Composer Dependencies Revamp

2017-03-19: This has been implemented for Tiki17. Next step is Composer Web Installer

(GOAL) Install Optional Libraries using Composer

Original text: Install optional libraries in Tiki via Composer

1.1.1. Challenges and opportunities

We could eventually turn off mods https://mods.tiki.org/ , which is only used currently for PluginR to get around licensing issues (GPL vs LGPL).

1.1.2. Security

- Permissions of files created by the Apache user
- Most of our recent security issues involved libraries managed by Composer but not used by everyone. All Tiki files have a feature and permission check which makes them only a risk if active but we don't have this for external libs.

1.1.3. License incompatibilities

- Sometimes, the best library is of an incompatible license (but still OSI compliant). See License
- We have spent a lot of time in the past to work with upstream projects to change the license (Bootstrap, Bootstrap Tour, etc.). But sometimes, it's just not possible.

1.1.4. Without shell access

- What we develop should work for users without shell access or, at least, they should have a documented procedure. For example, download this file abc.zip and unzip in folder /var/www/virtual/example.org/vendor_custom/
  - Mods has such documentation
- What is installed manually vs. custom libs installed by composer should be stored in a different directory.
  - The vendor/ directory can be deleted and everything is re-generated (great!).
    - We wouldn't want to have to track manual vs. system managed dependencies....

1.1.5. Single composer.json file

- Modifying the composer.json file, which will get overwritten on upgrades
- Jordi suggested to split in two files

1.1.6. End goal

1. A checkbox in tiki-admin.php that installs Composer Libraries when the relevant feature is activated.
   - This should run Composer if it can
     - If it can't, but for users with shell access: it should tell the user what shell command to run
     - For those with no shell access, it should tell the user what to install where (like Mods: http://twbasics.tikiforsmarties.com/Installing+Mods+Manually). This more or less is what people need to do now for ViewerJS and mPDF. Suggested location is a vendor_custom directory.
2. Tiki code that uses these libs would check for them,
   - ex.: https://sourceforge.net/p/tikiwiki/code/59054/tree/trunk/lib/pdflib.php#l48
3. If someone manually added a lib, and it's now out of date, there should be a warning in tiki-admin.php
   - If there is a security vulnerability, an email should be sent out. (This is actually something missing in Tiki in general.)
4. Tiki code that uses these libs would be maintained in common code base to keep things simpler.

- Coping with Complexity

5. If there are any post-install-cmd , they should be run

1.2. Possible Approaches

1.2.1. Use 2 composer files

One of the key objectives is to keep things simple for the user, allowing the user to install packages using composer (easily) and at the same time keep avoid problems for instance when using SVN because the tiki composer related files were changed when the user added a dependency with composer.

To simplify the life of the user, the intuitive way, is to allow the user to use the composer.json file in the root of the tiki. For that (in order to avoid collision with the code maintained with svn), the suggestion is to change the tiki structure, to match the following (relevant files / dirs only):

```
/
/composer.json.dist <- a sample composer.json
/vendor_bundled <- base folder for the dependencies bundled with tiki
/vendor_bundled/composer.json <- the composer file for tiki
```

As part of the instructions, we can suggest the user to rename `composer.json.dist` to `composer.json` (but at the end, should be ok for the user to generate his own `composer.json`)

The user will only need to do, for example, in the root of tiki:

```
$ composer require mpdf/mpdf
```

That will install the user dependencies in the “normal” composer folder.

For this to work, both sources (bundled packages and user packages) needs to be included in tiki, that can be done with the following patch:

```
Index: lib/init/initlib.php
===================================================================
--- lib/init/initlib.php  (revision 60173)
+++ lib/init/initlib.php  (working copy)
@@ -20,7 +20,7 @@
   exit;
 }

-if (! file_exists(__DIR__ . '/../../vendor/autoload.php')) {
+if (! file_exists(__DIR__ . '/../../vendor_bundled/autoload.php')) {
  echo "Your Tiki is not completely installed because Composer has not been run to fetch
package dependencies.\n";
  echo "You need to run 'sh setup.sh' from the command line.\n";
  echo "See https://dev.tiki.org/Composer for details.\n";
@@ -27,8 +27,13 @@
  exit;
 }

-require_once __DIR__ . '/../../vendor/autoload.php';
+require_once __DIR__ . '/../../vendor_bundled/autoload.php';

+if ( file_exists(__DIR__ . '/../../vendor/autoload.php')){
```

```
+ require_once __DIR__ . '/../../vendor/autoload.php';
+}
+
+
 /**
  * performs some checks on the underlying system, before initializing Tiki.
  * @package TikiWiki\lib\init
```

