**🛠️ Roboshop Project - Backend Setup: MongoDB**

**📌 Introduction to MongoDB**

**MongoDB** is a leading NoSQL database used for storing large volumes of unstructured or semi-structured data.

* Stores data in **BSON (Binary JSON)** format.
* Ideal for dynamic, scalable, real-time applications.

**🔧 MongoDB Installation and Configuration (Version: 7.x)**

**1️. Setup MongoDB Repository**

Create the repo file:

vim /etc/yum.repos.d/mongo.repo

Add the following content:

[mongodb-org-7.0]

name=MongoDB Repository

baseurl=https://repo.mongodb.org/yum/redhat/9/mongodb-org/7.0/x86\_64/

enabled=1

gpgcheck=0

**2️. Install MongoDB**

dnf install mongodb-org -y

**3️. Start and Enable MongoDB Service**

systemctl enable mongod

systemctl start mongod

**4️. Allow Remote Connections (Change Bind IP)**

By default, MongoDB binds only to localhost (127.0.0.1). To allow access from other servers:

1. Edit the config file:

vim /etc/mongod.conf

1. Update the **bindIp** under net section:

bindIp: 0.0.0.0

**5️. Restart MongoDB to Apply Changes**

systemctl restart mongod

Below is a detailed set of notes for the **Catalogue** microservice setup for your **Roboshop** project:

**03-Catalogue Microservice Setup**

**Overview**

**Catalogue** is responsible for serving the list of items displayed in the Roboshop application. This microservice is built in **NodeJS** and requires a version greater than 20. The steps below cover installing NodeJS, configuring the application, setting up the systemd service, and seeding the database with schema and master data.

**1. NodeJS Setup**

**Check Current Modules**

* **List Available NodeJS Modules:**
* dnf module list nodejs

**Disable the Default Module (NodeJS 16)**

* **Disable NodeJS 16:**
* dnf module disable nodejs -y

**Enable Required Module (NodeJS 20)**

* **Enable NodeJS 20 Module:**
* dnf module enable nodejs:20 -y

**Install NodeJS**

* **Install the NodeJS package:**
* dnf install nodejs -y

**2. Application Configuration**

**Create Application User**

* **Add a system user (non-root) to run the application:**
* useradd --system --home /app --shell /sbin/nologin --comment "roboshop system user" roboshop

*Note: The user* ***roboshop*** *is created as a daemon user and is not meant for interactive login.*

**Setup Application Directory**

* **Create a standard directory to hold the application code:**
* mkdir /app

**Download & Extract Application Code**

* **Download the Catalogue application code:**
* curl -o /tmp/catalogue.zip https://roboshop-artifacts.s3.amazonaws.com/catalogue-v3.zip
* **Change to the application directory and extract:**
* cd /app
* unzip /tmp/catalogue.zip

**Install Application Dependencies**

* **Install NodeJS dependencies with npm:**
* cd /app
* npm install

**3. Systemd Service Setup**

**Create a systemd Service File**

* **Create the Catalogue service file:**
* vim /etc/systemd/system/catalogue.service
* **Populate the file with the following content (be sure to replace <MONGODB-SERVER-IPADDRESS> with the actual IP address):**
* [Unit]
* Description = Catalogue Service
* [Service]
* User=roboshop
* Environment=MONGO=true
* Environment=MONGO\_URL="mongodb://<MONGODB-SERVER-IPADDRESS>:27017/catalogue"
* ExecStart=/bin/node /app/server.js
* SyslogIdentifier=catalogue
* [Install]
* WantedBy=multi-user.target

**Load and Start the Service**

* **Reload the systemd daemon (to register the new service):**
* systemctl daemon-reload
* **Enable and start the Catalogue service:**
* systemctl enable catalogue
* systemctl start catalogue

**4. Seeding the MongoDB Database**

**Setup MongoDB Client**

* **Ensure the MongoDB repository is properly configured by creating/updating /etc/yum.repos.d/mongo.repo with:**
* [mongodb-org-7.0]
* name=MongoDB Repository
* baseurl=https://repo.mongodb.org/yum/redhat/9/mongodb-org/7.0/x86\_64/
* enabled=1
* gpgcheck=0
* **Install the MongoDB shell client:**
* dnf install mongodb-mongosh -y

