The BackStage

RH Forum 2018

by Moisés Rivera
Team Lead for Cloud, Automation and Infrastructure

The BackStage

WHAT IS THE BACKSTAGE?

The Backstage is the laboratory of Ansible prepared for its realization in the Red Hat Forum 2018.

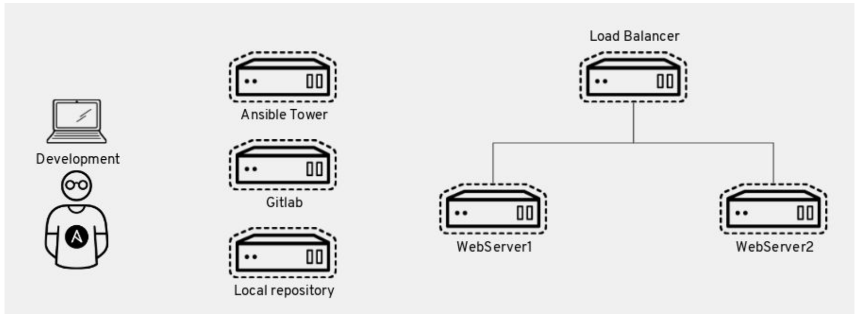
During the next two hours we will perform all the necessary operations to create the demo environment, used by the Red Hat Solution Architect, and that you may have seen in some of his presentations on Ansible.

Get ready, because we are going to work mainly with Ansible Tower, but we are also going to carry out operations with Gitlab, we will create workflows, surveys, credentials ... and the occasional playbook :)

WORK ENVIRONMENT

To start working, we will have a laptop, where you are reading this document :) with a series of virtual machines already created to be able to gain time and thus be able to perform all the operations that lie ahead.

The following diagram shows all the elements that we are going to use.



The following table shows the data of each of the elements:

FQDN	IP	Rol
localrepo.rhforum.com	192.168.110.160	Local repository with all the software necessary to perform the laboratory.
gitlab.rhform.com	192.168.110.161	Local instance of Gitlab that we will use as SCM of the laboratory.
tower.rhforum.com	192.168.110.162	Instance of Ansible Tower.
dev.rhforum.com	192.168.110.1	Development machine and host running all the VMs.
lb.rhforum.com	192.168.110.163	Load balancer and entry point to the Web Servers.
web1.rhforum.com	192.168.110.164	Web Server #1
web2.rhforum.com	192.168.110.165	Web Server #2

In addition to the above information, we will need the credentials that we will use during the lab. The following table shows the necessary user and password information.

User	Password	Rol
rhforum	rhforum	Standard work user.
rhforum	rhforum123.	User to use in GitLab.
admin	rhforum	Administrator user for Ansible Tower.

All the machines have the direct access as root via SSH disabled, but in case of needing it, comment it to verify if it is strictly necessary.

A priori, these are the data we need to get to work; so let's move forward, right? :)

METHODOLOGY

The methodological basis on which the laboratories are based is IaC (aka Infrastructure as Code).

From Wikipedia. https://en.wikipedia.org/wiki/Infrastructure_as_Code

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.[1] The IT infrastructure managed by this comprises both physical equipment such as bare-metal servers as well as virtual machines and associated configuration resources. The definitions may be in a version control system. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.

IaC approaches are promoted for cloud computing, which is sometimes marketed as infrastructure as a service (IaaS). IaC supports IaaS, but should not be confused with it.[1]

If we summarize this approach, we basically have:

- Definition of all our infrastructure, including services, in flat text configuration files.
- Use of a tool - Ansible - for the automated and orchestrated deployment of the defined services.
- Use of a single source of truth - code repository - as source and repository of the know-how of our project.

Certainly, we will perform manual operations, since the tools need a minimum configuration in order to start working ... what was before, the chicken or the egg? :)

LABORATORIES

During the remaining two hours, we will try to perform the following actions:

- Deployment of the production environment (aka, loadbalancers and webservers). Day 1 tasks.
- Modification of configurations in the production environment and content upload. Day 2 tasks.
- BONUS TRACK: Modification of SSH access to production machines.

In order to carry out all operations, we start with the infrastructure created, the Ansible code already written, and the vast majority of configured elements. If the reader is interested in how it was done, please, do not stop asking;)

The laboratories are designed to be carried out in a sequential order. Once done, we can interact with them the way we want.

LAB # 0. CLONING THE REPOSITORY CODE.

This laboratory is not absolutely necessary, but it is highly recommended to carry out the inspection of the code.

As the reader knows, there are many and diverse repositories of code, and it is not a new tool, this type of tools have been used by developers, since time immemorial :) but now it is our turn to the systems also use this type of tools , so without any fear, here we go :)

We can say that - currently - the standard repository is GitHub (https://github.com), but nothing is further from reality, since the most important thing is that we choose the repository that we choose, it is compatible between different brands; and that's where Git comes in.

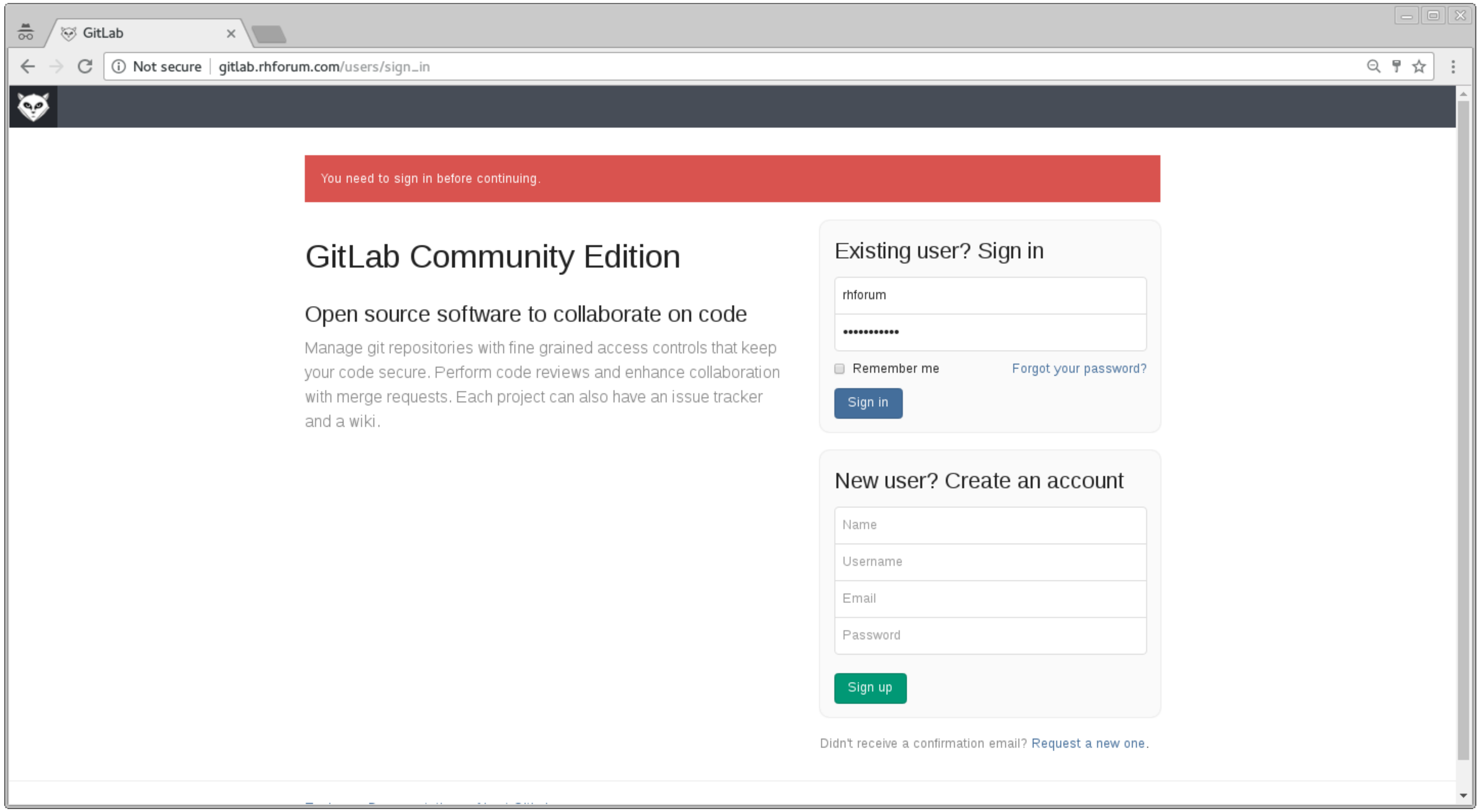
We for this lab have chosen the community version of GitLab https://about.gitlab.com. We have downloaded this version, and we have installed it in one of our VMs.

