THREDDS Clustering

Contents

1 Introduction

The THREDDS project is developing middleware to bridge the gap between data providers and data users. The goal is to simplify the discovery and use of scientific data and to allow scientific publications and educational materials to reference scientific data. Due to THREDDS's lack of horizontal scalability and automatic configuration management and deployment, this service usually deals with service downtimes and time consuming configuration tasks, mainly when an intensive use is done as is usual within the scientific community (e.g. climate).

This project aims to improve the scalability of the THREDDS Data Server through the implementation of a cluster of TDS instances that are deployed in the backend and that are only visible from the outside through a reverse proxy in the frontend. This project also aims to improve the management of the TDS catalogs by partitioning the hierarchy of catalogs into multiple TDS instances that are deployed in the backend, allowing high availability of the datasets and tackling the current problem of large waiting times after performing a THREDDS reboot when publishing new catalogs.

Instead of the classic installation and configuration of a single or multiple independent THREDDS servers, manually configured, this work presents an automatic provisioning, deployment and orchestration cluster of THREDDS servers.

1.1 Ansible - Automatic deployment and configuration

Ansible is a tool for automating configuration management and deployment and is used in TDS Clustering in order to automate the creation of the infrastructure.

This solution is based on Ansible playbooks, used to automate the deployment and management of the agents that conform the cluster. The playbooks are based on modules (or roles) of different backends and frontends load balancing setups and solutions. This implementation allows to configure different infrastructure and deployment setups, as more workers are easily added to the cluster by simply declaring them as Ansible variables and executing the playbooks.

1.2 HTTP Load balancing

The frontend load balancing system enables horizontal scalability by delegating requests to backend workers, consisting in a variable number of instances for the THREDDS server that are deployed inside Apache Tomcat containers. Through the clustering capacity of the Apache Tomcat server, in combination with the JK connector and the Apache Web Server, all HTTP features such as HTTP sessions, are available for the THREDDS cluster. This clustering also provides

fault-tolerance and better reliability since if any of the workers fail another instance of the cluster can take over it.

Additionally to the Apache httpd+mod_jk setup for the gateway, this project provides alternatives to perform the load balancing in the reverse proxy, such us HAproxy and nginx.

1.2.1 HTTP Load balancing caveats

In the context of HTTP load balancing, the load balancer becomes a bottleneck, since all the traffic goes through it. This is particularly problematic when large datasets are accessed by the clients, because the load balancer must perform heavy transfers of data at the kernel level, which involves a lot of overhead in data being copied from the network interface and the hard disk. This breaks the load balancer at the CPU level because of the excessive amount of work that it has to do.

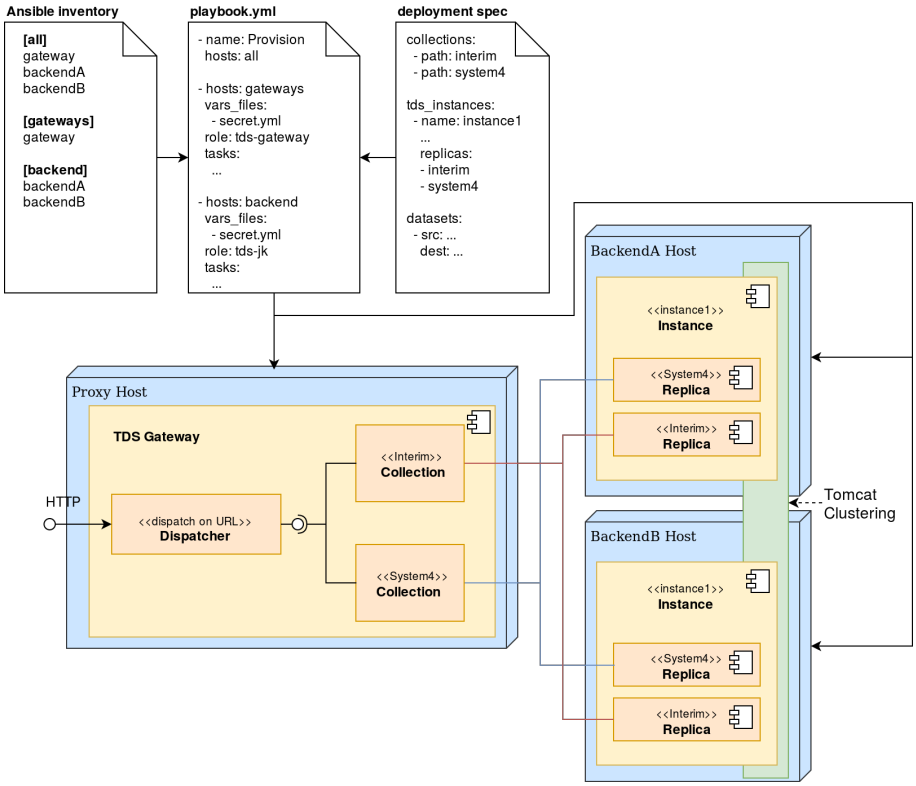
1.3 Direct Server Return

Direct Server Return is a type of load balancing in which the load balancer routes packets to the backends without changing anything in it but the destination MAC address. The backends process the requests and answer directly to the clients, without passing through the load-balancer.

This approach do away with the bottleneck created in the load balancer by the HTTP load balancing strategy, although it requires additional low level configuration and HTTP features are not available anymore.

2 Deployment model

The purpose of the deployment model is to define the common information needed by the different load balancing infrastructures. This information, defined using Ansible variables, will be used by the different roles and scenarios to perform their specific deployment of the THREDDS cluster. The following image provides a first overview of the deployment model.



In the figure we can identify the following agents:

- Reverse proxy or gateway
- Instances (THREDDS instances)
- Collections
- Replicas

All these elements are described in detail in the following sections.

2.1 The gateway

The gateway is the software that works as a reverse proxy and it is in charge of performing the load balancing, forwarding requests to the backend TDS instances deployed in Tomcat application servers. Multiple options are considered in this project to act as a gateway: httpd+mod_jk, HAproxy, Linux Virtual Server and others.

2.2 THREDDS instances

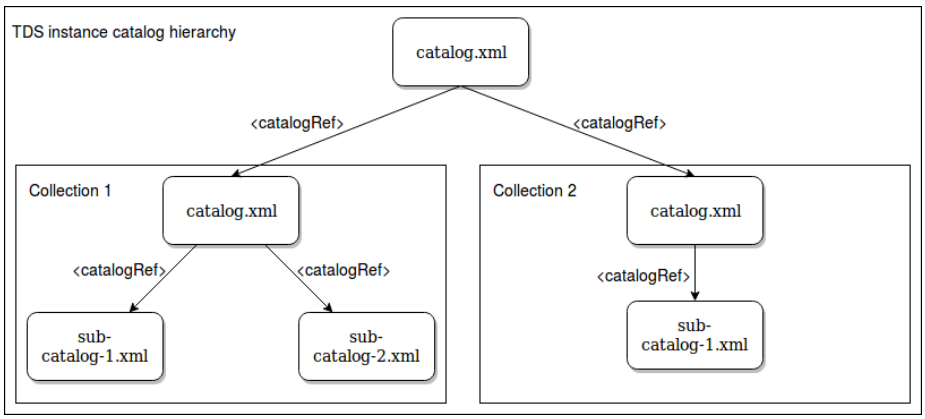
Each TDS instance, or simply instance, corresponds to a Tomcat server process running the THREDDS web application in the backend hosts. Each host can run one or more instances.

Each instance has the option to support any number of collections and each instance will respond only to requests targeting data available in the supported collections. This is done through the appropriate filtering of requests in the gateway.

Instances are clustered in the backend using Tomcat's clustering capabilities, which allows user session replication and in case that one of the instances become unavailable, another can take over it without any disruption of the service.

2.3 Collections and replicas

Collections are hierarchical structures compounded of a root THREDDS catalog and it's referenced catalogs. You build a collection by creating a catalog.xml file following the THREDDS Client Catalog Specification (see https://www.unidata.ucar.edu/software/thredds/current/tds/catalog/InvCatalogSpec.html#service).



The catalogs that form the collection will be copied into the TDS instances defined by the deployment model, inside the content/thredds directory. Also,

the root catalog of the collection will be referenced by the root catalog of the TDS instance. This allows the TDS instance to serve multiple collections by inserting multiple <catalogRef> elements into the TDS instance's root catalog.

2.4 How to define the deployment model

The deployment model is defined using the following variables:

- collections
- tds_instances
- datasets

The following is an example of the definition of these variables:

```
collections:
  - path: collection1 # Id of the collection and path in THREDDS
    top: True # Add a reference in the tds gateway to the collection
    catalogs:
      src: data/collection1
    services:
      - base: /thredds/catalog
      - base: /thredds/fileServer
      - base: /thredds/dodsC

  - path: collection2
    top: True
    catalogs:
      src: data/collection2
    services:
      - base: /thredds/dap4
      - base: /thredds/dodsC
      - base: /thredds/catalog
      - base: /thredds/fileServer
      - base: /thredds/ncss/grid

datasets:
  - src: data/datasets
    dest: '' # If empty, path is public/ in THREDDS
  - src: /home/user/tests/ncml-dataset
    dest: '/path/to/dest'

tds_instances:
  - name: instance1
    shutdown: 18000
    tds_debug: # optional, used to enable jpda and other debug features
```

```
  jpda_address: "{{ ansible_eth1.ipv4.address }}:8000"
connectors: # optional, connectors in conf/server.xml
  - protocol: HTTP/1.1
    port: 8080
replicas: # collections replicated by this tds instance
  - proxy: proxy # proxy's ansible inventory name
    host: "{{ ansible_eth1.ipv4.address }}"
    collections:
      - collection1
      - collection2
    properties: # <Connector> in conf/server.xml
      port: 18009
      protocol: AJP/1.3
```

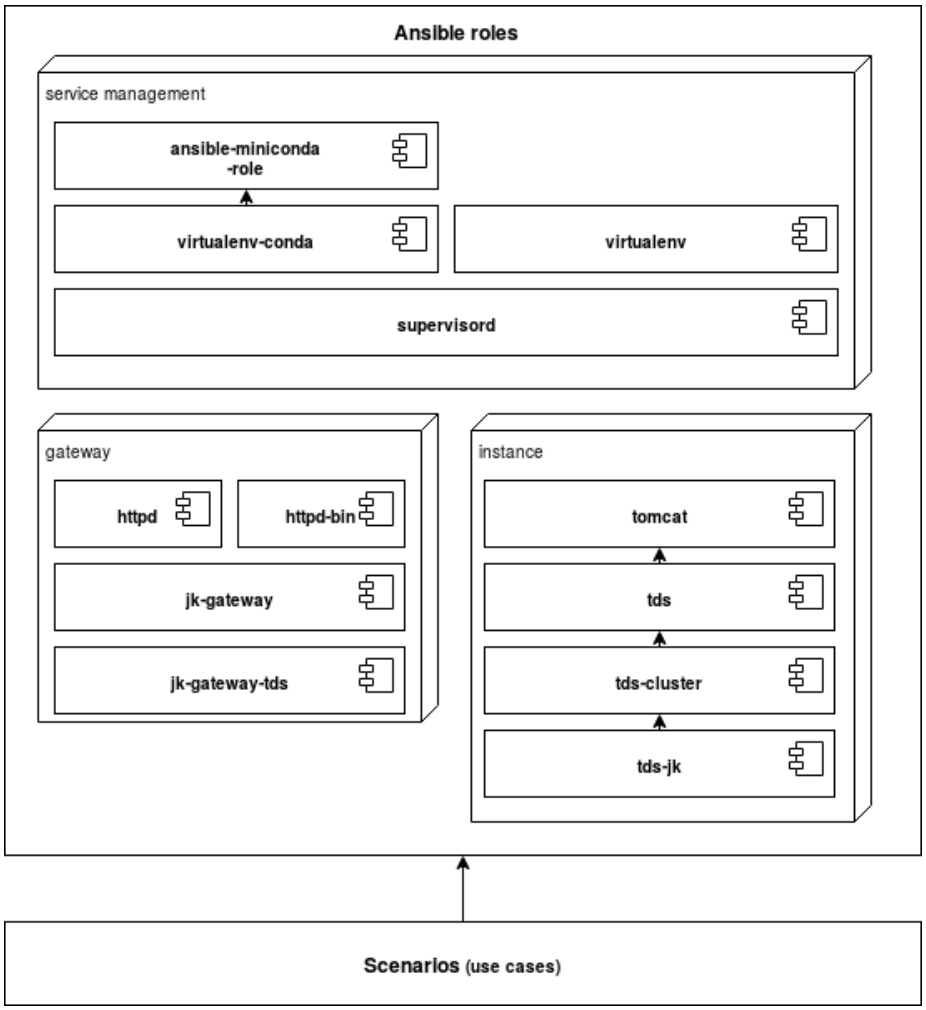
3 Roles and scenarios

3.1 Overview of roles

Ansible roles have been created to split the deployment of TDS Clustering into reusable parts. A user who wants to implement his specific TDS Clustering scenario has to define an Ansible inventory and a playbook, and also define the variables that the Ansible roles will use to do the deployment.

