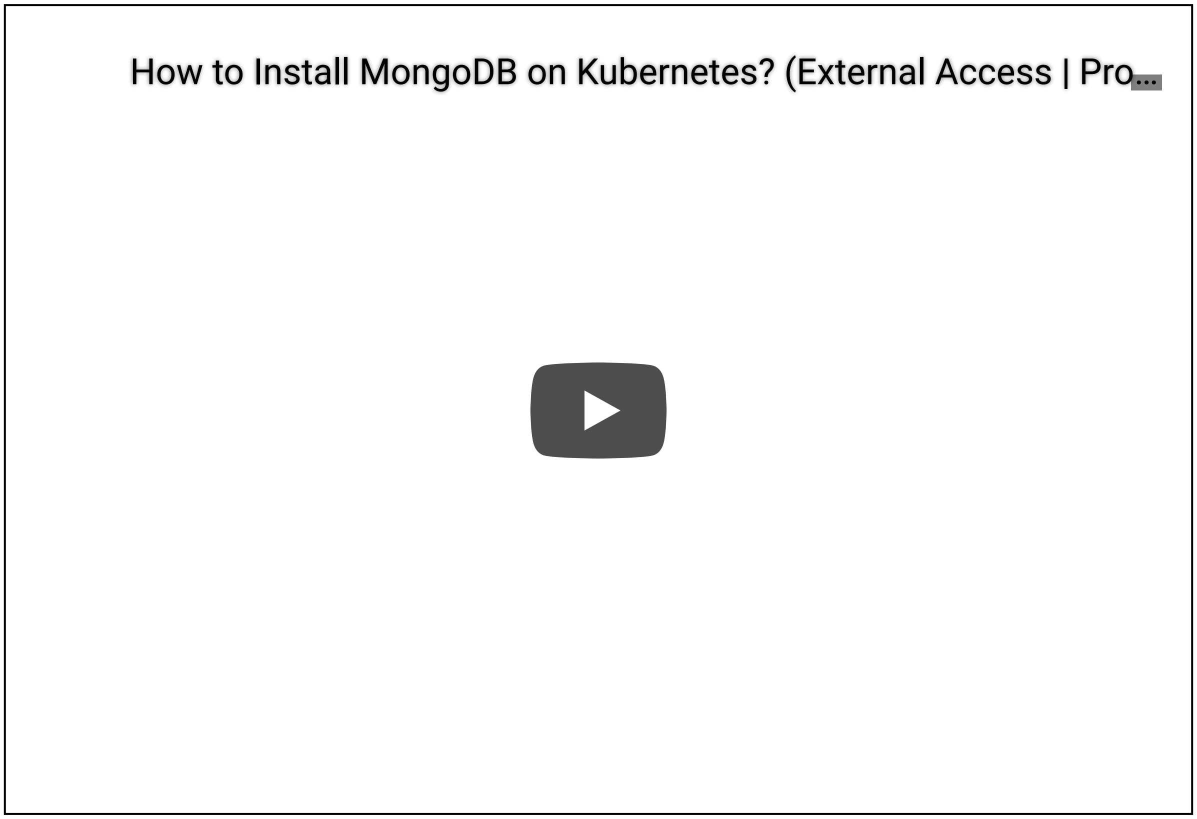
How to Install MongoDB on Kubernetes?


How to Install MongoDB on Kubernetes? (External Access | Pro...

- You can find the source code for this video in my GitHub Repo.
- 😆 - If you want to create EKS cluster using terraform, you can follow this tutorial.

Install MongoDB Kubernetes Operator

To install MongoDB, we're going to be using an open-source Kubernetes operator. Operator and MongoDB will be deployed in the same namespace. Let's start with it.

- Give it a name MongoDB and also important to add a label monitoring equal to Prometheus. Prometheus will only monitor namespaces that contain this label.

namespace.yaml

- Next, we need to create a custom resource definition for MongoDBCommunity. It extends Kubernetes and allows you to define a custom type that only be created by the corresponding operator. It will create a MongoDB cluster based on this definition and manage its lifecycle. Create crd.yaml file.
- Since it will require Kubernetes API server access, we need to create some RBAC policies. RBAC is a role-based access control system for Kubernetes. Create rbac folder and corresponding files.
- Then finally, the operator itself. It's going to be deployed as a simple deployment object. You can adjust a few parameters if you want, such as operator version, image repository, and others.

operator.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: mongodb
  name: mongodb-kubernetes-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: mongodb-kubernetes-operator
  strategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        name: mongodb-kubernetes-operator
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
```

