

Composer

Intro

Composer ဆိုတာ PHP အတွက် Dependency Management အတွက် အသုံးပြုနိုင်တဲ့ Tool တစ်ခု ဖြစ်ပါတယ်။ ကျွန်တော်တို့ Project တွေ ရေးတဲ့အခါမှာ မှီခို အသုံးပြုထားတဲ့ Library တွေကို Composer နဲ့ အသုံးပြုပြီး ကြေငြာပေးလို့ ရသလို သူ့အလိုလို အွန်လိုင်းက download ချ ထည့်သွင်းပေးတဲ့ နေရာမှာလဲ အသုံးပြုနိုင်ပါတယ်။

Dependency Management

Composer ဆိုတာက Package Manager မဟုတ်ပါဘူး၊ ဒါပေမယ့် Package တွေကို ကိုင်တွယ် ဖြေရှင်း ပေးတဲ့ tool တစ်ခုပါ။ Project တစ်ခုခြင်းစီအလိုက် မှီခိုထားရတဲ့ library တွေကို folder တစ်ခုထဲကို download ချပေးမယ်၊ လိုအပ်တာတွေ အကုန်ထည့်သွင်းပေးမှာ ဖြစ်ပါတယ်၊ (ပုံမှန် အားဖြင့်တော့ vendor ဆိုတဲ့ folder ထဲကို ထည့်ပေးပါတယ်)

Composer ရဲ့ အိုင်ဒီယာက အသစ်တော့ မဟုတ်ပါဘူး။ node.js တို့ ROR တို့မှာ ရှိတဲ့ npm, bundler တို့နဲ့ သဘောတရားခြင်း တူပါတယ်။


Composer အနေနဲ့ အောက်ပါ ပြဿနာတွေကို ဖြေရှင်းပေးနိုင်ပါတယ်

1. သင့်အနေနဲ့ project တစ်ခု တည်ဆောက်တဲ့အခါ မှီခိုထားတဲ့ libraries တွေ အများကြီး ရှိနေတယ်ဆိုရင်
2. သင့်ရဲ့ project မှာ အသုံးပြုထားတဲ့ libraries တွေဟာ အခြား Library တွေကို မှီခိုနေတယ်ဆိုရင်
3. သင့်ရဲ့ project မှာ သင်ကိုယ်တိုင် တည်ဆောက်ထားတဲ့ အရာတွေ အပေါ်မှာ မှီခိုဖို့ ကြေငြာချင်တယ် ဆိုရင်
4. composer အနေနဲ့ ဘယ် version ကို install လုပ်ရမယ်ဆိုတာ ရှာဖွေပေးမှာ ဖြစ်ပြီး install ပြုလုပ်ပေးပါလိမ့်မယ်။ (download ချပေးပါလိမ့်မယ်)

Declaring dependencies

ကျွန်တော်တို့ Project တစ်ခု ရေးတယ်ဆိုပါစို့။ အဲဒီ Project မှာ logging အတွက် Library တစ်ခုကို အသုံးပြုဖို့ လိုနေပါတယ်။ အဲဒီအတွက် monolog ဆိုတဲ့ library တစ်ခုကို အသုံးပြုဖို့ ရွေးချယ်လိုက်ပါတယ်။ (ကျွန်တော်တို့ composer အပေါ် မူတည်ပြီး အသုံးပြုလို့ရတဲ့ library တွေကို စုစည်းပေးထားတဲ့ site တစ်ခု ရှိပါတယ်။ အဲဒါကတော့ <https://packagist.org/> ပါ။

Create a new account | Login



Packagist

The PHP package archivist.

Submit Package

Packagist is the main **Composer** repository. It aggregates all sorts of PHP packages that are installable with Composer.

[Browse packages](#) or [submit your own](#).

Search packages...

