

[Home](#) » [Desenvolvimento](#) » [Migrations para aplicações PHP com Phinx](#)  28 de novembro de 2017  Juciellen Cabrera  1771 Views

Migrations para aplicações PHP com Phinx

Em tempos que se fala tanto de DevOps quero te mostrar como utilizar o **Phinx** para efetuar alterações no seu banco de dados à medida que a sua aplicação evolui, possibilitando que você automatize esse processo com uma ferramenta de integração contínua logo em seguida.


Imagine a seguinte situação: no seu ambiente de desenvolvimento você criou sua aplicação em PHP bem como o seu banco de dados. Logo em seguida você o colocou em produção. Até aí está tudo lindo, maravilhoso. Dando continuidade ao seu projeto você percebeu a necessidade de criar uma tabela nova. Ok, sem problemas, no seu ambiente de desenvolvimento você cria a bendita tabela. Mas como você procede para fazer isso em produção?

Talvez você não tenha percebido o problema agora, mas no futuro e provavelmente bem próximo, conforme as demandas forem aumentando, fica complicado administrar essas alterações do banco de dados em produção. E se você tiver um ambiente de homologação então, fazer a mesma alteração no banco de dados 3 vezes, já pensou?

Se você passa por esse tipo de situação quero lhe apresentar o **Phinx**.

O que é o Phinx?

De acordo com a documentação, o **Phinx** é uma ferramenta via linha de comando para gerenciamento de migrações de banco de dados. Com ele você poderá escrever scripts PHP para efetuar as alterações que você necessita no seu banco de dados e gerenciar isso via linha de comando.

Alguns frameworks fullstack de PHP tem recurso para *migrations*, inclusive o do CakePHP é baseado no próprio **Phinx**  le a pena falar dele ou talvez porque você não esteja usando *framework*, ou se trata de código legado com um *framework* sem esse recurso ou simplesmente pela facilidade que o **Phinx** proporciona.

Instalação

A instalação pode ser feita via composer:

```
01 juciellen@cabrera:~/Documentos/projetos/expressive$ php composer.phar require robmorgan/phinx
02 Using version ^0.9.1 for robmorgan/phinx
03 ./composer.json has been updated
04 Loading composer repositories with package information
05 Updating dependencies (including require-dev)
06 Package operations: 3 installs, 0 updates, 0 removals
07 - Installing symfony/filesystem (v3.3.13): Downloading (100%)
08 - Installing symfony/config (v3.3.13): Downloading (100%)
09 - Installing robmorgan/phinx (v0.9.1): Downloading (100%)
10 Writing lock file
11 Generating autoload files
```

A seguir crie os diretórios db/migrations para armazenar as suas migrations não esquecendo de dar as devidas permissões. Feito isso execute o comando abaixo no terminal:

```
1 juciellen@cabrera:~/Documentos/projetos/expressive$ php vendor/bin/phinx init
2 Phinx by CakePHP - https://phinx.org. 0.8.1
3
4 created ./phinx.yml
```

Repare que o arquivo **phinx.yml** foi criado na raiz do projeto. Esse é o arquivo de configuração do phinx.

Configurando o Phinx

Abaixo um modelo enxuto de phinx.yml.

```
01 paths:
02 migrations: %%PHINX_CONFIG_DIR%%/db/migrations
03
04 environments:
05 default_migration_table: phinxlog
06 default_database: development
07
08 development:
09 adapter: pgsql
10 host: localhost
11 name: db_test
12 user: SEU_USUARIO
13 pass: SENHA_SEGURA
14 port: 5432
15 charset: utf8
16
17 version_order: creation
```

Neste arquivo você precisará indicar os diretórios das suas migrations em **paths**, no nosso caso **db/migrations**.

Em **environments** deve conter as configurações para conexão com banco de dados em todos os ambientes em que você executará suas *migrations*. Eu deixei apenas as configurações do meu ambiente de desenvolvimento mas você pode acrescentar quantos blocos forem necessários, como homolog, production e etc. Eu estou usando *Postgres*, por isso o adapter é *pgsql*, mas consulte o manual do *phinx* para indicar o adapter de acordo com o SGDB que você estiver usando.

