Laravel: prime nozioni, installazione, tool

Prerequisiti: buona conoscenza di PHP, in particolare a oggetti.

Alcune risorse per un rapido ripasso:

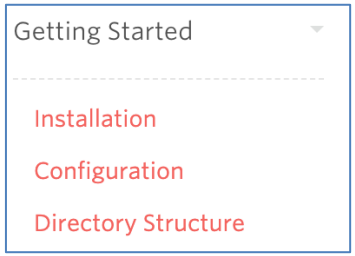
- https://www.ntu.edu.sg/home/ehchua/programming/#php
- https://developer.hyvor.com/tutorials/php/
- https://www.w3schools.com/php/
- https://www.php.net/manual/language.oop5.php
- https://www.tutorialspoint.com/php/php_object_oriented.htm
- https://www.tutorialspoint.com/php

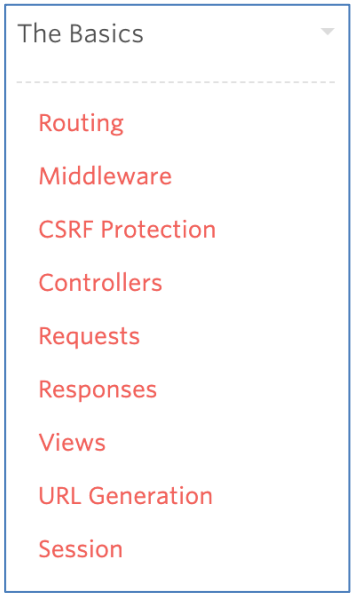
Queste note, comunque, sono piuttosto complete, anzi si noti che:

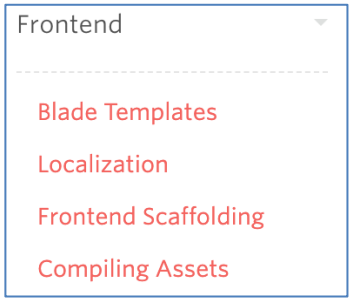
le slide con questo sfondo azzurro non servono per lo studio, ma solo per approfondimenti.

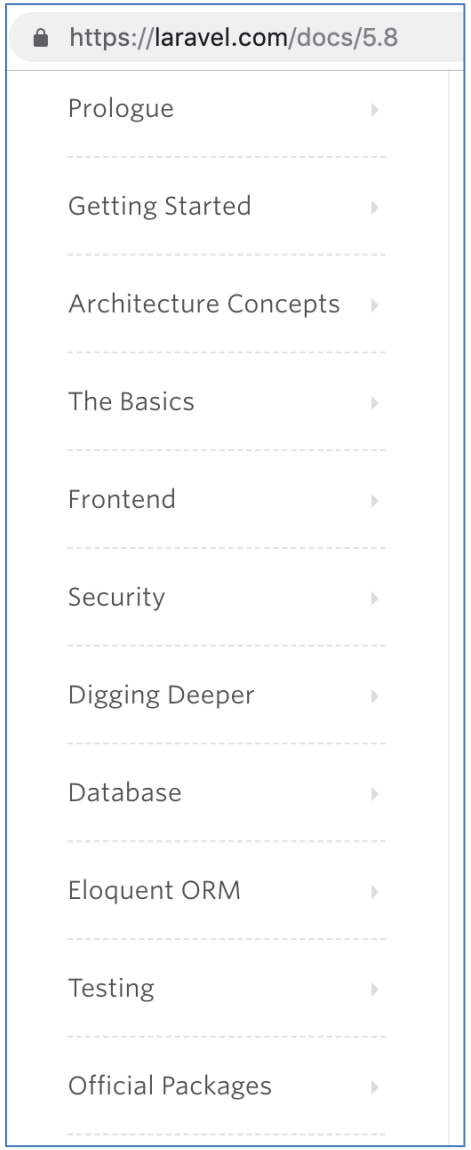
Laravel: risorse

- https://laravel.com/docs/ è il riferimento di base per Laravel
- Si tenga presente la sua dashboard laterale riprodotta qui a destra e, in particolare, per iniziare:









