Ansible for ONE 2.0

Installation guide

Ataccama

Table of contents

1. Ansible reference deployment

This repository contains Ansible manifests required for One20 v14.x platform installation. Once all requirements are met and inventories are properly prepared, this set of Ansible manifests helps you to install and configure One20 platform (including base & default data). Observability stack is deployed as well.

Warning

Please note, this package is intended for installing One20 platfom only in version 14.1 and above. It is not compatible with v13.x.

1.1 Note about users and privileges

Ansible can (and should) be installed under an ordinary (non-`root`) user. Whenever we talk about anything related to installing Ansible, we expect commands to be run under your own user. We include `sudo` in the command when `root` privileges are needed.

1.2 Version

This documentation is part of One 2.0 package version `14.5.2-1`.

1.3 Download PDF

You can download a PDF version of this guide.

2. Installation & requirements

2.1 Overview

ONE application is installed using Ansible: an open source tool for server orchestration (setup and management). The installation package contains Ansible configuration, but not the Ansible itself. Before you can start installing ONE, you have to install Ansible with all its dependencies.

Ansible is a Python application and requires specific versions of Python modules (libraries). To avoid clashing with already installed modules, we create a separate Python environment using Python `venv` module.

2.2 Proxy

If your controller accesses the Internet using a proxy server, you have to configure it. Run following commands in your console, substituting the correct protocol, address and port:

```
export http_proxy=http://1.2.3.4:3128
export https_proxy=http://1.2.3.4:3128
```

2.3 Ansible requirements

2.3.1 Installation requirements

A server with supported operating system (see below) with Internet connection is required.

No graphical environment is required (only the text terminal).

2.3.2 Supported operating systems

We currently support Ansible installations (both target and controller are supported) on Ubuntu 20.04 and RHEL 8. Using virtualized system (e.g. using VirtualBox) or remote server accessed using SSH is possible. Although CentOS 7 is supported for installing Gen20 platform, ***it is not supported to use CentOS7 as a Ansible controller***.

Running Ansible on Windows (controller or target) is unsupported.

Note about disk space: Ansible and it's dependencies requires about 2GB of disk space on Ansible controller.

2.3.3 System packages

Start by installing necessary system packages. The command is different for various Linux distributions:

Ubuntu

```
sudo apt update
sudo apt install unzip git python3-pip python3-venv curl
```

RedHat Enterprise Linux (RHEL)

```
sudo dnf makecache
sudo dnf install unzip git python3-pip curl
```

In the case when the ssh connections from the controller to the targets are authenticated with ssh password (instead of ssh keys without passphrase is recommended), `sshpass` package is required and needs to be installed as well.

2.3.4 Installation package

You should have the ONE installation package available in the form of `.zip` file. Unpack it into `~/one/`:

```
mkdir -p ~/one
unzip PACKAGE_NAME -d ~/one
```

2.3.5 Ansible

- Minimal Ansible version required is 4.5
- As the system (APT) module versions might differ from our requirements, we'll install Ansible into venv

If you already have an older version of Ansible installed in a virtual env (e.g. you are running an updated version of Ansible), it is recommended to delete the venv and start with a clean one. You can remove the old version by running

```
rm -r ~/venv
```

Removing venv should not cause any issues even if it contains the current Ansible version. It just wastes a little time.

Create a new venv and activate it.

```
python3 -m venv ~/venv
. ~/venv/bin/activate
```

Your prompt should now start with `(venv)`, indicating anything Python-related now uses modules from your venv. Whenever you want to use this Ansible repo, you have to have your venv active. You can activate the venv in current shell session using `. ~/venv/bin/activate` (note the dot and space).

Now we can install all the Python modules. Enter the directory with Ansible and run:

```
cd ~/one/ansible
pip install --upgrade 'pip>=20.3'
pip install wheel 'ansible<4.6'
pip install -r requirements-pip.txt
```

Danger

Before we can continue, all steps above must be successfully finished without an error. To install Ansible, we use Python virtual environment (venv), which allows us to install required packages version and without touching Python packages installed in OS. There is no other supported way how to install Ansible package (using `apt` or `ndf` for example).

2.3.6 Ansible config

Ansible must be properly configured before it can be used. There is an example of Ansible config file - `ansible-example.cfg`. To make sure it will always be used, copy it to your home directory:

```
cd ~/one/ansible
cp ansible-example.cfg ~/.ansible.cfg
```

One of the most important things here is the log part. With this config, Ansible keeps all playbook runs in a single log file defined in the config.

You should now have a working Ansible installation.

Note on the pip version

Pip uses a new dependency resolver since 20.3.0. Older versions (such as 20.0.2 that is part of Ubuntu 20.04) can't properly resolve the requirements file and fail because of conflicting dependencies.

2.4 Verification

2.4.1 Ansible version

Run:

```
ansible --version
```

It should give output similar to the one below. The Ansible version (at the first line) must be 2.11.X (all values of X are OK). The paths will reflect your own environment. The Python version must be at least 3.5.

```
ansible [core 2.11.5]
  config file = None
  configured module search path = ['/home/ansible/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python3.9/site-packages/ansible
  ansible collection location = /home/ansible/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.9.7 (default, Sep 28 2021, 18:41:28) [GCC 10.2.1 20210110]
  jinja version = 2.11.3
  libyaml = True
```

2.4.2 Ansible local verification

Use Ansible to run a `true` command on the controller (your own computer) to verify it is at least minimally functional. Run:

```
cd ~/one/ansible
ansible-playbook verify-local.yml
```

It should give the output below. The recap must not contain any failed steps. The warnings about missing inventory and host list are expected.

```
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [verify Ansible is usable locally] **************************************************************************

TASK [Gathering Facts] *******************************************************************************************
ok: [127.0.0.1]

TASK [run dummy command] *****************************************************************************************
changed: [127.0.0.1]

TASK [give result] ***********************************************************************************************
ok: [127.0.0.1] => {
    "msg": "Minimal functionality confirmed!"
}

PLAY RECAP *******************************************************************************************************
127.0.0.1                  : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

2.5 References

Official Ansible installation guide

3. Ansible inventory

3.1 Introduction

Having Ansible installed, the most important part is to prepare and customize the deployment before the installation. That means - prepare your inventory and set correctly some variables.

3.2 What is inventory

Ansible works against provided list of managed destination servers (called hosts) in your infrastructure at the same time. These are organized into groups and altogether the organized list of managed hosts is called an inventory.

You can prepare a new inventory either by creating it from scratch, or copying an example and modifying it.

In both cases, you will have to name it: use only letters, numbers, dashes and underscores. We strongly recommend using a customer name as the inventory name (e.g. `ataccama`). Whenever you see `<inventory-name>`, substitute your chosen inventory name, omitting the angle brackets.

3.2.1 Copying an example

We prepared example inventories, meant to be used as templates for the actual installation. These inventories are stored in `~/one/ansible/inventories/example_<variant-name>`. Find the one matching your chosen deployment scenario:

- **Ataccama ONE 2.0**
- installation of the plain Ataccama ONE 2.0 platform and observability stack, without any of the Ataccama AIP modules.
- Variant name: `plain`
- Playbook to run: `site.yml`
- **Ataccama ONE 2.0 + DQIT (issue tracker):**
- Ataccama ONE 2.0 platform, observability stack and DQ Issue Tracker
- Variant name: `dqit`
- Playbook to run: `site.yml`
- **Ataccama ONE 2.0 + all AIP components**
- full Ataccama installation with ONE 2.0 and all AIP components
- Variant name: `full`
- Playbook to run: `site.yml`
- **Plain DPE for hybrid deployment**
- plain DPE that can be used for hybrid deployment
- Variant name: `hybrid`
- Playbook to run: `hybrid-dpe.yml`
- **Standalone RDM deployment**
- RDM installation with all related components
- Variant name: `rdm_standalone`
- Playbook to run: `rdm_standalone.yml`
- **Standalone MDM deployment**
- MDM installation with all related components
- Variant name: `mdm_standalone`
- Playbook to run: `mdm_standalone.yml`
- **MDM + RDM deployment**
- MDM + RDM installation with all related components
- Variant name: `mdm_rdm`
- Playbook to run: `mdm_rdm.yml`

Then, run following command to copy it:

```
cp -r ~/one/ansible/inventories/example_<chosen_variant>/. ~/one/ansible/inventories/<inventory-name>
```

You now have a default inventory that needs to be filled with specific hosts and variables.

Note on component selection

- It is technically possible to choose any component combination by removing groups from inventory. This is officially unsupported: combinatorial explosion prevents us from testing all possible combinations.
- For MDM and DQIT there is a currently known issue with the installed server configuration, as we cannot distribute the demo configuration outside Ataccama. To get those components running, Ataccama consultant has to create the demo project for the customers specific usecase.

3.2.2 Inventory hosts

Inventory hosts are defined in hosts file - `~/one/ansible/inventories/<inventory-name>/hosts.yml`. In the supplied Ansible playbooks we work with the following predefined host groups.

- `one_app` - server for ONE 2.0 Applications (MMM, DPM, CS)
- `frontend` - frontend server (FE)
- `ai_server` - server for AI microservices
- `processing` - server for processing (DPE)
- `dependencies` - server for Ataccama dependencies
- `one_database` - database server for ONE 2.0
- `monitoring_server` - server for observability (monitoring + logging) tools
- `proxies` - server for nginx reverse proxy. Usually the same as `one_app` server.
- `rdm_server` (optional, AIP only) - RDM application and webapp server
- `rdm_database` (optional, AIP only) - database server for RDM
- `mdm_server` (optional, AIP only) - MDM and MDA application server
- `mdm_database` (optional, AIP only) - database server for MDM
- `mdm_frontend` (optional, AIP only) - MDM frontend application server. Usually the same as `frontend`.
- `dqit_server` (optional, DQIT only) - DQIT application and webapp server. Usually the same as `one_app`.
- `dqit_database` (optional, DQIT only) - database server for RDM. Usually the same as `one_database`.
- `monitoring_server` (optional) - monitoring and logging tools server
- `orchestration_server` (optional) - orchestration server (scheduler, online services, workflow)

Groups that are marked optional relate to optional Ataccama modules. If your deployment scenario does not contain a module, its groups won't be present in the inventory. This is expected.

The same server can be listed in multiple groups, depending on the agreed size of the installation.

As a result you should end up with a similar `hosts.yml` file.

