Chassis Documentation

Release 5.4.2

Ryan McCue & Bronson Quick

May 07, 2024

CONTENTS

Chassis is an open source virtual server for your WordPress site, built using Vagrant. (*Quickstart*)

Many people are still using their main computer for local development. However, this can be a huge pain when bringing new developers up to speed on a project, or trying to get non-developers (designers, managers, clients) running a site.

The concept of virtual machines for servers has become popular recently, but too many of these projects are focussed around specific uses, and include many things not everyone needs. They can end up massively increasing development time by introducing long waiting times, or huge context switches.

Chassis takes care of setting up a local server in an optimal configuration for WordPress, and removes all the hard work. It's fast (remember the Famous Five Minute Install?), and flexible to allow you to build how you want to.

Getting started is as easy as downloading Chassis and running a single command. (We're working on making it possible to run Chassis without ever touching the command line, but we're not quite there.)

You can contribute to Chassis on Github.

CHAPTER

ONE

USER GUIDE

1.1 Introduction

1.1.1 A Story

For a long time, you've had PHP running directly off your computer. You've had some combination of MySQL and Apache running too, all of them set up and working, but you have no idea how. Maybe they were already installed on your computer.

Everything's plodding along until one day, your host emails you to say they're upgrading you to PHP 5.5. You know you should test your code to make sure nothing breaks, but you have no idea how to change it. You try to follow a guide online by typing out the cryptic commands, but it never works for you.

Weird errors start popping up on your live site. Warnings about things that you don't get locally. Obscure bugs caused by something out of your control.

You've heard about this thing called "Vagrant", so you try setting that up. Problem is, you don't know the workings of Linux system administration. You try an existing project, but give up after waiting 30 minutes for it to set up.

Enter Chassis.

1.1.2 Philosophy

Chassis has a few important philosophies that guide it:

1. **Chassis is designed for everyone**

   Developers may be at home on the command line, but not everyone is. Regardless of your skill set, Chassis should be able to get you from nothing to a working WordPress install. More than that, if you're working on a project with a team, it should be able to get you set up to work on the project with minimal fuss.

2. **Chassis cannot be everything to everyone**

   We follow a minimalistic philosophy with Chassis. While some projects like VVV include basically every tool you might need, we only include the essential tools needed to run WordPress.

   We believe that if you're selling a flashlight, you should sell it with batteries included, but without the rest of the hardware store. Of course, if you want the other tools, we've included a flexible *extension system* to allow grabbing the rest.

   As a side-effect, this means Chassis is **fast**. The current initial boot time (after a fresh clone) is **under 3 minutes**.

3. **Chassis should be invisible**

   Once you've set up Chassis, we want to make sure we're out of your way as much as possible.

You should never need to think about how Chassis works, or how to set up databases, or adding host entries for DNS. We take care of all of this for you. The exact way that we set up and run the server shouldn't matter to you, unless you want to dive in and customise it.

We also never touch your codebase. Chassis keeps your plugins/themes in a `content/` directory and allows you to add extra configuration in this directory. However, we **never** touch this directory, allowing you to feel completely safe while using Chassis.

1.2 Quickstart

So you want to get Chassis running? Fantastic! We'll be here to guide you through the process.

For now, we're going to speed through some of these commands to get you started as quickly as possible. We've got a *command reference* waiting for you later.

Note: Psst, if you haven't seen the notation before, lines starting with `$` indicate commands you can type in, and other lines are output. Don't include the `$` when typing!

1.2.1 Prerequisites

Before using Chassis, this is how your system should be set up:

1.2.2 MacOS Intel chips

- Install VirtualBox
- Install Vagrant

1.2.3 MacOS Apple Silicon (M1/M2/M3) chips

- Install Parallels Desktop for Mac Pro Edition
- Install Vagrant
- Install the Vagrant Parallels plugin:

```
$ vagrant plugin install vagrant-parallels
```

- Change the Base Box mode to be `_mode: base` in one of your configuration files.

1.2.4 Windows

- Install VirtualBox
- Install Vagrant
- Make sure you have Zeroconf networking (Bonjour) set up:

If you have Bonjour Print Services or Creative Cloud installed, you already have Bonjour.

Otherwise, you need to install Bonjour on your system. The easiest way to do this is to install Bonjour Print Services. If you'd prefer not to do this, you can follow these instructions to install just Bonjour.

We also have some additional troubleshooting information.

1.2.5 Linux

- Install VirtualBox
- Install Vagrant
- Make sure you have Zeroconf networking (Bonjour) set up:
- You need to have Avahi installed on your system.

  For Ubuntu:

  ```
  $ sudo apt-get install avahi-dnsconfd
  ```

1.2.6 Installing

1. Clone the Chassis repo:

   ```
   $ git clone --recursive https://github.com/Chassis/Chassis <myproject>
   ```

   If you forget `--recursive` then run:

   ```
   $ git submodule update --init
   ```

   Note: Replace `<myproject>` with your preferred directory name.

2. Install your WordPress project:

   - **If you have an existing project**:

     Clone the content/ directory!

     ```
     $ cd <myproject>
     $ git clone git@github.com:yourcompany/yourproject.git content
     ```

   - **If you are starting a new project**:

     You will need to create a content folder:

     ```
     $ cd <myproject>
     $ mkdir -p content/{themes,plugins}
     ```

3. Boot up a Virtual Machine:

   ```
   $ vagrant up
   ```

   - **Windows**: Open a console with administrator privileges (Right-Click->Run as Administrator) and use this console to run `vagrant up`.

   Note: New Vagrant and VirtualBox users may see this error: `Stderr: VBoxManage: error: DHCP server already exists`

This error typically occurs when Vagrant and VirtualBox are both fresh installs, and you haven't used the networking tools before. This can be fixed by upgrading to Vagrant 1.7.0 or newer. (For older versions, a workaround is available)

4. Make a copy of `local-config-sample.php` and rename to `local-config.php`
5. Browse to http://vagrant.local and you should see your site! The default login credentials are `admin` and `password` and the login URL is http://vagrant.local/wp/wp-admin.

1.2.7 What's in the box?

By default we want to keep Chassis lean, below is a list of what we include:

- WordPress (latest stable version)
- PHP (version 8.1) (includes the cURL and GD extensions)
- nginx
- MySQL
- WP-CLI - A command line tool for WordPress.

Some tools including Git and cURL are installed during setup, but you shouldn't rely on these being available. Many more are available as default Ubuntu utilities.

Note that some tools like phpMyAdmin and Memcache are available instead as *extensions*, which are installed separately to keep Chassis fast.

Rather than providing everything under the sun, we provide a set of sensible defaults, along with the ability to change this as needed. This helps keep Chassis fast by designing for the common use-case first. This flexibility comes from two core parts: *configuration*, and *extensions*.

1.2.8 Updating

If you ever want to update Chassis, here's a quick two-step procedure:

```
# Pull and rebase (in case you have project-specific commits)
git pull --rebase

# Update submodules (Puppet modules)
git submodule update --init

# Update WP
git pull
```

Reinstall

During the development of plugins, it's often necessary to fill the WordPress databases with a bunch of test content, often using the *wp * generate* commands. You will often want to reset your WordPress install back to a clean state and not want to do a full *vagrant destroy -y && vagrant up* to do so. We have made a script that you can run to do this for you.

