SOFTWARE PRODUCTION ENGINEERING

Food Menu App

Akhil Puppala (IMT2021021)
Josh Jack (IMT2021515)

1. Introduction:

This is a general-purpose food menu application designed for restaurants, featuring functionalities such as login, signup, cart, and My Orders. The application is built using the MERN stack: MongoDB, Express.js, React, and Node.js.

Github repo link: https://github.com/AkhilPuppala/FoodDelivery.git

Docker frontend container: akhilpuppala/frontend-image

Docker backend container: akhilpuppala/backend-image

DevOps tools:

- **Source Control Management:** Git and GitHub
- **Continuous Integration Pipeline:** Jenkins
- **Containerization:** Docker
- **Container Orchestration:** Docker compose, Kubernetes
- **Front End:** React, bootstrap
- **Monitoring:** ELK Stack
- **Database:** MongoDB

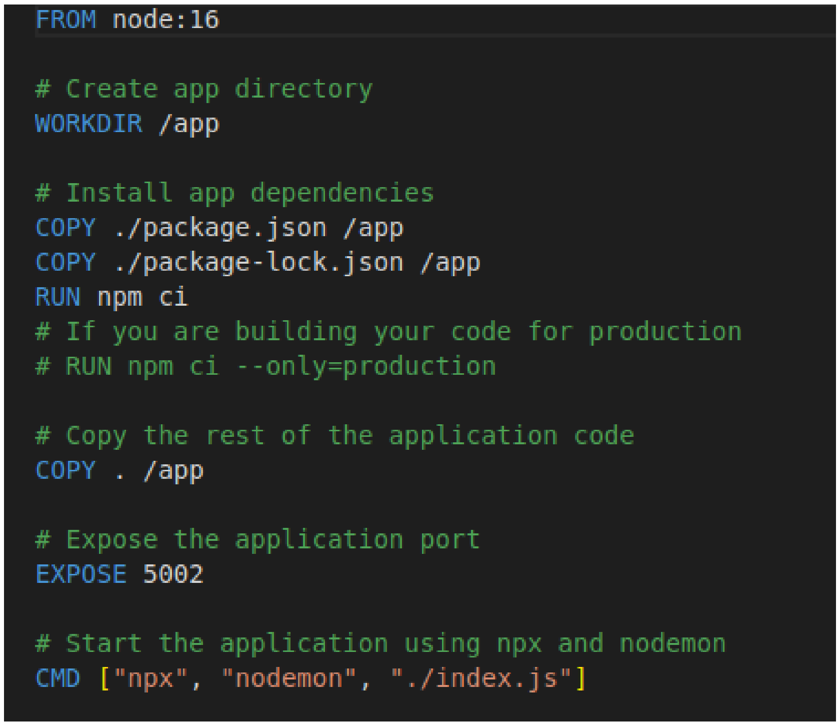
2. Features:

- **Register:** New users can register using this functionality.
- **Login:** Registered users can login using the credentials set while registering.
- **Cart:** The order can be added into the cart.
- **MyOrders:** Maintain your previous orders.

- **Checkout:** Shows the total cost of the items.

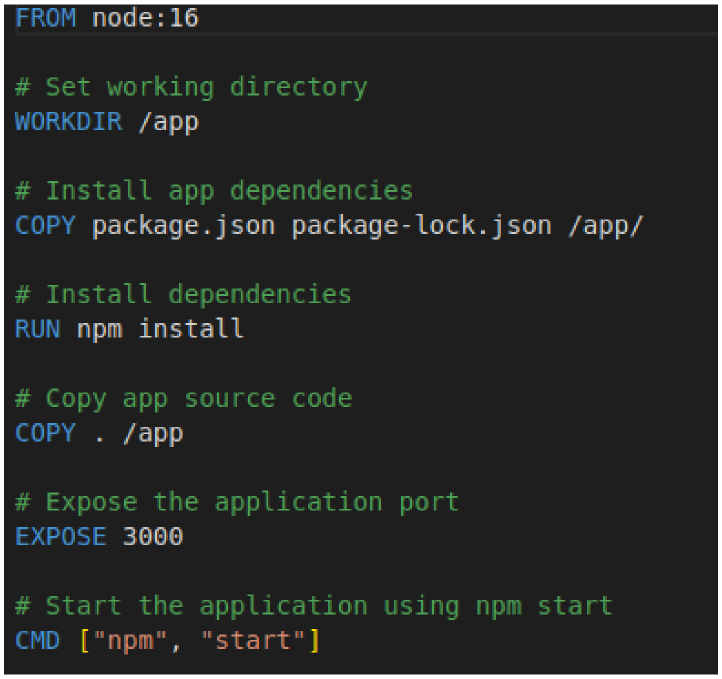
3. Docker:

Backend Docker file:



```
FROM node:16

# Create app directory
WORKDIR /app

# Install app dependencies
COPY ./package.json /app
COPY ./package-lock.json /app
RUN npm ci
# If you are building your code for production
# RUN npm ci --only=production

# Copy the rest of the application code
COPY . /app

# Expose the application port
EXPOSE 5002

# Start the application using npx and nodemon
CMD ["npx", "nodemon", "./index.js"]
```

Frontend docker file:



```
FROM node:16

# Set working directory
WORKDIR /app

# Install app dependencies
COPY package.json package-lock.json /app/

# Install dependencies
RUN npm install

# Copy app source code
COPY . /app

# Expose the application port
EXPOSE 3000

# Start the application using npm start
CMD ["npm", "start"]
```

These two Dockerfiles define the setup for containerizing a frontend and backend application, both built using Node.js.

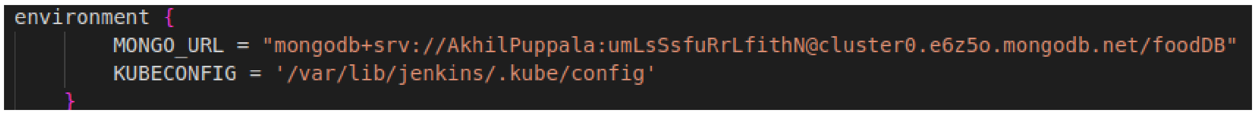
1. Frontend Dockerfile:
    - Uses the Node.js 16 image as the base to ensure compatibility with the project's dependencies.
    - Sets `/app` as the working directory within the container.
    - Copies `package.json` and `package-lock.json` to the container and installs dependencies with `npm install`.
    - Copies the rest of the application's source code into the container.
    - Exposes port 3000, which is the default port for most React applications.
    - Defines the container's startup command as `npm start` to launch the frontend application.
2. Backend Dockerfile:
    - Also uses Node.js 16 as the base image and sets `/app` as the working directory.

- Copies `package.json` and `package-lock.json` into the container, but uses `npm ci` for installing dependencies, which is optimized for reproducibility and CI/CD workflows.
- Copies the backend application code into the container.
- Exposes port 5002, which the backend application listens on.
- Uses `npx nodemon` as the startup command to automatically restart the server during development when file changes are detected.

Together, these Dockerfiles containerize the frontend and backend services, enabling them to run consistently across environments and facilitating integration into development or production workflows.

4. Jenkins:

1) Environment:



```
environment {
        MONGO_URL = "mongodb+srv://AkhilPuppala:umLsSsfuRrLfithN@cluster0.e6z5o.mongodb.net/foodDB"
        KUBECONFIG = '/var/lib/jenkins/.kube/config'
    }
```

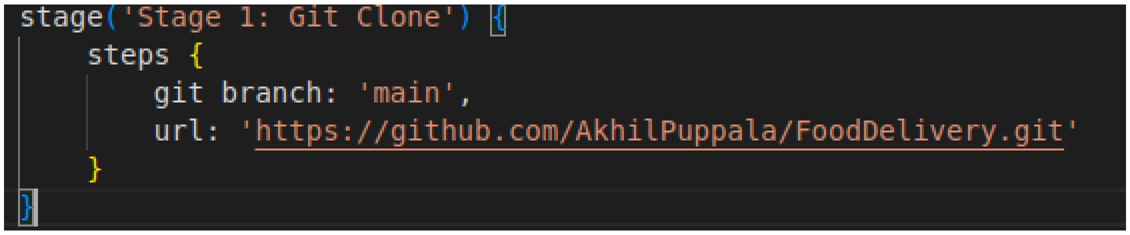
`MONGO_URL`:

A MongoDB connection string pointing to the `foodDB` database hosted on a MongoDB Atlas cluster. This variable is made available to the application during the deployment.

`KUBECONFIG`:

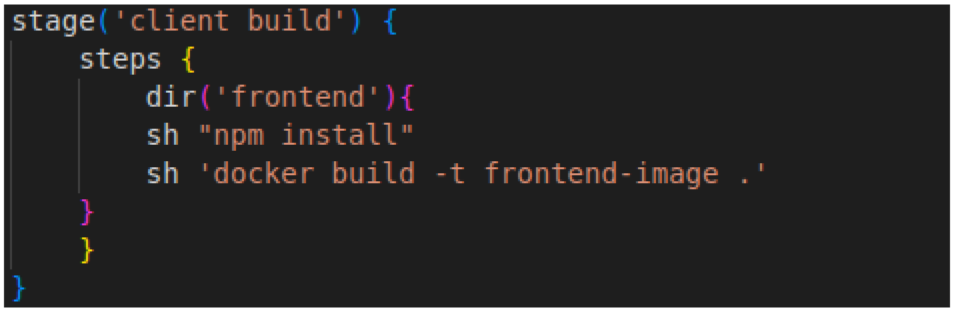
The path to the Kubernetes configuration file used by Jenkins to interact with the Kubernetes cluster. This is typically required for deploying the application to Kubernetes.

2) Stage-1: Git Clone


```
stage('Stage 1: Git Clone') {
    steps {
        git branch: 'main',
        url: 'https://github.com/AkhilPuppala/FoodDelivery.git'
    }
}
```

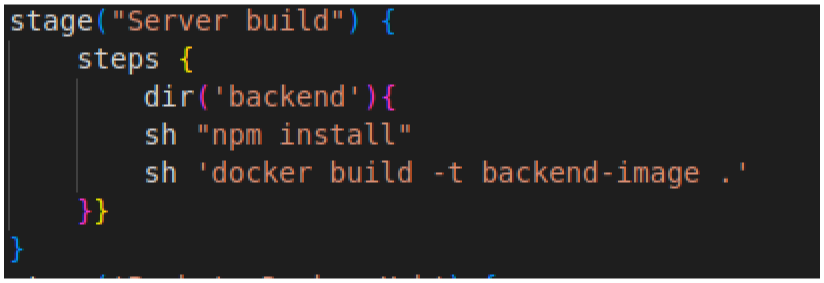
- This stage clones the application's source code from the GitHub repository.
- The `git` step pulls the code from the `main` branch of the specified repository (`https://github.com/AkhilPuppala/FoodDelivery.git`).

3) Stage-2: client build


```
stage('client build') {
    steps {
        dir('frontend'){
        sh "npm install"
        sh 'docker build -t frontend-image .'
    }
    }
}
```

- Navigates to the `frontend` directory.
- Runs `npm install` to install all frontend dependencies.
- Builds a Docker image for the frontend application named `frontend-image`.

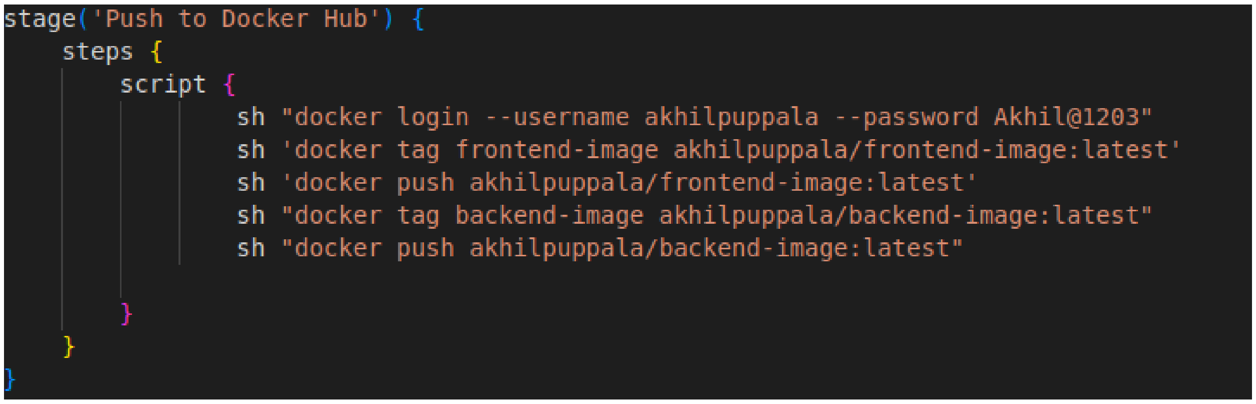
4) Stage-3: server build



```
stage("Server build") {
    steps {
        dir('backend'){
        sh "npm install"
        sh 'docker build -t backend-image .'
    }}
}
```

- Navigates to the `backend` directory.
- Runs `npm install` to install all backend dependencies.
- Builds a Docker image for the backend application named `backend-image`.

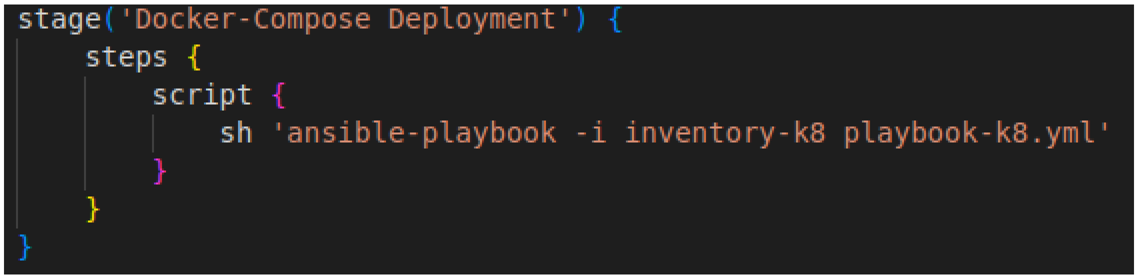
5) Stage-4: Push to docker hub



```
stage('Push to Docker Hub') {
    steps {
        script {
                sh "docker login --username akhilpuppala --password Akhil@1203"
                sh 'docker tag frontend-image akhilpuppala/frontend-image:latest'
                sh 'docker push akhilpuppala/frontend-image:latest'
                sh "docker tag backend-image akhilpuppala/backend-image:latest"
                sh "docker push akhilpuppala/backend-image:latest"

        }
    }
}
```

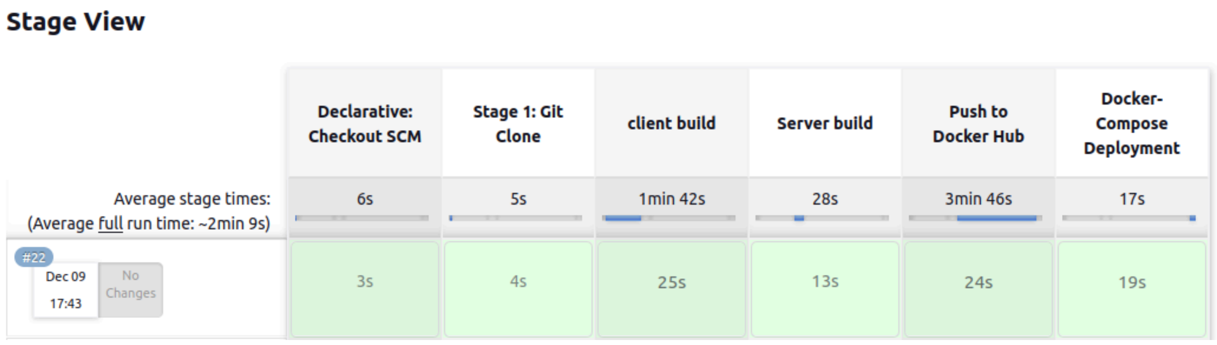
- Logs in to Docker Hub using the provided credentials
- Tags the frontend and backend Docker images with `latest` and assigns them to the `akhilpuppala` Docker Hub account.
- Pushes the tagged images to Docker Hub.