```
---
all:
  children:
    monitoring_server:
      hosts:
        monitoring.internal.domain.tld:   # monitoring server domain name
    ai_server:
      hosts:
        app.internal.domain.tld:          # ai server domain name
...
...
...
```

Creating custom modules combination

To create custom combination of modules, start with fresh copy of `example_plain` inventory and modify `hosts.yml` to contain host groups you need. Then proceed with installation as you would proceed with standard inventory.

You can create any combination of hosts groups, that are marked as *optional* in previous chapter. Rest have to be present everytime.

Example for ONE + MDM standalone:

```
all:
  children:
    monitoring_server:
      hosts:
        monitoring.internal.domain.tld:
    frontend:
      hosts:
        app.internal.domain.tld:
    proxies:
      hosts:
        app.internal.domain.tld:
    ai_server:
      hosts:
        app.internal.domain.tld:
    one_app:
      hosts:
        app.internal.domain.tld:
    processing:
      hosts:
        app.internal.domain.tld:
    dependencies:
      hosts:
        dependency.internal.domain.tld:
    one_database:
      hosts:
        app.internal.domain.tld:
# MDM module
    mdm_database:
      hosts:
        app.internal.domain.tld:
    mdm_server:
      hosts:
        app.internal.domain.tld:
    mdm_frontend:
      hosts:
        app.internal.domain.tld:
```

Verifying the inventory machines

To check that inventory machines are reachable and Ansible can connect to them, run the following command

```
ansible -u <username> -b --private-key <path-to-ssh-key> -i <path-to-inventory> all -m ping
```

The output should look similar to the example below. All servers must return SUCCESS: any other result signals error in connection, SSH access or sudo (if used).

```
monitoring.internal.domain.tld | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
app.internal.domain.tld | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
...
```

3.2.3 Inventory variables

Inventory variables are used to customize the deployment. You should create variable files in your inventory in the `group_vars` directory as well. Default variables are placed in `./ansible/defaults/` directory. To include the common global scoped variables, copy them from the `./ansible/defaults` directory in your inventory (see `./ansible/inventories/example/` for reference).

Description of all global variables is given below.

Variables can be defined per role, or for all roles at once. Unless specified otherwise, you should change the global variables. Set them in `group_vars` directory, file `vars.yml`, that is a part of your inventory:

Content of directory `inventories/<inventory-name>/`

```
.
├── group_vars
│ ├── _packages.yml <- default ONE package versions
│ ├── _vars.yml     <- default variables
│ └── secrets.yml      <- customer-specific overrides
└── hosts.yml
```

3.2.4 What is necessary to configure

- licenses
- You should have obtained the license from Ataccama beforehand, that allows to run the modules. For each module for which you have license there's variable `<module-name>_license_key` that should point to path to the license file on local. Default value for each of those variables is `licenses/license.plf`.
- content pack S3 access
- There is requirement to install basic content package. This package is installed by Ansible from S3 storage, which is behind the authentification. To be able to access S3, you need to manage credentials in advance. For more details see `defaults/_vars.yml` file, "MMM content_packs configuration" section. In other words, you need obtain `access_key` and `secret_key` to be able to install Gen2, without basic content installation will fail.
- If you want to install mmm content package that was created with the `jdbcMetastoreProcessing` plugin enabled, set `mmm_jdbc_metastore_processing: true` to your `vars.yml` file to enable its import.
- AIP modules configuration
- RDM and DQIT modules can be configured manually after installation
- MDM module needs to be configured during installation
- ansible can be set to use local files or remote project to configure MDM
- for more details see defaults/vars.yml file, "AIP project configuration" section
- database credentials
- jwt keys for module communication
- secrets (keycloak clients, monitoring endpoints credentials, etc...)
- nginx configuration

Example inventories contain `vars.yml` ready for modification, with comments to guide you through the process. Fill in every value that's missing or marked with `<placeholer-in-angle-brackets>`. Failure to fill in variable `XYZ` so usually manifests as errors containing message `'XYZ' is undefined`.

3.2.5 Licenses

Before the installation you should obtain license file(s) from Ataccama that allows execution of the modules according to the installation/license agreement.

Before running the installation you need to copy the license files on the same machine from which you are provisioning (running Ansible) and set the variables for the paths to the licenses for the modules that you will be installing. Default value for the license file location for each of the variables is `licenses/license.plf`.

```
## the following variables are to set path to licenses for modules that  form ONE 2.0
audit_license_file: /path/to/audit-license-file.plf
dpe_license_file: ...
dpm_license_file: ...
frontend_license_file: ...
mmm_license_file: ...
anomaly_detection_anomaly_detector_license_file: ...
term_suggestions_api_license_file: ...
term_suggestions_feedback_license_file: ...
term_suggestions_neighbors_license_file: ...
term_suggestions_recommender_license_file: ...
## the following variable is to set path to license for orchestration server for ONE 2.0
orchestration_server_license_file: ...
## the following variables are to set path to licenses for modules that form the MDM solution
mdm_license_file: ...
mdm_server_license_file: ...
## the following variables are to set path to licenses for modules that form the RDM solution
rdm_license_file: ...
rdm_server_license_file: ...
## the following variables are to set path to licenses for modules that form the DQIT solution
dqit_license_file: ...
dqit_server_license_file: ...
```

Each path is either absolute or relative to the `./ansible/files` dir.

Not providing the variables will result in an error, similar to this:

```
fatal: [<hostname>]: FAILED! => {"msg": "{{ dpm_files }}: [{'src': '{{ ataccama_internal_encryption.keystore }}', 'dest': '{{ dpm_install_dir }}/etc'},
{'src': '{{ ataccama_internal_encryption.passwordfile }}', 'dest': '{{ dpm_install_dir }}/etc'}, {'src': '{{ dpm_license_file }}',
'dest': '{{ dpm_install_dir }}/license/'}]: 'dpm_license_file' is undefined"}
```

3.2.6 Package versions

We keep package versions in `_packages.yml` file. Apart from the version, you can configure for each module `package_download type` :

- `maven_artifact` (default) - uses packages in Artifactory defined in `artifactory_repo_url`
- `remote_url` - uses remote location to download package, define `package_url` to specify url to the package
- `local` - uses package downloaded to your local machine, specify `package_location` to specify path

3.2.7 Nginx configuration

You need to properly set nginx hosts and certificates for the nginx endpoints. The following variables need to be set correctly:

- `nginx_cert_kind` - type: string (values: `selfsigned` , `letsencrypt` , `provided` , `ownca`)
- The variable is used to set the way how nginx certificates are created (self-signed, ownca-signed, let's encrypt or manually provided certs).
- Self-signed certificates are used if this value is left untouched. This behaviour is intended only for verification than ansible playbook is working. There is a big chance that most of the application won't work properly due to unknown certificate authority.
- As a customer you'd probably have your certificates prepared, so `provided` is the correct way.
- `nginx_certificates` - type: list
- This is a list of certificate files (and their private keys) that will be uploaded to the Nginx server. You can assign them to specific hosts in nginx_hosts later.
- each item in the list has the following keys:
    - `cn` (certificate common name - mandatory)
    - `kind` - certificate kind (`selfsigned` , `ownca` , `letsencrypt` or `provided`)
    - `cert` and `cert_key` - path to certificate and key, in case of provided certificates. Path is either absolute, or relative to `./ansible/files` directory.
- `nginx_domain` - type: string
- Nginx domain name for endpoints on which the application responds (e.g. `customer.domain.tld`)
- Used to unify the hostnames. Specifically, it is used to derive all other hostnames (as is apparent from `nginx_hosts`)
- Provided certificates should be issued for hostnames in this domain (either wildcard or one per each host in `nginx_hosts` - see below)
- This variable has to be set in every environment and default domain (dt.ataccama.dev) should not be used.
- `nginx_hosts` - type: dictionary
- Dictionary assigning nginx host (with associated configuration) to each application endpoint
- Optional, missing components still must be present, but should be marked `enabled: false` . This is technical limitation of Ansible. The configuration in examples contains code that will set this automatically.

It is technically possible to set hostnames unrelated to each other, and issue certificates for each of them. This is unsupported. See certificates section for more details about certificates.

For reasons of simplicity and clarity, it is recommended to use a single certificate (wildcard or SANs), name its files after the `nginx_domain` (e.g. `customer.domain.tld.crt` , `customer.domain.tld.key`) and leave the `nginx_hosts` unchanged.

Example full configuration with possible values:

```
nginx_cert_kind: provided
nginx_certificates:
  - cn: one.customer.domain.tld
    kind: provided
    cert: "/path/to/cert/{{ nginx_domain }}-fullchain.crt"
    cert_key: "/path/to/certkey/{{ nginx_domain }}.key"
nginx_domain: customer.domain.tld
nginx_hosts:
  one:
    enabled: true
    hostname: "one.{{ nginx_domain }}"
    crt: "{{ nginx_domain }}.crt"
    crt_key: "one.{{ nginx_domain }}.key"
  console:
    enabled: "{{ 'mdm_server' in groups.keys() | difference(['all', 'ungrouped']) | default(false) }}" # autogenerate enabled: true/false
    hostname: "console.{{ nginx_domain }}"
    crt: "{{ nginx_domain }}.crt"
    crt_key: "{{ nginx_domain }}.key"
  dpm:
    ...
  dqit:
    ...
  dqit_console:
    ...
  mda:
    ...
  minio:
    ...
  rdm:
    ...
  rdm_console:
    ...
  opensearch_dashboards:
    ...
  monitoring:
    ...
  alertmanager:
    ...
  grafana:
    ...
```

This will result in services being available on following domains when installation is finished:

- Keycloak admin console: https://one.customer.domain.tld/auth/
- Grafana: https://one.customer.domain.tld/grafana/
- OpenSearch Dashboards: https://kibana.customer.domain.tld/
- MMM FE: https://one.domain.tld
- ...

3.2.8 Keycloak secrets

To keep the installation secure, it is required to change keycloak credentials. Generate a strong, random password e.g. using a password manager. If you don't have a password manager handy, you can use following command:

```
< /dev/urandom tr -dc _A-Za-z0-9 | head -c16 ; echo
```

For Keycloak admin and Ataccama realm users, there are three predefined user variables:

- `keycloak_admin` - Keycloak admin console administrator,
- `keycloak_one_admin` - Ataccama ONE 2.0 realm keycloak administrator (can access all applications), and
- `keycloak_monitoring_user` - Ataccama ONE 2.0 realm keycloak user that can access only monitoring endpoints.

Change each one to a strong, unique password. Use separate password for each one.

However, note that current limitation is that the credentials cannot be updated for existing users when the realm is already imported.

Technical limitations of Ansible require you to fill in Keycloak configuration unrelated to passwords. Please, don't touch it: parts of Ansible rely on it. We are sorry.

```
keycloak_admin:
  username: <keycloak-admin-username>
  password: <super-strong-password>
keycloak_one_admin:
  username: <ataccama-admin-username>
  password: <super-strong-password>
keycloak_monitoring_user:
  username: monitoring_user
  password: <super-strong-password>
```

Keycloak module secrets

Each module is authenticating to Keycloak via its own client (token clients). The applications that have web UI also require public client, that has different authentication flow. Fill in all required passwords (the list in your inventory may be shorter, depending on chosen deployment scenario)

```
anomaly_detection_token_client_secret: <super-strong-password>
term_suggestions_token_client_secret: <super-strong-password>
mmm_admin_client_secret: <super-strong-password>
mmm_token_client_secret: <super-strong-password>
dpm_token_client_secret: <super-strong-password>
```

```
one_webapp_token_client_secret: <super-strong-password>
dpe_admin_client_secret: <super-strong-password>
dpe_token_client_secret: <super-strong-password>
rdm_admin_client_secret: <super-strong-password>
rdm_token_client_secret: <super-strong-password>
mdm_admin_client_secret: <super-strong-password>
mdm_token_client_secret: <super-strong-password>
mdm_webapp_public_client_secret: <super-strong-password>
runtime_server_token_client_secret: <super-strong-password>
dqit_webapp_token_client_secret: <super-strong-password>
dqit_steps_token_client_secret: <super-strong-password>
dqit_admin_client_secret: <super-strong-password>
audit_token_client_secret: <super-strong-password>
```

Generate secrets using pwgen

Secrets can be generated using the `pwgen` command and the following commandline from the ansible top level directory:

```
while IFS= read -r line;
  do printf "%s\n" "$line" | sed -re "s#((.*password|secret(_key)?):).*#\1 $(pwgen -n 30)#"
done < ansible/defaults/_secrets.yml  > ansible/inventories/<customer>/group_vars/all/secrets.yml
```

Keycloak demo users deployment

For demo purposes, there are prepared couple of users which can be deployed into Keycloak during installation.

```
keycloak_demo_users: false
```

By default, as those users are unsecured and *should not be deployed into production* environment, their deployment is disabled by default with `false` value. In case you need to deploy those users for demo purposes, you can set this variable to `true` to deploy those users into keycloak. We reccomend to change their passwords immediately after installation to secure them. It's not allowed to enable deployment of those users into production environment or into environment which is exposed into Internet.

Note: those users can be created only during first run of deployment, once Keycloak REALM (of which those users are part of) is deployed, demo users can no longer be added automatically afterwards.

3.2.9 Database secrets

You can configure owner and password for the databases. The following variables are for that.

All Ataccama ONE 2.0 modules currently use the same database server - and use the same owner. Can be configured through `postgres` variable.

```
db_user_one_password: <super-secret-owner-password>
```

DQIT, MDM and RDM modules each use its own database on each server and owner/password can be configured in their module hashes.

```
db_user_dqit_password: <super-secret-owner-password>
db_user_rdm_password: <super-secret-owner-password>
db_user_mdm_password: <super-secret-owner-password>
```

3.2.10 JWT secrets

All modules use jwt tokens to communicate and authenticate to each other. Each module has a variable `<module>_jwt_key` containing both public and secret part of the token.

Use our tool to generate all required tokens:

```
cd ~/one
tools/jwk_generator/generator.py > ~/one/ansible/inventories/<inventory-name>/group_vars/all/jwt_tokens.yml
```

The resulting file may contain tokens for unused modules: these will be simply ignored.

3.2.11 Firewall config

By default, we automatically configure firewall on all target servers. We do not cooperate with existing firewall setup, we override it instead.

The autoconfigured firewall allows incoming SSH (for management) and all ICMP services (to avoid breaking MTU path discovery and similar algorithms) from everywhere, and opens incoming TCP port for every service we install on its host. The configuration also limit source of every connection only to the appropriate servers. All other incoming connections are dropped. The firewall does not limit outgoing connections at all.

In case you need some custom rule (e.g. you use custom monitoring agent that need to be network-accessible), you can add custom rules to this autoconfigured firewall by setting the `custom_firewall_rules` variable. It is not possible to close an open port: we open only ports that are needed for proper functioning of ONE.

For example, this will open ports 11111 and 12345 on all target servers:

```
custom_firewall_rules:
  allowed_tcp_ports:
    - port: 11111
    - port: 12345
```

It is possible to open a range of ports too. This will open all ports in the range of 8080 to 8089 (both inclusive):

```
custom_firewall_rules:
  allowed_tcp_ports:
    - port: 8080:8089
```

If only some hosts require custom open ports, it is recommended to set this variable only for the respective hosts or groups. This will override the global setting in `group_vars/all/vars.yml`. Examples:

Setting this in `~/one/ansible/inventories/<inventory-name>/group_vars/frontend/vars.yml` will open port 23456 on frontend servers:

```
custom_firewall_rules:
  allowed_tcp_ports:
    - port: 23456
```

Setting this in `~/one/ansible/inventories/<inventory-name>/hosts.yml` will open port 33333 only on one of the three application servers (having multiple application servers is not common; this is only an example):

```
[...]
    one_app:
      hosts:
        <first-application-server-hostname>:
        <second-application-server-hostname>:
          custom_firewall_rules:
            allowed_tcp_ports:
              - port: 33333
        <third-application-server-hostname>:
[...]
```

If you set rules in more than one place, group settings will override the global settings, and host settings will override everything.

It is possible to configure more complex rules. If required, see details in `roles/firewall-rules/README.md`.

It is also possible to disable firewall management completely. In this case, an appropriate firewall must be configured beforehand. To disable firewall management, set following:

```
firewall_manage: false
```

Because the current firewall configuration is strict, we don't recommend managing the firewall on target servers using Ansible, and enforcing additional rules either using custom rules (see above) and at the network's edge.

3.2.12 MMM demo data

Currently, as a default the basic and default data coming from the MMM package are imported to the installation. If you wish to change the loaded data and lookups use the following variables that should point to url with the zipped data:

```
mmm_data_import_url: "https://example.com/demodata.zip"
mmm_lookups_import_url: "https://example.com/minio-lookups.zip"
```

3.2.13 Monitoring + alerting config

By default, there is a preconfigured non-working slack receiver in Alertmanager - that sends alerts from Alertmanager to Slack. If you wish to use this, it's enough to set the following two variables in your inventory.

```
prometheus_slack_channel: '#your-slack-channel-for-notifications'
alertmanager_slack_webhook_url: your-slack-webhook-url
```

Alerts that are being sent are about status of the environment - e.g. failed services, running out of memory, etc...

3.2.14 Observability tab

Warning

This is paid feature, enable it with care.

For enabling Observability stack in MMM frontend, you need to set following variable (default value is `false`).

