仕事で使えるComposer

無修正版

ピクシブ株式会社
うさみけんた @tadsan

pixiv

PHPカンファレンス北海道2016

お前誰よ

- うさみけんた（@tadsan）/ Zonu.EXE
  - GitHub/Packagistでは id:zonuexe
- プログラミング言語と日本語が好き
- 北海道工業大学(当時) 電子計算機研究部出身
- 2012年頃までRuby札幌とかPython札幌で活動
- 2012年11月に初めての就職（現職）
  - PHPは現職に所属してからまじめに取り組み

pixiv

おしながき

- Composerの基礎／導入
- ナウいPHPライブラリの探しかた
- Composerとデプロイについて
- そのほかもろもろのこと

pixiv

Version 1.0.0 was released on April 5, 2016

Composer #とは

Composerって何よ

- 依存関係管理ツール（Dependency manager）
- YumやAPTはシステムにライブラリを導入する
- Composerは、特定のプロジェクト単位に対して閉じて依存性を管理する
- npm(Node.js)やBundler(Ruby)に影響を受けた
- グローバル(特定のプロジェクトに依存しない)インストールも可能ではある

Get Composer!

- 2016年4月5日(今月) 正式版1.0.0リリース！
- https://getcomposer.org/download/

- PHAR形式で配布されてるので、
  PHP(5.3.2+)がある環境ならどこでも動作する
- Linux/OS XのみならずWindowsもサポートしてる
- Packagist, GitHub, PEARなどと連携可能

Composer VS pear

- pearはPHP標準のパッケージ管理コマンド
- 機能の豊富さ、利用できるパッケージの数、新鮮さのような違いはもちろんある。
- それ以上に、PHPプロジェクトと同時にデプロイ可能になったことが非常に重要
- ライブラリを導入する権限がインフラ／サーバー管理者からアプリエンジニアの手に落ちてきた（身分制度の変革）

Composerは何ではないか

- PHPのC extensionはインストールできない
  - インストール済か確認することはできる
  - 従来通りPECLなどを利用することになる

Composerの基礎

composer.json

- プロジェクトの情報を記述するファイル
  - 一般的にはプロジェクトのルートに置く
- 依存情報やオートローディング設定など
- パッケージ名はvendor/packageの形式
  - vendorは組織名やアカウント名をつける
  - Javaのように厳密に決まってるわけじゃない

ライブラリの依存を記述

- requireに依存するパッケージ+バージョンを記述

```
{
    "require": {
        "filp/whoops": "~2.1.0"
    },
    "require-dev": {
        "phpunit/phpunit": "^4.8"
    }
}
```

- composer require --dev "phpunit/phpunit:4.8"

バージョン記法（一部）

- "*": 全てのバージョンを受け入れる
- "1.2.3"：バージョンタグ固定
- "~1.2.3": バージョンタグ固定(マイナー更新する)
- "dev-xxxxx": ブランチxxxxxを指定
  - dev-master#82115e5e37fa046464cのようにブランチとコミットIDを指定できる(しないと危険)

composer.lock

- 依存パッケージのバージョンを固定するファイル
  - composer.jsonでは範囲や複数指定が可能
  - 固定することで、意図しないバージョンが勝手にインストールされることを防ぐ
  - 自動生成されるので手動編集しない方が良い

composer.lockの管理

- composer update vendor/package で更新
- 自動生成されるcomposer.lockをGitなどのバージョン管理の対象にするべきかは場合による
  - アプリケーションでは管理することが多い
  - ライブラリは基本的に管理しない
- 管理しない場合は .gitignore に設定しておく

Composerを使う

- 依存ライブラリはvendor/ディレクトリに入る
  - プロジェクトの .gitignore で除外しておく
- Composerに依存するPHPには以下の一行を書く

```
require __DIR__ . '/vendor/autoload.php';
```

- 何度も書く必要はなく、一度だけ読み込めばいい
- Composerでインストールされたライブラリは、include_onceする必要がない

オートローディング

- PHPerをinclude_onceから解き放つ福音

```
{
    "autoload": {
        "classmap": ["./src"]
    },
    "autoload-dev": {
        "classmap": ["./tests"]
    }
}
```

