Playlabs Documentation

Release 0.5

Yourlabs

Nov 29, 2018

Contents:

Playlabs provides a convenient wrapper for the ansible-playbook command and provides a set of ansible roles made to work together, and to orchestrate containers as much as possible and let only network level provisioning happen on hosts themselves, and combines straigthforward ansible patterns to install a docker orchestrated paas infra to prototype products for development to small-size production.

Pre-beta state: works for me, but parts are being rewriten independently until it's clean enough and declared stable, documentation is still in progress and so are tests.

CHAPTER 1

Playlabs: the obscene ansible distribution

Playlabs combines simple ansible patterns with packaged roles to create a docker orchestrated paas to prototype products for development to production.

Playlabs does not deal with HA, for HA you will need to do the ansible plugins yourself, or use kubernetes ... but Playlabs will do everything else, even configure your own sentry or kubernetes servers !

DISCLAMER: maybe it even works for you, but that's far from garanteed so far.

1.1 Install playlabs

Install with:

```
pip3 install --user -e git+https://yourlabs.io/oss/playlabs#egg=playlabs
```

Run the ansible-playbook wrapper command without argument to see the quick getting started commands:

```
~/.local/bin/playlabs
```

1.2 Quick start

You have a new host and you need your user to be installed with your public key, passwordless sudo, and secure SSH. The first command to run on a new host is `playlabs init`, ie.:

```
playlabs init root@1.2.3.4

# all options are ansible options are proxied
playlabs init @somehost --ask-become-pass

# example with a typical openstack vm
playlabs init ubuntu@somehost --ask-become-pass
```

Now your user can install roles:

```
playlabs install ssh,docker,firewall,nginx @somehost
```

And deploy a project, examples:

```
playlabs @somehost deploy image=betagouv/mrs:master
playlabs @somehost deploy
    image=betagouv/mrs:master
    plugins=postgres,django,uwsgi
    backup_password=foo
    prefix=ybs
    instance=hack
    env.SECRET_KEY=itsnotasecret
playlabs @somehost deploy
    prefix=testenv
    instance=$CI_BRANCH
    image=$CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

If you have that work, creating an inventory is the way to move on, wether you want to version configuration, add a deploy user for your CI, configure a secret backup password, add ssh-keys . . . :

```
playlabs scaffold ./your-inventory
```

Read on this README for gory details if you are already an Ansible user and only need to know about the patterns we're using playlabs for.

A more extensive and user-friendly documentation is in the docs sub-directory of playlabs and online @ https://playlabs.rtfd.io thanks to RTFD :)

1.3 playlabs init

Initializing means going from a naked system to a system with your own user, ssh key, dotfiles, sudo access, secure sshd, and all necessary dependencies to execute ansible, such as python3. It will also install your friend account if you have an ansible inventory repository where you store your friend list in yml.

You might need to pass extra options to ansible in some cases, for example if your install provides a passworded sudo, add `--ask-sudo-pass` or put the password in the CLI, since initializing will remove

```
playlabs init @somehost
playlabs init user:pass@somehost
playlabs init user@somehost --ask-sudo-pass
playlabs init root@somehost
```

1.4 playlabs install

If you want to deploy your project, then you need to install the paas which consists of three roles: docker, firewall, and nginx. The nginx role sets up two containers, nginx-proxy that watches the docker socket and introspects docker container environment variables, such as VIRTUAL_HOST, to reconfigure itself, it even supports uWSGI. The other container is nginx-letsencrypt, that shares a cert volume with the nginx-proxy container, and watches the docker socket for containers and introspect variables such as LETSENCRYPT_EMAIL, to configure the certificates.

Remember the architecture:

- nginx-proxy container recieves requests,
- nginx-letsencrypt container generates certificates,
- other docker containers have environment variables necessary for the above

The CLI itself is pretty straightforward:

```
playlabs install docker,firewall,nginx @somehost # the paas for the project role
playbabs install sendmail,netdata,mailcatcher,gitlab @staging
playbabs install sendmail,netdata,sentry user@production
```

The difference between traditionnal roles and playlabs roles, is that in playlabs they strive to have stuff running inside docker to leverage the architecture of the nginx proxy.

Playlabs can configure sendmail of course, but also has roles providing full-featured docker based mailservers or mailcatcher instances for your dev, training or staging environments for example.

This approach comes from migrating away from "building in production" to "building immutable tested chroots", away from "pet" to "cattle".

But if you're already an ansible hacker you're better off with ansible to do a **lot** more than than what docker-compose has to offer, such as managing users and roles, on your SDN as in your apps.

In fact, you will see role that consist of a single docker ansible module call, but the thing is that you can spawn it in one command and have it integrated with the rest of your server, and even rely on ansible to provision fine-grained RBAC in your own apps.

CHAPTER 2

User and groups management

The main feature of playlabs is your inventory, it's meant to make it easy for you to manage users and users to manage themselves on your infra & external services. For example, playlabs could provision ssh and ldap on an ldap server, but so far we haven't provisioned ldap servers with playlabs because we have playlabs ... wait wut ?

Anyway, when you're onboarding a hacker you can point them to your inventory repository url and also this documentation with the mission to add themselve.

2.1 Pre-requisite

Clone the inventory repository that you have been given if any. If it doesn't work, make sure that the git server knows your ssh public key if authenticating with SSH.

If you haven't been given an inventory repository to clone, create one with the scaffolt command (note that you can have as many inventories as you want):