The defined Ansible roles are:

- ansible-miniconda-role - Role to install miniconda
- virtualenv - Role to install a python virtualenv
- virtualenv-conda - Role to install a virtualenv using miniconda
- supervisord - Role to install supervisord in top of a virtualenv
- httpd - Role to install httpd from source code
- httpd-bin - Role to install httpd from system packages
- jk-gateway - Role to deploy the mod_jk gateway in top of httpd
- jk-gateway-tds - Role to deploy collections and replicas in top of mod_jk
- tomcat - Role to deploy a tomcat server that will contain multiple instances
- tds - Role to deploy the THREDDS Data Server in tomcat instances
- tds-cluster - Role to deploy common cluster configuration
- tds-jk - Bridge between jk-gateway and tds



Roles are further explained in the following chapters.

3.2 Overview of scenarios

Scenarios are concrete use cases that make use of the available Ansible roles to do deployments with specific settings. Roles do the deployment of the minimum infrastructure for the THREDDS cluster to run and any customization must be defined in the scenarios.

Any scenario will be composed of:

- inventory - Ansible inventory file, which defines the hosts involved in the deployment.

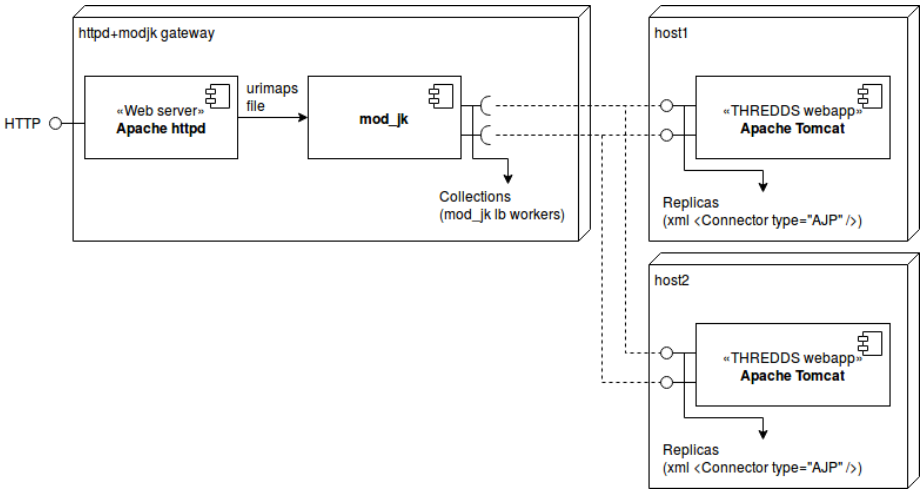
- main playbook - Ansible playbook that will contain two plays, one for the gateway hosts and other for the tds instances hosts, plus an optional third play that is in charge of the provisioning of the hosts.
- ansible.cfg - Ansible configuration file.
- user variables - Variables defined by the user that are required by the roles or that are defined to customize the deployment of the scenario.

4 Apache httpd+mod_jk gateway

When the gateway is built using the Apache httpd+mod_jk setup, it relies on the binary Apache JServ protocol to perform the load balancing. For each collection declared by the user through Ansible vars, a mod_jk worker of type lb is created in the gateway. This worker of type lb is populated with Tomcat workers based on the attribute replicas of the tds_instances Ansible var defined by the user.

In this context, collections are mod_jk workers of type load balancing compounded of replicas, that is, mod_jk balance workers. Each replica points to a tomcat instance deployed in the backend hosts that holds a copy of the contents served by the collection. Each request is routed, through the use of urimaps, to the TDS instances that support the collection in which the requested data resides.

Connectors of type AJP are used to connect the gateway with the instances. Usually, you only need one Connector element per gateway in the deployment, since Connector elements must reference their proxies. Each instance can contain one or more connectors (Connector xml elements in Tomcat's server.xml).



Additionally to the mod_jk workers described above, a mod_jk status worker is also created, which allows the user to check the state of the rest of the workers in a html view. See the role reference for more information.

4.1 Implementation example

The httpd+mod_jk implementation of THREDDS Clustering uses the information provided by the deployment model to create the appropriate mod_jk

workers and urimaps files, in addition to create the appropriate <Connector> elements in the tds instances (which will be of type AJP in this case).

