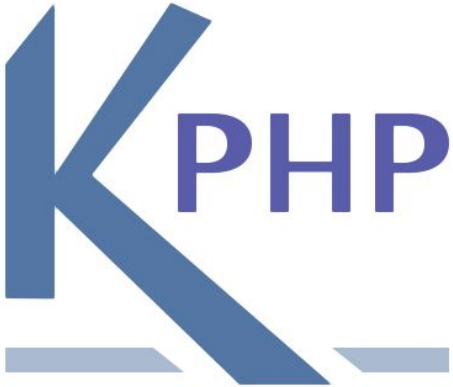
PHP scripts -> Release binaries

quasilyte @ Kazan PHP meetup 2021



Intro

- **Intro**
- Preparations
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- Using composer
- Useful recipes
- Rewriting Go app in KPHP

Releasing the PHP script

1. Prepare a phar archive
2. Ensure that client has PHP interpreter
3. Ensure that PHP version is compatible

Releasing the PHP script

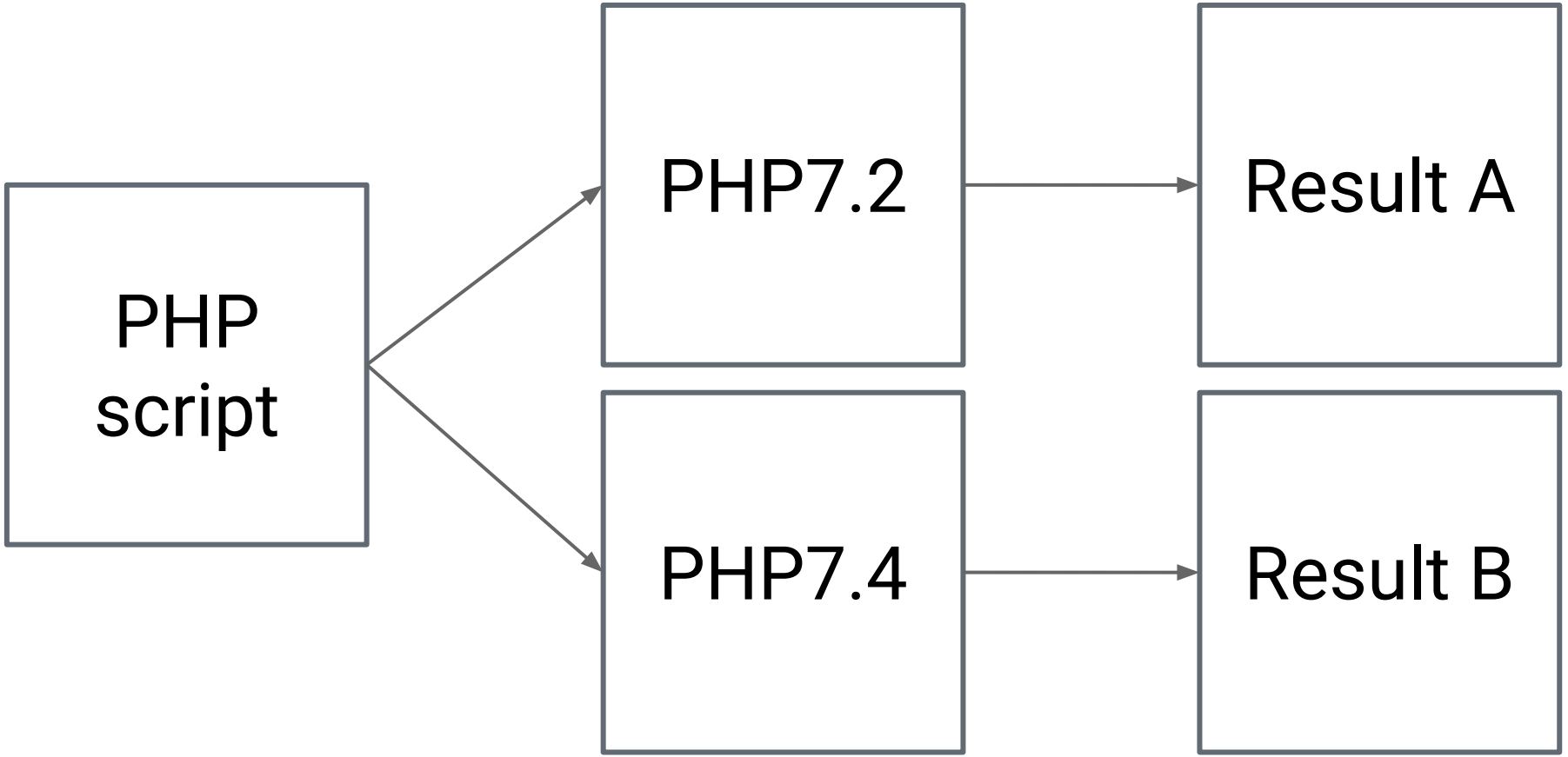
1. Prepare a phar archive
2. Ensure that client has PHP interpreter
3. Ensure that PHP version is compatible

