Automate web application deployment using Ansible

(GLPI, FusionInventory, Fail2ban)

By Ershad RAMEZANI

Table of Contents

Preface

To make its investments profitable, AdrarNum wants to expand its service offering with turnkey[1] Cloud solutions. The strategy consists of gradually migrating the services maintained and hosted by customers to its data center, moving from a classic IT service provider to a cloud service provider.

The goal: GLPI service based on PaaS[2]

AdrarNum wants to add to its catalog a GLPI solution in PaaS installed and preconfigured in less than an hour.

After product delivered, AdrarNum is responsible for updates on:

- Web server
- PHP latest version
- RDBMS[3]
- GLPI and its plugins

As the trend around deployment and configuration management products leaning towards Ansible, you are responsible for proposing a procedure for installing and maintaining the GLPI servers of future customers.

List of demands

- Installation of GLPI and all of the necessary components on a virtual machine that is dedicated to the customer using Ansible.
- Customers will only have access to servers in https.
- Only PHP versions higher than version 7 will be supported.
- The server administration tasks are the responsibility of AdrarNum but the GLPI application itself is the client's responsibility. For this purpose, upon delivery, the client will be provided with a URL, a username and password with access to GLPI.
- AdrarNum will also provide the client with a Fusion-Inventory agent installation script which will install the agent on client workstations from a download server.
- The client will have the choice to create local account into GLPI or to use an LDAP server of their choice.
- Securing the client server with Fail2Ban.

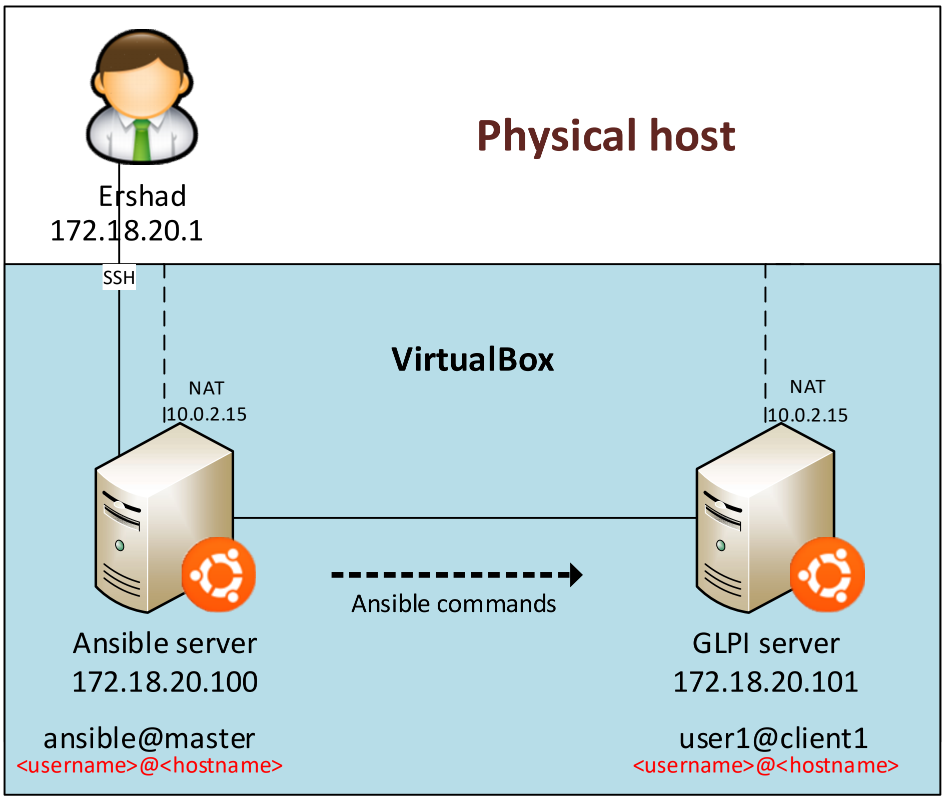
Project's Overview

For the purposes of this project, we'll start by introducing ansible, why it's the best choice, explaining how it works and its various components. Then we will install ansible on an Ubuntu-based machine and perform the basic configuration to start the communication between the ansible server and the node.

