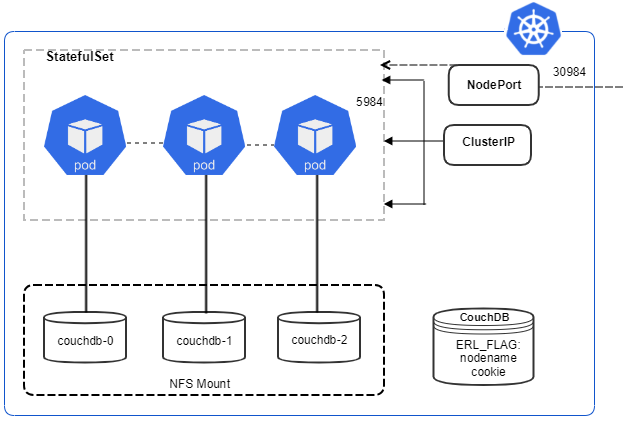
**Design Diagram**

Below diagram will give you a quick glimpse of the various components involved in the deployment. Rest of the article will provide step by step details on building each set of the component.

https://miro.medium.com/max/60/1*K3xwZLXnzNnADG6xAswBOA.png?q=20



**Storage Requirements**

A statefulset deployment needs a persistent volume for its storage requirements. This example is using a 10Gb Network File System (NFS) share to be used as Kubernetes persistent volume(PV) for storing database and related files.

First, mount the NFS share on a client system and create three directories as couchdb-0, couchdb-1 and couchdb-2. These directories will be used to attach Kubernetes persistent volume.

$ mkdir couchnfs  
$ mount nfs\_server\_ip:/shared\_folder/ couchnfs/  
$ cd couchnfs  
$ mkdir couchdb-0 couchdb-1 couchdb-2

After creating NFS shares, you will create a set of 3 Persistent Volumes named as couch-vol-0, couch-vol-1, couch-vol-2.  
Create a new file named pv.yaml and copy and save the content of the below manifest file to it. Replace the ‘server’ field with NFS server IP and ‘nfs\_share’ field in the ‘path’ field with full NFS path in this file. You can choose any name of your choice for these volumes.

**File:** pv.yaml

---  
apiVersion: v1  
kind: PersistentVolume  
metadata:  
 name: couch-vol-0  
 labels:  
 volume: couch-volume  
spec:  
 capacity:  
 storage: 10Gi  
 accessModes:  
 - ReadWriteOnce  
 nfs:  
 server: x.x.x.x  
 path: "/nfs\_share/couchdb-0"  
---  
apiVersion: v1  
kind: PersistentVolume  
metadata:  
 name: couch-vol-1  
 labels:  
 volume: couch-volume  
spec:  
 capacity:  
 storage: 10Gi  
 accessModes:  
 - ReadWriteOnce  
 nfs:  
 server: x.x.x.x  
 path: "/nfs\_share/couchdb-1"  
---  
apiVersion: v1  
kind: PersistentVolume  
metadata:  
 name: couch-vol-2  
 labels:  
 volume: couch-volume  
spec:  
 capacity:  
 storage: 10Gi  
 accessModes:  
 - ReadWriteOnce  
 nfs:  
 server: x.x.x.x  
 path: "/nfs\_share/couchdb-2"

After you have completed the above step, run below command to create persistent volumes.

$ kubectl create -f pv.yaml  
 persistentvolume/couch-vol-0 created  
 persistentvolume/couch-vol-1 created  
 persistentvolume/couch-vol-2 created

Verify the details of your newly created persistent volumes by running below command.

$ kubectl get pv  
 NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS   
 couch-vol-0 10Gi RWO Retain Available  
 couch-vol-1 10Gi RWO Retain Available  
 couch-vol-2 10Gi RWO Retain Available

**Deploy Statefulset**

Next, create three Kubernetes pods(each running a single container) using statefulset manifest file and connected via service ‘couch-service’. This manifest file has important details like ports, environment variables and ERL\_FLAGS variables which are required for setting up the cluster and joining nodes.

