Drifter Documentation

Release 1.2.0

Gilles Crettenand, Sylvain Fankhauser, Christian Stocker and con

Jan 23, 2021

Getting Started

Drifter is a framework to help provision developer boxes using Ansible and Vagrant.

CHAPTER 1

Goals

- Streamline our project setups
- Ease the “entry cost” for a new squad member
- Easy to use
- Lean: small codebase, easy to maintain and extend, focus only on Debian and Ubuntu
- Be adopted by Liip as a whole

CHAPTER 2

The idea behind the framework

The idea is to have a common ground for each project that can be improved over time, each project benefiting from the improvements.

This repository aims to contain multiples Ansible roles to manage the various part of the development stack needed to work on the various projects of Liip. If a someone need new roles, it is highly recommended that they are added to the common pool if they are deemed reusable.

Each squad can tailor its box to its need by modifying the Ansible playbook which should ultimately only contain role inclusion to maximize reuse.

When installed, Drifter creates a parameters file to hold various information about your project, a playbook file where you can choose what to install and finally a Vagrantfile where you can modify some Vagrant related parameters before the “main” Vagrantfile is included. This should offer enough flexibility for every project.

CHAPTER 3

What this framework is not ?

This framework does not aim to provide a way to deploy staging and production servers for your project. The roles are written with a development box in mind and are thus not fit for server provisioning. There are absolutely no security issues taken into consideration.

However, if your server is using a Debian based OS based on the stable release, both configurations should be close enough so that you won't run into issues.

CHAPTER 4

Intended Public

This project was first and foremost created to be used inside of Liip, but you are more than welcome to use it for personal projects or anywhere else you'd like to.

4.1 Requirements

- Vagrant >= 1.8.4
- Git >= 1.0

You also need a virtualization solution, either one of these:

- Virtualbox >= 4.3
- LXC >= 1.0 & vagrant-lxc >= 1.0.0.alpha.2

Optional dependencies:

- vagrant-hostmanager A Vagrant plugin that manages /etc/hosts files. (will be automatically used if installed, make sure it's at least 1.5.0 if you have it)

4.1.1 Install Requirements

Debian Stretch (testing) and Ubuntu Xenial 16.04

Open a terminal and run:

```
sudo apt-get install vagrant vagrant-lxc
vagrant plugin install vagrant-hostmanager
```

Older Debian and Ubuntu versions

Go to https://www.vagrantup.com/downloads.html to download and install the latest Vagrant version. Then open a terminal and run:

```
sudo apt-get install lxc redir     # this is needed for LXC provider
vagrant plugin install vagrant-lxc vagrant-hostmanager
```

Mac OS X

Download and install https://www.vagrantup.com/downloads.html.

Download and install https://www.virtualbox.org/wiki/Downloads.

Then open a terminal and run:

```
vagrant plugin install vagrant-hostmanager
```

You can also use `cask` to help with the installation::

```
brew cask install vagrant virtualbox
```

Windows

Install Virtualbox and Vagrant (>= 1.8.4) using the binaries available on their respective websites.

Also make sure that `core.autocrlf` is set to `input` (recommended) or at least `true` so that you don't get issues with Windows line-endings in the files that are in your box. You can set it by running the following command:

`git config --global core.autocrlf input`

For example if you get the following error when trying to provision the box:

```
TASK [base : ensure base packages are installed] ********************************

failed: [default] (item=[u'locales', u'procps', u'command-not-found', u'bash-
↪completion', u'zsh', u'bzip2', u'unzip', u'vim', u'ack-grep', u'highlight', u
↪'libxml2-utils', u'build-essential', u'wget', u'openssh-server', u'sudo', u
↪'imagemagick', u'iputils-ping', u'ncurses-term', u'python-pycurl']) => {"failed":␣
↪true, "item": ["locales", "procps", "command-not-found", "bash-completion", "zsh",
↪"bzip2", "unzip", "vim", "ack-grep", "highlight", "libxml2-utils", "build-essential
↪", "wget", "openssh-server", "sudo", "imagemagick", "iputils-ping", "ncurses-term",
↪"python-pycurl"], "module_stderr": ">>> /etc/sudoers.d/sudo-passwordless: syntax␣
↪error near line 1 <<<\nsudo: parse error in /etc/sudoers.d/sudo-passwordless near␣
↪line 1\nsudo: no valid sudoers sources found, quitting\nsudo: unable to initialize␣
↪policy plugin\n", "module_stdout": "", "msg": "MODULE FAILURE", "parsed": false}
```

That's because the sudoers file that gets copied in the box has the wrong format. Enabling `core.autocrlf` will fix the issue.

4.2 Usage

Drifter is going to be installed into your project as a git submodule. So if your project is not using Git as VCS, start by creating a git repo:

```
cd my-project && git init
```

Then to install Drifter, simply run the following command:

```
curl -sS https://raw.githubusercontent.com/liip/drifter/master/install.sh | /bin/bash
```

This will create a `Vagrantfile` in your root and a `virtualization` folder containing configuration files. You now have to follow those two steps:

- edit `virtualization/parameters.yml` to set parameters related to your project
- edit `virtualization/playbook.yml` to configure what to install in your box

You now just have to launch your Vagrant box and start hacking!:

```
vagrant up
```

4.3 Customization

You can customize what seems to us to be the most important options through two files:

- `virtualization/parameters.yml` for all project related parameters. Any value in this file will be passed to Ansible as a variable. You can override any role default values through this file. You can find details about possible parameters and values later in this documentation.
- `virtualization/playbook.yml` for provisioning. You can control which roles are used to build your box. This allows you to control what is installed in your box.

