Composer in Depth
Part 1

Nils Adermann
@naderman
Jordi Boggiano
@seldaek
Private Packagist
https://packagist.com
Dependency Management

- Dependency Management vs Package Management
- System state vs installation/update instructions

- Configuration Management
  - Tool to manage system state (puppet, salt, ansible, chef, ...)
  - Description of state (master.pp, top.sls, ...)
Working on “Libraries”
Publishing Packages

- README.md
  - What is it?
  - How do I use it?
  - How do I contribute? (Code of Conduct)

- Pick a License
  - SPDX
  - MIT, BSD, GPL
  - "proprietary"
Publishing Packages

- CHANGELOG.md
  - BC breaks
    - If necessary create UPGRADE.md
  - changes
  - bugfixes
  - new features
Semantic Versioning

x.y.z
(BC-break).(new functionality).(bug fix)

http://semver.org/
Semantic Versioning

Promise of Compatibility

\textbf{X.Y.Z}

- Must be used consistently
  
  Dare to increment \textbf{X}!

- Only valuable if BC/Compatibility promise formalized
  - See \url{http://symfony.com/doc/current/contributing/code/bc.html}
  - Document in Changelog
Continuous Integration for Libraries

- Multiple runs
  - `composer install` from lock file
  - `composer update` for latest deps
  - `composer update --prefer-lowest --prefer-stable` for oldest (stable) deps

- Potentially multiple composer.json files with different platform configurations
  - `COMPOSER=composer-customer1.json php composer.phar update`
  - `COMPOSER=composer-customer1.json php composer.phar install`
  - Don’t use this except for testing
Working on “Applications”
Simple Versioning

- There are no other packages depending on yours
- BC - for Composer consumption - doesn’t matter
- Options:
  - Don’t use versions at all, rely on your VCS
  - Increment a single integer
  - Use semver if you ship the application
How to update?

- "composer update"
  - no isolation of problems unless run very frequently

- "composer update <package...>
  - explicit conscious updates

- "composer update --dry-run [<package...>]
  - Understanding and preparing effects of updates
  - Read CHANGELOGs
  - composer outdated
## Versions Constraints

- **Exact Match:**
  - 1.0.0
  - 1.2.3-beta2
  - dev-master

- **Wildcard Range:**
  - 1.0.*
  - 2.*

- **Hyphen Range:**
  - 1.0-2.0
  - 1.0.0 - 2.1.0
  - >=1.0.0 <2.1
  - >=1.0.0 <=2.1.0

- **Unbounded Range:**
  - >= 1.0
  - *Bad!*

- **Next Significant Release**
  - ~1.2
  - ~1.2.3
  - >=1.2.0 <2.0.0
  - >=1.2.3 <1.3.0

- **Caret/Semver Operator**
  - ^1.2
  - ^1.2.3
  - Best Choice for Libraries
    - >=1.2.0 <2.0.0
    - >=1.2.3 <2.0.0

Operators: " " = AND, "||" = OR
Stabilities

- **Order**
dev -> alpha -> beta -> RC -> stable

- **Automatically from tags**
  1.2.3 -> stable
  1.3.0-beta3 -> beta

- **Automatically from branches**
  Branch -> Version (Stability)
  2.0 -> 2.0.x-dev (dev)
  master -> dev-master (dev)
  myfeature -> dev-myfeature (dev)

- **Choosing**
  "foo/bar": "1.3.*@beta"
  "foo/bar": "2.0.x-dev"

  "minimum-stability": "alpha"
$ php composer.phar validate

./composer.json is valid for simple usage with composer but has strict errors that make it unable to be published as a package:
See https://getcomposer.org/doc/04-schema.md for details on the schema
name: The property name is required
description: The property description is required
require.composer/composer: unbound version constraints (dev-master) should be avoided

Common: Version entry in composer.json conflicts with tag

$ php composer.phar self-update

$ php composer.phar update -vvv
Resolution Conflicts: Overly Strict Requirements

// composer.json

"require": {
  "cool/alice": "~1.3",
  "lazy/bob": "~1.2"
}

// dependencies

"name": "cool/alice",
"require": {
  "monolog/monolog": "~1.6"
}

"name": "lazy/bob",
"require": {
  "monolog/monolog": "1.3.*"
Resolution Conflicts: Overly Strict Requirements

Your requirements could not be resolved to an installable set of packages.

