----------------------------------------**EKS pod backup** ----------------------------------------------------------------

**Step1 :** got to pod login indise **kubectl exec -it mongodb-pod bash** (or)  
 **kubectl exec -it mongodb-pod -- bash** (or) **kubectl exec -it mongodb-pod -- /bin/bash** and login to shell mongodb **mongosh**

**Step2:** create user 🡪 **use devenv** now you can switch in to devenv

**Step3:** create the collection  **db.createCollection("azure")**

**Step4:** Insert Multiple Documents and collection

db.mycollection.insertMany([

{ name: "Bob", age: 30, city: "Chicago" },

{ name: "Charlie", age: 28, city: "San Francisco" }

])

db. **sri**.insertMany([

{ name: "Bob", age: 30, city: "Chicago" },

{ name: "pod", age: 29, city: "San Francisco" }

])

**Step5**: use this command single DB

kubectl exec -it mongodb-pod -- mongodump --db **mgdata** --out /home/ubuntu/**devopd**/

kubectl cp mongodb-pod:/home/ubuntu**/ags/** /home/ubuntu**/ags/**

ls -la /home/ubuntu**/uat/**

kubectl exec -it **mongodb-pod** -- mongodump --out /home/ubuntu/**allbackup**/

kubectl cp mongodb-pod:/home/ubuntu/**allbackup**/ /home/ubuntu/**allbackup**/

ls -la /home/ubuntu**/allbackup/**

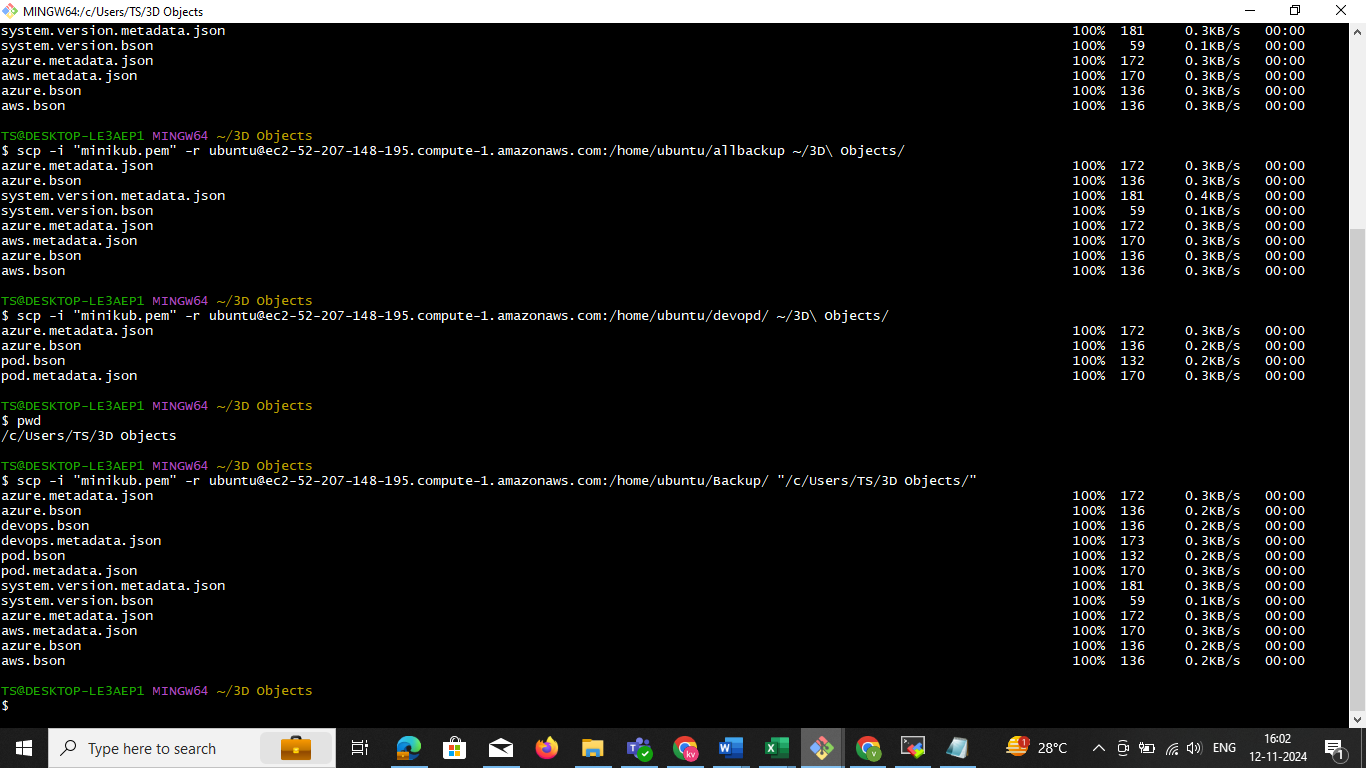
TS@DESKTOP-LE3AEP1 MINGW64 ~/3D Objects

$ pwd

/c/Users/TS/3D Objects

TS@DESKTOP-LE3AEP1 MINGW64 ~/3D Objects

scp -i "minikub.pem" -r ubuntu@ec2-52-207-148-195.compute-1.amazonaws.com:**/home/ubuntu/ser/ "/c/Users/TS/3D Objects/"**



**Automate Backups (Optional)**

You can automate the backup process by creating a **Kubernetes CronJob** that will run this backup command at regular intervals.

Here’s an example of how you could define a **CronJob** that runs the backup daily

apiVersion: batch/v1

kind: CronJob

metadata:

name: mongodb-backup

spec:

schedule: "1 \* \* \* \*" # Run daily based on REQ

jobTemplate:

spec:

template:

spec:

containers:

- name: mongodb-backup

image: mongo:latest

command:

- /bin/sh

- -c

- |

mongodump --archive=/data/db/mongodb-backup.archive --gzip &&

aws s3 cp /data/db/mongodb-backup.archive s3://your-bucket-name/mongodb-backup.archive

restartPolicy: OnFailure

In this example:

* The cron job runs daily at 2:00 AM (0 2 \* \* \*).
* The job will dump MongoDB and upload the backup to S3.

1. **MongoDB Backup**: Use mongodump to create a backup of the MongoDB database inside the pod.
2. **Upload to S3**: Use the AWS CLI (aws s3 cp) to upload the backup to an S3 bucket.