**Load the Schema / Master Data**

* **Seed the master data (products and quantity info) for the Catalogue service:**
* mongosh --host <MONGODB-SERVER-IPADDRESS> </app/db/master-data.js

**Verify Database Setup**

* **Connect to MongoDB to check if the data is loaded:**
* mongosh --host <MONGODB-SERVER-IPADDRESS>
* **Within the MongoDB shell, run:**
  + Show databases:
  + show dbs
  + Use the catalogue database:
  + use catalogue
  + List collections:
  + show collections
  + Retrieve products:
  + db.products.find()

**5. Final Frontend Configuration**

* **Remember to update the Catalogue server IP address in the frontend configuration file (/etc/nginx/nginx.conf) to point to the correct server.**

**🖥️ 01 - Frontend (RoboShop)**

The **Frontend** service in RoboShop is used to serve the **web interface** (static content) to users. It acts as the **presentation layer** and communicates with backend microservices using **Nginx** as a **web server and reverse proxy**.

**🔧 Step-by-Step Setup**

**✅ 1. List Available Nginx Modules**

dnf module list nginx

**✅ 2. Install Required Nginx Version (1.24)**

dnf module disable nginx -y

dnf module enable nginx:1.24 -y

dnf install nginx -y

**✅ 3. Enable and Start Nginx Service**

systemctl enable nginx

systemctl start nginx

🔍 **Check:** Access the server's IP in a browser to confirm Nginx is serving the default welcome page.

**✅ 4. Remove Default Content**

rm -rf /usr/share/nginx/html/\*

**✅ 5. Download Frontend Application Code**

curl -o /tmp/frontend.zip https://roboshop-artifacts.s3.amazonaws.com/frontend-v3.zip

**✅ 6. Extract Application Code**

cd /usr/share/nginx/html

unzip /tmp/frontend.zip

🔍 **Check:** Reload the browser; you should now see the **RoboShop Frontend UI**.

**🔁 Reverse Proxy Configuration for Backend Communication**

📍 Frontend connects to microservices like **catalogue, user, cart, etc.** using **proxy\_pass** via Nginx.

**✅ 7. Edit Nginx Configuration**

vim /etc/nginx/nginx.conf

**🔧 Replace Existing Content with:**

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log notice;

pid /run/nginx.pid;

include /usr/share/nginx/modules/\*.conf;

events {

worker\_connections 1024;

}

http {

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

types\_hash\_max\_size 4096;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

include /etc/nginx/conf.d/\*.conf;

server {

listen 80;

listen [::]:80;

server\_name \_;

root /usr/share/nginx/html;

include /etc/nginx/default.d/\*.conf;

error\_page 404 /404.html;

location = /404.html { }

error\_page 500 502 503 504 /50x.html;

location = /50x.html { }

location /images/ {

expires 5s;

root /usr/share/nginx/html;

try\_files $uri /images/placeholder.jpg;

}

## Reverse Proxy for Backend Microservices

location /api/catalogue/ { proxy\_pass http://<CATALOGUE-SERVER-IP>:8080/; }

location /api/user/ { proxy\_pass http://<USER-SERVER-IP>:8080/; }

location /api/cart/ { proxy\_pass http://<CART-SERVER-IP>:8080/; }

location /api/shipping/ { proxy\_pass http://<SHIPPING-SERVER-IP>:8080/; }

location /api/payment/ { proxy\_pass http://<PAYMENT-SERVER-IP>:8080/; }

## Health Check Endpoint

location /health {

stub\_status on;

access\_log off;

}

}

}

🔁 **IMPORTANT:** Replace all localhost values (e.g., in proxy\_pass) with the **actual IP addresses** of the respective microservices running on other servers.

