DO DESENVOLVIMENTO PARA A IMPLANTAÇÃO

ACH2006 – ENGENHARIA DE SISTEMAS DE INFORMAÇÃO
SIN5005 – TÓPICOS EM ENGENHARIA DE SOFTWARE

Daniel Cordeiro

Escola de Artes, Ciências e Humanidades | EACH | USP

AGENDA

- Disponibilidade & Responsividade
- Apdex
- Monitoramento
- Atualizações & flags de funcionalidades
- Protegendo os dados do cliente

DESENVOLVIMENTO VS. IMPLANTAÇÃO

Desenvolvimento
Testes para garantir que seu app funciona tal como projetado.

Implantação
Testes para garantir que seu app funciona quando usado de forma para o qual ele não foi projetado para ser usado.

MÁS NOTÍCIAS

- “Usuários são seres terríveis”
- Alguns bugs só aparecem quando o sistema está sob estresse
- Ambiente de produção ≠ ambiente de desenvolvimento
- O mundo está cheio de forças do mal
- ... e de idiotas

BOAS NOTÍCIAS: PAAS DEIXOU A IMPLANTAÇÃO MUITO MAIS FÁCIL

- arrume um Servidor Virtual Privado (*Virtual Private Server* ou VPS), talvez em uma plataforma de computação em nuvem
- instale & configure Linux, Rails, Apache, *mysqld*, openssl, sshd, ipchains, squid, qmail, logrotate, ...
- corrija (quase que toda semana) as vulnerabilidades de segurança
- descubra que você se encontra em uma *Library Hell*
- ajuste tudo que puder para conseguir o máximo possível por cada $ investido
- descubra um jeito de automatizar a escalabilidade horizontal

NOSSO OBJETIVO: SE MANTER A UM PAAS

PaaS gerencia...	Nós gerenciamos
As camadas “fáceis” de conseguir escalabilidade horizontal	Minimização da carga no banco de dados
Ajustes no desempenho dos componentes do sistema	Ajustes no desempenho da aplicação (ex: caching)
Segurança no nível da infraestrutura	Segurança no nível da aplicação

Mas isso é factível na prática?

- Pivotal Tracker & Basecamp rodam cada um em um único BD (computador commodity de 128GB < US$ 10 mil)
- Muitos apps SaaS não operam em escala global (interno ou de interesse limitado)
- DevOps is dead. Long live DevOps![1]

[1] https://techcrunch.com/2016/04/07/devops-is-dead-long-live-devops/

DESEMPENHO & SEGURANÇA

- Disponibilidade ou *uptime*

  Qual % do tempo em que o site está no ar & acessível?
- Responsividade

  Quanto tempo demora do clique do usuário até ele ver a resposta?
- Escalabilidade

  A medida que o # usuários aumenta, você consegue manter a responsividade sem aumentar o custo/usuário?

- Privacidade

  O acesso aos dados é limitado apenas aos usuários apropriados?
- Autenticação

  Podemos confiar que o usuário é quem ele diz ser?
- Integridade de dados

  É possível perceber uma violação dos dados mais sensíveis do usuário?

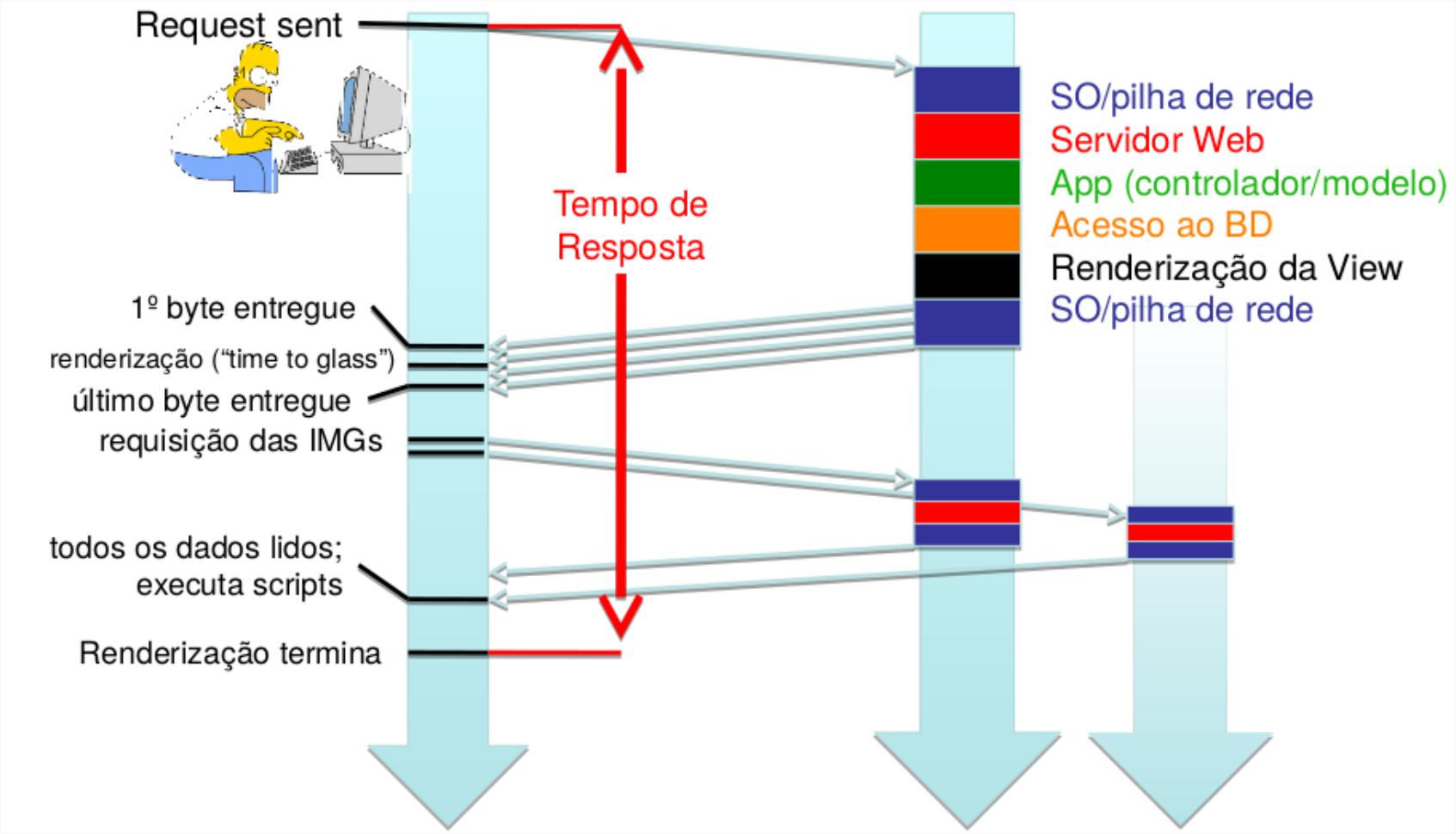
Quantificando a Disponibilidade & Responsividade

