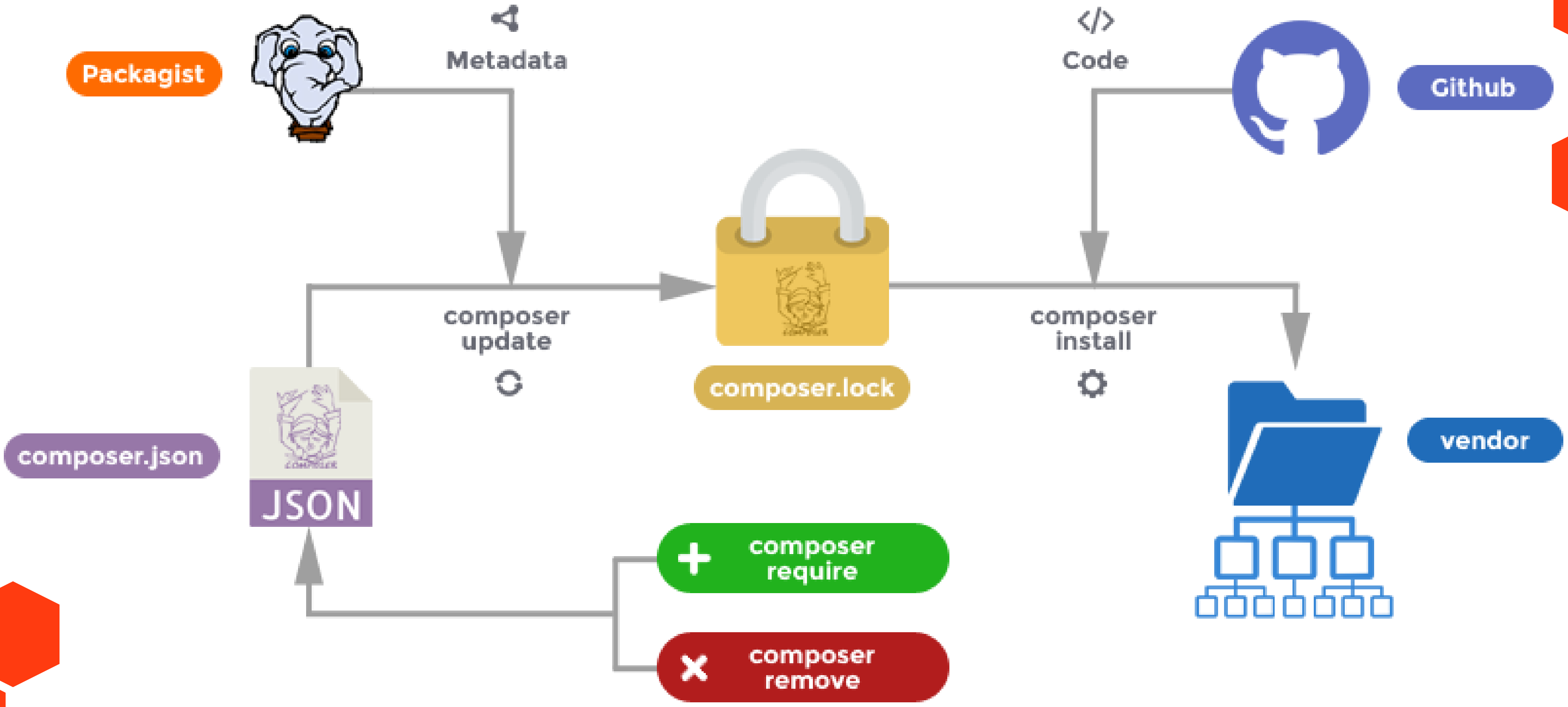
Developing and Deploying Magento with Composer: *Best Practices*

Nils Adermann
Co-Founder, Packagist Conductors
@naderman - n.adermann@packagist.com

