

Practice Session 10 : TOSCA Modelling & Orchestration

Make sure that you have already gone through [Lab-09](#).

In this practice session, you will learn multiple tools/technologies. To make it easier, we will keep the thing as simple as possible.

Before moving to the practice exercises, let's first recall following from the lecture session: *(make sure that you have gone through the [lecture 10](#))*

TOSCA

The TOSCA specification provides a language to describe service components and their relationships using a service topology, and it provides for describing the management procedures that create or modify services using orchestration processes.

Winery

Eclipse Winery is a web-based environment to graphically model TOSCA topologies and plans managing these topologies. The environment includes a type and template management component to offer creation and modification of all elements defined in the TOSCA specification. For more information visit here...<https://winery.readthedocs.io/en/latest/#>

Ansible

Red Hat Ansible Automation Platform is the IT automation technology that anyone can use. Recall the previous Lecture and Labs where you have already worked with Ansible.

radon-particle

This is the TOSCA definitions repository for the RADON project. The radon-particle github repository contains TOSCA blueprints, reusable definitions and extensions to deploy and manage your TOSCA-based applications.

It provides reusable TOSCA types of application runtimes, computing resources, and FaaS platforms in the form of abstract as well as deployable modeling entities.

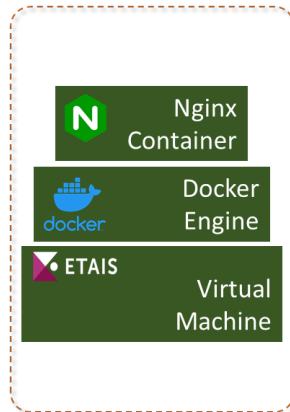
For more information on RADON project, visit <https://radonframework.eu/> , <https://radon-h2020.eu/>

TOSCA Orchestrator: xOpera

xOpera project includes a set of tools for advanced orchestration with an orchestration tool xOpera orchestrator or shorter opera.

Opera aims to be a lightweight orchestrator compliant with OASIS TOSCA and the current compliance is with the TOSCA Simple Profile in YAML v1.3.

A simple application



We will model the above application using Winery and deploy the application using xOpera. In this simple application, we will deploy an nginx web server on docker engine. The docker engine will be installed in a ubuntu VM.

The required node types are already created for this lab session.

1. Prerequisites and Winery Installation and Configuration

STEP-1: Prerequisites

- Make sure that two VMs are up and accessible
 - **microk8scontroller**: This will be used for modelling the above application. We will also install the orchestrator (e.g. xOpera) on this VM.
 - **microk8snode1**: The above modelled application will be deployed on this VM.
- Make sure that you can login to the **microk8scontroller** VM (over ssh) from your laptop.
- Uninstall the microk8s cluster on **microk8scontroller** using the command `sudo snap remove microk8s`
- Clean the disk space occupied by the docker images, stopped containers, volumes using `docker system prune -a` and press yes.
- Issue `docker logout` command to change the docker registry from gitlab to docker hub, as you have used gitlab registry in the previous lab.

STEP-2: Winery Installation and Configuration

- Login to <https://gitlab.cs.ut.ee> and fork <https://gitlab.cs.ut.ee/devops22fallpub/radon-particles> repository to your subgroup [Devops2023fall/students/devops2023Fall-<lastname>-<studyCode>](#)
 - While forking use the project name [prac10-radon-particles](#)
- Using Terminal, login to your **microk8scontroller** VM
- Go to the home directory. Clone the above forked repository.
 - The forked repository URL may look like below (if you have not changed the project slug while forking)

[Devops2023fall/students/devops2023Fall-<lastname>-<studyCode>/prac10-radon-particles](#)

- **Change directory** to [prac10-radon-particles](#)
- Install Winery in a docker container using the following command:

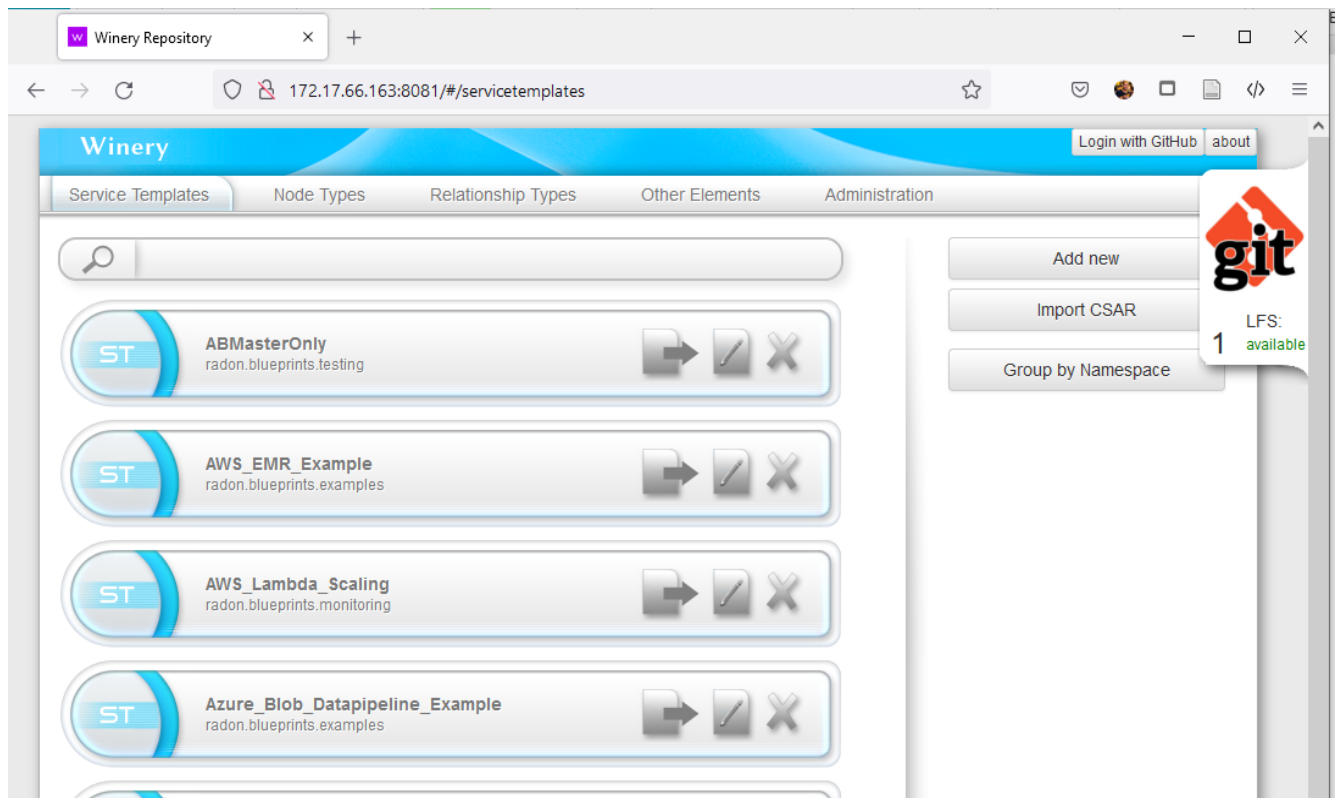
```
docker run -p 8081:8080 \  
-d --name prac10-tosca \  
-e PUBLIC_HOSTNAME=localhost \  
-e WINERY_FEATURE_RADON=true \  
-e WINERY_REPOSITORY_PROVIDER=yaml \  
-v $PWD:/var/repository \  
-u `id -u` \  
opentosca/radon-gmt
```

- You can issue `docker ps` command to see if the container is running
- The above [prac10-radon-particles](#) repository is now mounted to `/var/repository` directory inside the container
- References :
 - Winery Github repo: <https://github.com/eclipse/winery>
 - Winery documentation: <https://winery.readthedocs.io/en/latest/#>
 - Official radon particle repo: <https://github.com/radon-h2020/radon-particles>

2. Accessing Winery

To access the web UI of winery container

- Goto your web browser and visit http://<microk8s_VM_EXTERNAL_IP>:8081
- You should be able to get the homepage *similar* to below:



- Go through the tabs, such as Service Templates, Node Types, etc.

Eclipse Winery starts from the Service Template view. In this view, users can create new TOSCA service templates or maintain existing ones. A service template refers to an application template.

3. Installation of xOpera

xOpera is distributed as a Python package that is regularly published on PyPI. So the simplest way to test opera is to install it into a virtual environment.

- Make sure your login to **microk8scontroller** VM
- Update ubuntu: `sudo apt update`
- Create and change directory to a new opera directory: `mkdir $HOME/opera && cd $HOME/opera`
- Install python virtual environment if not present `sudo apt install python3-venv python3-wheel python-wheel-common`
- Create a virtual environment: `python3 -m venv .venv`
- Activate the virtual environment: `. .venv/bin/activate`
- Install pip: `pip install pip==21.3.1`
- Now it is the time to install opera: `pip install opera==0.6.8`

