# Table of Contents

## Quick Start



## Reference



## Guides



## Examples


*Automate, Automate, Automate!*

The key to optimize work is to *automate* as much as possible. Whether you are developing software, setting up infrastructure or even testing, if there is a chance to do it just take it.

Most of the times the boring and frequent tasks can be automated. There are a lot of different tools and technologies that can help you with that, but sometimes starting is simply an hassle by itself and one might end up giving up and doing it *"manually"*.

This is where **Athena** jumps in. The idea is quite simple, we minimize the start process by :

- automating the wiring of the dependencies and tools
- implement a plugin architecture to allow freedom and scalability
- throw in a wizard that takes care of the boring stuff

Once that part is done, you can reuse it as much as you can and you can even share with other people or teams.

Seems cool, right?

Well, it might also seem too easy or too abstract so let's dig into it.

# How does it work?

**Athena** aims to be of simple usage and architecture and consists of 2 pillars:

- the `Engine` : a declarative framework based in `bash` ( *say that again ???* Don't run away yet because it has proper unit tests. To know more about it, have a look at the bashUnit Testing Framework for `bash` . It has all of the common features an xUnit Framework gives, including Mocking.)
- the `Plugins` : they are typically *Services*, *Applications*, *Jobs*, etc. They can use any technology or any stack of technologies and they form the ecosystem that helps you handle different scenarios. To support the automation environments, the virtualisation technology used is docker.

As an example, you can use **Athena** to setup a webserver, test your website and handle the deployment. This example could be achieved by creating 3 `plugins` : `webserver` , `test` and `deploy` or you could simply create just one : `app` .

Hopefully the ecosystem will continue to grow and become big enough so that most of the scenarios will be already handled and you can use an existing `plugin` , build your own from scratch or even base yours on another one.

# Why did we choose `bash` ?

We wanted to make it simple and having the minimum dependencies possible. Using bash became a natural choice because that's where you usually start when you automate with scripts.

`bash` already has support for a lot of stuff, but the issue is that is not very declarative or at least does not have a very developer-like syntax, and this is why **Athena** was born. A simple, declarative and developer-friendly framework with testing support.

# Why should you use it?

Besides having a very straightforward and simple architecture, which makes it easy to debug, it provides you built-in support for solving the following topics :

- Version validation
- Error handling
- Proper display of messages
- Stacktrace
- Testing
- Routing
- Support for Multiple and configurable environments
- Hooks
- Standardized way of building stuff
- etc...

With this already taken care of you will only need to focus on your specific problem.

# Table of contents



# Quick start

Prerequisites

- You have a `bash` shell.
- You have Git installed.
- You have Docker installed.

There are three quick start options available:

**On Linux**

- Using a `debian package` from the releases :

```
$ sudo dpkg -i <downloaded_debian_package>
```

- Using `apt-get` :

```
$ sudo add-apt-repository ppa:athena-oss/athena
$ sudo apt-get update
$ sudo apt-get install athena
```

**On MAC OSX**

- Using Homebrew :

```
$ brew tap athena-oss/tap
$ brew install athena
```

**Note:** You might be required to allow Docker to access folders managed by Homebrew. In order to do this, go to Docker `Preferences > File Sharing` and add the folder `/usr/local/Cellar` .

**Alternative**

- Download the latest release

- Clone the repo: `git clone https://github.com/athena-oss/athena.git`

Go to the Documentation Website or download the Documentation PDF to find out more about using Athena.

# Examples

We have several examples here. Check these two **Athena** commands, so that you can have an idea how it looks like :

## running a file/directory validator

```
CMD_DESCRIPTION="Validates a file or directory for possible issues."

athena.usage 1 "<file|directory>"

if athena.plugins.base.check "$(athena.path 1)" ; then
  athena.ok "check passed"
  athena.exit 0
fi
athena.fatal "check failed"
```

## running a php webserver

```
CMD_DESCRIPTION="Starts the webserver."

athena.usage 2 "<source_directory> <port>"

# arguments are found below
source_directory="$(athena.path 1)"
port="$(athena.int 2)"

# clearing arguments from the stack
athena.pop_args 2

# options for container are found below
athena.plugin.use_external_container_as_daemon "php:7.0-apache"

# mounts the specified dir into the container
athena.docker.mount_dir "$source_directory" "/var/www/html"

# maps the specified host port to the port 80 of the container
athena.docker.add_option "-p $port:80"
```

# Plugins

Here is a list of some of the available plugins :

- PHP Plugin - Plugin for Test Automation using PHP as a development language.
- Selenium Plugin - Plugin to handle browser automation using Selenium.
- Proxy Plugin - Plugin to handle a proxy server using Browsermob-proxy.
- Appium Plugin - Plugin to handle mobile automation using Appium.
- AVD Plugin - Plugin to manage Android Virtual Devices.
- Gradle Plugin - Plugin for running gradle tasks.

# Contributing

Checkout our guidelines on how to contribute in CONTRIBUTING.md.

# Versioning

Releases are managed using github's release feature. We use Semantic Versioning for all the releases. Every change made to the code base will be referred to in the release notes (except for cleanups and refactorings).

# License

# Getting Started

Prerequisites

- You have a Bash shell.
- You have Git installed.
- You have Docker installed.

There are three quick start options available:

- On MAC OSX using Homebrew :

```
$ brew tap athena-oss/tap
$ brew install athena
```

- Download the latest release
- Clone the repo: `git clone https://github.com/athena-oss/athena.git`

Now that you have Athena you can explore the Examples section.

If you want to start by building your own plugin use the wizard :

```
$ ./athena wizard start [<name_of_the_plugin>]
```

You can also install plugins by using the following command :

```
$ ./athena plugins install <name> <repo>
```

# How the Engine works

The Athena Engine is a simple and declarative Bash framework, implemented using only Bash functions and it takes care of tasks like routing, errors, messages, stacktrace, testing, versioning, validation, etc.

## Structure

The directory structure is very simple as you can see in the following schema :

```
.
├── athena  # executable
├── ...
├── lib # location of the library functions
│   └── ...
├── plugins # location of the plugins
│   └── ...
└── tests   # location of the
    └── ...
```

## Namespaces

Even though it is not very tipical for Bash functions to have namespaces, we defined namespacing to avoid collision of functions. The pattern is

```
athena.<context>.<function_name>
```

The **context** is used to group functions, for instance, all functions related to handling arguments should be grouped under the `athena.argument.` namespace. As an example, to check if an argument exists you can use the following function :

```
athena.argument.argument_exists <argument_name>
```

The context also defines the name of the file where the functions are located. The pattern for the files is :

```
lib/[<shared_or_not>/]functions.<context>.sh
```

## Using the functions

You can use the functions of Athena in 2 contexts, *HOST* or *SHARED*. Using on the *HOST* means that the function is being used directly on the machine that has Athena installed, on the other hand *SHARED* means that you can use it either on the host or inside the container that will provide the environment for your plugin.

- HOST only functions
  - located in the `lib` folder
  - loaded automatically on the host
  - can be used only on the *PRE* and *POST* commands
- SHARED functions
  - located in the `lib/shared` folder
  - loaded automatically on the host and on the container when the default router is being used
  - can only be used on the inner commands

# How the Plugins work

Trends and technologies change fast and most likely your needs too. With this in mind, you can implement plugins using any technology or stack of technologies. `Plugins` can be solutions to handle a different set of scenarios, but more often than not, you may build them as Services, Applications, Jobs/Tasks. These `plugins` in turn have commands that allows you to manage or execute tasks of this context. As an example, your `plugin` can be a simple webserver that has commands to start and stop it.

# Structure

The directory structure is also very simple as you can see in the following schema :

```
plugins/example
├── bin
│   └── cmd          # location of the commands that execute the tasks
│       └── ...
├── dependencies.ini # file that contains the dependencies of this plugin
├── docker           # optional: contains the custom environments
│   └── ...
└── version.txt      # contains the version of the plugin
```

# Commands

Commands are used to execute tasks either on the host or inside the container or even on both. Some commands may require that you execute some tasks on the host before actually running inside the container, for example, if you need to prepare some directory structure on the host machine that will save the result of what was done inside the container.

In order to make this simple and generic enough, we decided to adopt the approach of having a 3-step execution that you can look at it like:

- prepare - runs on the HOST
- execute - runs on the CONTAINER
- cleanup - runs on the HOST

The sequence of the execution follows the order in the previous list. The implementation intends to be very simple on the usage side but also on the architecture side. To support a command you will need to have a file inside the commands directory that must follow the name pattern :

```
<name_of_command>[_<type>].sh
```

As an example, imagine that you want to implement the command `run` and that this command that needs a prepare, execute and cleanup steps, then you are required to have the files `run_pre.sh` , `run.sh` and `run_post.sh` .

The purpose of splitting this into different files is to sepparate the responsabilities and make it easier to maintain.

**Note:** All of these steps are optional, altough it does not make much sense having both a prepare and cleanup steps if no container will be used, because you can achieve the same result using only the prepare step.