```
ff_props_observability: true
```

3.2.15 SMTP (e-mail) server configuration

ONE Platform can be configured to send e-mail notifications using an external MTA (mail server); this functionality is disabled by default. The e-mails are submitted by MMM component. MMM uses STARTTLS to ensre the connection is properly encrypted and SMTP authentication to authenticate itself to the server.

You can enable SMTP in the inventory:

```
mmm_smtp_enabled: true
mmm_smtp_host: <mail server hostname>
mmm_smtp_port: <mail server port>
mmm_smtp_user: <user>
mmm_smtp_password: <password>
```

3.2.16 Availability check

- `check_availability` - type: bool

The variable is used to enable or disable availability checks for installed applications in roles. There is one exception, Keycloak. It is mandatory for all One 2.0 components, so it is always enabled.

3.2.17 Java options

Since 14.4, default installed Java is Adoptium distribution v 17. You can change this by modifying following properties:

```
java_distribution: adoptiumopenjdk # possible values: openjdk, adoptiumopenjdk
java_distribution_version: 17 # possible values: 11, 17
```

Its possible to use almost any combination of these values on any OS. See following table to check compatibility of your environment.

	Redhat 8	Ubuntu 20	Centos 7
Adoptium 11	✔	✔	✔
Adoptium 17	✔	✔	✔
OpenJDK 11	✔	✔	✔
OpenJDK 17	✔	✔	❌

You can set custom Java options (command-line parameters) for platform's services. The typical use is to increase the amount of heap memory Java can use:

```
audit_server_java_opts: -Xms256m -Xmx384m -XX:MaxRAM=500m
dpe_java_opts: -Xms1024m -Xmx2560m -XX:MaxRAM=3096m
mmm_fe_java_opts: -Xms128m -Xmx256m -XX:MaxRAM=300m
mdm_java_opts: -Xms256m -Xmx288m -XX:MaxRAM=425m
mdm_server_java_opts: -Xms1024m -Xmx3072m -XX:MaxRAM=3200m
mmm_be_java_opts: -Xms512m -Xmx1512m -XX:MaxRAM=1800m
rdm_java_opts: -Xms256m -Xmx288m -XX:MaxRAM=425m
```

3.2.18 Proxy configuration

Ansible needs access to the Internet to download dependencies. You can configure it to use a proxy server for this purpose.

In Linux / UNIX systems, proxy is conventionally configured by setting environment variables named `<PROTOCOL>_proxy` (e. g. `http_proxy`, `https_proxy`), containing an URL of the proxy server. To make Ansible to pass these variables to its (remote) tasks, set in your inventory:

```
environment_vars:
  http_proxy: http://1.2.3.4:3128
  https_proxy: http://1.2.3.4:3128
```

If your controller uses proxy to access the Internet too, you have to set these environment variables in your session too: refer to the installation section

3.2.19 Bastion host

It is an optional group in the inventory that contains a host used for service tasks related to cloud-allocated databases (creation of schema, users etc.). In the common case when databases are installed and managed by Ansible directly on the provided servers, bastion host is not necessary.

A dedicated server used as bastion is recommended (as it can be shut down after the installation); however, you can use the monitoring server instead.

However if you will be using a database as a service (AWS RDS, AWS RDS Aurora or Azure Databases) there is no need to install a database server. Instead, the bastion host is used by Ansible to set up databases configured by inventory variables

```
#Set below variables  if you want to use Azure PGSQL in OneGen2.0
cloud_managed_db: true   # this is used to indicate we want to use cloud managed database instance and must be set true.
cloud_provider: 'azure'  # this is used to indicate which cloud provider dataabase we are using during installation, Use AWS or Azure.
postgresql_domain: <pgsql_database_domain_name>  # this value is used to indicate private domain to which PGSQL database belong to Azure.
db_user_one: <database_username> # database user for one
db_user_one_password: <strong password>  # password for one user
db_user_dqit: <database_username> # database user for dqit
db_user_dqit_password: <strong password> # password for dqit user
db_user_rdm: <database_username> # database user for rdm
db_user_rdm_password: <strong password> # password for rdm user
db_user_mdm: <database_username> # database user for mdm
db_user_mdm_password: <strong password> # password for mdm user

postgresql_admin_user: <postgres_admin_username> # this is admin user of PGSQL database instance
postgresql_admin_password: <strong password>  # password of admin user of PGSQL database instance
one_db_host: <azure pgsql database host for one>  # database instance  for  one
rdm_db_host: <azure pgsql database host for rdm >  # database instance  for rdm
mdm_db_host: <azure pgsql database host for mdm>  # database instance  for mdm
dqit_db_host: <azure pgsql database host for dqit> # database instance  for dqit
```

Please have in mind that this bastion host needs to have allowed network access to the database service.

A dedicated server used as bastion is recommended (as it can be shut down after the installation); however, you can use the monitoring server instead.

However if you will be using a database as a service (AWS RDS, AWS RDS Aurora or Azure Databases) there is no need to install a database server. Instead, the bastion host is used by Ansible to set up databases configured by inventory variables

```
#Set below variables  if you want to use Azure PGSQL in OneGen2.0
cloud_managed_db: true   # this is used to indicate we want to use cloud managed database instance and must be set true.
cloud_provider: 'azure'  # this is used to indicate which cloud provider dataabase we are using during installation, Use AWS or Azure.
postgresql_domain: <pgsql_database_domain_name>  # this value is used to indicate private domain to which PGSQL database belong to Azure.
db_user_one: <database_username> # database user for one
db_user_one_password: <strong password>  # password for one user
db_user_dqit: <database_username> # database user for dqit
db_user_dqit_password: <strong password> # password for dqit user
db_user_rdm: <database_username> # database user for rdm
db_user_rdm_password: <strong password> # password for rdm user
db_user_mdm: <database_username> # database user for mdm
db_user_mdm_password: <strong password> # password for mdm user

postgresql_admin_user: <postgres_admin_username> # this is admin user of PGSQL database instance
postgresql_admin_password: <strong password>  # password of admin user of PGSQL database instance
one_db_host: <azure pgsql database host for one>  # database instance  for  one
rdm_db_host: <azure pgsql database host for rdm >  # database instance  for rdm
mdm_db_host: <azure pgsql database host for mdm>  # database instance  for mdm
dqit_db_host: <azure pgsql database host for dqit> # database instance  for dqit
```

Please have in mind that this bastion host needs to have allowed network access to the database service.

3.3 References

See official Ansible documentation.

4. Running Ansible

4.1 Running the playbook

The main playbook file is located in the `ansible/` directory and called `site.yml`. To deploy the entire application stack, run the following command from the root dir:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b
```

Ansible should be installed and configured before use. Please, read installation section before you'll run Ansible. Put your ssh user instead of `<your-username>`, and the path to your main inventory file (f.e. `hosts.yml`) instead of `<path-to-hosts-file>`. The path to the `ansible-playbook` command must be in the `$PATH` environment variable, so if you see errors such as:

```
$ ansible-playbook
Command 'ansible-playbook' not found
```

In case of errors during the playbook execution, please, read how to read and collect ansible logs in the config section.

4.2 Re-running the playbook

If you have changed something important, such as hosts in your inventory, you better re-run the entire playbook by running the same command as you did when you ran the playbook. If you would like to re-run only some tasks of the playbook, you can use the special Ansible flag - `--start-at-task`. For instance:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --start-at-task "<task-name>"
```

You can see the name of the task you want in the Ansible output. If you have Ansible output cleared or closed, you can look at the Ansible log. See config section. For instance, the name of the task you want is displayed in the log or in the output as:

```
TASK [one_module : Assert that module config defined correctly] ****************
```

To re-run the playbook from this task you should use the full task name from `[` to `]`, for instance:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --start-at-task "one_module : Assert that module config defined correctly"
```

4.3 Summary of the work done

As Ansible runs, it stores information of the work done on the target servers, and compiles it at the end into a set of summary files. These files are stored in the `summaries` directory, and can be processed into a report describing what was done. They are also ordinary YAML files, machine-readable and somewhat readable by humans. It then converts them into an easy-to-read HTML report.

You can generate summary files without running the actual installation by running only tasks with `summary` tag:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --tags summary
```

The report consists of two files: `summary_full.html` and `summary_short.html`. The `_short` file omits details that are of interest mostly to system administrators.

To open these files in a default browser, run

```
xdg-open summary_full.html
xdg-open summary_short.html
```

4.4 Restart an Ansible play at a specific task

In case the play fails, you can restart Ansible at a specific task. Note that it might be a wrong thing to do, e. g. if starting a service fails because an error in its configuration, retrying the restart will not fix the issue. This is common if the fix included inventory changes. (Re-running the whole playbook is always safe.)

To restart at a specific task, run the same command, but add `--start-at-task` parameter:

```
ansible-playbook --start-at-task "Name of the task" <rest of the parameters>
```

There's a limitation: Ansible can't restart at a task that was dynamically included (as we commonly do). If you try to do this, you will get an arror:

```
 [ERROR]: No matching task "Name of the task" found. Note: --start-at-task can only follow static includes.
```

In that case, you have to find the name of the include task itself and start at that one.

4.5 Restart services using Ansible

You can start, stop and restart Platform services using Ansible tags. Supported use-cases are stoppping and later starting Platform for the purpose of database maintenance and refreshing license files (followed by restart).

4.5.1 Updating license files

Update license files in your Ansible installation (generally in the `ansible/files` directory) and run:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --tags license,restart
```

The `license` tag limits Ansible so that only license-related tasks run. The `restart` tag is analogous except it relates to restarting services.

Generally, you want to run both tags together: the license file changes are not picked up until services are restarted. Also, the restart tasks run only when license files (or containing directories) change. It might be useful to run the license changes only and do the restart later manually (e. g. as part of a larger maintenance effort).

Please note the restarts are "edge triggered" - once the license files are changed, following runs won't restart any services. You can use the `stop` and `start` tags as desribed below and restart MANTA manually (also noted below).

