**Red Hat Co-op Summary**

* Vivek Shakya , MOC , CSRE Intern

Tasks performed during the Co-op

1. Learned to install OpenStack on a local system - <https://www.rdoproject.org/install/packstack/>
2. Learned Openstack operations and created a Hello-World Document for openstack users.

Google Drive Link - <https://docs.google.com/document/d/151KuguphqrbVYdRXWTZqV1eNSDnCTqz53Hmsm320s4E/edit?usp=sharing>

Github Link - <https://github.com/shakyav/CO-OP-RH-Docs/blob/master/MOC%20Openstack%20tutorial.docx>

1. Learned Ansible to automate openstack operations
2. Explored the methods for Service Health Check for OpenStack
3. Deployed the ELK stack and Heartbeat to monitor the Uptime for Openstack services

Document Link - <https://docs.google.com/document/d/1uClUFu4kQR0chtPXataFSzk_xu9TCh6BWAoMHoVisV8/edit?usp=sharing>

Github Link - <https://github.com/shakyav/CO-OP-RH-Docs/blob/master/Methods%20to%20Check%20the%20Health%20of%20Openstack%20Cloud.docx>

1. Created a Migration Plan for migrating from OpenShift 3.11 to Openshift 4.x

Document Link - <https://docs.google.com/document/d/1P6S_YFuoi9061a043lcTGazQQqGprfNOIxDZHSfFziE/edit?usp=sharing>

<https://docs.google.com/document/d/1WacbG2S8bP-bbdMUYQY37dkOy_8_72VfCsr64UZkdHE/edit?usp=sharing>

Github Link - <https://github.com/shakyav/CO-OP-RH-Docs/blob/master/MOC%20Openshift%20Migration%20from%20Version%203.docx>

Reference Link - <https://www.openshift.com/learn/topics/migration>

1. Learned how to Install OpenShift 4.x on OpenStack

Installation Document with issues - <https://docs.google.com/document/d/1LSbK21TMpgBq6FzO2watBz3XzRi5GpTFHTHv8fS_x3A/edit?usp=sharing>

GitHub Link -<https://github.com/shakyav/CO-OP-RH-Docs/blob/master/Installation%20steps%20for%20openshift%204.docx>

Correct Install Config -

apiVersion: v1

baseDomain: x86-openshift.osh.massopen.cloud

compute:

- architecture: amd64

hyperthreading: Enabled

name: worker

platform:

openstack:

type: c1.xlarge.moc\_openshift

replicas: 3

controlPlane:

architecture: amd64

hyperthreading: Enabled

name: master

platform:

openstack:

type: c1.xlarge.moc\_openshift

replicas: 3

metadata:

creationTimestamp: null

name: k2-shift

networking:

clusterNetwork:

- cidr: 10.128.0.0/14

hostPrefix: 23

machineNetwork:

- cidr: 10.0.0.0/16

networkType: OpenShiftSDN

serviceNetwork:

- 172.30.0.0/16

platform:

openstack:

cloud: kaizen\_oidc

computeFlavor: c1.xlarge.moc\_openshift

externalDNS: null

externalNetwork: external

lbFloatingIP: 128.31.25.27

octaviaSupport: "1"

region: "moc-kzn"

trunkSupport: "1"

publish: External

pullSecret:

sshKey: |

Add your own pull secret and SSh key

1. Issues after installation -

**a). frequent etcd leader changes** - because which cluster gives authorization error very frequently

**b). the cluster goes down completely after a day or two -** node status for control plane nodes changes from ready to not ready and the pod status changes from running to terminated for most of the pods that are required to access the cluster.

Note: Installation also fails while initializing the cluster because of frequent etcd leader changes . Also there are some recommendations from Shift on stack team to resolve the issues

1. Mid Co-op Project - Google-Drive <https://docs.google.com/presentation/d/1zz5aWuVpqpf6_pk15bwjXNK5ckEXL3G_bayyS7l7Jug/edit?usp=sharing>

Github link for the doc - <https://github.com/shakyav/CO-OP-RH-Docs/blob/master/Create%20an%20image%20registry%20and%20a%20knowledgebase%20of%20openshift%20migration%20issues%20for%20MOC%20Users.pptx>

Did not continue with this because of the above issues

1. Curated a Design document for OCP metering dashboard

Document Link - <https://docs.google.com/document/d/1mOJSRsq3q7BuzSiZYS4jjOgdXZZh7HSO2CibzMpIkqk/edit?usp=sharing>

Github link -<https://github.com/shakyav/CO-OP-RH-Docs/blob/master/Automated%20report%20generation%20for%20OPS%20Dashboard.docx>

