1. Complete **backup scripts** (DB dump, volume backup, rotation, और optional S3 upload).
2. **AWS EC2 step-by-step deployment guide** (instance, security groups, Docker, Docker Compose, deploy commands).
3. **Docker + Nginx + HTTPS setup** (nginx reverse proxy config + Certbot flow — certs host पर बनाएँ और container में mount करें).
4. **Resume storage architecture design** (local bind-mount + async copy to S3 + IAM policy + lifecycle).

नीचे हर चीज़ code + हर लाइन की सरल हिंदी व्याख्या के साथ दी गई है — सीधे copy-paste कर के इस्तेमाल कर लेना। अगर कोई फ़ाइल का नाम नहीं बताया है, तो ऊपर वाला project structure follow कर लेना (`project/ backend/ frontend/ docker-compose.yml nginx/ backup/ .env`).

---

1) .env (root) — centralized config

```
# .env (रूट में रखो, never commit to git)
POSTGRES_USER=achal
POSTGRES_PASSWORD=pass123
POSTGRES_DB=jobdb
BACKUP_DIR=/opt/backups
S3_BUCKET=my-job-portal-resumes   # अगर S3 use करोगे तो bucket name
AWS_REGION=ap-south-1            # अपने AWS region (option)
DOMAIN=example.com               # आपका domain
EMAIL=you@example.com            # Let's Encrypt renewal notices
```

क्यों: सभी secrets और config यहाँ से docker-compose में इस्तेमाल होंगे। `.env` सुरक्षित जगह रखो।

---

2) docker-compose.yml (full stack + nginx)

```
version: "3.8"

services:
  db:
    image: postgres:14
    restart: unless-stopped
    env_file:
      - .env
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=${POSTGRES_DB}
```

```
    volumes:
      - pg_data:/var/lib/postgresql/data
    networks:
      - jobnet

  backend:
    build: ./backend
    depends_on:
      - db
    env_file:
      - .env
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/${POSTGRES_DB}
      - SPRING_DATASOURCE_USERNAME=${POSTGRES_USER}
      - SPRING_DATASOURCE_PASSWORD=${POSTGRES_PASSWORD}
      - RESUME_DIR=/app/uploads
    ports:
      - "8080:8080"
    volumes:
      - ./backend/logs:/app/logs            # logs host पर रखो
      - ./uploads:/app/uploads             # resumes host bind mount
(persistent)
    networks:
      - jobnet

  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    networks:
      - jobnet

  nginx:
    build: ./nginx
    depends_on:
      - frontend
      - backend
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d
      - ./nginx/certs:/etc/letsencrypt/live   # host certs mount (certbot
created)
      - ./nginx/html:/usr/share/nginx/html
    networks:
      - jobnet

volumes:
  pg_data:

networks:
  jobnet:
```

ध्यान: `./uploads` host folder में सभी resume files store होंगी — यह bind mount तुम्हारे VPS पर physical files बनाता है। `/nginx/certs` में host-made certificates mount हो कर nginx container को HTTPS देता है।

---

3) nginx config (reverse proxy) —

nginx/conf.d/default.conf

```
server {
    listen 80;
    listen [::]:80;
    server_name ${DOMAIN};

    # HTTP -> HTTPS redirect
    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name ${DOMAIN};

    ssl_certificate /etc/letsencrypt/live/${DOMAIN}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/${DOMAIN}/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # Backend APIs proxied to backend service
    location /api/ {
        proxy_pass http://backend:8080/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # Frontend (static)
    location / {
        proxy_pass http://frontend:80;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

क्या कर रहा है: HTTP पर आने वाली request को HTTPS पर भेजेगा; `/api/*` backend को भेजेगा; बाकी सब frontend को। SSL certs host में Certbot से बनाएंगे और nginx को mount करेंगे।

---

4) Certbot (Let's Encrypt) — Host पर certificate बनाना और renew cron

हम certs host पर बनाएँगे और nginx container में mount करेंगे। इसका कारण: certbot को host पर root privileges चाहिए और mounted certs container में काम करे।

Certbot install & obtain cert (Ubuntu 示例)

```
# Ubuntu EC2 / VPS पर (run as sudo)
sudo apt update
sudo apt install -y certbot

# nginx temporarily up needs to listen on 80 to validate,
# इसलिए पहले docker-compose से nginx चालू रखो (80 पर) या host पे nginx use करो।
# अगर nginx container पहले से 80 पर है और domain DNS points to server, certbot के लिए
# हम standalone method use कर सकते हैं (port 80 temporarily taken).
sudo systemctl stop nginx || true   # अगर host nginx चल रहा हो तो बंद करो
sudo certbot certonly --standalone -d ${DOMAIN} --email ${EMAIL} --agree-tos
--non-interactive

# generated certs will be in /etc/letsencrypt/live/${DOMAIN}
# अब copy/mount कर दो project/nginx/certs में (ya symlink)
sudo mkdir -p /path/to/your/project/nginx/certs
sudo cp -r /etc/letsencrypt/live/${DOMAIN}
/path/to/your/project/nginx/certs/${DOMAIN}

# auto renew (certbot sets up /etc/cron.d certbot), but ensure after renew
you copy updated certs into project dir
# create a small script to sync renewed certs to project dir:
sudo tee /usr/local/bin/sync_certs.sh > /dev/null <<'EOF'
#!/bin/bash
DOMAIN="${DOMAIN}"
PROJECT_CERT_DIR="/path/to/your/project/nginx/certs/${DOMAIN}"
mkdir -p "$PROJECT_CERT_DIR"
cp -Lr /etc/letsencrypt/live/"$DOMAIN"/* "$PROJECT_CERT_DIR"/
cp -Lr /etc/letsencrypt/archive/"$DOMAIN"/* "$PROJECT_CERT_DIR"/ || true
EOF
sudo chmod +x /usr/local/bin/sync_certs.sh

# add to cron after certbot renew
sudo crontab -l > mycron || true
echo "0 3 * * * /usr/bin/certbot renew --quiet &&
/usr/local/bin/sync_certs.sh" >> mycron
sudo crontab mycron
rm mycron
```

नोट: ऊपर `/path/to/your/project` को अपने project root path से बदल देना। `sync_certs.sh` cert renew के बाद project directory में copy करता है ताकि docker-compose up में nginx container latest cert से serve करे।

---

5) PostgreSQL backup script — `backup/backup_db.sh` (detailed)

```
#!/bin/bash
# backup/backup_db.sh
set -euo pipefail

# load env if exists
DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)/.."
if [ -f "$DIR/.env" ]; then
  export $(grep -v '^#' "$DIR/.env" | xargs)
fi

BACKUP_DIR=${BACKUP_DIR:-/opt/backups}
TIMESTAMP=$(date +"%F_%H-%M")
COMPOSE_PROJECT_DIR=${DIR}
DB_CONTAINER_ID=$(cd "$COMPOSE_PROJECT_DIR" && docker-compose ps -q db)

mkdir -p "$BACKUP_DIR"

if [ -z "$DB_CONTAINER_ID" ]; then
  echo "Postgres container not found. Starting db..."
  cd "$COMPOSE_PROJECT_DIR"
  docker-compose up -d db
  DB_CONTAINER_ID=$(docker-compose ps -q db)
fi

echo "Taking pg_dump from container $DB_CONTAINER_ID ..."
docker exec -t "$DB_CONTAINER_ID" pg_dump -U "${POSTGRES_USER}"
"${POSTGRES_DB}" | gzip > "${BACKUP_DIR}/pg_backup_${TIMESTAMP}.sql.gz"

# optional: push to S3 (if aws cli configured)
if [ -n "${S3_BUCKET:-}" ]; then
  echo "Uploading backup to s3://${S3_BUCKET}/db_backups/"
  aws s3 cp "${BACKUP_DIR}/pg_backup_${TIMESTAMP}.sql.gz"
"s3://${S3_BUCKET}/db_backups/"
fi

