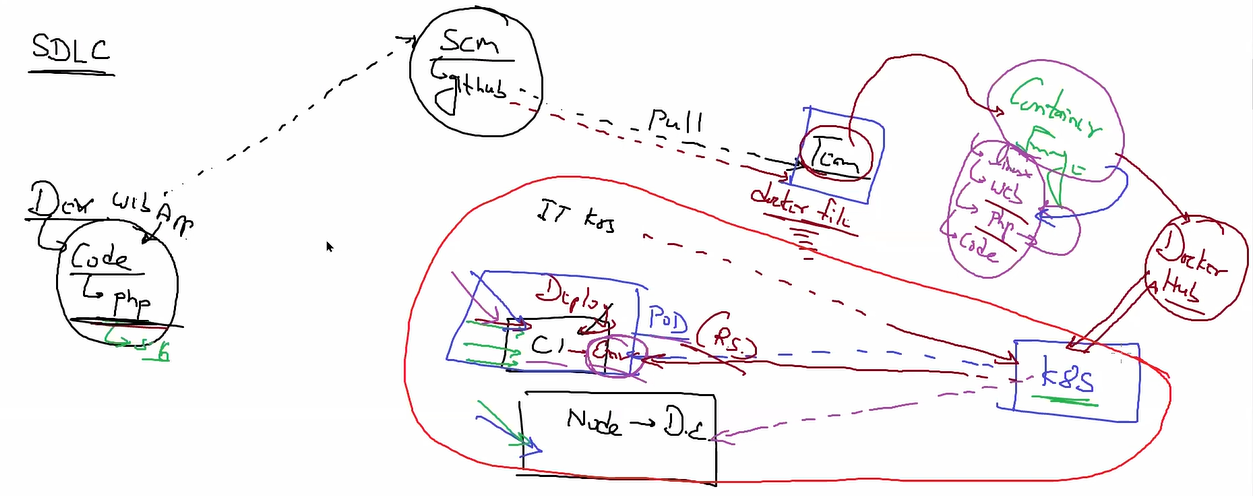
**Session 01 – Intro**

* K8s is container management tool.
* It will help you with respect to container.
* But we have to do many different things.
  + When developer will create a code, it will automatic uploaded to GitHub.
  + Then create an image of it.
  + Then we have to check… which version of php or java or Perl we are using.
  + After this we use k8s here.
  + Monitoring system.



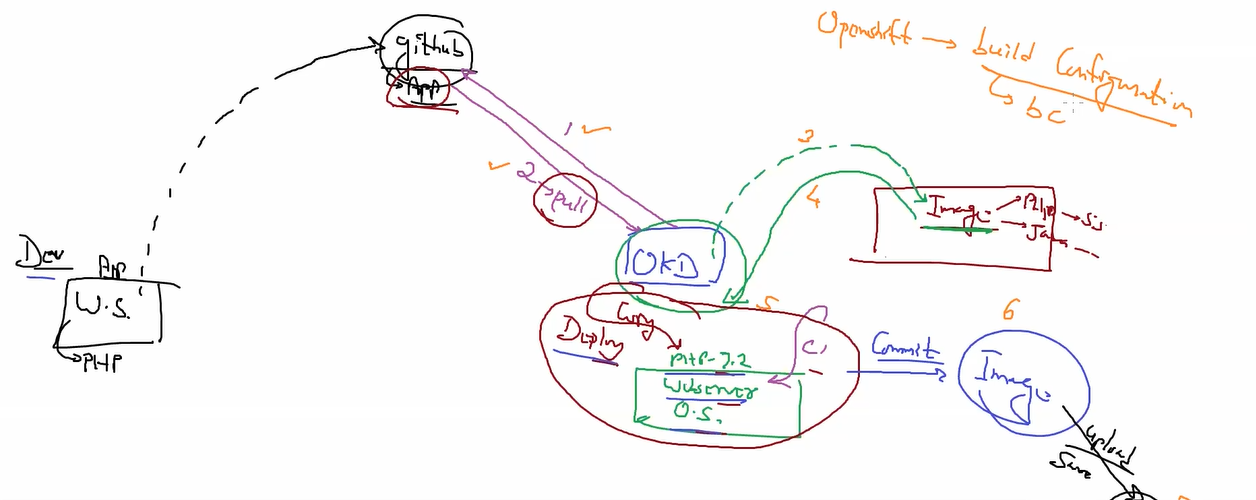
* We can also automate this thing using Jenkins, but industry say if you use multiple tools for one thing, then it will create a process very complex.
  + It also makes management harder.
  + We want some tool which can give you self-service.
  + The company should be reliable.
  + Company wants **unified solution**.
* **OpenShift** is a tool for this coming from **redhat** company.
  + OpenShift is a **complete management tool**.
  + In OpenShift we also have resources like k8s.
  + One of the resources in OpenShift is **BuildConfiguration** (BC), they automatically go to GitHub, figure out code is written in which language & version, then they launch interpreter runtime and run code for us.
* OpenShift is normally known as **Enterprise Great tool**.

