



Hands-on Workshop (Sandbox based)
Using OCP Virtualization to modernise an Application in slow-time

Step 1 - creating a VM

Log on to the Developer Sandbox (follow instructions on how to create an account if you don't already have one) - <https://console.redhat.com/openshift/sandbox>

Make sure you are in your (username)-dev projects (i.e. for me, utherp0, the project is utherp0-dev)

At the top left of the Ux is a pulldown giving three tabs, Administrator, Developer and Virtualization. For this workshop we will be primarily using the Administrator viewpoint as we will be switching between VMs and other important OpenShift Objects. Select the Administrator viewpoint and click on Virtualization/VirtualMachines.

Wait for the VM tab to render – it will say ‘No VirtualMachines found’. Click on the ‘Create VirtualMachine’ pulldown and select ‘From Template’. You will be taken to the Template Catalog tab.

Find the Tile for ‘Fedora VM’. Click on this tile.

Important – click on the ‘Optional Parameters’ pulldown. Change the CLOUD_USER_PASSWORD to openshiftcommons

Optional – set the VirtualMachine name to something personal. I tend to use ‘fedora-(my name)’.

Click on ‘Quick create VirtualMachine’.

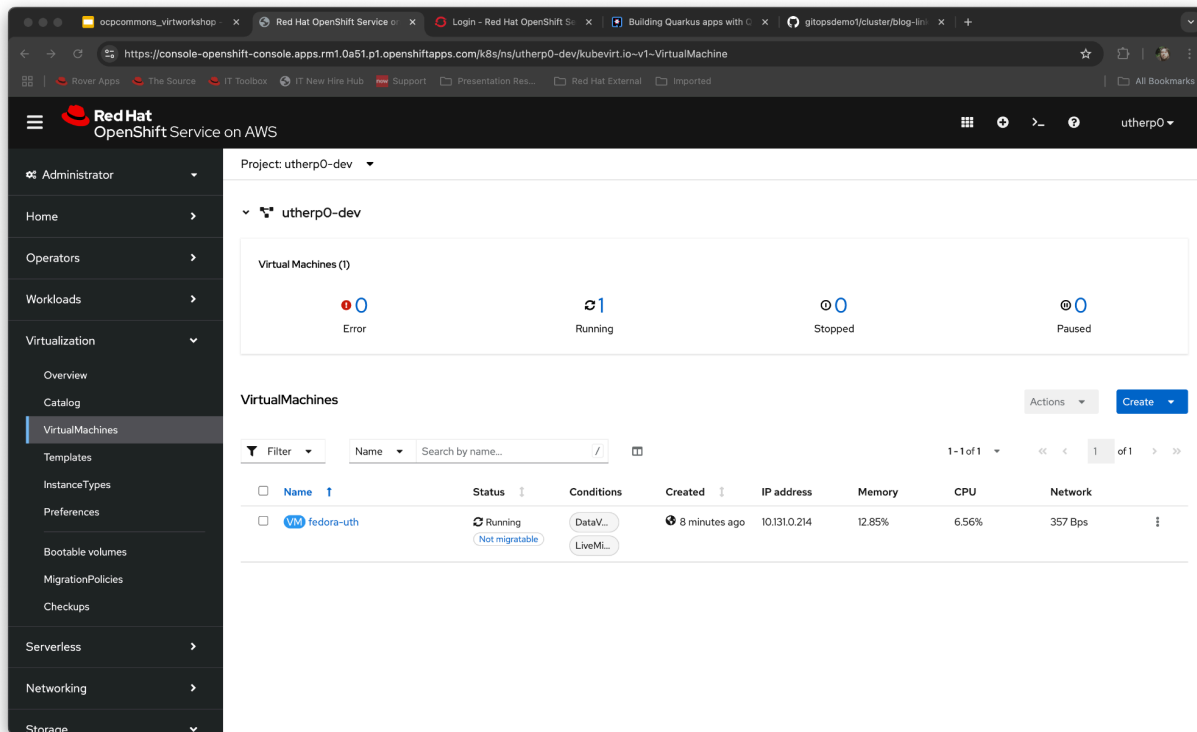
The VM overview tab will appear with the VM in a ‘Stopped’ state. Click on the play button (top right, next to Actions) to start the provisioning process. This may take a little while.