1.2.2. Use 2 composer files - Reverse Logic

The approach from "Use 2 composer files" can also be inverted and allow tiki to use the main project folder (as currently) and then use some folder like vendor_user for the user to have their own composer dependencies.
The key advantage of that approach is that no change needs to be done to tiki, but will be less intuitive for the user, and more error prone, if we move the user composer file to a subfolder

1.2.3. Use Pre-Packaged composer dependencies

For the cases where only FTP access is available, it should be possible to install packages using FTP.
For that we can leverage the existing directory `vendor_custom` and use the instructions bellow ("Packaging" composer packages)
Apart from the user action of uploading the dependencies to the vendor_custom folder, a change is required for tiki to autoload the "packages" from `vendor_custom`, like the one below:

```
Index: lib/init/initlib.php
===================================================================
--- lib/init/initlib.php  (revision 60173)
+++ lib/init/initlib.php  (working copy)
@@ -29,6 +29,15 @@

 require_once __DIR__ . '/../../vendor/autoload.php';

+foreach(new DirectoryIterator(__DIR__ . '/../../vendor_custom') as $fileInfo){
+  if (!$fileInfo->isDir() || $fileInfo->isDot()){
+    continue;
+  }
+  if (file_exists($fileInfo->getPathname() . '/autoload.php')){
+    require_once $fileInfo->getPathname() . '/autoload.php';
+  }
+}
+
 /**
  * performs some checks on the underlying system, before initializing Tiki.
  * @package TikiWiki\lib\init
```

1.3. Comparison of the Possible Approaches

Approach	Command Line Friendly	FTP Friendly	Web Friendly	Tiki knockout effect
Use 2 composer files	YES, you do composer require xx/yy and you are sorted	NO	YES, there may be some challenges, but we should be able to use a big enough timeout to run composer install commands.	Yes, some paths are hardcoded (frontend) to "/vendor", so they will need to be fixed
Use 2 composer files - Reverse Logic	NO, the user needs to go to a specific folder, but there is a composer file on the root folder, may cause some confusion	NO	YES, there may be some challenges, but we should be able to use a big enough timeout to run composer install commands.	NO
Use Pre-Packaged composer dependencies	MAYBE, if there is as part of the cli a option to download the package (but if you have command line access you might be better with the other options)	YES, you can download the packages from (let's imagine) some tiki package website, that might for instance allow packaging "on the fly" of any packagist package	YES, there may be some challenges, but we should be able to use a big enough timeout to run any command (like file_get_contents or similar) to download package and then uncompress from PHP.	NO

1.3.1. Suggested Approach

The suggested approach would be merging "Use 2 composer files" with "Use Pre-Packaged composer dependencies", this way we cover both command line access and ftp installation at the same time that we make the life of the tiki users as simple as possible.

1.3.2. Questions & Answers regarding the suggested Approach

1.3.2.1. Can composer (process) be run from the web ?

Yes. is a PHP software so we can either run it from the CLI as a standalone, or use it as a dependency and "call" from any PHP code by requiring the specific classes.
I think the deeper question is if we can build a web interface to composer, and the answer is yes, the only concern is if the server web user have the right permissions to write the files.

1.3.2.2. What is the point of renaming vendor to vendor_bundled ?

By doing that, users will use the main `vendor` folder together with a `composer.json` file in the root folder to manage their packages and a "special" folder called `vendor_bundled` would be used to hold the tiki `composer.json` and the tiki `vendor` folder.

This means that users can add and remove packages from tiki without touching the core tiki dependencies, and also they are going to find their dependencies on the location/folders that they are used to (like any other composer driven project)

1.3.2.3. Can we have 2 composer.json and just a vendor folder

Not that I'm aware.