## Example: run (prepare step)

```
CMD_DESCRIPTION="Runs a task."

athena.usage 1 "<source_directory>"

# arguments are found below
source_directory="$(athena.path 1)"
```

```
# clearing arguments from the stack
athena.pop_args 1

# do something with the $source_directory
...

# mounting the directory to be used inside the container
athena.docker.mount_dir "$source_directory" "/opt/workdir"
```

### Example: run (execute step)

```
# go inside the directory
cd /opt/workdir

# do something there
...
```

### Example: run (cleanup step)

```
# do something with the $source_directory
...
```

Now that you know how commands work, checkout the Shared API and Host API, to see what functions you can use inside the commands.

## Docker

The main purpose of Athena is to provide an abstraction of the automation environments and let you use or create your own automation logic without much hassle.

The technology chosen to implement the abstraction is Docker. If *Images*, *Containers*, *Dockerfile* doesn't ring a bell, we recommend that you read a bit about it in order to have an idea on how things work, but if you already do and just want to learn more on how to build your custom containers, please have a look at the Reference Page.

If you intend to just use existing images from DockerHub, then no worries because Athena also has support for it.

The functions that handle docker related operations are under the namespace of `athena.docker.` , read more about it on the Handling Docker page. Functions that are directly related to plugins are under `athena.plugin.` , read more about it on the Handling Plugin page.

**Note:** There are some functions in the plugin namespace that can be wrappers to other operations for simplicity of usage, for instance, they can assume the current plugin to set some options.

### Using containers

There are two types of containers that you can use:

- **External**

  The container name can be a tag that is registered in DockerHub, e.g.: `php:7.0-apache` , `java:7` , `debian:jessie` , etc.

- **Custom** (aka your own container)

  These containers must be built using a Dockerfile that needs to be located inside the plugin's `docker` folder.

  If the *Dockerfile* is located on the root of the `docker` directory, then this is called the *default container* and is used when you have a command that needs to run inside a container and don't explicitly say which one to use.

If you want to have multiple containers then you need to have a sub-folder inside the docker folder and there will be a *Dockerfile* and also a `version.txt` file to version this container. For the *default container*, the version of the plugin is also the version of the container.

```
...
├── docker
│   ├── Dockerfile      # default container
│   └── other           # other container
│       ├── Dockerfile
│       └── version.txt
└── version.txt
```

To use a different container than the *default* container, either an external or a custom one, you must specify it on the *prepare step*, to do so you can use the following functions :

```
athena.plugin.use_container <container name>
```

The container name can be either the tag from DockerHub or the name of the sub-folder in the `docker` directory.

**Hint:** There is also another function that is helpful when using external containers that should run as a daemon.

```
athena.plugin.use_external_container_as_daemon <container name> [instance_name]
```

# Using configurable images

Docker allows you to have configurable images by using the `ARG` instruction, read more about it here.

Using this mechanism, Athena allows you to create multiple environments using one single Dockerfile. This can be very useful because with this you can use

In order to achieve this, you need to use the `ARG` instructions in the Dockerfile and then have a file with key-value (commonly know as ini file), and when you want to use a specific configurable container, while executing a command, you just need to use the flag :

```
 --athena-env=<environment_file|name_of_environment>
```

The specified environment file must follow the *ini file* format and its values will be used for building the container. If a name is specified instead of a file, Athena expects for a file named `<name_of_environment>.env` to exist in the folder of the *Dockerfile*.

## Example

```
$ ./athena example run /path/to/dir --athena-env=/path/to/file/production.env
```

or

```
$ ./athena example run /path/to/dir --athena-env=production
```

# Building containers

Because we have a well defined structure, it is easy to build containers tagging them with the specifics of the location, environment and version. When executing a command that requires a container, Athena will try to find a container with the following pattern :

```
athena-plugin-<plugin_name>[-<custom_container>][-<environment>]-<instance_name>
```

The version is also used to find it (located in the `version.txt` file of the required container). If this container has not been built yet, Athena will do it for you, if it has, then it will use it for the execution of the command.

**Hint:** Once a specific container is built, it will only rebuild if there is an update on the version or if you specify an environment that has not been built yet.

### Example

```
athena-plugin-example-other-production-0:1.0.0
```

# Hooks

There is support for *hooks* in Athena. They can be used to perform any task on a given stage of a `plugin` usage. They must be located in the directory `bin/hooks` of the `plugin`.

### Pre and Post Plugin

These hooks are executed before and after any command is executed. To enable them you just need to add a file called `plugin_pre.sh` and/or `plugin_post.sh` in the *hooks* directory, add logic there and that's it, they're hooked.

# Reusing between commands

Sometimes you might to want to reuse functions or variables between commands. The way to do it is very simple as you can see bellow:

### Functions

- **PRE and POST commands**

  Implement your functions in the `bin/lib/functions.sh`

- **Inside containers**

  Implement them in the `bin/lib/functions.container.sh`.

### Variables

- **PRE and POST commands**

  Add them to `bin/variables.sh`.

- **Inside containers**

  Add them to `bin/variables.container.sh`

That's it, they are now automatically available and you can start using them.

# Custom logo

If you would like to have your own logo when executing the commands of your `plugin`, simply add a text file `.logo` to the root of your plugin with your ascii art.

# Default router

When you need to execute a command inside a container, Athena sets a default router that maps the command that you are executing to the right file.

Sometimes you might feel the need to use a particular docker image that has its own router and you actually want to preserve this behaviour. If this is the case, then you need to specify in the `pre` command file the following option :

```
athena.docker.set_no_default_router 1
```

# Most used functions (aliases)

To simplify the development of plugins, we created some aliases for functions that are frequently used in the commands. Below you can find a list of a few of them, but you can check the full list in the file `lib/shared/aliases.sh`.

## Handling arguments

- `athena.arg`
- `athena.args`
- `athena.nr_args_lt`
- `athena.arg_exists`
- `athena.pop_args`
- `athena.int`

## Handling Filesystem

- `athena.dir_exists_or_fail`
- `athena.path`

## Handling OS

- `athena.usage`
- `athena.exit`
- `athena.exit_with_msg`

## Handling Messages

- `athena.info`
- `athena.error`
- `athena.warn`
- `athena.ok`
- `athena.debug`
- `athena.fatal`
- `athena.print`

## Handling value types

- `athena.is_integer`

# Using CLI Functions

## Handling *argument*

### `athena.argument.append_to_arguments`

This function appends the given argumnets to the argument list ($ATHENA_ARGS).

**USAGE:** `athena.argument.append_to_arguments <argument...>`

**RETURN:** `--`

## athena.argument.arg

This function is a wraper for the athena.argument.get_argument function.

**USAGE:** `athena.argument.arg <argument position or name>`

**RETURN:** `string`

## athena.argument.args

This function is a wrapper for the athena.argument.get_arguments function. It returns the argument list ($ATHENA_ARGS).

**USAGE:** `athena.argument.args`

**RETURN:** `string`

## athena.argument.argument_exists

This function checks if an argument exists in the argument list $ATHENA_ARGS.

**USAGE:** `athena.argument.argument_exists <argument name>`

**RETURN:** `0 (true), 1 (false)`

## athena.argument.argument_exists_and_remove

This function checks if an argument exists (see athena.argument.argument_exists) in the argument list $ATHENA_ARGS and removes it if it exists.

**USAGE:** `athena.argument.argument_exists_and_remove <argument name> [<name of variable to save the value>]`

**RETURN:** `0 (true), 1 (false)`

## athena.argument.argument_exists_or_fail

This function checks if an argument exists (see athena.argument.argument_exists) in the argument list $ATHENA_ARGS. If no argument was given or the argument was not found script execution is exited and an error message is thrown.

**USAGE:** `athena.argument.argument_exists_or_fail <argument name>`

**RETURN:** `--`

## athena.argument.argument_is_not_empty

This function checks if the given arguments string is not empty.

**USAGE:** `athena.argument.argument_is_not_empty <arguments string>`

**RETURN:** `0 (true), 1 (false)`

## athena.argument.argument_is_not_empty_or_fail

This function checks if the given arguments string is not empty. If it is empty execution is stopped and an error message is thrown. If not empty the error code 0 is returned.

**USAGE:** `athena.argument.argument_is_not_empty_or_fail <argument string> [<name>]`

**RETURN:** `0 (true)`

## athena.argument.get_argument

This function returns the requested argument name or value if found in the argument list $ATHENA_ARGS. The function interpretes an given integer as argument index and a given string as argument name (e.g. for the list "a=3 b=5" "3" is return if "a" is requested and "a=3" is returned if "1" is requested).

**USAGE:** `athena.argument.get_argument <argument position or name>`

**RETURN:** `string`

## `athena.argument.get_argument_and_remove`

This function returns the argument name or value (see athena.argument.get_argument) and removes it from the $ATHENA_ARGS list.

