1) System Overview

1.1 Components

- **Primary Service (`stt_primary.py`)**
  - Faster-Whisper **small** (CT2).
  - Async FastAPI with per-process **GPU concurrency gate**.
  - Confidence-based routing: if **primary errors** or **low confidence**, calls fallback.
  - Endpoints:
    - `POST /v1/transcribe` (multipart)
    - `GET /v1/health` (JSON liveness)
    - `GET /v1/metrics` (JSON snapshot)
    - `GET /metrics` (Prometheus)
- **Fallback Service (`stt_fallback.py`)**
  - Faster-Whisper **medium** (CT2).
  - No routing, just transcribe.
  - Endpoints: `POST /v1/transcribe`, `GET /v1/health`, `GET /metrics`.
- **Client SDK (`stt_sdk.py`)**
  - Async HTTP client, retries, telemetry hook, rolling latency stats.
  - Optional `STTPool` to distribute across multiple primaries.

1.2 Data Flow (happy path)

Client → **Primary** (small):

- If **forced language** supplied → primary uses it, **lang-prob check disabled**.
- If **autodetect** → primary computes logprob & language probability.
- If confidence is sufficient → **returns**.
- Else → calls **Fallback** (same forced language if provided, else autodetect).

Primary always **releases its GPU slot** before contacting fallback.

---

2) Configuration Management

2.1 Config Files

Both services read a **.env file** using `config_loader.py`. By default:

- Primary: `stt_primary.env`
- Fallback: `stt_fallback.env`
- Override path by exporting `CONFIG_PATH=/path/to/file.env` at process start.

Primary (`stt_primary.env`)

```
# Model & device
CT2_SMALL_PATH=/models/whisper-small-ct2
DEVICE=cuda
DEVICE_INDEX=0
COMPUTE_TYPE=float16
CPU_THREADS=2

# Concurrency & versioning
MAX_CONCURRENCY=2
SERVER_COMMIT=prod-YYYY-MM-DD

# Routing thresholds
CONF_LOGPROB_THRESHOLD=0.55
CONF_LANGPROB_THRESHOLD=0.70
```

```
MIN_WORDS_FOR_CONF=3

# Fallback target
FALLBACK_URL=http://stt-fallback:8082
FALLBACK_API_KEY=fb_key_123

# Security / limits
API_KEYS=primary_key_abc
RATE_LIMIT_RPS=15
RATE_LIMIT_BURST=30
MAX_UPLOAD_MB=16
```

Fallback (`stt_fallback.env`)

```
CT2_MEDIUM_PATH=/models/whisper-medium-ct2
DEVICE=cuda
DEVICE_INDEX=0
COMPUTE_TYPE=int8
CPU_THREADS=2

API_KEYS=fb_key_123
RATE_LIMIT_RPS=60
RATE_LIMIT_BURST=120
MAX_UPLOAD_MB=16
```

Notes

- `API_KEYS` is a comma-separated allowlist for Bearer auth.
- Primary → Fallback uses `FALLBACK_API_KEY` which must appear in fallback's `API_KEYS`.
- Keep `CPU_THREADS` low (1–2) with GPUs to reduce contention and jitter.
- `MAX_UPLOAD_MB` protects latency; 16 MB is a safe default.

2.2 Versioning

Set `SERVER_COMMIT` to track deploy provenance (git SHA or date tag). Exposed in `/v1/metrics`.

---

3) Deployment Patterns

3.1 Runtime Requirements

- NVIDIA GPU with recent CUDA driver.
- ctranslate2 models available locally (fast SSD).
- Python ≥ 3.10.
- Install: `fastapi uvicorn pynvml httpx prometheus-client faster-whisper` (pin versions for prod).

3.2 Process Model

Run **one worker per GPU**:

```
CONFIG_PATH=stt_fallback.env uvicorn stt_fallback:app --host 0.0.0.0
--port 8082 --workers 1
CONFIG_PATH=stt_primary.env  uvicorn stt_primary:app  --host 0.0.0.0
--port 8081 --workers 1
```

Use the app's internal `MAX_CONCURRENCY` gate to tune throughput.