Getting Started

Define Your Dependencies

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": ">=2.0.3"
  }
}
```

Install Composer In Your Project

Publishing Packages

Define Your Package

Put a file named `composer.json` at the root of your package, containing this information:

```
{
  "name": "your-vendor-name/package-name",
  "require": {
    "php": ">=5.3.0",
    "another-vendor/package": "1.*"
  }
}
```

This is the strictly minimal information you have to give.

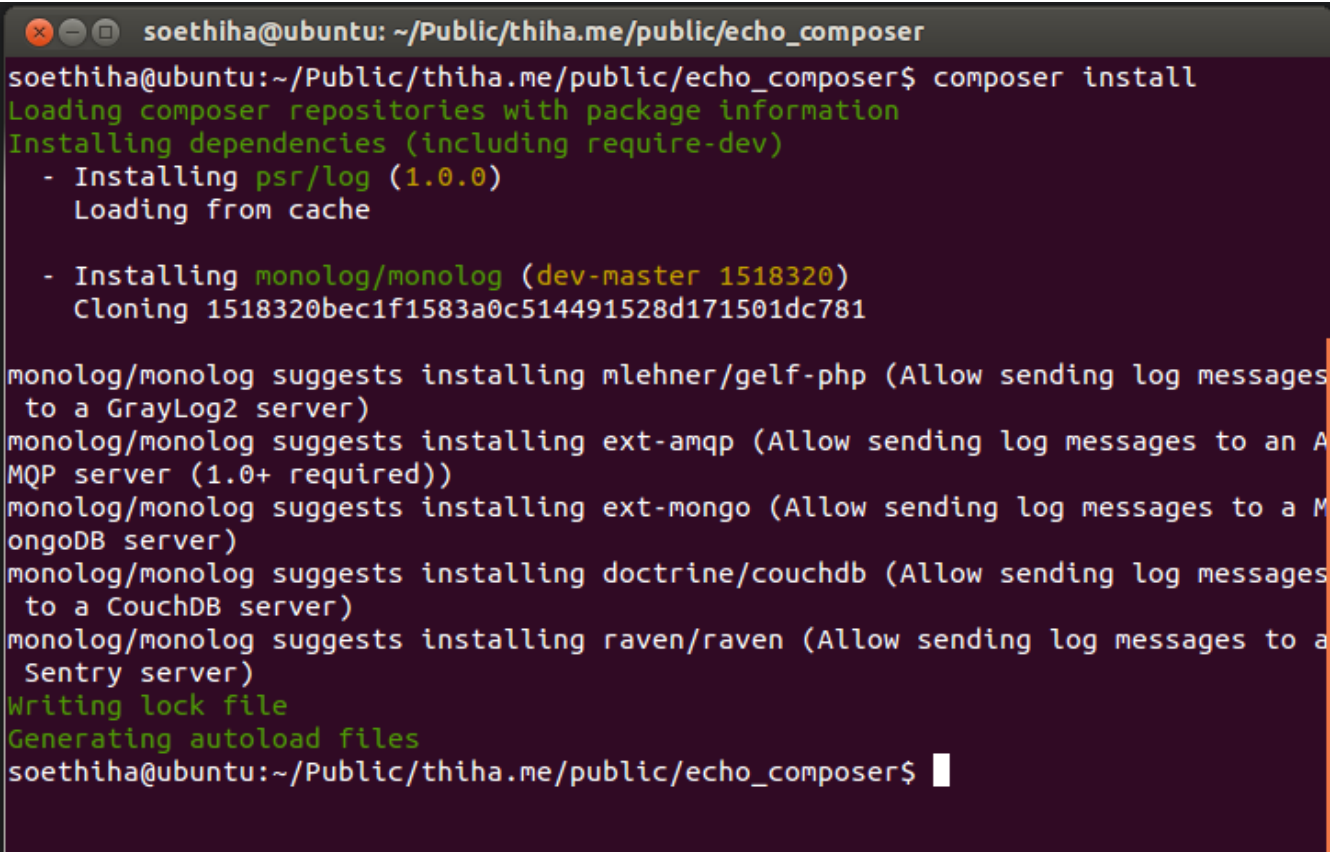
အဲဒီမှာ သွားရှာမယ်ဆိုရင် နောက်ဆုံးထွက်တဲ့ Version အမည်တွေက အစ သိရမှာ ဖြစ်ပါတယ်)

ကျွန်တော်တို့ ရှေ့ဆက်လိုက်ရအောင် ... monolog ကို အသုံးပြုဖို့အတွက် `composer.json` ဆိုတဲ့ file တစ်ခုကို တည်ဆောက်လိုက်ပြီး ကိုယ့် project အသုံးပြုမယ့် web server ထဲကို ထည့်ပေးလိုက်ဖို့ လိုပါတယ်။ အဲဒီအပြင် composer ကိုလဲ install လုပ်ဖို့လိုပါတယ်။ ဘယ်လို install လုပ်ရမယ်

ဆိုတာကိုတော့ <http://getcomposer.org/doc/00-intro.md> မှာ လေ့လာနိုင်ပါတယ်။ composer.json ဆိုတဲ့ file မှာ အောက်မှာ ရေးထားတဲ့အတိုင်း ရေးလိုက်ပါ။ ဒီနေရာမှာ တစ်ခု သတိထားဖို့ လိုတာက composer.json ဟာ JSON Format ကို လိုက်နာဖို့ လိုပါတယ်။ single quote နဲ့ ရေးလို့ မရပဲ double quote နဲ့သာ ရေးရမှာ ဖြစ်ပါတယ်။

