Lab 07 Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform

Contents

7.1 Introduction

In this lab exercise, we will deploy a cloud native application to an OpenShift cluster using the Tekton pipeline.

This is "**Lab 07 – Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform"** from an IBM Cloud Pak for Applications & App Modernization Proof of technology (PoT). The labs are not required to be executed in order. And, you may skip labs, and only perform the labs that suit your desired learning objectives.

The full set of labs in the PoT are:

Lab01 - Getting started with Docker
Lab02 - Explore RedHat OpenShift Container Platform
Lab03 - Getting started with Kubernetes
Lab04 – Liberty application deployment using Operators
Lab05 – IBM Cloud Pak for Applications - App Modernization using Transformation Advisor
Lab06 – App Modernization with Java EE Microservices and Liberty
Lab07 – Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform

This is a step by step guide to walk you through a quick example of how to create a Tekton pipeline to automate the build, push, and deploy a simple Node.js application on OpenShift.

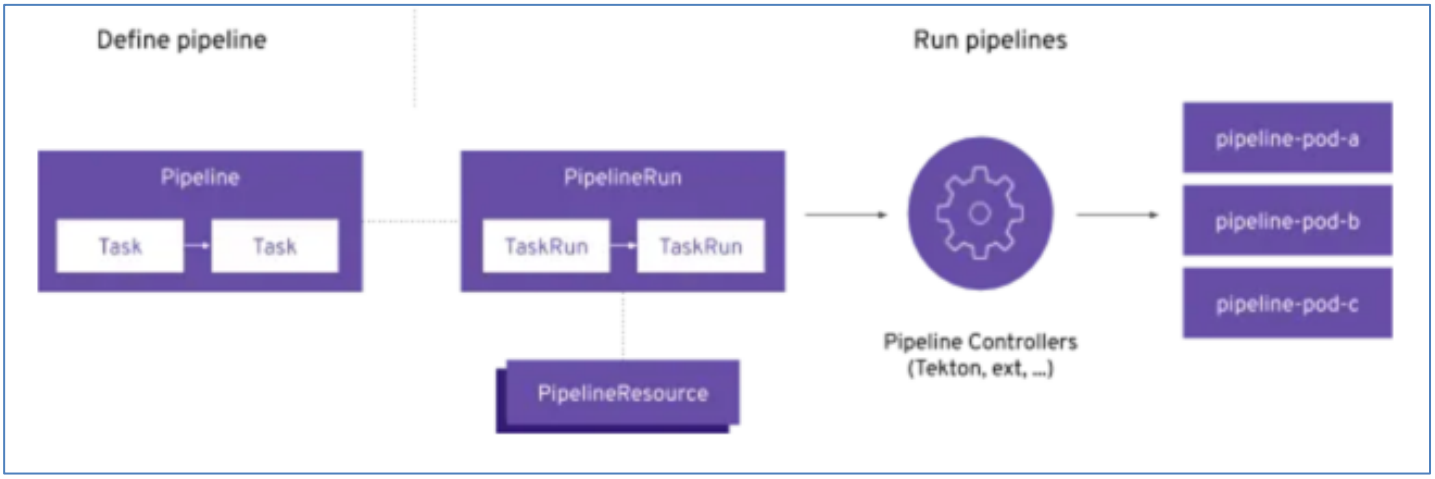
This example uses **Buildah** as the docker build engine. There are other options for the docker build engine, so it should be noted that this is not the only way to accomplish this task.

7.2 What is all this Tekton stuff?

Tekton defines a set of Kubernetes custom resource definitions (CRD) as standard constructs for creating Continuous Integration and Continuous Delivery (CI/CD) pipelines.

The following is a brief introduction to the Tekton CRDs.

- **Task**: A sequence of commands (steps) that are run in separate containers in a pod
- **Pipeline:** A collection of tasks that are executed in a defined order
- **PipelineResource:** Inputs (e.g. git repo) and outputs (e.g. image registry) to a pipeline
- **TaskRun:** Runtime representation of an execution of a task
- **PipelineRun:** Runtime representation of an execution of a pipeline



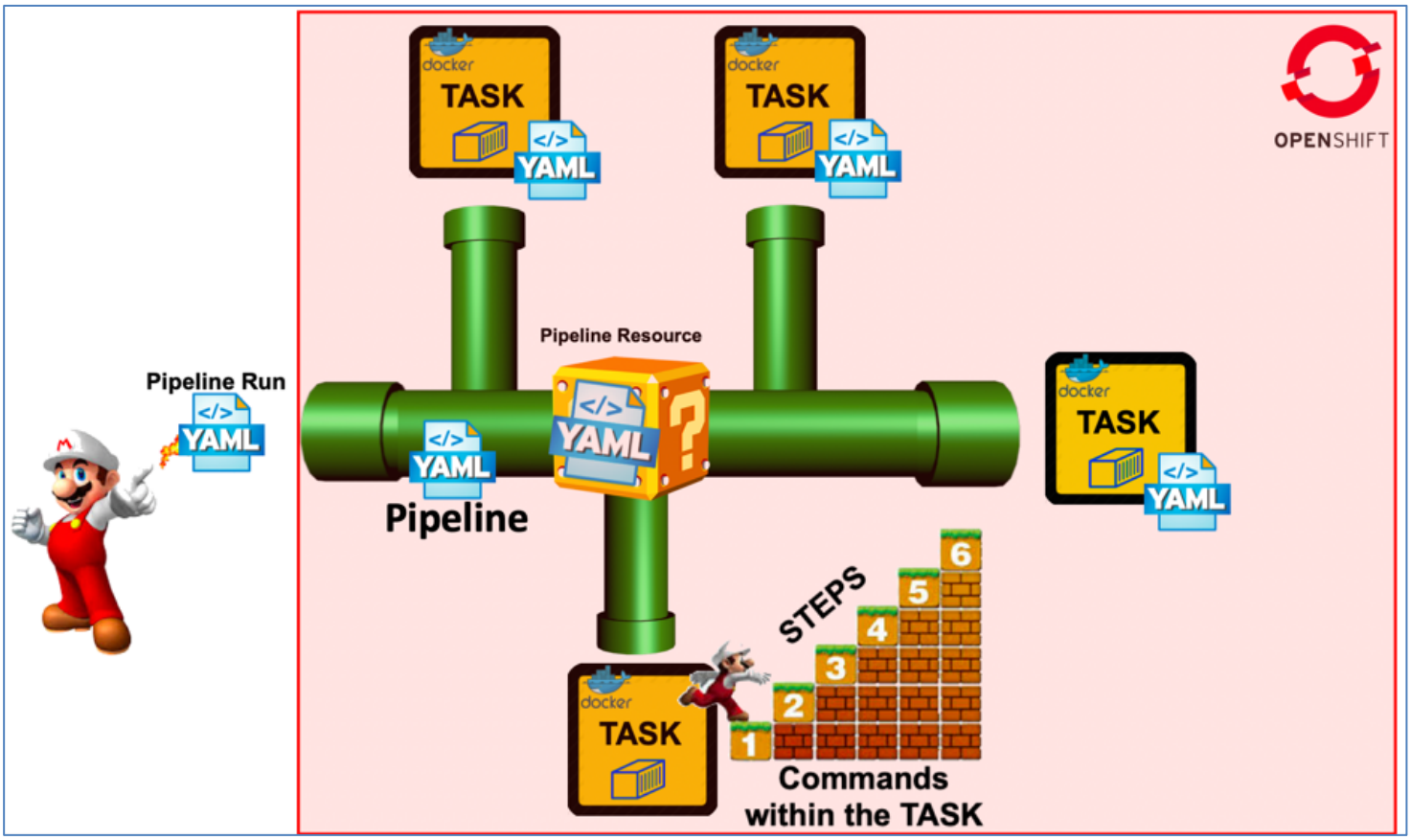
Let's look into a bit more detail what makes up a Tekton Pipeline. As explained above, all objects within a Tekton pipeline are Kubernetes objects.

Pipelines have **tasks**, which are actually a CRD that runs a container.

Within the **task** you define **steps**, which are commands that you will run inside the container.

Pipelines normally have **resources** associated with them, which can be accessed by all tasks within that pipeline.

It should be noted that tasks can be used within multiple pipelines, so it's good practice to use pipeline resources to define the resources used, such as GitHub repositories or docker hub image definitions.