3.3 systemd Units (bare-metal)

`/etc/systemd/system/stt-fallback.service`

```
[Unit]
Description=STT Fallback
After=network-online.target
Wants=network-online.target

[Service]
User=stt
```

```
Group=stt
WorkingDirectory=/opt/stt
Environment=CONFIG_PATH=/opt/stt/stt_fallback.env
ExecStart=/opt/stt/venv/bin/uvicorn stt_fallback:app --host 0.0.0.0
--port 8082 --workers 1
Restart=always
RestartSec=3

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

`/etc/systemd/system/stt-primary.service`

```
[Unit]
Description=STT Primary
After=stt-fallback.service

[Service]
User=stt
Group=stt
WorkingDirectory=/opt/stt
Environment=CONFIG_PATH=/opt/stt/stt_primary.env
ExecStart=/opt/stt/venv/bin/uvicorn stt_primary:app --host 0.0.0.0
--port 8081 --workers 1
Restart=always
RestartSec=3

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

3.4 Containers (Docker)

Minimal `Dockerfile` (per service):

```
FROM nvidia/cuda:12.1.0-runtime-ubuntu22.04
RUN apt-get update && apt-get install -y python3 python3-pip && rm -rf
/var/lib/apt/lists/*
WORKDIR /app
```

```
COPY requirements.txt .
RUN pip3 install --no-cache-dir -r requirements.txt
COPY . .
ENV CONFIG_PATH=/app/stt_primary.env
CMD
["uvicorn","stt_primary:app","--host","0.0.0.0","--port","8081","--wor
kers","1"]
```

Run with GPU:

```
docker run --gpus=all -p 8081:8081 --env-file stt_primary.env
stt/primary:tag
```

3.5 Kubernetes (GPU nodes)

- Use **NVIDIA device plugin** and **Node Feature Discovery**.
- Pin pods to GPU nodes with `nodeSelector`/`tolerations`.
- One pod (1 replica) per GPU; scale horizontally.

Example (primary):

```
apiVersion: apps/v1
kind: Deployment
metadata: { name: stt-primary }
spec:
  replicas: 1
  selector: { matchLabels: { app: stt-primary } }
  template:
    metadata: { labels: { app: stt-primary } }
    spec:
      nodeSelector: { "nvidia.com/gpu.present": "true" }
      containers:
        - name: stt-primary
          image: yourrepo/stt-primary:tag
          ports: [{ containerPort: 8081 }]
          env:
```

```
            - name: CONFIG_PATH
              value: /config/stt_primary.env
          volumeMounts:
            - name: cfg
              mountPath: /config
              readOnly: true
          resources:
            limits:
              nvidia.com/gpu: 1
              cpu: "4"
              memory: "8Gi"
            requests:
              nvidia.com/gpu: 1
              cpu: "2"
              memory: "4Gi"
      volumes:
        - name: cfg
          configMap: { name: stt-primary-config }
---
apiVersion: v1
kind: Service
metadata: { name: stt-primary }
spec:
  selector: { app: stt-primary }
  ports: [{ port: 8081, targetPort: 8081 }]
```

Config & secrets

- Put `.env` contents into a **ConfigMap** (non-secret) and API keys into a **Secret** (or bake keys directly into the file with limited RBAC).
- NetworkPolicy: allow only **clients** → **primary** and **primary** → **fallback**.

---

4) Security & Networking

4.1 Authentication

- Bearer tokens:
    - Clients → Primary: one of `API_KEYS`.
    - Primary → Fallback: `FALLBACK_API_KEY` must be accepted by fallback.

Key rotation (no downtime)

1. Add new key to **both** the service (`API_KEYS`) and clients.
2. Verify traffic with new key.
3. Remove old key.

4.2 Rate Limiting

- Token bucket per API key (or per IP when no key).
- Primary & fallback have separate `RATE_LIMIT_RPS` and `RATE_LIMIT_BURST`.
- Size `BURST` high enough for short spikes (suggest 2–4× `MAX_CONCURRENCY`).

4.3 Request Size Cap

- `MAX_UPLOAD_MB` enforced via `Content-Length`. Oversized → `413`.
- Keep clips short for latency; split long media on client if needed.