This file also has a volumeClaimTemplate to create persistent volume claim(PVC) to connect with PV’s which were created in the above step.

**File:** couchdb-statefulset.yaml

---  
apiVersion: apps/v1beta1  
kind: StatefulSet  
metadata:  
 name: couchdb  
spec:  
 serviceName: "couch-service"  
 replicas: 3  
 template:  
 metadata:  
 labels:  
 app: couch # *pod label*  
 spec:  
 containers:  
 - name: couchdb  
 image: couchdb:2.2.0  
 env:  
 - name: NODE\_NETBIOS\_NAME  
 valueFrom:  
 fieldRef:  
 fieldPath: metadata.name  
 - name: NODENAME  
 value: $(NODE\_NETBIOS\_NAME).couch-service # *FQDN in vm.args*  
 - name: COUCHDB\_USER  
 value: admin  
 - name: COUCHDB\_PASSWORD  
 value: password  
 - name: COUCHDB\_SECRET  
 value: monster  
 - name: ERL\_FLAGS  
 value: "-name couchdb@$(NODENAME)"  
 - name: ERL\_FLAGS  
 value: "-setcookie monster" # *the “password” used when nodes connect to each other.*  
 ports:  
 - name: couchdb  
 containerPort: 5984  
 - name: epmd  
 containerPort: 4369  
 - containerPort: 9100  
 volumeMounts:  
 - name: couch-pvc  
 mountPath: /opt/couchdb/data  
 volumeClaimTemplates:  
 - metadata:  
 name: couch-pvc  
 spec:  
 accessModes: ["ReadWriteOnce"]  
 resources:  
 requests:  
 storage: 10Gi  
 selector:  
 matchLabels:  
 volume: couch-volume

In the above yaml file, notice that the kind is ‘Statefulset’. A ‘Statefulset’ is ideal for deploying highly available database workloads. It enables an ordered graceful deployment, scaling and termination of pods. You can refer to Kubernetes documentation [here](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/) to know more about statefulsets.

This example will install CouchDB version 2.2.0 on these pods. You can select a supported version as per your requirement. To know more about supported docker versions of CouchDB, refer docker documentation [here](https://hub.docker.com/_/couchdb).

Create a new file named couchdb-statefulset.yaml and copy and save the content of above statefulset manifest file to it and run below command to create CouchDB statefulset. This will create three pods and PVC’s.

$ kubectl create -f couchdb-statefulset.yaml  
 statefulset.apps/couchdb created

**Environment Variables for Cluster Deployment**

Let’s look at some important environment variables as described in ‘env’ section in couchdb-statefulset.yaml file.  
metadata.name returns Pod’s name as value. This is equivalent to pod’s NetBIOS name.

As per the docker documentation, a CouchDB container needs NODENAME setting and erlang cookie for cluster setup. ERL\_FLAGS keys are used to set these environment variables. These settings are mentioned as -name couchdb@$(NODENAME) and -setcookie monster.

COUCHDB\_USER and COUCHDB\_PASSWORD are DB login credentials.  
VolumeMounts /opt/couchdb/data is the path where CouchDB data is stored.

Refer docker documentation [here](https://hub.docker.com/_/couchdb) to read more about these environment variables.

**Verifying Pods Status and DNS Names**

Check the running status of CouchDB pods by running below cmdlet. All the pods should have a ‘Running’ status to proceed with the next steps in this article.