**USAGE:** `athena.argument.get_argument_and_remove <argument position or name> [<name of variable to save the value>]`

**RETURN:** `string`

## `athena.argument.get_arguments`

This function will copy the $ATHENA_ARGS array into a variable provided as argument, unless it is being used in a shubshell, then a string containing all the arguments will be output.

**USAGE:** `athena.argument.get_arguments [array_name]`

**RETURN:** `0 (success) | 1 (failure)`

## `athena.argument.get_integer_argument`

This function returns the value for the given argument and if it is not an integer it will exit with error.

**USAGE:** `athena.argument.get_integer_argument <argument position or name> [<error string>]`

**RETURN:** `int`

## `athena.argument.get_path_from_argument`

This function extract a argument string or value (see athena.argument.get_argument) from the $ATHENA_ARGS list and checks if it is a valid directory path. If it is valid the path is return, if not script execution is exited and an error message is thrown.

**USAGE:** `athena.argument.get_path_from_argument <argument position or name>`

**RETURN:** `string`

## `athena.argument.get_path_from_argument_and_remove`

This function returns a valid direcory path if the given argument name or value (see athena.argument.get_path_from_argument) could be converted and removes the argument from the $ATHENA_ARGS list.

**USAGE:** `athena.argument.get_path_from_argument_and_remove <argument position or name>`

**RETURN:** `string`

## `athena.argument.is_integer`

This function checks if an argument is an integer.

**USAGE:** `athena.argument.is_integer <argument>`

**RETURN:** `0 (true), 1 (false)`

## `athena.argument.nr_args_lt`

This function returns the error code 0 if the number of arguments in $ATHENA_ARGS is less than the given number. If not the error code 1 is returned.

**USAGE:** `athena.argument.nr_args_lt <number>`

**RETURN:** `0 (true), 1 (false)`

### `athena.argument.nr_of_arguments`

This function returns the number of arguments found in the argument list $ATHENA_ARGS.

**USAGE:** `athena.argument.nr_of_arguments`

**RETURN:** `int`

### `athena.argument.pop_arguments`

This function pops a number of arguments from the argument list $ATHENA_ARGS.

**USAGE:** `athena.argument.pop_arguments <number>`

**RETURN:** `--`

### `athena.argument.prepend_to_arguments`

This function prepends the given argumnets to the argument list ($ATHENA_ARGS).

**USAGE:** `athena.argument.prepend_to_arguments <argument...>`

**RETURN:** `--`

### `athena.argument.remove_argument`

This function removes an argument from the argument list $ATHENA_ARGS if it is in the list.

**USAGE:** `athena.argument.remove_argument <argument|index>`

**RETURN:** `0 (successful), 1 (failed)`

### `athena.argument.set_arguments`

This function sets the argument list ($ATHENA_ARGS) to the given arguments.

**USAGE:** `athena.argument.set_arguments <argument...>`

**RETURN:** `--`

### `athena.argument.string_contains`

This function if a string contains a substring. With --literal, regex is not parsed, and there's a literal comparison.

**USAGE:** `athena.argument.string_contains <string> <sub-string> [--literal]`

**RETURN:** `0 (true), 1 (false)`

## Handling *color*

### `athena.color.print_color`

This function prints the given string in a given color on STDOUT. Available colors are "green", "red", "blue", "yellow", "cyan", and "normal".

**USAGE:** `athena.color.print_color <color> <string> [<non_colored_string>][<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_debug`

This function prints the given string on STDOUT formatted as debug message if debug mode is set.

**USAGE:** `athena.color.print_debug <string> [<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_error`

This function prints the given string on STDOUT formatted as error message.

**USAGE:** `athena.color.print_error <string> [<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_fatal`

This function prints the given string on STDOUT formatted as fatal message and exit with 1 or the given code.

**USAGE:** `athena.color.print_fatal <string> [<exit_code>] [<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_info`

This function prints the given string on STDOUT formatted as info message.

**USAGE:** `athena.color.print_info <string> [<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_ok`

This function prints the given string on STDOUT formatted as ok message.

**USAGE:** `athena.color.print_ok <string> [<redirect_number>]`

**RETURN:** `--`

### `athena.color.print_warn`

This function prints the given string on STDOUT formatted as warn message.

**USAGE:** `athena.color.print_warn <string> [<redirect_number>]`

**RETURN:** `--`

## Handling *fs*

### `athena.fs.absolutepath`

This function checks if the given argument is a valid absolute path to a directory or file. If not, execution is stopped and an error message is thrown. Otherwise the absolute path is returned.

**USAGE:** `athena.fs.absolutepath <file or directory name>`

**RETURN:** `string`

## `athena.fs.basename`

This function returns the basename of a file. If the file does not exist it will generate an error. It can be a full path or relative to the file.

**USAGE:** `athena.fs.basename <filename>`

**RETURN:** `string`

## `athena.fs.dir_contains_files`

This function checks if the given directory contains files with certain pattern (e.g.: *.sh). Globbing has 'dotglob' and 'extglob' (see BASH(1)) enabled.

**USAGE:** `athena.fs.dir_contains_files <directory> <pattern>`

**RETURN:** `0 (true), 1 (false)`

## `athena.fs.dir_exists_or_create`

This function checks if the given directory name is valid. If not the directory is been created. If the creation fails execution is stopped and an error message is thrown. 0 is returned if the directory exists or was created.

**USAGE:** `athena.fs.file_exists_or_fail <directory name>`

**RETURN:** `0 (true), 1 (false)`

## `athena.fs.dir_exists_or_fail`

This function checks if the given directory name is valid. If not execution is stopped and an error message is thrown. The displayed error message can be passed as second argument.

**USAGE:** `athena.fs.dir_exists_or_fail <directory name> <message>`

**RETURN:** `--`

## `athena.fs.file_contains_string`

This function checks if the filename contains the given string.

**USAGE:** `athena.fs.file_contains_string_<filename> <string>`

**RETURN:** `0 (true), 1 (true)`

## `athena.fs.file_exists_or_fail`

This function checks if the given filename is valid. If not execution is stopped and an error message is thrown. The displayed error message can be passed as second argument.

**USAGE:** `athena.fs.file_exists_or_fail <filename> <message>`

**RETURN:** `--`

## `athena.fs.get_cache_dir`

Returns the name of athena cache directory. If it does not exist, then it will be created and then returned.

**USAGE:** `athena.fs.get_cache_dir`

**RETURN:** `string`

### `athena.fs.get_file_contents`

This function checks if the given filename is valid. If not execution is stopped and an error message is thrown. If the given name is a valid filename the file content returned.

**USAGE:** `athena.fs.get_file_contents <filename>`

**RETURN:** `string`

### `athena.fs.get_full_path`

This function checks if the given argument is a valid directory or file and returns the absolute directory path of the given file or directory (a relative path is converted in an absolute directory path). If the path is not valid execution is stopped and an error message is thrown.

**USAGE:** `athena.fs.get_full_path <file or directory name>`

**RETURN:** `string`

## Handling *os*

### `athena.os.call_with_args`

This function will call the command/function passed as argument with all the arguments existing in $ATHENA_ARGS.

**USAGE:** `athena.os.call_with_args <command>`

**RETURN:** `<command> result | 1 (false) if no <command> was specified or it doesn' exist`

### `athena.os.enable_error_mode`

This function enables the error only output mode. To be used in conjunction with athena.os.exec.

**USAGE:** `athena.os.enable_error_mode`

**RETURN:** `--`

### `athena.os.enable_quiet_mode`

This function enables the no output mode. To be used in conjunction with athena.os.exec.

**USAGE:** `athena.os.enable_quiet_mode`

**RETURN:** `--`

### `athena.os.enable_verbose_mode`

This function enables the all output mode. To be used in conjunction with athena.os.exec.

**USAGE:** `athena.os.enable_verbose_mode`

**RETURN:** `--`

### `athena.os.exec`

This function wraps command execution to allow for switching output modes. The output mode is defined by using the athena.os.set*output** functions.

**USAGE:** `athena.os.exec <function> <args>`

**RETURN:** `int`

## athena.os.exit

This function exits Athena if called (if a forced exit is required). Default exit_code is 1.

**USAGE:** `athena.os.exit [<exit_code>]`

**RETURN:** `--`

## athena.os.exit_with_msg

This function exits Athena with an error message (see athena.os.exit).

**USAGE:** `athena.os.exit_with_msg <error message> [<exit_code>]`

**RETURN:** `--`

## athena.os.function_exists

This functions checks if the function with the given name exists.

**USAGE:** `athena.os.function_exists <name>`

**RETURN:** `0 (true) 1 (false)`

## athena.os.function_exists_or_fail

This functions checks if the function with the given name exists, if not it will abort the current execution.

**USAGE:** `athena.os.function_exists_or_fail <name>`

**RETURN:** `0 (true) 1 (false)`

## athena.os.get_base_dir

This functions returns the base directory of athena.

**USAGE:** `athena.os.get_base_dir`

## athena.os.get_base_lib_dir

This functions returns the base lib directory of athena.

**USAGE:** `athena.os.get_base_lib_dir`

## athena.os.get_command

This function returns the content of the $ATHENA_COMMAND variable. If it is not set execution is stopped and an error message is thrown.

