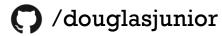
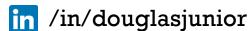
INTRODUÇÃO AO PHP COMPOSER







douglasjunior.me

massifrroma@gmail.com



Slides: https://git.io/vAd6S





- Introdução ao PHP Composer
- Instalação e configuração do ambiente.
- Criando projetos
- Instalando dependências de terceiros
- Autoloader
- Criando bibliotecas
- Importando bibliotecas



INTRODUÇÃO AO PHP COMPOSER

- Quem já trabalhou com PHP, sabe a dificuldade em se trabalhar com gerenciamento de pacotes.
- Por pacotes, entenda os módulos de terceiros que precisam ser integrados a sua aplicação.
 - Frameworks
 - Biblioteca de Log
 - Bib. formatação de data/hora
- Antes do PHP Composer, toda essa integração era manual e trabalhosa.







Dependency Manager for PHP



INTRODUÇÃO AO PHP COMPOSER

"Composer is a tool for dependency management in PHP. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you." – Composer Docs

"Composer é uma ferramenta para gerenciamento de dependências em PHP. Ele permite que você declare as bibliotecas nas quais seu projeto depende, e então ele irá gerenciá-las (instalar/atualizar) para você." – Composer Docs



INSTALAÇÃO DO COMPOSER

- O PHP Composer é distribuído através de um arquivo PHAR.
 - PHAR é uma extensão que permite que você distribuía aplicações PHP como um único arquivo de maneira fácil.
- Em qualquer sistema operacional, para instalar o Composer basta efetuar o Downlod do arquivo composer.phar.
- Porém, para incluir o comando composer nas variáveis de ambiente, a melhor maneira é seguir as instruções descritas na documentação:
 - https://getcomposer.org/doc/00-intro.md



- Para começar a usar o Composer, tudo que você precisa é criar um arquivo chamado composer. json na raiz de seu projeto.
- Este arquivo irá descrever as dependências de seu projetos e também alguns metadados que veremos no futuro.
- Exemplo:

```
{
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```



- A primeira (quase sempre a única) propriedade declarada é a chave require.
- Neste exemplo, estamos dizendo ao composer que o nosso projeto depende da biblioteca "monolog".
- As dependências são compostas do nome do pacote (monolog/monolog) e o número da versão (1.0.*).

```
{
   "require": {
     "monolog/monolog": "1.0.*"
   }
}
```



- O Composer usa as informações da chave "require" para buscar a dependências e versão correta dos pacotes, usando os repositórios que você especificar, ou no repositório público padrão <u>Packagist</u>.
- Os nomes dos pacotes consistem em "nome do distribuidor" e "nome do pacote".
 Isso permite que duas pessoas diferentes tenham dois pacotes com o mesmo nome, por exemplo:
 - igorw/json
 - seldaek/json



- Em nosso exemplo, estamos solicitando o pacote Monolog com a versão 1.0.*
- Isso significa que qualquer versão dentro da faixa 1.0 e 1.1 serão consideradas.
 - Ou seja >= 1.0 e < 1.1
- Você pode encontrar mais detalhes sobre as possibilidades de versões em:
 - https://getcomposer.org/doc/articles/versions.md



INSTALANDO DEPENDÊNCIAS

 Para que o Composer instale as dependências definidas no projeto, basta rodar o comando install.

```
php composer.phar install
```

Ou, se você definiu o Composer nas variáveis de ambiente:

```
composer install
```



INSTALANDO DEPENDÊNCIAS

- Se você está usando o comando install pela primeira vez, irá perceber que foi criado um arquivo composer.lock
- Este arquivo funciona como um "cache" da árvore de dependências presente no seu composer. json.
- Além disso, o composer.lock conserva a versão real com que as dependências foram instaladas em seu projeto.
- Sendo assim, é uma boa prática commitar o composer.lock em seu controlador de versões.



ATUALIZANDO DEPENDÊNCIAS

• Para atualizar as versões das dependências, basta executar o comando update:

composer update

Ou, para atualizar apenas um pacote específico:

composer update monolog/monolog



PACOTES DE PLATAFORMA

- O Composer também permite que você declare pacotes de plataforma, que são pacotes virtuais para coisas que são instaladas no sistema, mas não são um pacote PHP.
- Por exemplo, você pode definir o pacote PHP informando a versão mínima desejada.

```
{
    "require": {
        "php": ">=7.1",
        "monolog/monolog": "1.0.*"
    }
}
```



- Para bibliotecas que exigem informação de autoload, o Composer define no arquivo vendor/autoload.php
- Você pode simplesmente incluir este arquivo em seu projeto e está pronto para utilizar as classes disponíveis nas bibliotecas, sem nenhum trabalho extra.

