Cloud Computing 2022/23 spring

Practice 13: Cloud service modeling with TOSCA

In this practice session, we will try out a selection of Azure public cloud services. First, we shall see how to manually launch a VM in the Azure Portal. Then, we will learn how to automate the deployment of entire compositions of cloud services using TOSCA, Winery, and xOpera. You create service templates using TOSCA in the Winery graphical editor. Then you will use the xOpera orchestration engine to deploy the created models to the Cloud.

https://cloudify.co/ is one example of a platform that uses TOSCA for defining complex Cloud Service deployment templates.

References

1. Azure Cloud documentation: https://docs.microsoft.com/en-us/azure/?product=featured
2. xOpera guide: https://xlab-si.github.io/xopera-docs/
3. Winery documentation: https://winery.readthedocs.io/en/latest/user/getting-started.html

Exercise 13.1 Launching VMs in Azure

In this task, you will get the experience of deploying a VM in Azure, a process you are by now very familiar with in OpenStack.

PS! If you have any issues with Azure (For example, low remaining credit), you can create an OpenStack instance instead.

- Open the Azure Cloud portal: https://portal.azure.com/#home
- Search for the **Virtual machines** Azure service inside the Azure Cloud console
- Create a New Azure VM
  - Resource group: Create new one, and name it "Lab13ResourceGroup"
  - Region: North Europe
  - Image: `Ubuntu Server 20.04 LTS`
  - Size: Select "See all sizes" and observe the options. Choose @B2s@ or `D2as_v4`.
    - If you are not able to find/see D2as_v4, browse around and look for a cheaper VM type that has 4 GB RAM, 1-2 vCPU cores.
    - Take a look at the different options available in the table. Note the resources and cost. Write down your observation/discussion - how much would a VM similar to our OpenStacks "m2.tiny" used in previous labs cost on Azure? Note down the name of the similar VM, and list its specs.
  - Authentication type: SSH public key
  - SSH public key source: generate new (unless you have already created one inside Azure)
  - username: ubuntu
- Continue to the "Disks" section
  - Choose "Standard SSD".
- Continue to "Networking" tab.
  - You should see a network with your Resource Group name active.
  - Adjust "inbound ports", to **allow Port 80 and 22**.
  - Rest of the settings can stay the same.

- Deploy the VM ("**Review & Create**")
- NB! make sure you download and save the SSH key file. It is needed to connect to the deployed VM.

- After deployment, let's test if we can SSH into it.
  - Go to the VM resource page in Azure and copy the VM IP address
  - Using the SSH key, try to SSH into the VM.

If SSH is successful, let's install Docker to prepare for the next steps.

- Install Docker following this guide.
  - There are multiple options documented in the guide, a very convenient approach is to install using the convenience script.

    - Verify you have Docker working in the VM with `sudo docker --version`

Exercise 13.2 Running Winery

Let's now set up the web-based graphical service modeler Winery.

- (Optional: Restricting access to the Web application)
  - Notice we are running a VM in the **public web** with port 80 exposed. Potentially, anybody could visit your Winery instance. If you want, you can block access only for your own (public) IP. Here's how to do it:
  - In Azure Portal, open your Azure VM, and click on **Networking**
    - Locate the "inbound" rule for Port 80; open it by clicking.
    - Update the **Source IP Addresses** to: `YOUR_PUBLIC_IP` you can find out your public IP. For example, by visiting https://www.whatismyip.com/

- Let's create a Docker container with the Winery image from Docker Hub using the following CLI command:

```
sudo docker run -itd -p 80:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_FEATURE_RADON=true \
  -e WINERY_REPOSITORY_PROVIDER=yaml \
  -e WINERY_REPOSITORY_URL=https://github.com/UT-Cloud-Computing-Course/radon-particles \
  opentosca/radon-gmt
```

  - It should be up and running in about a minute. Use `docker logs` to track the status.
- You can access the Winery web interface using your browser on the URL:
  - `http://IP_OF_AZURE_VM:80` (Replace the IP_OF_AZURE_VM with the actual Azure VM IP).