6) Stage-5: Deployment


```
stage('Docker-Compose Deployment') {
    steps {
        script {
            sh 'ansible-playbook -i inventory-k8 playbook-k8.yml'
        }
    }
}
```

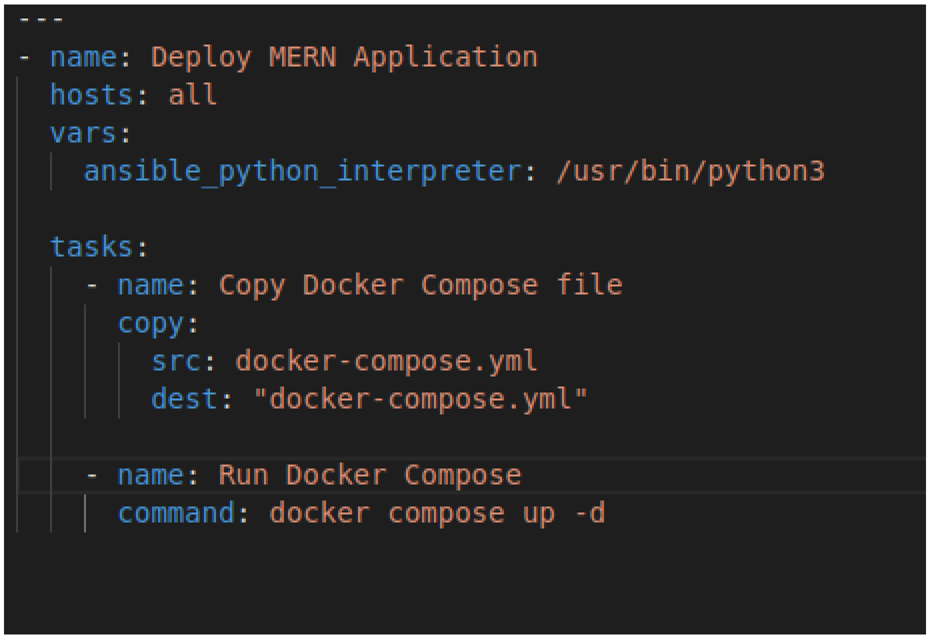
- Executes an Ansible playbook (`playbook-k8.yml`) using the inventory file `inventory-k8`.
- This playbook likely applies the Kubernetes manifests or a `docker-compose` setup for deploying the frontend and backend services.

5) Jenkins pipeline:


Stage View

	Declarative: Checkout SCM	Stage 1: Git Clone	client build	Server build	Push to Docker Hub	Docker-Compose Deployment
Average stage times: (Average full run time: ~2min 9s)	6s	5s	1min 42s	28s	3min 46s	17s
#22 Dec 09 17:43 No Changes	3s	4s	25s	13s	24s	19s

Playbook.yml has the permissions to the docker-compose and start docker images.



```yaml
---
- name: Deploy MERN Application
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3

  tasks:
    - name: Copy Docker Compose file
      copy:
        src: docker-compose.yml
        dest: "docker-compose.yml"

    - name: Run Docker Compose
      command: docker compose up -d
```

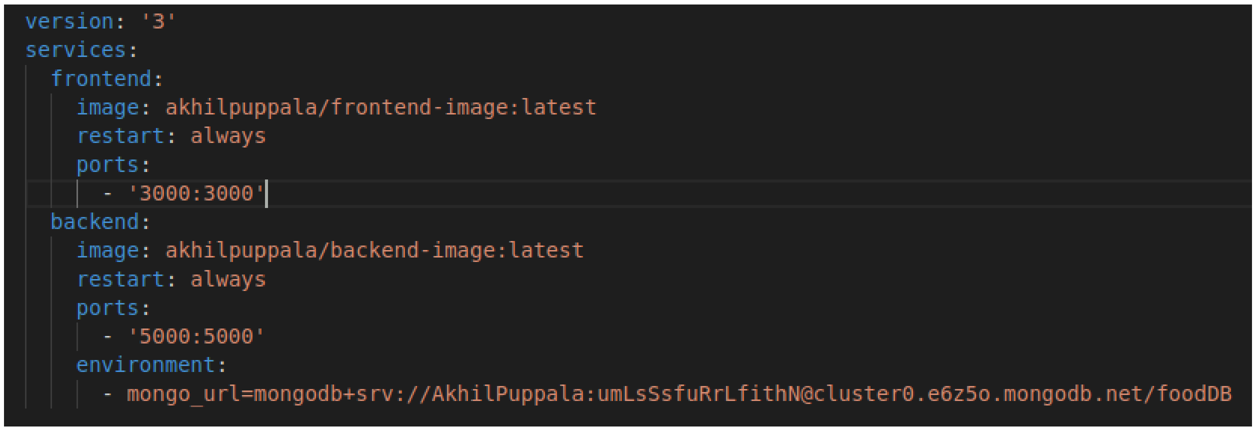
6) Deployments:

<u>Local host:</u>

- **Front end:** npm start
- **Back end:** npm nodemon ./index.js
- **URL:** http://localhost:3000/

7)Docker-compose:

<u>Docker-compose.yaml:</u>



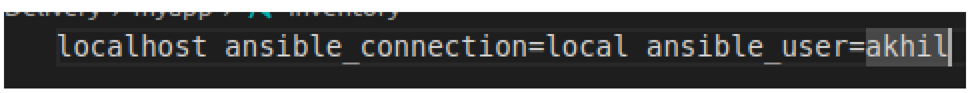
```yaml
version: '3'
services:
  frontend:
    image: akhilpuppala/frontend-image:latest
    restart: always
    ports:
      - '3000:3000'
  backend:
    image: akhilpuppala/backend-image:latest
    restart: always
    ports:
      - '5000:5000'
    environment:
      - mongo_url=mongodb+srv://AkhilPuppala:umLsSsfuRrLfithN@cluster0.e6z5o.mongodb.net/foodDB
```

This **Docker Compose file** defines two services:

1. **Frontend**:
    - Runs `akhilpuppala/frontend-image:latest`.
    - Exposes port `3000` to `http://localhost:3000`.
2. **Backend**:
    - Runs `akhilpuppala/backend-image:latest`.
    - Exposes port `5000` to `http://localhost:5000`.
    - Uses the `mongo_url` environment variable to connect to MongoDB.

Both services restart automatically if they fail.

Inventory:



```
localhost ansible_connection=local ansible_user=akhil
```

`localhost`:

- Refers to the local machine (where the playbook will run).