4.5.2 Stopping and starting services

To stop all database-using Platform services, run

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --tags stop
```

analogously, to start the same services:

```
ansible-playbook -u <your-username> -i <path-to-hosts-file> ansible/site.yml -b --tags start
```

When both tags are combined, the services are stopped and then immediately started again (efectively restarted).

The set of stopped / started services include Keycloak, but doesn't include MANTA. If you need MANTA restarted too, please do it manually.

4.6 Moving Ansible temporary files

Some systems have very limited space for temporary files, or use relatively slow filesystems (e. g. NFS) for root partition. It may be useful to move temporary files created by Ansible to a different directory.

Note that this is only best-effort configuration: Ansible calls external programs that might not respect the settings made here. Space-constrained `/tmp` and `/var/tmp` are still unsupported for production deployment; sufficient temporary file storage is a general requirement for reliable Linux / Unix system operation.

Furthermore, some tasks download or create files locally to the controller before uploading them to the target server. These tasks are configured to always use the default temporary directories (usually `/tmp`).

Subject to the caveats above, you can direct Ansible to store its temporary files to a specific directory by adding following variables to your inventory:

```
ansible_remote_tmp: <directory>
temp_folder: <directory>
environment_vars:
  TMP: <directory>
  TEMP: <directory>
  TMPDIR: <directory>
```

The directory `<directory>` must already exist on every target server and have `1777` privileges (note the sticky bit).

4.7 Securing Nginx deployment

To provide more secure Nginx connection via TLS security we have enforced more SSL parameters which require random DIFFIE_HELLMAN parameter file Default key size for DH parameter file is set to 2048.

```
diffie_hellman_key_size: 4096
```

5. Ansible installation tips & tricks

5.1 Get details from SSL certificate file

Using *openssl* binary we can check certificate and also key.

Check content of certificate (please note: the list of DNS hostnames can be different to official documentation, it's just an example):

```
$ openssl x509 -in /etc/ssl/one20/one.domain.tld -noout -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            03:52:e7:bd:81:26:42:c4:59:6e:37:24:61:90:3e:0d:0c:1f
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Let's Encrypt, CN = R3
        Validity
            Not Before: May 25 06:33:10 2021 GMT
            Not After : Aug 23 06:33:10 2021 GMT
        Subject: CN = one.domain.tld
...
...
...
X509v3 Subject Alternative Name:
                DNS:console.domain.tld, DNS:cs.domain.tld, DNS:dpm.domain.tld, DNS:dqit-console.domain.tld, DNS:dqit.domain.tld, DNS:kibana.domain.tld, DNS:...
...
...
```

Check certificate with private key (please, note using *sudo* in case of key):

```
$ openssl x509 -modulus -noout -in one.domain.tld-fullchain.crt |openssl md5
(stdout)= 9d9dbe3481db96465d9d798285e29d07

$ sudo openssl rsa -modulus -noout -in one.domain.tld.key |openssl md5
(stdout)= 9d9dbe3481db96465d9d798285e29d07
```

Output should be the same.

5.2 Check application logs

Following command should be run on VM where is desired component installed.

This can be run on app node, where MMM BE is installed:

```
journalctl -u mmm-backend.service
```

Will show you last messages from MMM BE.

and follow the log (live log stream):

```
journalctl -f -u mmm-backend.service
```

Command above can be used for every service which is running on particular node/mv.

5.3 Ansible Logs

Ansible can write logs of every run. Read about ansible configuration about how to enable this. If you use config from the config section, the log file will be located in `~/.ansible/ansible.log` . Ansible writes all the logs into this one file. If you would like to find the log of a specific run, you can find it using timestamps in the log file. If you would like to change the path to log file, you can do it using `log_path` setting in your Ansible configuration file.

5.3.1 References

Official Ansible logging documentation

6. Encrypting secrets

Optionally, it's possible to use SOPS to encrypt secrets such as JWK private keys, passwords and other secrets used by Ansible.

6.1 SOPS overview

SOPS is a program able to encrypt YAML and JSON files in such a way that mapping keys and overall structure is preserved, but values (mapping values, list members etc.) are unreadable without a proper key. It also protects the encrypted data from tampering using authenticated encryption (namely AES256 in GCM mode).

SOPS can use multiple cryptographic backends. We (unconditionally) use `age`, the modern file encryption tool.

6.1.1 How data are encrypted

(This part is not necessary to understand usage of SOPS, but might be useful to understand its security properties.)

SOPS uses a combination of symmetric and assymetric cryptography to encrypt the data.

When encrypting data, SOPS generates a (symmetric) *data encryption key (DEK)* and uses it to encrypt every value in the file. DEK is then encrypted using every recipients' public key, and appended (along with some metadata) under a `sops` key to the file. All encrypted data are *autenticated:* cryptographic checksums are computed to ensure no-one can change, add or remove any part of them.

When decrypting, SOPS checks the relevant authentication codes, then uses your private key to decrypt the DEK, and finally uses it to decrypt the data.

Details (ciphers, data authentication and more) can be found in the official documentation

6.2 Setup

This chapter assumes you have a working Ansible installation.

6.2.1 Binaries

The required tools (`age` and `sops`) are included in the package, and can be found in the `tools` directory. These are statically-linked binaries, and should work on any amd64 Linux system.

Ansible currently requires `sops` to be stored in a directory in `PATH` variable. We recommend adding the `tools` directory into your PATH:

```
PATH=$PATH:~/one/tools
```

Edit your `.bash_profile` to make the change permanent.

6.2.2 Keypair generation

Entites who can decrypt data (age calls them "recipients") are identified by their public key, and use their private key to decrypt data. Recipients can be both people and machines (e. g. deployment pipelines), every recipient needs its own key pair.

You can generate your own keypair by running

```
mkdir -p ~/.config/sops/age
age-keygen -o ~/.config/sops/age/keys.txt
```

The `age` command will store the private key in a default SOPS location, and print the public part to the console. In case you lose the public key, you can generate it fom the private key by running `age-keygen -y ~/.config/sops/age/keys.txt`.

6.2.3 Recipients

Collect all recipients' public keys into a single file called `recipients`. Store them ane after another, one key per line. You don't need any private keys except your own; if somebody sends you (even by mistake) their private key, consider it compromised and let them generate another key pair.

6.2.4 Encrypting files

You can encrypt any YAML file. We assume you have collected all secrets in two files (as directed in Inventory preparation chapter) stored in `ansible/inventories/<inventory-name>/group_vars/all/` called `secrets.yml` and `jwt_tokens.yml`, and will encrypt those two files. For technical reasons, it is not possible to correctly process files containing Jinja expressions: this is the reason we collect secrets into separate files.

SOPS plugin (that is responsible for decrypting the data for Ansible use) expects the encrypted files' names to end with `.sops.yml`. We use a SOPS configuration file (`ansible/.sops.yaml`) that makes SOPS to encrypt only these files. Therefore, you can simply run:

```
mv ansible/inventories/<inventory-name>/group_vars/all/secrets.yml ansible/inventories/<inventory-name>/group_vars/all/secrets.sops.yml
mv ansible/inventories/<inventory-name>/group_vars/all/jwt_tokens.yml ansible/inventories/<inventory-name>/group_vars/all/jwt_tokens.sops.yml
sops --encrypt --in-place ansible/inventories/<inventory-name>/group_vars/all/*.sops.yml
```

6.2.5 Configuring ansible for decryption

If you don't have an Ansible configuration file, create it by running

```
touch ~/.ansible.cfg
```

You have to enable Ansible SOPS plugin to allow Ansible to decrypt SOPS-encrypted files. Ensure you have following lines in your Ansible configuration file:

```
[defaults]
vars_plugins_enabled = host_group_vars,community.sops.sops
```

Your secrets are now encrypted, and Ansible is configured to decrypt them automatically using your private key stored in `~/.config/sops/age/keys.txt`. All other recipients can similarly decrypt the files.

There is also a file `ansible/ansible-example-encryption.cfg` that contains an example configuration.

6.3 Things to avoid

The encrypted data are cryptographically checksummed to prevent unauthorized changes. All changes to encrypted data must be done through SOPS:

```
sops ansible/inventories/<inventory-name>/group_vars/all/secrets.sops.yml # opens editor
```

6.4 References

SOPS: https://github.com/mozilla/sops

age: https://age-encryption.org/

6.5 Rationale for using age

GPG is bad. In the past, it might be a cutting edge of cryptography, but it aged poorly and it's not secure anymore. The overview article https://latacora.singles/2019/07/16/the-pgp-problem.html clearly points out its failings.

`age` is a much more simple and modern replacement. It has two implementations (giving us confidence that the design is sound and data formats are stable) in memory-safe languages. Its operational simplicity (recipient identified by plain keys, straightforward trust management etc.) prevents many classes of human error and its simple implementation reduces the probability of programmer errors.

While the implementation is young, the underlying cryptographic primitives (AES256, Curve25519) are well researched and designed for resiliency against common implementation issues.

7. Hybrid DPE installation using Ansible

This document serves as an installation guide for the hybrid DPE that is preconfigured to connect to Ataccama PaaS environment.

7.1 Ansible installation

As a first step, one needs to have Ansible installed on his control node - which can be either a local machine with any Linux distribution, or the installation can be run even from the DPE server / node itself. The server needs to have connectivity to the internet (at least for the time of the installation), as we need to download the necessary packages.

See installation section of the docs, which covers the installation of Ansible itself.

7.2 Prepare the inventory

Briefly, an inventory is a list of servers, that are provisioned via Ansible (see a more detailed description on what an inventory is).

To prepare the inventory for the hybrid DPE installation, use the example inventory from `inventories/example_hybrid` (copy it e.g. to `inventories/customer`). Then one has to modify both the hosts file and the variables files.

7.2.1 Hosts file

The hosts file is a file with the provisioned hosts. An example hosts file should look as follows.