```
playlabs scaffold your-inventory
```

2.2 Adding a new user

The users list and roles are defined in a YAML document that would be located in your repository at path `group_vars/all/users.yml`. Ansible offers a wide range of possibilities so it might also be elsewhere, but that's the convention used in the default playlabs inventory that you can generate with the `playlabs scaffold` command.

2.2.1 SSH Public key

Playlabs will use the SSH key it finds in the `keys/` inventory of the inventory repository. You can set it up as such:

```
# generate a key if you don't have any
ssh-keygen -t ed25519 -a 100

# create a branch for adding your user
git checkout -b $USER

# copy the public key to the keys subdirectory of the inventory repo
# if you have generated your key with the above it will be
cp ~/.ssh/id_ed25519.pub keys/$USER

# add to the inventory repository
git add keys/$USER
```

Then, read on the adding your user to the user list.

2.2.2 YAML user list

In the users.yml file, add a list item to the users variable. You should really use your local username if you want to have a nicer playlabs experience.

```
users:
  # ...
  - name: yourusername
    email: your@email.com
    roles:
      ssh: sudo
```

Add your modification with git and push it in a branch, then you can create a merge request on gitlab or whatever you use, ie:

```
git add -p group_vars/all/users.yml
git commit -m "Add $USER"
git push origin $USER
```

2.2.3 Kubernetes provisioning

Add `k8s: clusten-admin` or `cluster-admin: k8s` to the user `roles` ie.:

```
- name: jcarmack
  roles:
    ssh: sudo
    k8s: cluster-admin
```

Then, `playlabs install ssh,k8s @hostname` for example will add that user to ssh with sudo and make it a cluster-admin. It will create a signed certificate in the home directory of the user that they will be able to scp back and use to authenticate as cluster-admin with kubectl.

2.2.4 Password and secret variables

Secret content is handled with the ansible-vault command. You need to store your vault password in a file that will not be added to the inventory repository. The convention in playlabs is to name the file `.vault`. Then, ansible will recognize it with the `--vault-id .vault` command line argument.

Create a password for yourself:

```
ansible-vault create passwords/$USER
# or, automated:
echo -n your password | ansible-vault encrypt --vault-id .vault > passwords/$USER
```

SSH will not accept password authentication with playlabs by default, however your password will be useable with the rest of services installed with playlabs, even custom projects if their plugin support it, which is the case of the Django plugin, thanks to djcli.

2.3 Removing users

To remove a user, remove it from the `users` variables and then add its username to the `users_remove` list of `group_vars/all/users.yml` ie.:

```
users_remove:
- usernametodelete
```

2.4 Applying users

To apply users, you can run the `playlabs install ssh @host` command that will execute the SSH role, setting up the SSH users.

If you already have a host `inventory.yml` then you don't need to specify the hosts on the command line: all hosts that are in the ssh group will benefit from a `playlabs install ssh` call.

The convention accross playlabs is to have a tag named `users` so that we can also run roles partially in order to only update users with little efforts.

2.5 Reference

The users YAML document in the default repository serves as reference:

```
---
# This YAML document defines a list of users for playlabs ansible playbooks.
# You can have an automated job that will update users for example with
# `playlabs install ssh` and then users will get their credentials deployed on
# git push.
#
# You need the ansible vault password in cleartext a file that will not be
# tracked in git to edit secret variables such as passwords. You should create
# this file with the ``.vault`` name at the root of your inventory repository
# clone then you can use ansible-vault commands with the ``--vault-id .vault``
# argument ie.::
#
#     echo -n your password | ansible-vault encrypt --vault-id .vault > passwords/
↪hacker
#     ansible-vault view --vault-id .vault passwords/hacker
#     ansible-vault rekey --vault-id .vault passwords/hacker

users:
- name: hacker
```

(continues on next page)

(continued from previous page)