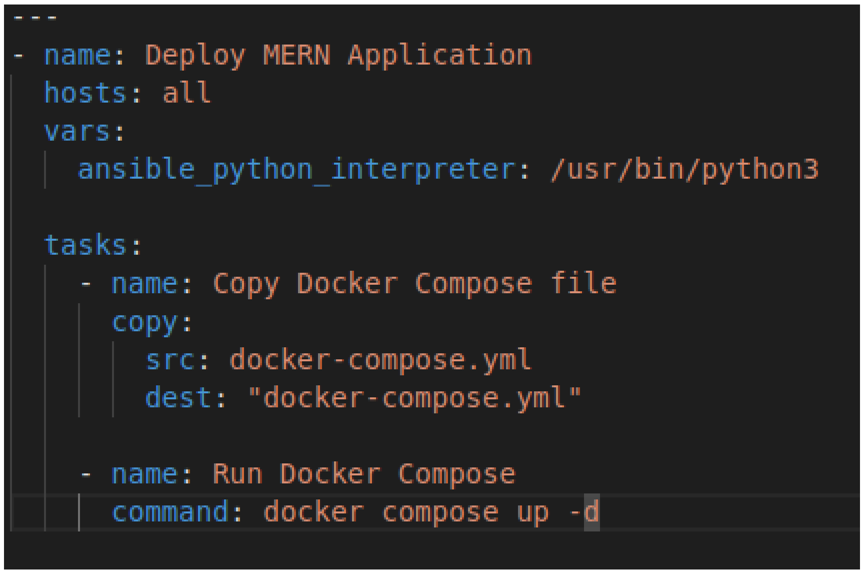
`ansible_connection=local`:

- Specifies that the connection type is **local**, meaning Ansible will execute tasks directly on the local system without requiring SSH.

`ansible_user=akhil`:

- Indicates that the user `akhil` will be used for running tasks.

Playbook.yaml:



```yaml
---
- name: Deploy MERN Application
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3

  tasks:
    - name: Copy Docker Compose file
      copy:
        src: docker-compose.yml
        dest: "docker-compose.yml"

    - name: Run Docker Compose
      command: docker compose up -d
```

Copy Docker Compose File:

- **Module**: `copy`
- **Action**:
  - Copies the `docker-compose.yml` file from the local source (`src`) to the destination directory on the target machine.
- **Result**: Ensures the necessary configuration for Docker Compose is available on the target machine.

Run Docker Compose:

- **Module**: `command`
- **Action**:

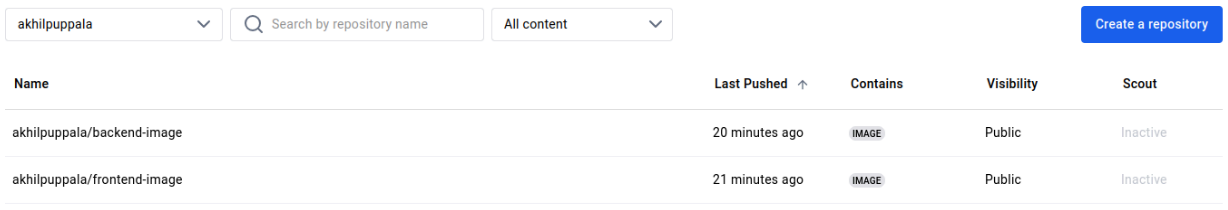
    - Runs `docker compose up -d` to start the application in detached mode (`-d`), launching the frontend and backend containers defined in the `docker-compose.yml`.
- **Result**: Deploys the MERN application services in the background.

<u>Flow when 'docker compose up' is used:</u>

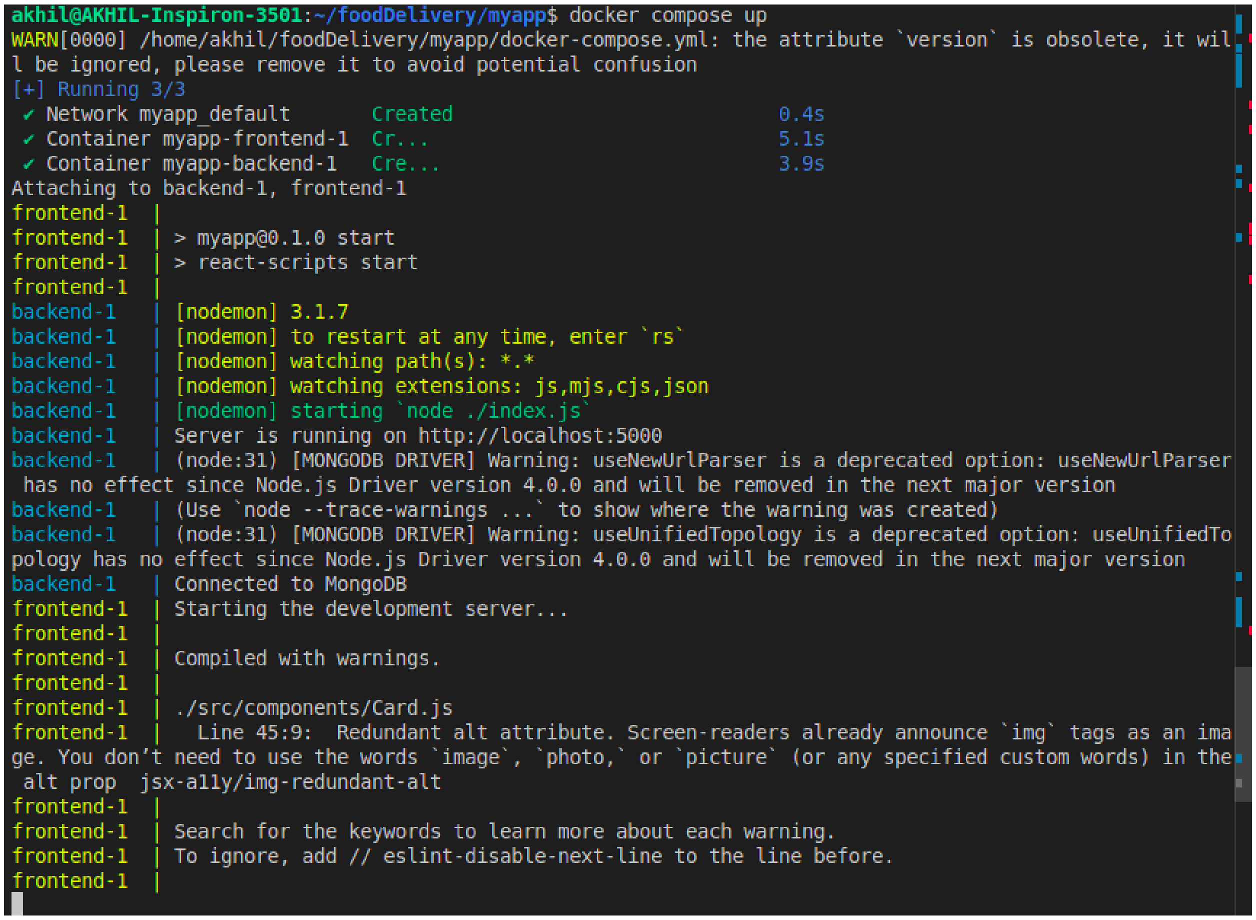
1. Ansible connects to the specified hosts (from the inventory file).
2. Copies the `docker-compose.yml` file to the target system(s).
3. Executes the `docker compose up` command to deploy the MERN application containers.

This playbook simplifies deployment by automating the setup and launch of Docker containers.

