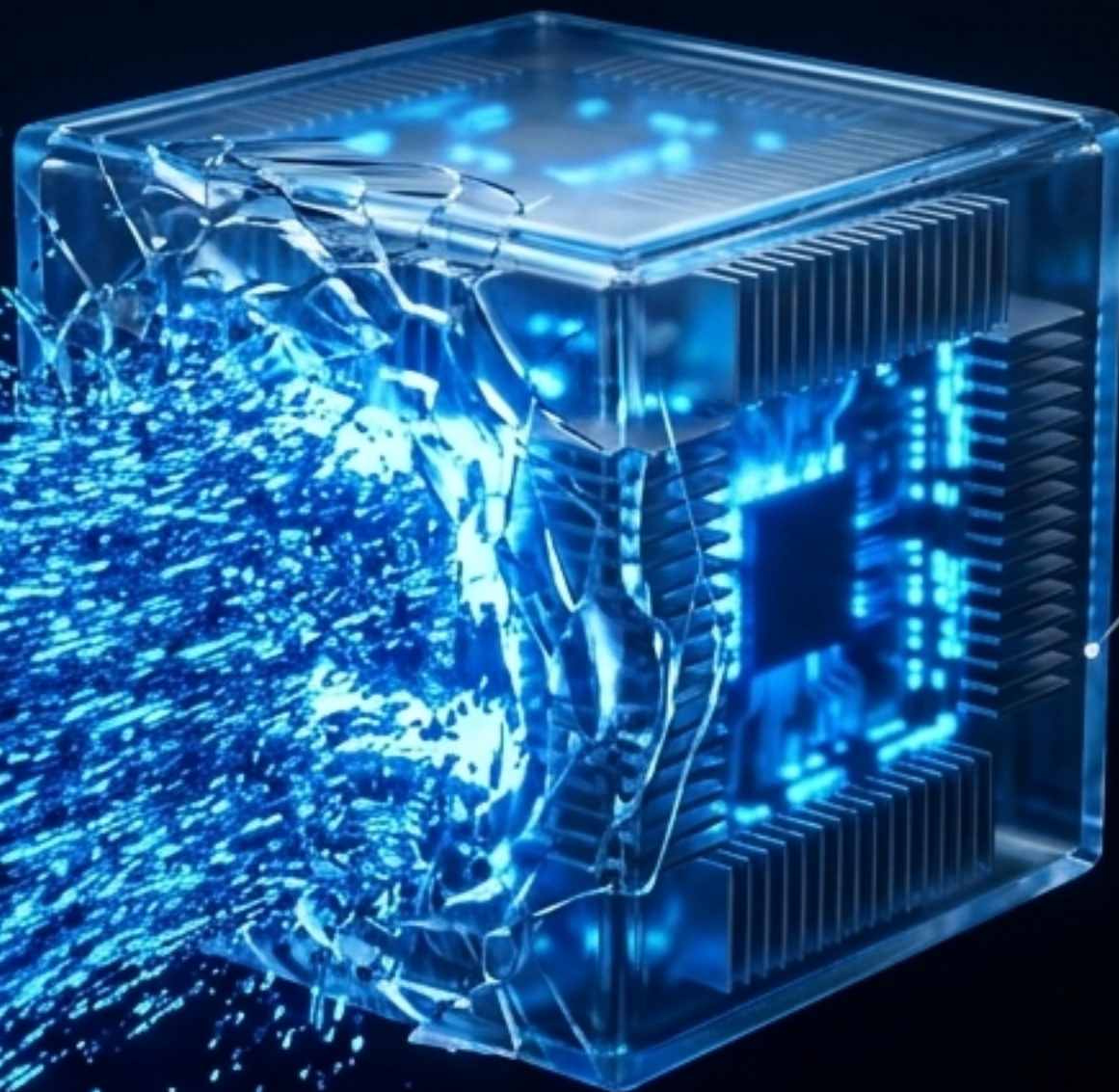




جاودان
JAVDAN

وقتی Docker دیگر کافی
نیست...

همه چیز داخل یک Docker Container
عالی عالی اجرا می شود. اما ناگهان...



50,000

کاربر همزمان وارد سایت
می شوند.
یک Container دیگر
جوابگو نیست.