To be able to enter the tool, we will open a browser, and type in the address of the tool, which in our case is:

```
http://gitlab.rhforum.com
```

and a screen like the one shown on the next page will appear. Now we will use the credentials that have been commented at the beginning of the document to be able to login with our laboratory user.

```
User: rhforum
Pass: rhforum123.
```



You need to sign in before continuing.

GitLab Community Edition

Open source software to collaborate on code

Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Existing user? Sign in

rhforum

••••••••••••

Remember me   Forgot your password?

Sign in

New user? Create an account

Name

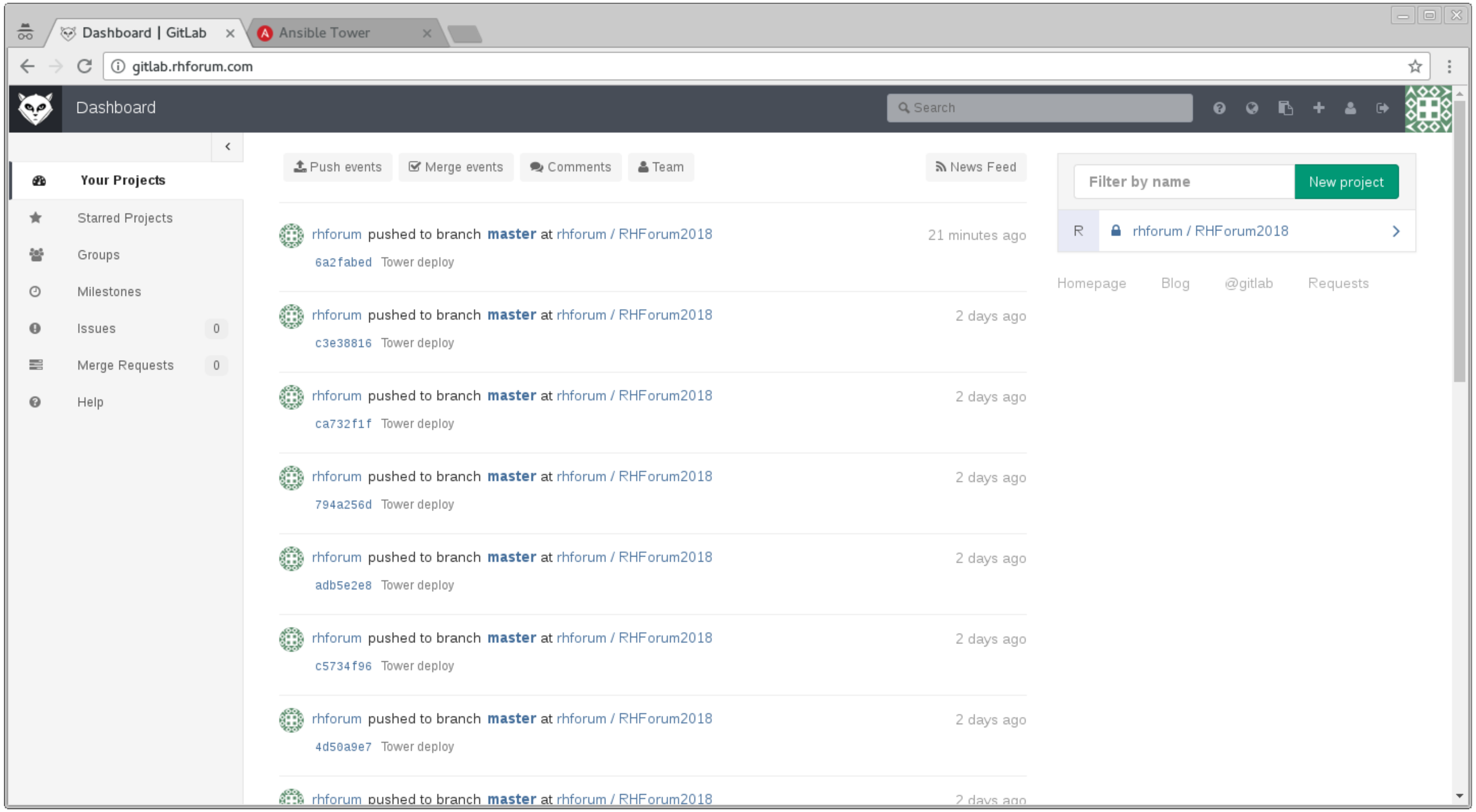
Username

Email

Password

Sign up

Didn't receive a confirmation email? Request a new one.



rhforum / RHForum2018

rhforum pushed to branch **master** at rhforum / RHForum2018 — 21 minutes ago
6a2fabed Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
c3e38816 Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
ca732f1f Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
794a256d Tower deploy

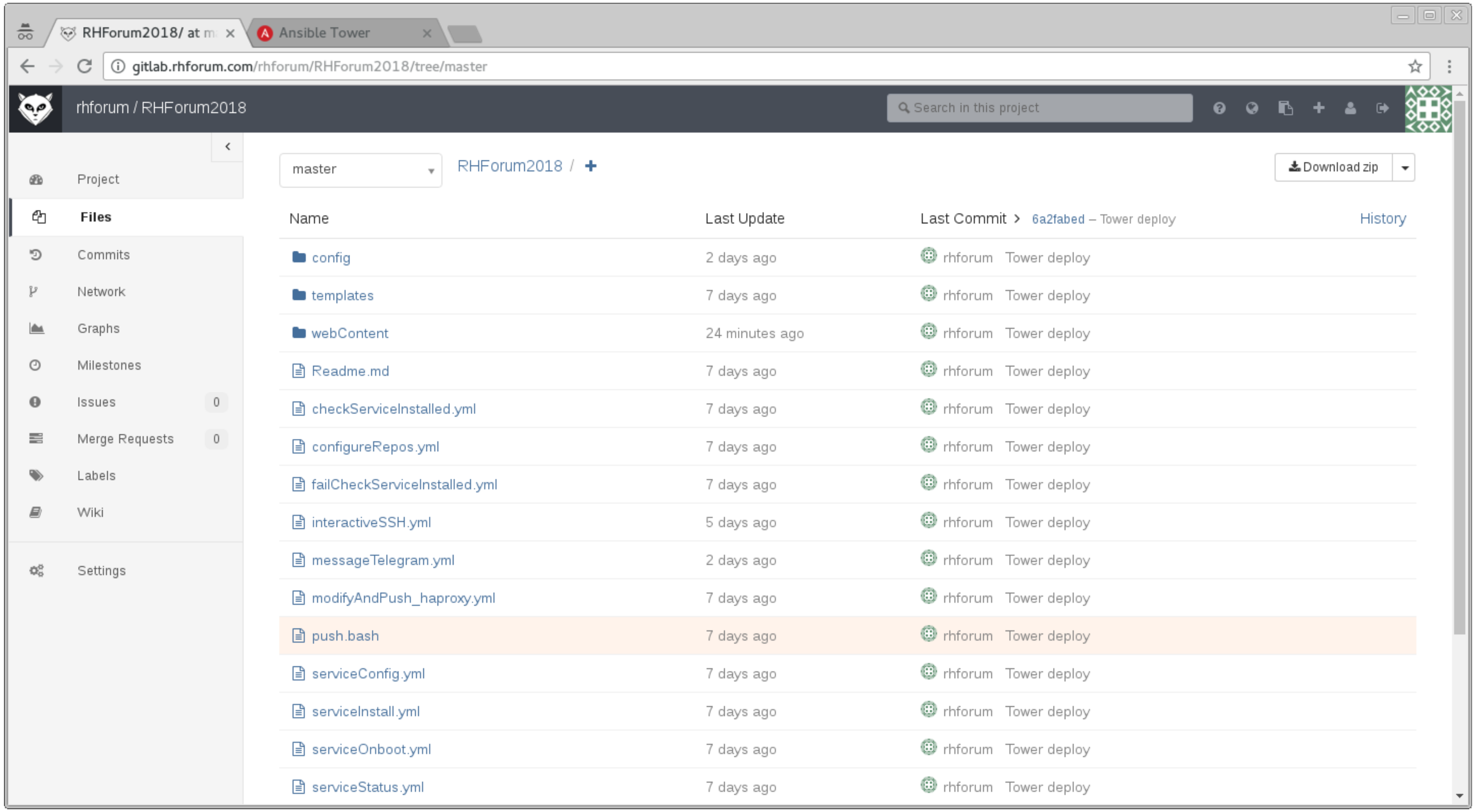
rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
adb5e2e8 Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
c5734f96 Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago
4d50a9e7 Tower deploy

rhforum pushed to branch **master** at rhforum / RHForum2018 — 2 days ago

As we can see, we have already created a project, RHForum2018. If we click on the link that appears on the right, we enter the project, and a new menu appears on the right. By clicking on Files, we can see the different files that comprise it.