**Session 02 – OKD // CRC // BC // DC // minishift installation// route**

* **RHOCP** is an enterprise tool and you have to purchase license for using it.
  + **OKD** is a **community edition** of it.
    - It is used for testing purpose.
    - It is upstream project of RHOCP.
* We have two ways for installing single node cluster in our VM.

1. Minishift
2. **CRC** (code ready container)
   1. It will not work on top of oracle vbox.

* Developer will push code to GitHub.
  + OKD go to GitHub, download the code and try to check it is written in which language, it has same pre created images for php, ruby and all other things.
  + It will launch the container and copy code in this image.
  + After all of this it will create a new image from this and push it into his private registry.
  + This complete process is known as build.
    - **Build configuration**



* + In future you want to launch pod/container using this image then you can easily launch.
  + They will launch pod using replicaset and private LB (svc).
    - Here they will use concept called **deployment configuration**.
* For starting it we have to use **minishift start --vm-driver virtualbox** 
  + - By default they use hyper V.
  + For going outside, internet connectivity we have to use **--network-nameserver 8.8.8.8**
  + So final command would be **minishift start --vm-driver virtualbox --network-nameserver 8.8.8.8**.
* You can access webUI using minishift console.
* You have to create a project for doing anything in OKD.
  + It is like namespace in k8s, but it has some more capability.
  + If you want to expose LB to outside world for this you have to route service.
    - Route = NodePort

**Session 03 – oc installation // console url // project // IS-IB**

* For managing openshift from CLI we have to use user command **OC**.
* We can download OC command from internet, but in our case minishift has already downloaded OC command for us.
  + To check where we have OC command, we have to use **minishift oc-env**.
  + We have to tell OC that openshift cluster is running on this URL.
    - You can check url using **minishift console --url**
  + By default minishift has created credentias with username: **system**, password: **admin**.
  + Now you can give these details to oc.
  + Oc login <https://localhost:8443> -u system:admin
* You can create a new project using **oc new-project p3**
  + You can check all project list using **oc get projects**.
    - oc get pods
    - oc get bc
    - oc new-app <https://github.com/>...
    - Oc logs -f bc/git-workshop
    - Oc get svc
  + When you expose in oc here they will create a NAT program so we can connect to cluster from outside world.
    - Oc get route
    - Oc expose svc git-workshop
      * Now if you see one route has been automatically created.
* Openshift using image and create another image which contains app (code + previous image).
  + Here first image is known as **Image Streaming**.
  + Second image is known as **image builder**.
  + Suppose developer has changed code.
    - You can start new build with,
    - **Oc start-build git-workshop**
  + First it will create a BC and then trigger DC (new pod).
    - Here for creating new pod they use rolling update strategy.