O TEMPO DE RESPOSTA É IMPORTANTE?

- Quão importante é o tempo de resposta?[2]
  - Amazon: +100ms → queda de 1% nas vendas
  - Yahoo!: +400ms → queda de 5–9% do tráfego
  - Google: +500ms → 20% menos buscas
- Estudos clássicos (Miller, 1968; Bhatti, 2000)
  - $<$ 100ms é "instantâneo"
  - $>$ 7s já é hora de desistir
- `http://code.google.com/speed`

[2] Fonte: Nicole Sullivan (Yahoo! Inc.), Design Fast Websites, `http://www.slideshare.net/stubbornella/designing-fast-websites-presentation`

PARA ONDE O TEMPO VAI (SERVIDOR/REDE)?



PARA ONDE O TEMPO VAI (CLIENTE)?

- Seletores CSS + Interpretador JavaScript = 41% do tempo de renderização no cliente
  - especialmente seletores que requerem percorrer a árvore DOM (ex: `div > li`)
- Navegadores competem na velocidade de seus interpretadores JavaScript ⇒ desempenho do seletor/parser é frequentemente o gargalo

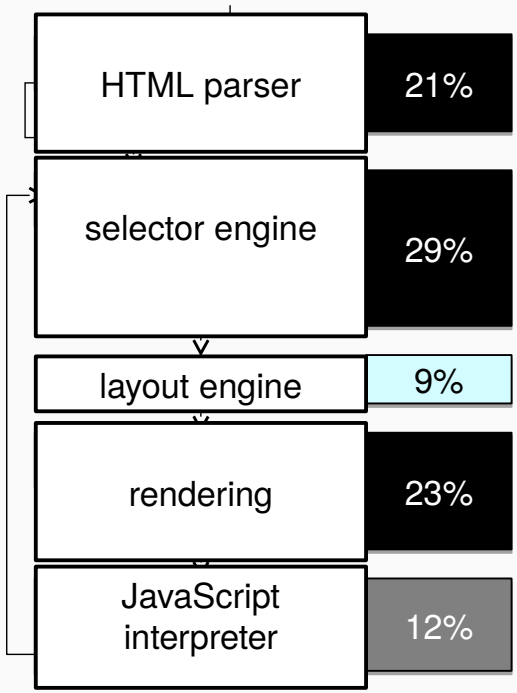


Figura 1: Cortesia de Leo Meyerovich, UC Berkeley

ESCALABILIDADE

- Um termo “usado e abusado” — significa coisas diferentes em áreas diferentes
- Para SaaS: a medida que o número de usuários aumenta, o tempo de resposta que o usuário obtém se mantém o mesmo
  - tempo de resposta é uma métrica chave para medir como usuário percebe o seu app
- Idealmente você também gostaria de: a medida que o número de usuários aumenta, o custo de servir cada usuário se mantém o mesmo (ou decresce)
  - uma métrica possível: usuários por servidor por $
  - captura o efeito da vazão (*throughput*) (ou “largura da banda”)

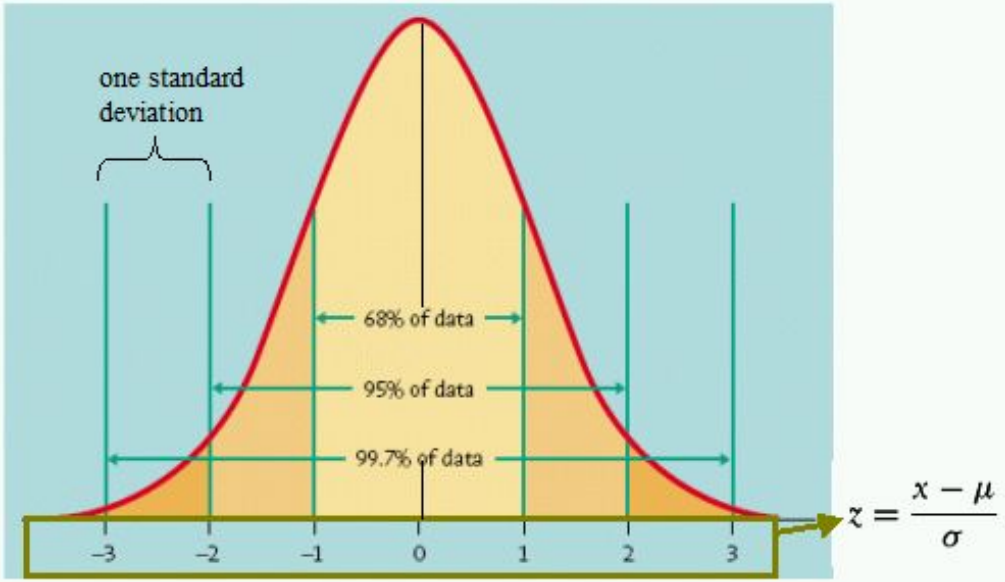
OBJETIVOS DE NÍVEL DE SERVIÇO

- *Service Level Objective* (SLO)
- Tempo para satisfazer a requisição do usuário (“latência” ou “tempo de resposta”)
- SLO: ao invés de medir o pior caso ou média mede a % de usuários que receberam um desempenho razoável
- Especifica a %, o tempo de resposta considerado ideal e uma janela de tempo
  - ex: 99% < 1 segundo, ao longo de uma janela de 5 minutos
  - por que a janela de tempo é importante?
- Acordo de nível de serviço (*service level agreement* ou SLA) é um SLO ao qual o provedor está contratualmente obrigado a oferecer