We are going to see the workers and urimaps files created given the following deployment model:

```
collections:
  - path: collection1
    top: True
    catalogs:
      src: data/collection1
    services:
      - base: /thredds/catalog
      - base: /thredds/fileServer
      - base: /thredds/dodsC

datasets:
  - src: data/datasets
    dest: ''

tds_instances:
  - name: instance1
    shutdown: 18000
    replicas:
      - proxy: proxy
        host: server1
        collections:
          - collection1
        properties:
          port: 18009
          protocol: AJP/1.3
```

The mod_jk urimaps file will look like the following:

```
/thredds/catalog/collection1=COLLECTION_collection1
/thredds/catalog/collection1/*=COLLECTION_collection1
/thredds/fileServer/collection1=COLLECTION_collection1
/thredds/fileServer/collection1/*=COLLECTION_collection1
/thredds/dodsC/collection1=COLLECTION_collection1
/thredds/dodsC/collection1/*=COLLECTION_collection1
```

The mod_jk workers file will look like the following:

```
worker.server1_18009.type=ajp13
worker.server1_18009.host=server1
worker.server1_18009.port=18009

worker.list=COLLECTION_collection1
worker.COLLECTION_collection1.type=lb
```

```
worker.COLLECTION_collection1.mount=/thredds/restrictedAccess/*

worker.COLLECTION_collection1.balance_workers=server_18009
```

In this configuration, a collection named collection1 is created in the gateway and it is replicated in the backend host running a THREDDS instance. The THREDDS instance provides a Connector listening in the specified port where the gateway can connect in order to forward requests. The worker names for the THREDDS instances are compounded of the hostname of the host where they are running plus the port in which the AJP connector is listening, allowing multiple instances to listen in the same host.

5 Roles reference

5.1 Role virtualenv-conda

The purpose of this role is to deploy a python virtualenv in top of the miniconda installed by the role ansible-miniconda-role. The ansible-miniconda-role is installed in order to install libnetcdf for the tds instances to use it in their ncss service.

5.1.1 Role dependencies

ansible-miniconda-role

5.1.2 Role usage

This role enforces you to use the following variables in your playbook (variables required from the role `ansible-miniconda-role`):

- `miniconda_python` - Set it to `2` or `3`
- `miniconda_prefix` - Path where miniconda will be deployed, e.g. `"{{ ansible_env.HOME }}/miniconda2"`
- `miniconda_env.name` - Name of the virtualenv created inside miniconda
- `miniconda_env.dependencies` - Libs to install in the virtualenv

5.1.3 Variables

- `venv_home`: `"{{ miniconda_prefix }}/envs/{{ miniconda_env.name }}"`

5.1.4 Documentation

5.2 Role virtualenv

The purpose of this role is to deploy a python virtualenv.

5.2.1 Role dependencies

None.

5.2.2 Role usage

5.2.3 Variables

- `venv_home`: `"{{ ansible_env.HOME }}"` - Path where the virtualenv will be created.
- `venv_version`: `15.1.0`
- `venv_file_name`: `"virtualenv-{{ venv_version }}.tar.gz"`
- `venv_file_url`: `"https://pypi.python.org/packages/d4/0c/9840c08189e030873387a73b90ada981`
- `venv_sw`: `"{{ venv_home }}/sw"`
- `tmp_dir`: `"/tmp"`

5.2.4 Documentation

5.3 Role supervisord

This role install supervisord in the virtualenv created by one of the roles that installs virtualenv.

5.3.1 Role dependencies

None.

5.3.2 Role usage

If you want the supervisord to be proctected with password, you have to declare the following variables:

- `supervisord_user`: User name
- `supervisord_password`: User password

If they are not declared, supervisord is not protected.

5.3.3 Variables

- `supervisord_etc`: `"{{ venv_home }}/etc"`
- `supervisord_var`: `"{{ venv_home }}/var"`
- `supervisord_programs`: `"{{ supervisord_etc }}/supervisord.d"`
- `supervisord_port`: `9001`

5.3.4 Documentation

5.4 Role httpd

This role deploys an httpd server, downloading sources into the control machine and compiling it in every host. It is intented to allow the deployment by a non root user.

5.4.1 Role dependencies

None.