**Session 04 – template // cakephp-mysql-persistent**

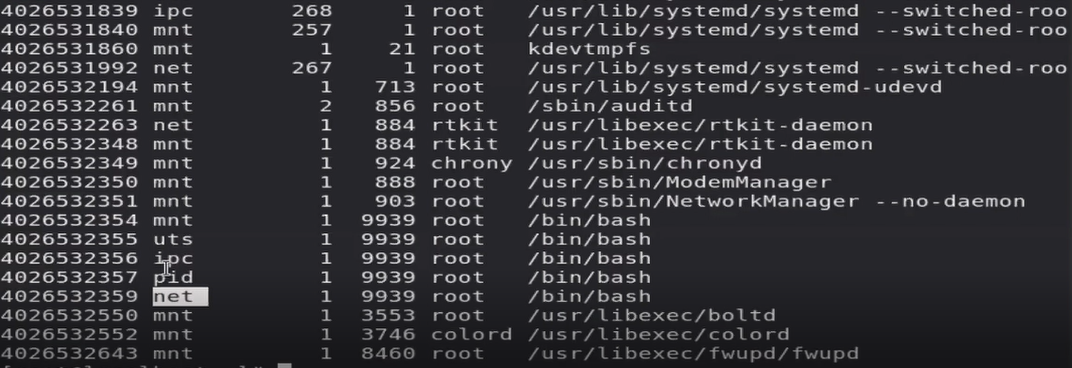
* In multitier architecture like wordpress-mysql we have to work with lots of things.
  + One pod is depend on another pod, pvc, secrets and lots of things.
  + Here we can use template concept.
* We can put all the things in one box and use that as a template.
  + Some templates are already available, but they are available in different projects.
  + So you need root user to go inside another projects.
  + Templates are available in openshift project.
  + **Oc new-app cakephp-mysql-persistent**
    - They will automatically detect this is a template and they will create complete setup for you.

**Session 05 – deployment configuration // deployment // env**

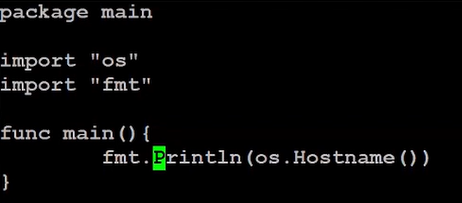
* Deployment config and deploy has one small difference.
* DC use RC & deployment use RS behind the seen.
* Wordpress-mysql deployment
  + **Env variable**

**Session 06 – runc // container //go //CRIO //podman //OCI//CRI shim**

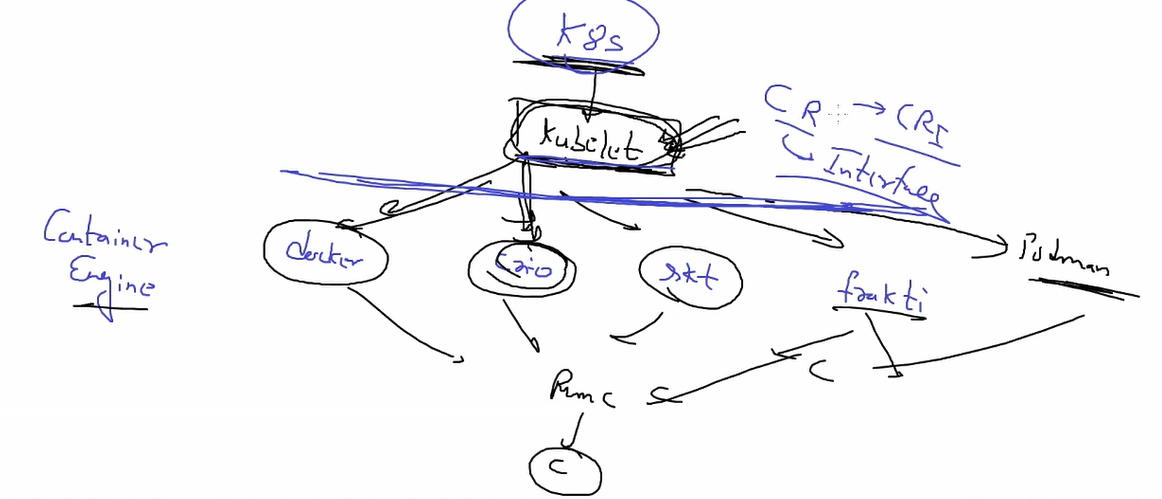
* Docker never go and launch a container for you.
* Docker will send an instruction to launch container, instruction is run by **container runtime**.
* One of the famous is **RunC**.
  + Run Time spec file is run by RunC.
  + They write everything in run time spec file.
* RunC has their own config file (name is also config).
* When you launch a container it will launch a complete new OS.
  + But base OS is running a process and complete OS is running inside this process.
  + It is like process inside process.
  + From kernel version 3 and above we have some advance concept like namespace.
  + We can create multiple **namespace**.
    - In container we give separate username, process tree
    - Because container has almost all the things like real OS in his namespace so he feels like he has isolate environment.
      * **Isolate environment == OS**
    - What ever OS needs we contains all the things in one box
  + If we check **lsns** command we can see all namespaces of kernel.



