Webアプリの
デプロイ今昔物語

PHP Conference Japan 2014　| 2014-10-11

https://joind.in/12041

HowTelevision

秋山顕治 (Kenji Akiyama)

株式会社ハウテレビジョン
エンジニア

@artifactsauce
@artifactsauce
/+KenjiAkiyama

気がつけば38歳

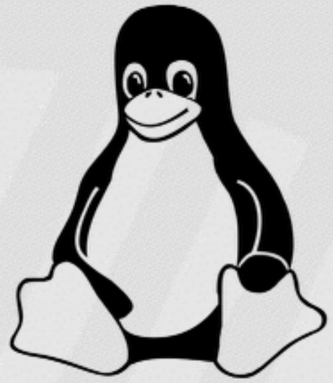
HowTelevision

外資就活ドットコム

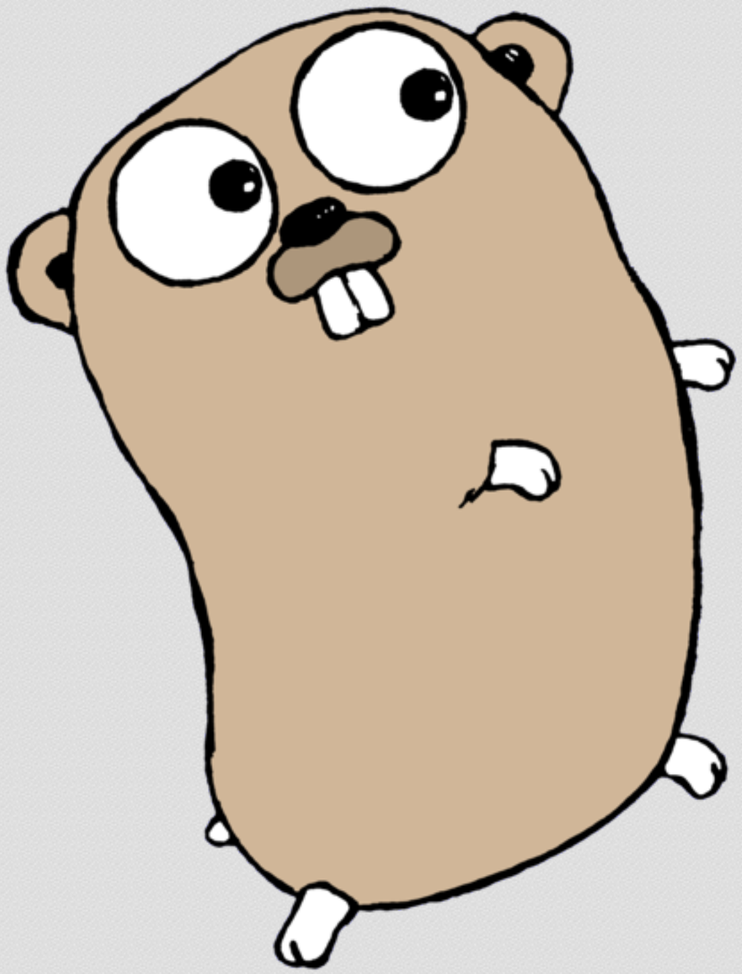
- 新卒向け求人メディア
- 2010年リリース
- 東大／慶応大の就活生の半数が会員登録している
- WordPressとCakePHPでできている

HowTelevision

スキルセット



最近は




Pearl


Pearl


仕事もプライベートも
地味な役割が大好き

以前のキャリア

理◯学研究所に13年ほど

- バイオインフォマティクス
  - 生物データの解析
  - データベース構築
  - Webアプリ開発

今回の趣旨

- 主に中小規模Webアプリを開発してきた人の視点で、Webアプリのデプロイにまつわる歴史を振り返りながら、なぜ今、このような形になっているのか？これからどのようになるのか？を考える。
- 自宅サーバー派やデータセンター利用派などはまた別の歴史を歩んでいますので、今回の視点からはズレてくると思います。

注意

- おおよその時代の流れをつかむことが目的であり、厳密なものではありません。
- 細かいことは気にしないで、ザァーっと聞いてください。

対象としている人

- Webアプリ開発の初心者
- デプロイツールを使い始めたばかりの人
- デプロイツールを使っているけど、使い始める前のことを知らない人

ソフトウェアデプロイメント（英: Software deployment）とは、ソフトウェアシステムを利用可能にする活動全般を指す用語である。デプロイメント（Deployment）とは「展開、配備、配置」などの意。

–Wikipedia

Webアプリのデプロイの歴史

- レンサバ時代
- VPS時代
- PaaS時代
- IaaS時代

レンサバ

- レンタルサーバーの略
- ホスティングサーバーまたは共用サーバーとも呼ばれる
- 複数の利用者にディスクスペースを貸し出すサービス
- リソース（CPU/Mem.など）は利用者で共有する
- CGI/SSIが動かせる（ものもある）

レンサバ時代

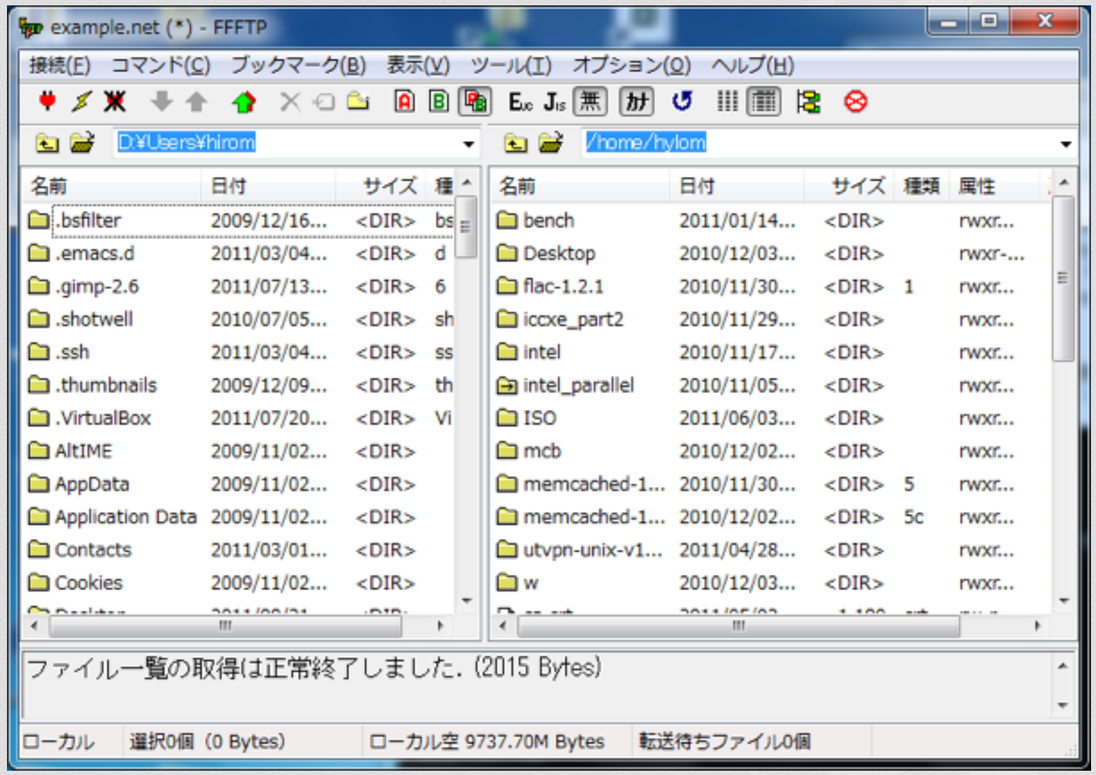
- 1990年台後半から2000年初頭か？
- レンタルサーバーくらいしか選択肢が無かった時代（技術的にも金銭的にも）
- 中小規模のサービスには占有サーバーなんて手が出ない。

この時代

- Perl 5.6.0（2000年）
  - CGI::Application 1.0（2000年）
- PHP 4.0（2000年）
- Windows 2000（2000年）
- Subversion（2000年）