Requisiti per l'engine PHP

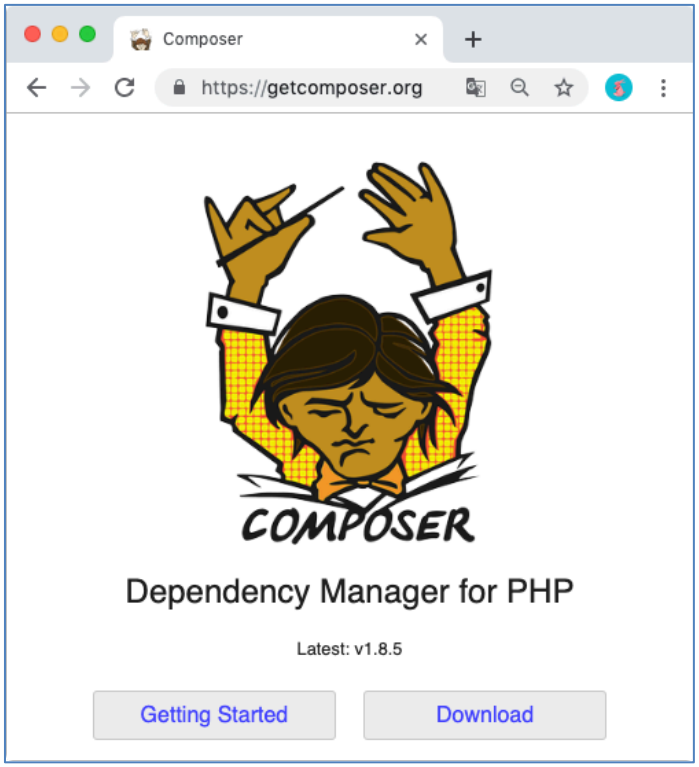
- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension
- BCMath PHP Extension

Sembrano tanti, ma dovrebbero essere soddisfatti da un'installazione PHP standard e comunque dovrebbero esserlo automaticamente grazie al tool *composer*

NB: se si installano ulteriori estensioni PHP (DLL) è possibile si debbano attivare con direttive nel file *php.ini*

Dipendenze PHP: il tool *composer*

- Tool (a riga di comando) per gestire dipendenze (tra librerie/pacchetti) in PHP
- Se un progetto PHP dipende da certe librerie (e queste da altre...), *composer*:
  - permette di dichiarare le dipendenze, anche a livello di versione
  - determina (tutti) i pacchetti necessari, li scarica e installa automaticamente



- Per default, *composer* opera a livello di singola applicazione PHP
- Gestisce cioè le dipendenze di quella app
- Opera installando le dipendenze (pacchetti) nella directory dell'app
- In alternativa, può operare (installare pacchetti necessari) globalmente
- NB: *globalmente* significa che i pacchetti vengono installati in *$HOME/.composer* cioè globalmente per lo specifico utente (non a livello di sistema)

Composer: installazione

```
$ php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# scarica lo script di boot di composer, omettiamo di controllare l'hash
$ php composer-setup.php      # installerà composer.phar nella directory corrente (/Users/gp in questo caso)
All settings correct for using Composer    Downloading...
Composer (version 1.8.5) successfully installed to: /Users/gp/composer.phar
Use it: php composer.phar
$ ls composer*   # come si vede (dal colore verde), composer.phar è già un eseguibile!
composer.phar composer-setup.php    # si può anche eliminare il file di setup composer-setup.php
```

- In Unix, il file *PHAR* (PHP archive) `composer.phar` si può lanciare direttamente, oppure lo si può passare a *PHP* come file da eseguire:

```
 $ ./composer.phar -V
Composer version 1.9.0 2019-08-02 20:55:32
$ php ./composer.phar -V
Composer version 1.9.0 2019-08-02 20:55:32
```

- *Composer* si può anche installare dandogli un nome diverso dal default (*composer.phar*), come qui sotto:

```
$ php composer-setup.php --filename=composer
Composer (version 1.8.5) successfully installed to: /Users/gp/composer
Use it: php compose # ovviamente questo comando va invocato dentro la cartella in cui si trova il file composer
```

Composer: installazione / 2

- Oltre al nome, si può anche scegliere la directory di installazione:

```
$ php composer-setup.php --install-dir=$HOME/bin --filename=composer
Composer (version 1.8.5) successfully installed to: /Users/gp/bin/composer
Use it: php /Users/gp/bin/composer
$ echo $PATH
/usr/local/bin:/usr/bin:/bin
$ export PATH=$HOME/bin:$PATH   # assicura che il composer appena generato sia nel PATH eseguibile
$ which -a composer
/usr/local/bin/composer  # questo composer era stato installato a livello di sistema (con apt / pacman / brew ...)
/Users/gp/bin/composer   # questo è il composer appena installato
```

- *Composer* è in grado di aggiornarsi:

```
 $ composer self-update
Updating to version 1.9.0 (stable channel).
   Downloading (100%)
Use composer self-update --rollback to return to version 1.8.5
~ $ composer -V
Composer version 1.9.0 2019-08-02 20:55:32
```