5.4.2 Exports

- `ansible_become`
- `httpd_document_root`

5.4.3 Variables

- `httpd_server_root`: `"{{ ansible_env.HOME }}/httpd"`
- `httpd_document_root`: `"{{ httpd_server_root }}/htdocs"`
- `httpd_version`: `2.4.25`
- `httpd_port`: `8000`
- `httpd_src`: `"{{ httpd_server_root }}/src"`
- `httpd_conf_path`: `"{{ httpd_server_root }}/conf"`
- `httpd_conf_file`: `"{{ httpd_conf_path }}/httpd.conf"`
- `httpd_version_major`: `"{{ httpd_versiontruncate(3, False, , 0) }}"`
- `httpd_mirror`: `https://archive.apache.org/dist/httpd/`
- `httpd_filename_unarchive`: `"httpd-{{ httpd_version }}"`
- `httpd_filename`: `"{{ httpd_filename_unarchive }}.tar.gz"`
- `httpd_download_url`: `"{{ httpd_mirror }}/{{ httpd_filename }}"`
- `httpd_to_be_removed`: `[build,build-1,icons,man,manual,src,include`
- `apr_version`: `"1.5.2"`
- `apr_filename_unarchive`: `"apr-{{ apr_version }}"`
- `apr_filename`: `"{{ apr_filename_unarchive }}.tar.gz"`

- apr_mirror: http://archive.apache.org/dist/apr
- apr_download_url: "{{ apr_mirror }}/{{ apr_filename }}"
- apr_install_base: "{{ httpd_server_root }}"
- apr_util_version: "1.5.4"
- apr_util_filename_unarchive: "apr-util-{{ apr_util_version }}"
- apr_util_filename: "{{ apr_util_filename_unarchive }}.tar.gz"
- apr_util_mirror: http://archive.apache.org/dist/apr
- apr_util_download_url: "{{ apr_util_mirror }}/{{ apr_util_filename }}"
- apr_util_install_base: "{{ httpd_server_root }}"
- pcre_version: "8.40"
- pcre_filename_unarchive: "pcre-{{ pcre_version }}"
- pcre_filename: "{{ pcre_filename_unarchive }}.tar.gz"
- pcre_mirror: http://ftp.cs.stanford.edu/pub/exim/pcre
- pcre_download_url: "{{ pcre_mirror }}/{{ pcre_filename }}"
- pcre_install_base: "{{ httpd_server_root }}"

5.4.4 Documentation

5.5 Role httpd-bin

This role deploys and httpd server using the system package manager. At the moment, it only works in yum based systems.

5.5.1 Role dependencies

None.

5.5.2 Exports

- ansible_become
- httpd_document_root

5.5.3 Variables

- `httpd_server_root`: `/etc/httpd`
- `httpd_conf_file`: `/etc/httpd/conf/httpd.conf`
- `httpd_document_root`: `/var/www/html`
- `httpd_port`: `80`
- `httpd_dependencies`: `["@Development tools", "httpd-devel", "apr", "apr-devel", "apr-util"`

5.5.4 Documentation

5.6 Role jk-gateway

Here follows the documentation of the role jk-gateway. This role deploys mod_jk in order to act as a reverse proxy. It can be used with an existing httpd installation or after executing the httpd role.

5.6.1 Role dependencies

Although this role does not define any dependencies in the `meta/` directory, it is supposed to be used on top of one of the `httpd` roles. It can also be used in top of an existing httpd installation that was previously carried out without any of the roles provided by this project, as it is the case of the ESGF scenario.

5.6.2 Exports

- `httpd_document_root`
- `mod_jk_conf_path`

5.6.3 Role usage

If you want to deploy the status worker, you have to declare the following variables:

- `mod_jk_status_user`: User to login
- `mod_jk_status_passwd`: Password to login

If they are not declared, the `.htpasswd` file nor the status worker are created.

5.6.4 Variables

These variables control the deployment of the reverse proxy based on the underlying httpd installation. Default values assume that httpd has been installed using one of the provided httpd roles or that httpd has been installed from system packages.

- `mod_jk_version`: `"1.2.43"`
- `mod_jk_filename_unarchive`: `"tomcat-connectors-{{ mod_jk_version }}-src"`
- `mod_jk_filename`: `"{{ mod_jk_filename_unarchive }}.tar.gz"`
- `mod_jk_mirror`: `http://archive.apache.org/dist/tomcat/tomcat-connectors/jk`
- `mod_jk_download_url`: `"{{ mod_jk_mirror }}/{{ mod_jk_filename }}"`
- `mod_jk_src`: `"{{ ansible_env.HOME }}/mod_jk_src"`
- `mod_jk_conf_path`: `"{{ httpd_server_root  default(/etc/httpd) }}/conf.d"`
- `mod_jk_shm`: `"{{ httpd_server_root  default(/etc/httpd) }}/logs/mod_jk.shm"`
- `mod_jk_log`: `"{{ httpd_server_root  default(/etc/httpd) }}/logs/mod_jk.log"`
- `mod_jk_static_catalog`: `True`
- `mod_jk_status_path`: `"{{ mod_jk_conf_path }}"`

The `mod_jk_static_catalog` variable is used to indicate if the role should create static html files imitating THREDDS html catalog files in the reverse proxy.