Whilst the VM is provisioning, switch to Storage/PersistentVolumeClaims (left hand navigation menu). You will see a PVC with the name of your VM in there; if the status is ‘Bound’ the VM has been prepared and is in the process of starting.

Now switch to Workloads/Pods (left hand navigation menu). You should see a Pod running here with the name ‘virt-launcher-(your VM name)-(random five characters)’. This is the OpenShift control point for the VM; all VM actions are driven through this Pod, by default this Pod lives on the SDN (as with all other Pods) and also VM metrics are exposed through this Pod. Click on the Pod name to go to the Pod Details page, and click on Metrics. This will show the current system resource consumption of the VM; this is also reachable directly from the VM overview.

Now click on Virtualization/VirtualMachines. This page may take a while to render; Sandbox is a shared system and resources are constrained. In normal OCP Virt systems this page is instantaneous.

Once the page renders you should have a VM listed like this:



Note the 'Not migratable' tag. This is because the storage classes available to Sandbox do not include a ReadWriteMany mode which is needed for live migration. Again, standard OCP Virt installations with capable hardware/CSI drivers will allow this. This is to do with moving the VM from worker node to worker node without turning it off; you need RWX mode so the system can create another control Pod (virt-launcher) while the previous one is running, for seamless handover of the VM between physical machines.

Click on the VM name.

You will see an overview of the VM, including a miniature version of the VM's output. Click on the 'Open web console' link. This will take you to a fullscreen version of the VNC console.

Important – there is a slight issue with the VNC console in that it can only support one connection. You will notice that the VNC console disconnects in around 10 seconds. Switch back to the Sandbox Ux tab and switch to Storage/PersistentVolumeClaims. Once this page has rendered, switch back to the VNC console tab. Now click 'Connect'.

This VNC page should now be stable. If you have further disconnections, simply click on 'Connect' to re-establish the console.

Step 2 - setting up the VM

We are now going to install the 'legacy' applications on our active VM; when using this in the real world, where you are more likely to be importing an active VM with Apps, the Applications will already be there. For this workshop we are going to create some.

Firstly, log on to the VM - the username will be 'fedora' and the password, if you changed it, will be 'openshiftcommons'. Note that the 'Guest login credentials' are shown at the top of the page.

Important - make sure the console has focus, this gets me every time.

When you have logged on perform an update; this is a complete machine, so everything you can do with a Fedora instance you can do with this. You can even install a windowing system, and the VNC console will reflect the XWindows user interface (and if you use Windows, you will see the Windows Ux).

Type:

```
Unset  
sudo dnf update -y
```

The VM will update itself to the latest OS components.

Once it has updated, let's add the Apache httpd webserver. Type:

```
Unset  
sudo dnf install httpd -y
```

Once it has completed, let's make it active and active at boot. Type:

```
Unset  
sudo systemctl start httpd  
sudo systemctl enable httpd
```

Check the httpd server is running by typing:

Unset

```
curl http://localhost
```

Now we are going to add the Quarkus CLI so we can add a simple RESTful microservice to our VM.

Type:

Unset

```
curl -Ls https://sh.jbang.dev | bash -s - trust add  
https://repo1.maven.org/maven2/io/quarkus/quarkus-cli/
```

Important – you can easily copy and paste from this document into the VM; simply highlight the text here, copy it (i.e. cmd-v for Mac), then on the Console page there is a ‘Paste to console’ button. Just remember to re-focus the console page by clicking on it afterwards.