To run the script, simply run the following command:

```
$ vagrant ssh
$ sh reinstall.sh
$ Drop all WordPress tables and reinstall? [y/N] y
```

This will drop all WordPress tables and reinstall WordPress.

1.3 Configuration

The core of Chassis is designed around you having everything needed for your site-specific code in the `content/` directory. This includes project-specific configuration (we load `content/config.php` in during `wp-config.php` loading) and your plugins and themes. This is designed such that you can keep your site self-contained in a separate repository, including API keys and such.

We also load 5 configuration files, allowing you to store project-specific configuration while still allowing overriding:

- `project/content/config.local.yaml` - project-specific overrides
- `project/content/config.yaml` - project-specific defaults
- `project/config.local.yaml` - global overrides
- `project/config.yaml` - global defaults
- `~/.chassis/config.yaml` - global defaults

This allows you to (e.g.) enable multisite for the entire project, while allowing specific users to override the hostname if it conflicts with existing projects.

Changing your Chassis box is a quick two-step process:

1. Change the relevant configuration file as above and save
2. Run `vagrant provision`

`vagrant provision` tells Vagrant to update the box with your new settings, and should take care of updating everything internally.

1.3.1 Base Box Mode

By default, we use a box built with the default settings. This speeds up initial provisioning time and reduces disk usage by sharing the common parts of the VM.

The `normal` box will use the Chassis box that we've built and uploaded to Vagrant Cloud which uses PHP 7.4.

To create a customised base box for your project you could have create a *config.local.yaml* file as follows:

```
hosts:
    - client.local

```

(continues on next page)

(continued from previous page)

```yaml
# PHP version
php: 7.2

# Maximum file upload size. This will set post_max_size and upload_max_filesize in PHP␣
↪and client_max_body_size in Nginx.
upload_size: 512M

# Values: normal, base, custom
_mode: base

extensions:
    - chassis/mailhog
    - chassis/xdebug
    - chassis/tester
    - chassis/sequelpro
```

If you have your own box in Vagrant Cloud or you wish to use another vendors Vagrant Cloud box then you can use a *custom* box in your *config.local.yaml*:

```yaml
# Values: normal, base, custom
_mode: custom

# The box you'd like to use.
box: bento/ubuntu-21.10
```

wp-config.php

wp-config.php

WordPress developers will often require customisation of optional constants that are defined in `wp-config.php`.

We discourage editing `wp-config.php` in Chassis as we have an optional php file `local-config.php` which can contain custom constants and overrides for WordPress.

If `local-config.php` exists it will be loaded before `wp-config.php` so any constants that you define in `local-config.php` will be the defaults.

1.3.2 PHP Version

php

PHP 7.4 is included with Chassis by default, plus we register the additional repositories for the other versions. We don't download them all automatically, to avoid extra download times, but switching is still pretty fast as we pre-register the APT repositories.

To switch to 5.6 for example:

1. Add `php: 5.6`
2. Run `vagrant provision`

You can use either a two-part version (`5.6`) or a three-part version (`5.6.1`) if you want to pick specifc versions. We support any version between 5.6.0 and 8.3.x.

1.3.3 PHP File Upload Size

`upload_size`

A file upload size of 1024M is included with Chassis by default. This sets `post_max_size` and `upload_max_filesize` in `php.ini` and sets `client_max_body_size` in Nginx.

To switch to 100M for example:

1. Add `upload_size: 100M` to one of your `.yaml` files.
2. Run `vagrant provision`

Note: Additional `php.ini` settings can be configured by using the Chassis Phpini extension.

1.3.4 WordPress Directory

`wpdir`

Note: Deprecated; use `paths.wp` instead.

Chassis also includes the latest-released version of WordPress by default, but we also allow swapping this out completely for users who want the flexibility. Our built-in version follows the pattern laid out by WordPress Skeleton, however you can change this easily if you want.

For example, to swap out the current version of WordPress for the latest development version:

1. Clone `https://github.com/WordPress/WordPress.git` to `wp-trunk/`
2. Add `wpdir: wp-trunk` to your `config.local.yaml`
3. Run `vagrant provision`

We want to make this as flexible as possible, without forcing you to run through any of these steps if you don't need to.

Note: If you're forking Chassis and want to maintain it yourself, you'll need to keep your copy of WP up-to-date as well. We recommend merging Chassis changes back to your forked version, but you can also do it yourself:

```
# To update to WP 4.2.1, e.g:
bash puppet/update-wp.sh 4.2.1
```

1.3.5 Multisite

`multisite`

Chassis includes built-in support for WordPress multisite, with both subdirectories and subdomains. This is turned off by default, but can be turned on easily.

- For multisite in subdirectories, set `multisite: Yes`
- For multisite on subdomains, set `multisite: subdomains`

When multisite is turned on, Chassis will set up WordPress in "subdirectory" mode; that is, sites will be created under the root, but using the same domain. A site called "test" would be created at `/test/`, for example.

Subdirectory mode is great, but subdomain mode is even better. With subdomains, a site called "test" would be available at `test.vagrant.local`. This is one of the most common ways that multisite is set up, since it also means you're separating your sites more cleanly.

If you're using `subdomains` then add your subdomains you'd like to use to a yaml file. e.g.

```
hosts:
  - vagrant.local
  - wat.vagrant.local
  - moo.vagrant.local
```

Then run `vagrant provision`. Once this is done, subdomains will work automatically in your browser. Create and remove sites at will, and Chassis will ensure it just works.

Each time you add a new subdomain you will need to add the subdomain to your yaml file and run *vagrant provision*.

These subdomains will be added to `/lib/systemd/system/chassis-hosts.service`

Note: Changing from multisite back to single site requires creating the box from scratch, using `vagrant destroy` before running `vagrant up`. This is due to WordPress' inability to switch back.

This will wipe your database, so make sure you export any sites' content that you need (via the WordPress exporter).

1.3.6 Default Admin

`admin`

When you first set up your site, Chassis will install WordPress and create the default admin user for you. By default, this user is set up as *admin* with the password *password* to keep it simple for local development.

To change this, simply set the `admin` configuration option to different values, like so:

```
admin:
    user: admin
    email: admin@example.com
    password: password
```

Warning: You must include all lines shown above (albeit with your custom configuration), even if you're not changing from the default.

Changing the default admin requires creating the box from scratch, using vagrant destroy before running vagrant up.

Note also that the indentation must be done with **spaces, not tabs** in YAML configuration.

1.3.7 Database Configuration

`database`

Similar to the admin user configuration, you can also override the default MySQL name, user, password, charset and prefix:

```
database:
    name: wordpress
    user: wordpress
    password: vagrantpassword
```

(continues on next page)

(continued from previous page)

```
    prefix: wp_
    charset: utf8mb4
    collation: utf8mb4_unicode_ci
```

(Again, don’t forget to include all lines, and use spaces for indentation.)

1.3.8 MySQL Configuration

mysql

Similar to the admin user configuration, you can also override the default MySQL options:

```
mysql:
    mysqld:
        sql_mode:
            - 'ERROR_FOR_DIVISION_BY_ZERO,NO_ZERO_DATE,NO_ZERO_IN_DATE'
```

(Again, don’t forget to include all lines, and use spaces for indentation.)