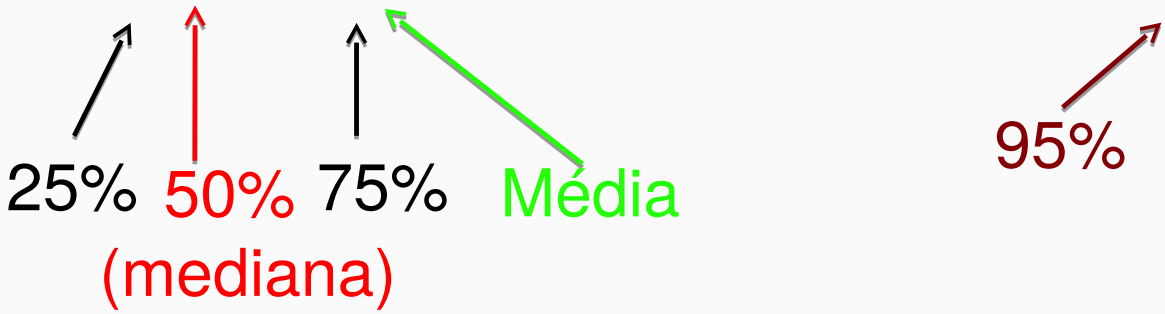
VISÃO SIMPLIFICADA (E FALSA) DO TEMPO DE RESPOSTA

- Com tempos de resposta seguindo uma *distribuição normal* em torno da média: o tempo de resposta está em média ± 2 desvio padrão com 95% de confiança
- Tempo de resposta *médio* T significa que:

- 95% dos usuários recebem $T + 2\sigma$
- 99,7% dos usuários recebem $T + 3\sigma$



UM EXEMPLO REAL



Courtesia de Bill Kayser, Distinguished Engineer, New Relic.
`http://blog.newrelic.com/breaking-down-apdex`.
Usado com permissão do autor.

APDEX: UM SLO SIMPLIFICADO

- Indicador de desempenho da aplicação (*Application Performance Index*)
- Dada uma latência T considerada limite para o usuário se sentir satisfeito:
  - requisições *satisfatórias* levam $t < T$
  - requisições *toleráveis* levam $T \leq t \leq 4T$
  - Apdex = (#satisfatórias + 0,5 #toleráveis) / #requisições
  - Valores entre 0,85 e 0,93 são considerados “bons”
- Cuidado! Pode esconder valores discrepantes se não for usado com cautela.
  - ex: ações críticas que ocorrem uma vez a cada 15 cliques, mas que levam 10x mais tempo $\Rightarrow (14 + 0)/15 > 0,9$

APDEX: VISUALIZAÇÃO

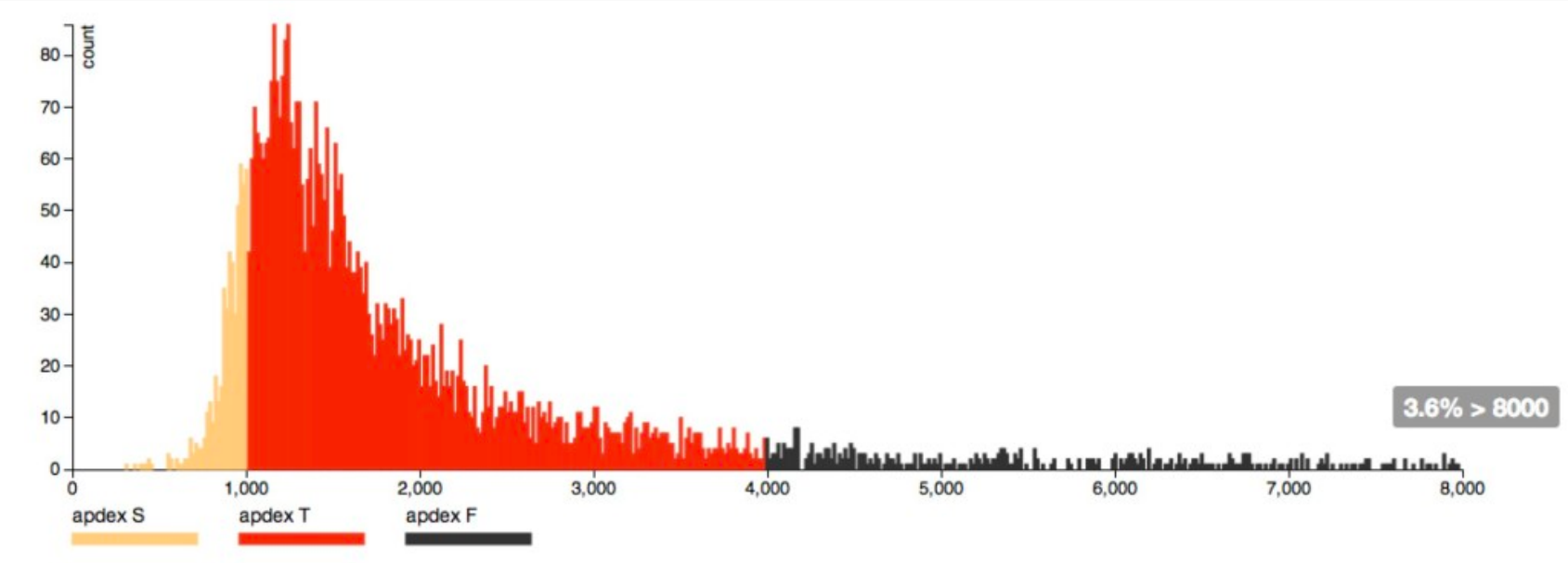


Figura 2: T = 1000ms, Apdex = 0,49

APDEX: VISUALIZAÇÃO (CONT.)