OK for PHP dev tools, not so much for anything else

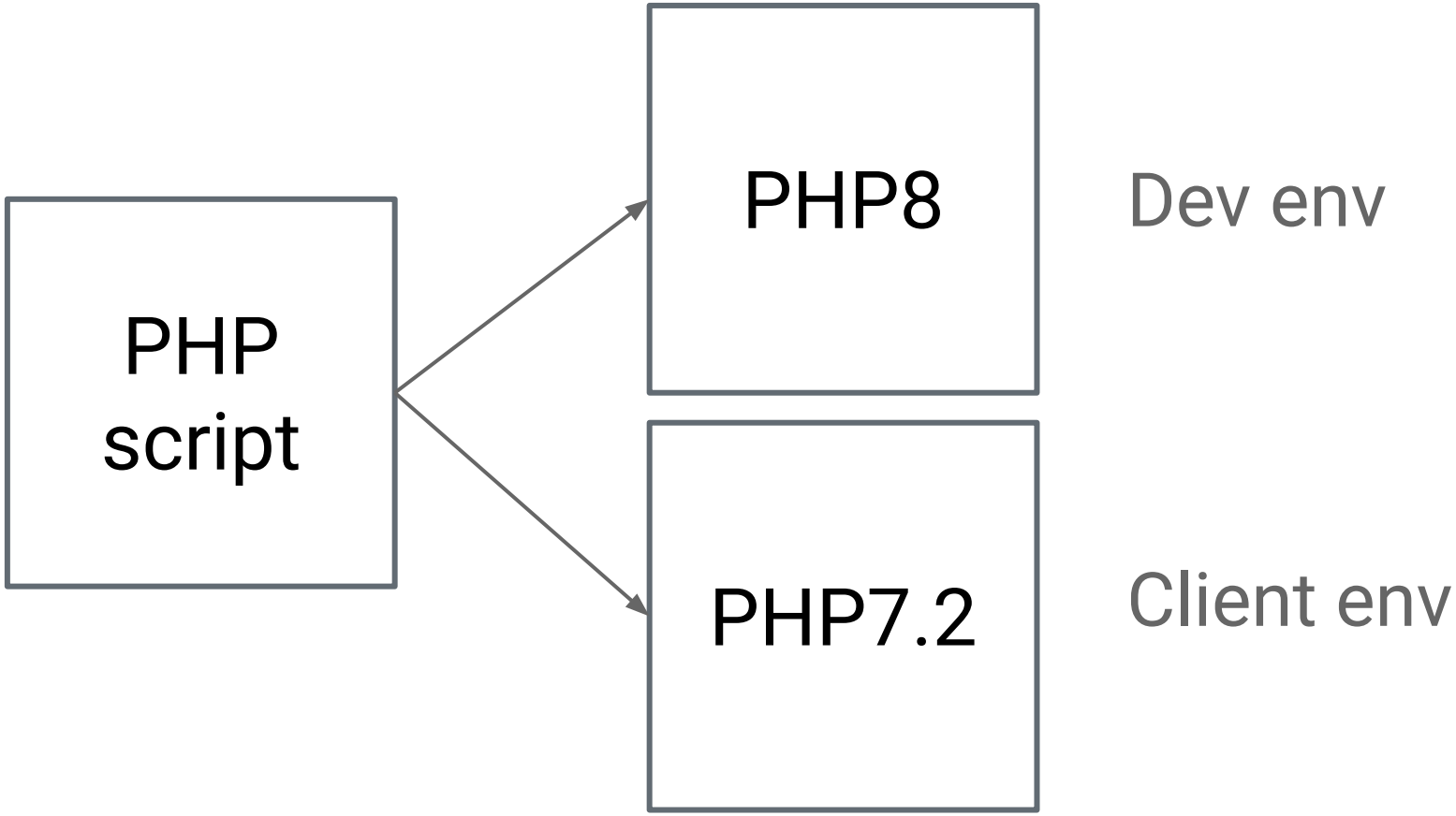
Releasing the PHP script

1. Prepare a phar archive
2. Ensure that client has PHP interpreter
3. Ensure that PHP version is compatible

