**Application walkthrough in bare metal**

**Abstract:**

User-api is a simple application for bootstrapping the concept of micro services. This is basically a springboot application (rest api) with embedded h2 as a database and it is capable of doing ***CRUD*** operation from backend. I used decomposition pattern for building this application so the database will not be shared across other services. All intercommunication across services will happen only through http call. Application can be easily be scaled horizontally using this pattern. As the architecture is distributed so several other technology (service discovery, log monitoring, centralised configuration management, etc...) are needed to build end to end application.

**Technology Used:**

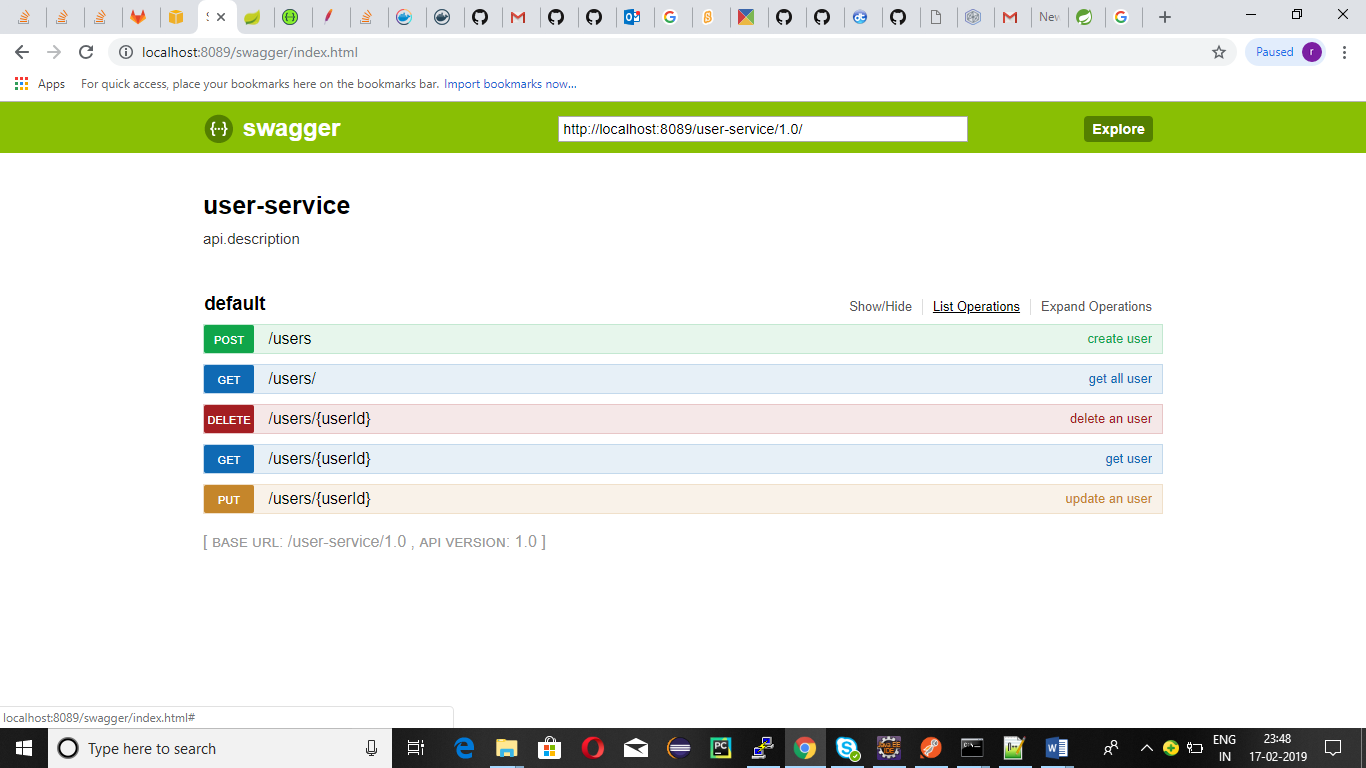
* Springboot(For embedded web container, embedded database, DI injection framework)
* Apache Camel(For rest dsl style controller, Integration Framework)
* Apache Maven (For build tool)
* Database(H2)
* Docker(Containerization)
* Kubernetes(Orchestration)