レンサバにおけるデプロイ

- 主流はWindowsクライアントからのFTPアップロード
- サーバー設定に手が出せないため、通信手段の選択肢がFTP以外に無い。



レンサバにおけるデプロイの問題点

- FTPなのでセキュリティが甘い
  - 暗号化されていないパスワードがネットワークに流れる。
- 共通アカウントでコンテンツ管理
  - アカウントがわかれば何でもできる。

アプリの肥大化

- 肥大化するWebアプリの処理速度を改善しようという試みが進む。
  - mod_perl（1996年）
  - FastCGI（1996年）

アプリの肥大化

- 肥大化するWebアプリの処理速度を改善しようという試みが進む。
  - mod_perl（1996年）
  - FastCGI（1996年）

レンサバではこれらの技術を利用できない

VPS時代

- Virtual Private Server
  - 物理コンピューター上で仮想的にコンピューターを複数起動する技術を用いて作られた仮想的なコンピューター
- VMWare (1999) / FreeBSD jail (2000) / Virtuozzo (2001) / Xen (2003) / Windows Server (2003) / Solaris Container (2004) / KVM (2007)

この時代

- 2000年初頭から2000年台前半
- デスクトップLinuxが使えるものになってきた。
- Perl 5.8.0（2002年）
- PHP 5.0（2004年）
- Ruby on Rails 初公開（2004年）

VPSのメリット

- リソース（CPU/Mem./HDDなど）以外は自由にできる。
  - ミドルウェアやプログラミング言語など
  - アカウント管理
  - 通信方法もSSHを使えるようになった
    - OpenSSHの普及は2000年から

VPSにおけるデプロイ

- Linuxクライアントからのrsync
  - コマンドラインからSSHを通してアクセスする
  - リモートサーバーとファイル同期をとる

すべてがうまくいった

もう悩みなんて無い

すべてがうまくいった

そんなわけない

もう悩みなんて無い

アプリの構成の複雑化

- プロセスの永続化
- 静的ファイルだけは別サーバー
- Javascriptの難読化／最小化
- キャッシュの保持

複雑化への対処

- スクリプトを書いたｗ
- 秘伝のタレが完成

デプロイツールの誕生

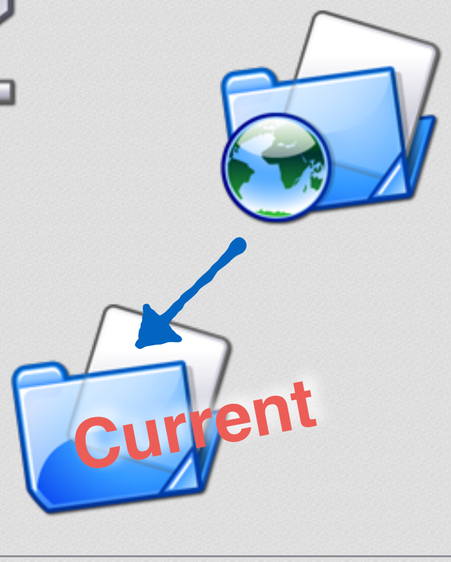
- Capistrano
  - 2006年にSwitchTowerから改名
  - 開発当初はShell Script
  - Rubyで書き直される
  - 現在のバージョンは3.2

Capistrano

- 複数のサーバー上でスクリプトを実行するためのオープンソースのツール
- 1つ以上のWebサーバ上のアプリケーションを新しいバージョンにする作業を自動化

Capistrano deploy

App Server



Git Server

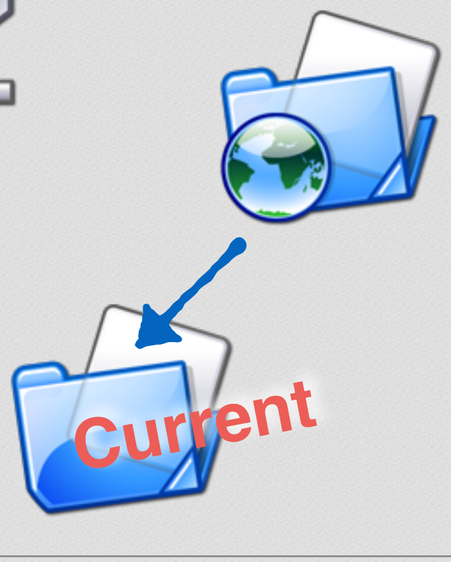
```
void
usage (char *name)
{
  printf ("usage:\n");
  printf ("%s -a [-c file]",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-s file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweld] [-z size] ");
#endif
```

デフォルトで用意されているdeployタスク

HowTelevision

Capistrano deploy

App Server



Git Server

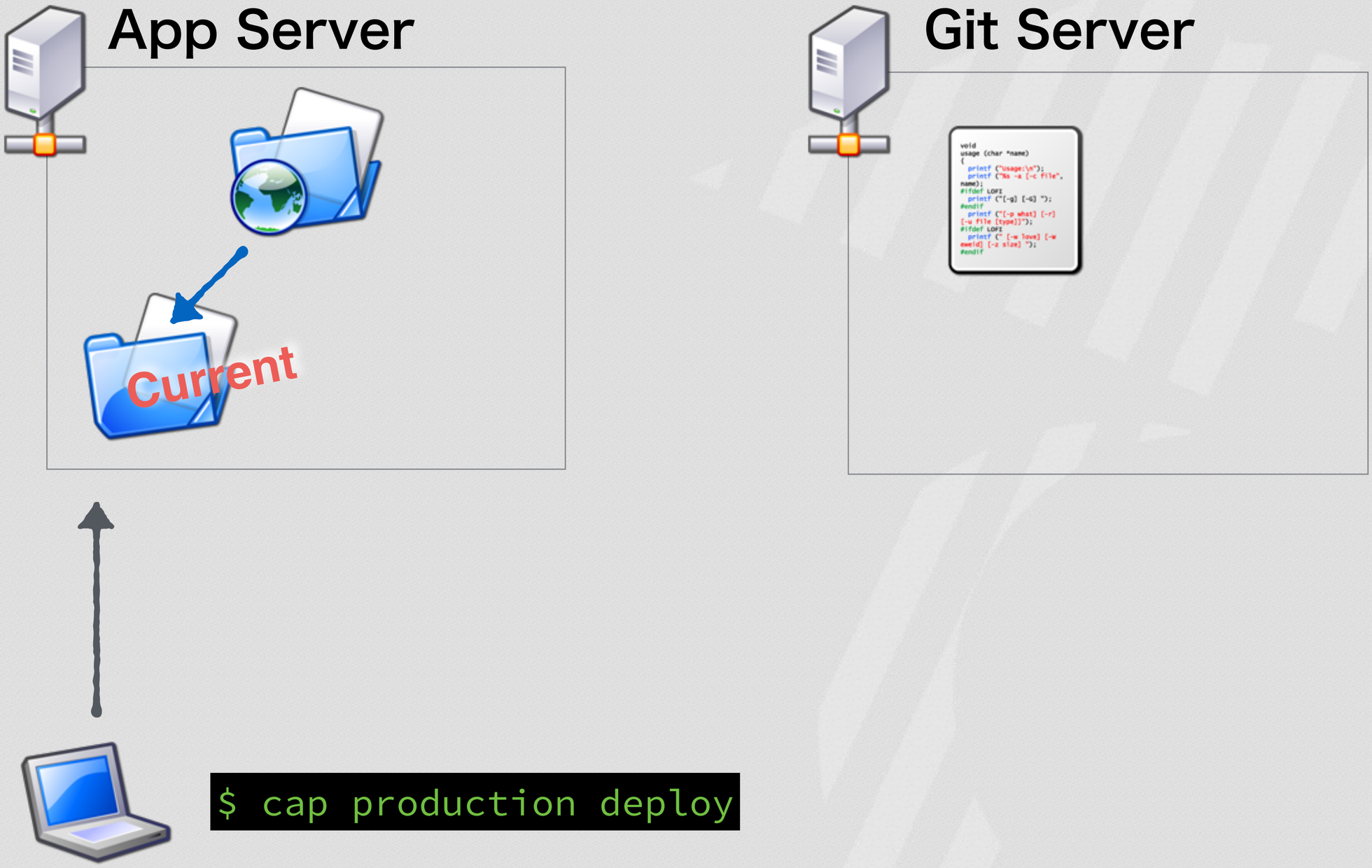
```
void
usage (char *name)
{
  printf ("usage:\n");
  printf ("%s -a [-c file]",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-s file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweld] [-z size] ");
#endif
```

```
$ cap production deploy
```

