DTaaS

Digital Twin as a Service (DTaaS)

DTaaS Development Team

Table of contents

1. What is DTaaS?

The Digital Twin as a Service (DTaaS) software platform is useful to **Build, Use and Share** digital twins (DTs).

🦾 **Build**: The DTs are built on the software platform using the reusable DT components available on the platform.

👩‍🔬👮‍♂️ **Use**: Use the DTs on the software platform.

🤝 **Share**: Share ready to use DTs with other users. It is also possible to share the services offered by one DT with other users.

There is an overview of the software available in the form of slides, video, and feature walkthrough.

1.1 License

This software is owned by The INTO-CPS Association and is available under the INTO-CPS License.

The DTaaS software platform uses third-party open-source software. These software components have their own licenses.

2. Admin

2.1 Overview

2.1.1 Goal

The goal is to set up the DTaaS infrastructure in order to enable your users to use the DTaaS. As an admin you will administrate the users and the servers of the system.

2.1.2 Requirements

OAuth Provider (Mandatory)

You need to have an OAuth Provider running, which the DTaaS can use for authorization. This is described further in the authorization section.

Domain name (Optional)

The DTaaS software is a web application and is preferably hosted on a server with a domain name like *foo.com*. However, it is possible to install the software on your computer and use access it at *localhost*.

Reverse Proxy (Optional)

The installation setup assumes that the foo.com server is behind a reverse proxy / load balancer that provides https termination. You can still use the DTaaS software even if you do not have this reverse proxy. If you do not have a reverse proxy, please replace https://foo.com with http://foo.com in client env.js file and in OAuth registration. Other installation configuration remains the same.

2.1.3 Install

The DTaaS can be installed in different ways. Each version serves a different purpose. Follow the installation that fits your usecase.

Installation Setup	Purpose
Trial installation on localhost	Install DTaaS on your computer for a single user; does not need a web server. *This setup also does not require reverse proxy and domain name.*
Trial installation on single host	Install DTaaS on server for a single user.
Production installation on single host	Install DTaaS on server for multiple users.
One vagrant machine	Install DTaaS on a virtual machine; can be used for single or multiple users.
Two vagrant machines	Install DTaaS on two virtual machines; can be used for single or multiple users.
	The core DTaaS application is installed on the first virtual machine and all the services (RabbitMQ, MQTT, InfluxDB, Grafana and MongoDB) are installed on second virtual machine.
Seperater Packages: client website and lib microservice	Can be used independently; do not need full installation of DTaaS.

2.2 Authorization

2.2.1 OAuth for React Client

To enable user authorization on DTaaS React client website, you will use the OAuth authorization protocol, specifically the PKCE authorization flow. Here are the steps to get started:

1. Choose Your GitLab Server:

- You need to set up OAuth authorization on a GitLab server. The commercial gitlab.com is not suitable for multi-user authorization (DTaaS requires this), so you'll need an on-premise GitLab instance.
- You can use GitLab Omnibus Docker for this purpose.
- Configure the OAuth application as an instance-wide authorization type.

2. Determine Your Website's Hostname:

- Before setting up OAuth on GitLab, decide on the hostname for your website. It's recommended to use a self-hosted GitLab instance, which you will use in other parts of the DTaaS application.

3. Define Callback and Logout URLs:

- For the PKCE authorization flow to function correctly, you need two URLs: a callback URL and a logout URL.
- The callback URL informs the OAuth provider of the page where signed-in users should be redirected. It's different from the landing homepage of the DTaaS application.
- The logout URL is where users will be directed after logging out.

4. OAuth Application Creation:

- During the creation of the OAuth application on GitLab, you need to specify the scope. Choose openid, profile, read_user, read_repository, and api scopes.

5. Application ID:

- After successfully creating the OAuth application, GitLab generates an application ID. This is a long string of HEX values that you will need for your configuration files.

6. Required Information from OAuth Application:

- You will need the following information from the OAuth application registered on GitLab:

GitLab Variable Name	Variable Name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.foo.com/
Application ID	REACT_APP_CLIENT_ID	
Callback URL	REACT_APP_REDIRECT_URI	https://foo.com/Library
Scopes	REACT_APP_GITLAB_SCOPES	openid, profile, read_user, read_repository, api

7. Create User Accounts:

Create user accounts in gitlab for all the usernames chosen during installation. The *trial* installation script comes with two default usernames - *user1* and *user2*. For all other installation scenarios, accounts with specific usernames need to be created on gitlab.

Development Environment

There needs to be a valid callback and logout URLs for development and testing purposes. You can use the same oauth application id for both development, testing and deployment scenarios. Only the callback and logout URLs change. It is possible to register multiple callback URLs in one oauth application. In order to use oauth for development and testing on developer computer (localhost), you need to add the following to oauth callback URL.

```
DTaaS application URL: http://localhost:4000
Callback URL: http://localhost:4000/Library
Logout URL: http://localhost:4000
```

The port 4000 is the default port for running the client website.