The `mod_jk_status_passwd` variable sets the path of the .htpasswd file that contains the user and password for the login in the jk status worker.

5.6.5 Documentation

5.7 Role jk-gateway-tds

This role further configures the mod_jk deployed by the role jk-gateway by registering the appropriate mod_jk workers and urimaps using the information defined in the collections and tds_instances variables. It also creates the static resources that will be served by the gateway instead of the backend TDS instances.

5.7.1 Role dependencies

- jk-gateway

5.7.2 Role usage

This role requires you to define the following variables, except that they are defined by other roles:

- httpd_document_root - Used in tasks and in templates
- collections - From the deployment model. Used in tasks

5.7.3 Variables

None.

5.7.4 Documentation

Tasks in this role are separated from the jk-gateway in order to allow other applications, apart from TDS, to be reverse proxied, i.e. TAP.

5.8 Role tomcat

This role performs the deployment of a tomcat server by downloading files from the Internet and following a configuration that does not require the user to have roor privileges.

5.8.1 Role dependencies

The role does not include any dependency in it's `meta/` directory but it is supposed to be used on top of the `supervisord-conda` role.

5.8.2 Variables

- `jre_version`: `8u141`
- `jre_filename_unarchive`: `"jre-{{ jre_version }}-linux-x64"`
- `jre_filename`: `"{{ jre_filename_unarchive }}.tar.gz"`
- `jre_mirror`: `http://download.oracle.com/otn-pub/java/jdk/`
- `jre_download_url`: `"http://download.oracle.com/otn-pub/java/jdk/8u172-b11/a58eab1ec24242`
- `jre_header`: `"Cookie: oraclelicense=accept-securebackup-cookie"`
- `jre_install_base`: `"{{ tomcat_home }}/jre"`
- `tomcat_home`: `"{{ ansible_env.HOME }}/tomcat-home"`

- tomcat_base: "{{ ansible_env.HOME }}/tomcat-base"
- tomcat_version: 8.0.42
- tomcat_version_major: "{{ tomcat_versiontruncate(1, True, , 0) }}"
- tomcat_filename_unarchive: "apache-tomcat-{{ tomcat_version }}"
- tomcat_filename: "{{ tomcat_filename_unarchive }}.tar.gz"
- tomcat_mirror: http://archive.apache.org/dist/tomcat
- tomcat_download_url: "{{ tomcat_mirror }}/tomcat-{{ tomcat_version_major }}/v{{ tomcat_v
- tomcat_to_be_removed: "[KEYS,LICENSE,NOTICE,RELEASE-NOTES,RELEASE-NOTES.html,RUNNING.txt

5.8.3 Documentation

5.9 Role tds

This role deploys a tds instance inside the tomcat server deployed by the 'tomcat' role.

5.9.1 Role dependencies

- tomcat

5.9.2 Role usage

This role requires that you define the following variables, from the deployment model, in your playbook: tds_instances, collections, datasets.

5.9.3 Variables

- tds_version: ESGF-5.0.1
- tds_filename_unarchive: "tds-{{ tds_version }}"
- tds_filename: "{{ tds_filename_unarchive }}.war"
- tds_mirror: http://artifacts.unidata.ucar.edu/content/repositories/unidata-releases/edu/
- tds_download_url: "{{ tds_mirror }}/{{ tds_version }}/tds-{{ tds_version }}.war"
- tds_debug: False

5.9.4 Documentation

TDS is downloaded from Unidata Nexus repository.

5.9.4.1 TdsClusterAuthorizer

TdsClusterAuthorizer is a custom Authorizer that always perform authentication in the THREDDS server using https, even when the initial request for a dataset is made in plane http.

5.10 Role tds-jk

This role is the glue between the role `jk-gateway` and the role `tds`. For every `tds_instance` defined for the `tds` role, it configures the `mod_jk` reverse proxy or gateway to redirect requests to the appropriate `tds_instance`.

5.10.1 Role dependencies

- tds

5.10.2 Role usage

See documentation for the role `tds`.

This role requires the following variables to be declared by the user when roles defining them are not used in the play.

- ansible_become
- httpd_document_root
- mod_jk_conf_path

5.10.3 Variables

5.10.4 Documentation

`mod_jk` sticky sessions are used.