**USAGE:** `athena.os.get_command <command>`

**RETURN:** `string`

## athena.os.get_executable

This function returns the executable for athena.

**USAGE:** `athena.os.get_executable`

**RETURN:** `string`

## `athena.os.get_host_ip`

This functions returns the ip of the host of athena.

**USAGE:** `athena.os.get_host_ip`

**RETURN:** `string`

## `athena.os.get_instance`

This function returns the value of the current instance as set in the $ATHENA_INSTANCE variable.

**USAGE:** `athena.os.get_instance`

**RETURN:** `string`

## `athena.os.get_prefix`

This functions returns the prefix that is used to create names for

**USAGE:** `athena.os.get_prefix`

**RETURN:** `string`

## `athena.os.getenv_or_fail`

This functions checks if the env variable with the given name exists, if not it will abort the current execution.

**USAGE:** `athena.os.getenv_or_fail <name>`

**RETURN:** `string`

## `athena.os.handle_exit`

This function handles the signals sent to and by athena.

**USAGE:** `athena.os.handle_exit <signal>`

**RETURN:** `--`

## `athena.os.include_once`

This function checks if a given Bash source file exists and includes it if it wasn't loaded before. If it was loaded nothing is done (avoid multiple sourcing).

**USAGE:** `athena.os.include_once <Bash source file>`

**RETURN:** `--`

## `athena.os.is_command_set`

This functions checks if $ATHENA_COMMAND variable is set.

**USAGE:** `athena.os.is_command_set`

**RETURN:** `0 (true) 1 (false)`

## `athena.os.is_debug_active`

This function returns the error code 0 if the debug flag ($ATHENA_IS_DEBUG) is set. If not it returns the error code 1.

**USAGE:** `athena.os.is_debug_active`

**RETURN:** `0 (true), 1 (false)`

## `athena.os.is_git_installed`

This function checks if the 'git' command is available (i.e. if git is installed). If not execution is stopped and an error message is thrown.

**USAGE:** `athena.os.is_git_installed`

**RETURN:** `--`

## `athena.os.is_linux`

This function checks if Athena runs on a Linux machine.

**USAGE:** `athena.os.is_mac`

**RETURN:** `0 (true), 1 (false)`

## `athena.os.is_mac`

This function checks if Athena runs on a Mac OS X.

**USAGE:** `athena.os.is_mac`

**RETURN:** `0 (true), 1 (false)`

## `athena.os.is_sudo`

This function checks if the $ATHENA_SUDO variable is set.

**USAGE:** `athena.os.is_sudo`

**RETURN:** `0 (true), 1 (false)`

## `athena.os.override_exit_handler`

This functions overrides the exit handler with the default signals to catch.

**USAGE:** `athena.os.override_exit_handler <function_name>`

**RETURN:** `--`

## `athena.os.print_stacktrace`

This function prints the stacktrace.

**USAGE:** `athena.os.print_stacktrace`

**RETURN:** `--`

## `athena.os.register_exit_handler`

This function register the exit handler that takes the decision of what to do when interpreting the exit codes and signals.

**USAGE:** `athena.os.register_exit_handler <function_name> <list_of_signals_to_trap>`

**RETURN:** `--`

## athena.os.return

This function assigns a value to a variable and overcomes the problem of assignment in subshells losing the current environment. It is meant to be used in the getters and expects that the function using it follows the convention athena.get_. NOTE: when used in subshell it will echo the value to be assigned to a variable.

**USAGE:** `athena.os.return <value> [<name_of_variable_to_assign_to>]`

**RETURN:** `string`

## athena.os.set_command

This function sets the $ATHENA_COMMAND variable to the given command string if it is not empty. If it is empty execution is stopped and an error message is thrown.

**USAGE:** `athena.os.set_command <command>`

**RETURN:** `--`

## athena.os.set_debug

This function sets the debug flag ($ATHENA_IS_DEBUG) to the given value. If no value is provided $ATHENA_IS_DEBUG is set to 0 (disabled).

**USAGE:** `athena.os.set_debug <debug value>`

**RETURN:** `--`

## athena.os.set_exit_handler

This functions registers the exit handler with the default signals to catch.

**USAGE:** `athena.os.set_exit_handler`

**RETURN:** `--`

## athena.os.set_instance

This functions sets the instance value.

**USAGE:** `athena.os.set_instance <value>`

**RETURN:** `--`

## athena.os.split_string

This function splits up a string on the specified field separator and will write the array into the given variable.

**USAGE:** `athena.os.split_string <string_to_split> <separator_character> <variable_name>`

**RETURN:** `--`

## athena.os.usage

This function prints the usage and exits with 1 and handles the name of the command automatically and the athena executable.

**USAGE:** `athena.os.usage [<min_args>] [<options>] [<multi-line options>]`

**RETURN:** `--`

# Handling *utils*

## `athena.utils.add_to_array`

This functions adds elements to the given array.

**USAGE:** `athena.utils.add_to_array <array_name> <element...>`

**RETURN:** `0 (true), 1 (false)`

## `athena.utils.array_pop`

This function pops elements from the given array, if argument 2 is an integer then it will pop as many times as specified.

**USAGE:** `athena.utils.array_pop <array_name> [number_of_times]`

**RETURN:** `0 (true), 1 (false)`

## `athena.utils.compare_number`

This function compares a number to another with the given operator (>, >=, <, <=)

**USAGE:** `athena.utils.compare_number <number_a> <number_b> <comparator>`

## `athena.utils.find_index_in_array`

This function returns the index of the element specified.

**USAGE:** `athena.utils.find_index_in_array <array_name> <needle> [strict]`

**RETURN:** `0 (true), 1 (false)`

## `athena.utils.get_array`

This function returns the elements of the given array in case of subshell assignment or stores them in a new variable if specified in argument 2.

**USAGE:** `athena.utils.get_array <array_name> [other_array_name]`

**RETURN:** `0 (true), 1 (false)`

## `athena.utils.get_version_components`

This function extracts the values from a Semantic Versioning 2 format into an array. index 0 contains the operation, index 1 the MAJOR version, index 2 MINOR version and index 3 the PATCH version.

**USAGE:** `athena.utils.get_version_components <sem_ver_string> <array_name_to_store>`

## `athena.utils.in_array`

This function checks if the element exists in the given array.

**USAGE:** `athena.utils.in_array <array_name> <element> [strict]`

**RETURN:** `0 (true), 1 (false)`

## `athena.utils.is_integer`

This function checks if a value is an integer.

**USAGE:** `athena.utils.is_integer <value>`

**RETURN:** `0 (true), 1 (false)`

## athena.utils.prepend_to_array

This function prepends the given elements to the specified array.

**USAGE:** `athena.utils.prepend_to_array <array_name> <element...>`

**RETURN:** `0 (true), 1 (false)`

## athena.utils.remove_from_array

This function removes the specified element from the array.

**USAGE:** `athena.utils.remove_from_array <array_name> <needle> [strict]`

**RETURN:** `0 (succeeded), 1 (failed)`

## athena.utils.set_array

This function assigns the given elements to the specified array.

**USAGE:** `athena.utils.set_array <array_name> <element...>`

**RETURN:** `0 (true), 1 (false)`

## athena.utils.validate_version

This function validates if the given version meets the expected version criteria.

**USAGE:** `athena.utils.validate_version <version_str> <expected_version|base_version end_version>`

## athena.utils.validate_version_format

This function validates if the given version follows Semantic Versioning 2.0.

**USAGE:** `athena.utils.validate_version_format <version>`

**RETURN:** `0 (true) 1 (false)`

- `athena.docker.stop_container`
- `athena.docker.volume_create`
- `athena.docker.volume_exists`
- `athena.docker.volume_exists_or_create`
- `athena.docker.wait_for_string_in_container_logs`
- `athena.plugin.build`

- Handling *plugin*
  - `athena.plugin.build`
  - `athena.plugin.check_dependencies`
  - `athena.plugin.get_available_cmds`
  - `athena.plugin.get_bootstrap_dir`
  - `athena.plugin.get_container_name`
  - `athena.plugin.get_container_to_use`
  - `athena.plugin.get_environment`
  - `athena.plugin.get_environment_build_file`
  - `athena.plugin.get_image_name`
  - `athena.plugin.get_image_version`
  - `athena.plugin.get_plg`
  - `athena.plugin.get_plg_bin_dir`
  - `athena.plugin.get_plg_cmd_dir`
  - `athena.plugin.get_plg_dir`
  - `athena.plugin.get_plg_docker_dir`
  - `athena.plugin.get_plg_hooks_dir`
  - `athena.plugin.get_plg_lib_dir`
  - `athena.plugin.get_plg_version`
  - `athena.plugin.get_plugin`
  - `athena.plugin.get_plugins_dir`
  - `athena.plugin.get_prefix_for_container_name`
  - `athena.plugin.get_shared_lib_dir`
  - `athena.plugin.get_subplg_version`
  - `athena.plugin.get_tag_name`
  - `athena.plugin.handle`
  - `athena.plugin.handle_container`
  - `athena.plugin.handle_environment`
  - `athena.plugin.init`
  - `athena.plugin.is_environment_specified`
  - `athena.plugin.plugin_exists`
  - `athena.plugin.print_available_cmds`
  - `athena.plugin.require`
  - `athena.plugin.run_command`
  - `athena.plugin.run_container`
  - `athena.plugin.set_container_name`
  - `athena.plugin.set_environment`
  - `athena.plugin.set_environment_build_file`
  - `athena.plugin.set_image_name`
  - `athena.plugin.set_image_version`
  - `athena.plugin.set_plg_cmd_dir`
  - `athena.plugin.set_plugin`
  - `athena.plugin.use_container`
  - `athena.plugin.use_external_container_as_daemon`
  - `athena.plugin.validate_plugin_name`
  - `athena.plugin.validate_usage`

# Using CLI Functions

## Handling *docker*

### `athena.docker`

This is a wrapper function for executing docker, which helps with mocking and tweaking.