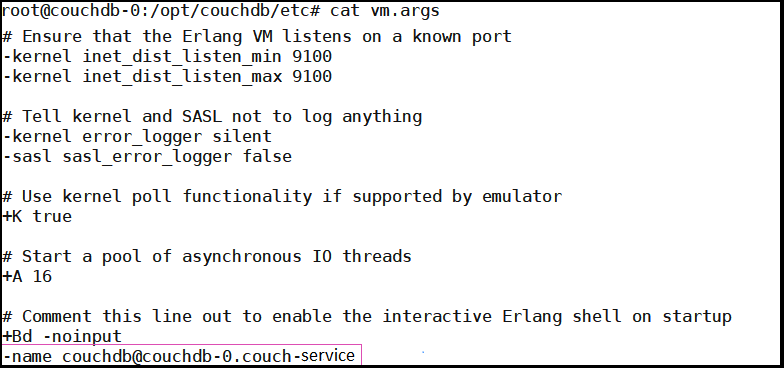
$ kubectl get pod  
 NAME READY STATUS RESTARTS AGE  
 couchdb-0 1/1 Running 0 1m  
 couchdb-1 1/1 Running 0 1m  
 couchdb-2 1/1 Running 0 1m

Next, ssh into pods using below command and verify that name field is using an FQDN.

$ kubectl exec -it couchdb-0 bash  
$ root@couchdb-0:/opt/couchdb# cd etc/  
$ root@couchdb-0:/opt/couchdb/etc# cat vm.args

Below output is from couchdb-0 . The -name field in etc/vm.args is using an FQDN ‘couchdb-0.couch-service’ which is a DNS name stored in kube-dns. This name should resolve to pod IP. All three pods should be able to resolve each other using their respective FQDN portion of the name i.e **couchdb-0.couch-service**.

https://miro.medium.com/max/60/1*zD8qyvpbl58vDcN-MmYSxQ.png?q=20



**Creating ClusterIP and NodePort Services**

A Kubernetes service describes a set of pods that perform the same task. The set of pods managed by service are determined by a label selector.

In the next step, you will deploy a headless service named ‘couch-service’ as type ClusterIP, reachable from within the cluster. You will also deploy a NodePort service named ‘couch-nodep-svc’ accessible from Kubernetes cluster’s worker Node’s IP on a static port 30984 to access CouchDB Fauxton web administration interface. It is not mandatory to create a NodePort service, but its useful for accessing our CouchDB web administration dashboard.

**File:**services.yaml

---  
apiVersion: v1  
kind: Service  
metadata:  
 name: couch-service  
 namespace: default  
 labels:  
 app: couch  
spec:  
 type: ClusterIP  
 clusterIP: None  
 ports:  
 - port: 5984  
 protocol: TCP  
 targetPort: 5984  
 selector:  
 app: couch  *# label selector*  
---  
kind: Service  
apiVersion: v1  
metadata:  
 name: couch-nodep-svc   
 labels:  
 app: couch  
spec:  
 type: NodePort *# NodePort service*  
 ports:  
 - port: 5984  
 nodePort: 30984 *# external port*  
 protocol: TCP  
 selector:  
 app: couch  *# label selector*

Save the above manifest file as services.yaml and run below command to create ClusterIP and NodePort services.

$ kubectl create -f services.yaml  
 service/couch-service created  
 service/couch-nodep-svc created

Run below command to verify the service details. You should notice two types of services, ClusterIP with IP address as None and port as 5984/TCP. NodePort service should have an IP address and port should be listed as 5984:30984/TCP.

$ kubectl get svc  
 NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
 couch-nodep-svc NodePort x.x.x.x <none> 5984:30984/TCP 8s  
 couch-service ClusterIP None <none> 5984/TCP 8s

**Cluster Configuration**

Now, you will follow CouchDB cluster setup API to create a cluster and join nodes to it. You can refer to CouchDB API documentation [here](https://docs.couchdb.org/en/2.2.0/cluster/setup.html#cluster-setup-wizard)

First, ssh into each pod and verify cluster setup by running below command on each pod locally. The output of this command should indicate that ‘cluster is already enabled’.

curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "enable\_cluster", "bind\_address":"0.0.0.0", "username": "admin", "password":"password", "node\_count":"3"}'

Next, choose any one pod as coordinator pod to run rest of the commands. For example, ssh into the pod couchdb-0 and run below commands to join other two nodes to the cluster. A successful output from these commands should be **{"ok":true}**