Multiple DTaaS applications

The DTaaS is a regular web application. It is possible to host multiple DTaaS applications on the same server. The only requirement is to have a distinct URLs. You can have three DTaaS applications running at the following URLs.

```
https://foo.com/au
https://foo.com/acme
https://foo.com/bar
```

All of these instances can use the same gitlab instance for authorization.

DTaaS application URL	Gitlab Instance URL	Callback URL	Logout URL	Application ID
https://foo.com/au	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab
https://foo.com/acme	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab
https://foo.com/bar	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab

If you are hosting multiple DTaaS instances on the same server, do not install DTaaS with a null basename on the same server. Even though it works well, the setup is confusing to setup and may lead to maintenance issues.

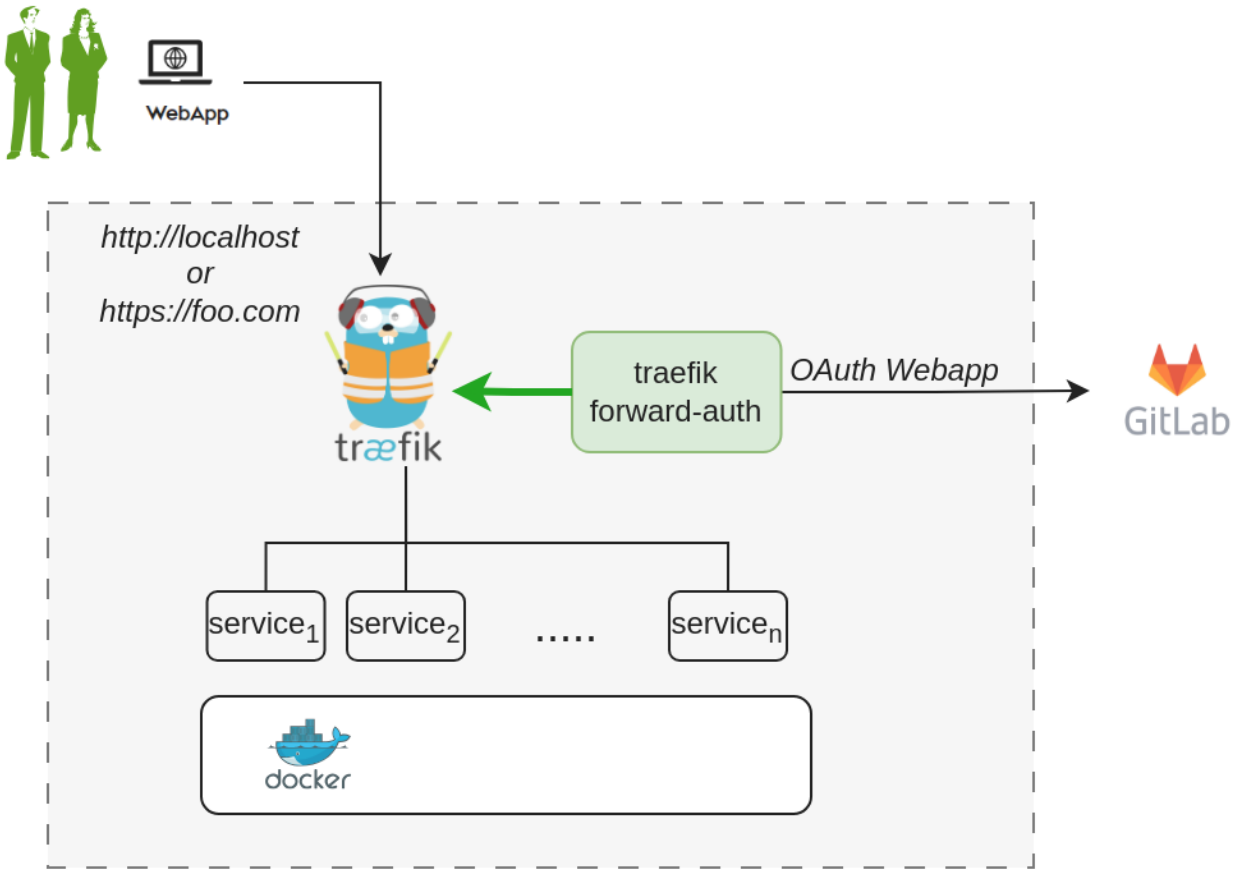
If you choose to host your DTaaS application with a basename (say bar), then the URLs in `env.js` change to:

```
DTaaS application URL: https://foo.com/bar
Gitlab instance URL: https://foo.gitlab.com
Callback URL: https://foo.com/bar/Library
Logout URL: https://foo.com/bar
```

2.2.2 OAuth for Traefik Gateway

The traefik gateway is used to serve the DTaaS. All the services provided as part of the application are secured at the traefik gateway. The security is based on Traefik forward-auth.

An illustration of the docker containers used and the authorization setup is shown here.



The **traefik forward-auth** can use any OAuth2 provider, but within the DTaaS gitlab is used as authorization provider. You will use the OAuth the web / server application authorization flow.

Here are the steps to get started:

1. Choose Your GitLab Server:

- You need to set up OAuth authorization on a GitLab server. The commercial gitlab.com is not suitable for multi-user authorization (DTaaS requires this), so you'll need an on-premise GitLab instance.
- You can use GitLab Omnibus Docker for this purpose.
- Configure the OAuth application as an instance-wide authorization type. Select option to generate client secret and also selection option for trusted application.