```
{
    "require": {
        "monolog/monolog": "1.6.*@dev"
    }
}
```

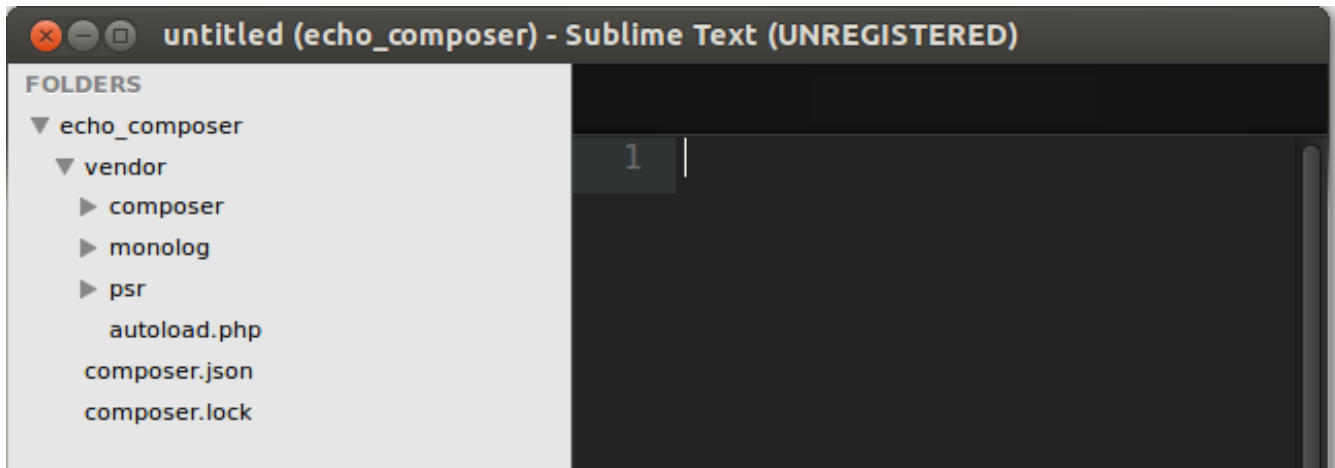
ပြီးရင် composer ကို install လုပ်ပေးဖို့ လိုပါမယ်။