デフォルトで用意されているdeployタスク

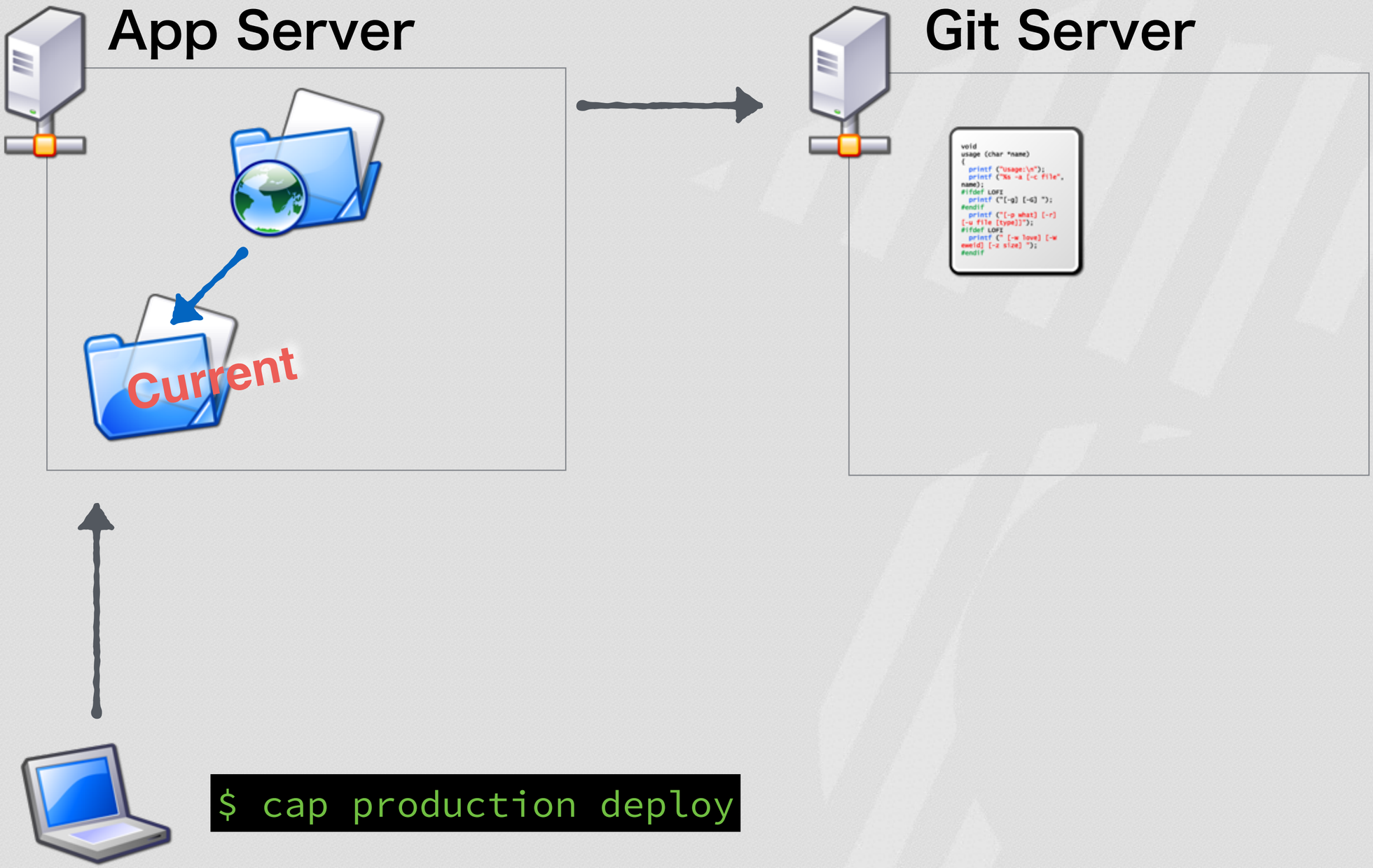
HowTelevision

Capistrano deploy



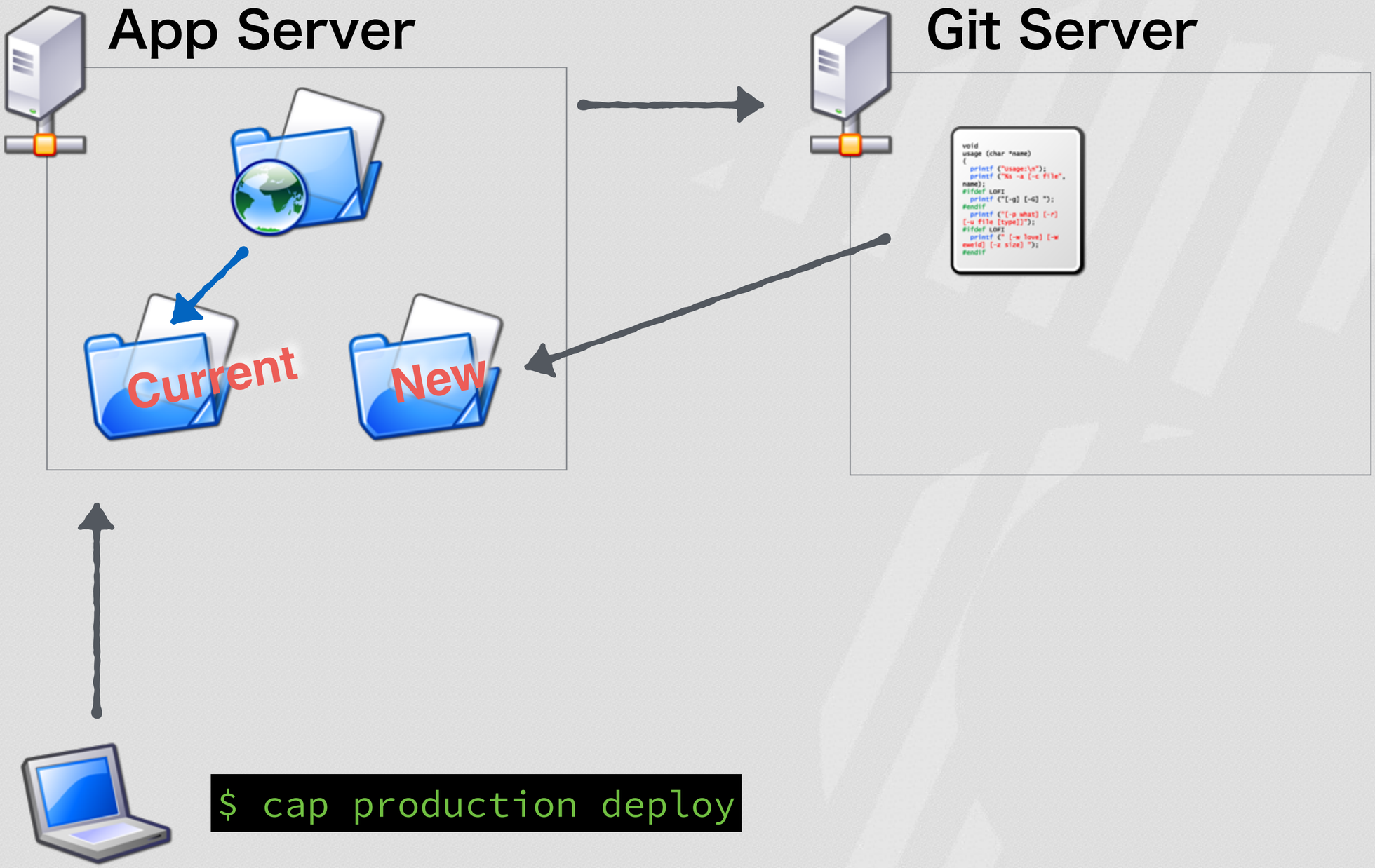
デフォルトで用意されているdeployタスク

Capistrano deploy



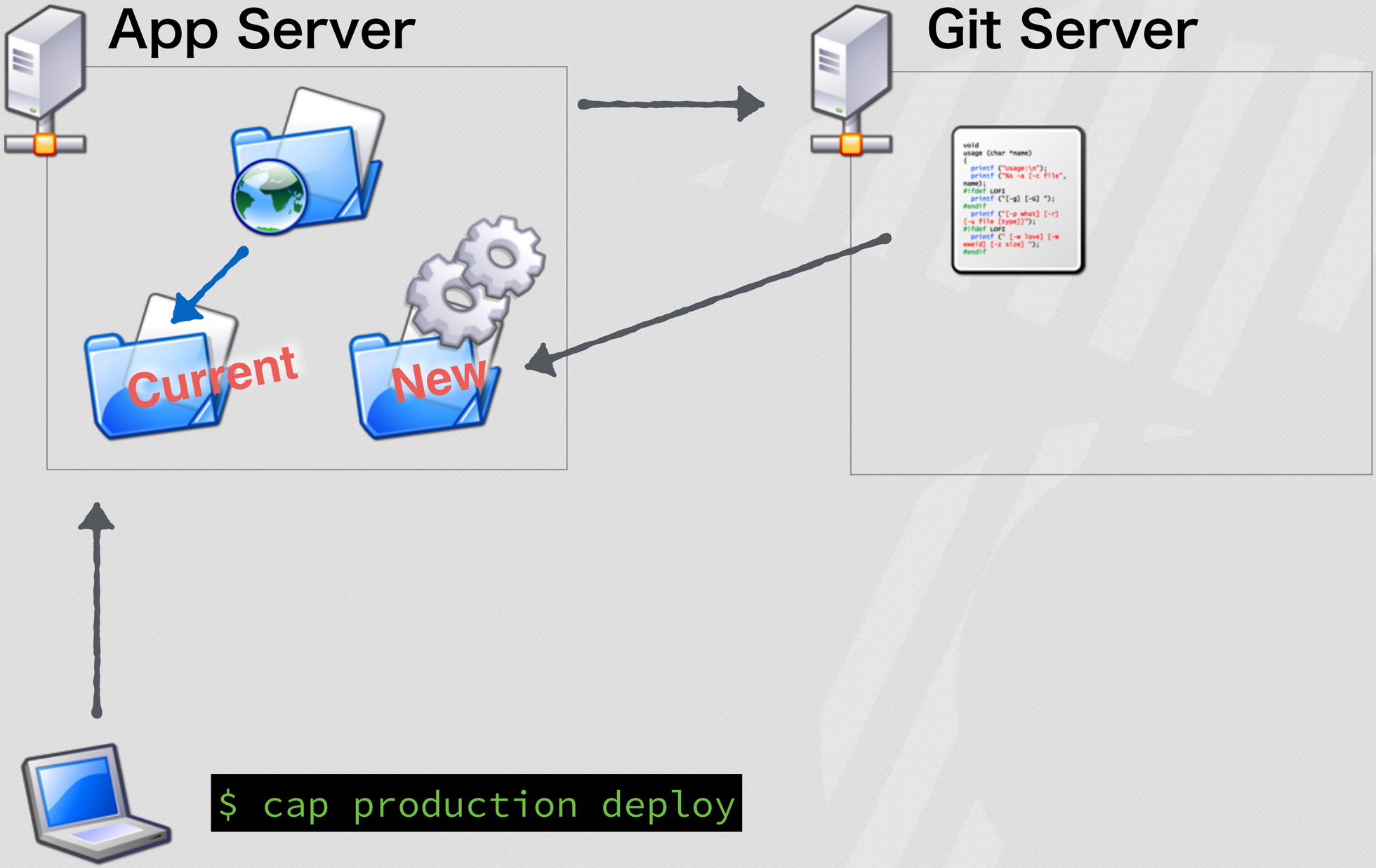
デフォルトで用意されているdeployタスク

Capistrano deploy



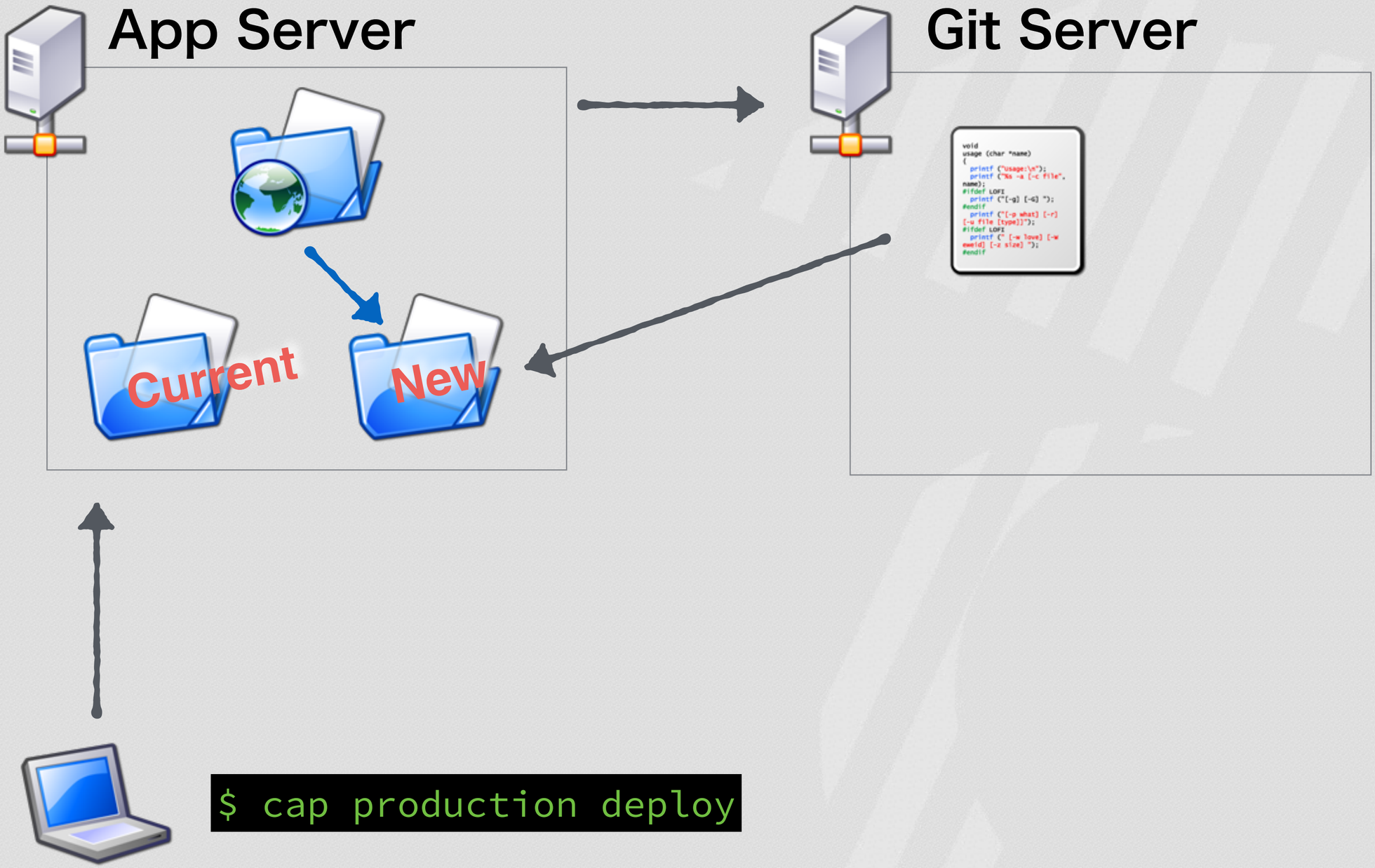
デフォルトで用意されているdeployタスク

Capistrano deploy



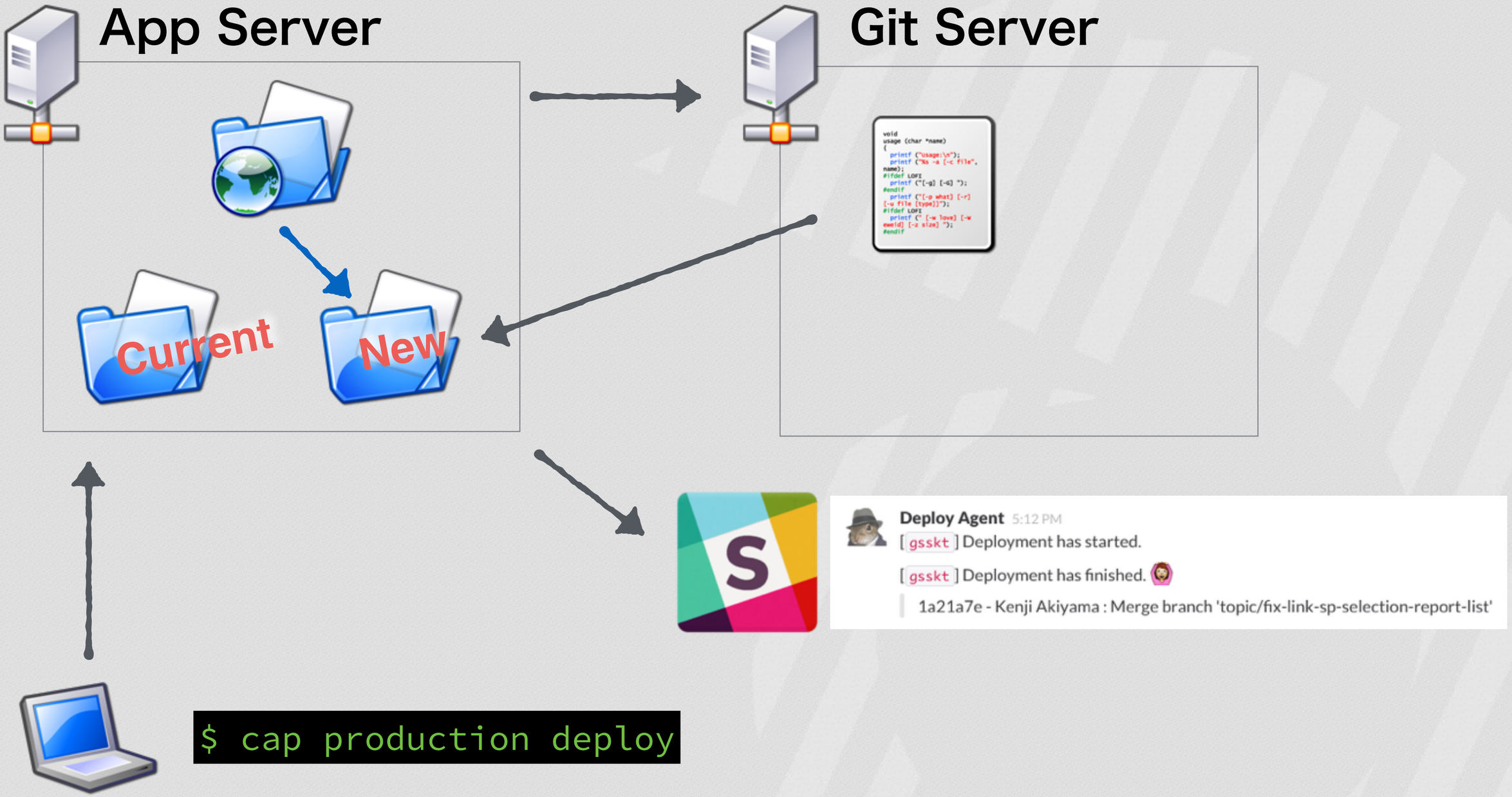
デフォルトで用意されているdeployタスク

Capistrano deploy



デフォルトで用意されているdeployタスク

Capistrano deploy



デフォルトで用意されているdeployタスク

HowTelevision

deploy task

- deploy:starting
  - deploy:check
    - git:check
    - deploy:check:directories
    - deploy:check:linked_dirs
    - deploy:check:make_linked_dirs
    - deploy:check:linked_files
- deploy:started
- deploy:updating
  - git:create_release
  - deploy:symlink:shared
    - deploy:symlink:linked_files
    - deploy:symlink:linked_dirs
- deploy:updated
- deploy:publishing
  - deploy:symlink:release
- deploy:published
- deploy:finishing
  - deploy:cleanup
- deploy:finished

過去形で記されているタスクは空のタスクなのでこれらを必要に応じて上書きする

Capistrano3のデプロイフレームワークの使い方

HowTelevision

基本方針

- デフォルトに寄せてみる

かつてのデプロイ手順