- *Composer* ha molti altri comandi (alcuni nella prossima slide)

```
$ composer
  Composer version 1.9.0 2019-08-02 20:55:32
Usage:  command [options] [arguments]
Options:
  -h, --help                     Display this help message
  -V, --version                  Display this application version
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
      --no-cache                 Prevent use of the cache
  -v|vv|vvv, --verbose           Increase the verbosity of messages:
Available commands:
  browse         Opens the package's repository URL or homepage in your browser.
  clear-cache    Clears composer's internal package cache.
  create-project Creates new project from a package into given directory.
  depends        Shows which packages cause the given package to be installed.
  diagnose       Diagnoses the system to identify common errors.
  dump-autoload  Dumps the autoloader.
  exec           Executes a vendored binary/script.
  global         Allows running commands in the global composer dir ($COMPOSER_HOME).
  help           Displays help for a command
  info           Shows information about packages.
  init           Creates a basic composer.json file in current directory.
  install        Installs project dependencies from the composer.lock file if present, or composer.json.
  list           Lists commands
  outdated       Shows installed packages that have updates available, including their latest version.
  prohibits      Shows which packages prevent the given package from being installed.
  remove         Removes a package from the require or require-dev.
  require        Adds required packages to your composer.json and installs them.
  run-script     Runs the scripts defined in composer.json.
  search         Searches for packages.
  self-update    Updates composer.phar to the latest version.
  suggests       Shows package suggestions.
  update         Upgrades dependencies to latest version according to composer.json, and updates composer.lock
  validate       Validates a composer.json and composer.lock.
```

composer e bash-completion

- Per *composer* è utile installare funzioni di auto-completamento, p.es. https://github.com/bramus/composer-autocomplete (ma in rete se ne trovano altri, più o meno completi)
- Una volta scaricato il file *composer-autocomplete*, lo si installa in */etc/bash_completion.d* (o */usr/local/etc/bash_completion.d* o *$BASH_COMPLETION_COMPAT_DIR*) e si avvia una nuova bash:

```
$ sudo cp composer-autocomplete /etc/bash_completion.d/
```

```
$   # nuova sessione Bash, d'ora in poi è attivo l'autocompletamento (tasto [TAB]) per il comando composer
$ composer [TAB]
about                depends              info                 run                  update
browse               dump-autoload        install              search               validate
...
config               home                 remove               suggests
create-project       i                    require              u
$ composer run-script [TAB]   # notevole il fatto che, se in composer.json, vi sono
il-mio-script                 # script dell'utente, il completamento automatico li trova e propone!
```

- Per disattivare il completamento (nelle successive sessioni *bash*):

```
my_app1 $ sudo rm /etc/bash_completion.d/composer-autocomplete
```

Installare... l'installer di Laravel

- Sotto si usa il *composer* (indipendentemente da dove sia stato installato) per installare il package (PHP) *laravel/installer* con le sue dipendenze
  - *laravel/installer* si installa global (meno utile installarlo per la specifica app)

```
$ composer global require laravel/installer      # global significa: installa nella directory
Changed current directory to /Users/gp/.composer # $COMPOSER_HOME (default ~/.composer)
Using version ^2.1 for laravel/installer
./composer.json has been created   # NB: nella directory $COMPOSER_HOME
Loading composer repositories with package information... Updating dependencies
Package operations: 12 installs, 0 updates, 0 removals
  - Installing symfony/process (v4.2.8): Loading from cache
  ...
  - Installing laravel/installer (v2.1.0): Loading from cache
symfony/contracts suggests installing psr/cache (When using the Cache contracts)
...
guzzlehttp/guzzle suggests installing psr/log (Required for using the Log middleware)
Writing lock file
Generating autoload files
```

- Nella dir di installazione .composer, compaiono l'eseguibile *laravel* e un link:

```
$ ls -ld ~/.composer/vendor/laravel/installer/laravel ~/.composer/vendor/bin/laravel
-rwxr-xr-x  ...      .composer/vendor/laravel/installer/laravel
lrwxr-xr-x  ...      .composer/vendor/bin/laravel -> ../laravel/installer/laravel
```

Usare (il tool) Laravel

- Si aggiunge la dir con (il link a) *laravel* al PATH, per semplificarne l'invocazione

```
~ $ ls .composer/vendor/bin/laravel
.composer/vendor/bin/laravel
~ $ export PATH=~/.composer/vendor/bin:$PATH
```

