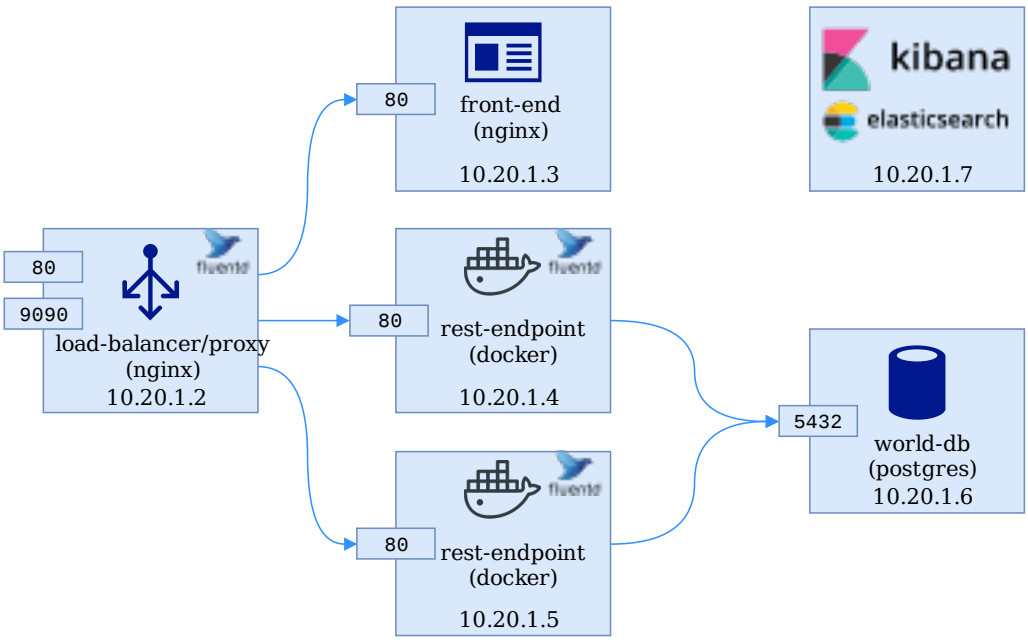
Lab 3

Playbook

During this lab you are going to provision and deploy a Postgres database using a playbook. It is part of a much larger deployment to browser countries. The full architecture is depicted below:



During this lab you'll provide the **world-db** database.

The first thing you want to do is to ensure your target machine is reset to it's initial state in case you have changed it state during demos or previous labs.

Open a terminal inside `~/course/ansible/machines/` and run the following command to restore `machine-6`

```
$ vagrant snapshot restore machine-6 initial
```

You'll be creating your playbook inside the `~/course/ansible/worlddb/` directory. The following files have already been provided to you:

- An `ansible.cfg` configuring the host file and the ssh user.
- An empty `worlddb.yaml` for you to complete

During this and subsequent labs, you'll be using the hosts file from the `~/course/ansible/setup/hosts` file. This file contains the hosts for the entire solution you will be building.

Whenever you want help on a module, you could go online or use `ansible-doc` . For example to get help on the `file` module, use:

```
$ ansible-doc  file
```

Run the playbook whenever you want. Also remember you can run it from a specific point:

```
$ ansible-playbook  worlddb.yaml --start-at-task="…"
```

If you want to see what a module has done, you can always use the "one-step-at-a-time" (`--step`) and then check with a ssh session if things are working as expected (example below lists all users if you wanted to check if a particular user was created)

```
$  ssh vagrant@10.20.1.6
#  sudo cut -d: -f1 /etc/passwd
```

Make sure you play with these kinds of techniques during this lab.

Let's start with some initial steps. During this task, you will eventually:

- Setup the basic structure of the playbook
- Declare play variables that you'll use later

Open the `worlddb.yaml` . The file is still empty and will have to be completed by you during this lab.