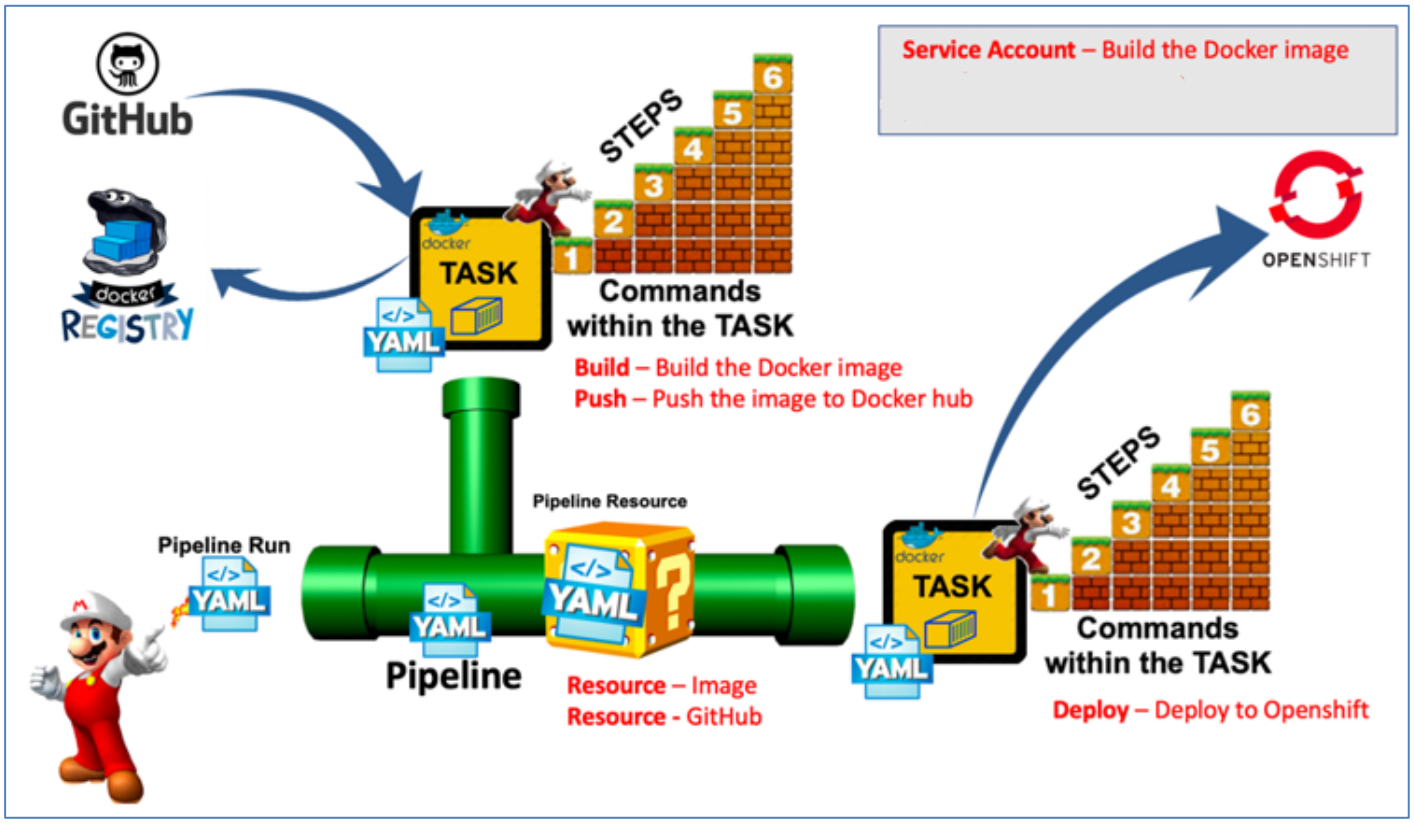

docker
TASK
YAML
docker
TASK
YAML
OPENSHIFT
Pipeline Run
YAML
Pipeline Resource
YAML
YAML
Pipeline
docker
TASK
YAML
STEPS
6
5
4
3
2
1
docker
TASK
Commands
within the TASK

7.3 What exactly are we building here?

Here, you will deploy a Tekton Pipeline along with Pipeline Resources, and two Task objects. The pipeline will pull your source code from GitHub and build the Docker image. Once the image is built, the image is pushed to a local Image repository in OpenShift. Lastly, the pipeline runs the task that deploys this containerized application to the OpenShift (Kubernetes) runtime.

There are a few things you will need to configure along with the pipeline, such as secrets and a service account. The lab guides you through all the steps, but you should take some time to learn more about the security roles that are associated with your service account which allow the service account user to push images to the OpenShift registry, and execute the pipeline resources

Here is a diagram of what you are going to build in this lab.



This is how I learned how to setup security roles and running deployments in TASKS. This is not required reading, but I do recommend you review this article. It has some good tips.

https://medium.com/@jerome_tarte/first-pipeline-with-tekton-on-ibm-cloud-pak-for-application-e82ea7b8a6b1

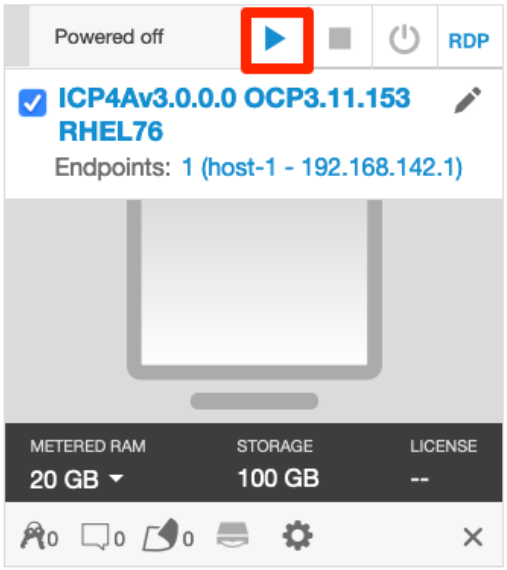
7.4 Lab Tasks

7.4.1 Let's get started

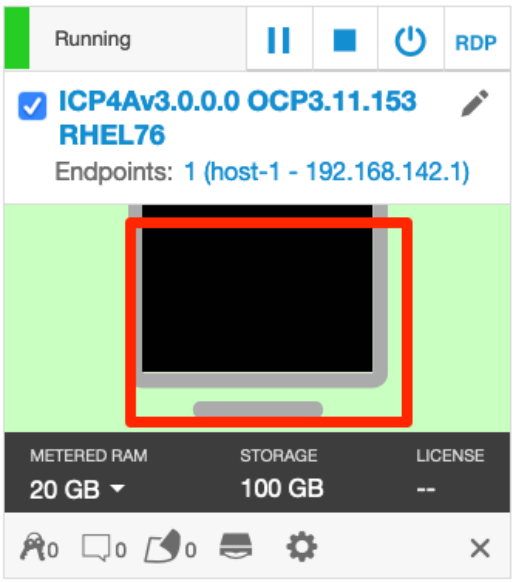
First, launch the lab environment and login to the VM.

On your laptop/workstation, locate the `ICP4Av3.0.0.0 OCP3.11.153 RHEL76` virtual machine

