Juju Practical :-

★ Steps to setup juju & LXD :-

i) To install LXD (It will create 'mini-vm's' for cluster nodes) :-

⇒ sudo snap install lxd

⇒ sudo lxd init [It will ask some ques., just click enter to select default) [use

ii) Install juju :-

⇒ sudo snap install juju --classic

iii) Bootstrap juju (connect juju to LXD) :-

⇒ juju bootstrap localhost my-controller

⇒ juju add-model k8s

★ Steps to deploy K8's Cluster :-

i) Deploy the Base Cluster :-

⇒ juju deploy charmed-kubernetes

ii) To Scale it to 6 nodes :- (3 master, 3 worker)

⇒ (3 Master) juju add-unit kubernetes-control-plane -n 2

⇒ (3 Worker) juju add-unit kubernetes-worker -n 2

☆ To stop / destroy current model:-

⇒ juju destroy-model k8s --no-wait --force --destroy-storag

⇒ Then it will ask the name of model to destroy, enter k8s

☆ Upgrade & Update packages :-

=> sudo apt update

=> sudo apt upgrade -y

=> sudo apt install -y curl wget snapd unzip

☆ Install juju :-

=> sudo snap install juju --classic

☆ Install microk8s (to provide Kubernetes cluster) :-

=> sudo snap install microk8s --classic

=> sudo usermod -a -G microk8s ubuntu

=> newgrp microk8s

=> microk8s enable dns hostpath-storage helm3

☆ Install Docker (to creat kind k8s nodes) :-

=> sudo apt install -y docker.io

=> sudo systemctl start docker

=> sudo systemctl enable docker

=> sudo usermod -aG docker ubuntu

=> newgrp docker

=> docker version

☆ Install kubectl :-

⇒ Go to k8s official documentation find the command to install, kubectl i.e.

⇒ curl . . . . . kubectl

⇒ chmod +x kubectl

⇒ sudo mv kubectl /usr/local/bin/

☆ Install KIND (to create 6 node k8s cluster):-

⇒ find command at kind documentation,

curl . . . . . kind

⇒ chmod +x kind

⇒ sudo mv kind /usr/local/bin

☆ Create 6 node cluster (3 master, 3 worker):-

⇒ nano kind-6node.yaml

⇒ kind create cluster --name k8s-6node --config kind-6node.yaml

⇒ kubectl get nodes

find the comma

---- kubectl

/bin/

de K8s cluster):-

entation,

---- kind

bin

, 3 worker):-

ode --config

⇒ In the yaml file, write :-

```
kind : Cluster
apiVersion : kind.x-k8s.io/v1alpha4
nodes :
- role : control-plane
- role :    ,,    ,,
- role :    ,,    ,,
- role : worker
- role :    ,,
- role :    ,,
```

☆ To add K8s to Juju & Bootstrap Controller :-

⇒ juju add-k8s kind-k8s

⇒ juju bootstrap kind-k8s juju-controller

⇒ juju add-model k8s-model

⇒ juju switch k8s-model

⇒ juju models . (to verify).

☆ Deploy stateful DBs using juju :-

⇒ juju deploy postgresql-k8s postgresql

⇒ juju trust postgresql --scope=cluster

(For Postgresql)

⇒ Similarly for MySQL & Mongo DB.

Note: So, juju does not have other mentioned databases like MariadB, galera and ferrit DB in its CharmHub, so those dB can't be up. I am trying researching on that part to make them available too.

⇒ juju status (to verify).

☆ Verify all PVs are bounded:-

⇒ kubectl get pvc -n k8s-middle

☆ Backup & DR (Velero):-

⇒ Goto velero documentation, which will tell to go to latest github version repo of velero to install it on linux., so do that using curl.

⇒ curl . - - - - - - - - -

⇒ tar -xvf - - - - - - - - -

⇒ sudo mv velero-v*/velero /usr/local/bin/

-- wait

☆ Velero :-

⇒ Tool for backup & DR for K8s.

⇒ DBs are stateful, hence data shouldn't be lost. So, backup & DR is req., achieved by velero.

⇒ Can backup, pods, deployments, stateful sets, services, config Maps, Secrets, Juju CRDs, Namespaces, PVCs

⇒ Does not copy raw disk data, it records PVC metadata.