gitlab.rhforum.com/rhforum/RHForum2018/tree/master

rhforum / RHForum2018

master — RHForum2018 / +

Name	Last Update	Last Commit > 6a2fabed – Tower deploy
config	2 days ago	rhforum Tower deploy
templates	7 days ago	rhforum Tower deploy
webContent	24 minutes ago	rhforum Tower deploy
Readme.md	7 days ago	rhforum Tower deploy
checkServiceInstalled.yml	7 days ago	rhforum Tower deploy
configureRepos.yml	7 days ago	rhforum Tower deploy
failCheckServiceInstalled.yml	7 days ago	rhforum Tower deploy
interactiveSSH.yml	5 days ago	rhforum Tower deploy
messageTelegram.yml	2 days ago	rhforum Tower deploy
modifyAndPush_haproxy.yml	7 days ago	rhforum Tower deploy
push.bash	7 days ago	rhforum Tower deploy
serviceConfig.yml	7 days ago	rhforum Tower deploy
serviceInstall.yml	7 days ago	rhforum Tower deploy
serviceOnboot.yml	7 days ago	rhforum Tower deploy
serviceStatus.yml	7 days ago	rhforum Tower deploy

The first action that we are going to carry out is the cloning of the same in our host to be able to inspect its content, although evidently, we can also do it directly from GitLab.

The first operation to be performed is to create an SSH key and put it inside our user profile in GitLab; In this way, you will not be asking us for passwords if we perform a push.

To generate the SSH key, we must proceed as follows:

```
[rhforum@th88 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/rhforum/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rhforum/.ssh/id_rsa.
Your public key has been saved in /home/rhforum/.ssh/id_rsa.pub.
The key fingerprint is:
77:a0:41:d9:0b:51:e2:75:f4:7c:e8:d8:9e:e0:27:a2 rhforum@th88
The key's randomart image is:
+--[ RSA 2048]----+
|        +=o.o    |
|       ooo.. o . |
|        o...  + .|
|         o.. + . |
|        S . + o  |
|         . o o . |
|          . o +  |
|         . . o   |
|        E        |
+-----------------+
```

Once generated, we make a cat of the public key, copy it and put it in GitLab. To perform this last operation, once we are inside GitLab, in the upper right corner, an icon similar to a person appears, where we can edit our profile. If we click, a new menu appears on the right side, where the SSH Keys are. By clicking on this option, we can add our new SSH key.

To perform the cloning, we will open a terminal from our machine, and execute the following commands:

```
[rhforum@dev ~]$ git clone http://gitlab.rhforum.com/rhforum/RHForum2018.git
Cloning into 'RHForum2018'...
Username for 'http://gitlab.rhforum.com': rhforum
Password for 'http://rhforum@gitlab.rhforum.com': rhforum123.
```

```
remote: Counting objects: 580, done.
remote: Compressing objects: 100% (330/330), done.
remote: Total 580 (delta 353), reused 388 (delta 239)
Receiving objects: 100% (580/580), 119.02 KiB | 0 bytes/s, done.
Resolving deltas: 100% (353/353), done.
Checking connectivity... done.
[rhforum@dev ~]$ cd RHForum2018/
[rhforum@dev RHForum2018]$ git config user.name rhforum
[rhforum@dev RHForum2018]$ git config user.email rhforum@domain.com
[rhforum@dev RHForum2018]$ git remote set-url --add origin git@gitlab.rhforum.com:rhforum/RHForum2018.git
[rhforum@dev RHForum2018]$ git remote set-url --delete origin http://gitlab.rhforum.com/rhforum/RHForum2018.git
[rhforum@dev RHForum2018]$ cat .git/config
[core]
	repositoryformatversion = 0
	filemode = true
	bare = false
	logallrefupdates = true
[remote "origin"]
	fetch = +refs/heads/*:refs/remotes/origin/*
	url = git@gitlab.rhforum.com:rhforum/RHForum2018.git
[branch "master"]
	remote = origin
	merge = refs/heads/master
[user]
	name = rhforum
	email = rhforum@domain.com
```

It is true that the reader may consider performing these operations in another way, but these - a priori - are valid for our laboratory.

From this moment, we can already perform the inspection of the code that we are going to use in the laboratory.

LAB # 1. DEPLOYMENT *DAY-1* TASKS.

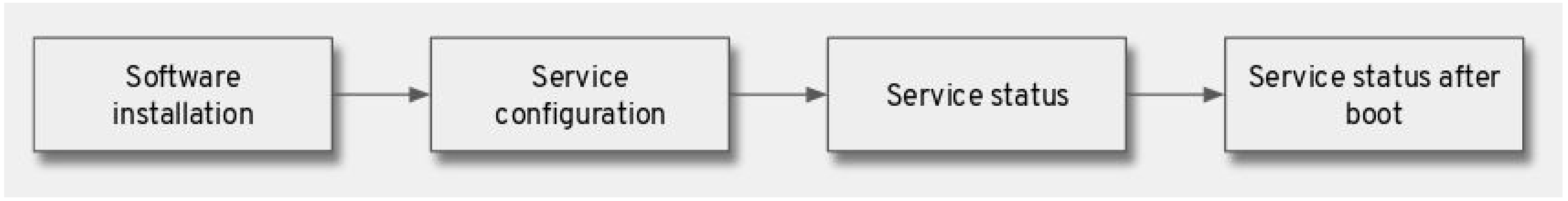
As we mentioned at the beginning of the lab, our methodological orientation will be IaC, which means, among many other things, that we must consider how we are going to parameterize / model our services and operations; thus the premises on which this laboratory has been built are:

- Any service must be able to be described by means of a configuration file.
- Decomposition of the service in reusable tasks.
- Creation of more sophisticated tasks based on atomic tasks.

A priori, the installation of a service - understood as a unitary service - we could decompose it in the following steps:

- Software installation
  - Configuration of the repositories to access the software.
  - Installation of the software itself.
- Service configuration
- Service status once the deployment has been made.
- Service status at the start of the machine.

Obviously, all these steps must be orchestrated to be certain that step n + 1 is done, if and only if step n has finished correctly; something similar to the following:



Well, once we have defined the steps we are going to perform, the first thing is to define the service in a configuration file; Below is the file of one of the services that we are going to deploy in the laboratory:

```
---
# Software repo config
service_software_repo_name: localrepo
service_software_repo_url: http://localrepo.rhforum.com/

# Service name string and service OS name
service_name: NGINX
service_os_name: rh-nginx18-nginx

# Software needed to install the service
service_pkgs: ['rh-nginx18']

# Config file vars. We need the template and the destination file
service_config_main_file: /etc/opt/rh/rh-nginx18/nginx/nginx.conf

service_config_files:
   - { template: ./templates/nginx_template.conf.j2, config: /etc/opt/rh/rh-nginx18/nginx/nginx.conf }

# Vars to configure the service
service_config_var_listen_port: 80
service_config_var_document_root: /opt/rh/rh-nginx18/root/usr/share/nginx/html

# Status on boot
service_onboot: on

# Service status after the deploy
service_status: start
...
```

Each of the sections / variables that appear have a meaning; and although a description appears in this regard, each of them is explained separately below:

- Configuration of repositories. It is necessary to be able to access the software that our service needs; Therefore, the first thing we define is the repository where the software is located.

  service_software_repo_name. Indicate the name that we are going to give to this repository.
  service_software_repo_url. Indicate the URL where we will be able to access the software.

- Names of services. There are times when the service at the level of the operating system is the same as we want to give it in our "login", but given that the other casuistry can be given, two variables have been included for it.

  service_name. Variable that is used to show the name of the service, has no greater implication of the messages per screen.
  service_os_name. Contains the name of the service at the operating system level. We use it to configure the start / stop of the same.