Limits the features you can use within the minimal supported PHP version



Different results could happen



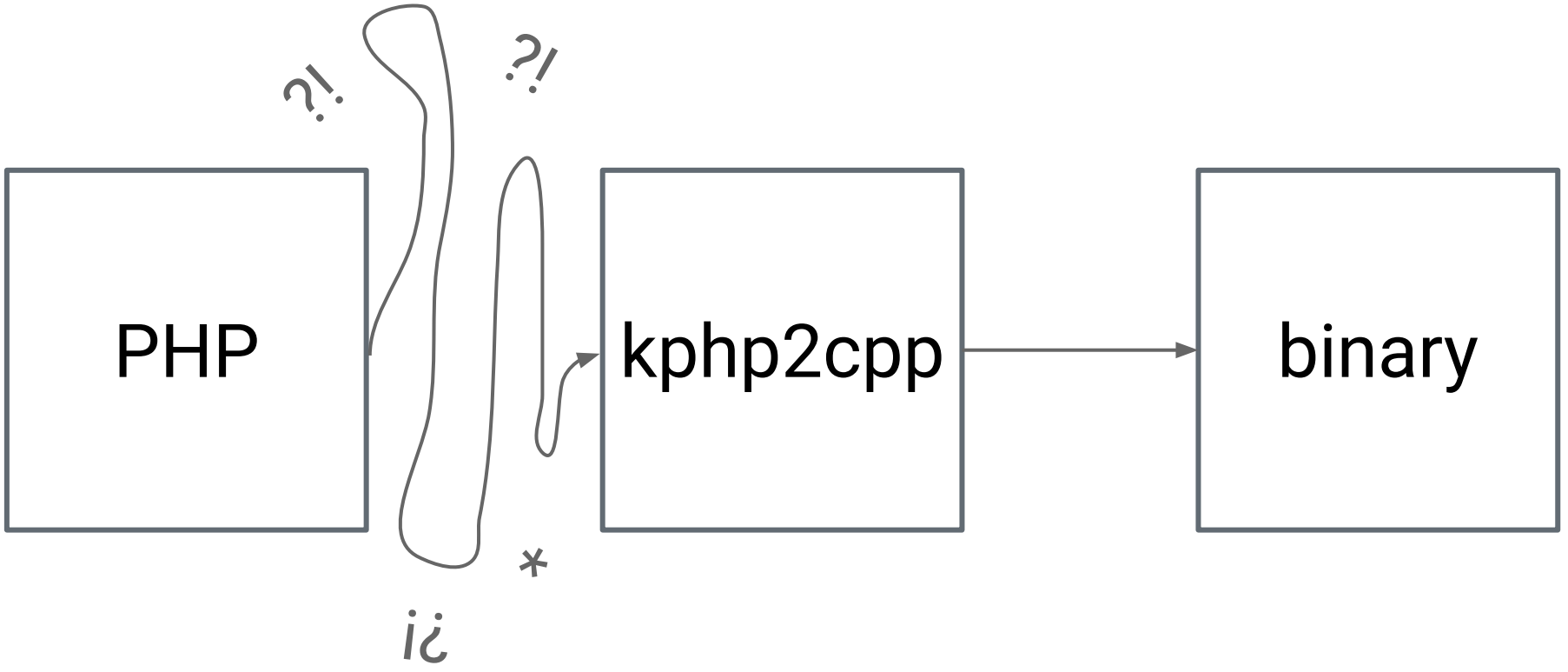
Can’t use all new PHP features...

Static binaries

- Portable among the chosen platform
- No (K)PHP version dependency

They’re easier to deploy.