4.4 TLS & Edge

- Place services behind your ingress / API gateway:
    - TLS termination, WAF, authz, request timeout policy.
- Restrict fallback to **only accept** traffic from primary subnets or via mTLS.

---

5) Observability

5.1 Metrics (Prometheus)

Scrape `GET /metrics`. Primary exports:

Counters

- `stt_requests_total{endpoint="/v1/transcribe"}`
- `stt_requests_exceptions_total{endpoint,type}`
- `stt_fallback_total{result="used|skipped|failed"}`

Histograms

- `stt_request_total_seconds` (end-to-end)
- `stt_primary_infer_seconds`
- `stt_fallback_infer_seconds`
- `stt_queue_wait_seconds`

Gauges

- `stt_queue_depth`
- `stt_in_flight`
- `stt_gpu_utilization_pct`
- `stt_gpu_mem_used_mb`
- `stt_gpu_mem_total_mb`

5.2 Logging

- Run Uvicorn with access logs; pipe to your log stack.
- Recommend JSON formatting at the ingress (requestID, clientIP, status, bytes, duration).
- Include response headers in error logs: `X-Error-Type`, `X-Error-Message`, `X-Fallback-Error`, `X-Req-Id`.

5.3 Suggested Grafana Panels

- **Latency**: p50/p95/p99 of `stt_request_total_seconds` by instance.
- **Fallback rates**: `sum(rate(stt_fallback_total{result="used"}[5m])) / sum(rate(stt_requests_total[5m]))`.
- **Fallback failures**: `rate(stt_fallback_total{result="failed"}[5m])`.
- **Throughput**: `rate(stt_requests_total[5m])`.
- **GPU Utilization**: `avg(stt_gpu_utilization_pct)`.
- **Queue depth & in-flight**: gauges over time.

5.4 Alerting (PromQL examples)

High tail latency

```
histogram_quantile(0.95, sum(rate(stt_request_total_seconds_bucket[5m])) by (le, instance)) > 1.0
```

-

Fallback failure spike

```
rate(stt_fallback_total{result="failed"}[5m]) > 0.5
```

-

5xx at ingress

```
sum(rate(http_requests_total{status=~"5..",app="stt-primary"}[5m])) > 1
```

-

GPU saturation

```
avg_over_time(stt_gpu_utilization_pct[10m]) > 90
```

-

---

6) SLOs & Capacity Planning

6.1 Example SLOs

- **Latency**: p95 end-to-end < **1.0 s** for N-second clips (define N).
- **Availability**: 99.9% successful `POST /v1/transcribe`.
- **Fallback failure rate**: < **0.5%** of total requests.

6.2 Concurrency & Throughput

- Use **Little's Law** (approx): `concurrency ≈ arrival_rate * avg_latency`.
- Start with:
  - `MAX_CONCURRENCY = 2` per GPU on primary.
  - Fallback `RATE_LIMIT_BURST ≥ 2 * MAX_CONCURRENCY * replicas`.
- Measure p95; raise `MAX_CONCURRENCY` gradually until p95 worsens.

6.3 Tuning Knobs

- `CPU_THREADS`: 1–2 recommended with GPU.
- `beam_size`: keep `1` (greedy) for latency; accuracy trade-off otherwise.

- Thresholds:
    - Lower `CONF_LOGPROB_THRESHOLD` to **reduce** fallbacks.
    - Lower `MIN_WORDS_FOR_CONF` (e.g., 2) for short commands.

---

7) Operations

7.1 Start / Stop

- Start fallback first, then primary.
- Check health:
    - `GET /v1/health` → `{ ok: true }`
    - Prometheus scraping OK.

7.2 Rolling Updates (zero downtime)

1. Deploy **new fallback** (leave old running). Verify health & metrics.
2. Switch **primary** config `FALLBACK_URL` to new fallback (or update service discovery).
3. Deploy **primary**.
4. Decommission old fallback.

7.3 Scale Out

- Add more **primary** replicas behind a load balancer.
- Ensure **fallback capacity** (RPS + BURST) can absorb worst-case routing.
- For extreme spikes, temporarily **force language at clients** to avoid lang-prob fallbacks.