The next step will be the creation of a LAMP[4] stack which is needed for the creation of our GLPI web application. we will start by creating a playbook that will deploy an apache web server and then we will add the other components one after the other by creating roles. Final step would be showing how to add the fusion-inventory plugin and fail2ban inside GLPI server. Progress as we go along will allow us to introduce the different modules available in Ansible.

[1] A turnkey or a turnkey project is a type of project that is constructed so that it can be sold to any buyer as a completed product.
[2] Platform as a Service is a type of cloud computing service model that offers a flexible, scalable cloud platform to develop, deploy, run, and manage apps.
[3] A relational database management system is a program used to create, update, and manage relational databases. Some of the most well-known RDBMSs include MySQL, PostgreSQL, MariaDB, Microsoft SQL Server, and Oracle Database.
[4] A LAMP stack is a bundle of four different software technologies that developers use to build websites and web applications. LAMP is an acronym for the operating system, Linux; the web server, Apache; the database server, MySQL; and the programming language, PHP.

For this lab I'm using VirtualBox. I installed two Ubuntu-server 22.04 VMs with host-only network interfaces in 172.18.20.0/24. I also have a Nat network interface which provides internet access to VMs for downloading packages, etc.



IT Process Automation (ITPA)

IT automation is the process of creating software and systems to replace repeatable processes and reduce manual work in data centers and cloud deployments. Automation tools, also called configuration management tools, conduct the tasks with minimum administrator intervention.

Puppet, Chef, Ansible, and SaltStack are four industry-leading configuration management tools that offer different paths to achieve a same goal in IT automation.

Type of IT automation

The most important way to classify IT automation is how the configuration are propagated. Based on this, we can distinguish between agent-based systems and agent-less systems.

Agent-based systems

Agent-based systems have two different components: a server, and a client called agent. Periodically, the client will contact the server to see if a new configuration for its machine is present. If a new configuration is present, the client will download it and apply it.

Agent-less systems

In agent-less systems, no specific agent is present. Communications are initialized by the server, which will contact the clients using standard protocols (usually via SSH and PowerShell).

Which one is better

An agent-based system can be less secure. Since all machines have to be able to initiate a connection to the server which could be attacked more easily than in an agent-less case. In an agent-based system, it is not possible to push an update immediately to client machines. Server will have to wait until those machines check-in. Tools such as Chef and Puppet are agent-based and ansible is agent-less.

What is Ansible

Ansible is an open source IT automation tool that automates provisioning, configuration management, application deployment, orchestration, and many other manual IT processes. Ansible users can use Ansible automation to install software, automate daily tasks, provision infrastructure and many more.

How does ansible work

Ansible works by connecting to client machines and pushing out small programs - called modules - to these machines also called nodes. Modules are used to accomplish automation tasks in Ansible. Ansible executes these modules and removes them when finished. Without modules, you'd have to rely on ad-hoc commands and scripting to accomplish tasks.

What's So Great About Ansible?

Easy-to-read syntax

Ansible uses the YAML file format which are easy to use. Ansible configuration management scripts are called playbooks. Ansible builds the playbook syntax on top of YAML, which is a language that was designed to be easy for humans to read and write.

Agentless

To manage servers with Ansible, Linux servers need to have SSH and Python installed, while Windows servers need WinRM[5] enabled and PowerShell. So there is no need to preinstall an agent or any other software on the host. On the control machine, it is best to install Python 3.8 or later.

We can easily achieve security by securing SSH and WinRM protocols with strong configurations and firewall settings.

Lots of modules

You use modules to perform tasks such as installing a package, restarting a service, or copying a configuration file. Ansible modules are declarative, it means you use them to describe the state you want the server to be in with no need to define which steps the module should take.

Procedural vs declarative

Procedural language outlines a specific set of steps that must be taken in order to arrive at the desired outcome. A declarative language tells the program what needs to be done and allows the program to figure out the necessary steps so the focus is on the outcome instead of how to get there.

Agentless and push-based

Ansible is push-based by default. Making a change looks like this:

- You: make a change to a playbook.
- You: run the new playbook.
- Ansible: connects to servers and executes modules that change the state of the servers.

Idempotency

Modules are also idempotent. It means that it is safe to run an Ansible playbook multiple times against a server. For example when we need a user, if the user does not exist, Ansible will create it. If it does exist, Ansible will not do anything.