- Software required to install the service. Specifies the list of software required to install the service. We should try to make the packaging of this software as standard as possible: .rpm, .dev, .msi ... although obviously it could also be a .tar, .tgz, .zip package ... the lab is intended for .rpm, but I'm sure that after doing so, it occurs to you how to do the deployments of .tar, .zip ...

  service_pkgs. List with the necessary packages to install the service. You can specify all of them, or leave the software manager to solve the dependencies by itself.

- Files and configuration variables. In this section, we will define the variables and configuration files available to the service. Our approach is that of configuration files with *Jinja2* syntax.

  service_config_main_file. File that indicates the path of the main configuration file of the service.
  service_config_files. List, where we configure the template file, and the path of the configuration file in the destination machine (s).
  service_config_var_ *. The variables with this name are used to specify the fields to be substituted in our configuration files with *Jinja2* syntax. All the necessary ones can be used.

- Service status. We indicate the status of the service once the deployment has been made and at the start of the machine.

  service_status. You can take the values of start, stop, restart or reload. Perform the indicated operation once the service has been deployed.
  service_onboot. You can take the True or False values. According to the indicated operation, this leaves the service configured at the start of the machine.

Once the introduction and approach of the laboratory is done, the first thing we are going to do is the connection to Ansible Tower, since it is the laboratory's focus tool.

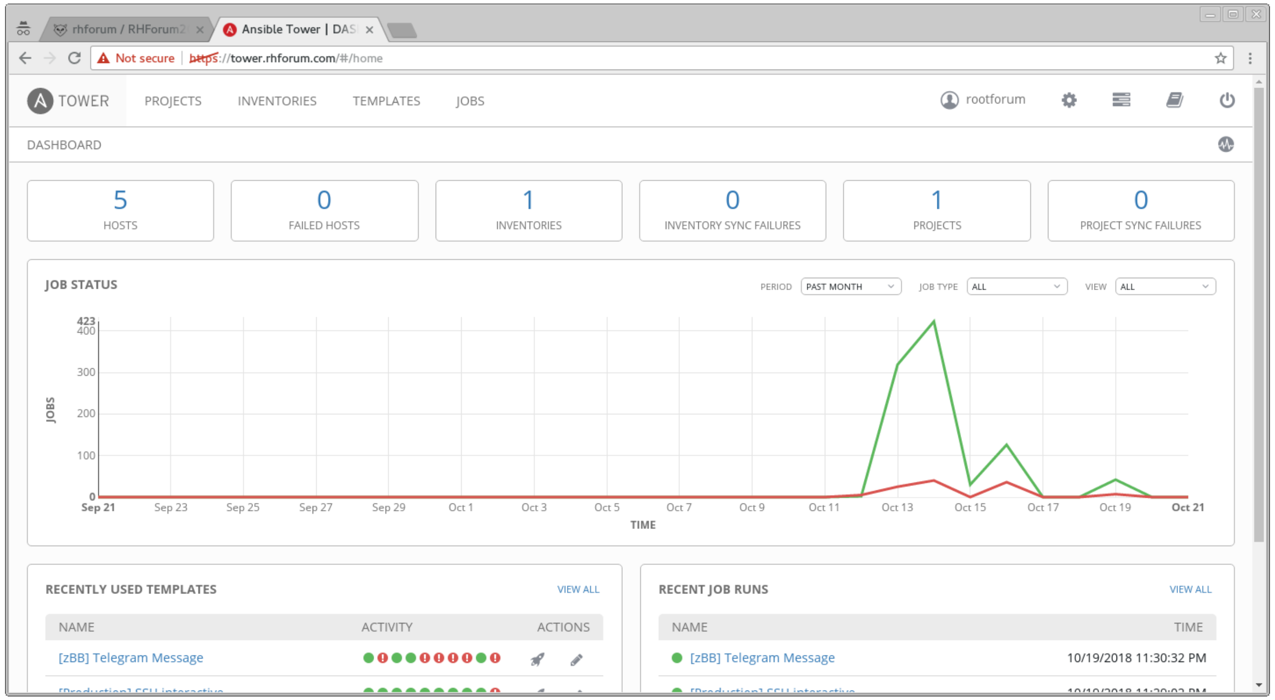
To do this, and in a browser, type the following url:

```
http://tower.rhforum.com
```

Now we will use the credentials that have been commented at the beginning of the document to be able to login with our laboratory user.

```
User: rootforum
Pass: rhforum123.
```

and we get a screen similar to the following:



5	0	1	0	1	0
HOSTS	FAILED HOSTS	INVENTORIES	INVENTORY SYNC FAILURES	PROJECTS	PROJECT SYNC FAILURES

JOB STATUS

PERIOD: PAST MONTH · JOB TYPE: ALL · VIEW: ALL

RECENTLY USED TEMPLATES — VIEW ALL

NAME	ACTIVITY	ACTIONS
[zBB] Telegram Message	●!●●!!!!●!	

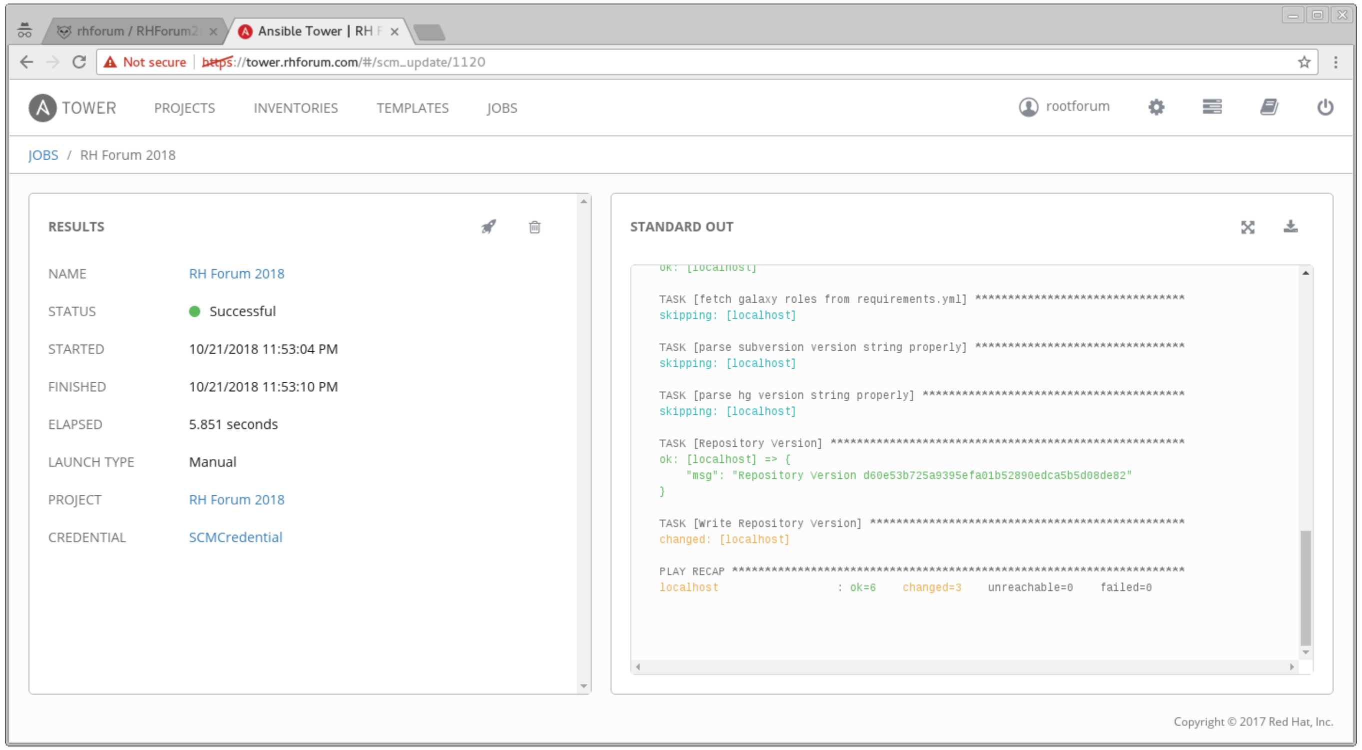
RECENT JOB RUNS — VIEW ALL

NAME	TIME
● [zBB] Telegram Message	10/19/2018 11:30:32 PM

We can take a quick look through the tool, verifying the inventory created, the credentials we are going to use, the definition of the project ... it is strongly recommended, take this tour :)

The first action we are going to execute is a synchronization of the project, in this way we verify the correct connection with our code repository, and we also make sure that the latest version of the code is in Ansible Tower.