1. Gitの開発ブランチから `master` ブランチへマージする
2. 作業の開始をSlackで全員に通知する
3. 前回のデプロイから変更のあったファイルだけをサーバーにコピーする
4. 現行バージョンのディレクトリを新たなバージョン名を付けてコピーする
5. サーバーにコピーしたファイルで新バージョンディレクトリのファイルを上書きする
6. 現行バージョンのディレクトリを指すSymlinkを、新バージョンディレクトリを指すように作成し直す
7. Cacheファイルを削除する
8. php-fpmを再起動する
9. 作業の終了をSlackで全員に通知する

かつてのデプロイ手順

```
# local

$ git checkout master
$ git pull
$ git merge DEV_BRANCH
$ PAGER=cat git diff --name-status LAST_TAG CRRT_TAG \
> > file_diff.txt
$ grep    '^D' file_diff.txt > file_del.txt
$ grep -v '^D' file_diff.txt > file_mod.txt
$ grep '\w\s*wordpress' file_diff.txt > file_wdp.txt
$ perl -i -ple '/\.scss$/ \
> && s/(sass|scss$)/css/g' file_mod.txt
$ bundle exec compass compile
$ mkdir -p DEST_DIR
$ perl -ple 's/^\w\s+//' file_mod.txt \
> | xargs -I% rsync -R % DEST_DIR
$ cp file_del.txt DEST_DIR
$ scp -r DEST_DIR REMOTE_HOST:~/DEST_DIR

# remote

$ cd DEPLOY_TO
$ sudo -u DEPLOY_USER cp -apr LAST_TAG CRRT_TAG
$ sudo -u DEPLOY_USER rsync -av DEST_DIR DEPLOY_TO/ \
> CRRT_TAG
$ cd DEPLOY_TO/CRRT_TAG
$ perl -ple 's/^D\s+//' file_del.txt | xargs rm
$ sudo -u DEPLOY_USER ln -snf DEPLOY_TO/CRRT_TAG WEB_ROOT
$ sudo find CAKE1_APP_CACHE -type f -exec rm {} \;