OpenSSH Authentication Explained

SSH service is required for the ansible to work. You can install OpenSSH service on both node server and ansible server with this command:

```
sudo apt install openssh-server
```

[5] Windows Remote Management is a Windows-native built-in remote management protocol that lets network administrators access, edit and update data from local and remote computers.

In reality we are going to be managing multiple hosts. By default this collection of hosts is listed in a file named *hosts* in */etc/ansible/hosts* directory. Inventory file's default address is defined in *ansible.cfg* file and can be modified.

Hosts file

The simplest way to list the servers in the *hosts* file is by listing the hostnames or IP addresses. If we use hostnames, they must be translatable by a DNS.

```
glpiserver.adrarnum.local #DNS record for a hostname is needed to be created on DNS server
```

Or

```
172.18.20.101
```

Behavioral Inventory Parameters

Instead of hostname or IP address we can use an *alias*. But in this case we have to define their hostname or IP address with variables. This kind of variable is called Behavioral Inventory Parameter. Here is a list of them with their descriptions and default value:

Name	**Default**	**Description**
Ansible_host	Name of host (alias)	Hostname or ip address to SSH to
Ansible_port	22	Port to SSH to
Ansible_user	User that is executing ansible	User to SSH as
Ansible_password	(None)	Password to use for SSH authentication

Each host has a default value for these parameters which can be overridden in ansible. We can change the default value in inventory *hosts* file or in *ansible.cfg*'s *[default]* section. For some of these Behavioral Inventory Parameter there is an equivalent option in *ansible.cfg*:

Behavioral Inventory Parameter	**Ansible.cfg option**
ansible_port	Remote_port
Ansible_user	Remote_user

An example of how to define them in *hosts* file:

```
glpiserver ansible_host=172.18.20.101 ansible_port=22 ansible_user=ansible
```

And the example in *ansible.cfg* (Notice that we are overriding these parameters' default value):

```
[default]
remote_port=2220
remote_user=ansible
```

Note that user could be overridden in playbook itself with the same *remote_user=user1* variable which has a priority to the two other variables for defining remote user.

Groups

We can perform configuration actions on groups of hosts, rather than on an individual host. Ansible automatically defines a group called *all*, which includes all the hosts in the inventory. We can define our specific groups:

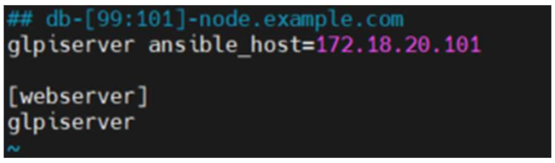
```
[webservers]
glpiserver ansible_host=172.18.20.101 ansible_port=2220 ansible_user=user1
```

Group vars

We can define variables for a group of hosts:

```
[webservers]
glpiserver ansible_host=172.18.20.101
[webservers:vars]
ansible_port=2220
ansible_password=12345
```

here is how our hosts file looks like at this moment:


```
## db-[99:101]-node.example.com
glpiserver ansible_host=172.18.20.101

[webserver]
glpiserver
~
```

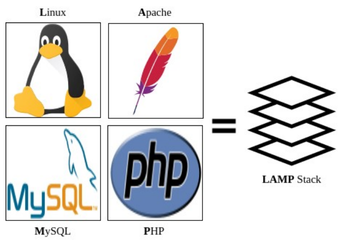
Web application and LAMP

Web application

A Web application is an application program that is stored on a remote server and delivered over the Internet through a browser interface.

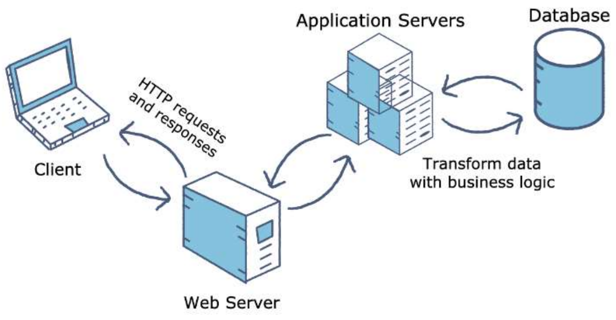
LAMP

A LAMP stack is a bundle of four different software technologies that developers use to build websites and web applications. LAMP is an acronym for the operating system, Linux; the web server, Apache; the database server, MySQL; and the programming language, PHP.