Now type:

Unset

```
curl -Ls https://sh.jbang.dev | bash -s - app install --fresh --force  
quarkus@quarkusio
```

To use the Quarkus functionality we have to restart the Shell. We also have to add a label to the VM itself, to allow us to direct traffic to it using a Kubernetes Service. Switch back to the Sandbox Ux.

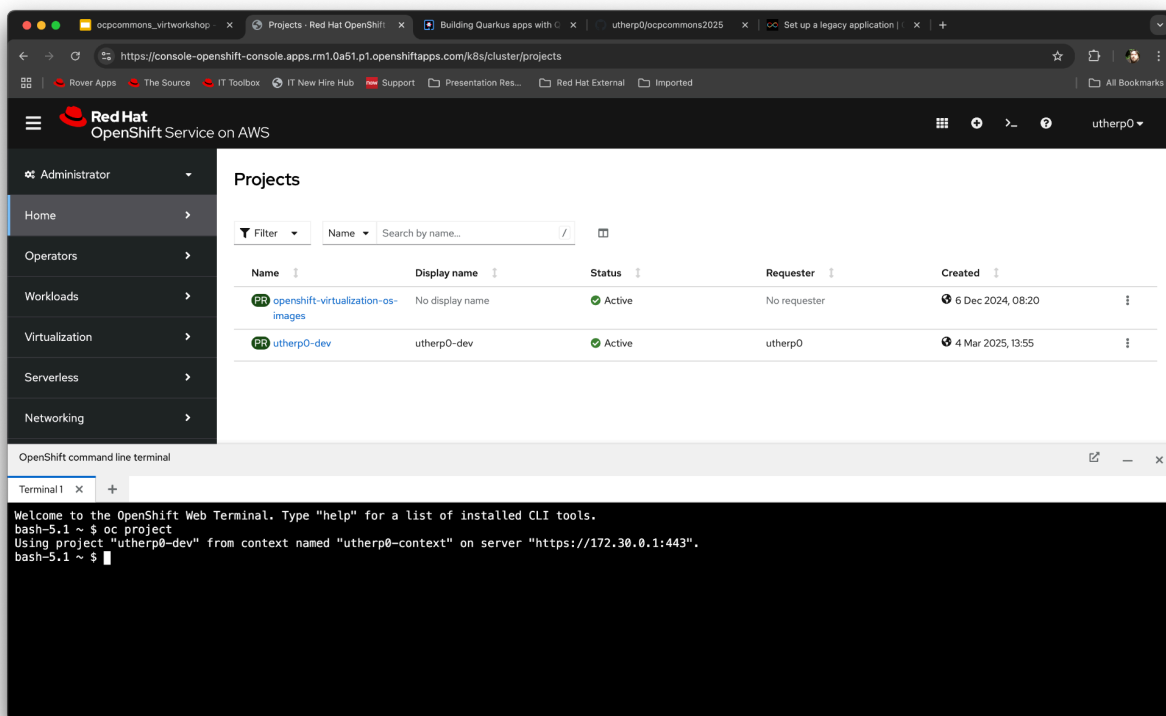
Click on Virtualization/VirtualMachines. Click on the three vertical dots at the far right (this is called a Kebab, which strikes me as amusing). Select ‘Stop’. This pulldown gives you all the VM controls you can push to the VM through the virt-launcher Pod. In this case we have stopped it. Make sure the status of the VM switches to ‘Stopped’.

Click on the VM name. When the overview appears, switch to the YAML tab. This tab allows you to directly change the specification of the VM Object itself. The majority of normal tasks are

available through opinionated components of the VM content page, but it is worth being aware of this facility. Switch back to the VM list by clicking on Virtualization/VirtualMachines.

Now we are going to use an extremely useful addition to the OpenShift Ux - the terminal window. At the top right click on the '>_ ' icon. This will start a terminal window in the bottom half of the webpage.

Once the Terminal window has started, make sure you are in the (name)-dev project:



Now check the state of your VM by typing:

```
Unset
oc get vm
```

It should respond with the name of your VM and a status of 'Stopped'.