```
          /usr/local/bin/entrypoint
        env:
        - name: WATCH_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: OPERATOR_NAME
          value: mongodb-kubernetes-operator
        - name: AGENT_IMAGE
          value: quay.io/mongodb/mongodb-agent:11.0.5.6963-1
        - name: VERSION_UPGRADE_HOOK_IMAGE
          value: quay.io/mongodb/mongodb-kubernetes-operator-version-
upgrade-post-start-hook:1.0.3
        - name: READINESS_PROBE_IMAGE
          value: quay.io/mongodb/mongodb-kubernetes-readinessprobe:1.0.6
        - name: MONGODB_IMAGE
          value: mongo
        - name: MONGODB_REPO_URL
          value: docker.io
        image: quay.io/mongodb/mongodb-kubernetes-operator:0.7.2
        imagePullPolicy: Always
        name: mongodb-kubernetes-operator
        resources:
          limits:
            cpu: 1100m
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 200Mi
        securityContext:
          readOnlyRootFilesystem: true
          runAsUser: 2000
      serviceAccountName: mongodb-kubernetes-operator
```

- Now let's move to the terminal and apply all of these files. I assume that you already have Kubernetes provisioned and kubectl configured to talk to the cluster.

```
kubectl apply -f k8s/mongodb/namespace.yaml
kubectl apply -f k8s/mongodb/crd.yaml
```

Install MongoDB on Kubernetes (Standalone/Single Replica)

There are a couple of ways to manage users in MongoDB. You can create users using the MongoDB operator or using the shell. I'll show you both ways along the way. Start with creating an admin user with the custom resource.

- First, we need to create a secret with a password. In this case, admin123.

secret.yaml

```yaml
---
apiVersion: v1
kind: Secret
metadata:
  name: admin-user-password
  namespace: mongodb
type: Opaque
stringData:
  password: admin123
```

- Now the main database configuration file. It's going to be a MongoDBComunity type. Then specify how many replicas do you want. If you use one member, the operator will create a standalone MongoDB instance. We will start with one and scale it up a little bit later. For now, only a single type is supported, which is ReplicaSet. Enterprise Operator supports various types, including sharded cluster. Community edition operator at this point only can deploy a cluster with multiple replicas only.

mongodb.yaml

```yaml
---
apiVersion: mongodbcommunity.mongodb.com/v1
kind: MongoDBCommunity
metadata:
  name: my-mongodb
  namespace: mongodb
spec:
  users:
  - name: admin-user
    db: admin
    passwordSecretRef:
      name: admin-user-password
    roles:
    - name: clusterAdmin
      db: admin
    - name: userAdminAnyDatabase
      db: admin
    scramCredentialsSecretName: my-scram
  additionalMongodConfig:
    storage.wiredTiger.engineConfig.journalCompressor: zlib
  statefulSet:
    spec:
      template:
        spec:
          containers:
          - name: mongod
            resources:
              limits:
                cpu: "1"
                memory: 2Gi
              requests:
                cpu: 500m
                memory: 1Gi
          affinity:
            podAntiAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                  - key: app
                    operator: In
                    values:
                    - my-mongodb
                topologyKey: "kubernetes.io/hostname"
      volumeClaimTemplates:
      - metadata:
          name: data-volume
        spec:
          accessModes:
          - ReadWriteOnce
          resources:
            requests:
              storage: 40G
```

is running and passing all the health checks.

```
kubectl get pods -n mongodb
```

- You can find persistent volume claims; we have one for 38 gigs and the second for two gigs used for logging by the operator.

```
kubectl get pvc -n mongodb
```

- Conveniently, the operator will generate a secret with the credentials and connection strings. You can get it from the secret. You have a standard connection string and a DNS Seed List Connection String.

```
kubectl get secret my-mongodb-admin-admin-user -o yaml -n mongodb
```

- You can grab the string and decode it using the base64 tool.

```
echo "HGC%#DG" | base64 -d
```

- Or, as a shortcut, you can use the jq command to parse the secret.

```
kubectl get secret my-mongodb-admin-admin-user -n mongodb -o json | jq -r
'.data | with_entries(.value |= @base64d)'
```

- Now let's connect to the database. We are going to be using a mongosh shell. You can use the port forward command to be able to access MongoDB locally.

```
kubectl port-forward my-mongodb-0 27017 -n mongodb
```

- To connect, provide the username and a password and use localhost with the default port number.

```
mongosh "mongodb://admin-user:admin123@127.0.0.1:27017/admin?
directConnection=true&serverSelectionTimeoutMS=2000"
```

- First, list all the available databases.

```
    user: 'aputra',
    pwd: 'devops123',
    roles: [ { role: 'readWrite', db: 'store' } ]
  }
);
```

- Then authenticate using its credentials.