2. Determine Your Website's Hostname:

- Before setting up OAuth on GitLab, decide on the hostname for your website. It's recommended to use a self-hosted GitLab instance, which you will use in other parts of the DTaaS application.

3. Define Callback and Logout URLs:

- For the web / server authorization flow to function correctly, you need two URLs: a callback URL and a logout URL.
- The callback URL informs the OAuth provider of the page where signed-in users should be redirected. It is the landing homepage of the DTaaS application. (either http://foo.com/_oauth/ or http://localhost/_oauth/)
- The logout URL is the URL for signout of gitlab and clear authorization within traefik-forward auth. (either http://foo.com/_oauth/logout or http://localhost/_oauth/logout). The logout URL is to help users logout of traefik forward-auth. The logout URL should not be entered into Gitlab OAuth application setup.

4. OAuth Application Creation:

- During the creation of the OAuth application on GitLab, you need to specify the scope. Choose *read_user* scope.

5. Application Credentials:

- After successfully creating the OAuth application, GitLab generates an *application ID* and *client secret*. Both these values are long string of HEX values that you will need for your configuration files.

6. Required Information from OAuth Application:

- You will need the following information from the OAuth application registered on GitLab:

GitLab Variable Name	Variable Name in .env of docker compose file	Default Value
OAuth Provider	OAUTH_URL	https://gitlab.foo.com/
Application ID	CLIENT_ID	*xx*
Application Secret	CLIENT_SECRET	*xx*
Callback URL	(to be directly entered in Gitlab OAuth registration)	
Forward-auth secret	OAUTH_SECRET	*random-secret-string* (password for forward-auth, can be changed to your preferred string)
Scopes	read_user	

Development Environment

The development environment does not required traefik forward-auth.

Configure Authorization Rules for Traefik Forward-Auth

The Traefik forward-auth microservices requires configuration rules to manage authorization for different URL paths. The *conf* file can be used to configure the specific rules. There are broadly three kinds of URLs:

PUBLIC PATH WITHOUT AUTHORIZATION

To setup a public page, an example is shown below.

```
rule.noauth.action=allow
rule.noauth.rule=Path(`/public`)
```

Here, 'noauth' is the rule name, and should be changed to suit rule use. Rule names should be unique for each rule. The 'action' property is set to "allow" to make the resource public. The 'rule' property defines the path/route to reach the resource.

COMMON TO ALL USERS

To setup a common page that requires Gitlab OAuth, but is available to all users of the Gitlab instance:

```
rule.all.action=auth
rule.all.rule=Path(`/common`)
```

The 'action' property is set to "auth", to enable Gitlab OAuth before the resource can be accessed.

SELECTIVE ACCESS

Selective Access refers to the scenario of allowing access to a URL path for a few users. To setup selective access to a page:

```
rule.onlyu1.action=auth
rule.onlyu1.rule=Path(`/user1`)
rule.onlyu1.whitelist = user1@localhost
```

The 'whitelist' property of a rule defines a comma separated list of email IDs that are allowed to access the resource. While signing in users can sign in with either their username or email ID as usual, but the email ID corresponding to the account should be included in the whitelist.

This restricts access of the resource, allowing only users mentioned in the whitelist.