Escrevendo *migrations*

Para criar uma *migration* você deve executar o seguinte comando no terminal, sempre a partir do diretório que estiver **phinx.yml**



```
1 juciellen@cabrera:~/Documentos/projetos/expressive$ php vendor/robmorgan/phinx/bin/phinx create
  MinhaPrimeiraMigration
```

O padrão de nome de *migration* é *CamelCase*. O resultado é o seguinte:

```
01 Phinx by CakePHP - https://phinx.org. 0.8.1
02
03 using config file ./phinx.yml
04 using config parser yaml
05 using migration paths
06 - /home/juciellen/Documentos/projetos/expressive/db/migrations
07 using seed paths
08 using migration base class Phinx\Migration\AbstractMigration
09 using default template
10 created db/migrations/20171124233848_minha_primeira_migration.php
```

Repare que o arquivo 20171124233848_minha_primeira_migration.php foi criado no diretório *db/migrations*, que indicamos na configuração. É nesse arquivo que escreveremos nossa *migration*.

```
01 <?php
02 use Phinx\Migration\AbstractMigration;
03
04 class MinhaPrimeiraMigration extends AbstractMigration
05 {
06     /**
07      * Change Method.
08      *
09      * Write your reversible migrations using this method.
10      *
11      * More information on writing migrations is available here:
12      * http://docs.phinx.org/en/latest/migrations.html#the-abstractmigration-class
13      *
14      * The following commands can be used in this method and Phinx will
15      * automatically reverse them when rolling back:
16      *
17      * createTable
18      * renameTable
19      * addColumn
20      * renameColumn
21      * addIndex
22      * addForeignKey
23      *
24      * Remember to call "create()" or "update()" and NOT "save()" when working
25      * with the Table class.
26     */
27     public function change()
28     {
29     }
30 }
31 }
```

Notem que o arquivo já vem com um método chamado **change**. É dentro desse método que você vai escrever o que vai ocorrer quando a *migration* for executada (você pode usar o método **up** para escrever a *migration*, mais a frente explico a diferença). Vamos escrever uma *migration* que cria a tabela **users** contendo os campos **id**, **name**, **email**, **password**, **active** e **created**.

```
01 public function change()
02 {
03     $this->table('users')
04         ->addColumn('name', 'string')
05         ->addColumn('email', 'string', [
06             'null' => false
07         ])
08         ->addColumn('password', 'string', [
09             'null' => false
10         ])
11         ->addColumn('active', 'boolean', [
12             'null' => false,
13             'default' => true
14         ])
15         ->addColumn('created', 'datetime', ['default'=>'CURRENT_TIMESTAMP'])
16         ->create();
17 }
18 }
```

Reparem que não criei o campo **id**, porque o *phinx* vai fazer isso automaticamente ao executar a *migration*.

Utilizei alguns métodos que vou explicar melhor:

table -> passando como parâmetro *users*, que é o nome da tabela que desejo trabalhar.

addColumn -> passando como parâmetros o nome da nova coluna, o tipo e as opções. Em opções é onde você pode indicar se o

campo pode ser nulo, definir um valor default e por aí vai.

create -> cria a tabela

Executando *migrations*

Para executar as migrations você deve executar o comando **migrate**. Caso você precise executar a *migration* em mais de um ambiente não esqueça de indicá-lo, no meu caso **migrate -e development**.

```
01 juciellen@cabrera:~/Documentos/projetos/expressive$ php vendor/robmorgan/phinx/bin/phinx migrate -e
02 development
03 Phinx by CakePHP - https://phinx.org. 0.8.1
04
05 using config file ./phinx.yml
06 using config parser yaml
07 using migration paths
08 - /home/juciellen/Documentos/projetos/expressive/db/migrations
09 using seed paths
10 using environment development
11 using adapter pgsq
12 using database db_test
13
14 == 20171124233848 MinhaPrimeiraMigration: migrating
15 == 20171124233848 MinhaPrimeiraMigration: migrated 0.1273s
16
17 All Done. Took 0.1428s
```

Após a execução a tabela foi criada, bem como a respectiva sequence. Simples não?

```
01 db_test=> \d
02 List of relations
03 Schema | Name | Type | Owner
04 -----+-----+-----+-----
05 public | phinxlog | table | jucabrera
06 public | users | table | jucabrera
07 public | users_id_seq | sequence | jucabrera
08 (3 rows)
09
10 db_test=> \d users
11 Table "public.users"
12 Column | Type | Modifiers
13 -----+-----+-----
14 id | integer | not null default nextval('users_id_seq'::regclass)
15 name | character varying(255) | not null
16 email | character varying(255) | not null
17 password | character varying(255) | not null
18 active | boolean | not null default true
19 created | timestamp without time zone | not null default now()
20 Indexes:
21 "users_pkey" PRIMARY KEY, btree (id)
```

O *phinx* gerencia as *migrations* executadas utilizando a tabela **phinxlog** criada por ele automaticamente no seu banco de dados, de modo que uma vez executada a *migration*, ela não será executada novamente.