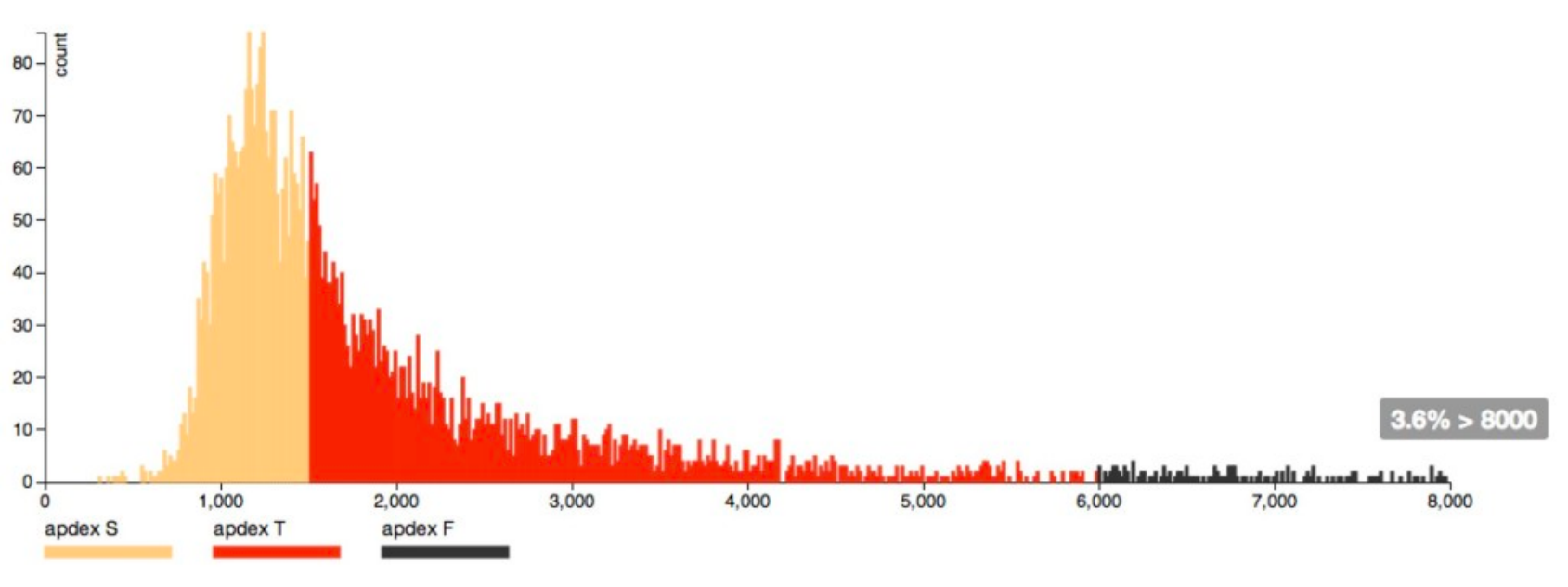


Figura 3: T = 1500ms, Apdex = 0,7

O QUE FAZER SE O SITE ESTIVER LENTO?

- Site pequeno: aumentar provisionamento
  - se aplica às camadas de apresentação & lógica
  - antes de computação em nuvem, era uma dor de cabeça
  - hoje em dia, pode ser totalmente automatizado (ex: Rightscale)
- Site grande: se preocupe
  - aumentar o provisionamento de um site com 10.000 computadores em 10% pode levar a 1.000 computadores ociosos
- Insight: os mesmos problemas que nos fazem optar por não usar algo PaaS-friendly são os problemas que vão nos atormentar quando estivermos maiores

Integração Contínua & Implantação Contínua

Implantações ontem e hoje:
Festa de lançamento do Windows 95

IMPLANTAÇÕES ONTEM E HOJE

- Facebook: ramo master é implantado uma vez por semana; objetivo é fazer isso uma vez por dia (Bobby Johnson, Diretor de Eng., no final de 2011)
- Amazon: várias implantações por semana
- StackOverflow: muitas implantações por dia (Jeff Atwood, cofundador)
- GitHub: dezenas de implantações por dia (Zach Holman)
- Lógica por trás disso: risco == # engenheiros–hora investidos no produto desde a última implantação!

Assim como o desenvolvimento e a incorporação de novas funcionalidades, **a implantação não deveria ser um evento**, mas sim algo que acontece o tempo todo.

IMPLANTAÇÃO BEM SUCEDIDA

Automação processo de implantação consistente

- sites PaaS como Heroku, CloudFoundry, etc. já fazem isso
- use ferramentas como o Capistrano para sites que você mesmo hospeda

Integração contínua (*continuous integration*) integre–teste o app após qualquer coisa que o desenvolvedor tenha feito

- código pré-lançamento dispara o sistema de CI
- como mudanças acontecem sempre, CI sempre está rodando
- estratégia comum: integre com GitHub

Veja: `https://help.github.com/articles/about-webhooks/`.

POR QUÊ CI?

- Diferenças entre ambientes de desenvolvimento e produção
- Testes de compatibilidade de navegador e de versão
- Testa a integração SOA quando os serviços remotos estão instáveis
- Fortalecimento (*hardening*): proteção contra ataques
- Testes de estresse / testes de longevidade para novas funcionalidades / caminhos de código
- Exemplo: o CI da Salesforce executa mais de 150 mil testes e relata bugs automaticamente quando algum deles falha

IMPLANTAÇÃO CONTÍNUA

- Push ⇒ CI ⇒ implantar *várias vezes por dia*
  - implantação pode feita automaticamente com o CI quando ele rodar
- Sendo assim, os lançamentos perdem o sentido?
  - continuam úteis para definir marcos vistos pelos usuários
  - crie “tag” em um *commit* específico com o nome do lançamento

    ```
    git tag 'happy-hippo' HEAD
    git push --tags
    ```
  - ou simplesmente use o *commit* ID do git para identificar o lançamento

