Project: React App — CI/CD, Deployment & Production Operations

Complete, production-ready documentation covering Jenkins pipeline, build & deploy scripts, node/agent setup, monitoring, troubleshooting and operational playbook.

Table of Contents

---

1. Project summary

This project contains a statically built React application served by Nginx inside a Docker container. CI/CD is driven by Jenkins using a declarative pipeline. The pipeline:

- checks out the repo
- builds the Docker image (on a build agent)
- pushes the image to Docker Hub
- deploys the app on target nodes using a `deploy.sh` script and a `docker-compose` file

Production nodes are configured as Jenkins inbound agents (systemd service) that auto-register to the controller. Monitoring is handled by **Uptime Kuma** on the production host and integrated with Slack and an optional Jenkins webhook.

---

2. Repository layout

(visual representation)

```
BUILD-APP/
├─ build/                    # build output or Docker build context
│  ├─ static/
│  ├─ Dockerfile
│  ├─ nginx.conf
│  └─ ...
├─ deploy.sh                 # deployment script run on nodes
├─ docker-compose.yaml       # compose file (root-level) — note: yaml extension
├─ Jenkinsfile               # pipeline definition
├─ jenkins_node_connect.sh   # helper for agent networking
└─ README.md
```

> Repo screenshot (reference): `file:///mnt/data/010daa09-dd6d-4e7f-bca8-7910fc0a93f8.png`

(If your UI converts local paths to URLs, the above path points to the repo tree screenshot.)

---

3. Jenkins pipeline — overview & full file

Stages - `Checkout Code` — detect the branch and stash workspace (`appsource`) for use on other agents. - `Build Docker Image` — runs on `build_agent`, builds Docker image from `build/` context using `-f build/Dockerfile` and tags it. - `Push Docker Image` — logs into Docker Hub (via credentials) and pushes the tagged image. - `Deploy to Environment` — unstashes the workspace on the target node and copies `docker-compose.yaml` and `deploy.sh` into the node home, then runs `deploy.sh` with the image tag.

Key pipeline details & rationale - We use `stash/unstash 'appsource'` to make repo files available across nodes. - Build runs on a labeled build agent (`build_agent`) that has Docker installed. - Deploy runs on the target node(s) labeled `project_1_dev` / `project_1_prod`. - We run the Docker build with `-f build/Dockerfile build/` so the build context is the `build` folder you have in the repo.

Important environment variables - `DEV_IMAGE` and `PROD_IMAGE` are set in pipeline environment block (e.g. `prem18062000/react-app-dev:latest-dev`).

> The canonical Jenkinsfile used in the project is stored directly in the repo as `Jenkinsfile`.

---

4. Build script (`build.sh`) — recommended

Create a small `build.sh` script to extract the build process from Jenkinsfile if you want to call it standalone (keeps pipeline logic unchanged — pipeline will `sh "./build.sh"` if desired).

Purpose: Build Docker image and optionally push it (depending on parameters).

Suggested `build.sh`

```
#!/usr/bin/env bash
set -euo pipefail

# Usage: build.sh <image_tag> [push]
IMAGE_TAG="$1"
PUSH=${2:-false}

echo "Building image: $IMAGE_TAG"
# assumes we're executed from repo root and build/ exists
sudo docker build -t "$IMAGE_TAG" -f build/Dockerfile build/

if [ "$PUSH" = true ]; then
  echo "Pushing image: $IMAGE_TAG"
  sudo docker push "$IMAGE_TAG"
fi

echo "Build finished: $IMAGE_TAG"
```

Make executable: `chmod +x build.sh`.

---

5. Deploy script (`deploy.sh`) — production-ready

This is the `deploy.sh` we used (the one already tested in your runs). It copies the `docker-compose.yaml` into a stable directory on the node, pulls the requested image, and starts containers using either `docker compose` (v2) or `docker-compose` (v1). It uses `IMAGE` environment variable injection so the compose file can reference `${IMAGE}`.

`deploy.sh`

```
#!/bin/bash
set -e

IMAGE="$1"
APP_DIR="$HOME/app"
COMPOSE_FILE="$APP_DIR/docker-compose.yaml"
TIMESTAMP() { date +"%Y-%m-%d %H:%M:%S"; }
log() { echo "[$(TIMESTAMP)] $*"; }

if [ -z "$IMAGE" ]; then
  log "ERROR: No image provided. Usage: ./deploy.sh <IMAGE>"
  exit 1
```

```
fi

log "--------------------------------------------------"
log "Starting Deployment"
log "Image to deploy: $IMAGE"
log "--------------------------------------------------"

# Verify docker
if ! command -v docker >/dev/null 2>&1; then
  log "ERROR: docker is not installed"
  exit 1
fi

# detect compose command
if command -v docker compose >/dev/null 2>&1; then
  COMPOSE_CMD="docker compose"
elif command -v docker-compose >/dev/null 2>&1; then
  COMPOSE_CMD="docker-compose"
else
  log "ERROR: docker compose/docker-compose not found"
  exit 1
fi

mkdir -p "$APP_DIR"
log "Copying docker-compose.yaml to $APP_DIR ..."
cp "$HOME/docker-compose.yaml" "$COMPOSE_FILE"

log "Pulling latest image: $IMAGE ..."
sudo docker pull "$IMAGE"

log "Stopping existing containers..."
$COMPOSE_CMD -f "$COMPOSE_FILE" down || true

log "Starting new containers..."
IMAGE="$IMAGE" $COMPOSE_CMD -f "$COMPOSE_FILE" up -d

log "Waiting for container warmup..."
sleep 3

log "Checking image list"
sudo docker images

log "Checking deployed containers..."
$COMPOSE_CMD -f "$COMPOSE_FILE" ps

log "Checking running containers..."
sudo docker ps
```

```
log "Deployment completed successfully!"
log "--------------------------------------------------"
```