7.4 Key Rotation

1. Append new key to `API_KEYS` in both services.
2. Roll clients to use the new key.
3. Remove old key from `API_KEYS`.

---

8) Runbooks (Troubleshooting)

8.1 Sporadic `502 Bad Gateway` from primary

Definition: Primary attempted fallback; fallback request failed.

Immediate checks

- Inspect response headers:
  - `X-Error-Type`, `X-Error-Message`, `X-Fallback-Error`, `X-Req-Id`.
- Check fallback logs for rate-limit (`429`), 5xx, or timeouts.

Common causes & fixes

- **Fallback 429** (rate limit):
  - Raise `RATE_LIMIT_BURST` and/or `RATE_LIMIT_RPS` on fallback.
  - Reduce unnecessary fallbacks (lower thresholds or **force language** at clients).
- **Network/timeout**:
  - Verify fallback is reachable; set low DNS TTL; ensure no packet loss.
  - Primary already retries once; consider increasing fallback replicas.
- **Payload too large (413)**:
  - Enforce smaller clips on client; raise `MAX_UPLOAD_MB` only if justified.

8.2 Always falling back

- If clients pass `language=xx`, lang-prob gate is disabled; only logprob/min-words can trigger fallback.
- If using autodetect on **very short** clips:
    - Reduce `MIN_WORDS_FOR_CONF` to 2.
    - Or **force language** at the client.

8.3 High tail latency (p95 ↑)

- Reduce `MAX_CONCURRENCY` or `CPU_THREADS`.
- Ensure model folders are on local SSD (no remote FS latency).
- Watch `stt_queue_wait_seconds` vs `stt_primary_infer_seconds` to identify queueing.

8.4 GPU OOM or memory creep

- Check `stt_gpu_mem_used_mb`; reduce concurrency or move to bigger GPU.
- Ensure only **1 worker per GPU**.
- Bounce process to reclaim memory if necessary; investigate long-running load patterns.

---

9) Resilience, DR & Compliance

- **Stateless** services: keep model artifacts in image or node-local SSD; redeployable anywhere.
- **Backups**: keep images and model artifacts in registries/artifact stores.
- **Multi-AZ**: run primary replicas across zones; keep a fallback cluster with equal or higher capacity.

- **Ingress timeouts**: set appropriately (e.g., 65s) to cover slower medium inferences.

---

10) Testing & Validation

10.1 Pre-prod checks

- Run `tests/smoke_test_stt.py` against your cluster with real audio:
    - Language **forced** and **autodetect**.
    - Mix of short & long clips.
- Confirm:
    - Low fallback rate for expected languages.
    - p95 latency within SLO.
    - No growth in process RSS during a 2-hour soak.

10.2 Load Test Hints

- Vary concurrency: 1, 2, 4, 8, 16…
- Observe:
    - p95 end-to-end, primary infer, fallback infer, queue wait.
    - GPU util and mem.
- Calibrate `MAX_CONCURRENCY` to the knee of the latency curve.

---

11) Change Management

- Tie deployments to `SERVER_COMMIT`.

- Use canary + metrics guardrails:
    - Abort if p95 > 1.5× baseline for 10 min.
    - Abort if fallback failures spike.
- Keep a **rollback** command ready (previous image tag and configs).

---

12) Appendix

12.1 Port Reference

- Primary: `8081` (HTTP)
- Fallback: `8082` (HTTP)
- Metrics: `/metrics`
- Health: `/v1/health`

12.2 Headers to Capture on Errors

- `X-Req-Id`, `X-Error-Type`, `X-Error-Message`, `X-Fallback-Error`
- Also capture timings: `X-Queue-Wait-ms`, `X-Primary-Infer-ms`, `X-Fallback-Infer-ms`, `X-Total-ms`.
- Routing: `X-Fallback-Used`, `X-Final-Model`, `X-Conf-Below`, `X-Lang-Check-Applicable`.

TL;DR Operations Checklist

- Models on fast local storage; 1 worker per GPU.

- Primary & fallback configured via `.env`; keys deployed and rotated safely.
- Rate limits sized (fallback burst ≥ 2–4× primary concurrency).
- Prometheus scraping `/metrics`; Grafana dashboards and alerts live.
- SLOs defined (latency, availability, fallback failure rate).
- Rollouts are canaried and observable; rollbacks ready.
- Runbooks printed and known: **502**, latency, fallbacks, GPU pressure.

Docker ops guide (Primary & Fallback as separate containers)

0) Repo layout (suggested)

```
.
├── config_loader.py
├── stt_primary.py
├── stt_fallback.py
├── stt_sdk.py
├── requirements.txt
├── docker/
│   ├── Dockerfile.primary
│   └── Dockerfile.fallback
├── configs/
│   ├── stt_primary.env
```

```
|   └── stt_fallback.env
└── models/
    ├── whisper-small-ct2/      # mount into containers at
/models/whisper-small-ct2
    └── whisper-medium-ct2/     # mount into containers at
/models/whisper-medium-ct2
```

> The containers read config files, not process env. We pass **`-e CONFIG_PATH=/config/....env`** and **bind-mount** those `.env` files into the containers.

---

1) Requirements on the host(s)

- **NVIDIA driver + NVIDIA Container Toolkit** installed on any host that will run a GPU container (primary and/or fallback).
- Docker Engine ≥ 20.10.

Tip: test GPU visibility with:

```
docker run --rm --gpus all nvidia/cuda:12.1.0-runtime-ubuntu22.04
nvidia-smi
```

---

2) Config files (container-friendly)

`configs/stt_fallback.env` (machine that will run the fallback)

```
CT2_MEDIUM_PATH=/models/whisper-medium-ct2
DEVICE=cuda
DEVICE_INDEX=0
COMPUTE_TYPE=int8
CPU_THREADS=2

API_KEYS=fb_key_123
RATE_LIMIT_RPS=60
```

```
RATE_LIMIT_BURST=240
MAX_UPLOAD_MB=16
```

`configs/stt_primary.env` (machine that will run the primary)

Set `FALLBACK_URL` to the fallback's reachable address.

```
CT2_SMALL_PATH=/models/whisper-small-ct2
DEVICE=cuda
DEVICE_INDEX=0
COMPUTE_TYPE=float16
CPU_THREADS=2

MAX_CONCURRENCY=2
SERVER_COMMIT=prod-2025-08-18

CONF_LOGPROB_THRESHOLD=0.55
CONF_LANGPROB_THRESHOLD=0.70
MIN_WORDS_FOR_CONF=3

FALLBACK_URL=http://<FALLBACK_HOST_OR_IP>:8082
FALLBACK_API_KEY=fb_key_123

API_KEYS=primary_key_abc
RATE_LIMIT_RPS=15
RATE_LIMIT_BURST=60
MAX_UPLOAD_MB=16
```

---

3) Dockerfiles (two separate images)

`docker/Dockerfile.fallback`

```
FROM nvidia/cuda:12.1.0-runtime-ubuntu22.04

# System deps
RUN apt-get update && apt-get install -y python3 python3-venv
python3-pip && rm -rf /var/lib/apt/lists/*
```

```
WORKDIR /app
COPY requirements.txt .
RUN pip3 install --no-cache-dir -r requirements.txt

# App code
COPY config_loader.py stt_fallback.py ./

# Default config path (override at runtime if needed)
ENV CONFIG_PATH=/config/stt_fallback.env

# Expose HTTP
EXPOSE 8082

# Healthcheck (simple)
HEALTHCHECK --interval=30s --timeout=3s --start-period=20s CMD curl
-fsS http://localhost:8082/v1/health || exit 1

CMD
["uvicorn","stt_fallback:app","--host","0.0.0.0","--port","8082","--wo
rkers","1"]
```

docker/Dockerfile.primary

```
FROM nvidia/cuda:12.1.0-runtime-ubuntu22.04

RUN apt-get update && apt-get install -y python3 python3-venv
python3-pip curl && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY requirements.txt .
RUN pip3 install --no-cache-dir -r requirements.txt

COPY config_loader.py stt_primary.py ./

ENV CONFIG_PATH=/config/stt_primary.env

EXPOSE 8081
```

```
HEALTHCHECK --interval=30s --timeout=3s --start-period=20s CMD curl
-fsS http://localhost:8081/v1/health || exit 1

CMD
["uvicorn","stt_primary:app","--host","0.0.0.0","--port","8081","--wor
kers","1"]
```