If those two mechanisms are not enough for you, you can also modify the `Vagrantfile`, but be aware that the risk of botching things up is far greater.

Currently you do not have a lot of control, but we will glad to add anything making sense to this file. Feel free to ask and we will comply ;)

4.4 Contributing

Before publishing your contributions please test your roles with the playground. To do so, go to the `playground` directory, enable any role you need in `playbook.yml` and set any parameter you want in `parameters.yml` and then run `vagrant up`. The box will use the roles of your working copy.

Please don't commit any change to the playground, unless you're fixing something in the playground.

4.5 System Roles

4.5.1 Base

This roles installs various useful software like vim, ack-grep, etc. It also put some configuration files of the vagrant user home directory.

It should always be included to have a common environment in all vagrant boxes.

4.5.2 Git

Install Git and some sane configuration and sync the username and e-mail from the host.

Fancy-diff is also installed by default and you can opt-in to sync your git configuration on each `vagrant up`.

Parameters

- **fancy_diff** : install fancy-diff
- **sync_git_with_host** : sync your host git config on each `vagrant up`

4.5.3 Supervisor

Install Supervisor so that you can manage long lived processes inside the box. A config file based on the parameters is also created for your service.

The service are automatically started on boot and restarted if they fail.

If you need multiple services, just include the role multiple times with the various parameters.

Parameters

- **service_name** : name of the service
- **user** : user to use to launch the service
- **command** : the command to launch
- **root_directory** : the base directory for the service
- **environment_vars** : environment vars you want to set

4.5.4 tmpfs

Configure a path to be mounted as a tmpfs (ie : in memory filesystem).

This can be used to speed up application, for example by putting their log directory in memory thus avoiding costly network transfers for shared directories.

Parameters

- **mount_path** : the path to replace with a tmpfs

4.5.5 SSL

If you set the `ssl` parameter to true in your `parameters.yml` file, ansible will create a Certification Authority (CA) and then create and sign SSL certificates for all hosts configured for your project.

The CA certificate will then be copied to your project `root_directory`. If you add this certificate to your trust store, you should be able to access your websites with HTTPS without any error messages from most browsers.

If the role is activated, both Apache and NGinx will be configured to use the created certificates.

WARNING: if the certificate is regenerated because you did a `vagrant destroy` or the hostname changed, you will need to re import the CA certificate into your trust store and in the meantime you might get errors from your browser. Chrome for example produce a pretty confusing error message about an attacker trying to steal your credentials.

4.5.6 SSH

Disable SSH strict host key checking if `ssh_no_stricthostkeychecking` is set to true in the parameters.

Also add the github and gitlab.liip.ch host key to the `known_hosts` file.

Parameters

- **ssh_no_stricthostkeychecking** : if set to true, disable SSH strict host key checking

4.5.7 Redis

To be completed.

4.5.8 RabbitMQ

To be completed.

4.5.9 LogStash

Currently only installs LogStach without any kind of configuration or nothing. This role is not usable as is.

Parameters

- **logstash_version**: version to install, defaults to 2.3

4.6 Webserver Roles

4.6.1 Apache

This roles installs Apache and the required virtual host configuration for your project.

Except for a static website, it should not be used directly because it is automatically included by other roles, for example PHP-Apache.

Parameters

- **web_directory** : Root directory for the virtual host, defaults to *root_directory*.
- **ssl** : Whether to activate HTTPS vhost, defaults to *false*. If enabled, the generated CA will be copied to the project directory.

4.6.2 NGinx

Install the NGinx web server and configure a virtual host based on the given site template. Except if you need to serve only static files, you should not have to include this role yourself, the Django or PHP-FPM roles do it automatically with the correct parameters.

The server logs are stored in `/var/log/nginx/<hostname>.(error|access).log`.

You can have your own site template in your project directory, for example *virtualization/templates/nginx.j2* and extend one of the default templates provided:

```
{% extends "default-site.j2" %}

{% block extra %}
    {{ super() }}

    # Here goes your custom Nginx rules
{% endblock %}
```

Then set the `site_template` parameter to `nginx.j2` when including the nginx role (or any other that depend on the nginx role):

```
roles:
    - { role: nginx, site_template: nginx.j2 }
```

If you want to use roles that include nginx, such as php-fpm, make sure you use the right parameter name (check the docs):

```
roles:
    - { role: php-fpm, nginx_site_template: nginx.j2 }
```

Parameters

- **site_template** : The virtual host template to use, defaults to "default-site.j2" for static websites only, possible values are:
- `default-site.j2`
- `django-site.j2` Site template for Django
- `drupal6-site.j2` Site template for Drupal6
- `drupal7-site.j2` Site template for Drupal7
- `drupal8-site.j2` Site template for Drupal8
- `php-site.j2` Site template for generic PHP
- `silex-site.j2` Site template for Silex
- `symfony2-site.j2` Site template for Symfony2
- `symfony4-site.j2` Site template for Symfony4
- **index** : what file do we use as an index ? defaults to 'false'
- **static_host** : Which static host to use for Django projects ? defaults to "false".
- **static_dir** : Which static URL dir to use for Django projects ? defaults to "false".
- **static_fs_dir** : Which static filesystem dir to use for Django projects ? defaults to "".