Make executable and commit: `chmod +x deploy.sh`.

---

6. Agent / Node bootstrap (`install.sh`) — Java, Docker, agent service

Purpose: Bootstraps a new EC2 instance so it automatically starts and registers as a Jenkins agent and supports Docker deployments.

Key tasks performed by the script: - Install OpenJDK 17 (for the agent compatibility) - Install Docker engine + compose plugin - Download `agent.jar` from controller and create a `systemd` service to run the agent - Start and enable the `jenkins-agent` service

Safe logging and verification: script prints progress and checks for failures.

Suggested `install.sh`

```
#!/bin/bash
set -euo pipefail

MASTER_IP="<JENKINS_CONTROLLER_IP>"   # replace
JENKINS_URL="http://${MASTER_IP}:8080"
AGENT_SECRET="<AGENT_SECRET_FROM_JENKINS>"
AGENT_NAME="project_1_prod"
WORKDIR="/home/ubuntu/jenkins-agent"
SERVICE_FILE="/etc/systemd/system/jenkins-agent.service"

log() { echo "[\$(date +'%Y-%m-%d %H:%M:%S')] \$*"; }

log "========================================"
log " Installing Java 17 (required for agent)"
log "========================================"
sudo apt update -y
sudo apt install -y openjdk-17-jre-headless
log "Java installed: \$(java -version 2>&1 | head -n1)"

log "========================================"
log " Installing Docker engine & compose plugin"
log "========================================"
# Minimal Docker install steps
sudo apt update -y
sudo apt install -y ca-certificates curl gnupg lsb-release
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/
keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
sudo tee /etc/apt/sources.list.d/docker.sources > /dev/null <<EOF
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: \$(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF
sudo apt update -y
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
log "Docker installed: \$(docker --version 2>&1 | head -n1)"

# Create workspace
sudo mkdir -p "$WORKDIR"
sudo chown -R ubuntu:ubuntu "$WORKDIR"
sudo chmod 755 "$WORKDIR"
cd "$WORKDIR"

log "Downloading agent.jar"
curl -fsSL -o agent.jar "${JENKINS_URL}/jnlpJars/agent.jar"
if [ ! -f agent.jar ]; then
  log "ERROR: agent.jar failed to download"
  exit 1
fi
sudo chown ubuntu:ubuntu agent.jar
sudo chmod 644 agent.jar

log "Creating systemd service"
sudo bash -c "cat > ${SERVICE_FILE}" <<EOF
[Unit]
Description=Jenkins Agent Service
After=network.target

[Service]
ExecStart=/usr/bin/java -jar ${WORKDIR}/agent.jar -url ${JENKINS_URL}/ -secret $
{AGENT_SECRET} -name ${AGENT_NAME} -webSocket -workDir ${WORKDIR}
User=ubuntu
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF
```

```
sudo chmod 644 "${SERVICE_FILE}"
sudo systemctl daemon-reload
sudo systemctl enable jenkins-agent
sudo systemctl restart jenkins-agent

log "Agent service status:"
sudo systemctl status jenkins-agent --no-pager -l || true

log "Bootstrap complete"
```

Notes: - Make sure `AGENT_SECRET` and `MASTER_IP` are set correctly. If you plan to run this script during instance user-data, pass those values via the EC2 user-data template. - The script avoids Java flags like `--enable-native-access` which are only supported by newer Java releases and caused failures on OpenJDK 11 in previously observed logs.

---

7. Dockerfile and docker-compose explanation

Dockerfile (multi-stage) - Builder stage: uses `node:18-alpine` to `npm ci` and `npm run build` when `package.json` exists. - Runner stage: `nginx:stable-alpine` copies the `build/` folder into `/usr/share/nginx/html` and uses `nginx.conf` tuned for SPAs.

Important notes: - Docker build ARG `BUILD_DIR` defaults to `build` so `COPY --from=builder /app/${BUILD_DIR} /usr/share/nginx/html` works regardless of whether you built from project root or the `build/` folder.

docker-compose.yaml You use a minimal single-service compose file that takes `IMAGE` from the environment injected by `deploy.sh`:

```
version: "3.9"
services:
  react-app:
    image: "${IMAGE}"
    container_name: react-app
    restart: always
    ports:
      - "80:80"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:80"]
      interval: 15s
      timeout: 5s
      retries: 5
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "3"
```

Make sure your compose file extension matches the file found on the node. In your runs you used `docker-compose.yaml` (yaml extension). Keep this consistent in the pipeline (copy the same filename you actually commit).