Limitation

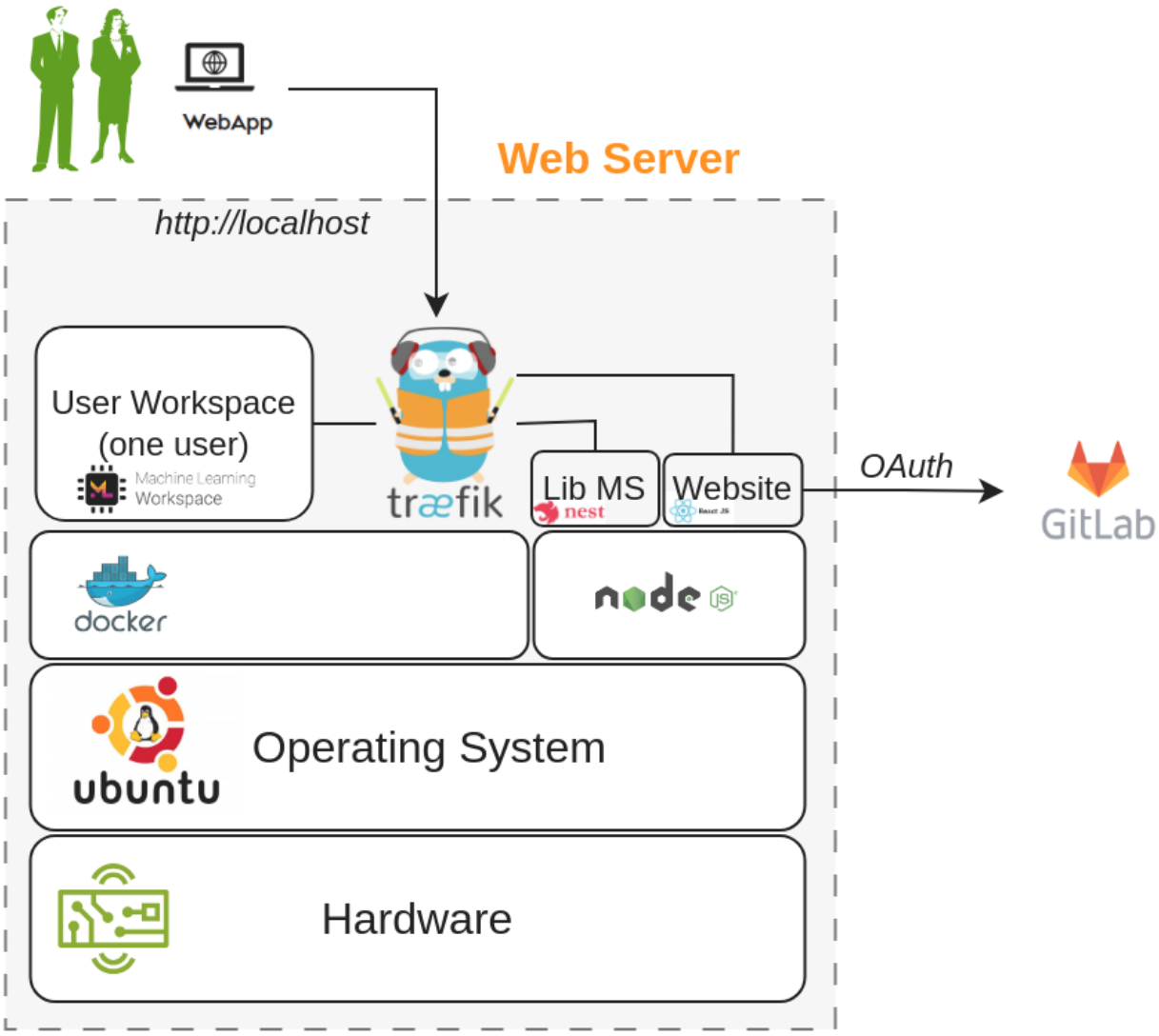
The rules in *conf* file are not dynamically loaded into the **traefik-forward-auth** microservice. Any change in the *conf* file requires retart of **traefik-forward-auth** for the changes to take effect. All the existing user sessions get invalidated when the **traefik-forward-auth** restarts.

Use a simple command on the terminal.

2.3 Host Install

2.3.1 Localhot Installation

To try out the software, you can install it on Ubuntu 22.04 Desktop Operating System. The setup requires a machine which can spare 4GB RAM, 2 vCPUs and 15GB Hard Disk space to a the DTaaS application. A successful installation will create a setup similar to the one shown in the figure.



A one-step installation script is provided on this page. This script sets up the DTaaS software for a single user. You can use it to check a test installation of the DTaaS software.

Pre-requisites

1. GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

> **Information**
>
> It is sufficient to have user-owned oauth application. You can create this application in your gitlab account.

You need the following information from the Gitlab OAuth application registered on Gitlab:

Gitlab Variable Name	Variable name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.com or https://gitlab.foo.com
Application ID	REACT_APP_CLIENT_ID	
Callback URL	REACT_APP_REDIRECT_URI	http://localhost/Library
Scopes	REACT_APP_GITLAB_SCOPES	openid, profile, read_user, read_repository, api

You can also see Gitlab help page for getting the Gitlab OAuth application details.

Install

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Run the following commands.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/feature/distributed-demo/deploy/single-script-install.sh
bash single-script-install.sh --env local --username <username>
```

The `--env local` argument is added to the script specifies `localhost` as the installation scenario. The `--username username` uses your Gitlab username to configure the DTaaS application.

Post install

After the single-install-script is successfully run. Please change Gitlab OAuth details in