Exercise 14.3 Designing a service template in Winery

In this task, we will design a new Cloud deployment service template. We will try to design a service model for deploying an Azure Serverless FaaS function. Recall how we manually deployed a serverless function to Azure Cloud during lab 4. In this task, we aim to automate the necessary steps for deploying the function, such as creating resource groups, storage resources, the function definition, and the deployment of code.

We will be using TOSCA models created in the RADON project, which are individually deployable and freely composable software service blocks that contain *Ansible* playbooks for installing, starting, configuring, stopping, and deleting them. There are models for AWS, Azure, Google Cloud, OpenStack, etc.

- The source code for these models is available here: http://github.com/radon-h2020/radon-particles
  - We are using a modified fork for this course: https://github.com/UT-Cloud-Computing-Course/radon-particles

NB! Best to avoid spaces, underscores, and numbers in names or properties while you configure things in this task!

In the Winery interface, Create a new Service template:

Define the name of the service template. Also, make sure to *disable* versioning.

Open the Service template graphical editor:

Open the Palette on the left side of the Canvas

Find `radon.nodes.azure` group.

Drag and drop the following node types to the canvas::

1. **AzurePlatform -** configures the Azure Python libraries needed
2. **AzureStorageAccount -** creates a new Storage account
3. **AzureHttpTriggeredFunction -** deploys a new Azure FaaS function.
    - Function code is provided as a zip file.
4. **AzureResourceGroup -** creates a new Azure *Resource Group*

And configure them as described in the following section.

Also, make sure first to activate showing the configuration values of the nodes by clicking on the **Properties** button on top of the canvas:

NB! Also, remember to change the `ResourceGroupName`, `StorageGroupName`, and `AzureFunctionName` with the names you want to assign for **New** Azure resources. Avoid using names of resources and functions **that already exist**; otherwise, it will be hard to test whether we succeed with the deployment later.

Configure the **Azure Platform** node:

- **Region -** `northeurope`
    - Availability zone inside the Azure cloud
- **Name -**
    - Freely chosen name for this TOSCA node type

Configure the **Azure Resource Group** node:

- **Resource group name -**
    - Resource group to be created
    - NB! Choose a unique name across Azure
- **Region -** `northeurope`
    - Availability zone inside Azure cloud
- **component_version:**
    - Not used, can leave empty

Configure the **Azure Storage Account** node:

- storage_account_name:
    - Name of the Storage account to be created
    - NB! Choose a unique name across Azure
- resource_group_name:
    - Name of the resource group (from the previous step)
- region: `northeurope`
- Can leave default values:
    - account_type: Standard_RAGRS
    - storage_kind: StorageV2
    - access_tier: Hot

Configure the **Azure HTTP Triggered Function** node:

- storage_account_name:
    - Name of the storage account (from the previous step)
- function_app_name:
    - Name of the function to be created
    - NB! Choose a unique name across Azure
- resource_group_name:
    - Name of the resource group (from the previous step)
- name:
    - Freely choosable name for this TOSCA node type
- zip_file: `/home/ubuntu/function.zip`
- region: `northeurope`
- Use these values:
    - runtime_version: 3.8
    - auth_level: anonymous
    - methods: []
    - functions_version: 4
    - route: none
    - os_type: Linux
    - route_prefix: api
    - runtime_type: python

NB! Make sure to SAVE the created Service template every once in a while. Otherwise, changes might be lost, To create connections (relationships) between the Azure Node types to each other, first, let's activate the Requirements and Capabilities view by clicking on the respective button on top of the canvas:

Then, we click and drag connections from the Requirements of one node type to the Capabilities of another node type.

