BattleArena v2.0 - Scale & Reliability

From Local Development to Production Infrastructure

Version: 2.0 — Infrastructure Evolution
Status: Production-Ready Platform
Timeline: Month 3–4

The Core Idea

"We need our game to handle 10,000 players without crashing. If something breaks, it should fix itself automatically."

v0 validated the concept. Popularity created load:

- 1,000+ DAU, $10,000+ MRR, 20% WoW growth

Current pain at 500 concurrent logins:

- API crashes and restarts
- Database overwhelmed, matchmaking freezes
- Single point of failure, manual 3 AM restarts
- No zero-downtime deploys

Solution: Production-grade orchestration with Kubernetes.

What Problem Are We Solving?

The Growth Problem

Monday 50 players: 50–100 ms, CPU 15%.
Saturday 1,000 players: API crashes, 5–10 s latency, CPU 100%, revenue loss.

We need to:

- Run multiple API replicas with load balancing
- Self-heal on crashes
- Auto-scale by traffic, scale down off-peak
- Deploy with zero downtime

Stakeholders

1. Gamers (End Users) — New Expectations

Needs: 99.9% uptime, fast responses at peak, no maintenance windows, safe progress/purchases.
Do not care: Tools or server counts. They want uninterrupted play.

2. Game Company (Us) — New Challenges

Team of 8, $50K MRR, $5K infra spend, raising Series A.
Needs: Auto-scale, self-heal, zero-downtime deploys, cost optimization, monitoring/alerting.

3. Operations Team (New Role)

Needs: Automated recovery, visibility, safe deployments, horizontal scaling, predictability.

4. Investors (New Stakeholder)

Ask: Scale to 100k, uptime, spike handling, DR plan.
Want: 99.9% uptime, autoscaling proof, clear roadmap.

Reliability Story

Saturday Night *Before* Kubernetes

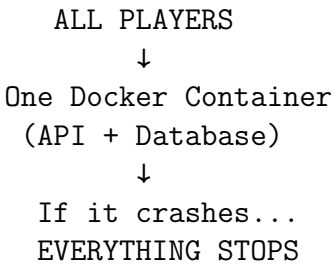
- 1,200 online, 8 s latency, API crash, global disconnects
- Manual recovery in minutes, lost revenue, reputation hit

Saturday Night *After* Kubernetes

- Autoscale from 3 to 6 API pods, <200 ms latency
- One pod crashes, K8s replaces it in seconds, no user impact

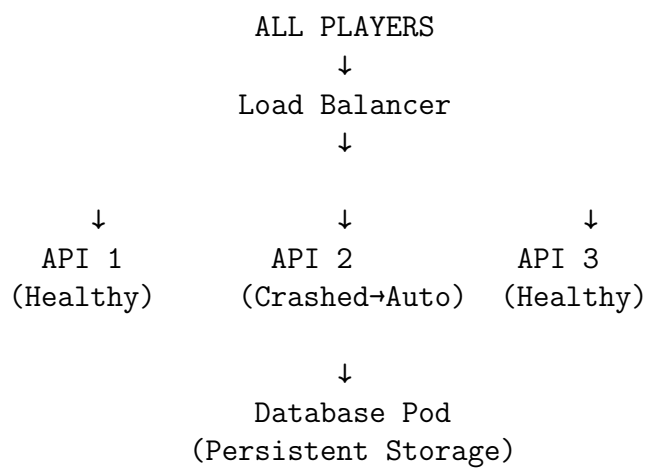
System Evolution

v0 (Docker Compose) — Single Point of Failure



Issues: No redundancy, manual restarts, no spike handling, deploys cause downtime.

v2 (Kubernetes) — Self-Healing



Benefits: Redundancy, self-heal, autoscale, rolling updates, persistent DB.

Key Concepts

Load Balancer: Routes to healthy pods, masks failures.
Self-Healing: Health checks, automatic restart/replace.
Auto-Scaling: Scale on CPU/memory/requests. Scale-down off-peak.

What v2.0 Delivers

For Players

99.9% availability, stable latency at peak, reliable purchases, seamless updates.

For Company

Self-heal, zero-downtime deploys, effortless scale, elastic cost, investor-ready metrics.

For Operations

Automated recovery, real-time dashboards/alerts, rolling updates + instant rollback, predictable infra.

Success Metrics for v2.0

Metric	Before (v0)	After (v2.0)	Improvement
Uptime	95%	99.9%	5× better
Deploy Frequency	1/wk	10/day	10× faster
Recovery Time	5–10 min	10–30 s	20× faster
Peak Capacity	500 players	10,000 players	20× scale
Infra Cost	\$5K fixed	\$3K–\$7K elastic	∼40% savings
Incident Response	Paged human	Self-healing	Quality of life

Economics of Kubernetes

Infra Costs

Before: One peak-sized server, \$5,000/mo, mostly idle.
After: Elastic replicas:

- Off-peak: $\tilde{1}$0 replicas, $\tilde{\$}$2,000/mo
- Normal: $\tilde{3}$0 replicas, $\tilde{\$}$4,000/mo
- Peak: $\tilde{5}$0 replicas, $\tilde{\$}$7,000/mo
- Average: $\tilde{\$}$3,500/mo

Hidden Savings

- Downtime cut from 5% to 0.1% $\Rightarrow$ $\tilde{\$}$39.2K/yr saved
- Dev productivity regained: $\tilde{6}$ dev-days/mo
- Support load reduced: $\tilde{\$}$50K/yr equivalent

Why Kubernetes

Not Just More Servers

Traditional: Manual build/scale/failover.
Kubernetes: Declare desired state; controller maintains it.

Desired State Philosophy

`‘‘Run 3 healthy API servers v2.0.’’`
K8s creates, heals, scales, and rolls updates to match.

Day in the Life

Monday 2 AM

50 players, 10 API pods, cost $\tilde{\$}$80/h, ops asleep.

Monday 3 PM

300 players, 30 API pods, $\tilde{\$}$150/h, 80 ms latency.

Monday 8 PM

1,000 players, 50 API pods, $\tilde{\$}$250/h, 120 ms latency.

8:23 PM Incident

One pod crashes; auto-recovered in $<$40 s; no user impact.

Tuesday 10 AM Deploy

Rolling update across 30 pods, 10 min total, zero downtime, instant rollback ready.

Definition of Done

Infrastructure

- 3+ API pods behind load balancer
- Persistent DB storage
- Liveness/readiness probes; self-heal
- HPA autoscaling on CPU

Operations

- Proven zero-downtime deploys
- One-command deploy; tested rollback
- Documented DR plan

Reliability

- 99.9% uptime over 30 days
- 2,000 concurrent players sustained
- Pod crash has zero player impact
- 5 deploys/day without downtime

Business

- 30% lower infra cost than v0
- 90% fewer downtime tickets
- Investor-ready metrics dashboard
- Team confidence in stability

Core Philosophy

Build infrastructure that grows with you.
Reliability, scalability, maintainability, cost-effectiveness, investor confidence.

TL;DR

v2.0 moves from a fragile single server to Kubernetes. Self-heals on failure, auto-scales on spikes, and ships updates with zero downtime. Players get 99.9% uptime. Costs drop ∼30–40%. Ops sleeps.