To do this, we click on **PROJECTS**, and when we see our project: *RH Forum 2018*, at the height of the name, on the right, we click on the icon that represents a cloud. At that moment, next to the name of the project, a green ball will begin to flash, indicating that the synchronization operation is being carried out. Once finished, the ball will stop flashing, and if its color is green, it will indicate that the synchronization has been done correctly. Whether it has finished well or not, if we click on it, we can see the operations that involve this synchronization.



RESULTS

NAME	RH Forum 2018
STATUS	● Successful
STARTED	10/21/2018 11:53:04 PM
FINISHED	10/21/2018 11:53:10 PM
ELAPSED	5.851 seconds
LAUNCH TYPE	Manual
PROJECT	RH Forum 2018
CREDENTIAL	SCMCredential

STANDARD OUT

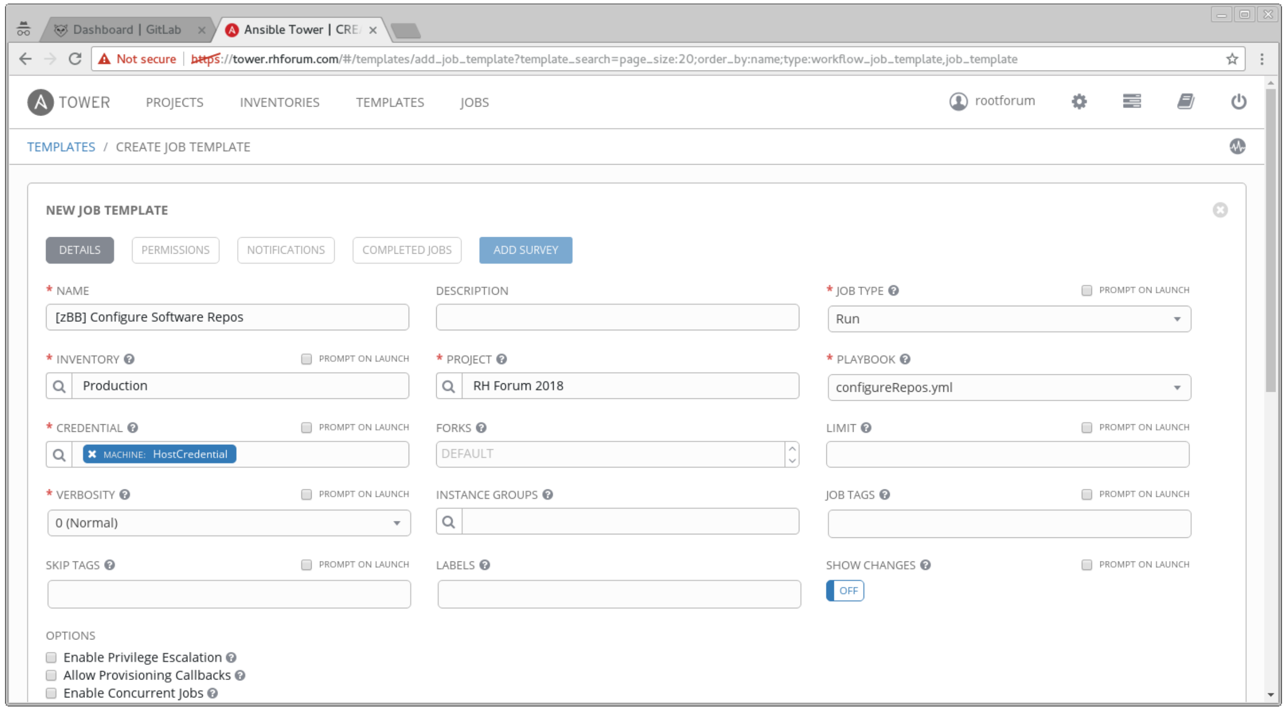
```
ok: [localhost]

TASK [fetch galaxy roles from requirements.yml] *******************************
skipping: [localhost]

TASK [parse subversion version string properly] *******************************
skipping: [localhost]

TASK [parse hg version string properly] ***************************************
skipping: [localhost]

TASK [Repository Version] *****************************************************
ok: [localhost] => {
    "msg": "Repository Version d60e53b725a9395efa01b52890edca5b5d08de82"
}

TASK [Write Repository Version] ***********************************************
changed: [localhost]

PLAY RECAP ********************************************************************
localhost                  : ok=6    changed=3    unreachable=0    failed=0
```

Copyright © 2017 Red Hat, Inc.

As we have commented, the great majority of the work of creation of the Jobs Templates is done, in order to be more agile in the laboratory. However, we are going to create a Job Template so that the reader can verify what is the way to do it. This Job Template will be used to make the configuration of the software repositories.

To do this, we will go to the **TEMPLATES** section, and there we click on the button on the right **+ADD**, and choose the option of **Job Template.** A screen appears, where we will use the following data to fill in the necessary and sufficient fields when creating a Job Template.

```
NAME: [zBB] Configure Software Repos
JOB TYPE: Run
INVENTORY: Production
PROJECT: RH Forum 2018
PLAYBOOK: configureRepos.yml
CREDENTIAL: HostCredential
```


Dashboard | GitLab
Ansible Tower | CRE
Not secure
~~https~~://tower.rhforum.com/#/templates/add_job_template?template_search=page_size:20;order_by:name;type:workflow_job_template,job_template
TOWER
PROJECTS
INVENTORIES
TEMPLATES
JOBS
rootforum
TEMPLATES / CREATE JOB TEMPLATE
NEW JOB TEMPLATE
DETAILS
PERMISSIONS
NOTIFICATIONS
COMPLETED JOBS
ADD SURVEY

* NAME: [zBB] Configure Software Repos	DESCRIPTION:	* JOB TYPE: Run (PROMPT ON LAUNCH)
* INVENTORY: Production (PROMPT ON LAUNCH)	* PROJECT: RH Forum 2018	* PLAYBOOK: configureRepos.yml
* CREDENTIAL: MACHINE: HostCredential (PROMPT ON LAUNCH)	FORKS: DEFAULT	LIMIT: (PROMPT ON LAUNCH)
* VERBOSITY: 0 (Normal) (PROMPT ON LAUNCH)	INSTANCE GROUPS:	JOB TAGS: (PROMPT ON LAUNCH)
SKIP TAGS: (PROMPT ON LAUNCH)	LABELS:	SHOW CHANGES: OFF (PROMPT ON LAUNCH)

OPTIONS
- Enable Privilege Escalation
- Allow Provisioning Callbacks
- Enable Concurrent Jobs

Finished putting the data, we click on the **Save** button, and we would have our first Job Templates done :)

But the idea is to perform an orchestrated deployment of the service, so we have to do a *Workflow*, where we will use different Jobs templates.

The following table shows the steps defined to perform the service deployment, and its mapping with the Jobs Templates that we have defined in Ansible Tower:

Software repositories configuration	[zBB] Configure Software Repos
Installing the service software	[zBB] Service Install
Service configuration	[zBB] Service Configure
Service status after deploy	[zBB] Service Status
Configuration of the service at the start of the machine	[zBB] Service Status On Boot

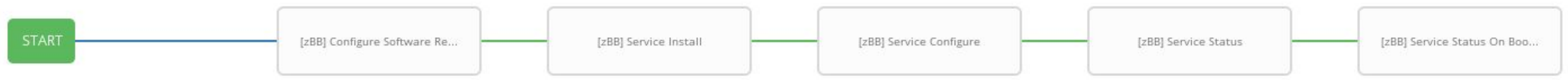
Thus, the next step is the creation of Workflow, so from the same screen where we created the previous jobs template, we click on the **+ADD** button, and now instead of Job Template, we choose *Workflow Template*. Once chosen, we use the following data to create it:

```
NAME: [Production] Service Deploy
ORGANIZATION: RHForum2018
```

Once the data is entered, click on the **Save** button, and we see how the **Survey** and **Workflow Editor** buttons are activated; we click on the latter, and the Workflows editor appears, where we will create our orchestration.

We will create each of the workflow steps - represented by a box - and assign the job templates specified in the previous table. In this case, we are only going to execute step *n + 1* when the *n* has been correct, as trick to create the workflow;)

The following figure shows the workflow created.



