>)

---

[\(http://elogroup.com.br/\)](http://elogroup.com.br/)