**cluster node 1:** ‘couchdb-1.couch-service’

curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "enable\_cluster", "bind\_address":"0.0.0.0", "username": "admin", "password":"password", "port": 5984, "node\_count": "3", "remote\_node": "couchdb-1.couch-service>", "remote\_current\_user": "admin", "remote\_current\_password": "password" }'curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "add\_node", "host":"couchdb-1.couch-service", "port": 5984, "username": "admin", "password":"password"}'

**cluster node 2:** ‘couchdb-2.couch-service’

curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "enable\_cluster", "bind\_address":"0.0.0.0", "username": "admin", "password":"password", "port": 5984, "node\_count": "3", "remote\_node": "couchdb-2.couch-service>", "remote\_current\_user": "admin", "remote\_current\_password": "password" }'curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "add\_node", "host":"couchdb-2.couch-service", "port": 5984, "username": "admin", "password":"password"}'

In the next step, run below command to finish the setup. This will finish the cluster setup and create the required databases.

curl -X POST -H "Content-Type: application/json" http://admin:password@127.0.0.1:5984/\_cluster\_setup -d '{"action": "finish\_cluster"}'

Finally, you can run below command to verify cluster membership.

curl http://admin:password@127.0.0.1:5984/\_membership

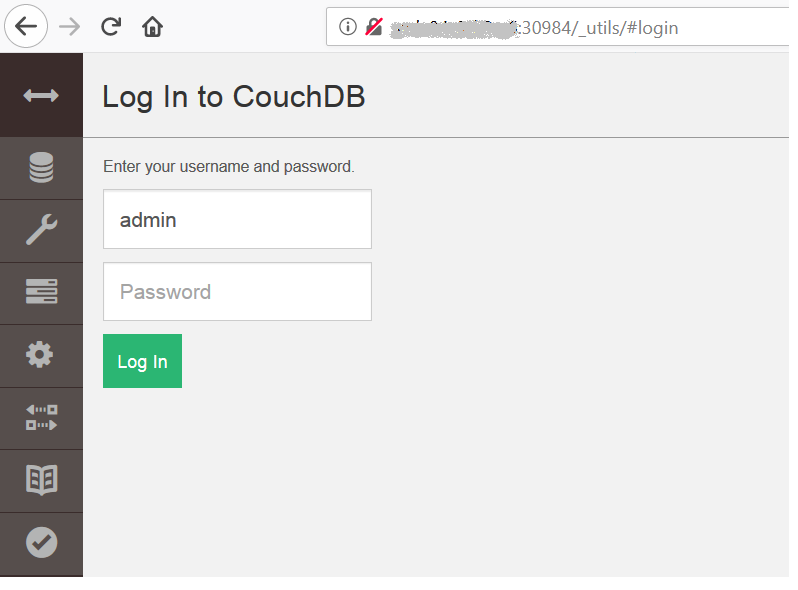
You should see all three nodes listed as cluster nodes.

{  
 "all\_nodes": [  
 "couchdb-1.couch-service",  
 "couchdb-2.couch-service",  
 "couchdb-0.couch-service"  
 ],  
 "cluster\_nodes": [  
 "couchdb-1.couch-service",  
 "couchdb-2.couch-service",  
 "couchdb-0.couch-service"  
 ]  
}

**Connecting to Cluster Web Administration Interface**

Now, your cluster is set up and you can log in to CouchDB Fauxton web administration interface by using NodePort service.  
Use the URL [http://Kubernetes-worker-node-ip:30984/\_utils](http://kubernetes-worker-node-ip:30984/_utils) and type ‘admin’ and ‘password’ to login to the web dashboard.

https://miro.medium.com/max/60/1*vTDz9DIsfN1cfX85d8CtVg.png?q=20



If you are able to view the web administration console, Congratulations!… you have successfully deployed a CouchDB cluster on your Kubernetes setup.