```
db.auth('aputra', 'devops123')
```

- Create a new store database.

```
use store
```

- Try to insert a record using the insertOne command.

```
db.employees.insertOne({name: "Anton"})
```

- Then, retrieve all the records in the collection using the find function.

```
db.employees.find()
```

Install MongoDB on Kubernetes (Replica Set)

Let's move on to the next example. Let's scale up the MongoDB to include two replica instances.

- To scale up, simply increase the number of members from 1 to 3.

mongodb.yaml

```
---
apiVersion: mongodbcommunity.mongodb.com/v1
kind: MongoDBCommunity
metadata:
  name: my-mongodb
  namespace: mongodb
```

```
kubectl apply -f k8s/mongodb/database/mongodb.yaml
```

- Now we have one primary instance and two replicas.

```
kubectl get pods -n mongodb
```

- Now we have one primary instance and two replicas. Since we still have the previous session, let's verify that replicas are up to date. You need to switch to the admin database first.

```
use admin
```

- Then, authenticate with admin credentials.

```
db.auth('admin-user', 'admin123')
```

- If you run the status, you find all the members.

```
rs.status()
```

- You can also check the replication if replicas are able to keep up with the primary.

```
rs.printSecondaryReplicationInfo()
```

- At this point, we have the MongoDB cluster ready for use. For the following example, we will secure MongoDB with TLS, but first, we need to clean up and delete the current deployment and persistent volume claims with corresponding volumes.

```
kubectl delete -f k8s/mongodb/internal/mongodb.yaml
kubectl delete pvc -l app=my-mongodb-svc
```

Install Cert-Manager on Kubernetes

```
    https://charts.jetstack.io
```

- Update index.

```
helm repo update
```

- Before deploying the cert-manager, I want to create Prometheus custom resources since we will use Prometheus to monitor all our components, including the certificates. You may get an error if you try to use apply since those files a huge. If you get an error, just use create instead of apply. It has to do with a limitation on the size of annotation.

```
kubectl create -f k8s/prometheus-operator/crds
```

- Next, create a namespace, and don't forget to include label monitoring equal to prometheus. Otherwise, the cert-manager will be ignored by Prometheus.

namespace.yaml

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: cert-manager
  labels:
    monitoring: prometheus
```

- To customize helm deployment, you can create a values file and override default variables. I want to include CRDs deployment as part of helm deployment. Then enable Prometheus monitoring. And define the service monitor object; this is a reason why we need to create Prometheus Operator CRDs first. Prometheus instance default must match Prometheus label as well.

helm-values.yaml

```yaml
---
installCRDs: true
```

Create a namespace first.

```
kubectl apply -f k8s/cert-manager/namespace.yaml
```

- Then deploy cert-manager and provide values file and specify the version of the helm chart.

```
helm install cert-105 jetstack/cert-manager \
  --namespace cert-manager \
  --version v1.6.1 \
  --values k8s/cert-manager/helm-values.yaml
```

- You will get three pods in the cert-manager namespace. Make sure that they are all up and running.

```
kubectl get pods -n cert-manager
```

Secure MongoDB with TLS/SSL

Next, we need to bootstap PKI. First of all, we need to create a self-sign Cluster Issuer to generate Certificate Authority.

self-signed-Issuer.yaml

```yaml
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: selfsigned
spec:
  selfSigned: {}
```

- Let's apply it in the terminal.

```
kubectl apply -f k8s/mongodb/certificates/self-signed-issuer.yaml
```

```yaml
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: devopsbyexample-io-ca
  namespace: cert-manager
spec:
  isCA: true
  duration: 43800h # 5 years
  commonName: devopsbyexample.io
  secretName: devopsbyexample-io-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned
    kind: ClusterIssuer
    group: cert-manager.io
```

- Let's apply it now.

```
kubectl apply -f k8s/mongodb/certificates/ca.yaml
```

- Make sure that the CA certificate is ready, and by default, it will be located in the cert-manager namespace.

```
kubectl get certificate -n cert-manager
```

- Next, we need to create a new Cluster Issuer based on the CA that we just generated. You don't have to create CA using the cert-manager. If your organization already has a certificate authority, you can simply import it as a secret and create this Cluster Issuer to sign new certificates using your existing CA.

ca-issuer.yaml

```yaml
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
```

- Now, we are ready to issue a certificate for the MongoDB cluster. First, it will only be accessible within the Kubernetes cluster. CA is false here. Those certificates are automatically renewed by the cert-manager that allows us to use a shorter duration, such as 90 days. The important part here, you can either use a common name with a wildcard which I don't recommend, or define alternative names using the dnsNames section. They must match internal MongoDB DNS names.

