
Ajenti

Release 2.2.1

Nov 22, 2022

1	Feature Overview	3
	Python Module Index	65
	Index	67

Ajenti is a highly extensible platform. The core of the platform provides HTTP server, Socket engine and Plugin container. The extensibility is implemented via a system of extension plugins.

The backend is written in Python (**Ajenti Core**). The frontend is written in Angular application hosted in the core plugin **shell**.

For more information about the architecture see the *[Architecture and how it works](#)*.

1.1 HTTP Server

- HTTP 1.1 Support.
- Websockets with fallback to XHR polling.
- Fast event-loop based processing.
- Flexible routing.
- Session sandboxing.
- SSL with client certificate authentication.

1.2 Performance

- >1000 requests per second.
- 30 MB RAM footprint + 5 MB per session.

1.3 API

- Highly modular Python API. Everything is a module and can be removed or replaced.
- Builtin webserver API supports routing, file downloads, GZIP, websockets and more.
- Transparent SSL client authorization.
- Plugin architecture
- Dependency injection
- Server-side push and socket APIs.

1.4 Security

- Pluggable authentication and authorization.
- Stock authenticators: UNIX account, password, SSL client certificate and Mozilla Persona E-mail authentication.
- Unprivileged sessions isolated in separate processes.
- Fail2ban rule

1.5 Frontend

- Clean, modern and responsive UI. Single-page, no reloads.
- Live data updates and streaming with Socket.IO support.
- Full mobile and tablet support.
- LESS support.
- Numerous stock directives.
- Angular framework

1.6 Platforms

- Debian 9 or later
- Ubuntu Bionic or later
- RHEL 8 or later
- Can be run on other Linux or BSD systems with minimal modifications.
- Supports Python 3.5+.

1.6.1 Installing

Caution: Supported operating systems:

- Debian 9 or later
- Ubuntu Bionic or later
- RHEL 8 or later

Other Linux-based systems *might* work, but you'll have to use manual installation method.

Automatic Installation

```
curl https://raw.githubusercontent.com/ajenti/ajenti/master/scripts/install.sh | sudo ↵  
↵bash -s -
```

Automatic Installation in virtual environment

Caution: Please note that this install method is still under tests. Ajenti starts successfully on the previously mentioned supported operating systems, but all functionalities were not tested. Be kind to report any problem with this install method as issue here : <https://github.com/ajenti/ajenti/issues>

```
curl https://raw.githubusercontent.com/ajenti/ajenti/master/scripts/install-venv.sh | 
↪ sudo bash -s -
```

Manual Installation

Native dependencies: Debian/Ubuntu

Enable Universe repository (Ubuntu only):

```
sudo add-apt-repository universe
```

```
sudo apt-get install build-essential python3-pip python3-dev python3-lxml libssl-dev 
↪ python3-dbus python3-augeas python3-apt ntpdate
```

Native dependencies: RHEL

Enable EPEL repository:

```
sudo dnf install epel-release
```

```
sudo dnf install -y gcc python3-devel python3-pip python3-pillow python3-augeas 
↪ python3-dbus chrony openssl-devel redhat-lsb-core
```

Install Ajenti 2

Upgrade PIP:

```
sudo pip3 install setuptools pip wheel -U
```

Minimal install:

```
sudo pip3 install ajenti-panel ajenti.plugin.core ajenti.plugin.dashboard ajenti.
↪ plugin.settings ajenti.plugin.plugins
```

With all plugins:

```
sudo pip3 install ajenti-panel ajenti.plugin.ace ajenti.plugin.augeas ajenti.plugin.
↪ auth-users ajenti.plugin.core ajenti.plugin.dashboard ajenti.plugin.datetime ajenti.
↪ plugin.filemanager ajenti.plugin.filesystem ajenti.plugin.network ajenti.plugin.
↪ notepad ajenti.plugin.packages ajenti.plugin.passwd ajenti.plugin.plugins ajenti.
↪ plugin.power ajenti.plugin.services ajenti.plugin.settings ajenti.plugin.terminal
```

Uninstall Ajenti 2

Ajenti is a collection of Python modules installed with pip, delivered with an init script (systemd or sysvinit). So it's necessary to remove the init script, then the Python librairies, and the configurations files.

Systemd

```
sudo systemctl stop ajenti.service
sudo systemctl disable ajenti.service
sudo systemctl daemon-reload
sudo rm -f /lib/systemd/system/ajenti.service
```

SysVinit

```
/etc/init.d/ajenti stop
rm -f /etc/init/ajenti.conf
```

Python3 modules

List all modules from Ajenti:

```
sudo pip3 list | grep aj
```

The result should be something like (eventually more or less plugins):

```
aj 2.1.43
ajenti-panel 2.1.43
ajenti.plugin.ace 0.30
ajenti.plugin.auth-users 0.31
ajenti.plugin.core 0.99
ajenti.plugin.dashboard 0.39
ajenti.plugin.filesystem 0.47
ajenti.plugin.passwd 0.24
ajenti.plugin.plugins 0.47
ajenti.plugin.session-list 0.4
ajenti.plugin.settings 0.30
```

Then simply remove all these modules:

```
sudo pip3 uninstall -y aj ajenti-panel ajenti.plugin.ace ajenti.plugin.auth-users_
↪ajenti.plugin.core ajenti.plugin.dashboard ajenti.plugin.filesystem ajenti.plugin.
↪passwd ajenti.plugin.plugins ajenti.plugin.session-list ajenti.plugin.settings
```

Configuration files

If you don't need it for later, just delete the directory */etc/ajenti/*:

```
sudo rm -rf /etc/ajenti/
```

1.6.2 Running Ajenti

Starting service

The automatic install script provides binary *ajenti-panel* and *initscript/job/unit ajenti*. You can ensure the service is running:

```
service ajenti restart
```

or:

```
/etc/init.d/ajenti restart
```

or:

```
systemctl restart ajenti
```

The panel will be available on **HTTPS** port **8000** by default. The default username is **root**, and the password is your system's root password.

Ajenti can also be run in a verbose debug mode:

```
ajenti-panel -v
```

Commandline options

- `-c, --config <file>` - Use given config file instead of default
- `-v` - Debug/verbose logging
- `--log <level>` - Fix log level : debug, info, warning or error
- `--dev` - Enables automatic resources build on each request
- `-d, --daemon` - Run in background (daemon mode)
- `--stock-plugins` - Run with provided plugins (default if option `--plugins` is not used)
- `--plugins <dir>` - Run with additional plugins
- `--autologin` - Will automatically log in the user under which the panel runs. **This is a security issue if your system is public.**

Debugging

If Ajenti does not start as intended, there are various ways to debug this, but it is good to know that the problem can have an origin in Python code or in Javascript code.

Debug Python problems

First of all, have a look at:

```
/var/log/ajenti/ajenti.log
```

It may contain some running errors which could be useful to understand the problem.

The traceback of a total crash would be stored in:

```
/var/log/ajenti/crash-DATE.log
```

If this log files do not provide enough informations, you can manually start Ajenti in debug mode as root:

```
systemctl stop ajenti
/usr/local/bin/ajenti-panel -v
```

This will increase the verbosity of Ajenti in `/var/log/ajenti/ajenti.log`, but you can also directly follow the progress of Ajenti start with:

```
systemctl stop ajenti
/usr/local/bin/ajenti-panel --dev
```

and then stop it as usual with `Ctrl + C`. Don't forget after this to restart the Ajenti process if necessary:

```
systemctl start ajenti
```

Debug Javascript problems

The best way to do it is to launch the developer tools in your browser, usually with `F12`, and to look if some errors are shown.

Submit the errors

The best way to help the development of Ajenti is then to submit the errors at <https://github.com/ajenti/ajenti/issues/new> with all informations (`traceback`, `OS`, `Python version`, ...).

1.6.3 Configuration files

All the configuration files are store in `/etc/ajenti`:

- **config.yml**: the main configuration file with all important parameters,
- **smtp.yml**: credentials to an email server relay, if you want to use some mail notifications or reset password functionality,
- **users.yml**: the default file which contains user account for the user authentication provider.

All configuration files use the [yaml format](#)

config.yml in details

Ajenti will use the following parameters :

auth block

```
auth:
  allow_sudo: true
  emails: {}
  provider: os
  users_file: /etc/ajenti/users.yml
```

Explanations:

- **allow_sudo:** **true** or **false** (allow users in the sudo group to elevate)
- **emails:** {} (not currently used)
- **provider:** authentication method to use, **os** (users from the os) or **users**
- **users_file:** if the users authentication provider is used, path to the users file (default **/etc/ajenti/users.yml**)

The parameter **user_config** was used to specified where the user configuration was stored, but is now deprecated, since it's bound to the **provider** (**os** or **users**) to avoid duplicates entries.

bind block

```
bind:
  host: 0.0.0.0
  mode: tcp
  port: 8000
```

Explanations:

- **host:** ip on which to listen (default **0.0.0.0**)
- **mode:** type of socket, **tcp** or **unix**
- **port:** port on which to listen, default **8000**

ssl block

```
ssl:
  enable: true
  certificate: /etc/ajenti/mycert.pem
  fqdn_certificate: /etc/letsencrypt/ajenti.pem
  force: false
  client_auth:
    enable: true
    force: true
    certificates:
      digest: ↪15:E8:5E:E5:D2:E8:75:0D:53:FF:22:A8:79:28:E5:BE:33:E0:37:07:FB:31:47:4D:61:69:AB:43:F8:5B:23:78
      name: C=NA,ST=NA,O=sajenti.mydomain.com,CN=root@ajenti.mydomain.com
      serial: 352674123960898230347891590646542168839110009016
      user: root
```

Explanations:

- **enable:** **true** or **false** to provide support for https. It's highly recommended to set it to **true**
- **certificate:** full path to default global certificate, used to generate client certificates, and for the https protocol, if the parameter **fqdn_certificate** is not set. The PEM file should contains the certificate itself, and the private key.
- **fqdn_certificate:** full path certificate for your FQDN (e.g. **/etc/ajenti/mycert.pem**). The PEM file should contains the certificate itself, and the private key.
- **force:** spawn a small listener on port 80 to enable a redirect from **http://hostname** to **https://hostname:port**.

- **client_auth:**

- **enable:** **true** or **false** to enable client authentication via certificates
- **force:** if **true**, only allows login with client certificate. If **false**, also permit authentication with password
- **certificates:** this entry contains all client certificates for an automatic login. It will be filled through the settings in [certificates](#)
 - * **digest:** digest of the certificate
 - * **name:** name of the certificate
 - * **serial:** serial of the certificate
 - * **user:** username

email block

```
email:
  enable: true
  templates:
    reset_email : /etc/ajenti/email/mytemplate_for_reset_password.html
```

Explanations:

- **enable:** **true** or **false**, if you want to enable the password reset function. But for this you need to set the smtp credentials in `/etc/ajenti/smtp.yml`
- **templates:** * **reset_email:** full path to template email for reset password functionality

The default template used to reset email password is located [here](#). The variables are automatically filled with jinja2.

Other global parameters

```
color: blue
language: en
logo: /srv/dev/ajenti/ajenti-panel/aj/static/images/Logo.png
max_sessions: 10
name: ajenti.mydomain.com
restricted_user: nobody
session_max_time: 1200
```

Explanations:

- **color:** secondary color of the CSS theme (possibles values are **default**, **bluegrey**, **red**, **deeporange**, **orange**, **green**, **teal**, **blue** and **purple**)
- **language:** language preference for all users, default **en**
- **logo:** full path to your own logo, default is [the one from Ajenti](#)
- **max_sessions:** max number of simultaneously sessions, default is **99**. If the max is reached, the older inactive session will be deactivated
- **name:** your domain name

- **restricted_user**: user to use for the restricted functionalities, like for the login page. It's an important security parameter in order to limit the actions in restricted environments : all actions in restricted environments will be done with this user's privileges. Default is **nobody**.
- **session_max_time**: max validity time in seconds before automatic logout. Default is **3600** (one hour).
- **trusted_domains** (*Ajenti* >= 2.2.1) : comma separated list of trusted domains under which it's possible to reach your *Ajenti* server. When the HTTP headers are tested, a valid origin will be considered as one of the domains listed. It's necessary to specify the protocol. It's mean that an entry should look like *http://my.domain.com*.
- **trusted_proxies** (*Ajenti* >= 2.2.1) : comma separated list of trusted proxies. This is actually used in order to get the real ip of the client.

smtp.yml in details

This file contains all the credentials of an email server which can be used as email relay to send some notifications, like an email to reset a forgotten password.

```
smtp:
  password: MyVeryStrongStrongPassword
  port: starttls
  server: mail.mydomain.com
  user: mail@mydomain.com
```

Explanations:

- **port**: **starttls** (will use 587) or **ssl** (will use 465)
- **server**: server hostname, like *mail.mydomain.com*
- **user**: user to authenticate
- **password**: password of the mail user

users.yml in details

Ajenti gives the possibility to use two authentication methods : **os** or **users**. If **users** is used, all user informations are stored in **users_file**. It's automatically filled with the user plugin.

The default path for the **users_file** is */etc/ajenti/users.yml* with following structure:

```
users:
  arnaud:
    email: arnaud@mydomain.com
    fs_root: /home/arnaud
    password: 73637279707.....
    permissions:
      packages:install: false
      sidebar:view:/view/cron: false
    uid: 1002
```

Explanations:

- **password**: hash of the password
- **permissions**: list of permissions of the user
- **uid**: related os uid to run the worker on
- **fs_root**: root directory

- **email**: email to use for password reset.

1.6.4 Securing

Fail2ban

Failed login attempts are logged in `/var/log/ajenti/ajenti.log`. A basic filter for Fail2ban is available here : <https://raw.githubusercontent.com/ajenti/ajenti/master/scripts/ajenti.conf>

You can enable it by copying it in `/etc/fail2ban/filter.d/ajenti.conf` and with the following lines in `/etc/fail2ban/jail.d/ajenti` :

```
[ajenti]
enabled = true
port    = 8000
bantime = 120
maxretry = 3
findtime = 60
logpath = /var/log/ajenti/ajenti.log
filter = ajenti
```

This is only an example : after 3 failed attempts (*maxretry*) the last 60 seconds (*findtime*), the found ip will be banned 2 minutes (*bantime*). You can naturally set other values related to your configuration.

1.6.5 Contributing to Ajenti

Translations

All translations are stored by [Crowdin](#), and any help is welcome. It's possible to translate directly all strings in the great interface of Crowdin and then we can include and compile it into the next release:

[Ajenti on Crowdin](#)

Testing

It's always good to have some users feedback, because we may oversee some problems. If you find an issue, please post it on [GitHub](#) with a complete description of the problem, and we will try to solve it and improve Ajenti.

Developping

There's two main axes to develop Ajenti :

- *Extension plugins*: like e.g. a plugin to manage the fstab file,
- *Core*: improve Ajenti on server side.

1.6.6 Plugin check_certificates

You can see with one look if your SSL certificates are still valid or not.

The screenshot shows the 'Ajenti Check certificates' interface. On the left is a sidebar with a menu containing 'GENERAL' (Dashboard, Plugins, Settings, Users) and 'TOOLS' (Check certificates, File Manager, Notepad, Session list, Terminal). Below these are 'SOFTWARE' options (Docker, Services). The main area displays a table of certificates:

Domain	Port	Issuer	End	Status
[redacted]	8006		Can not handle SSL on this port !	✖
[green]	443	Let's Encrypt	2022-06-04 01:03:02	✓
[green]	443	Let's Encrypt	2022-05-08 22:45:40	✓
[green]	443	Let's Encrypt	2022-06-04 01:03:02	✓
[red]	8000		Host refuse the connection !	✖
[green]	443	Let's Encrypt	2022-05-25 03:49:23	✓

Below the table is an 'Add host' button.

The list view let you see the hostname, the port, the issuer of the certificate, the end of the certificate, and the status of the connection.

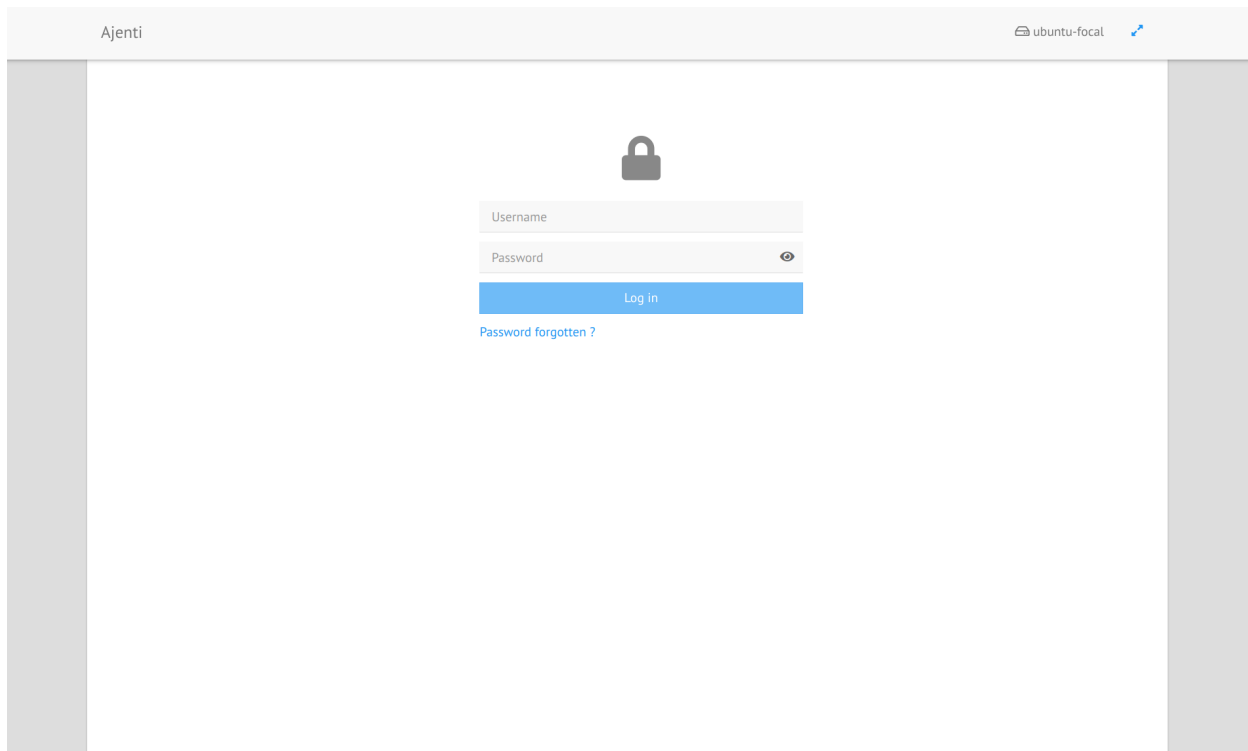
It's pretty easy to add or to remove an hostname. By default, a test will be done on port 443, the standard one for HTTPS. But you can naturally specify something else, like 8000 or 587.