* + - Here we can see we have multiple namespace running on 9939.
      * Here we have give separate mnt (/ storage), pid, network card, uts (hostname) and some other things.
* In RunC config file we written that which command to run which namespaces to include.
  + We use mnt namespace for creating / drive.
  + And we bundle image (entire data) to the / drive.
* If you think, RunC is **container runtime**, Containerd is **container engine** and docker is **container management tool**.
  + Docker engine is one who connect to runc and start stop container.
  + **Yum install runc -y**
  + **Runc spec**
    - It will create simple config file.
    - Here we need image but it will not manage by runc for this we have tool like docker podman so instead of this we can create a simple program in go language.
    - **Yum install go -y**
    - **Vi ak.go**



* + Go build -o ak
    - It will create executable file.
    - ./ak
    - Mkdir rootfs
    - Mv ak rootfs/
  + **Runc create myc1**
  + Runc list
    - Now you can see one container has been launched.
    - You can check lsns more namespace has been created.
  + **Nsenter -u -n -t <id>**
    - U for entering to user name space & n for network namespace.
    - Now you can see you are in different OS.
    - Technically you are in same OS but different namespace.
  + **Runc start myc1**
    - Here it will only print hostname, because we have set this in go language.
* In docker it will comes up with lots of facility, but sometimes we don’t require this.
  + Like container will give you **security networking** and some other things.
  + But mostly we use Kubernetes or openshift, & they have better facility for these things.
  + So we don’t want this thing from docker.
    - It will do some extra **overhead** to containers.
* In this setup when we use Kubernetes kubelet connect to **dockershim** (docker API), dockershim connect to docker, docker connect to containerd, container connect to runc and launch the container.
  + But docker in keep on changing many things and k8s and docker belongs to company so for k8s it is harder to sink with docker.
  + So they have depreciated docker and started using CRIO.
  + Kubelet connect to CRIO, CRIO connect to runc and launch the container.
* **CRIO – CRI and O for OCI.**
* Instead of directly connecting to container runtime, k8s has created one layer.
  + Now kubelet connect to this layer and on the fly decide which container runtime they want to use.
  + This layer is known as **CRI shim** (container runtime interface).
  + Here kubelet is client and CRI shim is server.



* One community has created standard for image create and all things.
  + So podman image can be used with docker.
  + This community name is **OCI** (open container initiative).
* Our final goal is run the container, so we have created an extra CLI work as a client, it will direct connect to runc and launch a container.
  + This command has extra facility then runc like managing container.
  + Here they don’t have daemon or service so container (service) never goes down.
  + This tool is known as **podman**.

