Tribunal Session Summary

Session: conclave-20260805-154324-for-a-solo-developer-bc71

Depth: T6

Advocates: 6 | Judges: 4 | Cost: $11.5448 | Time: 40m 23s

THE PROMPT: For a solo developer running a dozen small always-on services (trading bots, monitors, web apps) on a single Linux box, is systemd or Docker Compose the better management approach? Recommend one. PRE-FLIGHT ASSUMPTIONS (the asker did not specify these; a gate chose reasonable defaults — treat them as revisable constraints and flag any that prove load-bearing): - How are the services currently packaged and deployed? Assumed: 3. A substantial mix of host-native and containerized services. - What level of dependency and isolation separation do the services require? Assumed: 2. Moderate separation; some services have conflicting runtimes or dependencies.

THE RULING: Docker Compose is the better primary management approach for this scenario. The recommended architecture is a hybrid model where Compose serves as the operational surface for application services, while systemd handles boot-ordering and a closed, explicitly defined list of permanently host-native services.

The Prompt

For a solo developer running a dozen small always-on services (trading bots, monitors, web apps) on a single Linux box, is systemd or Docker Compose the better management approach? Recommend one. PRE-FLIGHT ASSUMPTIONS (the asker did not specify these; a gate chose reasonable defaults — treat them as revisable constraints and flag any that prove load-bearing): - How are the services currently packaged and deployed? Assumed: 3. A substantial mix of host-native and containerized services. - What level of dependency and isolation separation do the services require? Assumed: 2. Moderate separation; some services have conflicting runtimes or dependencies.

The Ruling

Docker Compose is the better primary management approach for this scenario. The recommended architecture is a hybrid model where Compose serves as the operational surface for application services, while systemd handles boot-ordering and a closed, explicitly defined list of permanently host-native services.

This recommendation is built on three key pillars: (1) Compose's superior dependency isolation for conflicting runtimes, (2) its declarative management of multi-service applications, and (3) the ability to unify the operational surface while maintaining systemd's host-level reliability. The approach specifically addresses the "substantial mix" assumption by providing a clear migration path for host-native services while acknowledging that certain host-coupled services (like VPNs and monitoring agents) should remain native.

Core Analysis

Architecture Overview

The recommended architecture consists of three key components:

1. **Docker Compose as Primary Operational Surface** - Manages application services through declarative YAML files - Groups services by "blast radius" (e.g., related trading bots and their dependencies) - Provides unified commands for deployment, logging, and rollback
2. **Systemd for Host-Level Integration** - Thin wrapper templates (`compose@.service`) for boot sequencing - Manages the Docker daemon and host-coupled services - Handles host-level dependencies (network, mounts, VPN interfaces)
3. **Closed Native Service List** - `tailscaled`/`wg-quick` (VPN connectivity) - `sshd`/`fail2ban` (host access) - `node_exporter` (host metrics) - `restic` backup timer - Docker image prune timer - `unattended-upgrades`

Key Mechanisms and Evidence

Dependency Isolation

- **Problem**: The "moderate separation" requirement implies conflicting runtimes (e.g., Python 3.8 vs 3.12, different library versions)
- **Solution**: Docker containers encapsulate dependencies at the service level
- **Evidence**:
  - Systemd cannot isolate library/runtime versions for native processes
  - Virtual environments (venv) break during host OS upgrades (e.g., Ubuntu 22.04 → 24.04)
  - Containerized services continue running unaffected by host library changes

Rollback Asymmetry

- **Container Rollback**: ~5 seconds via `git checkout HEAD~1 && docker compose up -d`
  - Requires pinned image tags (e.g., `my-bot:v42` not `latest`)
  - Bit-identical restoration when using explicit image retention
- **Native Rollback**: 30+ minutes per service
  - Requires rebuilding virtual environments
  - Network-dependent package resolution
  - Risk of dependency version conflicts

Operational Simplicity

- **Single Interface**: `docker compose` commands for all application services
- **Unified Logging**: `docker compose logs -f` across related services
- **Health Monitoring**: Built-in health checks with `depends_on: condition: service_healthy`
- **Resource Management**: `deploy.resources.limits` in Compose files

