Riptide Documentation

Marco Köpcke

Jan 05, 2022

Contents

Riptide is a set of tools to manage development environments for web applications. It's using container virtualization tools, such as Docker to run all services needed for a project.

It's goal is to be easy to use by developers. Riptide abstracts the virtualization in such a way that the environment behaves exactly as if you were running it natively, without the need to install any other requirements the project may have.

CHAPTER 1

Hello World!

A simple hello world web service:

```
# riptide.yml
project:
  name: hello-world
  src: .
  app:
    name: hello-world
    services:
      hello_world:
        image: strm/helloworld-http
        port: 80
        run_as_current_user: false
        roles:
          - main
```

To setup the project run:

```
# Setup project
riptide setup
# Start Riptide Proxy
riptide_proxy
```

After the setup head over to `http://hello-world.riptide.local` (assuming you are using the default configuration and DNS is set up). The Service will auto-start and after that you will be greeted with the message: `Hello from hello_world`.

CHAPTER 2

Riptide config files

If you need to edit the Riptide configuration files, here are the paths on where to find them:

- Linux: `~/.config/riptide`
- Windows: `C:\Users\USERNAME\AppData\Local\riptide`
- Mac: `/Users/USERNAME/Library/Application Support/riptide`

CHAPTER 3

Documentation

3.1 User Documentation

3.1.1 Introduction

Riptide is a set of tools to manage development environments for web applications. It's using container virtualization tools, such as Docker to run all services needed for a project.

It's goal is to be easy to use by developers. Riptide abstracts the virtualization in such a way that the environment behaves exactly as if you were running it natively, without the need to install any other requirements the project may have.

Riptide has to following major features:

- Environments can be defined in re-usable YAML files. The components of these files can be shared across multiple projects using Git repositories.
- Web services can be started without installing anything besides the engine (eg. Docker) and Riptide.
- Using Bash and Zsh integration CLI commands for projects can be defined an run in a shell just like ordinary commands.
- Riptide manages file and process permissions and tries to run everything as the same user that runs the Riptide command.
- Cross platform! Riptide works on Linux, Mac and Windows.
- Riptide is shipped with a simple proxy server that matches all your projects and services using DNS hostnames. It comes with out of the box SSL support and can also run behind another reverse proxy such as Nginx or Apache. The Proxy automatically starts and stops projects as you need them.
- You can work on multiple different projects at the same time, all requiring different versions of software and libraries without having to install anything.

Riptide is split into the following sub-projects:

- Riptide Library (`riptide-lib`): Main Python library that ties everything together.

- Riptide CLI (`riptide-cli`): CLI that you will interact with to manage your Riptide projects.
- Riptide Proxy (`riptide-proxy`): Proxy Server that proxies you the content of your web-services.
- Engines: The engine that actually starts and stops containers for services and commands. The following Engines are available:
  - `riptide-engine-docker`: Docker Engine
- Database Drivers: Optional components that make managing databases in your development environment easier. See Managing Databases for features. The following Database Drivers are available:
  - `riptide-db-mysql`: Driver for MySQL based databases

Riptide Projects include the definition of an app. The app defines what services to run and what commands are available. Services are parts of the application that are constantly running, such as the actual web-service or the database. Commands are utility CLI commands that are helpful in working with your application such as *node* or *npm* for NodeJS projects.

This guide shows you how to set up Riptide and how to interact with services and commands.

This is the user documentation, it assumes that someone has already set up a project for you. If you want to set-up a project yourself we recommend that you follow this guide first using the provided "Hello Wold" app. After that, read through the Configuration Guide.

3.1.2 Installing Riptide

To install Riptide, follow the guide for your operating system.

Note: This documentation was recently changed. If you have any issues, please contact us on Slack.

Linux

This guide will explain how to install Riptide under Linux distributions.

Installing Requirements

This guide assumes you want to run Riptide in the most common set-up using the Docker Engine. To use Riptide you need to have the following installed:

- Python 3.6+
- **pip for Python 3 (might come installed with Python)**
  - on Ubuntu `sudo apt-get install python3-pip`
- **Docker 16.0+**
  - Do **NOT** install Docker via Snap. Follow the instructions on the page linked.
  - Make sure to also follow the post-installation steps.
- **python-prctl requirements:**
  - on Ubuntu: `sudo apt-get install build-essential libcap-dev`
  - on Fedora: `sudo yum install gcc glibc-devel libcap-devel`

Python is available using package managers.

There is a good chance you already have Python installed. Try running `python3 --version` to check.

Installing Riptide system-wide

To install all Riptide components and the Docker implementation run the following command:

```
$ sudo pip3 install riptide-all
```

Make sure this command is run with sudo!

You can test if Riptide is working:

Installing Riptide in a Virtualenv

Riptide can also be installed in a Virtualenv. This is only recommended for advanced Python users. Please make sure, to use the correct Python interpreter of your Virtualenv when setting up the proxy server.

Updating Riptide

To update Riptide, run

```
$ [sudo] riptide_upgrade
```

If you installed Riptide system-wide and not in a Virtualenv, you **MUST** use sudo. Failing to to so may break your installation.

Get help and join the community

If you need some support or just want to chat with the community, join our Slack workspace.

Next steps

The next pages of this documentation will explain how to finish the setup of Riptide, how to setup the Proxy server and how to install the Bash/Zsh integration. It will also teach you how to use the Riptide CLI and Proxy server.

Please make sure to read through all of the following pages of this documentation to properly setup Riptide.

MacOS

This guide will explain how to install Riptide under MacOS.

Note: MacOS is not supported as well as the Linux setup. Most of the downsides of Riptide on MacOS come from the Docker Desktop implementation for MacOS.

Riptide has some Performance optimizations to increase the performance on Mac, but it will still be slower than running it on Linux.

If you have experience with Docker or Python on MacOS, we'd love your support in making Riptide on MacOS even better!

Installing Requirements

This guide assumes you want to run Riptide in the most common set-up using the Docker Engine. To use Riptide you need to have the following installed:

- Python 3.6+
- pip for Python 3 (might come installed with Python)
- Docker Desktop 16.0+

Python is available for Mac machines using package managers.

Note: If you know what the best way of installing Python 3 is, please let us know by updating this documentation on Github.

There is a good chance you already have Python installed. Try running `python3 --version` to check.

Installing Riptide system-wide

> **Warning:** It is currently unknown if sudo must or even can be used when installing Riptide system-wide. It seems to depend on the way Python is installed and also the MacOS version.
>
> Please try with sudo first and see if this works. Make sure to remember if you installed with or without sudo, as you will need to update Riptide the same way, see below.

To install all Riptide components and the Docker implementation run the following command:

```
$ [sudo] pip3 install riptide-all
```

Sudo may or may not be required, see warning above.

You can test if Riptide is working:

Installing Riptide in a Virtualenv

Riptide can also be installed in a Virtualenv. This is only recommended for advanced Python users. Please make sure, to use the correct Python interpreter of your Virtualenv when setting up the proxy server.

Updating Riptide

To update Riptide, run

```
$ [sudo] riptide_upgrade
```

Make sure to use or not use sudo, depending on if you did during your initial installation. Failing to do so, WILL break your installation.

Configuring shared folders

Docker Desktop for MacOS only allows the virtual machine running the Docker daemon limited access to your machine.

The default configuration is not enough to use Riptide. Please open the settings of Docker and navigate to the Shared Folders tab. Make sure the following entries are present:

- /Users
- /Volumes
- /private
- /tmp
- /var/folders
- /usr/local/lib/python3.7 (**Or wherever else Python is installed!**)

Additional MacOS related notes

Many additional settings or issues not described in this documentation may be directly related to the Docker Desktop for MacOS implementation.

Please see the documentation for Docker Desktop for Mac for further information.

Known issues under MacOS

- Riptide currently uses the default Docker Desktop Mac daemon. This setup is known to have significantly worse performance than the Linux version. Riptide has some Performance optimizations to increase performance.
- Due to the performance optimization settings, it might happen that changes to files are not immediately visible on the host system or the running containers. Some files are not updated on the host system at all (see Performance optimizations).

Note: If you are a Mac developer and want to improve this situation, please contact us. A possible solution for the perfomance issues may be something like a docker-sync implementation for Riptide.

Get help and join the community

If you need some support or just want to chat with the community, join our Slack workspace.

Next steps

The next pages of this documentation will explain how to finish the setup of Riptide, how to setup the Proxy server and how to install the Bash/Zsh integration. It will also teach you how to use the Riptide CLI and Proxy server.

Please make sure to read through all of the following pages of this documentation to properly setup Riptide.

Windows

This guide will explain how to install Riptide under Windows.

Note: Windows is not supported as well as the Linux setup. Most of the downsides of Riptide on Windows come from the Docker Desktop implementation for Windows.

Riptide has some Performance optimizations to increase the performance on Windows, but it will still be slower than running it on Linux.

Also we can not offer any Windows specific support at the moment.

If you have experience with Docker or Python on Windows, we'd love your support in making Riptide on Windows even better!

Installing Requirements

This guide assumes you want to run Riptide in the most common set-up using the Docker Engine. To use Riptide you need to have the following installed:

- Python 3.6+ * Download: Python website.
- pip for Python 3 (might come installed with Python)
- Docker Desktop 16.0+

There is a good chance you already have Python installed. Try running `python3 --version` to check.

Installing Riptide system-wide

To install all Riptide components and the Docker implementation run the following command:

```
$ pip3 install riptide-all
```

You can test if Riptide is working:

Installing Riptide in a Virtualenv

Riptide can also be installed in a Virtualenv. This is only recommended for advanced Python users. Please make sure, to use the correct Python interpreter of your Virtualenv when setting up the proxy server.

Updating Riptide

To update Riptide, run

    $ riptide_upgrade

Configuring shared drives

When installing Riptide on a drive other than C:, or when using projects from other drives, you may need to share this drive with the Docker VM. A notice about this should automatically open in this case.

Additional Windows related notes

Many additional settings or issues not described in this documentation may be directly related to the Docker Desktop for Windows implementation.

Please see the documentation for Docker Desktop for Windows for further information.

Known issues under Windows

- Riptide currently uses the default Docker Desktop Windows daemon. This setup is known to have significantly worse performance than the Linux version. Riptide has some Performance optimizations to increase performance.
- Due to the performance optimization settings, it might happen that changes to files are not immediately visible on the host system or the running containers. Some files are not updated on the host system at all (see Performance optimizations).

Note: If you are a Windows developer and want to improve this situation, please contact us. A possible solution for the perfomance issues may be something like a docker-sync implementation for Riptide or using Docker with WSL2 instead of using Docker Desktop. If you do, please share your experience!

Get help and join the community

If you need some support or just want to chat with the community, join our Slack workspace.

Next steps

The next pages of this documentation will explain how to finish the setup of Riptide, how to setup the Proxy server and how to install the Bash/Zsh integration. It will also teach you how to use the Riptide CLI and Proxy server.

Please make sure to read through all of the following pages of this documentation to properly setup Riptide.

Installing individual components

Instead of installing all Riptide components via the `riptide-all` package, you can also install individual parts of Riptide separately.

Core components:

```
$ [sudo] pip3 install riptide-proxy riptide-cli riptide-engine-docker
```

Database drivers, additional support for database management:

```
$ [sudo] pip3 install riptide-db-mysql # MySQL
```

Plugins, used to integrate Riptide better with special needs of some programming languages or frameworks:

```
$ [sudo] pip3 install riptide-plugin-php-xdebug # Required for the PHP debugger Xdebug
```

3.1.3 Configuration

This page will show you how to edit the system configuration file of Riptide (also refered to as "user configuration file").

Initial configuration

Create your system configuration file using `riptide config-edit-user`. This will open an editor.

Leave everything on default for now, individual settings will be explained below.

After creating the configuration file using this command, Riptide CLI is now ready to use! Continue to the next chapters to learn how to use it with a project and how to setup the Proxy Server.

Proxy server configuration

The configuration for the Proxy Server is described in the chapter Proxy Server Setup.

Repository configuration

The `repos` key contains a list of repositories, that are used by Riptide to look up components of projects.

By default the community repository is the only repository in this list. Please see Using Repositories for more info.

Engine configuration

The entry under `engine` defines which container engine implementation is used. Currently only `docker` is supported.

Editing the configuration file manually

You can use the command `riptide config-edit-user` to edit the configuration file.

Alternatively you can also directly edit the file "<CONFIG>/config.yml" in your favorite editor.

Advanced: Resolving hostnames & /etc/hosts file

Riptide uses a proxy server to route traffic to your projects. This proxy server uses hostnames to route traffic. These hostnames need to be routable to your local machine.

In order to make this easy for you, Riptide (by default) automatically updates the /etc/hosts file (may have a different path under different OSes). However in order to do so, **your local user needs write access to this file**. To change permissions under Linux, you can use the following command:

```
sudo setfacl -m u:<YOUR USERNAME>:rw  /etc/hosts
```

Replace `<YOUR USERNAME>` with your username.

If you don't want to change permissions to the file, you can instead add these entries manually. If Riptide can't update the file, it will prompt you with a message, whenever it needs updating:

```
[marco@kaptriel demo-projekt]$ riptide
Warning: Could not update the hosts-file (/etc/hosts) to configure proxy server routing.
> Give your user permission to edit this file, to remove this warning.
> If you wish to manually add the entries instead, add the following entries to /etc/hosts:
127.0.0.1       dummy__hello_world.riptide.local
```

Manual routing

Alternatively you can disable the automatic update of the hosts file by setting `update_hosts_file` to `false` in the configuration file.

In this case, you need to make sure, all project URLs are routed correctly via DNS.

Assuming you set the proxy server to run under `riptide.local` the following hostnames must be routable to your local machine using DNS:

- `riptide.local`
- `*.riptide.local`

3.1.4 Shell Integration

Riptide has integrations for the popular Bash and Zsh shells. We highly recommend installing these!

CLI Command Aliases

Riptide projects may define custom commands for you to use. Take for example a command called `mysql`. To run it *without* the integration you have to execute:

```
$ riptide cmd mysql -e "DESCRIBE Hello;"
```

If the shell integration is enabled, you can just run the command like you would any other shell command:

```
$ mysql -e "DESCRIBE Hello;"
```

> **Warning:** We highly recommend using the shell integration. The `riptide cmd` command does not support passing all arguments and options.

Install the integration

If you are using **Bash**, add the following line to your `.bashrc`:

```
. riptide.hook.bash
```

If you are using **Zsh**, add the following line to your `.zshrc`:

```
. riptide.hook.zsh
```

You need to re-open your terminals for the integration to be enabled (or source your bashrc/zshrc).

Warning: When using Riptide inside a virtualenv, you need to replace `riptide.hook.bash` with the full path to `riptide.hook.bash`. You can get that by calling `which riptide.hook.bash`. The same applies for the zsh integration.

Note: If you want to try these commands out yourself using the demo project from the following chapters, you may need to start the database first: `riptide start -s db`.

Warning: Whenever you set up a project for the first time, you need to exit and re-enter the project directory to use the commands.

Autocomplete

Riptide has limited experimental autocomplete support.

To enable it for **Bash**, add the following line to your `.bashrc`:

```
eval "$(_RIPTIDE_COMPLETE=source_bash riptide)"
```

To enable it for **Zsh**, add the following line to your `.zshrc`:

```
eval "$(_RIPTIDE_COMPLETE=source_zsh riptide)"
```

Replace `<full_path_to_riptide>` with the full path to the `riptide` command. On Mac and Linux you can get this path by executing `which riptide`.

You need to re-open your terminals or source the rc-file inside them for the integration to be enabled.

Warning: When using Riptide inside a virtualenv, you need to replace `riptide` with the full path to `riptide`. You can get that by calling `which riptide`.

3.1.5 Proxy Server Setup

The Riptide proxy server routes the traffic for your projects and services, you use it to access HTTP-based services of the project you are working on.

Proxy Server URL

The Proxy server URL can be configured by calling `riptide config-edit-user` and changing the value of `riptide.proxy.url`.

Enter the hostname you want your proxy server to be accessible at there.

By default Riptide will add entries to your system's hosts-file to make sure your projects can be routed at this address. See "Advanced: Resolving hostnames & /etc/hosts file" for more information..

If you change this address, and have hosts-file management enabled, you may need to run any command of the Riptide CLI to update the hosts-file with the new domains.

Proxy Server HTTP/HTTPS Ports

The proxy server can route HTTP and HTTPS traffic. You can change the ports by editing the system configuration (`riptide config-edit-user`) and changing the values of `riptide.proxy.ports`.

If you plan to use the proxy server standalone as your primary HTTP and HTTPS server on your machine, leave the defaults (80 and 443).

If you already have a web server on ports 80 and 443 and/or plan to use the Riptide proxy behind a reverse proxy (eg. Nginx or Apache), change the ports to something else, preferably a four-digit port combination (eg. 8080 and 8443).

You can also disable HTTPS by setting the value for `https` to `false`. Do this if you want to run the proxy server behind a reverse proxy with SSL termination.

Starting the Proxy Server

How to start the proxy server depends on your system.