**Building the application**:

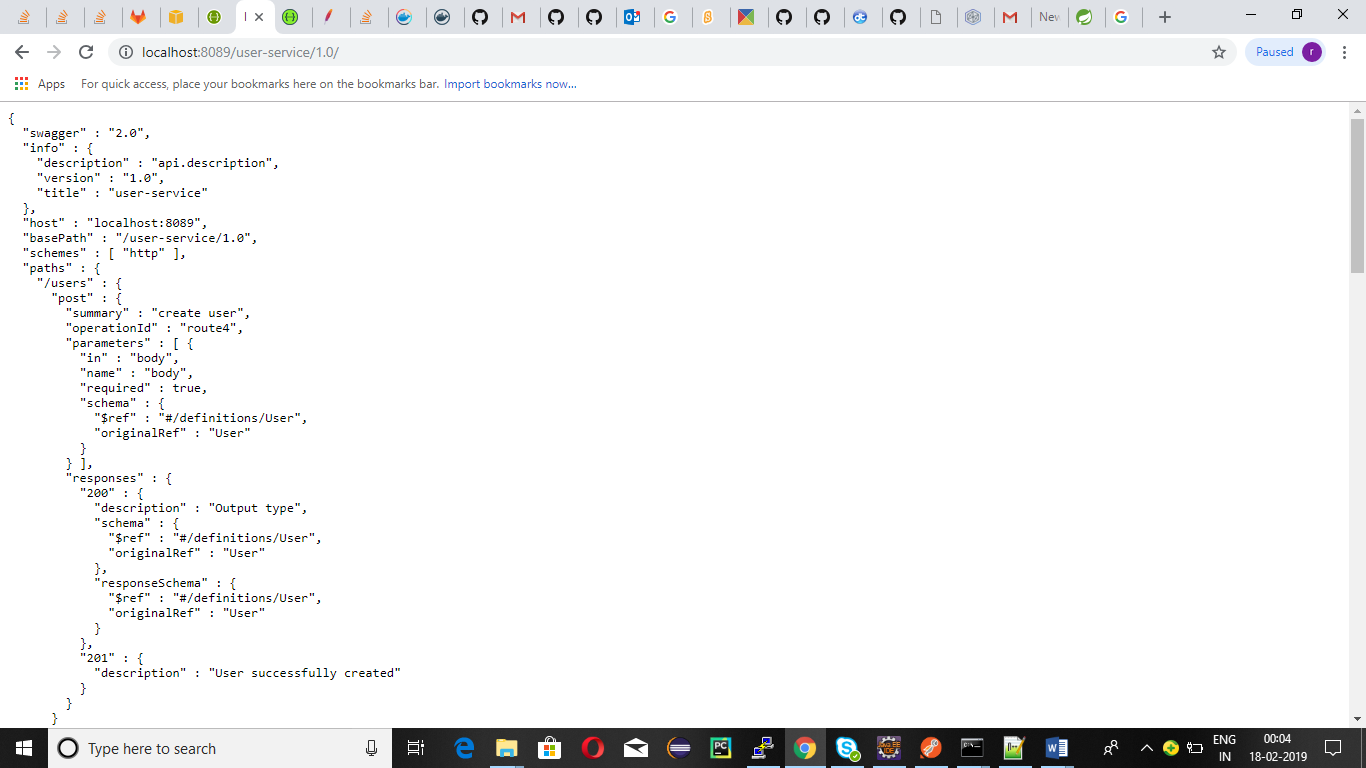
* To build the application execute 🡪 **mvn clean package** from project root dir.