1.3.9 MariaDB Configuration

mariadb

You can change the default database flavour to be MariaDB by overridding the default MySQL settings:

```
mysql:
    package_name:
        mariadb-server
    mysqld:
        log-error:
            /var/log/mysql/mariadb.log
        pid-file:
            /var/run/mysqld/mysqld.pid
        ssl-disable:
            true
    mysqld_safe:
        log-error:
            /var/log/mysql/mariadb.log
```

(Again, don’t forget to include all lines, and use spaces for indentation.)

1.3.10 Custom Host Names

hosts

By default, Chassis will set up `vagrant.local` as your main domain. If you’d like to change this, you can override the `hosts` configuration item. Note that this is a list, so it should have list items in the following format:

```
hosts:
    - vagrant.local
    - althost.local
```

The first host here will be set as the main host for the box and in WordPress. Subsequent hosts will be set as aliases of the main domain using nginx, and may be redirected by WordPress depending on your configuration or plugins.

Note: Domains ending in something other than *.local* won't have DNS set up for them automatically, so make sure to add these to your hosts file on your computer (not inside the virtual machine).

If you need to find out the IP address of your machine, run `vagrant ssh` to connect, then inside the box run `ifconfig eth1` and look for the line starting with `inet addr:`.

1.3.11 IP Address

`ip`

Chassis picks an IP address for your box automatically, using DHCP. If you'd prefer a static IP, you can specify this here with `ip: 192.168.1.114`

(Typically, this should be in the private routing range; either `192.168.x.x` or `10.x.x.x`)

1.3.12 APT Mirror

`apt_mirror`

To speed up package installation, Chassis can tell Ubuntu to use the closest mirror to you, rather than the main mirror (`ubuntu.com`). This typically speeds up installation by decreasing latency, however it may cause slowness with some slower or badly-behaving mirrors.

You can tell Chassis to do this by setting `apt_mirror: Yes`

If you have a specific mirror you'd like to use, you can set this as the value instead, such as:

```
apt_mirror: http://mirror.optus.net/ubuntu/
```

1.3.13 Synced Folders

`synced_folders`

By default Chassis syncs the `php` and `nginx` logs for you onto your local machine in the `logs` folder.

You may want to keep your themes and projects along-side Chassis, instead of inside it. You'll need to tell Chassis about these external directories, as it won't know how to map them. You can tell Chassis to map some external directories into the generated VM like so:

```
synced_folders:
  a/host/directory: a/vm/directory
  "this:ones:got:colons": another/vm/directory
```

1.3.14 NFS

nfs

Under the hood, Vagrant uses the default synced folders implementation for your system. In certain cases and uses, this might be too slow for everyday usage. You can instead use NFS under the hood, which has much better performance, but requires root on your computer.

```
nfs: true
```

We highly recommend also installing the vagrant-bindfs plugin, which ensures that users are correctly mapped into the virtual machine for you. If you're experiencing permissions errors, try installing this before anything else.

1.3.15 Paths

paths

If you're transplanting Chassis into an existing project, you can manually set some paths. These can be set to absolute paths, or relative paths.

```
paths:
   base: .
   wp: wordpress
   content: wordpress/wp-content
```

There are a few important things to note about `paths`:

1. Paths must be set in `config.local.yaml` in the same directory as your `Vagrantfile`.
2. Path's can't be used in `content/config.yaml`.
3. `base` is relative to `Vagrantfile`.
4. `content` and `wp` are relative to `base`.
5. When used, the internal mount point is changed to `/chassis` instead of `/vagrant`.
6. Any time you make a change to the paths you will need to run `vagrant reload` for those changes to take effect.

In case you have the Chassis folder nested within the project rather than wrapping it, eg: `base :  ..`, you will need to navigate to Chassis folder in order to carry out any Vagrant commands, like `vagrant up`/`vagrant halt` and `vagrant ssh`, because they only work from the folder that has a `VagrantFile`. There is a workaround for that, which is to use the `VAGRANT_CWD` variable, eg: `VAGRANT_CWD=chassis vagrant ssh`. And you can automate this by using dotenv files with `direnv``_. Install it and from the project directory execute:  ``direnv allow .; echo 'export VAGRANT_CWD=chassis' > .envrc`; replacing `chassis` with your relative Chassis directory name.

Note: When you change the `paths` configuration you will need to run `vagrant provision` for the changes to be applied.

1.3.16 Plugins

`plugins`

If you're using plugins from the WordPress.org repository you can add them in a list using the plugins slug. These will be downloaded, installed and activated for you.

Alternatively, if you want to install a plugin from a Git repository you can use a URL to a zip file of your plugin. e.g. `https://github.com/humanmade/S3-Uploads/archive/master.zip`

To find the slug just copy and paste the plugins slug from your browsers. For example the URL for Query Monitor is https://wordpress.org/plugins/query-monitor/ which makes the slug `query-monitor`.

```
plugins:
   - query-monitor
   - user-switching
   - https://github.com/humanmade/S3-Uploads/archive/master.zip
```

1.3.17 Themes

`themes`

If you're using themes from the WordPress.org repository you can add them in a list using the themes slug. These will be downloaded for you. The last theme in the list will be the theme that is activated for your site.

Alternatively, if you want to install a theme from a Git repository you can use a URL to a zip file of your theme. e.g. `https://github.com/humanmade/S3-Uploads/archive/master.zip`

To find the slug just copy and paste the plugins slug from your browsers. For example the URL for Twenty Sixteen is https://wordpress.org/themes/twentysixteen/ which makes the slug `twentysixteen`.

```
themes:
   - twentyfifteen
   - twentysixteen
   - https://github.com/WordPress/twentyseventeen/archive/master.zip
```

1.3.18 Site Title

`website`

You can customize the title Chassis uses when installing your local WordPress site.

```
website:
   name: My Local WordPress Site
```

1.3.19 Extensions

extensions

You can enable official Chassis extensions and third party extensions by listing their repo name in the `extensions` section:

Extension names can be specified in one of three ways:

- *extension-name*: Official Chassis extensions can be specified just by name.
- *user/repo*: Extensions on GitHub can be specified using the username and repo separated with a slash.
- *https://github.com/example/example.git*: Any other extension can be specified by its full git URL.

```yaml
extensions:
   - Tester
   - chassis/chassis-openssl
   - https://bitbucket.org/some/example.git
```

You can also remove extensions that you have previously installed. All configuration files will be remove from your Chassis server.

To remove an extension simply add new section to one of your *.yaml* configuration files:

```yaml
disabled_extensions:
   - chassis/mailhog
```

1.3.20 Machine Customisations

The underlying virtual machine managed by Vagrant can be customised, but depends on which provider you are using.

1.3.21 VirtualBox

virtualbox

When using VirtualBox, you can customise how much memory (in megabytes) and how many virtual CPUs will be assigned to the machine. The default values for both (`null`) are to use the VirtualBox defaults (1024 MB of RAM, and 2 vCPUs).

```yaml
virtualbox:
   memory: null
   cpus: null
```

machine_name

By default the machine name is "default". This can make it difficult to distinguish between virtual machines in the VirtualBox GUI or when listing VMs on the command line. Overriding the machine name makes it easier to tell which one is which.

Note that if the machine name is changed after it has already been created vagrant will not be able to find the VM. It is recommended to destroy the VM before making this change and then recreating it.

```yaml
machine_name: project.local
```

1.4 Guides

These guides will help you configure Chassis for specific types of development.