6 Scenarios reference

6.1 Scenario devel

This scenario serves as use case for development and testing of the TDS+`mod_jk` gateway infrastructure. It also serves as an example of different use cases that can be implemented.

6.1.1 Requirements

1. vagrant
2. ansible 2.5

6.1.2 Documentation

The main files in this directory are "source.yml" and "binary.yml". These playbooks deploy a httpd+mod_jk reverse proxy in the ansible host "proxy", that forward requests to a backend consisting on two hosts, "hostA" and "hostB", each running a tds instance. Configuration of variables can be found in the group_vars directory (see Ansible docs).

The inventory file, named inventory, defines the three hosts involved in this scenario. The "test-simple" group is only used for quick testing avoiding the deployment of hostB.

The data directory contains sample THREDDS catalog files and NetCDF datasets.

6.1.3 Usage

1. vagrant up && vagrant reload
2. ansible-playbook (source.ymlbinary.yml) –ask-vault-pass [–limit test-simple]
3. (in web browser) localhost:9000/thredds, localhost:9000/status-jk

6.1.4 Infrastructure description

- CentOS 7 vagrant machines
- httpd 2.4
- tomcat 8

- tds 4.6

6.1.5 Networking

Virtual machines are configured to use an internal network (see Vagrantfile) and each virtual machine has the following configuration:

- proxy - eth1, ip: 192.168.50.10
- hostA - eth1, ip: 192.168.50.11
- hostB - eth1, ip: 192.168.50.12

The following ports are forwarded from the host to the guests:

- host 9000 -> proxy 8080
- host 9001 -> hostA 8080
- host 9002 -> hostB 8080

6.1.6 JPDA debug

Tomcat instances are configured to be debugged used JPDA. JPDA listens in the port 8000 in both hostA and hostB.

6.2 Scenario tds_standalone

This scenario deploys a sandbox, using the framework provided by TDS Collections, that contains:

- A THREDDS instance available at http://localhost:8080/thredds by default
- content/thredds is available in the root directory where you deployed the sandbox
- toolsUI jar
- ./toolsUI inits the toolsUI interface in the background
- ./ncjdump is a shorcut to `java -cp toolsUI.jar ucar.nc2.NCdumpW "$@"`

The THREDDS instance is populated with the following data:

- THREDDS catalogs contained in `data/catalogs`
- Datasets contained in `data/datasets`
- A <dataset> entry is created in the main catalog for each file found in data/datasets, although this is not recursive

6.2.1 Requirements

1. ssh key pair in your ~/.ssh directory and authorized to ssh to localhost
2. ansible 2.5

6.2.2 Usage in localhost

1. `git clone --recursive -b devel https://github.com/SantanderMetGroup/ansible-thredds-clu`
2. `cd ansible-thredds-cluster/scenarios/zequi/tds_standalone`
3. `./run.sh -h`
4. Deploy the TDS instance `./run.sh -r /tmp/sandbox deploy`
5. (in web browser) `localhost:8080/thredds`
6. After adding new content to the `data` directory and perform only the update of the catalogs: `./run.sh -r /tmp/sandbox -u deploy`
7. Stop all the processes: `./run.sh -r /tmp/sandbox stop`
8. Start all the processes: `./run.sh -r /tmp/sandbox boot`

6.2.3 Usage in vagrant

Vagrantfile is provided if you want to test the deployment without polluting your system. The default ip for the virtual machines are `192.168.50.10` and `192.168.50.11`.

1. `vagrant up`
2. `vagrant ssh ubuntu` or `vagrant ssh centos`
3. `./test.sh`

6.2.4 Scenario's variables

- root: Default is "/tmp/sandbox"
- catalogs_path: Default is "data/catalogs"
- datasets_path: Default is "data/datasets"

6.3 Scenario spock

This scenario deploys the THREDDS Clustering in top of the test ESGF node deployed in spock. It serves as a use case for testing the THREDDS Clustering together with ESGF. Additionally, this scenario also creates mod_jk workers that allow access to the /tds5 and /udg-tap through spock.meteo.unican.es.

6.3.1 Usage

1. `svn co PATH_TO_TDSSPOCK_SVN_REPO .`
2. `ansible-playbook main.yml --ask-vault-pass`
3. `ansible-playbook main.yml --ask-vault-pass --tags update_catalogs`

6.3.2 Doc

ssh -L 4119:wn019:18008 -L 4120:wn020:18008 -L 4019:wn019:8000 -L 4020:wn020:8000 ui.macc.unican.es