If the port 587 is specified, Ajenti will try to open a STARTTLS connection, e.g. for email server.

This screenshot shows the same interface as before, but with a 'New hostname' dialog box open in the center. The dialog has two input fields: 'URL' with the value 'mail.' and 'Port (default is 443 if empty)' with the value '587'. At the bottom right of the dialog are 'ADD' and 'CANCEL' buttons. The background interface is dimmed.

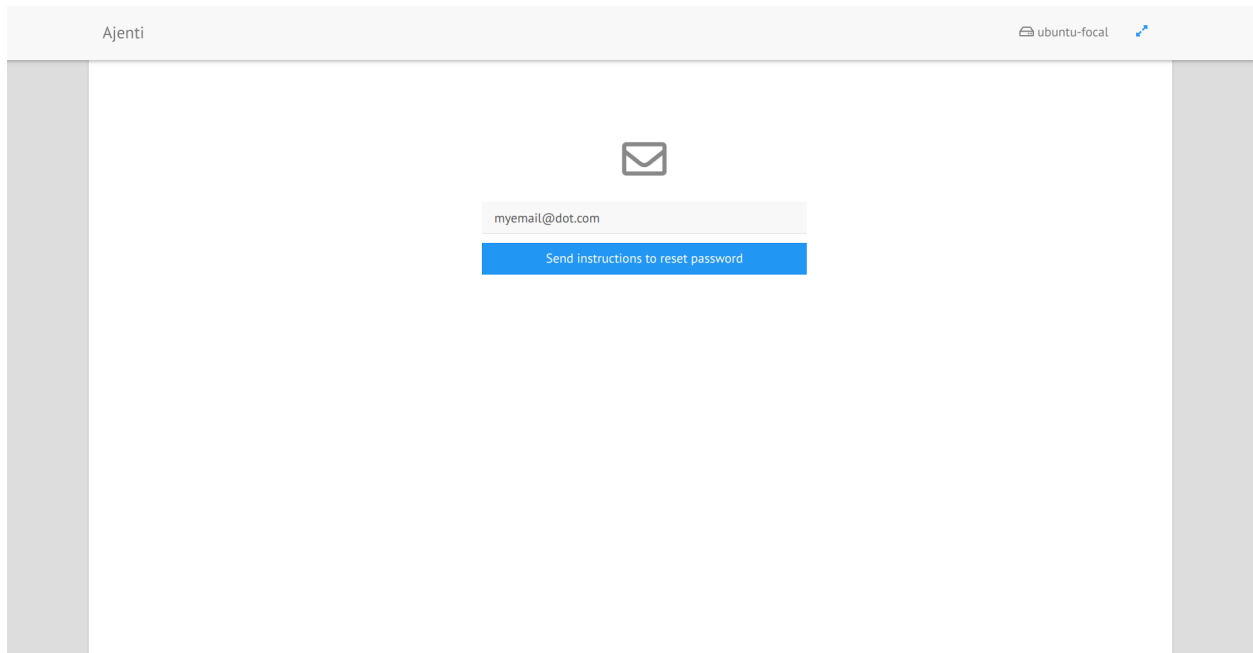
1.6.7 Plugin core

The main plugin of Ajenti is the core plugin.



This plugin manages:

- the authentication process,
- user environment setup,
- session management,
- the way the resources are delivered (CSS, JS, etc ...),
- the main template and the main style of Ajenti,
- the entries in the sidebar,
- error handling,
- password reset,
- configurations (Ajenti, user config).

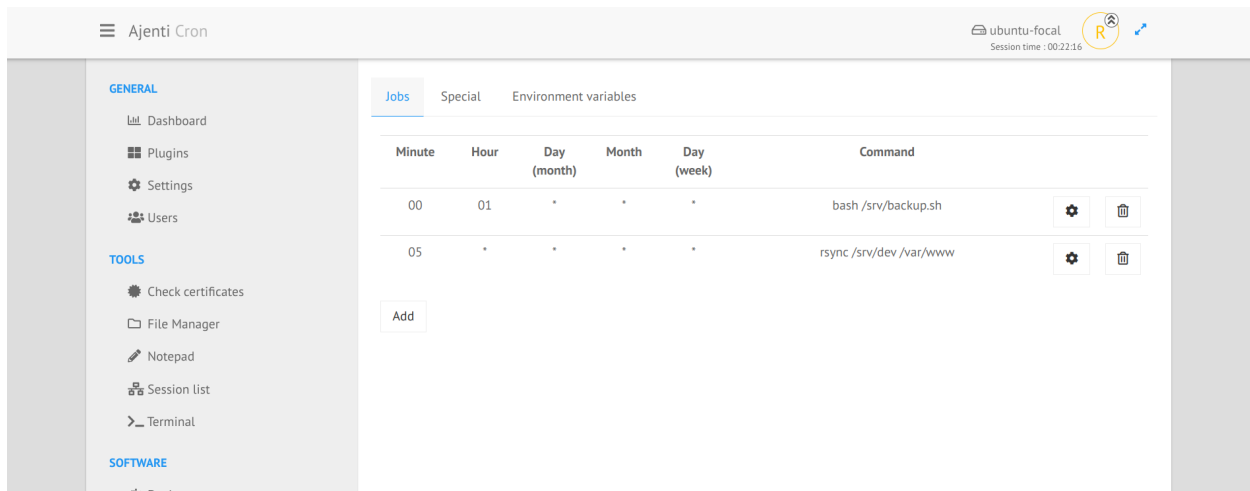


It delivers a lot of tools, services, components for the other plugins too:

- hotkeys,
- tasks,
- pushes,
- dialogs,
- progress spinner,
- navbox,
- messagebox,
- smartprogerss,
- customization,
- translations with gettext,
- notifications,
- socketio.

1.6.8 Plugin cron

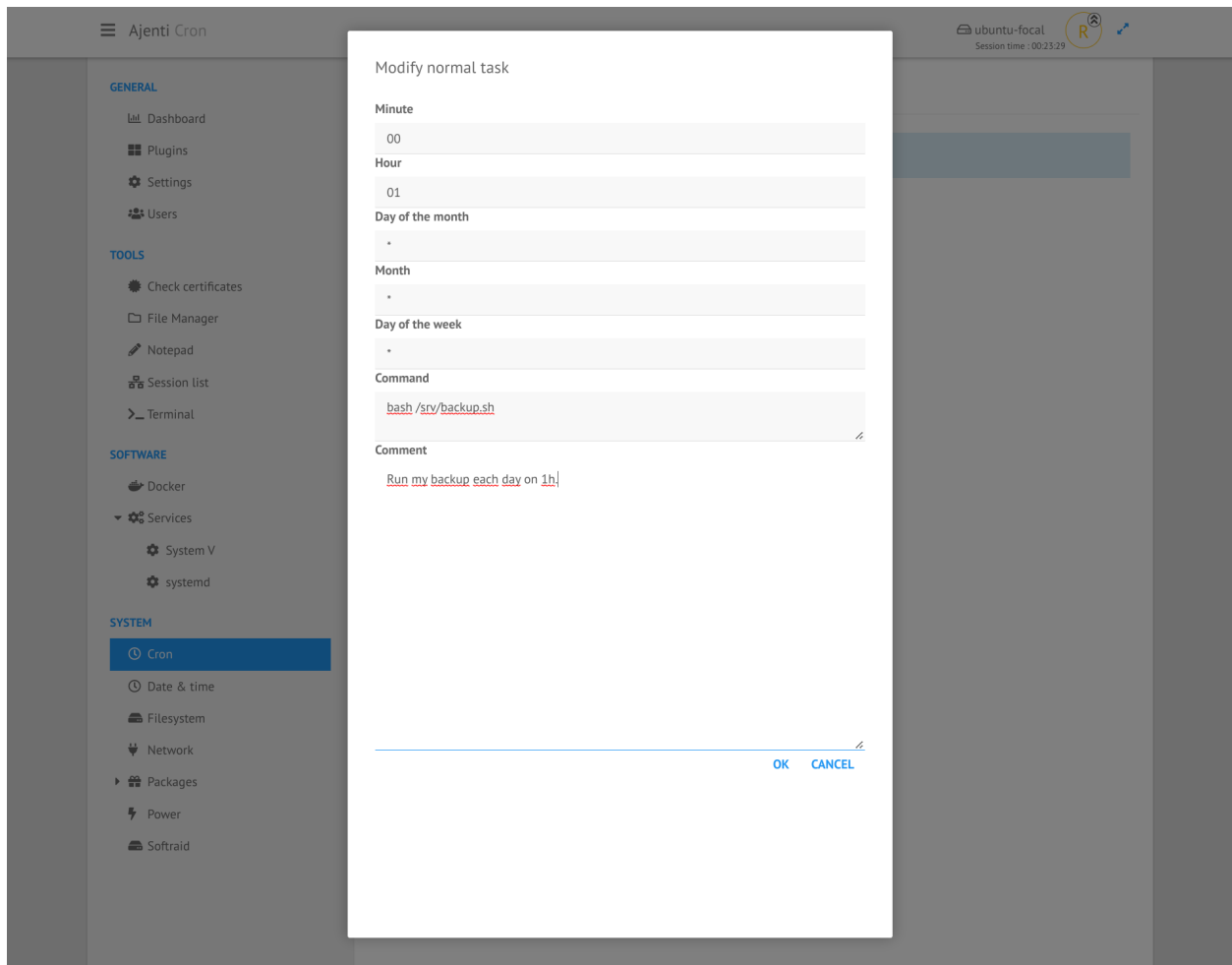
This plugin allows to handle all entries in a personal `cron` file.



This is quite equivalent as running `crontab -l -u USER` to manage your own cronjobs.

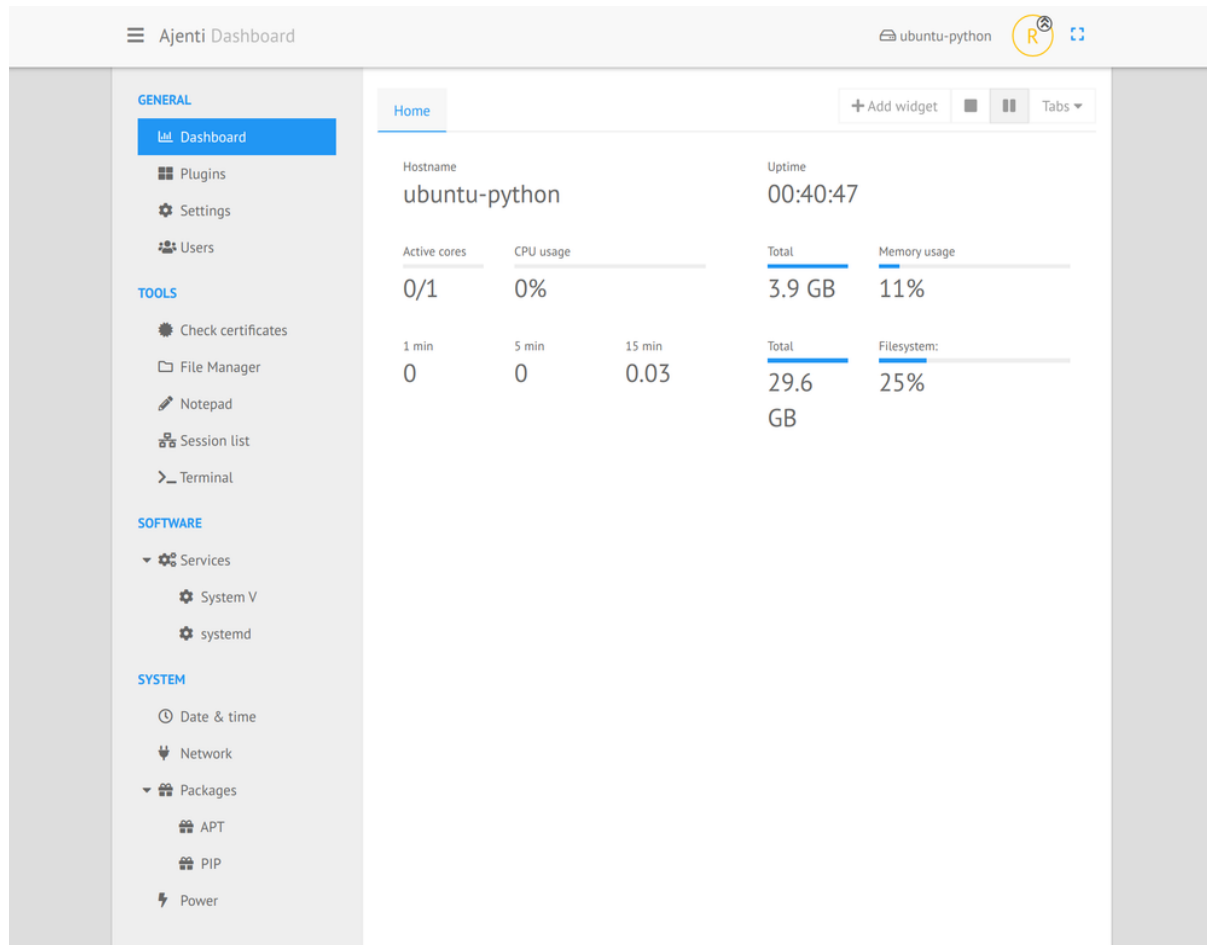
With this plugin, you can:

- add jobs,
- remove jobs,
- edit jobs,
- edit special entries (@yearly, etc ...),
- set environment variables,
- add comments.



1.6.9 Plugin dashboard

This is the default landing page after successfully authenticate.



It's possible to display the widgets of your choice, and to order them as you want with a simple drag&drop.

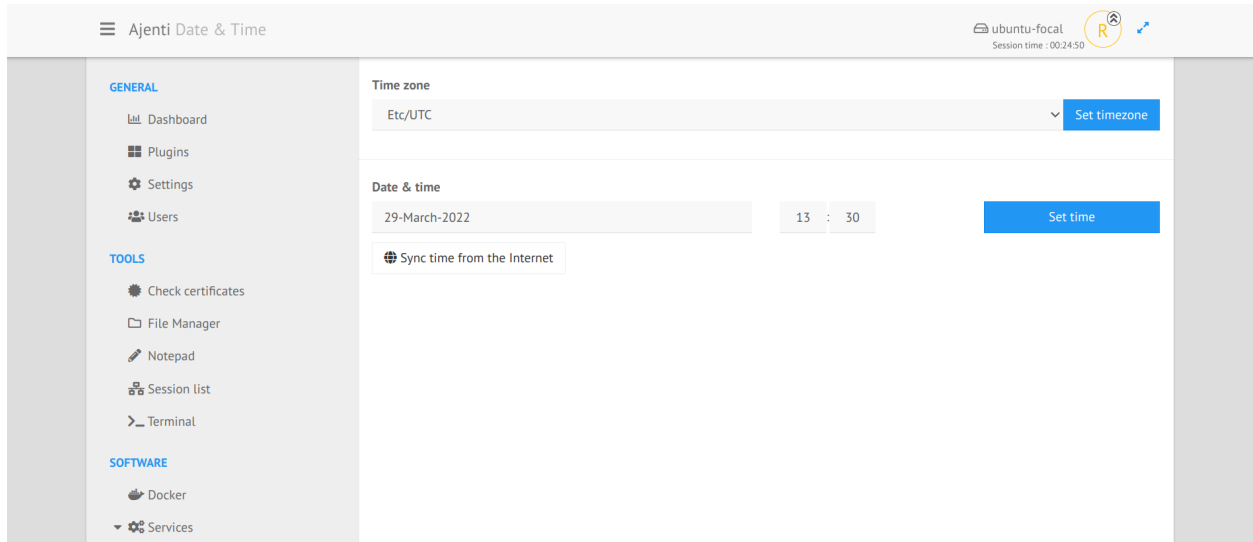
You can also add other tabs, and rename them the way you want.

The list of actual available widgets:

- Check certificates,
- CPU Usage,
- Disk space (you can choose the mount point),
- Hostname,
- Load average,
- Memory usage,
- Power state,
- Script (run your own command),
- Service (status of a service in systemd or sysv init),
- Sessions (logged in users),
- Traffic,
- Uptime.

1.6.10 Plugin datetime

This plugin displays the current time zone used, and time and date set on the server.

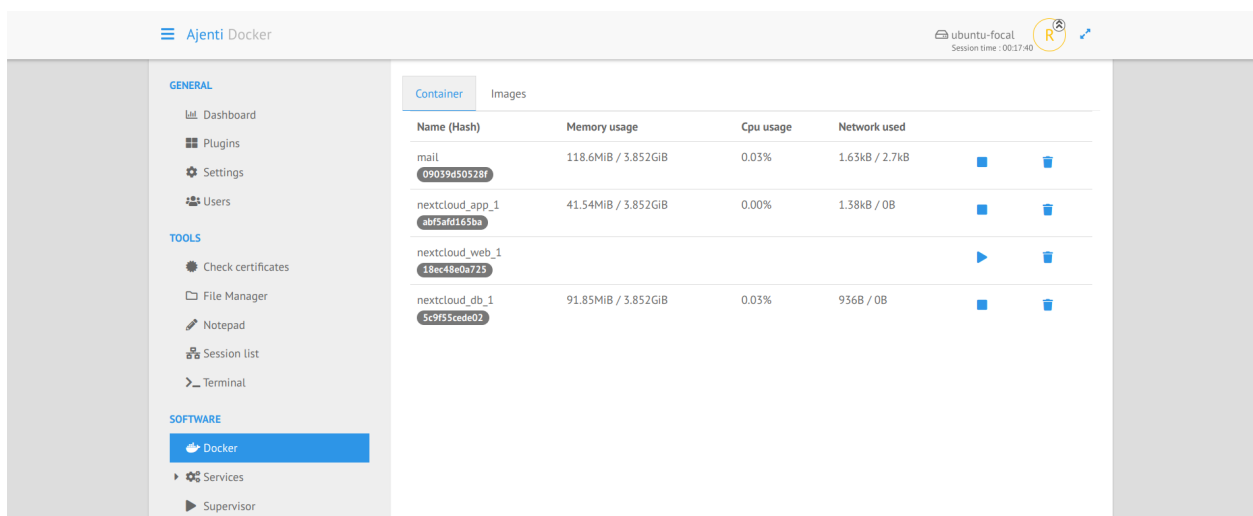


It's possible to:

- change the time zone used,
- set the time on the server,
- synchronize time using NTP (package `ntpdate` is for this necessary).