Você pode acompanhar isso através do comando status.

```
01 juciellen@cabrera:~/Documentos/projetos/expressive$ php vendor/robmorgan/phinx/bin/phinx status -e
02 development
03 Phinx by CakePHP - https://phinx.org. 0.8.1
04
05 using config file ./phinx.yml
06 using config parser yaml
07 using migration paths
08 - /home/juciellen/Documentos/projetos/expressive/db/migrations
09 using seed paths
10 using environment development
11 ordering by creation time
12
13 Status [Migration ID] Started Finished Migration Name
14 -----+-----+-----+-----
15 up 20171124233848 2017-11-24 22:41:47 2017-11-24 22:41:47 MinhaPrimeiraMigration
```

Rollback

Pode ocorrer de você perceber que esqueceu de criar algum campo, por exemplo, e queira desfazer o que acabou de fazer. Sendo assim você pode utilizar o **rollback** (use com moderação).

Para desfazer a última *migration* execute o comando **rollback -e development**.

```
01 juciellen@cabrera:~/Documentos/projetos/expressive$ php vendor/robmorgan/phinx
02 /bin/phinx rollback -e development
03 Phinx by CakePHP - https://phinx.org. 0.8.1
04
05 using config file ./phinx.yml
06 using config parser yaml
07 using migration paths
08 - /home/juciellen/Documentos/projetos/expressive/db/migrations
09 using seed paths
10 using environment development
11 using adapter pgsql
12 using database db_test
13 ordering by creation time
14
15 == 20171124233848 MinhaPrimeiraMigration: reverting
16 == 20171124233848 MinhaPrimeiraMigration: reverted 0.0429s
17
18 All Done. Took 0.0577s
```

O bacana de escrever a *migration* no método *change* utilizando os métodos específicos do *phinx* (table, addColumn, etc) é que em caso de *rollback* o *phinx* saberá exatamente como desfazer a ação. Se usarmos o método **up** ao invés do *change*, em caso de *rollback* se precisarmos desfazer o que foi definido no *up* vamos precisar escrever as instruções para isso no método **down**.

Após o *rollback*, meu banco não conterá mais a tabela *users* criada anteriormente.

```
1 db_test=> \d
2 List of relations
3 Schema | Name | Type | Owner
4 -----+-----+-----+-----
5 public | phinxlog | table | jucabrera
```

Como o *change* é invocado também no *rollback*, não o utilize em *migrations* de inserções de dados por exemplo, apenas para *migrations* de alteração de estrutura de tabelas.

Caso você precise dar *rollback* em várias migrations passe o parametro **-t CÓDIGO_DA_MIGRATION_LIMITE**

Utilizando SQL

Você também tem a opção de escrever uma instrução em **SQL** utilizando o método **execute** para casos em você não queira (ou não possa) utilizar os métodos onde toda a lógica de execução de comandos no banco de dados está abstraída. Isso ocorre por exemplo quando você quer executar comandos relacionados a dados e não à estrutura (INSERT, UPDATE, DELETE, SELECT).

```
1 public function up()
2 {
3     $this->execute("INSERT INTO users (name, email, password) VALUES
4         ('teste', 'teste@teste.com', 'senha_segura')");
5 }
```

Notem que utilizei o método *up* e não o *change*. Conforme mencionei acima, se você escrever no *change* e der *rollback*, instrução será executada novamente duplicando os dados.



Alertas

- Cuidado com os comandos que você escreve e o ambiente em que está executando a *migration*. Há chances de você executar DELETE ou UPDATE sem WHERE em produção. Se você não quer correr esse risco, remova o bloco production do seu phinx.yml.
- Se você estiver utilizando Git (espero que sim) não esqueça de colocar o phinx.yml no .gitignore senão já sabe né? Seus dados de acesso ao banco de dados estarão expostos.
- Use nomes intuitivos para suas *migrations*, para que pelo nome você rapidamente consiga localizar o que você precisa.

Conclusão

Não disse que o phinx era simples? Você não precisa ter um conhecimento amplo dessa biblioteca para conseguir utilizá-la. Com isso o seu processo de atualização do banco de dados nos outros ambientes vai ficar muito mais simples. Além disso outro fator super importante é conseguir utilizá-lo no seu processo de integração contínua (ou Continuous Integration – CI).

Para saber mais consulte a documentação do phinx nos links:

<http://docs.phinx.org/en/latest/>

<https://book.cakephp.org/3.0/en/phinx.html>

« Anterior

Acessos remoto centralizado com Apache Guacamole

Próxima »

Semântica em HTML5: mais acessibilidade e SEO em seus projetos

📁 Categoria ◀ Desenvolvimento ◀ DevOps

Compartilhe esse post:



ABOUT AUTHOR



Juciellen Cabrera 4 posts

Programadora e Instrutora de PHP em 4Linux | Rankdone. Membro das comunidades PHPSP e PHPWomen. Uma das poucas mulheres com certificação ZCPE no Brasil.