**Session 07 – podman**

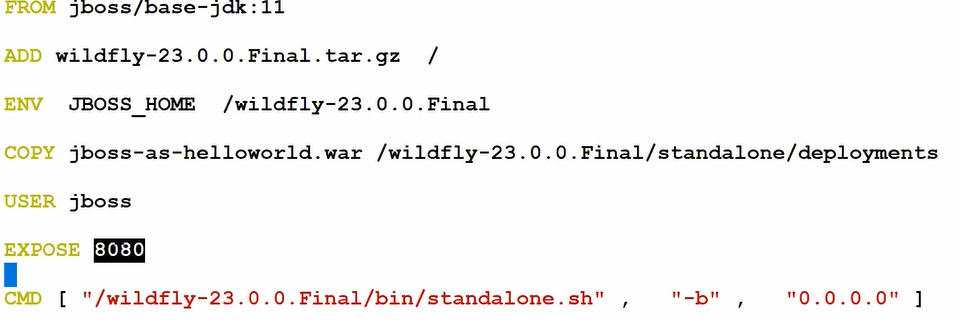
* Podman connect directly to the runc and launch the container.
  + You can say Podman is a **client tool** for **RunC**.
  + From rhel8 redhat gives you podman in your DVD.
  + But when you use OpenShift it will give you CRI-O.
    - Because Kubernetes need CRIO behind the seen.
  + We can also upload podman image in Quay registry.
  + It is opensource free registry.
* Yum install podman –y
  + Podman images
  + Podman pull vimal13/apache-webserver-php
  + Podman run --it --name myos vimal13/apache-webserver-php
  + Podman ps
  + Podman exec --it bash
  + Podma run -d --name db4 -e MYSQL\_ROOT\_PASSWORD=redhat -e MYSQL\_USER=vimal -e MYSQL\_PASSWORD=redhat -e MYSQL\_DATABASE=lwdb -v /mydata:/var/lib/mysql mysql
    - But we are mouting a storage and SELinux will not allow you this.
    - You can stop SELinux.

**Session 08 – jboss// wildfly // registery sequence//ADD**

* Here we are going to create JBOSS image.
  + Wildfly is a community edition of jboss.
  + For this we have to simple these simple steps.



* + - Cd /jbossws
    - Wget <https://download.jboss.org/wildfy/23.0.0.Final/wildfy-23.0.0.Final.tar.gz>
  + Tar -xzfv wildfly-23.0.0.Final.tar.gz
    - Export JBOSS\_HOME=/jbossws/wildfly-23.0.0.Final
    - Cd standalone
      * Here in same system jboss server, app & management portal also working.
    - Cd ..
    - Cd bin/
    - ./Standalone.sh
      * But it will only accessible from same system.
      * So here we can use this.
      * **./standalone.sh -b 0.0.0.0**
  + Now you can access same with IP:8080
    - Download demo code from GitHub to run.
    - Cp demo-code wildfly-23.0.0.Final/standalone/deployments/
  + You can download sample code from <https://github.com/vimallinuxworld13/jboss-sample-war-Dockerfile>
* We have to follow this same step inside docker container for creating image.
  + Podman first try to download image from redhat internal, then quay then docker hub.
  + Vim /etc/containers/registeries.conf
    - You can change registry sequence here.
    - Instead of copy keyword if you use ADD inside dockerfile.
    - It will copy and if file is tar then it will extract for you.
  + In JBOSS image it is mentioned that when we launch container it will launch with jboss user.
    - If you want this you have to mention this.
      * **USER AK**



* + Podman run -dit --name myos -p 1234:8080 myweb:v1

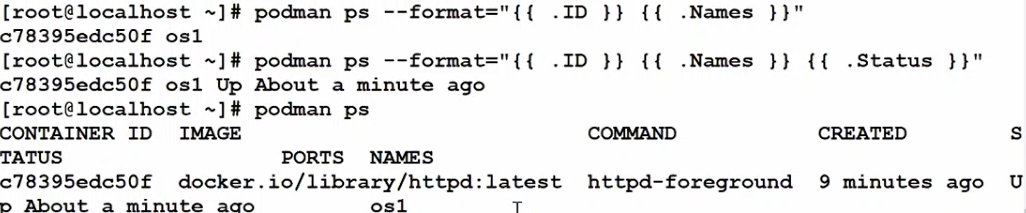
**Session 09 – Dockerfile//cmd//entrypoint// MAINTAINER// WORKDIR**

* You can add your details in the image which is stored in the metadata, for this you have to use MAITAINER keyword.
  + **MAINTAINER akshit <akshit.modi11@gmail.com>**
  + We can set any folder as a working directory, now when you launch a container you will automatically land on that directory.
    - **WORKDIR /var/www/html**

