Amadla

Release 0.1

May 28, 2023

CONTENTS

1 Contents **3**

1.1 How It Works 3

1.2 Setup 5

1.3 CI/CD 10

1.4 Application Compatibility 10

Amadla (/ah-mah-dlah/) is a set of IaC (Infrastructure as Code) tools to help deploy and manage various cloud environments, dedicated servers, and on-premise systems.

Amadla is a Zulu word meaning "power" or "strength".

Attention: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 How It Works

Amadla with the help of Vault to store secrets, Packer to create server images, Ansible for the provisioning and Terraform deployment of the infrastructure, makes it easier to deploy and manage: cloud environments, dedicated servers, on-premise, VMs, applications amd services.

Without Amadla you would have to manually create the server images, provision them and deploy the infrastructure. This is a time consuming process and prone to human error. Amadla automates this process and makes it easier to manage the infrastructure and the many other components.

It is important to note that Amadla is not a replacement for the tools mentioned above. It is a wrapper around them to make it easier to use them together.

In other words it is a abstraction layer on top of the tools mentioned above.

Note: Amadla cli is very dependant of git also. It uses git to pull the modules that are used for the other configuration files.

Amadla is written in Python and uses the following libraries:

- Click for the command line interface.
- PyYAML for parsing YAML files used for the Amadla configuration files.
- Jinja2 for templating.
- jsonschema for validating JSON files (for the validation of the Amadla configuration files that the user create and manage).
- requests for making HTTP requests.
- python-dotenv for loading environment variables from a *.env* file.

1.1.1 Overview

1. Amadla takes your Amadla configurations files and the needed secrets from Vault to then create Packer, Terraform and Ansible variable files.
2. Those variable files are then used by Packer, Ansible and Terraform to build the server images, provision them and deploy the infrastructure.

1.1.2 User Usage Flow

The simple basic steps that user needs to do to deploy the infrastructure are:

1. The user creates a Amadla configuration file (YAML file) that contains the information about the infrastructure they want to deploy.
2. Populates Vault with the secrets needed for the infrastructure.
3. The user runs the `amadla build` command.

1.1.3 Amadla CLI Flow

The simple steps that Amadla CLI does are:

1. The secrets (username/password, API Keys, etc) are pulled from Vault the user added before from Vault or from the environment variables (you can store secrets in the environment variables).
2. The Amadla configuration file is validated.
3. The Amadla configuration files are merged together to create the `amadla.lock` JSON merged configuration file.
4. With `amadla.lock` all the modules are `git` pulled.
5. The Packer, Ansible and Terraform variable files are created with `amadla.lock` and with the secrets that were pulled from Vault or the environment variables (they can be set in the same terminal session where Amadla CLI is executed or added to `.env` file).
6. Terraform variable file is validated.
7. Terraform using `terraform.tfstate` to store the status and state and uses it to determine what are the new elements that need to be modified, removed or added in the infrastructure.
8. Terraform uses Packer to create the server images.
9. Packer uses the variable files to create the server images.
10. Packer then uses Ansible to provision the server images.
11. Ansible uses the variable files to provision the server images (inside of Ansible is all the packages, applications, services and configurations for the server image).
12. Once the server images are created and provisioned, Terraform uses the variable files to deploy the infrastructure.

Note: You can just run a test instead of deploying the infrastructure. This is useful to test the Amadla configuration files and the secrets.

Note: The Amadla configuration files are validated using JSON Schema. The JSON Schema files are located in the `amadla/schemas` directory.

1.1.4 Amadla Configuration Files

The Amadla configuration files are YAML files that contain the information about the infrastructures, server images, the clouds, VMs, applications, etc you want to deploy.

The Amadla configuration files are located in the `amadla/configs` directory in the git project.

1.2 Setup

1.2.1 Instructions for local setup

Note: If you are using a container you can skip this section.