⇒ It ensures the same volume is reattached on restore.

☆ Steps of Backup & DR implementation :-

⇒ S3 bucket has been setup for backup of DBs.

⇒ For Backup :-

⇒ ~~no~~ name credentials-velero (For velero to access S3)

⇒ Add Access key id & secret.

⇒ Install ~~ve~~ velero now by :-

```
velero install \
-- provider aws \
--plugins velero/velero-plugin-for-aws:v1.9.0 \
-- bucket juju-velero-backup \
-- credentials-file ./credentials-velero \
-- backup-location-config region=ap-south-1(yours) \
-- use-volume-snapshots=false
```

-- wait

⇒ verify by, velero backup-location get

⇒ velero backup create db-backup --include-namespaces k8s-model
(this creates backup)

⇒ check from, velero backup get

⇒ Backup created.

⇒ For DR :- (To check if pod recovers or not)

⇒ delete pod by,
kubectl delete pod mysql-0 -n k8s-model

⇒ check by, kubectl get pods -n k8s-model

⇒ Now restore from backup,
velero restore create --from-backup db-backup

⇒ velero restore get (check if it is completed)

⇒ kubectl get pods -n k8s-model

⇒ juju status

⇒ Backup & DR are working now.

☆ To Scale th

⇒ Scale MySql

• juju scale-

• juju scale-

⇒ Then check

• juju statu

• juju

⇒ Delete a

• kubectl

☆ Application

⇒ Application
Virtual conne
enable th

⇒ First deplo

• juju dep

• juju d

• juju d

⇒ Create

• juju rel

☆ To Scale the Cluster with self-healing :-

⇒ Scale MySql & postgre pods :-

- juju scale-application mysql 3
- juju scale-application postgresql 3

⇒ Then check :-

- juju status
- juju run mysql/leader get-cluster-status

⇒ Delete a pod manually, check self-healing :-

- kubectl delete pod mysql-1 -n k8s-model

☆ Application relations, Secrets & Config. Management :-

⇒ Application relations :-
Virtual connection between Charms (juju applications) that enable them to automatically share data & configure each other.

⇒ First deploy an application :-

- juju deploy wordpress-k8s wordpress
- juju deploy mysql-k8s mysql

⇒ Create a relation :-

- juju relate wordpress mysql

⇒ Check :-

- juju status
- kubectl get nodes secrets -n k8s-model

⇒ Change configuration live :-

- juju config mysql profile=production.

⇒ Then app will restart,
config will be applied automatically,
& no need for manual secret handling.

⇒ Learnings :-

⇒ Docker was used because of KIND.

⇒ Kind creates K8's nodes as containers

⇒ Kind was used to create a multi-node K8s cluster on a single VM.

⇒ Kubectl was used to communicate to K8s, as K8s CLI.

⇒ Juju is used to manage applications, not infra.

⇒ Juju handles, deployment, scaling, Relations, secrets, config, day 2 operations.

⇒ Juju features used:-

i.) Juju Controller :- Central brain for juju

⇒ managed models, talked to k8s, tracked app state

ii.) Juju Models :- Logical separation of workloads.

⇒ Isolated k8s apps from controller logic, prevent accidental operations

iii.) Juju Kubernetes Operators (Charms):- to deploy complex apps easily.

⇒ Used charms like mysql-k8s, postgresql-k8s, mongodb-k8s, wordpress-k8s.

⇒ Handled clustering, databases, storage, restarts

iv.) Juju Relations :- To connect apps each other safely.

⇒ Create db, credentials, inject secrets without manual passwords or YAML.

v.) Juju Scaling :- To test HA (High Availability) & resilience

⇒ Scale the db units

⇒ PVCs were needed because they (DB) are stateful, so if pods die, data/state should be stored somewhere.

⇒ PVCs attached via k8s Storage Class

⇒ It stored dB data outside pods, will reattach data on pod restart.

⇒ Velero :-

- K8s does not have native backups, it used velero.
- Backed up the K8s objects, PVC references, Namespace state, all stored in S3 bucket.
- Does DR too, test by deleting any DB pod.