**Running the application**:

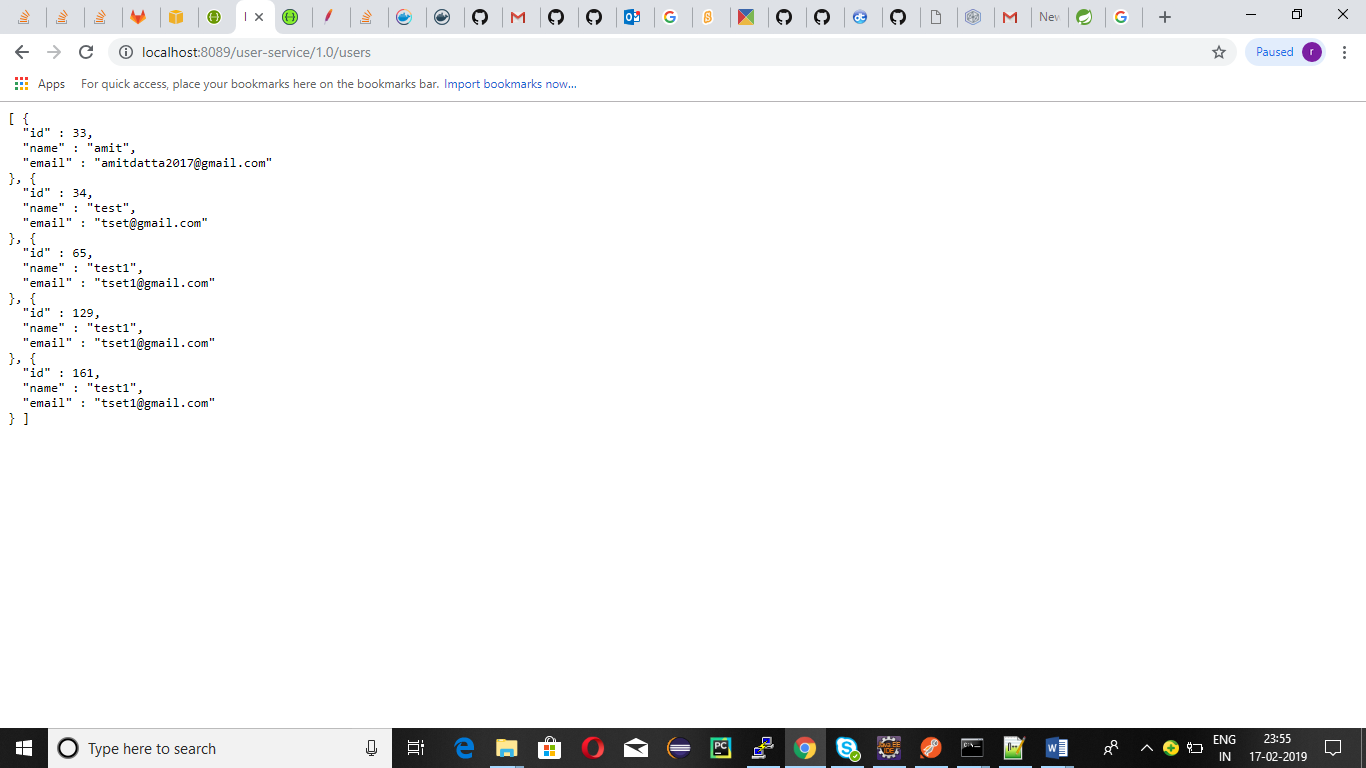
* Navigate to project-api dir execute 🡪 java –jar user-api-1.0.0.jar
* Swagger editor ui is integrated within the application so all application endpoint can be easily visualised 🡪 <http://localhost:8089/swagger/index.html>



* The request response json (swagger.json) can be extracted from the following application endpoint 🡪 <http://localhost:8089/user-service/1.0/>



* To fetch all user details from backend, make a get call from browser or using curl or using postman 🡪[http://localhost:8089/user-service/1.0/users](http://localhost:8089/user-service/1.0/users%20) . Postman can be used to check other endpoints of the application.



