Web apps em qualquer lugar

12 Factor App

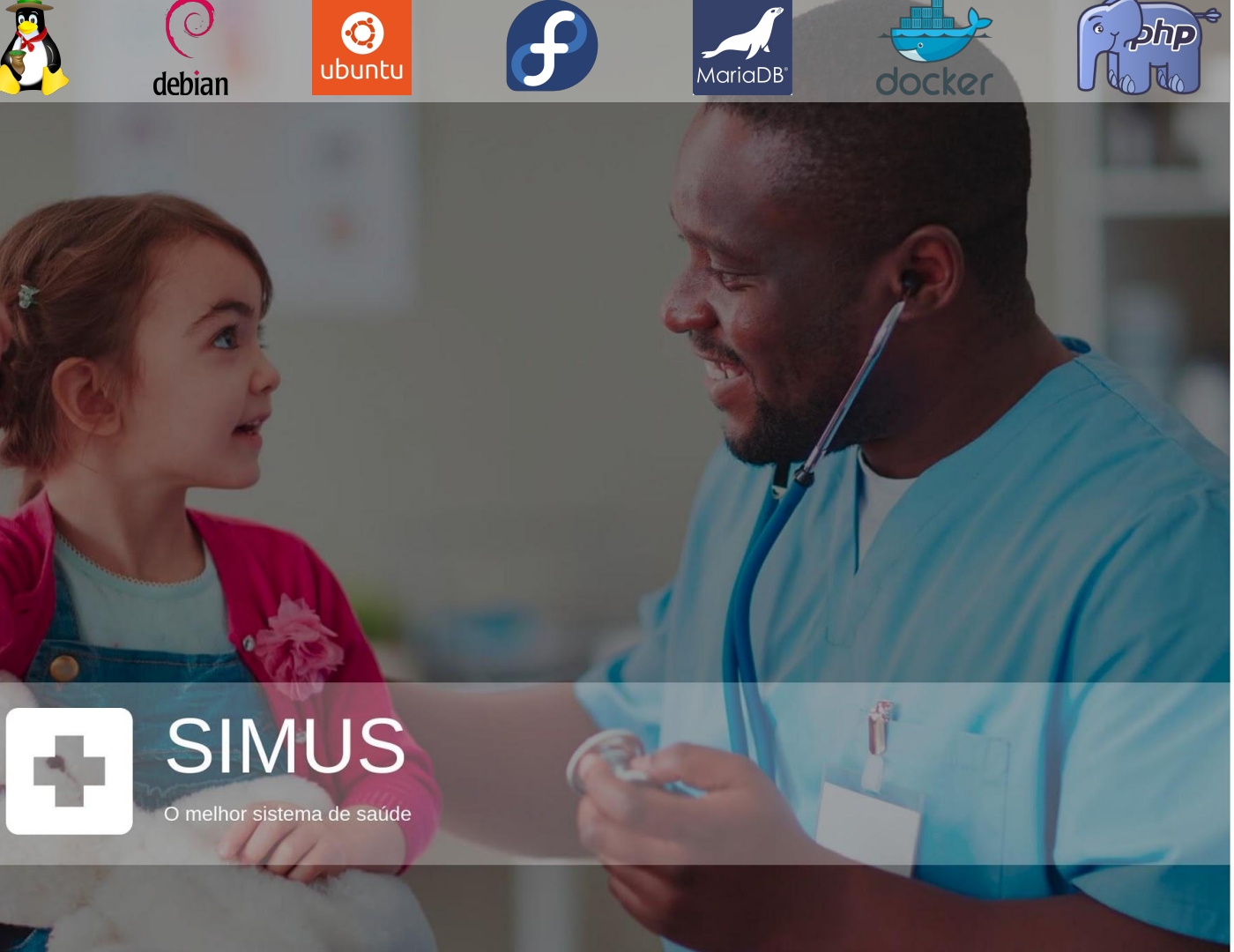
TONIN R. BOLZAN

Software Engineer ❤️ DevOps

CTO na Metrosoft S/A - Santa Maria/RS

12 anos de experiência

https://bolzan.io



METROSOFT S/A

Respiramos Software Livre

Especialista em Software para Saúde

SUS / Privado

Milhares de cidadãos atendidos diariamente.