Problem 1
- Installation request for lazy/bob ~1.2 -> satisfiable by lazy/bob[1.4.0].
- Installation request for cool/alice ~1.3 -> satisfiable by cool/alice[1.3.0].
- lazy/bob 1.4.0 requires monolog/monolog 1.3.* -> satisfiable by monolog/monolog[1.3.0, 1.3.1].
- cool/alice 1.3.0 requires monolog/monolog ~1.6 -> satisfiable by monolog/monolog[1.6.0, 1.7.0].
- Can only install one of: monolog/monolog[1.6.0, 1.3.0].
- Can only install one of: monolog/monolog[1.6.0, 1.3.1].
- Conclusion: don't install monolog/monolog 1.3.1
- Conclusion: don't install monolog/monolog 1.7.0
- Conclusion: don't install monolog/monolog 1.3.0
- Conclusion: don't install monolog/monolog 1.6.0
Resolution Conflicts: Overly Strict Requirements

// composer.json

"require": {
  "cool/alice": "~1.3",
  "lazy/bob": "~1.2"
}

// dependencies

"name": "cool/alice",
"require": {
  "monolog/monolog": "~1.6"
}

"name": "lazy/bob",
"require": {
  "monolog/monolog": "1.3.*"
Resolution Conflicts: Stabilities

// composer.json

"minimum-stability": "beta",
"require": {
    "monolog/monolog": "1.*",
    "symfony/symfony": "~2.4",
    "bad/package": "dev-master"
}

// dependencies

"name": "bad/package",
"require": {
    "monolog/monolog": "dev-master",
}
Resolution Conflicts: Stabilities

Your requirements could not be resolved to an installable set of packages.

Problem 1
- Installation request for bad/package dev-master -> satisfiable by bad/package[dev-master].
- bad/package dev-master requires monolog/monolog dev-master -> no matching package found.
Resolution Conflicts: Stabilities

// composer.json

"minimum-stability": "beta",
"require": {
    "monolog/monolog": "1.*",
    "symfony/symfony": "~2.4",
    "bad/package": "dev-master"
}

// dependencies

"name": "bad/package",
"require": {
    "monolog/monolog": "dev-master",
}
Resolution Conflicts: Stabilities

```json
// composer.json

"minimum-stability": "beta",
"require": {
    "monolog/monolog": "1.*@dev",
    "symfony/symfony": "~2.4",
    "bad/package": "dev-master"
}

// dependencies

"name": "bad/package",
"require": {
    "monolog/monolog": "dev-master"
}
```
- Installing monolog/monolog (dev-master 5ad421d)
  Cloning 5ad421d6a1d5d7066a45b617e5164d309c4e2852
// monolog

"name": "monolog/monolog",
"extra": {
    "branch-alias": {
        "dev-master": "2.0.x-dev"
    }
}

Your requirements could not be resolved to an installable set of packages.

Problem 1
- Installation request for monolog/monolog 1.*@dev -> satisfiable by monolog/monolog[1.12.0].
- Installation request for bad/package dev-master -> satisfiable by bad/package[dev-master].
- bad/package dev-master requires monolog/monolog dev-master -> satisfiable by monolog/monolog[dev-master].
- Can only install one of: monolog/monolog[1.12.0, dev-master].

We require “2.*@dev” instead
- Resolution works
- Project is probably broken:
  bad/package may not be compatible with 2.*
No error but unexpected result?