Docker containers:



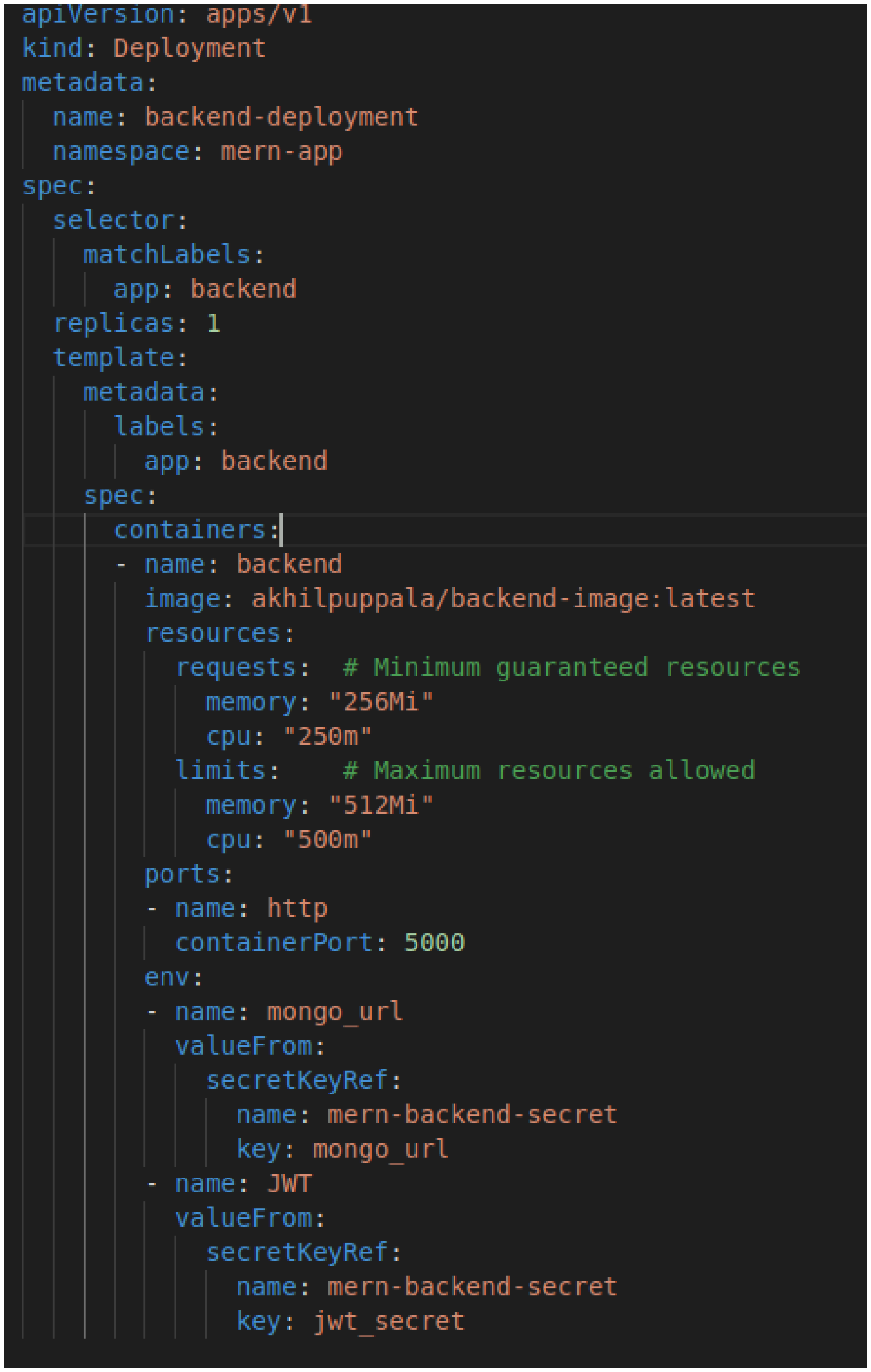
Name	Last Pushed	Contains	Visibility	Scout
akhilpuppala/backend-image	20 minutes ago	IMAGE	Public	Inactive
akhilpuppala/frontend-image	21 minutes ago	IMAGE	Public	Inactive



```
akhil@AKHIL-Inspiron-3501:~/foodDelivery/myapp$ docker compose up
WARN[0000] /home/akhil/foodDelivery/myapp/docker-compose.yml: the attribute `version` is obsolete, it wil
l be ignored, please remove it to avoid potential confusion
[+] Running 3/3
 ✔ Network myapp_default       Created                             0.4s
 ✔ Container myapp-frontend-1  Cr...                               5.1s
 ✔ Container myapp-backend-1   Cre...                              3.9s
Attaching to backend-1, frontend-1
frontend-1  |
frontend-1  | > myapp@0.1.0 start
frontend-1  | > react-scripts start
frontend-1  |
backend-1   | [nodemon] 3.1.7
backend-1   | [nodemon] to restart at any time, enter `rs`
backend-1   | [nodemon] watching path(s): *.*
backend-1   | [nodemon] watching extensions: js,mjs,cjs,json
backend-1   | [nodemon] starting `node ./index.js`
backend-1   | Server is running on http://localhost:5000
backend-1   | (node:31) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser
 has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
backend-1   | (Use `node --trace-warnings ...` to show where the warning was created)
backend-1   | (node:31) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTo
pology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
backend-1   | Connected to MongoDB
frontend-1  | Starting the development server...
frontend-1  |
frontend-1  | Compiled with warnings.
frontend-1  |
frontend-1  | ./src/components/Card.js
frontend-1  |   Line 45:9:  Redundant alt attribute. Screen-readers already announce `img` tags as an ima
ge. You don't need to use the words `image`, `photo,` or `picture` (or any specified custom words) in the
 alt prop  jsx-a11y/img-redundant-alt
frontend-1  |
frontend-1  | Search for the keywords to learn more about each warning.
frontend-1  | To ignore, add // eslint-disable-next-line to the line before.
frontend-1  |
```

8) Kubernetes:

Backend-deplyment.yaml:

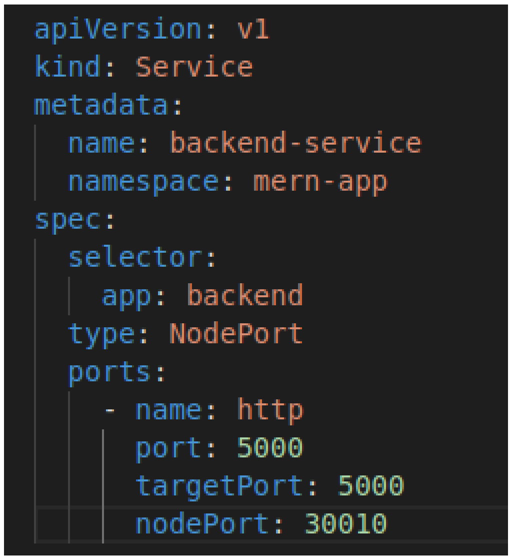


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  namespace: mern-app
spec:
  selector:
    matchLabels:
      app: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: akhilpuppala/backend-image:latest
        resources:
          requests:  # Minimum guaranteed resources
            memory: "256Mi"
            cpu: "250m"
          limits:    # Maximum resources allowed
            memory: "512Mi"
            cpu: "500m"
        ports:
        - name: http
          containerPort: 5000
        env:
        - name: mongo_url
          valueFrom:
            secretKeyRef:
              name: mern-backend-secret
              key: mongo_url
        - name: JWT
          valueFrom:
            secretKeyRef:
              name: mern-backend-secret
              key: jwt_secret
```

This Kubernetes **Deployment** runs a backend service in the `mern-app` namespace with:

- **1 replica** of a Pod using the image `akhilpuppala/backend-image:latest`.
- **Resource limits**: 256Mi/250m (min) and 512Mi/500m (max).
- Exposes **port 5000**.
- Injects sensitive data (`mongo_url` and `JWT`) from a Kubernetes Secret (`mern-backend-secret`).

Backend-service.yaml:



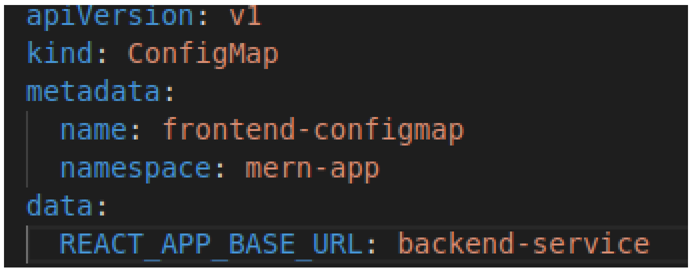
```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: mern-app
spec:
  selector:
    app: backend
  type: NodePort
  ports:
    - name: http
      port: 5000
      targetPort: 5000
      nodePort: 30010
```

This Kubernetes Service exposes the backend application to external traffic. Here's a brief explanation:

- **`apiVersion: v1` & `kind: Service`: Defines a Service resource.**
- **`metadata.name: backend-service`: Names the Service as `backend-service`.**
- **`metadata.namespace: mern-app`: Places the Service in the `mern-app` namespace.**
- **`spec.selector: app: backend`: Targets Pods labeled `app: backend`.**

- `type: NodePort`: Exposes the Service on a specific port of the cluster nodes.
- `ports`:
  - `port: 5000`: The Service's port for external communication.
  - `targetPort: 5000`: Forwards traffic to container port 5000.
  - `nodePort: 30010`: Allocates port 30010 on cluster nodes for external access.

<u>Frontend-config.yaml:</u>

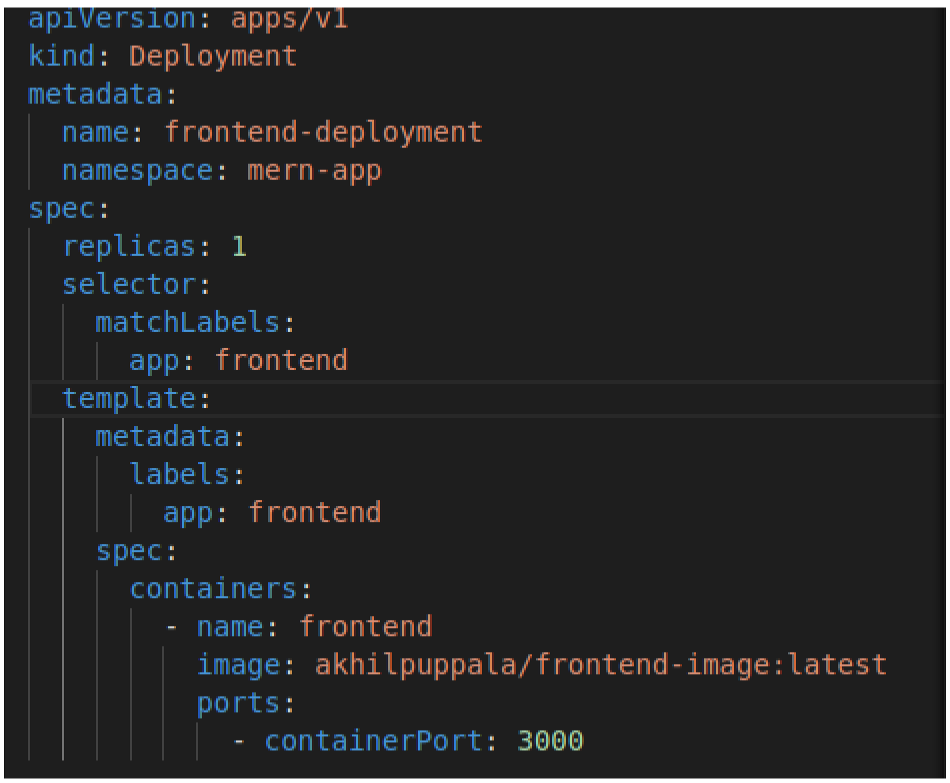


```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-configmap
  namespace: mern-app
data:
  REACT_APP_BASE_URL: backend-service
```

This Kubernetes ConfigMap provides configuration data for the frontend application. Here's a brief explanation:

- `apiVersion: v1` & `kind: ConfigMap`: Defines a ConfigMap resource.
- `metadata.name: frontend-configmap`: Names the ConfigMap as `frontend-configmap`.
- `metadata.namespace: mern-app`: Places the ConfigMap in the `mern-app` namespace.
- `data.REACT_APP_BASE_URL: backend-service`: Sets the environment variable `REACT_APP_BASE_URL` to `backend-service`, allowing the frontend to connect to the backend through this service name.

<u>Frontend-deployment.yaml:</u>



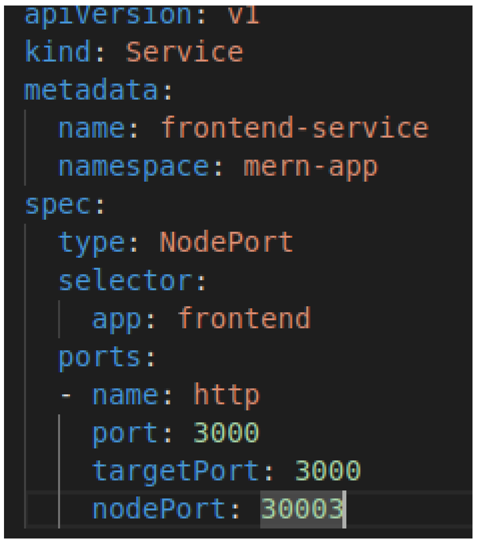
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  namespace: mern-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: akhilpuppala/frontend-image:latest
          ports:
            - containerPort: 3000
```

This Kubernetes **Deployment** runs the frontend service of the MERN application. Here's a brief explanation:

- **`apiVersion: apps/v1`** & **`kind: Deployment`**: Defines a Deployment resource.
- **`metadata.name: frontend-deployment`**: Names the Deployment as `frontend-deployment`.
- **`metadata.namespace: mern-app`**: Places the Deployment in the `mern-app` namespace.
- **`spec.replicas: 1`**: Runs one replica of the frontend Pod.
- **`spec.selector.matchLabels: app: frontend`**: Targets Pods labeled `app: frontend`.
- **`template.metadata.labels: app: frontend`**: Labels the Pods created by this Deployment.
- **`spec.containers`**:

  - **`name: frontend`**: Names the container `frontend`.
  - **`image: akhilpuppala/frontend-image:latest`**: Uses this Docker image for the container.
  - **`ports.containerPort: 3000`**: Exposes port 3000 inside the container for the frontend application.

Frontend-service.yaml:



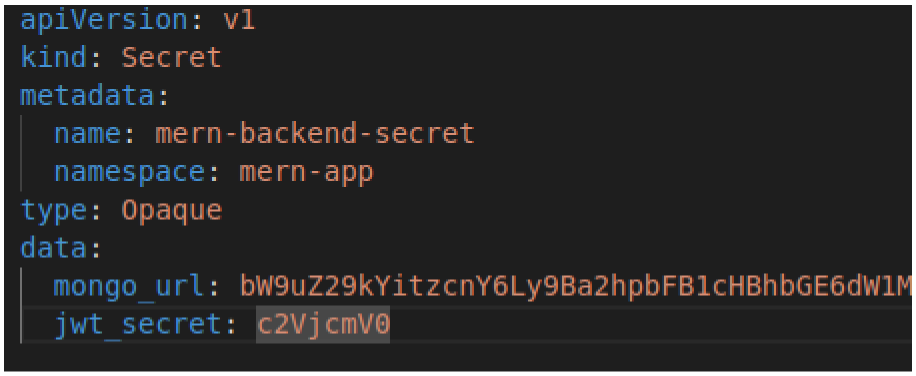
```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: mern-app
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
  - name: http
    port: 3000
    targetPort: 3000
    nodePort: 30003
```

This Kubernetes **Service** exposes the frontend application to external traffic. Here's a brief explanation:

- **`apiVersion: v1`** & **`kind: Service`**: Defines a Service resource.
- **`metadata.name: frontend-service`**: Names the Service as `frontend-service`.
- **`metadata.namespace: mern-app`**: Places the Service in the `mern-app` namespace.
- **`spec.type: NodePort`**: Exposes the Service on a specific port of the cluster nodes.
- **`spec.selector: app: frontend`**: Targets Pods labeled `app: frontend`.