```
c?php

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

$log = new Monolog\Logger('meu-proj');
$log->pushHandler(new Monolog\Handler\StreamHandler('app.log', Monolog\Logger::WARNING));
$log->addWarning('Mensagem de alerta!');
```



• Você também pode adicionar seus próprios arquivos ao autoloader no composer. json.

```
{
    "autoload": {
        "psr-4": {"ExpNordeste\\": "src/"}
    }
}
```

- O Composer vai registrar um autoloader PSR-4 para o namespace ExpNordeste.
- Isso permite mapear namespaces para seus diretórios.



• Por exemplo, crie uma classe Pessoa no arquivo com caminho src/Pessoa.php

```
<?php
namespace ExpNordeste;

class Pessoa {
   public $nome;
}</pre>
```

• E então instancie um objeto à partir desta classe:

```
$pessoa = new ExpNordeste\Pessoa();
$pessoa->nome = "Teste";

var_dump($pessoa);
```



• Exemplos de namespaces:

FULLY QUALIFIED CLASS NAME	NAMESPACE PREFIX	BASE DIRECTORY	RESULTING FILE PATH
\Acme\Log\Writer\File_Writer	Acme\Log\Writer	./acme-log-writer/lib/	./acme-log-writer/lib/File_Writer.php
\Aura\Web\Response\Status	Aura\Web	/path/to/aura-web/src/	/path/to/aura-web/src/Response/Status.php
\Symfony\Core\Request	Symfony\Core	./vendor/Symfony/Core/	./vendor/Symfony/Core/Request.php
\Zend\Acl	Zend	/usr/includes/Zend/	/usr/includes/Zend/Acl.php



CRIADO BIBLIOTECAS

- O simples fato de existir um composer.json no diretório do projeto, já faz deste projeto um pacote.
- Para que seu pacote seja "instalável", você precisa atribuir a ele um nome:

```
{
   "name": "nodeste/biblioteca",
   "require": {
     "monolog/monolog": "1.0.*"
   }
}
```



CRIADO BIBLIOTECAS

• Na maioria dos casos, você utiliza um gerenciador de versões (VCS) para armazenar o código (Git, Svn, etc), sendo assim, o Composer infere versões a partir do seu VCS e você não deve informar o número de versão no composer.json.

Porém, se você deseja manter os números de versão manualmente:

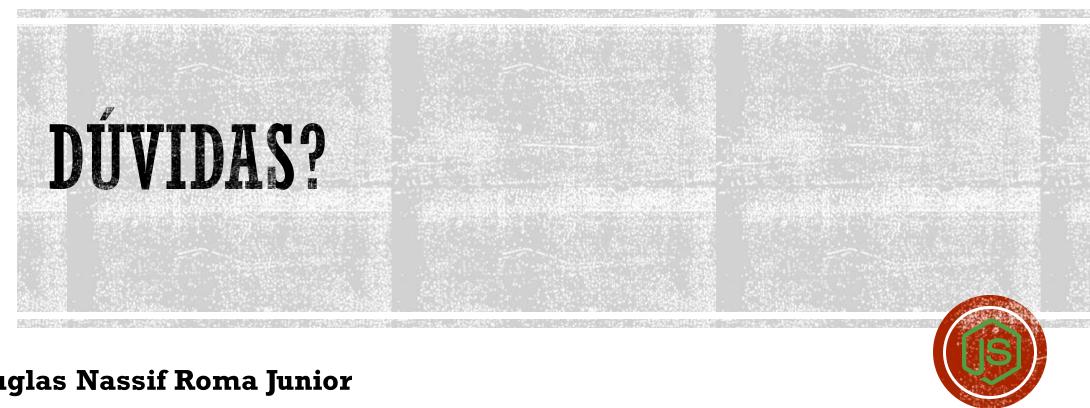
```
"name": "nodeste/biblioteca",
    "version": "1.0.0",
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```



INSTALANDO BIBLIOTECAS

 Uma vez que sua biblioteca está no controlador de versão, você pode configurar o caminho dos repositórios da seguinte maneira:

```
"name": "nordeste/projeto",
"repositories": [
   "type": "vcs",
    "url": "https://github.com/username/biblioteca"
"require": {
  "nordeste/biblioteca": "dev-master"
```



Douglas Nassif Roma Junior

- /douglasjunior
- in /in/douglasjunior
- douglasjunior.me

massifrroma@gmail.com

Slides: https://git.io/vAd6S