- If you receive some errors or warnings related to pyyaml versions, please ignore them.
- Check the installation by only issuing `opera` command

```
(.venv) ubuntu@winery:~/opera$ opera
error: the following arguments are required: command
usage: opera [-h] [-s {bash,zsh,tcsh}] [--version] {deploy,diff,info,notify,outputs,package,undeploy,unpackage,update,validate} ...

opera orchestrator

positional arguments:
  {deploy,diff,info,notify,outputs,package,undeploy,unpackage,update,validate}
    deploy              Deploy TOSCA service template or CSAR
    diff               Compare TOSCA service template to the one from the opera project storage and print out their differences
    info               Show information about the current project
    notify              Notify the orchestrator about changes after deployment and run triggers defined in TOSCA policies
    outputs             Retrieve deployment outputs (from TOSCA service template)
    package             Package service template and all accompanying files into a CSAR
    undeploy            Undeploy TOSCA service template or CSAR
    unpackage           Unpackage TOSCA CSAR to a specified location
    update             Update the deployed TOSCA service template and redeploy it according to the discovered template diff
    validate           Validate TOSCA service template or CSAR

options:
  -h, --help            show this help message and exit
  -s {bash,zsh,tcsh}, --shell-completion {bash,zsh,tcsh}
                        Generate tab completion script for your shell (default: None)
  --version, -v         Get current opera package version (default: None)
```

References :

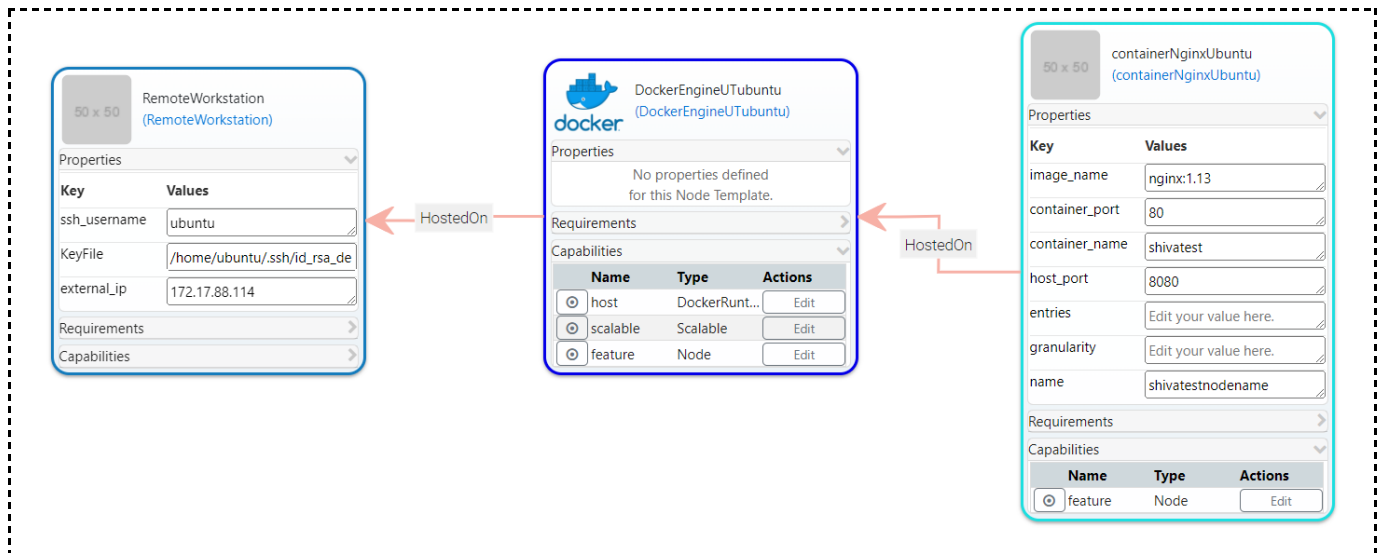
- xOpera Github repo: <https://github.com/xlab-si/xopera-opera>
- xOpera documentation is available here: <https://xlab-si.github.io/xopera-docs/index.html>
- Cli command references:
<https://xlab-si.github.io/xopera-docs/02-cli.html#cli-commands-reference>

Note: Remember that opera requires python 3 and a virtual environment. Before you issue any opera command, make sure that you have activated the virtual environment.

- `cd $HOME/opera`
- `. .venv/bin/activate`
- `opera`

4. Go through your application.

Let's first see the application. At the end of the modelling, your application *may* look like below:



At this point, you need to understand the essential properties of each node.

1. RemoteWorkstation:

The purpose here is to add the **microk8snode1** VM to the Ansible inventory list including the VM's External IP, login username and the ssh key file

Property name	Description
ssh_username	This is similar to the username that you use to login to the VM through ssh. For ubuntu VM this should be <code>ubuntu</code> .
KeyFile	This is the path to the ssh key file in the microk8scontroller VM. In this example, I have kept my ssh key file (<code>etais</code>) in the microk8scontroller VM inside the <code>/home/ubuntu/.ssh/</code> and here the value should be <code>/home/ubuntu/.ssh/etais</code>
external_ip	This is the external IP of the microk8snode1 VM where the application will be deployed.

