1. Prepare Application Code:

- Ensure your React frontend and Django backend codebases are ready and tested locally.

- Confirm that the Milvus database configuration is properly set up.

2. Create Dockerfiles:

- In the root directory of your React and Django projects, create Dockerfiles to define the Docker images.

- Include necessary dependencies and configurations in each Dockerfile.

- Ensure that Django Dockerfile includes commands to collect static files.

3. Configure Docker Compose:

- Create a `docker-compose.yml` file in the root directory.

- Define services for React frontend, Django backend, and Milvus database.

- Configure environment variables, ports, volumes, and dependencies for each service.

- Set up network configurations to ensure communication between services.

4. Build Docker Images:

- Open a terminal and navigate to the directory containing the `docker-compose.yml` file.

- Run `docker-compose build` to build Docker images for all services.

- Verify that the images are built successfully without any errors.

5. Start Docker Containers:

- After building the images, start the Docker containers by running `docker-compose up -d`.

- Ensure that all services start up without any issues.

- Monitor the container logs to check for any errors or warnings.

6. Verify Deployment:

- Access the React frontend by opening a web browser and navigating to the specified port (usually `http://localhost:<frontend\_port>`).

- Test various functionalities of the application to ensure they work as expected.

- Verify that the Django backend is serving API requests correctly.

- Check the Milvus database to ensure it's running and accessible.

7. Troubleshooting:

- If you encounter any issues during deployment, refer to the Docker logs for each container to identify the problem.

- Check the configuration files and Dockerfiles for any errors or misconfigurations.

- Use Docker commands (`docker ps`, `docker logs`, `docker-compose logs`, etc.) to inspect container status and logs.

- Consult documentation and online resources for troubleshooting Docker-related issues.

8. Monitoring and Maintenance:

- Set up monitoring tools to keep track of the health and performance of Docker containers.

- Implement logging and alerting mechanisms to detect and respond to any anomalies.

- Regularly update Docker images and configurations to incorporate security patches and improvements.

- Perform periodic maintenance tasks such as cleaning up unused containers, images, and volumes.

9. Scaling (Optional):

- If needed, scale the application by adjusting the number of container instances for each service in the Docker Compose file.

- Use container orchestration tools like Docker Swarm or Kubernetes for managing larger-scale deployments.

10. Documentation:

- Document the deployment process, including steps for setting up, configuring, and troubleshooting the application.

- Include information about environment variables, ports, dependencies, and any other relevant configurations.

- Provide instructions for maintaining and updating the deployed environment.

Following this runbook should help you deploy your React and Django application along with the Milvus database using Docker Compose efficiently and effectively.