Note the blue line of the **START** to the first of the steps, this line means that what is behind it will **always** run, while the green lines mean that the next step will be executed **if the previous one has finished correctly**, and the red ones - although in this workflow they do not appear - they mean that the next step will be executed **if the previous one has terminated incorrectly**.

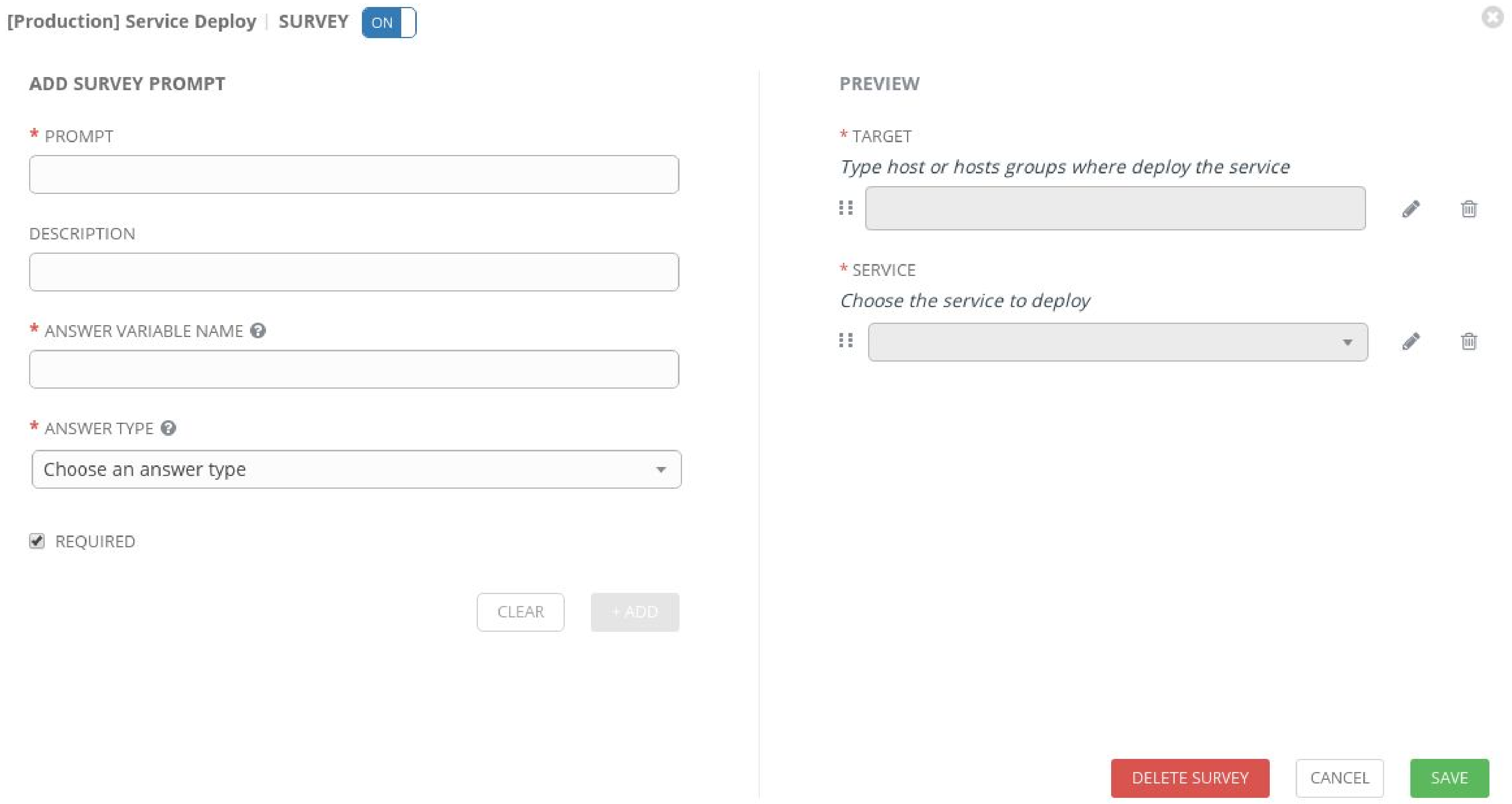
Finally, to finish our Workflow, we will provide it with a Survey. A Survey is a *data entry* that appears at the beginning of the execution of the Workflow, and that serves us so that the user interacts with the Workflow, choosing the different options that are requested / displayed on it.

To do this, we click on the **Add Survey** button, and we create our survey with the following data:

```
PROMPT: TARGET
DESCRIPTION: Type host or hosts groups where deploy the service
ANSWER VARIABLE NAME: target
ANSWER TYPE: Text
REQUIRED: Check
```

```
PROMPT: SERVICE
DESCRIPTION: Choose the service to deploy
ANSWER VARIABLE NAME: survey_service
ANSWER TYPE: Multiple Choice (single select)
MULTIPLE CHOICE OPTIONS:
                           apache
                           nginx
                           haproxy
REQUIRED: Check
```

Finished creating the Survey, we click on **Save**, and we would have our survey created. The following figure shows how it would look like.



[Production] Service Deploy | SURVEY ON

ADD SURVEY PROMPT

* PROMPT

DESCRIPTION

* ANSWER VARIABLE NAME

* ANSWER TYPE

Choose an answer type

☑ REQUIRED

PREVIEW

* TARGET

Type host or hosts groups where deploy the service

* SERVICE

Choose the service to deploy

Once we have saved the Survey, we proceed to save the Workflow, by clicking on the **Save** button.

At this point, we can proceed to deploy our services. To do this, we locate our Workflow, and we click on the rocket icon, to launch it.

We will launch the Workflow three times, each one with the following data:

```
# First execution.
TARGET: web1.rhforum.com
SERVICE: apache
```

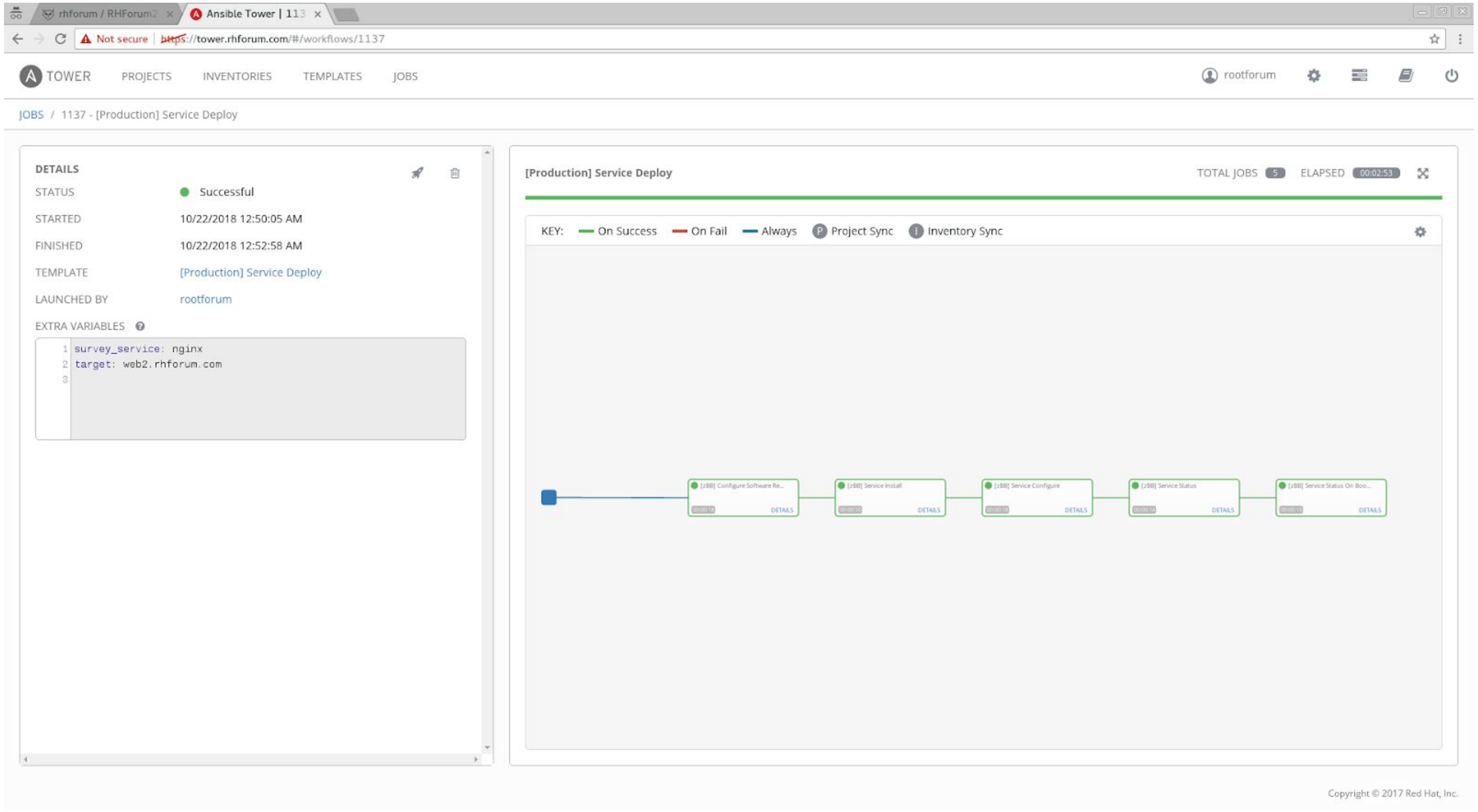
```
# Second execution.
TARGET: web2.rhforum.com
SERVICE: nginx
```

```
# Third execution.
TARGET: lb.rhforum.com
SERVICE: haproxy
```