Now Ansible will use the above information to login to your **microk8snode1** VM.

2. DockerEngineUTubuntu:

The purpose of this node is to install docker engine and other related libraries for ubuntuOS. Currently, this node has no properties to modify.

3. ContainerNginxUbuntu:

The purpose here is to create a container atop the docker engine in **microk8scontroller** VM. For this, the essential properties are:

Property name	Description
container_name	This is the name of the container.
image_name	This is the docker image name. The name should be given in <code><image_name>:[tag]</code> format (e.g. nginx:1.13). Make sure that the image is present in docker hub.
host_port	These two values are used to map/bind the port number of microk8snode1 VM and the container.
container_port	

Note:

- The [ContainerNginxUbuntu](#) node (which is in its development phase) can also be used to create any other container.

5. Model and Orchestrate the application.

In the above application, we will install a docker engine including all the related libraries on **microk8scontroller** VM. Once the docker engine is installed and started, we will create a nginx web server inside a container.

Make sure that you are in the **microk8scontroller** VM. It is assumed that you can login to **microk8scontroller** VM.

With the above information, let's model the application.

STEP-1: Creating the service template

- Access the winery UI: http://<microk8scontroller_external_IP>:8081
- Select the `Service Templates` tab.
- Click on `Add New` button to create a new service template
- Enter the name `myfirstToscaApp`
- Disable the `Versioning`
- Give `radon.blueprints.examples` as the `Namespace`
- Now the form/dialog box should look like below

Add a new Service Template

Name
myfirstToscaApp

Versioning:
☐ Enable Versioning

Namespace
radon.blueprints.examples

Template:

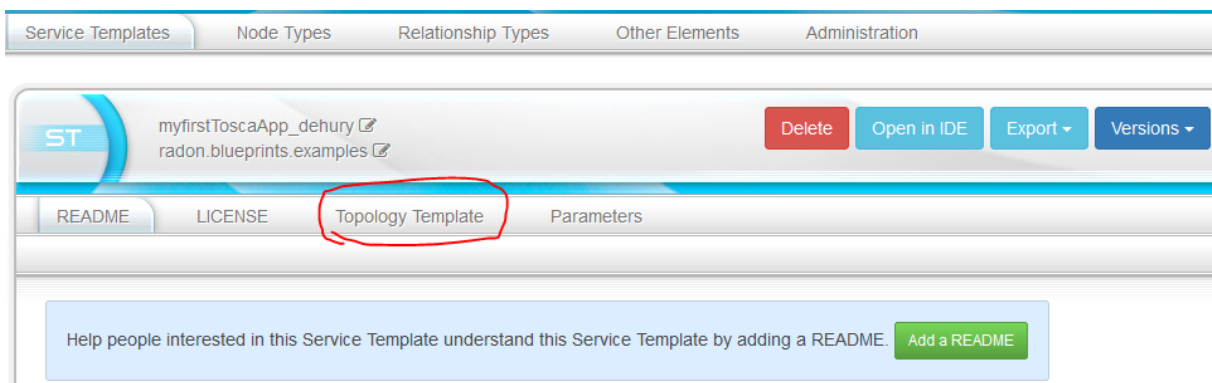
Cancel Add

- Now Click on Add button

STEP-2: Accessing topology modeller

Now you are inside the newly created service template or the application and ready to add the required nodes.

- Click on Topology Template -> Open Editor to access the topology modeller window, as shown below:

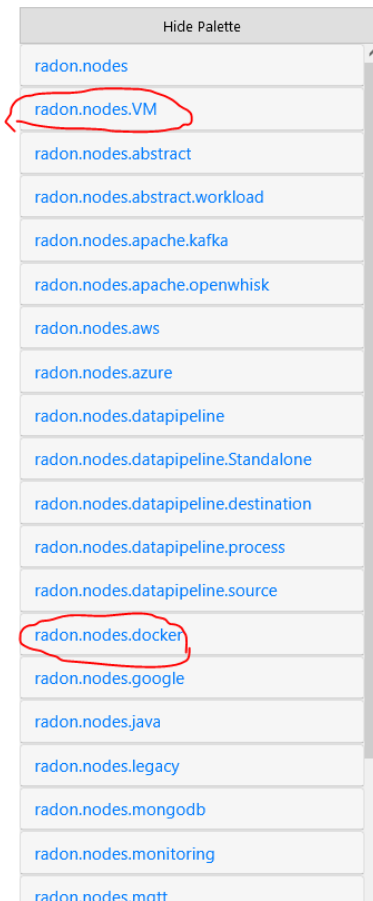


- This will open a new winery graphical modelling window.
- Go through the palette and get an overview of a list of node types available.