1. Created the Postman API’s to mock the SSMT backend for data display in the front-end

Let’s assume following the is API that we call to fetch the list of projects:

<http://localhost:8000/list_projects> - this API will return the list of projects running on OCP cluster in the last 1 hour along with the activation time in JSON format.

axios

.get(apiUrl)

.then(res => {

const tableData: Array<dataObject> = [];

res.data.forEach(clusterInfo => {

tableData.push({

namespace: clusterInfo['namespace'],

activationTime: clusterInfo['activation\_time']

});

});

this.setState({ ...this.state, isLoaded: true, clusterData: tableData });

})

.catch(err => {

this.setState({ ...this.state, isLoaded: false, err: err });

});

The above code snippet makes use of axios, it calls the api that I mentioned above and stores the response as ‘res’ . Then we iterate the response to read objects and store in the “tabledata” which is an array of type “dataObject”

Sample data that can be used to save as response for the postman request

[

{

"namespace": "openshift-service-catalog-controller-manager-operator",

"activation\_time": 1

},

{

"namespace": "sample",

"activation\_time": 1

},

{

"namespace": "hello-world",

"activation\_time": 1

}

]

apiUrl = apiUrl + '/' + startDate + '/' + endDate;

Here we are appending the startdate and End date to the base\_url and filter the data based on the date range

1. Learned React and Patternfly 4 for the SSMT-UI application
2. Created dynamic routes for the SSMT-UI application

const AppRoutes = () => (

<LastLocationProvider>

<Switch>

{routes.map(({ path, exact, component, title, isAsync }, idx) => (

<RouteWithTitleUpdates

path={path}

exact={exact}

component={component}

key={idx}

title={title}

isAsync={isAsync}

/>

))}

<Route exact path="/projectlist/:projectId" component={ProjectDetail} />

<PageNotFound title="404 Page Not Found" />

</Switch>

</LastLocationProvider>

);

To add any dynamic route you can add an entry like this <Route exact path="/projectlist/:projectId" component={ProjectDetail} /> in routes.tsx file

Here /:projectId is the unique identifier based on which backend will fetch you the results.

So basically what above line of code does is that , it renders the ProjectDetail component whenever the application encounters a route in /projectlist/:projectId format

Following is the code snippet from project\_detail.tsx

let { projectId } = useParams(); // this line is fetching the //project name from the route url which we passed as one of the //parameters

//console.log(projectId)

// following code is just making an API call and adding the //projectId to the API, hence it will fetch the details of the //projectID or name that we append to the API url

React.useEffect(() => {

axios.get("https://7303ef19-853c-4092-a338-05502ff1eecd.mock.pstmn.io/projectlist/" + projectId).then(res => {

cluster\_data = res.data;

cluster\_data.map((item: { namespace: any; }) => {

console.log(typeof (item.namespace))

setName(item.namespace)

setNode(item.node)

setStart(item.period\_start)

setEnd(item.period\_end)

setPod(item.pod)

setCpusage(item.pod\_usage\_cpu\_core\_seconds)

})

}).catch(err => {

if (err.response) {

//console.log(err.response + "--" + err.message)

}

else {

// console.log(err.message)

}

});

}, []);

1. Worked on user based login , Following code may help other developers to implement the user based login along with backend :

Once we submit the login form :

Use this code inside the handlesubmit() function to pass username and password to the API

axios.post("http://localhost:8000/login", {

userName,

password

}).then(result => {

if (result.status === 200) {

//setAuthTokens(result.data);

this.setState({isLoggedIn: true})

}

})

API can be different it doesn’t have to be the same as shown in the above code.

The above strategy works if the API is returning an Authentication token, but the issue with that is the authentication token expires after sometime so passing one authentication token may not be a good way to go ahead.

Another way is to pass the user credentials with every API call we make to fetch any data and verify the credentials in the backend each time.

For both the methods we have to create a Session context or Authentication context in the react app.

Following blog maybe of help when we plan to implement the user login.

<https://medium.com/better-programming/building-basic-react-authentication-e20a574d5e71>

1. Staging Environment -

Deployed the OCP 4.x cluster but it wasn’t stable as mentioned earlier

Deployed the CRC on one of the OpenStack VM’s but encountered access issues.

Try to follow the below documentation for enabling access to CRC running on a remote system.

<https://gist.github.com/sub-mod/a22b6e38f12d0df95b81261f8604605d#on-the-remote-host>

1. Explored the possible options for CI/CD pipeline that could be implemented for the OCP metering dashboard.

Installed the Jenkins server and integrated it with the push action for the SSMT-UI repo, but then decided to use GITHUB ACTIONS for creating the CI pipeline.