- Ora si può usare il tool *laravel* installato, per generare una applicazione Laravel, chiamata *my_app*, insieme con le dipendenze necessarie

```
$ laravel new my_app  # serve rete, per cabinet.laravel.com/latest.zip (v. oltre) e i pacchetti PHP (se non in cache)
Crafting application... Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 76 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Downloading (100%)
  - Installing doctrine/lexer (v1.0.1): Downloading (100%)
...
Generating optimized autoload files
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
> @php artisan key:generate -ansi      # artisan è il tool fondamentale di laravel, qui viene
Application key set successfully.      # invocato (automaticamente) per generare la chiave nel file .env
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
...
Package manifest generated successfully.
Application ready! Build something amazing.
```

- *my_app* vene installata in una sua directory chiamata *my_app*

Laravel: i ruoli di *composer*

- Ricapitolando: con *composer* si è installato *"global"* il pacchetto PHP *laravel/installer* (e altri pacchetti da cui questo dipende)
  - nella directory *$HOME/.composer/...* è quindi comparso il tool *laravel*:

```
~ $ .composer/vendor/laravel/installer/laravel
Laravel Installer 2.1.0
Usage:  command [options] [arguments]
Options:
  -h, --help            Display this help message
  -V, --version         Display this application version
  -v|vv|vvv, --verbose  Increase the verbosity of messages
Available commands:
  help  Displays help for a command
  list  Lists commands
  new   Create a new Laravel application
```

- In relazione a Laravel, e precisamente alle app come *my_app*, *composer* ha anche il ruolo di gestire (installare/aggiornare) le dipendenze dell'app da vari pacchetti/librerie PHP (poste sotto *vendor*):

```
my_app $ ls vendor/
autoload.php  doctrine       filp           league    nikic       phpdocumentor  psy          theseer
beyondcode    dragonmantank  fzaninotto     mockery   nunomaduro  phpoption      ramsey       tijsverkoyen
bin           egulias        hamcrest       monolog   opis        phpspec        sebastian    vlucas
composer      erusev         jakub-onderka  myclabs   paragonie   phpunit        swiftmailer  webmozart
dnoegel       fideloper      laravel        nesbot    phar-io     psr            symfony
```

laravel new e *composer*: lo scheletro dell'app

A questo punto, è utile approfondire quali passi compie il comando *laravel new an_app* per generare *an_app*:

1. scarica *http://cabinet.laravel.com/latest.zip*, lo "scheletro" dell'app, esclusa la directory *vendor* (con i pacchetti-dipendenze), ma con i file *composer.json* e *composer.lock*, che definiscono le dipendenze
2. decomprime *latest.zip* nella directory *an_app*

Per verifica, eseguiamo (1), (2) "a mano":

```
$ wget http://cabinet.laravel.com/latest.zip     # scarica template di app Laravel
$ mkdir an_app; cd an_app                        # crea directory dell'app
$ cd an_app
an_app $ unzip latest.zip

an_app $ ls      # troviamo la struttura di una app Laravel, con directory e file, ma senza vendor
app          composer.json  database      public     server.php  webpack.mix.js
artisan      composer.lock  package.json   resources  storage     yarn.lock
bootstrap    config         phpunit.xml    routes     tests
```

laravel new: cosa fa

NB: le azioni compiute da *laravel new* si possono ricostruire dal codice (parte del tool *laravel*) in *~/.composer/vendor/laravel/installer/src/NewCommand.php*), che invoca ***composer install --no-scripts*** e 3 script definiti in <*dir-base-app-generata*>*/composer.json*:

```
// ~/.composer/vendor/laravel/installer/src/NewCommand.php
...
protected function execute(InputInterface $input, OutputInterface $output) {
        ...
        $output->writeln('Crafting application... ');
        $this->download($zipFile ...)->extract($zipFile, $directory)
             ->prepareWritableDirectories($directory,$output)->cleanUp($zipFile);
        $composer = $this->findComposer();
        $commands = [
            $composer.' install --no-scripts',
            $composer.' run-script post-root-package-install',
            $composer.' run-script post-create-project-cmd',
            $composer.' run-script post-autoload-dump',
        ];
...
```

```
// file <directory-base-app-generata>/composer.json: descrive gli script possibile argomento di composer run-script
...
{   ...
    "scripts": {
        "post-root-package-install": [ "@php -r \"file_exists('.env') || copy('.env.example', '.env');\"" ],
        "post-create-project-cmd": [ "@php artisan key:generate --ansi" ],
        "post-autoload-dump": [
            "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
            "@php artisan package:discover --ansi" ]
}   }
```

laravel new e composer: le dipendenze

3. Dunque, ora *laravel new* userebbe *composer install --no-scripts* (che opera secondo le specifiche in *composer.lock* e *composer.json*), per scaricare i pacchetti-dipendenze (quelli da cui dipende l'app Laravel)