$ sudo find CAKE2_APP_CACHE -type f -name 'empty' -prune \
> -o -type f -exec rm {} \;
$ sudo service php5-fpm restart
```

自動化の恩恵

```
# local

$ git checkout master
$ git pull
$ git merge DEV_BRANCH
$ PAGER=cat git diff --name-status LAST_TAG CRRT_TAG \
> > file_diff.txt
$ grep    '^D' file_diff.txt > file_del.txt
$ grep -v '^D' file_diff.txt > file_mod.txt
$ grep '\w\s*wordpress' file_diff.txt > file_wdp.txt
$ perl -i -ple '/\.scss$/ \
> && s/(sass|scss$)/css/g' file_mod.txt
$ bundle exec compass compile
$ mkdir -p DEST_DIR
$ perl -ple 's/^\w\s+//' file_mod.txt \
> | xargs -I% rsync -R % DEST_DIR
$ cp file_del.txt DEST_DIR
$ scp -r DEST_DIR REMOTE_HOST:~/DEST_DIR

# remote

$ cd DEPLOY_TO
$ sudo -u DEPLOY_USER cp -apr LAST_TAG CRRT_TAG
$ sudo -u DEPLOY_USER rsync -av DEST_DIR DEPLOY_TO/ \
> CRRT_TAG
$ cd DEPLOY_TO/CRRT_TAG
$ perl -ple 's/^D\s+//' file_del.txt | xargs rm
$ sudo -u DEPLOY_USER ln -snf DEPLOY_TO/CRRT_TAG WEB_ROOT
$ sudo find CAKE1_APP_CACHE -type f -exec rm {} \;
$ sudo find CAKE2_APP_CACHE -type f -name 'empty' -prune \
> -o -type f -exec rm {} \;
$ sudo service php5-fpm restart
```