In a perfect world...



In the real world...

In the near future

The algorithm

1. Take the PHP script
2. … ??? <- we’ll concentrate on this
3. Profit! (Get the executable binary)

Supported platforms

Supported platforms

1. Linux

Supported platforms

1. Linux

That’s about it. ¯_(ツ)_/¯

Maybe macOS support will be added in the future.

Preparations

- Intro
- **Preparations**
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- Using composer
- Useful recipes
- Rewriting Go app in KPHP

KPHP_ROOT

```
$ git clone https://github.com/VKCOM/kphp.git
```

Cloned repository root is $KPHP_ROOT

Install deb packages

See compiling KPHP from sources

Build KPHP toolchain

```
$ export KPHP_ROOT=$(pwd)
$ mkdir build
$ cmake ..
$ make -j16
```

Conventional “kphp” shortcut

```
$ ln -s obj/bin/kphp2cpp kphp
```

Located in $KPHP_ROOT after you build it

Making binaries smaller

- Intro
- Preparations
- **Making binaries smaller**
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- Using composer
- Useful recipes
- Rewriting Go app in KPHP

Compiling the “Hello World”

```
<?php
echo "hello, world!\n";
```

Compiling the “Hello World”

```
<?php
echo "hello, world!\n";
```

```
$ kphp --mode=cli ./hello.php
```

Compiling the “Hello World”

```
<?php
echo "hello, world!\n";
```

```
$ kphp --mode=cli ./hello.php
```

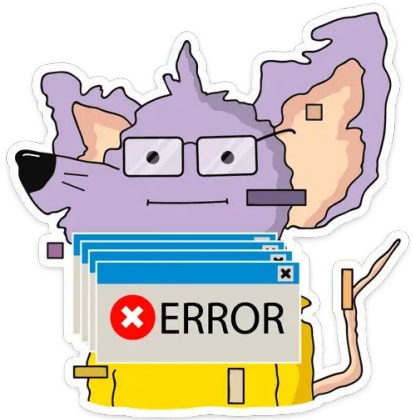
```
$ ./kphp_out/cli
hello, world!
```

Hello world size

```
$ du -h ./kphp_out/cli
...
```

Hello world size

```
$ du -h ./kphp_out/cli
25M
```



Hello world size

debug binary	25.0 M

Hello world size

debug binary	25.0 M
strip	8.3 M

Hello world size

debug binary	25.0 M
strip	8.3 M
strip + upx	2.8 M

Note: upx makes app start slightly slower

Hello world size

debug binary	25.0 M
strip	8.3 M
strip + upx	2.8 M

Stripped binary is an optimal solution

Fixing compilation errors

- Intro
- Preparations
- Making binaries smaller
- **Fixing compilation errors**
- Conditional compilation
- Dynamic dependencies
- Using composer
- Useful recipes
- Rewriting Go app in KPHP

Indirect member call

```
$this->$method(...)
```

Indirect member call

```
$this->$method(...)
```

```
if ($method === 'foo') {
  $this->foo(...)
} elseif ($method === 'bar') {
  $this->bar(...)
} elseif ...
```

Non-const require

```
require_once $filename;
```

Non-const require

```
require_once $filename;
```

```
if ($filename === 'foo.php') {
  require_once 'foo.php';
} elseif ($filename === 'bar.php') {
  require_once 'bar.php';
} elseif ...
```

Indirect new

```
return new $classname(...)
```

Indirect new

```
return new $classname(...)
```

```
if ($classname === 'Foo') {
  return new Foo(...);
} elseif ($method === 'Bar') {
  return new Bar(...);
} elseif ...
```

Conditional compilation

- Intro
- Preparations
- Making binaries smaller
- Fixing compilation errors
- **Conditional compilation**
- Dynamic dependencies
- Using composer
- Useful recipes
- Rewriting Go app in KPHP

#ifndef KPHP

```
#ifndef KPHP
  echo "PHP-only code\n";
if (false)
#endif
  echo "KPHP-only code\n";
```

KPHP parts

```
#ifndef KPHP
  echo "PHP-only code\n";
if (false)
#endif
  echo "KPHP-only code\n";
```

PHP parts

```
#ifndef KPHP
  echo "PHP-only code\n";
if (false)
#endif
  echo "KPHP-only code\n";
```

