Django Production-Readiness Assignment

Blue-green deploys • WebSockets • Observability

1. Context & Goal

We are rolling a **brand-new WebSocket service** into a Django-based stack.
Your task is to deliver a **self-contained repo** that:

- implements the WebSocket feature itself
- demonstrates you understand the ASGI concurrency model (event loop vs. thread-pool work)
- enables zero-downtime blue/green releases with Docker
- provides runtime observability: structured logs, metrics, health checks & alerts

The feature logic is intentionally simple (a message counter over WebSockets) so you can focus on production concerns.

2. Functional Requirements

Area	Requirement
WebSocket endpoint	`/ws/chat/` accepts a text message; replies with `{"count": n}` where `n` is the number of messages seen on that connection. On close, send `{"bye": true, "total": n}` and clean up.
Server-side push	Every 30 s broadcast a heartbeat `{"ts": <iso-timestamp>}` to all active sockets.

Area	Requirement
Graceful shutdown	When the container receives `SIGTERM`, finish in-flight messages, close sockets with code 1001, and exit within 10 s.
Reconnection (nice-to-have)	A reconnecting client may include its previous session UUID as a query param; resume the counter if still in memory.

3. Production-grade Requirements

Theme	Must-have	Nice-to-have / Suggestions
Blue-green	Two independent colour stacks (`app_blue`, `app_green`) behind a reverse proxy. A promotion script (bash or Make target) should: 1) build & start the next colour, 2) run smoke tests, 3) flip traffic, 4) retire the old colour.	Use Docker Compose with Nginx or Traefik as the router. If you prefer k8s, a single-node Kind cluster is fine.
Concurrency tuning	Show how many ASGI workers and thread-pool workers you run, and why (CPU-bound vs I/O). Provide a comment in `docker-compose.yml` or `helm/values.yaml`.	Use `uvicorn --workers`, `--loop`, and Django Channels' `ASYNC_CAPABLE` docs.

Theme	Must-have	Nice-to-have / Suggestions
Observability	**Metrics** endpoint `/metrics` exposing Prometheus-style counters: total messages, active connections, error count, shutdown time.	

Structured logs (JSON) for each event with `request_id`.
Health probes: `/healthz` (liveness) + `/readyz` (readiness toggled on startup/shutdown). | Wire Grafana dashboards via a pre-built `grafana/dashboards` JSON; or ship logs to Loki. | |
Alerting rule | Create one Prometheus rule (YAML) that triggers if active connections drop to 0 for >60 s. | | **Monitoring script** | A bash/py script `monitor.sh` that: 1) tails container logs for `ERROR`, 2) hits `/metrics` and prints the top-5 counters every 10 s. | | **CI task** | GitHub Actions or GitLab CI pipeline that: build-&-test, run `pytest`, execute the monitor script for 20 s, and archive logs/ artifacts. |

4. Non-Functional Targets

- **Throughput**: Demonstrate ≥ 5 000 concurrent sockets on your laptop.
- **Startup** < 3 s; **shutdown** < 10 s.

5. Deliverables

```
.
├─ docker/
│   ├─ Dockerfile
│   └─ compose.yml
├─ app/
│   ├─ manage.py
```

```
│   ├─ settings.py
│   └─ chat/
├─ scripts/
│   ├─ promote.sh
│   └─ monitor.sh
├─ ci/
└─ docs/
    ├─ DESIGN.md
    └─ OBSERVABILITY.md
```

README must include:

- the exact one-liner to spin up the full stack
- how to run the simple Locust/Autobahn load test (`make load-test`)
- how to flip blue→green

Optionally, add a short 5-minute Loom or GIF walking through the deploy & metrics.

6. Suggested Building Blocks (optional)

Concern	**Python / Django ecosystem pick**
ASGI server	`uvicorn[standard]` or `daphne`
WebSockets	`django-channels` 4.x
Thread-pool offload	`asgiref.sync.sync_to_async` & `ThreadPoolExecutor`
Metrics	`prometheus-client` or `django-prometheus`
Structured logs	`python-json-logger` or `structlog`
Graceful shutdown	`lifespan` hooks in Uvicorn, or Django's `request_finished` signal
Smoke tests	`pytest-asyncio` , `websockets` library

Concern	Python / Django ecosystem pick
Load test	`Locust`, `k6`, or a minimal `asyncio` script
Reverse proxy	`nginx:alpine`, `traefik:v3`

Feel free to swap for equivalents you know well; just document the choice.

7. Evaluation Checklist

Weight	Criterion	Details
40 %	**Production architecture**	Correct blue/green flow, health probes, graceful shutdown, Docker layering, config separation.
25 %	**ASGI & concurrency insight**	Proper use of workers vs thread-pool, no shared mutable state pitfalls, clear explanation in DESIGN.md.
15 %	**Observability**	Metrics coverage, log quality, alert rule validity, demo dashboards.
10 %	**Code quality & tests**	Idiomatic Django, typing where useful, pytest coverage.
10 %	**Docs & Dev-X**	`make dev-up` runs in ≤ 3 commands, README clarity, comments on trade-offs.

Bonus: reconnection support.

8. Submission

Create a repo, push your code, and email **Yugesh** (`yugesh.kothari@sigiq.ai`) or WhatsApp **+41 762621416** when ready. We'll schedule a review call to walk through your solution and discuss technical decisions.