

kubernetes and docker advance

Assignment documentation



August 4, 2025

Kubernetes assignment

# Requirement Understanding

**1.1 Functional / Non-functional Requirements**

* The database connection password should not be clearly visible in any Kubernetes YAML files (used Kubernetes Secrets).
* **REST API backend** built with Node.js, exposing multiple endpoints (Postman collection provided).
* **Persistent data store** with MongoDB.
* **Environment-agnostic deployment** leveraging container images, Kubernetes manifests, and version-controlled configuration.
* **Persistent storage** across pod restarts (Database).
* **Secure handling** of credentials (*no secrets in plaintext*).
* **Idempotent data seeding** when deploying to a fresh cluster.
* The data seeding configurable from outside the pod definition file and application code (used Kubernetes Config Map and Job).
* **Accessible externally** (able to call APIs from outside cluster, using Ingress). Whereas keep Database accessible internally (using headless).
* Easy deployment workflow via deploy.bat.
* **Multiple pod deployment** (API deployed on 4 pods whereas DB deployed on 1 pod)
* **Create and Read operations** for 1 domain entity (e.g. users).

# Assumptions

Following assumptions were made to guide the architecture:

1. Needs deployments for only single environment and/or client. **Namespace Isolation is not needed**: Deploy into a separate Kubernetes namespace (e.g. dev) for multi‑tenant or organized cluster structure.
2. Assumption only 1 service is exposed to external users. So used Single service Ingress. If needed to support multiple services, need to change the ingress type.
3. Seed data volume is small and consumed in one-time Job.
4. Currently only 1 Pod is supported for database. If need to support more Pods, corresponding connection string needs to modify in the secrets.
5. Pre-requisites are already installed on the system, in order to deploy to the Azure. Ex- Azure CLI, Kubernetes CLI, kubectl.
6. A screenshot of a computer

   AI-generated content may be incorrect.In order to deploy the services correctly, user has Permissions on the various services used in the assignment.

# Solution Overview

Solution implements a **2-tier architecture** using Kubernetes, consisting of:

1. **Service API Tier**: A Node.js backend service (backend\_node\_app)
2. **Database Tier**: A MongoDB database (database\_mongodb)

**Tech Stack**

* **Node.js** for the service tier (API)
* **MongoDB** as the database tier
* **Docker** for containerization
* **Kubernetes** for deployment
* **Postman** for API testing
* **Azure VM** (used optionally for hosting or pushing to Docker Hub)

## Service Tier Highlights (backend\_node\_app)

* Main Code File: backend\_node\_app/app.js
* Reads data from MongoDB using connection string from Kubernetes Secrets (02\_secret\_db-credential.yaml and 05\_secret\_connection-string-expose.yaml)
* Expose endpoints using Ingress
* Configured for 4 replicas (via Deployment YAML)

### Explanation:

#### Kubernetes Folder:

* It contains all the yaml files (deployment, Nodeport service and Ingress).
* A screenshot of a computer program

  AI-generated content may be incorrect.Deploy.bat file contains the command line scripts that will apply the above yaml files, to make a deployment bit easier (reduce typo). Also, it take arguments based on that it deploy the services. Example -
  + “deploy.bat i” - it will deploy Deployment.yaml, nodeport.yaml and Ingress controller and ingress.
  + “deploy.bat i” - it will not deploy Ingress. But use “kubectl port-forward” to enable us to access the pod (using nodeport) locally (tunnelling)

#### Models:

* It contains User models, it is mapped to users collection in the mongo db.

#### Postman:

* To avoid API testing effort, postman.json is provided. We can directly import the file and execute the API endpoints.

#### App.js

* It contains node js code that does the following
  + Make connection to the mongo db by using the values provided in the environment variables. (it uses secrets defined in database\_mongodb )
  + Expose multiple endpoints to get, save user details.

#### Dockerfile

* It contains commands to build the image.

#### Dummy.js

* It exposes same endpoints but return dummy text. It is kept for testing purpose only.