نقطه بحران



۳. اگر یک سرور خاموش
جمله‌های جدید را بسازد؟



۳. اگر یک سرور خاموش شد چه
کسی برنامه را منتقل کند؟



۱. چه کسی باید Containerهای
چه کسی دوباره آن را اجرا کند؟

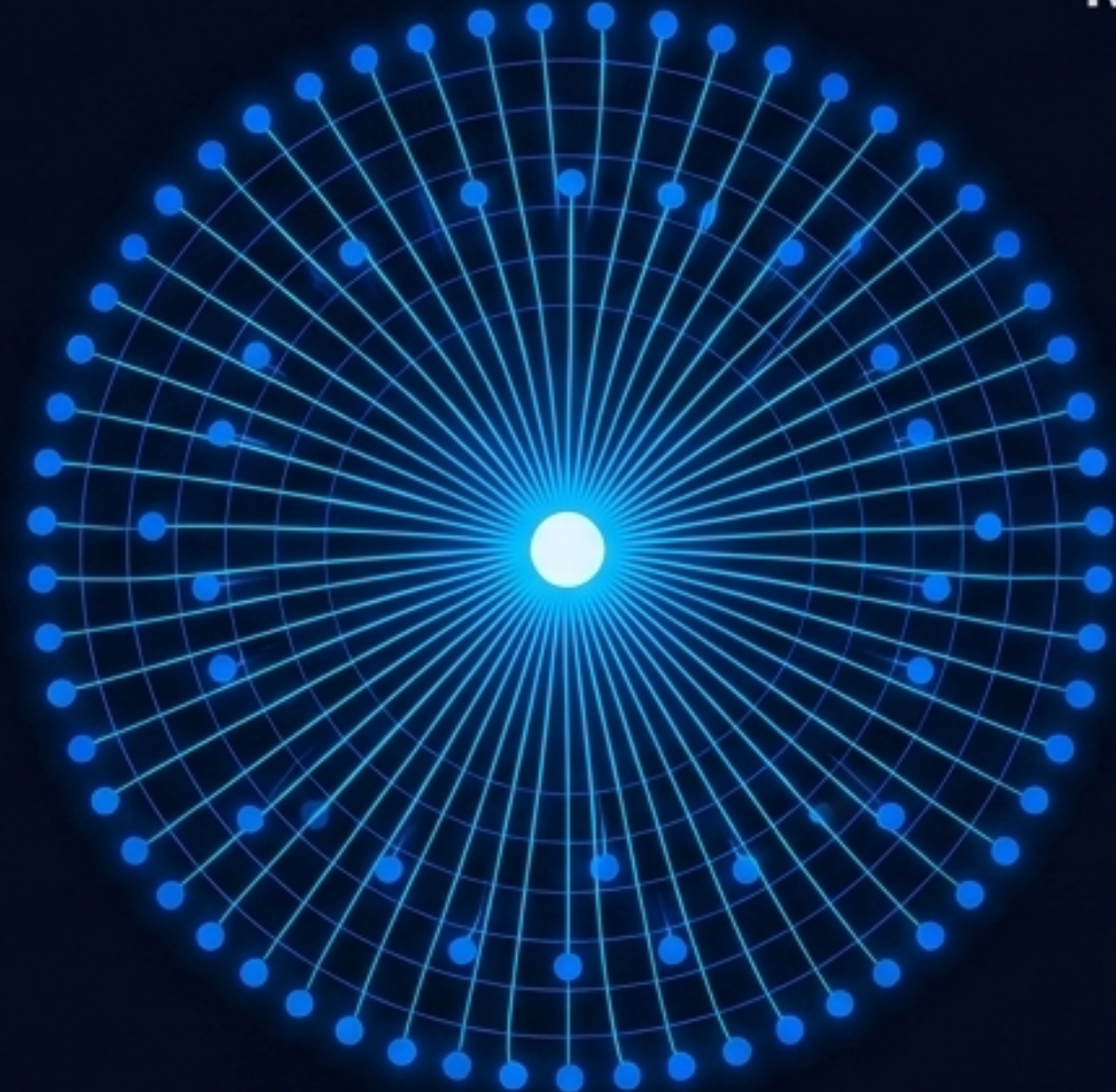
تولد Orchestration

فرض کنید یک ارکستر ۱۰۰ نفره دارید. بدون رهبر ارکستر، فقط سر و صدای بزرگ تولید می‌شود.