GLPI

GLPI is an open source IT Asset Management, issue tracking system and service desk system. This software is written in PHP and distributed as open-source software under the GNU General Public License. GLPI is a web-based application helping companies to manage their information system.


Application Servers
Database
HTTP requests and responses
Client
Transform data with business logic
Web Server

We can use LAMP stack to configure a GLPI server. To do that with ansible automation, we need to first create a LAMP stack in our infrastructure. A LAMP stack can be installed on one server or each of its software on a separate server depending on our infrastructure size and needs.

Organizing our GLPI Project

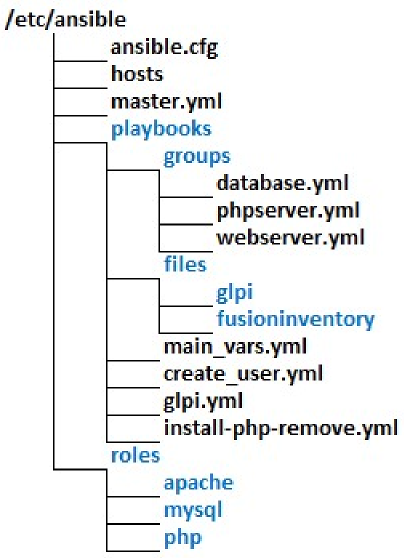
Choosing the best practice to create our ansible repository is an important action to take. We are going to present a pattern to organize an ansible project in an easy way. (Entire infrastructure is available in my Github, link in the last page of the article). Here are the rules to consider for this pattern:

- We will create our ansible project in ansible root directory in */etc/ansible*.
- In this directory we need to have three files:
    - *ansible.cfg*: Ansible configuration file we discussed earlier.
    - *hosts*: We also talked about inventory *hosts* file earlier.
    - *master.yml*: Which is a playbook that aligns the whole infrastructure.

Ansible.cfg and hosts already exist in */etc/ansible* but we will create *master.yml*.

- In addition to those three files, we create <u>two</u> folders:
  - *Playbooks*: This will contain:
    - The playbooks in yaml format.
    - *main_vars.yml* file for playbooks' variables.
    - *files* folder to place files and folders needed by playbooks.
    - *groups* folder for group management (explained later)
  - *Roles*: This will contain all of the roles we need in our project (explained in next section)

An empty roles folder exists by default in ansible root repository, but we will create the playbooks folder.



```
/etc/ansible
    ansible.cfg
    hosts
    master.yml
    playbooks
        groups
            database.yml
            phpserver.yml
            webserver.yml
        files
            glpi
            fusioninventory
        main_vars.yml
        create_user.yml
        glpi.yml
        install-php-remove.yml
    roles
        apache
        mysql
        php
```

In the schema you can see the structure of the ansible repository for our GLPI web application which needs a web server, a database server and a php server (considering each LAMP software on a separate server).

You can read more about best practices in ansible official website:

https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html

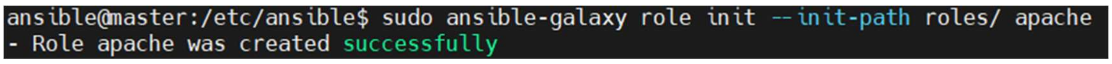
Roles

Playbooks are very good at executing operations, but they are not very good at configuring huge amounts of machines, because they will soon become messy. To solve this, Ansible has roles.

A role is a set of playbooks, templates, files, or variables used to achieve a specific goal. For instance, we could have a database role and a web server role so that those configurations stay cleanly separated. We can give each role to a different server or several roles to a server.

Anatomy of a role

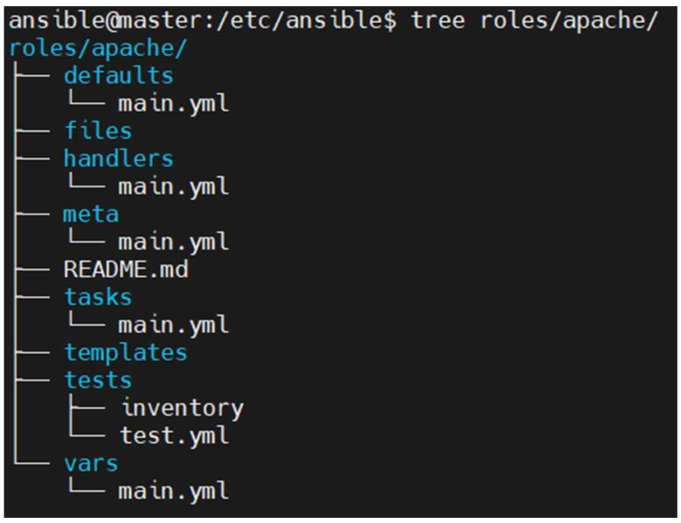
The structure of folders in a role is standard and you cannot change it much. You can create a role files and directories with ansible-galaxy command. Here is our first role, *apache*, created:



```
ansible@master:/etc/ansible$ sudo ansible-galaxy role init --init-path roles/ apache
- Role apache was created successfully
```