Quick test, in a folder, create a `composer.json` like this:

```
{
    "name": "rmelo/2composer",
    "require": {
        "composer/semver": "^1.4"
    }
}
```

then create inside a new folder called `bundled`, add the following `composer.json` inside that folder:

```
{
    "name": "rmelo/bundle",
    "config": {
        "vendor-dir": "../vendor/"
    },
    "require": {
        "monolog/monolog": "^1.22"
    }
}
```

Now do `composer install` in the folder `bundle`. After move to the main folder, if you do `composer install` then you should see something like:

```
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
  - Removing monolog/monolog (1.22.0)
  - Removing psr/log (1.0.2)
  - Installing composer/semver (1.4.2)
    Loading from cache

Generating autoload files
```

That means that the "bundled" dependencies are removed when doing the second composer install.

1.3.2.4. What will users be doing with their own required packages

One of the key use cases is to be able to offer as option libraries like mPDF, the license of which doesn't allow to be bundled with tiki, features is tiki could allow the user to chose for instance mpdf as a main library to generate pdf if the user install the library manually.
As a opportunity, tiki could chose to bundle the support for a wide range of plugins like today, but don't

bundle the libraries, instead ask the user to install "on demand" when the user activate the feature in tiki.

1.4. Other possible approaches that were discarded

1.4.1. Using a composer plugin like Composer Merge

Wikimedia uses a specific merge plugin to allow people to add local packages, info at https://github.com/wikimedia/composer-merge-plugin, where you can add packages (manually) to a composer.local.json and this dependencies are merged with the ones installed from the main composer.json

Description

- Key arguments
  - Only one composer.json file commit to the repo, a sample "composer.local.json-sample" is shiped with wikimedia, users can rename the file and add the dependencies
- Key problems
  - Doing "composer require xxx/yyy" will write to the main composer.json file (so there will be collisions when upgrading from the version control software)
  - Users need to manually add packages to the "local" file
  - Only address the issue for users with command line access - Or web access with right permissions if there is a we console with the right permissions in the FS

1.4.2. Using Symfony approach

In Symfony, if you look at the standard distribution, it is build using a repo with one composer.json file (and the project skeleton), that the require the framework main composer.json file.

Description

- Key arguments
  - Split between "Framework" and "Project", local dependencies are installed in the project, while all framework packages and dependencies are tied together with the framework composer.json
- Key problems
  - If you do composer require, you are still writing into the composer file of the project (so SCM collisions)
  - Doing this in tiki (that is not really designed like Symfony using a Front Controller) will lead to a bigger "skeleton" and a split brain having the "gateway" files in the project (most of the files in the root folder) that have business logic also an then the framework (without routing, and all the business logic of the root files) in the framework

1.4.3. Template

Description

- Key arguments
  - Argument 1
- Key problems
  - Problem 1

1.5. Composer Tricks & Useful info

1.5.1. “Packaging” composer packages

1.5.1.1. “Packaging” Instructions

If you want to create a package with composer, that can be distributed easily, you can do the following:

Create a composer.json file (using mpdf with dependencies as example):

```
{
    "require": {
        "mpdf/mpdf": "^6.1",
        "setasign/fpdf": "^1.8",
        "setasign/fpdi-fpdf": "^1.6"
    }
}
```

Run:

```
$ composer install
```

Rename the vendor dir and package:

```
$ cp composer.json composer.lock vendor/
$ mv vendor tiki-package-mpdf
$ tar czvf tiki-package-mpdf.tgz tiki-package-mpdf
```

1.5.1.2. How to use “Packages”

After that you can use this package in your project, example:

```
(project structure)
/
/simple_script.php
/generic_script.php
/lib
```

Commands:

```
$ cd lib
$ tar zxvf ~/tiki-package-mpdf.tgz
```

Add the following to the simple_script.php file:

```
<?php
if(!class_exists('mPDF')){
  echo("Class mPDF is not available!\n");
}

require_once __DIR__ . '/lib/tiki-package-mpdf/autoload.php';
```

```
if(class_exists('mPDF')){
  echo("And now it is!\n");
}
```

Or a generic approach that includes all “packages” under lib, using generic_script.php

```
<?php
if(!class_exists('mPDF')){
  echo("Class mPDF is not available!\n");
}

foreach(new DirectoryIterator(__DIR__ . '/lib') as $fileInfo){
  if (!$fileInfo->isDir() || $fileInfo->isDot()){
    continue;
  }
  if (file_exists($fileInfo->getPathname() . '/autoload.php')){
    require_once $fileInfo->getPathname() . '/autoload.php';
  }
}

if(class_exists('mPDF')){
  echo("And now it is!\n");
}
```