We are going to edit the specification of the created VM to add a label that we can use for assigning traffic to using a Kubernetes Service.

If you are old-school like me you will like/tolerate 'vi'. Type:

Unset

```
oc edit vm/(your vm name here)
```

Once in the editor, type:

Unset

```
/domain
```

And hit RETURN. Now press 'n' to go to the next mention of domain. It will take you to this point in the definition:

```
vm.kubevirt.io/os: fedora
vm.kubevirt.io/workload: server
creationTimestamp: null
labels:
  kubevirt.io/domain: fedora-uth
  kubevirt.io/size: small
network.kubevirt.io/headlessService: headless
```

Press 'i' to enter insert mode (it will say "-- INSERT --" at the bottom of the editor). Position the cursor after the word 'small', and press return. Hit 'delete' to move the cursor back in line with the text above (because YAML) then type:

Unset

```
special: myfedoravm
```

Now press [ESC] to get the editor command prompt, then type:

Unset

```
:wq!
```

This will save the VM definition with the new label in it. Click on the '_' icon at the top of the Terminal to compress it, so you can see more of the user interface.

We will now create two services for later use in the workshop. Click on Networking/Services in the left hand navigation. Click on 'Create Service' and replace the YAML in the editor with this:

```
Unset
apiVersion: v1
kind: Service
metadata:
  name: fedorahttp
spec:
  selector:
    special: myfedoravm
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```





Hit 'Create' to add this service. Click on Networking/Services again, and click on 'Create Service' again.

This time replace the YAML with:

```
Unset
apiVersion: v1
kind: Service
metadata:
  name: fedoraquarkus
spec:
  selector:
    special: myfedoravm
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

Hit 'Create' to add this service. Click on Networking/Services and you should see something like this:

Services

Name ▾	Search by name...	
Name ↑	Labels ↓	Pod selector ↓
 fedorahttp	No labels	 special=myfedoravm
 fedoraquarkus	No labels	 special=myfedoravm

Note the 'Pod selector' field.

Now click on Virtualization/VirtualMachines. Allow the list to render. Click on the kebab (the three vertical dots) and choose 'Restart' to reboot the VM. Wait until the status changes to 'Running'.

Note - the Sandbox is a shared resource and to keep it efficient it will auto shutdown the VM after an hour of use; if this is the case 'restart' will be grayed out; choose 'start' to get the VM running.

Now do a quick test (we haven't added the Quarkus yet so we will test the httpd service endpoint). Bring the Terminal back up (if you have iconified it there will be a 'Restore terminal' icon at the bottom right of the Ux).

In the terminal type:

```
Unset
curl http://fedorahttp
```

The terminal is running within the project and therefore has DNS resolution, **by name**, of the Service endpoints. You will see the webpage source from the httpd running in the VM.

Now we are going to add some external storage, from OpenShift, directly into the VM. Click on Storage/PersistentVolumeClaims, then click 'Create PersistentVolumeClaim' and select 'With Form'.

Set the PersistentVolumeClaim name to **appdisk**

Set the Size of the VM to 1Gb

Leave the Volume mode as 'Filesystem' and click 'Create'.

Now switch to the Virtualization/VirtualMachines. Let the list render, and then set the VM to stopped (kebab on the right).

Click on the VM name and then select the 'Configuration' tab. On the left hand side of this content pane click on 'Storage'.

Click on 'Add disk' and select 'Volume' from the drop-down. Leave the Name as generated, click on the PersistentVolumeClaim pulldown and select 'appdisk' (the PVC we have just created). Click 'Save' and the disk will be added to the list available to the VM.

Go back to Virtualization/VirtualMachines and restart the VM. You will notice it briefly waits for the storage to be created and bound before starting.

Now the Linux-y bit.....

Go back to the VNC console. Make sure the Sandbox Ux isn't on the VM page (to avoid disconnects). We are going to format and attach the disk for use; this is very cool as we are mapping a PVC from OpenShift into the VM as a usable piece of disk storage.