- `ports`:
  - `port: 3000`: The internal Service port for communication within the cluster.
  - `targetPort: 3000`: Forwards traffic to the container’s port 3000.
  - `nodePort: 30003`: Exposes the Service on port 30003 on all cluster nodes for external access.

Mern-backend-secret.yaml:



```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mern-backend-secret
  namespace: mern-app
type: Opaque
data:
  mongo_url: bW9uZ29kYitzcnY6Ly9Ba2hpbFB1cHBhbGE6dW1M
  jwt_secret: c2VjcmV0
```

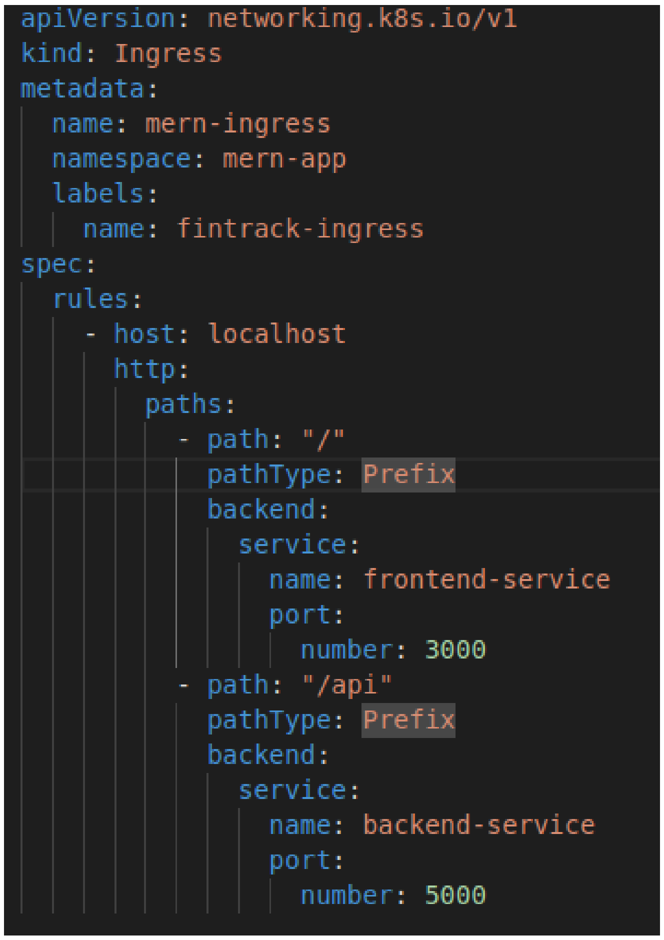
This Kubernetes **Secret** stores sensitive data such as MongoDB connection details and JWT secret. Here's a brief explanation:

- `apiVersion: v1` & `kind: Secret`: Defines a Secret resource.
- `metadata.name: mern-backend-secret`: Names the Secret as `mern-backend-secret`.
- `metadata.namespace: mern-app`: Places the Secret in the `mern-app` namespace.
- `type: Opaque`: Indicates the Secret holds arbitrary data (non-specific to a certain type).
- `data`:
  - `mongo_url`: The MongoDB connection string, encoded in Base64 (`bW9u...` is the encoded version of the actual connection string).

   ◦ **`jwt_secret`**: The JWT secret, also Base64-encoded (`c2VjcmV0` is the Base64 encoded value of "secret").

Kubernetes secrets are used to securely inject sensitive data (like passwords and tokens) into containers. The data in the Secret is encoded in Base64 to ensure it's handled securely.

Mern-ingress.yaml:



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mern-ingress
  namespace: mern-app
  labels:
    name: fintrack-ingress
spec:
  rules:
    - host: localhost
      http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 3000
          - path: "/api"
            pathType: Prefix
            backend:
              service:
                name: backend-service
                port:
                  number: 5000
```

This Kubernetes **Ingress** resource manages external access to services in the `mern-app` namespace. Here's a brief explanation:

- **`apiVersion: networking.k8s.io/v1`** & **`kind: Ingress`**: Defines an Ingress resource, used to manage HTTP and HTTPS routes to services.

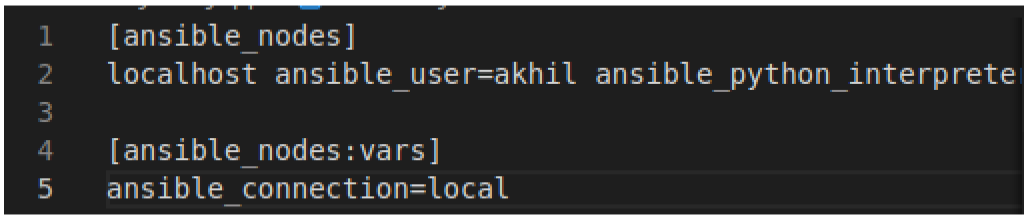
- **`metadata.name: mern-ingress`**: Names the Ingress resource `mern-ingress`.
- **`metadata.namespace: mern-app`**: Places the Ingress in the `mern-app` namespace.
- **`metadata.labels.name: fintrack-ingress`**: Labels the Ingress with `fintrack-ingress`.

`spec.rules` Section:

Defines routing rules for incoming HTTP requests.

- **`host: localhost`**: The Ingress will listen for requests sent to `localhost`.
- **`http.paths`**: Specifies path-based routing:
  - **`path: "/"`**: Routes requests to the frontend service when the URL path starts with `/`.
    - **`backend.service.name: frontend-service`**: Directs traffic to the `frontend-service`.
    - **`port.number: 3000`**: Forwards traffic to port `3000` on the `frontend-service`.
  - **`path: "/api"`**: Routes requests to the backend service when the URL path starts with `/api`.
    - **`backend.service.name: backend-service`**: Directs traffic to the `backend-service`.
    - **`port.number: 5000`**: Forwards traffic to port `5000` on the `backend-service`.

<u>Inventory-k8:</u>