3. **IAM Authentication**: Make sure the pod has access to AWS (via AWS credentials or IAM roles for service accounts).
4. **Automation**: Use a Kubernetes CronJob to automate the backup process.

Kubernetes, the **restartPolicy** field does indeed accept only three specific values:

1. **Always**
2. **OnFailure**
3. **Never**

**Explanation of Each Value**

* **Always:** Restarts the container whenever it stops, regardless of whether it exited successfully or with an error. This is the default restartPolicy for a Deployment, ReplicaSet, and DaemonSet, where the goal is to ensure continuous running of the application.
* **OnFailure**: Restarts the container only if it exits with a non-zero (failure) status. This is commonly used in Job and CronJob configurations where you want the job to retry on failure but not to restart on successful completion.
* **Never**: The container will never be restarted, regardless of its exit status. This is often used for Job resources where the task is only meant to run once and complete without retrying.

1. Image Pull Policy Types

a. Always

Kubernetes always pulls the image from the container registry, even if it is already present on the node.

Best for environments where images are frequently updated with the same tag (e.g., latest).

Default for images tagged latest.

b. IfNotPresent

Kubernetes pulls the image only if it is not already present on the node.

Reduces network usage and speeds up pod creation if the image is cached.

Default for images with a specific tag (e.g., v1.0.0) or no tag (defaults to latest).

c. Never

Kubernetes never pulls the image from the registry.

The image must already exist on the node; otherwise, the pod will fail to start.

Useful for testing locally built images that are manually loaded onto nodes.

2. Specifying Image Pull Policy

You can define the image pull policy in your pod or deployment YAML under the containers section:

Example with Always:

apiVersion: v1

kind: Pod

metadata:

name: pull-always-example

spec:

containers:

- name: my-container

image: my-image:latest

imagePullPolicy: Always

Example with IfNotPresent:

apiVersion: v1

kind: Pod

metadata:

name: pull-ifnotpresent-example

spec:

containers:

- name: my-container

image: my-image:v1.0.0

imagePullPolicy: IfNotPresent

apiVersion: v1

kind: Pod

metadata:

name: pull-never-example

spec:

containers:

- name: my-container

image: my-image:local

imagePullPolicy: Never

3. Default Image Pull Policy Behavior

If imagePullPolicy is not specified:

Images tagged latest: Defaults to Always.