---

8. Monitoring with Uptime Kuma

Quick setup summary - Run Uptime Kuma container on prod host (example binds to port `3001`). - Add an HTTP monitor that checks `http://<prod-ip>` every 30–60s. - Add a Notification channel (Slack / Webhook). - Optionally add a Webhook notification to trigger a Jenkins job when DOWN.

Why Uptime Kuma? - Lightweight, open-source, easy UI, supports many notification channels and webhooks.

---

9. Troubleshooting — examples from recent pipeline runs

Issue 1 — `tag does not exist: prem18062000/react-app-dev:latest-dev` - Occurred when pushing from a different agent than build agent and image wasn't present locally on push agent. - Resolution: ensure `push` runs on the same agent that built the image, or store/pull image between nodes. We solved by running build and push on the `build_agent` and using consistent stash/unstash for repo files.

Issue 2 — Jenkins looked for files in `/null/` path - Root cause: copying from `$WORKSPACE` on a different node where `$WORKSPACE` was either empty or different. - Resolution: use `stash 'appsource'` in checkout stage and `unstash 'appsource'` on targeted nodes, or use `writeFile/readFile` with explicit paths.

Issue 3 — permission denied on Docker socket when running compose - Cause: docker commands run as a user without access to `/var/run/docker.sock`. - Resolution: run `sudo docker ...` in the script or add the agent user to `docker` group. We used `sudo` in `deploy.sh` to pull image and `docker` or `docker compose` run by the agent user (ensure sudoers allow it or run as root).

Issue 4 — agent.service failing because Java flags unsupported - Observed logs: `Unrecognized option: --enable-native-access=ALL-UNNAMED`. - Cause: agent service used Java 11 which doesn't accept that flag. - Resolution: remove unsupported flags and install Java 17 when needed. Our final `install.sh` uses OpenJDK 17 and a minimal java invocation.

---

10. Operational runbook

Deploy checklist (pre-deploy) - Ensure build agent has Docker and network access to Docker Hub. - Ensure target node(s) have sufficient disk and Docker installed. - Confirm `DEV_IMAGE` / `PROD_IMAGE` naming and tags. - Ensure `docker-compose.yaml` and `deploy.sh` exist in repo root with correct filenames.

Deploy steps (what Jenkins performs) 1. Checkout code and stash files. 2. Build and push image on `build_agent`. 3. Unstash and copy compose + deploy script to node. 4. Run `deploy.sh <IMAGE>` on node. 5. Post-deploy: Smoke test (curl) and then rely on Uptime Kuma for long-term monitoring.

Rollback - If a new image is faulty, deploy the previous tag: `bash ~/deploy.sh <previous-image-tag>` or use compose to set the previous image and `compose up -d`.

Health checks - Use the `healthcheck` in `docker-compose.yaml`. Jenkins post-deploy does a quick curl loop to verify. - Uptime Kuma provides continuous health checks and alerts.

---

11. Appendix — quick commands & helpers

Useful Docker / Compose

```
# View containers
sudo docker ps -a
# View images
sudo docker images
# Pull image manually
sudo docker pull prem18062000/react-app-prod:latest-prod
# Run compose
sudo docker compose -f /home/ubuntu/app/docker-compose.yaml up -d
```

Agent troubleshooting

```
# show logs
sudo journalctl -u jenkins-agent -n 200 --no-pager
# see systemd service
sudo systemctl status jenkins-agent -l
# check agent.jar
ls -la /home/ubuntu/jenkins-agent/agent.jar
# run agent manually for debugging
java -jar /home/ubuntu/jenkins-agent/agent.jar -url http://<controller>:8080/ -
secret <secret> -name <name> -webSocket -workDir /home/ubuntu/jenkins-agent
```

Monitoring

```
# run uptime-kuma (in ~/monitoring)
sudo docker compose -f monitoring-compose.yaml up -d
# stop the app container to test alerts
sudo docker stop react-app
# start again
sudo docker start react-app
```

Where to find the files

- Jenkinsfile, `deploy.sh`, `docker-compose.yaml`, `Dockerfile` — repository root (see screenshot above).
- `install.sh` and `build.sh` — add them to repo root and ensure they are executable.

Final notes

- Keep the compose filename consistent across repository and pipeline (you used `docker-compose.yaml` earlier — match that exact name).
- Use `sudo` or appropriate group memberships for Docker commands when running as non-root.
- For automated agent bootstrap at EC2 creation, include `install.sh` contents in the EC2 user-data (and replace placeholders like `AGENT_SECRET`) so the instance registers to Jenkins during first boot.

If you want, I can: - Export this documentation as a downloadable Markdown file or PDF. - Create `build.sh` and `install.sh` as files in the repo and open a PR patch.