Firstly, determine the volume name using:

```
Unset  
sudo fdisk -l
```

It will probably be /dev/vdc - you are looking for the disk with ~1Gb of storage. Now we have to setup the partition table correctly using:

```
Unset  
sudo mkdir /mnt/appdisk  
sudo gdisk /dev/vdc (or the name your disk has from above)
```

In gdisk, hit 'n' for new and then take all the defaults. When it has completed, hit 'q' to exit.

Now format the disk for use with:

Unset

```
sudo mkfs -t ext4 /dev/vdc  
sudo mount /dev/vdc /mnt/appdisk  
sudo chmod 777 /mnt/appdisk
```

Now type:

Unset

```
sudo dnf install git -y
```

This will install git in the VM so we can pull our legacy application components.

Step 3 - the legacy application

We will start by cloning the git repo with our legacy application in it. In the VNC console tab, type:

Unset

```
cd  
mkdir git  
cd git  
git clone https://github.com/utherp0/ocpcommons2025
```

We also need to upgrade the JDK for Quarkus, so type:

Unset

```
jbang jdk install 23  
jbang jdk default 23
```

Now we can set the microservices application going:

Unset

```
cd /home/fedora/git/ocpcommons2025/quarkus/testservices  
quarkus run &
```

Note that for the sake of the workshop this is a transient application; if the VM is restarted you will have to re-run the Quarkus app.

Note - Quarkus takes a little time to start up; watch the log and when the App has settled, do the following. Repeat this test until the App responds (you may see 'failed to connect' while the App is starting up).

Test the application is serving requests:

```
Unset
curl http://localhost:8080/services/heartbeat
```

This will respond with the UTC millisecond time for the VM.

Add the legacy application webpages using:

```
Unset
sudo cp /home/fedora/git/ocpcommons2025/html/* /var/www/html/
```

Finally, we will expose the legacy application endpoints as routes in OpenShift. This abstraction of route to service will allow us to quickly shift the traffic ingress when we modernise the application next.

Switch back to the Sandbox Ux. Click on Networking/Routes. Click on 'Create Route'.

Using the Form view, set the name to **fedorahttp**

On the 'Service' pulldown, select fedorahttp

On the 'Target port' pulldown, select 80->80

Click on 'Secure Route'

Set 'TLS Termination' to Edge

Click 'Create'. This is the external FQDN route into the webserver currently running on the VM. Click on the link and the Fedora Webserver test page will pop-up. Add `"/test1.html"` to the URL bar and press return. This is our legacy web application endpoint.

Go back to Networking/Routes. Click on 'Create Route' again.

Using the Form view, set the name to **fedoraquarkus**

On the 'Service' pulldown, select fedoraquarkus

On the 'Target port' pulldown, select 8080->8080

Click on 'Secure Route'

Set 'TLS Termination' to Edge

Click 'Create'. This is the external FQDN route for our microservices running on the VM. Click on the link and a separate tab will appear saying 'Resource not found'. Add `"/services/heartbeat"` to the URL and press return.

Step 4 - Modernising part of the Application

The most powerful part of this approach is that you can now modernise components that are running on the VM in OpenShift into Containers without interrupting the Application as a whole.

First. Let's test the Application is working. On the Sandbox OCP Ux, go to Networking/Routes. Click on the URL shown for the Route 'fedorahttp'. You will see the Fedora Webserver test page. Now, add `"/test2.html"` to the URL bar and hit return.

This is our testbed for interacting with the microservices. Switch back to the Sandbox Ux and, again, on Networking/Routes, click on the URL shown for the Route 'fedoraquarkus'. You will get a 'Resource not found' page; highlight the route in the URL bar and copy it.

Switch back to the tab with the testbed webpage in it. Paste the Route just copied into the 'Service Route' text box. Now click on 'Heartbeat Service'. The page will render the heartbeat sent from the microservice hosted in the VM.