**Application walkthrough in docker**

**Docker installation:**

* sudo apt-get remove docker docker-engine docker.io containerd runc
* sudo apt-get update
* sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
* sudo apt-key fingerprint 0EBFCD88
* sudo add-apt-repository "deb [arch=amd64]https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"
* sudo apt-get update
* sudo apt-get install docker-ce docker-ce-cli containerd.io
* check docker version to confirm

**Building application docker image(optional)**

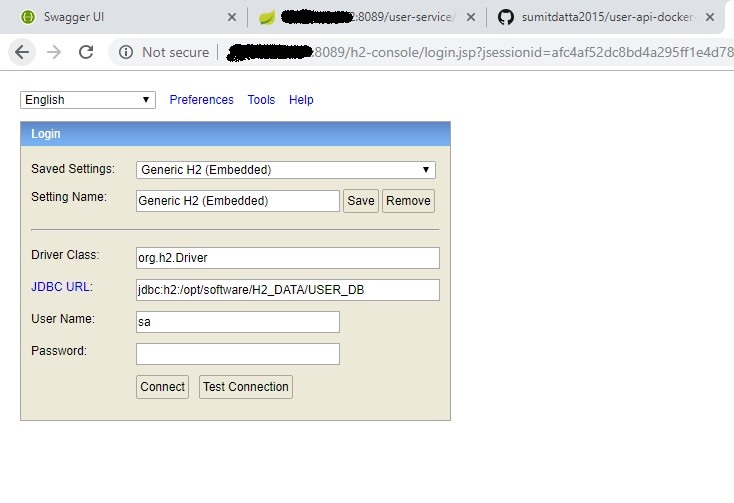
* user-api docker images can be built from the source jar itself or it can be pulled from docker hub. DockerHub image name is **sumitdatta2015/user-api**
* To build and tag image, you can use **docker build -t sumitdatta2015/user-api** [**https://github.com/sumitdatta2015/user-api-docker-build.git**](https://github.com/sumitdatta2015/user-api-docker-build.git)
* To build image from source separately, you can build the source using mvn clean package and use the Dockerfile in user-api resources folder.
* If you had private registry to store created dockerfile at the same time of build. You can use mvn clean package docker:build –DpushImagefrom user-api project. Please build the whole project before executing the so as to update build all its dependencies and mention your dockerhost in user-api post file.

**Pulling application docker images(optional in previous steps is done)**

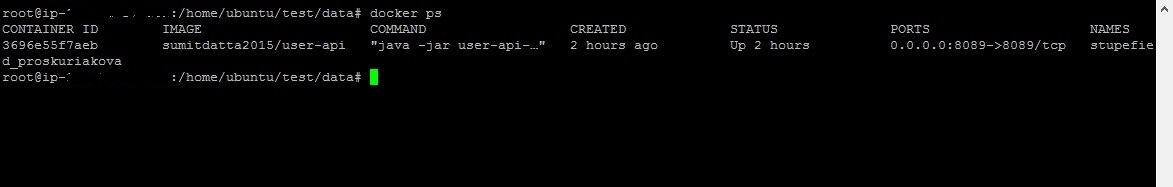
* docker pull sumitdatta2015/user-api

**Running application**

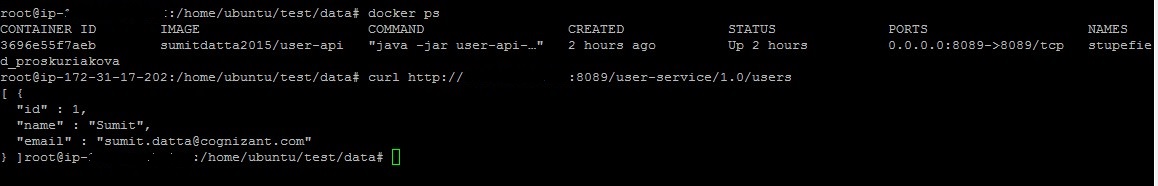
* user-api use two volumes one for storing application data (**H2\_DATA**) and another for application logs(**log**). Please make necessary folder in your host machine or network shared drive if you want to persist data.
* Application had in build H2 as database embedded. Use **application-host:application-port/h2-console** to connect.