```
all:
  children:
    processing:
      hosts:
        <dpe-server-hostname>:
        <dpe-2-server-hostname>: # optional, but at least one server in the processing group is required
          key1: value1           # for any server (host), one can optionally define also variables, which are host-specific (e.g. license)
        ...
```

When building your own inventory, the `<dpe-server-hostname>` yaml dictionary entries should be replaced with the right hosts.

It is possible to specify one or multiple hosts in the processing group, for each host you can have also host-specific variables (see above).

7.2.2 Variables

The variables are placed in `group_vars/all/vars.yml`. You need to modify the default ones and change the various endpoints and secrets to match the production PaaS environment values to be able to successfully connect to the PaaS environment.

Here we list the ones, that need to be changed/filled, as they are specific for each environment.

- `dpe_license_file` - path to the DPE license file on the Ansible controller. Either define it in group vars or for each host separately (when provisioning multiple DPEs). Default value is `licenses/license.plf`
- `keycloak_url` - PaaS Keycloak authentication server endpoint, e.g. `https://customer.dev.ataccama.online/auth/`
- `dpe_token_client`, `dpe_admin_client` - DPE token and admin clients credentials
- `minio_url` - PaaS MinIO (S3 storage) endpoint, e.g. `https://minio.customer.dev.ataccama.online`
- `minio` - configure MinIO credentials, through the following variables:

```
minio:
  access_key: <access-key>
  secret_key: <secret-key>
```

- `dpe_jwt_key` - private key of the on-premise DPE.
- `dpm` - DPM grpc host (e.g. `dpm-grpc.customer.dev.ataccama.online`), the gRPC and HTTP port numbers should remain unchanged.
- `dpm_jwt_key` - public key of the DPM module, containing all the respective parts (`name`, `content`, `fingerprint`).
- jwt key content must be in the form: `content: '{"kty":"EC", .... ,"alg":"ES256"}'`

Optional DPE configuration

One can provide additional configuration to DPE, whatever is for the client required. For that, add additional DPE application.properties to `dpe_additional config` variable. The setup in example is to set the DPE-DPM communication in bidirectional mode, that requires only just one connection opened between those two modules.

Also, we have preconfigured a few JDBC datasources as a starting example (which are most commonly used), and for those we download JDBC drivers. List of the downloaded drivers is contained in the `dpe_drivers` variable.

7.3 DPE installation

Having the inventory prepared, you can then proceed with the DPE installation. To install the DPE, run a playbook called `hybrid-dpe.yml`. The command to execute the playbook is the following, needs to be executed from the `ansible` directory,

```
ansible-playbook -i <path-to-inventory> -u <username> --private-key <path-to-private-key> -b hybrid-dpe.yml
### in case of the example inventory, connecting via `admin` account and it's private key located in `~/.ssh/admin-private` as follows:
ansible-playbook -i inventories/example_hybrid/ -u admin --private-key ~/.ssh/admin-private -b hybrid-dpe.yml
```

The user needs to have ssh access to all the hosts with root (e.g. password-less sudo) privileges.

7.3.1 Installation checks

At the beginning of the installation, several checks are being run to assure that all the PaaS endpoints can be reached from the DPE node. The successful outcome of these checks looks as follows (considering the example above, that has 2 DPE nodes).

```
TASK [Hybrid preinstallation checks] *************************************************************************************************************************

TASK [system : Check connectivity to Keycloak] ***************************************************************************************************************
ok: [dpe-1-server-hostname]
ok: [dpe-2-server-hostname]

TASK [system : Check connectivity to MinIO] ******************************************************************************************************************
ok: [dpe-1-server-hostname]
ok: [dpe-2-server-hostname]

TASK [system : Check connectivity to dpm grpc endpoint] ******************************************************************************************************
ok: [dpe-1-server-hostname]
ok: [dpe-2-server-hostname]
```

In case any of these checks fail, Ataccama PaaS environment cannot be reached from the DPE node(s), which can indicate a firewall / networking issue. Investigate and assure that the connection is fine before you proceed with the installation.

The successful outcome of running the play should look similarly as follows.

```
...

PLAY RECAP ***************************************************************************************************************************************************
dpe-1-server-hostname      : ok=86   changed=31   unreachable=0    failed=0    skipped=32   rescued=0    ignored=2
dpe-2-server-hostname      : ok=86   changed=31   unreachable=0    failed=0    skipped=32   rescued=0    ignored=2
```

DPE availability check

At the end of the play, a check that DPE is running fine is performed. The check can be turned off by setting the variable `check_availability: false`. The check is successful if the following task finishes without issues.

```
TASK [dpe : Wait for dpe to come up (check monitoring endpoint ready)] ***************************************************************************************
ok: [dpe-1-server-hostname]
ok: [dpe-2-server-hostname]
```

Else, if the task fails, one can see from the output, what is the cause for the DPE not starting properly. In the following example is e.g. the failure caused by DPE not being able to reach DPM (hence the monitoring endpoint is not returning 200).

```
fatal: [dpe-1-server-hostname]: FAILED! => {"attempts": 30, .... "json": {"components": {"db": {"details": {"database": "H2",
"validationQuery": "isValid()"}, "status": "UP"}, "diskSpace": {"details": {"exists": true, "free": 23339401216, "threshold": 10485760, "total": 31036686336},
"status": "UP"}, "dpm": {"details": {"state": "CONNECTING"}, "status": "DOWN"}, "livenessState": {"status": "UP"}, "ping": {"status": "UP"}, "readinessState":
{"status": "UP"}}, "groups": ["liveness", "readiness"], "status": "DOWN"}, "msg": "Status code was 503 and not [200]: HTTP Error 503: ", "redirected": false,
"status": 503, "transfer_encoding": "chunked", "url": "http://dpe-1-server-hostname:8034/actuator/health", "x_correlation_id": "8524f5"}
```

For more information on running the Ansible playbooks and other command line parameters, see Running Ansible section of this documentation.

8. Inventory examples

Example configuration snippets used to deploy Ataccama One Gen 2.0

8.1 Setup with on prem installed dependencies

```
db_user_one: <one_user>
db_user_one_password: <one_user_password>
db_user_dqit: <dqit_user>
db_user_dqit_password: <dqit_user_password>
db_user_rdm: <rdm_user>
db_user_rdm_password: <rdm_user_password>
db_user_mdm: <mdm_user>
db_user_mdm_password: <mdm_user_password>
...
postgres:
  host: "{{ groups['one_database'][0] }}"
  database:
    owner: "{{ db_user_one }}"
    password: "{{ db_user_one_password }}"
  port: "5432"
...
dqit:
  host: "{{ groups['dqit_server'][0] }}"
  ... # other dqit settings
  database:
    host: "{{ groups['dqit_database'][0] }}"
    port: "{{ postgres.port }}"
    name: dqit
    owner: {{ db_user_dqit }}
    owner_password: {{ db_user_dqit_password }}
...
rdm:
  ... # other rdm settings
  database:
    host: "{{ groups['rdm_database'][0] }}"
    port: "5432"
    name: rdm
    user: "{{ db_user_rdm }}"
    password: "{{ db_user_rdm_password }}"
...
mdm:
  ... # other mdm settings
  database:
    host: "{{ groups['mdm_database'][0] }}"
    port: "5432"
    user: "{{ db_user_mdm }}"
    password: "{{ db_user_mdm_password }}"
    databases:
      - name: mdc_db
        owner: mdm
      - name: it_db
        owner: mdm
      - name: esb_db
        owner: mdm
      - name: mda_cache
        owner: mdm
      - name: external
        owner: mdm
      - name: log_db
        owner: mdm
      - name: eh_db
        owner: mdm
```

8.2 Setup with AWS RDS PostgreSQL

```
cloud_provider: 'aws'
postgresql_admin_user: <admin_user>
postgresql_admin_password: <admin_password>
database_host_one: <aws rds db hostname>
database_host_dqit: <aws rds db hostname>
database_host_rdm: <aws rds db hostname>
database_host_mdm: <aws rds db hostname>
postgresql_domain: ''
db_user_one: <one_user>
db_user_one_password: <one_user_password>
db_user_dqit: <dqit_user>
db_user_dqit_password: <dqit_user_password>
db_user_rdm: <rdm_user>
db_user_rdm_password: <rdm_user_password>
db_user_mdm: <mdm_user>
```