System Packages

- python (11 or higher)
- pip (you might need to upgrade it after it is installed with `pip install --upgrade pip` or `pip3 install --upgrade pip`)
- make - optional (needed for the Makefile for generating documentations that is in the `docs/` directory)
- git (is needed for the pulling of modules)
- Terraform - for the infrastructure and VMs
- Vault - for the secrets management
- Packer - for the server and VM images
- Ansible - for the OS configurations, application installations, packages, and services (provisioning)

Tip: There are multiple ways to install the needed applications. You can download them directly and install them or even compile them or use a package manager on your system.

Mac OS X

Homebrew

```
brew install python
brew install git
brew tap hashicorp/tap
brew install hashicorp/tap/terraform hashicorp/tap/packer hashicorp/tap/vault
brew install ansible
```

Install `pip`

If your Python default version is 3 or higher:

```
curl https://bootstrap.pypa.io/get-pip.py | python
```

Or if your Python default version is 2 you might need to use this command:

```
curl https://bootstrap.pypa.io/get-pip.py | python3
```

Windows

The only it seems that you can install what is needed on Windows is by doing it manually by following the links above and installing them.

Note: Chocolatey might have some of the packages but it is not guaranteed that they will work.

Linux

Debian/Ubuntu

```
sudo apt install -y zip make ca-certificates git
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/
↪keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.
↪releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/
↪hashicorp.list
sudo apt update && sudo apt install -y terraform vault packer
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt install -y ansible
```

CentOS/RHEL

```
sudo yum install -y yum-utils git ansible make zip
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
sudo yum -y install terraform vault packer
```

Fedora

```
sudo dnf install -y dnf-plugins-core git ansible make zip
sudo dnf config-manager --add-repo https://rpm.releases.hashicorp.com/fedora/hashicorp.
↪repo
sudo dnf -y install terraform vault packer
```

Python Packages

For installing the packages and Amadla CLI you will need:

- poetry (`pip install poetry` or `pip3 install poetry`)

For generating the documentation you will need:

- sphinx (`pip install sphinx` or `pip3 install sphinx`)
- sphinx-rtd-theme (`pip install sphinx-rtd-theme` or `pip3 install sphinx-rtd-theme`)

All together:

```
pip install poetry sphinx sphinx-rtd-theme
```

Install Everything

The last step:

```
poetry install
```

OR

```
python -m poetry install
```

Note: Depending on your Python setup you might need to use `python3` instead of `python`.

Tip: You might run into an issue with keyring whilst you are trying to install Amadla CLI with `poetry` (it doesn't really show any errors unless you use the flag `--verbise` or `-vv` it just stays there with pending messages). To get around this issue you can try to use this environment variable: `export PYTHON_KEYRING_BACKEND=keyring.backends.null.Keyring`. Another solution is to upgrade `pip` with `pip install --upgrade pip` or `pip3 install --upgrade pip`.

1.2.2 Instructions for container setup

The container comes with the same packages as the local system setup except Vault you will need to use the Vault container or point via the configuration to wherever your Vault setup is.

Note: If you already have a container engine then you skip the System Packages setup.

System Packages

You can choose one of the following container engines:

- Podman (`sudo dnf install podman` or `sudo apt install podman`)
- Docker (`sudo dnf install docker-ce` or `sudo apt install docker-ce`)
- Kubernetes (`sudo dnf install kubectl` or `sudo apt install kubectl`)

Configurations

For Amadla you can use environment variables or a `.env` file. The `.env` file is loaded automatically if it exists in the root of the project.

```
touch .env
```

1.2.3 With Container

1. You need to have a Vault instance running.
2. Add your secrets to your Vault instance (or configure your Vault to load the secrets from a storage backend).
    - Adding secrets to Vault

1.2.4 Add Secrets

To add, remove, and list secrets you can follow the Secrets documentation.