- **expire_time** : Expiration time of static files, defaults to “6h”.
- **web_directory** : Root directory for the virtual host, defaults to *root_directory*.
- **ssl** : Whether to activate HTTPS vhost, defaults to *false*. If enabled, the generated CA will be copied to the project directory.

4.7 Database Roles

4.7.1 MySQL

Install and set up a MySQL server and then create the configured user and database.

The database administrative user is “root” with the “root” password.

This role must be included before the Django or PHP one if both are present so that the correct extension and configuration could be made.

Parameters

- **database_name** : the name of the database to create, set in parameters.yml
- **database_user**: the name of the user, defaults to the database name
- **database_password**: the password of the user, defaults to the database name
- **mysql_version**: the MySQL version to install, defaults to 5.6 and supports 5.6, 5.7 and 8.0 (more info on http://dev.mysql.com/downloads/repo/apt/)
- **mysql_character_set**: the database character set, defaults to “latin1”
- **mysql_collation**: the database collation, defaults to “latin1_swedish_ci”

4.7.2 PostgreSQL

Install and set up a PostgreSQL server and then create the configured user and database.

This role must be included before the Django or PHP one if both are present so that the correct extension and configuration could be made.

Parameters

- **database_name** : the name of the database to create, set in parameters.yml
- **database_user**: the name of the user, defaults to the database name
- **database_password**: the password of the user, defaults to the database name
- **database_template**: the template to use, defaults to “template0”
- **database_encoding**: character encoding, defaults to UTF-8
- **database_lc_collate**: database collation, defaults to en_US.UTF-8
- **database_lc_ctype**: database ctype, defaults to en_US.UTF-8

4.7.3 PostGIS

Install and set up a PostgreSQL server with the PostGIS extension enabled.

The *postgres* role is declared as a dependency and does not need to be activated explicitly in *playbook.yml*.

For each of the supported OS, this role installs the recommended PostgreSQL/PostGIS combination package: - **Debian 8 (Jessie)**: postgresql-9.4-postgis-2.1 - **Debian 9 (Stretch)**: postgresql-9.6-postgis-2.3 - **Debian 10 (Buster)**: postgresql-11-postgis-2.5 - **Ubuntu 14 (trusty)**: postgresql-9.3-postgis-2.1 - **Ubuntu 16 (xenial)**: postgresql-9.5-postgis-2.2

4.7.4 MemCached

To be completed.

4.8 PHP Roles

4.8.1 PHP

Install PHP and various extensions : curl, intl, gd, imagemagick, . . .

The version can be changed and defaults to 5.6. All version are however not available on all OS versions, an error message will be displayed by Ansible if you chose an impossible combination.

Available versions are:

- **Debian Stretch & Jessie**: 5.6, 7.0, 7.1 and 7.2
- **Ubuntu Trusty**: 5.5, 5.6, 7.0, 7.1 and 7.2.

Development specific configuration options are also put into place, for example to activate error outputting.

A database driver is also installed if one of the MySQL or PostgreSQL roles was included before.

If you want to install xdebug, you'll need to also add the specific role : `php-xdebug`.

There are also roles for some more specific extension that could be found below.

This role is automatically included by roles PHP-Apache and PHP-FPM, so you should not include it yourself.

Parameters

- **php_sury_apt_key_id**: if you're installing PHP on Debian >= jessie, this parameter allows you to change the APT key id of the Sury repository. Defaults to B188E2B695BD4743
- **php_version** : version to install, defaults to 5.6
- **php_error_reporting** : php error reporting, defaults to "E_ALL | E_STRICT"
- **php_assert_exceptions** : php assert exceptions for 7.0 and above, defaults to false
- **php_max_execution** _time** : script max exectution time, defaults to "3600"
- **php_memory_limit** : memory limit, defaults to "4G"
- **php_upload_max_filesize** : maximal size of uploaded file, defaults to "128M"
- **php_date_timezone** : timezone, defaults to "Europe/Zurich"
- **php_default_charset** : default charset, defaults to "UTF-8"

- **php_default_socket_timeout** : socket timeout, defaults to 120

4.8.2 PHP-Apache

Install the PHP mod for Apache along with Apache and PHP. You only need to install this role, PHP and Apache will be automatically added as dependencies. For details about PHP config, see above.

The default vhost template from the Apache role is used.

4.8.3 PHP-FPM

Install PHP-FPM so that you can use NGinx. You only need to install this role, PHP and NGinx will be automatically added as dependencies. For details about PHP config, see above.

You can change the site template used using the parameter defined below. The templates can be found in the NGinx role.

Parameters

- **nginx_site_template**: template to use for site configuration, defaults to “php-site.j2”
- **nginx_index**: index in nginx configuration, defaults to “index.php”

4.8.4 PHP-XDebug

Installs the XDebug extension for PHP.