```
[ansible_nodes]
localhost ansible_user=akhil ansible_python_interprete

[ansible_nodes:vars]
ansible_connection=local
```

This is an **Ansible inventory file** that defines a group of nodes called `ansible_nodes` with connection settings and variables for running playbooks. Here's a brief explanation:

Inventory Groups

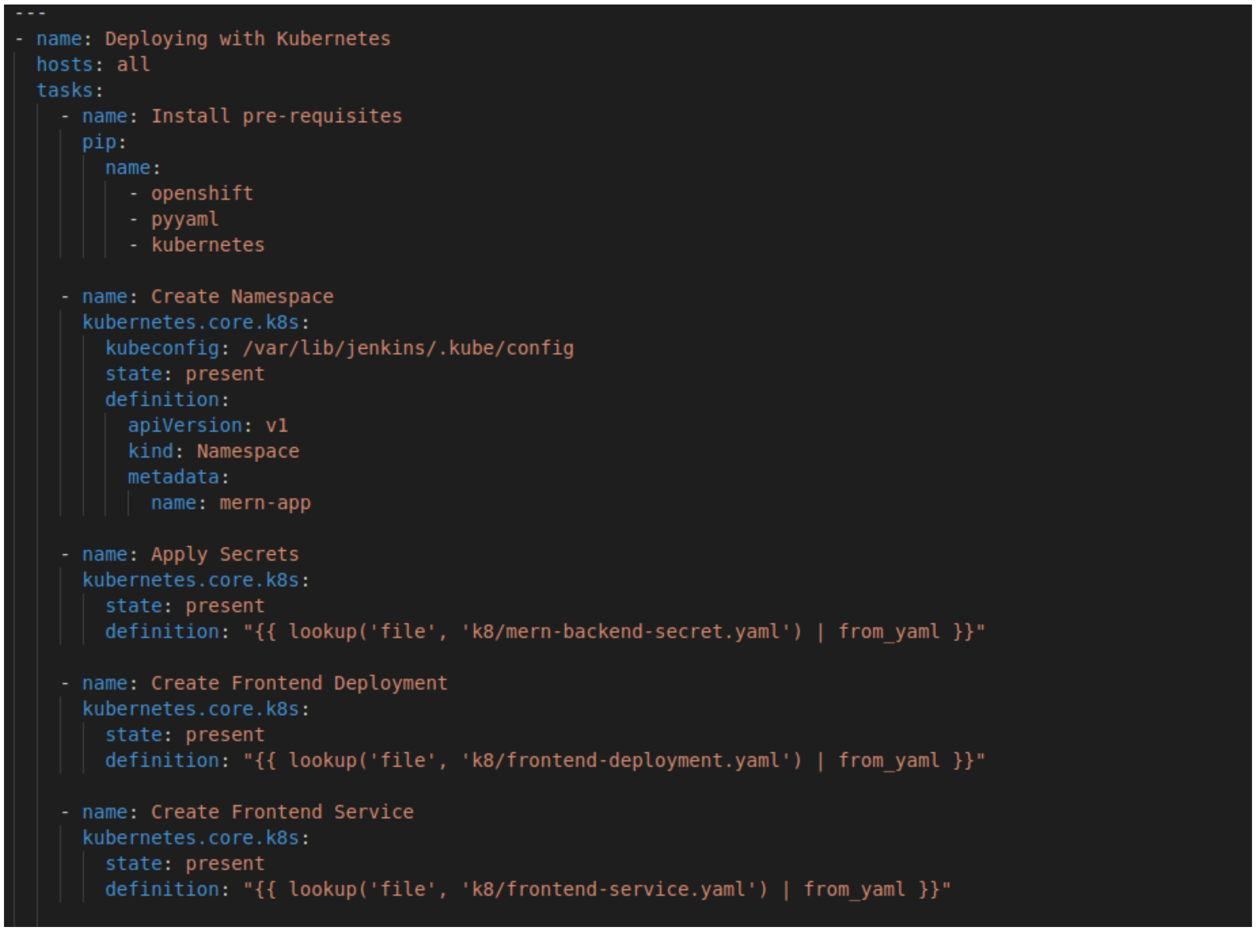
- **`[ansible_nodes]`**: A group named `ansible_nodes` containing the following node:
  - **`localhost`**: Specifies that Ansible will manage the local machine.
  - **`ansible_user=akhil`**: Specifies that the user to log in as is `akhil`.
  - **`ansible_python_interpreter=/usr/bin/python3`**: Indicates the Python interpreter to use on the target node (Python 3 in this case).

Group Variables

- **`[ansible_nodes:vars]`**: Variables that apply to all nodes in the `ansible_nodes` group.
  - **`ansible_connection=local`**: Specifies that the connection is local, meaning no SSH is used since it's managing the local machine.

This configuration is useful for testing or running Ansible on the same machine without needing SSH.

Playbook-k8.yaml:



```yaml
---
- name: Deploying with Kubernetes
  hosts: all
  tasks:
    - name: Install pre-requisites
      pip:
        name:
          - openshift
          - pyyaml
          - kubernetes

    - name: Create Namespace
      kubernetes.core.k8s:
        kubeconfig: /var/lib/jenkins/.kube/config
        state: present
        definition:
          apiVersion: v1
          kind: Namespace
          metadata:
            name: mern-app

    - name: Apply Secrets
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', 'k8/mern-backend-secret.yaml') | from_yaml }}"

    - name: Create Frontend Deployment
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', 'k8/frontend-deployment.yaml') | from_yaml }}"

    - name: Create Frontend Service
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', 'k8/frontend-service.yaml') | from_yaml }}"
```

This is an **Ansible playbook** for deploying a Kubernetes-based MERN application. Here's a very brief breakdown:

Purpose:

The playbook automates the deployment of a MERN stack application to a Kubernetes cluster.

Key Sections:

1. **Install Pre-requisites**:
    - Installs Python packages (`openshift`, `pyyaml`, `kubernetes`) for interacting with Kubernetes.
2. **Create Namespace**:

    - Ensures the Kubernetes namespace `mern-app` exists.
3. **Apply Kubernetes Resources**:
    - **Secrets, ConfigMap, Deployments, Services, and Ingress** are created by loading YAML definitions from corresponding files (e.g., `mern-backend-secret.yaml`, `frontend-deployment.yaml`).

Highlights:

- **`kubernetes.core.k8s` Module**: Manages Kubernetes resources using `kubeconfig` for authentication.
- **File Lookups**: Reads YAML files for resource definitions using `lookup('file')` and `from_yaml`.

This playbook ensures your Kubernetes resources are consistently created or updated without manual intervention.

Flow of running the command: ansible-playbook -i inventory-k8 playbook-k8.yml

The following flow occurs:

---

1. Inventory Setup

- **File**: `inventory-k8`
  Ansible uses the inventory file to identify the target host(s). In this case:
    - `localhost` is the target.
    - The connection is local (`ansible_connection=local`).
    - The user is `akhil`.
    - Python 3 is used as the interpreter.

---

2. Playbook Execution

- **File**: `playbook-k8.yml`
  The playbook defines the tasks to execute on the `localhost`. The flow proceeds as follows:

Step 1: Install Pre-requisites

- The `pip` module ensures required Python libraries (`openshift`, `pyyaml`, `kubernetes`) are installed on the local system to interact with Kubernetes.

Step 2: Create Kubernetes Namespace

- The `kubernetes.core.k8s` module ensures the `mern-app` namespace is present in the cluster.
- Uses the kubeconfig file located at `/var/lib/jenkins/.kube/config`.

Step 3: Apply Secrets

- Reads the `k8/mern-backend-secret.yaml` file using `lookup('file')` and converts its contents to YAML with `from_yaml`.
- Creates or updates the Secret (`mern-backend-secret`) in the `mern-app` namespace.

Step 4: Create Frontend Deployment

- Reads the `k8/frontend-deployment.yaml` file and applies its configuration.
- Ensures the frontend application pods are deployed in the cluster.

Step 5: Create Frontend Service

- Reads the `k8/frontend-service.yaml` file and applies its configuration.
- Exposes the frontend pods to external traffic on NodePort `30003`.

Step 6: Create Backend Deployment

- Reads the `k8/backend-deployment.yaml` file and applies its configuration.
- Ensures the backend application pods are deployed.

Step 7: Create Backend Service

- Reads the `k8/backend-service.yaml` file and applies its configuration.
- Exposes the backend pods on NodePort `30010`.

Step 8: Create ConfigMap

- Reads the `k8/frontend-configmap.yaml` file and applies its configuration.
- Configures the frontend with the backend service URL (`REACT_APP_BASE_URL`).

Step 9: Create Ingress

- Reads the `k8/mern-ingress.yaml` file and applies its configuration.
- Sets up an Ingress controller to route:
  - `/` requests to the frontend service.
  - `/api` requests to the backend service.

---

3. Kubernetes Cluster Flow

After all tasks are executed:

1. The namespace `mern-app` is created to logically group the application's resources.
2. Secrets, ConfigMap, Deployments, Services, and Ingress resources are applied in the cluster:
    - **Frontend** and **backend pods** are deployed.
    - Services expose the pods via NodePorts.
    - The Ingress routes traffic to the correct service based on the URL path.

Outcome

- The MERN stack application is fully deployed in the `mern-app` namespace.
- You can access:
    - The frontend at `http://<node-ip>:30003` or via the Ingress (`http://localhost/`).
    - The backend API at `http://<node-ip>:30010` or via the Ingress (`http://localhost/api`).