Upgrades & Flags de funcionalidades

O PROBLEMA COM OS UPGRADES

- O que acontece se o código novo é instalado só em alguns servidores?
  - durante a atualização, alguns terão versão n, enquanto outros terão versão $n + 1$... será que isso funciona?
- O que acontece se o código novo depender de uma migração de esquema do BD?
  - a versão $n + 1$ do esquema quebra o código atual
  - código novo não funcionará com o esquema atual

O UPGRADE INGÊNUO

1. Desligue o serviço (deixe ele offline)
2. Aplique a migração destrutiva, incluindo a cópia de dados
3. Implante o novo código
4. Volte o serviço

`http://pastebin.com/5dj9k1cj`

Pode resultar em um *downtime* inaceitável

UPGRADES INCREMENTAIS COM FLAGS DE FUNCIONALIDADES

1. Faça uma migração não-destrutiva
   `http://pastebin.com/TYx5qaSB`
2. Implante o método protegido por uma flag de funcionalidade
   `http://pastebin.com/qqrLfuQh`
3. Ligue a flag de funcionalidade; se ocorrer um desastre, desligue
4. Quando todos os registros tiverem sido migrados, implante o novo código sem a flag de funcionalidade
5. Aplique uma migração para remover as colunas velhas

“DESFAZENDO” UM UPGRADE

- O desastre acontece... podemos usar uma migração *down*?
  - será que ele foi suficientemente testado?
  - a migração é reversível?
  - você tem certeza de que mais ninguém aplicou uma migração irreversível?
- Ao invés disso, use flags de funcionalidade
  - migrações *down* são principalmente para *desenvolvimento*

OUTROS USOS PARA FLAGS DE FUNCIONALIDADE

- Pré-checagem: faça o laçamento gradual da funcionalidade, aumentando aos poucos o número de usuários
  - para limitar o desempenho de problemas, por exemplo
- Testes A/B
- Funcionalidades complexas com código inserido em múltiplas implantações
- A gema `rollout` cobre esses e outros casos

Monitoramento

TIPOS DE MONITORAMENTO

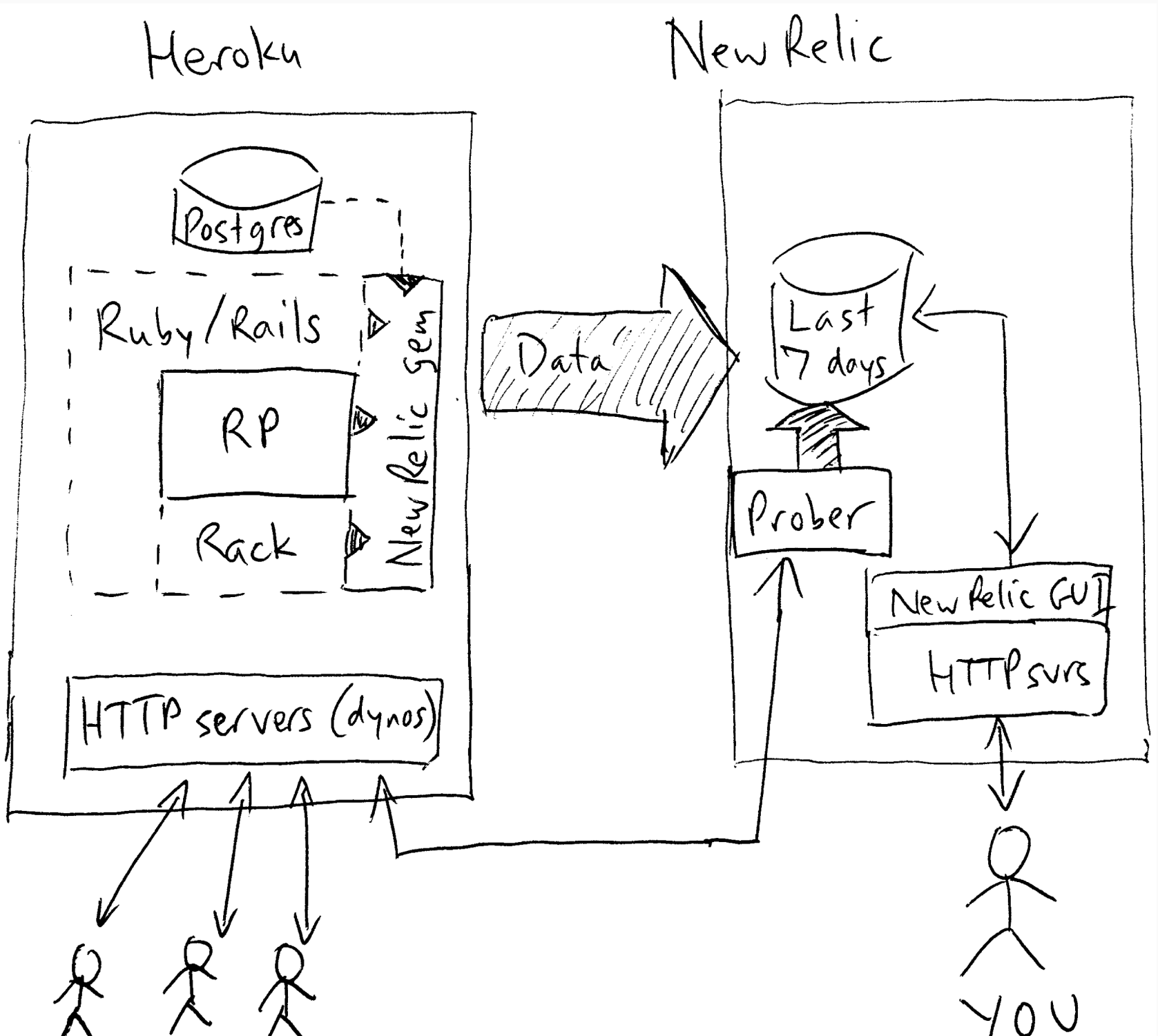
- “Se você não estiver monitorando, então provavelmente não está funcionando”
- Em tempo de desenvolvimento (*profiling*)
  - identifica possíveis problemas de desempenho/estabilidade *antes* deles irem para produção
- Em produção:
  - interno: instrumentação do código do app e/ou arcabouço (Rails, Rack, etc.)
  - externo: sondagem ativa por outro(s) site(s)