```
  email: hacker1337@example.pcom
  roles:
    netdata: [sysadmin, domainadmin, dba, webmaster, proxyadmin]
    ssh: [sudo]
    # superuser on all project instances
    project: [superuser]
    # setting role on group works both ways: don't have groups and roles with
    # the same name
    superuser:
    - ci
    - project-staging
    - sentry

# The example inventory provides one deploy user, than has no sudo
# access, except for the backup scripts that it cannot write.
# He has an ssh account because playlabs found a key in keys/deploy.pub.
# For deploy user, we have a key without password, that is supposed to be
# crypted with ansible-vault before commit:
#
#     $ ssh-keygen -t ed25519 -a 100 -f keys/deploy
#     $ echo -n your vault password > .vault
#     $ ansible-vault encrypt --vault-id .vault keys/deploy
- name: deploy
  sudo:
  - /home/*/backup.sh
  - /home/*/docker-run.sh

# The productowner user does not have ssh access because it does not have a
# public key in keys/productowner.pub.
#
# However, productowner have a password to pass through htaccess security, for
# roles and projects that have it enabled.
#
# To generate your own crypted password run this command:
#
#     $ echo -n your vault password > .vault
#     $ echo -n password | ansible-vault encrypt --vault-id .vault > passwords/
↪yourproductowner
- name: productowner
  superuser:
  - project-staging

# Playlabs will remove users in this list
users_remove:
- name: olduser

# Name of the user that will be able to write /home/service/docker-image
deploy_user: deploy
```

CHAPTER 3

Hosts inventory

While running playlabs with `@hostname` arguments is nice to experiment with, it won't scale with many machines nor will be convenient to automate playlabs calls. Most roles require an inventory to be really fun.

3.1 Pre-requisite

Clone the inventory repository that you have been given if any. If it doesn't work, make sure that the git server knows your ssh public key if authenticating with SSH.

If you haven't been given an inventory repository to clone, create one with the scaffolt command (note that you can have as many inventories as you want):

```
playlabs scaffold your-inventory
```

3.2 Adding a new host

Hosts are defined in the `inventory.yml` file of the inventory repository, use the `all` variable to add them in no specific group:

```
all:
  hosts:
    yourhost.com:  # adds a host with no extra option
    otherhost:
      fqdn: yourdomain.tld
      ansible_port: 22
      ansible_host: 123.12.12.23
```

3.3 Setting host groups

You can link hosts to groups in the `children` variable of the `inventory.yml` YAML document. For example, if you want `playlabs install ssh,netdata` without argument (for CI likely) to apply on `otherhost`, then this will work:

```
children:
  netdata-ssh:
    hosts:
      otherhost
```

CHAPTER 4

Managing infra variables in the inventory

4.1 Global variables

Variables that are used by convention accross roles:

```
letsencrypt_uri=https...
letsencrypt_email=your@...
```

4.2 Role variables

Base variable are defined in *playlabs/roles/rolename/vars/main.yml* and start with the *rolename_*, they can be overridden in your inventory's *group_vars/all/rolename.yml*.

The base variable will default to the same variable without the *rolename_* prefix:

```
# Set project_image project role variable from the command line
image=your/image:tag
```

4.3 Role structure

Default roles live in playlabs/roles and share the standard directory structure with ansible roles, that you can scaffold with the ansible-galaxy tool.

Playlabs use roles as alternatives as docker-compose when possible, rather than polluting the host with many services.

4.4 Project variables

The project role base variables calculate to be overridable by prefix/instance:

```
# project_{image,*} base value references project_staging_{image,*} from inventory
instance=staging

# project_{image,*} base value references mrs_production_{image,*} from inventory
instance=production prefix=mrs
```

4.5 Project plugins variable

The project role has a special plugins variable that can be overridden in the usual way, but it will also try to find it by introspecting the docker image for the *PLAYLABS_PLUGINS* env var ie:

```
ENV PLAYLABS_PLUGINS postgres,django,uwsgi,sentry
```

4.6 Plugin variables

Plugin variables are loaded by the project role for each plugin that it loads if any.

Base plugin variables start with *project_pluginname_* and the special *project_pluginname_env* variable should be a dict, they will be all merged to add environment variables to the project container, project_env will be a merge of all them plugin envs.

Plugin env vars should preferably use overridable variables.

CHAPTER 5

Projects deployments and lifecycles

WIP doc

5.1 Pre-requisite

You need a sudo access on the remote machine, which can typically be obtained with the `playlabs init` command ie.:

```
playlabs init root@1.2.3.4

# all options are ansible options are proxied, so this also works
playlabs init @somehost --ask-become-pass
```

The ssh, docker and firewall playlabs roles must be installed on the server:

```
playlabs install ssh,docker,firewall,nginx @somehost
```

5.2 Deploying a docker image

Examples:

```
playlabs @somehost deploy image=betagouv/mrs:master
playlabs @somehost deploy
    image=betagouv/mrs:master
    plugins=postgres,django,uwsgi
    backup_password=foo
    prefix=ybs
    instance=hack
    env.SECRET_KEY=itsnotasecret
playlabs @somehost deploy
```

(continues on next page)

(continued from previous page)

```
    prefix=testenv
    instance=$CI_BRANCH
    image=$CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

5.3 Deployments

The *project* role is made to be generic and cover infrastructure needs to develop a project, from development to production. Spawn an environment, here with an example image this repo is tested against:

```
playlabs @yourhost deploy betagouv/mrs:master '{"env":{"SECRET_KEY" :"itsnotasecret"}}
↪'
```

It will use the IP address by default if ansible finds it, set the dns with the dns option `dns=yourdns.com`, or set it in `project_staging_dns` yaml variable of *your-inventory/group_vars/all/project.yml*.

This is because the default prefix is `project` and the default instance is `staging`. Let's learn a new way of specifiying variables, add to your variables:

```
yourproject_production_image: yourimage:production
yourproject_production_env:
  SECRET_KEY: itsnotsecret
  # the above value could be encrypted with ansible-vault s_encrypt
```

Then you can deploy as such:

```
playlabs @yourhost deploy prefix=yourproject instance=production
```

If you configure yourhost in your inventory, in group "yourproject-production", then you don't have to specify the host anymore:

```
playlabs @yourhost project prefix=$CI_PROJECT instance=$CI_BRANCH
```

5.4 Project plugins

PostgreSQL or Django or uWSGI support are provided through project plugins, which you may activate as such:

- specify `-p postgres,uwsgi,django`
- configure `yourprefix_yourinstance_plugins=[postgres, uwsgi, django]`
- add to Dockerfile `ENV PLAYLABS_PLUGINS postgres,uwsgi,django`

The order of plugins matters, having postgres first ensures postgres is started before the project image.

Plugins are directories located at the root of playlabs repo, but at some point we can imagine loading them from the image itself.

Plugins contain the following:

- vars.yml: variables that are auto-loaded
- deploy.pre.yml: tasks to execute before deploy of the project image
- deploy.post.yml: tasks to execute after deploy of the project image
- backup.pre.sh: included in backup.sh template before the backup

- backup.post.sh: included in backup.sh template before the backup
- restore.pre.sh: included in restore.sh template before the restore
- restore.post.sh: included in restore.sh template before the restore

Default plugins live in playlabs/plugins and have the following files:

- *backup.pre.sh* take files out of containers and add them to the $backup variable
- *backup.post.sh* clean up files you have taken out after the backup has been done
- *restore.pre.sh* clear the place where you want to extract data from the restic backup repository
- *restore.post.sh* load new data and clean after the project was restarted in the snapshot version,
- *deploy.pre.yml* ansible tasks to execute before project deployment, ie. spawn postgres
- *deploy.post.yml* ansible tasks to execute after project deployment, ie. create users from inventory
- *vars.yml* plugin variables declaration

5.5 Operations

By default, it happens in /home/yourprefix-yourinstance. Contents depend on the activated plugins.

In the /home/ directory of the role or project there are scripts:

- *docker-run.sh* standalone command to start the project container, feel free to have on that one
- *backup.sh* cause a secure backup, upload with lftp if inventory defines dsn
- *restore.sh* recovers the secure backup repository with lftp if inventory desfines dsn. Without argument‘ list snapshots. With a snapshot argument‘ proceed to a restore of that snapshot including project image version and plugin data
- *prune.sh* removes un-needed old backup snapshots
- *log* logs that playlabs rotates for you, just fill in log files, it will do a copy truncate though, but works until you need prometheus or something

For backups to enable, you need to set backup_password, either with -e, either through yourprefix_yourinstance_backup_password.

The restic repository is encrypted, if you set the lftp_dsn or yourprefix_yourinstance_lftp_dsn then it will use lftp to mirror them. If you trash the local restic repository, and run restore.sh, then it will fetch the repository with lftp.

CHAPTER 6

CLI Options

Some of the variables you can like

```
-e key=value                    # set variable "key" to "value"
-e '{"key":"value"}'            # same in json
-i path/to/inventory_script.ext # load any numbers of inventory variables
-i 1.2.4.4,                     # add a host by ip to this play
--limit 1.2.4.4,                # limit play execution to these hosts
--user your-other-user          # specify a particular username
--noroot                        # don't try becoming root automatically
```

Note : all ansible-playbook arguments should work.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search