Meet Magento™

PRIVATE PACKAGIST

COMPOSER

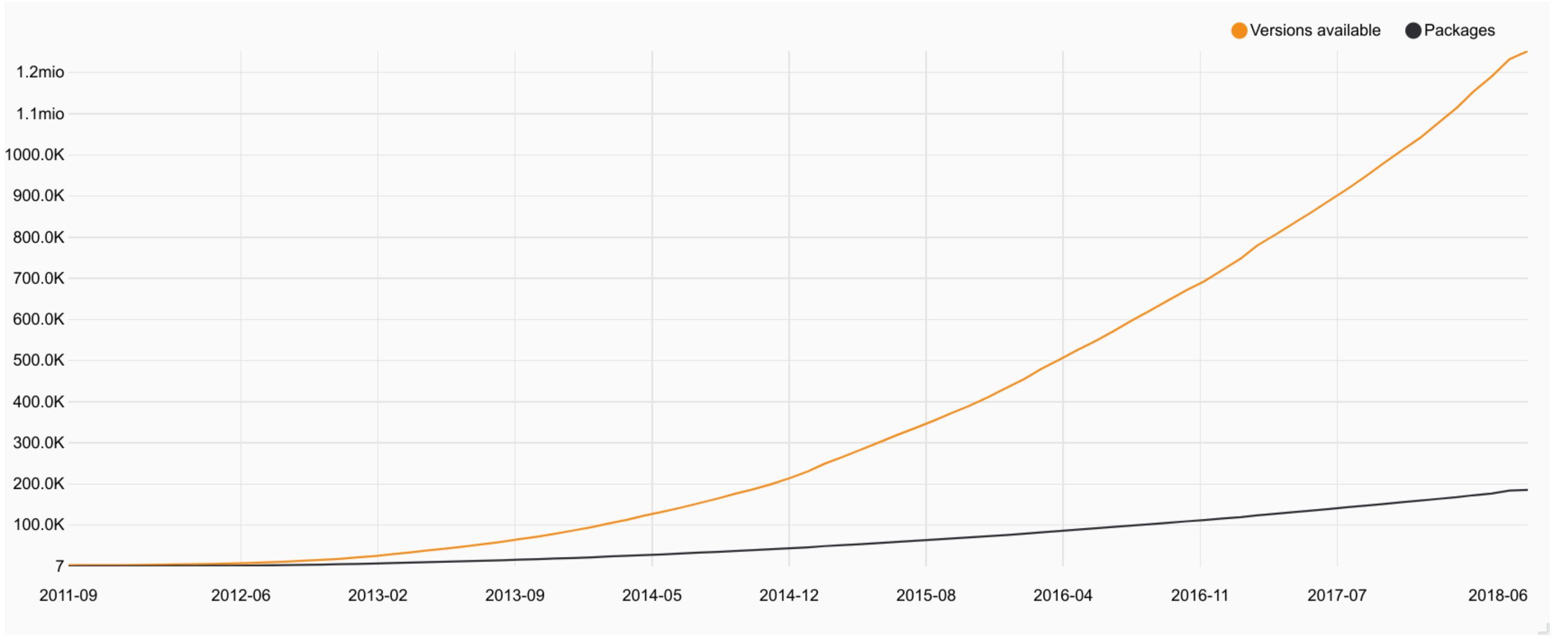

Packagist
Metadata
Code
Github
composer update
composer.lock
composer install
composer.json
JSON
vendor
composer require
composer remove

Package Repositories

- Third Parties
  - Packagist https://packagist.org
  - Magento Marketplace https://marketplace.magento.com
  - Individual vendors' repositories
- Private Packages
  - Any git/svn/mercurial/... repository
  - GitHub, Bitbucket, GitLab
  - Private Packagist https://packagist.com

Leveraging Open-Source Packages

- Nearly 200k packages on packagist.org
  - Many useful well tested, maintained and secure packages
  - Large amounts of unmaintained, insecure, broken, or poorly working code



Leveraging Open-Source Packages

- Evaluate packages every time before you write code yourself

- Selection criteria
  - Quality of documentation (changelogs?)
  - Development activity (commits, issues, PRs)
  - Number of maintainers
  - Installation counts, GitHub stars
  - Complexity

- It's all trade-offs – no golden rule

Magento Marketplace

- Apply similar criteria as for open-source packages

- Additional factors to consider
  - Cost
  - Licenses
  - Reviews / Ratings
  - Extension Quality Program

Using your private code with Composer

-
  ```
  "repositories": [
      {"type": "path", "url": "../core"}
  ],
  ```

-
  ```
  "repositories": [
      {"type": "vcs",
       "url": "https://github.com/naderman/symfony" }
  ],
  ```

-
  ```
  "repositories": [
      {"type": "composer",
       "url": "https://repo.packagist.com/my-org/" }
  ],
  ```

Development Environment *Best Practices*

create-project instead of cloning

- `composer create-project --repository-url=https://repo.magento.com/magento/project-community-edition <path>`
  - composer.json will have the correct contents
    - Different from forking the community edition

- Magento/project-community-edition is a metapackage
  - No code
  - Defines dependencies on a number of other packages

- Only clone if you're trying to contribute to a repository directly

Managing Updates: 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

Managing Updates: Stabilities

- Order dev -> alpha -> beta -> RC -> stable
- Automatically from tags
  - 1.2.3 -> stable
  - 1.3.0-beta3 -> beta
- Automatically from branches
  - branch name -> 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"