* Example for Catalogue:
* location /api/catalogue/ { proxy\_pass http://192.168.1.101:8080/; }

**✅ 8. Restart Nginx to Apply Changes**

systemctl restart nginx

**✅ Integration Notes**

| **Component** | **Description** | **Config Location** | **Action** |
| --- | --- | --- | --- |
| **Catalogue** | Backend for products | proxy\_pass in nginx.conf | Use IP of catalogue server |
| **MongoDB** | Used internally by catalogue (not by frontend) | Set in /etc/systemd/system/catalogue.service | Use MongoDB IP |
| **Other APIs** | user, cart, etc. | nginx.conf | Setup when those services are ready |

**✅ Final Test Checklist**

* Frontend UI accessible via browser
* Catalogue items are visible (means proxy + MongoDB working)
* Nginx restarted without errors: systemctl status nginx
* All proxy IPs correctly configured
* You can ping backend servers from frontend

**🔰 Goal:**

Assign domain names to your services like:

* frontend.roboshop.in → Frontend (Nginx server)
* catalogue.roboshop.in → Catalogue service
* mongodb.roboshop.in → MongoDB server

**✅ Prerequisites:**

* You have a **domain name** (e.g., roboshop.in)
* Your domain is managed in **Route 53 hosted zone**
* You know the **public IPs** (or private if inside a VPC) of your **Frontend, Catalogue**, and **MongoDB** servers

**🛠️ Step-by-Step: Creating DNS Records in Route 53**

**✅ Step 1: Open Hosted Zone**

1. Go to **AWS Console** → **Route 53**
2. Click on **"Hosted Zones"**
3. Choose your domain name (e.g., roboshop.in)

**✅ Step 2: Add A Records for Each Service**

**📍 For Frontend (Nginx Server):**

1. Click **"Create Record"**
2. Name: frontend
3. Record type: **A – IPv4 address**
4. Value: <Frontend server's public IP>
5. TTL: 300 (default is fine)
6. Click **"Create records"**

🧪 **Test:**  
Visit http://frontend.roboshop.in → Should load RoboShop UI

**📍 For Catalogue:**

1. Click **"Create Record"**
2. Name: catalogue
3. Type: **A**
4. Value: <Catalogue server public IP>
5. TTL: 300
6. Click **Create**

🧪 **Test:**  
From Nginx server:

curl http://catalogue.roboshop.in:8080/health

**📍 For MongoDB:**

MongoDB is **not accessed via browser**, only by backend services.  
Still, a DNS record helps for easier configuration.

1. Click **"Create Record"**
2. Name: mongodb
3. Type: **A**
4. Value: <MongoDB server public/private IP>
5. TTL: 300
6. Click **Create**

🧪 **Test:**  
From catalogue server:

ping mongodb.roboshop.in

**✅ Step 3: Update Configs with DNS Names**

**🔁 In Nginx (on Frontend):**

Replace:

proxy\_pass http://<CATALOGUE\_IP>:8080/;

with:

proxy\_pass http://catalogue.roboshop.in:8080/;

Then:

systemctl restart nginx

**🔁 In Catalogue Service (/etc/systemd/system/catalogue.service):**

Replace:

Environment=MONGO=true

Environment=MONGO\_URL="mongodb://<MONGO-IP>:27017/catalogue"

with:

Environment=MONGO=true

Environment=MONGO\_URL="mongodb://mongodb.roboshop.in:27017/catalogue"

Then:

systemctl daemon-reexec

systemctl restart catalogue

**🎯 Summary**

| **Service** | **Domain** | **IP Target** | **Purpose** |
| --- | --- | --- | --- |
| Frontend | frontend.roboshop.in | Public IP of Nginx | UI access via browser |
| Catalogue | catalogue.roboshop.in | Public/Private IP | API for products |
| MongoDB | mongodb.roboshop.in | Internal/External IP | DB access by backend service |

**💡 Why is the User Component dependent on MongoDB and Redis?**

**1. MongoDB:**

MongoDB is used as the **primary database** for the **User** component. Here's why:

* It stores **user-related data** like:
  + Usernames
  + Passwords (hashed)
  + Email, address, phone number, etc.
  + Login attempts, account status, timestamps

MongoDB is **NoSQL** and works well with flexible schemas, which is useful for dynamic user data that might change over time.

✅ **Example Use:**  
When a user registers or logs in, data is stored/retrieved from **MongoDB**.