- composer why [--tree] foo/bar
  mydep/here 1.2.3 requires foo/bar (^1.0.3)

- composer why-not [--tree] foo/bar ^1.2
  foo/bar 1.2.3 requires php (>=7.1.0 but 5.6.3 is installed)
The Lock File

Contents
- all dependencies including transitive dependencies
- Exact version for every package
- download URLs (source, dist, mirrors)
- Hashes of files

Purpose
- Reproducibility across teams, users and servers
- Isolation of bug reports to code vs. potential dependency breaks
- Transparency through explicit updating process
Commit The Lock File

- If you don’t
  - composer install without a lock file is a composer update
  - Conflict can randomly occur on install
  - You may not get the same code
  - You no longer manage change
    Change is managing you!

- The lock file exists to be committed!
The Lock file will conflict
Day 0: “Initial Commit”

**Project**

- zebra 1.0
- giraffe 1.0

**master**

`composer.lock`
- zebra 1.0
- giraffe 1.0

**dna-upgrade**

`composer.lock`
- zebra 1.0
- giraffe 1.0
Week 2: Strange new zebras require duck

**Week 2: Strange new zebras require duck**

- **Project**: zebra 1.1
  - **master**: composer.lock
    - zebra 1.1
    - giraffe 1.0
    - duck 1.0
- **Project**: zebra 1.0
  - **dna-upgrade**: composer.lock
    - zebra 1.0
    - giraffe 1.0
Week 3: Duck 2.0
Week 4: Giraffe evolves to require duck 2.0
Text-based Merge

project

master

composer.lock
- zebra 1.1
- giraffe 1.2
- duck 1.0
- duck 2.0

Merge results in invalid dependencies
Reset composer.lock

```
git checkout <refspec> -- composer.lock  
git checkout master -- composer.lock
```
Apply the update again

```
composer update giraffe --with-dependencies
```

```
Project

zebra 1.1

master

composer.lock
- zebra 1.1
- giraffe 1.2
- duck 2.0

giraffe 1.2

duck 2.0
```
How to resolve lock merge conflicts?

- composer.lock cannot be merged without conflicts
  - contains hash over relevant composer.json values

- git checkout <refspec> -- composer.lock
  - git checkout master -- composer.lock

- Reapply changes
  - composer update <list of deps>
Deployment
Never Deploy without a Lock File

Do not run composer update during deployments
composer install performance

- Use `--prefer-dist` to avoid git clones
  - Will always download zip files if possible (default for stable versions)

- Store `~/.composer/cache` between builds
  - How depends on CI product/setup you use
Autoloader Optimization

- composer install --optimize-autoloader
  - composer dump-autoloading --optimize
- composer install --optimize-autoloader --classmap-authoritative
  - composer dump-autoloading --optimize --classmap-authoritative
- composer install --optimize-autoloader --apcu-autoloader
  - composer dump-autoloading --optimize --apcu

https://getcomposer.org/doc/articles/autoloader-optimization.md
Autoloader Optimization

- Use this one
  `composer dump-autoloading --optimize --classmap-authoritative`

- Requires PHP7 to be optimal
  - opcache can keep static array definition in shared memory
  - no loading overhead on PHP request startup

- Will not search for classes not in lookup table
  - not useful for development
  - not useful for dynamically generated code (don’t do that!)
Platform Requirements

- Platform repository
  - implicitly defined additional package repository
  - contains packages for
    - PHP
    - extensions
    - system libraries (e.g. libxml)
  - packages cannot be updated/installed/removed
$ ./composer.phar show --platform