Managing Updates: Semantic Versioning

x.y.z

(BC-break).(new functionality).(bug fix)

https://semver.org

Managing Updates: Semantic Versioning

Promise of Compatibility

X.Y.Z

- Must be used consistently
  - Dare to increment X!
- Only valuable if BC/compatibility promise formalized
  - https://devdocs.magento.com/guides/v2.0/contributor-guide/backward-compatible-development/
  - http://symfony.com/doc/current/contributing/code/bc.html
  - Document breaks in changelog

Managing Updates

- `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`

Managing Updates: Unexpected Results

```
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)
```

Managing Updates: Partial Updates

```
{
    "name": "zebra/zebra",
    "require": {
        "horse/horse": "^1.0"
}}
{
    "name": "giraffe/giraffe",
    "require": {
        "duck/duck": "^1.0"
}}
```

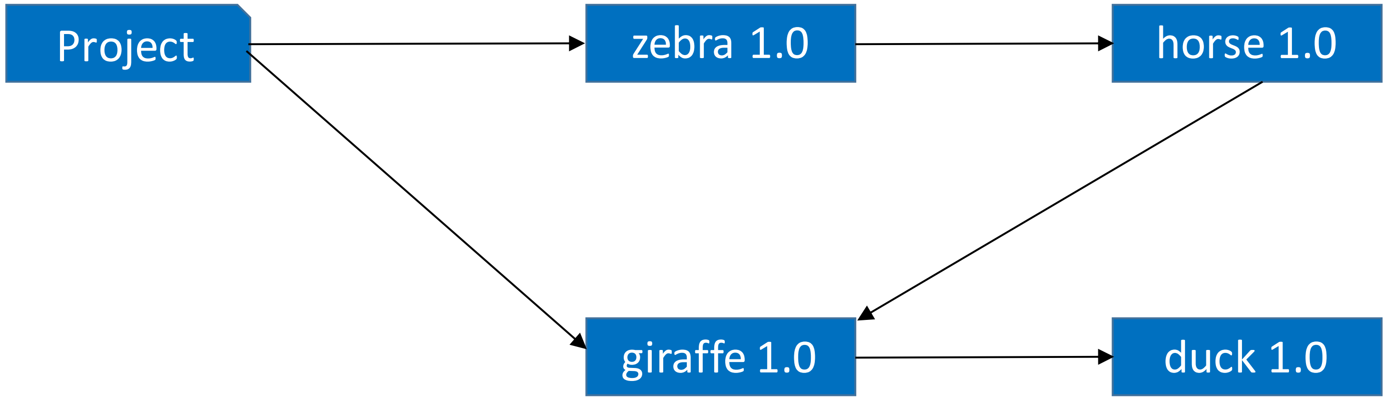
Managing Updates: Partial Updates

```
{
    "name": "horse/horse",
    "require": {
        "giraffe/giraffe": "^1.0"
}}
{
    "name": "duck/duck",
    "require": {}
}
```

Managing Updates: Partial Updates

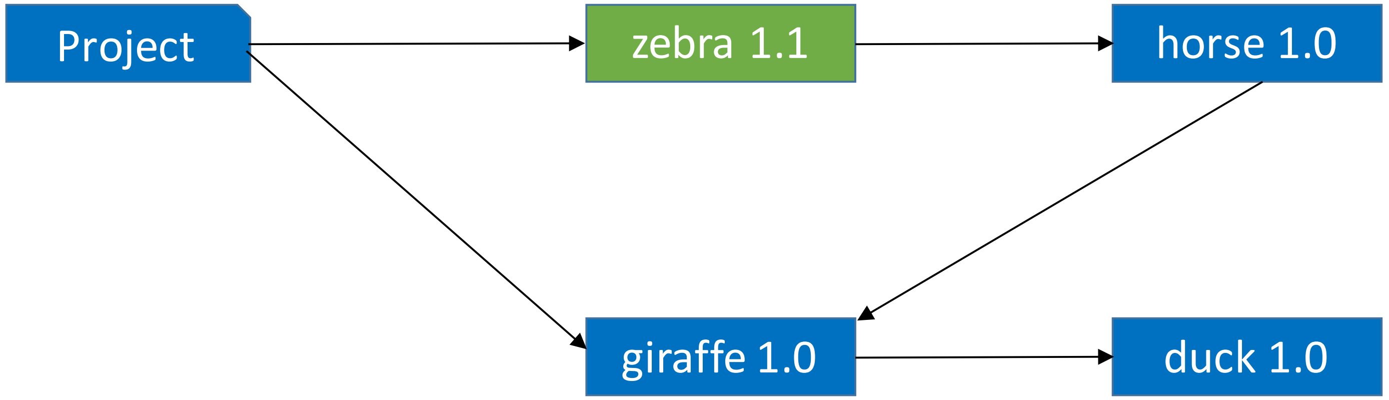
```
{
    "name": "my/project",
    "require": {
        "zebra/zebra": "^1.0",
        "giraffe/giraffe": "^1.0"
    }
}
```