```
soethiha@ubuntu: ~/Public/thiha.me/public/echo_composer
soethiha@ubuntu:~/Public/thiha.me/public/echo_composer$ composer install
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing psr/log (1.0.0)
  Loading from cache
- Installing monolog/monolog (dev-master 1518320)
  Cloning 1518320bec1f1583a0c514491528d171501dc781
monolog/monolog suggests installing mlehner/gelf-php (Allow sending log messages
to a GrayLog2 server)
monolog/monolog suggests installing ext-amqp (Allow sending log messages to an A
MQP server (1.0+ required))
monolog/monolog suggests installing ext-mongo (Allow sending log messages to a M
ongoDB server)
monolog/monolog suggests installing doctrine/couchdb (Allow sending log messages
to a CouchDB server)
monolog/monolog suggests installing raven/raven (Allow sending log messages to a
Sentry server)
Writing lock file
Generating autoload files
soethiha@ubuntu:~/Public/thiha.me/public/echo_composer$
```

composer.json file ရှိတဲ့ အထဲကို command line က သွားပြီး composer install လို့ ရိုက်လိုက်မယ် ဆိုရင် ပုံမှာပြထားတဲ့အတိုင်း monolog ကို install လုပ်သွားတာ တွေ့ရပါလိမ့်မယ်။ composer အနေနဲ့

monolog ကို download ချရုံသာမက autoload.php ဆိုတဲ့ file တစ်ခုကိုလဲ စီစဉ်ပေးပါလိမ့်မယ်။



ကိုယ့်အနေနဲ့ ကိုယ့်ရဲ့ project မှာ monolog ကို သုံးချင်တယ်ဆိုရင် autoload.php file ကို require လုပ်ပေးရုံပါပဲ။

ကျွန်တော် အပေါ်မှာ တည်ဆောက်ထားတဲ့ ပုံစံအရ echo_composer ဆိုတဲ့ folder ထဲမှာ composer.json ရှိနေပါတယ်။ ကျန်တာတွေကတော့ Composer က install လုပ်လိုက်လို့ အွန်လိုင်းက download ချလိုက်တာတွေ ဖြစ်ပါတယ်။ ကျွန်တော်တို့ monolog ကို သုံးချင်တယ်ဆိုရင် index.php မှာ ကျွန်တော် ရေးထားတဲ့အတိုင်း ရေးပြီး စမ်းလို့ရပါပြီ။

```
<?php

require "vendor/autoload.php";

use Monolog\Logger;
use Monolog\Handler\StreamHandler;

$log = new Logger('name');
$log->pushHandler(new StreamHandler(__DIR__ . '/log/project.log', Logger::WARNING));

$log->addWarning('You site has been under attacked!');
```

```
$log->addError('Stupid Error!');
```

```
?>
```

ကျွန်တော်တို့ composer.json မှာ ရေးခဲ့တဲ့ အပိုင်းကို ပြန်သွားကြရအောင်