composer-plugin-api 1.1.0 The Composer Plugin API
ext-apcu 5.1.8 The apcu PHP extension
ext-ctype 7.2.5 The ctype PHP extension
ext-curl 7.2.5 The curl PHP extension
ext-date 7.2.5 The date PHP extension
ext-dom 20031129 The dom PHP extension
ext-fileinfo 1.0.5 The fileinfo PHP extension
ext-filter 7.2.5 The filter PHP extension
ext-ftp 7.2.5 The ftp PHP extension
ext-hash 1.0 The hash PHP extension
ext-iconv 7.2.5 The iconv PHP extension
ext-intl 1.1.0 The intl PHP extension
ext-json 1.6.0 The json PHP extension
ext-libxml 7.2.5 The libxml PHP extension
... 
lib-curl 7.59.0 The curl PHP library
lib-ICU 58.2 The intl PHP library
lib-libxml 2.9.5 The libxml PHP library
lib-openssl 2.5.5 LibreSSL 2.5.5
lib-pcre 8.41 The pcre PHP library
php 7.2.5 The PHP interpreter
php-64bit 7.2.5 The PHP interpreter, 64bit
php-ipv6 7.2.5 The PHP interpreter, with IPv6 support
Platform Requirements

```
{
    "require": {
        "php": "^7.1.1"
    }
}
```

$ php -v
PHP 5.6.10

$ composer update

Your requirements could not be resolved to an installable set of packages.

Problem 1
- This package requires php ^7.1.1 but your PHP version (5.6.10) does not satisfy that requirement.
Platform Requirements

- What if you maintain multiple projects on your local system to be deployed to different platforms?
  - e.g. Server A running PHP 7.0, Server B running PHP 7.2
- What if you want to build Composer automation tools
  - Private Packagist at packagist.com runs on a single PHP version, managed projects have lots of different requirements
Platform Requirements

```
{
    "require": {
        "php": "^7.1.1"
    }
}
```

$ php -v
PHP 5.6.10

$ composer update --ignore-platform-reqs
Success

No idea if dependencies even work on PHP 7.1.1
Platform Requirements

```
"require": {
  "php": "^7.1.1",
  "ext-intl": "*"
}
"config": {
  "platform": {
    "php": "7.1.2",
    "ext-intl": "1.1.0"
  }
}
```

```
$ php -v
PHP 5.6.10

$ composer update
Success
```
Platform Requirements

- Watch out if you are using Plugins!
  - Composer plugins (Composer installers are plugins, too)
    - Packages with type “composer-plugin”
    - Will be installed before all other packages if dependencies allow it
    - Code will be executed in Composer process during update/install
  - Can be disabled with **--no-plugins**
    - no easy way to run them on prod later

- Watch out if you are using scripts
  - Use **--no-scripts**
  - Run them separately in production with **composer run-script <name>**
Platform Requirements

```
"require": {
    "php": "^7.1.1",
    "ext-intl": "*"
}
"config": {
    "platform": {
        "php": "7.1.2",
        "ext-intl": "1.1.0"
    }
}
```

```
$ composer update
Success
- Create ZIP
- deploy to prod

PHP Fatal Error

Prod was actually still on PHP 5.6
```
Platform Requirements

```
"require": {
    "php": "^7.1.1",
    "ext-intl": "*"
}
"config": {
    "platform": {
        "php": "7.1.2",
        "ext-intl": "1.1.0"
    }
}
```

- dev$ composer update
- Create ZIP
- upload to prod
- composer check-platform-reqs
  - no error? switch to new code
Summary

- Library CI: composer update
  --prefer-lowest --prefer-stable
- git checkout <branch> -- composer.lock
- composer dump-autoload --optimize
  --classmap-authoritative
- composer why/why-not
- composer show --platform
  {"config":{"platform":{"php":"7.2.5"}}}
  composer check-platform-reqs
  Watch out for plugins & scripts!