What we are going to do now is re-host the webserver component in a Container instead of the VM.

Switch back to the Sandbox Ux and switch to the Developer viewpoint (top left of the Ux). This will show the project (i.e. your account-dev) and no resources found (if you are starting from an account where you have been experimenting you may have other Pods in here).

Click on '+Add'. Then the 'All Services' tile. Click on 'Builder Images' in the left hand panel of the Developer Catalog.

Now click on the 'Apache HTTP Server' tile. We are going to build a 'source-2-image' application based on the Apache HTTP with our webpages hosted (from the git repository).

When the overview for Apache HTTP Server appears, click on 'Create'.

When the 'Create Source-to-Image' form appears, enter the following in the 'Git Repo URL' text field:

Unset
`https://github.com/utherp0/ocpcommons2025`

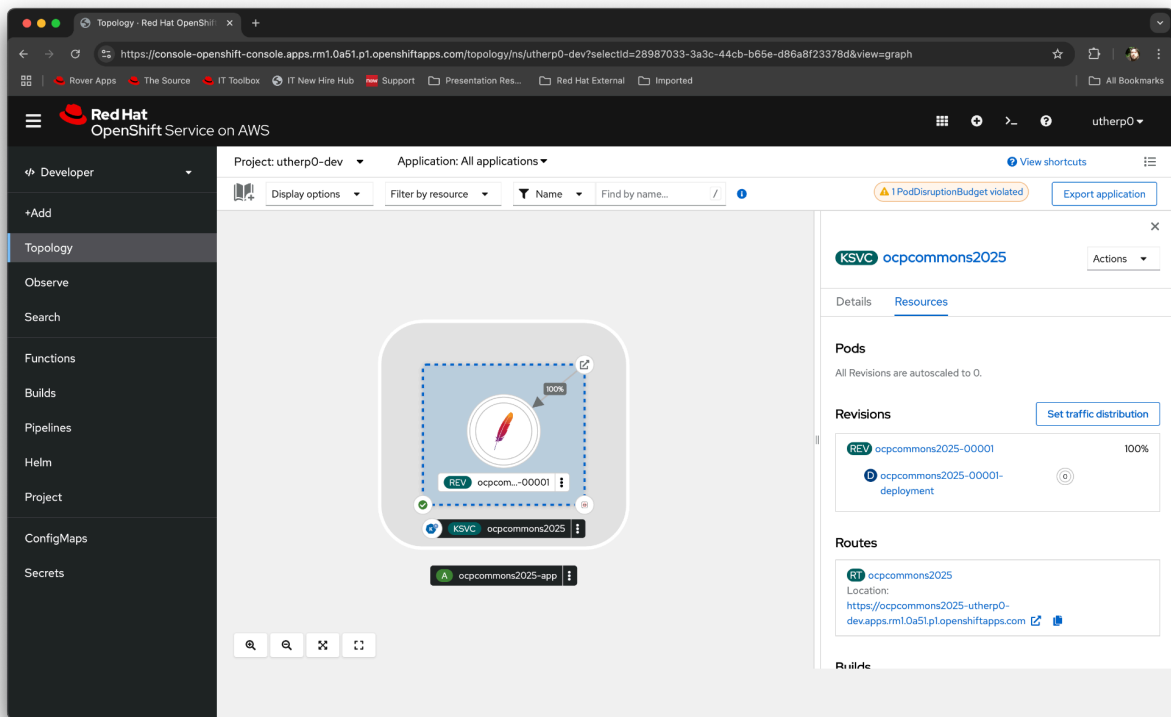
Now click on 'Advanced Git options'.

In the 'Context dir' textbox, add:

Unset
`/html`

Leave everything else as default - we will be deploying the webserver Container as Serverless to make it much more efficient. Click on 'Create' to build the Application.

Eventually the build will complete and the Topology content pane will look like this:



We have a webserver deployed with our web components as a Serverless application; this container will only be active while serving requests.