```
db_user_mdm_password: <mdm_user_password>
...
postgres:
  host: "{{ database_host_one }}"
  database:
    owner: "{{ db_user_one }}"
    password: "{{ db_user_one_password }}"
  port: "5432"
  postgresql_server:
    hostname: "{{ database_host_one }}"
    admin_user: "{{ postgresql_admin_user }}"
    admin_password: "{{ postgresql_admin_password }}"
    parameters: "?sslmode=require"
    users:
      - name: "{{ db_user_one }}"
        password: "{{ db_user_one_password }}"
        role_attr_flags: CREATEDB,CREATEROLE,NOSUPERUSER,INHERIT,LOGIN
        databases:
          - name: mmm
            extensions:
              - btree_gin
          - name: events
          - name: dpm
          - name: ai
          - name: keycloak
          - name: audit
      - name: "{{ db_user_dqit }}"
        password: "{{ db_user_dqit_password }}"
        role_attr_flags: CREATEDB,CREATEROLE,NOSUPERUSER,INHERIT,LOGIN
        databases:
          - name: dqit
...
dqit:
  host: "{{ database_host_dqit }}"
  ... # other dqit settings
  database:
    host: "{{ database_host_dqit }}"
    port: "{{ postgres.port }}"
    name: dqit
    owner: {{ db_user_dqit }}
    owner_password: {{ db_user_dqit_password }}
  postgresql_server:
    hostname: "{{ database_host_dqit }}"
    admin_user: "{{ postgresql_admin_user }}"
    admin_password: "{{ postgresql_admin_password }}"
    parameters: "?sslmode=require&amp;gssEncMode=disable"
    users:
      - name: "{{ db_user_dqit }}"
        password: "{{ db_user_dqit_password }}"
        role_attr_flags: CREATEDB,CREATEROLE,NOSUPERUSER,INHERIT,LOGIN
        databases:
          - name: dqit
...
rdm:
  ... # other rdm settings
  database:
    host: "{{ database_host_rdm }}"
    port: "5432"
    name: rdm
    user: "{{ db_user_rdm }}"
    password: "{{ db_user_rdm_password }}"
  postgresql_server:
    hostname: "{{ database_host_rdm }}"
    admin_user: "{{ postgresql_admin_user }}"
    admin_password: "{{ postgresql_admin_password }}"
    parameters: "?sslmode=require&amp;gssEncMode=disable"
    users:
      - name: "{{ db_user_rdm }}"
        password: "{{ db_user_rdm_password }}"
        role_attr_flags: CREATEDB,CREATEROLE,NOSUPERUSER,INHERIT,LOGIN
        databases:
          - name: rdm
...
mdm:
  ... # other mdm settings
  database:
    host: "{{ database_host_mdm }}"
    port: "5432"
    user: "{{ db_user_mdm }}"
    password: "{{ db_user_mdm_password }}"
    databases:
      - name: mdc_db
        owner: "{{ db_user_mdm }}"
      - name: it_db
        owner: "{{ db_user_mdm }}"
      - name: esb_db
        owner: "{{ db_user_mdm }}"
      - name: mda_cache
        owner: "{{ db_user_mdm }}"
      - name: external
        owner: "{{ db_user_mdm }}"
      - name: log_db
        owner: "{{ db_user_mdm }}"
      - name: eh_db
```

```
        owner: "{{ db_user_mdm }}"
  postgresql_server:
    hostname: "{{ database_host_mdm }}"
    admin_user: "{{ postgresql_admin_user }}"
    admin_password: "{{ postgresql_admin_password }}"
    parameters: "?sslmode=require&amp;gssEncMode=disable"
    users:
      - name: "{{ db_user_mdm }}"
        password: "{{ db_user_mdm_password }}"
        role_attr_flags: CREATEDB,CREATEROLE,NOSUPERUSER,INHERIT,LOGIN
        databases:
          - name: mdc_db
          - name: it_db
          - name: esb_db
          - name: mda_cache
          - name: external
          - name: log_db
          - name: eh_db
```

8.3 Setup with Azure PostgreSQL

Below are variables that must be set in vars.yml

```
cloud_managed_db: true   # this is used to indicate we want to use cloud managed database instance and must be set true
cloud_provider: 'azure'  # this is used to indicate which cloud provider  database we are using during installation

# This value is used to indicate private domain to which PGSQL database belong to Azure.
# This must be a top-level domain.
# E.g., if the fqdn is postgres.some.customer.domain, the value should be 'some.customer.domain'.
postgresql_domain: <pgsql_database_domain_name>

# Azure Postgres names are used in the 'user@domain' format.
# Please, put only user part there.
db_user_one: <database_username> # database user for one
db_user_one_password: <strong password>  # password for one user
db_user_dqit: <database_username> # database user for dqit
db_user_dqit_password: <strong password> # password for dqit user
db_user_rdm: <database_username> # database user for rdm
db_user_rdm_password: <strong password> # password for rdm user
db_user_mdm: <database_username> # database user for mdm
db_user_mdm_password: <strong password> # password for mdm user

postgresql_admin_user: <postgres_admin_username> # this is admin user of PGSQL database instance without @domain part
postgresql_admin_password: <strong password>  # password of admin user of PGSQL database instance

# The value must be a second-level domain.
# E.g., if the fqdn is postgres.some.customer.domain, the value should be 'postgres'.
one_db_host: <azure pgsql database host for one>  # database instance  for  one
rdm_db_host: <azure pgsql database host for rdm >  # database instance  for rdm
mdm_db_host: <azure pgsql database host for mdm>  # database instance  for mdm
dqit_db_host: <azure pgsql database host for dqit> # database instance  for dqit


all:
  children:
    monitoring_server:
      hosts:
        <mon-server-hostname>:
    ai_server:
      hosts:
        <one-server-hostname>:
    testing_data:
      hosts:
        <dependency-server-hostname>:
    one_app:
      hosts:
        <one-server-hostname>:
    processing:
      hosts:
        <procesing-server-hostname>:
    dependencies:
      hosts:
        <dependency-server-hostname>:
    proxies:
      hosts:
        <one-server-hostname>:
    rdm_server:
      hosts:
        <one-server-hostname>:
    dqit_server:
      hosts:
        <one-server-hostname>:
    frontend:
      hosts:
        <one-server-hostname>:
    mdm_server:
      hosts:
        <mdm-server-hostname>:
```

```
    orchestration_server:
      hosts:
        <orch-server-hostname>:
    bastion:
      hosts:
        <bastion-server-hostname>:
```

If you don't have a dedicated bastion server, you can use a monitoring server as the bastion. It's needed only to operate with DBs. Bastion server is used as a proxy (jump host) to forward SQL commands into DBs which cannnot be accesed directly from orchestration server.

8.4 Setup with AWS RDS

```
cloud_managed_db: true   # this is used to indicate we want to use cloud managed database instance and must be set true
cloud_provider: 'aws'  # this is used to indicate which cloud provider database we are using during installation


# AWS RDS users. You may omit user variables if defaults suit.
db_user_one: <database_username> # database user for one
db_user_one_password: <strong password>  # password for one user
db_user_dqit: <database_username> # database user for dqit
db_user_dqit_password: <strong password> # password for dqit user
db_user_rdm: <database_username> # database user for rdm
db_user_rdm_password: <strong password> # password for rdm user
db_user_mdm: <database_username> # database user for mdm
db_user_mdm_password: <strong password> # password for mdm user

postgresql_admin_user: <postgres_admin_username> # this is the admin user of AWS RDS
postgresql_admin_password: <strong password>  # password of the AWS RDS admin user

# The value must be a fqdn (fully qualified domain name).
# Note that postgresql_domain variable is not needed in the case of AWS RDS.
one_db_host: <aws rds endpoint for one>  # database endpoint for one
rdm_db_host: <aws rds endpoint for rdm >  # database endpoint for rdm
mdm_db_host: <aws rds endpoint for mdm>  # database endpoint for mdm
dqit_db_host: <aws rds endpoint for dqit> # database endpoint for dqit


all:
  children:
    monitoring_server:
      hosts:
        <mon-server-hostname-or-ip>:
    ai_server:
      hosts:
        <one-server-hostname-or-ip>:
    testing_data:
      hosts:
        <dependency-server-hostname-or-ip>:
    one_app:
      hosts:
        <one-server-hostname-or-ip>:
    processing:
      hosts:
        <procesing-server-hostname-or-ip>:
    dependencies:
      hosts:
        <dependency-server-hostname-or-ip>:
    proxies:
      hosts:
        <one-server-hostname-or-ip>:
    rdm_server:
      hosts:
        <one-server-hostname-or-ip>:
    dqit_server:
      hosts:
        <one-server-hostname-or-ip>:
    frontend:
      hosts:
        <one-server-hostname-or-ip>:
    mdm_server:
      hosts:
        <mdm-server-hostname-or-ip>:
    orchestration_server:
      hosts:
        <orch-server-hostname-or-ip>:
    bastion:
      hosts:
        <bastion-server-hostname-or-ip>:

# In the case of bastion server you can use ssh proxy command
#  vars:
#    ansible_ssh_common_args: |-
#      -o ProxyCommand="ssh -W %h:%p -q <ansible_operator_ssh_username>@<bastion-server-external-hostname-or-ip>"
#
```

If you don't have a dedicated bastion server, you can use a monitoring server as the bastion. It's needed only to operate with DBs. In the case of AWS RDS Aurora you must use read-write endpoints as database hosts. Bastion server is used as a proxy (jump host) to forward SQL commands into DBs which cannnot be accesed directly from the controller.

```
  bastion:
    hosts:
      <bastion_server_hostname>:
```

Bastion server is used as a proxy (jump host) to forward SQL commands into DBs which cannnot be accesed directly from orchestration server.

8.5 Setup with AWS S3

Since we are not going use Minio as your object storage, we should remove minio role from dependencies.yml playbook. Your AWS account should also possess necessary privileges to manipulate with object in your S3 bucket (e.g. s3:ListBucket + s3:*Object*).

```
minio_url: <S3 bucket url>
minio:
  secret_key: <aws secret key>
  access_key: <aws access key>
```

8.6 Setup with Nginx monitoring enabled

Nginx has the `ngx_http_stub_status_module` that provides access to basic status information. See official ngx_http_stub_status_module page for details. Metrics are scraped by `nginx-prometheus-exporter`.