  - Formalize BC promises for users of your libraries
  - SemVer: Don’t be afraid to increase the major version
  - Document changes to dependencies

  Commit the composer.lock file!
Composer in Depth
Part 2

Nils Adermann
@naderman

Jordi Boggiano
@seldaek

Private Packagist
https://packagist.com
A brief history of Composer

- Symfony & phpBB plugins
- Apr 2011 - First Commit
- Sep 2011 - Packagist.org
- Apr 2012 - First 1,000 Packages
- Apr 2013 - First 10,000 Packages
- Jun 2014 - Toran Proxy
A brief history of Composer

- Symfony & phpBB plugins
- Apr 2011 - First Commit
- Sep 2011 - Packagist.org
- Apr 2012 - First 1,000 Packages
- Apr 2013 - First 10,000 Packages
- Jun 2014 - Toran Proxy
- Dec 2016 - Private Packagist
- Jun 2018 - 185,000+ Packages with 1,200,000+ Versions
What is Dependency Management?

- Assembly
- Dependency Change Management
- Risk Analysis & Reduction

May happen at build time or at runtime
Dependency Assembly

- Installation of Libraries, Tools, etc.
  - composer install
  - apt-get install foo
  - Application of Configuration Management (Puppet, Chef, Ansible, Salt, ...)

- Configuration for Connections to Services, external APIs
  - Authentication
  - Glue Code

- Connection to Services (usually at Runtime)
Dependency Assembly

Past:
- Step-by-Step installation instructions
- Readmes, Delete and reinstall individual packages

Today:
- Description of a system state (e.g. composer.json, top.sls)
- Tools to move the system into the state (e.g. composer, salt)
Dependency Change Management

- Dependency Change
  - Adding, Removing, Updating, Replacing of Libraries
  - Replacing APIs
  - composer update

- Dependency Change Management
  - Balance Risks, Consequences, Cost & Advantages
  - Architecture Decisions which enable “Change”
    - Example: Abstraction to replace concrete service
Composer Architecture Decisions

- Separate independent tools and services
  - Avoid PEAR confusion and problems

- Build reusable code to allow for other tools and services to emerge
  - Check out https://github.com/composer
composer update/install

- Load all package metadata
- Resolve dependencies to create transaction (install/remove/update)
- Create lock file
- Download or checkout files from locations in lock file
Satis

- Static File Generator
- Big config file of all packages
- Archive creation for downloads possible
- No hooks to trigger updates
- Not suitable for building further tools or services on top of it
- Considerably cost to setup & maintain
Private Packagist

- Your own Composer repository done right
  - SaaS or on-premises - https://packagist.com
- Easy setup
  - Integration with GitHub, Gitlab, Bitbucket
- Authentication
- Permission Management
- Foundation for future functionality to simplify dependency management
Load package metadata?

- Package repositories
- Path repositories
- VCS repositories
- Composer Repositories
  - Packagist (packagist.org)
  - Satis
  - Private Packagist (packagist.com)
"repositories": [  {  "type": "package",  "package": {  "name": "vendor/package",  "version": "1.0.0",  "dist": {  "url": "http://example.org/package.zip",  "type": "zip"  },  "source": {  "url": "git://example.org/package.git",  "type": "git",  "reference": "tag name, branch name or commit hash"  }  }  }],  "require": {  "vendor/package": "1.0.0"  }

Monorepo

- repo/projectA/composer.json
  "repositories": [
    {
      "type": "path", "url": "./projectB"
    }
  ],
  "require": {
    "vendor/projectB": "dev-master"
  }

- repo/projectB/composer.json
  "name": "vendor/projectB",
  "version": "dev-master"
VCS Repository

"repositories": [
  {
    "type": "vcs",
    "url": "git://example.org/MyRepo.git"
  }
]

- Information is inferred from composer.json files in tags & branches
- dist download URLs only for known hosts, e.g. github, bitbucket, gitlab
Using Forked Packages

Custom repositories have priority:

```
"repositories": [
 {
  "type": "vcs",
  "url": "https://github.com/naderman/symfony"
 }
],
"require": {
  "symfony/symfony": "dev-master"
}
```
Using Forked Packages

Custom branches are available (composer show -v symfony/symfony)