```
~/DTaaS/client/build/env.js
```

Post-install Check

Now when you visit http://localhost, you should be able to login through Gitlab OAuth Provider and access the DTaas web UI.

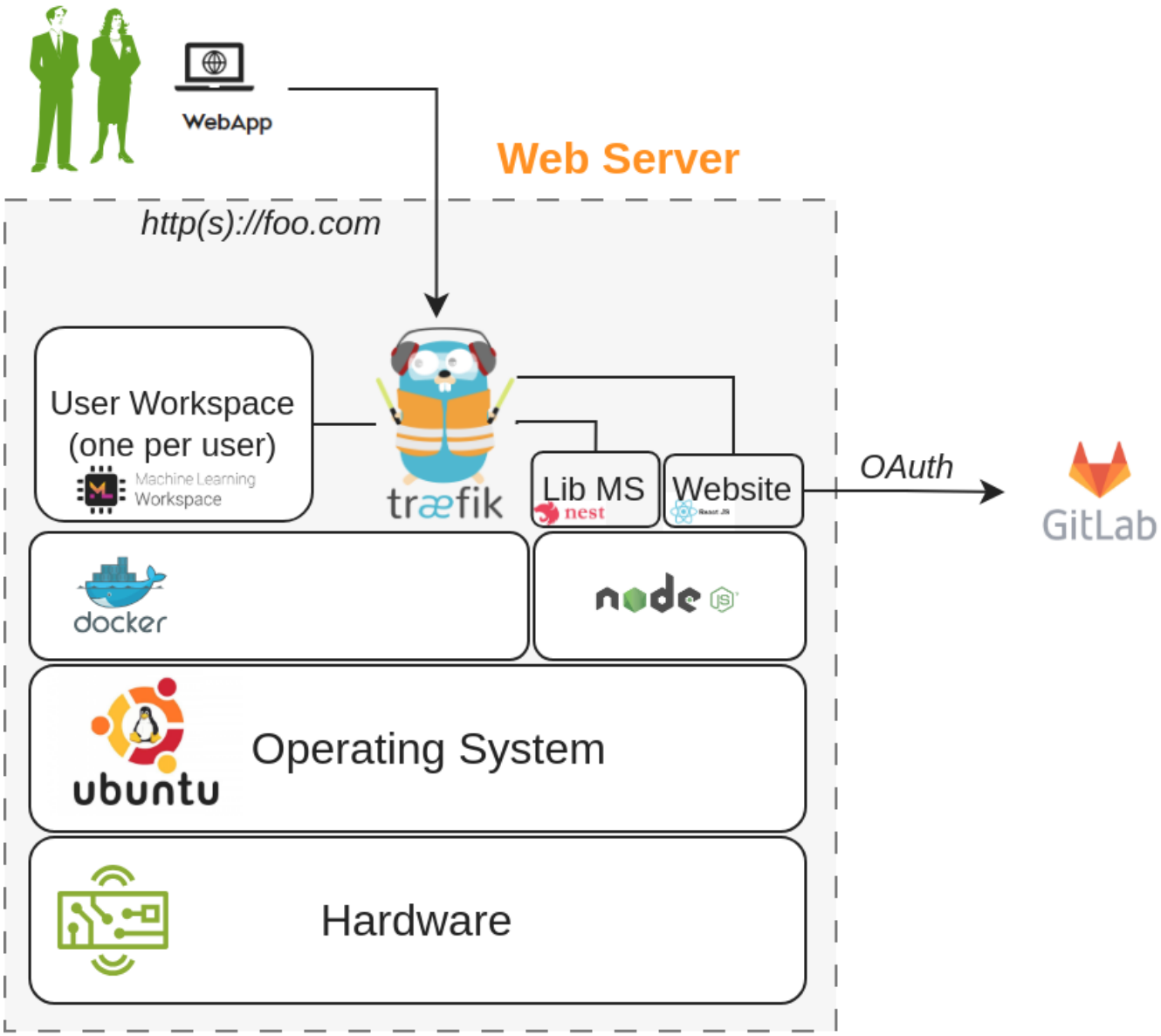
If you can following along to see all the screenshots from user website. Everything is correctly setup.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.3.2 Trial Installation

To try out the software, you can install it on Ubuntu Server 22.04 Operating System. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space to the DTaaS application (or the vagrant box hosting the DTaaS application). A successful installation will create a setup similar to the one shown in the figure.



A one-step installation script is provided on this page. This script sets up the DTaaS software with default gateway credentials for a single user. You can use it to check a test installation of DTaaS software.

Information

It is sufficient to have user-owned oauth application.

Pre-requisites

1. DOMAIN NAME

You need a domain name to run the application. The install script assumes **foo.com** to be your domain name. You will change this after running the script.

2. GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

You need the following information from the Gitlab OAuth application registered on Gitlab:

Gitlab Variable Name	Variable name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.com or https://gitlab.foo.com
Application ID	REACT_APP_CLIENT_ID	
Callback URL	REACT_APP_REDIRECT_URI	https://foo.com/Library
Scopes	REACT_APP_GITLAB_SCOPES	openid, profile, read_user, read_repository, api

You can also see Gitlab help page for getting the Gitlab OAuth application details.

Install

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Run the following commands.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/feature/distributed-demo/deploy/single-script-install.sh
bash single-script-install.sh --env trial --username <username>
```

The `--env trial` argument is added to the script specifies `trial` as the installation scenario. The `--username username` uses your Gitlab username to configure the DTaaS application.

> **Warning**
>
> This test installation has default gateway credentials and is thus highly insecure.

Post install

After the install-script. Please change **foo.com** and Gitlab OAuth details to your local settings in the following files.

```
~/DTaaS/client/build/env.js
~/DTaaS/servers/config/gateway/dynamic/fileConfig.yml
```

Post-install Check

Now when you visit https://foo.com, you should be able to login through Gitlab OAuth Provider and access the DTaas web UI.

If you can following along to see all the screenshots from user website. Everything is correctly setup.