1.4.1 Migrations

Importing A Production Database Into Chassis

1. Use `ssh` to connect to your production server. Your host should provide instructions for doing this.
2. Export your production database with WP-CLI `wp db export <filename.sql>`.
3. Download your the contents of your `wp-content` folder on production to your local content folder.
4. Provision a new Chassis instance.
5. Copy the export into the `content` folder.
6. Run `vagrant ssh` in a terminal to SSH into your Chassis box.
7. Run `cd /vagrant/content`
8. Run `wp db import <filename.sql>`
9. Run `wp search-replace '//www.yoursite.com/wp-content' '//vagrant.local/content'`
10. Run `wp cache flush`
11. Start developing.

Alternatively you can use the SequelPro or phpMyAdmin extensions to handle importing and exporting of your databases.

You could also look at using the db_backup extension. You can commit an SQL export as a file called `chassis-backup.sql` and it should automatically import on install. You will still need to search and replace the URLs.

Exporting A Development Database Into A Production Database

1. Add the production domain under `- vagrant.local` in `content/config.local.yaml` e.g. `- www.yoursite.com`.
2. Reprovision with Puppet `vagrant provision`.
3. SSH into your Chassis Box `vagrant ssh`.
4. Use WP-CLI to search and replace e.g. `wp search-replace '//vagrant.local' '//www.yoursite.com'`.
5. Use WP-CLI to search and replace the content urls. e.g. `wp search-replace '//www.yoursite.com/content' '//www.yoursite.com/wp-content'`.
6. Export the database using WP-CLI `wp db dump --add-drop-table`.
7. Log in to phpMyAdmin on your production server.
8. Drop the database on your site and import your database dump.
9. Upload your the contents of your `content` folder to your `wp-content` folder on production.
10. You're done!

N.B. If you getting the Error Establishing a Database Connection message then you'll probably need to edit the `$table_prefix` in `wp-config.php`.

Alternatively you can use the SequelPro or phpMyAdmin extensions to handle importing and exporting of your databases.

1.4.2 WP-CLI

WP-CLI is the command-line interface for WordPress. You can update plugins, configure multisite installations and much more, without using a web browser.

We bundle the latest version of WP-CLI in Chassis. You can access WP-CLI by running `vagrant ssh` from a terminal. You can check the WP-CLI details by running `wp --info` inside the Chassis box. This should result in something like the following:

```
vagrant@vagrant:~$ wp --info
OS:  Linux 4.15.0-112-generic #113-Ubuntu SMP Thu Jul 9 23:41:39 UTC 2020 x86_64
Shell:       /bin/bash
PHP binary:  /usr/bin/php7.4
PHP version: 7.4.10
php.ini used:       /etc/php/7.4/cli/php.ini
WP-CLI root dir:    phar://wp-cli.phar/vendor/wp-cli/wp-cli
WP-CLI vendor dir:  phar://wp-cli.phar/vendor
WP_CLI phar path:   /home/vagrant
WP-CLI packages dir:
WP-CLI global config:       /home/vagrant/.wp-cli/config.yml
WP-CLI project config:
WP-CLI version:     2.4.0
```

WP-CLI Commands

You can see a full list of WP-CLI commands by running `wp help` inside your Chassis box. Some common commands are as follows:

- `wp config list` - Lists variables, constants, and file includes defined in wp-config.php file.
- `wp cron event list` - Lists scheduled cron events.
- `wp cron event run` - Runs the next scheduled cron event for the given hook.
- `wp plugin list` - Gets a list of plugins.
- `wp plugin install <plugin_slug> --activate` – Install and activate a plugin from the WordPress plugin directory.
- `wp post list` - Gets a list of posts.
- `wp post-type list` - Lists registered post types.
- `wp search-replace <old> <new>` - Searches through all rows in a selection of tables and replaces appearances of the first string with the second string. Add the `--dry-run` parameter to test without making changes.
- `wp shell` - Evaluate PHP statements and expressions interactively, from within a WordPress environment. e.g. `get_bloginfo( 'name' );`
- `wp site empty` - Empties a site of its content (posts, comments, terms, and meta).
- `wp site create --slug=awesome` - Creates a site in a multisite installation.

There is extensive documentation and examples of all the WP-CLI commands on the WP-CLI website.

WP-CLI Packages

WP-CLI has additional WP-CLI packages which can be installed to add additional helper commands for your WordPress development needs.

We have made a Chassis WP-CLI extension that allows you to automatically install WP-CLI packages which have been published in the package index.

For example you could add the following section to one of your `.yaml` configuration files to automatically install the Chassis WP-CLI extension and the WP-CLI Server command.

```
wp_cli:
 packages:
     - wp-cli/server-command
     - wp-cli/restful
```

1.4.3 Setting up SSL

To add an SSL to Chassis you need to do the following steps:

1. Add `- chassis/chassis_openssl` to one of your `.yaml` configuration files or run `git clone https://github.com/Chassis/chassis_openssl.git extensions/chassis_openssl`.
2. Run `vagrant provision`. This will create a `vagrant.local.cert` or `<yoursitename>.local.cert` and a `vagrant.local.key` or *<yoursitename>.local.key* in the root directory of your Chassis folder.
3. Modify the `WP_SITEURL` and `WP_HOME` constants to use `https://` instead of `http://`.
4. If you are using a Mac run `sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain <yoursitename>.local.key`.
5. If you are using Windows run `certutil -enterprise -f -v -AddStore "Root" "<yoursitename>.local.key"`.
6. Alternatively, you can read the Chassis OpenSSL readme for GUI options.

1.4.4 WordPress Core Development

We have created an official Chassis extension for [WordPress Core](https://github.com/Chassis/core_dev) development. Follow the instructions in the README to get setup and help contribute to [WordPress Core](https://make.wordpress.org/docs/handbook/devhub/#to-get-involved).

1.4.5 Site Health

Out of the box Chassis will not meet the Site Health requirements checks in WordPress core. If you'd like to achieve 100% Site Health you will need to do the following:

1. Create a *local-config.php* with the following constants:

```
<?php
define( 'AUTOMATIC_UPDATER_DISABLED', false );
define( 'WP_DEBUG_LOG', false );
define( 'WP_DEBUG', false );
```