```
{
    "require": {
        "monolog/monolog": "1.6.*@dev"
    }
}
```

require ဆိုတာကတော့ dependencies အတွက် ကိုယ့် project မှာ ဘယ် library တွေ လိုမယ်၊ ဘယ် Library တွေ သုံးမယ်ဆိုတာ ပြပါတယ်။ တစ်ဖက်မှာ package name ကို ရေးရပြီး နောက်တစ်ဖက်မှာ Package Version ကို ရေးပေးရပါတယ်။

1.0.* ဆိုရင် 1.0.0, 1.0.1, 1.0.2 စသည်ဖြင့် install လုပ်ပေးပါလို့ ဆိုလိုပါတယ်။ ကိုယ့်အနေနဲ့ package version ကို ပုံစံ လေးမျိုးနဲ့ သတ်မှတ်နိုင်ပါတယ်။

- **Exact version:** သင့်အနေနဲ့ သင့် project အတွက် dependency library ကို တိကျတဲ့ version သတ်မှတ်နိုင်ပါတယ်။ ဥပမာ - 1.0.2
- **Range:** comparison operators တွေဖြစ်တဲ့ >, >=, <, <=, != စတာတွေကို အသုံးပြုပြီး project မှာ လိုအပ်မယ့် dependency library ရဲ့ version တွေကို သတ်မှတ်နိုင်ပါတယ်။ ဥပမာ - >= 1.0။ သင့်အနေနဲ့ multiple range ကို သတ်မှတ်ချင်တယ် ဆိုရင်တော့ , နဲ့ ခံပြီး သတ်မှတ်နိုင်ပါတယ်။ ဥပမာ - >=1.0, < 2.0
- **Wildcard:** သင့်အနေနဲ့ * ဆိုတဲ့ wildcard pattern ကို အသုံးပြုပြီး သင်လိုတဲ့ package version ကို သတ်မှတ်နိုင်ပါတယ်။ ဥပမာ - 1.0.* ဆိုရင် >=1.0, <1.1 နဲ့ အဓိပ္ပါယ်တူပါတယ်။
- **Next Significant Release (Tilde Operator):** ဒါကတော့ Package version အသစ်ထွက်တိုင်း သုံးမယ်ဆိုတဲ့ သဘောဖြစ်ပါတယ်။ ~ ဆိုတဲ့ pattern နဲ့ သုံးရပါတယ်။

ပုံမှန်အားဖြင့် stable release တွေကိုသာ အသုံးပြုကြပြီး ကိုယ့်အနေနဲ့ Development Version, Alpha

Version, Beta Version စတာတွေကို သုံးချင်တယ်ဆိုရင် @dev, @beta စသည်ဖြင့် ထည့်ပြီး သုံးနိုင်ပါတယ်။