**USAGE:** `athena.docker <args>`

**RETURN:** `--`

### `athena.docker.add_autoremove`

This function adds the --rm flag (automatically remove the container when it exits) to the docker run option string ($ATHENA_DOCKER_OPTS).

**USAGE:** `athena.docker.add_autoremove`

**RETURN:** `--`

### `athena.docker.add_daemon`

This function adds the daemon flag to the docker run option string ($ATHENA_DOCKER_OPTS).

**USAGE:** `athena.docker.add_daemon`

**RETURN:** `--`

### `athena.docker.add_env`

This function adds an environment variable to the docker run option string ($ATHENA_DOCKER_OPTS).

**USAGE:** `athena.docker.add_env <variable name> <variable value>`

**RETURN:** `--`

### `athena.docker.add_envs_from_file`

This function adds environment variables from the given file (ini format).

**USAGE:** `athena.docker.add_envs_from_file <filename>`

**RETURN:** `--`

### `athena.docker.add_envs_with_prefix`

This function adds environment variables with the given prefix to the docker run option string.

**USAGE:** `athena.docker.add_envs_with_prefix <prefix>`

**RETURN:** `--`

### `athena.docker.add_option`

This function adds the given option to the docker run option string ($ATHENA_DOCKER_OPTS).

**USAGE:** `athena.docker.add_option <your option>`

**RETURN:** `--`

## `athena.docker.build`

This is a wrapper function for executing docker build, which helps with mocking and tweaking.

**USAGE:** `athena.docker.build <args>`

**RETURN:** `--`

## `athena.docker.build_container`

This function builds a docker image using the given tag name, version and docker directory (Dockerfile must exists in the given directory). If a docker image with tag:version already exists nothing is done. If not the function checks if it is in the right directory, loads build environment variables if provided (see athena.docker.get_build_args), and builds the docker image. If the function is called in a wrong directory or the build is unsuccessful execution is stopped and an error message is thrown.

**USAGE:** `athena.docker.build_container <tag name> <version> <docker directory>`

**RETURN:** `--`

## `athena.docker.build_from_plugin`

This function builds a docker image for a plugin. Plugin name, sub-plugin name, and version must be provided. If no valid docker directory or Dockerfile is found execution is stopped and an error message is thrown.

**USAGE:** `athena.docker.build_from_plugin <plugin name> <sub-plugin name> <plugin version>`

**RETURN:** `--`

## `athena.docker.cleanup`

This function cleans up the container for the current plugin in case is not running.

**USAGE:** `athena.docker.cleanup`

**RETURN:** `--`

## `athena.docker.container_has_started`

This function checks if container has started

**USAGE:** `athena.docker.container_has_started`

**RETURN:** `0 (true) 1 (false)`

## `athena.docker.disable_auto_cleanup`

This function disables the automatic removal of the container.

**USAGE:** `athena.docker.disable_auto_cleanup`

**RETURN:** `--`

## `athena.docker.disable_privileged_mode`

This function disables the privileged mode of the container.

**USAGE:** `athena.docker.disable_privileged_mode`

**RETURN:** `--`

## `athena.docker.exec`

This is a wrapper function for executing docker exec, which helps with mocking and tweaking.

**USAGE:** `athena.docker.exec <args>`

**RETURN:** `--`

## athena.docker.get_build_args

This function generates and stores, in the given array, the build arguments from the build args file returned by athena.docker.get_build_args_file or does nothing if no file was found.

**USAGE:** `athena.docker.get_build_args <array_name>`

**RETURN:** `string | 1 (false)`

## athena.docker.get_build_args_file

This function checks if a docker build environment file is defined in the $ATHENA_PLG_DOCKER_ENV_BUILD_FILE variable. If not it returns 1. If defined it checks if the file exists and returns the name of the file.

**USAGE:** `athena.docker.get_build_args_file`

**RETURN:** `string | 1 (false)`

## athena.docker.get_ip

This function returns the ip address of the docker machine. It checks for the 'docker-machine' and 'boot2docker' commands to do this (default on Mac). If not found it searches for the docker0 device (default on Linux) and returns the localhost ip if found. If no docker0 device is available the function assumes to run inside a docker container and checks if a docker daemon is running in this container. If so localhost is returned. If not it returns the default route ip address.

**USAGE:** `athena.docker.get_ip`

**RETURN:** `string`

## athena.docker.get_ip_for_container

This function returns the container internal ip provided by docker.

**USAGE:** `athena.docker.get_ip_for_container <container_name>`

**RETURN:** `string`

## athena.docker.get_options

This function outputs the extra options to be passed for running docker. As an alternative you can also assign to a given array name.

**USAGE:** `athena.docker.get_options [array_name]`

**RETURN:** `string`

## athena.docker.get_tag_and_version

## athena.docker.handle_run_type

This function checks if either the daemon or the autoremove flag is set in the docker run option string ($ATHENA_DOCKER_OPTS). If one of both is set it returns the error code 0. If none is set it sets the autoremove flag (--rm) and returns the error code 1.

**USAGE:** `athena.docker.handle_run_type`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.has_option`

This function checks if the given option is already set.

**USAGE:** `athena.docker.has_option <option> [strict]`

**RETURN:** `0 (true) 1 (false)`

## `athena.docker.image_exists`

This function checks if a docker image with the given tag name and version exists.

**USAGE:** `athena.docker.image_exists <image name> <version>`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.images`

This is a wrapper function for executing docker images, which helps with mocking and tweaking.

**USAGE:** `athena.docker.images <args>`

**RETURN:** `--`

## `athena.docker.inspect`

This is a wrapper function for executing docker inspect, which helps with mocking and tweaking.

**USAGE:** `athena.docker.inspect <args>`

**RETURN:** `--`

## `athena.docker.is_auto_cleanup_active`

This function checks if the automatic removal of the container is active.

**USAGE:** `athena.docker.is_auto_cleanup_active`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.is_container_running`

This function checks if a docker container with the given name is running. If no container with the given name is running all stopped containers with this name are removed (to avoid collisions).

**USAGE:** `athena.docker.is_container_running <container name>`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.is_current_container_not_running_or_fail`

This function checks if the container assigned for running is already running and if it is then exits with an error message.

**USAGE:** `athena.docker.is_current_container_not_running_or_fail [msg]`

**RETURN:** `0 (false)`

## `athena.docker.is_current_container_running`

This function checks if the container assigned for running is already running.

**USAGE:** `athena.docker.is_current_container_running`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.is_default_router_to_be_used`

This function checks if the default router should be used.

**USAGE:** `athena.docker.is_default_router_to_be_used`

**RETURN:** `0 (true) 1 (false)`

## `athena.docker.is_privileged_mode_enabled`

This function checks if the docker privileged mode is enabled.

**USAGE:** `athena.docker.is_privileged_mode_enabled`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.is_running_as_daemon`

This function checks if docker option -d is already set.

**USAGE:** `athena.docker.is_running_as_daemon`

**RETURN:** `0 (true) 1 (false)`

## `athena.docker.list_athena_containers`

This function returns a list of athena custom containers.

**USAGE:** `athena.docker.list_athena_containers`

**RETURN:** `string`

## `athena.docker.logs`

This is a wrapper function for executing docker logs, which helps with mocking and tweaking.

**USAGE:** `athena.docker.logs <args>`

**RETURN:** `--`

## `athena.docker.mount`

This function adds the given volume to the docker run option string ($ATHENA_DOCKER_OPTS). If source or target directory are not specified it stops execution and throws an error message and if source is neither a file or a directory also stops the execution.

**USAGE:** `athena.docker.mount <source> <target>`

**RETURN:** `--`

## `athena.docker.mount_dir`

This function adds the given volume to the docker run option string ($ATHENA_DOCKER_OPTS). If source or target directory are not specified it stops execution and throws an error message or if source is not a directory also stops.

**USAGE:** `athena.docker.mount_dir <source directory> <target directory>`

**RETURN:** `--`

## `athena.docker.mount_dir_from_plugin`

This function adds the given volume to the docker run option from a relative path to the current plugin.

