LDF Document -

[Kubernetes]

****

---

1. Introduction to Kubernetes Architecture
2. Basic Kubernetes objects
   > installation
3. Kubeadm, minikube, k3s, k8s

---

4. Hand-on Practice of k8s main components
5. Pods, ReplicaSet and Deployments
6. AutoScaling

---

7. Rolling updates and rollback (automation - eg. yaml file)

---

8. Config Maps and Secret

---

9. Services and Type of Services (Binding Services)

---

10. Role based access Control (RBAC)
    > Dashboard Setup
    > Example RBAC Binding (Somethings)

---

11. Kubernetes Volume
    >

---

12. Stateless and Stateful applications

13. Liveness, Readiness and Startup Probes
14. Helm and Helm Chart (Kubernetes Package Manager)

[kubectl commands explanation]

****

```
$ install minikube
$ minikube init
$ kubectl get pods
$ kubectl get po
$ kubectl get po -A
$ kubectl get pods - - namespace <namespace>
$ kubectl get nodes
$ kubectl get pods -A -o wide

$ kubectl logs <POD ID>
$ kubectl get events << the whole cluster events
$ kubectl get events --sort-by=.metadata.creationTimestamp
```

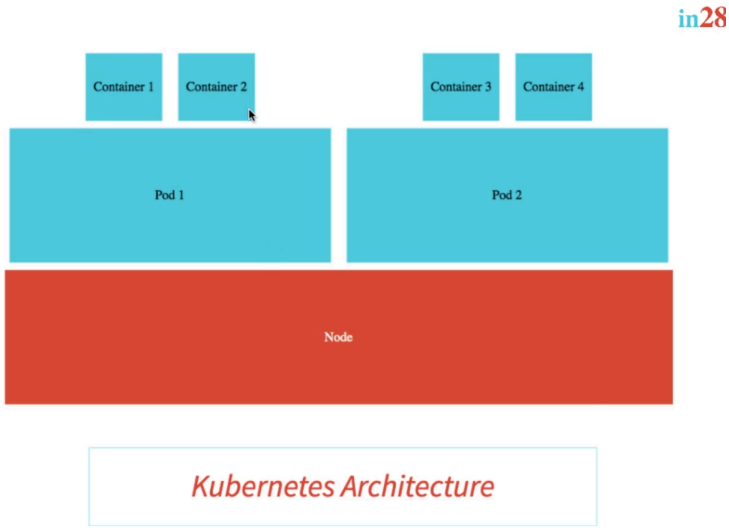
PODs, ReplicaSet, Deployments, Services

```
$ kubectl get nodes
$ kubectl get pods
$ kubectl get deployment
$ kubectl get replicaset
$ kubectl get services

$ kubectl explain pods
```

Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.

```
$ kubectl describe pod < pod name >
```



Kubernetes Architecture

ReplicaSet

```
$ kubectl scale deployment hello-world-rest-api --replicas=3
```

Desired State == Current State [pods numbers]

Deployment

```
$ kubectl set image deployment  hello-world-rest-api
hello-world-rest-api=DUMMY-IMAGE:TEST
```

```
$ kubectl create deployment hello-world-rest-api
--image=in28min/hello-world-api:0.0.1.RElEASE --replicas=3
```

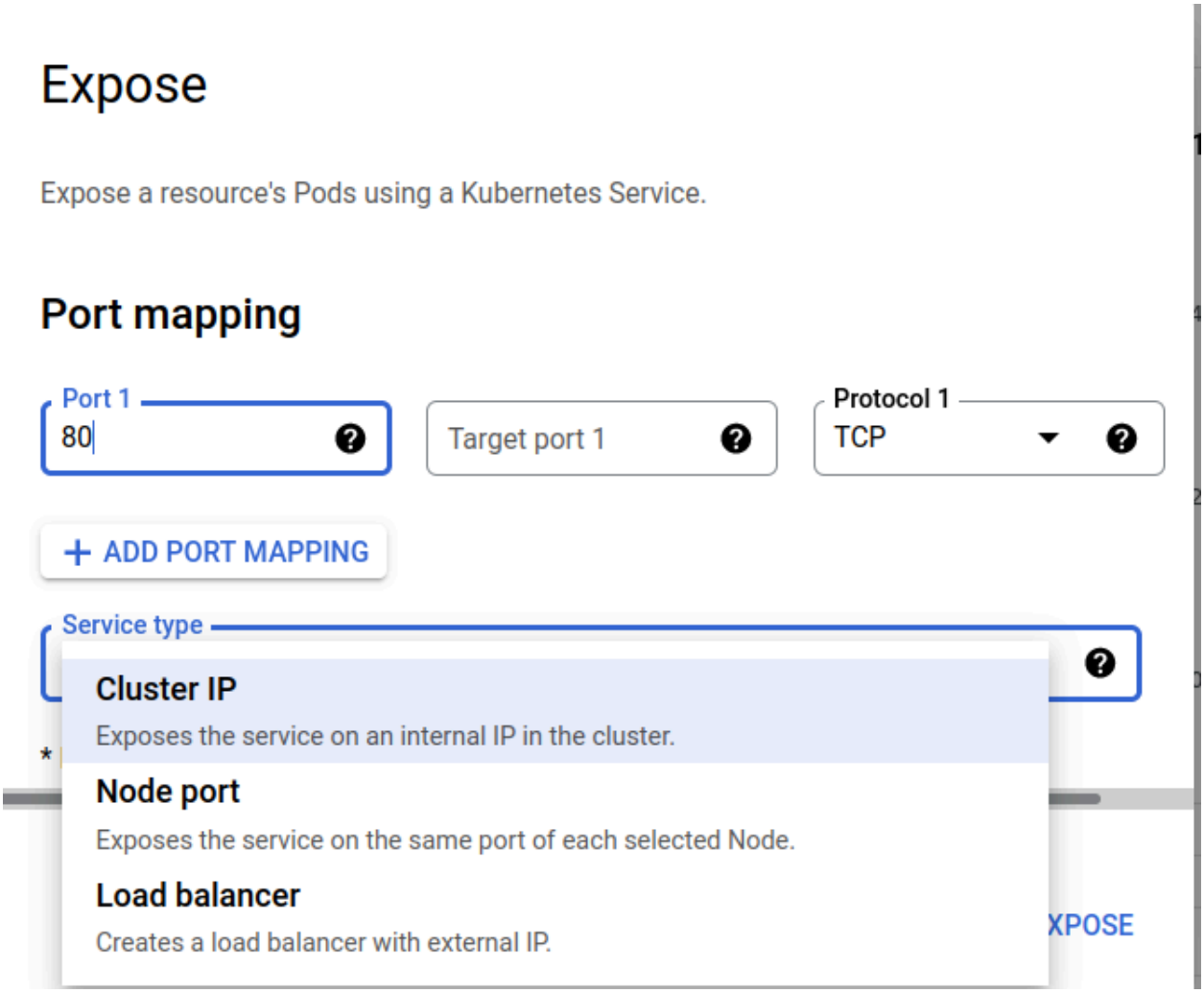
Although we set the image of the hello-world-rest-api deployment, the image is wrong. That is why the newly deployed image is not working in our desired deployment.

```
ranga@cloudshell:~ (solid-course-258105)$ kubectl get pods
NAME                                  READY   STATUS             RESTARTS   AGE
hello-world-rest-api-58ff5dd898-5x4vl   1/1     Running            0          14m
hello-world-rest-api-58ff5dd898-vn7d6   1/1     Running            0          16m
hello-world-rest-api-58ff5dd898-zp2nq   1/1     Running            0          14m
hello-world-rest-api-85995ddd5c-msjsm   0/1     InvalidImageName   0          2m6s
ranga@cloudshell:~ (solid-course-258105)$
```

```
st-api-85995ddd5c-msjsm to gke-in28minutes-cluster-default-pool-19eb1ee2-g1xw
2m28s       Warning   InspectFailed             Pod          Failed to apply default image tag "DUMMY_IMA
GE:TEST": couldn't parse image reference "DUMMY_IMAGE:TEST": invalid reference format: repository name mu
st be lowercase
2m28s       Warning   Failed                    Pod          Error: InvalidImageName
4m17s       Normal    SandboxChanged            Pod          Pod sandbox changed, it will be killed and r
e-created.
ranga@cloudshell:~ (solid-course-258105)$
```