Noise



Music



نوازنده‌ها Docker هستند. رهبر ارکستر Kubernetes و Docker Swarm هستند.



مقیاس واقعی زیرساخت

50

سروریس

300

کانتینر

20

سرور

مدیریت دستی چنین سیستمی تقریباً غیرممکن است.
Orchestrator این مدیریت را خودکار می‌کند.

معرفی Kubernetes

مدیر هوشمند کانتینرها

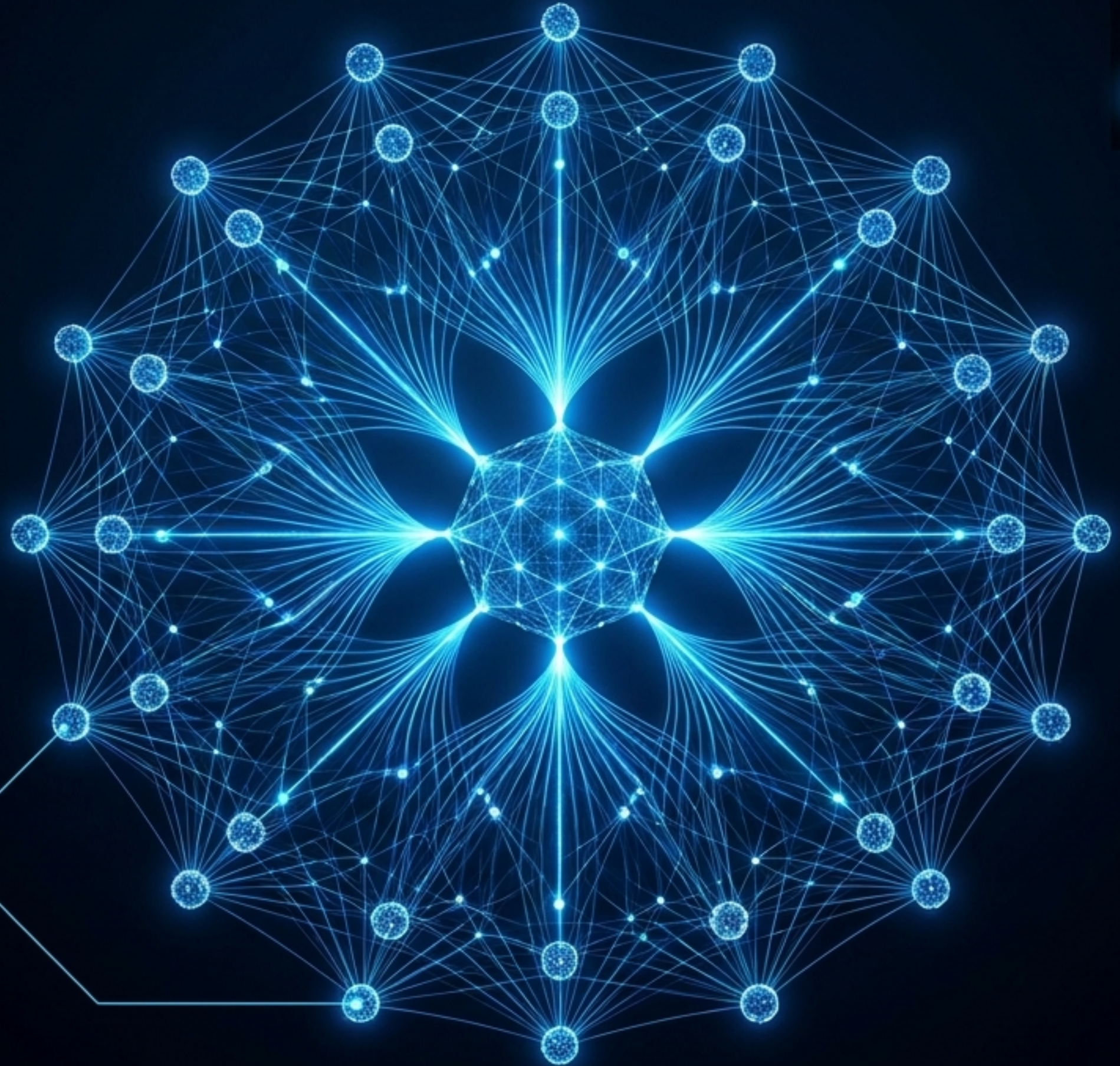
“

شما فقط دستور می‌دهید:
«می‌خواهم همیشه ۵ نسخه از
از این برنامه اجرا شود.»

”

■ نسخه‌های خراب را جایگزین می‌کند.

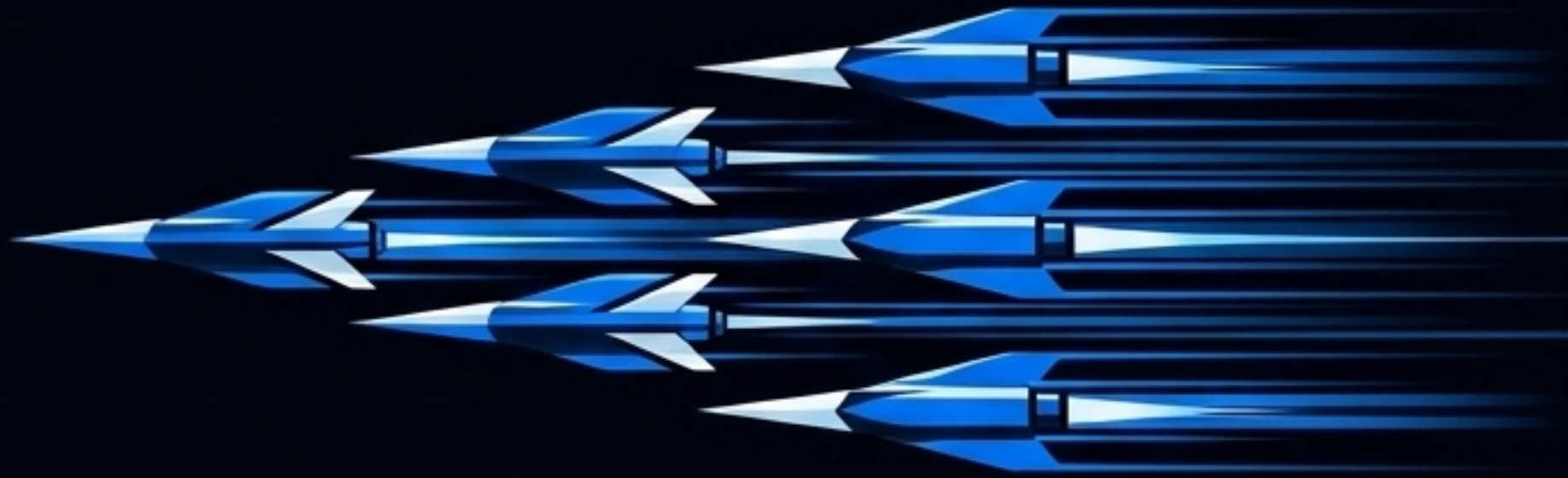
■ برنامه را به سرورهای سالم منتقل می‌کند.





معرفی Docker Swarm جایگزین سبک و سریع

دقیقاً برای همان هدف ساخته شده، اما ساده‌تر.



یادگیری راحت‌تر مانی کوچتر

راه‌اندازی سریع‌تر

ایده‌آل برای پروژه‌های کوچک

هسته مشترک

تفاوت در اصل کار نیست، در امکانات و مقیاس است. هر دو انجام می‌دهند:



اجرای چندین کانتینر



توزیع بار بین سرورها (Load Balancing)

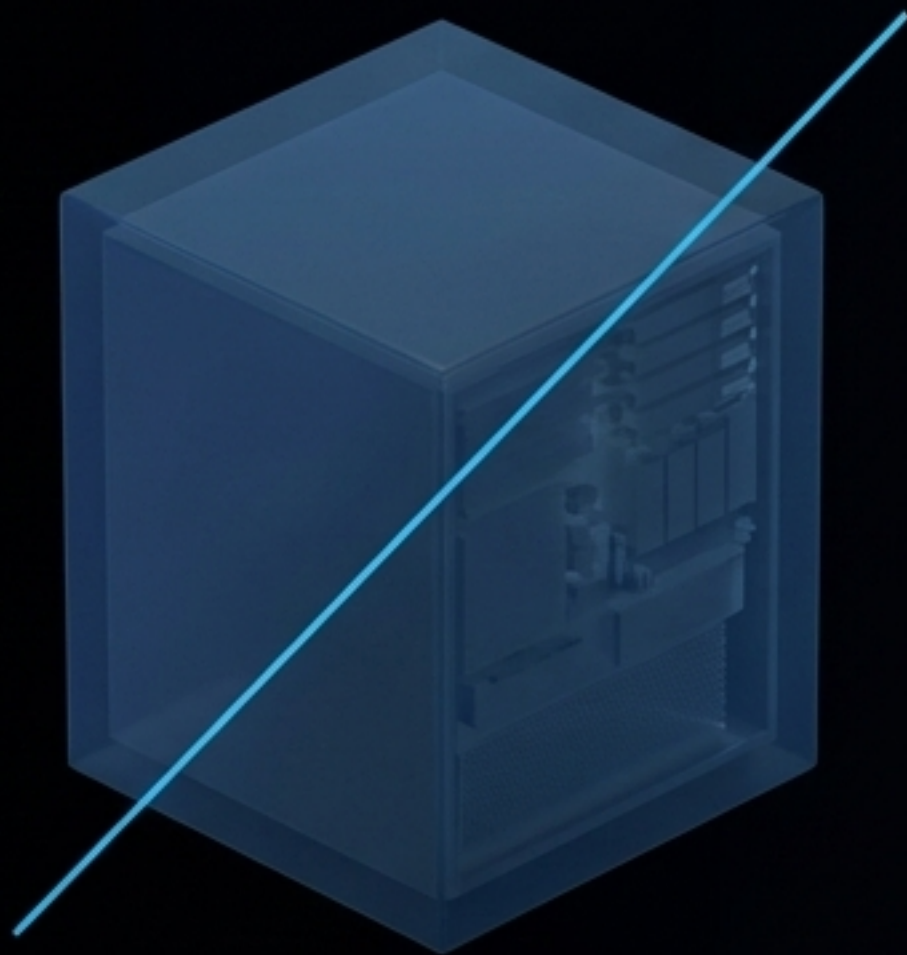


راه‌اندازی خودکار نسخه‌های خراب



مقیاس‌پذیری و به‌روزرسانی

مکانیزم اول: Horizontal Scaling

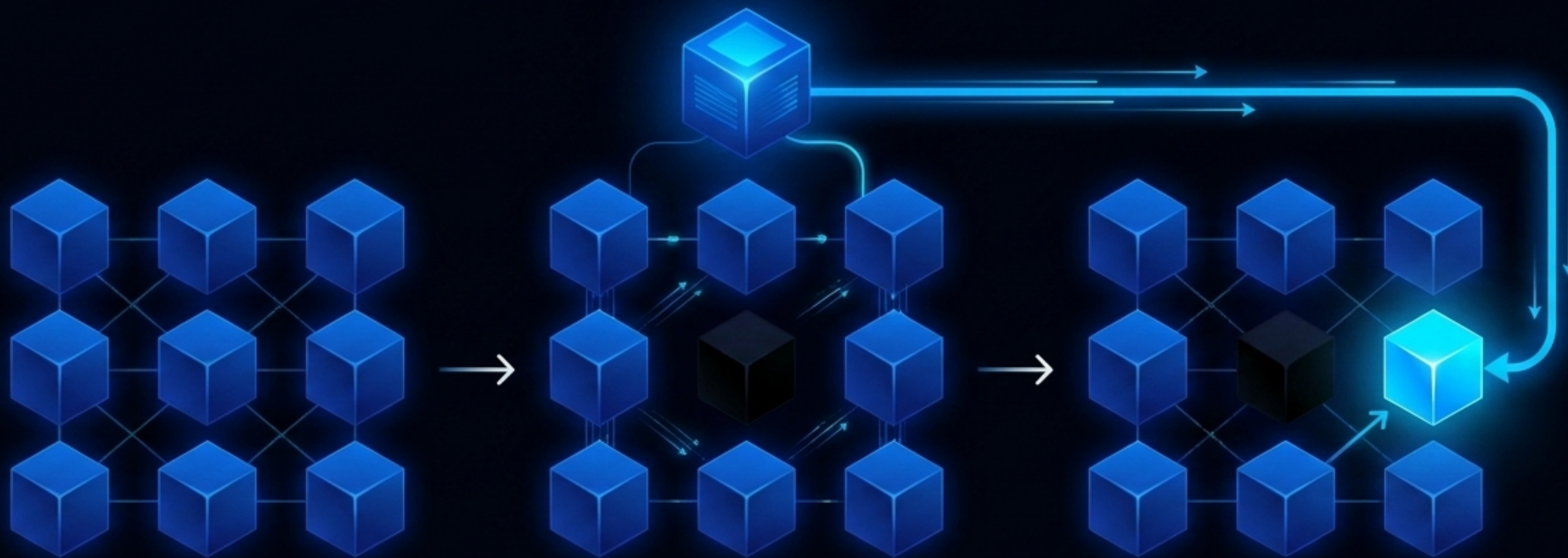


به جای خرید یک سرور غول‌پیکر، Orchestrator چندین نسخه از برنامه را روی چندین سرور ارزان‌تر اجرا می‌کند.

آماده برای رشد از ۱۰۰ کاربر به ۱۰۰,۰۰۰ کاربر در چند ثانیه.

مکانیزم دوم: Self-Healing

اگر یک سرور خاموش شود، کاربر حتی متوجه خرابی نمی‌شود.



تشخیص خرابی → توقف ترافیک به نود خراب → اجرای فوری روی سرور سالم

مکانیزم سوم: Load Balancing

فرض کنید ۱۰ نسخه از برنامه در حال اجراست. اگر همه کاربران به یک نسخه وصل شوند، آن نسخه از کار می‌افتد.



Orchestrator درخواست‌ها را بین همه کانتینرها به زیبایی تقسیم می‌کند تا فشار پخش شود.

مکانیزم چهارم: Rolling Updates

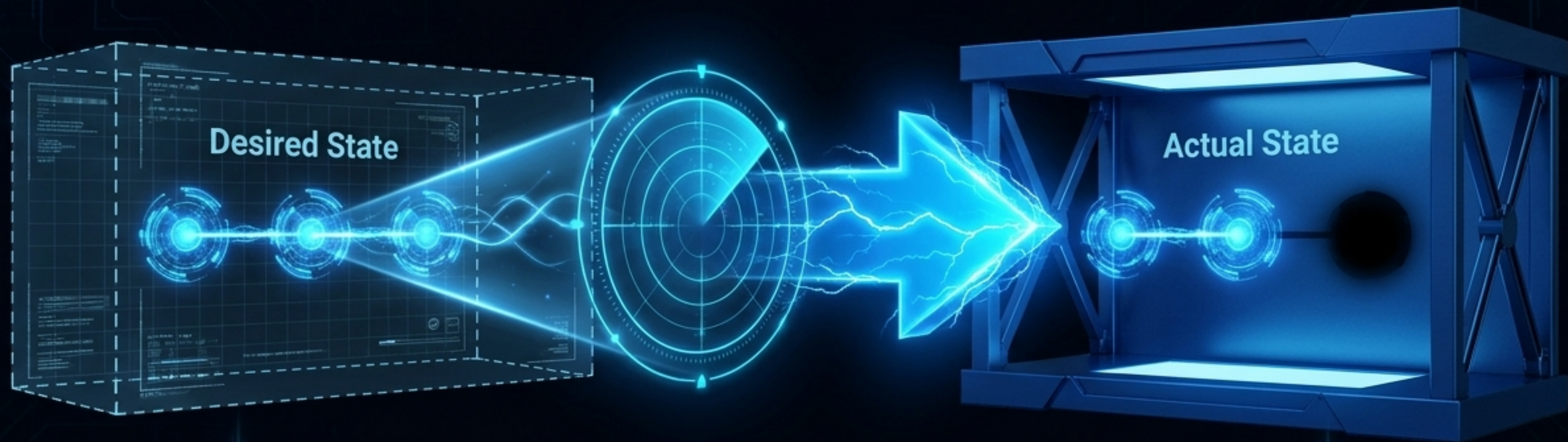
پایان دوران خاموش کردن سایت برای آپدیت.