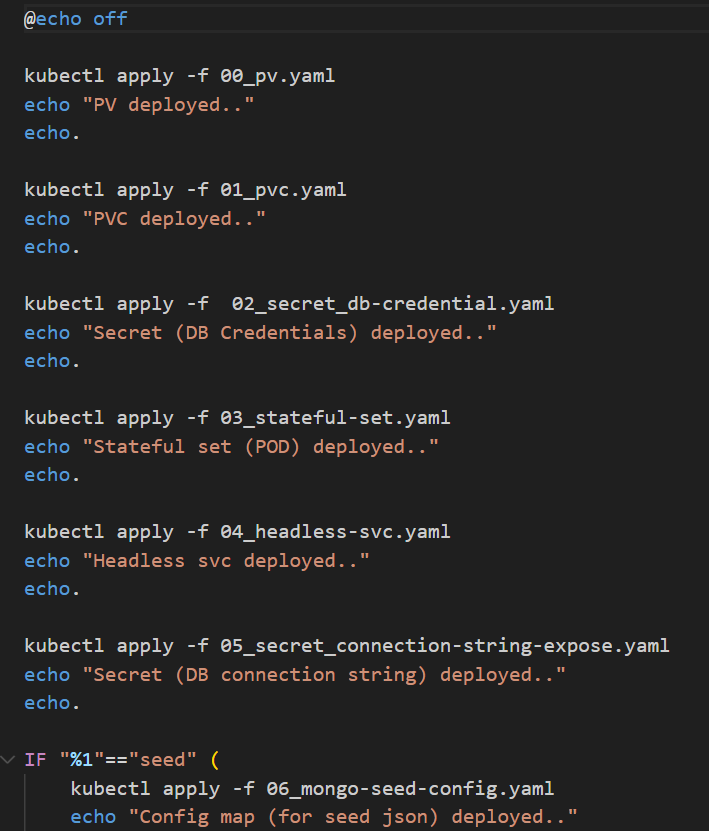
## Database Tier Highlights

* Deployed using a StatefulSet
* Uses Persistent Volume (PV) and Persistent Volume Claim (PVC) to ensure data persistence
* Includes mongo-seed job for inserting seed records
* Database credentials managed securely via Secrets
* Database not exposed outside the cluster

### Explanation:

#### Kubernetes Folder:

* It contains various yamls for creating PV, PVC, Secrets, configmap, and so on.
* Used 2 secrets “mongo-creds” and “db-connection” to store credentials and dbname/connection string separately.
* Used ConfigMap, for the seed data in json format (seed.json).
* Used Job, that will use config map to get seed.json and secrets to get db connection string, then execute command to insert seed data to db.
* It also contains deploy.bat that will deploy all the services in 1 go, based on parameter. If “seed” parameter is passed then seeding will run otherwise empty db is deployed.



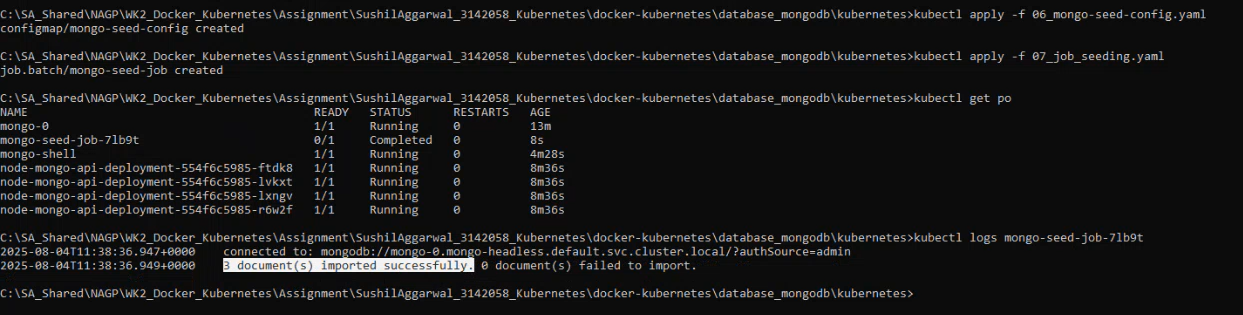
### Kubernetes YAMLs

Located in:

* backend\_node\_app/kubernetes – for API microservice
* database\_mongodb/kubernetes – for MongoDB
  + Includes: pv.yaml, pvc.yaml, stateful-set.yaml, secret\_db-credential.yaml, mongo-seed-config.yaml, and service definitions

# Screenshots:

1. Below screenshot showing the PODs created, after executing all Yaml. Also, it shows the result of seed job.
   1. Reference this time in the Video:- 35:34