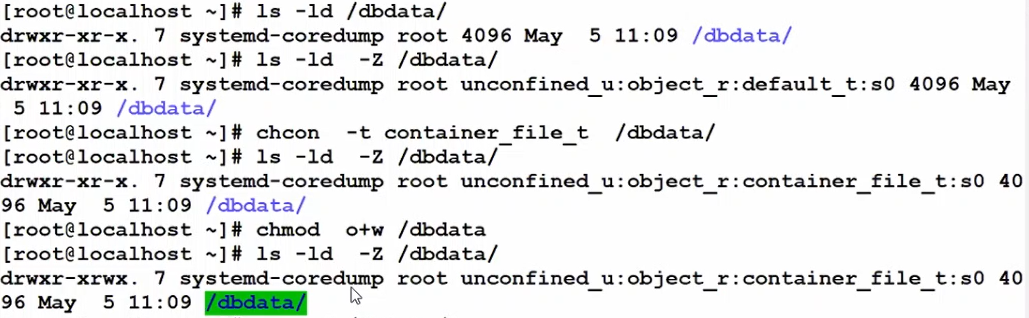


**Session 10 – short commands//selinux**

* We know in inspect command we can get data in JSON format.
* So we can retrieve IP address from it using,
  + **Podman inspect os1 -f “{{ .NetworkSettings.IPAddress}}”**
* You can retrieve id or/and name of container with this command.
  + **Podman ps –format=”{{.ID}} {{ .Names}} {{ .Status}}”**



* Sometimes we have to face SElinux issues.
  + We can set read write and all permissions for files and folders.
  + But there are also some **process permissions**, which is managed by SElinux.
  + So if you have SELinux enabled, you cannot mount container with base OS.
* You can check selinux permission using,
  + **Ls -ld -Z /dbdata/**
  + Here this line (unconfined\_u:object\_r:default\_t:s0) is known as **context**, so you can change context using,
    - **Chcon -t container\_file\_t /dbdata/**
  + This means if any container wants to mount then they can go.
    - Here we also using this folder for writing so we can also give write permission to folder using,
      * **Chmod o+w /dbdata**



* **Session 01 – Intro** 
  + Intro to OpenShift
  + Need of OpenShift
  + Management tool
  + Build Configuration
* **Session 02 – OKD // CRC // BC // DC // minishift installation// route**
  + OKD – community edition of RHOCP
  + CRC not work in vbox.
  + Build configuration
  + Deployment configuration
  + **minishift start --vm-driver virtualbox --network-nameserver 8.8.8.8**
  + route
* **Session 03 -- oc installation // console url // project // IS-IB**
  + Oc cli command
  + **minishiftoc-env**
  + **minishift console –url**
  + Oc login <https://localhost:8443> -u system:admin
  + Image streaming (base image)
  + Imgae builder (new custom image)
* **Session 04 – template // cakephp-mysql-persistent**
  + Template
    - Can create complete architecture using template
    - Contains PVC, secrets multitier architecture
  + Available in **openshift** project
* **Session 05 – deployment configuration // deployment // env**
  + Deployment configuration – use rc
  + Deployment – use rs
  + Env variable
* **Session 06 – runc // container //go //CRIO //podman //OCI//CRI shim**
  + Container management tool – docker
  + Container engine -- containerd
  + Continer runtime – runc
    - Config file
    - **Isolate environment == OS**
    - Lsns
    - Supports go language
  + Docker shim
  + Container runtime intefacse
  + Open container initiative
  + Podman
    - By redhat
    - Don’t have server
* **Session 07 – podman// quay** 
  + Client tool for runc
  + Quay registry
* **Session 08 – jboss// wildfly // registery sequence// ADD**
  + Jboss
    - Community edition – wildfly
  + Standalone
  + ADD
    - Copy inside container and also untar file if it is tar file.
  + Podman has registry sequence.
    - Vim /etc/containers/registeries.conf
* **Session 09 – Dockerfile//cmd//entrypoint// MAINTAINER // WORKDIR**
  + MAINTAINER
    - Here you can mention who is maintaining image
  + WORKDIR
    - To change con land position.
  + CMD vs entrypoint
  + Build time commands & runtime commands.
* **Session 10 – small commands//selinux**
  + Podman inspect os1 -f “{{ .NetworkSettings.IPAddress}}”
  + Podman ps –format=”{{.ID}} {{ .Names}} {{ .Status}}”
  + Ls -ld -Z /dbdata/
  + Chcon -t container\_file\_t /dbdata/
  + Chmod o+w /dbdata