Let's start with some basic configuration:

- make sure this play accesses the hosts in the `db-servers` group `- hosts: db-servers`
- by default the user privilege should be elevated to the root user `become: true`
- Set the following variables `vars:`

variable	value
`postgres_dir`	`/var/lib/postgresql`
`pgdata_dir`	`pgdata` subdirectory of `postgres_dir`

The second variable will be our location for the postgres data (which differs from the default, therefore this requires configuration later in the lab.)

You are now ready to add your first task. You'll need to ensure the user `postgres` exists. This user will need to be in the `postgres` group. You'll also define a password for the user (hashed).

We will set the shell of this user to `/usr/bin/psql` . This means when we do `su postgres` it will eventually open the `psql` client tool. The home directory of the user will be set to `/var/lib/postgresql/` (for which you already defined a playbook variable)

- Ensure the postgesql group exists
  - use the correct module (you might want to check the system modules (https://docs.ansible.com/ansible/latest/modules/list_of_system_modules.html)) `group`
  - the name of the group is `postgres` `name: postgres`
- Ensure postgres user is available using the correct module `user` :

property	value
name	`postgres`
group	`postgres`
home	use `postgres_dir` variable

property	value
shell	`/usr/bin/psql`
password	see further below

- Ensure the home directory exists:
  - Use the correct module `file`
  - The name of the directory is available in the `postgres_dir` variable `path: "{{postgres_dir}}"`
  - Make sure the directory exists `state: directory`
  - Set the owner and group to be `postgres` `owner:…` and `group: …`

The password needs to be hashed. We will use `masterkey` as the password value. To hash it use `mkpasswd --method=sha-512` . If this is not installed use the following python 3 script:

```
$ pip install passlib --user
$ python3 -c 'import crypt,getpass; print(crypt.crypt(getpass.getpass(), crypt.
mksalt(crypt.METHOD_SHA512)))'
```

Before you run the play so far in your playbook, perform a syntax check first `--syntax-check` and then perform a dry-run check `--check` or `-C`

If you want run your playbook (however we are not done yet)

You will need to install the following packages and then configure the service

- `postgresql-server`
- `postgresql-contrib`

Installation

Use a single task to install the two packages:

- Do you remember the correct module `package` with `name` and a list
- After that ensure the postgresql service is *enabled*
  - use the correct module `service`
  - and ensure `postgresql` `name:…` is enabled `enabled: true`

Service Configuration

The service needs to know the location of our custom data directory. This is achieved through the environment variable `PGDATA` . On a systemd-enabled system this is achieved by adding service configuration. In our case we need to make a file named `/etc/systemd/system/postgresql.service.d/pgdata.conf` with the following contents:

```
[Service]
Environment=PGDATA=/var/lib/postgresql/pgdata
```

In order to achieve this, create a task:

- Which module could you use for this `ini_file`
- The name of the file to create/update is `/etc/systemd/system/postgresql.service.d/pgdata.conf` `path`
- set the above value using the (make sure you use your `postgres_dir` variable) `section` , `option` and `value`
- also ensure this file is created in case in does not exist yet `create: yes` (which is also default)

Reload the configuration

After this we need to reload the systemd daemon (this is important for the next steps to work):

- use the appropriate module for this `systemd`
- the name of the service is postgresql `name: …`
- ensure the daemon is reloaded `daemon_reload: true`

With postgres installed we can now configure the database. Eventually during this step you will:

- Initialise a database inside our custom location
- Make postgres accessible from outside (by default it only opens a port on localhost)
- Allows password-based authentication for database users
- Start the postgres service

Initialise the database

The database needs to be initialised. This is achieved using the postgres utility `postgresql-setup` with an option of `initdb`. It will be your task to invoke this and make sure if does not get invoked if initialisation has already been performed.