References

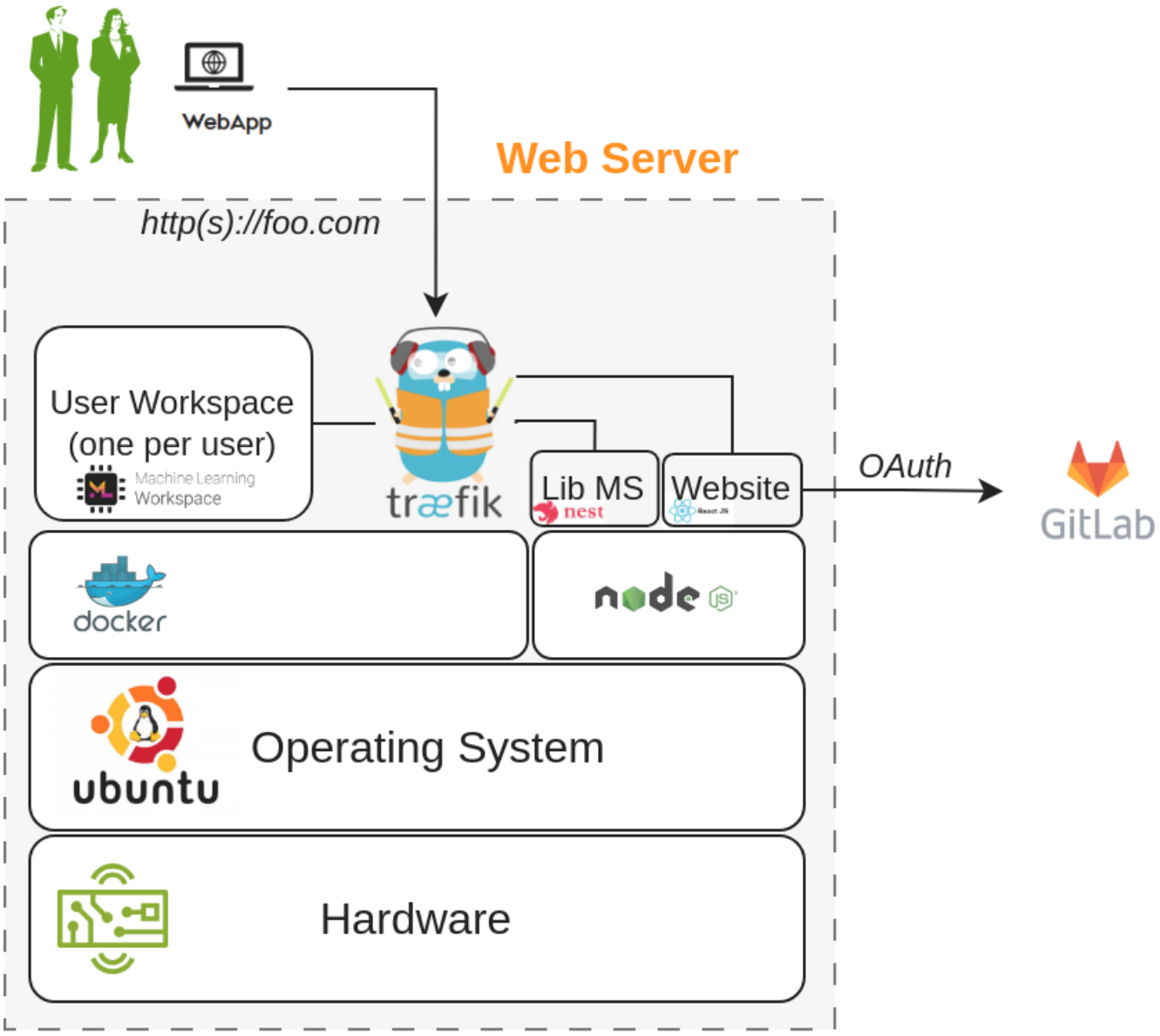
Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.3.3 DTaaS on Linux Operating System

These are installation instructions for running DTaaS application on a Ubuntu Server 22.04 Operating System. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space.

A dummy **foo.com** URL has been used for illustration. Please change this to your unique website URL. It is assumed that you are going to serve the application in only HTTPS mode.

A successful installation will create a setup similar to the one shown in the figure.



Please follow these steps to make this work in your local environment. Download the **DTaaS.zip** from the releases page. Unzip the same into a directory named **DTaaS**. The rest of the instructions assume that your working directory is **DTaaS**.

> **Note**
>
> If you only want to test the application and are not setting up a production instance, you can follow the instructions of trial installation.

Configuration

You need to configure the Traefik gateway, library microservice and react client website.

The first step is to decide on the number of users and their usenames. The traefik gateway configuration has a template for two users. You can modify the usernames in the template to the usernames chosen by you.

TRAEFIK GATEWAY SERVER

You can run the Traefik gateway server in both HTTP and HTTPS mode to experience the DTaaS application. The installation guide assumes that you can run the application in HTTPS mode.

The Traefik gateway configuration is at *deploy/config/gateway/fileConfig.yml*. Change `foo.com` to your local hostname and user1/user2 to the usernames chosen by you.

Do not use `http://` or `https://` in *deploy/config/gateway/fileConfig.yml*.