```
nginx_status_stub_enabled: true
nginx_exporter_port: 9113
```

8.7 Setup with partially custom repositories

You can partially replace the system repositories and repositories installed by the playbook with the custom ones. For example, when it's not possible to use HTTP-based repositories. The current limitation is that you can't replace repositories installed by external roles.

8.7.1 Ubuntu

```
system_custom_repos_content: |
  deb https://mirror.dkm.cz/ubuntu/ focal main restricted
  deb https://mirror.dkm.cz/ubuntu/ focal-updates main restricted
  deb https://mirror.dkm.cz/ubuntu/ focal universe
  deb https://mirror.dkm.cz/ubuntu/ focal-updates universe
  deb https://mirror.dkm.cz/ubuntu/ focal multiverse
  deb https://mirror.dkm.cz/ubuntu/ focal-updates multiverse
  deb https://mirror.dkm.cz/ubuntu/ focal-backports main restricted universe multiverse
  deb https://mirror.dkm.cz/ubuntu/ focal-security main restricted
  deb https://mirror.dkm.cz/ubuntu/ focal-security universe
  deb https://mirror.dkm.cz/ubuntu/ focal-security multiverse

anomaly_detection_anomaly_detector_focal_libffi6_repo: https://mirror.dkm.cz/ubuntu
anomaly_detection_anomaly_detector_focal_libffi6_path: pool/main/libf/libffi/libffi6_3.2.1-8_amd64.deb

fluentbit_apt_repo: "deb https://packages.fluentbit.io/ubuntu/{{ ansible_distribution_release }} {{ ansible_distribution_release }} main"

postgresql_apt_repo: "deb https://apt.postgresql.org/pub/repos/apt focal-pgdg main"
postgresql_apt_repo_key: "https://www.postgresql.org/media/keys/ACCC4CF8.asc"

azure_apt_repo_baseurl: https://packages.microsoft.com/repos/azure-cli/
azure_repo_key_url: https://packages.microsoft.com/keys/microsoft.asc

adoptopenjdk_apt_repo: 'deb https://adoptopenjdk.jfrog.io/adoptopenjdk/deb focal main'
adoptopenjdk_apt_repo_key: 'https://adoptopenjdk.jfrog.io/adoptopenjdk/api/gpg/key/public'
adoptopenjdk_apt_repo_key_id: '8ED17AF5D7E675EB3EE3BCE98AC3B29174885C03'
```

8.7.2 RedHat/CentOS

```
system_custom_repos_content: |
  [rhui-rhel-8-for-x86_64-baseos-rhui-rpms]
  name=Red Hat Enterprise Linux 8 for x86_64 - BaseOS from RHUI (RPMs)
  baseurl=https://rhui-1.microsoft.com/pulp/repos/content/dist/rhel8/rhui/$releasever/x86_64/baseos/os
          https://rhui-2.microsoft.com/pulp/repos/content/dist/rhel8/rhui/$releasever/x86_64/baseos/os
          https://rhui-3.microsoft.com/pulp/repos/content/dist/rhel8/rhui/$releasever/x86_64/baseos/os
  enabled=1
  gpgcheck=1
  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
  sslverify=1
  sslclientcert=/etc/pki/rhui/product/content.crt
  sslclientkey=/etc/pki/rhui/private/key.pem

fluentbit_yum_repo: https://packages.fluentbit.io/centos/7/$basearch/

postgresql_yum_repo_url: "https://download.postgresql.org/pub/repos/yum/reporpms/EL-{{ ansible_distribution_major_version }}-{{ ansible_architecture }}/pgdg-redhat

azure_yum_repo_baseurl: https://packages.microsoft.com/yumrepos/azure-cli

__adoptopenjdk_packages:
  - adoptopenjdk-8-hotspot
  - adoptopenjdk-11-hotspot
  - adoptopenjdk-12-hotspot
  - adoptopenjdk-13-hotspot
  - adoptopenjdk-14-hotspot
  - adoptopenjdk-15-hotspot

adoptopenjdk_rpm_rhel8_repo: "https://adoptopenjdk.jfrog.io/artifactory/rpm/centos/8/x86_64/"
adoptopenjdk_rpm_rhel8_repo_key: "https://adoptopenjdk.jfrog.io/adoptopenjdk/api/gpg/key/public"
```

9. Certificates

The Gen2 platform requires a TLS configuration to protect traffic between users and application endpoints, and to protect traffic between Gen2 components.

9.1 Supported certificate types

You can issue self-signed certificates, certificates signed by your own CA, or LetsEncrypt certificates using Ansible, or use Ansible to install the provided certificates.

9.1.1 Provided certificates

The preferable way of installing certificates is to use customer-provided certificates. All other options should be used only for POC, testing purposes, or as a temporary solution. To use provided certificates, you should put them in the `ansible/files/certificates` folder, and then set the next variables in your inventory:

```
nginx_cert_kind: provided
nginx_certificates:
  - cn: one.customer.domain.tld
    kind: provided
    ca: "certificates/customerCA.crt"
    cert: "certificates/{{ nginx_domain }}-fullchain.crt"
    cert_key: "certificates/{{ nginx_domain }}.key"
```

You can use a custom folder with the certificates. In this case, you should put the full folder's path in `ca`, `cert` and `cert_key` variables. If the customer uses an external well-known CA to sign his certificates, you can omit `ca` variable. If the `nginx_certificates` list contains more than one certificate, you must specify `ca` variable only once, f.e., in the first element of the list. If you specify more than one `ca` variables, Ansible will show you an error. CA certificate will also be added to system truststore and to java keystore. See the information below for the details about how to use custom CA certificates in One Desktop and for One WebApp.

9.1.2 Own CA-signed certificates

There is an option to create your own CA certificate and issue certificates signed by this CA. The result is a usable Platform, except for end user workstations (web browsers etc.): you'll need to make the CA certificate trusted manually. To use this, you can set the next variables in your inventory:

```
nginx_cert_kind: ownca
nginx_certificates:
  - cn: one.customer.domain.tld
    kind: ownca
    altnames:
      - grafana.customer.domain.tld
      - kibana.customer.domain.tld
      - ...
```

CA issuing can be configured by defining the next variables:

```
ownca_common_name: one-ca
ownca_privatekey_passphrase: null
ownca_email_address: null
ownca_ca_not_after: "+3650d"
ownca_certs_not_after: "+3650d"
```

CA certificate will also be added to system truststore and to java keystore. This option should be used ONLY for testing purposes or for POC. See the information below for the details about how to use custom CA certificates in One Desktop and for One WebApp.

9.1.3 LetsEncrypt certificates

This option can be used ONLY for internal testing purposes and cannot be used on a customer's side. Only Microsoft Azure environments are supported. To enable LetsEncrypt certificates issuing, set the next variables in your inventory:

```
le_email: your_email
le_cloud_provider: azure
le_azure_subscription_id: your_azure_subscription_id
le_azure_resource_group: your_azure_resource_group
le_azure_dns_zone_name: your_azure_dns_zone_name
le_azure_key_vault_name: your_azure_key_vault_name

nginx_certificates:
  - cn: "*.your_nginx_domain"
    kind: letsencrypt
```

9.1.4 Self-signed certificates

Self-signed certificates can be used ONLY to check if Ansible installation works, f.e. on your local machine. This option isn't guaranteed to create a functioning Platform. The latest OpenSSL versions don't support self-signed certificates, so a system installed using them won't work. Generally, you should use the Own CA as described above for POCs, specialized installs etc. To use self-signed certificates, you can set the next variables in your inventory:

```
nginx_cert_kind: selfsigned
nginx_certificates:
  - cn: one.test.domain.tld
    kind: selfsigned
    altnames:
      - grafana.test.domain.tld
      - kibana.test.domain.tld
      - ...
```

9.2 Use a custom CA with One Desktop and One WebApp

You can't use One Desktop and One WebApp right after the installation that was performed using a custom (provided or own-issued) CA certificate. To use them, you should add this CA to the system truststore and to the One Desktop java keystore on the computer where One Desktop and One WebApp are supposed to be used.

9.2.1 Adding CA to a system truststore

Windows

- Right-click the CA certificate file, choose Install Certificate. The Certificate Import wizard appears.
- In the wizard, choose Next. Then, when you are prompted for the Certificate Store, choose Place all certificates in the following store. Select the **Trusted Root Certification Authorities** store.
- Complete the remaining steps of the wizard and click Finish.

Linux

- Copy the CA certificate to the ca-certificates folder. For Debian-based distros: `/usr/local/share/ca-certificates` For RedHat-based distros: `/etc/pki/ca-trust/source/anchors`

- Update certificates with this command: For Debian-based distros: `sudo /usr/sbin/update-ca-certificates` For RedHat-based distros: `sudo /bin/update-ca-trust`

- You must have latest version of ca-certificates package installed in Debian based distros.

MacOS

- Open the CA certificate with Keychain application. If the CA certificate has `.crt` or `.cer` extension, Keychain should open by double-clicking on the file.
- Choose System from the Keychain option and press OK.
- Choose Always Trust everywhere on the next window and click the Always Trust button.

- Then you will notice that the certificate is added to the system entry.

9.2.2 Adding CA to the One Desktop java keystore

Use the next command to add the CA certificate to the One Desktop java keystore:

```
/path-to-one-desktop-installed/jre/bin/keytool \
  -import \
  -keystore /path-to-one-desktop-installed/jre/lib/security/cacerts \
  -alias one-ca \
  -file path-to-your-ca-certificate \
  -storepass changeit \
  -noprompt
```

10. Glossary

Docs

this documentation

Controller

the machine where `ansible-playbook` is run

Target

any machine where the Platform will be installed, or more generally, any machine managed by Ansible. List of targets is a part of the inventory.

Inventory, Ansible inventory

a set of files describing where and how will the Platform be installed

venv:

Python virtual environment: a tool that allows installing Python modules (libraries) without affecting the main OS

pip

Python's package manager, used to install Python modules in the venv