   Anche questo passo (3) si può eseguire "a mano":

```
an_app $ composer help install
  install     Installs project dependencies from composer.lock if present, or composer.json
an_app $ composer install --no-scripts
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 80 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Loading from cache
  ...
  - Installing phpunit/phpunit (7.5.15): Loading from cache
symfony/routing suggests installing doctrine/annotations (For the annotation loader)
...
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
Generating optimized autoload files
```

- I pacchetti installati compaiono nella (nuova) directory *vendor*
- Inoltre, in *vendor/composer*, compaiono gli "autoload files" generati (v. oltre per l'*autoloading*, funzionalità di caricamento implicito dei file delle classi PHP)

I suggerimenti di *composer*

Come si vede, dopo avere installato (dalla cache, o dai repository remoti) i pacchetti specificati in *composer.lock* e *composer.json, composer* suggerisce di installarne degli altri, eventualmente specificando la versione:

```
an_app $ composer install --no-scripts
Loading composer repositories with package information. Installing dependencies ... from lock file
Package operations: 80 installs, 0 updates, 0 removals
  ... Installing ...
symfony/routing suggests installing doctrine/annotations (For using the annotation loader)
...
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
```

Per installare, p.es. *phpunit/php-invoker*, va integrato *composer.json*, con:

```
an_app $ composer help require
  require        Adds required packages to your composer.json and installs them

an_app $ composer require phpunit/php-invoker:2.0
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing phpunit/php-invoker (2.0.0): Downloading (100%)
Writing lock file
Generating optimized autoload files     # NB: come si vede qui, dopo l'installazione (con install o require)
...                                     # di un pacchetto, composer rigenera gli autoload files in vendor/composer
```

NB: questo è solo un esperimento; di norma i *suggests* si applicano dopo *laravel new*

Composer e autoloading

Torniamo sull'installazione dei pacchetti da cui dipende l'app Laravel:

```
an_app $ composer install --no-scripts
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 80 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Loading from cache
  ...
Generating optimized autoload files
```

- I pacchetti installati compaiono nella (nuova) directory *vendor*
- Inoltre, in *vendor/composer*, compaiono gli "autoload files" generati
- j

laravel new: cosa fa / 2

Ricordiamo che in *~/.composer/vendor/laravel/installer/src/NewCommand.php* (nel metodo *execute()*), *laravel new* conclude invocando tre script, che si ritrovano alla fine del file *composer.json*, nella directory base della app generata con *laravel new*:

```
// ~/.composer/vendor/laravel/installer/src/NewCommand.php
...
protected function execute(InputInterface $input,
                           OutputInterface $output) {
        ...
        $commands = [
            $composer.' install --no-scripts',
            $composer.' run-script post-root-package-install',
            $composer.' run-script post-create-project-cmd',
            $composer.' run-script post-autoload-dump',
        ];
...
```

```
// file <dir-base-app>/composer.json

{
    "name": "laravel/laravel",
...
    "autoload": {
        "psr-4": {
            "App\\": "app/"
        }, ...
    },
...
    "scripts": {
        "post-root-package-install": [
            "@php -r \"file_exists('.env') || copy('.env.example', '.env');\"" ],
        "post-create-project-cmd": [
            "@php artisan key:generate --ansi" ],
        "post-autoload-dump": [
            "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
            "@php artisan package:discover --ansi" ]
}   }
```

laravel new: ultimi ritocchi

Con questi tre script, completiamo il nostro *laravel new* "a mano":

4. *post-root-package-install*: via *php* crea il file di configurazione *.env* (qui invochiamo direttamente il corpo dello script)

```
an_app $ php -r "file_exists('.env') || copy('.env.example', '.env');"
```

5. *post-create-project-cmd*: con il tool *artisan* dell'app, genera una chiave e la memorizza nel file *.env* (qui si invoca direttamente il corpo dello script)

```
an_app $ php artisan key:generate --ansi
Application key set successfully.
```

6. infine, si lancia a mano l'ultimo script (stavolta non direttamente il suo corpo): così che il metodo statico *Illuminate\Foundation\ComposerScripts::postAutoloadDump* possa essere invocato facilmente:

```
// file <dir-base-app>/composer.json
...
    "scripts": { ...
        "post-autoload-dump": [
            "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
            "@php artisan package:discover --ansi" ]
```

```
an_app $ composer run-script post-autoload-dump
```

Quest'ultimo passo è relativamente involuto (ma non è indispensabile capirlo, per i nostri scopi). Cercheremo di illustrarlo nella prossima slide.