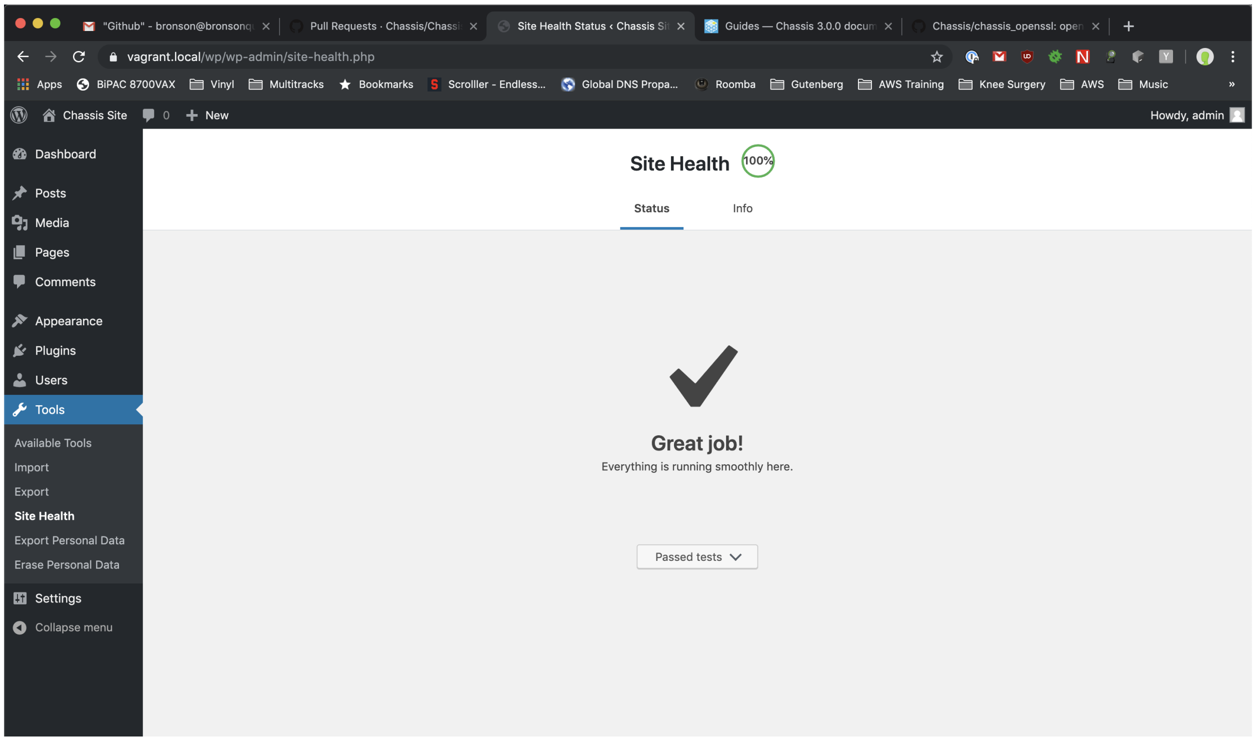
2. Delete the inactive themes and plugins in both *content* and *wp/wp-content*.
3. Create a yaml configuration file with the following extensions:

```yaml
extensions:
    - chassis/chassis_openssl
    - chassis/imagick
    - chassis/bcmath
```

4. Run `vagrant provision`.
5. Follow the Setting up SSL guide to setup your site for SSL.
6. Create a file called `ssl.php` in *content/mu-plugins* with the following contents:

```php
<?php
add_filter( 'https_ssl_verify', '__return_false' );
```

7. **Caution:** Delete your `.git` folders.
8. You'll now have 100% Site Health!



Tester Extension

The Tester extension adds and configures PHPUnit for WordPress tests. You can install other extensions if you want, but there's nothing else required.

Configuration

Save this configuration as `config.local.yaml` in the Chassis root directory to configure Chassis to look for WordPress in your `wpcore/` checkout.

```yaml
paths:
    # Use the Chassis box normally...
    base: .
    content: content

    # But use my development copy of WordPress for the WP source
    wp: ../wpcore/src

# Also use your development copy of WordPress for the unit test
# framework, and for the unit tests themselves
synced_folders:
    ../wpcore: /vagrant/extensions/tester/wpdevel

# Set the host to ``core.local`` to distinguish from other chassis boxes
hosts:
    - core.local

# Explicitly set database configuration to avoid warning with Tester
database:
    name: wordpress
    user: wordpress
    password: vagrantpassword
    prefix: wp_

# Run in multisite mode (totally optional)
multisite: true
```

WordPress will now be loaded from the `wpcore/` checkout, not the `wp/` directory within Chassis. However, in order for WP-CLI and other tools to be able to find Chassis' `wp-config.php` we need to add a dummy configuration file at `wpcore/src/wp-config.php`:

```php
<?php
// Fool WP-CLI into recognising this as a valid config file
if ( false ) {
    require ABSPATH . './wp-settings.php';
}

require '/vagrant/wp-config.php';
```

In normal circumstances editing Chassis' own `wp-config.php` file is discouraged in favor of using `local-config.php`. However in this case you're working around WP-CLI with the dummy `wp-config.php` file, so you will need to make one change to Chassis' own `wp-config.php` file to wrap the line which requires `wp-settings.php` in a conditional check:

```
if ( ! defined( 'WP_CLI' ) ) {
    require_once( ABSPATH . 'wp-settings.php' );
}
```

This will prevent WP-CLI from loading `wp-settings.php` twice.

With this configuration you should now be ready to develop against WordPress Core.

1.4.6 Vagrant Share

Vagrant Share enables the ability to generate a temporary URL which you can share with people to allow them access to your local Chassis installation.

1. **Install Vagrant Share**

   Run the following command in a terminal to install the Vagrant Share plugin. `vagrant plugin install vagrant-share`.

2. **Install ngrok**

   Vagrant Share requires `ngrok` to be installed on the host machine. You can verify if this is installed by running `which ngrok` in a terminal. If there is no output then you will need to download and install ngrok. Once you've downloaded `ngrok` unzip it: `unzip /path/to/ngrok-stable-darwin-amd64.zip`. Move ngrok: `mv /path/to/ngrok /usr/local/bin/ngrok`. If you have Homebrew installed you can install ngok by running `brew install eqnxio/ngrok/ngrok`.

3. **Run Vagrant Share**

   You now need to run `vagrant share --http=vagrant.local:80`. If you're using a custom domain then you will need to use that e.g. `vagrant share --http=<your-custom-domain>.local:80`. You will then have a temporary URL generated for you. e.g. `http://<id>.ngrok.io`. You need to leave this running in the background.

4. **Share your site**

   Navigate to the URL that ngrok generated.

1.4.7 Testing a Chassis site from a Windows Virtual Machine

When developing on a Mac or Linux host system, you may want to verify your website works properly in Internet Explorer or other windows-only browsers. Microsoft helpfully provides [virtual machine images for testing in Legacy Edge and IE11](https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/). The following instructions detail how to access your Chassis site from within a browser running in a separate Windows 10 virtual machine.

1. **Get your Chassis machine's IP**

   From the command line within your Chassis folder, run `vagrant ssh -c 'ifconfig'`. This sends a message into your Chassis environment to output the VM's internal network information. What we're looking for is the IP address assigned to the VM, which in our example case looks like this:

   ```
   eth1      Link encap:Ethernet  HWaddr 08:00:27:5d:c7:4f
             inet addr:172.28.128.17  Bcast:172.28.128.255  Mask:255.255.255.0
   ```