- 詳しくはWEB+DB Vol. 81に書きました(宣伝)

サブコマンド（一部）

- `composer install`: 必要なパッケージをインストール
- `composer list`: 機能一覧を出力
- `composer self-update`: composer.pharを更新
- `composer search`: パッケージ検索
- `composer validate`: composer.jsonのチェック
- `composer create-project`:
  フレームワークの雛形からプロジェクトを新規作成

Composerを導入する

Composerの配置を考える

- 公式サイトのスクリプトでは、ダウンロードまでしかしてくれない（Windowsにはしてくれる）
- 仕事の性質に配置方法には一考の余地がある
  - サーバーごと
  - ユーザーごと
  - プロジェクトごと

サーバーごとの場合

- プロジェクトメンバーが共有の開発サーバーにログインして作業する場合にとれる方法

- /usr/local/bin/composer.phar など

- 良い点：Composerのバージョン差異がなくなる

- 悪い点：サーバー管理者が更新しなきゃいけない
  サーバー多いとめんどくさい

ユーザーごとの場合

- プロジェクトメンバーが自分のPCで開発する場合とか、ライブラリをいろいろ作る場合とか

- $HOME/bin/composer.phar など（好みによる）

- 良い点：メンテナンスするライブラリの数が多い場合は管理が楽

- 悪い点：バージョンを同期させるのがめんどう
  PATH通したりsudoさせたり説明めんどう

プロジェクトごとの場合

- 大規模な開発チームで一つのリポジトリを継続的に開発していく場合とか
- プロジェクト内の /composer.phar など
- 良い点：メンバー間のバージョン差異はなくせる
- 悪い点：プロジェクト数が多いと運用が面倒
  ディスクスペースの浪費（1.6MB程度）

結局どれがいいの？

- PHPに詳しくないひとのことを考えると、プロジェクトごとにcomposer.pharを置くのが一番楽じゃないかと思う
- セットアップスクリプトでダウンロードさせてもいいけど、pixivでは git commit しちゃってる
- 毎月更新するとして1.6MBのディスクスペースの浪費は、運用の手間を考えれば甘受できるレベル

xdebugとComposer

- xdebugが有効な環境でComposer使うと怒られる

```
You are running composer with xdebug enabled. This has a major
impact on runtime performance. See https://getcomposer.org/xdebug
```

- xdebugは開発中は有用なので外したくない
- 実際遅くなるのでシェルスクリプトを噛ませたい
- composer.phar を php -n で起動できればいい
  - php.iniを無効化する＝拡張が有効にならない

Composerのラッパー

- プロジェクトローカルに置く場合…
  - composer.pharを.composer.pharにリネーム
  - かわりに ./composer を利用
  - chmod +x ./composer で実行権限を付与
  - ファイルの中身は以下の二行

```
#!/bin/sh
php -n $(dirname $0)/.composer.phar "$@"
```

環境変数

- Composerに必要なファイルはCOMPOSER_HOME環境変数のディレクトリにインストール

- デフォルトでユーザディレクトリにインストール
  (Linux/OS X) $HOME/.composer
  (Windows) %APPDATA%¥Composer

- システムの設定を工夫すれば共有ディレクトリにインストールすることも可能ではある
  (やったことないけど)

Composerを導入

- UNIX系OSでユーザごとにインストールするとき $HOME/local/bin/composer に配置しておくと便利（.bash_profileや.zshenvなどでPATHを通す）

```
# .bash_profile/.bashrc
PATH=$HOME/.composer/vendor/bin:$HOME/local/bin:$PATH
# .zshenv/.zshrc
path=(
    ~/.composer/vendor/bin(N-/)
    ~/local/bin(N-/)
    $path
)
```