Laravel scopre i pacchetti installati

Consideriamo l'ultimo script lanciato da *laravel new*:

```
an_app $ composer run-script post-autoload-dump
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
```

Lo script analizza i pacchetti installati, per renderli disponibili al codice Laravel.

A questo scopo, occorre registrare (nelle sedi opportune, cache, etc.) ciascuno di questi pacchetti con i suoi componenti detti *service provider* e *façade*.

Lo script esegue due azioni :

1. invoca il metodo *postAutoloadDump*, che cancella (da file e cache) ogni informazione su package precedentemente registrati
2. esegue il comando *artisan package:discover* per

Per dettagli: https://divinglaravel.com/laravels-package-auto-discovery

Laravel e composer / 2

`composer dump-autoload` (da https://getcomposer.org/doc/03-cli.md#dump-autoload-dumpautoload)

If you need to update the autoloader (e.g. because of new classes in a classmap package), you can use dump-autoload to do that without having to go through an install or update.

Additionally, it can dump an optimized autoloader that converts PSR-4 packages into classmap ones for performance reasons. In large applications with many classes, the autoloader can take up a substantial portion of every request's time. Using classmaps is less convenient in development, but using this option you can still use PSR-4 for convenience and classmaps for performance.

`composer dump-autoload --optimize (-o)`: Convert PSR-0/4 autoloading to classmap to get a faster autoloader. Recommended especially for production, but can take a bit of time to run.

- con

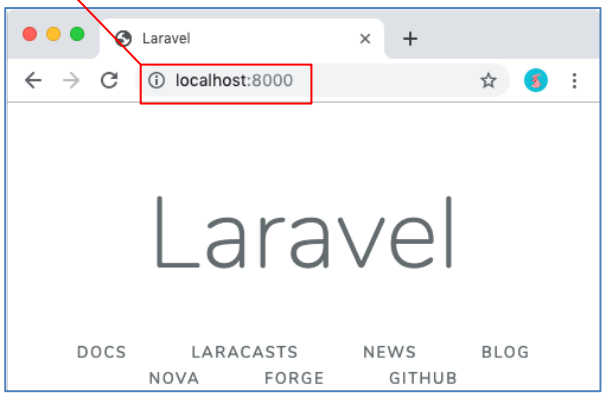
Servire app Laravel

```
$ cd my_app              # si entra nella directory della app Laravel generata

My_app $ file artisan    # in essa c'è il file artisan, script PHP di gestione della app
Artisan: a /usr/bin/env php script text executable, ASCII text

my_app $ php artisan     # ... help con numerosissimi comand, tra cui serve
...
```

- Ora è già possibile servire l'app (template) generata a clienti locali (si presume si usi una macchina di sviluppo), con *php artisan serve* :

```
my_app $ php artisan serve   # avvia un server, ora si può puntare il browser a localhost:8000
Laravel development server started: <http://127.0.0.1:8000>
[Thu May 16 03:08:18 2019] 127.0.0.1:56160 [200]: /favicon.ico
```

- NB: questo "web server" *artisan*, implementato in PHP, è adatto alla fase di sviluppo, ma non "in produzione"
- *artisan* è il tool fondamentale per Laravel, con i tantissimi comandi e sottocomandi



Servire altre app Laravel

- Per passare a generare un'altra app, basta cambiare directory (presumibilmente la *home*)
- E usare di nuovo il wizard *laravel new*
  - stavolta i pacchetti che compongono il framework sono già in *cache*, quindi non verranno scaricati

```
$ laravel new my_app
Crafting application...
Package operations: 76 installs, 0 updates, 0 removals
  - Installing doctrine/inflector (v1.3.0): Loading from cache
...
Generating optimized autoload files
...
Application key set successfully.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
...
Package manifest generated successfully.
Application ready! Build something amazing
```

- Per servire questa app *insieme* alla precedente, va impiegato un altro port

```
$ cd my_app
my_app $ php artisan serve -port=8001
Laravel server started:
<http://127.0.0.1:8001> ...
```

Servire direttamente da PHP

- Il comando *artisan serve*, oltre a effettuare qualche verifica e mostrare dei diagnostici, non fa altro che invocare l'interprete PHP sul file *server.php*, generato automaticamente dal tool *laravel new*, nella directory base (p.es. *my_app*) della applicazione
- Quindi, un secondo (ed equivalente) modo di "servire" un'applicazione Laravel ai clienti, è invocare dalla shell:

```
$ cd my_app
my_app $ php -S localhost:8001 server.php
Listening on http://localhost:8001
Document root is /Users/gp/laracode/my_app
Press Ctrl-C to quit.
```