Managing Updates: Partial Updates


$$\begin{tikzcd}
\text{Project} \arrow[r] \arrow[dr] & \text{zebra 1.0} \arrow[r] & \text{horse 1.0} \arrow[dl] \\
 & \text{giraffe 1.0} \arrow[r] & \text{duck 1.0}
\end{tikzcd}$$

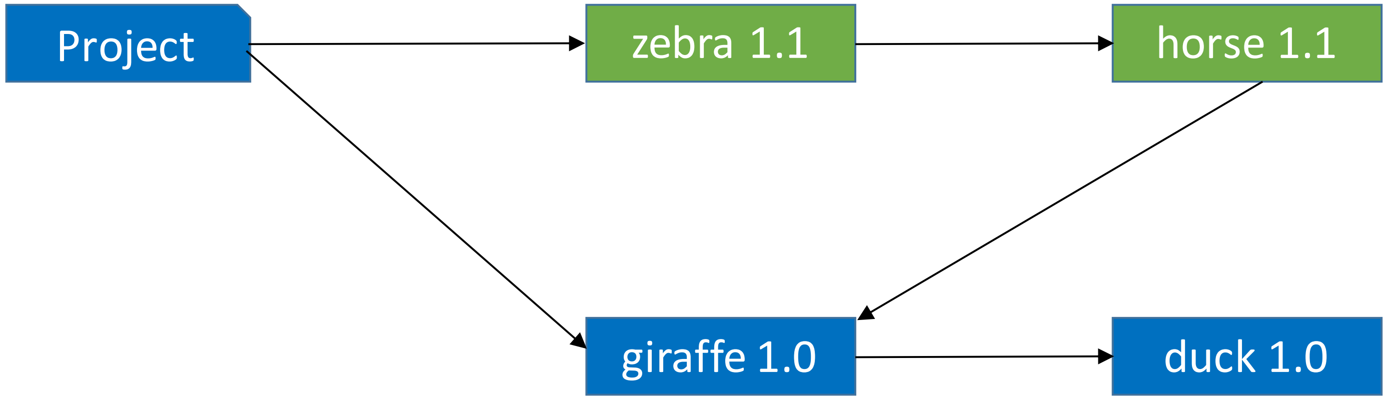
Now each package releases version 1.1

Managing Updates: Partial Updates



```
$ composer update zebra/zebra
  Updating zebra/zebra (1.0 -> 1.1)
```