Autoloading

ကျွန်တော်တို့ အနေနဲ့ ကျွန်တော်တို့ လိုအပ်တဲ့ library တွေကို composer.json မှာ require keyword နဲ့ ကြေငြာလိုက်တယ် ဆိုရင် autoload.php ဆိုတဲ့ file တစ်ခုကို composer က ဖန်တီးပေးပါလိမ့်မယ်။ ကျွန်တော်တို့ project ရေးတဲ့အခါမှာ require 'vendor/autoload.php' ဆိုပြီး ရေးလိုက်တာနဲ့ ကျွန်တော်တို့ သုံးချင်တဲ့ library တွေကို သုံးလို့ ရပါပြီ။ ဒါပေမယ့် ကျွန်တော်တို့မှာ သူများ ရေးပြီးသား library တွေတင် မကပါဘူး၊ ကိုယ်တိုင်ရေးထားတဲ့ Library တွေလဲ ရှိနေနိုင်ပါတယ်။ အဲဒီအတွက် ကိုယ်ရေးထားတဲ့ Library တွေကိုလဲ composer.json မှာ ထည့်သွင်း ကြေငြာပေးဖို့ လိုပါမယ်။

ကိုယ့် Project မှာ ကိုယ်ရေးထားတဲ့ Library တွေကို ထည့်သွင်းကြေငြာပေးတဲ့ နေရာမှာ လိုအပ်ချက်အရ သုံးမျိုး ရှိနိုင်ပါတယ်။

1. **Namespace:** ကိုယ့် project မှာ namespace အနေနဲ့ autoload လုပ်ချင်တယ်ဆိုရင် PSR-0 standard အရ autoload လုပ်နိုင်ပါတယ်။
2. **Class:** ကိုယ့် project မှာ class ကို တစ်ခါတည်း ချိတ်ဆက်ပြီး autoload လုပ်နိုင်ပါတယ်။
3. **File:** ကိုယ့် project အတွက် လိုမယ့် file ကို တန်းပြီး autoload လုပ်နိုင်ပါတယ်။

အဲဒီ သုံးခုကို ကျွန်တော် နမူနာတစ်ခုနဲ့ ရှင်းပြသွားပါမယ်။

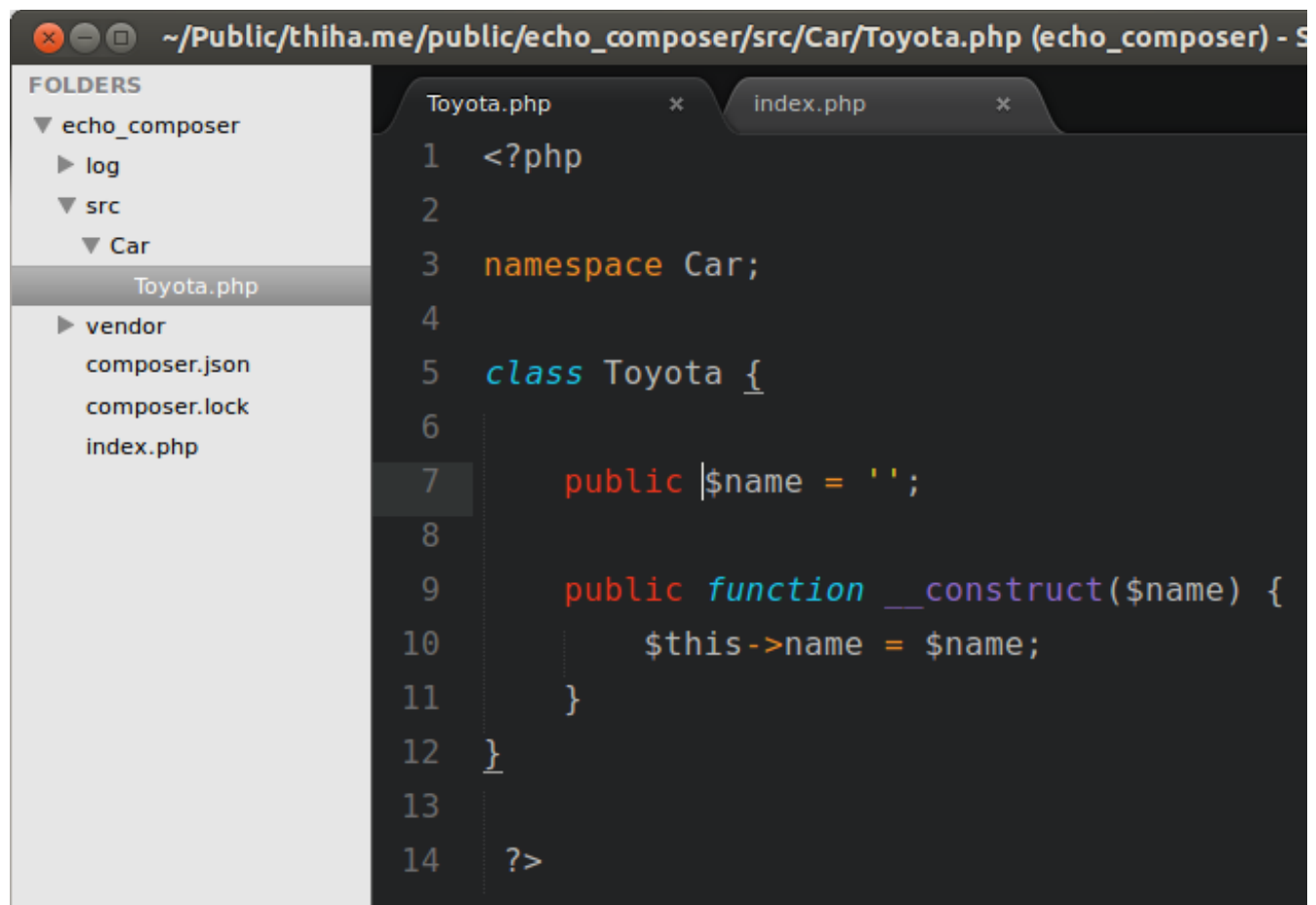
Namespace Autoload

ပထမ ဦးဆုံးနမူနာ ရေးကြည့်မယ် အပိုင်းကတော့ Namespace Autoload ပဲ ဖြစ်ပါတယ်။ Composer အနေနဲ့ Namespace ကို autoload လုပ်ဖို့အတွက် PSR-0 Standard ကို လိုက်နာထားပါတယ်။

ကျွန်တော်တို့ သုံးမယ့် Library က src/Car/Toyota.php ဆိုပါစို့ ...