PHPのライブラリを探す

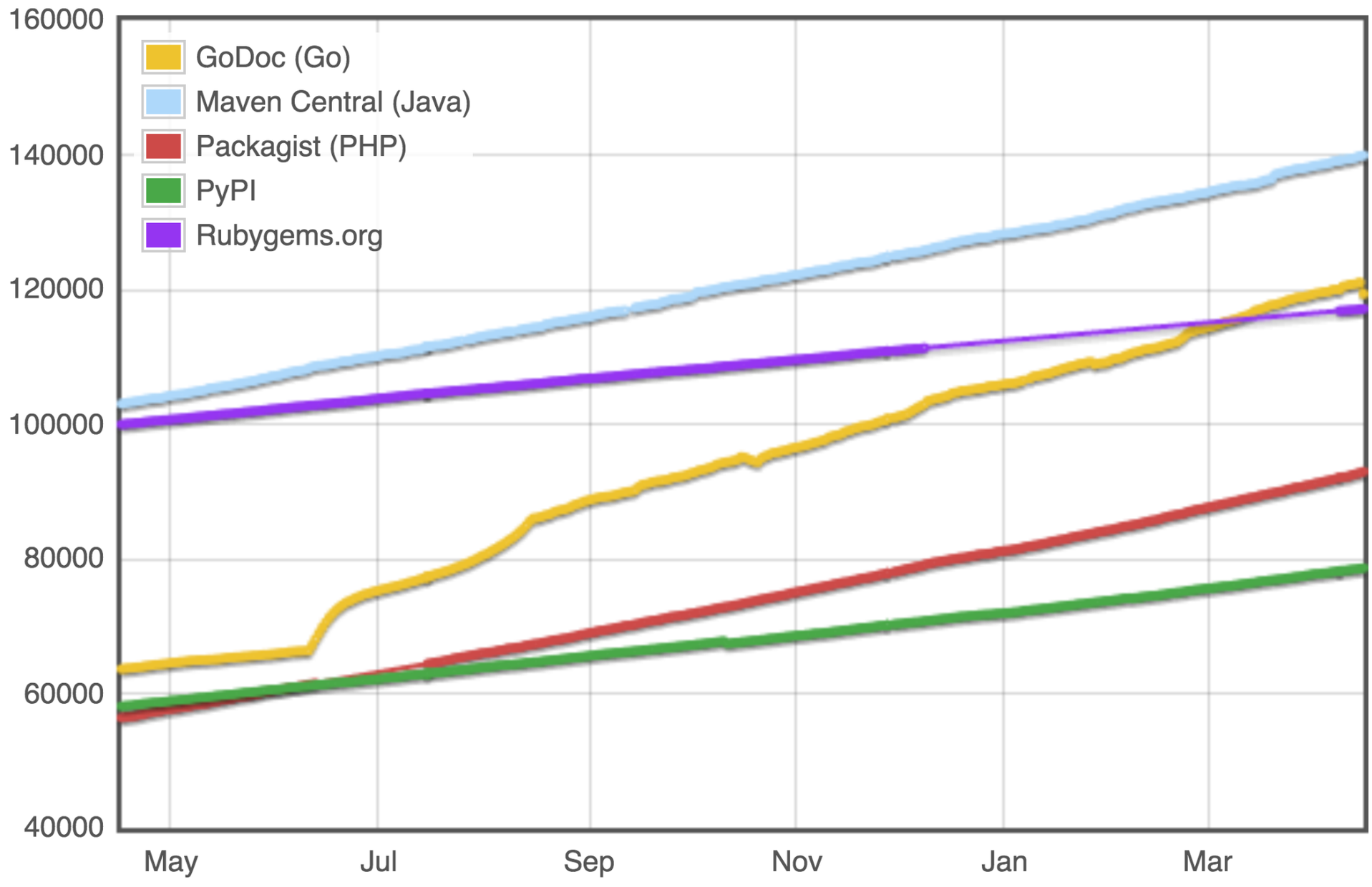
Packagist

The PHP Package Repository

Packagistって何だ

- https://packagist.org/
- PHPのパッケージリポジトリ
- 2012年から4月から活動してる（４周年！）
- 現在92,973パッケージ登録されてる
- ちなみにPEARのパッケージ数は602