* Sample docker run command to do host mapping 🡪 **docker run -d -v /home/ubuntu/test/data/app\_data:/opt/software/H2\_DATA -v /home/ubuntu/test/data/app\_log:/opt/software/log -p 8089:8089 sumitdatta2015/user-api**
* To check running container 🡪 sudo docker ps



* To check end to end application endpoint



**Application walkthrough in kubernetes**

**Kubernetes installation:**

Provision minimum 2 nodes one for master and one for slave in aws. For master minimum 3 GB and 2 core is required. For node minimum 1 GB and 1 core is required.

**Execute the following commands in the Master and Slave machines**

* Switch to the **root** user (sudo su)
* apt-get update && apt-get install -y apt-transport-https
* curl -s <https://packages.cloud.google.com/apt/doc/apt-key.gpg> | apt-key add -
* cat <<EOF >/etc/apt/sources.list.d/kubernetes.list  
  deb <http://apt.kubernetes.io/> kubernetes-xenial main  
  EOF
* apt-get update
* apt-get install [docker.io](http://docker.io/)
* apt-get install -y kubelet kubeadm kubectl kubernetes-cni
* Run docker as daemon in both Master and Slave nodes as given below(optional):

1. Open /lib/systemd/system/docker.service
2. Add the below line and save the file:
3. ExecStart=/usr/bin/dockerd -H fd:// -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2376 $DOCKER\_OPTS
4. **Execute the following commands to reload and restart docker**
   1. systemctl daemon-reload
   2. systemctl restart docker

NB: Expose necessary port in security group in aws so that master slave machines can communicate.

**Execute the following commands in the Master**

* kubeadm init

**In case, kubernetes cluster configuration located in $HOME/.kube/config then run the following command**

       To start using your cluster, run below commands:

mkdir -p $HOME/.kube

          sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

           sudo chown $(id -u):$(id -g) $HOME/.kube/config

          export KUBECONFIG=$HOME/.kube/config

NB: Add “export KUBECONFIG=$HOME/.kube/config” to ~/.bashrc file in order to export the conf file. If this is not added, the export will be applicable only to the current putty session. For a new session, the Kubernetes API will be unreachable.

**In case, kubernetes configuration located in $HOME/admin.conf then run the following command**

    To start using your cluster, run below commands as root user or prefix with sudo:

          cp /etc/kubernetes/admin.conf $HOME/

          chown $(id -u):$(id -g) $HOME/admin.conf

          export KUBECONFIG=$HOME/admin.conf

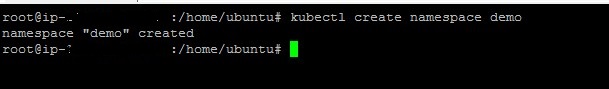
NB: Add “export KUBECONFIG=$HOME/admin.conf” to ~/.bashrc file in order to export the conf file. If this is not added, the export will be applicable only to the current putty session. For a new session, the Kubernetes API will be unreachable.

**Execute the following commands in the Slave**

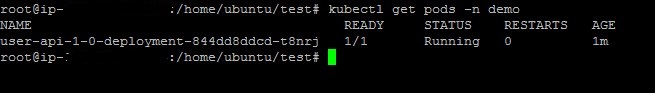
* kubeadm join --token <token id> <Master IP>:6443
* To check all the nodes had joined use the command in master node 🡪 kubectl get nodes.

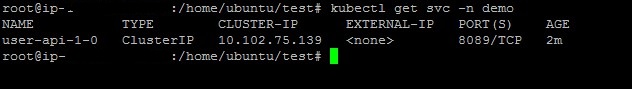
**Running the application:**

* First create a namespace with name demo using 🡪 **kubectl create namespace demo**.



* Run kubernetes scripts(can be found in project parent dir) to create pv,pvc,service,pod 🡪 **kubectl create –f user-api.yml**
* Check pods and services running





* Attach the created volumes only if you have nfs as in cluster, files should be shared among slave node.
* To check end to end application endpoint