```
{
    "require": {
        "monolog/monolog": "1.6.*@dev"
    },
    "autoload": {
        "psr-0": {"Car\\": "src/"}
    }
}
```

ကျွန်တော် အပေါ်မှာ ရေးထားတဲ့ ပုံစံအတိုင်း ရေးပေးဖို့ လိုပါမယ်။ folder structure ကတော့ အောက်မှာ ပြထားတဲ့ ပုံအတိုင်း ဖြစ်ပါတယ်။



src က folder name ပါ။ Car ဆိုတဲ့ folder က namespace ရဲ့ အမည်ဖြစ်ပါတယ်။ Toyota.php ကတော့ Class name အတွက် file ဖြစ်ပါတယ်။ composer.json မှာလဲ update လုပ်ပြီးပြီး namespace, class တွေလဲ ဆောက်ပြီးပြီး ဆိုရင်တော့ command line က သွားပြီး

`composer dump-autoload`

ဆိုပြီး autoload file ကို update လုပ်ပေးဖို့ လိုပါတယ်။ ပြီးရင်တော့ ကျွန်တော်တို့ Project မှာ Car/Toyota ကို သုံးလိုရပါပြီ။ ကျွန်တော် နမူနာအနေနဲ့ သုံးပြပါမယ်။

```
require "vendor/autoload.php";

use Car\Toyota;

$car = new Toyota('Belta');
echo $car->name;
```

run ကြည့်လိုက်မယ်ဆိုရင် Belta ဆိုပြီး ထွက်လာပါလိမ့်မယ်။ ကျွန်တော်တို့ အနေနဲ့ Car ဆိုတဲ့ Namespace အောက်မှာ Nissan, Mazda ဆိုတဲ့ Class တွေ ရှိလာမယ်ဆိုရင် Car ဆိုတဲ့ Folder အောက်မှာ ထည့်သွားရုံပါပဲ။ ဥပမာ အနေနဲ့ Car ဆိုတဲ့ folder အောက်မှာ Nissan.php ဆိုပြီး ထပ်ရေးကြည့်ပါမယ်။

```
<?php

namespace Car;

class Nissan {
    public $name = "";
```



```

    public function __construct($name) {
        $this->name = $name;
    }
}

?>

```

အဲဒီ Nissan ဆိုတဲ့ Class ကို သုံးချင်တယ်ဆိုရင် အောက်မှာ ပြထားတဲ့အတိုင်း ရေးပြီး သုံးနိုင်ပါတယ်။

```

require "vendor/autoload.php";

use \Car;

$toyota_car = new Car\Toyota('Belta');
$nissan_car = new Car\Nissan('AD Van');
echo $toyota_car->name . "<br />";
echo $nissan_car->name;

```

Use နေရာမှာ \Car ဆိုတဲ့ Namespace ကိုပဲ ကြော်ငြာထားပါတယ်၊ ပြီးတော့မှ Car\Toyota Car\Nissan ဆိုပြီး သုံးလို့ရပါတယ်။ run ကြည့်မယ်ဆိုရင်

Belta

AD Van

ဆိုပြီး result ထွက်လာပါလိမ့်မယ်။

Class Autoload

Namespace autoload အနေနဲ့လဲ သွားနိုင်သလို Class ကိုတစ်ခါတည်း ညွှန်းပြီးလဲ သုံးနိုင်ပါတယ်။
classmap လို့ composer မှာ သုံးပါတယ်။

ကျွန်တော်တို့ Project မှာ Namespace အနေနဲ့ သုံးတာ မဟုတ်ပဲ Class ကို တိုက်ရိုက်ညွှန်းပြီး သုံးတာလဲ ဖြစ်နိုင်ပါတယ်။ အဲဒီလို သုံးချင်တဲ့အခါ classmap နဲ့ composer.json မှာ ကြေငြာပြီး သုံးနိုင်ပါတယ်။

```
{
    "require": {
        "monolog/monolog": "1.6.*@dev"
    },
    "autoload": {
        "psr-0": {"Car\\": "src/"},
        "classmap": [
            "config"
        ]
    }
}
```

classmap ဆိုပြီး autoload မှာ ထပ်ဖြည့်ထားပါတယ်။ ဆိုလိုတဲ့ အဓိပ္ပါယ်ကတော့ config ဆိုတဲ့ folder ထဲက file တွေအားလုံးကို စစ်မယ်၊ အဲဒီ file တွေထဲက class တွေကို mapping လုပ်မယ်လို့ အဓိပ္ပါယ် ရပါတယ်။ ကျွန်တော်တို့ စမ်းသပ်ကြည့်ရအောင် ...

config ဆိုတဲ့ folder ထဲမှာ Database.php ဆိုတဲ့ file တစ်ခု တည်ဆောက်လိုက်ပါမယ်။ အဲဒီအထဲမှာတော့ ကျွန်တော် ရေးထားတဲ့အတိုင်း ရေးလိုက်ပါ။

```
<?php

class DB {
    public $dbname = "";

    public function __construct($dbname) {
        $this->dbname = $dbname;
    }
}

?>
```

အဲဒီလို ရေးပြီးရင် command line ကို သွားပြီး composer dump-autoload ဆိုပြီး ထပ်ခေါ်ပေးဖို့ လိုပါတယ်။

```
composer dump-autoload
```

အဲဒီလို ခေါ်လိုက်မှသာ လိုအပ်တဲ့ class တွေကို mapping လုပ်ပေးမှာ ဖြစ်ပါတယ်။ အဲဒီလို classmap လုပ်ပြီးပြီဆိုရင်တော့ သုံးလို့ ရပါပြီ။ ထုံးစံအတိုင်း index.php ကို သွားပြီး စမ်းကြည့်ရအောင် ...