Apache role

Running this command creates the following files and directories for *apache* role:



```
ansible@master:/etc/ansible$ tree roles/apache/
roles/apache/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

The most important folder within the role is the *tasks* folder, because this is the only mandatory folder in it. It has to contain a *main.yml* file that will be the list of tasks to be executed. Other folders that are often present in the roles are *templates* and *files*. The first one will be used to store templates used by the *template* task, while the second will be used to store files that are used by the *copy* task.
We can transform our *configure_apache.yml* playbook into a role. We used a template named *virtualhost.conf.j2* for that playbook, so we have to place that file properly in the *roles/apache/templates* folder.

We should also place the *webserver* folder with its *index.html* file into the apache role's *files* folder as our playbook expecting *webserver* folder to be present in *files* folder.

We transfer the *configure_apache.yml* file content to the *webserver/tasks/main.yml* file. There are only two changes to make:

1. The *hosts*, *remote_user(or user)*, *become*, *vars* and *tasks* lines should be deleted.
2. The indentation of the rest of the file should be fixed accordingly.

task/main.yml

The following is the code for our apache in *tasks/main* to show how the file starts and the indentations. Note that the *handlers* are also omitted.

```
---
- name: install apache
  apt:
    name: apache2
    update_cache: true
    cache_valid_time: 3600
    state: latest
  notify: Restart Apache

- name: Restart Apache
  meta: flush_handlers

- name: copy the virtualhost to the node
  template:
    src: virtualhost.conf.j2
    dest: "/etc/apache2/sites-available/{{ project_name }}.conf"
  notify: Restart Apache

- name: activate the virtualhost
  file:
    src: "/etc/apache2/sites-available/{{ project_name }}.conf"
    dest: "/etc/apache2/sites-enabled/{{ project_name }}.conf"
    state: link
    mode: '0777'
  notify: Restart Apache

- name: transfer website content to the node
  synchronize:
    src: "{{ project_name }}"
    dest: "/var/www/"

- name: disable apache default page
  shell: /usr/sbin/a2dissite 000-default.conf
  when: disable_default

- name: create key pair on the node
  command: >
    sudo openssl req -new -newkey rsa:4096 -x509
    -sha256 -days 365 -nodes -out {{ project_name }}.crt
    -keyout {{ project_name }}.key
    -subj '/CN={{ hostname }}.{{ project_name }}' -days 365
  args:
    chdir: "{{ apache_conf_path }}"
    creates: "{{ apache_conf_path }}/{{ project_name }}.crt"
  notify: Restart Apache

- name: enable ssl module
  shell: /usr/sbin/a2enmod ssl

- name: enable rewrite module
  tags: module
  shell: /usr/sbin/a2enmod rewrite
  notify: Restart Apache
```

Variable Precedence

We've seen several ways of defining variables:

- vars: parameter directly inside a play,
- vars_files: parameter which points to a yaml file elsewhere,
- inside the roles as role's default variable files,
- in the inventory hosts file and