**USAGE:** `athena,docker.mount_dir_from_plugin <relative_path_from_plugin> <target_directory>`

**RETURN:** `--`

## `athena.docker.network_create`

Create a new docker network with .

**USAGE:** `athena.docker.network_create <name> [opts...]`

**RETURN:** `0 (true), exit 1 (failed)`

## `athena.docker.network_exists`

Check if docker network with the exists.

**USAGE:** `athena.docker.network_exists <name> [opts...]`

**RETURN:** `0 (true), exit 1 (failed)`

## `athena.docker.network_exists_or_create`

Check if a network with the already exists, if not the network is created.

**USAGE:** `athena.docker.network_exists_or_create <name> [opts...]`

**RETURN:** `0 (true), 1 (false)`

## `athena.docker.print_or_follow_container_logs`

Either print or follow the output of one or more container logs.

**USAGE:** `athena.docker.print_or_follow_container_logs <containers> [-f]`

**RETURN:** `--`

## `athena.docker.remove_container_and_image`

This function removes a docker container and the associated image.

**USAGE:** `athena.docker.remove_container_and_image <tag name> <version>`

**RETURN:** `--`

## `athena.docker.rm`

This is a wrapper function for executing docker rm, which helps with mocking and tweaking.

**USAGE:** `athena.docker.rm <args>`

**RETURN:** `--`

## `athena.docker.rmi`

This is a wrapper function for executing docker rmi, which helps with mocking and tweaking.

**USAGE:** `athena.docker.rmi <args>`

**RETURN:** `--`

## `athena.docker.run`

This is a wrapper function for executing docker run, which helps with mocking and tweaking.

**USAGE:** `athena.docker.run <args>`

**RETURN:** `--`

## `athena.docker.run_container`

This function runs a container.

**USAGE:** `athena.docker.run_container <container_name> <tag_name>`

**RETURN:** `--`

## `athena.docker.run_container_with_default_router`

This function runs a container using the default router. The ATHENA_COMMAND and ATHENA_ARGS will be set dynamically within the router inside the container so that even executing something inside an already running container will have the correct COMMAND being executed with the correct ARGS.

**USAGE:** `athena.docker.run_container_with_default_router <container_name> <tag_name> <command>`

**RETURN:** `--`

## `athena.docker.set_no_default_router`

This function specifies that the default router should not be used.

**USAGE:** `athena.docker.set_no_default_router [value]`

**RETURN:** `--`

## `athena.docker.set_options`

This function sets the options to be passed to docker.

**USAGE:** `athena.docker.set_options <options>`

**RETURN:** `--`

## `athena.docker.stop_all_containers`

This function stops and removes docker containers which run in this instance with the given name. If '--global' is set as additional argument all (regardless the instance) docker containers with the given name are stopped/removed. Since the containers are stopped/removed in parallel the function waits until all containers were stopped and removed successfully. OPTION: --global

**USAGE:** `athena.docker.stop_all_containers <name_to_filter> [<option>]`

**RETURN:** `--`

## `athena.docker.stop_container`

This function stops a docker container with the given name if running or the current container f container name is not specified. In any case (running or already stopped) the containers with the given name will be removed including associated volumes.

**USAGE:** `athena.docker.stop_container [container name]`

**RETURN:** `--`

### `athena.docker.volume_create`

Create a new docker volume with .

**USAGE:** `athena.docker.volume_create <name>`

**RETURN:** `0 (true), exit 1 (failed)`

### `athena.docker.volume_exists`

Check if docker volume with the exists.

**USAGE:** `athena.docker.volume_exists <name>`

**RETURN:** `0 (true), exit 1 (failed)`

### `athena.docker.volume_exists_or_create`

Check if a volume with the already exists, if not the volume is created.

**USAGE:** `athena.docker.volume_exists_or_create <name>`

**RETURN:** `0 (true), 1 (false)`

### `athena.docker.wait_for_string_in_container_logs`

This function checks if a certain string can be found in the container logs. If not the container is considered to be not running and the function keeps rechecking every second. If 300 seconds are reached it stops execution and throws an error message.

**USAGE:** `athena.docker.wait_for_string_in_container_logs <container> <log message>`

**RETURN:** `0 (true)`

### `athena.plugin.build`

This function gets the name ($ATHENA_PLUGIN), docker directory (see athena.plugin.get_plg_docker_dir), tag name (see athena.plugin.get_tag_name), and version ($ATHENA_PLG_IMAGE_VERSION) of the current plugin and builds a docker image from these resources. If no valid Dockerfile exists or the build is unsuccessful execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.build`

**RETURN:** `--`

## Handling *plugin*

### `athena.plugin.build`

This function builds the container for the current plugin.

**USAGE:** `athena.plugin.build`

**RETURN:** `--`

### `athena.plugin.check_dependencies`

This function checks if the given plugin has dependencies (i.e. it checks the content of dependencies.ini). It checks each specified dependency by name and version. If a plugin specified as dependency is not installed or has the wrong version number the error code 1 is returned. If all dependencies are installed the error code 0 is returned.

**USAGE:** `athena.plugin.check_dependencies <plugin name>`

**RETURN:** `0 (true), 1 (false)`

## `athena.plugin.get_available_cmds`

This function prints the usage info list of all commands found for this plugin (in $ATHENA_PLG_CMD_DIR).

**USAGE:** `athena.plugin.get_available_cmds`

**RETURN:** `--`

## `athena.plugin.get_bootstrap_dir`

This function returns the bootstrap directory.

**USAGE:** `athena.plugin.get_bootstrap_dir`

**RETURN:** `string`

## `athena.plugin.get_container_name`

This function returns a generic container name generated from the current plugin and instance settings (i.e. $ATHENA_PLUGIN and $ATHENA_INSTANCE variables will be considered for the container name generation).

**USAGE:** `athena.plugin.get_container_name`

**RETURN:** `string`

## `athena.plugin.get_container_to_use`

This function checks if a container was set for running (in the $ATHENA_PLG_CONTAINER_TO_USE variable). If so the container name is returned. If not the error code 1 is returned.

**USAGE:** `athena.plugin.get_container_to_use`

**RETURN:** `string`

## `athena.plugin.get_environment`

This function returns the value of the current plugin environment as set in the $ATHENA_PLG_ENVIRONMENT variable. If $ATHENA_PLG_ENVIRONMENT is not set error code 1 is returned.

**USAGE:** `athena.plugin.get_environment`

**RETURN:** `string`

## `athena.plugin.get_environment_build_file`

This function returns the current docker build environment file name as set in the $ATHENA_PLG_DOCKER_ENV_BUILD_FILE variable. If $ATHENA_PLG_DOCKER_ENV_BUILD_FILE is not set error code 1 is returned.

**USAGE:** `athena.plugin.get_environment_build_file`

**RETURN:** `string`

## `athena.plugin.get_image_name`

This function returns the value of the current plugin image name as set in the $ATHENA_PLG_IMAGE_NAME variable.

**USAGE:** `athena.plugin.get_image_name`

**RETURN:** `string`

## athena.plugin.get_image_version

This function returns the value of the current plugin image version as set in $ATHENA_PLG_IMAGE_VERSION variable.

**USAGE:** `athena.plugin.get_image_version`

**RETURN:** `string`

## athena.plugin.get_plg

This function returns the name of the current plugin as set in the $ATHENA_PLUGIN variable.

**USAGE:** `athena.plugin.get_plg`

**RETURN:** `string`

## athena.plugin.get_plg_bin_dir

This function returns the plugin binary directory name and checks if the plugin root exists. If not, execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_bin_dir [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plg_cmd_dir

This function returns the plugin command directory name and checks if the plugin root exists. If not execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_cmd_dir [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plg_dir

This function returns the plugin root directory name and checks if it exists. If it does not exist execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_dir [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plg_docker_dir

This function returns the plugin docker directory name and checks if the plugin root exists. If not execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_docker_dir <plugin name>`

**RETURN:** `string`

## athena.plugin.get_plg_hooks_dir

This function returns the plugin hooks directory and checks if the plugin root root exists. If not, execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_hooks_dir [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plg_lib_dir

This function returns the plugin library directory name and checks if the plugin root exists. If not, execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.get_plg_lib_dir [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plg_version

This function returns the version of a plugin as set in its version.txt.

**USAGE:** `athena.plugin.get_plg_version [plugin name]`

**RETURN:** `string`

## athena.plugin.get_plugin

This function wraps the athena.plugin.get_plg function.

**USAGE:** `athena.plugin.get_plugin`

**RETURN:** `string`

## athena.plugin.get_plugins_dir

This function returns the directory name where plugins are installed (i.e. the value of the $ATHENA_PLGS_DIR variable).

**USAGE:** `athena.plugin.get_plugins_dir`

**RETURN:** `string`

## athena.plugin.get_prefix_for_container_name

This function returns the prefix for creating a container name.