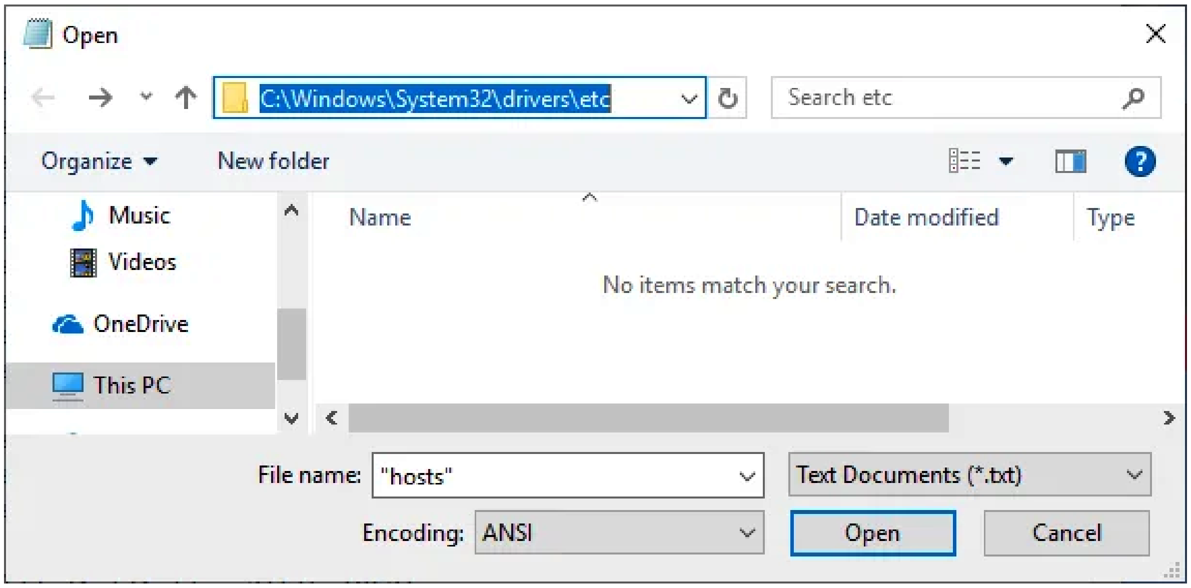
2. **Boot your Windows VM**

3. **Tell Windows the IP for your Chassis VM**

   Now that Windows is running and we've got our IP (`172.28.128.17` in this example), right-click on Notepad within your Windows VM and select "run as administrator". Once Notepad is open, go to "Open a file".

   In Notepad's "open file" dialogue, navigate to the folder `C:\Windows\System32\drivers\etc`. This folder may appear empty. This is a trick: it is not empty.

   Type "hosts" into the filename dialogue,



then hit "Open" to open the hidden hosts file.

At the bottom of the file, add the IP from step 1 alongside your Chassis system's hostname. For example,

```
172.28.128.17    chassis.local
```

4. **Test your Chassis site in IE or Edge**

   If you open a browser within your Windows VM and navigate to *chassis.local*, it should now connect to your Chassis site.

   Note: Windows will not be able to access other services running on your host OS, such as Webpack DevServer. For this reason, it is usually best to test IE and Edge using a static production build of your web application.

Debugging

If you see an error when you run `vagrant provision` then try running it again. If you're still having trouble accessing the URL try a `vagrant reload` after you've run a successful `vagrant provision`

1.5 Inside the Box

While Chassis handles the setup of the virtual machine for you, it can be useful to know exactly what's happening inside of it.

1.5.1 Networking

Chassis VMs use the networking features provided by Vagrant and the underlying VM management (VirtualBox, VMWare Fusion, etc). The VM is connected to the host using a private network, with a dynamically assigned IP address (unless a static IP is *specified in your config*).

DNS resolution is handled by Avahi, which is a Zeroconf/Bonjour service. This responds to DNS-SD (DNS Service Discovery, also called mDNS) requests, which are requested by the host for `.local` domains. This requires a DNS-SD daemon on your host; macOS automatically includes this (Bonjour), most Linux systems also include it (Avahi), and

Windows requires it to be installed (Bonjour for Windows, installed automatically by iTunes, Skype, Adobe Creative Suite, and others).

For additional domain names specified in the config, and WordPress multisite domains, other names are registered with Avahi using the `chassis-hosts` daemon (source in `puppet/chassis-hosts.py`). Files inside the `/etc/chassis-hosts/conf.d` directory are automatically watched by the daemon, and updates are pushed into the configuration on change. (The daemon uses D-Bus to communicate with Avahi.)

`local-config-hosts.php` attaches to the WordPress hooks for site creation, deletion, and update, and writes all the domains it knows about into `/etc/chassis-hosts/conf.d/subdomains`.

1.5.2 Synced Folders

Synced folders are handled by the VM manager. Chassis automatically uses the default implementation provided by the VM manager, unless the `nfs` config option is used.

Folders are set to be world-writable and world-readable (777) to ensure compatibility with WordPress plugins that expect files to be writable.

By default, the Chassis root directory is synced to `/vagrant`. When a custom root directory is specified in the configuration, an additional synced folder is set to map the root directory to `/chassis` on the system. If the `wp` or `content` directories aren't under the root directory, additional synced folders are added for each to `/chassis/wp` and `/chassis/content` respectively.

1.5.3 Nginx

HTTP requests are served up by nginx.

For single-site WordPress, the `site.nginx.conf.erb` template (in `puppet/modules/chassis/templates`) is used, while `multisite.nginx.conf.erb` is used for multisite configuration. These templates are used to generate the site's configuration, which is placed in `/etc/nginx/sites-available/<domain>`, and symlinked into `/etc/nginx/sites-enabled`.

The primary nginx configuration is loaded from `/etc/nginx/nginx.conf` by the system, and this file is provisioned from the `nginx.conf.erb` template. This is a standard nginx setup, based on the file bundled with the Ubuntu package.

Further configuration files are included by the main configuration, which loads in `/etc/nginx/*.conf` to allow for further system-level configuration. The site config also loads `/etc/nginx/sites-available/<domain>.d/*` to allow Chassis extensions to further configure sites.

For any request not for a specific file, nginx passes the request via FastCGI to PHP.

1.5.4 PHP

PHP runs in FastCGI mode (`php-fpm`), using a Unix socket at `/var/run/php5-fpm.sock`. The PHP configuration at `/etc/php/<version>/fpm/php.ini` is provisioned from the `php.ini.erb` template. This is a standard PHP configuration based on the `php.ini` included in the Ubuntu package. The CLI configuration is also provisioned to the same template into `/etc/php/<version>/cli/php.ini`.

Additionally, PHP loads all files from `/etc/php/<version>/fpm/conf.d` (and corresponding CLI directory) in alphabetical order. Chassis extensions can place additional configuration into this directory. PHP extensions installed via packages (e.g. `php-xdebug`) will automatically place configuration into this directory to load the extension (typically with a filename like `10-xdebug.ini`).

1.5.5 Database

The root MySQL credentials, if you need them, are:

Username: `root`

Password: `password`

1.6 Extending Chassis

Chassis includes a flexible extension system to allow setting up the server exactly how you like.

Find available extensions on the Extensions Index.

1.6.1 Installing extensions

Extensions can be *specified in your configuration*, and will be automatically downloaded by Chassis. You can also manually add extensions.

Chassis automatically loads all subdirectories of your `extensions/` directory as extensions, so extensions can be downloaded into this directory.

Example: Installing The Memcached Extension

To install the memcached extension you would do the following:

1. Open your terminal and navigate to your root Chassis folder.
2. `git clone git@github.com:Chassis/memcache.git extensions/memcache`.
3. `vagrant provision`.

1.6.2 Globally installing extensions

In addition to per-project, extensions can be globally installed. This is useful for development tools that you want to apply to all your Chassis boxes, which aren't specific to any project.

Chassis loads global extensions from `~/.chassis/extensions` in the same way that it loads project-specific extensions.

To install a global extension clone it into `~/.chassis/extensions`. For example, if you'd like to make the mailhog extension global then you would run the following command in a terminal.

`git clone https://github.com/Chassis/Mailhog ~/.chassis/extensions/mailhog`

If a project includes an extension of the same name in its `extensions/` directory, the project's extension will be loaded instead of the globally-installed extension.