You can modify the config file `/etc/php5/conf.d/20-xdebug.ini` to change the configuration and restart your Apache or PHP-FPM. XDebug is also configured to trigger debugging and profiling in response to the related query string or cookie, so you should be able to install a browser extension to make it work this way.

Parameters

- **xdebug_idekey**: value of the `xdebug.idekey` setting, defaults to *XDEBUG-DRIFTER*.

4.8.5 PHP-Redis

Installs the Redis extension for PHP. Redis and PHP are installed as a dependency.

Concerning Redis itself, the documentation is in the “System” section of the documentation.

4.8.6 PHP-MemCached

Installs the MemCached extension for PHP. MemCached and PHP are installed as a dependency.

Concerning MemCached itself, the documentation is in the “System” section of the documentation.

4.8.7 Composer

Installs Composer, the PHP package manager. The PHP role is defined as a dependency. You can set the install dir, a link in `/usr/local/bin` will be set up whichever the install dir is so that composer can be accessed globally.

If composer is already installed, this role will update it instead.

Parameters

- **composer.dir** : where to install the binary, default "opt/composer"

4.8.8 PhiVE

Installs PhIVE support (Phar Installation and Verification Environment (PHIVE). A link in `/usr/local/bin` will be set up so that `phive` can be accessed globally.

If PhIVE is already installed, this role will update it instead.

Parameters

- **phive.dir** : where to install the binary and the downloaded phar(s), default to "opt/phive"

4.9 Python Roles

4.9.1 Python

Install Pip and Virtualenv along with dev dependencies. Dependencies to build the Pillow package are also installed.

Both Python 2 and Python 3 are always installed, for example to facilitate tests on multiple python version, the parameter below only change the behavior of python related roles.

Parameters

- **python_version**: version of Python to use. Can be 2 or 3, defaults to "3"
- **pip_version** : the version of pip to install in the virtual environment. Defaults to 9.0.1.
- **setuptools_version** : the version of setuptools to install in the virtual environment. Defaults to 28.8.0.
- **python3_install_from_source**: whether to install Python from source (true) or use the distribution version (false). Defaults to false
- **python3_source_version**: Python version like 3.5.5, defaults to "3.6.5"

4.9.2 Virtualenv

Create a python virtual environment and install application requirements via pip. The environment will also get pip-tools installed.

The virtual environment is automatically activated upon box login.

- **pip_requirements** : filename of the requirements file, defaults to "requirements/dev.txt"

- **env_root** : directory where the virtual environment must be created, defaults to "~/ENV"
- **pip_requirements_dir** : name of the requirements directory that contain the *.in* files. If set, Drifter will run `pip-compile` on these files upon provisioning.
- **pip_tools_version** : the version of pip-tools to install in the virtual environment. Defaults to 1.8.2.

4.9.3 Django

Uses the `virtualenv` or the `pipenv` role (depending on the `django_use_pipenv` parameter) to create and install a virtual environment for Django.

Configure database access via environment variable and then run migrations.

You need to include either to `mysql` or `postgresql` roles before this one.

This role depends on the Virtualenv and NGinx roles. The NGinx role is configured to use the "django-site.js" site template on the port "8000".

Parameters

- **django_root** [root directory of the Django project, default to] the "root_directory" variable defined in parameters.yml
- **django_use_pipenv**: whether to use Pipenv to install requirements. Defaults to false.
- **django_use_virtualenv**: whether to use Virtualenv to install requirements. Defaults to the opposite of **django_use_pipenv**. If both **django_use_pipenv** and **django_use_virtualenv** are false, you're responsible for installing your project requirements.

4.10 Ruby Roles

4.10.1 Ruby

Install `rbenv` along with `Bundler`. Complete the installation with the `bundle install` command if a Gemfile is found in the project root directory.

Parameters

- **ruby_version**: this should be the exact version name (such as 2.3.3). Find a list of accepted version with `rbenv install -l`. Default is 2.4.1.
- **ruby_build_version**: version of ruby-build to use. Default is v20190423.

4.10.2 Rails

Simply add roles `nodejs` and `ruby` in your playbook.yml. Note that rails will not be installed unless specified in your Gemfile.

Using mysql or postgres? then include `mysql` or `postgresql` role before `ruby`.

Run server

You have two options. First, in the box, run `rails server` or `puma`, then open your browser on `http://{hostname}:3000`

Second option is to add the nginx role with the rails template:

```
- { role: nginx, web_directory: "/vagrant/public", site_template: "rails-site.j2",␣
↪proxy_port: 3000 }
```

Then you can just open `http://{hostname}`.

4.11 Java Roles

4.11.1 Java

Installs a Java Runtime Environment using the OpenJDK Debian package.

Parameters

- **java_jre_version**: JRE version to install, defaults to 7. Set your version according to your needs and your Linux distribution.
- **java_jre_package**: default is `openjdk-{{ java_jre_version }}-jre`.

4.11.2 JDK

Installs a Java Development Kit using the OpenJDK Debian package.

Parameters

- **java_jdk_version**: JDK version to install, defaults to 7. Set your version according to your needs and your Linux distribution.
- **java_jdk_package**: default is `openjdk-{{ java_jdk_version }}-jdk`.

4.11.3 Maven

Installs Maven via `apt-get`.