Critical Hardening Measures

1. **Daemon Reliability** - Enable `live-restore` in `/etc/docker/daemon.json` - Add systemd drop-in to prevent daemon abandonment: `ini [Service] StartLimitIntervalSec=0 `
2. **Log Management** - Configure journald driver in `daemon.json`: `json { "log-driver": "journald", "log-opts": { "tag": "{{.Name}}" } } `
3. **Graceful Shutdown** - Set `stop_grace_period: 30s` for trading bots - Ensures proper order cancellation on SIGTERM
4. **Secrets Security** - Mount secrets as read-only files: `yaml services: trading_bot: volumes: - /srv/secrets/bot.env:/run/secrets/env:ro ` - Avoid environment variables (visible via `docker inspect`)

Migration Strategy

1. **Quantified Effort** - Simple services: 1-3 hours each - Services with native extensions: 2-4 hours each - Total for 6 services: 12-18 hours

2. **Prioritization** - Migrate drift-prone services first (Python/Node with complex dependencies) - Validate with 48-hour parallel dry-run - Keep old systemd units disabled but present during transition
3. **Validation Steps** - Test graceful shutdown behavior - Verify log aggregation - Confirm health check functionality - Document rollback procedure

Falsification Conditions

The recommendation flips to **systemd-primary** if:

- All 12 services share identical runtime requirements
- No dependency conflicts exist between services
- The "substantial mix" assumption proves false

How We Got Here

Scorecard

Judge	Claude Opus 5	GPT-5.6 Sol	Grok 4.5	Magistral Medium	Gemini 3.1 Pro	DeepSeek V4 Pro	Winner
DeepCogito v2.1	ACCEPT	REJECT	REJECT	REJECT	REJECT	REJECT	Claude Opus 5
GPT-OSS 120B	ACCEPT	ADOPT	ADOPT	REJECT	REJECT	REJECT	Claude Opus 5
Nemotron 3 Ultra 550B	ACCEPT	ADOPT	ADOPT	REJECT	REJECT	ADOPT	Claude Opus 5
Kimi K2.6	ACCEPT	ADOPT	REJECT	REJECT	REJECT	REJECT	Claude Opus 5

Convergence Assessment

The deliberation showed strong epistemic convergence, with advocates refining positions based on technical evidence rather than social pressure. Key indicators:

1. **Evidence-Driven Shifts** - All advocates abandoned flawed positions when presented with technical counter-evidence - Multiple advocates independently discovered the same critical issues (e.g., docker.service StartLimitBurst)
2. **Artifact Refinement** - Migration cost estimates increased 25x (from 30 minutes to 12-18 hours) as advocates incorporated real-world complexity - Initial oversimplifications (single docker-compose.yml) evolved into blast-radius grouped designs

3. **Technical Depth** - Advocates engaged with low-level mechanisms (FIFO buffers, dynamic linking, cgroups) - Positions became more nuanced rather than more polarized

Closing Statements

Claude Opus 5:

- Emphasized risk management over raw feature comparison
- Quantified both migration costs and risks of inaction
- Provided explicit falsification conditions
- Strongest on operational realities and failure modes

GPT-5.6 Sol:

- Advocated systemd-primary with three-substrate model
- Strong on supervision semantics but weaker on migration costs
- Conceded systemd's default restart limitations
- Most rigorous on host-level integration

Grok 4.5:

- Proposed Quadlet as convergence path
- Strong on networking trade-offs but vague on migration costs
- Acknowledged live-restore mitigation
- Best on container runtime alternatives

Key Moments

- **The Alpine Bind-Mount Debacle (Round 1)**
- Magistral Medium's initial example of bind-mounting host binaries into Alpine containers was challenged due to glibc/musl incompatibility
- Forced all advocates to adopt proper containerization standards (COPY not bind-mounts, glibc-compatible bases)
- **Discovery of Docker Daemon Abandonment Risk (Round 6)**
- Claude Opus 5 identified that the vendor `docker.service` unit includes `StartLimitBurst=3`
- Could lead to permanent daemon failure after crashes
- Led to mandatory `StartLimitIntervalSec=0` drop-in
- **The Ubuntu 22.04 EOL Revelation (Round 6)**
- Claude Opus 5 dated the Ubuntu 22.04 standard support end (April 2027)
- Provided concrete timeline for Python 3.10 venv breakage
- Transformed abstract risk into scheduled event
- **The FIFO Back-Pressure Mechanism (Round 7)**
- Claude Opus 5 identified that container stdout writes block when daemon is down