If you are using Vault for your secrets the next step:

Depending on how you installed Vault you can start it with:

- Vault - `vault server -dev` if it is installed locally.
- Podman - `podman run --rm -d --name vault -p 8200:8200 vault:latest`.
- Docker - `docker run --rm -d --name vault -p 8200:8200 vault:latest`.
- Kubernetes - `kubectl run vault --image vault --port 8200`.
- podman-compose - `podman-compose up -d` (or `docker-compose` if you are using Docker).

Vault can store can secrets in many ways.

If you are using a file then you will need to add the parameter `-v ./vault/file:/vault/data` to the command.

If you are using some other storage backend then you will need to configure it.

Make sure you have a Vault server running and that you have the following environment variables set:

- `VAULT_ADDR` - The address of the Vault server
- `VAULT_TOKEN` - The token to authenticate with Vault
- `VAULT_LOG_LEVEL` - (optional) The log level for Vault (if not set it will use the log level of Amadla CLI)

Important: If you are using containers don't forget to set the environment variables for the container.

For a full list of environment variables see the Vault documentation.

Tip: To help you troubleshoot the Linux environment variables you can use these commands (should be the same on Mac OS X and other UNIX based systems):

- `export | grep VAULT` - To see all the environment variables that are set
- `echo $VAULT_ADDR` - To see the value of a specific environment variable
- `unset VAULT_ADDR` - To unset a specific environment variable
- `unset $(compgen -v | grep VAULT)` - To unset all the environment variables that start with `VAULT`
- `printenv` - To see all the environment variables that are set (you can always pipe this to `grep` to filter the output)

For the Windows environment using PowerShell you can use these commands:

- `Get-ChildItem Env:VAULT*` - To see all the environment variables that are set
- `$Env:VAULT_ADDR` - To see the value of a specific environment variable
- `Remove-Item Env:VAULT_ADDR` - To unset a specific environment variable
- `Get-ChildItem Env:VAULT* | Remove-Item` - To unset all the environment variables that start with `VAULT`
- `Get-ChildItem Env:` - To see all the environment variables that are set (you can always pipe this to `Select-String` to filter the output)

Using Windows CMD:

- `set | findstr VAULT` - To see all the environment variables that are set
- `echo %VAULT_ADDR%` - To see the value of a specific environment variable
- `set VAULT_ADDR=` - To unset a specific environment variable
- To unset all the environment variables that start with "VAULT" is a bit more complex in CMD and may require a script, it's not a single command.
- `set` - To see all the environment variables that are set

Note: If you use Vault you need to make sure that the Vault server is reachable from the Amadla CLI container. If you use Docker Compose you can use the `network_mode: host` option to make sure that the container can reach the Vault server.

1.2.5 Install Amadla CLI Dependencies

```
poetry install
```

Or you can run it with `python<python version> -m`:

```
python3 -m poetry install
```

Note: For the none development environment you can run `poetry install --no-dev` or `python3 -m poetry install --no-dev`. This will not install the development dependencies that are listed in `[tool.poetry.dev-dependencies]` in `pyproject.toml` file.

1.2.6 Start Amadla

You can try to run the CLI with:

```
poetry run amadla
```

Or you can run it with `python<python version> -m`:

```
python3 -m poetry run amadlacli
```

The other commands: - `clean` - Remove `.terraform` folder, state and plan files - `down` - Plan and apply a tf destroy

1.3 CI/CD

CI/CD tools that Amadla supports: - GitHub Workflow - Jenkins

You can use any other CI/CD tools that you whish but for the moments those are the only ones that Amadla will maintain.

CI/CD helps to quickly put everything together and deploy the things that you want automatically.

With GitHub Workflow you will be able to just fork this repo and then whenever you make a change to the code, and you commit and push the GitHub Workflow will execute the IaC code and deploy everything without the need of any other actions on your part.

More documentation to come.

1.4 Application Compatibility

1.4.1 Making Applications Compatible

Container

Full Server

X Application