**2. Redis:**

Redis is used for **caching** and **session/token management** in the User component. Here's why:

* Redis is **ultra-fast** because it stores data in **RAM**.
* Commonly used for:
  + **Storing user sessions** (e.g., once a user logs in, the session/token can be stored in Redis for quick validation)
  + **OTP storage** for email/phone verification (short-lived, temporary data)
  + **Login rate limiting** (e.g., blocking IPs after failed login attempts)
  + **Caching frequently used user data** to reduce MongoDB load

✅ **Example Use:**  
When a user logs in successfully, their session/token is stored in Redis with a timeout like 30 minutes.

**❓ What is Redis?**

Redis = **Remote Dictionary Server**

| **Feature** | **Details** |
| --- | --- |
| Type | In-memory, key-value data store |
| Speed | Very fast (data stored in **RAM**) |
| Use Cases | Cache, Session Store, Pub/Sub, Leaderboards, OTP storage, Rate limits |
| Access Pattern | Key-Value (e.g., session\_123 = user\_data) |
| Port | Default is **6379** |

**⚙️ Redis Configuration Explained:**

**✅ Install Redis 7:**

dnf module disable redis -y

dnf module enable redis:7 -y

dnf install redis -y

**✅ Allow External Access:**

By default, Redis only allows access from localhost for security.

To allow other components (like the **User service** on a different server) to connect:

1. **Change bind address:**
2. bind 0.0.0.0

(Accepts connections from all IPs)

1. **Disable protected mode:**
2. protected-mode no

(So it doesn't block remote access)

**✅ File to update:**

vim /etc/redis/redis.conf

**✅ Start Redis:**

systemctl enable redis

systemctl start redis

netstat -lntp

portnumber: 6379

**🔒 Security Tip:**

Once everything works, consider:

* **Restricting access** to Redis via firewall (firewalld or iptables)
* **Using authentication** (add requirepass <password> in redis.conf)
* Binding to internal IP only if all services are inside VPC or same private network

**🔁 Summary**

| **Dependency** | **Purpose** |
| --- | --- |
| MongoDB | Stores persistent user data like login info, profiles |
| Redis | Stores fast-access data like sessions, OTPs, rate limits |

In the **RoboShop project**, **Redis** plays a crucial backend role — even though it's not directly visible to end users, it is **extremely important for speed, scalability, and smooth functioning**. Here's exactly how Redis is **useful to this project**, especially in the **User** component:

**🧠 Redis Use Cases in RoboShop Project**

**1. Session Management (Login Tokens)**

* After a user logs in, a **token or session ID** is generated and stored in Redis.
* Every time the user performs a secure action (like checking their cart, payment, etc.), the app checks this token in Redis to verify if the session is still valid.
* Sessions are **temporary**, so Redis is perfect because it’s fast and allows **auto-expiry** using TTL (Time To Live).

✅ **Why Redis?** → RAM-based, ultra-fast access for millions of users.

**2. OTP Storage for Verification**

* When a user signs up or resets their password, an **OTP** (One-Time Password) is sent.
* This OTP is stored temporarily in Redis, e.g. for **5 minutes**.

✅ **Why Redis?** → Redis supports setting key expirations — OTPs automatically expire, which is ideal.

**3. Login Rate Limiting (Security)**

* Redis can store **login attempt counts per IP or user ID**.
* If a user fails to log in 5 times in 2 minutes, block them temporarily.

✅ **Why Redis?** → Real-time counter tracking and TTLs prevent brute-force attacks efficiently.

**4. Caching Frequently Accessed Data**

* Information like **user preferences**, **user profile**, or even **catalogue metadata** can be cached in Redis.
* Instead of querying MongoDB every time, the app first checks Redis.

✅ **Why Redis?** → Reduces load on MongoDB and improves performance.

**5. Inter-Component Communication (Advanced Use)**

* Redis supports **Pub/Sub messaging**, which advanced microservices can use to notify other components.
* Example: After user registration, send a message to Email Service.

(Not always implemented in beginner versions, but Redis supports this.)