Running conditional code

```
$ php -f script.php
PHP-only code
```

Running conditional code

```
$ php -f script.php
PHP-only code
```

```
$ ./kphp/cli
KPHP-only code
```

Dynamic dependencies

- Intro
- Preparations
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- **Dynamic dependencies**
- Useful recipes
- Rewriting Go app in KPHP

KPHP shared libs

```
$ ldd ./kphp_out/cli
...
```

KPHP shared libs

```
$ ldd ./kphp_out/cli
linux-vdso.so.1    libc.so.6
libpthread.so.0    ld-linux-x86-64.so.2
libcrypto.so.1.1   libdl.so.2
librt.so.1         libgcc_s.so.1
libstdc++.so.6     libm.so.6
```

KPHP shared libs

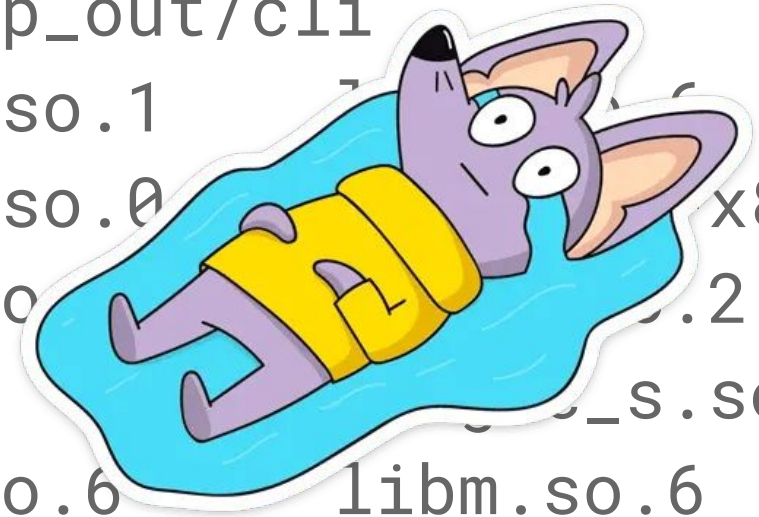
```
$ ldd ./kphp_out/cli
linux-vdso.so.1     libc.so.6
libpthread.so.0     ld-linux-x86-64.so.2
libcrypto.so.1.1    libdl.so.2
librt.so.1          libgcc_s.so.1
libstdc++.so.6      libm.so.6
```

11

+ pthread

KPHP shared libs

```
$ ldd ./kphp_out/cli
linux-vdso.so.1
libpthread.so.0                    x86-64.so.2
libcrypto.so                       .2
librt.so.1                         _s.so.1
libstdc++.so.6                     libm.so.6
```



+ pthread

Compiling with a spell

```
$ export KPHP_EXTRA_LDFLAGS='...'
$ kphp --mode=cli script.php
```

KPHP_EXTRA_LDFLAGS

```
export KPHP_EXTRA_LDFLAGS='
  -L${KPHP_ROOT}/objs/flex
  /opt/curl7600/lib/libcurl.a
  -l:libssl.a
  -l:libcrypto.a
  -l:librt.a
  -static-libstdc++
  -static-libgcc
  -ldl'
```

KPHP shared libs (2)

```
$ ldd ./kphp_out/cli
linux-vdso.so.1
libdl.so.2
libm.so.6
libc.so.6
ld-linux-x86-64.so.2
```

KPHP shared libs (2)

```
$ ldd ./kphp_out/cli
linux-vdso.so.1
libdl.so.2
libm.so.6
libc.so.6
ld-linux-x86-64.so.2
```