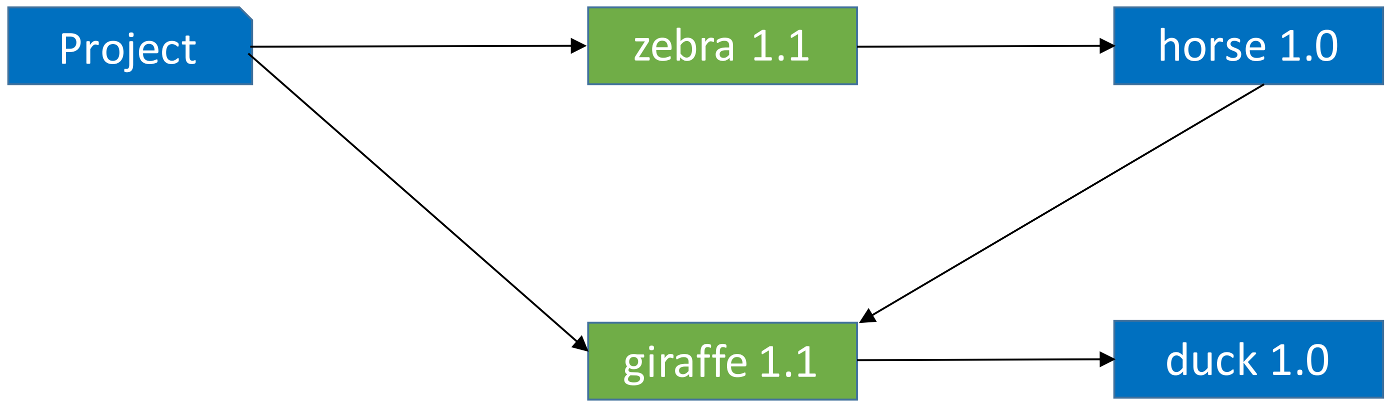
Managing Updates: Partial Updates


$$\begin{tikzcd}
\text{Project} \arrow[r] \arrow[dr] & \text{zebra 1.1} \arrow[r] & \text{horse 1.1} \arrow[dl] \\
& \text{giraffe 1.0} \arrow[r] & \text{duck 1.0}
\end{tikzcd}$$

```
$ composer update zebra/zebra --with-dependencies
  Updating horse/horse(1.0 -> 1.1)
  Updating zebra/zebra (1.0 -> 1.1)
```

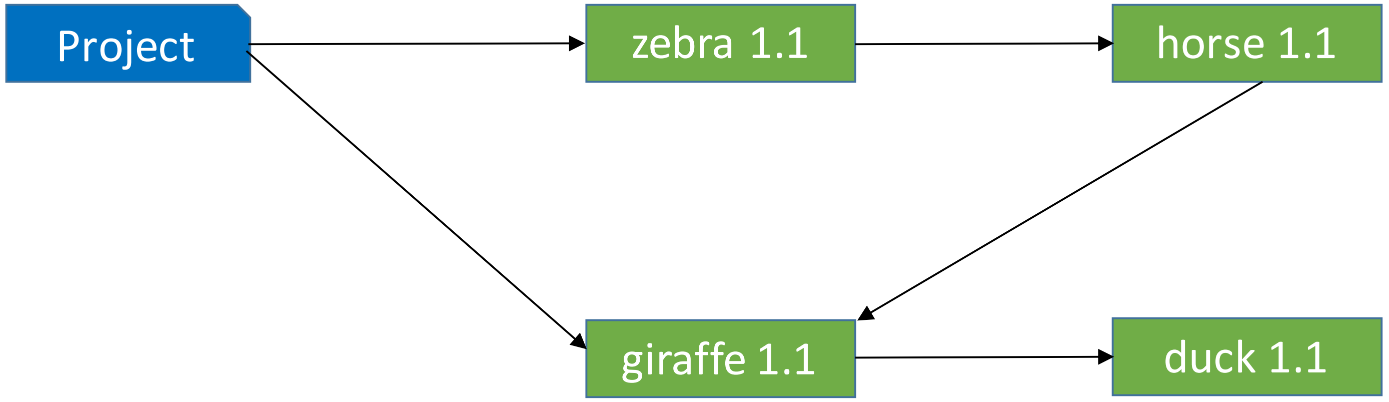
Meet Magento™ #MM18DE

Managing Updates: Partial Updates


$$\begin{tikzcd}
\text{Project} \arrow[r] \arrow[dr] & \text{zebra 1.1} \arrow[r] & \text{horse 1.0} \arrow[dl] \\
 & \text{giraffe 1.1} \arrow[r] & \text{duck 1.0}
\end{tikzcd}$$

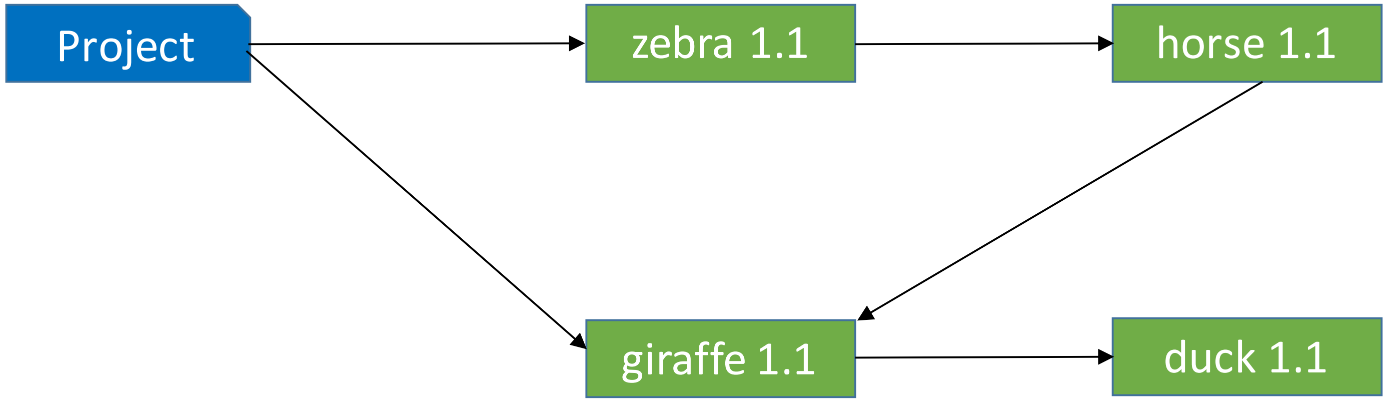
```
$ composer update zebra/zebra giraffe/giraffe
  Updating zebra/zebra (1.0 -> 1.1)
  Updating giraffe/giraffe(1.0 -> 1.1)
```