Set up the following relationships:

- Define the **hosted on** relationship to indicate which node should be deployed on which other node:
  1. Drag `HostedOn` requirement from **AzureStorageAccount** into **AzurePlatform** `host` capability
  2. Drag `HostedOn` requirement from **AzureHttpTriggeredFunction** into **AzurePlatform** `host` capability
  3. Drag `HostedOn` requirement from **AzureResourceGroup** into **AzurePlatform** `host` capability
- Define the **Depends on** type relationship to define which node should be deployed before other nodes are deployed:
  1. Drag `DependsOn` requirement from **AzureHttpTriggeredFunction** into **AzureStorageAccount** `feature` capability
     - *storage account should be deployed before the function*
  2. Drag `DependsOn` requirement from **AzureStorageAccount** into **AzureResourceGroup** `feature` capability
     - *resource group should be deployed before the storage account*

The result should look something like this:

NB! Check that the relationships you created between nodes in Winery and their direction match this image. Otherwise, you may run into issues later.

Download the created Service template as a `.csar` file container (basically a zip container) by clicking on the **Manage** button on the top left, exit the Editor view and choose **Export** and **Download**:

NB! For verification, you can check that your service template looks similar to the example image.

Take a screenshot of the finalized Service template in Winery

Exercise 13.4 Setting up RADON xOpera

In this task, we will set up xOpera TOSCA orchestrator, which is able to parse the models we have previously created and deploy them automatically.

opera aims to be a lightweight orchestrator compliant with OASIS TOSCA. XOpera takes the TOSCA service template and manages the life cycle of the nodes inside the service. Life cycle refers to creating, configuring, starting, stopping, and deleting the individual nodes. The life cycle commands of TOSCA node types are implemented as Ansible playbooks, included inside each TOSCA node type.

- Connect to the Azure VM over SSH and update the Linux repositories `sudo apt update`
- Install the python virtual environment `sudo apt install -y python3-venv python3-wheel python-wheel-common`
- Install Azure CLI package:
  - `curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash`
- Create a new opera directory: `mkdir opera`
- Move into the created folder: `cd opera`
- Create a python virtual environment `python3 -m venv .venv`
- Activate the virtual environment:
  - `source .venv/bin/activate`
  - Now you should see **(.venv)** $ in-front of the command line prompt
- Update pip `pip install --upgrade pip`
- Install opera inside the virtual environment: `pip install opera==0.6.8`

NB! Every time you want to execute opera in the following tasks, make sure that you are in the correct folder and have activated the Python Virtual environment.

1. `cd opera`
2. `source .venv/bin/activate`
3. `opera`

Exercise 13.5 Deploying a service template

Now we will use the xOpera orchestrator to deploy the previously created (and saved as a .csar file) TOSCA service template.

1. Copy the downloaded `.csar` file into the VM.
   - Just like you have moved files into VMs in the previous practice sessions. (e.g. using `scp`). See the "Tips" section at the bottom of this guide if you forgot how to do it.
   - Place it in the `~/opera` folder

Also, let's copy the function code `function.zip` file into the VM. This .zip contains the Serverless function implementation. The implementation and deployment guide we use in this lab task have been created by scholars from the Imperial College of London: Shreshth Tuli, Runan Wang, and Giuliano Casale for their Performance Engineering course.

The function we are going to use in this task is related to style-transfer machine learning models. Artistic style transfer models mix the content of an image with the style of another image. The style transfer model used in this function is called Rain Princess.

- Get the function **.zip** and copy it to the VM:
  - The file MUST be located inside the home folder of the Ubuntu user (`/home/ubuntu/function.zip`)
  - You can download the zip file containing the function from here: https://owncloud.ut.ee/owncloud/s/LjmcTJNK4SpnRjs
  - You will find the password for the link in the Zulip topic `practice-session-13`

Now that we have the .csar describing the deployment, and the .zip function the .csar uses, let's deploy them to Azure!