Images with a specific tag: Defaults to IfNotPresent.

Images with no tag: Defaults to IfNotPresent (interpreted as latest).

4. Controlling Image Pull Behavior Cluster-Wide

Limit registry access using imagePullSecrets: For private registries, define secrets to authenticate image pulls.

apiVersion: v1

kind: Pod

metadata:

name: private-registry-example

spec:

imagePullSecrets:

- name: my-registry-secret

containers:

- name: my-container

image: my-private-image:v1.0.0

Node-level image preloading:

Preload images on nodes to reduce registry dependency and set imagePullPolicy: Never.

Admission Controllers: Use tools like OPA Gatekeeper or custom admission webhooks to enforce specific pull policies.

5. Best Practices

For production environments: Use specific tags (e.g., v1.2.3) and IfNotPresent to avoid unintentional updates.

For development or CI/CD pipelines: Use latest tags and Always to ensure the most recent image is pulled.

Secure access to private registries: Use imagePullSecrets.

Use Never cautiously: Ensure images are correctly preloaded on all nodes if using Never.

Let me know if you'd like help configuring an imagePullPolicy for a specific scenario!

**--------------------------------------------------EKS Cronjob ---------------------------------**

**Allowed restart Policy for Different Kubernetes Resources**

* **Pod**: Can use Always, OnFailure, or Never.
* **Deployment**, **ReplicaSet**, and **DaemonSet**: Only Always is allowed, as these controllers manage Pod lifecycles to maintain desired states.
* **Job** and **CronJob**: Can use OnFailure or Never. These resources handle workloads where tasks should complete and stop rather than run continuously.

**Example Usage**

yaml

Copy code

# Pod with OnFailure restartPolicy

apiVersion: v1

kind: Pod

metadata:

name: example-pod

spec:

containers:

- name: example-container

image: busybox

command: ["sh", "-c", "echo Hello Kubernetes!"]

restartPolicy: OnFailure

If you have specific needs around failure handling, combining OnFailure with configurations like backoffLimit and activeDeadlineSeconds (for Jobs and CronJobs) can provide finer control over retries and timing.

----------------------------------------shell script for back in s3 local and S3---------------------------

**Updated Shell Script: backup\_and\_transfer\_to\_s3.sh**

**#!/bin/bash**

**# Variables**

S3\_BUCKET\_NAME="your-s3-bucket-name” **# Replace with your actual S3 bucketname**

**S3\_DESTINATION\_PATH="s3://${S3\_BUCKET\_NAME}/mongodb-backup/"**

**LOCAL\_BACKUP\_DIR="/home/ubuntu/allbackup/"**

**# Step 1: Run mongodump on the MongoDB pod**

**kubectl exec -it mongodb-pod -- mongodump --out "$LOCAL\_BACKUP\_DIR"**

**# Step 2: Copy the backup files from the pod to the EC2 instance**

kubectl cp mongodb-pod:/home/ubuntu/allbackup/ "$LOCAL\_BACKUP\_DIR"

**# Step 3: List the backup directory to confirm files are present**

ls -la "$LOCAL\_BACKUP\_DIR"

**# Step 4: Copy the backup files from EC2 to S3 bucket**

aws s3 cp "$LOCAL\_BACKUP\_DIR" "$S3\_DESTINATION\_PATH" --recursive

**# Optional Step 5: Copy the backup files from EC2 to the local machine (if needed)**

scp -i "minikub.pem" -r ubuntu@ec2-52-207-148-195.compute-1.amazonaws.com:/home/ubuntu/ssc/ "/c/Users/TS/3D Objects/"