Operacional e Gestão

DEVOPS
DAYS

TcheLinux
Santa Maria

MeetUp Tech
Santa Maria

Rock the Software
Freedom Day
www.softwarefreedomday.org

FLISOL

12 Factor App

É uma metodologia para construir SaaS Web

- Construir aplicações web que **possam escalar**
- Não **gaste seu tempo** desnecessariamente quando o fizerem
- **Suportar os Cloud** Providers e CI/CDs modernos
- **Evitar Lock-in** - portabilidade
- **Automatizar** tarefas
- **Agnóstico** - Aplicável a qualquer linguagem ou tecnologia
- São padrões e recomendações, **não imposições**
- Criado pela Heroku em 2011 - Adotado por todo mundo

`https://12factor.net/pt_br/`

Os Doze Fatores

I. Base de Código
Uma base de código com rastreamento utilizando controle de revisão, muitos deploys

II. Dependências
Declare e isole as dependências

III. Configurações
Armazene as configurações no ambiente

IV. Serviços de Apoio
Trate os serviços de apoio, como recursos ligados

V. Build, Release, Run
Separe estritamente os builds e execute em estágios

VI. Processos
Execute a aplicação como um ou mais processos que não armazenam estado

VII. Vínculo de porta
Exporte serviços por ligação de porta

VIII. Concorrência
Dimensione por um modelo de processo

IX. Descartabilidade
Maximizar a robustez com inicialização e desligamento rápido

X. Dev/prod semelhantes
Mantenha o desenvolvimento, teste, produção o mais semelhante possível

XI. Logs
Trate logs como fluxo de eventos

XII. Processos de Admin
Executar tarefas de administração/gerenciamento como processos pontuais

Fator I. Base de Código

Teoria

Uma base de código com rastreamento utilizando controle de revisão, muitos deploys;

Múltiplos repositórios podem existir se compartilhar o registro raiz. Ex.: git submodules;

Múltiplas bases de código compoem um sistema distribuido, cada repositório é uma app;

Múltiplas apps compartilhando uma base de código é uma violação;

Uma base de código gera multiplos builds/deploys - dev, prod, staging

Prática

Apenas uma base de código por aplicação

Repositório de código = Git, Mercurial ou Subversion

builds/deploys podem ser branches dentro do vcs

Fator II. Dependências

Teoria

Declare e isole explicitamente as dependências

Podem ser dependências globais (site packages) ou locais (vendoring)

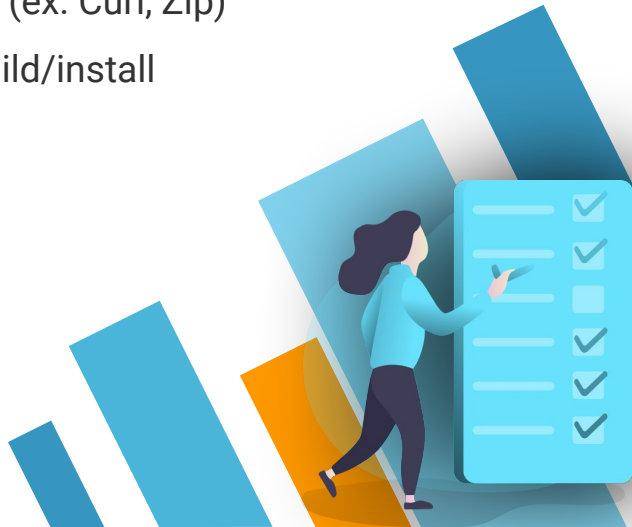
Declarar todas as dependências, completa e exatamente, por meio de um arquivo

Nunca confiar na existência implícita de pacotes em todo o sistema (ex. Curl, Zip)

Centralização da instalação em um único comando de validação/build/install

Prática

```
S/O    = apt, dnf
PHP    = composer, pecl, pear
Python = pip
Ruby   = rubygems
```



Fator III. Configurações

Teoria

Configurações são variações entre ambientes ou deploys (dev, prod, staging)

Você deve armazenar as configurações em var. de ambiente de forma granular, sem agrupamento

Colocar configurações no código em constantes, objetos ou strings é uma violação

Estrita separação entre configuração e código, código não deve variar entre ambientes

Prática

Ex.: Credenciais para AWS S3, facebook, google, string de conexão BD

Seus arquivos .env não devem ser versionados no código

Eles devem ser carregados no ambiente e não na aplicação

Pode utilizar um .env para cada ambiente

Seu código não deve saber em que ambiente está rodando

Você abriria o código da sua aplicação no github sem se preocupar?