**USAGE:** `athena.plugin.get_prefix_for_container_name [plugin name]`

**RETURN:** `string`

## athena.plugin.get_shared_lib_dir

This function returns the shared lib directory.

**USAGE:** `athena.plugin.get_shared_lib_dir`

**RETURN:** `string`

## athena.plugin.get_subplg_version

This function returns the version of a sub-plugin as set in its version.txt.

**USAGE:** `athena.plugin.get_subplg_version <plugin name> <sub-plugin name>`

**RETURN:** `string`

## `athena.plugin.get_tag_name`

This function generates and returns a tag name from current plugin settings (i.e. $ATHENA_PLG_IMAGE_NAME, $ATHENA_PLUGIN, and $ATHENA_PLG_ENVIRONMENT variables will be considered for the tag name generation).

**USAGE:** `athena.plugin.get_tag_name`

**RETURN:** `string`

## `athena.plugin.handle`

This function handles the routing of the plugin.

**USAGE:** `athena.plugin.handle <command> <command_dir> <lib_dir> <bin_dir> <hooks_dir>`

**RETURN:** `0 (sucessfull), 1 (failed)`

## `athena.plugin.handle_container`

This function checks if the name, docker directory, and container of the current plugin are set. If no container is set it checks if a Dockerfile is available in the docker directory and will run athena.docker.build with it. If no Dockerfile is available it will return doing nothing (some plugins might not need a container). If a container was already set it will check if its docker directory (e.g. of a given sub-plugin) with version.txt exists, sets image version ($ATHENA_PLG_IMAGE_VERSION) and plugin name ($ATHENA_PLUGIN) accordingly, and builds the docker image for it. If no valid docker directory is found execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.handle_container`

**RETURN:** `--`

## `athena.plugin.handle_environment`

This function checks if the name, environment, and container of the current plugin are set. If so it checks if a build environment file exists and sets the $ATHENA_PLG_DOCKER_ENV_BUILD_FILE variable pointing to it. If not execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.handle_environment`

**RETURN:** `--`

## `athena.plugin.init`

This function checks if the given plugin was initialised (i.e. if athena.lock is set in the plugin directory). If not it checks the plugin dependencies (using athena.plugin.check_dependencies) and then runs the plugin init script if successful. If the required plugins (dependencies) are not installed it stops execution and throws an error message.

**USAGE:** `athena.plugin.init <plugin_name>`

**RETURN:** `--`

## `athena.plugin.is_environment_specified`

This function checks if the current plugin environment ($ATHENA_PLG_ENVIRONMENT) is set. If set error code 0 is returned. If not error code 1 is returned.

**USAGE:** `athena.plugin.is_environment_specified`

**RETURN:** `0 (true), 1 (false)`

## athena.plugin.plugin_exists

This function checks if the plugin root directory of the given plugin exists. If not execution is stopped and an error message is thrown. If a version is given as second argument it checks if it complies with the found plugin version. If not an error message is thrown.

**USAGE:** `athena.plugin.plugin_exists <plugin name> <version>`

**RETURN:** `0 (true), 1 (false)`

## athena.plugin.print_available_cmds

This function prints the usage screen of the given plugin including all commands found in the plugin directory ($ATHENA_PLG_CMD_DIR).

**USAGE:** `athena.plugin.print_available_cmds <plugin_name>`

**RETURN:** `--`

## athena.plugin.require

This function checks if the plugin root directory of the given plugin name exists. If not it stops execution and throws an error message. If it exists it sources 'bin/variables.sh' and 'bin/lib/functions.sh' if available in the plugin.

**USAGE:** `athena.plugin.require <plugin name> <version>`

**RETURN:** `--`

## athena.plugin.run_command

This function runs the given command from the plugin.

**USAGE:** `athena.plugin.run_command <command_name> <plugin_cmd_dir>`

**RETURN:** `int`

## athena.plugin.run_container

This functions runs the given container.

**USAGE:** `athena.plugin.run_container <command>`

**RETURN:** `0 (true), 1 (false)`

## athena.plugin.set_container_name

This function sets the current container name in the $ATHENA_CONTAINER_NAME variable to the given value.

**USAGE:** `athena.plugin.set_container_name <container name>`

**RETURN:** `--`

## athena.plugin.set_environment

This function sets the current plugin environment in the $ATHENA_PLG_ENVIRONMENT variable to the given value. If no plugin environment is provided execution will be stopped.

**USAGE:** `athena.plugin.set_environment <plugin environment>`

**RETURN:** `--`

## athena.plugin.set_environment_build_file

This function sets the current docker build environment file name in the $ATHENA_PLG_DOCKER_ENV_BUILD_FILE variable to the given value.

**USAGE:** `athena.plugin.set_environment_build_file <docker build environment file name>`

**RETURN:** `--`

## athena.plugin.set_image_name

This function sets the current plugin image name in the $ATHENA_PLG_IMAGE_NAME variable to the given value.

**USAGE:** `athena.plugin.set_image_name <image name>`

**RETURN:** `--`

## athena.plugin.set_image_version

This function sets the current plugin image version in the $ATHENA_PLG_IMAGE_VERSION variable to the given value.

**USAGE:** `athena.plugin.set_image_version <image version>`

**RETURN:** `--`

## athena.plugin.set_plg_cmd_dir

This functions sets the plg cmd dir(s). The parameter should be one or more directories separated by colons.

**USAGE:** `athena.plugin.set_plg_cmd_dir <dir(s)>`

**RETURN:** `--`

## athena.plugin.set_plugin

This function sets the current plugin in the $ATHENA_PLUGIN variable to the given value.

**USAGE:** `athena.plugin.set_plugin <plugin name>`

**RETURN:** `--`

## athena.plugin.use_container

This function sets the container that will be used for running (i.e. assigning the given value to $ATHENA_PLG_CONTAINER_TO_USE variable). If no value is provided it stops the execution and throws an error message.

**USAGE:** `athena.plugin.use_container <container name>`

**RETURN:** `--`

## athena.plugin.use_external_container_as_daemon

This function uses an external container as a daemon and disables the default router.

**USAGE:** `athena.plugin.use_external_container_as_daemon <container name> [instance_name]`

**RETURN:** `--`

## athena.plugin.validate_plugin_name

This function checks if the given argument (e.g. ) is not empty. If the given string is empty execution is stopped and an error message is thrown.

**USAGE:** `athena.plugin.validate_plugin_name <$VARIABLE>`

**RETURN:** `--`

## `athena.plugin.validate_usage`

This function checks the number of arguments in the given list. If no argument is given it shows the available commands of the given plugin and exits. If another argument than 'init' or 'cleanup' is given it checks if the plugin was initialised.

**USAGE:** `athena.plugin.validate_usage <plugin_name> <argument1> <argument2> ...`

**RETURN:** `--`

# Global CLI flags

These are flags that can be used with any command of any `plugin`.

## Enabling debug messages

To enable the debug messages append your command with :

```
--athena-dbg
```

## Disabling the logo

To disable the logo when invoking the commands append your command with :

```
--athena-no-logo
```

## Specifying the environment

In case your plugin supports multiple environments for the same container append your command with :

```
--athena-env=<name|file_with_environment_config>
```

## Overriding the container's dns nameserver

```
--athena-dns=<nameserver_ip>
```

# Contributing to Athena

Athena is an OLX open source project that is both under very active development and is also being used to automate stuff at OLX. We're still working the details to make contributing to this project as easy and transparent as possible. Hopefully with the help of this document and your feedback we will eventually make it.

# Our Development Process

Some of our core contributors will be working directly on GitHub. These changes will be public from the beginning.

## `master` changes fast

We move fast and most likely things will break. Every time there is a commit our CI server will run the tests and hopefully they will pass all times. We will do our best to properly communicate the changes that can affect the application API and always version appropriately in order to make easier for you to use a specific version.

### Pull Requests

The core contributors will be monitoring for pull requests. When we get one, we will pull it in and apply it to our codebase and run our test suite to ensure nothing breaks. Then one of the core contributors needs to verify that all is working appropriately. When the API changes we may need to fix internal uses, which could cause some delay. We'll do our best to provide updates and feedback throughout the process.

*Before* submitting a pull request, please make sure the following is done:

1. Fork the repo and create your branch from `master`.
2. If you've added code that should be tested, add tests!
3. If you've changed APIs, update the documentation.
4. Ensure the test suite passes ( `bashunit tests` ).

# Bugs

### Where to Find Known Issues

We will be using GitHub Issues for our public bugs. We will keep a close eye on this and try to make it clear when we have an internal fix in progress. Before filing a new issue, try to make sure your problem doesn't already exist.

### Reporting New Issues

A good issue description starts with information about your Athena Environment. Since Athena 0.10.1 you can run `athena new-issue` in order to generate a short Markdown preamble for your issue. Please copy and paste the output of that command in your issue description alongside all steps necessary to reproduce your problem. Please also provide the actual error output and specify what you were expecting instead.