"repositories": [
    {
        "type": "vcs",
        "url": "https://github.com/naderman/symfony"
    }
],
"require": {
    "symfony/symfony": "dev-my-patch"
}
Using Forked Packages

Aliases allow other dependencies to resolve against custom branches:

```
"require": {
    "symfony/symfony": "dev-my-branch as 3.1.0"
    "other/package": "1.23"
}

"name": "other/package"
"require": {
    "symfony/symfony": "^3.1"
}
```
Define the appropriate installation method

```
"config": {
  "preferred-install": {
    "acme/*": "source",
    "*": "dist"
  }
},
"require": {
  "acme/foo": "^1.2",
  "other/package": "^1.4"
}
```
"repositories": [
  {
    "type": "composer",
    "url": "https://satis.example.org/"
  },
  {
    "type": "composer",
    "url": "https://repo.packagist.com/my-org"
  },
  {
    "packagist.org": false
  }
]
Composer Repository: Satis

packages.json:
{
    packages: {
        "seld/private-test": {
            "dev-master": {
                name: "seld/PRivate-test",
                version: "dev-master",
                version_normalized: "9999999-dev",
                source: {
                    ....
                },
                dist: {
                    ....
                },
                require: {
                    php: ">=5.3.0",
                    ...
                }
            }
        }
    }
}
Composer Repository: packagist.org

packages.json:
{
    packages: [ ],
    notify: "/downloads/%package%",
    notify-batch: "/downloads/",
    providers-url: "/p/%package%$%hash%.json",
    search: "/search.json?q=%query%&type=%type%",
    provider-includes: {
        p/provider-2013$%hash%.json: {
            sha256: "eb67fda529996db6fac4647ff46cf41bb31065536e1164d0e75f911d160f6b9f"
        },
        ...
        p/provider-archived$%hash%.json: {
            sha256: "444a8f22af4bc0e2ac0c09eda1f5edc63158a16e9d754100d7f774b930a38ae6"
        },
        p/provider-latest$%hash%.json: {
            sha256: "b0e0065f1e36f061b9fd2bbb096e7986321421f9eecd3d5e68dc4780d7295c33"
        }
    }
}
Composer Repository: Private Packagist

packages.json:
{
    packages: {
        "seld/private-test": {
            "dev-master": {
                name: "seld/PRivate-test",
                ...
            }
        }
    }
    providers-lazy-url: "/myorg/p/%package%.json",
    mirrors: [
        {
            preferred: true
        }
    ]
}
Composer with Private Dependencies

- composer.json
  - require:
    - foo/bar: ^1.3

- composer.lock
  - foo/bar: 1.3.4
  - foo/dep: 1.2.1

- vendor/foo/
  - bar/Bar.php
  - bar/Bax.php
  - dep/Dep.php
  - dep/Doo.php

- foo-bar.git
- foo-dep.git
Composer with Private Dependencies: Private Packagist

composer.json
require:
foo/bar: ^1.3

composer.lock
foo/bar: 1.3.4
foo/dep: 1.2.1

composer install
vendor/foo/bar/Bar.php
bar/Bax.php
dep/Dep.php
dep/Doo.php

packages.json
foo-bar-1.3.4.zip
foo-dep-1.2.1.zip

private packagist

Atlassian
Bitbucket

foo-bar.git
foo-dep.git

composer update
git clone
https, unzip

json https
https, unzip
Risk Analysis: Availability

Affects Assembly

Examples:
- Open Source Library deleted
- Payment Service unavailable
- EU VATId Service out of order
- Jenkins not accessible
Risk Reduction: Availability

- Software is available when you have a copy
  - composer cache
  - Forks
  - Private Packagist or Satis
Composer with Open Source Dependencies

- composer.json
  - require:
    - foo/bar: ^1.3
- composer update
- composer.lock
  - foo/bar: 1.3.4
  - foo/dep: 1.2.1
- composer install
- vendor/foo/
  - bar/Bar.php
  - bar/Bax.php
  - dep/Dep.php
  - dep/Doo.php
- packages.json
- json https
- foo-bar.git
- foo-dep.git
- git clone