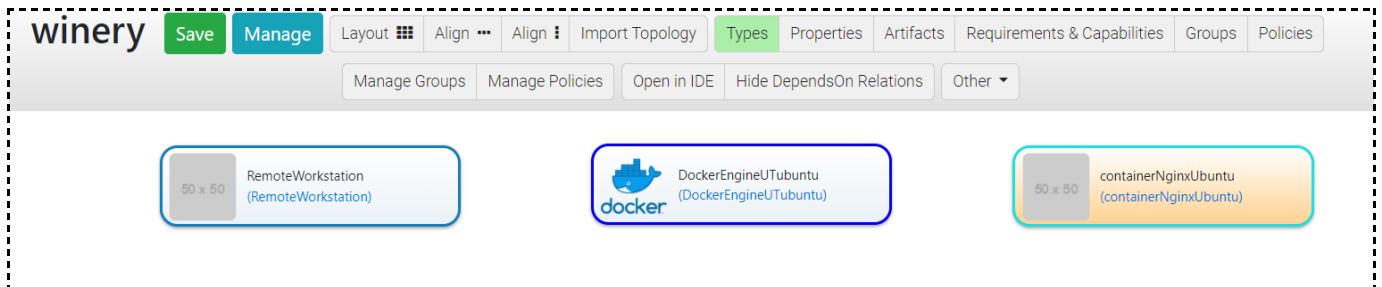
STEP-3: Adding required nodes

Now, you are ready to model your application by adding the required nodes (a node simply represents an application component).

- First add the following required nodes by dragging the nodetypes to the canvas:
 - RemoteWorkstation present under `radon.nodes.VM`
 - DockerEngineUTubuntu available under `radon.nodes.docker`
 - containerNginxUbuntu available under `radon.nodes.docker`
- You can get this nodetype from left side palette, as shown below:



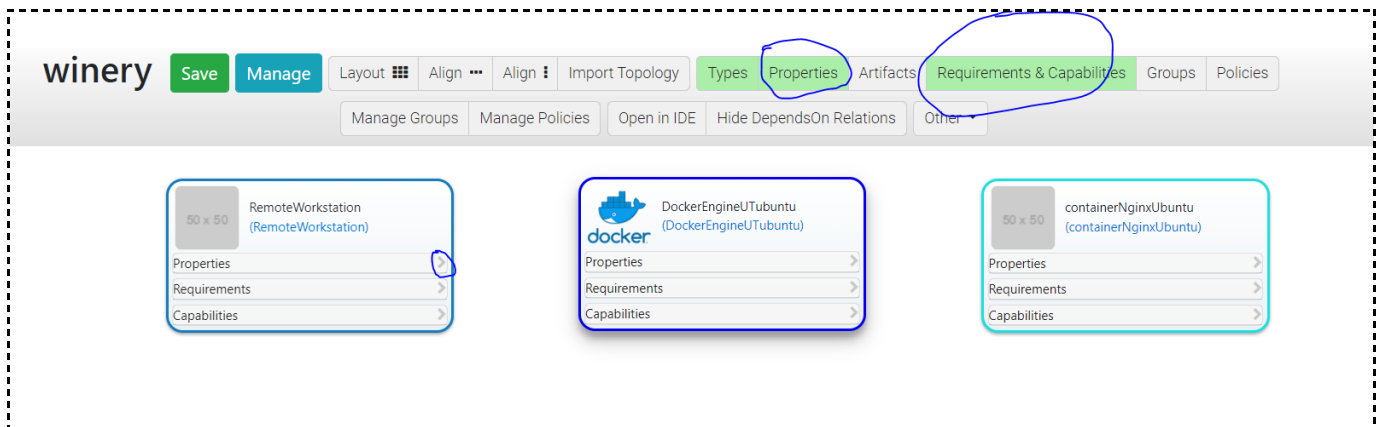
- After adding the nodes, the canvas may look like below:



- Now you need to provide the properties and establish the relationships among them.

STEP-4: Assigning values to properties

- Click on Properties and Requirement & Capabilities button, as shown below:



- Select **RemoteWorkstation** node and you will see the meta information and the list of properties in the right side of the window.
- Edit the **Name** of the node and provide the following properties value, as shown in below figure:
 - **ssh_username**: This should be **ubuntu**.
 - **KeyFile**: In the prerequisites step, I have already kept my ssh key file in **/home/ubuntu/.ssh** directory (in **microk8scontroller** VM). Update this value according to your environment. Ex: **/home/ubuntu/.ssh/etals** in my case
 - **external_ip**: This is the external ip of the **microk8snode1** VM. You should update this according to your **microk8snode1** VM.