__1. The VM should already be running. If not, Launch the Lab environment by clicking the **Run this VM** icon.

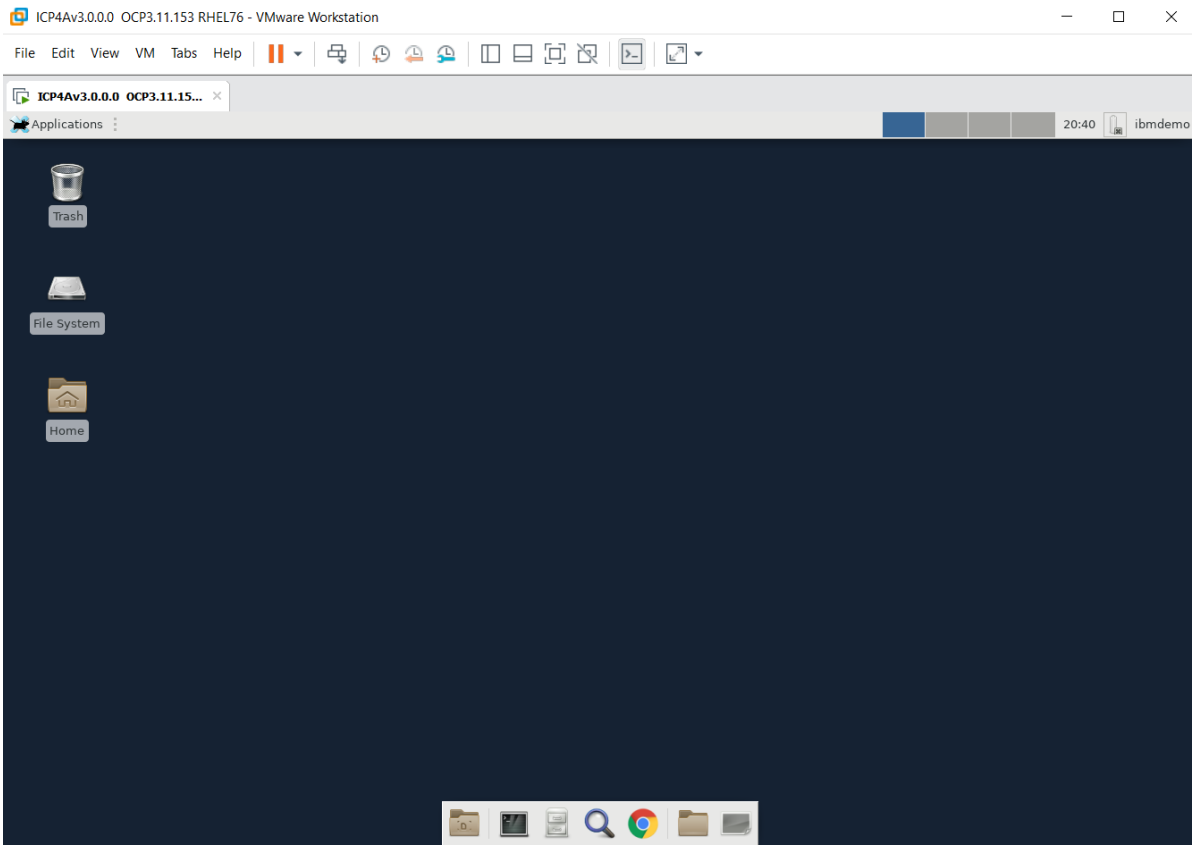


__2. After the VM is running, click its icon to access the VM's desktop.



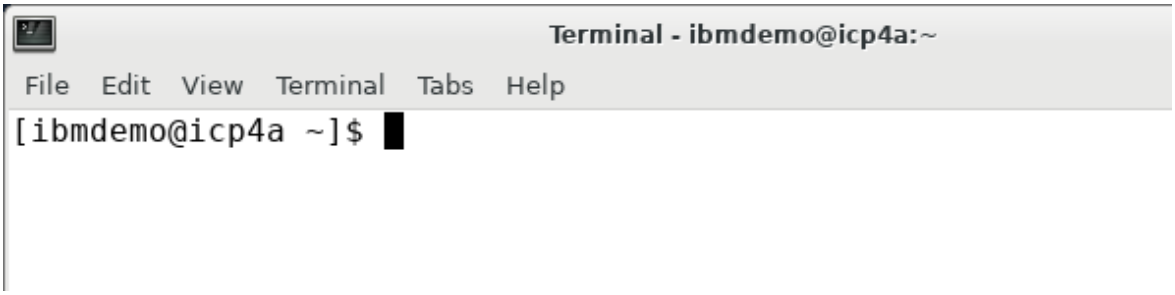
__3. After the VM machine powers on, log with the `ibmdemo` user using the password `passw0rd`.

The `ICP4Av3.0.0.0 OCP3.11.153 RHEL76` virtual machine running and its Desktop is displayed in a web browser window.



__4. Click `Terminal` from the bottom of the desktop to open a command line terminal.

You'll be running in the terminal as the user `ibmdemo`



7.4.2 Clone the Git repository used for this lab and explore the contents

__1. Clone the **tekton1-lab** GitHub repository to the local VM.

__a. From the terminal window, run the following commands to clone the repo:

```
cd ~/student
git clone https://github.com/kpostreich/tekton1-lab.git
cd tekton1-lab
```

```
[ibmdemo@icp4a ~]$ cd ~/student
[
ibmdemo@icp4a student]$ git clone https://github.com/kpostreich/tekton1-
lab.git
Cloning into 'tekton1-lab'...
remote: Enumerating objects: 76, done.
remote: Counting objects: 100% (76/76), done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 76 (delta 9), reused 71 (delta 7), pack-reused 0
Unpacking objects: 100% (76/76), done.

ibmdemo@icp4a student]$cd tekton1-lab
[ibmdemo@icp4a tekton1-lab]$
```

These commands above clone the public repo named **tekton1-lab** to the local directory under **/home/ibmdemo/student/tekton1-lab** directory.

__b. List the directory contents using the "ls" command

You will find the following key resources:

- **Dockerfile** – Used to build the NodeJS Express Application
- **app.js** – The NodeJS Application
- **tekton-pipeline** (folder) – YAML files to create the Pipeline resources for this lab

```
[[ibmdemo@icp4a tekton1-lab]$ ls
app.js  Dockerfile    package-lock.json  readme-images  routes           views
bin     package.json  public             README.md      tekton-pipeline
[ibmdemo@icp4a tekton1-lab]$
```

In the GitHub repo, you will find all the YAML files in the **tekton-pipeline** sub folder.

__2. Enter "`cd tekton-pipeline`" then type "`ls`" to go to the lab directory and list the contents

In the **tekton1-lab/ab/tekton-pipeline** directory, you will find all the YAML files needed to create the Tekton pipeline resources to build and deploy a simple NodeJS Express application to OpenShift.

```
[ibmdemo@icp4a tekton1-lab]$ cd tekton-pipeline

[ibmdemo@icp4a tekton-pipeline]$ ls

deployment.yaml    img-resource.yaml  pipeline-run.yaml  pv.yaml
service.yaml  task.yaml
git-resource.yaml  oc-deploy.yaml     pipeline.yaml      service-account.yaml
taskRun.yaml  Templates
```

You will find the following key resources:

- **service- account.yaml** – Creates a new OpenShift Service Account (functional User) that is used to run the pipelines and access the OpenShift Image registry, and deploy the application to OpenShift
- **pv.yaml** – Creates a persistent volume used by the pipeline to store data
- **git-resource.yaml** – Creates the Pipeline resource that references the input GitHub repo that contains the source for the application to be built and deployed via the pipeline
- **image-resource.yaml** – Creates the Pipeline resource that references the output Docker image registry where the Docker image will be pushed via the pipeline
- **task.yaml** – Creates the build and push Tekton tasks
- **pipeline.yaml** – Creates the pipeline that invokes the tasks defined
- **oc-deploy.yaml** – Creates the Tekton deployment Task to deploy the application to OpenShift
- **deployment.yaml** – Invoked by the oc-deployment task to create the OpenShift Deployment for the application
- **service.yaml** - Invoked by the oc-deployment task to create the OpenShift Service for the application
- **pipeline-run.yaml** – Runtime execution of the pipeline to build and deploy the app

7.4.3 Login to OpenShift and create a new project for this lab

__1. Type `oc login` to login to OpenShift. Use `ocpadmin` for the username and `ocpadm1n` (note the "1", not "i") for the password

```
ibmdemo@icp4a]$ oc login
Authentication required for https://icp4a.pot.com:8443 (openshift)
Username: ocpadmin
Password:
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

  * default
    istio-system
    ta
  Truncated output
Using project "default".
```

__2. Type "`oc new-project tekton-lab`" which will create a new project named tekton-lab, and switch your context to that project

Note: Ensure you create the new project with the name "**tekton-lab**". Otherwise, you will be required to review and modify all YAML files that reference this OpenShift project (Namespace), prior to running the YAML files to create the pipeline resources.

```
[ibmdemo@icp4a tekton1-lab]$ oc new-project tekton-lab

Now using project "tekton-lab" on server "https://icp4a.pot.com:8443".

You can add applications to this project with the 'new-app' command. For
example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
[ibmdemo@icp4a tekton1-lab]$
```

7.4.4 Create an OpenShift Service Account and its security contexts / Roles

It is a good OpenShift practice to create a service account for your applications. A service account provides an identity for processes that run in a Pod.

In this step we will create a new service account with the name **"tekton-sa**".

__1. Create a new service account names **tekton-sa** in the **tekton-lab** project

```
oc create serviceaccount tekton-sa -n tekton-lab
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create serviceaccount tekton-sa -n tekton-
lab
serviceaccount/tekton-sa created
[ibmdemo@icp4a tekton-pipeline]$
```

__2. Add Privileged access to the Service Account required to run pipelines and deploy apps to OpenShift

```
oc adm policy add-scc-to-user privileged -n tekton-lab -z tekton-sa
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc adm policy add-scc-to-user privileged -n
tekton-lab -z tekton-sa

scc "privileged" added to: ["system:serviceaccount:tekton-lab:tekton-sa"]
[ibmdemo@icp4a tekton-pipeline]$
```

The **tekton-sa** Service Account needs privileged access because the pipeline will be creating pods when it runs and it needs this authority to create the pods.

NOTE: The "-n" and "-z" params on this command are in reference to the namespace and service account name.

__3. Add "**Edit**" role to the Service Account to allow for deployments to OpenShift

```
oc adm policy add-role-to-user edit -n tekton-lab -z tekton-sa
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc adm policy add-role-to-user edit -n
tekton-lab -z tekton-sa

role "edit" added: "tekton-sa"
[ibmdemo@icp4a tekton-pipeline]$
```

The **tekton-sa** Service Account requires the EDIT role so that it has the proper authority to make the deployment. This happens within the deployment task during the pipeline execution.

__4. Add **system:image-builder** Role to allow the Service Account to push images to the mage registry.

The pipeline build pods require the **system:image-builder** role, which allows pushing images to any image stream in the project using the internal Docker registry.

```
oc adm policy add-role-to-user system:image-builder -n tekton-lab -z tekton-sa
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc adm policy add-role-to-user system:image-
builder -n tekton-lab -z tekton-sa

role "system:image-builder" added: "tekton-sa"
[ibmdemo@icp4a tekton-pipeline]$
```

7.4.5 Create Secret with Login Token for the Service Account

Next, create a NEW Kubernetes secret with the login token for the **tekton-sa** Service Account.

This is needed by the account for an automated login for the deploy task of the pipeline.

The **first command** extracts the token from the “tekton-sa-token” secret and store it in a file (**token.txt**).

The **second command** creates a new secret using that token. The deploy task will use the token within this secret to login and issue the deploy command during the pipeline.

__1. Run the following commands to create the new secret for the service account to login to OpenShift while running the Pipeline.

__a. Get the token from the service account and store it in a file

```
oc get secret $(oc get secret -n tekton-lab | grep tekton-sa-token |
head -1 | awk '{print $1}') -n tekton-lab -o jsonpath="{.data.token}"
| base64 -d > token.txt
```

__b. Verify the token was written to the **token.txt** file

```
cat token.txt
```

```
[[ibmdemo@icp4a tekton-pipeline]$ cat token.txt
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50I
iwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJ0ZWt0b24tbGFiIiwia3V
iZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6InRla3Rvbi1zYS10b2tlbi1ia
21tcyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJ
0ZWt0b24tc2EiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51a
```

__c. Create a new secret with the token extracted from the service account

```
oc create secret generic tekton-lab-deployer-secret --from-
literal=user=sa --from-file=token=token.txt -n tekton-lab
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create secret generic tekton-lab-deployer-
secret --from-literal=user=sa --from-file=token=token.txt -n tekton-lab

secret/tekton-lab-deployer-secret created
     [ibmdemo@icp4a tekton-pipeline]$
```

__d. Verify the new secret

```
oc describe secret tekton-lab-deployer-secret
```

```
[[ibmdemo@icp4a tekton-pipeline]$ oc describe secret tekton-lab-deployer-
secret
Name:         tekton-lab-deployer-secret
Namespace:    tekton-lab
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
token:  858 bytes
user:   2 bytes
[ibmdemo@icp4a tekton-pipeline]$
```

7.4.6 Create the Tekton “PipelineResources” for the applications build and deployment

Next, you will define two **PipelineResources** to be used by the Tekton pipeline

- **git-resource.yaml** creates a Tekton **PipelineResource** identifying the GitHub **repository** from which the pipeline will pull its data during a build.
- **img-resource.yaml** creates a Tekton **PipelineResource** identifying the **image location**. The tag for that image must be changed every time the application is updated, and the pipeline executed.

__1. Ensure the **Terminal** window is in the **/home/ibmdemo/student/tekton1-lab/tekton-pipeline** directory, where the Pipeline YAML files are located.

```
cd /home/ibmdemo/student/tekton1-lab/tekton-pipeline
```

__2. Review the contents of **git-resource.yaml**

```
cat git-resource.yaml
```

- The name of the PipelineResource is **tekton1-git**
- The source **type** is “**git**”
- The **url** to the source git repo is defined in the “**url**” parameter.

Note: Do NOT MODIFY the YAML for this lab!

```
[ibmdemo@icp4a tekton-pipeline]$ cat git-resource.yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: tekton1-git
spec:
  type: git
  params:
    - name: revision
      value: master
    - name: url
      value: https://github.com/kpostreich/tekton1-lab.git
[ibmdemo@icp4a tekton-pipeline]$
```

__3. Review the contents of **img-resource.yaml**

```
cat img-resource.yaml
```

- The name of the PipelineResource is **tekton-image**
- The **type** is "**image**"
- The **url** parameter defines the location of the image registry where the built image will be pushed during the execution of the pipeline.

Note: Do NOT MODIFY the YAML for this lab!

```
[ibmdemo@icp4a tekton-pipeline]$ cat img-resource.yaml

apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: tekton1-image
spec:
  type: image
  params:
    - name: url
      value: docker-registry.default.svc:5000/tekton-lab/tekton1:latest
[ibmdemo@icp4a tekton-pipeline]$
```

__4. Run the following commands to create the PipelineResources using the YAML files

```
oc create -f git-resource.yaml

oc create -f img-resource.yaml
```

```
[[ibmdemo@icp4a tekton-pipeline]$ oc create -f git-resource.yaml

pipelineresource.tekton.dev/tekton1-git created

[ibmdemo@icp4a tekton-pipeline]$ oc create -f img-resource.yaml

pipelineresource.tekton.dev/tekton1-image created
[ibmdemo@icp4a tekton-pipeline]$
```

__5. List the new PipelineResources

```
oc get pipelineresources
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc get pipelineresources
NAME             AGE
tekton1-git      2m
tekton1-image    2m
[ibmdemo@icp4a tekton-pipeline]$
```

7.4.7 Create an OpenShift (Kubernetes) persistent volume for the Tekton Tasks to store its data while executing the pipeline

The Tekton **PipelineRun** request storage through a Persistent Volume Claim (PVC). The PVC is backed by a Persistent Volume (PV).

In the lab environment, the PV is created using the **pv.yaml** file. This PV is defined as HostPath, and references **/var/lib/containers** path where the privileged Service Account has access.

The Tekton **Task** that you will create later in the lab references this volume for storage during the build steps.

__1. Review the **pv.yaml** file that is used to create the persistent volume that Tekton Task requires

```
cat pv.yaml
```

```
[ibmdemo@icp4a tekton-pipeline]$ cat pv.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv0001
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/var/lib/containers"
  persistentVolumeReclaimPolicy: Retain
[ibmdemo@icp4a tekton-pipeline]$
```

__2. Run the **pv.yaml** to create the persistent volume defined above, and verify it is created as expected

```
oc create -f ./pv.yaml

oc get pv pv0001
```

```
ibmdemo@icp4a tekton-pipeline]$ oc create -f ./pv.yaml

persistentvolume/pv0001 created
[ibmdemo@icp4a tekton-pipeline]$

[ibmdemo@icp4a tekton-pipeline]$ oc get pv pv0001
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM    STORAGECLASS   REASON
AGE
pv0001   5Gi        RWO            Retain           Available                           2m
[ibmdemo@icp4a tekton-pipeline]$
```

7.4.8 Create a Tekton Task to build the Docker image, and push the image to the OpenShift Image Registry

Recap:

You have created the PipelineResources, which define the input and output for the build.

You have created a Service account with the proper privileges and roles, and credentials to run the pipeline, push images to the image registry, and deploy pods to OpenShift.

Now you are ready to create the actual Tekton **Task**, with two steps:

- **build** the docker image from the source GitHub repo
- **pushes** the image to the OpenShift image registry

Of course, there are several ways to build a docker image inside a task (docker command, kaniko …).

For this lab, you will use **buildah**. (https://buildah.io/)

Buildah is a command-line tool for building Open Container Initiative-compatible (that means Docker- and Kubernetes-compatible, too) images quickly and easily.

Buildah is easy to incorporate into scripts and build pipelines.

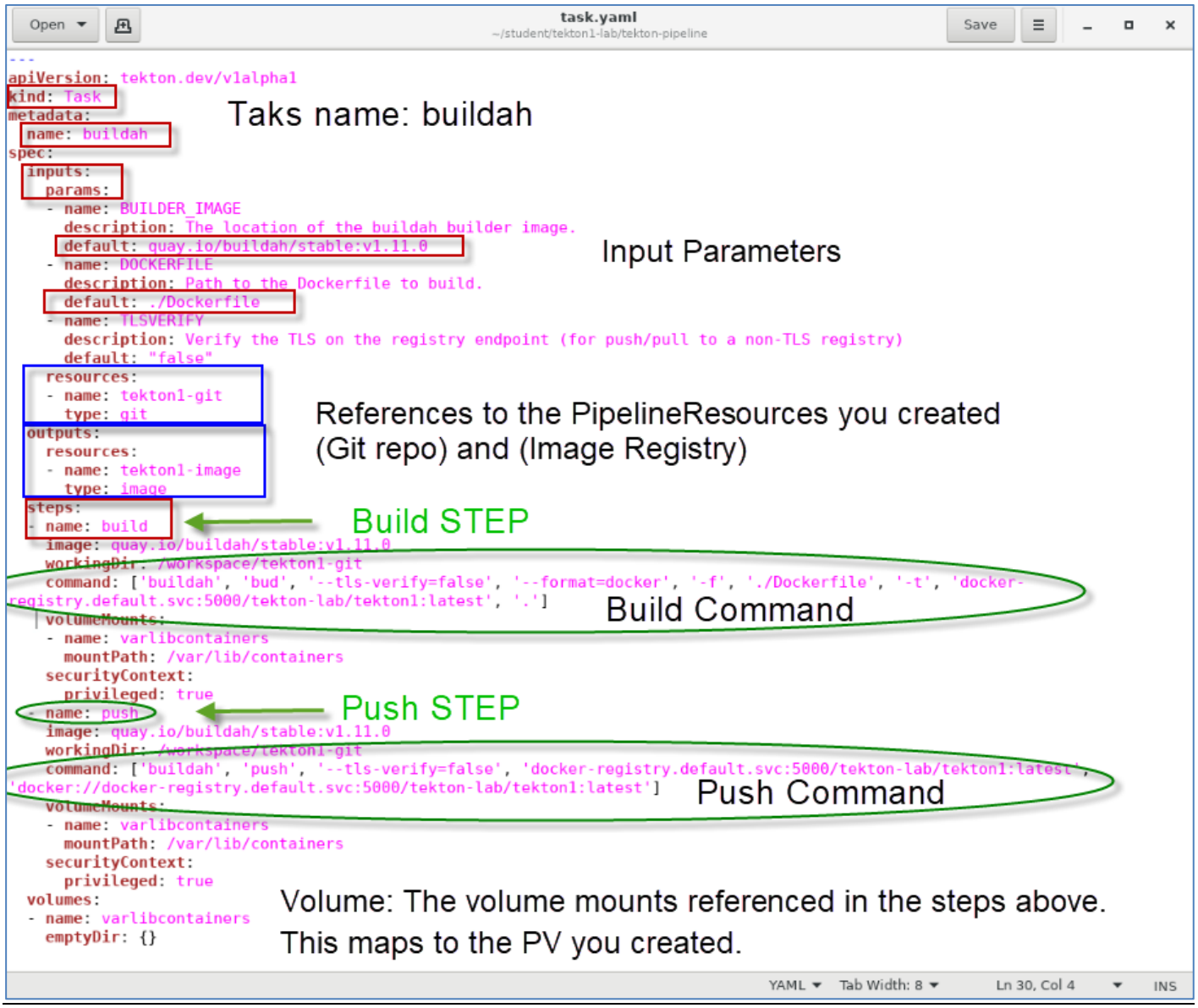
__3. Review the **task.yaml** file using gedit command. **DO NOT MODIFY THE FILE**

```
gedit task.yaml
```

The Task resource defines its spec:

- The input resource
- The output resource
- Local parameters used during the execution of the task
- Steps. In this example, “build” and “push” is described
- The image used for the task execution. here, it is quay.io/buildah/stable.
- Its environment. The env variables are defined, based on configmap and/or *secret*. Here, a *secret* is used to define the authentication information.
- The commands to execute in the “build” and “push” steps. The first one builds the image, the second pushes it to the target repository.

In general, steps are used to isolate individual commands, and illustrated below.




```
---
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: buildah
spec:
  inputs:
    params:
    - name: BUILDER_IMAGE
      description: The location of the buildah builder image.
      default: quay.io/buildah/stable:v1.11.0
    - name: DOCKERFILE
      description: Path to the Dockerfile to build.
      default: ./Dockerfile
    - name: TLSVERIFY
      description: Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)
      default: "false"
    resources:
    - name: tekton1-git
      type: git
  outputs:
    resources:
    - name: tekton1-image
      type: image
  steps:
  - name: build
    image: quay.io/buildah/stable:v1.11.0
    workingDir: /workspace/tekton1-git
    command: ['buildah', 'bud', '--tls-verify=false', '--format=docker', '-f', './Dockerfile', '-t', 'docker-
registry.default.svc:5000/tekton-lab/tekton1:latest', '.']
    volumeMounts:
    - name: varlibcontainers
      mountPath: /var/lib/containers
    securityContext:
      privileged: true
  - name: push
    image: quay.io/buildah/stable:v1.11.0
    workingDir: /workspace/tekton1-git
    command: ['buildah', 'push', '--tls-verify=false', 'docker-registry.default.svc:5000/tekton-lab/tekton1:latest',
'docker://docker-registry.default.svc:5000/tekton-lab/tekton1:latest']
    volumeMounts:
    - name: varlibcontainers
      mountPath: /var/lib/containers
    securityContext:
      privileged: true
  volumes:
  - name: varlibcontainers
    emptyDir: {}
```

__a. **Close the Gedit editor** when you have finished reviewing the contents**.**

DO NOT SAVE ANY CHANGES!

__4. Create the Task, using the task.yaml file, then list the new “buildah” task.

```
oc create -f ./task.yaml
oc get tasks
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create -f ./task.yaml

task.tekton.dev/buildah created

bmdemo@icp4a tekton-pipeline]$ oc get tasks
NAME       AGE
buildah    1m
```

7.4.9 Create the Deployment Task

To manage the deployment of this simple Node.js Express application, tasks are needed to specify a **Deployment** (controller for pods) and a **Service** definition in OpenShift.

The **oc-deployment.yaml** file defines a Tekton **Task** that in turn invokes a command to run the **deployment.yaml** to create the deployment and service for the sample application. To enable this action, each task will define with a step using the **quay.io/openshift/origin-cli:latest** docker image.

__1. Review the **oc-deploy.yaml** file using cat command. **DO NOT MODIFY THE FILE**

```
cat oc-deploy.yaml
```

```
[ibmdemo@icp4a tekton-pipeline]$ cat oc-deploy.yaml
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: deploy-cm
spec:
  inputs:
    resources:
      - name: tekton1-git
        type: git
    params:
      - name: pathToContext
        type: string
        default: /workspace/tekton1-git
      - name: targetNamespace
        type: string
        default: tekton-lab
  steps:
    - name: oc-service
      image: quay.io/openshift/origin-cli:latest
      env:
        - name: REG_PWD
          valueFrom:
            secretKeyRef:
              name: tekton-lab-deployer-secret
              key: token
      command: ["/bin/bash", "-c"]
      args:
        - oc apply -f /workspace/tekton1-git/tekton-pipeline/deployment.yaml --token=$REG_PWD -n tekton-lab
[ibmdemo@icp4a tekton-pipeline]$
```

__2. Review the **deployment.yaml** file that is used to create the Deployment and Service for the application, and is invoked by **deploy-cm** Task you reviewed in the previous step

```
cat deployment.yaml
```

The **Deployment** specifies 1 replica (pod), and is deployed using the Docker image that is pushed to the OpenShift image registry.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: tekton1
  name: tekton1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tekton1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: tekton1
    spec:
      containers:
      - image: docker-registry.default.svc:5000/tekton-lab/tekton1:latest
        name: tekton1
        resources: {}
```

The **Service** defines how the application will be accessed

```
# This the service yaml

apiVersion: v1
kind: Service
metadata:
  labels:
    app: tekton1
  name: tekton1-svc
spec:
  ports:
    - port: 3000
      protocol: TCP
      targetPort: 3000
  selector:
    app: tekton1
  type: LoadBalancer
```

__3. Run the **oc-deploy.yaml** to create the Tekton Deployment Task. Then list the new task

```
oc create -f ./oc-deploy.yaml

oc get tasks
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create -f ./oc-deploy.yaml
task.tekton.dev/deploy-cm created

[ibmdemo@icp4a tekton-pipeline]$ oc get tasks
NAME         AGE
buildah      47m
deploy-cm    7s
```

7.4.10 Create the Pipeline that invokes the build/push and deploy Tasks you created

Now that that tasks have been created, they can be incorporated and orchestrated in a Pipeline. The pipeline in the lab does the following:

- First, the pipeline runs the **buildah** task that performs the **build** and **push** steps
- Once the build-push task completes, the **deploy-cm** task is executed to deploy the app to OpenShift
- The pipeline orchestrates the order of the task execution using the **runAfter** tag in the pipeline definition. If the build-push task fails, the deploy task will not run.

__1. Review the **pipeline.yaml** file

```
cat pipeline.yaml
```

Snippet showing the tasks in the pipeline.yml file

```
tasks:
  - name: build-push
    taskRef:
      name: buildah
    params:
      - name: BUILDER_IMAGE
        value: "quay.io/buildah/stable:v1.11.0"
      - name: DOCKERFILE
        value: "/Dockerfile"
      - name: TLSVERIFY
        value: "false"
    resources:
      inputs:
        - name: tekton1-git
          resource: tekton1-git
      outputs:
        - name:  tekton1-image
          resource: tekton1-image
  - name: deploy-to-cluster
    taskRef:
      name: deploy-cm
    params:
      - name: pathToContext
        value: "/workspace/tekton1-git"
      - name: targetNamespace
        value: "tekton-lab"
    resources:
      inputs:
        - name: tekton1-git
          resource: tekton1-git
    runAfter:
      - build-push
```

__2. Use the pipeline.yaml file to Create the pipeline. Then list the new pipeline

```
oc create -f ./pipeline.yaml
oc get pipelines
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create -f ./pipeline.yaml

pipeline.tekton.dev/tutorial-pipeline created

[ibmdemo@icp4a tekton-pipeline]$ oc get pipelines
NAME                AGE
tutorial-pipeline   1m
[ibmdemo@icp4a tekton-pipeline]$
```

7.4.11 Run the Pipeline

To execute the pipeline, a **PipelineRun** artefact should be created.

A **PipelineRun** starts a **Pipeline** and ties it to the **Git** and **image** resources that should be used for the specific invocation. It automatically creates and starts the TaskRuns for each Task in the Pipeline.

__1. Review the **pipeline-run.yaml** file

```
cat pipeline-run.yaml
```

The **PipelineRun** identifies the pipeline to run, and provides the resources and parameters used during its execution. It also defines the Service Account that runs the pipeline.

```
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  name: tutorial-pipeline-run-1
spec:
  serviceAccount: tekton-sa
  pipelineRef:
    name: tutorial-pipeline
  resources:
    - name: tekton1-git
      resourceRef:
        name: tekton1-git
    - name: tekton1-image
      resourceRef:
        name: tekton1-image
  params:
    - name: BUILDER_IMAGE
      value: quay.io/buildah/stable:v1.11.0
    - name: DOCKERFILE
      value: ./Dockerfile
    - name: TLSVERIFY
      value: "false"
    - name: pathToContext
      value: /workspace/tekton1-git
    - name: targetNamespace
```

__2. Execute the PipelineRun using the YAML file

```
oc create -f ./pipeline-run.yaml
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc create -f ./pipeline-run.yaml

pipelinerun.tekton.dev/tutorial-pipeline-run-1 created
```

Next, let's do some basic queries to ensure the pipeline is executing. Then, you will launch the Tekton dashboard to view the PipelineRun.

__3. First, verify that the tasks persistent volume claim (PVC) is bound to the persistent volume (PV) that you created for the lab.

```
oc get pvc
```

Note: If the PVC is not bound, the pipeline will hang and wait for storage to be available.

```
[ibmdemo@icp4a tekton-pipeline]$ oc get pvc

NAME                            STATUS    VOLUME    CAPACITY   ACCESS MODES
STORAGECLASS   AGE
tutorial-pipeline-run-1-pvc     Bound     pv0001    5Gi        RWO
4s
```

__4. Now, check that the pipeline pod is running, and there the READY state of the containers is **not stuck** at 0/5. The READY state will continue to progress as the build tasks execute and complete.

```
oc get pods
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc get pods
NAME                                                READY     STATUS
RESTARTS   AGE
tutorial-pipeline-run-1-build-push-4jpqb-pod-c33d32   4/5       Running   0
1m
```

A fully completed and successful **pipelineRun** will result in the pod states below.

Note: It may take 10 minutes to run the pipeline, as it pulls the required docker images from DockerHub, builds the docker image for the app, pushes the image to the OpenShift image registry, and deploys the application.

- The **tekton1-<pod ID>** is the application that was deployed via the pipeline.
- The **tutorial-pipeline-run-1-build-push** pod is the pod that ran the build/push tasks
- The **tutorial-pipeline-run-1-deploy-to-cluster** pod is the pod that ran the deploy task

```
[ibmdemo@icp4a tekton-pipeline]$ oc get pods
NAME                                                 READY     STATUS
RESTARTS   AGE
tekton1-76dcb78d7d-nh4c8                             1/1       Running
tutorial-pipeline-run-1-build-push-4jpqb-pod-c33d32          0/5       Completed
tutorial-pipeline-run-1-deploy-to-cluster-g9r5s-pod-f4f00b   0/2       Completed
```

7.4.12 Access the Tekton Dashboard to view the PipelineRun status and logs

The Tekton Dashboard is available in the lab environment. The Tekton Dashboard is a general purpose, web-based UI for Tekton Pipelines and Tekton triggers resources. It allows users to manage and view Tekton resource creation, execution, and completion.

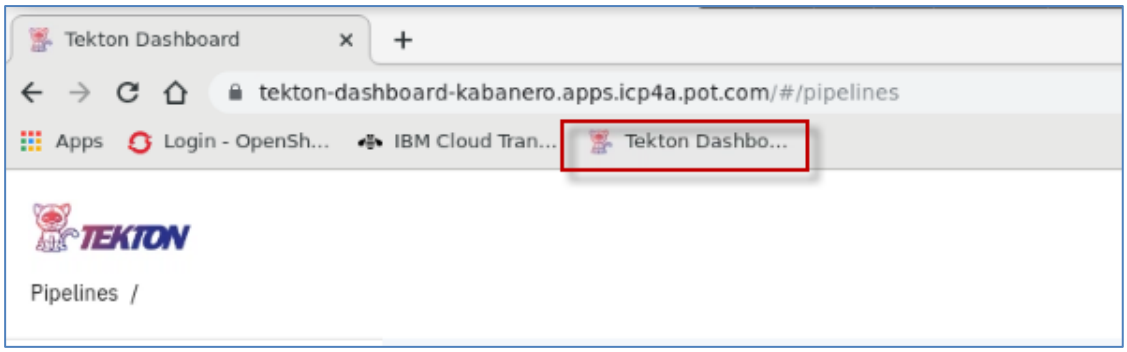
Note: If the Pipeline fails, refer to the "**Appendix: troubleshooting**" section of this lab to learn how to view the logs and cleanup and restart a failed pipeline.

Note: If you modified any of the pipeline YAML files or used different names, namespaces, etc, you may have to update YAML files to fix the issues if you changed the names, namespaces, etc that are coded in the YAML files.

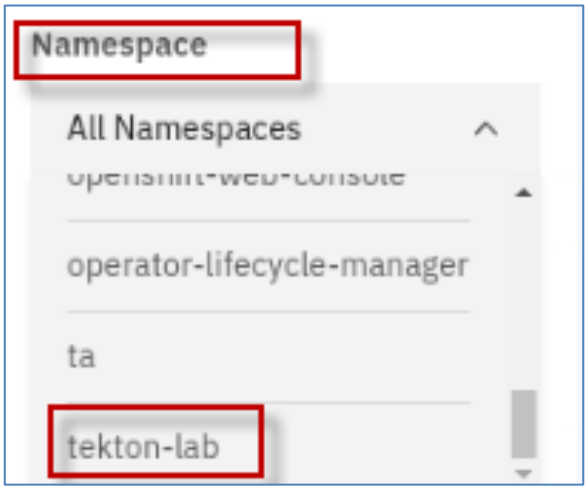
__1. Access the Tekton Dashboard

__a. Click the Chrome browser icon  located at the bottom of the VM window

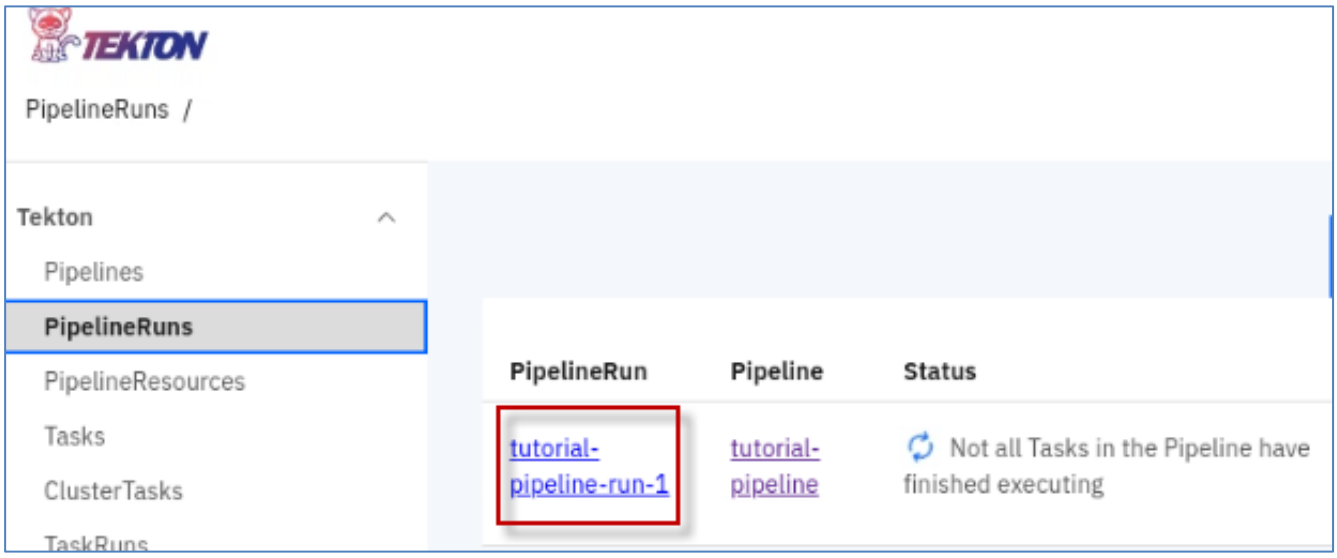
__b. From the browser, click the **Tekton Dashboard** Bookmark located on the bookmark toolbar



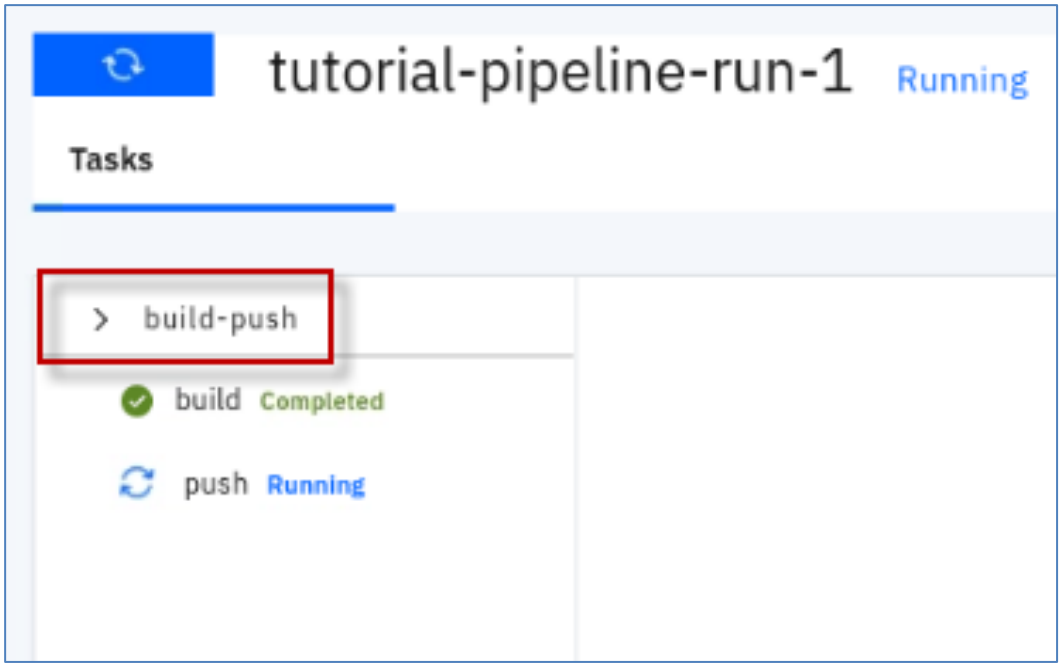
__2. From the Tekton Dashboard, use the **Namespace** pulldown menu, and select the **tekton-lab** namespace to filter the resources to the namespace used in the lab.



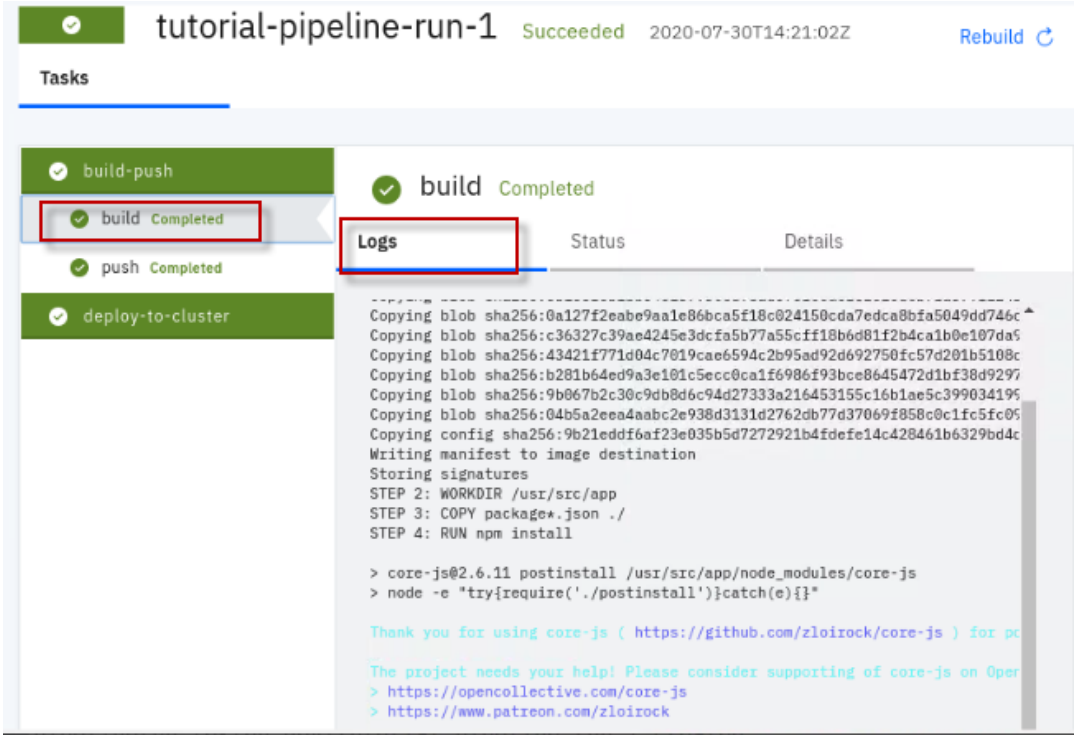
__3. From the Tekton Dashboard, click the **PipelineRuns** menu option to view your PipelineRun. Then click on the tutorial-pipeline-run-1 pipeline to view the details.



__4. Expand the **build-push** Task.

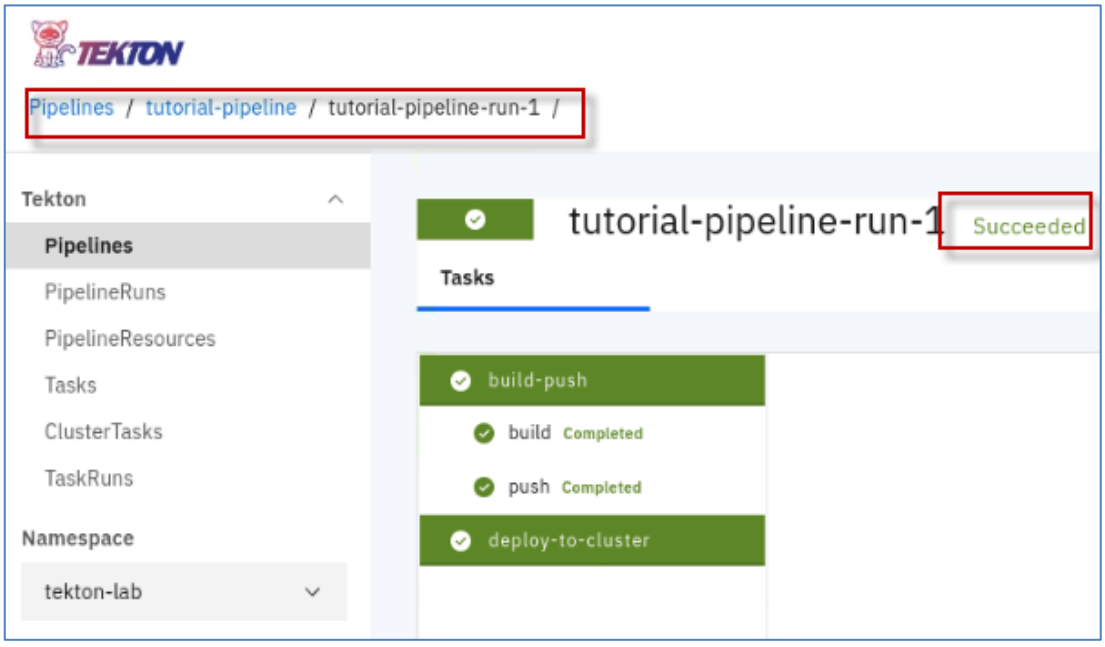


__5. Then you can click the **build** step or the **push** step to see the logs and status.



__6. Once the build-push tasks complete, the deploy-to-cluster task will execute to deploy the application. You can view any of the logs for the tasks.

A successful PipelineRun will look like this:



7.4.13 Validate the application is deployed and runs as expected

Upon successful completion of the pipeline, the sample NodeJS Express application is deployed to OpenShift.

In this section, you will view the application resources that were deployed to OpenShift and validate the sample application runs as expected.

__1. Use the following commands to verify the application is deployed and the pod is running

```
oc get deployments

oc get pods | grep tekton1
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc get deployments

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
tekton1   1         1         1            1

[ibmdemo@icp4a tekton-pipeline]$ oc get pods | grep tekton1

tekton1-76dcb78d7d-scfls                                  1/1       Running
```

__2. Use the following commands to verify the service was created

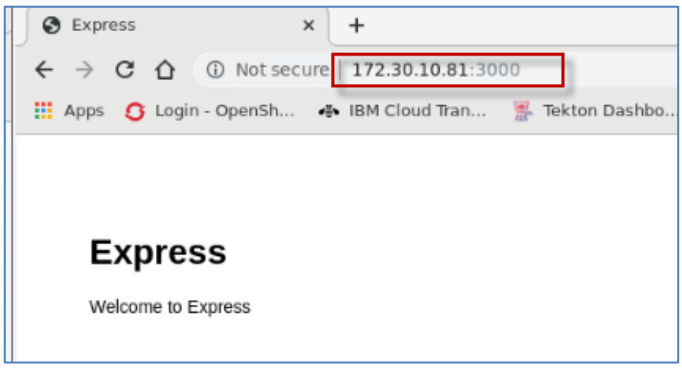
```
oc get services
```

```
[ibmdemo@icp4a tekton-pipeline]$ oc get services

NAME          TYPE           CLUSTER-IP      EXTERNAL-IP                   PORT(S)
tekton1-svc   LoadBalancer   172.30.10.81    172.29.215.72,172.29.215.72   3000:30580/TCP
```

__3. Test the application from the web browser. Use the CLUSTER-IP and PORT as the URL

http://<CLUSTER-IP>:3000



7.5 Conclusion

Congratulations! You have completed the lab and are on your way to developing robust pipelines for CI/CD of your application deployments using containers and RedHat OpenShift Container Platform.

In this lab, you learned how to create the Tekton resources to automate CI/CD for microservices deployed to OpenShift.

- PipelineResource
- Task
- Pipeline
- PipelineRun

You learned how to configure a Service Account with proper authorization and roles to be able to push Docker images to an image registry and authenticate to OpenShift and deploy the application via the Tekton Pipeline.

End of Lab 07: Using Tekton pipelines for CI/CD of microservices to RedHat OpenShift Container Platform

Appendix: Troubleshooting and restarting a failed PipelineRun

If any of the tasks fail in the pipeline, you will need to review the logs from the failed task to determine the issue. Once you resolve the problem, you will need to execute a new PipelineRun.

Tip: View the logs using the Tekton Dashboard you saw in the lab to determine the failure message.

Here is my advice for re-running a new pipeline after a failed attempt.

Note that every pipeline must have a **unique name**. The name is hard coded in the YAML files used to create the pipeline resources.

__1. Run the following commands to cleanup a failed PipelineRun and start a new one