certificate.yaml

```yaml
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: mongodb
  namespace: mongodb
spec:
  isCA: false
  duration: 2160h # 90d
  renewBefore: 360h # 15d
  dnsNames:
  - my-mongodb-0.my-mongodb-svc.mongodb.svc.cluster.local
  - my-mongodb-1.my-mongodb-svc.mongodb.svc.cluster.local
  - my-mongodb-2.my-mongodb-svc.mongodb.svc.cluster.local
  secretName: mongodb-key-pair
  privateKey:
    algorithm: RSA
    encoding: PKCS1
    size: 4096
  issuerRef:
    name: devopsbyexample-io-ca
    kind: ClusterIssuer
    group: cert-manager.io
```

- You can go to the terminal and create that certificate.

```
kubectl apply -f k8s/mongodb/internal/certificate.yaml
```

the secret with certificates. Optionally, if you want to secure your existing database with TLS, you need to set optional to true. Since it's a new deployment, we don't need that.

mongodb.yaml

```yaml
---
apiVersion: mongodbcommunity.mongodb.com/v1
kind: MongoDBCommunity
metadata:
  name: my-mongodb
  namespace: mongodb
spec:
  members: 3
  type: ReplicaSet
  version: "5.0.5"
  security:
    authentication:
      modes:
      - SCRAM
  users:
  - name: admin-user
    db: admin
    passwordSecretRef:
      name: admin-user-password
    roles:
    - name: clusterAdmin
      db: admin
    - name: userAdminAnyDatabase
      db: admin
    scramCredentialsSecretName: my-scram
  security:
    tls:
      enabled: true
      certificateKeySecretRef:
        name: mongodb-key-pair
      caCertificateSecretRef:
        name: mongodb-key-pair
      # optional: true
    authentication:
      modes:
      - SCRAM
  additionalMongodConfig:
    storage.wiredTiger.engineConfig.journalCompressor: zlib
```

```
      cpu: "1"
      memory: 2Gi
    requests:
      cpu: 500m
      memory: 1Gi
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - my-mongodb
        topologyKey: "kubernetes.io/hostname"
volumeClaimTemplates:
- metadata:
    name: data-volume
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 40G
```

- Now we can deploy the MongoDB cluster.

```
kubectl apply -f k8s/mongodb/internal/mongodb.yaml
```

- We can try to connect to the database using the TLS. Since it does not have external access yet, we can only connect to MongoDB inside the Kubernetes cluster. SSH to the pod with mongodb shell. It can be an existing MongoDB instance.

```
kubectl exec -it my-mongodb-0 -c mongod -- bash
```

- Now use mongosh with TLS. Provide a CA file and a certificate. When you create a headless service in Kubernetes, it will automatically create SRV DNS record. That's why you can use DNS Seed List Connection Format.

- Don't forget to delete the database and persistent volumes.

```
kubectl delete -f k8s/mongodb/internal/mongodb.yaml
kubectl delete pvc -l app=my-mongodb-svc
```

Configure External Access on AWS

For the last example, we will add external access and secure it with tls as well.

- Create a similar secret that will contain an admin password.

secret.yaml

```
---
apiVersion: v1
kind: Secret
metadata:
  name: external-admin-user-password
  namespace: mongodb
type: Opaque
stringData:
  password: admin123
```

- Then the certificate. The only difference here is that this certificate needs to be valid for internal access as well as external. It should have two sets of DNS names. devopsbyexample.io is my public DNS domain that I will use to create external MongoDB access. We will use the same CA issuer.

certificate.yaml

```
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: mongodb-external
  namespace: mongodb
```

```yaml
  - my-mongodb-0.devopsbyexample.io
  - my-mongodb-1.devopsbyexample.io
  - my-mongodb-2.devopsbyexample.io
  secretName: mongodb-external-key-pair
  privateKey:
    algorithm: RSA
    encoding: PKCS1
    size: 4096
  issuerRef:
    name: devopsbyexample-io-ca
    kind: ClusterIssuer
    group: cert-manager.io
```

- Also, for MongoDB, we need to add one more section - replicaSetHorizon. It will allow access to the database from the Kubernetes cluster as well as outside of it.