- Could wedge trading bots during extended outages
- Strengthened case for journald log driver

Reality Checks

Claim	Challenge	Result
Magistral Medium: Containerizing host-native services is trivial (30 min/service)	**Claude Opus 5:** Actual migration requires 1-3h for simple services, 6-12h for host-coupled	Conceded; estimate revised to 12-18h total
Gemini 3.1 Pro: Systemd wrapper provides "iron-clad supervision"	**GPT-5.6 Sol:** Type=oneshot cannot supervise container runtime state	Full concession; position abandoned
DeepSeek V4 Pro: Single docker-compose.yml simplifies management	**Claude Opus 5:** Monolithic file creates coupled blast radius	Conceded; adopted blast-radius grouping
Grok 4.5: Docker daemon crashes take down all containers	**Claude Opus 5:** live-restore keeps containers running across daemon restarts	Position revised; risk downgraded

Next Steps

Next Step	Why It Matters	Confidence
Inventory current services	Determines migration effort and falsification condition	100%
Test rollback procedure	Validates bit-identical restoration capability	90%
Benchmark network latency	Confirms container networking overhead is acceptable	85%
Implement secret rotation	Ensures secure credential management	80%
Schedule migration windows	Minimizes downtime during transition	75%

Dissenting Opinions

No dissenting opinions were filed in this session.

Glossary

Term	Definition
Blast Radius	The set of services affected by a failure or restart event
FIFO Back-Pressure	Blocking of container writes when log buffers fill during daemon outages
Live-Restore	Docker feature that keeps containers running during daemon restarts
Rollback Asymmetry	The ratio of reverse-deploy to forward-deploy effort (lower is better)
SPOF	Single Point of Failure - a component whose failure causes system-wide outage
Systemd Drop-In	A configuration file that overrides specific unit settings without modifying the original unit file

How The Conclave Works

The Conclave is a structured multi-model deliberation system. Instead of asking one AI model a question and trusting its answer, the Conclave convenes a panel of independent models, forces them to argue, and subjects their conclusions to judicial review. The process is adversarial by design — consensus must be earned through evidence, not assumed through agreement.

Advocates are independent AI models (e.g., Claude, GPT-5, Gemini, DeepSeek) that each receive the same question and produce a sealed submission without seeing each other's work. Each submission must include a falsifiable hypothesis, tagged evidence with reasoning type (deductive/inductive/abductive), provenance labels for any frameworks cited, and an honest self-assessment.

Challenges follow submissions. Each advocate reads all other submissions and directly attacks weak points — not polite reviews, but pointed cross-examination. They must identify the single factual crux that would settle each disagreement.

Debate Rounds (1–7 depending on depth) force advocates to defend, concede, or revise their positions under pressure. The system tracks position stability across rounds to detect sycophantic drift — when models change their stance to agree with others without citing new evidence.

Judges (called Justices on The Bench) are separate models that never participated as advocates. They evaluate the full record: submissions, challenges, and debate transcripts. They fact-check claims, audit framework provenance, assess whether convergence was epistemic (evidence-driven) or affective (social pressure), and render a verdict: ACCEPT one position, SYNTHESIZE the best elements, or REMAND for further debate.

Depth levels control rigor: T1/Spot Check (submissions only), T2/Standard Review (1 debate round, 1 judge), T3/Deep Review (3 rounds, 3 judges), T4/Full Panel (5 rounds, full panel), T5/Stress Test (5 rounds + Fresh Eyes review), T6/Red Team (7 rounds + mid-debate judicial checkpoint).

The system is deterministic code — it cannot be sycophantic. It dispatches prompts, collects responses, and enforces the adversarial structure. The models argue; the code referees.