```
oc delete -f ./pipeline-run.yaml
```

Note: The deployment and service resources may not exist, depending on where the pipeline failed. So, the delete command may state “not found”. Ignore the message

```
oc delete deployment tekton1

oc delete service tekton1-svc

oc create -f ./pipeline-run.yaml
```

Appendix: SkyTap Tips for labs

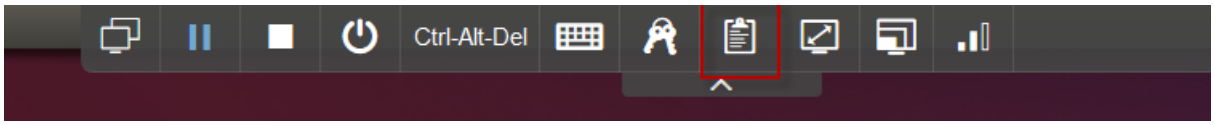
How to use Copy / Paste between local desktop and Skytap VM

Using copy / Paste capabilities between the lab document (PDF) on your local workstation to the VM is a good approach to more efficiently work through a lab, while reducing the typing errors that often occur when manually entering data.

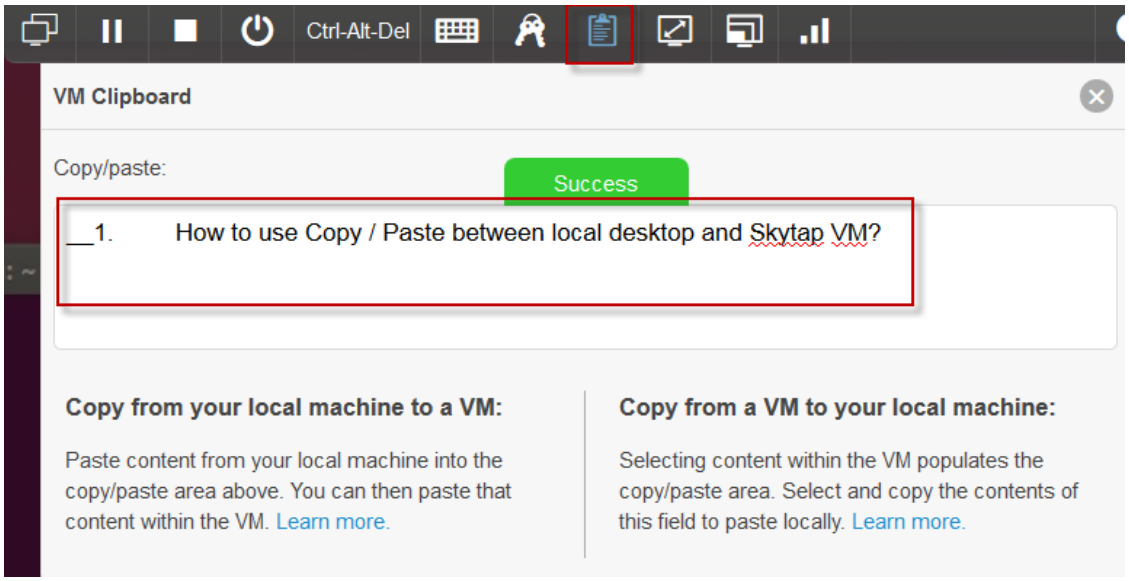
__1. In SkyTap, you will find that any text copied to the clipboard on your local workstation is not available to be pasted into the VM on SkyTap. So how can you easily accomplish this?

__a. First copy the text you intend to paste, from the lab document, to the clipboard on your local workstation, as you always have (CTRL-C)

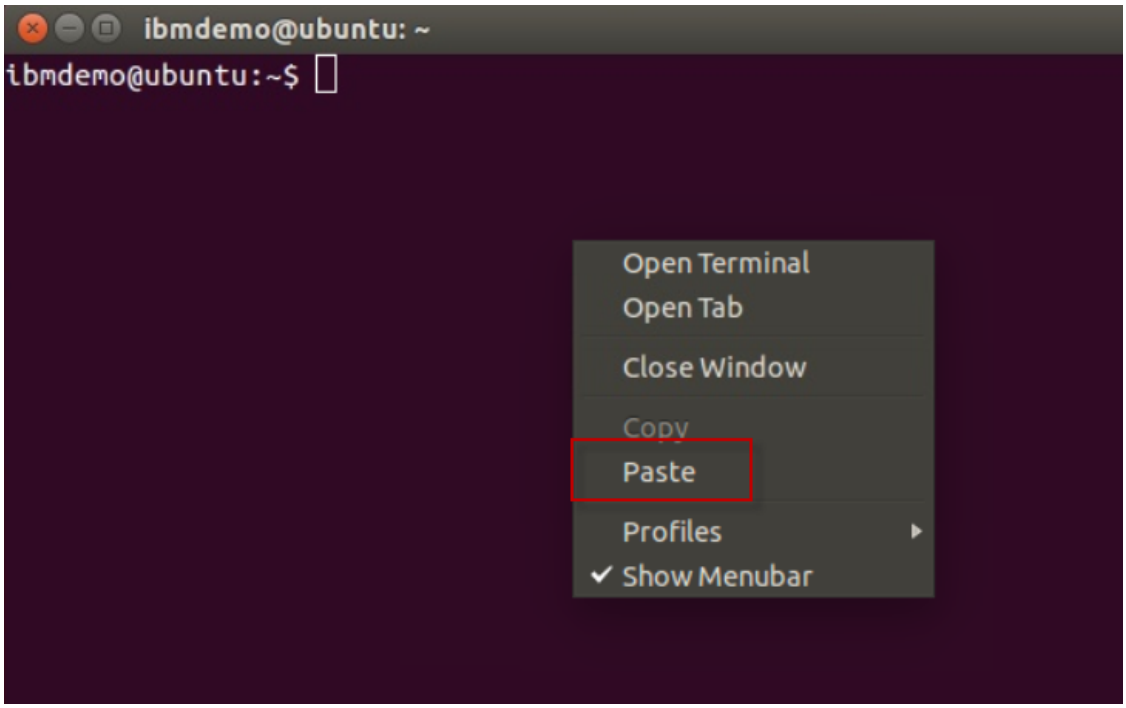
__b. Return to the SkyTap environment and click on the Clipboard at the top of the SkyTap session window.



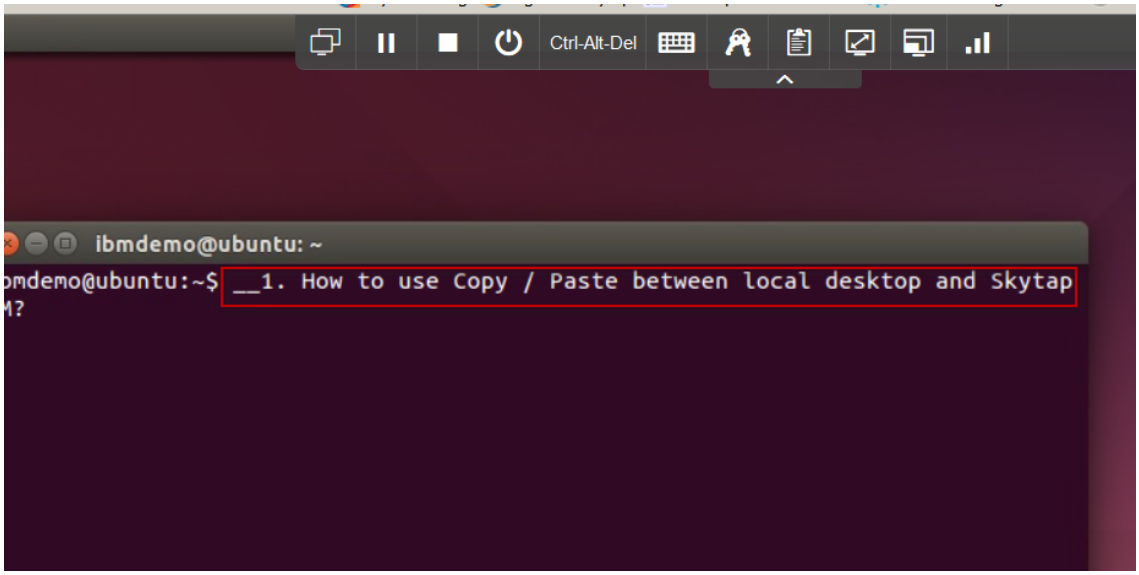
__c. Use **CTRL-V** to paste the content into the Copy/paste VM clipboard. Or use the **paste** menu item that is available in the dialog, when you right mouse click in the clipboard text area.



__d. Once the text is pasted, just navigate away to the VM window where you want to paste the content. Then, use **CTRL-C**, or right mouse click & us the **paste menu item** to paste the content.



__e. The text is pasted into the VM



Note: The very first time you do this, if the text does not paste, you may have to paste the contents into the Skytap clipboard twice. This is a known Skytap issue. It only happens on the 1st attempt to copy / paste into Skytap.