- Atlassian Bitbucket
Composer with Open Source Dependencies: Private Packagist

- composer.json
  - require:
    - foo/bar: ^1.3

- composer.lock
  - foo/bar: 1.3.4
  - foo/dep: 1.2.1

- composer install
  - vendor/foo/bar/Bar.php
  - bar/Bax.php
  - dep/Dep.php
  - dep/Doo.php

- composer update

- packages.json
  - foo-bar-1.3.4.zip
  - foo-dep-1.2.1.zip

- git clone
- https, unzip

- PRIVATE PACKAGIST

- json https

- Atlassian Bitbucket
  - foo-bar.git
  - foo-dep.git
Downloading files from the lock file

{  
  "content-hash": "bb557b05609c879265a30bc052ef77e4",
  "packages": [
    
    {
      "name": "aws/aws-sdk-php",
      "version": "3.25.6",
      "source": {
        "type": "git",
        "url": "https://github.com/aws/aws-sdk-php.git",
        "reference": "fe98140a4811abbe9104477b167dc3c7f9a8391b"
      },
      "dist": {
        "type": "zip",
        "reference": "fe98140a4811abbe9104477b167dc3c7f9a8391b"
      },
      "require": {
        "guzzlehttp/guzzle": "^5.3.1|^6.2.1",
      }
    }
  ]
}
Downloading files from the lock file with Private Packagist

```
"packages": [
    {
        "name": "aws/aws-sdk-php",
        "version": "3.25.6",
        "source": {
            "url": "https://github.com/aws/aws-sdk-php.git",
            ...
        },
        "dist": {
            "type": "zip",
            "reference": "fe98140a4811abbe9104477b167dc3c7f9a8391b",
            "mirrors": [
                {
                    "url": "https://repo.packagist.com/phpbb/dists/%package%/%version%/%reference%.%type%",
                    "preferred": true
                }
            ]
        }
    }
]
```
Risk Reduction: (New) Dependencies

Quality Criteria for software libraries (and services)

- Number of Maintainers / Developers
- Actively Developed?
- How many users?
  - Packagist shows installation count
- Where is a library being installed from?
  - GitHub, self-hosted svn server? -> Availability
- Alternatives / how easy to replace? Complexity?
  - Could you take over maintenance?
Risk Reduction: Compatibility

Semantic Versioning (Semver) promises Compatibility

\[x.y.z\]

- Must be used consistently
- Only valuable if BC/Compatibility promise formalized
- Otherwise choose narrower Version Constraints, check more frequently
  - e.g. \(~1.2.3\) instead of \(^1.2.3\)
Risk Reduction: Compatibility

- Automated
  - Tests
  - Static Analysis

- Manual
  - Read Changelogs (and write them!)
  - Experience which libraries break BC
Risk Minimization: Compliance / Legal

- Affects Change Management
- Example
  - Viral Copy-Left License not compatible with proprietary product
- composer licenses
- Private Packagist License Review
Assessing & Managing Risk

- Formulate a Plan B
- Identify problems which are probable and which have great effects

- **Dependencies are great!** They can save tons of money and time
- Only spend resources on reducing risk until the risk is acceptable

- Private Packagist can help you manage and reduce these risks by being the one central place for all your third party code
How is Private Packagist helping?

- Faster and more reliable composer operations
- Work with private dependencies more efficiently
  - Automatic synchronization of packages, teams, users, permissions
  - Authentication Tokens
- One central place for all your dependencies
  Improved understanding of and control over open-source usage
  - Statistics and references between internal code and open-source code
  - License review
- Much more to come!
Thank you!

https://packagist.com

Questions / Feedback?

E-Mail: contact@packagist.com
Twitter: @naderman & @seldaek