Assignment : 1

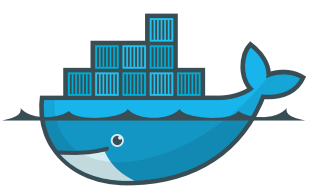
PRESENTED TO	PRESENTED BY	PRESENTED ON
Gurucool	**Arpit Kumar Singh**	- - -

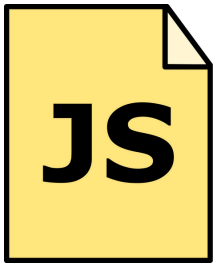
Introduction

The Gurucool Backend System Docker image is a containerized version of a robust and scalable backend system designed to efficiently manage requests from multiple users using a queue structure. This Docker image encapsulates the entire backend system, including the Node.js server, user authentication, request queuing with RabbitMQ, MongoDB database integration, and worker processes for request processing.

Technologies Used

- **MongoDB Atlas:** Database for storing and managing data.
- **Express.js:** Backend framework for handling API requests.
- **Docker:** Docker image encapsulates the entire backend system.
- **Node.js:** Runtime environment for server-side code.
- **Rabbitmq:** RabbitMQ is utilized as the messaging and queuing system in the backend..



User Authentication:

- Implements user authentication functionality.
- Handles login requests and generates JWT tokens.
- Validates user credentials against a database.
- Returns appropriate responses for successful and failed authentication attempts.

```
// Login
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username && u.password === password);
  if (user) {

    jwt.sign({ user }, jwtSecret, { expiresIn: '1h' }, (err, token) => {
      if (err) {
        res.status(500).json({ error: 'Failed to create token' });
      } else {
        res.json({ token });
      }
    });
  } else {
    res.status(401).json({ error: 'Invalid credentials' });
  }
});
```

```
// Protected
app.get('/protected', verifyToken, (req, res) => {
  jwt.verify(req.token, jwtSecret, (err, authData) => {
    if (err) {
      res.sendStatus(403);
    } else {
      res.json({ message: 'Authenticated successfully', data: authData });
    }
  });
});
```

```
// Middleware
function verifyToken(req, res, next) {
  const bearerHeader = req.headers['authorization'];
  if (typeof bearerHeader !== 'undefined') {
    const bearerToken = bearerHeader.split(' ')[1];
    req.token = bearerToken;
    next();
  } else {
    res.sendStatus(403);
  }
}
```

Worker Process :

- Implements worker processes responsible for pulling requests from RabbitMQ queues and processing them.
- Executes queued requests sequentially, ensuring FIFO (First-In-First-Out) order.
- Logs processing status and handles message acknowledgement to ensure reliable message processing.

```
const amqp = require('amqplib/callback_api');


amqp.connect('amqp://localhost', (error0, connection) => {
  if (error0) {
    throw error0;
  }

  connection.createChannel((error1, channel) => {
    if (error1) {
      throw error1;
    }
    const queue = 'Gurucool';

    channel.assertQueue(queue, {
      durable: true
    });
    console.log(`Worker process is waiting for messages in ${queue}`);

    channel.consume(queue, (msg) => {
      const content = msg.content.toString();
      console.log(`Worker process received message: ${content}`);

      setTimeout(() => {
        console.log(`Worker process processed message: ${content}`);
        channel.ack(msg);
      }, 1000);
    }, {
      noAck: false
    });
  });
});
```

Publish.js:

```
const amqp = require('amqplib/callback_api');

// RabbitMQ connection URL
const url = 'amqp://localhost';

// Message to publish
const message = 'Hello RabbitMQ!';

// Connect to RabbitMQ server
amqp.connect(url, (error0, connection) => {
  if (error0) {
    throw error0;
  }
  // Create channel
  connection.createChannel((error1, channel) => {
    if (error1) {
      throw error1;
    }
    const queue = 'Gurucool';
    // Declare queue
    channel.assertQueue(queue, {
      durable: true
    });
    // Publish message to queue
    channel.sendToQueue(queue, Buffer.from(message), {
      persistent: true
    });
    console.log(`Message '${message}' sent to queue '${queue}'`);
  });
});
```