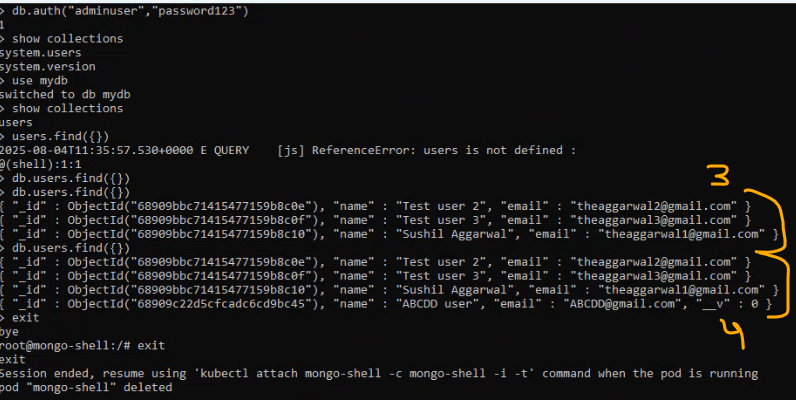


1. Below screenshot shows when new new user is entered from API, then user is added to the collection

Reference this time in video: 36:16

A screenshot of a computer

AI-generated content may be incorrect.



1. In below screenshot, even after deleting the pod, the data persists. Endpoint returns correct user details.

Refer this time in video: 40:46

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Screenshot showing deployment.yaml is using the latest image (deployed live in the video)

Refer this time in the video: 24:23

A screen shot of a computer

AI-generated content may be incorrect.

Screenshot showing connection string and other confidential details are referenced from the secrets.

Refer this time in the video:- 33:40

A screenshot of a computer program

AI-generated content may be incorrect.

# Justification for the Resources Utilized

**4.1 Deployment vs. StatefulSet**

* **Node.js API** is stateless (each instance is interchangeable); thus, a Deployment is ideal.
* **MongoDB** stores state and requires stable network identity and persistent storage; a **StatefulSet** ensures that:
  + Database pod has its own PVC and keeps its data if restarted or rescheduled.
  + Pods are assigned predictable DNS names
  + Pods are created and terminated in a defined order.

**4.2 Headless Service + PV/PVC**

* Headless service provides **direct DNS resolution** to individual MongoDB pods (e.g. mongo-0.mongo-headless.default.svc.cluster.local), which is needed to connect APIs to the database.
* MongoDB instance use its own **persistent volume claim**, bound to persistent volume. This prevents data loss and avoids contention issues

**4.3 Secret and ConfigMap Usage**

* **Secret** is used to store sensitive credentials (e.g. MongoDB root user password, URI), avoiding plaintext in YAML or environment configuration.
* **ConfigMap** is ideal for the seed file (seed.json), which is non-sensitive and may vary between environments (e.g. development vs testing).
* This separation ensures **security best practices** and **environmental portability**.

**4.4 Job for Seeding**

* Creating a Kubernetes **Job** allows for running the seeding process independently, enabling idempotent initialization of the database.
* This avoids embedding the seed logic in the application startup and allows for clear declarative state of seed execution.

**4.5 NodePort + Ingress**

* **NodePort** exposes the API service on each node’s fixed port range and enables inward traffic from the cluster node’s IP or firewall.
* **Ingress** provides routing based on domain name and URL path, TLS termination capabilities, and consolidates access into one point, avoiding multiple LoadBalancer resources in cost-constrained or local environments [Medium](https://nidhiashtikar.medium.com/kubernetes-differences-between-ingress-and-nodeport-1e014e587fc2?utm_source=chatgpt.com).

**4.6 Docker Hub + deploy.bat**

* **Docker Hub** is used as a central registry to store container image builds, making deployment repeatable across clusters and environments.
* deploy.bat simplifies developer workflow with a single command to apply all resources in the correct order, promoting consistency and reducing errors during redeployment or refresh.

**How It Works at Runtime**

1. **Deploy MongoDB** first — the StatefulSet ensures each replica comes up in order, each with dedicated storage.
2. **Apply Job** to seed data—reads seed.json from ConfigMap.
3. **Deploy Node.js API**, with environment variables pointing at the seeded MongoDB connection string.
4. **Expose Node.js via Service**, then route through **Ingress** (user can call https://<external-ip>/api/).