نسخه جدید کم کم جایگزین نسخه قدیمی می شود. کاربران هیچ قطعی احساس نمی کنند و در صورت بروز خطا، سیستم خودکار به نسخه قبل برمی گردد.

مغز متفکر Kubernetes

مفهوم Desired State



شما به Kubernetes نمی‌گویید «الان چه کاری انجام بده».
فقط وضعیت مطلوب را تعریف می‌کنید.

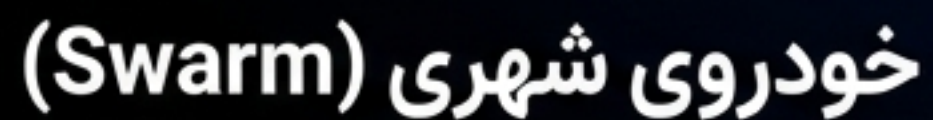
سیستم دائماً وضعیت واقعی را با وضعیت مطلوب مقایسه کرده
و اختلافات را به صورت خودکار اصلاح می‌کند.

مزیت رقابتی Docker Swarm

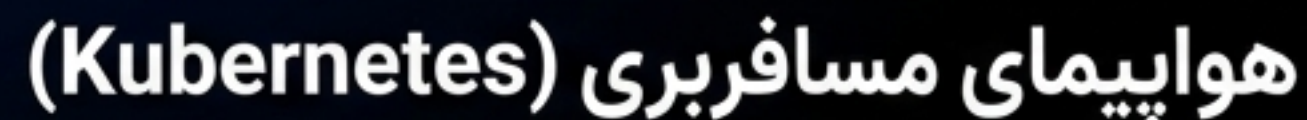
سادگی مطلق. چند دستور اجرا می‌کنید، سرورها عضو خوشه می‌شوند و سرویس‌ها اجرا می‌شوند.

```
docker swarm init  
docker service create --replicas 3 my-web-app
```

تیم‌های کوچک در کمتر از چند ساعت می‌توانند یک Swarm کامل را راه‌اندازی کنند.



ساده، سریع و کم‌دردسر.



هدایت سخت‌تر، اما ضروری برای زمانی که تعداد مسافران و مسیرها بسیار زیاد است.



هزینه واقعی Kubernetes

“تا وقتی واقعاً به Kubernetes نیاز ندارید، سراغ آن نروید.”

- Reddit & X

امکانات پیشرفته (Auto Scaling, Secrets Management, Network Policy) رایگان نیستند. هزینه آن‌ها پیچیدگی شدید و منحنی یادگیری تند است. ابزار بزرگ‌تر همیشه انتخاب بهتر نیست.



ماتریس تصمیم‌گیری

Docker Swarm	Kubernetes
تیم کوچک	صدها کانتینر
پروژه تازه شروع شده	تیم‌های متعدد
زیرساخت ساده	زیرساخت ابری
زمان کم برای یادگیری	مقیاس‌پذیری خودکار حیاتی (High Availability)
سادگی < امکانات	شرکت در حال رشد سریع

استراتژی مهاجرت بدون ریسک

تیم‌هایی که با Docker Compose یا Swarm شروع می‌کنند و پس از رشد واقعی به K8s مهاجرت می‌کنند، هزینه و ریسک بسیار کمتری را تجربه می‌کنند.



از انتخاب Kubernetes در ابتدای مسیر فقط به خاطر ترند بودن خودداری کنید.

Docker

Swarm

Kubernetes

قانون طلایی زیرساخت

“

«زیرساخت را برای نیاز امروزت انتخاب کن، مگر نه برای
رؤیای پنج سال بعد.»

”

Docker
اجرای کانتینر.



Swarm
سادگی و راه‌اندازی سریع برای تیم‌های کوچک.



Kubernetes
مقیاس‌پذیری بی‌نهایت برای سازمان‌های بزرگ.





جاودان
JAVDAN

جایی که عمران با هوش ملاقات می‌کند.