`requirements.txt` (pin as you prefer)

```
fastapi==0.111.0
uvicorn==0.30.3
httpx==0.27.0
pydantic==2.8.2
faster-whisper==1.0.0
ctranslate2==4.3.1
numpy==1.26.4
prometheus-client==0.20.0
pynvml==11.5.0
```

> If your CT2/Faster-Whisper version differs for your models, pin accordingly.

---

4) Build images

From repo root:

```
# Fallback
docker build -f docker/Dockerfile.fallback -t stt-fallback:1.0 .

# Primary
docker build -f docker/Dockerfile.primary -t stt-primary:1.0 .
```

(Optional) Push to a registry for multi-machine deployment:

```
docker tag stt-fallback:1.0 registry.example.com/your/stt-fallback:1.0
docker tag stt-primary:1.0  registry.example.com/your/stt-primary:1.0
```

```
docker push registry.example.com/your/stt-fallback:1.0
docker push registry.example.com/your/stt-primary:1.0
```

---

5) Run locally (single box)

Assuming `./models` has both model folders:

```
# 1) Start Fallback on 8082
docker run -d --name stt-fallback \
  --gpus all \
  -p 8082:8082 \
  -e CONFIG_PATH=/config/stt_fallback.env \
  -v $(pwd)/configs/stt_fallback.env:/config/stt_fallback.env:ro \
  -v $(pwd)/models/whisper-medium-ct2:/models/whisper-medium-ct2:ro \
  stt-fallback:1.0

# 2) Start Primary on 8081
# Make sure stt_primary.env has FALLBACK_URL=http://127.0.0.1:8082
docker run -d --name stt-primary \
  --gpus all \
  -p 8081:8081 \
  -e CONFIG_PATH=/config/stt_primary.env \
  -v $(pwd)/configs/stt_primary.env:/config/stt_primary.env:ro \
  -v $(pwd)/models/whisper-small-ct2:/models/whisper-small-ct2:ro \
  stt-primary:1.0
```

Quick smoke:

```
curl -s http://localhost:8082/v1/health
curl -s http://localhost:8081/v1/health
```

---

6) Run on two different machines

Machine B (Fallback host)

```
# Copy or mount the medium model to /opt/stt/models/whisper-medium-ct2
# Copy configs/stt_fallback.env to /opt/stt/configs/stt_fallback.env
(edit API_KEYS as needed)

docker run -d --name stt-fallback \
  --gpus all \
  -p 8082:8082 \
  -e CONFIG_PATH=/config/stt_fallback.env \
  -v /opt/stt/configs/stt_fallback.env:/config/stt_fallback.env:ro \
  -v /opt/stt/models/whisper-medium-ct2:/models/whisper-medium-ct2:ro
\
  registry.example.com/your/stt-fallback:1.0
```

- Ensure port **8082** is reachable from the primary host (open firewall / security groups).
- Keep `API_KEYS` on fallback containing the **primary's** `FALLBACK_API_KEY`.

Machine A (Primary host)

Edit `configs/stt_primary.env`:

```
FALLBACK_URL=http://<MACHINE_B_IP_OR_DNS>:8082
FALLBACK_API_KEY=fb_key_123
```

Run:

```
docker run -d --name stt-primary \
  --gpus all \
  -p 8081:8081 \
  -e CONFIG_PATH=/config/stt_primary.env \
  -v /opt/stt/configs/stt_primary.env:/config/stt_primary.env:ro \
  -v /opt/stt/models/whisper-small-ct2:/models/whisper-small-ct2:ro \
  registry.example.com/your/stt-primary:1.0
```

Sanity:

```
# from the PRIMARY box
curl -s http://localhost:8081/v1/health
```

```
# optional: test fallback reachability directly from primary machine
curl -s http://<MACHINE_B_IP_OR_DNS>:8082/v1/health
```

---

7) Security & auth (Docker specifics)

- **Primary inbound**: clients must send `Authorization: Bearer primary_key_abc` if `API_KEYS` is set in `stt_primary.env`.
- **Primary → Fallback**: the primary sends `Authorization: Bearer <FALLBACK_API_KEY>`. This key must appear in **fallback's** `API_KEYS`.
- Prefer running both behind your reverse proxy / API gateway for TLS/WAF. If exposing public endpoints, add network ACLs.