自動化の恩恵

```
# local

$ git checkout master
$ git pull
$ git merge DEV_BRANCH
$ PAGER=cat git diff --name-status LAST_TAG CRRT_TAG \
> > file_diff.txt
$ grep    '^D' file_diff.txt > file_del.txt
$ grep -v '^D' file_diff.txt > file_mod.txt
$ grep '\w\s*wordpress' file_diff.txt > file_wdp.txt
$ perl -i -ple '/\.scss$/ \
> && s/(sass|scss$)/css/g’ file_mod.txt
$ bundle exec compass compile
$ mkdir -p DEST_DIR
$ perl -ple 's/^\w\s+//' file_mod.txt \
> | xargs -I% rsync -R % DEST_DIR
$ cp file_del.txt DEST_DIR
$ scp -r DEST_DIR REMOTE_HOST:~/DEST_DIR

# remote

$ cd DEPLOY_TO
$ sudo -u DEPLOY_USER cp -apr LAST_TAG CRRT_TAG
$ sudo -u DEPLOY_USER rsync -av DEST_DIR DEPLOY_TO/ \
> CRRT_TAG
$ cd DEPLOY_TO/CRRT_TAG
$ perl -ple 's/^D\s+//' file_del.txt | xargs rm
$ sudo -u DEPLOY_USER ln -snf DEPLOY_TO/CRRT_TAG WEB_ROOT
$ sudo find CAKE1_APP_CACHE -type f -exec rm {} \;
$ sudo find CAKE2_APP_CACHE -type f -name 'empty' -prune \
> -o -type f -exec rm {} \;
$ sudo service php5-fpm restart
```

```
# local

$ cap production deploy
```

自動化の恩恵

```
# local

$ git checkout master
$ git pull
$ git merge DEV_BRANCH
$ PAGER=cat git diff --name-status LAST_TAG CRRT_TAG \
> > file_diff.txt
$ grep    '^D' file_diff.txt > file_del.txt
$ grep -v '^D' file_diff.txt > file_mod.txt
$ grep '\w\s*wordpress' file_diff.txt > file_wdp.txt
$ perl -i -ple '/\.scss$/ \
> && s/(sass|scss$)/css/g' file_mod.txt
$ bundle exec compass compile
$ mkdir -p DEST_DIR
$ perl -ple 's/^\w\s+//' file_mod.txt \
> | xargs -I% rsync -R % DEST_DIR
$ cp file_del.txt DEST_DIR
$ scp -r DEST_DIR REMOTE_HOST:~/DEST_DIR

# remote

$ cd DEPLOY_TO
$ sudo -u DEPLOY_USER cp -apr LAST_TAG CRRT_TAG
$ sudo -u DEPLOY_USER rsync -av DEST_DIR DEPLOY_TO/ \
> CRRT_TAG
$ cd DEPLOY_TO/CRRT_TAG
$ perl -ple 's/^D\s+//' file_del.txt | xargs rm
$ sudo -u DEPLOY_USER ln -snf DEPLOY_TO/CRRT_TAG WEB_ROOT
$ sudo find CAKE1_APP_CACHE -type f -exec rm {} \;
$ sudo find CAKE2_APP_CACHE -type f -name 'empty' -prune \
> -o -type f -exec rm {} \;
$ sudo service php5-fpm restart
```

```
# local

$ cap production deploy
```

taken by wiedmaier

HowTelevision

自動化の恩恵

１時間

taken by hipponotized

自動化の恩恵

1時間

taken by hipponotized

5分

taken by photons

HowTelevision

自動化の恩恵

1時間

taken by hipponotized

12倍

5分

taken by photons

HowTelevision

どんなコード？

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake_dir}/#{part_dir}"
        end
      end
    end

    within "#{fetch(:release_path)}/cakephp2/app" do
      composer_path = "#{fetch(:release_path)}/cakephp2/app/composer.phar"
      if test "[ -f #{composer_path} ]" then
        execute "./composer.phar", 'install'
      end

      if fetch(:run_migration) then
        execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
      end
    end

    if fetch(:sass_compile_path) then
      within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
        execute :compass, 'compile', '--boring'
      end
    end

  end
end
```

どんなコード？

CakePHPライブラリへ
リンクを張る

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake_dir}/#{part_dir}"
        end
      end
    end

    within "#{fetch(:release_path)}/cakephp2/app" do
      composer_path = "#{fetch(:release_path)}/cakephp2/app/composer.phar"
      if test "[ -f #{composer_path} ]" then
        execute "./composer.phar", 'install'
      end

      if fetch(:run_migration) then
        execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
      end
    end

    if fetch(:sass_compile_path) then
      within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
        execute :compass, 'compile', '--boring'
      end
    end

  end
end
```

どんなコード？

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake
        end
      end
    end

    within "#{fetch(:release_path)}/cakephp2/app" do
      composer_path = "#{fetch(:release_path)}/cakephp2/app/composer.phar
      if test "[ -f #{composer_path} ]" then
        execute "./composer.phar", 'install'
      end

      if fetch(:run_migration) then
        execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
      end
    end

    if fetch(:sass_compile_path) then
      within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
        execute :compass, 'compile', '--boring'
      end
    end

  end
end
```

CakePHPライブラリへ
リンクを張る

composerでプラグイン
をインストールする

どんなコード？

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake
        end
      end

      if fetch(:composer_path) then
        within "#{fetch(:release_path)}/cakephp2/app" do
          execute "#{fetch(:release_path)}/cakephp2/app/composer.phar
          #{composer_path} ]" then
          composer.phar", 'install'
        end
      end

      if fetch(:run_migration) then
        execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
      end
    end

    if fetch(:sass_compile_path) then
      within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
        execute :compass, 'compile', '--boring'
      end
    end

  end
end
```

CakePHPライブラリへ
リンクを張る

データベースマイグレー
ションを実行する

composerでプラグイン
をインストールする

どんなコード？

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake
        end
      end
    end

    within "#{fetch(:release_path)}/cakephp2/app" do
      if test "[ -e #{fetch(:release_path)}/cakephp2/app/composer.phar
        execute "#{composer_path} ]" then
        execute "./composer.phar", 'install'
      end
    end

    if fetch(:run_migration) then
      execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
    end
  end

  if fetch(:sass_compile_path) then
    within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
      execute :compass, 'compile', '--b
    end
  end

  end
end
```

CakePHPライブラリへ
リンクを張る

データベースマイグレー
ションを実行する

composerでプラグイン
をインストールする

compassでsassファイル
をコンパイルする

どんなコード？

```
task :updated do
  on roles(:app) do
    ["cakephp1","cakephp2"].each do |cake_dir|
      ["cake", "lib", "index.php", "plugins", "vendors"].each do |part_dir|
        part_path = fetch(:deploy_to) + '/' + cake_dir + '/' + part_dir
        if test "[ -e #{part_path} ]" then
          execute :ln, '-snf', part_path, "#{fetch(:release_path)}/#{cake
        end
      end
      within "#{fetch(:release_path)}/cakephp2/app" do
        if test "[ -e #{composer_path} ]" then
          execute "composer.phar", 'install'
        end
        if fetch(:run_migration) then
          execute '../lib/Cake/Console/cake', 'Migrations.migration', 'run', 'all'
        end
      end

      if fetch(:sass_compile_path) then
        within "#{fetch(:release_path)}/#{fetch(:sass_compile_path)}" do
          execute :compass, 'compile', '--b
        end
      end

    end
  end
end
```

CakePHPライブラリへ
リンクを張る

composerでプラグイン
をインストールする

データベースマイグレー
ションを実行する

compassでsassファイル
をコンパイルする

アプリケーション特異的なタスクを追記していく

どんなコード？

```
task :restart do
  on roles(:app, :wp) do
    execute :sudo, :service, 'php5-fpm', 'restart'
  end
end

after :publishing, :restart

before :restart, :clear_cache do
  on roles(:app) do
    if fetch(:app_cache1_path) then
      execute :sudo, :find, fetch(:app_cache1_path), '-type f -exec rm {} \;'
    end
    if fetch(:app_cache2_path) then
      execute :sudo, :find, fetch(:app_cache2_path), '-type f -name "empty" -prune -o -type f -exec rm {} \;'
    end
  end
end
```

どんなコード？

php-fpmをリスタートする