Managing Updates: Partial Updates



```
$ composer update zebra/zebra giraffe/giraffe --with-dependencies
  Updating duck/duck(1.0 -> 1.1)
  Updating giraffe/giraffe(1.0 -> 1.1)
  Updating horse/horse(1.0 -> 1.1)
  Updating zebra/zebra(1.0 -> 1.1)
```

Managing Updates: Partial Updates



```
$ composer update zebra/zebra --with-all-dependencies
  Updating duck/duck(1.0 -> 1.1)
  Updating giraffe/giraffe(1.0 -> 1.1)
  Updating horse/horse(1.0 -> 1.1)
  Updating zebra/zebra(1.0 -> 1.1)
```

Managing Updates: The Lock File

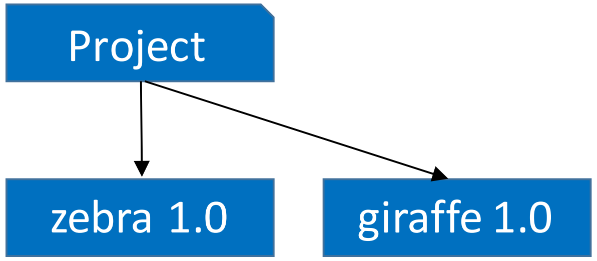
- Contents
  - All dependencies including transitive dependencies
  - Exact version for every package
  - Download URLs (source, dist, mirrors)
  - Hashes of files
- Purpose
  - **Reproducibility** across teams, users, and servers
  - **Isolation** of bug reports to code vs. potential dependency breaks
  - **Transparency** through explicit updating process

Commit The Lock File

Every composer install without a lock file is a catastrophe waiting to happen

The Lock File Will Conflict

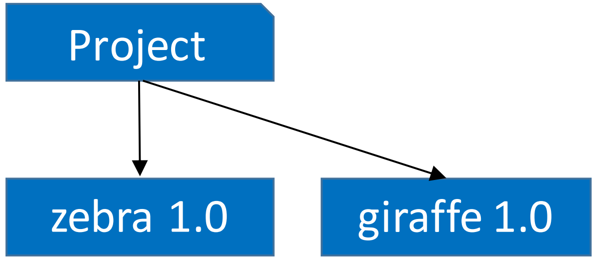
Day 0: "Initial Commit"



master

composer.lock

- zebra 1.0
- giraffe 1.0

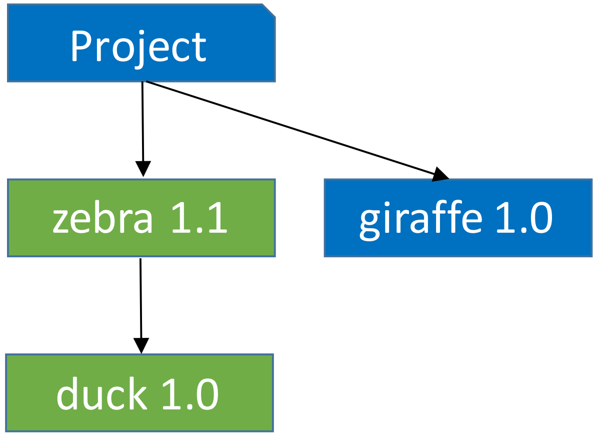


dna-upgrade

composer.lock

- zebra 1.0
- giraffe 1.0

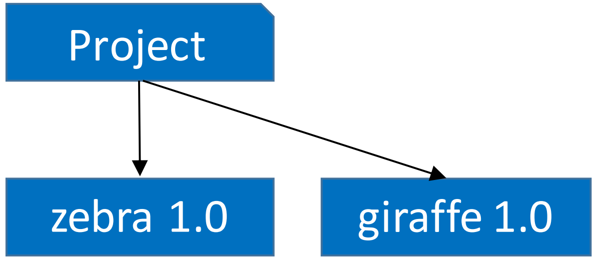
Week 2: Strange new zebras require duck


$$\begin{tikzcd}
\text{Project} \arrow[d] \arrow[dr] & \\
\text{zebra 1.1} \arrow[d] & \text{giraffe 1.0} \\
\text{duck 1.0} &
\end{tikzcd}$$

master

composer.lock

- zebra 1.1
- giraffe 1.0
- duck 1.0


$$\begin{tikzcd}
\text{Project} \arrow[d] \arrow[dr] & \\
\text{zebra 1.0} & \text{giraffe 1.0}
\end{tikzcd}$$