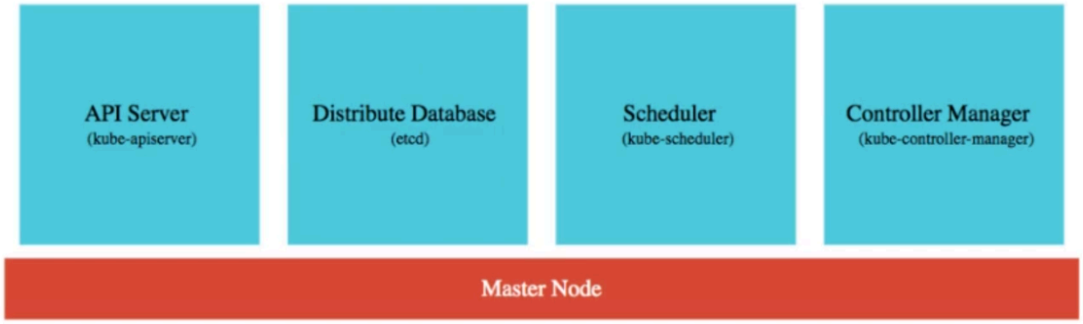
```
$ kubectl apply -f ./my-manifest.yaml
$ kubectl apply -f ./my1.yaml -f ./my2.yaml
$ kubectl apply -f ./dir < create kube object from manifest dir>
```

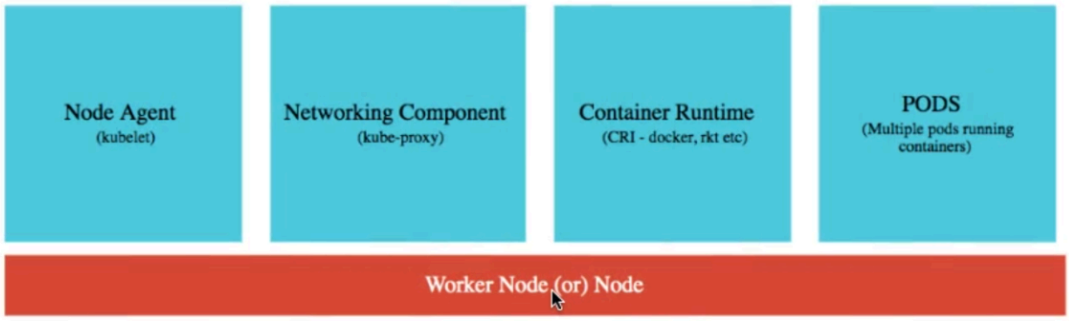
Services



The services are actually not belonging to the related deployment. It belongs only with pods which are running for our deployment.

Master and Worker Nodes





Master Nodes Components

API Server (kube-api-server)
Distributed Database (Etcd)
Scheduler (kube-scheduler)
Controller Manager (kube-controller Manager)

Worker Nodes Components

Node Agent (Kubelet)
Networking Component (kube-proxy)
Container Runtime (CRI - Docker, rkt, etc.)
PODs (Multiple Pods Running Containers)

What if the master node goes down, the worker nodes and their applications go down?

The answer is NO ! . The applications inside the pods and their relative nodes will continue to work.

```
$ kubectl get componentstatuses
```

```
ranga@cloudshell:~ (solid-course-258105)$ kubectl get componentstatuses
NAME                 STATUS    MESSAGE              ERROR
etcd-0               Healthy   {"health": "true"}
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-1               Healthy   {"health": "true"}
ranga@cloudshell:~ (solid-course-258105)$
```

- Install Google Cloud SDK and kubectl to connect your local machine and Google Kubernetes Engine

```
rangaraokaranam$ kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.3", GitCommit:
"5e53fd6bc17c0dec8434817e69b04a25d8ae0ff0", GitTreeState:"clean", BuildDate:"2019-06
-06T01:44:30Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"darwin/amd64"}
Unable to connect to the server: dial tcp 192.168.99.105:8443: i/o timeout
rangaraokaranam$ gcloud container clusters get-credentials in28minutes-cluster --zon
e us-central1-a --project solid-course-258105
Fetching cluster endpoint and auth data.
kubeconfig entry generated for in28minutes-cluster.
rangaraokaranam$ kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.3", GitCommit:
"5e53fd6bc17c0dec8434817e69b04a25d8ae0ff0", GitTreeState:"clean", BuildDate:"2019-06
-06T01:44:30Z", GoVersion:"go1.12.5", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"13+", GitVersion:"v1.13.11-gke.9", Gi
tCommit:"6c1e92d07f5717440f751666d4aad6943015d3cb", GitTreeState:"clean", BuildDate:
"2019-10-11T23:14:17Z", GoVersion:"go1.11.13b4", Compiler:"gc", Platform:"linux/amd6
4"}
rangaraokaranam$
```

Initialize gcloud sdk

```
lucas@SecXX:~$ gcloud init
Welcome! This command will take you through the configuration of gcloud.

Your current configuration has been set to: [default]

You can skip diagnostics next time by using the following flag:
  gcloud init --skip-diagnostics

Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).

You must log in to continue. Would you like to log in (Y/n)?  Y

Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&scope=openid+https%3A%2F%2Fwww
.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2F
auth%2Fsqlservice.login+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=nREXwvlJimvCdVHu2VuuflTKgV88vg&access_type=offline&c
ode_challenge=X7YRUUjgop-eZW5XzcbratiZCV1bE2TxMrZhH32A3fc&code_challenge_method=S256

You are logged in as: [server5cloud.kst@gmail.com].

Pick cloud project to use:
 [1] fit-bulwark-358207
 [2] versatile-now-358216
 [3] zayy-wal
 [4] Enter a project ID
 [5] Create a new project
Please enter numeric choice or text value (must exactly match list item):
Please enter a value between 1 and 5, or a value present in the list:  2

Your current project has been set to: [versatile-now-358216].
```

Rollout and Rollback Deployment

```
$ kubectl rollout history deployment <deployment-name>
```

```
rangaraokaranam$ kubectl rollout history deployment hello-world-rest-api
deployment.extensions/hello-world-rest-api
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

rangaraokaranam$
```

```
$ kubectl set image deployment <deployment-name>
deploymentname=id/image-name:release --record=true
```

```
rangaraokaranam$ kubectl set image deployment hello-world-rest-api hello-world-rest-
api=in28min/hello-world-rest-api:0.0.3.RELEASE --record=true
deployment.extensions/hello-world-rest-api image updated
rangaraokaranam$
```

--record=true (This option will record your execution or changes which can easily be traced back to our rollout history.

```
rangaraokaranam$ kubectl rollout history deployment hello-world-rest-api
deployment.extensions/hello-world-rest-api
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
3         kubectl set image deployment hello-world-rest-api hello-world-rest-api=in2
8min/hello-world-rest-api:0.0.3.RELEASE --record=true

rangaraokaranam$
```

We can handle the rollout and rollback process easily by changing REVISION No.

```
$ kubectl rollout undo deployment <deployment-name>
--to-revision=(REVISION.NO)
```

```
rangaraokaranam$ kubectl rollout undo deployment hello-world-rest-api --to-revision=
1
deployment.extensions/hello-world-rest-api rolled back
```

As we saw the following figure.

```
rangaraokaranam$ kubectl rollout history deployment hello-world-rest-api
deployment.extensions/hello-world-rest-api
REVISION  CHANGE-CAUSE
2         <none>
3         kubectl set image deployment hello-world-rest-api hello-world-rest-api=in2
8min/hello-world-rest-api:0.0.3.RELEASE --record=true
4         <none>

rangaraokaranam$
```

We rolled back to the REVISION No 1. That's why No.1 disappeared and No.4, the new release is upcoming.