At the top right of the roundel is an icon which is the Route to this app – click on this and you will get the RHEL test page (this version of the Apache HTTP server is hosted on the RHEL UBI, Universal Base Image).

Add “/test2.html” to the URL for the test page and press return. You will see the web components now hosted as a Container. Switch back to the Sandbox Ux tab and switch to Administrator viewpoint. Click on Networking/Routes. Click on the fedoraquarkus URL again – you will get the ‘Resource not found’ page. Copy the URL from the URL bar.

Switch back to the webserver tab (showing test2.html). Paste the route into the Service Route textbox. Hit ‘Heartbeat Service’.

We now have an OCP hosted version of the webserver.

Step 5 – invisible network switchover

For this final step we will create a static deployment of the webserver on the Sandbox; this is to show the simplicity of switchover from Service to Service. You can do the same thing with Service->Serverless Service but it is slightly more complex.

Go to the Sandbox Ux and switch to the Developer viewpoint (top left of the Ux).

Click '+Add', then 'All Services'.

Click 'Builder Images' and then click the tile for 'Apache HTTP Server'.

Click 'Create'.

Set the 'Git Repo URL' to:

Unset
<https://github.com/utherp0/ocpcommons2025>

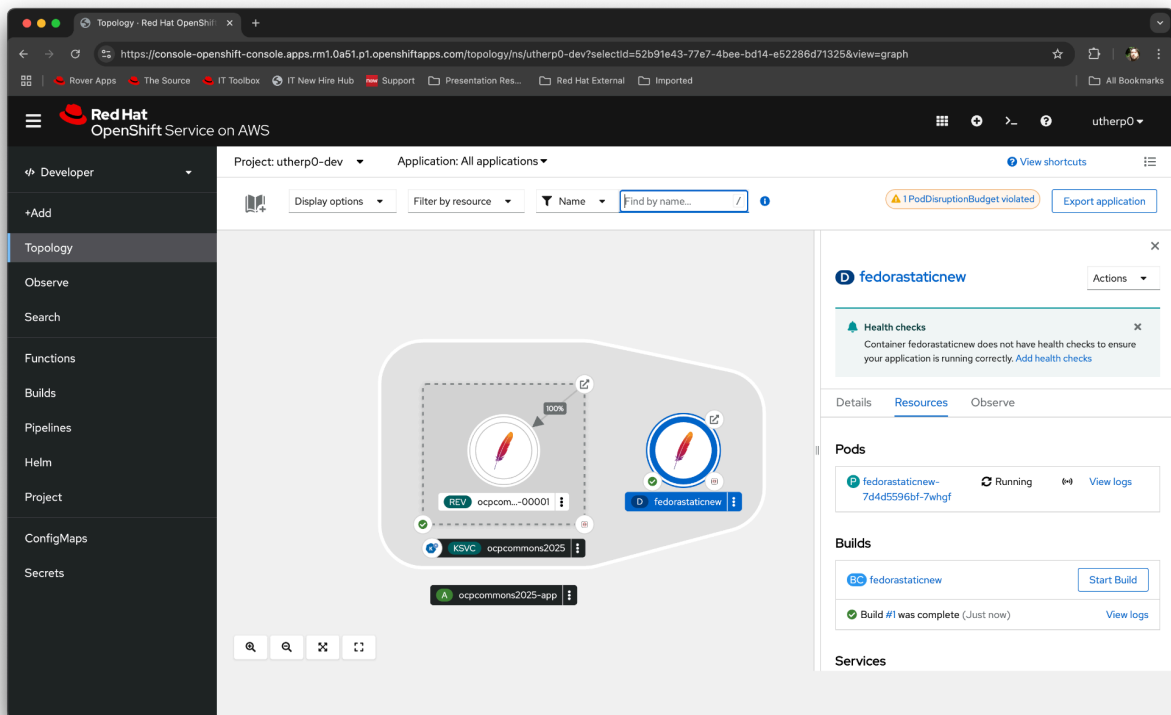
Click on 'Show Advanced Git options'. Set the context dir to /html

Set the 'General/Name' to 'fedorastaticnew'

Change the 'Deploy/Resource type' to 'Deployment'

Hit 'Create'.