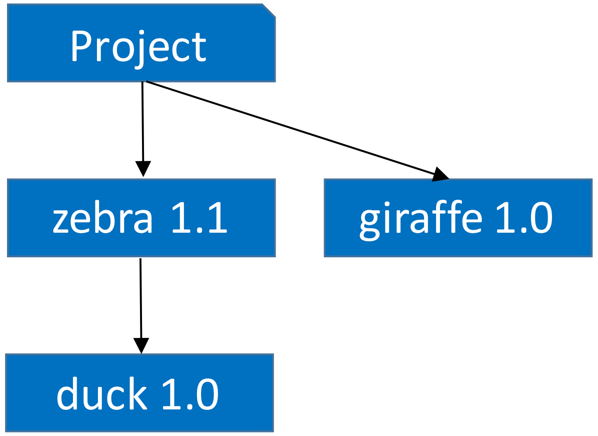
dna-upgrade

composer.lock

- zebra 1.0
- giraffe 1.0

Week 3: Duck 2.0

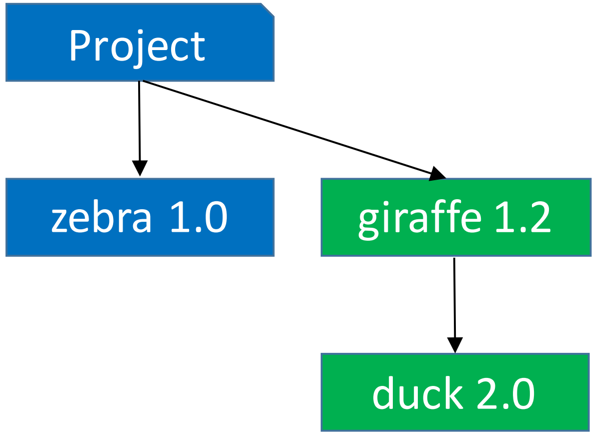
Week 4: Giraffe evolves, requires duck 2.0



master

composer.lock

- zebra 1.1
- giraffe 1.0
- duck 1.0

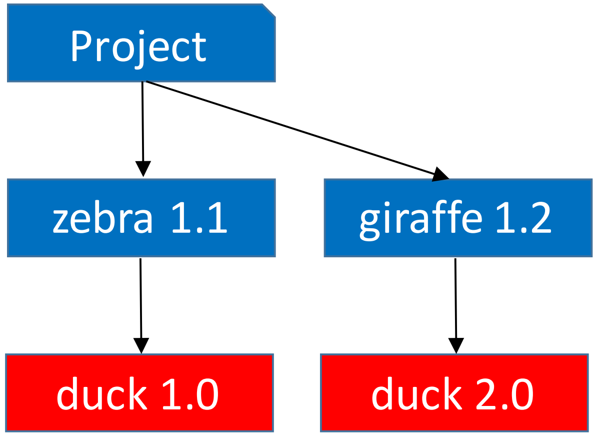


dna-upgrade

composer.lock

- zebra 1.0
- giraffe 1.2

Text-based Merge

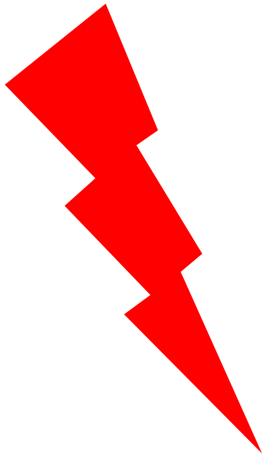


master

composer.lock

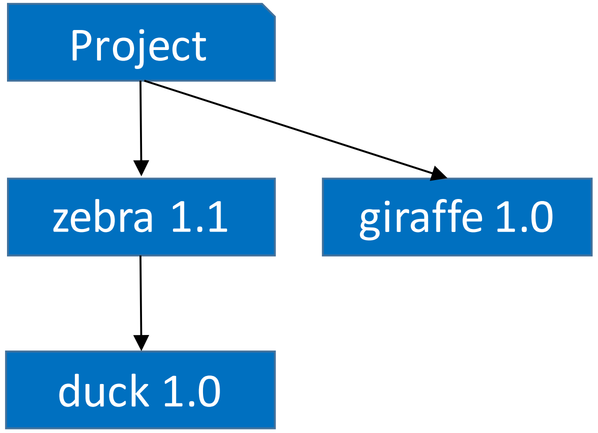
- zebra 1.1
- giraffe 1.0
- **duck 1.0**
- **duck 2.0**