We can verify which operations are running at a given moment within the Workflow, since a *small ball* appears flashing in the current step. If it ends, and its color is green, it is that that step has ended correctly; if it is red, it is that it has ended incorrectly.

During the execution, we can click on **DETAILS**, within each of the steps, to verify the specific tasks of that execution.

The figure on the next page shows the finished Workflow for the second execution, where on the web2.rhforum.com machine, we installed a nginx web server.



DETAILS

STATUS	● Successful
STARTED	10/22/2018 12:50:05 AM
FINISHED	10/22/2018 12:52:58 AM
TEMPLATE	[Production] Service Deploy
LAUNCHED BY	rootforum

EXTRA VARIABLES

```
survey_service: nginx
target: web2.rhforum.com
```

[Production] Service Deploy — TOTAL JOBS 5 — ELAPSED 00:02:53

KEY: 🟩 On Success 🟥 On Fail 🟦 Always Ⓟ Project Sync Ⓘ Inventory Sync

Once the deploys of the three services are finished, we must proceed to verify them; for this, from a browser, we open three tabs, and type the following urls, each one in one of the tabs:

```
http://web1.rhforum.com
```

The default page of Apache is returned.

```
http://web2.rhforum.com
```

The default page of Nginx is returned.

```
http://lb.rhforum.com
```

The page of Apache and Nginx is returned, depending on how the page is reloaded, since the HAproxy is configured with a Round Robin policy.

If the reader wishes to inspect the complete execution of the Workflows, we can go to the **JOBS** menu within Ansible Tower, and verify the execution of each of the Jobs Templates.

LAB # 2. MODIFICATION OF CONFIGURATION AND UPLOAD OF CONTENT. *DAY-2* TASKS.

In many cases, automation and orchestration tools are usually identified to perform deployment tasks, and nothing is further from reality, since many of the operations that are performed on the environments, are *day-2* tasks such as:

- Patched
- Configuration modification
- Verification of compliance...

These *day-2* tasks, contradict the evolution of the IaC, which is the Immutable Infrastructure, where its paradigm says that you can not perform any modification operation in the infrastructure in execution, and therefore, you have to perform a new deployment if is it necessary to make some modification ... does it sound like something to the reader? What is the basic technology that complies with this? :)

But while this evolution of our environments is being carried out, we can and **must** use tools such as Ansible and Ansible Tower to standardize any operation that is carried out on the platform.

Thus, as part of the laboratory, in the case of of *day-2* tasks, we have included two tasks:

- Modification of the balancer configuration. Addition or subtraction of servers to the balancing pool.
- Upload of content to web servers.

For the first task, a Workflow has been created for this task, where the main characteristic of it is that the configuration changes are not made directly on the balancer, but are performed on the service definition file of the haproxy, a *commit* is executed to the repository, and then the configuration is applied.

To do this, from the **TEMPLATES** section of Ansible Tower, locate the Workflow with name: **[Production] PROXY Config Modification**, and execute it as indicated above, by clicking on the rocket icon.

Just launch the Workflow, a Survey appears, where we will use the following information for its execution:

```
TARGET SERVICE: haproxy
TARGET HOST: web1.rhforum.com
HTTP SERVICE: apache
OPERATION: DELETE
```

In this case, to verify that the host web1.rhforum.com has been removed from the HAproxy configuration, we can do it in several ways:

- Reloading the connection from the browser to the url: **lb.rhforum.com**.
- Verifying the **config/haproxy_config.yml** file from a connection to GitLab.
- Verifying the contents of the HAproxy configuration file **/etc/haproxy/haproxy.conf**.

The reader is recommended to perform more than one execution of this Workflow, with the following data entries.

```
# Data Entry
TARGET SERVICE: haproxy
TARGET HOST: web1.rhforum.com
HTTP SERVICE: apache
OPERATION: ADD

# Expected result
The web1.rhforum.com server appears again in the HAproxy configuration.
```

```
# Data Entry
TARGET SERVICE: haproxy
TARGET HOST: web2.rhforum.com
HTTP SERVICE: apache
OPERATION: DELETE

# Expected result
The workflow ends up sending an error message saying that the web2.rhforum.com does not have an Apache server installed.
```

Surely the reader has noticed that this Workflow can be improved, taking the premise that, at least there should always be a host within the balancing pool, since if one of the hosts is deleted first, and then the other, in the second execution, Workflow fails ... the reader is invited to verify a possible solution in this regard.

The second *day-2* task that we are going to carry out, is the update of the static content of our web server.

A priori, this task is basically simple: copy the content that will be in a source directory, to the machine(s) that we specify. The issue here is that we must take into account that we have different types of web servers, and therefore, we can give the casuistry - among others - that the route where you should put the content, is not the same for each of them .

So much so, that if we inspect the configuration files of the apache and nginx services, we can verify:

```
[rhforum@dev RHForum2018]$ grep -i document_root config/apache_config.yml
service_config_var_document_root: /var/www/html
[rhforum@dev RHForum2018]$ grep -i document_root config/nginx_config.yml
service_config_var_document_root: /opt/rh/rh-nginx18/root/usr/share/nginx/html
```

Therefore two possibilities open to us:

- Ask the user to enter the type of service where we are going to update the content.
- That the automation itself is capable of discerning what service is acting, and act accordingly.

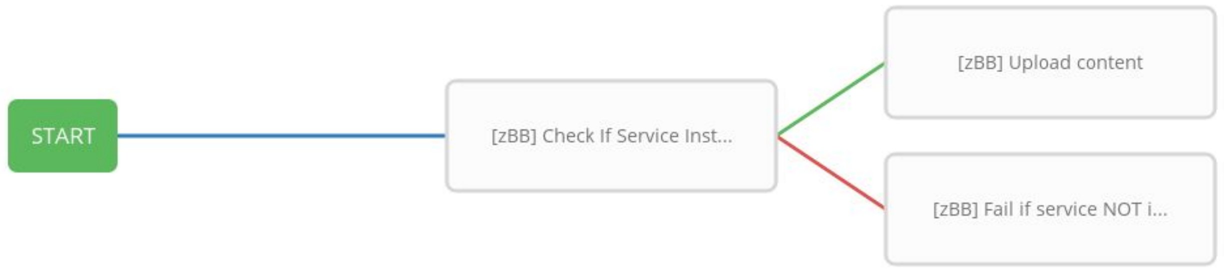
It is clear that the best option is the second one, but in the laboratory, the first option has been chosen, leaving the second as an exercise for the reader.

Looking at the first of them, we have to verify if the target machine has the service installed that they introduce us, and given that in the Workflow of modification of the proxy configuration we already perform this operation, we are going to reuse the Jobs Templates that we do, and we will create a Workflow to manage the static content.

Thus, from the **TEMPLATES** section of Ansible Tower, we will perform a Workflow with the following data:

```
NAME: [Production] Upload Web Content
ORGANIZATION: RHForum2018
```

Once finished creating the Workflow, it will be as follows:


START
[zBB] Check If Service Inst...
[zBB] Upload content
[zBB] Fail if service NOT i...