```
$db = new DB('test');
echo $db->dbname;
```

run ကြည့်မယ်ဆိုရင် DB ဆိုတဲ့ class ကို autoload လုပ်ပြီးသား ဖြစ်သလို သုံးလို့ရနေတာ တွေ့ရပါလိမ့်မယ်။ ဒီနေရာမှာ တစ်ခု သတိထားဖို့လိုတာက config ဆိုတဲ့ folder ထဲမှာ တစ်ခုခု ထည့်မယ်ဆိုတာနဲ့ composer dump-autoload ပြန်လုပ်ပေးဖို့ လိုတာပါ။ အဲဒီလို အသစ်ထည့်တိုင်း classmap ပြန်မလုပ်ချင်ရင် Namespace Autolaod နဲ့ သုံးနိုင်ပါတယ်။

File Autoload

Namespace Autoload, Class Autoload ပြီးရင်တော့ File Autoload ပဲ ကျန်ပါတော့တယ်။ ဒါကတော့ ကျွန်တော်တို့ PHP မှာ သုံးနေကြအတိုင်းပဲ ဖြစ်ပါတယ်။ RebornCMS မှာဆိုရင်တော့ helper function တွေကို file autoload နဲ့ သုံးထားပါတယ်။ ကျွန်တော်တို့ Project မှာလဲ file autoload ကို စမ်းကြည့်ရအောင်

```
{
    "require": {
        "monolog/monolog": "1.6.*@dev"
    },
    "autoload": {
        "psr-0": {"Car\\": "src/"},
        "classmap": [
            "config"
        ],
        "files": [
            "helper/helper.php"
        ]
    }
}
```

files ဆိုပြီး autoload မှာ helper/helper.php ဆိုတဲ့ file ကို ချိတ်လိုက်ပါတယ်။ helper ဆိုတဲ့ folder တစ်ခုကို တည်ဆောက်လိုက်ပြီး helper.php ဆိုတဲ့ file တစ်ခုကို helper folder ထဲမှာ ဆောက်လိုက်ပါ။ ပြီးရင် ကျွန်တော် ရေးထားသလို ရေးလိုက်ပါ။

```
<?php

function Currency_Converter($amount) {
    return $amount * 980;
}

?>
```

အဲဒီလို ဆောက်ပြီးသွားပြီ ဆိုရင်တော့ ထုံးစံအတိုင်း composer dump-autoload ဆိုပြီး ပြန်လုပ်ပေးဖို့ လိုပါလိမ့်မယ်။ ပြီးရင် Currency_Converter ဆိုတဲ့ function ကို စမ်းသုံးကြည့်ပါမယ်။ index.php မှာ ထပ်ဖြည့်ပြီး ရေးကြည့်ပါမယ်။

```
$kyats = Currency_Converter(13000);
echo $kyats;
```

run ကြည့်မယ်ဆိုရင်တော့ kyats ရဲ့ value ကို ရိုက်ထုတ်ပြတာ တွေ့ရပါလိမ့်မယ်။

ကျွန်တော်တို့ ဒီလောက်ဆိုရင် Composer ရဲ့ Dependency Management အပိုင်းရယ်၊ autoloading အပိုင်းတွေ ဖြစ်တဲ့ namespace autoload, class autoload, file autoload အပိုင်းတွေကို သဘော ပေါက်နားလည်လိမ့်မယ်လို့ ယူဆပါတယ်။

ရှေ့ဆက်ပြီး Composer ကို အသုံးပြုပြီး Package တစ်ခု တည်ဆောက်တဲ့ ပုံစံကို ဆက်ရေးမှာ ဖြစ်ပါတယ်။