Authorization

This step requires `htpasswd` commandline utility. If it is not available on your system, please install the same by using

```
sudo apt-get update
sudo apt-get install -y apache2-utils
```

You can now proceed with update of the gateway authorization setup. The dummy username is `foo` and the password is `bar`. Please change this before starting the gateway.

```
rm deploy/config/gateway/auth
touch deploy/config/gateway/auth
htpasswd deploy/config/gateway/auth <first_username>
password: <your password>
```

The user credentials added in *deploy/config/gateway/auth* should match the usernames in *deploy/config/gateway/fileConfig.yml*.

Lib microservice

The library microservice requires configuration. A template of this configuration file is given in *deploy/config/lib* file. Please modify this file as per your needs.

The first step in this configuration is to prepare a filesystem for users. An example file system is in `files/` directory. You can rename the top-level user1/user2 to the usernames chosen by you.

The simplest possibility is to use `local` mode with the following example. The filepath is the absolute filepath to `files/` directory. You can copy this configuration into *deploy/config/lib* file to get started.

```
PORT='4001'
MODE='local'
LOCAL_PATH ='filepath'
LOG_LEVEL='debug'
APOLLO_PATH='/lib'
GRAPHQL_PLAYGROUND='true'
```

React Client Website

GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

You need the following information from the OAuth application registered on Gitlab:

Gitlab Variable Name	Variable name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.foo.com/
Application ID	REACT_APP_CLIENT_ID	
Callback URL	REACT_APP_REDIRECT_URI	https://foo.com/Library
Scopes	REACT_APP_GITLAB_SCOPES	openid, profile, read_user, read_repository, api

You can also see Gitlab help page for getting the Gitlab OAuth application details. Remember to Create gitlab accounts for usernames chosen by you.

UPDATE CLIENT CONFIG

Change the React website configuration in *deploy/config/client/env.js*.

```
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: 'prod',
    REACT_APP_URL: 'https://foo.com/',
    REACT_APP_URL_BASENAME: 'dtaas',
    REACT_APP_URL_DTLINK: '/lab',
    REACT_APP_URL_LIBLINK: '',
    REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',
    REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',
    REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/lab',
    REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',

    REACT_APP_CLIENT_ID: '934b98f03f1b6f743832b2840bf7cccaed93c3bfe579093dd0942a433691ccc0',
    REACT_APP_AUTH_AUTHORITY: 'https://gitlab.foo.com/',
    REACT_APP_REDIRECT_URI: 'https://foo.com/Library',
    REACT_APP_LOGOUT_REDIRECT_URI: 'https://foo.com/',
    REACT_APP_GITLAB_SCOPES: 'openid profile read_user read_repository api',
  };
};
```

Update the installation script

Open `deploy/install.sh` and update user1/user2 to usernames chosen by you.

Perform the Installation

Go to the DTaaS directory and execute

```
source deploy/install.sh
```

You can run this script multiple times until the installation is successful.

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Post-install Check

Now you should be able to access the DTaaS application at: https://foo.com

If you can following all the screenshots from user website. Everything is correctly setup.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.4 Vagrant

2.4.1 DTaaS Vagrant Box

This README provides instructions on creating a custom Operating System virtual disk for running the DTaaS software. The virtual disk is managed by **vagrant**. The purpose is two fold:

- Provide cross-platform installation of the DTaaS application. Any operating system supporting use of vagrant software utility can support installation of the DTaaS software.
- Create a ready to use development environment for code contributors.

There are two scripts in this directory:

Script name	Purpose	Default
`user.sh`	user installation	✅
`developer.sh`	developer installation	❌

If you are installing the DTaaS for developers, the default installation caters to your needs. You can skip the next step and continue with the creation of vagrant box.

If you are a developer and would like additional software installed, you need to modify `Vagrantfile`. The existing `Vagrantfile` has two lines:

```
1       config.vm.provision "shell", path: "user.sh"
2       #config.vm.provision "shell", path: "developer.sh"
```

Uncomment the second line to have more software components installed. If you are not a developer, no changes are required to the `Vagrantfile`.