Id

RemoteWorkstation_0

Name

RemoteWorkstation

Type

{radon.nodes.VM}RemoteWorkstation

Properties

Key	Values
ssh_username	ubuntu
KeyFile	/home/ubuntu/.ssh/etals
externalIp	172.17.88.243

Delete

- Now repeat the above step for other nodes.
- For **DockerEngineUTubuntu** node:
 - **name**: you may leave to default
 - **properties**: nothing to update here
- For **containerNginxUbuntu** node:

- container_name: give some name (e.g. mynginxserver)
- container_port: "80"
- image_name: nginx:1.13
- host_port: "8082"
- name: give some name (e.g. defaultNginxServer_chinmaya)

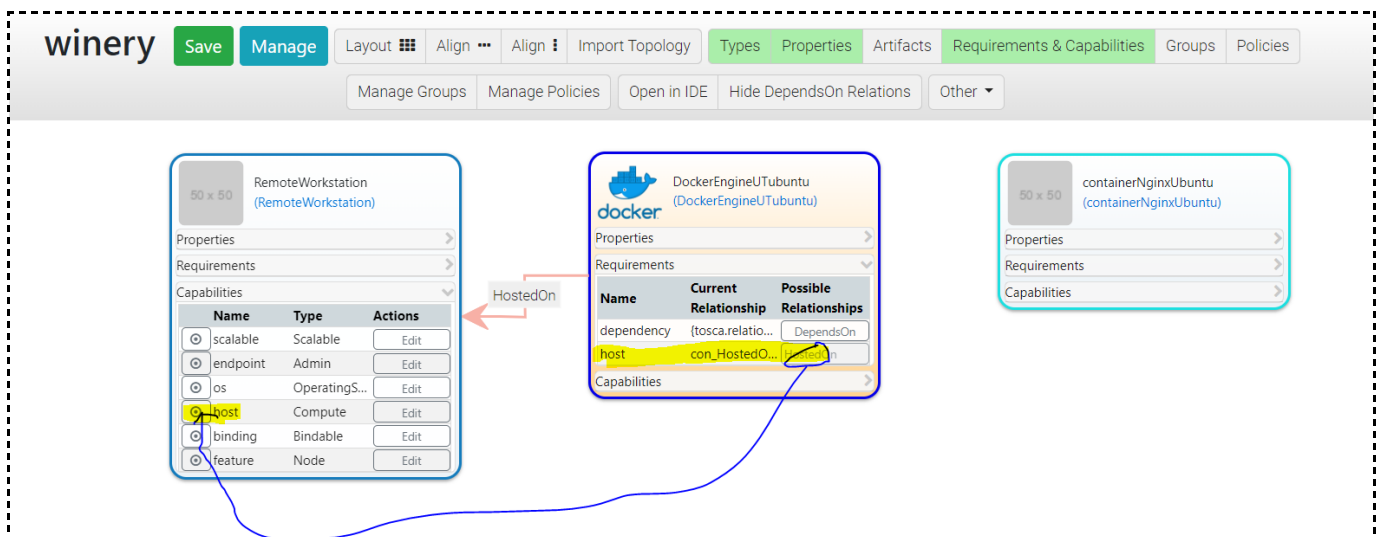
Note:

- Remember to use (") in container_port and host_port values
- You Use your name not "chinmaya" while naming the containerNginxUbuntu node

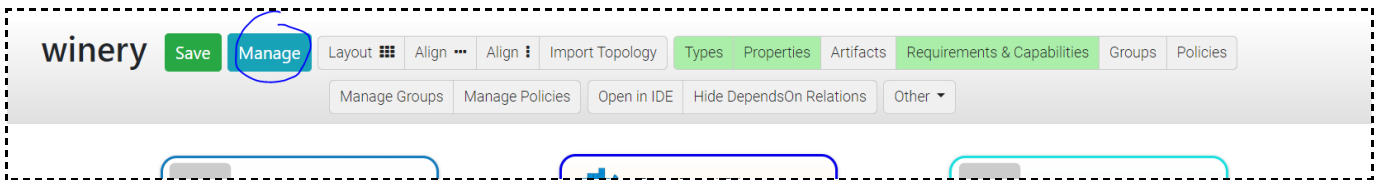
STEP-5: Connecting all the nodes

In this step, we will establish the relationships among all the added nodes.