**🔄 Flow Example: User Login using Redis**

1. User submits login form.
2. Backend checks MongoDB → User exists + password matches.
3. Backend generates a session token.
4. Stores:
5. session\_user123 = <token\_data> (expires in 30 mins)
6. Later, any request with this token will be validated using Redis (not MongoDB).

**📌 Summary Table**

| **Use Case** | **Redis Role** | **Benefit** |
| --- | --- | --- |
| User Login Sessions | Store token with TTL | Fast, avoids DB queries |
| OTP Verification | Store temporary OTP | Auto-expiry, secure |
| Rate Limiting | Track login attempts | Prevent brute-force attacks |
| Data Caching | Store frequently used data | Improve performance |
| Pub/Sub Messaging (optional) | Notify services via channels | Event-driven architecture support |

**🎯 Why Redis is Essential to RoboShop**

Without Redis:

* Sessions would be stored in DB → slower performance
* No fast way to expire OTPs or tokens
* Higher load on MongoDB → slower user experience
* Increased risk of brute-force attacks without rate-limiting

With Redis:  
✅ Performance  
✅ Security  
✅ Scalability  
✅ Simplicity in session/OTP/token handling

**🛡️ What is protected-mode in Redis?**

* protected-mode is a **safety feature** in Redis to **prevent unauthorized access**.
* When enabled (yes), Redis **only allows connections from localhost (127.0.0.1)**.
* If any external machine tries to access Redis, it will **reject the connection**, even if the port is open.

**🤔 Why are we changing protected-mode to no?**

In the RoboShop project:

**🔗 Redis is a shared service:**

* Redis is installed on one server.
* But the **User component** (which depends on Redis) might be running on **another server**.
* For the User component to talk to Redis **over the network**, Redis must **accept external connections** (not just localhost).

**✅ So, we change:**

bind 127.0.0.1

➡️ To:

bind 0.0.0.0

And:

protected-mode yes

➡️ To:

protected-mode no

This tells Redis:

* **“Allow connections from other servers”** (like the User service)
* **“Do not block access just because it's not from localhost”**

**⚠️ Important Security Note**

Setting protected-mode no and bind 0.0.0.0 **exposes Redis** over the network.

So in production:

* You should **secure it with a firewall**
* Or **bind only to your private subnet**, not public IP
* Or configure **authentication (requirepass)** in Redis

But for internal use like RoboShop in a **private cloud or lab**, this setup is okay.

**🧠 Summary**

| **Setting** | **Meaning** | **Why We Change It** |
| --- | --- | --- |
| protected-mode yes | Only accept local (127.0.0.1) calls | Secure but blocks other servers |
| protected-mode no | Accept remote connections | Needed for microservices like user |

**🚀 User Component Setup Guide**

**Component Name:** User  
**Function:** Manages user registrations and login for the RoboShop e-commerce platform  
**Dependencies:** Node.js, MongoDB, Redis

**🧱 Step 1: Install Node.js 20**

By default, CentOS provides Node.js 16. We need Node.js 20.

dnf module disable nodejs -y

dnf module enable nodejs:20 -y

dnf install nodejs -y

**👤 Step 2: Create Application User**

Applications should not run as root. We’ll create a system user named roboshop.

useradd --system --home /app --shell /sbin/nologin --comment "roboshop system user" roboshop

**📁 Step 3: Set Up Application Directory**

Create a standard directory for the app and download the code.

mkdir /app

curl -L -o /tmp/user.zip https://roboshop-artifacts.s3.amazonaws.com/user-v3.zip

cd /app

unzip /tmp/user.zip

**📦 Step 4: Install Dependencies**

Node.js apps use npm to install required packages.

cd /app

npm install

**⚙️ Step 5: Configure systemd Service**

Create a new service so systemd can manage the app.

Create the file:

vim /etc/systemd/system/user.service

Add the following content (replace <REDIS-IP> and <MONGODB-IP>):

[Unit]

Description=User Service

[Service]

User=roboshop

Environment=MONGO=true

Environment=REDIS\_URL='redis://<REDIS-IP>:6379'

Environment=MONGO\_URL='mongodb://<MONGODB-IP>:27017/users'

ExecStart=/bin/node /app/server.js

SyslogIdentifier=user

[Install]

WantedBy=multi-user.target

**🔄 Step 6: Start the Service**

Reload systemd to detect the new service and start it.

systemctl daemon-reload

systemctl enable user

systemctl start user

**🔍 1. netstat -lntp**

**✅ Purpose:**

To check which services are **listening on which ports** on the current server.