[View all posts by this author →](#)



VOCE PODE GOSTAR TAMBEM



DevOps

Openshift: criação de cluster e deploy de uma aplicação

Esse post tem como objetivo criar um cluster de OpenShift Origin com um master e dois nós, a fim de fazer o deploy automático e definir o fluxo de integração

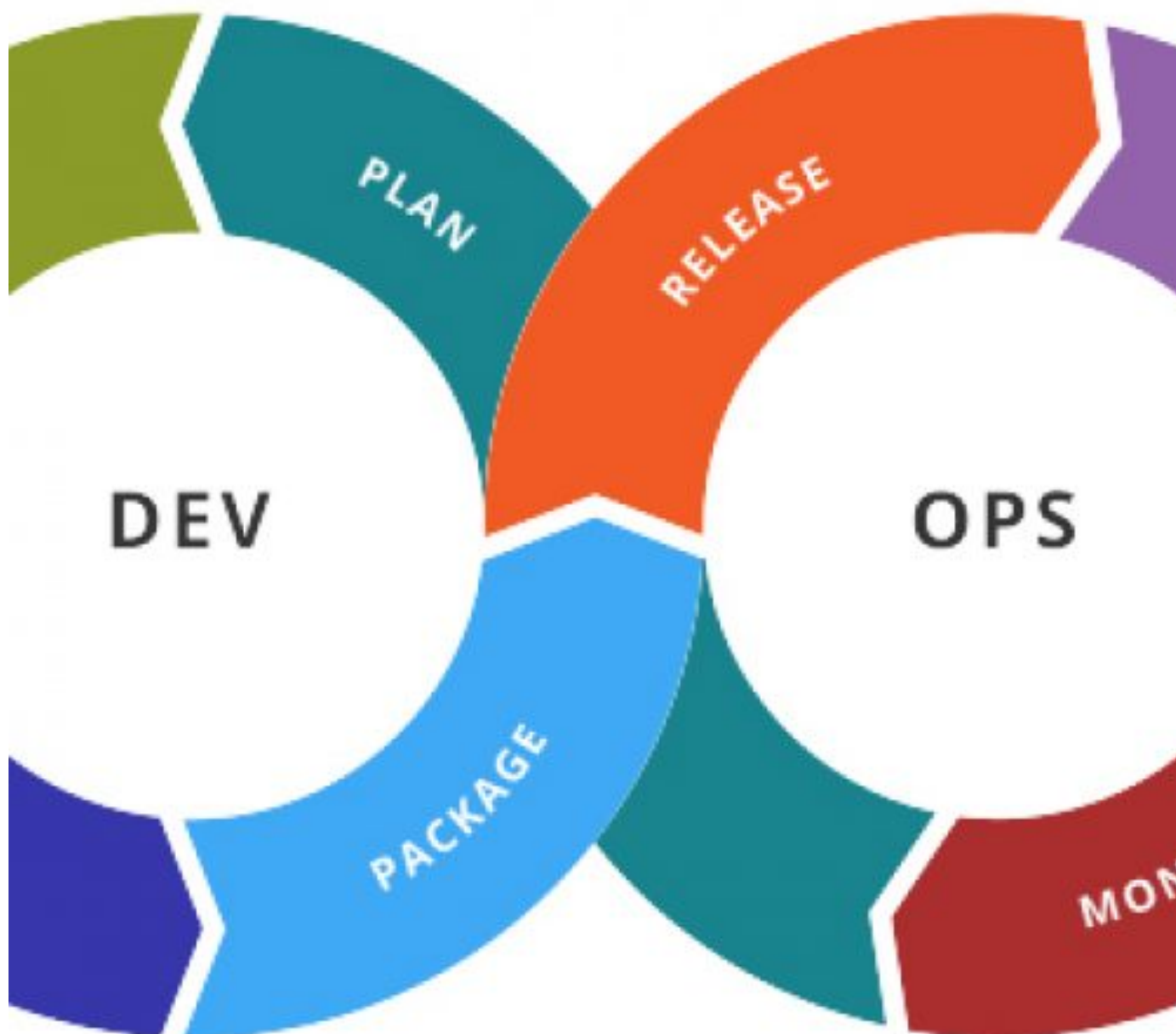




Minio — Tenha seu próprio S3 Storage

Minio é um storage de objetos, de alta performance e distribuído. A grande vantagem de usá-lo, reside em sua total compatibilidade com o Amazon S3. Neste artigo explico como subir





DevOps

DevOps – Como implementar essa poderosa estratégia em sua empresa?

DevOps é um termo que tem ganhado destaque no mundo corporativo. Considerado uma poderosa estratégia, surgiu da necessidade de agilizar entregas na área de tecnologia da informação, sempre buscando ações







4Linux - www.4linux.com.br 11 2125-4747 / 11 2125-4748 Rua Vergueiro, 3.057 Vila Mariana - São Paulo, SP CEP: 04101-300