1. Log into the Azure Cloud inside the VM:
   - PS! Unfortunately, because we are using the University of Tartu Azure tenant, we do not have access to Azure API credentials to automate the login step also using xOpera. So we have to manually log in before calling xOpera.
   - `az login`
   - Follow the steps you see when running this command.
2. Make sure Python Virtual environment is active
   - `cd ~/opera`
   - `source .venv/bin/activate`
3. Use xOpera to Deploy the TOSCA CSAR file:
   - `opera deploy -r SERVICE.csar`
     - Replace SERVICE with the correct name of the file.
     - `-r` specifies that xOpera should resume any ongoing deployment that might have failed.
     - If you run into any errors, you may have to use `opera deploy -c SERVICE.csar` instead if you want to make sure xOpera deploys everything again from a clean state and does not resume from the latest state.
   - If you get any errors, scroll up and try to find the actual error message inside the JSON output of the command.
     - NB! Check the following section for an example of where the error can be located.
4. xOpera will install necessary Azure libraries inside the local python Virtual environment and deploy and configure Azure resource group, storage account, and FaaS function.
   - If there are no errors with the opera deployment command, you will see output about deployments of the 4 different components (ResourceGroup, Function, ..) taking place.
     - Otherwise, check below how to troubleshoot/debug.
5. Once the deployment is successful, you can log into Azure Portal and verify that the function and other resources have been deployed correctly.
   - You should see the *Storage account* and *Function* appear under your created *Resource Group*
   - Going to https://FUNCTONNAME.azurewebsites.net/ should show it is deployed
   - https://FUNCTONNAME.azurewebsites.net/api/httpexample shows one of the three APIs inside the function(s)

- You should see xOpera print out the status that all four node templates were deployed successfully, and you should see no lengthy error messages.
  - Check the end of this practice session page for debugging info.
- Take a screenshot of the xOpera output
- Answer: In which order are the 4 components deployed?
  - Why are they being deployed in this order?

Example of a successful xOpera deployment:

```
(.venv) examples/hello$ opera deploy service.yaml
[Worker_0]   Deploying my-workstation_0
[Worker_0]   Deployment of my-workstation_0 complete
[Worker_0]   Deploying hello_0
[Worker_0]     Executing create on hello_0
[Worker_0]   Deployment of hello_0 complete
```

- If you run into any errors, you may have to use `opera deploy -c SERVICE.csar` instead if you want to make sure xOpera deploys everything again from scratch.

NB! If you get any xOpera errors, check the **Debugging xOpera** section, at the end of this practice guide

Exercise 13.6 Invoking the deployed Serverless Function

Let's try using the serverless function. Check what is the function endpoint URL by going to the Azure Portal, and searching for the Function App service.

You should see 3 different functions under your Function App like so:

- Go to your function app service -> Functions -> onnx -> Get function URL.
  - It will include an additional code component, something like this: `https://functionname.azurewebsites.net/api/onnx?code=5HbZjm8m...`

One of the endpoints of the function called onnx can be used for image processing. Let's test it. The ONNX takes as input an image file (jpg!), and returns a processed version of it. It can be invoked with an HTTP request like so:

- `curl -s --data-binary @input_picture.jpg https://YOUR_FUNCTION_URL_HERE -o output.jpg`
  - Tip: to invoke it, try using your laptop instead of the Azure VM, that way you can quickly see if the output.jpg is what you expected. (On Windows you may need to use Git Bash to use *curl* like this)

Debugging function call.

- If the function does not return an image, try the command without redirecting output to an image file:
    - `curl -s --data-binary @input_picture.jpg https://YOUR_FUNCTION_URL_HERE`
        - This might now print out the error text that may indicate why the function call or image creation/saving failed.