We are performing the same source-2-image build as we did for the Serverless component, but instead standing it up as an 'always-on' Deployment. Once the build has completed, the topology page should look similar to:



Click on the Route icon (top right) of the 'fedorastaticnew' roundel. You will get the RHEL Test Page; again, the s2i build creates the webserver on a RHEL UBI rather than Fedora, as with the VM.

Switch back to the Sandbox Ux, and go to the Administrator viewpoint; then click on 'Networking/Routes'.

At this point you should have (at least) three Routes; we are interested in fedorahttp, fedoraquarkus and fedorastaticnew

Click on the URL for 'fedorahttp' - this pushes traffic to the VM hosted webserver (you will see the Fedora Webserver testpage).

Switch back to the Sandbox Ux; whilst still on the Routes page, click on the name 'fedorahttp' and you will be taken to the Route details page. Click on 'Actions' and then choose 'Edit Route'.

In the 'Edit Route' form, click on the 'Service' drop down and select 'fedorastaticnew'. Then click on 'Target port' and select 8080->8080

Click 'Save'. Now go back to the Networking/Routes page and click on the URL for the 'fedorahttp'.

You will now see the RHEL Test Page; this route is now taking you to the Container. From an external perspective the consumers of the URL haven't changed the target, but are now getting webpages from a Container.

To test this, go back to the VM console (if the tab has been closed, go to Virtualization/VirtualMachines in the Administrator viewpoint, click on the VM and then click on the 'Web console' link, remembering to pop back to the Sandbox Ux and switch to another content pane, for instance Networking/Services, to avoid disconnects.

In the VM console, type:

```
Unset
sudo systemctl stop httpd
sudo systemctl status httpd
```

Press 'Q' to get out of the status page; the httpd engine will be offline in the VM (note that if you reboot or restart it will recreate, use 'systemctl disable' to remove it if you want to).

Now, in the VM console, type:

```
Unset
curl http://localhost
```

And you will see the socket is no longer bound.

Switch back to the Sandbox Ux, then Networking/Routes

Click on the URL for 'fedoraquarkus'; when you see the 'Resource not found' window copy the URL

Switch back to the Sandbox Ux, then Networking/Routes

Click on the URL for 'fedorahttp'; you will see the RHEL test page (application is now serving from the Container on the old URL that used to serve from the VM). Add /test2.html to the URL and press return.

Paste the fedoraquarkus URL into the Service Route textbox - this is a direct link to the Quarkus microservices running in the VM.

Now hit 'Heartbeat Service'. You now have the new Application (Containerised) talking to the microservices on the VM.

Summary and next steps

In the workshop we have created a Fedora VM hosted on OpenShift Virtualization. We have added a composite application, web frontend and microservices, and exposed them as reachable services/routes in OpenShift. We have modernised the web frontend into a Container and switched traffic flow without interrupting user experience.

This workshop can be run at any time on Sandbox. If you fancy a challenge, try doing the following:

1. Modernise the Quarkus hosting application using the Java S2I, or using the quarkus CLI directly from the VM to deploy the application as an image (hint, <https://quarkus.io/guides/deploying-to-openshift>)
2. Try converting the storage we added to the VM. It is stored as a PVC within OpenShift, but that PVC contains a disk.img file that is mounted. Come up with a strategy for either a: copying the contents to a PVC filesystem (hint, install 'oc' in the VM and try playing with 'oc rsync' after creating a Pod/PVC or b: options for mounting a disk.img as a file system (hint, loopback)
3. Have a play with Podman and Podman Desktop; in particular, have a look at Podman Desktop bootc plugin, which can build full VM images from container file layer systems, allowing you to store your OS as container images and apply updates using OSTREE. Try creating a bootc VM OS image and uploading it.