Global extensions can also be disabled in the same way local extensions are disabled. e.g:

```
disabled_extensions:
   - chassis/mailhog
```

1.6.3 Updating extensions

The Chassis extensions often received bug fixes, upgrades and enhancements and a developers Chassis project may require an extension to be updated if a provision is failing. Currently you need to update extensions manually. For example, let's pretend that the Xdebug extension has an update that you require. You would need to do the following:

1. Open your terminal and navigate to your root Chassis folder.
2. `cd extensions\xdebug`
3. `git pull`
4. `vagrant provision`

You can run steps 3 & 4 in each extensions directory and run step 4 to update all of your extensions.

1.6.4 Creating your own

Chassis extensions can be created by anyone who knows the Puppet configuration language.

Note: *"What's Puppet?"*, I hear you cry. Puppet is a tool that allows you to "provision" servers; that is, make them consistent by installing software packages and setting configuration. It also includes its own neat language for the configuration, giving it a huge amount of power.

Before you get worried that we're crazy (although we might be), and that you're learning something useless, Puppet is one of the most widely used tools for configuring servers, and has a huge community behind it. Everyone from Twitter to the New York Times to Google uses Puppet to set up their servers. Anything you pick up here is applicable in the wider Puppet ecosystem too for setting up your own servers.

Chassis internally uses Puppet to configure your virtual machine and set up all the niceties that you've come to expect from Chassis, such as switching PHP versions or configuring MySQL.

If you don't already know Puppet, we recommend reading the Learning Puppet series, which guides you into understanding and using Puppet. We've already got Puppet set up and installed inside your Chassis virtual machine, so you can follow any of the steps by simply running `puppet` inside the box.

Once you've got a grip on Puppet, we'd recommend having a quick read of how Chassis works internally. We store all our files in the `puppet/` directory.

1.6.5 Extensions API (v2)

Note: Chassis v2 introduced a formal extensions API. This is a backwards-incompatible change from our initial extensions (now v1), so extensions need to opt-in with the `chassis.yaml` metadata file.

If your extension does not specify the version, it will be assumed to be v1. This will generate a deprecation warning in the future, and may be removed, so it is highly recommended to update your extensions.

Making your own extension is easy. There's two required pieces for an extension: an extension metadata file, and a Puppet class.

An extension exists as a subdirectory of the *extensions* directory in Chassis. For example, the memcache extension lives in the *extensions/memcache/* directory.

An example extension is included with Chassis, and lives in the *extensions/example/* directory. We highly recommend reading the docs and code inside this extension to understand how it works.

Extension Metadata

Extensions specify information about themselves in the extension metadata file, a YAML file specifying options to the extension. This metadata file lives at `chassis.yaml` inside your extension's directory.

Extensions **must** specify the following options:

- `version`: Extension API version. Currently `2`.

Extensions may also specify the following options:

- `dependencies`: Extensions that the extension depends on. List of strings, where each string is an extension name (see *Site Title*).
- You can also add custom configuration variables to your extension configuration and use them in your extension. e.g.:

  ```
  your_variable: your_value
  ```

Puppet Class

To integrate into Chassis' provisioning, Chassis loads a class with the same name as your extension. This class must be defined in `extensions/{name}/modules/{name}/manifests/init.pp`, and the name of the class must match the extension's directory name.

Note: You can't use hyphens in your extension name as Puppet interpolates those as minus signs. Please use underscores as this will ensure your extension will work in future versions of Puppet and Chassis.

This class receives a single hash parameter of `$config`, which contains the Chassis configuration specified in the `config.yaml` files.

Internally, Chassis registers `extensions/{name}/modules` as an additional module path for Puppet. This means you can add third-party modules into your `modules/` directory as needed.

For example, an extension called `example` located in the `extensions/example/` directory would have the following class at `extensions/example/modules/example/manifests/init.pp`:

```
class example(
  $config
) {
  notify { "PHP version is ${config[php]}": }
}
```

Currently, a class is always required, even if you don't need to provision anything using Puppet. If you don't need provisioning, you can use an empty class:

```
class example( $config ) {}
```

WordPress Configuration

Chassis automatically loads `local-config.php` from your extension's directory when loading `wp-config.php`. If you need to run PHP inside the WordPress context, use this file.

This file can be committed to your extension's repository, or provisioned by Puppet, if you need to set additional settings.

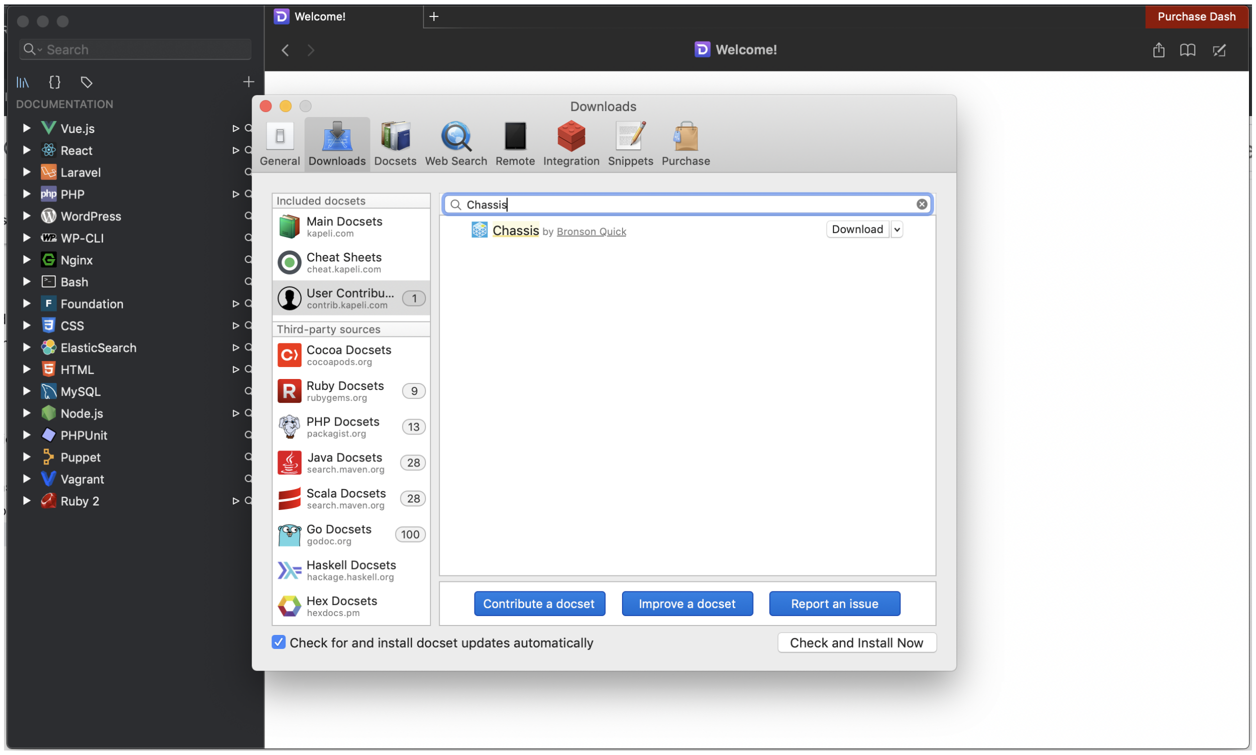
For example, the memcache extension sets configuration for the object cache inside WordPress to set the hostname. The `local-config.php` file hence contains:

```
<?php
$memcached_servers = array( '127.0.0.1:11211' );
```

1.7 Reference

1.7.1 Dash Documentation

The Chassis documentation can be downloaded and viewed in Dash on Mac. Simply search for Chassis in the User Contributed docs.