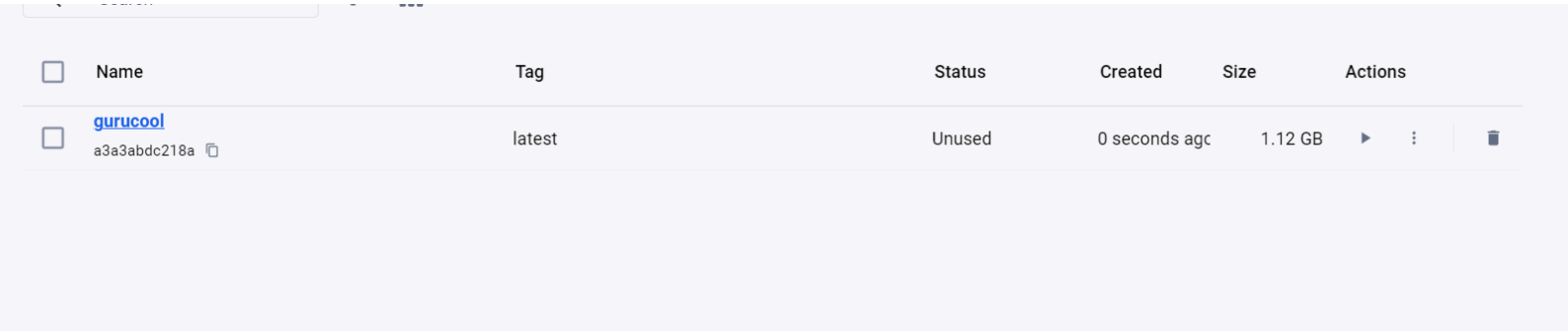
Docker:

- We created a Dockerfile to define the instructions for building our Docker image.
- The Dockerfile included commands to set up the Node.js environment, install dependencies, and copy the application code.

```
FROM node:latest

WORKDIR /home/gurucool

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app.js"]
```

1. **Docker Image Building**:
   - We used the docker build command to build a Docker image based on the Dockerfile.
   - The image was tagged with a name (gurucool) using the -t flag.

```
PS C:\Users\arpit\OneDrive\Desktop\Gurucool> docker build -t gurucool .
[+] Building 152.1s (10/10) FINISHED                                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                                             0.0s
 => => transferring dockerfile: 178B                                                                                             0.0s
 => [internal] load metadata for docker.io/library/node:latest                                                                   3.2s
 => [internal] load .dockerignore                                                                                                0.0s
 => => transferring context: 2B                                                                                                  0.0s
 => [1/5] FROM docker.io/library/node:latest@sha256:cbd62dc7ba7e50d01520f2c0a8d9853ec872187fa806ed61d0f87081c220386d           143.4s
 => => resolve docker.io/library/node:latest@sha256:cbd62dc7ba7e50d01520f2c0a8d9853ec872187fa806ed61d0f87081c220386d             0.0s
 => => sha256:622db8d5507df7bf5bf41eedc84fdb3a31a41f053affc8a48d7caec1122ab1ad 2.00kB / 2.00kB                                   0.0s
 => => sha256:a1c1026b1a58d8dabc8554808ad296d41a28ebe1cd3ee27264bce9793b708e27 7.41kB / 7.41kB                                   0.0s
 => => sha256:1468e7ff95fcb865fbc4dee7094f8b99c4dcddd6eb2180cf044c7396baf6fc2f 49.58MB / 49.58MB                                37.5s
 => => sha256:2cf9c2b42f41b1845f3e4421b723d56146db82939dc884555e077768e18132f4 24.05MB / 24.05MB                                24.5s
 => => sha256:c4c40c3e3cdf945721f480e1d939aac857876fdb5c33b8fbfcf655c63b0b9428 64.14MB / 64.14MB                                56.1s
 => => sha256:cbd62dc7ba7e50d01520f2c0a8d9853ec872187fa806ed61d0f87081c220386d 1.21kB / 1.21kB                                   0.0s
 => => sha256:c05cc1123d7e335d59b0f465c23b7ad2ad27f4875b6c3eab41c65a9b50efa382 211.18MB / 211.18MB                              109.6s
 => => sha256:04f2356c02d21ef160986f7994210be28dab889b9ba0724e126bd2ee4ed21cd6 3.37kB / 3.37kB                                  38.5s
 => => extracting sha256:1468e7ff95fcb865fbc4dee7094f8b99c4dcddd6eb2180cf044c7396baf6fc2f                                        5.8s
 => => sha256:eb8bd4891eecea478e2d9939d8592535190e186d434faaf15b74633b198129a3 52.12MB / 52.12MB                                70.1s
 => => extracting sha256:2cf9c2b42f41b1845f3e4421b723d56146db82939dc884555e077768e18132f4                                        1.7s
```



Conclusion

In conclusion, we successfully designed and implemented a robust backend system using Node.js, RabbitMQ, MongoDB, and Docker. Throughout the process, we focused on achieving the following objectives:

1. **User Authentication**: Implemented secure user authentication using JWT tokens to authenticate users before they can enqueue requests.
2. **Request Queueing**: Established a queue for each client to handle requests in a First-In-First-Out (FIFO) manner, ensuring efficient request management.
3. **Request Processing**: Developed a process to handle and execute requests sequentially, ensuring that each request is processed in the order it was received.
4. **Concurrency Management**: Managed multiple clients and their queues concurrently, enabling the system to handle a large number of users and requests simultaneously.