--------------------------**Without S3 Bucket Step-------------**

**#!/bin/bash**

**# Variables**

**LOCAL\_BACKUP\_DIR="/home/ubuntu/allbackup/"**

**# Step 1: Run mongodump on the MongoDB pod**

**kubectl exec -it mongodb-pod -- mongodump --out "$LOCAL\_BACKUP\_DIR"**

**# Step 2: Copy the backup files from the pod to the EC2 instance**

**kubectl cp mongodb-pod:/home/ubuntu/allbackup/ "$LOCAL\_BACKUP\_DIR"**

**# Step 3: List the backup directory to confirm files are present**

**ls -la "$LOCAL\_BACKUP\_DIR"**

**# Optional Step 4: Copy the backup files from EC2 to the local machine (if needed)**

**scp -i "minikub.pem" -r ubuntu@ec2-52-207-148-195.compute-1.amazonaws.com:/home/ubuntu/ssc/ "/c/Users/TS/3D Objects/"**

**Instructions to Set Up with Cron**

1. **Make the Script Executable**: Save the script as backup\_and\_transfer\_to\_s3.sh and give it execute permissions:

chmod +x /path/to/backup\_and\_transfer\_to\_s3.sh

1. **Open Crontab Editor**: Run the following command to edit the crontab:

crontab -e

1. **Schedule the Script in Cron**: Add the following line to schedule the script. For instance, to run it daily at midnight:

0 0 \* \* \* /path/to/backup\_and\_transfer\_to\_s3.sh >> /path/to/backup\_log.txt 2>&1

Replace /path/to/backup\_and\_transfer\_to\_s3.sh with the actual path of the script file. This cron job will:

* + Run the script at midnight every day.
  + Log the output and errors to backup\_log.txt.

1. **Save and Close Crontab**: Save and close the crontab file to activate the cron job.

**Additional Notes**

* Ensure AWS CLI is configured on your EC2 instance with the necessary permissions to access the specified S3 bucket.
* Replace "your-s3-bucket-name" with your actual S3 bucket name.

This setup will now automatically:

1. Create the MongoDB backup in the pod.
2. Copy it to the EC2 instance.
3. Upload it to S3.
4. Optionally transfer it to your local machine if needed.

--------------------------------------------------SHELL----------------------------------------------------- shell script back**10 minutes**:

**#!/bin/bash**

**while true**

**do**

**kubectl exec -it <pod-name> -- mongodump --out /var/backups/mongobackups/$(date +'%m-%d-%y\_%H-%M')**

**sleep 600 # Wait for 10 minutes**

**done**

**Explanation:**

1. **Replace <pod-name>**: Substitute <pod-name> with the actual name of your MongoDB pod.
2. **sleep 600**: This makes the script pause for 10 minutes between executions.
3. **Save the script**: Save this script as backup-script.sh.

**How to Use:**

1. **Make the script executable**:

chmod +x backup-script.sh

1. **Run the script**:

./backup-script.sh

**Stopping the Script:**

* Use Ctrl+C to stop the script manually.
* Alternatively, run it in the background using nohup:

nohup ./backup-script.sh &

------------------------------**Updated Script for Day and Week Backups**---------------------------

#!/bin/bash

while true

do

# Generate timestamps

DATE=$(date +'%m-%d-%y\_%H-%M') # Current date and time

DAY=$(date +'%A') # Day of the week (e.g., Monday, Tuesday)

WEEK=$(date +'%U') # Week number of the year

# Define backup paths

DAILY\_PATH="/var/backups/mongobackups/daily/$DAY"

WEEKLY\_PATH="/var/backups/mongobackups/weekly/week-$WEEK"

# Ensure directories exist

kubectl exec -it <pod-name> -- mkdir -p $DAILY\_PATH

kubectl exec -it <pod-name> -- mkdir -p $WEEKLY\_PATH

# Perform daily backup

kubectl exec -it <pod-name> -- mongodump --out "$DAILY\_PATH/$DATE"

# Perform weekly backup (e.g., on Sunday)

if [ "$DAY" == "Sunday" ]; then

kubectl exec -it <pod-name> -- mongodump --out "$WEEKLY\_PATH/$DATE"

fi

# Wait for 5 minutes

sleep 300

#!/bin/bash

while true

do

# Generate timestamps

DATE=$(date +'%m-%d-%y\_%H-%M') # Current date and time

DAY=$(date +'%A') # Day of the week (e.g., Monday, Tuesday)

WEEK=$(date +'%U') # Week number of the year

# Define backup paths

DAILY\_PATH="/var/backups/mongobackups/daily/$DAY"

WEEKLY\_PATH="/var/backups/mongobackups/weekly/week-$WEEK"