4.11.4 Solr

Install `solr` via the tarballs available on the Apache repository.

A specific user is created and `solr` is automatically started at boot using `supervisor`.

You can choose any `solr` version (compatible with Java7) via download. However the provided start command might need some adjustment.

To create a Solr core, use both the `solr_core_name` and `solr_core_conf` parameter.

Parameters

Those parameters controls base feature for `solr`. There's also a list below of "internal" parameters that you'll might need to tweak if you want to use a version different than 4.X or 5.X

- **solr_version**: Solr version to install, defaults to 5.3.1. You should be able to use all 5.X and 4.X version, but some tuning might be needed.
- **solr_base_dir**: Solr base directory, this is not directly used by the role, defaults to */opt/solr*.
- **solr_install_dir**: Solr installation directory, defaults to `{{ solr_base_dir }}`.
- **solr_config_dir**: Solr configuration directory, defaults to */opt/solr/server/solr*.
- **solr_port**: defaults to 8984.
- **solr_core_name**: Create a new Solr core/index with such name; by default no indexes are created. If this parameter is defined, `solr_core_conf` must be defined as well.
- **solr_core_conf**: Specifies the Solr core/index configuration folder to use for the index, it will be symlinked to the *conf* folder of the index. Refer to the documentation for the file structure required by Solr. Example: `solr_core_conf=/vagrant/solr/conf`.

4.11.5 ElasticSearch

To be completed

4.12 Webpack

Current Webpack version: 4.5.0

This role provides a pretty simple setup to handle assets (javascripts, stylesheets, images, fonts and SVG icons) through Webpack in your project.

It creates a *webpack.config.js* that is preconfigured to handle:

- Sass files to create stylesheets. Stylesheets are processed through Autoprefixer for browser compatibility and CSSNano for optimisations.
- JavaScript files through Babel with babel-preset-env to use next generation JavaScript today
- SVG icons through svg-sprite-loader and make a single sprite file out of it
- Images (svg, png, jp(e)g, gif, webp)
- Fonts (woff, woff2, eot, ttf, otf)

4.12.1 Installation

Once enabled, this role will create a `webpack.config.js`, a `babel.config.js` and a `package.json` that includes all the required dependencies by default.

Existing files (webpack.config.js, babel.config.js and package.json) will not be overridden. If those files already exist, the installation will be incomplete and might not work as expected.

Parameters

- **webpack_directory**: where should the webpack.config.js be created, defaults to `{{ root_directory }}/`
- **webpack_create_config**: Create the webpack.config.js & babel.config.js, defaults to `true`
- **webpack_browserslist**: Define Browserslist in `package.json`, defaults to:

  ```
  - "> 0.5%"
  - "not op_mini all"
  - "not dead"
  ```

Post-install

The default configuration expects a couple of things:

- The main JavaScript file to live at `assets/scripts/common.js` (not created by the role)
- All the assets to live in `assets/...` or in `node_modules`
- The SVG icons to be included in the sprite to live in `assets/icons/`

Other defaults:

- Built files are bundled into the `dist/` folder
- Generated icons sprite is named `icons.svg`

You can change all these default values by editing the `webpack.config.js` file. If you need help, you should check out the Webpack config documentation.

4.12.2 Default tasks

Development

```
npm start
```

Starts `webpack-dev-server` at example.lo:3000, compile on-the-fly and reload the browser automatically. All requests not handled by webpack will be proxified to example.lo.

Replace example.lo by the "hostname" you set in the "parameters.yml".

Notice that webpack-dev-server does not write the files to the disk. To debug which files are being served, go to example.lo:3000/webpack-dev-server.

Production

```
npm run build
```

Will bundle all the assets, optimized for production, in the `dist` folder by default.

4.12.3 Loading assets

The default public path for bundled assets is `/`.

To load the main JavaScript file, use:

```
<script type="text/javascript" src="common.js"></script>
```

CSS styles are extracted to a separate file. For example, all the CSS required by the `common` bundle, would be extracted as `common.css` and should be loaded like this:

```
<link rel="stylesheet" type="text/css" href="common.css">
```

If you need help to import files such as CSS, images or fonts, take a look to the Webpack asset management guide.

4.13 Gulp Role

Existing configuration files (Gulpfile.js, gulp.config.js, webpack.config.js, package.json) will not be overridden.

- Install `gulp` globally in the Vagrant box
- Create a prefilled `Gulpfile.js` with useful tasks
  - Watch & live reload with BrowserSync
  - Compile Sass with Autoprefixer & source-maps
  - Bundle JavaScript with Webpack, preconfigured with Babel (optional)
  - Lossless images optimization with ImageMin
- Create associated `gulp.config.js` and `webpack.config.js`
- Add the necessary dependencies to `package.json` (only if the file doesn't exist yet)

After the first provisioning, you should edit the `gulp.config.js` and `webpack.config.js` to match your project structure.

