
Task: Deploying a Node.js Application Using Nginx as a Reverse Proxy

Objective:

The goal is to set up a multi-server environment for building, storing, and deploying a **Node.js application** with **Nginx** as the reverse proxy server to manage traffic to the Node.js application. This setup will involve three servers:

- **Node.js Build Server**: The server where the code is built and packaged.
- **Nexus Repository Server**: The server where the application artifacts are stored.
- **Deploy Server**: The server where the application will be deployed and run with Nginx managing the traffic.

Prerequisites:

- **Node.js Application**: Students will use an existing Node.js application repository for this task. Example: Node.js Web Application.
- **Three Servers**:
    - **Build Server** to build and package the Node.js app.
    - **Nexus Repository Server** to store artifacts.
    - **Deploy Server** with Nginx, where the application will be deployed and run.

---

1. Node.js Build Server Setup

Task Overview:

The **Build Server** is responsible for cloning the Node.js application repository, installing the dependencies, building the application, and pushing it to Nexus for storage.

Steps:

1. **Clone the Application Repository**: Clone the Node.js application from the provided GitHub repository to the Build Server.

2. **Install Dependencies**: Use `npm` to install all the dependencies from the `package.json` file.

3. **Build the Application**: Ensure that the application is properly built. This may involve running unit tests, packaging the app, or bundling assets (e.g., if using Webpack or other build tools).

4. **Publish to Nexus**:

    - Configure npm to use Nexus for publishing (update `.npmrc` if necessary).

    - Publish the application or Docker image to Nexus (if you're using Docker for deployment).

Deliverables:

- Successfully cloned and installed application dependencies.

- Artifacts (npm package or Docker image) uploaded to Nexus.

---

2. Nexus Repository Server Setup

Task Overview:

The **Nexus Repository Server** will store the packaged application artifacts (npm packages or Docker images). This server will be responsible for storing and making the artifacts available for deployment.

Steps:

1. **Install Nexus Repository Manager**: Install Nexus Repository Manager on the server.

2. **Create npm or Docker Repositories**:

    - Create a repository in Nexus for storing npm packages (or Docker images, if applicable).

3. **Configure Authentication**: Ensure that the Build and Deploy servers can authenticate with Nexus to fetch the artifacts.

4. **Verify Artifact Storage**: Verify that the application artifacts (npm package or Docker image) were successfully uploaded to Nexus and are available for the Deploy Server to download.

Deliverables:

- Nexus server up and running with proper configuration for npm and/or Docker repositories.
- Authentication configured for Build and Deploy Servers.
- Artifacts successfully uploaded to Nexus.

---

3. Deploy Server Setup (with Nginx as Reverse Proxy)

Task Overview:

The **Deploy Server** will be responsible for hosting the Node.js application and serving it to the internet via Nginx. The application will be run in the background, and Nginx will route traffic to the Node.js app.

Steps:

1. **Install Node.js and Nginx**:
    - **Node.js**: Install Node.js and npm on the deploy server if it's not already installed.
    - **Nginx**: Install Nginx to act as the reverse proxy for the Node.js application.
2. **Configure Nginx**:
    - Set up Nginx to reverse proxy HTTP requests to the Node.js application running on a specific port (e.g., port 3000).
    - Create an Nginx configuration file for the Node.js app. Here’s an example configuration:

Example Nginx configuration (`/etc/nginx/sites-available/nodeapp`):

```
server {
    listen 80;
    server_name your_domain_or_ip;

    location / {
        proxy_pass http://localhost:3000;  # Port where the Node app runs
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

3.
    - Enable the Nginx configuration and restart Nginx:

```
sudo ln -s /etc/nginx/sites-available/nodeapp /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

4.
5. **Fetch the Artifact from Nexus**:
    - On the Deploy Server, use `npm install` to fetch the packaged artifact (npm package) from Nexus. If it's a Docker image, use Docker commands to pull the image.

Example for npm artifact:

```
npm install --registry=http://<NEXUS_URL>/repository/npm-repository/
```

6.
7. **Run the Application**:
    - After fetching the application artifact, deploy it by running the application with `npm start` or by running it inside a Docker container (if the app is containerized).

Example command to start Node.js app:

```
npm start
```

8.
    - If using Docker, run the app within a container.

9. **Verify the Deployment**:
    - Ensure the application is running by accessing it via a browser using the public IP address of the Deploy Server (e.g., `http://<deploy-server-ip>`).
    - The Nginx server will route requests to the Node.js application, and the app should respond with the appropriate content.

Deliverables:

- Nginx properly configured as a reverse proxy.
- Node.js application running behind Nginx and accessible via a browser.
- A working deployment of the Node.js application using Nginx as the reverse proxy.

---

4. End-to-End Testing

Task Overview:

Once the setup is complete, students will test the entire process to ensure the application is properly built, stored in Nexus, and deployed with Nginx.

Steps:

1. **Trigger the Build**: On the Build Server, students will trigger the build process, publish the artifact to Nexus, and ensure it's available for deployment.
2. **Verify Artifact in Nexus**: Log in to Nexus and verify that the artifact is correctly uploaded and stored in the repository.
3. **Deploy the Artifact**: On the Deploy Server, students will pull the artifact from Nexus and deploy it using `npm install` (for npm packages) or `docker pull` (for Docker images).
4. **Access the Application**: Once the application is deployed, students will test that it's accessible from the browser through Nginx.

Deliverables:

- A fully deployed Node.js application running behind Nginx.
- Documentation detailing the process and any issues encountered during the setup.

Conclusion

By the end of this task, students will have learned how to:

- Build, package, and store a Node.js application in Nexus.
- Deploy the application on a server with **Nginx** as a reverse proxy.
- Ensure that the application is running and accessible to end-users.

This exercise simulates a real-world **DevOps** pipeline and helps students understand how to integrate **build**, **repository**, and **deployment** steps in a distributed environment using Nginx.

Assessment Criteria:

- **Server Configuration**: The Build, Nexus, and Deploy servers must be set up correctly and work seamlessly together.
- **Artifact Management**: The artifact must be successfully built, stored in Nexus, and deployed to the Deploy Server.
- **Deployment**: The application must be accessible through Nginx, running correctly.
- **Documentation**: Clear documentation on the setup, challenges faced, and solutions provided.