- Click on the Capabilities of RemoteWorkstation node
 - Find host capability name
- Click on the Requirements of DockerEngineUTubuntu node
 - Find HostedOn relationship type
- Connect HostedOn requirement to host capability by click-n-drag, as shown below:

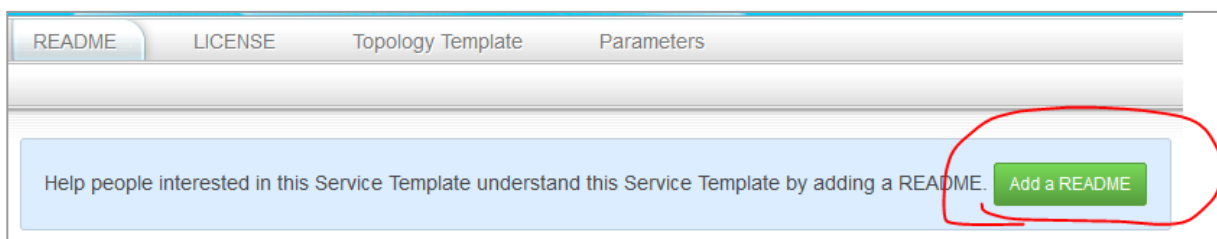


- Similarly connect containerNginxUbuntu node to DockerEngineUTubuntu node.
- Click on Requirements of containerNginxUbuntu node
 - Find HostedOn relationship
- Click on Capabilities of DockerEngineUTubuntu node
 - Find host capability name
- Establish the connection from HostedOn requirement to host capability.
- Now cross-verify the properties and click on Save button..
- Once saved, Click on **Manage**



STEP-6: Export the service template (or your TOSCA-based Application)

Add a README with your content. You can describe the application in your own words in a few sentences.



Now let's export the application by clicking on Export -> Download option.

This will download the whole application blueprint in CSAR format, i.e.

`myfirstToscaApp.csar`. A CSAR file is a ZIP file, it can be extracted, edited by hand and re-packaged.

Optional : Extract the CSAR package and go through all the internal files and folders. Once you unzip, you can start looking at

`_definitions/radonblueprints__myfirstToscaApp.tosca` file. You can see all the nodes and their corresponding properties values.

STEP-7: Orchestrate the application

- Upload the csar file (the zipped package) to the **microk8scontroller** VM (inside [/home/ubuntu/opera](#)). You can use scp command to copy the files from your laptop to microk8scontroller VM as shown below:

(Ex: `scp -i .\etais ..\Downloads\myfirstToscaApp.csar ubuntu@172.17.91.251:/home/ubuntu/opera`)

```
PS C:\Users\poojara\Desktop> scp -i .\etais ..\Downloads\myfirstToscaApp.csar ubuntu@172.17.91.66:/home/ubuntu/opera
myfirstToscaApp.csar
PS C:\Users\poojara\Desktop>
```

100% 67KB 4.3MB/s 00:00

- Login to the **microk8scontroller** VM
- Change directory to [/home/ubuntu/opera](#)
- Activate the virtual environment if it is not activated. : `. .venv/bin/activate`

- Now deploy TOSCA application: `opera deploy myfirstToscaApp.csar`
- **Screenshot**: Take the screen shot of the output (similar to the one given below) and save the image file with the name **Screenshot-5-7-OperaDeploy.jpg**. The screenshot should also include the command that you have entered.

```
(.venv) ubuntu@winery:~/opera$ opera deploy --clean test2_v2.csar
The clean state deploy option might have unexpected consequences on the already deployed blueprint.
Do you want to continue? (Y/n): y
[Worker_0] Deploying RemoteWorkstation_0_0
[Worker_0] Executing create on RemoteWorkstation_0_0
[Worker_0] Executing configure on RemoteWorkstation_0_0
[Worker_0] Executing start on RemoteWorkstation_0_0
[Worker_0] Deployment of RemoteWorkstation_0_0 complete
[Worker_0] Deploying DockerEngineUTubuntu_0_0
[Worker_0] Executing configure on DockerEngineUTubuntu_0_0
[Worker_0] Deployment of DockerEngineUTubuntu_0_0 complete
[Worker_0] Deploying containerNginx_0_0
[Worker_0] Executing create on containerNginx_0_0
[Worker_0] Executing configure on containerNginx_0_0
[Worker_0] Executing start on containerNginx_0_0
[Worker_0] Deployment of containerNginx_0_0 complete
```

If needed: `rm -rf .opera/`

- Once all the nodes are deployed you may verify if the nginx web server is installed and accessible.
- For this visit http://<microk8s-node1_VM_external_IP>:8082 and you should be able to see the default nginx web server page.
 - **Screenshot**: Take the screenshot of the entire default nginx web server page including the browser's address bar. **IP** and **port number** of the nginx server should be visible in the screenshot. Save the screenshot with the name **Screenshot-5-7-nginx-page.jpg**.
- Login to **microk8snode1** VM.
 - Execute **docker ps -a** command. If permission denied then use `sudo docker ps -a`
 - **Screenshot**: Take the screenshot of output of above **docker ps -a** command. Here, it is expected that the nginx container is up and running. The following information should be visible in the screenshot:
 - IMAGE
 - COMMAND
 - CREATED
 - STATUS
 - PORTS
 - NAMES

Save the screenshot with the name **Screenshot-5-7-nginx-container.jpg**.