Generate Yaml Configuration for Deployment

```
$ kubectl get deployment <deployment-name>
$ kubectl get deployment <deployment-name> -o yaml
$ kubectl get deployment <deployment-name> -o yaml > dpl.yaml

$ kubectl get service <deployment-name> -o yaml > dpl-svc.yaml
$ kubectl get all  << all components running beneath Default namespace
$ kubectl get all -o wide
```

```
rangaraokaranam$ kubectl delete all -l app=hello-world-rest-api
pod "hello-world-rest-api-58ff5dd898-6ctr2" deleted
pod "hello-world-rest-api-58ff5dd898-m9cwn" deleted
service "hello-world-rest-api" deleted
deployment.apps "hello-world-rest-api" deleted
rangaraokaranam$
```

Delete all pods, service, deployment related to hello-world-rest-api LABELs.

```
$ kubectl delete all -l app=<LABELs Name want to be Deleted>
```

Lets practice,

Previously, We just exposed our running deployment in yaml format. So, modify those yaml for deployment and services and apply again for testing. Lets see the example demonstration.

```
rangaraokaranam$ kubectl apply -f deployment.yaml
deployment.extensions/hello-world-rest-api created
service/hello-world-rest-api created
rangaraokaranam$ kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-58ff5dd898-czzlx   1/1     Running   0          19s
pod/hello-world-rest-api-58ff5dd898-f4kv8   1/1     Running   0          20s

NAME                           TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
      AGE
service/hello-world-rest-api   LoadBalancer   10.0.9.221    <pending>     8080:32208/
TCP   19s
service/kubernetes             ClusterIP      10.0.0.1      <none>        443/TCP
      50m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world-rest-api    2/2     2            2           21s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-58ff5dd898   2         2         2       21s
```

```
$ kubectl apply -f deployment.yaml
```

Sometimes, We want to trace and watch live for the services or deployment what they are doing. By using --watch option. Lets see the demonstration.

```
$ kubectl get svc --watch
```

```
rangaraokaranam$ kubectl get svc --watch
NAME                  TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)
 AGE
hello-world-rest-api  LoadBalancer   10.0.9.221    35.193.152.162   8080:32208/TCP
 73s
kubernetes            ClusterIP      10.0.0.1      <none>           443/TCP
 51m
```

Benefit of curl / client URL

Curl is a developer tool used to transfer from and to a server. Mostly retrieved html source content from a web server.

```
ranga@cloudshell:~ (solid-course-258105)$ curl 35.193.152.162:8080/hello-world
Hello World  V3 x6gqwranga@cloudshell:~ (solid-course-258105)$ watch curl http://35.193.152.162:8080/hell
o-world
```

```
(solid-course-258105) ×
Every 2.0s: curl h...  cs-6000-devshell-vm-9b36b50a-054d-43b8-b2ff-3ec5fba01bfc: Tue Jan 28 19:00:21 2020

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0 100    21  100    21    0
    0     56      0 --:--:-- --:--:-- --:--:--    56
Hello World  V1 f4kv8
```

```
$ watch curl http://domain:port/sub-url
```

Sometimes, We modify and save yaml configuration. But We couldn't trace changes before and after changing yaml. Lets see the diff option usage in kubectl CMD.

```
$ kubectl diff -f deployment.yaml
```

```
^Crangaraokaranam$ kubectl diff -f deployment.yaml
diff -u -N /var/folders/y_/x4jdvdkx7w94q5qsh745gzz00000gn/T/LIVE-784808359/extension
s.v1beta1.Deployment.default.hello-world-rest-api /var/folders/y_/x4jdvdkx7w94q5qsh7
45gzz00000gn/T/MERGED-873255642/extensions.v1beta1.Deployment.default.hello-world-re
st-api
--- /var/folders/y_/x4jdvdkx7w94q5qsh745gzz00000gn/T/LIVE-784808359/extensions.v1bet
```

```
PullPolicy":"IfNotPresent","name":"hello-world-rest-api"}],"restartPolic
"terminationGracePeriodSeconds":30}}}}
    creationTimestamp: "2020-01-28T13:28:17Z"
-   generation: 1
+   generation: 2
    labels:
      app: hello-world-rest-api
    name: hello-world-rest-api
@@ -16,7 +16,7 @@
    uid: 0b7949bb-41d2-11ea-8e69-42010a80009c
  spec:
    progressDeadlineSeconds: 2147483647
-   replicas: 2
+   replicas: 3
    revisionHistoryLimit: 2147483647
    selector:
      matchLabels:
exit status 1
```

As we see the changes, we actually changed replicas count into three by annotating plus signs. Diff option can render the output that shows the differences before and after changing configuration yaml.

Labels And Selector

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world-rest-api
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: hello-world-rest-api
    spec:
```

Before we've talked about Labels and Selector. Lets see rolling update strategy in k8s cluster.

MaxSurge -
MaxUnavailable -

```
spec:
  replicas: 3
  minReadySeconds: 45
  selector:
    matchLabels:
      app: hello-world-rest-api
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

minReadySeconds : 45 << 45 seconds will take while the new releases are deploying.

Replica-Set in Depth

With a replica set, it cannot handle release. That is why we need deployment to rollout or rollback for handling releases easily.

```yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  labels:
    app: hello-world-rest-api
  name: hello-world-rest-api
  namespace: default
spec:
  replicas: 3
  minReadySeconds: 45
  selector:
    matchLabels:
      app: hello-world-rest-api
```

As above configuration, We changed the kind of yaml from Deployment to ReplicaSet. After that,

```yaml
    containers:
    - image: in28min/hello-world-rest-api:0.0.2.RELEASE
      imagePullPolicy: IfNotPresent
      name: hello-world-rest-api
    restartPolicy: Always
    terminationGracePeriodSeconds: 30
---
```

To test release with ReplicaSet, set the image of deployment to new version and apply that configuration again.

```
rangaraokaranam$ kubectl apply -f deployment.yaml
replicaset.extensions/hello-world-rest-api configured
service/hello-world-rest-api unchanged
rangaraokaranam$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-world-rest-api-dkbgk    1/1     Running   0          3m29s
hello-world-rest-api-hj62t    1/1     Running   0          3m29s
hello-world-rest-api-zgjtv    1/1     Running   0          3m29s
rangaraokaranam$
```

See, the new changes in ReplicaSet will not be affected. Because the ReplicaSet couldn't handle the release process. There is only one way to force release into new changes. That is deleting pods.

```
rangaraokaranam$ kubectl delete pod hello-world-rest-api-dkbgk
pod "hello-world-rest-api-dkbgk" deleted
rangaraokaranam$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-world-rest-api-hj62t    1/1     Running   0          4m10s
hello-world-rest-api-pjvk6    1/1     Running   0          6s
hello-world-rest-api-zgjtv    1/1     Running   0          4m10s
rangaraokaranam$
```

After we've deleted, the replica set will determine that if the current state and desired state of running pods were not equal, it will read again our new configuration and apply it with our new changes. As we see the above photo, the highlight one is running with our new yaml configuration which has already set the new image.

```
Every 2.0s: curl h...  cs-6000-devshell-vm-9b36b50a-054d-43b8-b2ff-3ec5fba01bfc: Tue Jan 28 19:40:34 2020

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0 100    21  100    21    0
    0     55      0 --:--:-- --:--:-- --:--:--    55
Hello World  V3 zgjtv
```

```
Every 2.0s: curl h...  cs-6000-devshell-vm-9b36b50a-054d-43b8-b2ff-3ec5fba01bfc: Tue Jan 28 19:42:36 2020

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0c100    21  100    21    0
    0     54      0 --:--:-- --:--:-- --:--:--    54
Hello World  V2 pjvk6
```

We haven't deleted all the pods. So, the three running pods are not equal versions. Only one pod is running with V3 and the left two are with V2 as we see the above curl watching output

Configure Multiple Kubernetes Deployments with one service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: hello-world-rest-api
    version: v1
  name: hello-world-rest-api-v1
  namespace: default
spec:
  replicas: 2
  minReadySeconds: 45
```

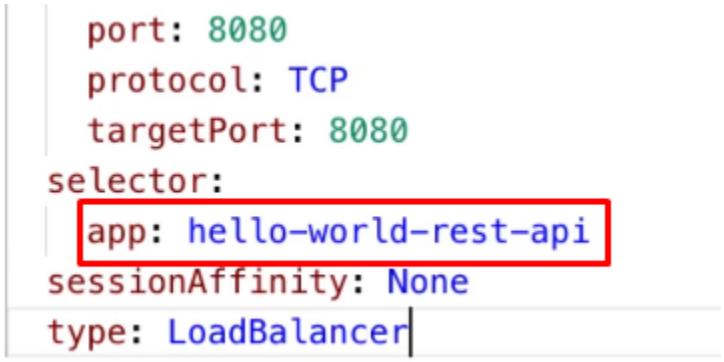
Lets see we have two deployments belonging to one service.