As a result, you should get an output.jpg appearing where you ran the command, it's contents are a processed version of the image. Example input and output:

If you are interested in how the Azure TOSCA node types are defined and how their life cycle commands (create, delete) are implemented using Ansible, then you can check their GitHub repository folders: https://github.com/UT-Cloud-Computing-Course/radon-particles/tree/master/nodetypes/radon.nodes.azure

Deliverables:

1. Upload the CSAR file
2. Link to the Function (ONNX) deployed in Azure
3. Take a screenshot of the finalized Service template in Winery
4. Take a screenshot of the xOpera output
5. Answer: Discuss the VM Size options of Azure (See task 3.1)
6. Answer: In which order are the 4 components deployed at task 13.5? Why are they being deployed in this order? What controls the deployment order?
7. Delete the Azure VM

Task	Lab 13 - Tosca
Current submission	ZIP (16:28 02.05.2023) If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File No file chosen
Comment	
	Submit

NB! Remember to shut down or remove the Azure VM after you have finished! Otherwise, it will keep spending your credits.

Tips

- How to move files between hosts using SCP:

These examples include the "-i" parameter, which indicates which keyfile to use to authenticate

- Copy file from a remote host to localhost
    - `$ scp -i /path/to/ssh_public_key.pem username@REMOTE_VM_IP:/home/ubuntu/file.txt /path/to/local`
    - Above command takes file.txt from /home/ubuntu folder of the remote VM, and copies it to your machines folder with path: `/path/to/local`

- Copy file from local machine to remote host:
    - `$ scp -i /path/to/ssh_public_key.pem file.txt username@REMOTE_VM_IP:/remote/directory/`
    - Above command sends the file "file.txt" from the current working directory to the folder "/remote/directory/" of the remote machine.

Debugging xOpera

In case of errors xOpera may output 2 types of errors:

1. Python error, which indicates something went wrong
    - Usually, the actual error is not displayed here, and you can scroll past the Python errors
2. Error inside the Ansible JSON output - if something is wrong on the Ansible side.
    - Anything related to Azure APIs, wrong configuration, etc., will be shown inside the JSON output. Look at the following picture of how to find such errors.

Example of how to find an error inside the xOpera output:

Possible solutions to potential issues

1. If you get an error that a Storage account or resource group does not exist:
    1. Check that you have specified their names exactly the same way in all the nodes.
    2. Check that the relationships in Winery match the example image. Otherwise, TOSCA nodes may be deployed in the wrong order, and the function is deployed before the storage account exists, and deployment will thus fail.
2. How to get xOpera to redeploy node types it previously already deployed.
    - You can use the clean state deployment option to make sure xOpera reconfigures everything again:
    - `opera deploy --clean-state shivaservice.csar`
3. `The storage account named mystorage is already taken.`
    - For Azure storage accounts, you must provide a name for the resource that is unique across Azure.
    - Solution is to rename the storage account name in Winery and deploy the service again.
    - Use clean state deployment:
        - `opera deploy --clean-state shivaservice.csar`
    - Make sure you rename the storage account name also inside the other nodes if they use the storage account name.
4. If the resulting image is 0 bytes, it means that the request did not return an image file.
    - It may be because you are not using the full Azure function endpoint.
    - Check what is the function endpoint URL by going to the Azure Portal, and searching for the Function App service.
        - Go to your function app service -> Functions -> onnx -> Get function URL.
        - NB! Function will not have the same endpoint that we used while testing locally inside the VM
        - It will include an additional `code` component, something like this: `https://functionname.azurewebsites.net/api/onnx?code=5HbZjm8m...`
5. If you get an error about an Unprotected Private key file when using SSH or SCP commands to connect to the VM:
    - It means that the file permissions of the downloaded secret SSH key file should be set lower so other users inside your system do not have permission to see the secret key file.
    - Using the Linux command `chmod 600 PATH_TO_KEY_FILE` should solve this issue
6. Error: Service not available
    - Creating a Resource group, storage account, and function inside a different Cloud Availability zone worked as an alternative.
        - Choose North Europe instead of West Europe.
        - The respective option specifying north europe when creating the function would be: `--consumption-plan-location northeurope`