# rotate: keep last 30 backups locally
ls -1t "${BACKUP_DIR}"/pg_backup_*.sql.gz 2>/dev/null | tail -n +31 | xargs -
r rm --
echo "Backup saved: ${BACKUP_DIR}/pg_backup_${TIMESTAMP}.sql.gz"
```

लाइन-बाय-लाइन (हिंदी):

- `set -euo pipefail` — script strict mode (error पर रुक जाएगा)
- `.env` load — ताकि POSTGRES_USER आदि मिल जाएँ
- `DB_CONTAINER_ID` — docker-compose से db container id लाता है
- `docker exec ... pg_dump` — container के अंदर से DB dump निकालकर gzip कर देता है
- `aws s3 cp` — (अगर S3 configure किया) backup S3 पर भेजता है
- rotation — 30 से पुरानी backup फाइलें हटाता है

Permission: `chmod +x backup/backup_db.sh` और crontab में schedule करो (नीचे)।

Cron example (daily 2 AM):

```
# edit crontab (as deploy user)
crontab -e
# add:
0 2 * * * /path/to/project/backup/backup_db.sh >> /var/log/backup_db.log 2>&1
```

---

6) Restore script — `backup/restore_db.sh`

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

if [ -z "${1:-}" ]; then
  echo "Usage: $0 /path/to/backup.sql.gz"
  exit 1
fi

BACKUP_FILE="$1"
DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)/.."

cd "$DIR"
# stop backend+frontend to avoid connections
docker-compose down

# start db only
docker-compose up -d db
DB_ID=$(docker-compose ps -q db)

echo "Restoring $BACKUP_FILE into container $DB_ID ..."
gunzip -c "$BACKUP_FILE" | docker exec -i "$DB_ID" psql -U "${POSTGRES_USER}"
-d "${POSTGRES_DB}"

# bring up other services
docker-compose up -d
echo "Restore complete."
```

क्या करता है: backup फ़ाइल से DB को restore करता है, बीच में services down/up कर देता है ताकि consistency रहे।

---

7) Volume backup & restore (Postgres volume) — commands

Backup named volume `pg_data` to host tar:

```
docker run --rm \
  -v pg_data:/data \
  -v $(pwd)/backup:/backup \
  alpine \
  sh -c "cd /data && tar czf /backup/pg_data_$(date +%F).tgz ."
```

Restore into volume:

```
docker run --rm \
  -v pg_data:/data \
  -v $(pwd)/backup:/backup \
  alpine \
  sh -c "cd /data && tar xzf /backup/pg_data_2023-01-01.tgz"
```

क्यों: named volume filesystem को temporary container से access कर के tar बनाया जा सकता है।

---

8) Resume storage architecture (design + flow)

Goals:

- Uploads immediate response (fast).
- Files persisted on host for quick access.
- Long-term durable storage on S3 (for disaster recovery & cross-cloud migration).
- Metadata (filename, s3_path, uploader_id) stored in PostgreSQL.

Components:

1. **Local bind mount** (`./uploads` में) — immediate store, serve and process.
2. **Async uploader worker** — backend triggers background job (e.g., use Spring Async / RabbitMQ / simple cron) to upload file to S3 and update DB reference.
3. **Postgres metadata table**:

```
CREATE TABLE resumes (
  id bigserial primary key,
```

```
  user_id bigint,
  original_filename text,
  local_path text,
  s3_key text,
  uploaded_at timestamptz default now(),
  checksum text,
  size bigint
);
```

4. **S3 bucket** with lifecycle rule (move to Glacier after 30 days, versioning enabled).
5. **IAM user & policy** with least privilege for upload/delete.

Simple flow:

- User uploads file → backend saves to `/app/uploads/uuid.pdf` (bind mounted to host `./uploads`).
- Backend inserts metadata with `local_path` and empty `s3_key`.
- Background worker uploads file to `s3://my-job-portal-resumes/YYYY/MM/DD/uuid.pdf` and updates `s3_key`.
- After successful S3 upload, optionally delete local file older than X days to save disk.