```
        app: hello-world-rest-api
        version: v1
    spec:
      containers:
      - image: in28min/hello-world-rest-api:0.0.1.RELEASE
        imagePullPolicy: IfNotPresent
        name: hello-world-rest-api
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: hello-world-rest-api
    version: v2
  name: hello-world-rest-api-v2
  namespace: default
```

We wanted to balance load between version 1 and version 2 of deployments.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-world-rest-api
  name: hello-world-rest-api
  namespace: default
spec:
  ports:
  - nodePort: 32208
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: hello-world-rest-api
    version: v2
  sessionAffinity: None
  type: LoadBalancer
```

If we didn't change the selector for which version of deployment, The LoadBalancer service will select all versions of hello-world-rest-api deployment. The main fact is we are considering two deployments with one service.



As in the above photo, the selector wasn't defined with a specific version of deployment. So, It will take effect on all versions of deployments.

Delete Deployment using LABELS

$ kubectl delete all -l app=<LABEL>

```
rangaraokaranam$ kubectl delete all -l app=hello-world-rest-api
pod "hello-world-rest-api-v1-797dd4b5dc-7sjcv" deleted
pod "hello-world-rest-api-v1-797dd4b5dc-kksz6" deleted
pod "hello-world-rest-api-v2-549fbbbf7c-4ckqt" deleted
pod "hello-world-rest-api-v2-549fbbbf7c-cmqwc" deleted
service "hello-world-rest-api" deleted
deployment.apps "hello-world-rest-api-v1" deleted
deployment.apps "hello-world-rest-api-v2" deleted
rangaraokaranam$
```

Now, let's deep dive into microservices.

Deep Dive to Microservices

```
spec:
  containers:
  - name: currency-exchange
    image: in28min/currency-exchange-devops
    imagePullPolicy: IfNotPresent
    ports:
    - name: liveness-port
      containerPort: 8000
    resources: #CHANGE
      requests:
        cpu: 100m
        memory: 512Mi
      limits:
        cpu: 500m
        memory: 1024Mi #256Mi
    readinessProbe:
      httpGet:
        path: /
        port: liveness-port
      failureThreshold: 5
      periodSeconds: 10
      initialDelaySeconds: 60
    livenessProbe:
      httpGet:
        path: /
        port: liveness-port
      failureThreshold: 5
      periodSeconds: 10
      initialDelaySeconds: 60
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
```

1.
2.
3.
4.
5.

- FailureThreshold : 5 (Acceptable failure counts)
- periodSecond : 10 (Check Container Health in every 10 seconds)
- initialDelaySeconds: 60 (Wait 60 seconds for initial stage to start container)

Lets dive into the yaml configuration.

No.1 - We exposed container port 8000

No.2 - Request the resource 100m of CPU and 512Mb of Memory.

No.3 - To check the container health at the startup time to ready

No.4 - To check the container health at the up and running time.

```
.u.e.EnvironmentConfigurationLogger : CURRENCY_CONVERSION_SERVICE_HOST - 10.0.4.7
2020-01-28 15:38:24.155  INFO [currency-conversion,,,] 1 --- [           main] i.
.u.e.EnvironmentConfigurationLogger : CURRENCY_EXCHANGE_PORT_8000_TCP_PROTO - tcp
2020-01-28 15:38:24.158  INFO [currency-conversion,,,] 1 --- [           main] i.
.u.e.EnvironmentConfigurationLogger : LD_LIBRARY_PATH - /usr/lib/jvm/java-1.8-ope
k/jre/lib/amd64/server:/usr/lib/jvm/java-1.8-openjdk/jre/lib/amd64:/usr/lib/jvm/j
-1.8-openjdk/jre/../lib/amd64
2020-01-28 15:38:24.161  INFO [currency-conversion,,,] 1 --- [           main] i.
.u.e.EnvironmentConfigurationLogger : CURRENCY_EXCHANGE_SERVICE_HOST - 10.0.4.155
2020-01-28 15:38:24.161  INFO [currency-conversion,,,] 1 --- [           main] i.
```

Whenever we check the logs of running containers, we can see some environment variables that are automatically created by kubernetes service. The fact is that fetched the service name and followed by their related configuration. That is why it comes out with its own environment variables.

```
rangaraokaranam$ kubectl get svc
NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP      PORT
GE
currency-conversion  LoadBalancer   10.0.4.72     35.239.203.172   8100
m59s
currency-exchange    LoadBalancer   10.0.4.155    35.193.152.162   8000
m42s
kubernetes           ClusterIP      10.0.0.1      <none>           443/
h5m
rangaraokaranam$ CURRENCY_EXCHANGE_SERVICE_HOST
```

As we see, the service name was currency-exchange. So, to form environment variables, it changes the Capital letter of the service name and replaces the hyphen instead of the dash followed by its specific requirement variable name. (CURRENCY_EXCHANGE_SERVICE_HOST)

Kubernetes Service Discovery

We set up the currency-exchange and currency-conversion example project in k8s cluster with two deployments and one service to talk to each respective container.

These two containers need to fix the problem when the currency conversion service is launched. let's say the currency exchange service was not available, and a little while later, the currency exchange service was started up.

In that kind of situation, kubernetes will not provide the currency exchange service information to the currency conversion service, and that's not cool.

How do we avoid that? The way we can avoid that is by configuring a specific URl, as the URL for currency exchange service, inside the currency conversion service. Let's see.

```
containers:
- image: in28min/currency-conversion:0.0.1-RELEASE #CHANGE
  imagePullPolicy: IfNotPresent
  name: currency-conversion
  env:
    - name: CURRENCY_EXCHANGE_SERVICE_HOST
      value: http://currency-exchange
restartPolicy: Always
terminationGracePeriodSeconds: 30
```

We have to set the value of CURRENCY_EXCHANGE_SERVICE_HOST to http://currency-exchange. Where does it come from? Right Here. it comes from the service name.

```
rangaraokaranam$ kubectl get svc
NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP      PORT
GE
currency-conversion  LoadBalancer   10.0.4.72     35.239.203.172   8100
m59s
currency-exchange    LoadBalancer   10.0.4.155    35.193.152.162   8000
m42s
kubernetes           ClusterIP      10.0.0.1      <none>           443/
h5m
rangaraokaranam$ CURRENCY_EXCHANGE_SERVICE_HOST
```

A more reliable approach than depending on the environment variables that are injected in. This URL would always be available, as soon as a currency exchange service instance comes up. This URL would get active, and the currency conversion service would be able to talk to the currency exchange service.

How does it work?

Simple !

Kubernetes has already provided the automated DNS. As soon as a service starts up in kubernetes, it will register with a kubernetes DNS and then you can find out the addresses of all the currency exchange instances, by sending a request to this specific URL. The currency exchange link and there is the name of the service. See the previous photos as we mentioned earlier.

We don't really need to do anything special for service discovery. The other interesting thing is when we use this kind of URL, we will also get the Load Balancing for free.

- To test Service Discovery
- To test Load Balancing between containers

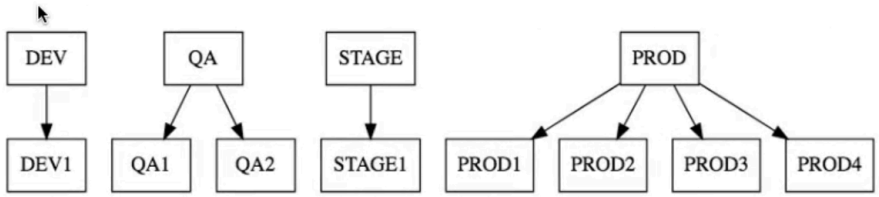
Centralized Configuration Management / Configmap

Config map allows us to decouple the configuration details from the container image. Using configmaps, we pass the configuration data as key-value pairs, which are consumed by PODs or any other system components and controllers, in the form of environment variables, sets of commands and arguments, or volumes.

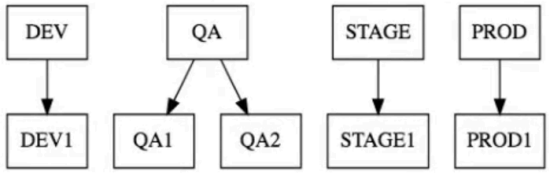
We can create ConfigMaps from literal values, from configuration files, from one or more files or directories.

Example creation from command line.

```
$ kubectl  create configmap my-config --from-literal=key1=value1
--from-literal=key2=value2
```



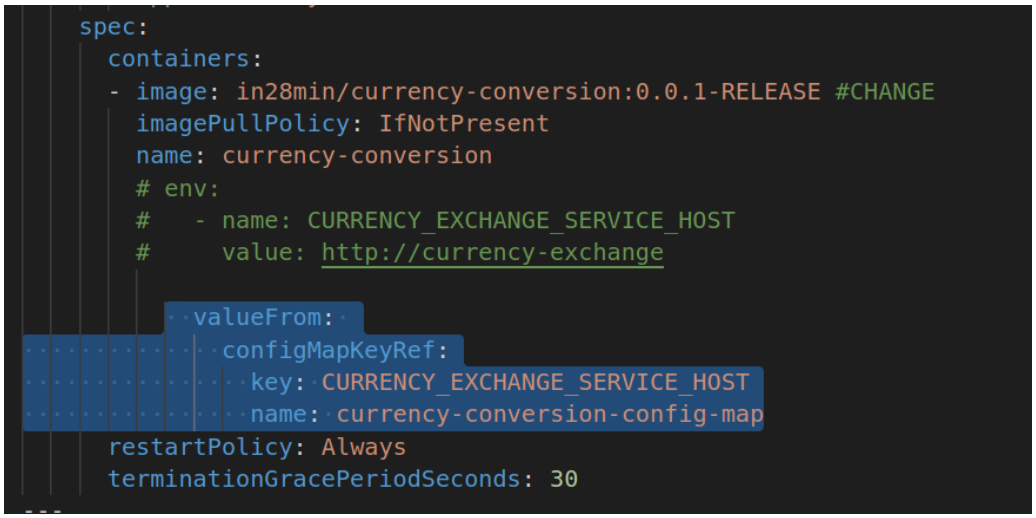
Currency Exchange Service



Currency Conversion Service

As we see in the above photos, we have many environments of deployment. That is why we are going to need to set the environment variable with the config

map. The aim is we don't want to hard code the configuration variable in every deployment.

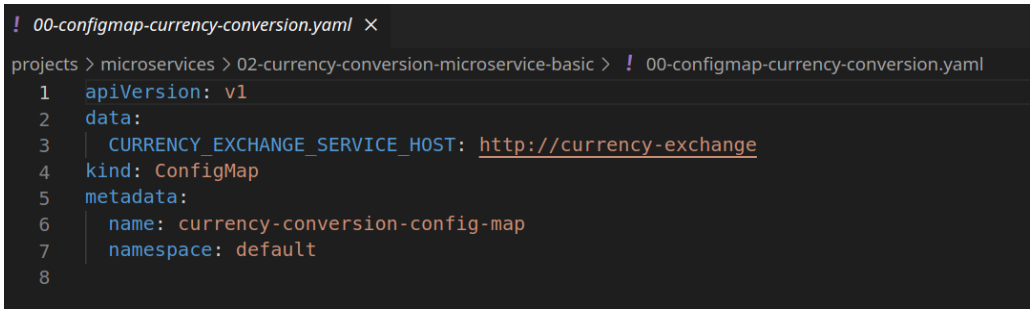


```
spec:
  containers:
  - image: in28min/currency-conversion:0.0.1-RELEASE #CHANGE
    imagePullPolicy: IfNotPresent
    name: currency-conversion
    # env:
    #   - name: CURRENCY_EXCHANGE_SERVICE_HOST
    #     value: http://currency-exchange

    valueFrom:
      configMapKeyRef:
        key: CURRENCY_EXCHANGE_SERVICE_HOST
        name: currency-conversion-config-map
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
```

The currency-conversion container needs to set the env variables CURRENCY_EXCHANGE_SERVICE_HOST. In the above photo, the highlight part is getting environment variables from the config map named CURRENCY_EXCHANGE_SERVICE_HOST.

So we're going to need configmap deployment yaml file. Right here.



00-configmap-currency-conversion.yaml

projects > microservices > 02-currency-conversion-microservice-basic > 00-configmap-currency-conversion.yaml

```
1  apiVersion: v1
2  data:
3    CURRENCY_EXCHANGE_SERVICE_HOST: http://currency-exchange
4  kind: ConfigMap
5  metadata:
6    name: currency-conversion-config-map
7    namespace: default
8
```

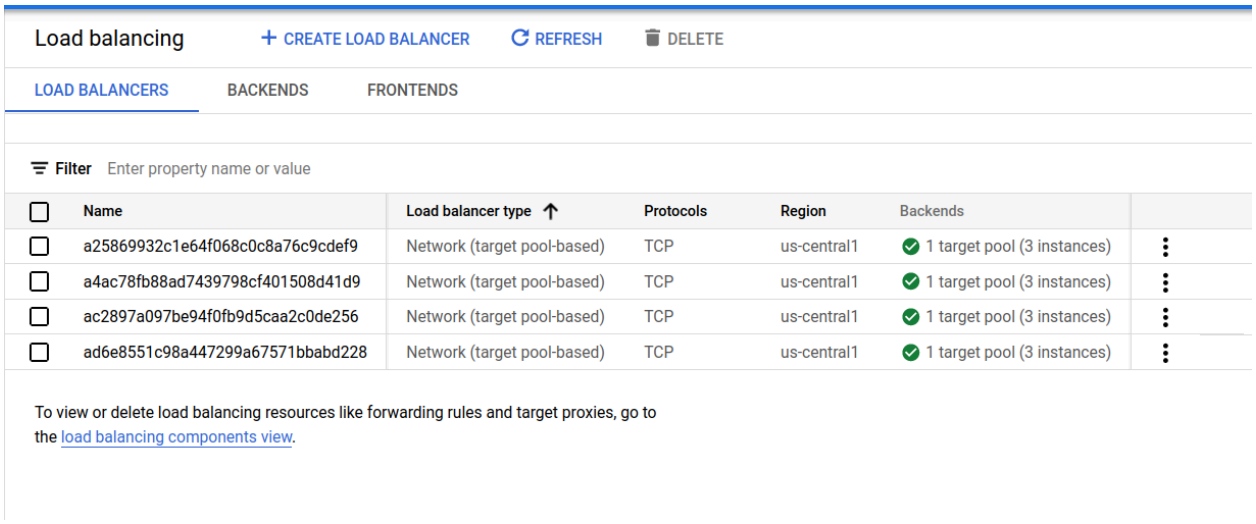
To Be Continued ....

Ingress Controller

Do We want to have 100 load balancers for each 100 pods? The answer is NO.

Because the facts are, load balancers are very expensive. We need more centralized service that can route the external request to the appropriate microservices inside the cluster.

That is why the kubernetes provided an ingress controller to allow incoming connection with reliable routing and load balancing.