Fator IV. Serviços de Apoio

Teoria

Trate os serviços de apoio como recursos anexos que o app consuma externamente

Cada serviço distinto é um recurso, um banco master/slave se comporta como 2 recursos

Normalmente são serviços locais, mas podem se tornar remotos

A troca de um serviço de apoio por outro equivalente não deve ter alterações de código

Prática

Permitir a troca do banco de dados de produção pelo backup de testes

Permitir a troca de MySQL pela Amazon RDS sem mexer no código, só configurações

Permitir a troca do envio de email local pelo MailGun

Ex.: MySQL, Redis, RabbitMQ, eMail SMTP, WS SOAP/Rest

Ex.: Acesso a Disco (VFS)

Fator V. Build, release, run

Teoria

Separe estritamente os estágios de construção, lançamento e execução (pipeline)

Uma base de código limpa se torna um deploy de produção depois de uma build

Impossível alterar o código em tempo de execução (sorry wordpress)

Prática

```
docker build -t $IMAGE_NAME -f Dockerfile .
docker push $IMAGE_NAME
docker run $IMAGE_NAME

php composer.phar install --no-dev -o
Ex.: Docker, Capistrano, deployer
```

Fator VI. Processos

Teoria

Execute o app como um ou mais processos **stateless** e **share-nothing**

Toda a persistência deve ser armazenada em serviços de apoio, até a sessão

O FS pode ser usado como unidade de transição temporária, mas não como armazenamento

Não confie no estado atual da aplicação

Prática

Um upload do browser para o Nginx/Apache utiliza um arquivo temporário

Uma reinicialização pode quebrar o seu app?

Fator VII. Vínculo de porta

Teoria

Exporte todos os serviços através de vínculos de porta

Um app doze-fatores é completamente auto-contido, exportando somente o serviço final

Não depende do domínio ou de uma porta específica

Prática

Exportar HTTP, WebSockets, XMPP, FastCGI diretamente para um porta

Fator VIII. Concorrência

Teoria

Criar mais processos deve ser uma operação simples e confiável

Apps nunca deveriam daemonizar ou escrever arquivos PID

Confie no gerente de processos do sistema operacional (docker, systemd, supervisord)

Prática

Iniciar 1 instancia do app e criar N deve ser natural

Não faça sua aplicação controlar o próprio estado

Fator IX. Descartabilidade

Teoria

Maximize robustez com inicialização rápida e desligamento gracioso

Os processos podem ser iniciados ou parados a qualquer momento

Robustos contra morte súbita

Prática

Seu app deve iniciar rápido e começar a responder requisições em menos de 1 segundo

Deletar seu app e cria-lo novamente pode quebrar o seu app?

FatorX. Dev/prod semelhantes

Teoria

Mantenha o desenvolvimento, homologação e produção o mais similares possível

Evite as Lacunas entre os ambientes

Lacuna de tempo: Dev pode escrever um código e ter o deploy feito em minutos ou horas

Lacuna de pessoal: Devs devem envolver-se em realizar o deploy e acompanhar em produção

Lacuna de ferramentas: Mantenha desenvolvimento e produção o mais similares possível

Prática

Utilize a mesma IDE, Sistema Operacional, etc... que o resto da equipe

Utilize os mesmo serviços de apoio em desenvolvimento e produção

Evitar o famoso “funciona no meu computador”

Fator XI. Logs

Teoria

Trate logs como fluxos de eventos, normalmente no stdout/stderr, evite fixar em um arquivo

Não se preocupe com o roteamento, buffer ou armazenagem do seu fluxo de saída

Pense que alguém fora da sua aplicação deve ser responsável por esse gerenciamento

Prática

Saídas stdout/stderr ficam a cargo do gerente de processo (docker, systemd, supervisord)

Em ambientes de desenvolvimento o fluxo de eventos pode ser no terminal

Em produção pode ser coletado por um agregador de logs como o Fluentd

Fator XII. Processos de Admin

Teoria

Rode tarefas de administração/gestão em processos pontuais

Códigos de administração devem ser fornecidos com o código da aplicação

Deve usar a mesma técnica de isolamento de dependência. Ex.: php composer

Devem ser executados em um ambiente idêntico

Prática

Não colocar processos administrativos em urls web, rode-os em cli

Ex.: Executar migrações de base de dados

Ex.: Executar um console REPL (bash, php -a, python, irb)

Perguntas

Obrigado

Slides: https://bit.do/tchelinuxsc19