**🔧 Example Output:**

tcp 0 0 0.0.0.0:6379 0.0.0.0:\* LISTEN 1234/redis-server

tcp 0 0 0.0.0.0:27017 0.0.0.0:\* LISTEN 2345/mongod

✅ If you see Redis (6379) and MongoDB (27017) listening on 0.0.0.0, **they are accessible from other servers.**

❌ If bound to 127.0.0.1, update config files:

* /etc/redis/redis.conf: change bind 127.0.0.1 to bind 0.0.0.0
* /etc/mongod.conf: set bindIp: 0.0.0.0

**📡 2. telnet redis.<yourdomain> 6379**

**✅ Purpose:**

To check if **User service server** can reach **Redis server** via port 6379.

**🔧 Example:**

telnet redis.roboshop.internal 6379

**✅ Expected Output:**

Trying <redis-IP>...

Connected to redis.roboshop.internal.

Escape character is '^]'.

This means Redis is reachable!

❌ If you get:

Connection refused

→ Redis is not listening on that IP/port (check Redis config and firewall).

❌ If you get:

Unable to resolve host

→ DNS name is wrong or not resolvable (check hostname or add to /etc/hosts).

**📡 3. telnet mongodb.<yourdomain> 27017**

**✅ Purpose:**

To check if **User service** can connect to **MongoDB server** on port 27017.

**🔧 Example:**

telnet mongodb.roboshop.internal 27017

**✅ Expected Output:**

Trying <mongo-IP>...

Connected to mongodb.roboshop.internal.

Escape character is '^]'.

Same troubleshooting as Redis applies here if it fails.

**✅ Summary Checks**

| **Service** | **Port** | **Command** | **What to See** |
| --- | --- | --- | --- |
| Redis | 6379 | telnet redis.<domain> 6379 | Connected |
| MongoDB | 27017 | telnet mongodb.<domain> 27017 | Connected |
| Both | — | netstat -lntp | Listening on 0.0.0.0 |

**🛒 Cart Microservice Overview**

* **Function:** Manages **user's cart items** — add to cart, update quantity, delete items.
* **Tech Stack:** Node.js (>20), Redis, Catalogue Service (for product info).

**🔗 Why Cart Depends on:**

**✅ 1. Redis**

Redis is used by the Cart service to:

* **Store temporary cart data** in memory (user’s cart items before placing an order).
* **Provide high-speed read/write** of cart items (since Redis is in-memory and super fast).
* Acts like a **session store** to cache user-specific cart details.

📌 Example: When a user adds a product to their cart, the cart data is stored in Redis against their session/user ID.

**✅ 2. Catalogue**

The Catalogue service provides:

* **Product info** like name, price, availability.
* Cart service uses this to:
  + Show item details while displaying the cart.
  + Ensure the product is available and price is accurate.

📌 Example: When user adds a product with ID 123, Cart queries Catalogue to fetch Product Name, Price, etc.

**⚙️ Step-by-Step Cart Setup Summary**

**📌 Step 1: Enable Node.js 20**

dnf module disable nodejs -y

dnf module enable nodejs:20 -y

dnf install nodejs -y

**📌 Step 2: Create Roboshop Application User**

useradd --system --home /app --shell /sbin/nologin --comment "roboshop system user" roboshop

**📌 Step 3: Setup App Directory**

mkdir /app

**📌 Step 4: Download Cart App Code**

curl -L -o /tmp/cart.zip https://roboshop-artifacts.s3.amazonaws.com/cart-v3.zip

cd /app

unzip /tmp/cart.zip

**📌 Step 5: Install App Dependencies**

cd /app

npm install

**📌 Step 6: Create Cart Systemd Service**

Use vim /etc/systemd/system/cart.service and paste:

[Unit]

Description = Cart Service

[Service]

User=roboshop

Environment=REDIS\_HOST=<REDIS-IP>

Environment=CATALOGUE\_HOST=<CATALOGUE-IP>

Environment=CATALOGUE\_PORT=8080

ExecStart=/bin/node /app/server.js

SyslogIdentifier=cart

[Install]

WantedBy=multi-user.target

👉 Replace <REDIS-IP> and <CATALOGUE-IP> with actual internal IPs or DNS like redis.roboshop.internal.