6. Deliverable: Commit and push the changes

Here, you will push the changes that are made to the cloned

<https://gitlab.cs.ut.ee/groups/devops2023-fall/students/devops2023fall-<lastname>-<studycode>/prac-10-radon-particles> repository.

For this you may follow below steps:

- Using Terminal, login to your **microk8scontroller** VM
- **Change directory** to [prac-10-radon-particles](#).
Command: `mv prac-10-radon-particles Lab-10-radon-particles`
- Create a directory **Deliverable** in the project's root directory
Command: `mkdir Deliverable`
- Keep the following inside **Deliverable** directory
 - **Screenshot-5-7-OperaDeploy.jpg**
 - **Screenshot-5-7-nginx-page.jpg**
 - **Screenshot-5-7-nginx-container.jpg**
 - **myfirstToscaApp.csar**, this file you have downloaded in Step 5.6

Command: `scp -i <key> Screenshot-5-7-<name>.png`

`ubuntu@<VM_external_IP>:/home/ubuntu/prac-10-radon-particles/Deliverable`

You can copy .csar file from opera directory:

`cp opera/myfirstToscaApp.csar`

`prac-10-radon-particles/Deliverable/myfirstToscaApp.csar`

- Now commit and push the changes to your <https://gitlab.cs.ut.ee/groups/devops2023-fall/students/devops2023fall-<lastname>-<studycode>/prac-10-radon-particles> repository with the commit message "Uploading deliverables".

Submission:

1. Zip the **Deliverable** directory only which includes three screenshots and one .csar file and Upload the zip file to the course wiki page.
2. You may **Stop** the Virtual Machines and you can start using the same in the next **practice session**.

Don't delete your VMs

7. Bonus task : Develop your own TOSCA node type

- This task is beyond only use of existing node types. Here you basically need to create a new nodetype similar to `containerNginxUbuntu` node type.
- Before going to develop your own node type, first explore the `Node Types` tab from the winery UI. In the search box, let's enter `containerNginxUbuntu` and select the `containerNginxUbuntu` node type.
- Once you click on the `containerNginxUbuntu`, you will see several tabs, such as `Property Definitions`, `Attribute Definitions`, `Capability Definitions`, `Requirement Definitions`, `Inheritance`, `Artifacts`, `Interfaces` etc....

Go through those tabs, especially the tabs mentioned above.

Once finished, you can create your own node type with following properties and attributes:

Node type name: `containerGeneric`

Namespace: `radon.nodes.docker`

Disable Versioning

Properties, attributes, inheritance, artifacts, interfaces: similar to `containerNginxUbuntu`

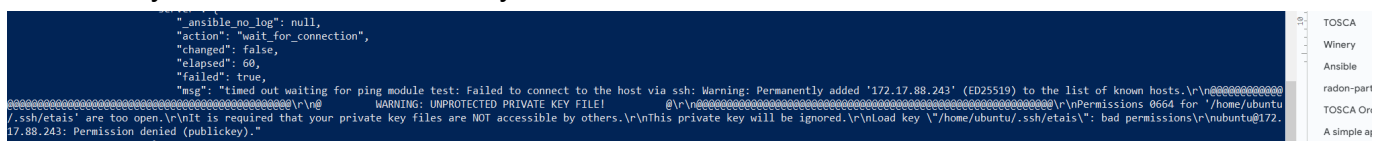
You are welcome to ask PI (Chinmaya Dehury) in any of the groups' time for more explanation and guide to do bonus tasks. But make sure that you have finished the above exercises.

Submission of bonus task:

- Similar to above Deliverable section commit and push the changes with the commit message "`adding bonus task`"
- Demonstrate that the newly developed node type is working. For this you may update the same service template created before.
 - For the demonstration purpose, contact PI.

Possible solutions to the common problems:

1. If you receive error w.r.t key file as shown below



Solution is to change the permissions of the key file stored under `/home.ubuntu/.ssh` using the command `sudo chmod 600 /home.ubuntu/.ssh/key_file_name`