```
task :restart do
  on roles(:app, :wp) do
    execute :sudo, :service, 'php5-fpm', 'restart'
  end
end

after :publishing, :restart

before :restart, :clear_cache do
  on roles(:app) do
    if fetch(:app_cache1_path) then
      execute :sudo, :find, fetch(:app_cache1_path), '-type f -exec rm {} \;'
    end
    if fetch(:app_cache2_path) then
      execute :sudo, :find, fetch(:app_cache2_path), '-type f -name "empty" -prune -o -type f -exec rm {} \;'
    end
  end
end
```

どんなコード？

```
task :restart do
  on roles(:app, :wp) do
    execute :sudo, :service, 'php5-fpm', 'restart'
  end
end

after :publishing, :restart

before :restart, :clear_cache do
  on roles(:app) do
    if fetch(:app_cache1_path) then
      execute :sudo, :find, fetch(:app_cache1_path), '-type f -exec rm {} \;'
    end
    if fetch(:app_cache2_path) then
      execute :sudo, :find, fetch(:app_cache2_path), '-type f -name "empty" -prune -o -type f -exec rm {} \;'
    end
  end
end
```

php-fpmをリスタートする

:publishingタスクの後に実行してね♡

どんなコード？

```
task :restart do
  on roles(:app, :wp) do
    execute :sudo, :service, 'php5-fpm', 'restart'
  end
end

after :publishing, :restart

before :restart, :clear_cache do
  on roles(:app) do
    if fetch(:app_cache1_path) then
      execute :sudo, :find, fetch(:app_cache1_path), '-type f -exec rm {} \;'
    end
    if fetch(:app_cache2_path) then
      execute :sudo, :find, fetch(:app_cache2_path), '-type f -name "empty" -prune -o -type f -exec rm {} \;'
    end
  end
end
```

php-fpmをリスタートする

:publishingタスクの後に実行してね♡

リスタートする前にキャッシュを消しといてね♡

どんなコード？

```
task :restart do
  on roles(:app, :wp) do
    execute :sudo, :service, 'php5-fpm', 'restart'
  end
end

after :publishing, :restart

before :restart, :clear_cache do
  on roles(:app) do
    if fetch(:app_cache1_path) then
      execute :sudo, :find, fetch(:app_cache1_path), '-type f -exec rm {} \;'
    end
    if fetch(:app_cache2_path) then
      execute :sudo, :find, fetch(:app_cache2_path), '-type f -name "empty" -prune -o -type f -exec rm {} \;'
    end
  end
end
```

php-fpmをリスタートする

:publishingタスクの後に実行してね♡

リスタートする前にキャッシュを消しといてね♡

既存のタスクの前後に任意のタスクを Hookする

どんなコード？

```
namespace :notify do
  namespace :update do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Deployment has started.")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Deployment has finished. :ok_woman:\n> #{fetch(:commit_message)}")
    end
  end

  namespace :rollback do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Rollback has started.\nCurrent Revision is \`#{fetch(:latest_revision)}\`")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Rollback has finished. :ok_woman:\nCurrent revision is
\`#{fetch(:current_revision)}\`")
    end
  end
end

before 'deploy:starting', 'notify:update:start'
after 'deploy:finishing', 'notify:update:finish'
before 'deploy:reverting', 'notify:rollback:start'
after 'deploy:finishing_rollback', 'notify:rollback:finish'
```

Capistranoによるデプロイの実行時にSlack通知する方法

HowTelevision

どんなコード？

```
namespace :notify do
  namespace :update do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Deployment has started.")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Deployment has finished. :ok_woman:\n> #{fetch(:commit_message)}")
    end
  end

  namespace :rollback do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Rollback                                                    test_revision)}\`")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Rollback
\`#{fetch(:current_revision)}\`")
    end
  end
end

before 'deploy:starting', 'notify:update:start'
after 'deploy:finishing', 'notify:update:finish'
before 'deploy:reverting', 'notify:rollback:start'
after 'deploy:finishing_rollback', 'notify:rollback:finish'
```

デプロイやロールバックの開始終了
の前後に通知を送る

Capistranoによるデプロイの実行時にSlack通知する方法

どんなコード？

```
namespace :notify do
  namespace :update do
    task :start do
      _send_message("[\`#{fetch(:app                                ment has started.")
    end

    task :finish do
      _send_message("[\`#{fet                                     as finished. :ok_woman:\n> #{fetch(:commit_message)}")
    end
  end

  namespace :rollback do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Rollback                                                    test_revision)}\`")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Rollback
\`#{fetch(:current_revision)}\`")
    end
  end
end

before 'deploy:starting', 'notify:update:start'
after 'deploy:finishing', 'notify:update:finish'
before 'deploy:reverting', 'notify:rollback:start'
after 'deploy:finishing_rollback', 'notify:rollback:finish'
```

定義したRubyの関数を呼び出す

デプロイやロールバックの開始終了の前後に通知を送る

Capistranoによるデプロイの実行時にSlack通知する方法

HowTelevision

どんなコード？

```
namespace :notify do
  namespace :update do
    task :start do
      _send_message("[\`#{fetch(:app                                  ment has started.")
    end

    task :finish do
      _send_message("[\`#{fet                                       as finished. :ok_woman:\n> #{fetch(:commit_message)}")
    end
  end

  namespace :rollback do
    task :start do
      _send_message("[\`#{fetch(:application)}\`] Rollback                                                    test_revision)}\`")
    end

    task :finish do
      _send_message("[\`#{fetch(:application)}\`] Rollback
\`#{fetch(:current_revision)}\`")
    end
  end
end