4.13.1 Parameters

- **gulp_directory**: where should the gulpfile be created, defaults to `<root_directory>/`
- **gulp_create_config**: Create the gulp.config.js used by the default Gulpefile.js, defaults to `true`
- **gulp_use_webpack**: Setup Webpack alongside Gulp, defaults to `true`
- **gulp_use_purescript**: Add PureScript support to Webpack, defaults to `false`
- **gulp_browserslist**: Define Browserslist in `package.json`, defaults to:

  ```
  - Last 2 versions
  - IE 11
  ```

4.13.2 Default tasks

Watch & live reload proxy

Run BrowserSync, watch for changes in files, compile and reload browser afterwards with:

```
npm start
```

Build for production

```
npm run build
```

Optimize images

For performance reason, this task is not included in the watch/build tasks. You should run it manually according to your needs.

Optimize jp(e)g, png, gif & svg files with:

```
gulp images
```

4.14 Browser Roles

Browser roles are available for frontend testing. They all depend on the role xvfb, which is a headless X server, except for Phantomjs.

4.14.1 Firefox

Install Firefox with Geckodriver to be used with selenium.

Parameters

- **firefox_version**: The version of Firefox to be installed, defaults to *latest*. Should be greater than 47.0. The full list of supported versions can be found on the Firefox releases page.

4.14.2 Chrome

Install Chrome with Chromedriver to be used with selenium.

There are no parameters available for this one, as they don't really provide older versions. Therefore, always the newest Chrome browser will be installed.

4.14.3 PhantomJS

Install PhantomJS.

Parameters

- **phantomjs_version**: The version of PhantomJS to be installed, defaults to *2.1.1*. The full list of supported versions can be found on the PhantomJS releases page.

4.14.4 Example usage with pytest and splinter

This is the recommended way to use those browser for testing in python with *pytest*. *pytest-splinter* is a pytest plugin that provides easy access to several webdrivers.

First, you have to install some packages via pip (see *Virtualenv* for instruction on how to properly do this):

- pytest
- pytest-splinter
- pytest-xvfb (When you want to use firefox or chrome)

In order to run your tests, you can simply invoke pytest. By default the Firefox webdriver will be used, but it's possible to change this with the option –splinter-webdriver=chrome. More info available on the pytest-splinter project page.

4.15 Other Roles

4.15.1 Ruby

Install Ruby, Gem integration for Debian and dev dependencies.

Any Debian ruby package should then be also recognized as a Gem. You can however continue to install Gems using the `gem` utility if you need a specific version or an unavailable package.

4.15.2 NodeJS

Install NodeJS and NPM.

Parameters

- **nodejs_version** : The version to install, currently supports 13.x, 12.x, 11.x, 10.x, 9.x, 8.x, 7.x, 6.x, 5.x, 4.x, 0.12 and 0.10, default being 12.x.
- **nodejs_distro** : Is automatically set to either 'jessie', 'stretch', etc based on available information, you can also put an Ubuntu codename here.
- **nodejs_create_package_json**: create a `package.json` file based on the settings below during provisioning. Defaults to `true`.
- **nodejs_package_json_template**: template to use for the creation of the initial `package.json` file. Defaults to

  `package.json.j2`, or `package.json.gulp.j2` if you're using the gulp role. See the `provisioning/roles/nodejs/templates` directory for the list of available templates.
- **nodejs_package_json_path**: where should the package.json file be created, defaults to `<root_directory>/package.json`
- **nodejs_package_json_author**: Author that should be put in the package.json file, defaults to `Liip AG`
- **nodejs_install_package_json**: Run `npm install` on each provisioning. Defaults to `true`.

4.15.3 OpenLDAP

Install an OpenLDAP (slapd) server.

It will open the standard LDAP ports (389 for `ldap://`, 636 for `ldaps://`), and the `ldap-utils` (shipping `ldapsearch` is also installed.

Parameters

- **ldap_organization** : Fulltext organization name, defaults to 'EvilCorp Ltd'
- **ldap_organization_domain** : Organization domain name, defaults to `evilcorp.example.com`
- **ldap_admin_password** : Password of the original cn=admin,dc=evilcorp,dc=example,dc=com user, defaults to 'admin'

4.15.4 RMT - Release Management Tool

Install RMT in the box. Once done you must run *php /home/vagrant/.config/composer/vendor/liip/rmt/RMT* to init it for your project. Then for the next steps go to https://github.com/liip/RMT#usage

4.15.5 Redis

To be completed.

4.15.6 Gitlab CI

See *CI*.

4.16 PHP

4.16.1 PHP Debugging with Drifter & PHPStorm

As said earlier, the PHP role installs php-xdebug which is configured to try to connect to any listener on the host.

If you are using Chrome and PHPStorm, debugging a script can be done by following these steps:

1. Install the Xdebug helper chrome extension
2. In your browser address bar, click on the little bug and select "Debug"
3. In PHPStorm, open the "Run" menu and select "Start Listen for PHP Debug connection"
4. Reload the page in your browser
5. A dialog should open in PHPStorm to ask you which file you want to debug, choose the entry point of your application

4.17 CI

4.17.1 Integrating with Gitlab CI

(These instructions are specifically for the Liip Gitlab CI, but may be used on other Gitlab CI's as well)

To make it easy to run tests on the Gitlab CI runners, this packages provides some general purpose scripts, so you don't have to reinvent the wheel all the time.