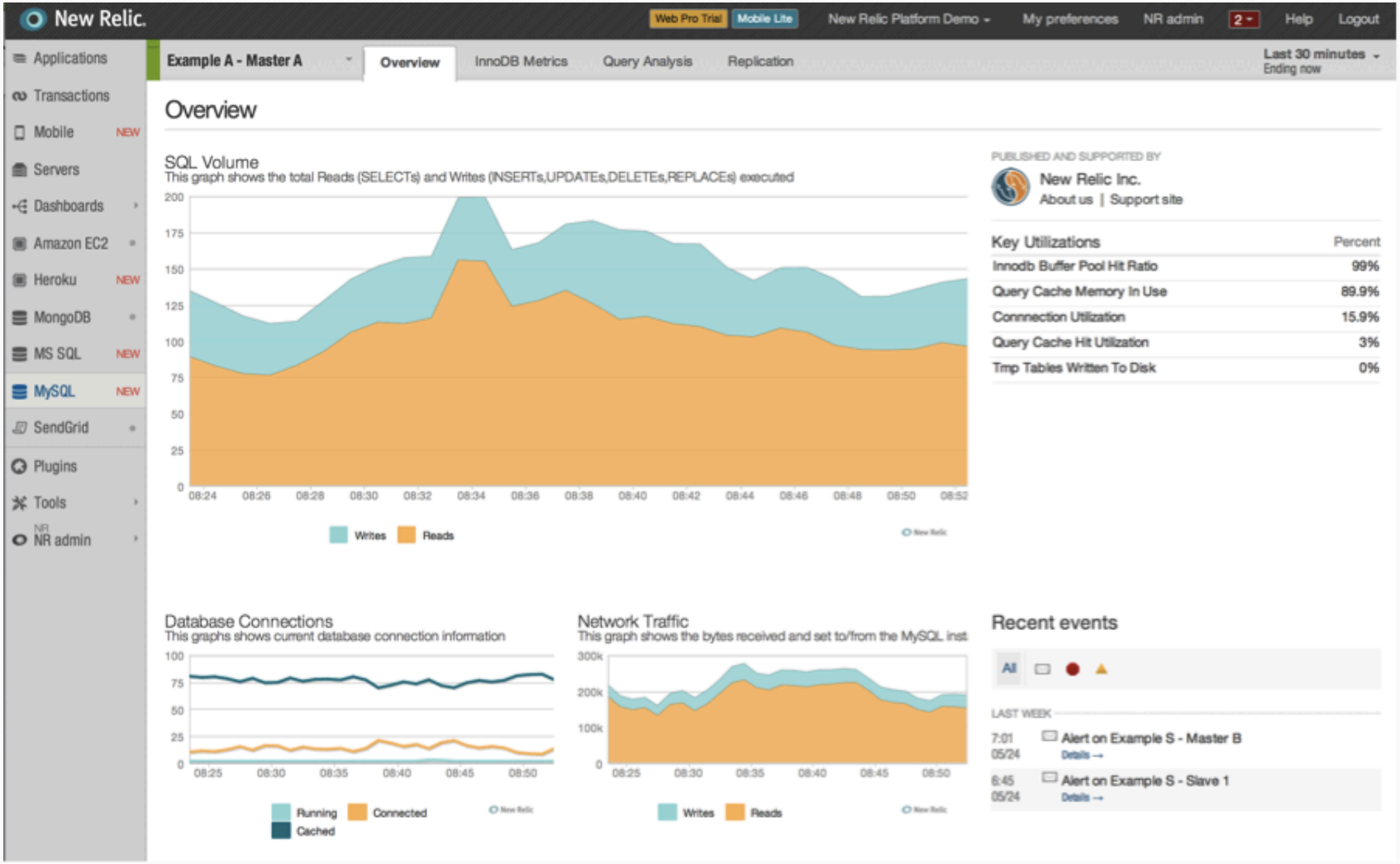
PARA QUÊ USAR MONITORAMENTO EXTERNO?

- Detectar se o site está fora do ar
- Detectar se o site está lento por motivos que não podem ser detectados pelo monitoramento interno
- Pegar pontos de vista de usuários de lugares diferentes da Internet
- Exemplo: Pingdom

MONITORAMENTO INTERNO

- pré-SaaS/PaaS: local
  - info coletada e armazenada localmente, ex: Nagios
- Hoje: hospedado
  - info coletada no seu app, mas armazenada de forma centralizada
  - info disponível mesmo quando o app está fora do ar
- Exemplo: New Relic
  - tem um modo de desenvolvimento e um modo de produção
  - nível básico de serviço é gratuito para apps no Heroku


Heroku
New Relic
Postgres
Ruby/Rails
RP
Rack
New Relic gem
Data
Last 7 days
Prober
New Relic GUI
HTTP svrs
HTTP servers (dynos)
YOU


New Relic.
Web Pro Trial
Mobile Lite
New Relic Platform Demo
My preferences
NR admin
2
Help
Logout
Applications
Transactions
Mobile NEW
Servers
Dashboards
Amazon EC2
Heroku NEW
MongoDB
MS SQL NEW
MySQL NEW
SendGrid
Plugins
Tools
NR admin
Example A - Master A
Overview
InnoDB Metrics
Query Analysis
Replication
Last 30 minutes
Ending now
Overview
SQL Volume
This graph shows the total Reads (SELECTs) and Writes (INSERTs,UPDATEs,DELETEs,REPLACEs) executed
200
175
150
125
100
75
50
25
0
08:24
08:26
08:28
08:30
08:32
08:34
08:36
08:38
08:40
08:42
08:44
08:46
08:48
08:50
Writes
Reads
PUBLISHED AND SUPPORTED BY
New Relic Inc.
About us | Support site
Key Utilizations
Percent
Innodb Buffer Pool Hit Ratio
99%
Query Cache Memory In Use
89.9%
Connnection Utilization
15.9%
Query Cache Hit Utilization
3%
Tmp Tables Written To Disk
0%
Database Connections
This graphs shows current database connection information
100
75
50
25
0
08:25
08:30
08:35
08:40
08:45
08:50
Running
Connected
Cached
Network Traffic
This graph shows the bytes received and set to/from the MySQL inst
300k
200k
100k
0
08:25
08:30
08:35
08:40
08:45
08:50
Writes
Reads
Recent events
All
LAST WEEK
7:01
05/24
Alert on Example S - Master B
Details →
6:45
05/24
Alert on Example S - Slave 1
Details →