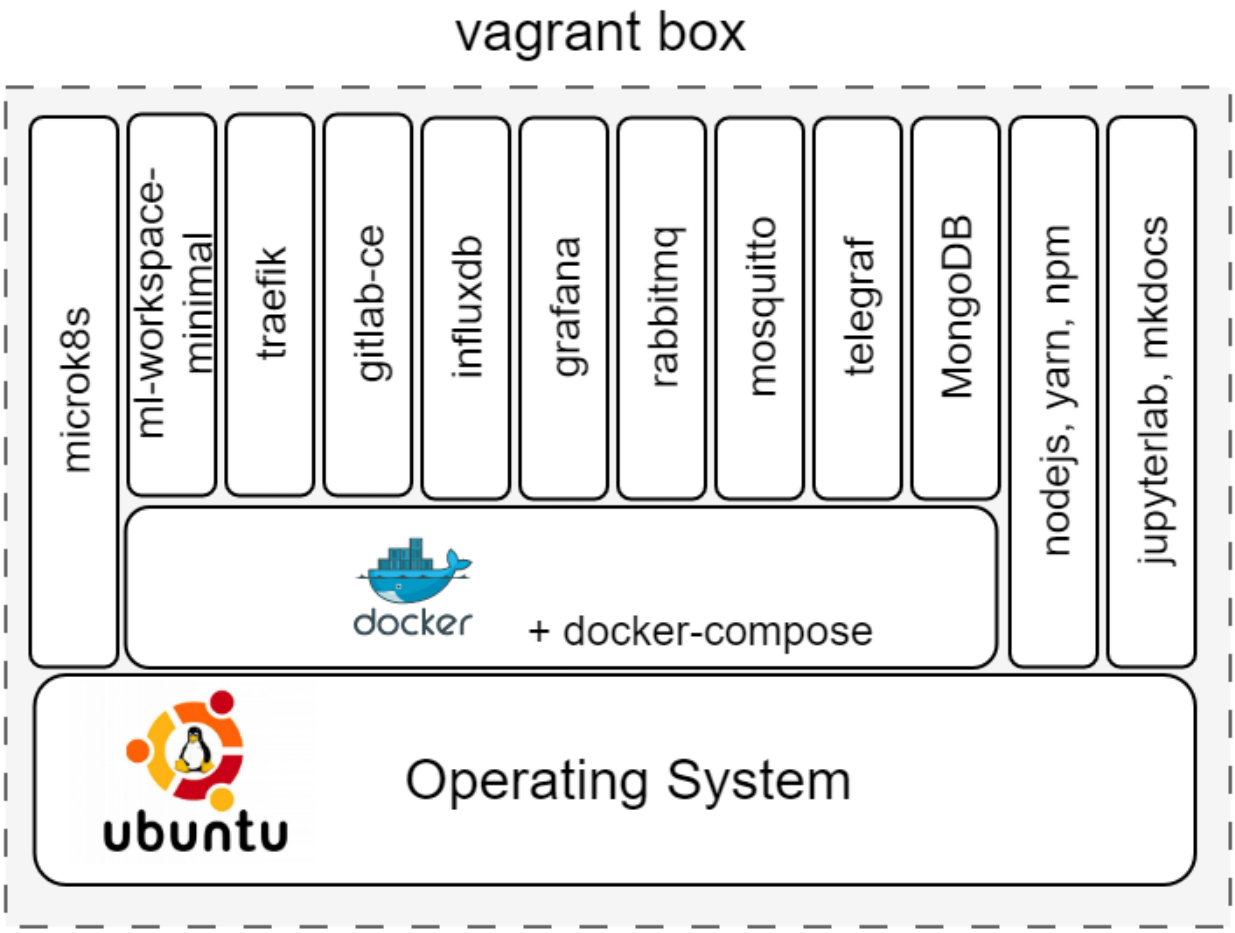
This vagrant box installed for users will have the following items:

1. docker v24.0
2. nodejs v20.10
3. yarn v1.22
4. npm v10.2
5. containers - ml-workspace-minimal v0.13, traefik v2.10, gitlab-ce v16.4, influxdb v2.7, grafana v10.1, rabbitmq v3-management, eclipse-mosquitto (mqtt) v2, mongodb v7.0

This vagrant box installed for developers will have the following items additional items:

- docker-compose v2.20
- microk8s v1.27
- jupyterlab
- mkdocs
- container - telegraf v1.28

At the end of installation, the software stack created in vagrant box can be visualised as shown in the following figure.



The upcoming instructions will help with the creation of base vagrant box.

```
#create a key pair
ssh-keygen -b 4096 -t rsa -f vagrant -q -N ""

vagrant up

# let the provisioning be complete
# replace the vagrant ssh key-pair with personal one
vagrant ssh

# install the oh-my-zsh
sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
# install plugins: history, autosuggestions,
git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions

# inside ~/.zshrc, modify the following line
plugins=(git zsh-autosuggestions history cp tmux)

# to replace the default vagrant ssh key-pair with
# the generated private key into authorized keys
cp /vagrant/vagrant.pub /home/vagrant/.ssh/authorized_keys

# exit vagrant guest machine and then
# copy own private key to vagrant private key location
cp vagrant .vagrant/machines/default/virtualbox/private_key

# check
vagrant ssh #should work

# exit vagrant guest machine and then
vagrant halt

vagrant package --base dtaas \
--info "info.json" --output dtaas.vagrant

# Add box to the vagrant cache in ~/.vagrant.d/boxes directory
vagrant box add --name dtaas ./dtaas.vagrant

# You can use this box in other vagrant boxes using
#config.vm.box = "dtaas"
```

References

Image sources: Ubuntu logo

2.4.2 DTaaS on Single Vagrant Machine

These are installation instructions for running DTaaS software inside one vagrant Virtual Machine. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space to the vagrant box.

Create Base Vagrant Box

Create **dtaas** Vagrant box. You would have created an SSH key pair - *vagrant* and *vagrant.pub*. The *vagrant* is the private SSH key and is needed for the next steps. Copy *vagrant* SSH private key into the current directory (`deploy/vagrant/single-machine`). This shall be useful for logging into the vagrant machines created for two-machine deployment.

Target Installation Setup

The goal is to use the **dtaas** Vagrant box to install the DTaaS software on one single vagrant machine. A graphical illustration of a successful installation can be seen here.