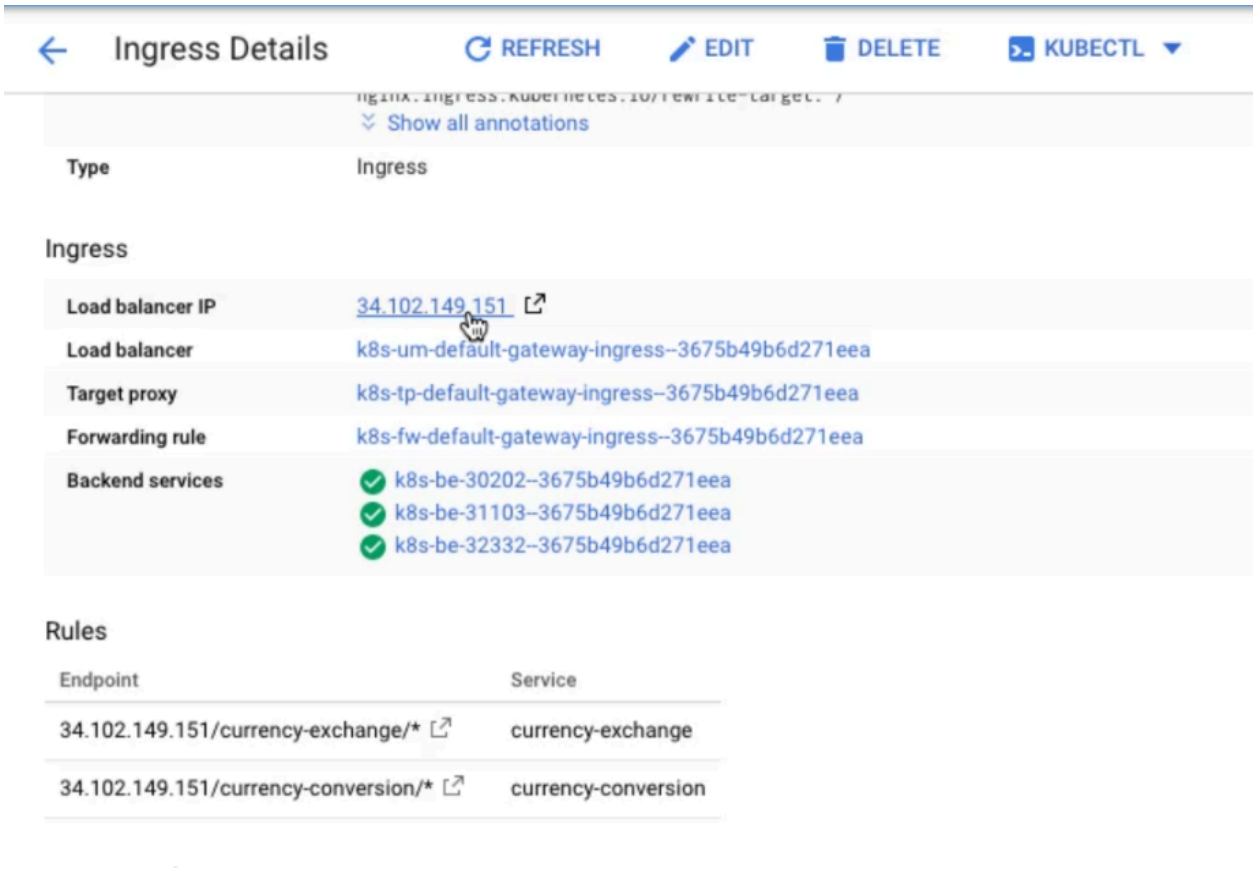


projects > microservices > ! ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gateway-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /currency-exchange/
        pathType: Prefix
        backend:
          service:
            name: currency-exchange
            port:
              number: 8000
      - path:  /currency-conversion/
        pathType: Prefix
        backend:
          service:
            name: currency-conversion
            port:
              number: 8100
```

In the above photo, the ingress controller yaml configuration file. There are two rules if the end user calls with the currency-exchange url the ingress rules let u in the currency-exchange service that is their related services.

If the end user calls with the currency-conversion url , the ingress rules let u into the currency-conversion service that is their related services.



Before you begin, switch your services type LoadBalancer to the NodePort and the upcoming ingress rules will route with the NodePort.

We will need no more load balancers. The ingress will provide route and load balancer problems.

If you want to add another microservice to the ingress rules. Just write the rules for that service and just apply the configured ingress configuration file.

ETCd

Etcd is a distributed reliable key-value store that is simple, secure and fast. And it is also a database that stores information regarding the cluster, such as nodes, pods, configs, secrets, accounts, roles, bindings and others. in a key-value format. Key=Value.

Every information you see when you run the kubectl get command is from the ETCD database. ETCD starts a service that listens on port 2379 by default.

Etcd is a leader-based distributed system. Ensure that the leader periodically sends heartbeats on time to all followers to keep the cluster stable.
Operating etcd clusters for Kubernetes

Two types of etcd backing up

Built-in snapshot

etcd supports built-in snapshot. A snapshot may either be taken from a live member with the `etcdctl snapshot save` command or by copying the `member/snap/db` file from an etcd data directory that is not currently used by an etcd process. Taking the snapshot will not affect the performance of the member.

Below is an example for taking a snapshot of the keyspace served by `$ENDPOINT` to the file `snapshotdb`:

```
ETCDCTL_API=3 etcdctl --endpoints $ENDPOINT snapshot save snapshotdb
```

Verify the snapshot:

```
ETCDCTL_API=3 etcdctl --write-out=table snapshot status snapshotdb
```

```
+----------+----------+------------+------------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE |
+----------+----------+------------+------------+
| fe01cf57 |       10 |          7 | 2.1 MB     |
+----------+----------+------------+------------+
```

Volume snapshot

If etcd is running on a storage volume that supports backup, such as Amazon Elastic Block Store, back up etcd data by taking a snapshot of the storage volume.

Snapshot using etcdctl options

We can also take the snapshot using various options given by etcdctl. For example

```
ETCDCTL_API=3 etcdctl -h
```

will list various options available from etcdctl. For example, you can take a snapshot by specifying the endpoint, certificates etc as shown below:

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> \
  snapshot save <backup-file-location>
```

where `trusted-ca-file`, `cert-file` and `key-file` can be obtained from the description of the etcd Pod.

Note:
If any API servers are running in your cluster, you should not attempt to restore instances of etcd. Instead, follow these steps to restore etcd:

- stop *all* API server instances
- restore state in all etcd instances
- restart all API server instances

We also recommend restarting any components (e.g. `kube-scheduler`, `kube-controller-manager`, `kubelet`) to ensure that they don't rely on some stale data. Note that in practice, the restore takes a bit of time. During the restoration, critical components will lose leader lock and restart themselves.

```
[root@kubemaster ~]# kubectl get pods -n kube-system
NAME                                 READY   STATUS    RESTARTS   AGE
coredns-f9fd979d6-9qgt7              1/1     Running   4          64d
coredns-f9fd979d6-h49kc              1/1     Running   4          64d
etcd-kubemaster                      1/1     Running   2          7h29m
kube-apiserver-kubemaster            1/1     Running   8          31h
kube-controller-manager-kubemaster   1/1     Running   40         64d
kube-flannel-ds-44b7c                1/1     Running   22         64d
kube-flannel-ds-tp64c                1/1     Running   2          64d
kube-flannel-ds-wp7qk                1/1     Running   5          64d
kube-proxy-5lx62                     1/1     Running   22         64d
kube-proxy-9hw5w                     1/1     Running   5          64d
kube-proxy-flxzs                     1/1     Running   2          64d
kube-scheduler-kubemaster            1/1     Running   28         25d
[root@kubemaster ~]# _
```

Firstly, check if the etcd pod is running inside the kube-system namespace.