- You will need to run the shell command `postgresql-setup initdb`. Use the correct module `command` with its free form
- This command needs to be ran as the `postgres` user `become` and `become_user`
- The result of this is a file named `/var/lib/postgresql/pgdata/postgresql.conf`. How can you ensure it won't run when it already ran before `creates:…` (please mind how you pass this property `args:`)

Listen to all ports

By default Postgres only listens on localhost. Later in the course, we will need to access this from another computer. We therefore need to configure Postgres to listen to all ip addresses. This is accomplished by changing the line `#listen_addresses = 'localhost'` to `listen_addresses = '*'` inside `/var/lib/postgresql/pgdata/postgresql.conf`:

- What module should you use to ensure a particular line is in a file `lineinfile`
- Specify the name of the file (use your variable) `path: …`
- look for the correct line (a regular expression is `^#?listen_addresses`) `regexp: …`
- Replace this line with `listen_addresses = '*'` `line: …`

Allow password login

By default Postgres does not allow password authentication. To enable this, we need to make sure the following line is inside the `/var/lib/postgresql/pgdata/pg_hba.conf`:

```
host    all             all             0.0.0.0/0               md5
```

- Use the appropriate module `lineinfile` again
- Ensure the above line is in the file (no need for `regexp` here)

Start Postgres

Finally make sure the `postgresql` service is started (or restarted in case it was already running).

- choose your module `service`
- The name of the service is `postgresql` `name: …`
- make sure it is (re)started `state: restarted`

During this task you will:

- Create a new database named `worlddb`
- Populate the database with a tables and data
- Create a database user (`student`) to connect to this database

You will perform these steps using postgresql modules (https://docs.ansible.com/ansible/latest/modules/list_of_database_modules.html#postgresql-database-modules)

Create the database

Use the postgresql_db (https://docs.ansible.com/ansible/latest/modules/postgresql_db_module.html#postgresql-db-module) module:

- run this as the `postgres` user
- the name of the database is `worlddb` `name: …`
- make sure the database is created if not there `state: present`

Populate the database

We have provided you with a sql script `files/structure.sql` . You will first need to copy this file over and then run it against our newly created database.

- Use the correct module to copy the file to the target system(s) `copy`
- run this as the `postgres` user so that permissions are set correctly on the file
- The source is `files/structure.sql` `src`
- For the destination use the `/tmp` folder for now `dest`

After this, use the same postgresql_db (https://docs.ansible.com/ansible/latest/modules/postgresql_db_module.html#postgresql-db-module) as before:

- run this as the `postgres` user
- the name of the database is `worlddb`
- ensure a sql script will be ran `state: restore`
- the sql file is named `files/structure.sql` `target: …`

Create our database user

As a final step, create a user named `student` with password `masterkey` . This user will need all permissions to the tables `city` , `country` and `countrylanguage` .

Use the postgresql_user (https://docs.ansible.com/ansible/latest/modules/postgresql_user_module.html#postgresql-user-module) module:

- run this as the `postgres` user
- the name of the database is `worlddb` `db: …`
- the name of the user is `student` `name: …`
- set the password in clear text to `masterkey` `password`
- Set the permissions to `"CONNECT/city:ALL/country:ALL/countrylanguage:ALL"` `priv: …`

You probably already ran the playbook a couple of times, but it is time for a final run. So please go ahead and run your finalised playbook (perform a syntax check and dry-run if you want)

After ansible has completed the playbook, check if you are able to connect to the database. We will use a postgres docker container to run the `psql` client from another machine (in this case your host)

```
$ docker run --rm -it  postgres:9-alpine  psql -U student -h 10.20.1.6 worlddb
Password for user student:
psql (9.6.10, server 9.2.24)
Type "help" for help.

worlddb=#
```

Yes! you are able to connect, so it seems your configuration is correct. Let's check if you have the correct privileges

To list all the public tables run the `\dt` operation:

```
worlddb=# \dt
```

You should see three geographic tables (`city` , `country` and `countryLanguage`).

Make sure you can query for example the `country` table:

```
worlddb=# select code, name from country;
```

You should see some 240 countries.

Exit the psql client:

```
worlddb=# \q
```