# How to Get in Touch

- Github Issues
- Mailing list - Athena in Google Groups

# Development best practices

## Common

- Global Variables :
  - *SHOULD* be avoided and used only to store global state
  - *MUST* be handled using getters and setters
  - *MUST* be named in uppercase, e.g.: `MY_VARIABLE_NAME`
  - *MUST* be prefixed with ATHENA *when defined in Athena engine and prefixed with ```ATHENA_PLG*${PLUGIN*NAME}*``` when defined in a plugin, e.g.:
    - `ATHENA_MY_VARIABLE_NAME`
    - `ATHENA_PLG_SELENIUM_MY_VARIABLE_NAME`
- Local variables :
  - *MUST* be declared with the local keyword
  - *MUST* be named in lowercase
- CLI functions:
  - *MUST* be tested and documented
  - Documentation *MUST* follow the following format :

    ```
    # Description
    # USAGE: <name_of_function> [argument...]
    # RETURN: <type_of_return>
    ```

    while arguments have the rules:
    - mandatory arguments needs to be inside < > like `<varname1>`
    - optional arguments needs to be inside [ ] like `[varname2]`
    - one or more arguments needs to be written as `varname3...` like `<directory...>` meaning one or more directories are mandatory

    and return values the rules:
    - `<type_of_return>` should be written as `--` if no return value exists
    - return types should be written in lower case like `string` or `int`
  - when used as a core athena function *MUST* follow the naming schema `athena.${context}.${function_name}`
  - when used as a plugin function (inside the `lib` directory) *MUST* follow the naming schema `athena.plugins.${plugin_name}.${function_name}`
  - *MUST* always return 0 when success and not 0 when fail
- Plugin Commands
  - *MUST NOT* have a shebang line
  - Arguments *MUST* only be accessed or setted using the functions in `athena.argument` library

## Plugins

- Library functions *MUST* be located inside `${PLUGIN}/bin/lib` and when multiple contexts are handled *MUST* follow the naming schema `${PLUGIN}/bin/lib/functions.${CONTEXT}.sh` , e.g.: `java/bin/lib/functions.api.sh`
- Library functions *MUST* follow the following name schema : `athena.plugins.${plugin_name}.${function_name}`
- Folders that are supposed to be mounted in the docker container *MUST* be located in `${PLUGIN}/mnt/${context}` , e.g.: `java/mnt/api`
- Source code used per context *MUST* follow a recommended standard, e.g.: PHP *MUST* follow PSR-2, JAVA *MUST* follow Google (https://google.github.io/styleguide/javaguide.html) or other widely adopted

- External libraries *MUST* not be “shipped” with the plugin and *MUST* have a *License* that allows us to use it as we see fit, e.g.: Apache 2.0 License
- Documentation *MUST* be provided
- Examples on HOW TO USE *MUST* be provided
- The init command *MUST* NOT be used directly

# License

By contributing to Athena, you agree that your contributions will be licensed under the Apache License Version 2.0 (APLv2).

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or

```
Derivative Works a copy of this License; and
```

(b) You must cause any modified files to carry prominent notices

```
stating that You changed the files; and
```

(c) You must retain, in the Source form of any Derivative Works

```
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and
```

(d) If the Work includes a "NOTICE" text file as part of its

```
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.
```

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS



Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

```
    http://www.apache.org/licenses/LICENSE-2.0
```

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

# Versioning

Releases are managed using github's release feature. We use Semantic Versioning for all the releases. Every change made to the code base will be referred to in the release notes (except for cleanups and refactorings).

# Let's create a plugin using the wizard

```
$ athena wizard start
       ___   __  __
      /   | / /_/ /_  ___  ____  ____ _
     / /| |/ __/ __ \/ _ \/ __ \/ __  /
    / ___ / /_/ / / /  __/ / / / /_/ /
   /_/  |_\__/_/ /_/\___/_/ /_/\__,_/
                              v0.4.0
  ==================================


Welcome to the plugin creator wizard.

[Plugin] what is the name of the plugin?
simple-webserver

[Plugin:simple-webserver] how many commands do you want do add?
1

[Plugin:simple-webserver] what is the name of the command #1? (Cannot contain spaces or underscore and should be in l
owercase)
start

[Command:start] what is the description?
Starts the webserver.

[Command:start] will you require a docker container (Y/n)?


[Command:start] do you want to create your own container (Y/n)?
n

[Command:start] what will be the image that you will be using, e.g.: debian:jessie, ubuntu:latest, php:7.0-apache, et
c...
php:7.0-apache

[Command:start] will the container run as a daemon(y/N)?
y

[Command:start] will you execute some tasks after the container starts/executes (y/N)?


[Command:start] how many mandatory arguments does it have *(0)?
2

[Command:start] what is the name of the argument #1?
source directory

[Command:start] what is the name of the argument #2?
port

[Command:start] do you want to save (Y/n)?


[OK] Command 'start' was created for plugin 'simple-webserver'.

[INFO] To use it execute "athena simple-webserver start"
```

# Using the plugin for the first time

```
$ athena simple-webserver
       ___   __  __
      /   | / /_/ /_  ___  ____  ____ _
     / /| |/ __/ __ \/ _ \/ __ \/ __  /
```

```
    / ___ / /_/ / / /  __/ / / / /_/ /
   /_/  |_\__/_/ /_/\___/_/ /_/\__,_/
                              v0.4.0
  ==================================
  [simple-webserver v1.0.0]
  ----------------------------------

[INFO] Athena plugin 'simple-webserver' has not been initialized!
[INFO] Athena plugin 'simple-webserver' is initialising...

usage: athena simple-webserver <command> [arg...]

These are the available commands for plugin [simple-webserver]:
        start  Starts the webserver.
```

# Execute the start command without parameters

```
$ athena simple-webserver start
       ___   __  __
      /   | / /_/ /_  ___  ____  ____ _
     / /| |/ __/ __ \/ _ \/ __ \/ __  /
    / ___ / /_/ / / /  __/ / / / /_/ /
   /_/  |_\__/_/ /_/\___/_/ /_/\__,_/
                              v0.4.0
  ==================================
  [simple-webserver v1.0.0]
  ----------------------------------

usage: athena simple-webserver start <source_directory> <port>
```

# Code generated by the wizard plus a few additions

```
CMD_DESCRIPTION="Starts the webserver."

athena.usage 2 "<source_directory> <port>"

# arguments are found below
source_directory="$(athena.path 1)"
port="$(athena.int 2)"

# clearing arguments from the stack
athena.pop_args 2

# options for container are found below
athena.plugin.use_external_container_as_daemon "php:7.0-apache"

# mounts the specified dir into the container
athena.docker.mount_dir "$source_directory" "/var/www/html"

# maps the specified port into the port 80 of the container
athena.docker.add_option "-p $port:80"
```

# Execute the command with the right parameters

```
$ athena simple-webserver start ~/tech-demo  80 --athena-dbg
       ___   __  __
      /   | / /_/ /_  ___  ____  ____ _
     / /| |/ __/ __ \/ _ \/ __ \/ __  /
    / ___ / /_/ / / /  __/ / / / /_/ /
   /_/  |_\__/_/ /_/\___/_/ /_/\__,_/
                              v0.4.0
  ==================================
```

```
  [simple-webserver v1.0.0]
  ---------------------------------

[DEBUG] starting container athena-plugin-simple-webserver-0 for command 'start' ...

[DEBUG] Time: 0 minutes and 0 seconds elapsed.
```

## Run the info command to check if the container is running

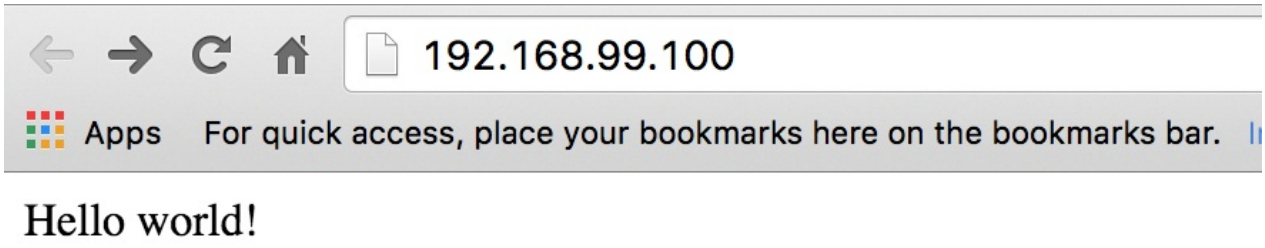
```
$ athena info
       ___   __  __
      /   | / /_/ /_  ___  ____  ____ _
     / /| |/ __/ __ \/ _ \/ __ \/ __  /
    / ___ / /_/ / / /  __/ / / / /_/ /
   /_/  |_\__/_/ /_/\___/_/ /_/\__,_/
                              v0.4.0
  ================================


[INFO] Running containers [image|container|status]:

 * php  athena-plugin-simple-webserver-0  [UP]  To access it use http://192.168.99.100:80


[INFO] No custom images were found!
```

## Now check the browser, and *voilá* it's working:




Hello world!