Merge results in invalid dependencies



Reset composer.lock

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



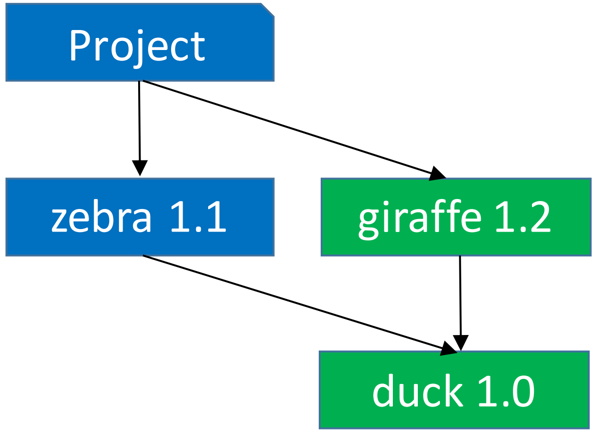
dna-upgrade

composer.lock

- zebra 1.1
- giraffe 1.0
- duck 1.0

Apply the update again

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



master

composer.lock

- zebra 1.1
- giraffe 1.2
- duck 2.0

Resolving composer.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`**
- Repeat: `composer update <list of deps>`
    - Store parameters in commit message
    - Separate commit for the lock file update

Publishing Packages

- `composer validate`
  - Will inform you about problems like missing fields and warn about problematic choices like unbound version constraints

- Do not publish multiple packages under the same name, e.g. CE/EE
  - **Names must be unique**

Continuous Integration for Packages

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

Development Tools

- **require-dev** in composer.json
  - These packages won't be installed if you run `composer install --no-dev`
  - Use for testing tools, code analysis tools, etc.
- **`--prefer-source`**
  - Clone repositories instead of downloading and extracting zip files
  - Default behaviour for dev versions
  - Allows you to push changes back into dependency repos

Deployment
Best Practices

What properties should a deployment process have?

- Unreliable or slow deployment process
    - You will be scared to deploy
    - You will not enjoy deploying
- Consequence: You will not deploy often
    - Infrequent deploys increase risks
        - You will not be able to spot problems as quickly
        - Problems will fester over time
- Vicious Cycle
    - **Reliability and speed** are key to breaking it

Composer install performance

- `--prefer-dist`
    - Will always download zip files over cloning repositories
- Store `~/.composer/cache/` between builds
    - How to do this depends on CI product/setup you use

Autoloader Optimization

- `composer install` **`--optimize-autoloader`**
    - `composer dump-autoload –optimize`
- `composer install` **`--optimize-autoloader --classmap-authoritative`**
    - `composer dump-autoload –optimize --classmap-authoritative`
- `composer install` **`--optimize-autoloader --apcu-autoloader`**
    - `composer dump-autoload –optimize --apcu`

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

Reduce Dependence on External Services

- **Build process** ***(move more into this)***
  - Install dependencies (Composer, npm, …)
  - Generate assets (Javascript, CSS, …)
  - Create an artifact with everything in it
- **Deployment process** ***(make this as small as possible)***
  - Move the artifact to your production machine
    - sftp, rsync, apt-get install, …
  - Machine dependent configuration
  - Database modifications
  - Start using new version

Never Deploy without a composer.lock

Reduce Dependence on External Services

- Composer install loads packages from URLs in composer.lock
  - Packagist.org is metadata only
  - *Open-Source dependencies could come from anywhere*

- Solutions to unavailability
  - Composer cache in ~/.composer/cache
    - Unreliable, not intended for this use
  - Fork every dependency
    - Huge maintenance burden
  - Your own Composer repository mirroring all packages
    - Private Packagist

Summary

Development

- Make a checklist for new dependencies
- composer create-project
- SemVer: Don't be afraid to increase **X**
- Formalize BC promises for users of your libraries
- composer update [--dry-run] <package..>
- git checkout <branch> -- composer.lock then replay composer update
- Document changes to dependencies

Deployment

- Document & automate build process
- Composer install --prefer-dist --optimize-autoloader --no-dev
- Use a highly available Composer repository (Private Packagist)
- Deploy more frequently
- Focus on reliability and speed of your deployment process
- Deploying should not be scary

Nils Adermann - @naderman – n.adermann@packagist.com

PRIVATE PACKAGIST