Other alternatives have been developed in cooperation with Dash.app's developer Kapeli:

- Helm Dash for Emacs.
- Velocity for Windows.
- Zeal for Linux, macOS, and Windows.

1.7.2 Commands

This is a very quick reference of Vagrant commands we find useful. For the full documentation, check out the Vagrant command reference.

```
# Start the VM
vagrant up

# SSH in to the VM
vagrant ssh

# Reprovisioning (e.g. after updating this repository)
vagrant provision

# Suspending (sleeping) the VM
vagrant suspend

# Halting (shutting down) the VM
vagrant halt

# Reboot (halt and up)
vagrant reload

# Destroying the VM (if your VM is completely broken)
vagrant destroy
```

1.7.3 Building a new Base Box for Vagrant Cloud

The Chassis base boxes are generated by the Chassis team and they are uploaded to Vagrant Cloud.

The Chassis base box will follow the SEM Versioning 2.0.0 specification which is `MAJOR.MINOR.PATCH`. For example:

1. MAJOR version 2.0.0 - This would be a new Ubuntu distribution. e.g. Bionic Beaver.
2. MINOR version 1.1.0 - This would be when we have added a new Chassis feature. e.g. *upload_size*
3. PATCH version 1.0.1 - This would be when the Ubuntu base box has had an update or a bug fix.

To generate and upload a new base box a Vagrant Cloud Chassis team member will need to do the following:

1. Open a new issue in the Chassis repository with details about the base box updates.
2. Create a new branch off `main` for the base box updates.
3. Backup any custom yaml configuration files you have as they will be deleted during this process.
4. Delete any extensions in your `extensions` folder (apart from `example` as it's required).
5. Bump the version number of the base box in the build script, yaml file and documentation.
6. Change `_mode:  normal` to `_mode:  base` in `config.yaml`.
7. Run `vagrant destroy -f`.
8. Run `vagrant up`.
9. Run `sh buildbox.sh` in the root directory of Chassis.
10. Login to your Vagrant Cloud account when you are prompted to do so.
11. Wait for the upload to complete.

12. Submit a Pull Request and wait for one of the Chassis team members to approve it.

When you're building a new base box it's a good idea to check the versions of the following software for when write the release notes:

1. PHP - `php -v`
2. software-properties-common - `apt policy software-properties-common`
3. curl - `curl --version`
4. pip - `pip -V`
5. avahi-daemon - `avahi-daemon -V`
6. puppet - `puppet --version`
7. ruby - `ruby -v`

1.7.4 Troubleshooting

1.7.5 Logs

Chassis synchronises the Nginx and PHP log files from the Virtual Machine to your local development folder. These logs can be useful if your provisions are failing and you require further information.

There are three logs that we synchronise for you:

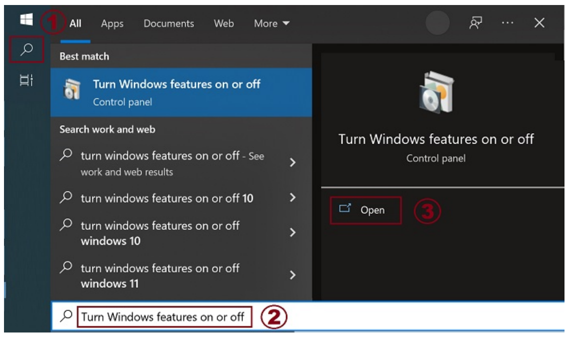
1. `logs/php/php_errors.log` - This shows any PHP errors or warnings that have occurred.
2. `logs/nginx/error.log` - This shows any errors that have occurred.
3. `logs/nginx/access.log` = This shows details of any requests that Nginx has served.

1.7.6 Vagrant gets stuck on *default: SSH auth method: private key* on Windows machines

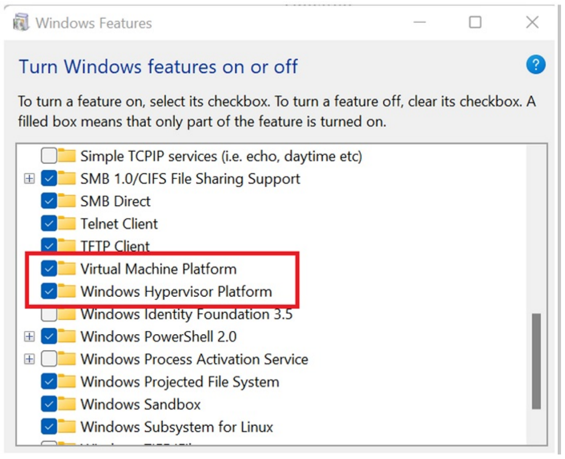
```
$ vagrant up
default: SSH auth method: private key
```

To resolve this issue you'll need to do the following:

1. Click the Search icon.
2. Type Turn Windows features on or off in the search field.
3. **Click Open.**


1
All
Apps
Documents
Web
More
Best match
Turn Windows features on or off
Control panel
Search work and web
turn windows features on or off - See work and web results
turn windows features on or off 10
turn windows features on or off windows 10
turn windows features on or off windows 11
Turn Windows features on or off
Control panel
Open
3
Turn Windows features on or off
2

4. **Uncheck both Virtual Machine Platform and Windows Hypervisor Platform.**



To turn a feature on, select its checkbox. To turn a feature off, clear its checkbox. A filled box means that only part of the feature is turned on.

- [ ] Simple TCPIP services (i.e. echo, daytime etc)
- [x] SMB 1.0/CIFS File Sharing Support
- [x] SMB Direct
- [x] Telnet Client
- [x] TFTP Client
- [x] Virtual Machine Platform
- [x] Windows Hypervisor Platform
- [ ] Windows Identity Foundation 3.5
- [x] Windows PowerShell 2.0
- [ ] Windows Process Activation Service
- [x] Windows Projected File System
- [x] Windows Sandbox
- [x] Windows Subsystem for Linux

5. Click OK and restart your computer.

6. Change directories back to your Chassis project and run `vagrant up`.

1.7.7 Character encoding on Windows machines

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'bento/ubuntu-16.04'...
C:/Vagrant/embedded/gems/2.1.1/gems/childprocess-0.6.3/lib/childprocess/windows/process_
↪builder.rb:43:in `join': incompatible character encodings: Windows-1252 and UTF-8␣
↪(Encoding::CompatibilityError)
```

To resolve this issue you'll need to create a symlink and set the VAGRANT_HOME path. e.g.

```
$ cd C:\users
$ mklink /J username usérnamé
$ setx VAGRANT_HOME "C:/users/username"
```

Alternatively you can set the path for VirtualBox to a directory that doesn't have special characters. To do this open the VirtualBox application, select 'File' -> 'Settings' -> 'General' and change the default path for VM's (e.g. to "C:VirtualBox VMs").

1.7.8 Server IP address could not be found. (DNS_PROBE_FINISHED_NXDOMAIN)

This typically occurs on Mac when the DNS configuration has not been set up properly during the installation. SSH into the Chassis machine and run `sudo service avahi-daemon restart`

1.7.9 Further issues

In case the previous tips didn't solve your issue, it is recommended to update your copy of Virtualbox, Vagrant, Chassis, and the Chassis box (note that updating the box will recreate your instance from scratch), and/or create a bug report on the GitHub repo.

INDEX