Module Counts



http://www.modulecounts.com/

Packagistの特徴

- GitHubなどのリポジトリと簡単に連携できる
  - API連携した状態なら、git tagをつけるだけでかんたんに新しいバージョンをリリースできる
- 有名なライブラリ／フレームワークは登録済み
  - 歴史的にフレームワークはzipファイルで公開されてきたが、現在はComposerでもインストールできる

ライブラリを探す

- とりあえず awesome-php に載ってるものを触ってみる
  - 各ジャンルの手堅いライブラリの一覧
  - たいていのコンポーネントは大手が作ってる
    - SymfonyとかHoaProjectとか
- Twitterの@call_user_funcをフォローすると新着のライブラリがいっぱい流れてくる

Composerとデプロイ

デプロイ、どうしてますか？

- 稼働中のサーバーに高速にコードを配置したい
- rsyncなどで愚直にファイルを配置すると、
  最中のPHPファイルが書き変わっちゃって
  処理途中のユーザーリクエストがエラーで落ちる
- Composerにも自動生成ファイルがあるので、
  デプロイを工夫しないと、やはり落ちることが

アトミックデプロイ

- エラーが出ないようにデプロイ途中の不完全な状態がユーザーの目に触れないよう工夫する
- 弊社事例(rsync+デプロイスクリプト手書き)はWEB+DB PRESS vol. 84に載ってる
- 最近はDeployer http://deployer.org/ がいい感じにアトミックデプロイしてくれるらしい(環境に合せて検証の必要あり)

Satis - Package Repository Generator

- https://github.com/composer/satis
- 要はプライベートなPackagistを自前運用できる
- パッケージのアーカイブを保存できるので、
  もしGitHubが不通になってしまっても
  デプロイできない事態を避けることができる
- デプロイ先のサーバーとネットワーク的に
  近いところにあればデプロイ時間の短縮になる

Satisの運用

- 必要なライブラリをsatis.jsonに列挙する
  - 記法はcomposer.jsonと同じ
    - "~2.1.0"記法よりは">=2.1.0"が良い
- satis.jsonとビルドスクリプトを置いたリポジトリを用意して、毎日定時とsatis.jsonが更新されたときに新鮮なパッケージを取得

Satisを使う場合のcomposer.json

```
{
    "repositories": [
        {
            "packagist": false
        },
        {
            "type": "composer",
            "url": "http://satis.pixiv.private/"
        }
    ]
}
```

パッケージ作りたい

基本

- PackagistはGitHubでログインできるので、GitHubに上げると非常に楽
- ブラウザで https://packagist.org を開いて説明通りに入力していくだけ
- 新しいバージョンのリリースは、git tag 1.4.5 && git push --tags で完了

お作法

- Composerはオートローディングが基本なので、不必要なinclude_onceは書かない
- PSR-1, PSR-2, PSR-4 を読んでおけば、だいたい綺麗なコードが書ける気がする
  - PSR-1, PSR-2はインフィニットループさんによる日本語訳がWebにあります
- とりあえずnamespaceは付けといた方が良い

Composerが遅い

Packagistは遠い

- GitHub/Packagistは日本から遠いところにあるので、ネットワークコストが非常に高い
  - これはSatisを置けば解決する
- Satisを用意できない／用意するまでもないときは、Hirakuさんという方がミラーサイト
  https://packagist.jp/
  を用意してくれてるので、これを使える

Composerは遅い

- ComposerはPHPの互換性を重視してるので、最適な実装ではない
- Hirakuさんが並列ダウンロードできるようにした
  - composerを速くするプラグイン・prestissimoを作った http://blog.tojiru.net/article/432944706.html
- GitHubで既に★1660個ついてる！

グローバルに
ライブラリをインストール

どんなときにグローバル？

- 実際出番はあまりない
- コマンドとしてインストールしたいライブラリ
- いちいちプロジェクトを作るまでもない
  使い捨てコードから利用したいライブラリ
- グローバルインストールしても、
  勝手に読み込まれるようなことはない

PsySHはべんり

- composer global require psy/psysh
- PHPの対話環境（REPL）
- RubyのIRB, Pryのようなもの
- ただし実行中に関数やクラスの再定義はできないので注意すること
- psysh.el もあるよ（宣伝／未完成）