# Ensure directories exist

kubectl exec -it <pod-name> -- mkdir -p $DAILY\_PATH

kubectl exec -it <pod-name> -- mkdir -p $WEEKLY\_PATH

# Perform daily backup

kubectl exec -it <pod-name> -- mongodump --out "$DAILY\_PATH/$DATE"

# Perform weekly backup (e.g., on Sunday)

if [ "$DAY" == "Sunday" ]; then

kubectl exec -it <pod-name> -- mongodump --out "$WEEKLY\_PATH/$DATE"

fi

# Wait for 5 minutes

sleep 300

done

**Explanation:**

1. **Timestamps**:
   * $DATE: Includes the full timestamp (MM-DD-YY\_HH-MM).
   * $DAY: Day of the week (e.g., Monday, Tuesday).
   * $WEEK: Week number of the year (e.g., week-45).
2. **Paths**:
   * DAILY\_PATH: Stores daily backups in /var/backups/mongobackups/daily/<day-of-week>.
   * WEEKLY\_PATH: Stores weekly backups in /var/backups/mongobackups/weekly/<week-number>.
3. **mkdir -p**: Ensures that the backup directories exist before running mongodump.
4. **Weekly Backup Logic**:
   * Only performs weekly backups on Sundays (if [ "$DAY" == "Sunday" ]).
5. **Interval**:
   * Script waits for 5 minutes (sleep 300) between executions.

**How It Works:**

* Every 5 minutes, the script creates a backup in the daily folder based on the current day.
* On Sundays, it also creates a backup in the weekly folder for the current week.

**Running the Script:**

1. Save the script as backup-day-week.sh.
2. Make it executable:

chmod +x backup-day-week.sh

1. Run it:

./backup-day-week.sh

This will ensure you have organized daily and weekly backups automatically!

---------------------------------------------**Docker** ----------------------------------------------------------

use **admin**

db.auth("username", "password")

use **admin**

db.grantRolesToUser("admin", [{ role: "dbAdmin", db: "http" }, { role: "readWrite", db: "http"}])

use **admin**

**Grant the root Role to the User** Run the following command to assign the root role to the user, which grants full permissions:

db.updateUser("admin", { roles: ["root"] })

// **Switch to admin database (Note: role management must be done from admin database)**

use **admin**

// **Grant multiple roles to admin user for demotask database**

db.grantRolesToUser(

"admin",

[

{ role: "readWrite", db: "demotask" },

{ role: "dbAdmin", db: "demotask" },

{ role: "dbOwner", db: "demotask" }

]

)

// **Verify the roles**

db.getUser("admin")

Here’s an example of how you could define a **CronJob** that runs the backup daily

apiVersion: batch/v1

kind: CronJob

metadata:

name: mongodb-backup

spec:

schedule: "1 \* \* \* \*" # Run daily based on REQ

jobTemplate:

spec:

template:

spec:

containers:

- name: mongodb-backup

image: mongo:latest

command:

- /bin/sh

- -c

- |

mongodump --archive=/data/db/mongodb-backup.archive --gzip &&

aws s3 cp /data/db/mongodb-backup.archive s3://your-bucket-name/mongodb-backup.archive

restartPolicy: OnFailure

To automate this backup process, including uploading it to S3, you can use the following script and set it up with cron (often misinterpreted as “corntaps”). Here’s an updated script that includes an S3 upload step and instructions on scheduling it using cron.

Updated Shell Script: backup\_and\_transfer\_to\_s3.sh

bash

Copy code

#!/bin/bash

# Variables

S3\_BUCKET\_NAME="your-s3-bucket-name" # Replace with your actual S3 bucket name

S3\_DESTINATION\_PATH="s3://${S3\_BUCKET\_NAME}/mongodb-backup/"

LOCAL\_BACKUP\_DIR="/home/ubuntu/allbackup/"

# Step 1: Run mongodump on the MongoDB pod

kubectl exec -it mongodb-pod -- mongodump --out "$LOCAL\_BACKUP\_DIR"

# Step 2: Copy the backup files from the pod to the EC2 instance

kubectl cp mongodb-pod:/home/ubuntu/allbackup/ "$LOCAL\_BACKUP\_DIR"

# Step 3: List the backup directory to confirm files are present

ls -la "$LOCAL\_BACKUP\_DIR"