---

8) Observability in containers

- **Prometheus**: scrape
  - Primary: `http://<primary-host>:8081/metrics`
  - Fallback: `http://<fallback-host>:8082/metrics`
- **Kibana/Logs**: aggregate container stdout/stderr; include response headers in error logs.
- **Health**: `GET /v1/health` → `{ "ok": true, ... }`.

---

9) Capacity & tuning when containerized

- Keep **one container per GPU** (`--workers 1` is already set).
- Tune **`MAX_CONCURRENCY`** in the `.env` file; start with 2 and watch p95.

- Fallback `RATE_LIMIT_BURST` should be ≥ **2–4×** total primary concurrency across replicas to absorb bursts.
- Keep models on **fast local disk** and mount read-only.

---

10) Common gotchas (Docker)

- **No GPU in container** → ensure `--gpus all` and the NVIDIA Container Toolkit are installed.
- **Primary can't reach fallback** → check `FALLBACK_URL` host/IP, firewall, or DNS.
- **Random 502s** → check headers `X-Error-Message`/`X-Fallback-Error`; increase fallback `RATE_LIMIT_BURST` or pass a fixed `language` from clients to reduce fallbacks.
- **High latency** → lower `MAX_CONCURRENCY` or `CPU_THREADS`; verify models are local, not on network storage.

---

11) (Optional) local two-container demo with Compose

If you want a quick local demo without cross-machine networking:

Note: GPU flags in Compose vary by version; the most reliable path is still `docker run --gpus all`. If you do use Compose v2 with GPU support enabled, you can add:

```
# docker-compose.yml (GPU support may require Docker Compose v2 +
toolkit)
services:
  fallback:
    image: stt-fallback:1.0
    ports: [ "8082:8082" ]
    environment:
      CONFIG_PATH: /config/stt_fallback.env
    volumes:
```

```
      - ./configs/stt_fallback.env:/config/stt_fallback.env:ro
      - ./models/whisper-medium-ct2:/models/whisper-medium-ct2:ro
    deploy:
      resources:
        reservations:
          devices:
            - capabilities: [gpu]

  primary:
    image: stt-primary:1.0
    ports: [ "8081:8081" ]
    environment:
      CONFIG_PATH: /config/stt_primary.env
    volumes:
      - ./configs/stt_primary.env:/config/stt_primary.env:ro
      - ./models/whisper-small-ct2:/models/whisper-small-ct2:ro
    depends_on: [ fallback ]
    deploy:
      resources:
        reservations:
          devices:
            - capabilities: [gpu]
```

If Compose doesn't honor GPU on your setup, use the `docker run` commands above.

---

12) Quick API check from your workstation

```
# Fallback health
curl -s http://<fallback-host>:8082/v1/health | jq

# Primary health
curl -s http://<primary-host>:8081/v1/health | jq

# Transcribe (replace AUDIO.wav and API key)
curl -s -X POST http://<primary-host>:8081/v1/transcribe \
  -H "Authorization: Bearer primary_key_abc" \
  -F "audio=@AUDIO.wav" \
```

```
  -F "language=en" | jq
```

Recap

- **Two separate images** → `stt-primary` and `stt-fallback`.
- **Two separate containers** → can run on **different machines**; just set `FALLBACK_URL` in the primary's `.env` to point to the fallback.
- Use **bind-mounts** for configs and models; set `CONFIG_PATH` to the mounted file; pass `--gpus all`.
- Expose **8081** (primary) and **8082** (fallback); scrape `/metrics`; secure with Bearer keys and network ACLs.

If you want Helm charts for K8s GPUs or a makefile that builds/tags/pushes both images and starts them with the exact mounts/flags, say the word and I'll drop those in too.