Setup

Using the gitlabci role

The automatic way. Just uncomment/add the folloing line in your `playbook.yml` and the needed files will be created automatically on the next provsioning (if they don't exist already)

```
- { role: gitlabci }
```

It installs the following files (which should be added to git afterwards)

.gitlab-ci.yml

The config file for Gitlab CI, it tells the CI what exactly to run

scripts/gitlabci.sh

The script called first by the gitlab runner. It updates the submodules and then calls `./virtualization/drifter/ci/start.sh`, which does start vagrant, provisions it and calls your actual test script.

scripts/run_tests.sh

This is where your actually test calls go. This is run within your vagrant box.

virtualization/provisionbuild.dat

To prevent provisioning on the ci runners all the time and save lots of time, provisioning is only run, when this file changes. Therefore if you change something in your provisioning scripts, also changes this file to a different value (doesn't matter which one, as long as it's different, but some kind of timestamp assures that it's different)

If you don't add this file to your project, then provisioning will be run every time a ci build is started.

Be aware, that the CI deletes all files before each run, which are not in your git repository. This eg. means that your vendor (if you use composer) or node_modules folders are gone and not recreated, if provisioning doesn't go through. To still keep your important directories, add this to .gitlab-ci.yml (see also https://docs.gitlab.com/ce/ci/yaml/#cache for more details)

```
cache:
  paths:
    - bin/
    - vendor/
  key: sharedcache
```

Using a different folder than scripts/

If you prefer to install those files in a different folder than scripts/ you can add the following line in your `parameters.yml`, eg:

```
ci_scripts_folder: bin/
```

You can also adjust the files afterwards and uncomment the gitlabci role in `playbook.yml` again (otherwise the files will be created again after each provisioning)

Installing it manually

Copy the following files to some location (we have a /scripts/ folder, but you can choose any directory)

```
SCRIPTS_FOLDER=./scripts/
cp virtualization/drifter/provisioning/roles/gitlabci/templates/gitlab-ci.yml .gitlab-
↪ci.yml
cp virtualization/drifter/provisioning/roles/gitlabci/templates/gitlabci.sh $SCRIPTS_
↪FOLDER/gitlabci.sh
cp virtualization/drifter/provisioning/roles/gitlabci/files/run_tests.sh $SCRIPTS_
↪FOLDER/run_tests.sh
date +%Y%m%d%H%M%S > virtualization/provisionbuild.dat
```

And adjust `.gitlab-ci.yml` and `$SCRIPTS_FOLDER/gitlabci.sh` with the corrects paths.

Add your tests

Put your test scripts into `$SCRIPTS_FOLDER/run_tests.sh` and they should be run the next time you push something to gitlab (also make sure you enable one of the go-based gitlab runners for your project, the ones labeled with “go”, “shell”, and “lxc”)

You can also use any other file, but then adjust the env variable `CI_TEST_SCRIPT` in $SCRIPTS_FOLDER/gitlabci.sh

Customization

Global project cache

On each runner, there’s a global project cache (shared with all projects), which can be mounted, uncomment `export DO_GLOBAL_PROJECTS_CACHE=true` in `$SCRIPTS_FOLDER/gitlabci.sh` and that will be mounted into `/home/vagrant/.projects_cache`. We for example add the php composer cache dir there into `/home/vagrant/.projects_cache/composer_cache`, so that not every project has to download the same project all over again.

NPM would maybe be another canditate.

As this is shared with all projects, be careful where to put things there.

4.18 Running and writing tests

The `box` pytest fixture allows you to get a full fledged Vagrant box (powered by LXC). Start by provisioning it, and then run commands in the box using the `execute` method:

```
def test_mysql_role_installs_mysql(box):
    box.provision(roles=['mysql'], parameters={'mysql_version': '5.7'})
    assert '5.7' in box.execute('mysql --version')
```

By default boxes use a specific Debian image (refer to `tests/conftest.py` for the exact distribution). You can specify the OS to use by passing the `os` argument to the `provision` method:

```
def test_mysql_role_installs_mysql_on_ubuntu(box):
    box.provision(roles=['mysql'], parameters={'mysql_version': '5.7'}, os='drifter/
↪trusty64-base')
    assert '5.7' in box.execute('mysql --version')
```

To run the tests, start by installing the requirements:

```
pip3 install pytest pyyaml
```

And then execute the `pytest` command to run the tests. Test boxes are automatically discarded when the test run is over so you don't have to clean anything.

4.18.1 Running a specific test / debugging

When a test fails, you can either re-run only the failing tests by passing the `--lf` option to pytest, or by using the `-k` option, followed by a part of the name of the test (for example `pytest -k mysql_role_installs`).

If you want to break at a failing test (for example to spawn a shell into the box and check what's going on), add the `--pdb` option to pytest and, once you're into pdb, retrieve the box id:

```
(Pdb) p box.box.get_lxc_id()
drifter-base-boxes_default_1519465979666_71466
```

Then run `lxc-attach -n drifter-base-boxes_default_1519465979666_71466` to get a shell to the box. Once you're done, use the `q` command to exit the debugger and destroy the box.