6

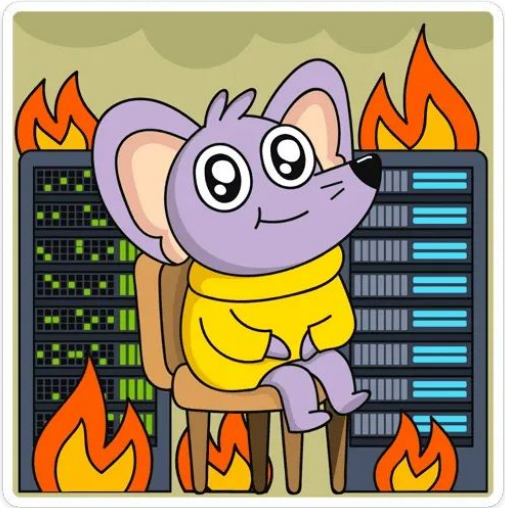
+ pthread

PHP shared libs

```
$ ldd $(which php)
linux-vdso.so.1      libpcre2-8.so.0
libargon2.so.1       libz.so.1
libresolv.so.2       libsodium.so.23
libutil.so.1         libc.so.6
librt.so.1           ld-linux-x86-64.so.2
libm.so.6            libicuuc.so.65
libdl.so.2           liblzma.so.5
libxml2.so.2         libicudata.so.65
libssl.so.1.1        libstdc++.so.6
libcrypto.so.1.1     libgcc_s.so.1
```

PHP shared libs

```
$ ldd $(which php)
linux-vdso.so.1          libpcre2-8.so.0
libargon2.so.1           libz.so.1
libresolv.so.2           libsodium.so.23
libutil.so.1             libc.so.6
librt.so.1               ld-linux-x86-64.so.2
libm.so.6                libicuuc.so.65
libdl.so.2               liblzma.so.5
libxml2.so.2             libicudata.so.65
libssl.so.1.1            libstdc++.so.6
libcrypto.so.1.1         libgcc_s.so.1
```



21

+ pthread

Intro

- Intro
- Preparations
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- **Using composer**
- Useful recipes
- Rewriting Go app in KPHP

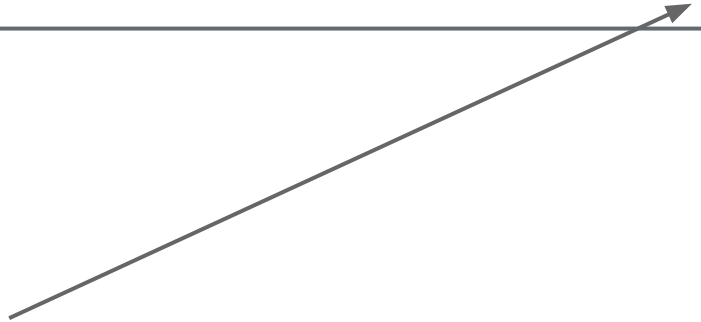
Initialize as a normal composer package

```
$ composer init ...
$ composer require ...
$ composer install ...
```

KPHP only cares about the vendor/ folder

The composer-root argument

```
$ kphp --composer-root $(pwd) ...
```



Path to a project root with composer.json

autoload.php

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

autoload.php

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

In PHP, it will initialize the composer autoloader

autoload.php

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

In KPHP, we mostly ignore this line (but autoload “files” will be executed **here**)

Class autoloading

Class autoloading

- autoload.psr4 supported

Class autoloading

- autoload.psr4 supported
- autoload.files supported

Class autoloading

- autoload.psr4 supported
- autoload.files supported
- autoload.classmap coming soom

The composer-root argument

```
$ kphp --composer-no-dev ...
```

By default, both require and require-dev are loaded; require-dev can be disabled

Useful recipes

- Intro
- Preparations
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- Using composer
- **Useful recipes**
- Rewriting Go app in KPHP

Building with clang

```
$ kphp --cxx clang++ ...
```

```
$ KPHP_CXX=--cxx kphp
```

Show all type errors

```
$ kphp --show-all-type-errors ...
```

```
$ KPHP_SHOW_ALL_TYPE_ERRORS=1 kphp
```

Display compilation progress

```
$ kphp --show-progress ...
```

```
$ KPHP_SHOW_PROGRESS=1 kphp
```

Other useful resources

- ktest - run PHPUnit tests with KPHP
- t.me/kphp_chat - unofficial KPHP group
- 3v3l.org - run code with 250+ PHP versions
- KPHP FAQ

Rewriting Go app in KPHP

- Intro
- Preparations
- Making binaries smaller
- Fixing compilation errors
- Conditional compilation
- Dynamic dependencies
- Using composer
- Useful recipes
- **Rewriting Go app in KPHP**

PHP scripts -> Release binaries

quasilyte @ Kazan PHP meetup 2021