- Una terza possibilità è usare un web server con un modulo PHP, p.es. Apache (ne parleremo più avanti)

Il tool *artisan*

Si è visto che nella directory di una app Laravel è presente *artisan,* script PHP, di cui si sono usati i comandi *serve*, *key:generate*, *package:discover*

Si tratta di un tool per lo sviluppatore Laravel, ricchissimo di funzionalità:

```
app8 $ php artisan
Laravel Framework 5.8.33
Usage:  command [options] [arguments]
Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
Available commands:
  clear-compiled       Remove the compiled class file
  down                 Put the application into maintenance mode
  dump-server          Start the dump server to collect dump information.
  env                  Display the current framework environment
  help                 Displays help for a command
  inspire              Display an inspiring quote
  list                 Lists commands
  migrate              Run the database migrations
  optimize             Cache the framework bootstrap files
  preset               Swap the front-end scaffolding for the application
  serve                Serve the application on the PHP development server
  tinker               Interact with your application
  up                   Bring the application out of maintenance mode
 app
  app:name             Set the application namespace
```

```
 auth
  auth:clear-resets    Flush expired password reset tokens
 cache
  cache:clear          Flush the application cache
  cache:forget         Remove an item from the cache
  cache:table          Create a migration for the cache database
table
 config
  config:cache         Create a cache file for faster ...
  config:clear         Remove the configuration cache file
 db
  db:seed              Seed the database with records
 event
  event:cache          Discover and cache application's events ...
  event:clear          Clear all cached events and listeners
  event:generate       Generate missing events and listeners ...
  event:list           List the application's events and ...
 key
  key:generate         Set the application key
 make
  make:auth            Scaffold basic login and registration ...
  make:channel         Create a new channel class
  make:command         Create a new Artisan command
  make:controller      Create a new controller class
  make:event           Create a new event class
  make:exception       Create a new custom exception class
  make:factory         Create a new model factory
  make:job             Create a new job class
  make:listener        Create a new event listener class
  make:mail            Create a new email class
  make:middleware      Create a new middleware class
  make:migration       Create a new migration file
  make:model           Create a new Eloquent model class
  make:notification    Create a new notification class
  make:observer        Create a new observer class
  make:policy          Create a new policy class
  make:provider        Create a new service provider class
  make:request         Create a new form request class
  make:resource        Create a new resource
  make:rule            Create a new validation rule
  make:seeder          Create a new seeder class
  make:test            Create a new test class
 migrate
  migrate:fresh        Drop all tables and re-run all
migrations
  migrate:install      Create the migration repository
  migrate:refresh      Reset and re-run all migrations
  migrate:reset        Rollback all database migrations
  migrate:rollback     Rollback the last database migration
  migrate:status       Show the status of each migration
 notifications
  notifications:table  Create migration for notifications table
 optimize
  optimize:clear       Remove the cached bootstrap files
 package
  package:discover     Rebuild the cached package manifest
 queue
  queue:failed         List all of the failed queue jobs
  queue:failed-table   Create a migration for the failed ...
  queue:flush          Flush all of the failed queue jobs
  queue:forget         Delete a failed queue job
  queue:listen         Listen to a given queue
  queue:restart        Restart queue worker daemons after job
  queue:retry          Retry a failed queue job
  queue:table          Create a migration for jpbs queue
  queue:work           Start processing jobs on the queue
 route
  route:cache          Create a route cache file for faster ...
  route:clear          Remove the route cache file
  route:list           List all registered routes
 schedule
  schedule:run         Run the scheduled commands
 session
  session:table        Create a migration for session DB table
 storage
  storage:link         Create symbolic link for public/storage
 vendor
  vendor:publish       Publish publishable assets from packages
 view
  view:cache           Compile application's Blade templates
  view:clear           Clear all compiled view files
```

artisan e bash-completion

- Per *artisan* sarebbe assai utile il completamento automatico (col tasto [TAB]) sulla *bash*, data la miriade di comandi e opzioni
- Questa funzionalità è (quasi) nativa per *zsh*, mentre per *bash* si può usare lo script da https://gist.github.com/tuanpht/2c92f39c74f404ffc712c9078a384f39

  NB: aggiungere come ultimo rigo `complete -F _artisan artisan`
- Il file scaricato e modificato, chiamato p.es. *artisan.auto*, va posto in */etc/bash_completion.d/* o in */usr/local/etc/bash_completion.d/* o *$BASH_COMPLETION_COMPAT_DIR* (tipicamente nella *home*):

```
$ sudo cp artisan.auto /etc/bash_completion.d/
```