1.6.11 Plugin docker

This plugin allows to show all running containers and images from a locally docker instance.



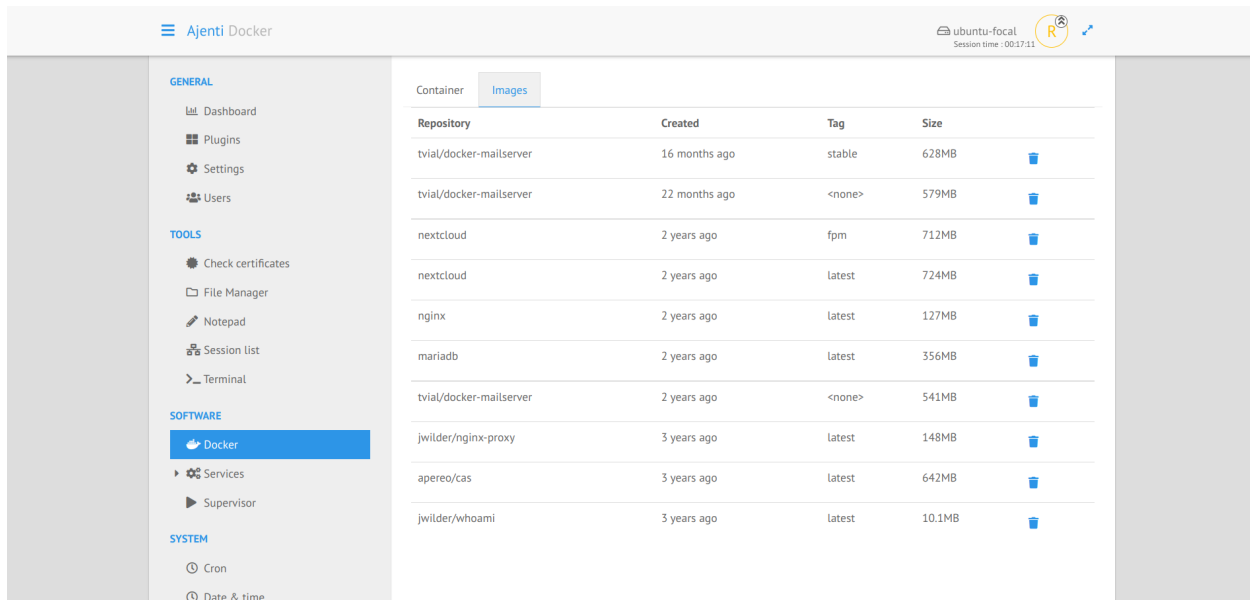
The default tab shows all containers, with their names and id, and you can:

- start/stop a container,
- remove a container,

- see memory usage, cpu usage and network I/O

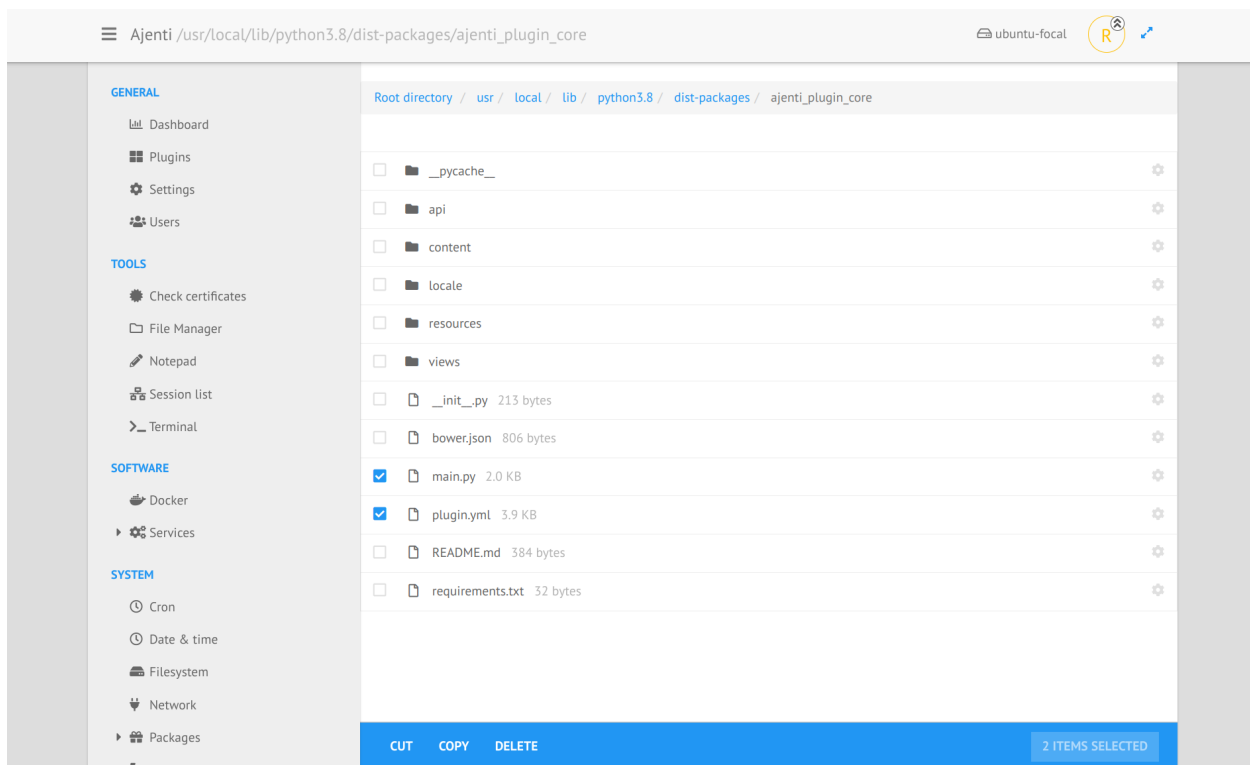
On the second tab, you will see the stored images with their sizes.

You can easily choose which one you want to delete.



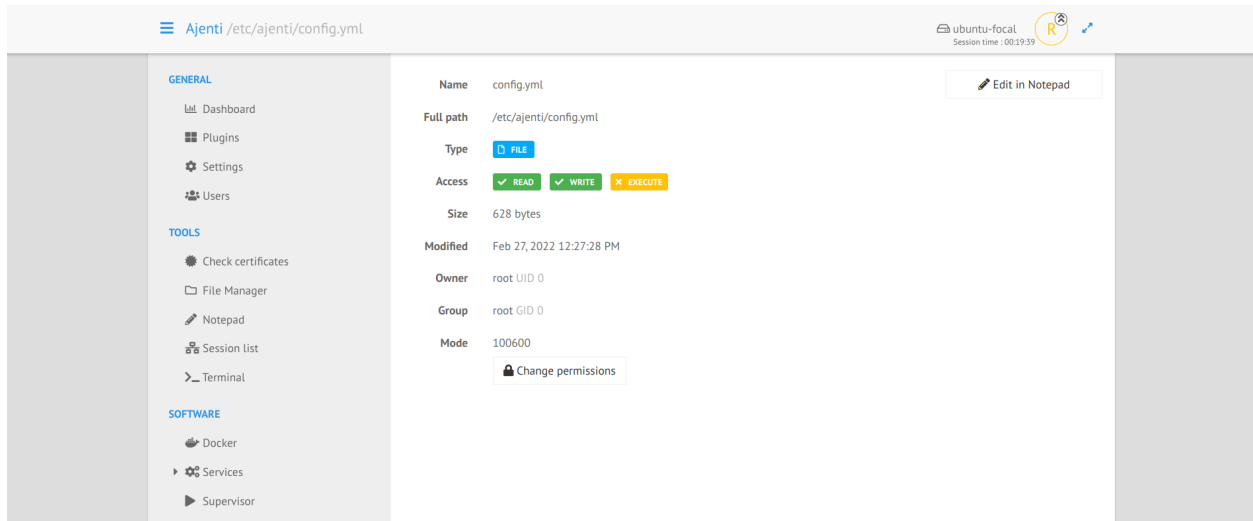
1.6.12 Plugin filemanager

This plugin let you navigate on the server filesystem and perform all common operations on files and directories.



Currently, it's possible to:

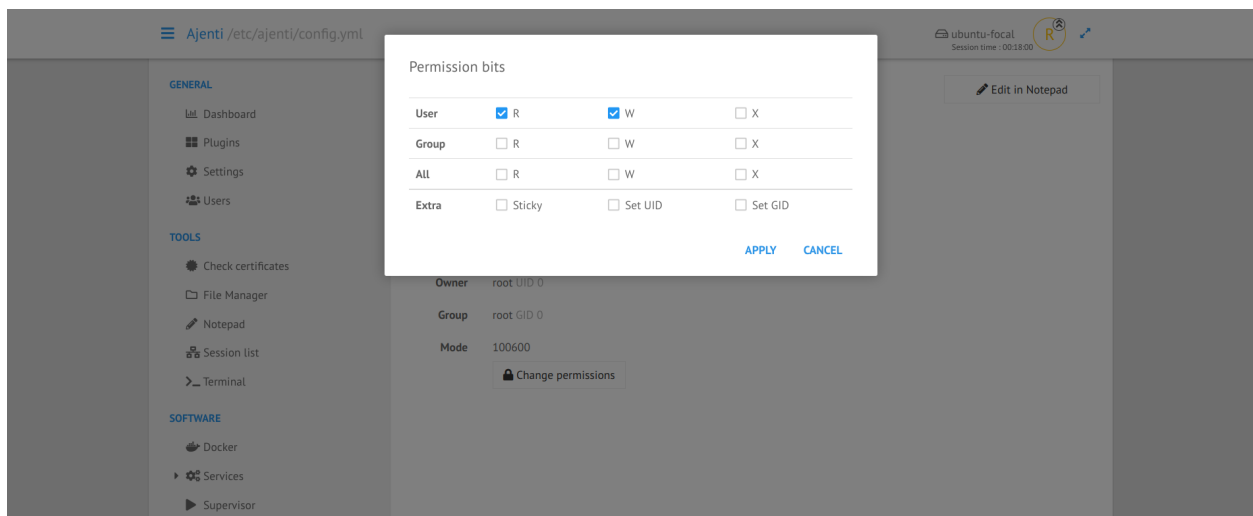
- create new files, new directories,
- upload a file through drag&drop,
- navigate in many tabs,
- cut, copy, delete files and directories (you must first select at least one object),
- display the properties of an object,
- easily navigate between directory with the breadcrumb.



In the properties view you will see all common informations (permissions, last change date, owner, etc ... the same as the command `stat`).

If the file is plain text, a button `Edit in Notepad` will appear and let you modify the file.

You can also change the permissions of the file:



1.6.13 Plugin fstab

The first tab shows the output of the `mount` command with some util informations like:

- filesystem type,
- mountpoint,
- used space,
- total size.

The button on the right let you unmount the desired device, but you should use it with caution (don't try to unmount the root fs!).

Ajenti Fstab

ubuntu-focal
Session time : 00:24:41

GENERAL

Dashboard

Plugins

Settings

Users

TOOLS

Check certificates

File Manager

Notepad

Session list

Terminal

SOFTWARE

Docker

Services

System V

systemd

SYSTEM

Cron

Date & time

Filesystem

Network

Packages

Power

Softraid

Mounted

Fstab

Device	FS Type	Mountpoint	Used	Size	Usage	
udev	devtmpfs	/dev	0 bytes	1.9 GB		
tmpfs	tmpfs	/run	1016.0 KB	393.6 MB		
/dev/sda2	ext4	/	7.4 GB	14.7 GB		
tmpfs	tmpfs	/dev/shm	0 bytes	1.9 GB		
tmpfs	tmpfs	/run/lock	0 bytes	5.0 MB		
tmpfs	tmpfs	/sys/fs/cgroup	0 bytes	1.9 GB		
/dev/loop0	squashfs	/snap/core18/2344	55.6 MB	55.6 MB		
/dev/loop4	squashfs	/snap/core18/2128	55.5 MB	55.5 MB		
/dev/loop2	squashfs	/snap/core20/1376	62.0 MB	62.0 MB		
/dev/loop6	squashfs	/snap/txd/22754	78.9 MB	78.9 MB		
/dev/loop3	squashfs	/snap/snapd/15177	43.8 MB	43.8 MB		
/dev/loop5	squashfs	/snap/txd/21624	73.3 MB	73.3 MB		
tmpfs	tmpfs	/run/user/0	0 bytes	393.6 MB		
/dev/loop7	squashfs	/snap/core20/1405	62.0 MB	62.0 MB		

The second tab lists all entries in `/etc/fstab` and let you add/modify or delete the entries.
But you should also be careful here with what you are doing.

Ajenti Fstab

ubuntu-focal
Session time : 00:24:02

GENERAL

Dashboard

Plugins

Settings

Users

TOOLS

Check certificates

File Manager

Notepad

Session list

Mounted

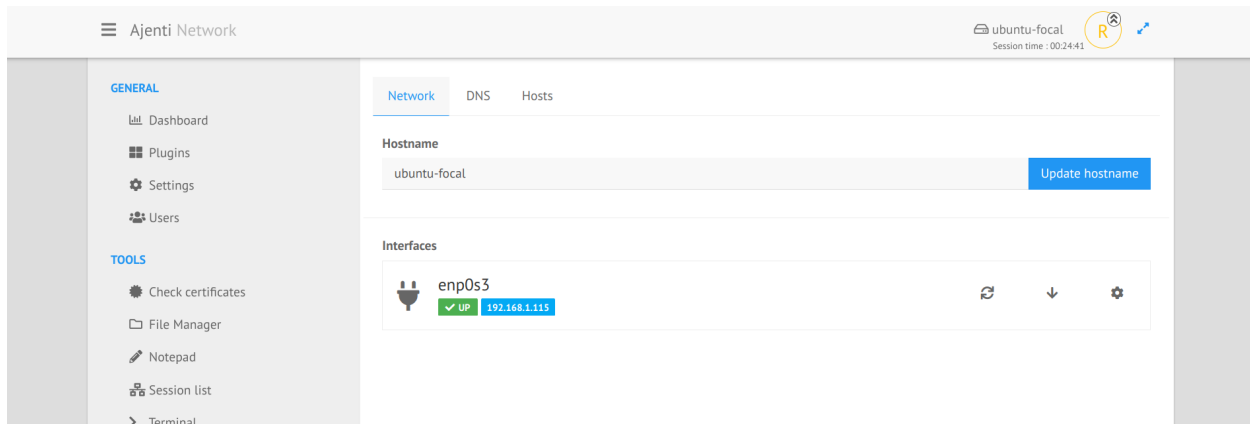
Fstab

Device	Mountpoint	Type		
/dev/disk/by-uuid/01d07049-8a6b-40ea-a5e4-435b56ca4f22	/	ext4		
/swap.img	none	swap		

Add

1.6.14 Plugin network

This plugin contains the utilities to show the most important informations about your network interfaces.



Tab network

You will see all network interfaces, their IP and status. It's possible to bring an interface up or down and change some of their properties (not yet implemented for systems running with `netplan`). It's also possible to update the hostname name.

Tab DNS

This tab enable DNS management (add or delete DNS server).

Tab Hosts

Lists all entries in the file `/etc/hosts`, and modify or delete any single of them.

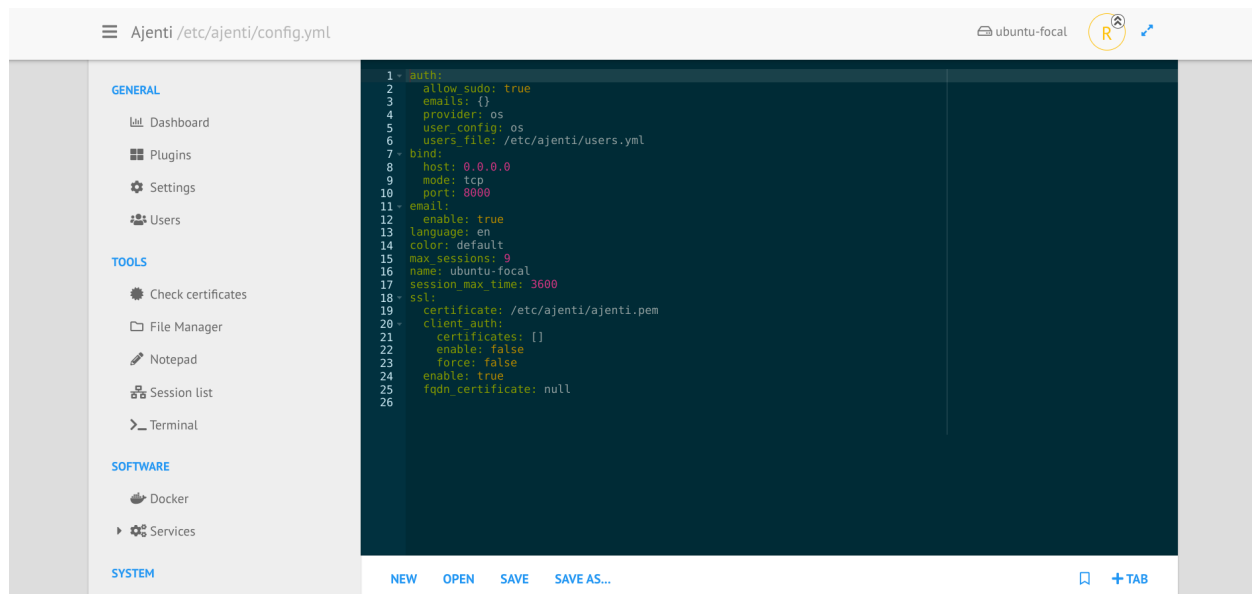
1.6.15 Plugin notepad

Based on the [ACE editor](#), you can:

- edit all plain text files,
- create a new file,
- save an existing file in another location,
- manage all of these files with tabs.

Hotkey:

- Ctrl + O : open file
- Ctrl + N : new file
- Ctrl + S : save file



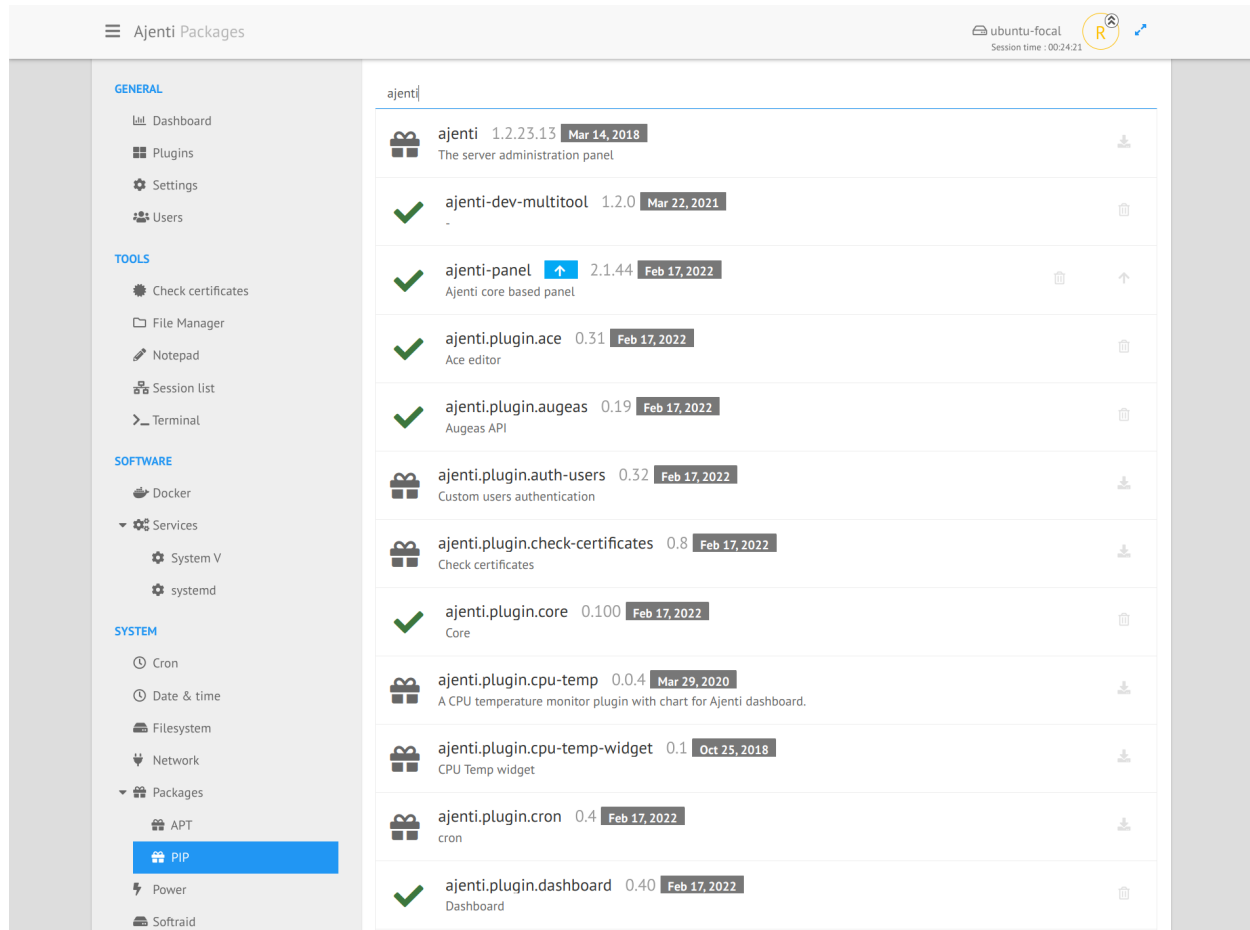
1.6.16 Plugin packages

In order to manage the packages installed on your server, the plugin packages provides a quick search to filter the packages matching the search query.

Actually, the supported package engines are APT and PIP.

It's necessary to enter at least 3 chars in the search to automatically get a packages list, and then perform usual operations:

- see if a package is installed,
- see the version,
- see if a newer version is available,
- install/update a package,
- remove a package.



1.6.17 Plugin plugins

Ajenti is pretty flexible and allow anyone to write its own plugin (backend Python and frontend AngularJS).

In order to manage all plugins and their versions, the plugin `plugins` lists all available plugins, shows if they are installed, or if an update if published.

The main plugin `core` can not be uninstalled, because Ajenti can not run without it, but you can check whenever a new version is available.

Updating or removing a plugin is this way pretty easy.

Ajeti Plugins

ubuntu-python

GENERAL

Dashboard

Plugins

Settings

Users

TOOLS

Check certificates

File Manager

Notepad

Session list

Terminal

SOFTWARE

Services

System V

systemd

SYSTEM

Date & time

Network

Packages

APT

PIP

Power

Core

Ajeti core 2.1.34, no upgrades available.

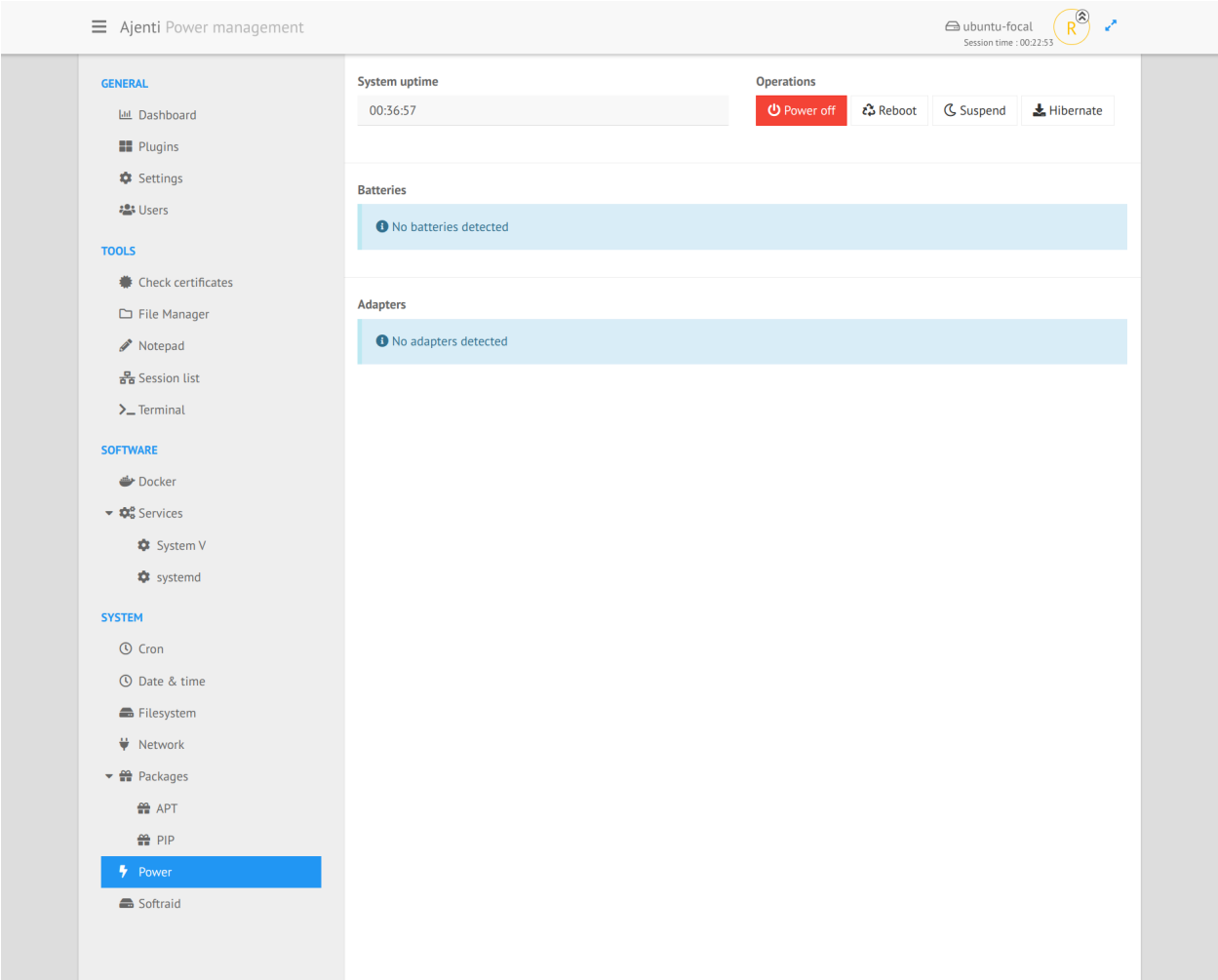
Installed plugins

Ace editor	ace 0.26	
Augeas API	augeas 0.16	
Check certificates	check_certificates 0.3	
Core	core 0.94	
Custom users authentication	auth_users 0.29	
Dashboard	dashboard 0.37	
Date & time	datetime 0.37	
File Manager	filemanager 0.25	
Filesystem API	filesystem 0.44	
Network	network 0.22	
Notepad	notepad 0.25	
Packages	packages 0.31	
Plugins	plugins 0.45	
Power management	power 0.20	
Services	services 0.28	
Session list	session_list 0.1	

1.6.18 Plugin power

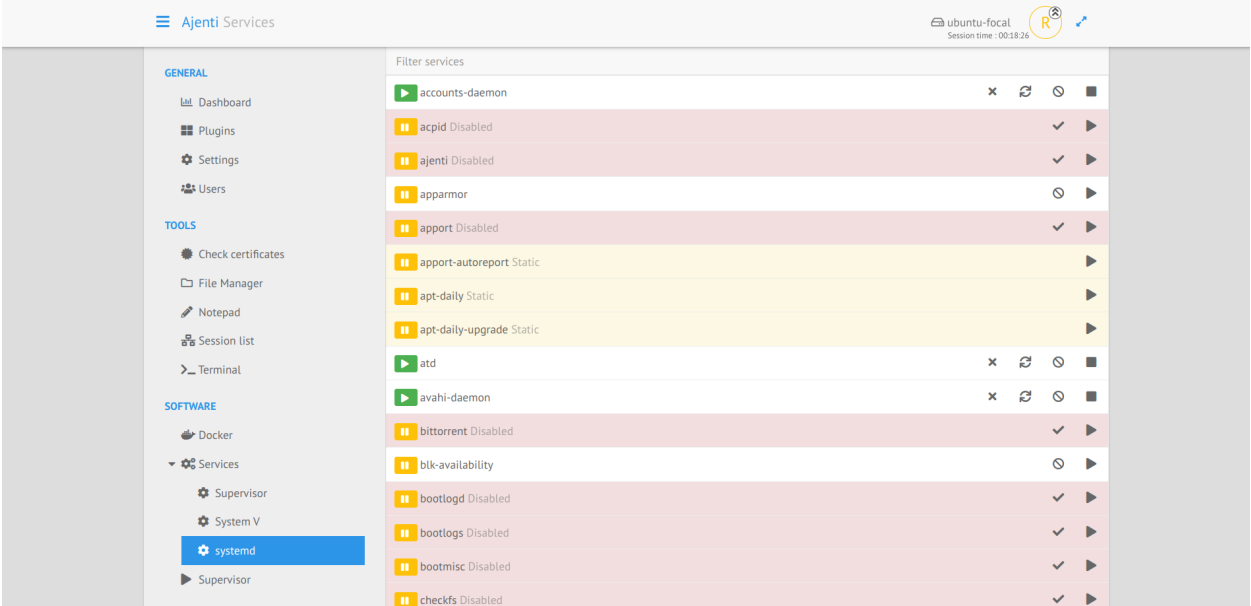
Basically handle all around power management on your server.

Uptime appears, and you can also reboot or shutdown the server if needed.



1.6.19 Plugin services

The plugin services shows the status of services in systemd or in system V init.



For the `systemd` unit services, you can:

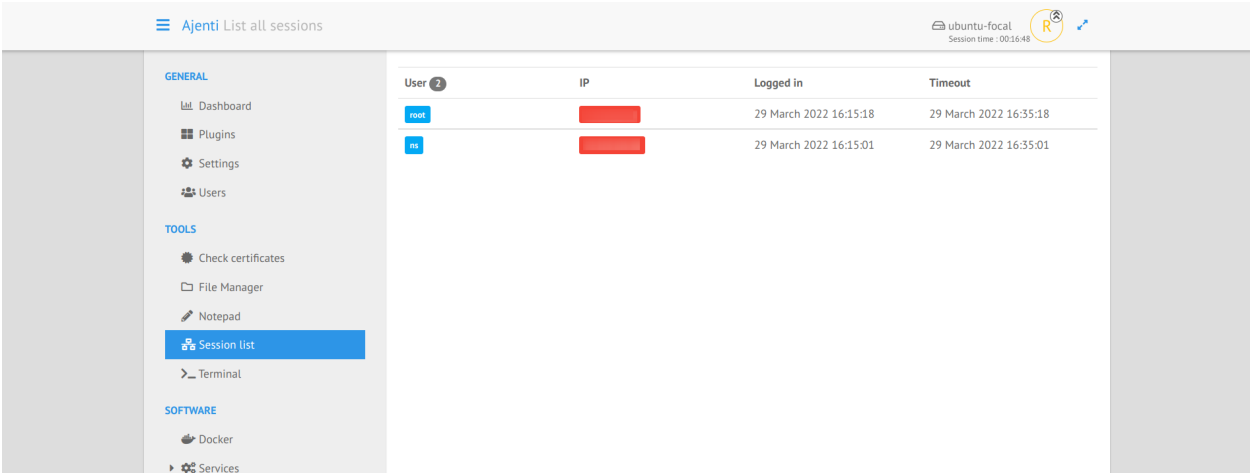
- start/stop/restart the service,
- enable/disable the service, if not static.

For the `system V` init services, you can:

- start/stop/restart the service,
- kill a running service.

1.6.20 Plugin session_list

this plugin displays the logged users, their ip and the timeout.



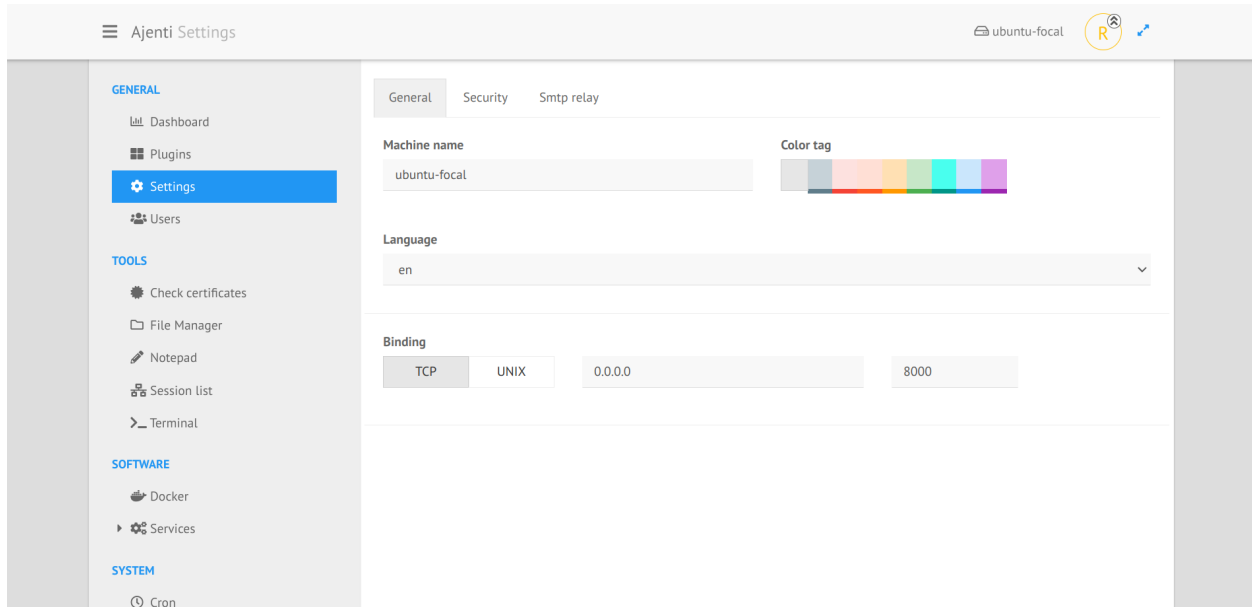
1.6.21 Plugin settings

This page gives access to the settings stored in `/etc/ajenti/config.yml` and `/etc/ajenti/smtp.yml`.

For a full description of the configuration files, please see [Configuration files](#).

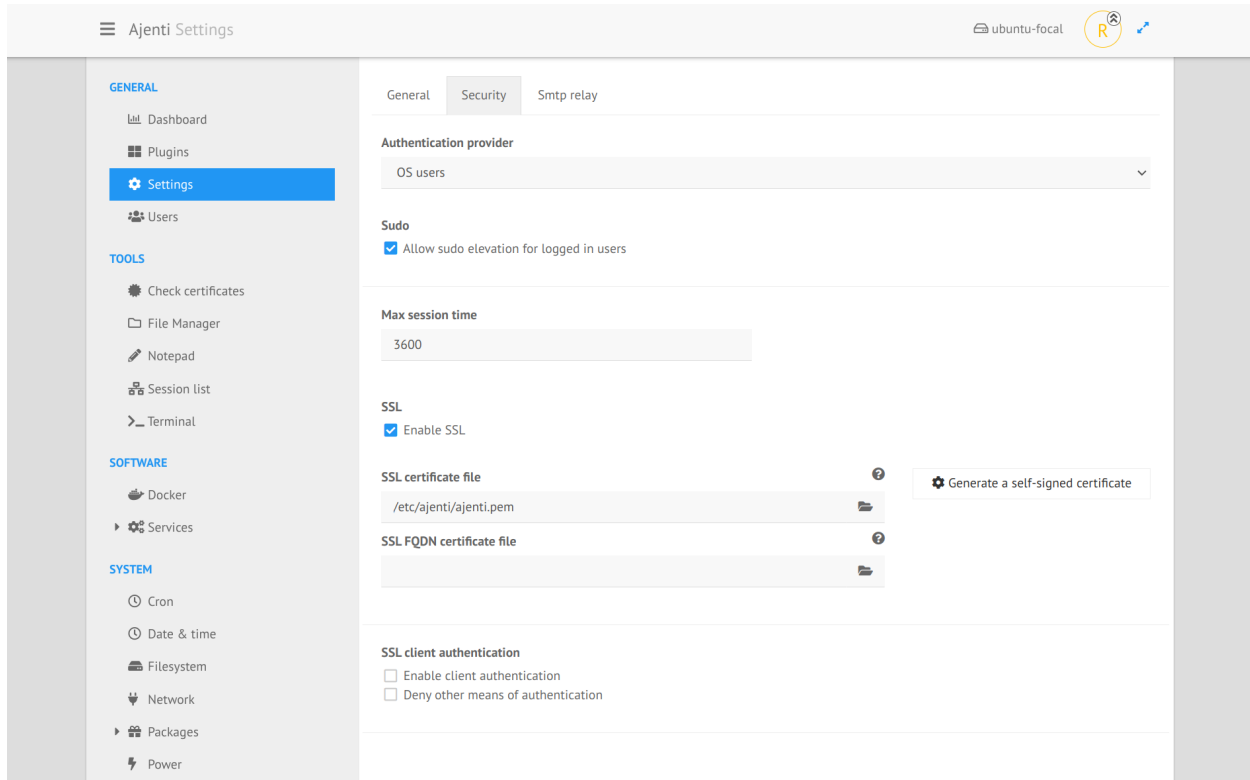
After changing the settings, it's necessary to restart the panel.

Tab General



This tab contains the binding settings, language, hostname set in Ajenti et color style.

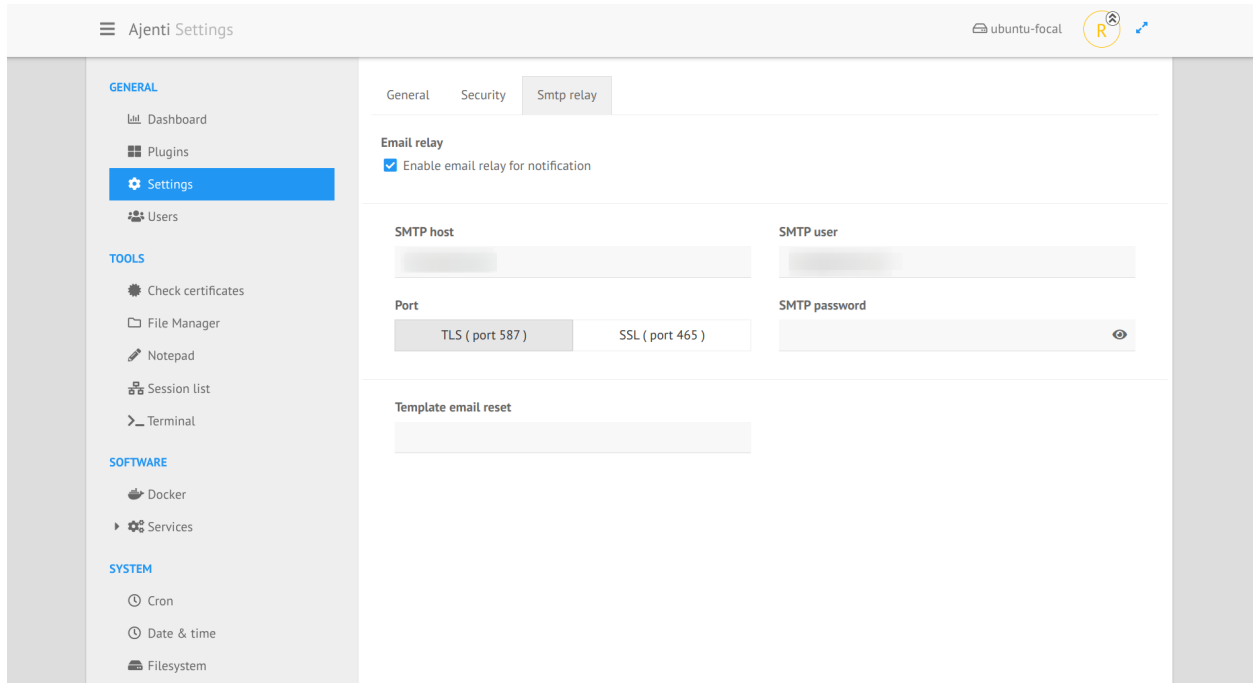
Tab Security



You can choose:

- the authentication provider (OS or USERS),
- allow sudo elevation or not,
- set the timeout of a session,
- configure SSL and certificates,
- configure SSL and certificates for client authentication.

Tab Smtplib relay



This tab provides the credentials saved in `/etc/ajenti/smtp.yml`.

1.6.22 Plugin terminal

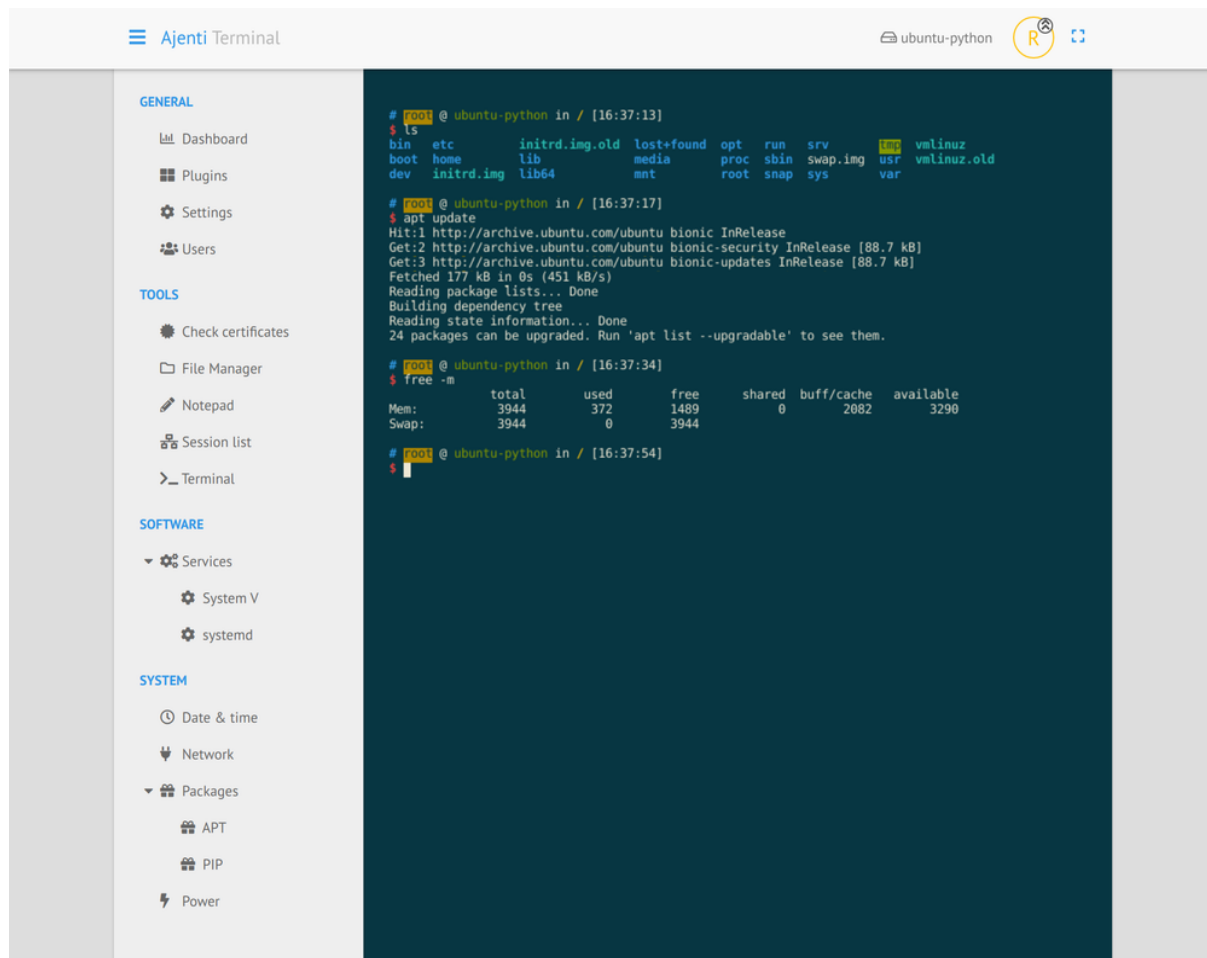
It would be really cool to have an terminal access on the server. That's exactly what this plugin does!

You have the possibility to launch a command (and naturally see the result) or to open a whole terminal on the server. You will get the same environment as your user on the system.

Type `exit` or `Ctrl + D` to come back to the terminal list.

Hotkeys

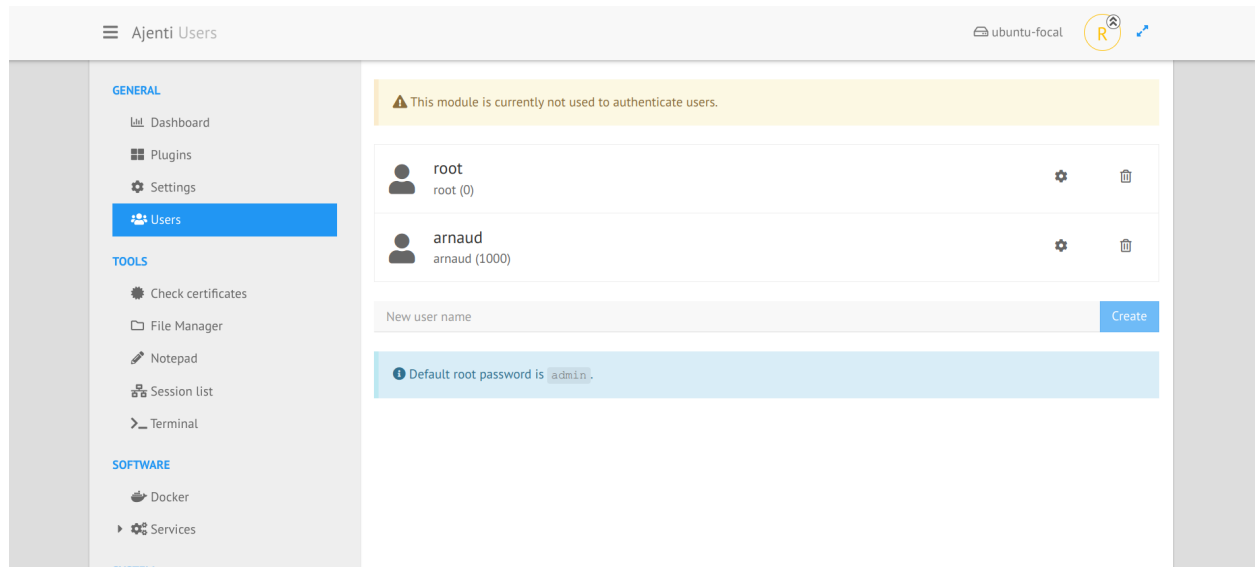
- `Ctrl + C` : copy
- `Ctrl + V` : paste
- `Ctrl + D` : exit



1.6.23 Plugin users

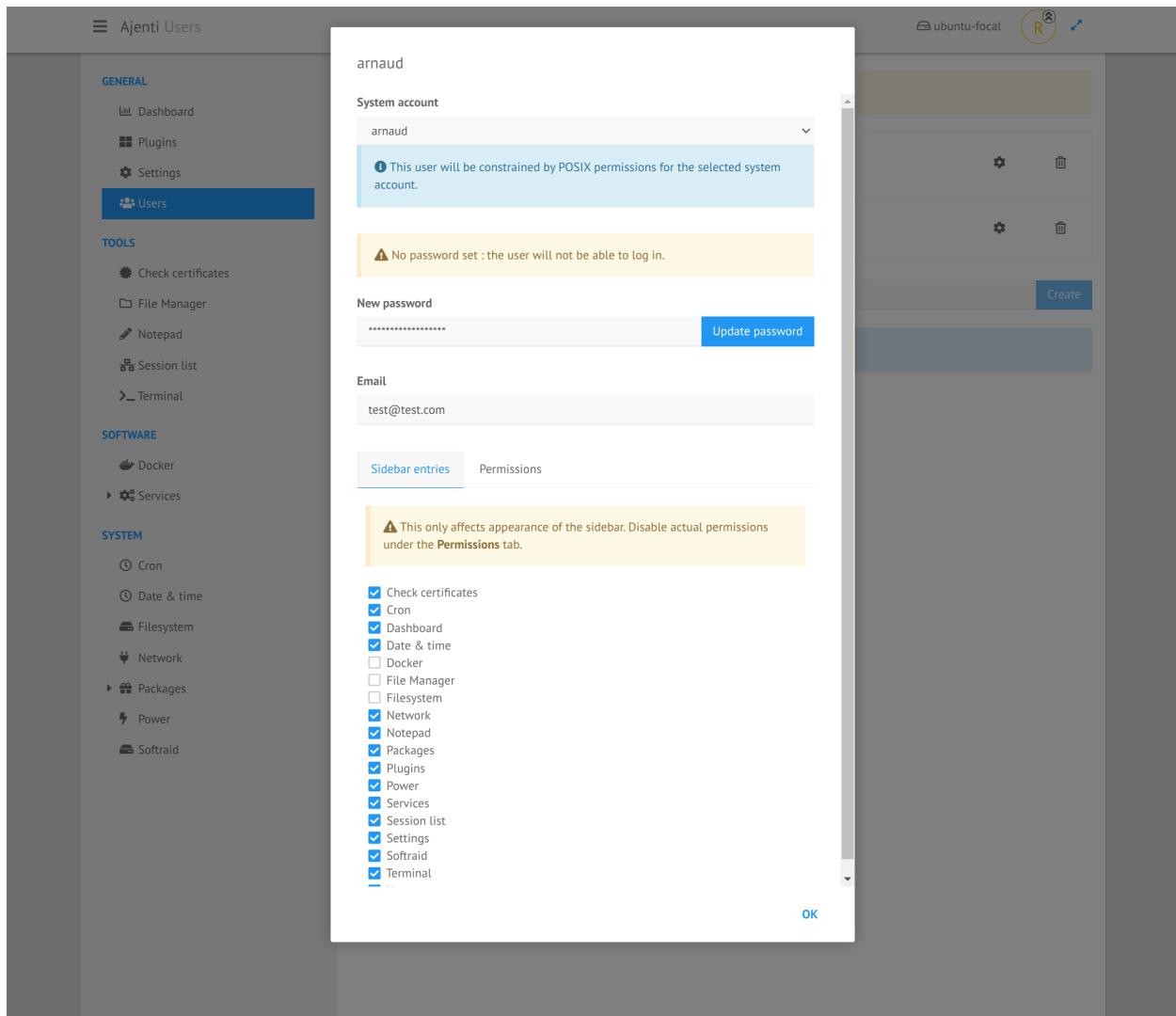
The default authentication provider used in Ajenti is the OS provider which allows all users of the system to log in.

The plugin `auth_users` provides an alternative way to authenticate users, and to create custom users. All users data are stored in plain text, in `/etc/ajenti/users.yml` (but this is configurable).



The default view presents a list of current users and let you:

- add a new user,
- manage the properties of an existing user,
- delete an existing user.



The property modal window displays some utilities per account:

- system account: all user accounts must be bound to a system account in order to set the privileges. An user bound to `root` will have all privileges, but an user bound to a system user account like `arnaud` will only have the privileges of the system user `arnaud`.
- password change: only a hash is stored, not the password itself,
- set the email: for notifications or password reset function,
- select the sidebar entries and permissions of the user.

Don't forget to **SAVE** the changes when updating an user.

1.6.24 Architecture and how it works

Backend

Ajenti project itself consists of **Ajenti Core** and a set of stock plugins forming the **Ajenti Panel**.

Ajenti Core

Represents the core backend and it's the entry point of Ajenti.

- HTTP server
- IoC container
- Base classes and Interfaces
- Simplistic web framework
- Set of core components aiding in client-server communications

Ajenti Panel

- Startup script
- Plugins developed for the Ajenti Core (filemanager, terminal, notepad, etc.)

Modus operandi

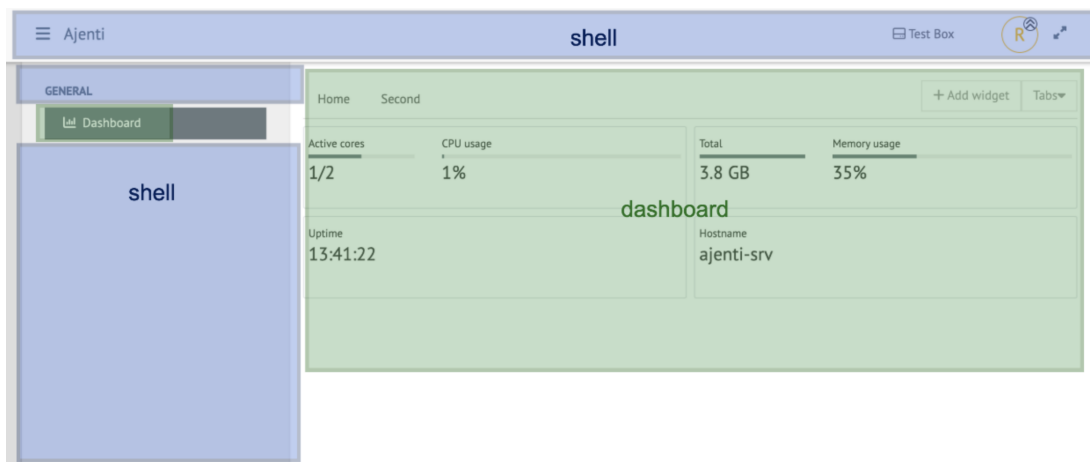
During bootstrap, Ajenti Core will locate and load Python modules containing Ajenti plugins (identified by a `plugin.yml` file). It will then register the implementation classes found in them in the root IoC container. Some interfaces to be implemented include `aj.api.http.HttpPlugin`, `aj.plugins.core.api.sidebar.SidebarItemProvider`.

Ajenti Core runs a HTTP server on a specified port, managing a pool of isolated session workers and forwarding requests to these workers, delivering them to the relevant `aj.api.http.HttpPlugin` instances. It also supports Socket.IO connections, forwarding them to the relevant `aj.api.http.SocketEndpoint` instances.

Ajenti contains a mechanism for session authentication through PAM login and `sudo` elevation. Standard core plugin provides HTTP API for that.

Authenticated sessions are moved to isolated worker processes running under the corresponding account.

Frontend



The frontend can be divided into two main parts:

- core part (plugin `shell` and `ngx-ajenti`)

- extension plugins (ace, dashboard, filemanager,..)

Screenshot

shell (plugin)

Serves as a container for other plugins. Plugins are implemented as micro-frontends and are loaded within the shell. It uses [@angular-architects/module-federation](#) package of Angular Architects. For deep dive into Webpack 5's module federation usage with Angular see the [link](#).

- Basic navigation (Header, Sidebar, Routing,..)
- Container for other plugins
- Config management

ngx-ajenti (plugin)

Represents the shared library.

- Authentication and Identity management
- Global (TS) services and components
- Navigation (Header, Sidebar, Routing,..)
- Config management
- Plugin manager

1.6.25 Ajenti Dev Multitool

```
sudo pip install ajenti-dev-multitool
```

ajenti-dev-multitool is a mini-utility to help you with common plugin development tasks.

ajenti-dev-multitool typically operates on all plugins found in current directory and below.

- `--run` will launch the globally installed Ajenti with plugins from the current directory. `--run-dev` will additionally enable developer mode.
- `--build-frontend` builds the frontend resources.
- `--setuppy "<setup.py-command-with-args>"` runs a `setuptools` command on the plugin package. A `setup.py` file is generated automatically. Example: `ajenti-dev-multitool --setuppy 'sdist upload --sign --identity "John Doe"'`

1.6.26 User Interface

Basics

Ajenti frontend is a Angular based single-page rich web application.

Your plugins can extend it by adding new Angular components and routes.

Client-server communication is facilitated by AJAX requests to backend API (`HttpClient`) and a `Socket.IO` connection (`socket` and `push` Angular services).

Client styling is based on a customized Bootstrap build.

Example

Warning: This part is obsolete. The demo-plugins repo must be converted from AngularJS to Angular.

Basic UI example can be browsed and downloaded at https://github.com/ajenti/demo-plugins/tree/master/demo_2_ui

The basic UI plugin includes:

- an AngularJS `module` containing a `route` and a `controller`:
- an AngularJS `view template` (HTML)

1.6.27 Handling HTTP Requests

This page describes how to handle HTTP request on the backend side.

Example

Basic HTTP API example can be browsed and downloaded at https://github.com/ajenti/demo-plugins/tree/master/demo_4_http

Plugins can provide their own HTTP endpoints by extending the `aj.api.http.HttpPlugin` abstract class.

Example:

```
import time
from jadi import component

from aj.api.http import get, HttpPlugin

from aj.api.endpoint import endpoint, EndpointError, EndpointReturn

@Component(HttpPlugin)
class Handler(HttpPlugin):
    def __init__(self, context):
        self.context = context

    @get(r'/api/demo4/calculate/(?P<operation>\w+)/(?P<a>\d+)/(?P<b>\d+)')
    @endpoint(api=True)
    def handle_api_calculate(self, http_context, operation=None, a=None, b=None):
        start_time = time.time()

        try:
            if operation == 'add':
                result = int(a) + int(b)
            elif operation == 'divide':
                result = int(a) / int(b)
            else:
                raise EndpointReturn(404)
        except ZeroDivisionError:
            raise EndpointError('Division by zero')
```

(continues on next page)

(continued from previous page)

```

return {
    'value': result,
    'time': time.time() - start_time
}

```

@endpoint (api=True) mode provides automatic JSON encoding of the responses and error handling.

If you need lower-level access to the HTTP response, use @endpoint (page=True):

```

@get (r'/api/test')
@endpoint (page=True)
def handle_api_calculate(self, http_context):
    http_context.add_header('Content-Type', '...')
    content = "Hello!"
    #return http_context.respond_not_found()
    #return http_context.respond_forbidden()
    #return http_context.file('/some/path')
    http_context.respond_ok()
    return content

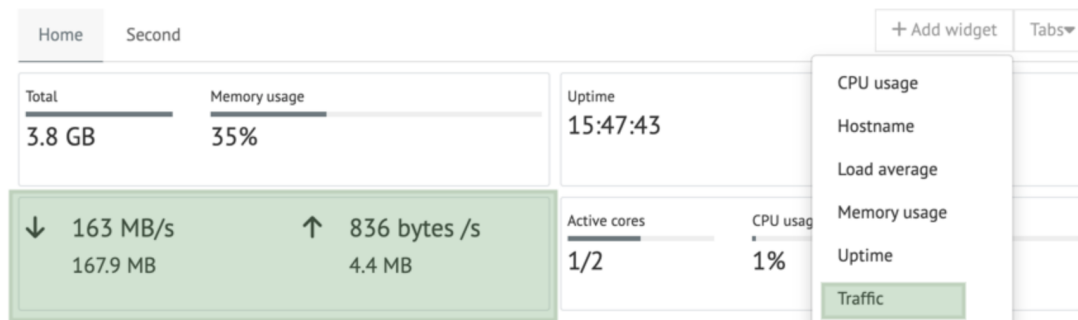
```

See [aj.http.HttpContext](#) for the available http_context methods.

1.6.28 Dashboard Widgets

The dashboard (plugin) provides a way how to extend dashboard with some extra widgets. This is done by implementing a new module containing the new widget(s).

Example of a Traffic widget (located in the `traffic` module)



Example implementation

Elements to be implemented

- Backend: Widget class
- Backend: Widget config endpoint (Optional)
- Frontend: WidgetComponent
- Frontend: Widget config component (Optional)

Backend: Widget class

This class must implement the `aj.plugins.dashboard.widget`. It's used for the registration in the backend and as a provider for the widget data. Dashboard will issue periodic requests to your `aj.plugins.dashboard.api.Widget` implementations. If user creates multiple widgets of same type, a single instance will be created to service their requests.

Example widget class:

```
@component (Widget)
class TrafficWidget (Widget):
    id = 'traffic'
    name = _('Traffic')

    ..

    def get_value(self, config):
        ...

    return { .. }
```

Backend: Widget config endpoint (Optional)

This is required only if the widget is configurable. The endpoint is implemented as a handler from the `HttpPlugin`. The decorator `@url` will register the endpoint in the backend.:

```
@component (HttpPlugin)
class Handler (HttpPlugin):
    ..

    @url (r'/api/traffic/interfaces')
    @endpoint (api=True)
    def handle_api_interfaces(self, http_context):
        ..
        return ..
```

Frontend: WidgetComponent

This is the actual UI shown to the user. It's implemented as a Angular component. This component must be exposed in the `webpack.config.js` as part of the `ModuleFederationPlugin`.

Widget component implementation: https://github.com/ajenti/demo-plugins/tree/master/demo_5_widget/frontend/components/demowidget/

Webpack registration: https://github.com/ajenti/demo-plugins/tree/master/demo_5_widget/frontend/webpack.config.js#L35

1.6.29 Extension plugins

This page describes the way how to setup the development environment for the development of extension plugins.

Required knowledge

- Python 3, Typescript, Angular, HTML

Steps

- 1. Setup Ajenti (core)
- 2. Install build tools
- 3. Setup plugin environment

1. Setup Ajenti (core)

The Ajenti (core) is required for the development and run of any plugin. There are two development scenarios:

Develop only an extension plugin

Install the Ajenti(Core): *Installation guide*

Develop an extension plugin + Ajenti(core) and the same time

Run the Ajenti(Core) in the development mode *Core*

2. Install build tools

Follow the steps in *Build tools*

4. Plugin development

4.1 Edit existing plugin

See the *plugins-new/Readme.txt*

4.2 Create a new plugin

Create a new plugin in the current directory:

```
ajenti-dev-multitool --new-plugin "Some plugin name"
```

Build frontend:

```
ajenti-dev-multitool --build-frontend
```

Start start the backend:

```
#If Ajenti(core) was installed
sudo ajenti-dev-multitool --run-dev
#Navigate to http://localhost:8000/

#If Ajenti(core) is running in the dev mode:
make rundev
```

See the *plugins-new/Readme.txt* to see how to start the frontend

What's inside a plugin?

- Backend: Python modules, which contain `jadi.component` classes (*components*).
- Frontend (optional): Angular components, services and LESS files.

Example plugin structure:

```
some_plugin_name
├── backend/
│   ├── controllers
│   │   └── dashboard.py
│   ├── __init__.py
│   └── requirements.txt
├── frontend/
│   ├── e2e/
│   └── src/
│       ├── components
│       │   ├── uptime-widget.component.html
│       │   ├── uptime-widget.component.less
│       │   └── uptime-widget.component.ts
│       ├── services
│       │   └── dashboard.service.ts
│       └── dashboard.module.ts
├── locale/
├── plugin.yml    #plugin description
└── README.md
```

Example plugins

See the demo-plugins git repo for some example plugin implementations.

Warning: This part is obsolete. The demo-plugins repo must be converted from AngularJS to Angular.

Download plugins from here: <https://github.com/ajenti/demo-plugins> or clone this entire repository.

Prep work:

```
ajenti-dev-multitool --build-frontend
```

Run:

```
ajenti-dev-multitool --run-dev
```

1.6.30 Core

This page describes the way how to setup the development environment for the development of the core.

Attention: For plugin/extension development see *Extension plugins*

Required knowledge

- Python 3.x, async programming with `gevent`, HTML, Angular, Typescript, LESS

Prerequisites

Minimal set of software required to build and run Ajenti: `git`, `Node.js`

Debian/Ubuntu extras: `python3-dbus` (ubuntu)

Steps

There are two ways how to setup the core.

- Automatic (Recommended)
- Manual

Automatic Installation (Backend + Frontend)

The following script will perform a complete automatic installation under Debian or Ubuntu, using virtual environment with *Python*. The virtual environment is then located in `/opt/ajenti` and the cloned git repository in `/opt/ajenti/ajenti`. This install script will install a lot of dependencies, this may take several minutes.

```
curl https://raw.githubusercontent.com/ajenti/ajenti/master/scripts/install-dev.sh |   
↪ sudo bash -s -
```

After a successful installation, do the following to activate the dev mode:

- Activate the virtual environment : `source /opt/ajenti/bin/activate`
- Navigate in the git repository : `cd /opt/ajenti/ajenti`
- Launch a rundev recipe : `make rundev` (quit with `Ctrl+ C`)
- Call `https://localhost:8000` in your browser (you will get some warnings because of the self-signed certificate, it's perfectly normal.

Manual installation - Backend

Download the source code:

```
git clone git://github.com/ajenti/ajenti.git
```

Install the dependencies:

```
# Debian/Ubuntu  
sudo apt-get install build-essential python3-pip python3-dev python3-lxml libffi-dev   
↪ libssl-dev libjpeg-dev libpng-dev uuid-dev python3-dbus  
  
# RHEL  
sudo dnf install gcc python3-devel python3-pip libxslt-devel libxml2-devel libffi-  
↪ devel openssl-devel libjpeg-turbo-devel libpng-devel dbus-python  
  
cd ajenti
```

(continues on next page)

(continued from previous page)

```
sudo pip3 install -r ajenti-core/requirements.txt
sudo pip3 install ajenti-dev-multitool
```

Install the build tools

Follow: *Build tools*

Ensure that resource compilation is set up correctly and works (optional):

```
make build
```

Launch Ajenti backend in dev mode:

```
make rundev
```

Navigate to <http://localhost:8000/>.

Hint: Additional debug information will be available in the console output and browser console. Reloading the page with Ctrl-F5 (Cache-Control: no-cache) will unconditionally rebuild all resources

Manual installation - Frontend

The setup the core frontend is needed to build and run the plugins: `ngx-ajenti` and `shell`

The way how to do it is described here in the `plugins-new/README.md` See the Readme <https://github.com/daniel-schulz/netzint-ajenti/blob/dev/plugins-new/README.md>

For more info see What's Ajenti and how it works.

1.6.31 Build tools

This setup is required for the development of the *Core* and the *Extension plugins*.

Steps

Install Curl:

```
sudo apt install curl
```

Install NodeJS - you can use the NodeSource repositories for quick setup:

```
# Using Ubuntu
curl -sL https://deb.nodesource.com/setup_17.x | sudo -E bash -
sudo apt-get install -y nodejs

# Using Debian, as root
curl -sL https://deb.nodesource.com/setup_17.x | bash -
apt-get install -y nodejs

# Using RHEL or centos, as root
curl -sL https://rpm.nodesource.com/setup_17.x | bash -
```

Install Yarn - Enable the official Yarn repository, import the repository GPG key, and install the package.:

```
# Using Ubuntu
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.
↪list.d/yarn.list
sudo apt update
sudo apt install --no-install-recommends yarn
```

Install Angular CLI:

```
sudo yarn global add @angular/cli
```

Install Gettext:

```
# Ubuntu or Debian:
sudo apt-get install gettext

# RHEL or CentOS
dnf install gettext
```

Install Ajenti Dev Multitool:

```
pip3 install ajenti-dev-multitool
```

(More info about the *Ajenti Dev Multitool*)

1.6.32 API: jadi

`jadi.get_fqdn(cls)`

Returns a fully-qualified name for the given class

`jadi.interface(cls)`

Marks the decorated class as an abstract interface.

Injects following classmethods:

.all(context)

Returns a list of instances of each component in the `context` implementing this `@interface`

Parameters `context` (*Context*) – context to look in

Returns `list(cls)`

.any(context)

Returns the first suitable instance implementing this `@interface` or raises *NoImplementationError* if none is available.

Parameters `context` (*Context*) – context to look in

Returns `cls`

.classes()

Returns a list of classes implementing this `@interface`

Returns `list(class)`

`jadi.component(iface)`

Marks the decorated class as a component implementing the given `iface`

Parameters `iface` (*interface()*) – the interface to implement

`jadi.service(cls)`

Marks the decorated class as a singleton service.

Injects following classmethods:

```

    .get (context)
        Returns a singleton instance of the class for given context
        Parameters context (Context) – context to look in
        Returns cls

class jadi.Context (parent=None)
    An IoC container for interface () s, service () s and component () s

    Parameters parent (Context) – a parent context

    get_component (cls)

    get_components (cls, ignore_exceptions=False)

    get_service (cls)

exception jadi.NoImplementationError (cls)

```

1.6.33 API: aj

```

aj.config = <module 'aj.config' from '/home/docs/checkouts/readthedocs.org/user_builds/ajenti/venv/lib/python3.6/site-packages/ajenti/config.py'>
    Configuration dict

aj.platform = 'debian'
    Current platform

aj.platform_string = 'Ubuntu 18.04.5 LTS'
    Human-friendly platform name

aj.platform_unmapped = 'ubuntu'
    Current platform without “Ubuntu is Debian”-like mapping

aj.version = '2.2.1'
    Ajenti version

aj.server = None
    Web server

aj.debug = False
    Debug mode

aj.init ()

aj.exit ()

aj.restart ()

```

1.6.34 API: aj.api.http

```

class aj.api.http.BaseHttpHandler
    Base class for everything that can process HTTP requests

    handle (http_context)
        Should create a HTTP response in the given http_context and return the plain output

        Parameters http_context (aj.http.HttpContext) – HTTP context

class aj.api.http.HttpMasterMiddleware (context)

    handle (http_context)
        Should create a HTTP response in the given http_context and return the plain output

```

Parameters `http_context` (*aj.http.HttpContext*) – HTTP context

class `aj.api.http.HttpMiddleware` (*context*)

handle (*http_context*)

Should create a HTTP response in the given `http_context` and return the plain output

Parameters `http_context` (*aj.http.HttpContext*) – HTTP context

class `aj.api.http.HttpPlugin` (*context*)

A base interface for HTTP request handling:

```
@component
class HelloHttp(HttpPlugin):
    @get('/hello/(?P<name>.+)' )
    def get_page(self, http_context, name=None):
        context.add_header('Content-Type', 'text/plain')
        context.respond_ok()
        return 'Hello, f"{name}"!'
```

handle (*http_context*)

Finds and executes the handler for given request context (handlers were methods decorated with `url()` and will be decorated with e.g. `@get` and `@post` in the future)

Parameters `http_context` (*aj.http.HttpContext*) – HTTP context

Returns response data

class `aj.api.http.SocketEndpoint` (*context*)

Base interface for Socket.IO endpoints.

destroy ()

Destroys endpoint, killing the running greenlets

on_connect (*message*)

Called on a successful client connection

on_disconnect (*message*)

Called on a client disconnect

on_message (*message, *args*)

Called when a socket message arrives to this endpoint

plugin = `None`

arbitrary plugin ID for socket message routing

send (*data, plugin=None*)

Sends a message to the client.

Parameters

- **data** – message object
- **plugin** (*str*) – routing ID (this endpoint's ID if not specified)

spawn (*target, *args, **kwargs*)

Spawns a greenlet in this endpoint, which will be auto-killed when the client disconnects

Parameters **target** – target function

`aj.api.http.delete` (*pattern*)

Exposes the decorated method of your `HttpPlugin` via HTTP

Parameters **pattern** (*str*) – URL regex (^ and \$ are implicit)

Return type

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.get(pattern)`Exposes the decorated method of your `HttpPlugin` via HTTP**Parameters** `pattern` (*str*) – URL regex (^ and \$ are implicit)**Return type**

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.head(pattern)`Exposes the decorated method of your `HttpPlugin` via HTTP**Parameters** `pattern` (*str*) – URL regex (^ and \$ are implicit)**Return type**

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.patch(pattern)`Exposes the decorated method of your `HttpPlugin` via HTTP**Parameters** `pattern` (*str*) – URL regex (^ and \$ are implicit)**Return type**

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.post(pattern)`Exposes the decorated method of your `HttpPlugin` via HTTP**Parameters** `pattern` (*str*) – URL regex (^ and \$ are implicit)**Return type**

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.put(pattern)`Exposes the decorated method of your `HttpPlugin` via HTTP**Parameters** `pattern` (*str*) – URL regex (^ and \$ are implicit)**Return type**

function

Named capture groups will be fed to function as `**kwargs``aj.api.http.requests_decorator_generator(method)`Factorization to generate request decorators like `@get` or `@post`.**Parameters** `method` (*basestring*) – Request method decorator to generate, like `get` or `post`**Returns****Return type**

`aj.api.http.url` (*pattern*)

Exposes the decorated method of your *HttpPlugin* via HTTP. Will be deprecated in favor of new decorators (`@get`, `@post`, ...)

Parameters `pattern` (*str*) – URL regex (^ and \$ are implicit)

Return type

function

Named capture groups will be fed to function as `**kwargs`

1.6.35 API: `aj.api.endpoint`

exception `aj.api.endpoint.EndpointError` (*inner, message=None*)

To be raised by endpoints when a foreseen error occurs. This exception doesn't cause a client-side crash dialog.

Parameters

- **inner** – inner exception
- **message** – message

exception `aj.api.endpoint.EndpointReturn` (*code, data=None*)

Raising `EndpointReturn` will return a custom HTTP code in the API endpoints.

Parameters

- **code** – HTTP code
- **data** – response data

`aj.api.endpoint.endpoint` (*page=False, api=False, auth=True*)

It's recommended to decorate all HTTP handling methods with `@endpoint`.

`@endpoint(auth=True)` will require authenticated session before giving control to the handler.

`@endpoint(api=True)` will wrap responses and exceptions into JSON, and will also provide special handling of `EndpointsError`

Parameters

- **auth** (*bool*) – requires authentication for this endpoint
- **page** (*bool*) – enables page mode
- **api** (*bool*) – enables API mode

1.6.36 API: `aj.config`

class `aj.config.UserConfigService` (*context*)

classmethod `get` (*context*)

`get_provider` ()

1.6.37 API: aj.core

`aj.core.restart()`

`aj.core.run(config=None, plugin_providers=None, product_name='ajenti', dev_mode=False, debug_mode=False, autologin=False)`

A global entry point for Ajenti.

Parameters

- **config** (`aj.config.BaseConfig`) – config file implementation instance to use
- **plugin_providers** (`list(aj.plugins.PluginProvider)`) – list of plugin providers to load plugins from
- **product_name** (`str`) – a product name to use
- **dev_mode** (`bool`) – enables dev mode (automatic resource recompilation)
- **debug_mode** (`bool`) – enables debug mode (verbose and extra logging)
- **autologin** (`bool`) – disables authentication and logs everyone in as the user running the panel. This is EXTREMELY INSECURE.

1.6.38 API: aj.entry

`aj.entry.handle_crash(exc)`

`aj.entry.start(daemonize=False, log_level=20, dev_mode=False, **kwargs)`

A wrapper for `run()` that optionally runs it in a forked daemon process.

Parameters **kwargs** – rest of arguments is forwarded to `run()`

1.6.39 API: aj.http

class `aj.http.HttpContext(env, start_response=None)`

Instance of `HttpContext` is passed to all HTTP handler methods

env

WSGI environment dict

path

Path segment of the URL

method

Request method

headers

List of HTTP response headers

body

Request body

response_ready

Indicates whether a HTTP response has already been submitted in this context

query

HTTP query parameters

add_header(key, value)

Adds a given HTTP header to the response

Parameters

- **key** (*str*) – header name
- **value** (*str*) – header value

classmethod **deserialize** (*data*)

dump_env ()

fallthrough (*handler*)

Executes a *handler* in this context

Returns handler-supplied output

file (*path*, *stream=False*, *inline=False*, *name=None*)

Returns a GZip compressed response with content of file located in *path* and correct headers

get_cleaned_env ()

gzip (*content*, *compression=6*)

Returns a GZip compressed response with given *content* and correct headers

Parameters **compression** (*int*) – compression level from 0 to 9

Return type *str*

json_body ()

redirect (*location*)

Returns a HTTP 302 Found redirect response with given *location*

remove_header (*key*)

Removed a given HTTP header from the response

Parameters **key** (*str*) – header name

respond (*status*)

Creates a response with given HTTP status line

respond_forbidden ()

Returns a HTTP 403 Forbidden response

respond_not_found ()

Returns a HTTP 404 Not Found response

respond_ok ()

Creates a HTTP 200 OK response

respond_server_error ()

Returns a HTTP 500 Server Error response

respond_unauthenticated ()

Returns a HTTP 401 Unauthenticated response

run_response ()

Finalizes the response and runs WSGI's `start_response()`.

serialize ()

```

class aj.http.HttpMiddlewareAggregator (stack)
    Stacks multiple HTTP handlers together in a middleware fashion.

    Parameters stack (list(aj.api.http.BaseHttpHandler)) – handler list

    handle (http_context)
        Should create a HTTP response in the given http_context and return the plain output

        Parameters http_context (aj.http.HttpContext) – HTTP context

class aj.http.HttpRoot (handler)
    A root WSGI middleware object that creates the HttpContext and dispatches it to an HTTP handler.

    Parameters handler (aj.api.http.BaseHttpHandler) – next middleware handler

    dispatch (env, start_response)
        Dispatches the WSGI request

```

1.6.40 API: aj.plugins

```

class aj.plugins.PluginProvider
    A base class for plugin locator

    provide ()
        Should return a list of found plugin paths

        Returns list(str)

class aj.plugins.DirectoryPluginProvider (path)
    A plugin provider that looks up plugins in a given directory.

    Parameters path – directory to look for plugins in

    provide ()
        Should return a list of found plugin paths

        Returns list(str)

class aj.plugins.PythonPathPluginProvider
    A plugin provider that looks up plugins on $PYTHONPATH

    provide ()
        Should return a list of found plugin paths

        Returns list(str)

exception aj.plugins.PluginLoadError

exception aj.plugins.PluginCrashed (exception)

    describe ()

class aj.plugins.Dependency

    exception Unsatisfied

    describe ()

    reason ()

    build_exception ()

```

```
    check()
    value
    yaml_loader
        alias of yaml.loader.SafeLoader
    yaml_tag = '!Dependency'
class aj.plugins.ModuleDependency (module_name=None)

    exception Unsatisfied

        reason()
    description = 'Python module'
    is_satisfied()
    yaml_tag = '!ModuleDependency'
class aj.plugins.PluginDependency (plugin_name=None)

    exception Unsatisfied

        reason()
    description = 'Plugin'
    is_satisfied()
    yaml_tag = '!PluginDependency'
class aj.plugins.OptionalPluginDependency (plugin_name=None)

    exception Unsatisfied

        reason()
    description = 'Plugin'
    is_satisfied()
    yaml_tag = '!OptionalPluginDependency'
class aj.plugins.BinaryDependency (binary_name=None)

    exception Unsatisfied

        reason()
    description = 'Application binary'
    is_satisfied()
    yaml_tag = '!BinaryDependency'
class aj.plugins.FileDependency (file_name=None)
```

```

exception Unsatisfied

    reason()

    description = 'File'

    is_satisfied()

    yaml_tag = '!FileDependency'

class aj.plugins.PluginManager(context)
    Handles plugin loading and unloading

    classmethod get(context)

    get_content_path(name, path)

    get_crash(name)

    get_loaded_plugins_list()

    load_all_from(providers)
        Loads all plugins provided by given providers.

        Parameters providers (list(PluginProvider)) –

```

1.6.41 Angular: ajenti.core

This Angular module contains core components of Ajenti frontend.

Services

```

class config()

    config.data
        Config file content object

    config.load()
        Gets complete configuration data of the backend

        Returns promise

    config.save()
        Updates and saves configuration data

        Returns promise

    config.getUserConfig()
        Gets per-user configuration data of the backend

        Returns promise → per-user Ajenti config object

    config.setUserConfig(config)
        Updates and saves per-user configuration data

        Arguments
            • config(object) – updated configuration data from getUserConfig()

        Returns promise

```

class core()

`core.pageReload()`
Reloads the current URL

`core.restart()`
Restarts the Ajenti process

class hotkeys()

Captures shortcut key events

`hotkeys.ENTER, ESC`
Respective key codes

`hotkeys.on(scope, handler, mode='keydown')`
Registers a hotkey handler in the provided `scope`

Arguments

- **scope** (*\$scope*) – *\$scope* to install handler into
- **handler** (*function(keyCode, rawEvent)*) – handler function. If the function returns a truthy value, event is cancelled and other handlers aren't notified.
- **mode** (*string*) – one of `keydown`, `keypress` or `keyup`.

class identity()

Provides info on the authentication status and user/machine identity

`identity.user`
Name of the logged in user

`identity.effective`
Effective UID of the server process

`identity.machine.name`
User-provided name of the machine

`identity.isSuperuser`
Whether current user is a superuser or not

`identity.auth(username, password, mode)`
Attempts to authenticate current session as `username:password` with a mode of `normal` or `sudo`

`identity.login()`
Redirects user to a login dialog

`identity.logout()`
Deauthenticates current session

`identity.elevate()`
Redirects user to a sudo elevation dialog

class messagebox()

Provides interface to modal messagebox engine

`messagebox.show(options)`
Opens a new messagebox.

Arguments

- **options** (*object*) –
- **options.title** (*string*) –

- **options.text** (*string*) –
- **options.positive** (*string*) – positive action button text. Clicking it will resolve the returned promise.
- **options.negative** (*string*) – negative action button text. Clicking it will reject the returned promise.
- **options.template** (*string*) – (optional) custom body template
- **options.scrollable** (*boolean*) – whether message body is scrollable
- **options.progress** (*boolean*) – whether to display an indeterminate progress indicator in the message

Returns a Promise-like object with an additional `close()` method.

class notify()

```
notify.info(title, text)
notify.success(title, text)
notify.warning(title, text)
notify.error(title, text)
    Shows an appropriately styled notification
notify.custom(style, title, text, url)
    Shows a clickable notification leading to url.
```

class pageTitle()

Alters page `<title>` and global heading.

```
pageTitle.set(text)
    Sets title text
```

```
pageTitle.set(expression, scope)
    Sets an title expression to be watched. Example:
```

```
$scope.getTitle = (page) -> someService.getPageTitle(page)
$scope.page = ...

pageTitle.set("getTitle(page)", $scope)
```

class push()

Processes incoming push messages (see [aj.plugins.core.api.push](#)). This service has no public methods.

This service broadcasts events that can be received as:

```
$scope.$on 'push:pluginname', (message) ->
    processMessage(message)...
```

class tasks()

An interface to the tasks engine (see [aj.plugins.core.api.tasks](#)).

tasks.tasks

A list of task descriptors for the currently running tasks. Updated automatically.

```
tasks.start(cls, args, kwargs)
    Starts a server-side task.
```

Arguments

- **cls** (*string*) – full task class name (`aj.plugins.pluginname....`)
- **args** (*array*) – task arguments
- **kwargs** (*object*) – task keyword arguments

Returns a promise, resolved once the task actually starts

Directives

autofocus ()

Automatically focuses the input. Example:

```
<input type="text" autofocus ng:model="..." />
```

checkbox ()

Renders a checkbox. Example:

```
<span checkbox ng:model="..." text="Enable something"></span>
```

dialog ()

A modal dialog

Example:

```
<dialog ng:show="showDialog">
  <div class="modal-header">
    <h4>
      Heading
    </h4>
  </div>
  <div class="modal-body scrollable">
    ...
  </div>
  <div class="modal-footer">
    <a ng:click="..." class="btn btn-default btn-flat">
      Do something
    </a>
  </div>
</dialog>
```

Arguments

- **ngShow** (*expression*) –
- **dialogClass** (*string*) –

floating-toolbar ()

A toolbar pinned to the bottom edge. Example:

```
<div class="floating-toolbar-padder"></div>

<floating-toolbar>
  <a ng:click="..." class="btn btn-default btn-flat">
    Do something useful
  </a>
```

(continues on next page)

(continued from previous page)

```

</floating-toolbar>

<!-- accented toolbar for selection actions -->

<floating-toolbar class="accented" ng:show="haveSelectedItems">
  Some action buttons here
</floating-toolbar>

```

ng-enter()

Action handler for Enter key in inputs. Example:

```
<input type="text" ng:enter="commitStuff()" ng:model="..." />
```

progress-spinner()**root-access()**

Blocks its inner content if the current user is not a superuser.

smart-progress()

An improved version of ui-bootstrap's progressbar

Arguments

- **animate** (*boolean*) –
- **value** (*float*) –
- **max** (*float*) –
- **text** (*string*) –
- **maxText** (*string*) –

Filters**bytesFilter** (*value*, *precision*)**Arguments**

- **value** (*int*) – number of bytes
- **precision** (*int*) – number of fractional digits in the output

Returns string, e.g.: 123.45 KB

ordinalFilter (*value*)**Arguments**

- **value** (*int*) –

Returns string, e.g.: 121st

pageFilter (*list*, *page*, *pageSize*)

Provides a page-based view on an array

Arguments

- **list** (*array*) – input data
- **page** (*int*) – 1-based page index
- **pageSize** (*int*) – page size

Returns array

1.6.42 Angular: ajenti.ace

ACE code editor integration

Directives

ace-editor()

Arguments

- **ngModel** (*binding*) –
- **aceOptions** (*object*) – (optional) options for ace.setOptions()

1.6.43 Angular: ajenti.augeas

Services

class augeas()

augeas.get (*endpoint*)

Reads an Augeas tree from server side.

Returns promise → AugeasConfig

augeas.set (*endpoint, config*)

Overwrites an Augeas tree on the server side.

Returns promise

class AugeasNode()

AugeasNode.**name**

AugeasNode.**value**

AugeasNode.**parent**

AugeasNode.**children**

AugeasNode.**fullPath**()

class AugeasConfig()

This is a JS doppelganger of normal Augeas API. In particular, it doesn't support advanced XPath syntax, and operates with regular expressions instead.

AugeasConfig.**get** (*path*)

Returns AugeasNode

AugeasConfig.**set** (*path, value*)

AugeasConfig.**model** (*path*)

Returns a getter/setter function suitable for use as a ngModel

AugeasConfig.**insert** (*path, value, index*)

```
AugeasConfig.remove (path)
AugeasConfig.match (path)
Returns Array (string)
AugeasConfig.matchNodes (path)
Returns Array (AugeasNode)
```

1.6.44 Angular: ajenti.filesystem

Services

```
class filesystem()
```

```
filesystem.read (path)
Returns promise → content of path
filesystem.write (path, content)
Returns promise
filesystem.list (path)
Returns promise → array
filesystem.stat (path)
Returns promise → object
filesystem.chmod (path, mode)
Arguments

- mode (int) – numeric POSIX file mode

Returns promise
filesystem.createFile (path, mode)
Arguments

- mode (int) – numeric POSIX file mode

Returns promise
filesystem.createDirectory (path, mode)
Arguments

- mode (int) – numeric POSIX file mode

Returns promise
filesystem.downloadBlob (content, mime, name)
  Launches a browser-side file download
Arguments

- content (string) – Raw file content
- mime (string) – MIME type used
- name (string) – Default file name for saving

```

Returns promise

Directives

file-dialog()

File open/save dialog. Example:

```
<file-dialog
  mode="open"
  ng:show="openDialogVisible"
  on-select="open(item.path) "
  on-cancel="openDialogVisible = false">
</file-dialog>

<file-dialog
  mode="save"
  ng:show="saveDialogVisible"
  on-select="saveAs(path) "
  on-cancel="saveDialogVisible = false"
  name="saveAsName">
</file-dialog>
```

Arguments

- **ngShow** (*expression*) –
- **onSelect** (*expression(item)*) – called after opening or saving a file. *item* is an object with a *path* property.
- **onCancel** (*expression*) – (optional) handler for the cancel button
- **mode** (*string*) – one of open, save
- **name** (*binding*) – (optional) name for the saved file
- **path** (*binding*) – (optional) current

path-selector()

An input with a file selection dialog:

```
<path-selector ng:model="filePath"></path-selector>
```

1.6.45 Angular: ajenti.passwd

Services

class passwd()

passwd.list()

Returns promise → array of the users registered in the system

1.6.46 Angular: ajenti.services

Services

class services()

`services.getManagers()`

Returns promise → array of the available service managers

`services.getServices(managerId)`

Returns promise → array of the available services in the ServiceManager

`services.getService(managerId, serviceId)`

Returns promise → object, gets a single service from the manager

`services.runOperation(managerId, serviceId, operation)`

Arguments

- **operation** (*string*) – typically start, stop, restart, reload; depends on the service manager

Returns promise

1.6.47 Angular: ajenti.terminal

Services

class terminals()

`terminals.list()`

Returns promise → array of opened terminal descriptors

`terminals.kill(terminalId)`

Kills a running terminal process

Returns promise

`terminals.create(options)`

Creates a new terminal

Arguments

- **options.command** (*string*) –
- **options.autoclose** (*boolean*) –

Returns promise → new terminal ID

`terminals.full(terminalId)`

Returns promise → full content of the requested terminal

1.6.48 Plugin: aj.plugins.core.api.push

class aj.plugins.core.api.push.**Push**(context)

A service providing push messages to the client.

classmethod get (context)

push (plugin, msg)

Sends a push message to the client.

Parameters

- **plugin** – routing ID
- **msg** – message

register ()

1.6.49 Plugin: aj.plugins.core.api.sidebar

class aj.plugins.core.api.sidebar.**Sidebar**(context)

build ()

Returns a complete tree of sidebar items.

Returns dict

classmethod get (context)

class aj.plugins.core.api.sidebar.**SidebarItemProvider**(context)

Interface for providing sidebar items.

provide ()

Should return a list of sidebar items, each in the following format:

```
{
    'id': 'optional-id',
    'attach': 'category:general', # id of the attachment point or None for
↪top level
    'name': 'Dashboard',
    'icon': 'bar-chart',
    'url': '/view/dashboard',
    'children': [
        ...
    ]
}
```

Returns list(dict)

1.6.50 Plugin: aj.plugins.core.api.tasks

class aj.plugins.core.api.tasks.**Task**(context, *args, **kwargs)

Tasks are one-off child processes with progress reporting. This is a base abstract class.

abort ()

name = None

Display name

push (*plugin, message*)

An interface to `aj.plugins.core.api.push.Push` usable from inside the task's process

report_progress (*message=None, done=None, total=None*)

Updates the task's process info.

Parameters

- **message** – text message
- **done** – number of processed items
- **total** – total number of items

run ()

Override this with your task's logic.

send_log_event (*method, message, *args, **kwargs*)

start ()

Starts the task's process

class `aj.plugins.core.api.tasks.TasksService` (*context*)

abort (*_id*)

format_tasks ()

classmethod **get** (*context*)

notify (*message=None*)

remove (*_id*)

send_update ()

start (*task*)

1.6.51 Plugin: `aj.plugins.augeas.api`

1.6.52 Plugin: `aj.plugins.auth-users.api`

1.6.53 Plugin: `aj.plugins.dashboard.widget`

class `aj.plugins.dashboard.widget.Widget` (*context*)

Base interface for dashboard widgets.

get_value (*config*)

Override this to return the widget value for the given config dict.

id = `None`

name = `None`

Display name

1.6.54 Plugin: aj.plugins.check_certificates.api

1.6.55 Plugin: aj.plugins.datetime.api

1.6.56 Plugin: aj.plugins.network.api

1.6.57 Plugin: aj.plugins.packages.api

1.6.58 Plugin: aj.plugins.power.api

1.6.59 Plugin: aj.plugins.services.api

a

- [aj](#), [45](#)
- [aj.api.endpoint](#), [48](#)
- [aj.api.http](#), [45](#)
- [aj.config](#), [48](#)
- [aj.core](#), [49](#)
- [aj.entry](#), [49](#)
- [aj.http](#), [49](#)
- [aj.plugins](#), [51](#)
- [aj.plugins.core.api.push](#), [62](#)
- [aj.plugins.core.api.sidebar](#), [62](#)
- [aj.plugins.core.api.tasks](#), [62](#)
- [aj.plugins.dashboard.widget](#), [63](#)

j

- [jadi](#), [44](#)

A

`abort()` (*aj.plugins.core.api.tasks.Task method*), 62
`abort()` (*aj.plugins.core.api.tasks.TasksService method*), 63
`ace-editor()` (*built-in function*), 58
`add_header()` (*aj.http.HttpContext method*), 49
`aj` (*module*), 45
`aj.api.endpoint` (*module*), 48
`aj.api.http` (*module*), 45
`aj.config` (*module*), 48
`aj.core` (*module*), 49
`aj.entry` (*module*), 49
`aj.http` (*module*), 49
`aj.plugins` (*module*), 51
`aj.plugins.core.api.push` (*module*), 62
`aj.plugins.core.api.sidebar` (*module*), 62
`aj.plugins.core.api.tasks` (*module*), 62
`aj.plugins.dashboard.widget` (*module*), 63
`all()` (*jadi. method*), 44
`any()` (*jadi. method*), 44
`augeas()` (*class*), 58
`augeas.get()` (*augeas method*), 58
`augeas.set()` (*augeas method*), 58
`AugeasConfig()` (*class*), 58
`AugeasConfig.get()` (*AugeasConfig method*), 58
`AugeasConfig.insert()` (*AugeasConfig method*), 58
`AugeasConfig.match()` (*AugeasConfig method*), 59
`AugeasConfig.matchNodes()` (*AugeasConfig method*), 59
`AugeasConfig.model()` (*AugeasConfig method*), 58
`AugeasConfig.remove()` (*AugeasConfig method*), 58
`AugeasConfig.set()` (*AugeasConfig method*), 58
`AugeasNode()` (*class*), 58
`AugeasNode.children` (*global variable or constant*), 58

`AugeasNode.fullPath()` (*AugeasNode method*), 58
`AugeasNode.name` (*global variable or constant*), 58
`AugeasNode.parent` (*global variable or constant*), 58
`AugeasNode.value` (*global variable or constant*), 58
`autofocus()` (*built-in function*), 56

B

`BaseHttpHandler` (*class in aj.api.http*), 45
`BinaryDependency` (*class in aj.plugins*), 52
`BinaryDependency.Unsatisfied`, 52
`body` (*aj.http.HttpContext attribute*), 49
`build()` (*aj.plugins.core.api.sidebar.Sidebar method*), 62
`build_exception()` (*aj.plugins.Dependency method*), 51
`bytesFilter()` (*built-in function*), 57

C

`check()` (*aj.plugins.Dependency method*), 51
`checkbox()` (*built-in function*), 56
`classes()` (*jadi. method*), 44
`component()` (*in module jadi*), 44
`config` (*in module aj*), 45
`config()` (*class*), 53
`config.data` (*global variable or constant*), 53
`config.getUserConfig()` (*config method*), 53
`config.load()` (*config method*), 53
`config.save()` (*config method*), 53
`config.setUserConfig()` (*config method*), 53
`Context` (*class in jadi*), 45
`core()` (*class*), 53
`core.pageReload()` (*core method*), 54
`core.restart()` (*core method*), 54

D

`debug` (*in module aj*), 45
`delete()` (*in module aj.api.http*), 46

Dependency (class in *aj.plugins*), 51
 Dependency.Unsatisfied, 51
 describe() (*aj.plugins.Dependency.Unsatisfied* method), 51
 describe() (*aj.plugins.PluginCrashed* method), 51
 description (*aj.plugins.BinaryDependency* attribute), 52
 description (*aj.plugins.FileDependency* attribute), 53
 description (*aj.plugins.ModuleDependency* attribute), 52
 description (*aj.plugins.OptionalPluginDependency* attribute), 52
 description (*aj.plugins.PluginDependency* attribute), 52
 deserialize() (*aj.http.HttpContext* class method), 50
 destroy() (*aj.api.http.SocketEndpoint* method), 46
 dialog() (built-in function), 56
 DirectoryPluginProvider (class in *aj.plugins*), 51
 dispatch() (*aj.http.HttpRoot* method), 51
 dump_env() (*aj.http.HttpContext* method), 50

E

endpoint() (in module *aj.api.endpoint*), 48
 EndpointError, 48
 EndpointReturn, 48
 env (*aj.http.HttpContext* attribute), 49
 exit() (in module *aj*), 45

F

fallthrough() (*aj.http.HttpContext* method), 50
 file() (*aj.http.HttpContext* method), 50
 file-dialog() (built-in function), 60
 FileDependency (class in *aj.plugins*), 52
 FileDependency.Unsatisfied, 52
 filesystem() (class), 59
 filesystem.chmod() (*filesystem* method), 59
 filesystem.createDirectory() (*filesystem* method), 59
 filesystem.createFile() (*filesystem* method), 59
 filesystem.downloadBlob() (*filesystem* method), 59
 filesystem.list() (*filesystem* method), 59
 filesystem.read() (*filesystem* method), 59
 filesystem.stat() (*filesystem* method), 59
 filesystem.write() (*filesystem* method), 59
 floating-toolbar() (built-in function), 56
 format_tasks() (*aj.plugins.core.api.tasks.TasksService* method), 63

G

get() (*aj.config.UserConfigService* class method), 48
 get() (*aj.plugins.core.api.push.Push* class method), 62
 get() (*aj.plugins.core.api.sidebar.Sidebar* class method), 62
 get() (*aj.plugins.core.api.tasks.TasksService* class method), 63
 get() (*aj.plugins.PluginManager* class method), 53
 get() (in module *aj.api.http*), 47
 get() (*jadi.* method), 45
 get_cleaned_env() (*aj.http.HttpContext* method), 50
 get_component() (*jadi.Context* method), 45
 get_components() (*jadi.Context* method), 45
 get_content_path() (*aj.plugins.PluginManager* method), 53
 get_crash() (*aj.plugins.PluginManager* method), 53
 get_fqdn() (in module *jadi*), 44
 get_loaded_plugins_list() (*aj.plugins.PluginManager* method), 53
 get_provider() (*aj.config.UserConfigService* method), 48
 get_service() (*jadi.Context* method), 45
 get_value() (*aj.plugins.dashboard.widget.Widget* method), 63
 gzip() (*aj.http.HttpContext* method), 50

H

handle() (*aj.api.http.BaseHttpHandler* method), 45
 handle() (*aj.api.http.HttpMasterMiddleware* method), 45
 handle() (*aj.api.http.HttpMiddleware* method), 46
 handle() (*aj.api.http.HttpPlugin* method), 46
 handle() (*aj.http.HttpMiddlewareAggregator* method), 51
 handle_crash() (in module *aj.entry*), 49
 head() (in module *aj.api.http*), 47
 headers (*aj.http.HttpContext* attribute), 49
 hotkeys() (class), 54
 hotkeys.ENTER, ESC (global variable or constant), 54
 hotkeys.on() (*hotkeys* method), 54
 HttpContext (class in *aj.http*), 49
 HttpMasterMiddleware (class in *aj.api.http*), 45
 HttpMiddleware (class in *aj.api.http*), 46
 HttpMiddlewareAggregator (class in *aj.http*), 50
 HttpPlugin (class in *aj.api.http*), 46
 HttpRoot (class in *aj.http*), 51

I

id (*aj.plugins.dashboard.widget.Widget* attribute), 63
 identity() (class), 54
 identity.auth() (*identity* method), 54

[identity.effective](#) (*global variable or constant*), [54](#)
[identity.elevate\(\)](#) (*identity method*), [54](#)
[identity.isSuperuser](#) (*global variable or constant*), [54](#)
[identity.login\(\)](#) (*identity method*), [54](#)
[identity.logout\(\)](#) (*identity method*), [54](#)
[identity.machine.name](#) (*global variable or constant*), [54](#)
[identity.user](#) (*global variable or constant*), [54](#)
[init\(\)](#) (*in module aj*), [45](#)
[interface\(\)](#) (*in module jadi*), [44](#)
[is_satisfied\(\)](#) (*aj.plugins.BinaryDependency method*), [52](#)
[is_satisfied\(\)](#) (*aj.plugins.FileDependency method*), [53](#)
[is_satisfied\(\)](#) (*aj.plugins.ModuleDependency method*), [52](#)
[is_satisfied\(\)](#) (*aj.plugins.OptionalPluginDependency method*), [52](#)
[is_satisfied\(\)](#) (*aj.plugins.PluginDependency method*), [52](#)

J

[jadi](#) (*module*), [44](#)
[json_body\(\)](#) (*aj.http.HttpContext method*), [50](#)

L

[load_all_from\(\)](#) (*aj.plugins.PluginManager method*), [53](#)

M

[messagebox\(\)](#) (*class*), [54](#)
[messagebox.show\(\)](#) (*messagebox method*), [54](#)
[method](#) (*aj.http.HttpContext attribute*), [49](#)
[ModuleDependency](#) (*class in aj.plugins*), [52](#)
[ModuleDependency.Unsatisfied](#), [52](#)

N

[name](#) (*aj.plugins.core.api.tasks.Task attribute*), [62](#)
[name](#) (*aj.plugins.dashboard.widget.Widget attribute*), [63](#)
[ng-enter\(\)](#) (*built-in function*), [57](#)
[NoImplementationError](#), [45](#)
[notify\(\)](#) (*aj.plugins.core.api.tasks.TasksService method*), [63](#)
[notify\(\)](#) (*class*), [55](#)
[notify.custom\(\)](#) (*notify method*), [55](#)
[notify.error\(\)](#) (*notify method*), [55](#)
[notify.info\(\)](#) (*notify method*), [55](#)
[notify.success\(\)](#) (*notify method*), [55](#)
[notify.warning\(\)](#) (*notify method*), [55](#)

O

[on_connect\(\)](#) (*aj.api.http.SocketEndpoint method*), [46](#)
[on_disconnect\(\)](#) (*aj.api.http.SocketEndpoint method*), [46](#)
[on_message\(\)](#) (*aj.api.http.SocketEndpoint method*), [46](#)
[OptionalPluginDependency](#) (*class in aj.plugins*), [52](#)
[OptionalPluginDependency.Unsatisfied](#), [52](#)
[ordinalFilter\(\)](#) (*built-in function*), [57](#)

P

[pageFilter\(\)](#) (*built-in function*), [57](#)
[pageTitle\(\)](#) (*class*), [55](#)
[pageTitle.set\(\)](#) (*pageTitle method*), [55](#)
[passwd\(\)](#) (*class*), [60](#)
[passwd.list\(\)](#) (*passwd method*), [60](#)
[patch\(\)](#) (*in module aj.api.http*), [47](#)
[path](#) (*aj.http.HttpContext attribute*), [49](#)
[path-selector\(\)](#) (*built-in function*), [60](#)
[platform](#) (*in module aj*), [45](#)
[platform_string](#) (*in module aj*), [45](#)
[platform_unmapped](#) (*in module aj*), [45](#)
[plugin](#) (*aj.api.http.SocketEndpoint attribute*), [46](#)
[PluginCrashed](#), [51](#)
[PluginDependency](#) (*class in aj.plugins*), [52](#)
[PluginDependency.Unsatisfied](#), [52](#)
[PluginLoadError](#), [51](#)
[PluginManager](#) (*class in aj.plugins*), [53](#)
[PluginProvider](#) (*class in aj.plugins*), [51](#)
[post\(\)](#) (*in module aj.api.http*), [47](#)
[progress-spinner\(\)](#) (*built-in function*), [57](#)
[provide\(\)](#) (*aj.plugins.core.api.sidebar.SidebarItemProvider method*), [62](#)
[provide\(\)](#) (*aj.plugins.DirectoryPluginProvider method*), [51](#)
[provide\(\)](#) (*aj.plugins.PluginProvider method*), [51](#)
[provide\(\)](#) (*aj.plugins.PythonPathPluginProvider method*), [51](#)
[Push](#) (*class in aj.plugins.core.api.push*), [62](#)
[push\(\)](#) (*aj.plugins.core.api.push.Push method*), [62](#)
[push\(\)](#) (*aj.plugins.core.api.tasks.Task method*), [62](#)
[push\(\)](#) (*class*), [55](#)
[put\(\)](#) (*in module aj.api.http*), [47](#)
[PythonPathPluginProvider](#) (*class in aj.plugins*), [51](#)

Q

[query](#) (*aj.http.HttpContext attribute*), [49](#)

R

[reason\(\)](#) (*aj.plugins.BinaryDependency.Unsatisfied*

method), 52
reason() (*aj.plugins.Dependency.Unsatisfied method*), 51
reason() (*aj.plugins.FileDependency.Unsatisfied method*), 53
reason() (*aj.plugins.ModuleDependency.Unsatisfied method*), 52
reason() (*aj.plugins.OptionalPluginDependency.Unsatisfied method*), 52
reason() (*aj.plugins.PluginDependency.Unsatisfied method*), 52
redirect() (*aj.http.HttpContext method*), 50
register() (*aj.plugins.core.api.push.Push method*), 62
remove() (*aj.plugins.core.api.tasks.TasksService method*), 63
remove_header() (*aj.http.HttpContext method*), 50
report_progress() (*aj.plugins.core.api.tasks.Task method*), 63
requests_decorator_generator() (*in module aj.api.http*), 47
respond() (*aj.http.HttpContext method*), 50
respond_forbidden() (*aj.http.HttpContext method*), 50
respond_not_found() (*aj.http.HttpContext method*), 50
respond_ok() (*aj.http.HttpContext method*), 50
respond_server_error() (*aj.http.HttpContext method*), 50
respond_unauthenticated() (*aj.http.HttpContext method*), 50
response_ready (*aj.http.HttpContext attribute*), 49
restart() (*in module aj*), 45
restart() (*in module aj.core*), 49
root-access() (*built-in function*), 57
run() (*aj.plugins.core.api.tasks.Task method*), 63
run() (*in module aj.core*), 49
run_response() (*aj.http.HttpContext method*), 50

S

send() (*aj.api.http.SocketEndpoint method*), 46
send_log_event() (*aj.plugins.core.api.tasks.Task method*), 63
send_update() (*aj.plugins.core.api.tasks.TasksService method*), 63
serialize() (*aj.http.HttpContext method*), 50
server (*in module aj*), 45
service() (*in module jadi*), 44
services() (*class*), 61
services.getManagers() (*services method*), 61
services.getService() (*services method*), 61
services.getServices() (*services method*), 61
services.runOperation() (*services method*), 61
Sidebar (*class in aj.plugins.core.api.sidebar*), 62

SidebarItemProvider (*class in aj.plugins.core.api.sidebar*), 62
smart-progress() (*built-in function*), 57
SocketEndpoint (*class in aj.api.http*), 46
spawn() (*aj.api.http.SocketEndpoint method*), 46
start() (*aj.plugins.core.api.tasks.Task method*), 63
start() (*aj.plugins.core.api.tasks.TasksService method*), 63
start() (*in module aj.entry*), 49

T

Task (*class in aj.plugins.core.api.tasks*), 62
tasks() (*class*), 55
tasks.start() (*tasks method*), 55
tasks.tasks (*global variable or constant*), 55
TasksService (*class in aj.plugins.core.api.tasks*), 63
terminals() (*class*), 61
terminals.create() (*terminals method*), 61
terminals.full() (*terminals method*), 61
terminals.kill() (*terminals method*), 61
terminals.list() (*terminals method*), 61

U

url() (*in module aj.api.http*), 47
UserConfigService (*class in aj.config*), 48

V

value (*aj.plugins.Dependency attribute*), 52
version (*in module aj*), 45

W

Widget (*class in aj.plugins.dashboard.widget*), 63

Y

yaml_loader (*aj.plugins.Dependency attribute*), 52
yaml_tag (*aj.plugins.BinaryDependency attribute*), 52
yaml_tag (*aj.plugins.Dependency attribute*), 52
yaml_tag (*aj.plugins.FileDependency attribute*), 53
yaml_tag (*aj.plugins.ModuleDependency attribute*), 52
yaml_tag (*aj.plugins.OptionalPluginDependency attribute*), 52
yaml_tag (*aj.plugins.PluginDependency attribute*), 52