ALGUMAS FERRAMENTAS DE MONITORAMENTO

O que é monitorado?	Nível	Ferramenta	Hospedada
Disponibilidade	site	pingdom.com	Sim
Exceções não capturadas	site	airbrake.io	Sim
Ações de controladores lentas ou consultas ao BD	app	newrelic.com	Sim
cliques, tempo entre ações	app	Google Analytics	Sim
Saúde do processo & telemetria (MySQL, servidor Apache, etc.	processo	god, monit, nagios	Não

O QUE MEDIR?

- Testes de estresse ou teste de carga: até que ponto meu sistema aguenta antes que...
  - ... o desempenho de torne inaceitável?
  - ... ele comece a engasgar e morra?
- Normalmente, um componente será um gargalo
  - uma visão em particular, ação, consulta, ...
- Testadores de carga podem ser simples ou sofisticados
  - acessam uma única URI repetidas vezes
  - acessam uma sequência determinada de URIs repetidas vezes
  - repetem um arquivo de log

BUGS LONGEVOS

- Vazamento de recurso (RAM, descritores de arquivos, tabelas de sessão) são exemplos clássicos
- Algumas infraestruturas de software tais como o Apache já fazem algum *rejuvenescimento*
  - aka: reiniciam ao longo do tempo
- Relacionado: ficar sem sessões
  - solução: guarde todo o `session[]` em cookie (Rails ≥ 3 faz isso por padrão)

Defendendo os dados dos clientes

ATAQUES COMUNS AOS APPS

1. Espionagem & SSL
2. Homem no meio / Sequestro de sessão
3. Injeção de SQL
4. Falsificação de requisição cross-site (CSRF)
5. Cross-site scripting (XSS)
6. Atribuição maciça de atributos sensíveis
7. ... mais no livro e em `https://guides.rubyonrails.org/security.html`

SSL (SECURE SOCKETS LAYER)

- Ideia: *encriptar* tráfego HTTP para frustrar espiões
- Problema: para criar um *canal seguro*, as duas partes precisam compartilhar um segredo primeiro
- Mas na web, as duas partes nem se conhecem
- Solução: *criptografia de chave pública* (Rivest, Shamir, & Adelman, Prêmio Turing em 2002)

O QUE O SSL FAZ E O QUE ELE NÃO FAZ

- Cada participante tem uma chave composta de 2 partes:
  - parte pública: todos podem conhecer
  - parte privada: participante mantém em segredo
  - dada uma parte, não é possível deduzir a outra
- Mecanismo: dados *criptografados* com uma parte só podem ser *decriptografados* com a outra
  - se uma mensagem pode ser decriptografada com a chave pública de Bob, então Bob necessariamente a criptografou usando sua chave privada
  - se eu usar a chave pública de Bob para criar uma mensagem, só ele poderá lê-la

COMO SSL FUNCIONA (SIMPLIFICADAMENTE)

1. `bob.com` prova sua identidade a uma Autoridade Certificadora (CA)
2. CA usa sua chave privada para criar um "certificado" que amarra essa identidade ao domínio "bob.com"
3. Certificado é instalado no servidor `bob.com`
4. Navegador visita `https://bob.com`
5. As chaves públicas da CA estão embutidas no navegador, que pode verificar se o certificado casa com o domínio
6. Uma troca de chaves usando o algoritmo *Diffie-Hellman* é usada para iniciar um canal criptografado para as comunicações futuras

Use o método `force_ssl` do Rails para forçar algumas ações a usar SSL

O QUE SSL FAZ E O QUE NÃO FAZ

- ✔ Garante ao navegador que `bob.com` é legítimo
- ✔ Previne espiões de ler (ou corromper) o tráfego entre o navegador e `bob.com`
- ✔ Cria trabalho computacional adicional para o servidor

O que ele não faz:

- ✘ Garante ao servidor quem é o usuário
- ✘ Garante algo sobre o que é feito com os dados depois que ele chega ao servidor
- ✘ Garante algo sobre outras vulnerabilidades do servidor
- ✘ Protege o navegador contra malware se o servidor for do mal

- View: `= text_field_tag 'name'`
- App: `Moviegoer.where("name='#params[:name]'")`
- Usuário malfeitor preenche o campo com:
  `BOB'); DROP TABLE moviegoers; --`
- `SELECT * FROM moviegoers WHERE (name='BOB');`
  `DROP TABLE moviegoers; --'`
- Solução: `Moviegoer.where("name=?", params[:name])`