4.18.2 Passwordless tests running

To run the tests without any password (useful for CI integration), add the following to your sudoers (replace *johndoe* by your user name):

```
johndoe ALL=(ALL) NOPASSWD: /usr/binlxc-info -iH -n *, /usr/bin/lxc-start -n *, /usr/
↪bin/lxc-stop -k -n *, /usr/bin/lxc-attach -n * -- *, /usr/bin/lxc-copy -s -B␣
↪overlayfs -n * -N *, /usr/bin/lxc-stop -k -n *, /usr/bin/lxc-destroy -n *
```

4.19 The future ?

The framework will evolve as we use it on more projects. It is not the goal to refrain you from doing anything. It will be improved as we need it, the goal is to serve Liip teams !

What could be done if the need arise :

- Better installer with questions to automatically create the config files instead of manual editing

4.20 Create boxes

4.20.1 Current way

Used for the current boxes available on https://vagrantbox-public.liip.ch/

See https://gitlab.liip.ch/liip/drifter-base-boxes

Ansible

If you plan on using the `ansible_local` provisioner, `ansible` must be installed in the box with at least a version of 1.9.0 otherwise the roles won't work.

4.20.2 Other way (older, may still work)

LXC

```
git clone https://github.com/team-rawbot/vagrant-lxc-base-boxes
cd vagrant-lxc-base-boxes
make jessie
```

If you're getting errors when trying to install the base packages, check your default LXC config (`/etc/lxc/default.conf`) and adapt it to your setup:

```
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
```

VirtualBox

Install veewee and then:

```
git clone https://github.com/team-rawbot/veewee-definitions definitions
veewee vbox build liip-jessie64
veewee vbox export liip-jessie64
```

4.21 Migration instructions

4.21.1 Version 2.0

Ansible version

This version of Drifter requires Ansible version >= 2.7 installed. If you're using `ansible_local = true` in your *Vagrantfile* (which is the default), you'll need to make sure Ansible 2.7 is installed on the guest. If you have an existing Vagrantfile in your project root, check if the default version is set to something old. If it is set, edit the line to say:

```
@ansible_version = config['ansible_version'] || "2.7.0"
```

You can also specify the parameter in `virtualization/parameters.yml` as:

```
ansible_version: 2.7.0
```

Then run the provisioning using `vagrant provision`, this should install the correct Ansible version.

If you're using `ansible_local = false`, you'll need to make sure the Ansible version installed on the host is at least 2.7. Instructions will depend on how you installed Ansible (OS package manager, pip, etc).

Ansible templates location

The new Ansible version changed the way templates are discovered. You might have templates paths set in your application, especially if you extend a Drifter template in one of your templates (eg. `{% extends "nginx/templates/default-site.j2" %}`).

In such cases, you'll need to replace the template path with only the template name. For example `nginx/templates/default-site.j2` would become `default-site.j2`.

4.21.2 Version 1.0

Ansible version and `ansible_local`

Changes were made to the roles that requires to use of at least the version 1.9.0 of `ansible`. This means Debian stable users have to install `ansible` via the Backports if they don't want to use the `ansible_local` provisioner.

Also, the default is now `ansible_local` also for LXC and the Vagrant version was bumped to 1.8.4 in this case to get rid of the bug that caused issues before. This is the new recommanded provisioner.

Old Vagrantfile format

The support for the old Vagrantfile format has been removed in this version. You should follow the steps detailed in the migration from 0.1.0 to 0.2.0 if you haven't done it already.

Virtualbox and LXC URLs

It's not possible to specify separate boxes for LXC and Virtualbox via Drifter anymore. You need to move to the new JSON box format in order to be able to do it. You can have a look at https://vagrantbox-public.liip.ch/drifter-jessie64-base.json for an example.

The `lxc_box_name`, `lxc_box_url`, `vbox_box_name` and `vbox_box_url` have been removed in favor of `box_name` and `box_url`.

Git

Git installation and configuration is now in its own role. It was added to the `playbook.yml.dist` file, but existing project should also add it to their playbook if they want to have git installed.

You should also have a look at the `git` role documentation inside the System roles for the new features.

PHP roles names

`redis-php` and `memcached-php` roles have been renamed to follow the already in place convention. You'll now have to use `php-redis` and `php-memcached`

Flash & Django roles

The `flask` and `django` roles now use the new `virtualenv` role. This means the parameter for the requirements is now named `pip_requirements`.

The default value for this parameter has also been changed to "requirements/dev.txt".

4.21.3 Version 0.1.0 to 0.2.0

In order for the framework to work correctly on Windows, we removed the symlinks to the Vagrantfile stored in the submodule. The content of the VagrantfileExtra.rb is now in the Vagrantfile at your project root. This new file then loads a more complete Vagrantfile that is in the submodule.

In order to migrate, follow those steps (commands assume you are in your project root directory) :

1. Remove the Vagrantfile symbolic link from the root : `rm -f Vagrantfile`
2. Copy the VagrantfileExtra.rb file to your root and rename it : `mv virtualization/VagrantfileExtra.rb Vagrantfile`
3. Add the `get` method to your `CustomConfig` class in the `Vagrantfile` (copy the snippet below inside the class)
4. Add the loading of the complete Vagrantfile to the project Vagrantfile : `echo "load 'virtualization/drifter/Vagrantfile'" >> Vagrantfile`

   def get(name, default = nil) if self.respond_to?(name) self.send(name) elsif default.nil? raise "[CONFIG ERROR] '#{name}' cannot be found and no default provided." else default end end