**📌 Step 7: Start and Enable Service**

systemctl daemon-reload

systemctl enable cart

systemctl start cart

**✅ Optional: Connectivity Test (like you did for Redis and MongoDB)**

From Cart server:

telnet redis.roboshop.internal 6379

telnet catalogue.roboshop.internal 8080

**🎯 Goal**

**Create DNS records in Route 53** so that internal services can communicate using names like:

* cart.roboshop.internal
* user.roboshop.internal
* redis.roboshop.internal

**✅ Steps to Create Route 53 Records**

**1️. Go to AWS Console → Route 53 → Hosted Zones**

You should already have a **hosted zone** like:

roboshop.internal

If not, create one as a **Private Hosted Zone** (for internal resolution within the VPC).

**2️. Inside Hosted Zone roboshop.internal, create A records for:**

| **Name** | **Type** | **Value (IP of EC2 Instance)** |
| --- | --- | --- |
| cart | A | IP address of EC2 instance running Cart |
| user | A | IP address of EC2 instance running User |
| redis | A | IP address of EC2 instance running Redis |

➡️ Example for Cart:

Record name: cart

Record type: A

Value: 10.0.2.15 (Example internal IP of Cart EC2)

TTL: 300

Repeat for user and redis.

**🛠️ Where to Make Changes in Configuration Files**

After creating DNS records, update the respective **.service** files with the DNS names instead of IPs.

**🔁 1. Cart Component**

Edit:

vim /etc/systemd/system/cart.service

Replace:

Environment=REDIS\_HOST=redis.roboshop.internal

Environment=CATALOGUE\_HOST=catalogue.roboshop.internal

**🔁 2. User Component**

Edit:

vim /etc/systemd/system/user.service

Replace:

Environment=REDIS\_URL=redis://redis.roboshop.internal:6379

Environment=MONGO\_URL=mongodb://mongodb.roboshop.internal:27017/users

**🔁 3. Redis Doesn't Call Anyone**

Redis just **receives** traffic, so **no config updates needed** on Redis. You only need to make sure **its name (redis.roboshop.internal)** is correctly used by User, Cart, and others.

**🔄 Final Steps After Changes**

Run these commands on **each EC2 instance** where you made .service file updates:

systemctl daemon-reload

systemctl restart cart # on cart server

systemctl restart user # on user server

**✅ Verify DNS Resolution**

Use this command on any instance:

nslookup redis.roboshop.internal

nslookup user.roboshop.internal

nslookup cart.roboshop.internal

If you get the correct internal IPs, DNS is working fine.

**🎯 Why Changes Are Needed in Frontend?**

The frontend makes HTTP API calls to the backend microservices (like user, cart, catalogue). So, if you change the backend service endpoints (from IPs to DNS names like user.roboshop.internal), the **frontend must also be updated to use these DNS names**.

**✅ Steps to Update Frontend Configuration**

**1️. Go to the Frontend EC2 Instance**

**2️. Edit the frontend service configuration file:**

Assuming you're using Nginx for frontend reverse proxying:

vim /etc/nginx/nginx.conf

**3️. Update API URLs**

Update this file to use DNS names like:

nginx

location /api/user/ {

proxy\_pass http://user.roboshop.internal:8080/;

}

location /api/cart/ {

proxy\_pass http://cart.roboshop.internal:8080/;

}

location /api/catalogue/ {

proxy\_pass http://catalogue.roboshop.internal:8080/;

}

🔁 Update all other relevant proxy\_pass lines similarly (e.g. shipping, payment, etc. when you reach them).

**4️. Restart Nginx to apply changes:**

systemctl restart nginx