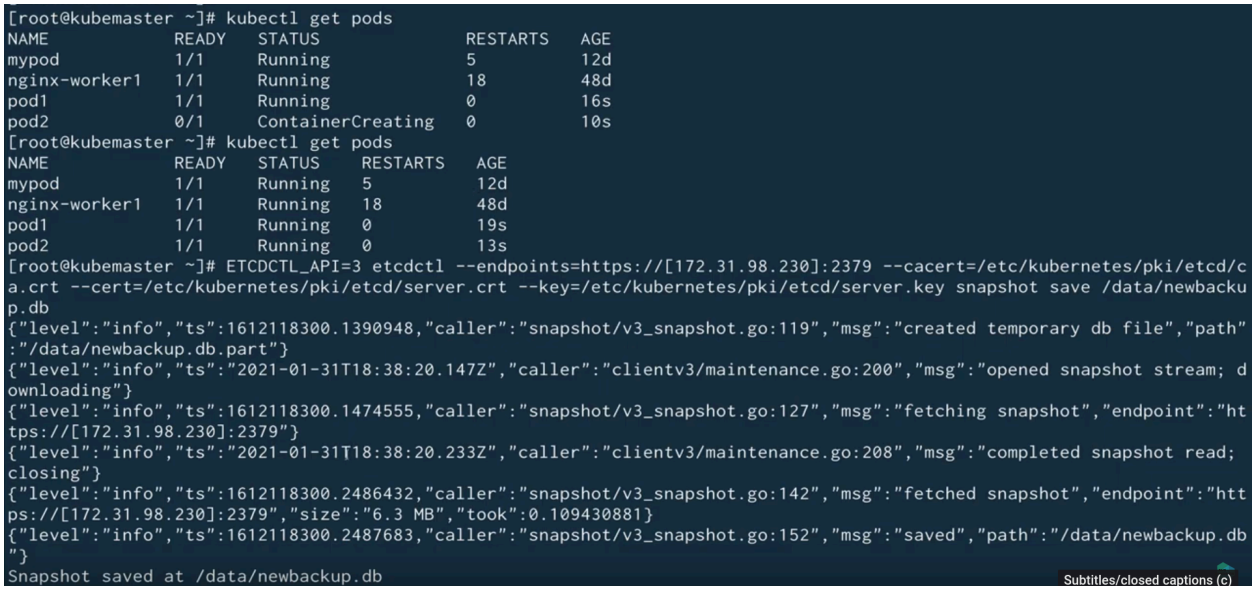
$ kubectl get pods -n kube-system

```
[root@kubemaster ~]# kubectl describe pod etcd-kubemaster -n kube-system
```

And describe etcd-kubemaster pod

```
  Command:
    etcd
    --advertise-client-urls=https://172.31.98.230:2379
    --cert-file=/etc/kubernetes/pki/etcd/server.crt
    --client-cert-auth=true
    --data-dir=/var/lib/etcd
    --initial-advertise-peer-urls=https://172.31.98.230:2380
    --initial-cluster=kubemaster=https://172.31.98.230:2380
    --key-file=/etc/kubernetes/pki/etcd/server.key
    --listen-client-urls=https://127.0.0.1:2379,https://172.31.98.230:2379
    --listen-metrics-urls=http://127.0.0.1:2381
    --listen-peer-urls=https://172.31.98.230:2380
    --name=kubemaster
    --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    --peer-client-cert-auth=true
    --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    --snapshot-count=10000
    --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
  State:          Running
    Started:      Sun, 31 Jan 2021 18:22:26 +0000
  Last State:     Terminated
    Reason:       Completed
```

```
ETCDCTL_API=3 etcdctl --endpoints 10.2.0.9:2379 \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
snapshot save /data/newbackup.db
```



In the above photo, the backup up process is done. The output saved to /data/newbackup.db.

Now, we want to restore it etc database to the new cluster.

Before restoring the process, we have to check and could have a little configuration like path on etcd.yaml inside the etc directory.

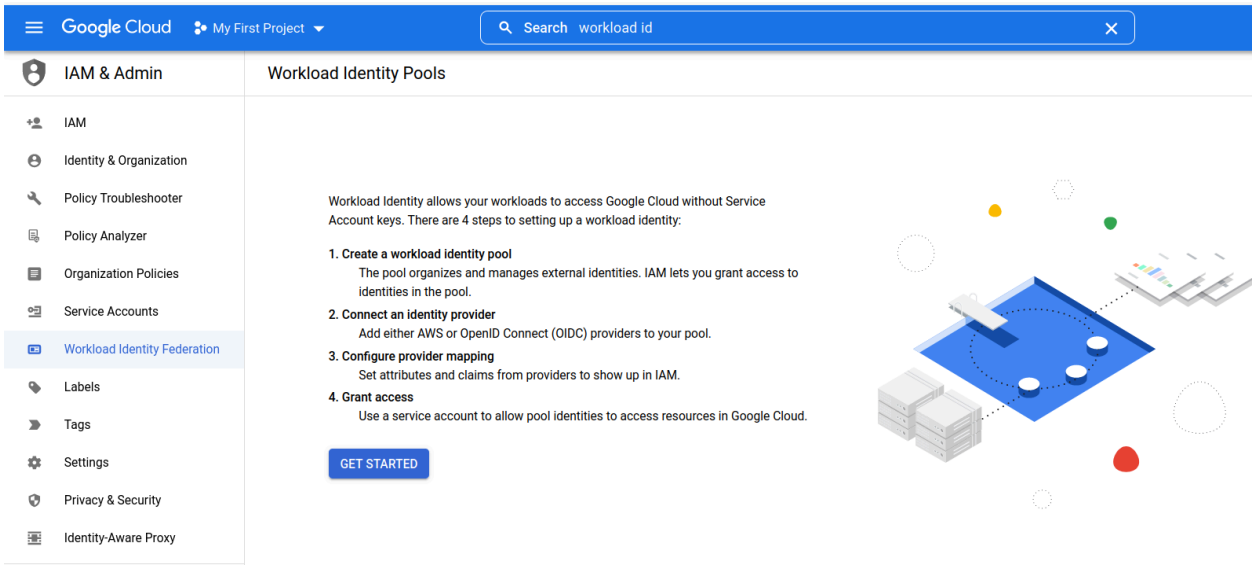
Restoring etcd database to new cluster

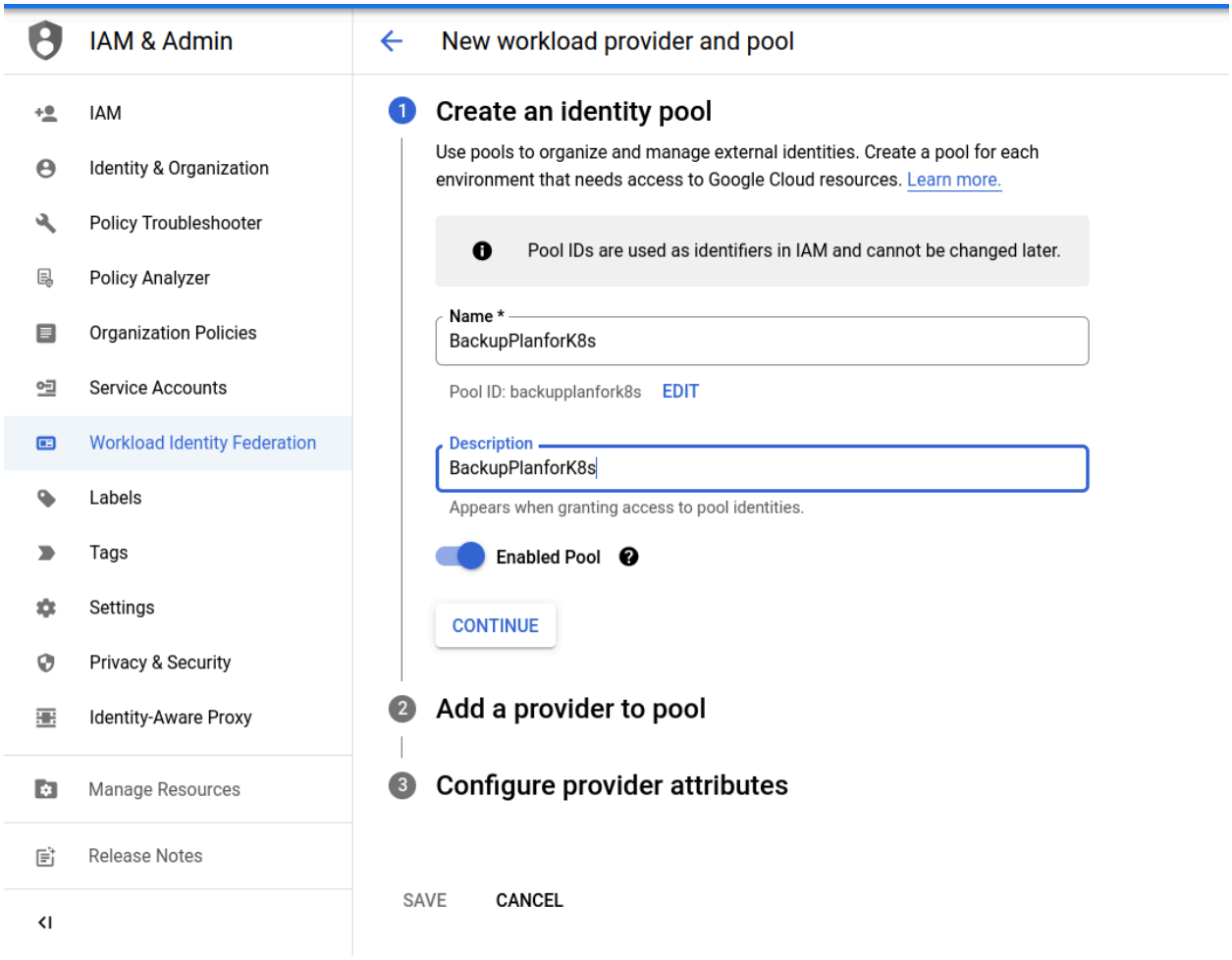
[Disaster recovery | etcd]

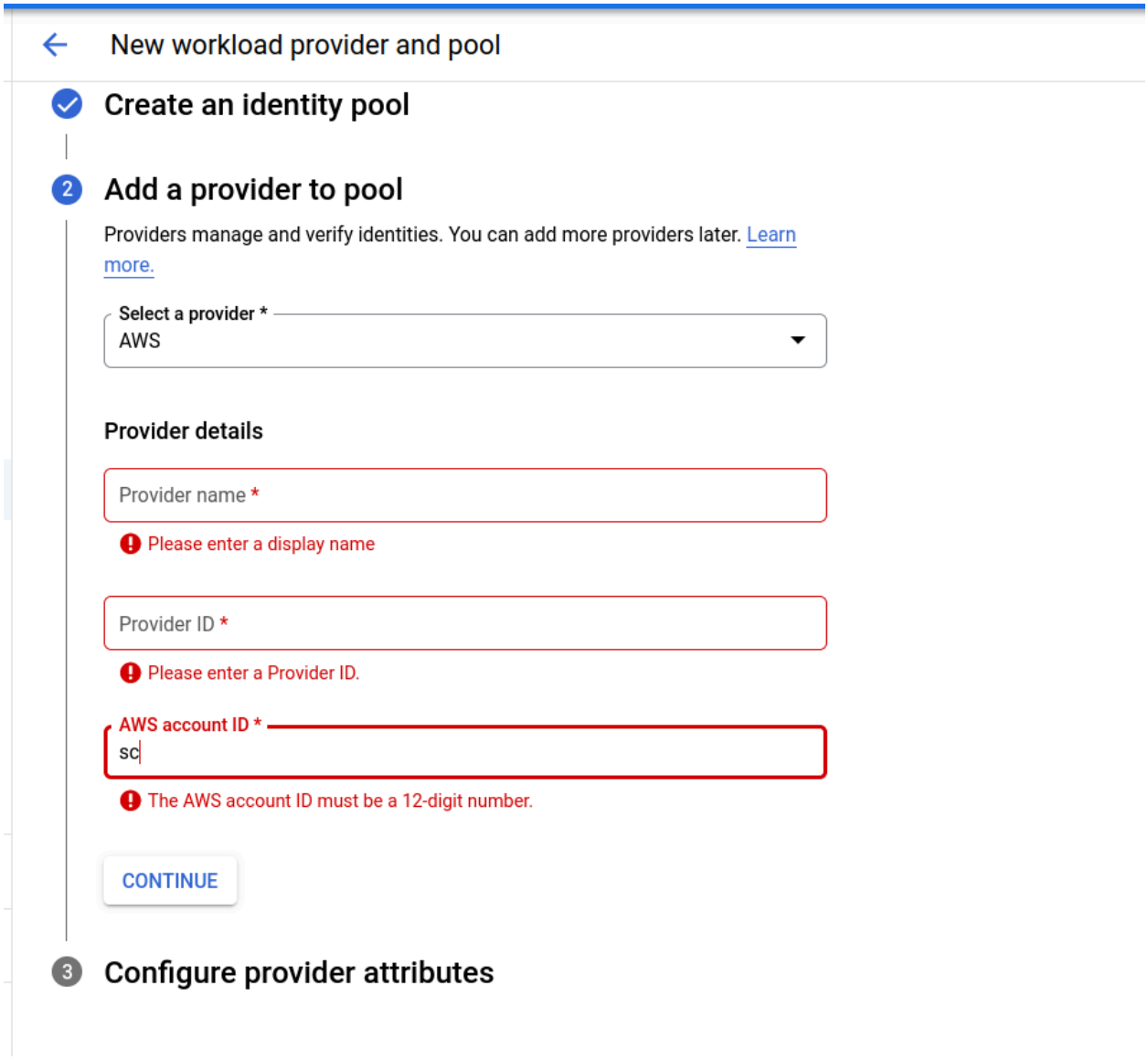
To be continue .....

Etcd Backup in GKE

Firstly set the workload identity




IAM & Admin
IAM
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Workload Identity Federation
Labels
Tags
Settings
Privacy & Security
Identity-Aware Proxy
Manage Resources
Release Notes
New workload provider and pool
1 Create an identity pool
Use pools to organize and manage external identities. Create a pool for each environment that needs access to Google Cloud resources. Learn more.
Pool IDs are used as identifiers in IAM and cannot be changed later.
Name *
BackupPlanforK8s
Pool ID: backupplanfork8s EDIT
Description
BackupPlanforK8s
Appears when granting access to pool identities.
Enabled Pool
CONTINUE
2 Add a provider to pool
3 Configure provider attributes
SAVE
CANCEL


New workload provider and pool
Create an identity pool
2
Add a provider to pool
Providers manage and verify identities. You can add more providers later. Learn more.
Select a provider *
AWS
Provider details
Provider name *
Please enter a display name
Provider ID *
Please enter a Provider ID.
AWS account ID *
sc
The AWS account ID must be a 12-digit number.
CONTINUE
3
Configure provider attributes

Useful Commands Cheat Sheets for kubectl

$ kubectl get pods --all-namespace

$ kubectl get pods --all-namespace -l app=<Label to select>

$ kubectl get service --all-namespace --sort-by=.spec.type

(.spec.type) meaning . is the current deployment configuration file.

$ kubectl cluster-info

```
rangaraokaranam$ kubectl cluster-info
Kubernetes master is running at https://35.226.9.8
GLBCDefaultBackend is running at https://35.226.9.8/api/v1/namespaces/kube-system/se
rvices/default-http-backend:http/proxy
Heapster is running at https://35.226.9.8/api/v1/namespaces/kube-system/services/hea
pster/proxy
KubeDNS is running at https://35.226.9.8/api/v1/namespaces/kube-system/services/kube
-dns:dns/proxy
Metrics-server is running at https://35.226.9.8/api/v1/namespaces/kube-system/servic
es/https:metrics-server:/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

To further debug and diagnose cluster problems, use this one.

$ kubectl cluster-info dump

$ kubectl top node

```
rangaraokaranam$ kubectl top node
NAME                                                CPU(cores)   CPU%   MEMORY(byte
s)   MEMORY%
gke-in28minutes-cluster-default-pool-c13cdcda-0p6j   50m          5%     808Mi
     30%
gke-in28minutes-cluster-default-pool-c13cdcda-s4xk   66m          7%     1082Mi
     41%
gke-in28minutes-cluster-default-pool-c13cdcda-thzt   95m          10%    781Mi
     29%
rangaraokaranam$
```

Shortcut for components (Describe with Demo)

- Replica Set = rs
- Service = svc
- Pods = po
- Namespace = ns
- Nodes = no

```
rangaraokaranam$ kubectl get no
NAME                                                  STATUS   ROLES    AGE    VERSIO
N
gke-in28minutes-cluster-default-pool-c13cdcda-0p6j    Ready    <none>   129m   v1.13.
11-gke.23
gke-in28minutes-cluster-default-pool-c13cdcda-s4xk    Ready    <none>   129m   v1.13.
11-gke.23
gke-in28minutes-cluster-default-pool-c13cdcda-thzt    Ready    <none>   129m   v1.13.
11-gke.23
```