Sample Java (pseudo) uploader logic:

```
// save file locally
String localPath = "/app/uploads/" + uuid + ".pdf";
Files.copy(inputStream, Paths.get(localPath));

// insert metadata with localPath

// async upload
s3Client.putObject(bucket, s3Key, new File(localPath));
updateMetadataSetS3Key(id, s3Key);

// optionally remove local file older than 7 days via cron
```

---

9) IAM Policy (least privileged) — example for S3 upload/delete

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3AccessForResumes",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
```

```
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-job-portal-resumes",
        "arn:aws:s3:::my-job-portal-resumes/*"
      ]
    }
  ]
}
```

कभी सॉफ्टकोडेड credentials मत रखो — EC2 पर role attach करो (recommended) या secrets manager use करो.

---

10) AWS EC2 step-by-step deployment (quick & complete)

A. EC2 Instance (t2.medium / t3.medium recommended for small production)

1. Create EC2 instance (Ubuntu 22.04 LTS). Attach **security group**:
   - TCP 22 (your-ip only) — SSH
   - TCP 80 (0.0.0.0/0)
   - TCP 443 (0.0.0.0/0)
   - TCP 8080/3000 only if required (but nginx will proxy)
2. Attach IAM Role to EC2 with S3 access (if using S3). This avoids storing AWS keys.

B. SSH and install Docker & Docker Compose

```
# SSH into server
ssh -i mykey.pem ubuntu@ec2-public-ip

# update & install
sudo apt update && sudo apt upgrade -y
# install docker
sudo apt install -y docker.io
sudo systemctl enable --now docker
# allow deploy user to use docker without sudo
sudo usermod -aG docker $USER
# install docker-compose plugin
sudo apt-get install -y docker-compose-plugin
# logout/login required for group change
```

C. Clone project & env

```
cd /opt
sudo git clone https://your-repo.git job-portal
cd job-portal
sudo cp .env.example .env   # fill values
sudo chown -R ubuntu:ubuntu .   # make deploy user owner
```

D. Build & start

```
docker compose build
docker compose up -d
# check
docker compose ps
docker compose logs -f backend
```

E. Obtain certs & copy to project (see Certbot step earlier)

- Ensure DNS A record points to EC2 public IP for `${DOMAIN}`.
- Run certbot on EC2 to create certs for domain and copy to project `nginx/certs/${DOMAIN}`.
- Restart docker-compose:

```
docker compose down
docker compose up -d
```

F. Setup backups cron

```
# ensure backup scripts executable
chmod +x backup/backup_db.sh backup/restore_db.sh
# edit crontab for deploy user
crontab -e
# add (daily 2am)
0 2 * * * /opt/job-portal/backup/backup_db.sh >> /var/log/backup_db.log 2>&1
```

---

11) Extra: Docker image export for migration (Hostinger → AWS)

```
# save images
docker save myproject_backend:latest -o backend_latest.tar
# scp to other host
scp backend_latest.tar ubuntu@ec2-ip:/home/ubuntu/
# on target host
docker load -i backend_latest.tar
```

---

12) Security hardening checklist (production)

- Use HTTPS everywhere (Let's Encrypt or managed cert).
- Attach IAM role to EC2 (no access keys on disk).
- Keep `.env` private and not in repo.
- Use database user with least privileges.
- Use firewall (ufw) to limit ports.
- Use non-root user inside containers (set `USER` in Dockerfile).
- Regularly patch OS and update images.
- Enable RDS (managed DB) in AWS for large scale — migrate DB to RDS for durability.

---

13) Useful commands cheat sheet (quick)

```
# docker / compose
docker compose build
docker compose up -d
docker compose down
docker compose logs -f backend
docker ps
docker volume ls
docker volume inspect pg_data

# backup & restore
./backup/backup_db.sh
./backup/restore_db.sh /path/to/pg_backup_2025-11-19_02-00.sql.gz

# certs
sudo certbot certonly --standalone -d ${DOMAIN} --email ${EMAIL} --agree-tos
--non-interactive
```