# Step 4: Copy the backup files from EC2 to S3 bucket

aws s3 cp "$LOCAL\_BACKUP\_DIR" "$S3\_DESTINATION\_PATH" --recursive

# Optional Step 5: Copy the backup files from EC2 to the local machine (if needed)

scp -i "minikub.pem" -r ubuntu@ec2-52-207-148-195.compute-1.amazonaws.com:/home/ubuntu/ssc/ "/c/Users/TS/3D Objects/"

Instructions to Set Up with Cron

Make the Script Executable: Save the script as backup\_and\_transfer\_to\_s3.sh and give it execute permissions:

bash

Copy code

chmod +x /path/to/backup\_and\_transfer\_to\_s3.sh

Open Crontab Editor: Run the following command to edit the crontab:

bash

Copy code

crontab -e

Schedule the Script in Cron: Add the following line to schedule the script. For instance, to run it daily at midnight:

bash

Copy code

0 0 \* \* \* /path/to/backup\_and\_transfer\_to\_s3.sh >> /path/to/backup\_log.txt 2>&1

Replace /path/to/backup\_and\_transfer\_to\_s3.sh with the actual path of the script file. This cron job will:

Run the script at midnight every day.

Log the output and errors to backup\_log.txt.

Save and Close Crontab: Save and close the crontab file to activate the cron job.

Additional Notes

Ensure AWS CLI is configured on your EC2 instance with the necessary permissions to access the specified S3 bucket.

Replace "your-s3-bucket-name" with your actual S3 bucket name.

This setup will now automatically:

Create the MongoDB backup in the pod.

Copy it to the EC2 instance.

Upload it to S3.

Optionally transfer it to your local machine if needed.

#!/bin/bash

while true

do

kubectl exec -it <pod-name> -- mongodump --out /var/backups/mongobackups/$(date +'%m-%d-%y\_%H-%M')

sleep 600 # Wait for 10 minutes

done

#!/bin/bash

while true

do

kubectl exec -it <pod-name> -- mongodump --out /var/backups/mongobackups/$(date +'%m-%d-%y\_%H-%M')

sleep 600 # Wait for 10 minutes

Done

--------------------------------------------------------------------------------------------------------------------------------------

To perform an incremental backup on a MongoDB pod in an EKS cluster, use the mongodump command with the --oplog option, which leverages the MongoDB oplog to capture only changes since the last backup, effectively creating an incremental backup; you'll need to manage a timestamp file to track the last backup point and incorporate it into your command to target only new data.

Key components of the command:

MongoDB: The MongoDB command to create a database backup.

--op-log: Specifies to use the op-log for incremental backups.

--query (optional): Filter data based on specific criteria.

--archive: Create a compressed archive of the backup.

mongodump --uri "mongodb://<mongodb-pod-hostname>:<port>" --oplog --archive "<backup-directory>/mongo\_incremental\_backup\_$(date +%Y-%m-%dT%H-%M-%SZ).gz" --gzip

Important considerations:

Timestamp tracking:

Create a separate file to store the timestamp of the last incremental backup.

Read this timestamp before each backup to use in your query to filter out previously backed up data.

Backup directory:

Specify a dedicated directory on your EKS cluster to store incremental backups.

Accessing the MongoDB pod:

Replace <mongodb-pod-hostname> with the actual hostname of your MongoDB pod in the EKS cluster.

How to implement incremental backups:

1. Set up a timestamp file:

Create a file (e.g., last\_backup\_timestamp) in your backup directory to store the last backup timestamp.

2. Write a script:

Create a shell script that:

Reads the last backup timestamp from the file.

Uses the --query option with a timestamp filter in the mongodump command to capture only new data.

Updates the timestamp file with the current time after a successful backup

#!/bin/bash

LAST\_TIMESTAMP=$(cat last\_backup\_timestamp)

mongodump --uri "mongodb://<mongodb-pod-hostname>:<port>" --oplog --query '{ "timestamp": { "$gt": ISODate("'"$LAST\_TIMESTAMP"'") } }' --archive "<backup-directory>/mongo\_incremental\_backup\_$(date +%Y-%m-%dT%H-%M-%SZ).gz" --gzip

echo $(date +%Y-%m-%dT%H-%M-%SZ) > last\_backup\_timestamp