mongodb.yaml

```yaml
---
apiVersion: mongodbcommunity.mongodb.com/v1
kind: MongoDBCommunity
metadata:
  name: my-mongodb
  namespace: mongodb
spec:
  members: 3
  type: ReplicaSet
  version: "5.0.5"
  security:
    authentication:
      modes:
      - SCRAM
  users:
  - name: admin-user
    db: admin
    passwordSecretRef:
      name: external-admin-user-password
    roles:
    - name: clusterAdmin
      db: admin
    - name: userAdminAnyDatabase
      db: admin
    scramCredentialsSecretName: my-scram
```

```
      certificateKeySecretRef:
        name: mongodb-external-key-pair
      caCertificateSecretRef:
        name: mongodb-external-key-pair
    authentication:
      modes:
      - SCRAM
  additionalMongodConfig:
    storage.wiredTiger.engineConfig.journalCompressor: zlib
  statefulSet:
    spec:
      template:
        spec:
          containers:
          - name: mongod
            resources:
              limits:
                cpu: "1"
                memory: 2Gi
              requests:
                cpu: 500m
                memory: 1Gi
          affinity:
            podAntiAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                  - key: app
                    operator: In
                    values:
                    - my-mongodb
                topologyKey: "kubernetes.io/hostname"
      volumeClaimTemplates:
      - metadata:
          name: data-volume
        spec:
          accessModes:
          - ReadWriteOnce
          resources:
            requests:
              storage: 40G
```

- To create external access, you can use NodePort, but the better approach would be to create a load balancer for each pod. Usually, cloud load balancers will charge you only based on the number of connections. This approach will work in most of the clouds. This example will be

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: my-mongodb-0
  namespace: mongodb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
spec:
  type: LoadBalancer
  ports:
  - name: mongodb
    port: 27017
    protocol: TCP
  selector:
    app: my-mongodb-svc
    statefulset.kubernetes.io/pod-name: my-mongodb-0
---
apiVersion: v1
kind: Service
metadata:
  name: my-mongodb-1
  namespace: mongodb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
spec:
  type: LoadBalancer
  ports:
  - name: mongodb
    port: 27017
    protocol: TCP
  selector:
    app: my-mongodb-svc
    statefulset.kubernetes.io/pod-name: my-mongodb-1
---
apiVersion: v1
kind: Service
metadata:
  name: my-mongodb-2
  namespace: mongodb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
spec:
  type: LoadBalancer
  ports:
```

- Alright, let's deploy it now.

```
kubectl apply -f k8s/mongodb/external/secret.yaml
kubectl apply -f k8s/mongodb/external/certificate.yaml
kubectl apply -f k8s/mongodb/external/mongodb.yaml
kubectl apply -f k8s/mongodb/external/services.yaml
```

- We need to create DNS records using those load balancers. We have 3 LBs. Go to your DNS hosting and create DNS records. My domain is hosted with google domains. We need to create 3 CNAME records for each load balancer.

```
my-mongodb-0 CNAME 300 <lb-0>
my-mongodb-1 CNAME 300 <lb-1>
my-mongodb-2 CNAME 300 <lb-2>
```

- If you want to use a new connection string type, you can create SRV record.

```
_mongodb._tcp.my-mongodb SRV 0 50 27017 my-mongodb-0.devopsbyexample.io.
                             0 50 27017 my-mongodb-1.devopsbyexample.io.
                             0 50 27017 my-mongodb-2.devopsbyexample.io.
```

- Next, we need to retrieve the CA certificate and a certificate key file. You can get it from secrets or ssh to one of the pods and grab it from there.

```
kubectl exec -it my-mongodb-0 -c mongod -- bash
```

- First, let me cat the CA certificate.

```
cat /var/lib/tls/ca/ca.crt
vim ca.crt # paste content from previous command
```

- Next is the certificate key file. It will contain cert and a private key. Same thing here.

```
cat /var/lib/tls/server/*.pem
vim code certificateKey.pem # paste content from previous command
```

```
  "mongodb+srv://admin-user:admin123@my-mongodb.devopsbyexample.io/admin?
ssl=true&serverSelectionTimeoutMS=2000"
```

Install Prometheus and Grafana on Kubernetes

Finally, I'll show you how to monitor Mongodb with Prometheus inside the Kubernetes cluster.

- Let's quickly deploy Prometheu. You can find the code here.

```
kubectl apply -f k8s/prometheus-operator/rbac
kubectl apply -f k8s/prometheus-operator/deployment
kubectl apply -f k8s/prometheus
kubectl apply -f k8s/mongodb/exporter
kubectl apply -f k8s/cadvisor
kubectl apply -R -f k8s/grafana
```

Monitor MongoDB with Prometheus

- Use port forward to access Grafana locally. Go to localhost 3000 and use admin as a user and password devops123.

```
kubectl port-forward svc/grafana 3000 -n monitoring
```

Clean

- `kubectl delete -R -f k8s`