before 'deploy:starting', 'notify:update:start'
after 'deploy:finishing', 'notify:update:finish'
before 'deploy:reverting', 'notify:rollback:start'
after 'deploy:finishing_rollback', 'notify:rollback:finish'
```

定義したRubyの関数
を呼び出す

Rubyで拡張する

デプロイやロールバックの開始終了
の前後に通知を送る

Capistranoによるデプロイの実行時にSlack通知する方法

HowTelevision

デプロイ自動化の恩恵

- ミスが格段に減り、開発速度が上がった
- デプロイに対する心理的障壁が下がった
- コード化されることで作業が明確になった

すべてがうまくいった

もう悩みなんて無い

すべてがうまくいった

そんなわけない

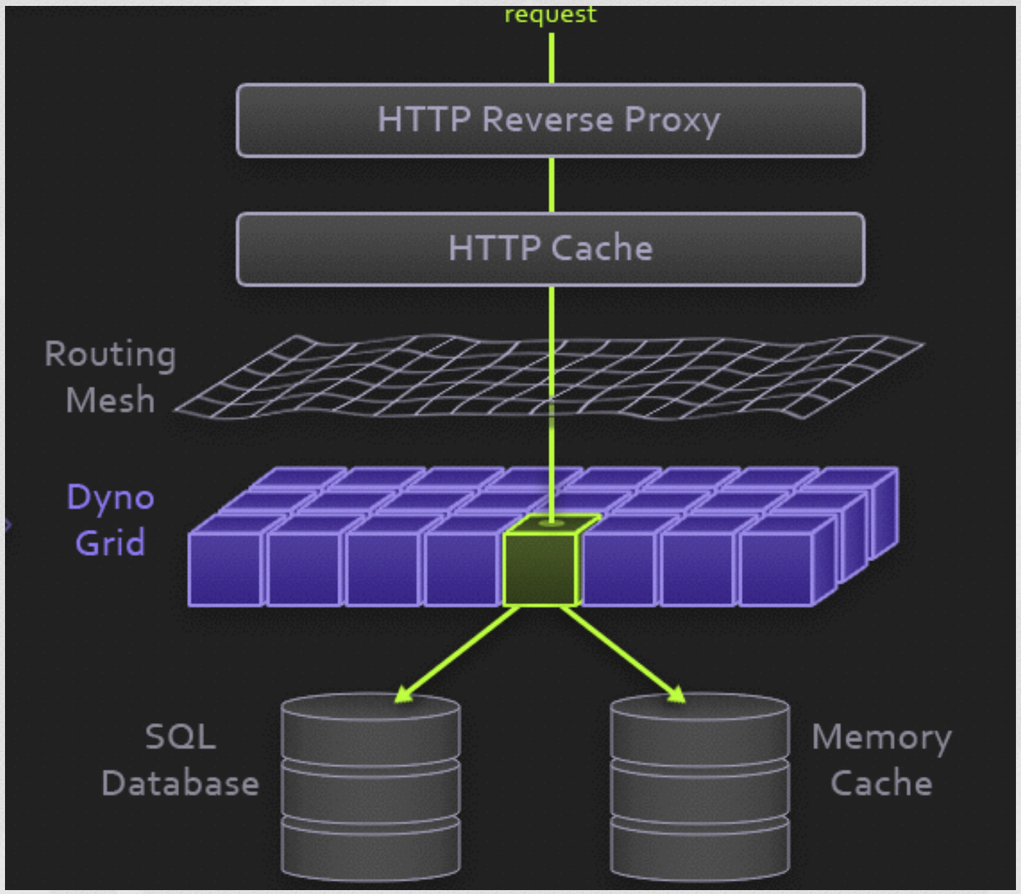
もう悩みなんて無い

VPSの問題

- スケールアウトするのはとても大変
- やっぱり大手企業には敵わないのか…。

PaaS時代

- Platform as a Service
  - ユーザーのシステムを稼働させる事ができるプラットフォーム自体を、インターネット経由でサービスとして利用できる形態
- Engine Yard（2006年）
- Heroku（2007年）



この時代

- Git（2005年）
- Ruby 1.9（2007年）
- Python 3.0（2008年）

PaaSのメリット

- ホストの管理はサービス事業者任せ
- 負荷分散は自動
- デプロイは `git push` だけ

PaaSの隠れたメリット

- ベストプラクティス的なところを強制する
  - データベース・アクセス用のパスワードをリポジトリに含めちゃダメよ♡
- アプリをポータブルにすることを強制する
  - アップロードファイルやログ、セッション情報などをアプリから分離させる

アプリの構成のさらなる複雑化

- Alternativ JSのコンパイル
- Alternative CSSのコンパイル

この時代

- Sass（2007年）
- LESS（2009年）
- Coffeescript（2009年）
- Ruby on Railsに含まれる（2011年）

IaaS時代

- Infrastructure as a Service
  - Amazon Web Service正式公開（2006年）
  - Google Compute Engine正式公開（2013年）

“良いVPS”では無かった

- APIによる操作
- 周辺サービスの充実
  - 分散ストレージ（S3）
  - データベース（DynamoDB／RDS）
  - DNS（Route 53）
  - CDN（CloudFront）
  - etc…

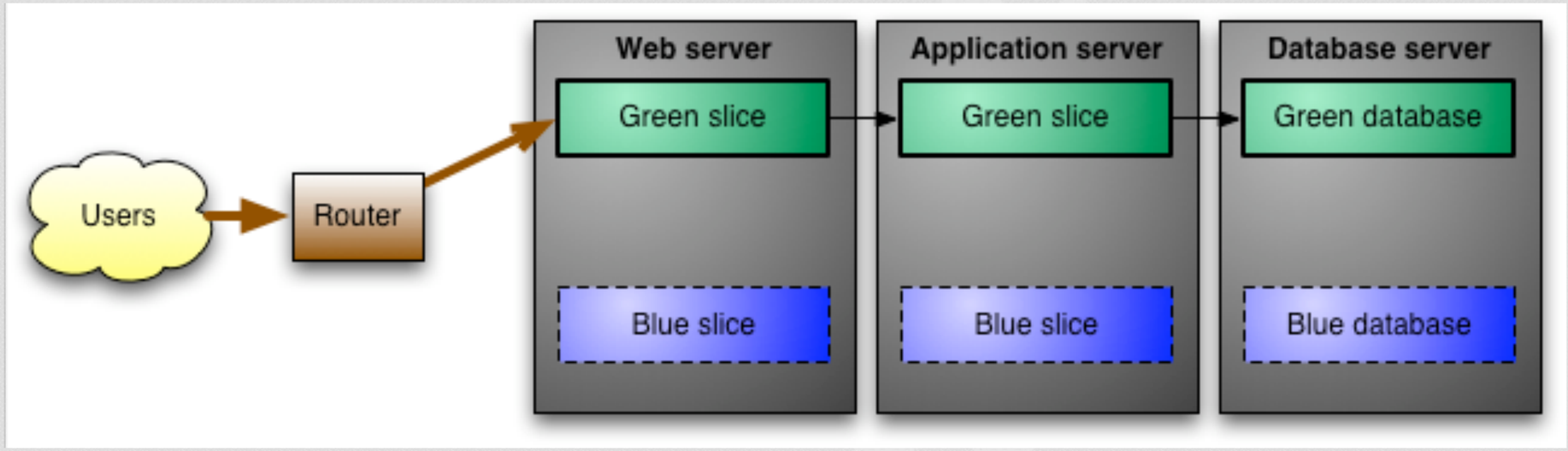
“良いVPS”では無かった

- APIによる操作
- 周辺サービスの充実
  - 分散ストレージ（S3）
  - データベース（DynamoDB／RDS）
  - DNS（Route 53）
  - CDN（CloudFront）
  - etc…

サーバーを使い捨てできるようになった

Immutable Infrastructure

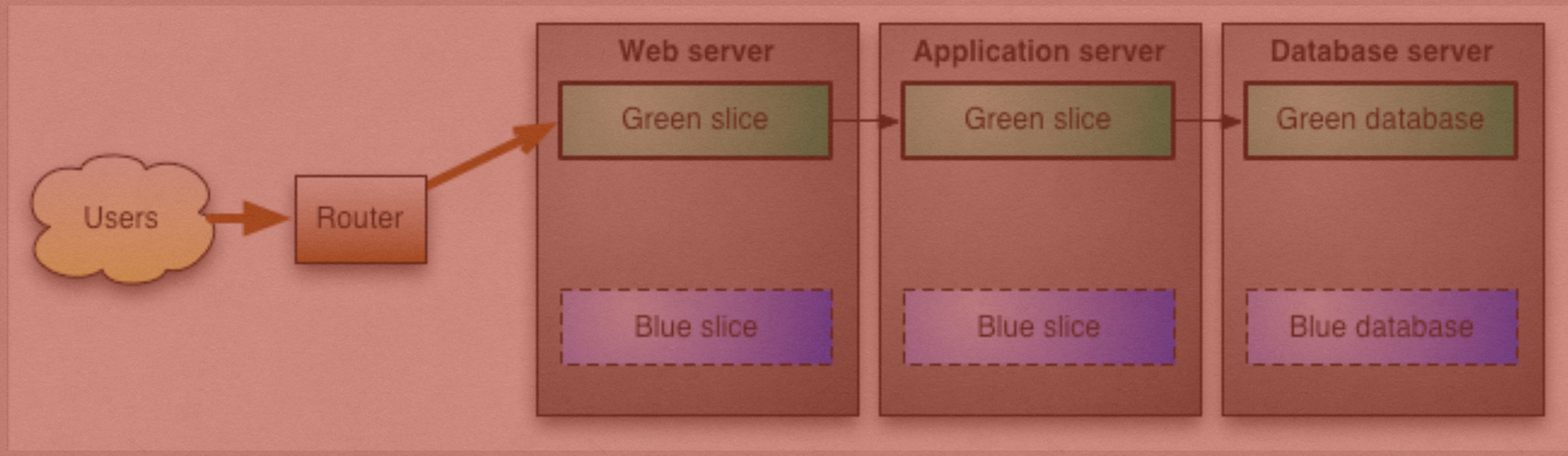
- サーバーの状態を管理しない
  - サーバーの内容を変更するのではなく、完成したサーバーと現行サーバーを切り替える



Immutable Infrastructure

- サーバーの状態を管理しない
  - サーバーの内容を変更するのではなく、完成したサーバーと現行サーバーを切り替える

アプリがポータブルになったから実現



その恩恵

- 大規模にスケールアウトしているサービスだけでなく、1ホストで事足りるサービスにすら、安全にバージョンアップができるようになった。

すべてがうまくいった

もう悩みなんて無い

すべてがうまくいった

今のところはね

もう悩みなんて無い

まとめ

- デプロイの手法という視点で大まかな歴史をなぞった。
  - どういった経緯でデプロイの手法が発展してきたのか？
- どういった技術が使われてきたのか？
- どのように思想が変わってきたのか？

完

ありがとうございました。

https://joind.in/12041