Linux, ports <= 1024

If you are on Linux and the port number of either HTTP or HTTPS is below 1024, you need to start the Proxy Server as root. These elevated privileges are required for applications to be able to bind ports below 1024. After binding the port the proxy will automatically drop all it's privileges to the user executing sudo.

To start the server in this scenario:

```
$ sudo riptide_proxy
Was running as root. Changing user to marco.
Starting Riptide Proxy on HTTPS port 443
Starting Riptide Proxy on HTTP port 80
```

After starting the proxy server head over to the URL you configured for the proxy server and you should see a landing page for the proxy server.

Mac , ports <= 1024

Running the proxy server with ports lower than 1024 may or may not be possible on your system based on how your machine is set up.

You may be able to follow the "all other platforms" section of this guide. However if this does not work for you please use higher ports and configure firewall rules.

All other platforms

On all other platforms and on Linux when using ports above 1024, you can start the proxy server as your current user without sudo or an Administrator Command Line:

```
$ riptide_proxy
Starting Riptide Proxy on HTTPS port 443
Starting Riptide Proxy on HTTP port 80
```

After starting the proxy server head over to the URL you configured for the proxy server and you should see a landing page for the proxy server.

Start the Proxy on system boot

You may want to start the proxy server automatically whenever you log in, this section describes how to do so for different platforms.

Linux, ports <= 1024 (Systemd)

When using the proxy server with ports below 1024, the server needs to be run as root. This means for autostart it has to be configured as a system level service.

Create the following unit file under `/etc/systemd/system/riptide.service`:

```
[Unit]
Description=Riptide

[Service]
ExecStart=<PROXY> --user=<USERNAME>
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

You need to replace `<USERNAME>` with your username and `<PROXY>` with the full path to the proxy executable which you can get by calling `which riptide_proxy`.

After that you need to reload the Systemd daemon:

```
sudo systemctl daemon-reload
```

To enable autostart:

```
sudo systemctl enable riptide
```

To start the proxy server right away:

```
sudo systemctl start riptide
```

Linux, ports > 1024 (Systemd)

When using ports above 1024 it is best to configure the proxy server as a user level unit. This means that the proxy server is directly bound to your user account and will autostart on login.

Create the following unit file under `~/.config/systemd/user/riptide.service`:

```
[Unit]
Description=Riptide

[Service]
ExecStart=<PROXY>
Restart=on-failure

[Install]
WantedBy=default.target
```

You need to replace `<PROXY>` with the full path to the proxy executable which you can get by calling `which riptide_proxy`.

After that you need to reload the Systemd daemon:

```
sudo systemctl daemon-reload
```

To enable autostart:

```
systemctl --user enable riptide
```

To start the proxy server right away:

```
systemctl --user start riptide
```

Other platforms

There is no info on how to do this on other platforms here yet. Please start the proxy server manually as described above.

Running the Proxy Server behind Nginx or Apache

You may want to run Riptide behind an Nginx or Apache proxy. This is especially useful if you work on projects that don't use Riptide.

This guide will show you how to do that, assuming you set the HTTP port of Riptide proxy to 8888 and disabled HTTPS. This guide assumes Nginx or Apache will terminate SSL for you.

Nginx

```
server {
  listen 80;
  listen [::]:80;

  # Configure SSL if desired
  #listen *:443 ssl http2;
  #listen [::]:443 ssl http2;
  #ssl_certificate ...
  #ssl_certificate_key ...

  server_name <INSERT PROXY HOSTNAME HERE>;
  server_name *.<INSERT PROXY HOSTNAME HERE>;

  client_max_body_size 2G;

  location / {
      proxy_pass            http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>;
      proxy_read_timeout    90000;
      proxy_send_timeout    90000;
      proxy_connect_timeout 90000;
      send_timeout          90000;

      proxy_set_header      X-Real-IP $remote_addr;
      proxy_set_header      X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header      Host $host;
      proxy_set_header      X-Forwarded-Proto $scheme;
```

(continues on next page)

(continued from previous page)

```
  }

  # WebSocket Reverse Proxy
  location /___riptide_proxy_ws {
    proxy_pass http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>;
    proxy_http_version 1.1;
    proxy_set_header Host $host;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
  }

}
```

Apache

The modules `proxy`, `proxy_http` and `proxy_wstunnel` must be enabled.

```
<VirtualHost *:80>
    ServerName <INSERT PROXY HOSTNAME HERE>
    ServerAlias *.<INSERT PROXY HOSTNAME HERE>

    RewriteCond %{HTTP:Upgrade} =websocket [NC]
    RewriteRule ^/___riptide_proxy_ws    ws://127.0.0.1:<INSERT PROXY HTTP PORT HERE>/
↪___riptide_proxy_ws [P,L]

    ProxyPreserveHost On
    ProxyTimeout 90000
    ProxyPass / http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>/
    ProxyPassReverse / http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>/
</VirtualHost>

<IfModule mod_ssl.c>
<VirtualHost *:443>
    ServerName <INSERT PROXY HOSTNAME HERE>
    ServerAlias *.<INSERT PROXY HOSTNAME HERE>

    RewriteCond %{HTTP:Upgrade} =websocket [NC]
    RewriteRule ^/___riptide_proxy_ws    wss://127.0.0.1:<INSERT PROXY HTTP PORT HERE>
↪/___riptide_proxy_ws [P,L]

    ProxyPreserveHost On
    ProxyTimeout 90000
    ProxyPass / http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>/
    ProxyPassReverse / http://127.0.0.1:<INSERT PROXY HTTP PORT HERE>/
</VirtualHost>
</IfModule>
```

Import the SSL certificate authority

If you enable the HTTPS feature of the proxy server, you probably want to import the certificate authority (CA) into your browser, so that you don't get an SSL warning every time you restart the proxy server or enter a different project.

Location

The CA file is located under "<CONFIG>/riptide_proxy/ca.pem".

The file is created on the first startup of the proxy server. You can also place your own CA file here.

Chrome

1. Navigate to `chrome://settings/certificates?search=SSL`
2. Go to the tab for certificate authorities
3. Click Import and import the CA file, mark it as trusted to identify websites.

Firefox

1. Navigate to `about:preferences#privacy`
2. Search for "Certificates" and press the "View Certificates…" button.
3. On the "Authorities" tab "Import…" the CA certificate. Trust the certificate to identify websites.

Auto-Start services

The proxy server can automatically start projects if you access the URL for a service. To disable this set `riptide.proxy.autostart` to `false` in the system configuration. `true` enables it.

3.1.6 Project Setup

The first time you want to use a project, it has to be set up.

For this part of the guide, we will be using a demo project to guide you through the setup process.

This demo project contains everything you may encounter while setting up real projects, so we recommend you place this demo project into an empty directory and follow this guide first before setting up a real project.

Demo project (place in `riptide.yml` in empty directory):

```
project:
  name: dummy
  src: .
  app:
    name: dummy
    import:
      dummy_directory:
        target: dummy-files
        name: Anything-this-is-just-a-demo
    notices:
      usage: This usage text shows you additional things you need to do when running␣
↪this project.
    services:
      hello_world:
        image: strm/helloworld-http
        port: 80
        run_as_current_user: false
```

(continues on next page)

(continued from previous page)

```
        roles:
          - main
      db:
        image: mysql:8.0
        roles:
          - db
        driver:
          name: mysql
          config:
            database: dummy
            password: mysql
        run_as_current_user: false
    commands:
      mysql:
        image: "{{ parent().get_service_by_role('db').image }}"
        command: "mysql -hdb -uroot -pmysql dummy"
```

Running the first-time setup

First, make sure all repositories and Docker images are up to date:

```
$ riptide update
Updating Riptide repositories...
    ...

Updating images...
    [service/hello_world] Pulling 'strm/helloworld-http':
        Done!
    [command/db] Pulling 'mysql:8.0':
        Done!
    [command/mysql] Pulling 'mysql:8.0':
        Done!
    Done!
```

You should run this command regularly to make sure your images and repositories are always up to date. See the Docker documentation for more details on images. See Using Repositories for more information on repositories.

To run the first-time setup run:

```
$ riptide setup
Thank you for using Riptide!
This command will guide you through the initial setup for dummy.
Please follow it very carefully, it won't take long!
> Press any key to continue...
```

This will update all repositories and images and start the setup. After starting the setup, press any key:

```
> BEGIN SETUP

Usage notes for running dummy with Riptide:
    This usage text shows you additional things you need to do when running this␣
↪project.

> Do you wish to run this interactive setup? [Y/n]
```

Riptide will then show you the usage notes that are defined for the app your project is using. This usage note may contain additional steps you need to run **after** the setup. If you need to view this again, run `riptide notes` after the setup.

Confirm that you want to run the interactive setup by pressing `y`.

Tip: If you accidentally press `n` or make a mistake later during the setup, you can always restart it by passing the `--force` option.

After pressing `y` you will be asked what kind of setup you want to do:

```
> INTERACTIVE SETUP
> Are you working on a new project that needs to be installed or do you want to␣
↪Import existing data? [n/I]
```

If you press `n` Riptide will exit and show you instructions for the first-time installation of the application you are using. Follow these instructions.

If you press `i` you will be guided through the import of existing data. What can be imported depends on the project. For this dummy project, a MySQL database can be imported, Riptide will tell you this after you pressed `i`:

```
> EXISTING PROJECT
  > DATABASE IMPORT
> Do you want to import a database (format mysql)? [Y/n]
```

For this demo, open a text editor and put the following contents in a file called `demo.sql`:

```
CREATE TABLE Hello (
    World varchar(255)
);
```

Enter `y` to confirm that you want to import an SQL file:

```
Enter the path to the SQL file.
```

Enter the path to the SQL file that you just downloaded:

```
Enter the path to the SQL file. demo.sql
-----
Starting services...

mysql: 2/6|                          | Pulling image... Downloading :...
```

You can see that the database is now starting, your SQL file will be imported shortly:

```
-----
Starting services...

mysql: 6/6|| Started!

Waiting for database...
Importing into database environment default... this may take a while...

Database environment default imported.

-----
```

After the database is imported, the project may ask you to import other directories, such as directories containing media files or configuration specific to the application:

```
-----
    > FILE IMPORT
        > dummy_directory IMPORT
    > Do you wish to import Anything-this-is-just-a-demo to <project>/dummy-files? [Y/
↪n]
```

In our example it doesn't really matter. You may try this out by confirming with `y` and entering a path to a directory. It will be copied into the dummy-files directory inside the current directory:

```
> Do you wish to import Anything-this-is-just-a-demo to <project>/dummy-files? [Y/n] y
Enter path of files or directory to copy: /tmp/test_dir
-----
Importing dummy_directory (dummy-files) from /tmp/test_dir
Copying... this can take some time...
Done!
-----
```

After the import, or after you skipped it, Riptide will inform you that it is done:

```
> IMPORT DONE!
All files were imported.

DONE!

...
```

Next steps

The project is now set-up. If you are setting up a real project, there may need to be some additional steps you have to do now, that you were told in the usage notes. If you need to view these notes again run `riptide notes`. This will show you both the general usage notes, that may contain things you need to do after importing an existing project, and installation notes, for starting from scratch.

If you want to import databases or files later on, see Managing Databases and Importing Files.

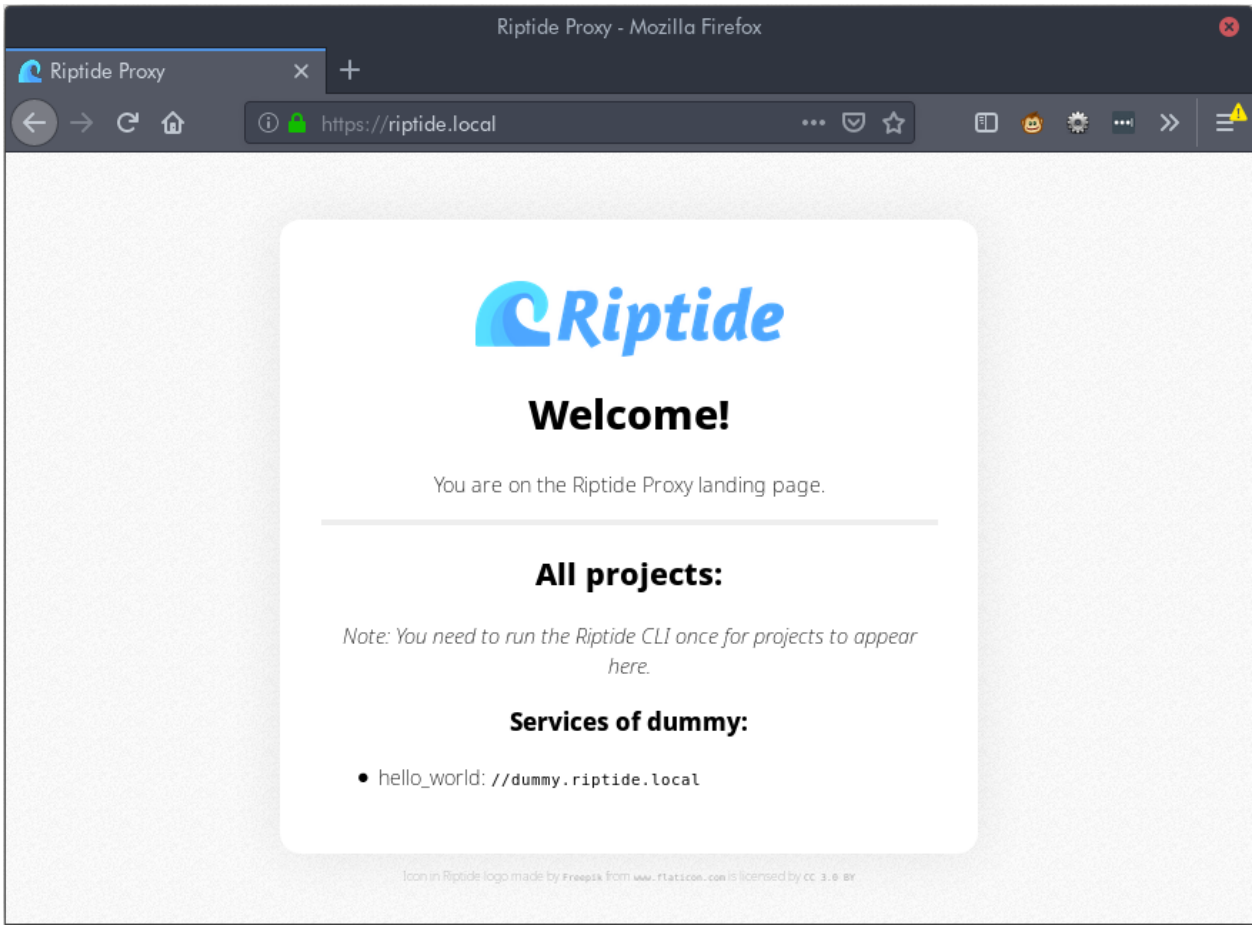
3.1.7 Working with Riptide

Now you have everything set up! It is time to access your project through Riptide.

This part of the guide will show you how to do daily tasks with Riptide.

Access your projects web services

First make sure that the proxy server is started. After that head to the URL you configured for the proxy. You should be greeted with a landing page:



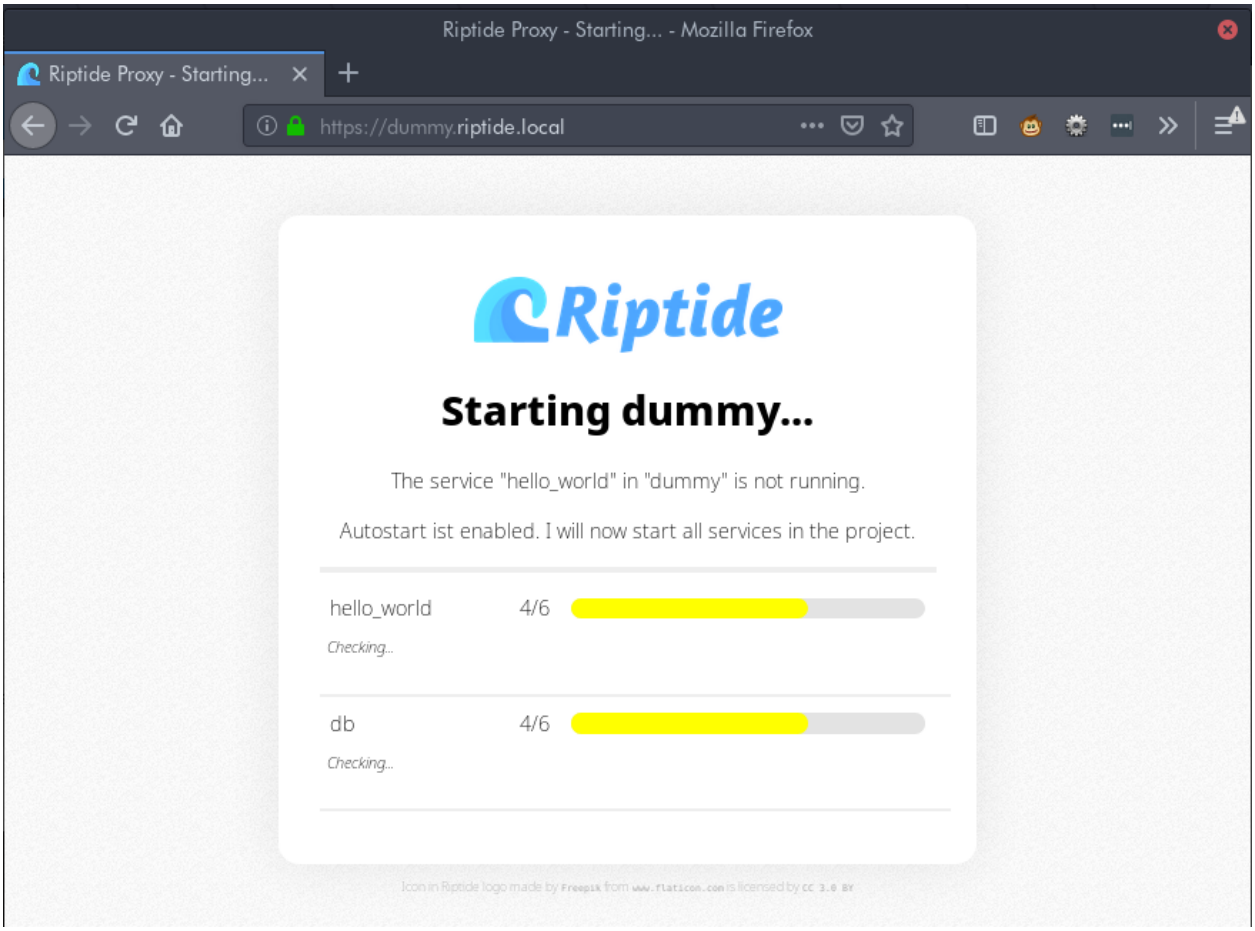
As you can see, projects with all services that have HTTP capabilities are listed here. You can click on a link to access the service.

All service URLs have the following structure:

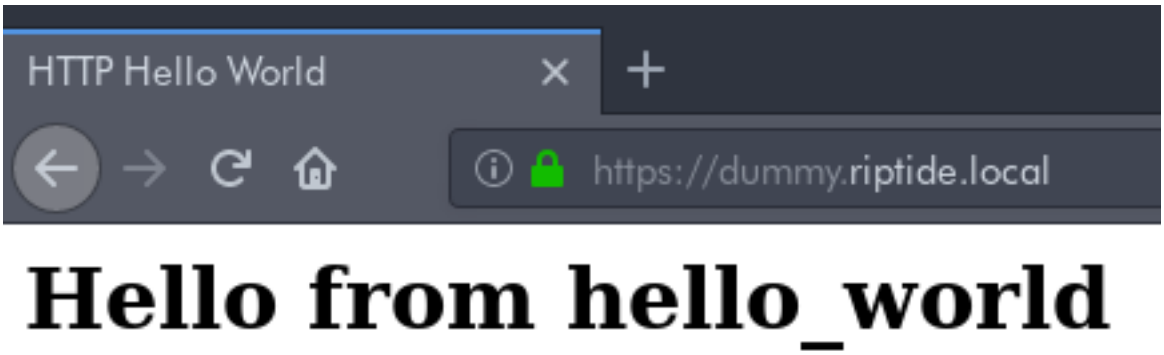
```
<project>(--<service>).<proxy-url>
```

If your service is configured to be the main service of the URL, it's url is simply the URL of the project (eg. `dummy.riptide.local`). If your project has multiple services, than the other services are accessible by adding two dashes to the name of the project (eg. `dummy--service2.riptide.local`).

Accessing the dummy project's hello_wold service as shown in the screenshot above, will present you with the autostart page of this web service:



After your project has started up, you will see the contents served by your web service:



Access other TCP/UDP ports

A project may provide other, non HTTP services. Riptide allows services to define additional ports which will be bound to your local machine on a port that will always stay the same.

For example, if you have two projects, both with a MySQL server that would normally run on port 3306, then the first project may reserve the port 3306 and the second one 3307. The ports will always stay the same for these projects so you can configure your SQL software accordingly.

To view the additional ports for a project, run `riptide status` after the services have been started:

```
Status:
    hello_world:
        Running.
        Access via http://dummy.riptide.local

    db:
        Running.
        Additional Ports:
            Port MySQL Port (3306) reachable on localhost:3309
```

Running CLI commands

Note: This section assumes you have the Shell Integration set up. If not, prefix all commands with `riptide cmd`.

A project may define helpful shell commands for you to use.

To list them run `riptide cmd`:

```
$ riptide cmd
  Commands:
      - mysql
```

To run a command, simply execute it on the shell (you need to be inside the project directory):

```
$ mysql --help
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql  Ver 8.0.15 for Linux on x86_64 (MySQL Community Server - GPL)
...
```

Warning: When using the shell integration and you have just set up a project, you need to leave and re-enter the project to use commands.

Warning: Piping (`|`, `<`, `>`) is not supported for Riptide commands. If you need to pipe input, you may be able to run the command directly in the *shell of a service*.

Warning: The `--help` flag does not work as expected when running commands with `riptide cmd`, it will always show the help for `riptide cmd` instead. Please set up the shell integration if you need the `--help` flag.

Starting and stopping services via CLI

You can start and stop services on the CLI by using the `start`, `restart` and `stop` commands. You can pass the `-s` flag to only affect certain services (comma separated):

```
$ riptide stop -s hello_world,db
Stopping services...

hello_world: 2/3|                  | Stopping...
db         : 3/3|| Stopped!
```

To view the names and status of all services run `riptide status`.

Running services in foreground

Sometimes you might need to run a command in foreground mode (attached to your console; interactively) like you would run other commands. This may be needed if you want to debug the service. For example when using NodeJS you can configure this with the debugger of your IDE to start and stop your application service via the IDE and have it attach it's debugger.

To run a service in foreground use `start-fg`. In this example a service named `varnish` is run in foreground:

```
$ riptide start-fg -s www varnish
(1/3) Starting other services...
Starting services...

www: 2/2|| Already started!

(2/3) Stopping varnish...
Stopping services...

varnish: 3/3|| Stopped!

(3/3) Starting in varnish foreground mode...
bind(): Cannot assign requested address
child (37) Started
Child (37) said Child starts
```

Please note that some service options are ignored when running a service interactively:

- The logging options for stdout and stderr are ignored. Instead stdout and stderr are directly sent to the terminal.
- `pre_start` and `post_start` commands are not run.
- The `src` role is added to the service. This means that the source code of your application will always be available for the service.
- `working_directory` is ignored. The working directory is set to the directory you are currently in. If you are not currently inside the project, the working directory is set to the root of the project.

A note about paths and directories

Please note that all containers used to run your application use a separate file system from your own.

The path configured in the `src` setting inside the `riptide.yml` is available for all services with the `src` role and all commands under `/src`.

If you see paths in logs and other places `/src` always represents the project `src` setting.

You CAN NOT access files on your machine that are outside of the `src` directory. Under normal circumstances, this will be no problem. When you start commands and are inside the project `src`-folder you can access files like normal, because Riptide will automatically run the command in the correct directory inside the container.

However **you can not use any paths that are outside the project's ‘‘src‘‘ directory**.

Let's take the following example: We have a directory tree like so:

```
/home/me/my_projects
 -> project
    -> riptide.yml
    -> a_file
-> other_directory
    -> b_file
```

The `src` setting is set to `.`, meaning that all commands and services have the entire `/home/me/my_projects` directory mounted to `/src`.

Because of this, the following will work as expected. `my_command` will be able to access `a_file`:

```
$ pwd
/home/me/my_projects/project
$ riptide cmd my_command a_file
$ riptide cmd my_command ./a_file
$ riptide cmd my_command /src/a_file
```

However the following will **NOT** work. `my_command` will find neither `a_file` nor `b_file`:

```
$ pwd
/home/me/my_projects/project
$ riptide cmd my_command /home/me/my_projects/project/a_file
$ riptide cmd my_command ../other_directory/b_file
$ riptide cmd my_command /home/me/my_projects/other_directory/b_file
```

Directly access the shell of a service

This should usually not be required, but you can directly access the shell of the containers the services run in by running `riptide exec service_name`.

If you need root access inside of the container, pass the flag `--root`.

3.1.8 Managing Databases

If your project uses a supported database you may be able to use the database features of Riptide. These features allow you to manage different environments of your database and to switch between them, for example if you are developing a new feature.

Note: Database environments are an abstraction over whatever database management your database software comes with. It completely isolates the entire physical database files in different directories.

If you switch the environment you tell Riptide to use a different directory for the data of your database.

Listing environments and status

`riptide db-status` shows you the current database environment and `riptide db-list` lists them.

Note: If these commands fail with the message "No such command", then database management is not available for your project.

Creating a new environment

Use `riptide db-new NAME` to create a new EMPTY database environment. This also switches the current environment to this new one. See *Copying.html* instead if you want to copy an existing environment.

Importing and exporting

You can import and export dumps of the currently active database environment. The format of this dump depends on the database driver that the project is using.

`riptide db-import FILE` to import, `riptide db-export FILE` to export.

Note: Depending on the database driver this may export/import the entire database server with all "sub-databases" or only one active "sub-database". For the MySQL driver for example it only exports and imports one configured primary database.

Copying

To copy an existing environment, use `riptide db-copy FROM TO`, where `FROM` is the name of the environment to copy from and `TO` the name of the new environment.

Switches to the new environment.

Deleting

To delete an environment use `riptide db-drop NAME`. You can not delete the active environment.

Warning: This can not be undone.

3.1.9 Importing Files

Riptide supports the definition of directories that are used to import files into during the project setup.

You can also import files after the setup is completed by running `riptide import-files KEY PATH_TO_IMPORT`.

`KEY` is the import key. You can find his by executing `riptide config-dump` to output the entire project configuration. The import keys are defined under `riptide.project.app.import`.

`PATH_TO_IMPORT` is the path of the directory to import.

3.1.10 Log Files

A service in a project may define log files. These log files may be from the standard output, the standard error, a file inside of the service container or the output of a utility command inside the service container.

To view the log files, open the directory `<project>/_riptide/logs/`. You will find a directory in there for each service that defines logs. Inside the directories are the log files.

Log files don't get cleared after a service reboots. If you want to clear them manually, stop the service and remove the files.

3.1.11 Using Repositories

Parts of project configuration may be stored in external repositories. Repositories make it easy to share Services, Commands or other parts of the configuration across multiple projects.

You can check if your project uses repositories by opening the project's `riptide.yml`. If it contains `$ref`-keys then parts of the configuration are merged with documents from the repositories. If, for example, the `app` entry contains a `$ref` entry with the value `app/demo`, then Riptide searches for the App by searching for `app/demo.yml` inside all your configured repositories. The `app/demo.yml` is loaded as your App and then the contents under `app` in the project's `riptide.yml` are merged together.

More information about repositories, can be found in the configuration guide.

You can change repositories by running `riptide config-user-edit`. Repositories are defined as a list under the `repo` key. Riptide repositories are Git repositories. Enter the clone-URLs for your repositories there.

You can update (pull) the current contents of all repositories by running `riptide update`. This command also updates all project images.

The repositories are stored in the "<CONFIG>/repos" folder. Since they are ordinary Git repositories you can pull and push repositories that are stored there.

3.1.12 PHP Debugging with XDebug

By default XDebug is disabled when using PHP projects, for performance reasons.

If you want to enable XDebug, make sure you have the `riptide-plugin-php-xdebug` package installed. This is installed by default since Riptide 0.5.0.

See Status of XDebug

To see if XDebug is currently enabled for a project, use the following command:

```
$ riptide xdebug
Xdebug status for riptidedocs: Disabled.
Detected Xdebug version: 3
Mode: debug
Extra configuration:
Request trigger: no (xdebug.start_with_request=yes)
```

Enable / Disable XDebug

Use the following commands to toggle XDebug for all running services and all Riptide commands for a single project:

```
$ riptide xdebug off
$ riptide xdebug on
```

Xdebug Mode

By default Xdebug is configured to run in "develop" mode. Using the `-m/--mode` option you can change the mode Xdebug should use. You can comma-seperate multiple modes.

This setting only applies to Xdebug 3.

Activation method

By default `xdebug.start_with_request` is set to `yes`, so the only triggers configuring whether or not debugging should happen are this command and your IDE listening for connections or not.

If you want, you can use `--request/-r` to set this value to `trigger`. The debugger will then only connect if it detects the cookie for it (see Xdebug documentation). To disable this again use `--no-request/-R`.

This setting only applies to Xdebug 3.

Additional configuration

You can pass additional configuration as comma-seperated key-value pairs using the option `--config/-c`. This is the same format as used by the `XDEBUG_CONFIG` environment variable.

For example, to set the `xdebug.log` and `xdebug.log_level` settings:

> riptide xdebug -c 'log=/tmp/xdebug.log,log_level=10' on

Xdebug Version

The plugin tries to automatically detect the Xdebug version. For this it tries the following:

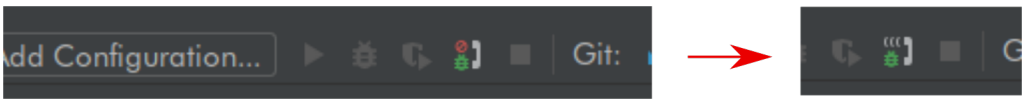
- First it checks if the environment variable (that Riptide is running with!) `RIPTIDE_XDEBUG_VERSION` is set to either `2` or `3`.
- (**RECOMMENDED**) Otherwise it checks if the label `php_xdebug_version` of the image assigned to the first service in the app with the role *php*' is set to either `2` or `3`.
- Otherwise, it checks if in the currently loaded configuration the environment variable `RIPTIDE_XDEBUG_VERSION` is set to either `2` or `3` at ANY service or comamnd in the app (absolutely not recommended).

Based on the determined value, it will set the correct Xdebug configuration. If no version could be detected, Riptide will assume version 2 and output a warning.

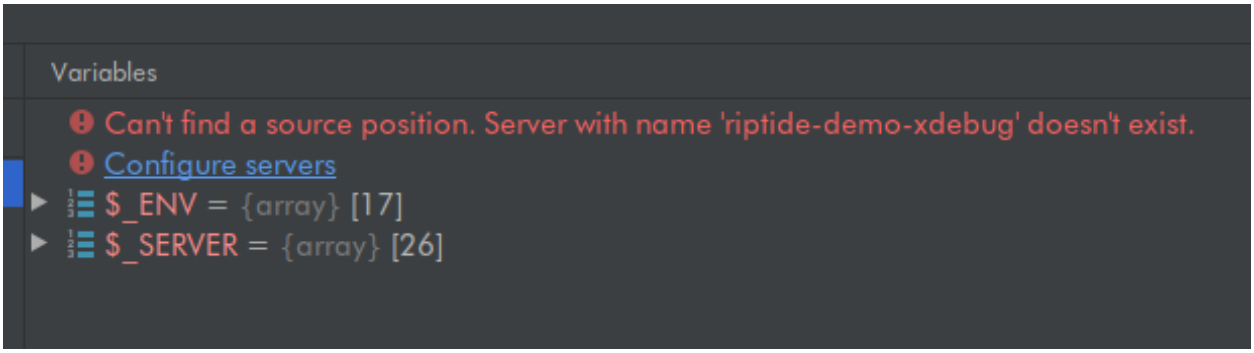
Enable XDebug in PHPStorm

When XDebug is enabled, it will automatically try to connect.

Enable Remote Debugging in PHPStorm to accept debugging connections:

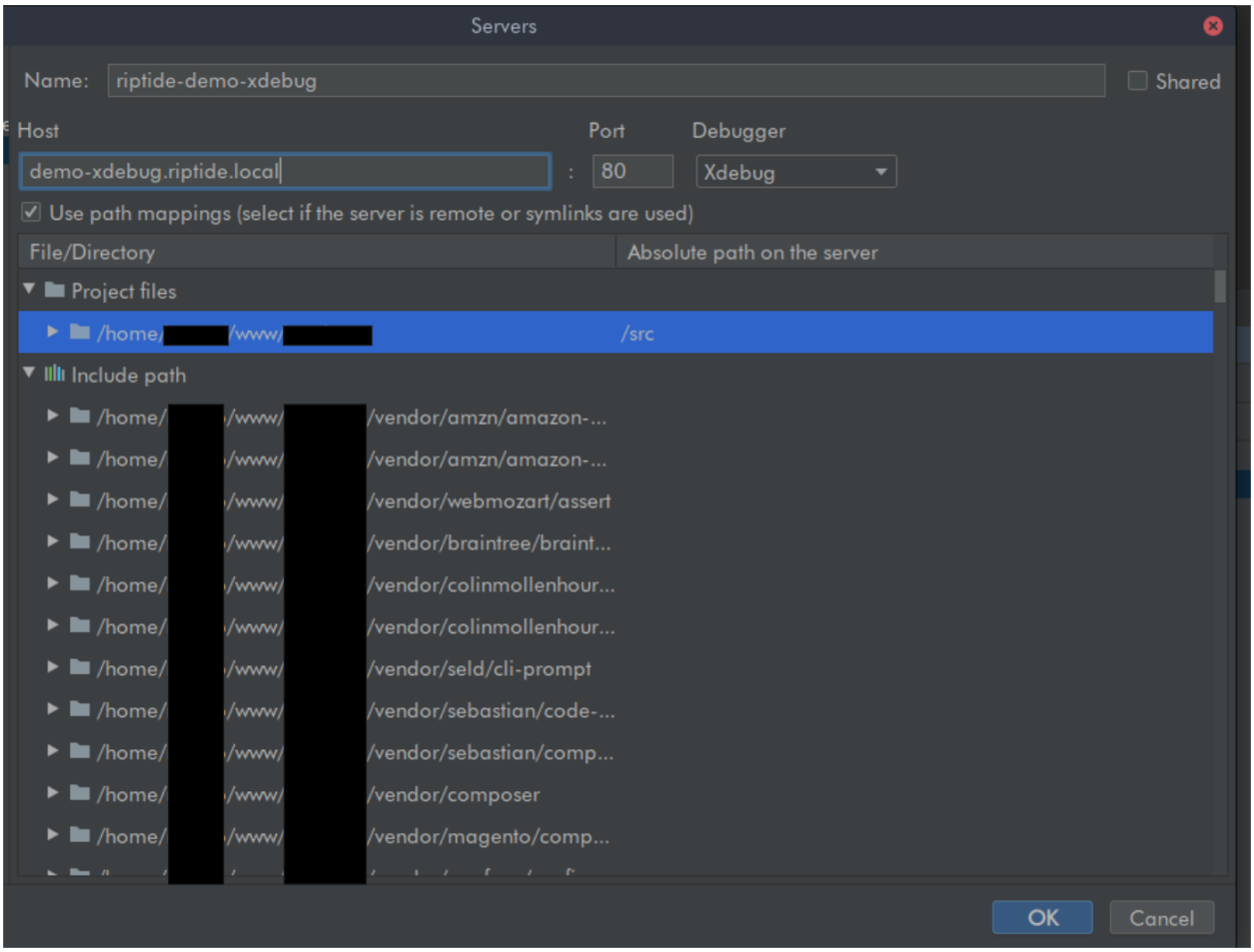


When a service container tries to connect for the first time, you have to configure path mappings. You will see the following warning in the debugging window of PhpStorm:



Click on "Configure Server". Create a new server entry and as a "Name" enter the name that was shown in the previous warning (in this case `riptide-demo-xdebug`). As "Host" enter the URL of the project/service.

Check "Use path mappings". Select the sub-directory your `src` entry of your `riptide.yml` points to. In this example `src` is `.`, so we will select the top project directory. Enter `/src` as "Absolute Path on the server" and save.



The debugger should now work for this project.

3.1.13 Performance Optimizations

Riptide has some settings for performance optimizations that may be enabled on any platform, but only bring benefits on some.

Riptide also has some fixed built-in performance optimizations for specific platforms.

Configurable Performance Optimizations

These performance optimizations can be toggled in the Riptide system configuration (`riptide config-edit-user`).

They are found under the `performance` key. The default value for all settings is `auto`, which means that Riptide will automatically decide, of the performance option should be enabled or not.

Named Volumes instead of Host-Path volumes (dont_sync_named_volumes_with_host)

If enabled, volumes, that have a volume_name set, are not mounted to the host system and are instead created as volumes with the volume_name. Otherwise they are created as host path volumes only. Enabling this increases performance on some platforms.

Please note, that Riptide does not delete named volume data for old projects. Please consult the documentation of the engine, on how to do that.

"auto" enables this feature on Mac and Windows, when using the Docker container backend.

Switching this setting on or off breaks existing volumes. They need to be migrated manually. See Update notes for version 0.5.0

Do not synchronize unimportant paths with the host system (dont_sync_unimportant_src)

Normally all Commands and Services get access to the entire source directory of a project as volume. If this setting is enabled, `unimportant_paths` that are defined in the App are not updated on the host system when changed by the volume. This means changes to these files are not available, but file access speeds may be drastically increased on some platforms.

Currently all files written inside the container are lost on container restart. The files are currently written to RAM.

"auto" enables this feature on Mac and Windows, when using the Docker container backend.

This feature can be safely switched on or off. Projects need to be restarted for this to take effect.

Platform-specific optimizations

MacOS

Under MacOS when using Docker, the performance setting `delegated` is set for volumes. This means that sometimes changes to files within the container are not immediately visible on the host system.

See the Docker documentation for more details.

3.2 Configuration Guide

Welcome to the Riptide Configuration Guide.

This guide will guide you through writing your own Riptide projects. It explains the mechanics of the YAML-based configuration language and the different entities that control how Riptide projects behave.

The Chapter How Riptide works explains the mechanics of the Riptide configuration and what repositories are.

The Chapter Entities contains the specification of all entities.

Examples shows you how to get quickly started by creating projects based on templates from the Riptide Community Repository.

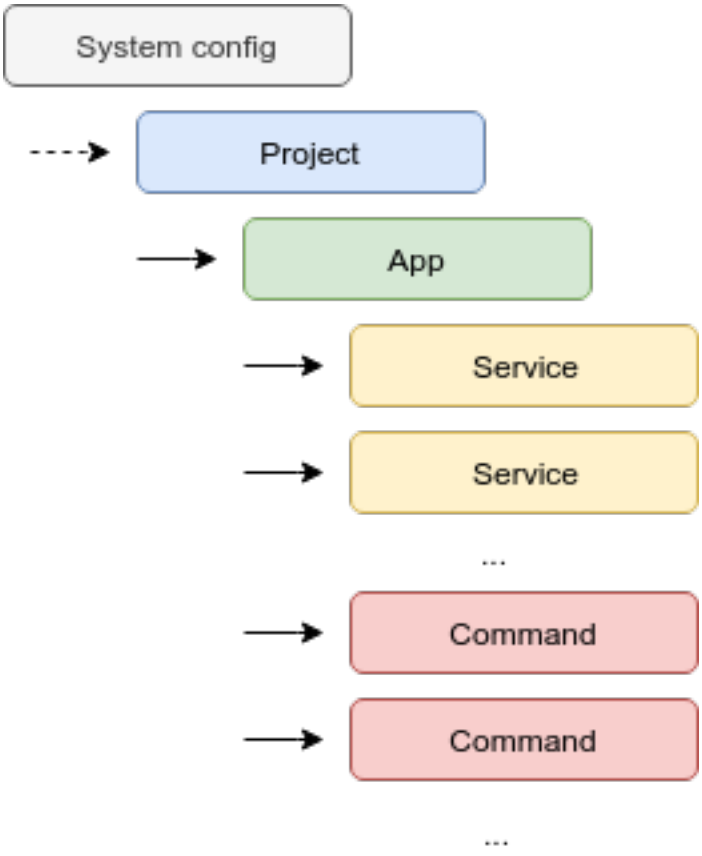
3.2.1 How Riptide works

Overview / Hierarchy

Riptide's configuration is made up of a hierarchy of entities (also called objects or documents).

The currently loaded configuration is based on the system configuration under "<CONFIG>/config.yml" and the currently loaded project, which is read from a `riptide.yml`. Additionally, if a `riptide.local.yml` exists, it's contents are merged on top of the `riptide.yml`.

Projects contain an App, which contains Services and Commands. And so the configuration forms a hierarchy of different entities:



The entire (fully processed) configuration can be viewed by using the CLI command `riptide config-dump`. Projects are internally inserted under the system configuration:

```
$ riptide config-dump
# Riptide configuration

# This is the final configuration that was created by merging all configuration files␣
↪together
# and resolving all variables.
# Includes some internal system keywords (keys with $, except $ref).
riptide:
  engine: docker
  proxy:
    autoexit: 15
    autostart: true
    ports:
      http: 80
      https: 443
    url: riptide.local
  repos:
  - git@github.com:Parakoopa/riptide-repo-private.git
  update_hosts_file: true
  project:
    $path: /home/example/riptide/demo-project/riptide.yml
    app:
      commands:
        echo_me:
          $name: echo_me
          command: echo
          image: alpine
      name: dummy
      services:
        hello_world:
          $name: hello_world
          image: strm/helloworld-http
          port: 80
          run_as_current_user: false
          roles:
            - main
    name: dummy
    src: ./src
```

The files that make up each configuration entity are simple YAML files with a header depending on their type.

Example project entity that contains one app entity under app. This app contains service entities under “services”.

```
project:
  name: foo
  src: .
  app:
    name: bar
    services:
      hello_world:
        image: strm/helloworld-http
        port: 80
        run_as_current_user: false
        roles:
          - main
```

Creating Projects

To create projects, create a new file called "riptide.yml" in the root of your project.

This file is a YAML file with the header "project". The rest of the file is filled with keys and values, based on the project's schema. See Projects for more details.

Schemas

Each entity has a schema that defines it. The configuration files you create must fit to this schema. The schema for all entities is explained in the following sections.

Variables

Strings in entity documents may contain variables. These variables are references to fields in the same document.

Example:

```yaml
project:
  name: foo
  src: .
  notes:
    usage: "Image - {{ app.services.hello_world.image }}"
  app:
    name: bar
    services:
      hello_world:
        image: strm/helloworld-http
```

Result:

```yaml
project:
  name: foo
  src: .
  notes:
    usage: "Image - strm/helloworld-http"
  app:
    name: bar
    services:
      hello_world:
        image: strm/helloworld-http
```

Helper Functions

In addition to variables, helper functions (also called "Variable Helpers") can be used to perform advanced tasks. All entities have one helper called `parent()` that returns the parent entity.

Example:

```yaml
app:
  name: bar
  services:
    hello_world:
      image: '{{ parent().name }}'
```

Result:

```
app:
  name: bar
  services:
    hello_world:
      image: bar
```

In this example `parent()` is called on the service called `hello_world`. The parent of this service is the app, and so `parent().name` returns the name of the app.

Repositories

Entities can contain references to other documents.

Example:

```
app:
  name: bar
  services:
    hello_world:
      $ref: /service/hello-world
      command: 'this will override the comnmand in /service/hello-world'
```

Riptide will load the entity contained in the file `service/hello-world.yml` inside one of the repositories, that is specified in the system configuration and merge it with the one defined here.

More information on repositories can be found under "How Repositories work".

Details about how documents are processed

All of the properties described here are based on the Python library Configcrunch.

If you want additional information about the behaviour of Configcrunch, please have a look at the Configcrunch documentation.

How Repositories work

As explained in the User Documentation repositories contain apps, services and commands that can be used inside projects.

To use an entity from a repository, simply reference it in your project file via `$ref`:

```
project:
  name: demo
  src: .
  app:
    $ref: /app/demo
    services:
      hello_world:
        command: 'this will override the comnmand for hello_world in app/demo.yml'
```

Riptide will look trough all cloned repositories that are defined in the system configuration under `repos`. It will start with the first repository and search for a file named `app/demo.yml`.

If it finds this file it will merge the contents of your project file on top of `app/demo.yml`. It will then do the same for all other repositories defined under `repos`, so you can configure multiple repositories that override each other and build a hierarchy of repositories using this technique.

You can view the merged result by calling `riptide config-dump`.

Entities defined in the repository can also reference other entities, even using relative paths.

Repositories are cloned and updated whenever `riptide update` is run.

Complex example

This is a complex example using two repositories and multiple references. By looking at this example it should become clear, how repositories work.

This is the base project file for our example:

```yaml
project:
  name: demo
  src: .
  app:
    $ref: /app/demo
    services:
      hello_world:
        command: 'this will override the comnmand for hello_world in app/demo.yml'
      additional_service:
        $ref: /service/demo
        image: 'this will override the image in service/demo.yml'
```

And this is the content of our system configuration's `repos` setting:

```yaml
repos:
  - https://repos.example/repo1.git
  - https://repos.example/repo2.git
```

`repo1.git` contains the following files:

```yaml
# <repo1.git>/app/demo
app:
  services:
    hello_world:
      image: alpine
      command: 'echo hello world'
```

```yaml
# <repo1.git>/service/demo
service:
  image: ubuntu
  command: demo
```

`repo2.git` contains the following files:

```yaml
# <repo2.git>/app/demo
app:
  services:
    hello_world:
      image: debian
```

The end result is the following project file:

```
project:
  name: demo
  src: .
  app:
    services:
      hello_world:
        image: debian
        command: 'this will override the comnmand for hello_world in app/demo.yml'
      additional_service:
        image: 'this will override the image in service/demo.yml'
        command: demo
```

Removing values

During the merging process it is possible to remove values entirely using the special keyword `$renove`.

Example (remove a service from a loaded app):

```
project:
  name: demo
  src: .
  app:
    $ref: /app/demo
    services:
      hello_world: $remove
```

Details about how documents are processed

More information about the properties of Riptide's configuration language, can be found in the section Overview / Hierarchy.

The configuration language is based on the Python library Configcrunch.

If you want additional information about the behaviour of Configcrunch, please have a look at the Configcrunch documentation.

Riptide Community Repository

The Riptide Community Repository is the default repository for Riptide.

It contains many helpful apps, services and commands maintained by the community. This part of the guide will use entities from this repository to build projects.

A list of all entities can be found here.

Is something missing from the repository? Feel free to fork the repository and add the things you need. Or set up your own and use both. If you built something, please contribute to the community repository to make it even better!

3.2.2 Entities

System (User) Configuration

The system configuration is the main configuration file that defines global behaviour for Riptide, such as the proxy server configuration.

It is located under "<CONFIG>/config.yml".

Schema

The Schema defines what the contents of the YAML configuration files are.

classmethod Config.**schema**() → schema.Schema

proxy

url: str Base-URL for the proxy server. The name of projects and/or services will be appended to it.

For example *projectname.riptide.local* would route to the project *projectname* if *riptide.local* is specified.

ports

http: int HTTP port that the proxy server should listen on

https: Or[int,bool] HTTPS port that the proxy server should listen on, or false to disable HTTPS

autostart: bool Whether or not the proxy server should auto-start all services for a project if a user enters the URL for a service.

[autostart_restrict]: List[str] If set, only the IPv4 ip addresses specified by the netmasks in this list are allowed to trigger the auto-start process via the proxy server. For other clients, projects are not automatically started. Useful if you share a network with co-workers and don't want them to start your projects.

[compression]: bool If true, the proxy server doesn't decompress any data, and instead passes the compressed data of the backend server (if compressed). Experimental.

engine: str Engine to use, the Python package for the engine must be installed.

repos: List[str] List of URLs to Git repositories containing Riptide Repositories.

update_hosts_file: bool Whether or not Riptide should automatically update the system's host file.

[project]: Project If a project is loaded, Riptide inserts the project here. Do not manually insert a project into the actual system configuration file.

performance Various performance optimizations that, when enabled, increase the performance of containers, but might have some other drawbacks.

Values can be true/false/auto. "auto" enables an optimization, if beneficial on your platform.

dont_sync_named_volumes_with_host: Or['auto',bool] If enabled, volumes, that have a volume_name set, are not mounted to the host system and are instead created as volumes with the volume_name. Otherwise they are created as host path volumes only. Enabling this increases performance on some platforms.

Please note, that Riptide does not delete named volume data for old projects. Please consult the documentation of the engine, on how to do that.

"auto" enables this feature on Mac and Windows, when using the Docker container backend.

Switching this setting on or off breaks existing volumes. They need to be migrated manually.

dont_sync_unimportant_src: Or['auto', bool] Normally all Commands and Services get access to the entire source directory of a project as volume. If this setting is enabled, `unimportant_paths` that are defined in the App are not updated on the host system when changed by the volume. This means changes to these files are not available, but file access speeds may be drastically increased on some platforms.

"auto" enables this feature on Mac and Windows, when using the Docker container backend.

This feature can be safely switched on or off. Projects need to be restarted for this to take effect.

Example Document:

```
riptide:
  proxy:
    url: riptide.local
    ports:
      http: 80
      https: 443
    autostart: true
    autostart_restrict:
      - 127.0.0.1/32
  engine: docker
  repos:
    - https://github.com/theCapypara/riptide-repo.git
  update_hosts_file: true
  performance:
    dont_sync_named_volumes_with_host: auto
    dont_sync_unimportant_src: auto
```

Helper Functions

Helper Functions (also called "Variable Helpers") can be used in the configuration files to perform some advanced tasks.

Config.**get_config_dir**()

Variable Helper

Can be used inside configuration files.

Returns the path to the Riptide system configuration directory

Example usage:

```
something: '{{ get_config_dir() }}'
```

Example result:

```
something: '/home/thomas/.config/riptide'
```

Config.**get_plugin_flag**(*inp: str*) → any

Variable Helper

Can be used inside configuration files.

Returns the value (usually true/false, but can also be other data) of a flag set by a Riptide plugin.

If the flag or plugin is not found, false is returned.

Parameters **inp** – plugin-name.flag-name

Projects

Projects represent one web development project.

They are loaded from `riptide.yml` files. Additionally, if a `riptide.local.yml` exists, it's contents are merged on top of the `riptide.yml`.

A project consists of one app.

Schema

The Schema defines what the contents of the YAML configuration files are.

classmethod `Project.`**schema**`()` → schema.Schema

name: str Unique name of the project.

src: str Relative path of the source code directory (relative to riptide.yml). Services and Commands only get access to this directory.

app: App App that this project uses.

[links]: List[str] Links to other projects (list of project names).

Riptide will add all service containers in this project in the TCP/IP networks of all projects specified here. This way services in your project can communicate with services from other projects and vice-versa. If a project in this list does not exist, Riptide will ignore it.

Please make sure, that service names are not re-used across projects that are linked this way, this could lead to unexpected results during service host name resolution.

[default_services]: List[str] List of services to start when running *riptide start*. If not set, all services are started. You can also control which services to start using flags. See *riptide start –help* for more information.

[env_files]: List[str] A list of paths to env-files, relative to the project path, that should be read-in by services and command when starting. See the `read_env_file` flag at `Service` and `Command` for more information.

Defaults to ["./.env"].

Example Document:

```
project:
  name: test-project
  src: src
  app:
    $ref: apps/reference-to-app
```

Helper Functions

Helper Functions (also called "Variable Helpers") can be used in the configuration files to perform some advanced tasks.

`Project.`**parent**`()` → Config

Variable Helper

Can be used inside configuration files.

Returns the system configuration document.

Example usage:

```
something: '{{ parent().proxy.url }}'
```

Example result:

```
something: 'riptide.local'
```

Apps

An app defines all the different services (sub-applications) and commands that are required to run a web development project for a specific framework or application.

An app consists of a number of services and commands, a list of files that can be imported during the initial setup and some usage notes.

Schema

The Schema defines what the contents of the YAML configuration files are.

classmethod App.**schema**() → schema.Schema

name: str Name describing this app.

[notices]

[usage]: str Text that will be shown when the interactive setup wizard ist started.

This text should describe additional steps needed to finish the setup of the app and general usage notes.

[installation]: str Text that will be shown, when the user selects a new installation (from scratch) for this app.

This text should explain how to execute the first-time-setup of this app when using Riptide.

[import]

{key} Files and directories to import during the interactive setup wizard.

target: str Target path that the file or directory should be imported to, relative to the directory of the riptide.yml

name: str Human-readable name of this import file. This is displayed during the interactive setup and should explain what kind of file or directory is imported.

[services]

{key}: Service Services for this app.

[commands]

{key}: Command Commands for this app.

[unimportant_paths]: List[str] Normally all files inside containers are shared with the host (for commands and services with role 'src'). This list specifies files that don't need to be synced with the host. This means, that these files will only be uploaded to the container on start and changes will not be visible on the host. Changes that are made on the host file system may also not be visible inside the container. This increases performance on non-native platforms (Mac and Windows).

This feature is only enabled if the system configuration performance setting `dont_sync_unimportant_src` is enabled. If the feature is disabled, all files are shared with the host. See the documentation for that setting for more information.

All paths are relative to the src of the project. Only directories are supported.

Example Document:

```
app:
  name: example
  notices:
    usage: Hello World!
  import:
    example:
      target: path/inside/project
      name: Example Files
  services:
    example:
      $ref: /service/example
  commands:
    example:
      $ref: /command/example
```

Helper Functions

Helper Functions (also called "Variable Helpers") can be used in the configuration files to perform some advanced tasks.

App.**parent**() → Project

Variable Helper

Can be used inside configuration files.

Returns the project that this app belongs to.

Example usage:

```
something: '{{ parent().src }}'
```

Example result:

```
something: '.'
```

App.**get_service_by_role**(*role_name: str*) → Optional[riptide.config.document.service.Service]

Variable Helper

Can be used inside configuration files.

Returns any service with the given role name (first found) or None.

Example usage:

```
something: '{{ get_service_by_role("main")["$name"] }}'
```

Example result:

```
something: 'service1'
```

Parameters **role_name** – Role to search for

App.**get_services_by_role**(*role_name: str*) → List[riptide.config.document.service.Service]

Variable Helper

Can be used inside configuration files.

Returns all services with the given role name.

Parameters **role_name** – Role to search for

Services

A service is the definition of a software container that contains one of the applications required to run the entire app.

Since services are container definitions, they need to contain at least the name of an image to run. All other fields are optional.

Schema

The Schema defines what the contents of the YAML configuration files are.

classmethod Service.**schema**() → schema.Schema

[$name]: str Name as specified in the key of the parent app.

Added by system. DO NOT specify this yourself in the YAML files.

[roles]: List[str] A list of roles for this service. You can use arbitrary strings and get services by their assigned roles using *get_service_by_role()*.

Some roles are pre-defined and have a special meaning:

***main*:** This service is the main service for the app.

Some commands will default to this service and the proxy URL for this service is shorter. Usually services are accessible via `http://<project_name>--<service_name>.<proxy_url>`, however the main service is accessible via `http://<project_name>.<proxy_url>`.

Only one service is allowed to have this role.

***src*:** The container of this service will have access to the source code of the application.

It's working directory will be set accordingly.

***db*:** This service is the primary database. A database driver has to be set (see key `driver`).

This service is then used by Riptide for *database management </user_docs/db.html>*.

image: str Docker Image to use

[command]: str or map

If this is not set: The default command in the image is used and considered in the "default" command group (see below).

If it is a string: Command to run inside of the container. Default's to command defined in image. This command will be in the "default" command group (see below).

If it is a map: A list of commands that this service supports. Keys are the "command group", values the commands to run. Each service must have a command defined for the "default" command group. You can speficy a command group to use when using *riptide start*. Default is the "default" command group, this one is also used by the Riptide Proxy autostart feature. For more information on this see the *–cmd* flag of *riptide start*.

Example:

```
comamnd:
  default: "npm run default"
  debug: "npm run debug"
```

Warning: Avoid quotes (", ') inside of commands, as those may lead to strange side effects.

[port]: int HTTP port that the web service is accessible under. This port will be used by the proxy server to redirect the traffic.

If the port is not specified, the service is not accessible via proxy server.

[logging] Logging settings. All logs will be placed inside the "_riptide/logs" directory.

[stdout]: bool Whether or not to log the stdout stream of the container's main command. Default: false

[stderr]: bool Whether or not to log the stderr stream of the container's main command. Default: false

[paths]

{key}: str Additional text files to mount into the logging directory. Keys are filename's on host (without .log) and values are the paths inside the containers.

[commands]

{key}: str Additional commands to start inside the container. Their stdout and stderr will be logged to the file specified by the key.

[pre_start]: List[str] List of commands to run, before the container starts. They are run sequentially. The startup will wait for the commands to finish. Exit codes (failures) are ignored.

Each of these commands is run in a separate container based on the service specification. Each command is run in a "sh" shell.

[post_start]: List[str] List of commands to run, after container starts. They are run sequentially. The startup will wait for the commands to finish. Exit codes (failures) are ignored.

Each of these command's is run inside the service container (equivalent of `docker exec`). Each command is run in a "sh" shell.

[environment] Additional environment variables

{key}: str Key is the name of the variable, value is the value.

[working_directory]: str Working directory for the service, either

- absolute, if an absolute path is given
- relative to the src specified in the project, if the role "src" is set.
- relative to the default working directory from the image, if the role is not set.

Defaults to `..`

[config] Additional configuration files to mount. These files are NOT directly mounted. Instead they are processed and the resulting file is mounted.

All variables and variable helpers inside the configuration file are processed.

Processed config files are either written to _riptide/processed_config and mounted to containers or (if they are under the source tree of the project and the service has the role 'src') copied to the path in the project and mounted with the rest of the source tree. A '.riptide_info.txt' is added then to explain the origin of this file.

Example configuration file (demo.ini):

```
[demo]
domain={{domain()}}
project_name={{parent().parent().name}}
```

Resulting file that will be mounted:

```
[demo]
domain=projectname.riptide.local
project_name=projectname
```

{key}

from: str Path to the configuration file, relative to any YAML file that was used to load the project (including "riptide.yml" and all yaml files used inside the repository; all are searched). Absolute paths are not allowed.

to: str Path to store the configuration file at, relative to working directory of container or absolute.

[force_recreate: bool] False by default. If false, command containers that use this config file will not try to recreate the processed file if it already exists. If true command containers will also recreate the file every time they are started. Started services always recreate the processed file on start, regardless of this setting.

[additional_ports] Additional TCP and/or UDP ports that will be made available on the host system. For details see section in user guide.

{key}

title: str Title for this port, will be displayed in `riptide status`

container: int Port number inside the container

host_start: int First port number on host that Riptide will try to reserve, if the port is already occupied, the next one will be used. This port will be reserved and permanently used for this service after that.

[additional_volumes] Additional volumes to mount into the container for this command.

{key}

host: str Path on the host system to the volume. Avoid hardcoded absolute paths.

container: str Path inside the container (relative to src of Project or absolute).

[mode]: str Whether to mount the volume read-write ("rw", default) or read-only ("ro").

[type]: str Whether this volume is a "directory" (default) or a "file". Only checked if the file/dir does not exist yet on the host system. Riptide will then create it with the appropriate type.

[volume_name]: str Name of a named volume for this additional volume. Used instead of "host" if present and the dont_sync_named_volumes_with_host performance setting is enabled. Volumes with the same volume_name have the same content, even across projects. As a constraint, the name of two volumes should only be the same, if the host path specified is also the same, to ensure the same behaviour regardless of if the performance setting is enabled.

[driver] The database driver configuration, set this only if the role "db" is set.

Detailed documentation can be found in a separate section.

name: str Name of the database driver, must be installed.

config: ??? Specification depends on the database driver.

[run_as_current_user]: bool Whether to run as the user using riptide (True) or image default (False).

Default: True

Riptide will always create the user and group, matching the host user and group, inside the container on startup, regardless of this setting.

Some images don't support switching the user, set this to false then. Please note that, if you set this to false and also specify the role 'src', you may run into permission issues.

[run_pre_start_as_current_user]: 'auto' or bool Whether to run pre start commands the user using riptide or image default. Default is 'auto' which means the value of *run_as_current_user* will be used.

[run_post_start_as_current_user]: 'auto' or bool Whether to run post start commands the user using riptide or image default. Default is 'auto' which means the value of *run_as_current_user* will be used.

[allow_full_memlock]: bool Whether to set memlock ulimit to -1:-1 (soft:hard). This is required for some database services, such as Elasticsearch. Note that engines might ignore this setting, if they don't support it.

Default: False

[read_env_file]: bool If enabled, read the environment variables in the env-files defined in the project (`env_files`). Default: True

Example Document:

```
service:
  image: node:10
  roles:
    - main
    - src
  command: 'node server.js'
```

(continues on next page)

(continued from previous page)

```
port: 1234
logging:
  stdout: true
  stderr: false
  paths:
    one: '/foo/bar'
  commands:
    two: 'varnishlog'
pre_start:
  - "echo 'command 1'"
  - "echo 'command 2'"
post_start:
  - "echo 'command 3'"
  - "echo 'command 4'"
environment:
  SOMETHING_IMPORTANT: foo
config:
  one:
    from: ci/config.yml
    to: app_config/config.yml
working_directory: www
additional_ports:
  one:
    title: MySQL Port
    container: 3306
    host_start: 3006
additional_volumes:
  temporary_files:
    host: '{{ get_tempdir() }}'
    container: /tmp
```

Helper Functions

Helper Functions (also called "Variable Helpers") can be used in the configuration files to perform some advanced tasks.

`Service.`**`parent`**`()` → App

Variable Helper

Can be used inside configuration files.

Returns the app that this service belongs to.

Example usage:

```
something: '{{ parent().notices.usage }}'
```

Example result:

```
something: 'This is easy to use.'
```

`Service.`**`system_config`**`()` → Config

Variable Helper

Can be used inside configuration files.

Returns the system configuration.

Example usage:

```
something: '{{ system_config().proxy.ports.http }}'
```

Example result:

```
something: '80'
```

Service.**volume_path**() → str

Variable Helper

Can be used inside configuration files.

Returns the (host) path to a service-unique directory for storing container data.

Example usage:

```
additional_volumes:
    cache:
        host: '{{ volume_path() }}/cache'
        container: '/foo/bar/cache'
```

Example result:

```
additional_volumes:
    cache:
        host: '/home/peter/my_projects/project1/_riptide/data/service_name/cache'
        container: '/foo/bar/cache'
```

Service.**get_working_directory**() → str

Variable Helper

Can be used inside configuration files.

Returns the path to the working directory of the service **inside** the container.

Warning: Does not work as expected for services started via "start-fg".

Example usage:

```
something: '{{ get_working_directory() }}'
```

Example result:

```
something: '/src/working_dir'
```

Service.**domain**() → str

Variable Helper

Can be used inside configuration files.

Returns the full domain name that this service should be available under, without protocol. This is the same domain as used for the proxy server.

Example usage:

```
something: 'https://{{ domain() }}'
```

Example result:

```
something: 'https://project--service.riptide.local'
```

Service.**os_user**() → str

Variable Helper

Can be used inside configuration files.

Returns the user id of the current user as string (or 0 under Windows).

This is the same id that would be used if "run_as_current_user" was set to *true*.

Example usage:

```
something: '{{ os_user() }}'
```

Example result:

```
something: '1000'
```

Service.**os_group**() → str

Variable Helper

Can be used inside configuration files.

Returns the id of the current user's primary group as string (or 0 under Windows).

This is the same id that would be used if "run_as_current_user" was set to *true*.

Example usage:

```
something: '{{ os_group() }}'
```

Example result:

```
something: '100'
```

Service.**host_address**() → str

Variable Helper

Can be used inside configuration files.

Returns the hostname that the host system is reachable under inside the container.

Example usage:

```
something: '{{ host_address() }}'
```

Example result:

```
something: 'host.riptide.internal'
```

Service.**home_path**() → str

Variable Helper

Can be used inside configuration files.

Returns the path to the home directory inside the container.

Example usage:

```
something: '{{ home_path() }}'
```

Example result:

```
something: '/home/riptide'
```

Service.**get_tempdir**() → str

Variable Helper

Can be used inside configuration files.

Returns the path to the system (host!) temporary directory where the user (should) have write access.

Example usage:

```
something: '{{ get_tempdir() }}'
```

Example result:

```
something: '/tmp'
```

Helpfer Functions for configuration files

The helper functions listed here can only be used inside files used with the `config` setting of services.

`riptide.config.service.config_files_helper_functions.`**`read_file`**(*config_file_path: str*, *file_to_read_in: str*) → str

Reads the contents of a file, relative to the configuration file being processed.

Can not access files in parent directories. Variables in the included file are not processed.

The parameter `config_file_path` is automatically filled by Riptide.

Example usage:

```
{{ read_file('example.txt') }}
```

Example result:

```
contents of example.txt
```

Commands

A command is the specification for a container that can be started interactively by the user. This is used to start CLI command containers.

Commands can either be invoked via `riptide cmd` or directly via Riptide's shell integration.

Commands either run as separate containers in the same container network as services (normal commands), or are started in running service containers.

Schema

The Schema defines what the contents of the YAML configuration files are.

`classmethod` `Command.`**`schema`**() → schema.Schema

Can be either a normal command, a command in a service, or an alias command.

`classmethod` `Command.`**`schema_normal`**()

Normal commands are executed in seperate containers, that are running in the same container network as the services.

[$name]: str Name as specified in the key of the parent app.

Added by system. DO NOT specify this yourself in the YAML files.

image: str Docker Image to use

[command]: str Command to run inside of the container. Default's to command defined in image.

> **Warning:** Avoid quotes (", ') inside of the command, as those may lead to strange side effects.

[additional_volumes] Additional volumes to mount into the container for this command.

{key}

host: str Path on the host system to the volume. Avoid hardcoded absolute paths.

container: str Path inside the container (relative to src of Project or absolute).

[mode]: str Whether to mount the volume read-write ("rw", default) or read-only ("ro").

[type]: str Whether this volume is a "directory" (default) or a "file". Only checked if the file/dir does not exist yet on the host system. Riptide will then create it with the appropriate type.

[volume_name]: str Name of a named volume for this additional volume. Used instead of "host" if present and the dont_sync_named_volumes_with_host performance setting is enabled. Volumes with the same volume_name have the same content, even across projects. As a constraint, the name of two volumes should only be the same, if the host path specified is also the same, to ensure the same behaviour regardless of if the performance setting is enabled.

[environment] Additional environment variables

{key}: str Key is the name of the variable, value is the value.

[config_from_roles]: List[str] List of role names. All files defined under "config" for services matching the roles are mounted into the command container.

[read_env_file]: bool If enabled, read the environment variables in the env-files defined in the project (`env_files`). Default: True

Example Document:

```
command:
  image: riptidepy/php
  command: 'php index.php'
```

classmethod Command.**schema_in_service**()

Command is run in a running service container.

If the service container is not running, a new container is started based on the definition of the service.

[$name]: str Name as specified in the key of the parent app.

Added by system. DO NOT specify this yourself in the YAML files.

in_service_with_role: str Runs the command in the first service which has this role.

May lead to unexpected results, if multiple services match the role.

command: str Command to run inside of the container.

> **Warning:** Avoid quotes (", ') inside of the command, as those may lead to strange side effects.

[environment] Additional environment variables. The container also has access to the environment of the service. Variables in the current user's env will override those values and variables defined here, will override all other.

{key}: str Key is the name of the variable, value is the value.

[read_env_file]: bool If enabled, read the environment variables in the env-files defined in the project (`env_files`). Default: True

Example Document:

```
command:
  in_service_with_role: php
  command: 'php index.php'
```

classmethod Command.**schema_alias**()

Aliases another command.

[$name]: str Name as specified in the key of the parent app.

Added by system. DO NOT specify this yourself in the YAML files.

aliases: str Name of the command that is aliased by this command.

Helper Functions

Helper Functions (also called "Variable Helpers") can be used in the configuration files to perform some advanced tasks.

Command.**parent**() → App

Variable Helper

Can be used inside configuration files.

Returns the app that this command belongs to.

Example usage:

```
something: '{{ parent().notices.usage }}'
```

Example result:

```
something: 'This is easy to use.'
```

Command.**system_config**() → Config

Variable Helper

Can be used inside configuration files.

Returns the system configuration.

Example usage:

```
something: '{{ system_config().proxy.ports.http }}'
```

Example result:

```
something: '80'
```

Command.**volume_path**() → str

Variable Helper

Can be used inside configuration files.

Returns the (host) path to a command-unique directory for storing container data.

Example usage:

```
additional_volumes:
    command_cache:
        host: '{{ volume_path() }}/command_cache'
        container: '/foo/bar/cache'
```

Example result:

```
additional_volumes:
    command_cache:
        host: '/home/peter/my_projects/project1/_riptide/cmd_data/command_name/
↪command_cache'
        container: '/foo/bar/cache'
```

Command.**os_user**() → str

Variable Helper

Can be used inside configuration files.

Returns the user id of the current user as string (or 0 under Windows).

This is the same id that would be used if "run_as_current_user" was set to *true*.

Example usage:

```
something: '{{ os_user() }}'
```

Example result:

```
something: '1000'
```

Command.**os_group**() → str

Variable Helper

Can be used inside configuration files.

Returns the id of the current user's primary group as string (or 0 under Windows).

This is the same id that would be used if "run_as_current_user" was set to *true*.

Example usage:

```
something: '{{ os_group() }}'
```

Example result:

```
something: '100'
```

Command.**host_address**() → str

Variable Helper

Can be used inside configuration files.

Returns the hostname that the host system is reachable under inside the container.

Example usage:

```
something: '{{ host_address() }}'
```

Example result:

```
something: 'host.riptide.internal'
```

`Command.home_path() → str`

Variable Helper

Can be used inside configuration files.

Returns the path to the home directory inside the container.

Example usage:

```
something: '{{ home_path() }}'
```

Example result:

```
something: '/home/riptide'
```

`Command.get_tempdir() → str`

Variable Helper

Can be used inside configuration files.

Returns the path to the system (host!) temporary directory where the user (should) have write access.

Example usage:

```
something: '{{ get_tempdir() }}'
```

Example result:

```
something: '/tmp'
```

3.2.3 Examples

NodeJS Hello World

This section will guide you through the setup of a simple NodeJS project using the Riptide repository.

This guide assumes you have Riptide fully set up, with shell integration enabled and a running proxy server (for this guide we assume `https://riptide.local` as base URL of your proxy server). It also assumes you have the `repos` part of the configuration set to only the Riptide Community Repository (the default).

NodeJS does NOT need to be installed for this guide.

Preparing the project

For this guide we will set up a very simple Express-based web server. You can probably adapt this guide to more complex applications.

Create a new directory and create an `index.js` in it with the following contents:

```
// Source: https://expressjs.com/starter/hello-world.html
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

Create a `package.json` containing express as a dependency:

```
{
  "name": "js-helloworld",
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

Creating a basic riptide.yml

Create a `riptide.yml` with the following contents:

```
project:
  name: js-helloworld
  src: .
  app:
    name: js-helloworld
    services:
      nodejs:
        image: node:10
        command: 'node index.js'
        port: 3000
        roles:
          - src
          - main
```

This file contains one project named `js-helloworld`. We specify with `src` that the source code for this project is in the same directory that the `riptide.yml` is in.

This project contains an app called `js-helloworld`. This app has one service called `nodejs`. This service is the container specification for our Hello World app.

The service `nodejs` needs a Docker image with Node.js in it, so we specify the `image` `node:10`. Our script is in the `index.js`, so we tell Riptide to run `node index.js` as the `command` of our service.

Our Hello World app (http) runs on port `3000`, so we tell Riptide this by setting `port` to it.

The final step is adding `roles`. Roles define the behaviour of services.

The `src` role gives our service access to the source code (the `index.js` file). The `main` role sets the service as the main service for our project.

Adding commands for NPM

Next we need to add the `node` and `npm` commands to our project, so that we can run `npm` to install express from the `package.json`.

Add the following under `app` in the `riptide.yml`:

```
commands:
  node:
    $ref: /command/node/10
  npm:
    $ref: /command/npm/node10
```

This adds two new commands, one containing NodeJS and one containing npm. All npm processes started will also have access to the directory `.npm` in your home directory and your `.npmrc`.

Those commands come from the Riptide repository, if you want to know how they work, visit the repository:

- /command/node/10
- /command/npm/node10

Running the project setup

Run `riptide setup --skip` to initiate the project. Since we have not added any setup instructions or files to import, we just skip the setup with the `--skip` flag.

Installing requirements

If you have the shell integration enabled, leave and enter the directory again, this will load the configured `npm` and `node` commands. You can now run `npm install`, which will install express and create a directory named `node_modules`.

Starting the project

Since the project's dependencies (express) are now installed, you can open the front page of the Proxy server (`https://riptide.local`). You will find a new project called `js-helloworld`.

Click on the link and the project will start. After it starts you will see the "Hello World!" message telling you, that the project works.

Enable logging

If you want to enable logging, add the following lines to the service `nodejs`:

```
logging:
  stdout: true
  stderr: true
```

You can restart the project by using `riptide restart`. After the restart you will find logging files in `_riptide/logs/nodejs`.

Adding files for import and setup instructions

For our simple example there are no files to import and we don't really need any setup instructions.

However the `riptide setup` command supports usage notes and importing files, as you can see in the User Documentation. You can also see an example project there.

To add usage notes, add the following to the `app`:

```
notices:
  usage: >-
    This is a demo usage note.

    You can also use variables here: {{ services.nodejs.image }}

  installation: >-
    This will be shown when the user chooses to set up a new project.
```

The user (and you) can view those notes by calling `riptide notes`. They are also shown during `riptide setup`. The first one is shown in the beginning during the setup and the second if the uses chooses to install a new project. Use the first notice for general usage notes and post installation steps and the second as a guide for setting up completely new projects.

You can also specify files to import. During `riptide setup` the user will be asked if they want to import the file or directory. When they choose to do it, Riptide will copy the files and directories inside the project.

Example:

```
import:
  example:
    target: "readme.txt"
    name: Readme file
```

If you run `riptide setup --force` you can run the setup wizard for your project again.

You will see the notice, and if you choose to setup an existing project, you can specify a "Readme file" to import to `readme.txt`. Try it out and you will see, that Riptide copies the directory or file you specify to `readme.txt` inside your project.

PHP Hello World

This section will guide you through the setup of a simple PHP project using the Riptide repository.

We will use an Apache web server, the other guide (PHP with Database, Redis and Composer) shows how to use an Nginx server.

This guide assumes you have Riptide fully set up, with shell integration enabled and a running proxy server (for this guide we assume `https://riptide.local` as base URL of your proxy server). It also assumes you have the `repos` part of the configuration set to only the Riptide Community Repository (the default).

PHP and Apache do NOT need to be installed for this guide.

Preparing the project

For this guide we will set up a very simple PHP file.

Create a new directory and create an `index.php` in it with the following contents:

```
<?php echo "Hello World!"; ?>
```

Creating a basic riptide.yml

Create a `riptide.yml` with the following contents:

```
project:
  name: php-helloworld
  src: .
  app:
    name: php-helloworld
    services:
      php:
        $ref: /service/php/7.2/apache
        roles:
          - src
          - main
```

This file contains one project named `php-helloworld`. We specify with `src` that the source code for this project is in the same directory that the `riptide.yml` is in.

This project contains an app called `php-helloworld`. This app has one service called `php`. This service is the container specification for our Hello World app.

The service `php` needs Apache and PHP so we tell it to load `/service/php/7.2/apache` from the Riptide repository. You can find more details and the YAML file for this on Github.

The final step is adding `roles`. Roles define the behaviour of services.

The `src` role gives our service access to the source code (the `index.php` file). The `main` role sets the service as the main service for our project.

Running the project setup

Run `riptide setup --skip` to initiate the project. Since we have not added any setup instructions or files to import, we just skip the setup with the `--skip` flag.

Starting the project

Open the front page of the Proxy server (`https://riptide.local`). You will find a new project called `php-helloworld`.

Click on the link and the project will start. After it starts you will see the "Hello World!" message telling you, that the project works.

Enable logging

If you want to enable additional logging, add the following lines to the service `php`:

```
logging:
  stdout: true
  stderr: true
```

You can restart the project by using `riptide restart`. After the restart you will find logging files in `_riptide/logs/php`. Apache error log is in `stderr.log` and Apache access log in `stdout.log`.

Adding files for import and setup instructions

For our simple example there are no files to import and we don't really need any setup instructions.

However the `riptide setup` command supports usage notes and importing files, as you can see in the User Documentation. You can also see an example project there.

To add usage notes, add the following to the `app`:

```
notices:
  usage: >-
    This is a demo usage note.

    You can also use variables here: {{ services.php.image }}

  installation: >-
    This will be shown when the user chooses to set up a new project.
```

The user (and you) can view those notes by calling `riptide notes`. They are also shown during `riptide setup`. The first one is shown in the beginning during the setup and the second if the uses chooses to install a new project. Use the first notice for general usage notes and post installation steps and the second as a guide for setting up completely new projects.

You can also specify files to import. During `riptide setup` the user will be asked if they want to import the file or directory. When they choose to do it, Riptide will copy the files and directories inside the project.

Example:

```
import:
  example:
    target: "readme.txt"
    name: Readme file
```

If you run `riptide setup --force` you can run the setup wizard for your project again.

You will see the notice, and if you choose to setup an existing project, you can specify a "Readme file" to import to `readme.txt`. Try it out and you will see, that Riptide copies the directory or file you specify to `readme.txt` inside your project.

PHP with Database, Redis and Composer

This section will guide you through the setup of a complex PHP project using the Riptide repository.

We will use a Nginx web server and PHP-FPM.

This guide assumes you have Riptide fully set up, with shell integration enabled and a running proxy server (for this guide we assume `https://riptide.local` as base URL of your proxy server). It also assumes you have the `repos` part of the configuration set to only the Riptide Community Repository (the default).

PHP and Nginx do NOT need to be installed for this guide.

Preparing the project

For this guide we will set up a PHP file.

Create a new directory and create an `index.php` in it with the following contents:

```
<?php
require_once "vendor/autoload.php";

$hello = new Rivsen\Demo\Hello();
echo $hello->hello();
```

This PHP file tries to load the autoloader supplied by Composer and then tries to load a class from the rivsen/hello-world package.

We will also be setting up Redis and a MySQL database, however our simple PHP example will not use them. You can experiment with your own PHP code to access them, this guide will give you everything you need for this (aside from PHP knowledge).

We also need a `composer.json` with the requirement for this package:

```
{
    "name": "phpcomplex-helloworld",
    "require": {
        "rivsen/hello-world": "*"
    }
}
```

Instead you can also run `composer require rivsen/hello-world` later on, after we added the `composer` command.

Creating a riptide.yml with nginx and php-fpm

Create a `riptide.yml` with the following contents:

```
project:
  name: phpcomplex-helloworld
  src: .
  app:
    name: phpcomplex-helloworld
    services:
      nginx:
        $ref: /service/nginx/latest
        roles:
          - src
          - main
        config:
          default_nginx_conf:
            from: default_nginx.conf
            to: '/etc/nginx/conf.d/default.conf'
        pre_start:
          # Wait for php (otherwise nginx crashes) :(
          - "until ping -c5 php &>/dev/null; do :; done"
      php:
        $ref: /service/php/7.2/fpm
        roles:
```

(continues on next page)

(continued from previous page)

```
        - src
        - php
```

The PHP service is nearly the same as in the `riptide.yml` of the simple example.

We just added the role `php` and switched the reference to the `fpm` variant. This variant does not container Apache but instead PHP and PHP-FPM. The role `php` is used, so that we can later use the `/command/php/from-service` command from the repository.

We added the new service nginx. This service is also based on a service from the repository and also get's access to the source code.

In `config` we tell Riptide to take the `default_nginx.conf` and put it to `/etc/nginx/conf.d/default.conf` in the container.

The `default_nginx.conf` contains the server settings for Nginx. We connect PHP and Nginx there. This is the contents of this file:

```
server {
    listen 80;
    root /src;
    index index.php;
    server_name {{ domain() }};
    location ~* \.php$ {
        fastcgi_index   index.php;
        fastcgi_pass    php:9000;
        include         fastcgi_params;
        fastcgi_param   SCRIPT_FILENAME    $document_root$fastcgi_script_name;
        fastcgi_param   SCRIPT_NAME        $fastcgi_script_name;
    }
}
```

As you can see, we tell Nginx to use the service `php` as a FastCGI backend for all php files. The service `php` contains php-fpm and nginx will communicate with it to process php files.

Since this is a `config` file, variables and variable helper functions can be used in this file. In this case we use the *`domain()`* helper. Riptide will process the file, look for all template strings (`{{ something }}`) and replace them. The helper *`domain()`* returns the domain of the proxy server that our project is accessible under. So when the service is started this line will actually say something like `server_name phpcomplex-helloworld.riptide.local;`.

In `pre_start` for `nginx` we make sure that `nginx` doesn't get started before `php` does, because otherwise nginx would crash.

Adding commands for Composer

Next we need to add the `php` and `composer` commands to our project, so that we can run `composer` to install express from the `composer.json`.

Add the following under `app` in the `riptide.yml`:

```
commands:
  php:
    $ref: /command/php/from-service
  composer:
    $ref: /command/composer/with-host-links
```

This adds two new commands, one containing PHP and one containing PHP and the newest Composer version. All composer processes started will also have access to the directory `.composer` in your home directory and `.ssh`.

Those commands come from the Riptide repository, if you want to know how they work, visit the repository:

- /command/php/from-service
- /command/composer/with-host-links

Installing requirements

If you have the shell integration enabled, leave and enter the directory again, this will load the configured `php` and `composer` commands. You can now run `composer install`, which will install the dependencies and create a directory named `vendor`.

Running the project setup

Run `riptide setup --skip` to initiate the project. Since we have not added any setup instructions or files to import, we just skip the setup with the `--skip` flag.

Starting the project

Open the front page of the Proxy server (`https://riptide.local`). You will find a new project called `php-helloworld`.

Click on the link and the project will start. After it starts you will see the "Hello World!" message telling you, that the project works.

Adding Redis

To add redis, add a new service under `services`:

```
redis:
  $ref: /service/redis/4.0
```

You can start this service using the Riptide CLI:

```
$ riptide start
Starting services...

nginx: 2/2|| Already started!
php  : 2/2|| Already started!
redis: 4/6|                    | Checking...
```

Try to write PHP code to access Redis! Since the service is named `redis`, you will be able to access Redis under the hostname `redis`.

Adding MySQL

To add a MySQL database, add a new service under `services`:

```
db:
  $ref: /service/mysql/5.6
  driver:
    name: mysql
    config:
      database: db
      password: password
```

You can specify the database and password. Username is always `root`.

This is using the MySQL service from the repository and the MySQL database driver. The database driver enables the database management features of Riptide.

Database driver are separate packages that need to be installed. The package for MySQL can be installed via `pip install riptide-db-mysql` (Github).

When you start the database via `riptide start` you can access it.

Try to write PHP code to access the database! Since the service is named `db`, you will be able to access MySQL under the hostname `db`.

The database driver also provides a way to directly access the database. When you enter `riptide status` you can see the port on which you can access the database from the host system.

Enable logging

See the simple example.

Adding files for import and setup instructions

See the simple example.

Sphinx Documentations

This section will guide you through the setup of a sphinx project using the Riptide repository.

Sphinx is a documentation generation tool. This documentation uses Sphinx, so we will use this documentation as an example.

The project we set up uses sphinx-autobuild as a file-watcher and HTTP server for the documentation.

This guide assumes you have Riptide fully set up, with shell integration enabled and a running proxy server (for this guide we assume `https://riptide.local` as base URL of your proxy server). It also assumes you have the `repos` part of the configuration set to only the Riptide Community Repository (the default).

Sphinx and Python do NOT need to be installed for this guide.

Preparing the project

For this guide we will use this documentation as an example.

Clone `https://github.com/Parakoopa/riptide-docs.git` somewhere via Git, or download the repository via Github.

Delete the `riptide.yml` file (we don't want Spoilers :)).

Creating a basic riptide.yml

Create a `riptide.yml` with the following contents:

```
project:
  name: riptidedocs
  src: .
  app:
    $ref: /app/sphinx/latest
    services:
      sphinx:
        environment:
          REQUIREMENTS_FILE: "requirements_docs.txt"
          SPHINX_SOURCE: source
          SPHINX_BUILD: build
```

This file contains one project named `riptidedocs`. We specify with `src` that the source code for this project is in the same directory that the `riptide.yml` is in.

This project contains an app. We load this app from the repository (`/app/sphinx/latest`, *Github <https://github.com/Parakoopa/riptide-repo/tree/master/app/sphinx>*) This app has one service called `sphinx`. This service is the container specification for our Hello World app.

The service `sphinx` is already specified in the app we loaded from the repository.

However we need to set some important environment variables. `SPHINX_SOURCE` and `SPHINX_BUILD` contain the paths to the source and build directories. With `REQUIREMENTS_FILE` an additional file can be specified that contains requirements that will be installed on start. In our case, Riptide is installed before the documentation server starts.

Running the project setup

Run `riptide setup --skip` to initiate the project. Since we have not added any setup instructions or files to import (and `/app/sphinx/latest` also doesn't define any), we just skip the setup with the `--skip` flag.

Starting the project

Since the project's dependencies (express) are now installed, you can open the front page of the Proxy server (`https://riptide.local`). You will find a new project called `riptidedocs`.

Click on the link and the project will start. After the start you may get a Gateway Timeout. Wait a while and refresh and you should see this documentation.

Commands

`/app/sphinx/latest` also defines commands. You can list them with `riptide cmd`.

Magento

This section will guide you through the setup of a Magento 2 project using the Riptide repository.

Magento is an eCommerce platform written in PHP. Riptide has templates for both Magento 1 and Magento 2 projects. This guide will only describe the Magento 2 app. In the Repository Documentation you can find more information about the Magento 1 and Magento 2 templates.

This guide is an example on how to use Riptide with more complex applications and a guide for setting up Magento 2.

This guide assumes you have Riptide fully set up, with shell integration enabled and a running proxy server (for this guide we assume `https://riptide.local` as base URL of your proxy server). It also assumes you have the `repos` part of the configuration set to only the Riptide Community Repository (the default).

PHP or Magento do NOT need to be installed for this guide.

Creating a project

To create a project, create a new file named `riptide.yml` with the following contents:

```
project:
  name: magento-demo
  src: src
  app:
    $ref: /app/magento2/ce/2.3
```

This file defines a project named `magento-demo`. The project uses the app `/app/magento2/ce/2.3` from the Riptide repository (Github).

We choose the `src` directory to install Magento in.

The Magento 2 app comes with PHP-FPM, Nginx, Varnish, MySQL, Redis, RabbitMQ and Mailhog as mail catcher. The default configuration should be suited for most needs, but the next steps of this guide will also show you, how to customize your Magento 2 installation.

You can change the version (`2.3`) if you want to install another Magento version.

The default database name is `magento2`, the user is `root` and the password `magento2`. If you want to change these settings, you have to change the database driver configuration for the `db` service like so:

```
project:
  name: magento-demo
  src: src
  app:
    $ref: /app/magento2/ce/2.3
    services:
      db:
        driver:
          config:
            password: demo
            database: demo
```

Project setup

To get started, run the project setup (`riptide setup`). You will be asked, if you want to start with an existing option or install a new Magento 2 installation.

For this guide we will choose to install a new Magento 2 shop. If you choose to install an existing shop, Riptide will ask you to import a MySQL dump and media files (`pub/media`).

Magento installation

After the project setup, when selecting to install a new project, the following message will be displayed to you:

```
> NEW PROJECT
Okay! Riptide can't guide you through the installation automatically.
Please read these notes on how to run a first-time-installation for magento2-ce-2.3.

Installation instructions:
     To install Magento run the following commands on the command line:

      # 0. Download the Magento source code (replace with 'enterprise-edition' if you␣
↪want):
      mkdir -p <project_directory_root>/src
      cd <project_directory_root>/src
      riptide cmd composer create-project --repository=https://repo.magento.com/ --
↪ignore-platform-reqs magento/project-community-edition ./

      # 1. Dump the autoloader
      cd ./
      riptide cmd composer dump-autoload

      # 2. Start the database and redis
      riptide start -s redis,db

      # 3. Install Magento using the CLI.
      riptide cmd magento setup:install \
        --base-url=https://magento-demo.riptide.local/ \
        --db-host=db \
        --db-name=demo \
        --db-user=root  \
        --db-password=demo \
        --admin-firstname=Admin \
        --admin-lastname=Admin \
        --admin-email=email@yourcompany.com \
        --admin-user=admin \
        --admin-password=admin123 \
        --language=en_US \
        --currency=USD \
        --timezone=America/Chicago \
        --use-rewrites=1

      # 3. (Optional) install sample data
      riptide cmd magento sampledata:deploy

      # 4. Run setup:upgrade
      riptide restart -s redis
      riptide cmd magento setup:upgrade

    You can change the settings in step 3 to your likings, see the installation guide␣
↪at
      https://devdocs.magento.com/guides/v2.3/install-gde/install/cli/install-cli.html
```

These instructions may vary for you. Follow the instructions shown to you to set up your shop.

If shell integration is correctly set up you can also omit the prefix `riptide cmd` from commands. You may need to close and reopen your terminal once for this to work.

If you get started with an existing project, follow the project setup for existing projects. Please note that you need to follow the instructions shown at the beginning of the setup wizard then, You can also show the instructions again by running `riptide setup`. This will show both the instructions for new installations (on the top) and for existing projects ("General Usage notice") on the bottom.

Starting Magento

After you installed Magento, go to the front page of your proxy server (eg. `https://riptide.local`). You will find the various services for the Magento 2 shop there:

The service named varnish is the main entrypoint for your shop. Click on the link. This will open your installed Magento 2 shop.

Mailhog

The Magento app comes with a mail catcher (Mailhog). You can find it by accessing the link of the `mail` service on the proxy server front page. This mail catcher will collect all emails sent by the Magento shop.

Commands (magento, composer, n98-magerun2)

The Magento app comes with a variety of commands for you to use. You can list them with `riptide cmd`:

```
Commands:
    - php
    - magerun
    - n98-magerun (alias for magerun)
    - n98-magerun2 (alias for magerun)
    - magerun2 (alias for magerun)
    - mysql
    - magento
    - composer
```

You have access to the PHP interpreter used for the shop (`php`). The `mysql` command gives you direct access to the database (see below). `magento` is the `bin/magento` command. You can NOT access `bin/magento` directly. Instead use the `magento` command provided by Riptide:

```
$ magento cache:flush
Flushed cache types:
config
layout
block_html
collections
reflection
db_ddl
compiled_config
eav
customer_notification
config_integration
config_integration_api
```

(continues on next page)

(continued from previous page)

```
full_page
config_webservice
translate
vertex
```

In addition to those commands, you also have access to composer (`composer`) and n98-magerun2 by Netz98.

Additionally you can open a console to the containers for the services by using `riptide exec <service_name`, eg. `riptide exec php`. You can also open a root console by passing the flag `--root`.

Accessing the database

To access the database, you have to start it first (either via the Proxy server or by running `riptide start`.

You can access the database directly simply by executing `mysql`. Additionally you can access the database using your favorite SQL client. To get the port you can access the database from, see this section of the User Documentation.

Adding own services and commands

If you want to add your own services and commands, simply add new entries under `services` or `commands` in the project file:

```yaml
project:
  name: magento-demo
  src: src
  app:
    $ref: /app/magento2/ce/2.3
    services:
      db:
        driver:
          config:
            password: demo
            database: demo
      styleguide:
        image: node:8
        roles:
          - src
        working_directory: styleguide
        command: node_modules/.bin/gulp serve
        port: 3000
        pre_start:
          - npm install
          - node_modules/.bin/gulp clean
          - node_modules/.bin/gulp build
    commands:
      node:
        $ref: /command/node/8
      npm:
        $ref: /command/npm/node8
```

Configuration management

The Magento Riptide app comes with support for the configuration management tool mageconfigsync. If installed the file *app/etc/config.yml* with the environment *dev* is loaded into the database on each start of the project.

If you want to run your own configuration management tools, add the appropriate commands to the *post_start* step of the php service.

3.3 Riptide Community Repository

This is the documentation for all entities in the Riptide Community Repository.

This repository is the default repository for Riptide. You can use the apps, services and commands in your own projects.

More information about entities and repositories, and guide's on how to use the entities from this repository, can be found in the Configuration Guide.

Is something missing from the repository? Feel free to fork the repository and add the things you need. Or set up your own and use both. If you built something, please contribute to the community repository to make it even better!

3.3.1 Apps

Angular

Angular, platform for building mobile and desktop web applications.

Minimal project configuration for a basic Angular app.

Run `riptide cmd npm install` before trying to start the project.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/app/angular

Index

`/app/angular/base`

Angular base variant, based on Node 12.

Services

www

Accessible via Proxy?: yes

Runs as the user using Riptide?: yes

The `ng serve` Angular webpack server.

Roles

Has roles: `src`, `main`

Has access to source code (`src`). This is the `main` service.

Commands

node

Based on: /command/node/12

NodeJS, version 12.

npm

Based on: /command/npm/node12

NPM, latest version using Node 12.

yarn

Based on: /command/yarn/node12

Yarn package manager, latest version using Node 12.

ng

Based on: /command/node/12

Angular CLI. Runs the `ng` command of the `node_modules` directory.

Grav

Grav Web-platform.

Web server is based on Apache. The Riptide app comes with a mail catcher.

Comes with a default user (name: `riptide`, password `12345`).

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/app/grav

Index

`/app/grav/base`

Grav base variant.

Imports

Key	Title	Target	Description
images	Images	images	Image files
pages	CMS Pages	user/pages	CMS pages
plugins	Plugin configuration	user/plugins	Configuration and code for some plugins

Services

php

Based on: /service/php/7.2/apache

PHP Version 7.2 with the Apache 2 webserver.

Roles

Has roles: `src`, `php`, `main`

Has access to source code (`src`) and is marked as main PHP service (`php`). This is the `main` service.

Config

If you want to change additional Magento settings, we recommend adding additional `bin/magento config:set` to `post_start` or using a module for configuration management.

Name	Target	Should be replaced?	Description
user_config	user/config/system.yaml	maybe (if your page requires custom configuration)	System configuration file.
security_config	user/config/security.yaml	maybe	Default security configuration (salt).
riptide_account	user/accounts/riptide.yaml	no	Default system user (riptide). Password is 12345.

mail

Based on: /service/mailhog/latest

Mailhog, used as mail catcher.

Roles

Has roles: `mail`

Role required for PHP service.

Commands

php

Based on: /command/php/from-service

PHP command.

grav

`bin/grav` command.

Runs in the `php` service.

composer

Based on: /command/composer/with-host-links

Composer package manager.

npm

Based on: /command/npm/node12

NPM JavaScript package manager. Might be useful for frontend building.

node

Based on: /command/node/12

NodeJS. Might be useful for frontend building.

Magento 1

Magento eCommerce platform, version 1.

The Riptide app comes with Redis and a mail catcher.

Web server is based on Apache.

Uses mageconfigsync for configuration management, if installed. If you want to use mageconfigsync with Riptide create a file `app/etc/config.yml` with an environment `dev`.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/app/magento1

Index

`/app/magento1/base`

Magento 1 base variant.

Imports

Key	Title	Target	Description
media_files	Media Files	media	Media files, such as product images

Services

www

Based on: /service/php/7.2/apache

Apache and PHP 7.2.

Roles

Has roles: `src`, `php`, `main`

Has access to source code (`src`) and is marked as main PHP service (`php`).

Config

If you want to change additional Magento settings, we recommend using a module for configuration management.

Name	Target	Should be replaced?	Description
local_xml	app/etc/local.xml	no	Magento 1 local.xml, contains all database and base url settings, etc. pp.

Post Start

Waits for `magerun db:info` to work (= db to start up).

Runs mageconfigsync to load configuration from the `dev` environment from the file `app/etc/config.yml`. If mageconfigsync is not installed this step silently fails.

Clears cache.

db

Based on: /service/mysql/5.6

MySQL 5.6 database.

Driver

Configuration:

User: root

Password: magento1

Database: magento1

redis

Based on: /service/redis/latest

Redis, used for Cache and Session.

rabbitmq

Based on: /service/rabbitmq/3.6

RabbitMQ, may be used as message broker.

mail

Based on: /service/mailhog/latest

Mailhog, used as mail catcher.

Roles

Has roles: `mail`

Role required for PHP service.

Commands

php

Based on: /command/php/from-service

PHP command.

magerun, n98-magerun

n98-magerun by Netz98 for Magento development.

Additional volumes

Name	Source	Source path	Target path	Description
local_xml	Config from another service	(config 'local_xml' from service 'php')	app/etc/local.xml	local.xml for Magento
config	Home Directory	~/.n98-magerun	~/.n98-magerun (ro)	Magerun configuration

composer

Based on: /command/composer/with-host-links

Composer package manager.

mysql

Based on: /command/mysql/from-service-db

MySQL client that load's the configuration from the service with role `db`.

The client auto-connects to the database from this service.

`/app/magento1/ce/1.9`

Based on: /app/magento1/base

Configuration for different versions of Magento Open Source, version 1.

/app/magento1/ee/1.14

Based on: /app/magento1/base

Configuration for different versions of Magento Commerce, version 1.

Magento 2

Magento eCommerce platform, version 2.

The Riptide app comes with Varnish, Redis, RabbitMQ and a mail catcher.

Web server is based on Nginx. The "Apache" variants contain web servers based on Apache.

Uses mageconfigsync for configuration management, if installed. If you want to use mageconfigsync with Riptide create a file `app/etc/config.yml` with an environment `dev`.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/app/magento2

Index

/app/magento2/base

Magento 2 base variant, using Nginx.

Imports

Key	Title	Target	Description
media_files	Media Files	pub/media	Media files, such as product images

Services

php

Based on: /service/php/7.2/fpm

PHP-FPM Version 7.2

Roles

Has roles: `src`, `php`

Has access to source code (`src`) and is marked as main PHP service (`php`).

Config

If you want to change additional Magento settings, we recommend adding additional `bin/magento config:set` to `post_start` or using a module for configuration management.

Name	Target	Should be replaced?	Description
env_php	app/etc/env.php	no	Magento 2 env.php, contains all database and redis settings, etc. pp.

Post Start

Waits for `bin/magento` to work (= redis and db to start up).

Changes settings, such as the base url.

Runs mageconfigsync to load configuration from the `dev` environment from the file `app/etc/config.yml`. If mageconfigsync is not installed this step silently fails.

Clears cache.

www

Based on: /service/nginx/latest

Nginx, linked with the PHP service.

Reads the `nginx.conf.sample` provided by Magento for additional server configuration.

Roles

Has roles: `src`, `varnish`

Has access to source code (`src`) and is marked as backend server for Varnish (`varnish`).

Config

Name	Target	Should be replaced?	Description
env_php	app/etc/env.php	no	Magento 2 env.php, contains all database and redis settings, etc. pp.
magento_nginx_conf	/etc/nginx/conf.d/default.conf	no	Nginx server settings

Pre Start

Waits for `php` to start. Would crash otherwise.

varnish

Based on: /service/varnish/4

Varnish cache server. Uses `www` as backend server.

Roles

Has roles: `main`

Config

Key	Target	Should be replaced?	Description
vcl	/etc/varnish/default.vcl	maybe	Magento 2 default VCL

db

Based on: /service/mysql/5.6

MySQL 5.6 database.

Driver

Configuration:

User: root

Password: magento2

Database: magento2

redis

Based on: /service/redis/latest

Redis, used for Cache and Session.

rabbitmq

Based on: /service/rabbitmq/3.6

RabbitMQ, may be used as message broker.

mail

Based on: /service/mailhog/latest

Mailhog, used as mail catcher.

Roles

Has roles: `mail`

Role required for PHP service.

Commands

php

Based on: /command/php/from-service

PHP command.

magerun, magerun2, n98-magerun, n98-magerun2

n98-magerun2 by Netz98 for Magento development.

Additional volumes

Name	Source	Source path	Target path	Description
env_php	Config from another service	(config 'env_php' from service 'php')	app/etc/env.php	env.php for Magento
config	Home Directory	~/.n98-magerun2	~/.n98-magerun2 (ro)	Magerun2 configuration

magento

`bin/magento` command. Not included in image, read from working directory instead.

Additional volumes

Name	Source	Source path	Target path	Description
env_php	Config from another service	(config 'env_php' from service 'php')	app/etc/env.php	env.php for Magento

composer

Based on: /command/composer/with-host-links

Composer package manager.

mysql

Based on: /command/mysql/from-service-db

MySQL client that load's the configuration from the service with role `db`.

The client auto-connects to the database from this service.

`/app/magento2/apache`

Based on: /app/magento2/base

Variant of Magento using the Apache web-server instead.

Services

php

Based on: /service/php/7.2/apache

Apache web server + PHP.

Roles

Has roles: `src`, `php`, `varnish`

Has access to source code (`src`), is marked as main PHP service (`php`) and is marked as backend server for Varnish (`varnish`).

www

Is removed.

The apache web-server with a PHP CGI module is in the "php" service.

/app/magento2/ce/X

Based on: /app/magento2/base

Configuration for different versions of Magento Open Source, version 2. Using Nginx.

Available versions:

- 2.3

/app/magento2/ee/X

Based on: /app/magento2/base

Configuration for different versions of Magento Commerce, version 2. Using Nginx.

Available versions:

- 2.3

/app/magento2/ce/X-apache

Based on: /app/magento2/apache

Configuration for different versions of Magento Open Source, version 2. Using Apache.

Available versions:

- 2.3

/app/magento2/ee/X-apache

Based on: /app/magento2/apache

Configuration for different versions of Magento Commerce, version 2. Using Apache.

Available versions:

- 2.3

Sphinx

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/app/sphinx

Index

`/app/sphinx/latest`

Latest version of Sphinx and sphinx-autobuild.

sphinx-autobuild is a file watcher and HTTP server for Sphinx.

Change the `SPHINX_SOURCE` and `SPHINX_BULD` of the `sphinx` service.

Please note that, after you started this project, it takes a while to actually finish starting. In the meantime you will get a Gateway Timeout.

Services

sphinx

Accessible via Proxy?: yes

Runs as the user using Riptide?: yes

sphinx-autobuild server.

Uses the riptidepy/sphinx image, based on Python 3.7.

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
SPHINX_SOURCE	yes	yes (default: "source")	source	Directory that contains the conf.py
SPHINX_BUILD	yes	yes (default: "build")	build	Build output directory
REQUIREMENTS_FILE	no	no	requirements.txt	This file is read on startup and the dependencies in it are installed first.

Additional volumes

Name	Source	Source path	Target path	Description
py_packages	Data folder	_riptide/data/___/site_packages	~/.local/lib/python3.7/site-packages	Installed Python packages (see REQUIREMENTS_FILE)

Commands

make, sphinx-build, sphinx-autogen, sphinx-apidoc

make, sphinx-build, sphinx-autogen and sphinx-apidoc commands.

Use image of `sphinx` service.

Additional volumes

See `sphinx` service.

sphinx-doctest

Runs the sphinx doctest command:

```
python -msphinx -b doctest {{ parent().services.sphinx.get_working_directory() }}/{{␣
↪parent().services.sphinx.environment.SPHINX_SOURCE }}
```

Uses image of `sphinx` service.

Additional volumes

See `sphinx` service.

3.3.2 Services

Elasticsearch

Elasticsearch is a search engine based on Lucene.

When using this service please make sure to override the image to match the exact Elasticsearch version you want to use.

The template provided is tested for Elasticsearch 6.

Requires Riptide >= 0.2.7

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/elasticsearch

Index

`/service/elasticsearch/latest`

Runs as the user using Riptide?: yes

Accessible via Proxy?: no

Latest version of Elasticsearch.

Mailhog

Mailhog is a mail catching application that can be used to test SMTP functionality of applications.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/mailhog

Index

/service/mailhog/latest

Runs as the user using Riptide?: yes

Accessible via Proxy?: yes

Latest version of Mailhog.

MySQL

MySQL relational database management system.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/mysql

Index

/service/mysql/base

Accessible via Proxy?: no

Runs as the user using Riptide?: no

Latest version of MySQL. Configure database name and password via the driver settings.

Suggested Roles

Has roles: `db`

This service is a database and has the role db set. See the `driver` section for more details.

Additional volumes

Name	Source	Source path	Target path	Description
n/a	Data folder	_riptide/data/___/___	/var/lib/mysql	Database data, managed by Riptide's database management tools

Additional ports

Name	Title	Container Port	Host Start Port	Description
mysql	MySQL Port	3306	3306	MySQL Port

Post Start

Waits for the database to finish start-up.

Driver

Database driver: `mysql <https://github.com/Parakoopa/riptide-db-mysql>`

Configuration:

User: root

Password: mysql

Database: mysql

`/service/mysql/5.4`, `/service/mysql/5.5`, `/service/mysql/5.6`, `/service/mysql/5.7`, `/service/mysql/8.0`, `/service/mysql/9.0`

Based on: /service/mysql/base

Additional versions of MySQL. If you need other versions, use the `base` version and change the image tag.

nginx

nginx web server.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/nginx

Index

`/service/nginx/latest`

Accessible via Proxy?: yes

Runs as the user using Riptide?: no

Latest version of nginx. Make sure to override the file `/etc/nginx/conf.d/default.conf` with a `server` configuration.

Config

Name	Target	Should be replaced?	Description
nginx_conf	/etc/nginx/nginx.conf	no, add files to conf.d instead	Main Nginx configuration
(Not provided!)	/etc/nginx/conf.d/default.conf	yes	Nginx server configuration

php

PHP scripting language. Uses the riptidepy/php images.

There are variants for PHP 7.1 - 7.4. Some include the Apache web server (`apache` variants), others include php-fpm (`php-fpm` variants) and some only the interpreter (`cli` variants).

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/php

Index

/service/php/base

Accessible via Proxy?: no

Runs as the user using Riptide?: yes

Base of all php variants.

Role Requirements

Role: `mail`

The service to use as SMTP server (eg. /service/mailhog). If you don't have a SMTP service, add the role to this service instead. You will not be able to send emails then.

Suggested Roles

Suggested roles: `src`, `php`

This service should have access to the source code of the application via the role `src`.

If this is your main PHP service, add the role `php`. You can then use the /command/php/from-service template for commands.

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
XDEBUG_CONFIG	no	yes, (default: "remote_host={{ host_address() }}")	remote_host={{ host_address() }	Configuration for Xdebug
PHP_IDE_CONFIG	no	yes, (default: "serverName=riptide-{{ parent().parent().name }}")	serverName=riptide-{{ parent().parent().name }}	PhpStorm path mapping key

Config

Key	Target	Should be replaced?	Description
php_ini	/etc/php.d/z_riptide.ini	no, add own files to the php.d directory	Disables opcache
msmtprc	/etc/msmtprc	no	SMTP configuration, see "Role Requirements"

`/service/php/base-apache`

Based on: /service/php/base

Runs as the user using Riptide?: yes, via environment variables `APACHE_RUN_USER` and `APACHE_RUN_GROUP`

Accessible via Proxy?: yes

Variant that contains the Apache web server and integrates the PHP CGI module.

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
APACHE_RUN_USER	yes	yes, (default: "#{{ os_user() }}")	#{{ os_user() }}, www-data, #1000	User to run Apache as
APACHE_RUN_GROUP	yes	yes, (default: "#{{ os_group() }}")	#{{ os_group() }}, www-data, #1000	Group to run Apache as

`/service/php/7.1/apache`, `/service/php/7.2/apache`, `/service/php/7.3/apache`, `/service/php/7.4/apache`

Based on: /service/php/base-apache

Variant that contains the Apache web server and integrates the PHP CGI module. PHP 7.1 - 7.3.

/service/php/7.1/cli, /service/php/7.2/cli, /service/php/7.3/cli, /service/php/7.4/cli

Based on: /service/php/base

Variant that only contains the PHP interpreter. PHP 7.1 - 7.3.

/service/php/7.1/fpm, /service/php/7.2/fpm, /service/php/7.3/fpm, /service/php/7.4/fpm

Based on: /service/php/base

Variant that contains PHP-FPM. PHP 7.1 - 7.3.

RabbitMQ

RabbitMQ message broker.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/rabbitmq

Index

/service/rabbitmq/base

Accessible via Proxy?: no

Runs as the user using Riptide?: no

Latest version of RabbitMQ.

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
RABBITMQ_NODENAME	yes	yes, (default: "rabbit@localhost")	rabbit@localhost	
RABBITMQ_DEFAULT_USER	yes	yes, (default: "rabbit")	rabbit	
RABBITMQ_DEFAULT_PASS	yes	yes, (default: "rabbit")	rabbit	
RABBITMQ_USE_LONGNAME	no	yes, (default: "true")	true, false	

Additional volumes

Name	Source	Source path	Target path	Description
rabbitmq	Data folder	_riptide/data/___/rabbitmq	/var/lib/rabbitmq	RabbitMQ Data

`/service/rabbitmq/3.6`

Based on: /service/rabbitmq/base

Version 3.6 of RabbitMQ.

Redis

Redis is an open source, in-memory data structure store, used as a database, cache and message broker.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/redis

Index

`/service/redis/base`

Accessible via Proxy?: no

Runs as the user using Riptide?: no

Latest version of Redis.

`/service/redis/4.0`

Based on: /service/redis/base

Version 4.0 of Redis.

Varnish

Varnish Cache is an HTTP caching server.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/service/varnish

Index

`/service/varnish/4`

Accessible via Proxy?: yes

Runs as the user using Riptide?: no

Varnish version 4. Make sure to replace the default VCL.

Role Requirements

Role: `varnish`

Use this in your VCL as backend server:

```
backend default {
    .host = "{{ parent().get_service_by_role('varnish')['$name'] }}";
}
```

Your `varnish` target should have an HTTP server running on port 80.

Suggested Roles

Suggested roles: `main`

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
VCL_CONFIG	yes	yes (default: "/etc/varnish/default.vcl")	/etc/varnish/default.vcl	Path to the VCL, should NOT be changed

Config

Key	Target	Should be replaced?	Description
vcl	/etc/varnish/default.vcl	yes	VCL configuration for Varnish

Logging

Name	Type	Path / Command	Description
varnish.log	Command	varnishlog	varnishlog

Pre Start

Wait's for the backend service server to be reachable (otherwise varnish would crash).

3.3.3 Commands

Composer

Composer PHP package manager.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/composer

Index

`/command/composer/base`

Latest version of Composer. No links to host system (see `with-host-links`).

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
COMPOSER_HOME	no	yes (default: "{{ home_path() }}/.composer")	/home/riptide/.composer	Directory that composer config and cache is stored in

Additional volumes

Name	Source	Source path	Target path	Description
tmp	Other	/tmp ({{ get_tempdir() }})	/tmp	Temporary directory

`/command/composer/with-host-links`

Based on: /command/composer/base

Composer with volumes for the user's `.composer` and `.ssh` directories.

Additional volumes

Name	Source	Source path	Target path	Description
composer	Home directory	~/.composer	~/.composer	.composer
ssh	Home directory	~/.ssh	~/.ssh	SSH configuration

MySQL

MySQL relational database management system.

Client command to be used with /service/mysql.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/mysql

Index

/command/mysql/from-service-db

MySQL client that load's the configuration from the service with role `db`.

The client auto-connects to the database from this service.

Role Requirements

Role: `db`

MySQL service that the configuration will be loaded from.

Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/node

Index

/command/node/base

Latest Node.js version. Change image tag for other versions.

/command/node/X

Based on: /command/node/base

Different Node.jS versions. Available versions:

- 8

- 10
- 11
- 12

Other versions can be used by changing the version of the image.

NPM

NPM Node.js package manager.

This command template can also be used for other Node.js commands (by changing the command), if they require access to the npm cache.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/npm

Index

/command/npm/base

Latest NPM version with the latest Node.js version.

Additional volumes

Name	Source	Source path	Target path	Description
npm	Home directory	~/.npm	~/.npm	NPM cache
npmrc	Home directory	~/.npmrc	~/.npmrc	NPM config
ssh	Home directory	~/.ssh	~/.ssh	SSH configuration

/command/npm/nodeX

Based on: /command/npm/base

Latest NPM with different Node.js versions. Avaiable Node.js versions:

- 8
- 10
- 11
- 12

php

PHP scripting language. Uses the riptidepy/php images.

There are variants for PHP 7.1 - 7.3 and one that uses the configuration of the service with role `php`.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/php

Index

/command/php/base

Latest PHP version.

Environment variables

Key	Required?	Already set?	Example Value(s)	Description
XDEBUG_CONFIG	no	yes, (default: "remote_host={{ host_address() }}")	remote_host={{ host_address() }	Configuration for Xdebug
PHP_IDE_CONFIG	no	yes, (default: "serverName=riptide-{{ parent().parent().name }}")	serverName=riptide-{{ parent().parent().name }}	PhpStorm path mapping key

/command/php/X

Based on: /command/php/base

Different PHP versions. Available versions:

- 7.1
- 7.2
- 7.3

/command/php/from-service

Based on: /command/php/base

Uses the PHP configuration of a service with role `php`. This variant can also be used to send mails.

Role Requirements

Role: `php`

A service based on /service/php.

Additional volumes

These volumes are used from the PHP service's config.

Name	Source	Source path	Target path	Description
php_ini	Config from another service	(config 'php_ini' from service with role 'php')	/etc/php.d/z_riptide.ini	PHP service php settings
msmtprc	Config from another service	(config 'msmtprc' from service with role 'php')	/etc/msmtprc	SMTP configuration

Yarn

Yarn Node.js package manager.

This command template can also be used for other Node.js commands (by changing the command), if they require access to the yarn cache.

Link to entity in repository: https://github.com/Parakoopa/riptide-repo/tree/master/command/yarn

Index

/command/yarn/base

Latest Yarn version with the latest Node.js version.

Additional volumes

Name	Source	Source path	Target path	Description
yarn	Home directory	~/.yarn	~/.yarn	Yarn cache
yarnrc	Home directory	~/.yarnrc	~/.yarnrc	Yarn config
ssh	Home directory	~/.ssh	~/.ssh	SSH configuration

/command/yarn/nodeX

Based on: /command/npm/base

Latest Yarn with different Node.js versions. Avaiable Node.js versions:

- 8
- 10
- 11
- 12

3.4 Plugin Development

Riptide can be extended with plugins. To write a plugin, create a Python package, that has the following entry point defined:

```
[riptide.plugin]
php-xdebug=riptide_plugin_php_xdebug.plugin:PhpXdebugPlugin
```

Replace `php-xdebug` with the identifier of your plugin and the rest with the entry point of your plugin, implementing AbstractPlugin (see below).

3.4.1 Plugin Interface

class riptide.plugin.abstract.**AbstractPlugin**

Bases: abc.ABC

A Riptide plugin extends the functionality of Riptide.

For this it can:

- Add new CLI commands to riptide-cli.
- Set flags, which can be retrieved from the configuration using a variable helper
- Directly read and modify all parts of the configuration entities loaded.
- Communicate with the loaded engine.

after_load_cli(*main_cli_object*)

Called after the last CLI of Riptide CLI has loaded. Can be used to add CLI commands using Click. The passed object is the main CLI command object.

after_load_engine(*engine: riptide.engine.abstract.AbstractEngine*)

After the engine was loaded. `engine` is the interface of the configured engine.

after_reload_config(*config: Config*)

Called whenever a project is loaded or if the initial configuration is loaded without a project.

get_flag_value(*config: Config*, *flag_name: str*) → any

Return the value of a requested plugin flag. Return False if not defined. The current config is passed, to give a context about the calling project. Please note, that flag values are usually loaded before after_reload_config!

3.5 Updates

This section documents breaking changes that were introduced by Riptide updates.

3.5.1 lib 0.2.0

Using configuration files in commands

Previously it was possible to use configuration files defined for services in commands, using the following syntax:

```
app:
  services:
    example:
      config:
        env_php:
          from: assets/env.php
          to: '{{ get_working_directory() }}/app/etc/env.php'
  commands:
    dummy:
      additional_volumes:
        env_php:
          host: "{{ parent().services.example.config('env_php') }}"
          container: "{{ parent().services.example.get_working_directory() }}/app/
↪etc/env.php"
```

This is no longer possible, due to issues related to this functionality. Instead, you can use this new syntax, which will add all configuration files of the services tagged with the listed roles to the command. This achieves the same goal.

```
app:
  services:
    example:
      roles:
        - myrole
      config:
        env_php:
          from: assets/env.php
          to: '{{ get_working_directory() }}/app/etc/env.php'
  commands:
    dummy:
      config_from_roles:
        - myrole
```

The variable helper function `config` for Services has been removed. `config_from_roles` was added to the schema for Commands.

Detailed examples can be seen in the Riptide Community Repository commit ee1a766.

The Riptide Community Repository contains a branch `0.1` that is automatically used when running `riptide update` with versions older than 0.2. This branch is still compatible with the old version, but will not receive updates.

3.5.2 0.5.0

Riptide 0.5.0 introduces Performance optimizations. These include the change, that under Mac and Windows most volumes are now no longer stored on the host file system, but in internal named volumes of the VM instead.

This also affects volumes of projects, which were previously stored under `_riptde/data` in projects, such as MySQL databases.

If you need the data, you need to manually export the data of these volumes before upgrading Riptide and then import it after the upgrade.

Linux is not affected by these changes (unless the performance option is manually enabled). This guide assumes you are using MacOS or Windows.

Exporting and importing databases

To export and import databases, use the `riptide db-export` and `riptide-db-import` commands, see Managing Databases.

Exporting and importing other data

Riptide does not offer export or import functionality for other data.

I already upgraded, what now?

If you already upgraded Riptide to 0.5.0, you can disable the performance option and access the data again to export it. You can then enable it again and import the data.

To disable the performance optimization, run `riptide config-edit-user` and set `performance.dont_sync_named_volumes_with_host` to `false`.

To enable it again after the export, set `performance.dont_sync_named_volumes_with_host` to `auto`

If you unable to migrate, you can also leave the performance option disabled.

- genindex
- modindex
- search

Index

V