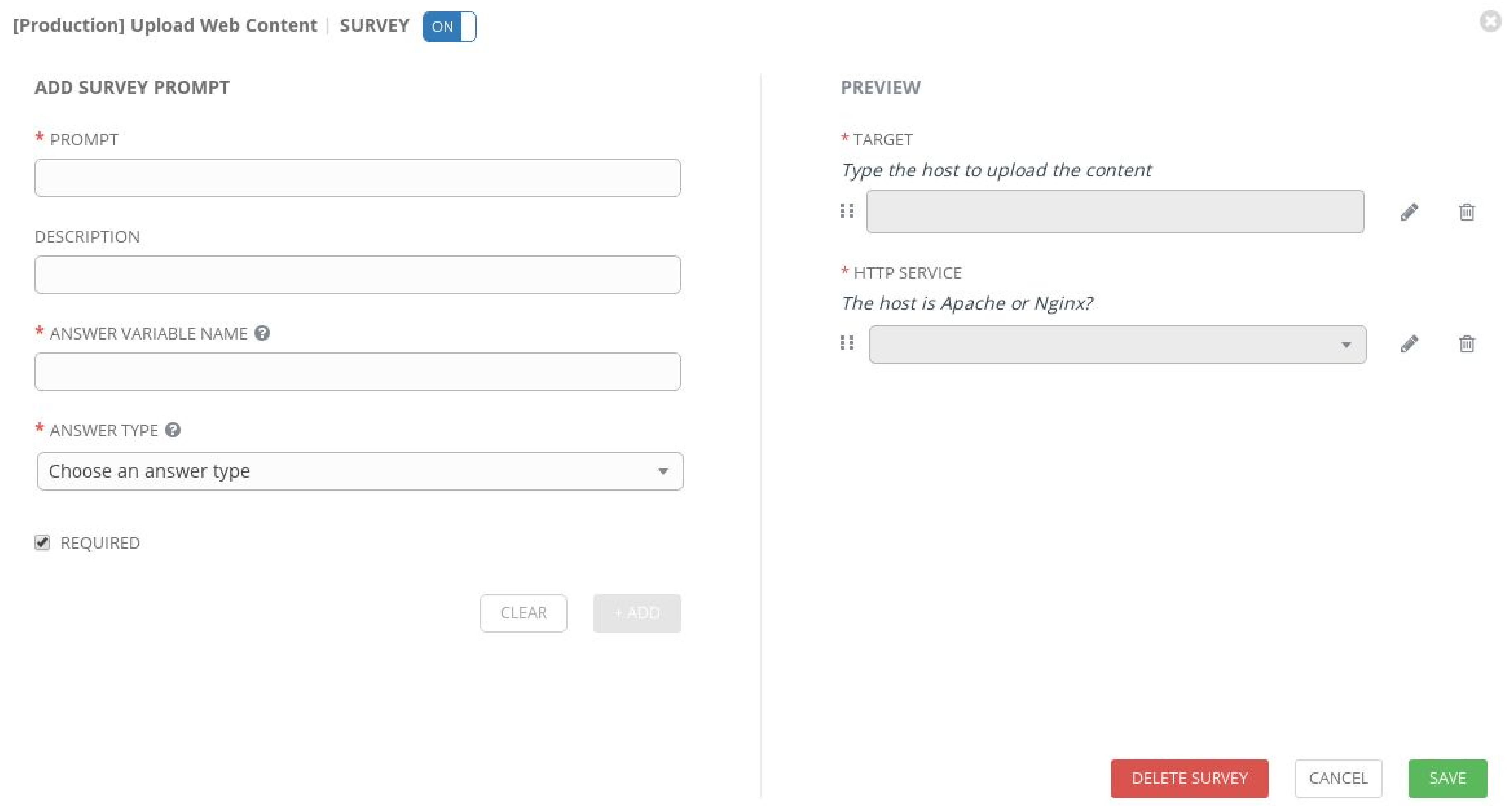
Finally, and given that we need to request data from the user, we will create a Survey with the following data:

```
PROMPT: TARGET
DESCRIPTION: Type the host to upload the content
ANSWER VARIABLE NAME: target_http
ANSWER TYPE: Text
REQUIRED: Check
```

```
PROMPT: HTTP SERVICE
DESCRIPTION: The host is Apache or Nginx?
ANSWER VARIABLE NAME: survey_service_http
ANSWER TYPE: Multiple Choice (single select)
MULTIPLE CHOICE OPTIONS:
                           apache
                           nginx
REQUIRED: Check
```

The Survey is shown on the next page. We save all our changes in the Workflow that we just created and execute it with the following data:

```
TARGET: web1.rhforum.com
HTTP SERVICE: apache
```


[Production] Upload Web Content | SURVEY ON

ADD SURVEY PROMPT

* PROMPT

DESCRIPTION

* ANSWER VARIABLE NAME

* ANSWER TYPE

Choose an answer type

☑ REQUIRED

CLEAR + ADD

PREVIEW

* TARGET

Type the host to upload the content

* HTTP SERVICE

The host is Apache or Nginx?

DELETE SURVEY CANCEL SAVE

To verify that the content has changed, and that Apache's default page is no longer displayed, the reader is left to try to connect to the particular web server or the balancer.

Finally, we can do another type of control - following the same example - that would be using code. To verify this, you would have to create a Job Template with the following data:

```
NAME: [Production] Upload Web Content - Playbook
JOB TYPE: Run
INVENTORY: Production
PROJECT: RH Forum 2018
PLAYBOOK: uploadWebContent_witchCheck.yml
CREDENTIAL: HostCredential
```

Once the Job Template has been created, we must add the same Survey that we have added in this section to the previous Workflow, to upload the content to the web servers.

LAB # 3. BONUS TRACK: SSH ACCESS MODIFICATION TO PRODUCTION MACHINES

Last but not least, this last lab has been prepared, where we will be able to activate or deactivate the SSH interactive access via console to the production machines.

Interactive access via SSH to production machines is disabled. Let's do it:

```
[rhforum@dev RHForum2018]$ ssh rhforum@web1.rhforum.com
rhforum@web1.rhforum.com's password:
PTY allocation request failed on channel 0
Connection to web1.rhforum.com closed.
[rhforum@dev RHForum2018]$ ssh rhforum@web2.rhforum.com
rhforum@web2.rhforum.com's password:
PTY allocation request failed on channel 0
Connection to web2.rhforum.com closed.
[rhforum@dev RHForum2018]$ ssh rhforum@lb.rhforum.com
rhforum@lb.rhforum.com's password:
PTY allocation request failed on channel 0
Connection to lb.rhforum.com closed.
```

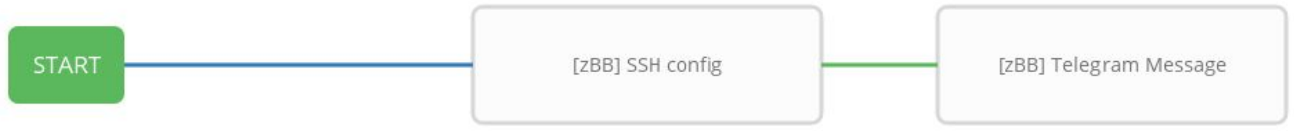
To activate or deactivate the interactive SSH, the WorkFlow has been prepared: **[Production] SSH interactive**.

Try to execute it, filling in the data requested by the Survey, and you will see that if you set it to ON, you can later access via SSH, that is, with the rhforum user, since root is deactivated :)

```
[rhforum@dev RHForum2018]$ ssh rhforum@web1.rhforum.com
rhforum@web1.rhforum.com's password:
Last login: Fri Oct 19 23:00:02 2018 from 192.168.110.162
[rhforum@web1 ~]$ hostname
web1.rhforum.com
[rhforum@web1 ~]$ logout
Connection to web1.rhforum.com closed.
[rhforum@dev RHForum2018]$ ssh root@web1.rhforum.com
root@web1.rhforum.com's password:
Permission denied, please try again.
root@web1.rhforum.com's password:
```

But since this operation is a critical operation, we can modify the Workflow, and add the sending a message to Telegram.

To do this, we will modify the Workflow indicated, and add one more step to the Workflow, remaining as follows:



It is not strictly necessary, but we must add to the Survey that has been created, one more field, with the following data:

```
PROMPT: MESSAGE
ANSWER VARIABLE NAME: survey_message
ANSWER TYPE: Text
DEFAULT ANSWER: RH Forum 2018
REQUIRED: Check
```

Once the Survey is saved, try again to execute it, and in the message, use the one that is by default, or put one to know that your message arrives correctly… although here we depend on the internet connection :)