```
        state: restarted
```

Let’s add our playbook to the *master.yml* file:

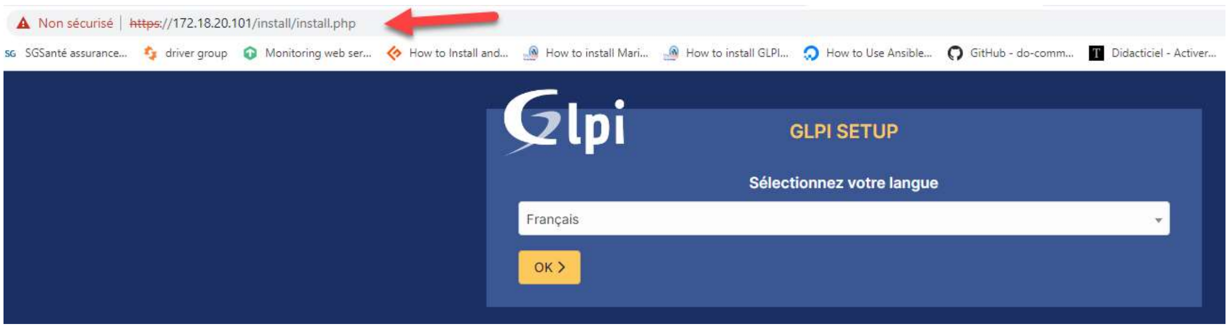
```
---
- import_playbook: playbooks/groups/webserver.yml
- import_playbook: playbooks/groups/database.yml
- import_playbook: playbooks/groups/phpserver.yml
- import_playbook: playbooks/glpi.yml
```

Final execution and verification of GLPI

We will execute the master file again with this command:

```
ansible-playbook master.yml --skip-tag=set_root_pass
```

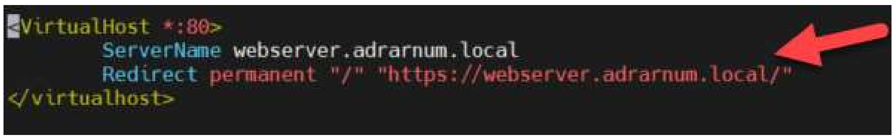
When the playbook execution is finished, we can open the *glpi* webpage on https://172.18.20.101:


Non sécurisé | ~~https~~://172.18.20.101/install/install.php
GLPI SETUP
Sélectionnez votre langue
Français
OK

Remember that we configured redirection in the virtualhost so opening *http*, redirects toward this link:

https://{{ project_name }}.{{ domain_name }}/

Here is the link in virtualhost on the node according to the variables we defined in *main_vars.yml*:

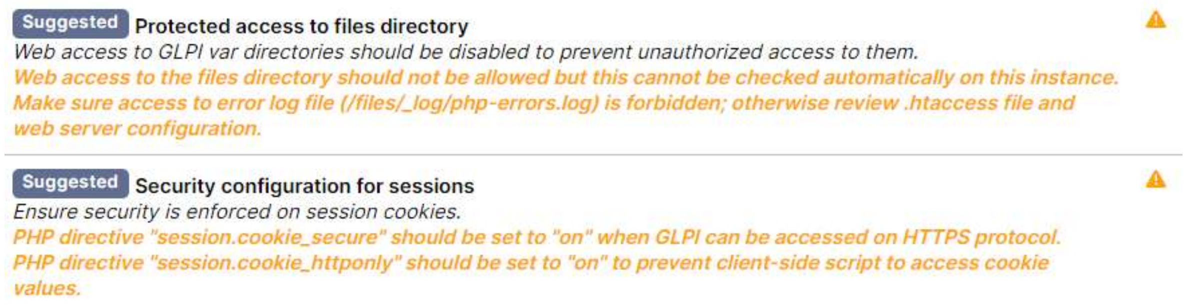


```
<VirtualHost *:80>
        ServerName webserver.adrarnum.local
        Redirect permanent "/" "https://webserver.adrarnum.local/"
</virtualhost>
```

Therefore, we need to add this address into our DNS our client machines *hosts* file.

Compatibility checks

After selecting the language and finishing up with basic configuration, we will have a page for compatibility check. If the installation is successful as we planned, apart from two, all the other tests must a check valid. Here are the two:



Suggested **Protected access to files directory**
Web access to GLPI var directories should be disabled to prevent unauthorized access to them.
Web access to the files directory should not be allowed but this cannot be checked automatically on this instance. Make sure access to error log file (/files/_log/php-errors.log) is forbidden; otherwise review .htaccess file and web server configuration.

Suggested **Security configuration for sessions**
Ensure security is enforced on session cookies.
PHP directive "session.cookie_secure" should be set to "on" when GLPI can be accessed on HTTPS protocol. PHP directive "session.cookie_httponly" should be set to "on" to prevent client-side script to access cookie values.

In the first warning, as it says the log file should not be accessible. We can test it simply by trying to access to this *url*: