Lab 11

AWX

During this lab you are goint to setup the playbook you've been working on in AWX. You'll use a local gitlab as your SCM. To make it as real as possible, let's do some house cleaning and make sure all configuration on the machine and your host are removed:

Reset all your virtual machines

Open a terminal to `~/course/ansible/machines` and run

```
$ vagrant snapshot restore initial
```

Also remove the machine's certificates from your trusted list:

```
$ for i in {2..7}; do ssh-keygen -R 10.20.1.$i; done
```

And remove your public SSH key from the servers using the `unauthorise.yaml` playbook inside `~/course/ansible/setup`

```
$ ansible-playbook unauthorise.yaml
```

Time to install AWX. There are various installation options and these differ from Ansible Tower. What AWX and Ansible Tower installations have in common is that they both use ansible for the actual installation.

The options we have available are OpenShift, Kubernetes and Docker/Docker Compose. We will choose the latter option.

Start by cloning the awx git repository

```
$ cd ~/Downloads/
$ git clone https://github.com/ansible/awx/ \
      --depth=1 --branch 1.0.8 \
      && rm -fr awx/.git \
      && cd awx
```

Then enter the `installation` directory

```
$ cd installer
```

Review the `inventory`. Notice that by default it will apply the setup playbook to the localhost. We recommend this as well. If you want, you can spin up another machine by increasing the number of machines in `~/course/ansible/machines`. However we recommend running it on your host as you will need at least 2GB of memory on the target machine.

Within the `inventory` change the following variables:

- Set the postgres data directory to `/home/student/course/ansible/awx/pgdata`
- Set the host port to 8888: `host_port=8888`
- Indicate we want to use docker-compose, uncomment and set to true: `use_docker_compose=true`
- change the value for the docker compose directory to `docker_compose_dir=/home/student/course/ansible/awx/`

Then run the installer

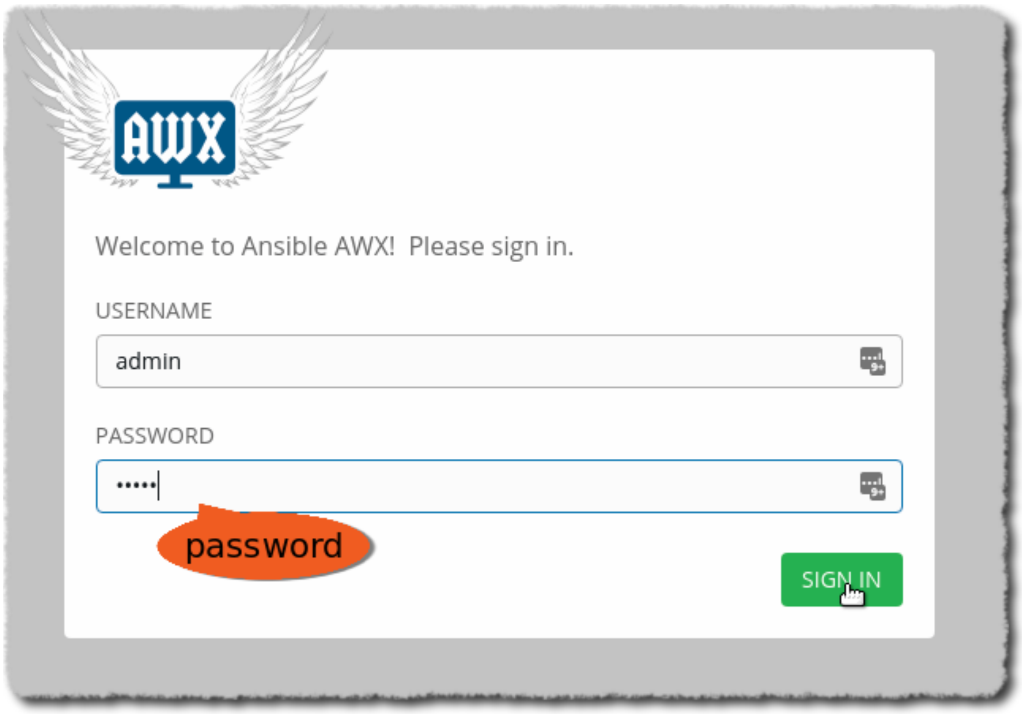
```
$ ansible-playbook -i inventory install.yml
```

This might take a while, so you might want to grab a coffee or a tea.

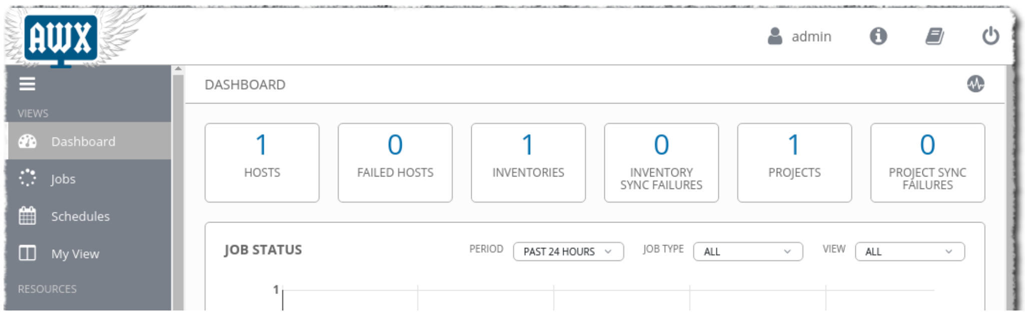
After the installer has completed you should have:

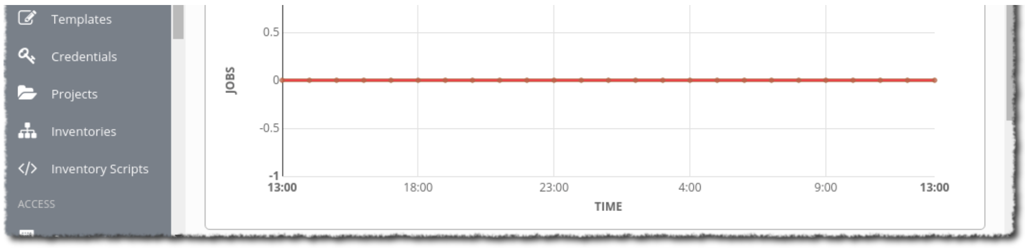
- Five docker containers running (`docker ps`):
  - `ansible/awx_web`
  - `ansible/awx_task`
  - `postgres`
  - `ansible/awx_rabbitmq`
  - `memcached`
- you should also have a `docker-compose.yml` file at `~/course/ansible/awx/`

Open your browser at http://localhost:8888 (http://localhost:8888). You will be presented with a login screen. The default username and password are *admin*/*password*:



After that you will see the *Dashboard*





You will be using gitlab for your ansible playbook project. You must add this as a docker service to the generated `docker-compose.yml` .

Open the docker-compose file and add the following to the end of `/home/student/course/ansible/awx/docker-compose.yml` (you'll need to indent it as shown below, as it is parts of the compose `services` in this yaml file)

```
  gitlab:
    image: gitlab/gitlab-ce:11.2.1-ce.0
    hostname: gitlab
    ports:
    - "8080:80"
    - "8022:22"
    volumes:
    - ${HOME}/course/gitlab/config:/etc/gitlab
    - ${HOME}/course/gitlab/logs:/var/log/gitlab
    - ${HOME}/course/gitlab/data:/var/opt/gitlab
    environment:
    - GITLAB_PORT=8080
    - GITLAB_SSH_PORT=8022
```

And then start gitlab (the other services have already been started by the AWX installation process)

```
$ docker-compose up -d gitlab
```

After some time you should be able to go to http://localhost:8080/ (http://localhost:8080/). (this might take some considerable time, if in doubt check the logs `docker-compose logs -f gitlab`).

Once the **GitLab Community Edition** page appears, change the password of the root user to **masterkey** and then login with **root** and your password

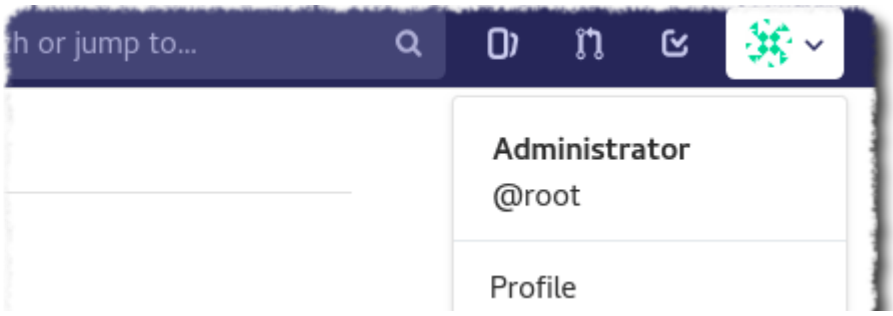
Provide key to gitlab

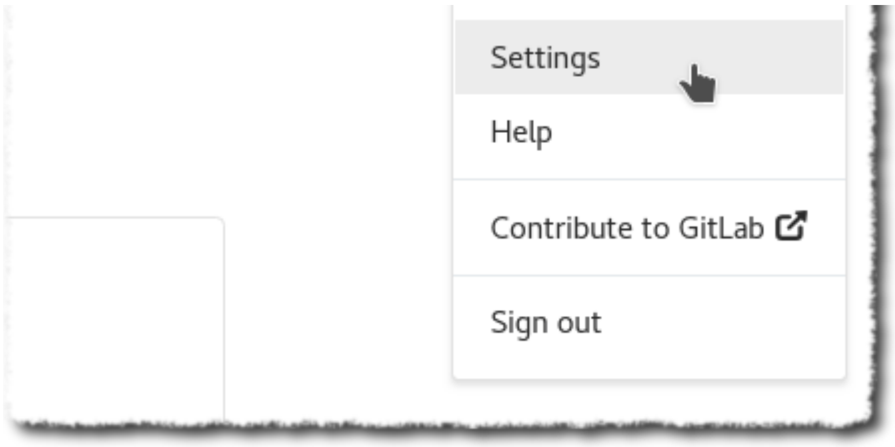
In order to push updates to gitlab you will need to register your public SSH key.

Copy your public key to your clipboard (e.g, by piping it to `xclip`)

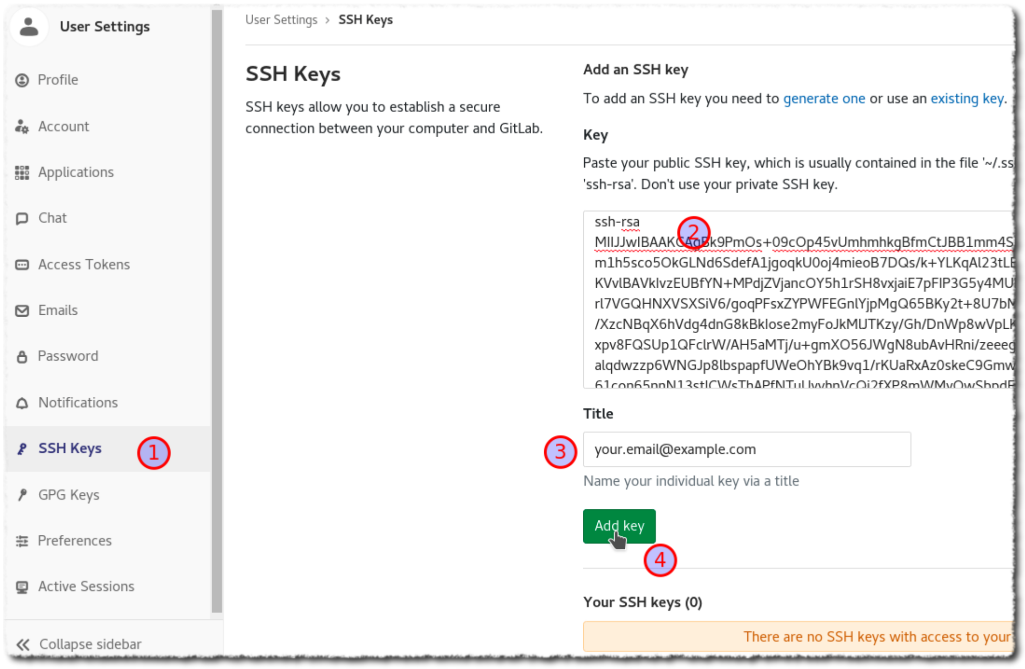
```
$ cat ~/.ssh/id_rsa.pub | xclip -selection c
```

Then inside github go to settings





On the sidebar menu choose **SSH Keys**, Then paste your key in the text area and click on **Add key**



SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to generate one or use an existing key.

Key

Paste your public SSH key, which is usually contained in the file '~/.ss
'ssh-rsa'. Don't use your private SSH key.

```
ssh-rsa
MIIJJwIBAAKCAgBk9PmOs+09cOp45vUmhmhkgBfmCtJBB1mm4S
m1h5sco5OkGLNd6SdefA1jgoqkU0oj4mieoB7DQs/k+YLKqAl23tLE
KVvlBAVkIvzEUBfYN+MPdjZVjancOY5h1rSH8vxjaiE7pFIP3G5y4MU
rl7VGQHNXVSXSiV6/goqPFsxZYPWFEGnlYjpMgQ65BKy2t+8U7bN
/XzcNBqX6hVdg4dnG8kBklose2myFoJkMUTKzy/Gh/DnWp8wVpLK
xpv8FQSUp1QFclrW/AH5aMTj/u+gmXO56JWgN8ubAvHRni/zeeeg
alqdwzzp6WNGJp8lbspapfUWeOhYBk9vq1/rKUaRxAz0skeC9Gmw
61con65npN13stlCWsThAPfNTuUvvhnVcQi2fXP8mWMvQwSbpdF
```

Title

your.email@example.com

Name your individual key via a title

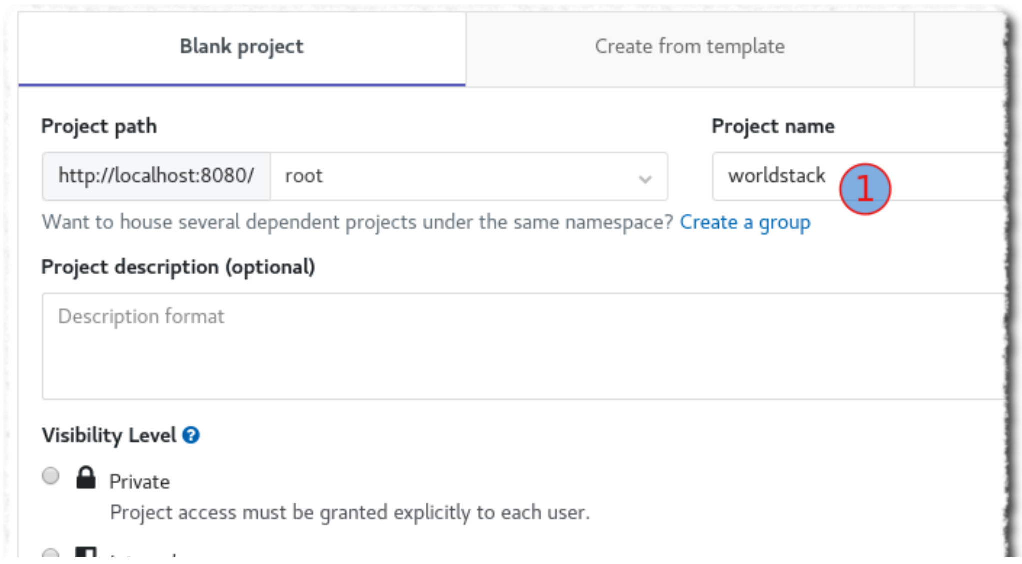
Add key

Your SSH keys (0)

There are no SSH keys with access to your

Define a new project

Create a new gitlab project. Name it **worldstack** and make it **public** (do *not* initialise the project with README):



Project path

http://localhost:8080/ root

Project name

worldstack

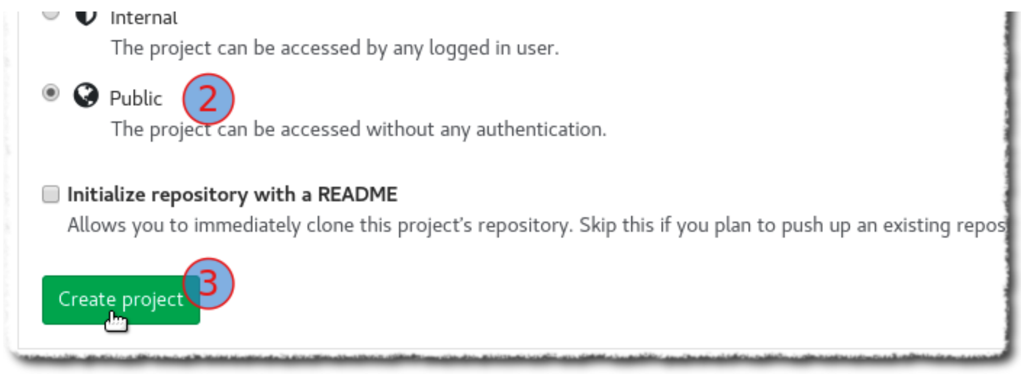
Want to house several dependent projects under the same namespace? Create a group

Project description (optional)

Description format

Visibility Level

Private
Project access must be granted explicitly to each user.


Internal
The project can be accessed by any logged in user.
Public
The project can be accessed without any authentication.
Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repos

Configure your SSH Client

You will soon push a project to gitlab. But before you can do so you'll have to configure your ssh client a little further.

Our gitlab's ssh port is not `22` but `8022`. You can't set this port number when using git. We can however set it as the default SSH port for our gitlab server.

Perform the following steps:

- Add a more meaningful hostname for your localhost, to be used when interacting with gitlab:

  ```
  $ echo "127.0.0.1 gitlab.example.com" | sudo tee --append /etc/hosts
  ```

- Then set the default port for this server to '8022'

  ```
  $ echo "Host gitlab.example.com" >> ~/.ssh/config
  $ echo "Port 8022" >> ~/.ssh/config
  ```

- Then ensure you set the correct permissions on this file:

  ```
  $ sudo chmod 600  ~/.ssh/config
  ```

You are now all set to push a project to your server

For git you need to specify your identity (this is used in commit messages)

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Navigate to the `~/course/ansible/worldstack` directory. This contains the whole stack without the EFK logging stack and the use or roles (If you did want to use roles you would have to add a `roles/requirements.yml` with a list of required roles. AWX would then download those roles to the local checkout of the project)

Now let's make the current `worldstack` directory a git project, stage the project files and commit your first version:

```
$ git init
$ git add .
$ git commit -m "initial commit"
```

With our first commit behind us, we want to push it to the gitlab server. First add the address to the remote git repository and then push your master branch to it:

```
$ git remote add  origin git@gitlab.example.com:root/worldstack.git
$ git push -u origin master
```

You will be asked if you trust the server's certificate, answer this with "yes". Note if you were to restart your gitlab server, you would have to remove this key using `ssh-keygen -R [gitlab.example.com]:8022`)

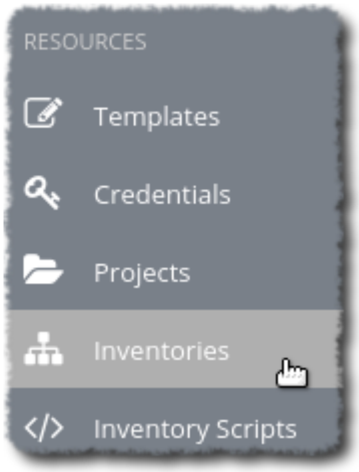
Go ahead and have a look at your project in gitlab.

Let's now setup a project in AWX

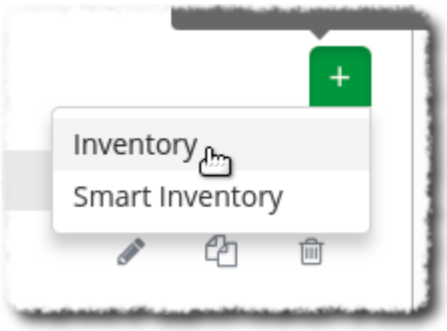
You have been using `~/course/ansible/setup/hosts` when running your playbooks. You will now enter these details into AWX.

Open http://localhost:8888 (http://localhost:8888) (if you need to login again, recall it is *admin*/*password*)

Then select *inventories* on the left menu



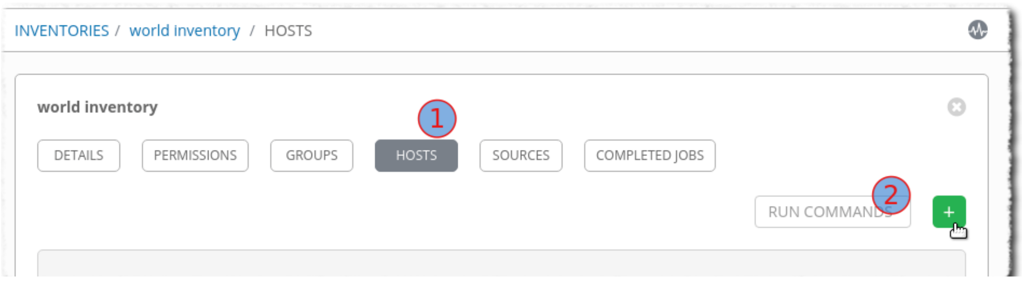
To add an inventory, click the "plus"-button on the left and choose **inventory**



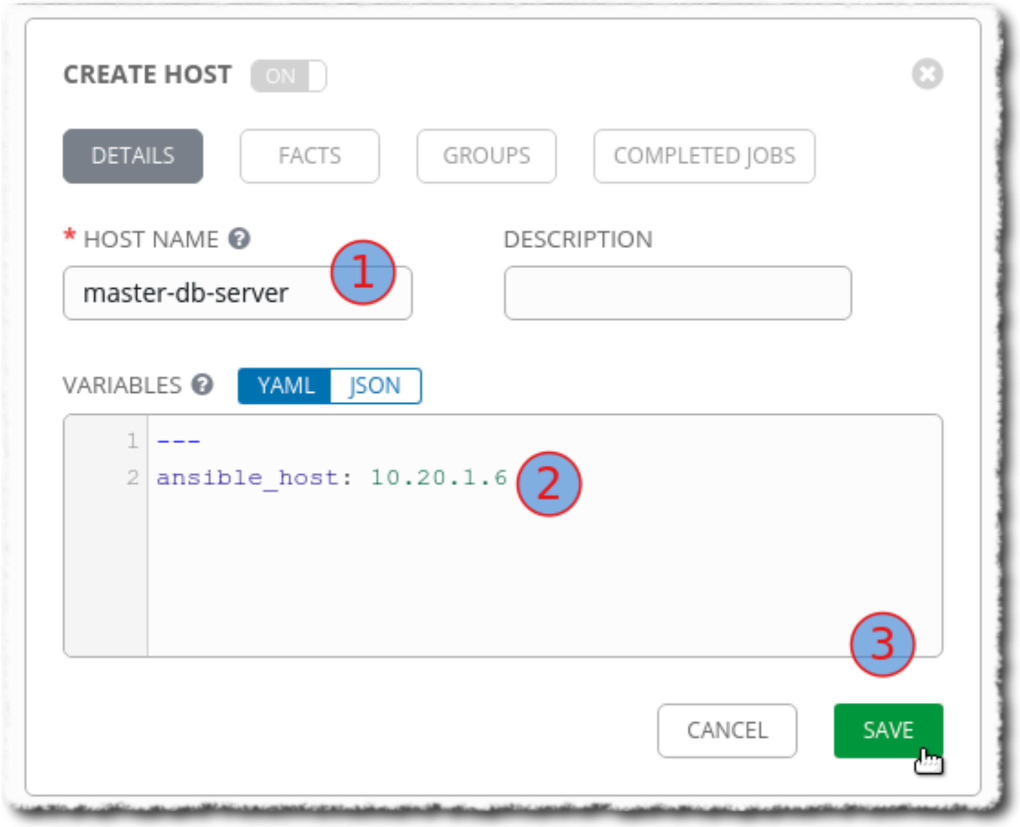
On the form, name your inventory *world inventory* and then immediately click the **save** button. This will enable the other buttons such as **groups** and **hosts**.

Add the hosts

Choose the **hosts** and click the green "plus"-button to add your first host:



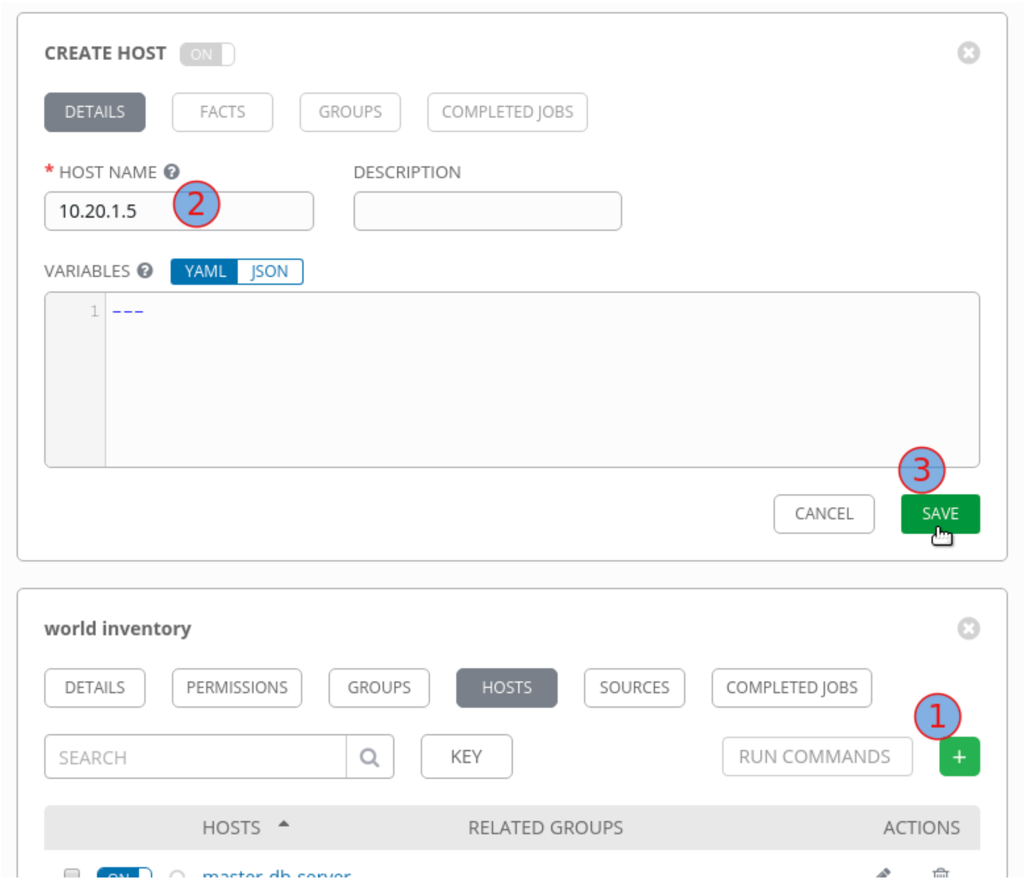
Then add the host for your database node



HOST NAME: master-db-server

```
---
ansible_host: 10.20.1.6
```

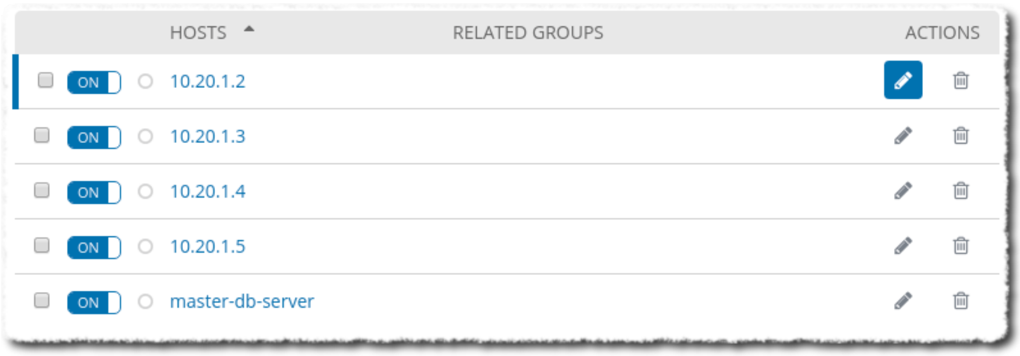
Then add the other ones. For these you don't need to use the `ansible_host` , just use the ip address for the **host name** as shown below for one of the hosts you'll need to add:



HOST NAME: 10.20.1.5

```
---
```

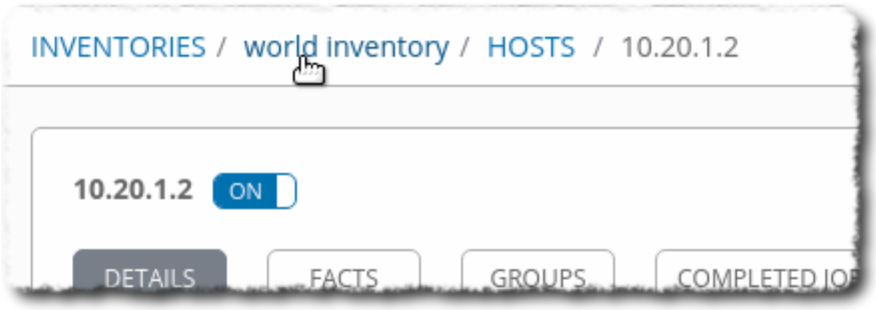
After adding all your hosts, you should now have this list:



	HOSTS	RELATED GROUPS	ACTIONS
ON	10.20.1.2		
ON	10.20.1.3		
ON	10.20.1.4		
ON	10.20.1.5		
ON	master-db-server		

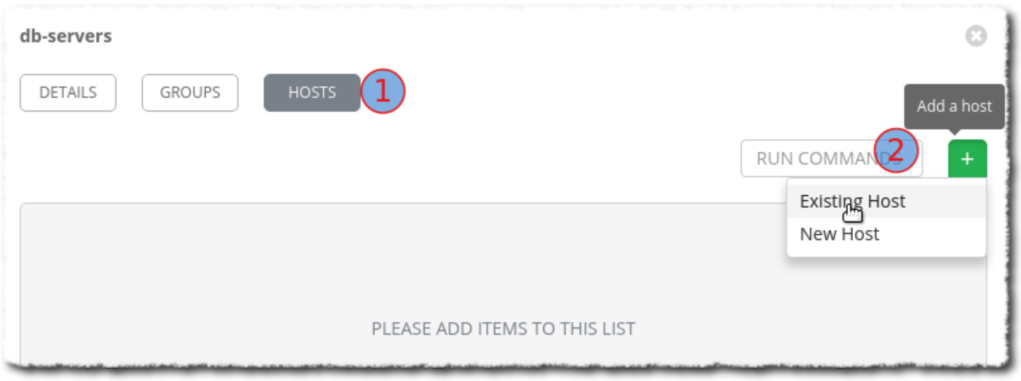
Groups

On the breadcrumbs in the top, choose to go back to your inventory:



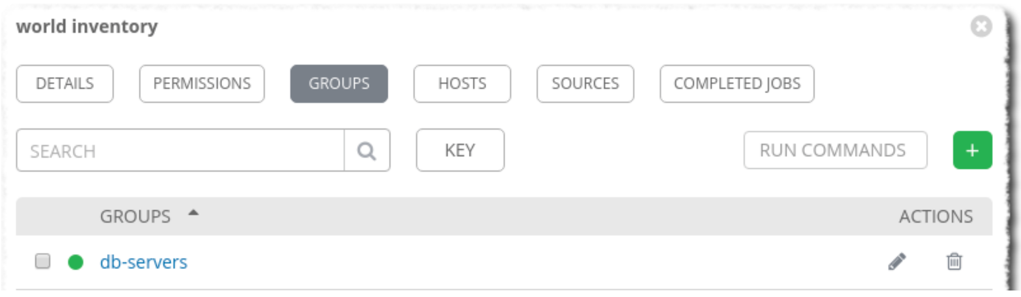
Then add a group by clicking on the green "plus"-button. Name the group **db-servers**. Save it so that the other options become available

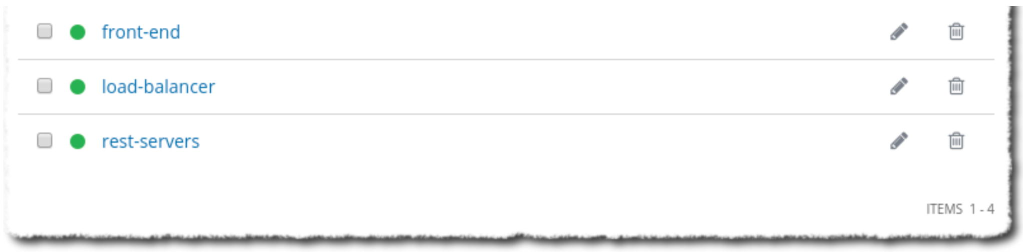
Then click on **hosts** and add an existing host:



Then choose for this **db-servers** group, the host **master-db-server**. Mirror the groups you have been using in `~/course/ansible/setup/hosts`.

Eventually you should have the following groups populated with their hosts.



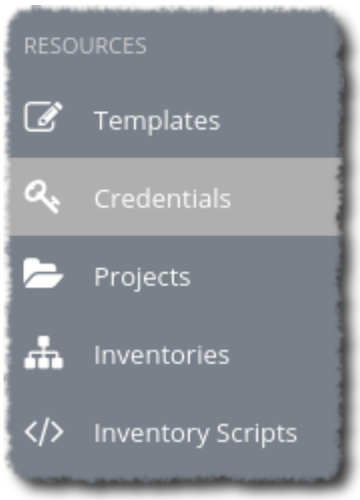


You will be adding two credentials:

- one to access the machines (we will use password authentication)
- and a second credential for the vault we are using

The SSH vagrant user

AWX needs to now which credentials to use to our machines. On the sidebar menu, choose **Credentials**



On the credentials page, click the "plus"-button to add a new credential. Use the following information for your credential:

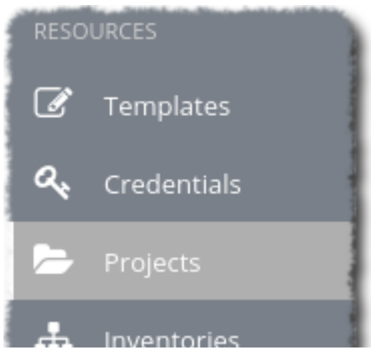
- Name the credential **world user**
- The credential type is **Machine** (use the magnifying glass to be able to select this value)
- Our credentials only consist of a username and password (**vagrant**/**vagrant**)

Our vault password

Then go through similar steps to add:

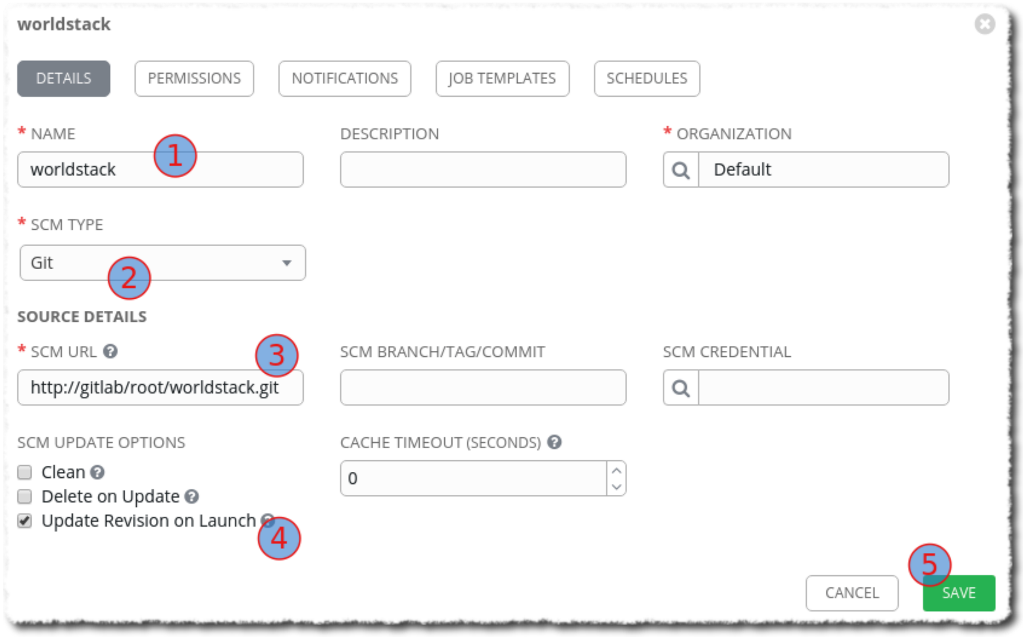
- A credential named `world vault`
- of type **Vault**
- The vault password is **tiger**

Let's now define our project in AWX. From the sidebar menu choose **Projects**



Define your new project using the following values:

- name it worldstack
- choose git as the **SCM Type**
- The **SCM URL** is http://gitlab/root/worldstack.git (http://gitlab/root/worldstack.git) (you can copy this from your project in gitlab, make sure to use the http variant and not ssh.)
- Check the box for **Update Revision on Launch**


worldstack

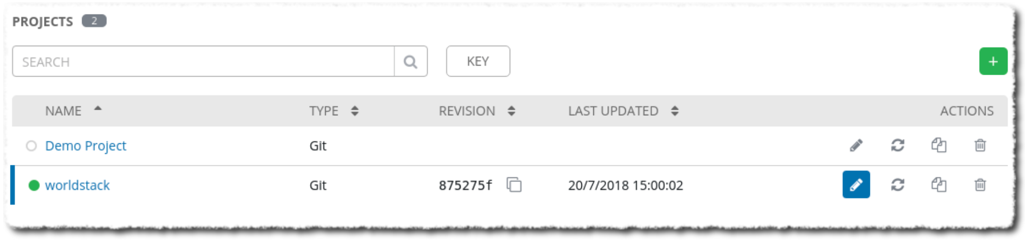
DETAILS PERMISSIONS NOTIFICATIONS JOB TEMPLATES SCHEDULES

* NAME: worldstack	DESCRIPTION:	* ORGANIZATION: Default
* SCM TYPE: Git		

SOURCE DETAILS

* SCM URL: http://gitlab/root/worldstack.git	SCM BRANCH/TAG/COMMIT:	SCM CREDENTIAL:
SCM UPDATE OPTIONS: ☐ Clean ☐ Delete on Update ☑ Update Revision on Launch	CACHE TIMEOUT (SECONDS): 0	

After saving, you should see the revision and last updated date change (you can always trigger a refresh, you could do that for example for the **Demo project** to test that out once)


PROJECTS 2

NAME	TYPE	REVISION	LAST UPDATED	ACTIONS
Demo Project	Git			
worldstack	Git	875275f	20/7/2018 15:00:02	

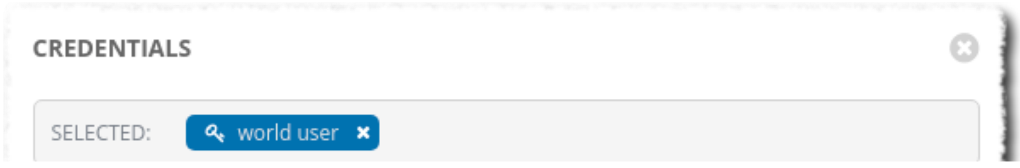
From the sidebar menu choose **Templates**.

Then add a new **Job Template** using the familiar "plus"-button

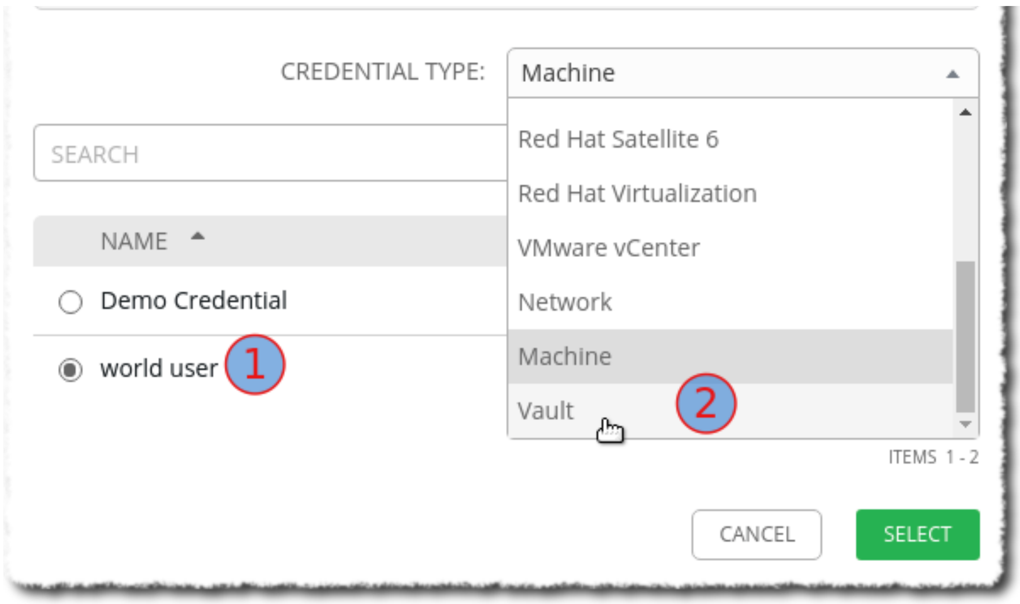
Some of the basic configuration:

- The name of this template is **worldstack Job Template**
- Choose our inventory **world inventory**
- Choose our project **world inventory**
- Choose our playbook `main.yaml` from the dropdown list

For the credentials, check the **world user** *Machine* credential. Then after having done that choose *vault* from the list of credentials types:

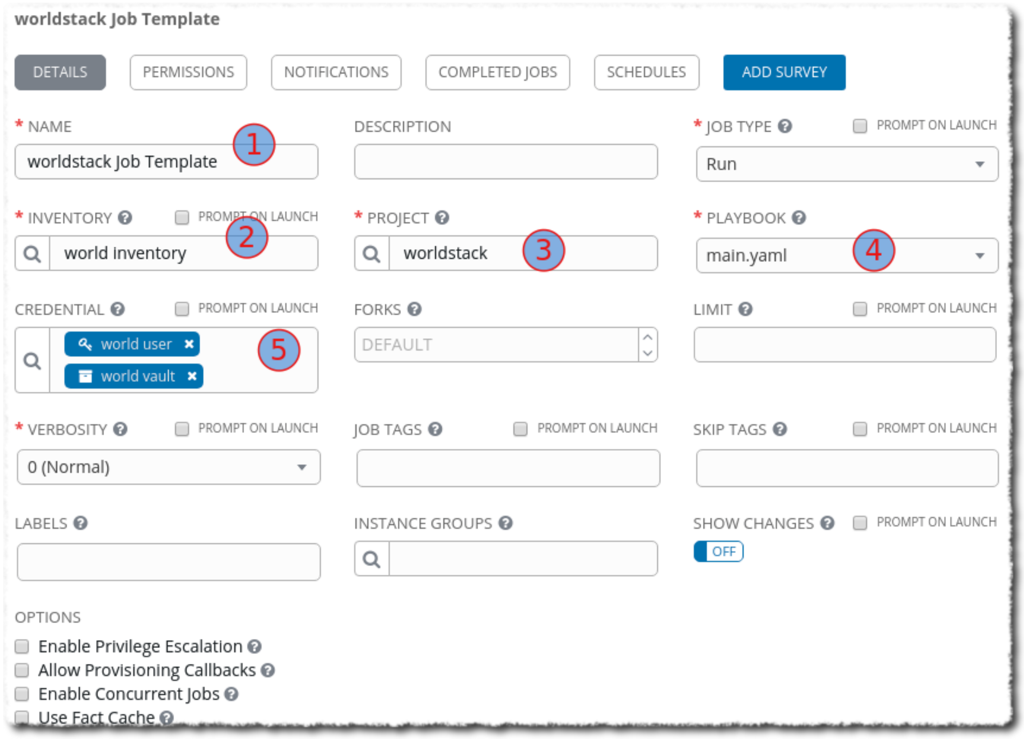

CREDENTIALS

SELECTED: world user



This now allows you to check our **world vault** credentials. When both are selected, close the dialog with the **select** button.

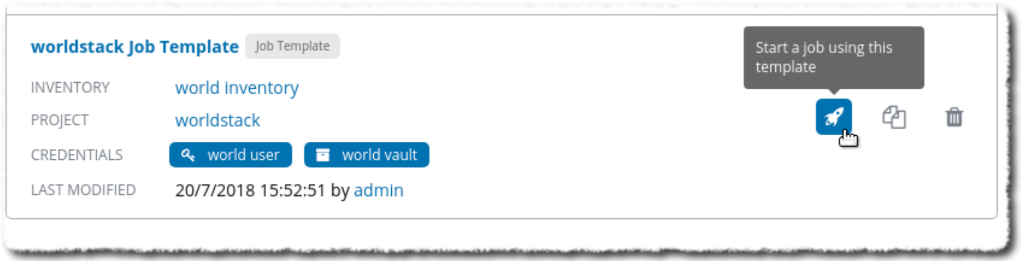
Eventually you project template should look like this:


worldstack Job Template

NAME: worldstack Job Template	DESCRIPTION:	JOB TYPE: Run
INVENTORY: world inventory	PROJECT: worldstack	PLAYBOOK: main.yaml
CREDENTIAL: world user, world vault	FORKS: DEFAULT	LIMIT:
VERBOSITY: 0 (Normal)	JOB TAGS:	SKIP TAGS:
LABELS:	INSTANCE GROUPS:	SHOW CHANGES: OFF

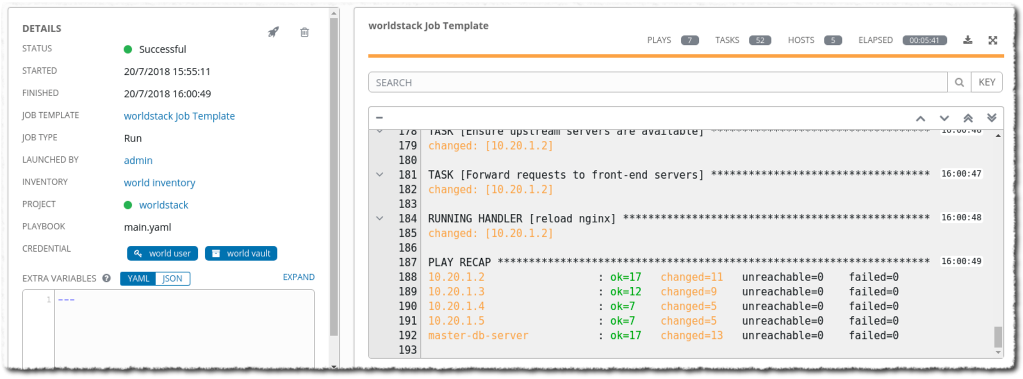
Click the **Save** button at the bottom of the screen.

If you scroll down on your template or on the templates page, you can launch the template by clicking the launcher icon:


worldstack Job Template Job Template

INVENTORY	world inventory
PROJECT	worldstack
CREDENTIALS	world user, world vault
LAST MODIFIED	20/7/2018 15:52:51 by admin

You should see the job page. After a couple of seconds the log on the left hand side will get entries. Enjoy the show, after a while it should complete:


DETAILS

STATUS	● Successful
STARTED	20/7/2018 15:55:11
FINISHED	20/7/2018 16:00:49
JOB TEMPLATE	worldstack Job Template
JOB TYPE	Run
LAUNCHED BY	admin
INVENTORY	world inventory
PROJECT	● worldstack
PLAYBOOK	main.yaml
CREDENTIAL	world user, world vault

EXTRA VARIABLES: YAML

```
---
```

worldstack Job Template

PLAYS 7 TASKS 52 HOSTS 5 ELAPSED 00:05:41

```
TASK [Ensure upstream servers are available] **********************************   16:00:46
changed: [10.20.1.2]

TASK [Forward requests to front-end servers] **********************************   16:00:47
changed: [10.20.1.2]

RUNNING HANDLER [reload nginx] ************************************************   16:00:48
changed: [10.20.1.2]

PLAY RECAP ********************************************************************   16:00:49
10.20.1.2                  : ok=17    changed=11    unreachable=0    failed=0
10.20.1.3                  : ok=12    changed=9     unreachable=0    failed=0
10.20.1.4                  : ok=7     changed=5     unreachable=0    failed=0
10.20.1.5                  : ok=7     changed=5     unreachable=0    failed=0
master-db-server           : ok=17    changed=13    unreachable=0    failed=0
```

Congratulations you have successfully used AWX to manage machines.

This is not an important step, perhaps more fun than anything else. It will expose you to the REST API provided by AWX, so there is some value.

The thing is AWX nor Ansible Tower come with webhooks support out of the box. But is does come with a REST api. We can use that to setup a webhook to launch a job when we push updates to our gitlab project.

Let's explore the AWX REST API a little. Open http://localhost:8888/api (http://localhost:8888/api). This is a navigational API, so drive it down to the **job_template** for your template (e.g, http://localhost:8888/api/v2/job_templates/7/ (http://localhost:8888/api/v2/job_templates/7/)). It is important to find your template id, and pencil this down somewhere as you'll be needing it later.

Add a webhook endpoint

We need a small webhook endpoint that then triggers AWX to launch a job. There is a project named adnanh/webhook (https://github.com/adnanh/webhook/) that does that. It allows you to extract information from the webhook request and use that as parameters to run as hell command. For this lab we will keep things simple, but it will get the idea and to make it more suffocated and flexible is not a very difficult. We will in fact hard-code the request URL to launch the AWX Job and not use any values from the webhook request.

We will run this webhook utility inside a docker container. The docker image edc4it/webhook (https://hub.docker.com/r/edc4it/webhook/) is available on docker hub. As you have done before you will add a docker compose service to the generated `docker-compose.yml` file in `~/course/ansible/awx/` .

Add the following after the gitlab service you added earlier on:

```
  webhook:
    image: edc4it/webhook:2.6.8
    hostname: webhook
    volumes:
    - $HOME/course/ansible/hooks:/etc/webhook
    command: "-hooks=/etc/webhook/hooks.json -hotreload -verbose -nopanic"
```

Run the docker container for our webhook service (run this command inside `~/course/ansible/awx/`):

```
$ docker-compose up -d webhook
```

Defining the hook

We are mounting the container's `/etc/webhook` to `~/course/ansible/hooks/` on your host. If you go to that directory you'll find a file named `hooks.json` . Open this file:

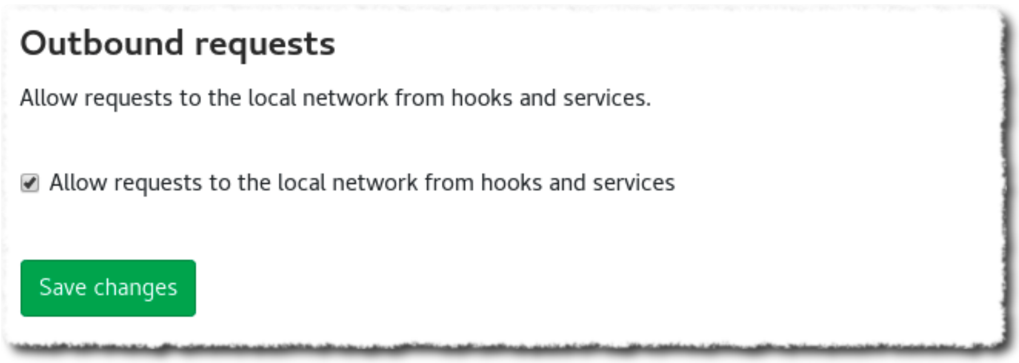
- review its contents (as mentioned earlier on, we are hard coding everything. Have a look at some more interesting examples (https://github.com/adnanh/webhook/blob/master/docs/Hook-Examples.md) of writing hooks. This hopefully gives you an idea on how to use this in a real environment)
- Notice we are hard coding the username and password
- Notice we are sending a POST request
- You will need to provide the url as the last parameter. From this container the url to AWX is `http://web:8052` . With that as a base set the value: `http://web:8052/api/v2/job_templates/…/launch/` (replace the three dots with the id of your template. It is also very important you add the training slash)

Configure GitLab

Before we can add the webhook to your gitlab project, we need to perform some configuration inside GitLab. Our webhook is local and by default GitLab does not allows such urls. We therefore need to change its configuration.

Click on the **wrench** to enter the *admin area* and then on **Settings** (most likely the last menu item on the side bar menu, it has a cogwheel as an icon)

Then search for **Outbound requests** and click **expand** it using the button on the right. Then enable to "Allow request to local networks …":


Outbound requests

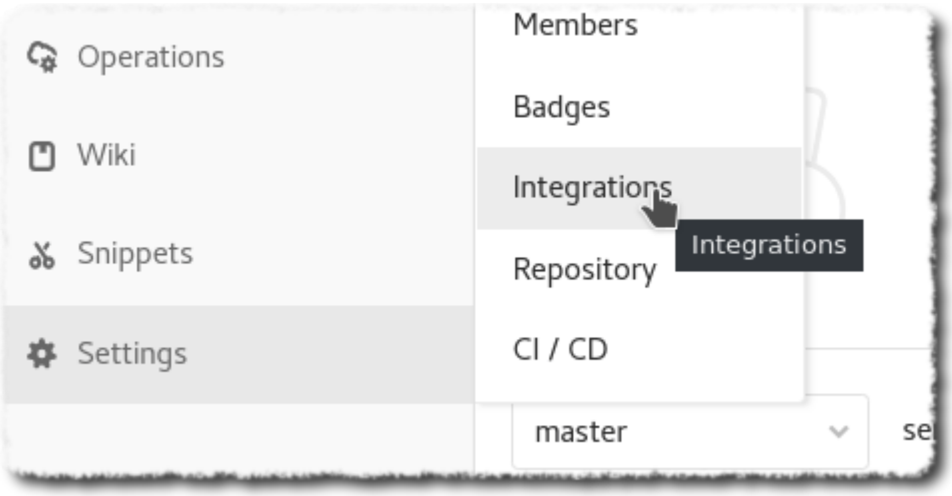
Allow requests to the local network from hook and services.

[✓] Allow requests to the local network from hooks and services

Click on **Save Changes**

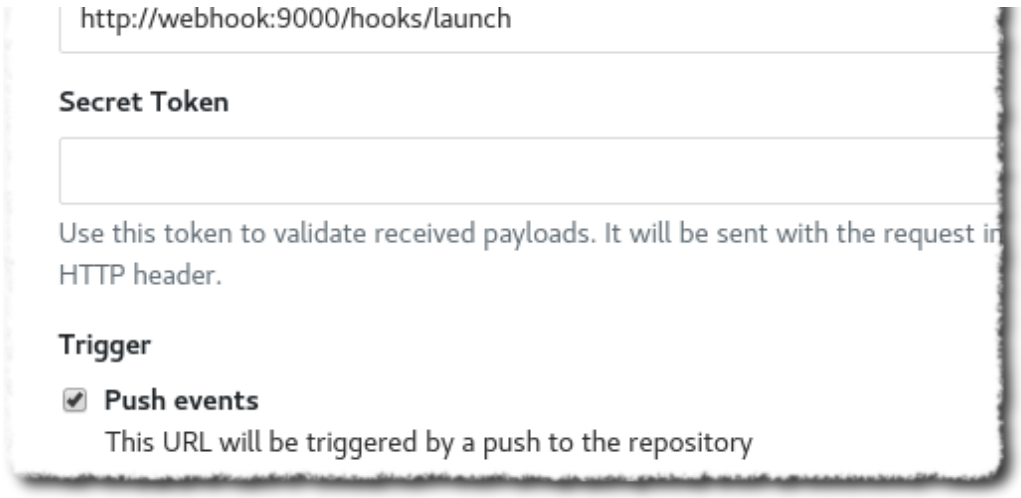
Add the hook to our project

Go to your project in GitLab. Then choose **Integrations** from the project's **Settings**:



Then set the webhook using `http://0.0.0.0:9000/hooks/launch` also know that for this lab, only enabling the **Push events** trigger is sufficient.

URL


http://webhook:9000/hooks/launch

Secret Token

Use this token to validate received payloads. It will be sent with the request in
HTTP header.

Trigger

✓ Push events
This URL will be triggered by a push to the repository

Click **Add webhook** at the bottom of the page

Test the webhook

On the same page, when you scroll down, you'll find a **Test** dropdown box. From it choose **Push events**

This should start the job in AWX (head over to the **jobs** page)

Push an update

Now move back to your "development console" (`~/course/ansible/worldstack`). We need to change a file so that we can commit/push a new version of our application that will trigger an AWX job. Let's change something that is easily observable. We will change the title on the html page.

Open `files/index.html` and change the title to "European Countries"

```
<a class="navbar-brand" href="/">European Countries</a>
```

When you issue a `git status` you should see our modification

Stage the file and commit locally

```
$ git add files/index.html
$ git commit -m "changed the title"
```

You are now ready to push, keep an eye on the AWX console while you do this:

```
$ git push origin master
```

A job should start. After it has completed (it should not take a long time), check http://10.20.1.2/ (http://10.20.1.2/). It should reflect the change you made to the title.