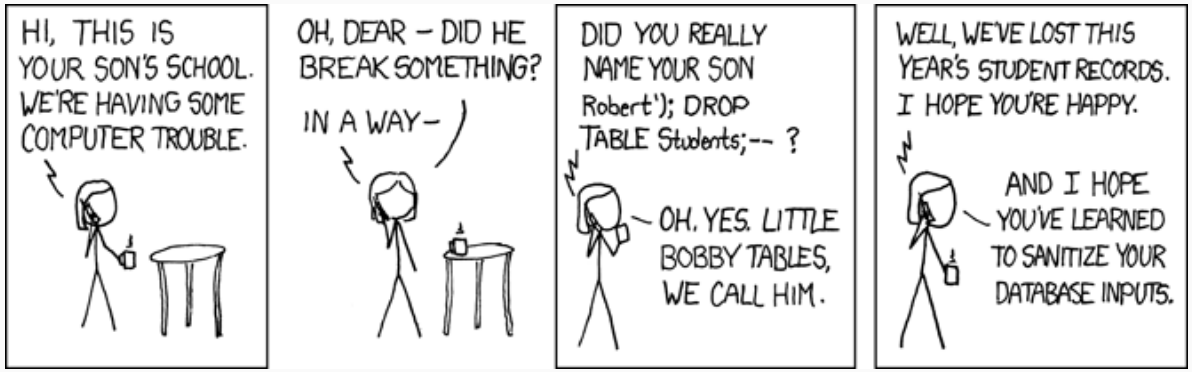


Figura 4: Exploits of a Mom: `https://xkcd.com/327/`

FALSIFICAÇÃO DE REQUISIÇÃO CROSS-SITE

1. Alice se loga em `banco.com` e agora tem um *cookie*
2. Alice vai em `blog.malfeitor.com`
3. A página contém a tag
   `<img src="http://banco.com/extrato_mensal"/>`
4. `malfeitor.com` recolhe informações sobre a conta de Alice

Soluções

- (fraco) verificar o campo `Referer` no cabeçalho HTTP
- (forte) incluir `session nonce` em cada requisição
  - `csrf_meta_tags` em `layouts/application.html.haml`
  - `protect_from_forgery` em `ApplicationController`
  - os auxiliares para criação de formulário automaticamente incluem nonce
- Mais infos em `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`

Perspectiva Planeje-e-Documente sobre Desempenho, Lançamentos, Robustez e Segurança

P-E-D E DESEMPENHO

- Assim como robustez e segurança, desempenho é considerado um requisito não-funcional
  - pode ser parte dos testes de aceitação
- Os ciclos de vida Planeje-e-Documente ignoram desempenho porque:
  - otimizações de desempenho são consideradas desculpas para má prática de EngSoft
  - é um problema coberto em outros cursos/livros

GERENCIAMENTO DE LANÇAMENTOS EM P-E-D

- Caso especial do gerenciamento de configurações
- Lançamentos em P-e-D incluem tudo: código, arquivos de configuração, dados e documentação
- Esquema de enumeração dos lançamentos em P-e-D; ex: Rails versão 3.2.12
  - .12 representa o número de lançamentos menos importantes (*minor release*)
  - .2 representa um lançamento mais importante (*major release*)
  - 3 representa um novo lançamento com mudanças tão drásticas que poderiam quebrar a API do app

P-E-D E ROBUSTEZ

- Confiança via redundância
  - Diretriz: não permita pontos únicos de falha
- Por quanta redundância o cliente pode pagar?
- Tempo médio entre falhas (*Mean Time To Failure* ou MTTF)
  - inclui software e seus operadores, assim como hardware
- Indisponibilidade $\approx$ tempo médio entre reparos (MTTR)
  - melhorar o MTTR pode ser mais fácil do que melhorar o MTTF, mas podemos tentar melhorar ambos

- P-e-D assume que o processo de desenvolvimento de software em uma organização pode ser melhorado
  - $\Rightarrow$ produto de software mais confiável
  - registre todos os aspectos do projeto para ver o que pode ser melhorado
- Obtenha o padrão ABNT NBR ISO 9001[3] se a empresa usar:
  - um processo
  - um método para ver se o processo é seguido
  - um registro dos resultados para melhorar o processo
- Aprova o *processo*, não a qualidade do código resultante

[3] Gestão da qualidade e garantia da qualidade

- Confiabilidade depende de probabilidades, mas segurança nos defende contra oponentes inteligentes
  - *Common Vulnerabilities and Exposures* (CVE) lista os ataques mais comuns. Veja `http://www.cert.br/` e `http://www.cvedetails.com/`.
- Algumas técnicas melhoram a confiabilidade contra prevenção de ataques:
  - estouro de buffer (*buffer overflow*), estouro aritmético (*arithmetic overflow*), condições de corridas
- Testes de penetração realizados por um time de especialistas em segurança (*tiger team*) podem testar a segurança

3 princípios de segurança

1. Menor privilégio um usuário ou componente do software não deve ter mais privilégios — isso é, acesso à informação ou recursos — do que o necessário para realizar a sua tarefa
    - princípio “need-to-know” de documentos sigilosos
2. Padrões à prova de falha a não ser que alguém explicitamente dê permissão para um usuário ou componente do software acessar um objeto, o acesso a esse objeto deveria ser negado
    - o padrão deve ser negar o acesso
3. Aceitação psicológica o mecanismo de proteção não deve tornar o app mais difícil de usar se comparado a um app que não use proteção
    - precisa ser fácil de usar para que os mecanismos de segurança sejam seguidos de forma rotineira

Falácias, armadilhas e comentários finais

ARMADILHA: OTIMIZAR PREMATURAMENTE SEM MEDIR

- Velocidade é uma funcionalidade que os usuários esperam ter
  - otimizar para o 99º percentil, não para a “média”
- Escalabilidade horizontal » desempenho por máquina, mas muitas coisas podem ficar lentas
- Monitoramento é o seu amigo: meça duas vezes, corte uma

FALÁCIA: ”MEU SOFTWARE É UM APP EM 3 CAMADAS, PORTANTO ELE É ESCALÁVEL”

- É difícil obter escalabilidade de bancos de dados
  - mesmo que consiga, você quer que as operações “caras” fiquem longe do seu SLO
- Dica: faça cache em vários níveis
  - cache de página inteiro, de fragmentos, de consultas
  - invalidação de cache é uma preocupação transversal
  - As funcionalidades de RoR para *crosscutting concerns* permitem que você as especifique declarativamente
- Use PaaS por tanto tempo quanto for possível

FALÁCIA: ”NINGUÉM VAI ATACAR MEU SITE PEQUENO”

- Os hackers podem estar atrás dos seus usuários, não dos seus dados
- Assim como com desempenho, segurança é uma preocupação transversal — difícil de adicionar depois do problema acontecer
- Fique atualizado com as melhores práticas e ferramentas — dificilmente você vai conseguir fazer melhor sozinho
- Prepare-se para catástrofes: faça cópias de segurança do site e banco de dados regularmente