- Aperta una nuova shell, si verifica che il completamento è attivo:

```
$   # nuova sessione Bash, con autocompletamento per artisan attivo
$ complete -p artisan
complete -F _artisan artisan
```

- È utile avere un comando *artisan* da autocompletare...

artisan e bash-completion / 2

- Infine, occorre un comando di bash '*artisan*'; lo si ottiene con *alias* ed ecco attivo il completamento con tab:

```
my_app $ alias artisan='php artisan'   # si potrebbe inserire in .bashrc etc.
my_app $ artisan [TAB]                 # premendo il tasto TAB si attiva il completamento automatico
app:name             help               make:migration       migrate:rollback     route:cache
auth:clear-resets    inspire            make:model           migrate:status       route:clear
cache:clear          key:generate       make:notification    notifications:table  route:list
...
down                 make:event         make:seeder          queue:flush          up
dump-server          make:exception     make:test            queue:forget         vendor:publish
env                  make:factory       migrate              queue:listen         view:cache
event:cache          make:job           migrate:fresh        queue:restart        view:clear
...
my_app $ artisan migrat [TAB]    # il completamento funziona anche con sottocomandi e opzioni
artisan migrat
migrate           migrate:install   migrate:reset     migrate:status
migrate:fresh     migrate:refresh   migrate:rollback
```

- Per disattivare l'autocompletion (mantenendo l'alias) nella sessione corrente:

```
my_app1 $ complete -r artisan
my_app1 $ complete -p artisan
-bash: complete: artisan: nessun completamento specificato
```

- Per disattivarlo in permanenza, in successive sessioni *bash*, e per ogni app:

```
my_app1 $ sudo rm /etc/bash_completion.d/artisan
```

artisan e bash-completion, con PHP

- Per *artisan* sarebbe assai utile il completamento automatico (col tasto [TAB]) sulla *bash*, data la miriade di comandi e opzioni
- Questa funzionalità è (quasi) nativa per *zsh*, mentre per *bash* si può ottenere con codice PHP, che va installato attraverso *composer*, localmente a ciascuna app
  - ciò perché è un complemento di *artisan*, che è installato localmente all'app

```
my_app $ composer require balping/artisan-bash-completion
...
  - Installing balping/artisan-bash-completion (v1.0.0): Loading from cache
```

- Il pacchetto installato contiene anche lo script di Bash per l'autocompletion; lo si installa in */etc* (o */usr/local/etc* o *$BASH_COMPLETION_COMPAT_DIR*) (NB: lo si fa una volta sola, per la prima app per cui è desiderato, qui è *my_app*):

```
my_app $ sudo cp vendor/balping/artisan-bash-completion/artisan /etc/bash_completion.d/
```

- Aperta una nuova shell, si verifica che il completamento è attivo:

```
my_app $  # nuova sessione Bash, con autocompletamento per artisan attivo
my_app $ complete -p artisan
complete -F _artisan artisan
```

- Serve però ancora un comando *artisan* da autocompletare...

artisan e bash-completion con PHP / 2

- Infine, occorre un comando di bash '*artisan*'; lo si ottiene con *alias* ed ecco attivo il completamento con tab:

```
my_app $ alias artisan='php artisan'   # si potrebbe inserire in .bashrc etc.
my_app $ artisan [TAB]                 # premendo il tasto TAB si attiva il completamento automatico
app:name             help               make:migration       migrate:rollback     route:cache
auth:clear-resets    inspire            make:model           migrate:status       route:clear
cache:clear          key:generate       make:notification    notifications:table  route:list
...
down                 make:event         make:seeder          queue:flush          up
dump-server          make:exception     make:test            queue:forget         vendor:publish
env                  make:factory       migrate              queue:listen         view:cache
event:cache          make:job           migrate:fresh        queue:restart        view:clear
...
my_app $ artisan migrat [TAB]    # il completamento funziona anche con sottocomandi e opzioni
artisan migrat
migrate           migrate:install   migrate:reset     migrate:status
migrate:fresh     migrate:refresh   migrate:rollback
```

- Per disattivare l'autocompletion (mantenendo l'alias) nella sessione corrente:

```
my_app1 $ complete -r artisan
my_app1 $ complete -p artisan
-bash: complete: artisan: nessun completamento specificato
```

- Per disattivarlo in permanenza, in successive sessioni *bash*, e per ogni app:

```
my_app1 $ sudo rm /etc/bash_completion.d/artisan
```