



adaptTo()

EUROPE'S LEADING AEM DEVELOPER CONFERENCE
28th – 30th SEPTEMBER 2026

Advancing AEM Elasticity

Earlier Signals, Faster Readiness, Better Resilience

Meet the Speakers



Dominik Süß

Senior SRE — Adobe



Tudor Rogoz

Senior Software Engineer — Adobe

The Vulnerability Window



Decision to scale

Vulnerable window

Capacity ready



Existing pods absorb everything until new capacity arrives

That moment is where incidents escalate into outages.

The Binding Constraint



CPU

Processing cycles



Memory

Object graphs, footprints



I/O

Latency, queries, calls

Whichever hits its limit first becomes the bottleneck.

How Scaling Relates to Constraints



Vertical Scaling (ScaleOps)

Right-sizes CPU/memory requests to match actual usage

**400m used, 800m requested
→ right-sized to 500m**



Horizontal Scaling (HPA / KEDA)

Distributes load across more pods when the constraint is distributable

**500 req ÷ 100/pod
→ scale to 5 pods**

Why Current Scaling Isn't Fast Enough

60-70%

Pattern A

Slow Accumulation

*Marketing campaign, +30% traffic
over 45 min*

20-25%

Pattern B

Rapid Spike

*CDN cache purge, origin traffic
spikes 10x in 90 seconds*

10-15%

Pattern C

Persistent Failure

*Background job monopolizes CPU on
every pod*

The Timeline Problem



Hidden infrastructure delay

Node provisioning added 30-60s in the past → fixed with Kubernetes overprovisioning (placeholder pods reserve capacity in advance)

The Two Dimensions of Elasticity



Signal Timing

Reactive CPU %, 30-60s lag
→ Leading indicators, 5-60 min lead



Readiness Speed

Median ~3.4 min to ready
~85% under 5 min; slow tail 10-20 min

**Earlier detection + a shorter path to ready
compound — each shrinks the same window**

Signal Quality Problem

HPA's signal: CPU % = actual usage ÷ CPU request

500m request

120%

HPA scales out



1000m request

60%

HPA stops scaling



Same 600m actual load — different outcome.

Why Signal Quality Matters

Use signals independent of request size



Absolute CPU usage

Tracks real consumption, not a ratio



Server load ratio

Direct indicator of constraint pressure



Response time P95

Leading signal of demand buildup



Queue depth

Work accumulation pattern

Real Anti-Patterns from Production



Serialized Repository Traversal

Blocks startup, triggers probe timeout



Expensive Alias Resolution

Hundreds resolved serially, 20+ min startup



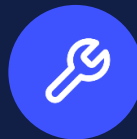
Index/Cache Lifecycle

Computed during or async after startup



Blocking External Calls

Unpredictable latency stalls every new pod



Inefficient Repoint

Per-user rules instead of group inheritance

The Startup Profile

Median ~3.4 min to ready · ~85% under 5 min · p90 ~6 min · p95 ~8 min · p99 11-16 min



Scheduling (seconds) + Init ~55s + Main container: JVM + OSGi → ready ~150s = ~3.4 min median

The main-container boot dominates — scheduling is not the bottleneck



Healthy environment

~2.5 min — consistently, every pod



Troubled environment

10-20 min — chronically, every pod

~10× spread — the slow tail is environment-specific and in your control, not random infrastructure noise

What You Can Optimize

STEP 1



Measure

Break down startup by phase,
find the slowest 20%



STEP 2



Move

Blocking operations
to async/background



STEP 3



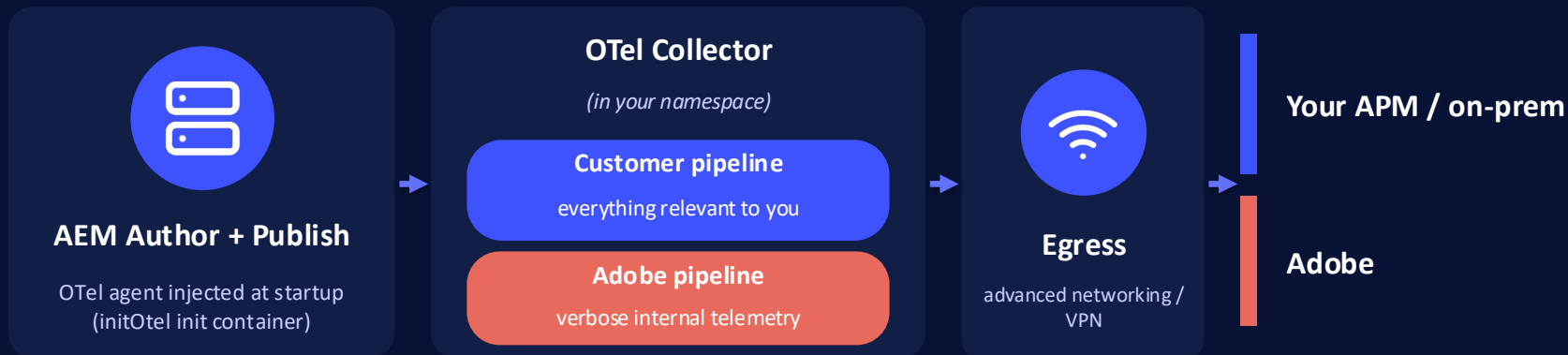
Replicate

Golden publish: pre-compute
once, replicate to all



Measure Startup with OpenTelemetry

Configure via Cloud Manager / Config Pipeline — OTLP endpoint · agent variants: OTel · Grafana Alloy · Splunk (Azure App Insights coming)



Track startup time & early-runtime bottlenecks — traces + JVM/OSGi metrics, in your own APM

Beta / Early Access — subject to change, not guaranteed to ship as-is

Starting Warm

Be ready to serve productive traffic the moment the pod is up — no warm-up window



Skip the warm-up

Don't depend on warm-up routines
- avoid live computation wherever you can.



Import, don't compute

Avoid synchronous backend calls but import



Indexes replicate warm

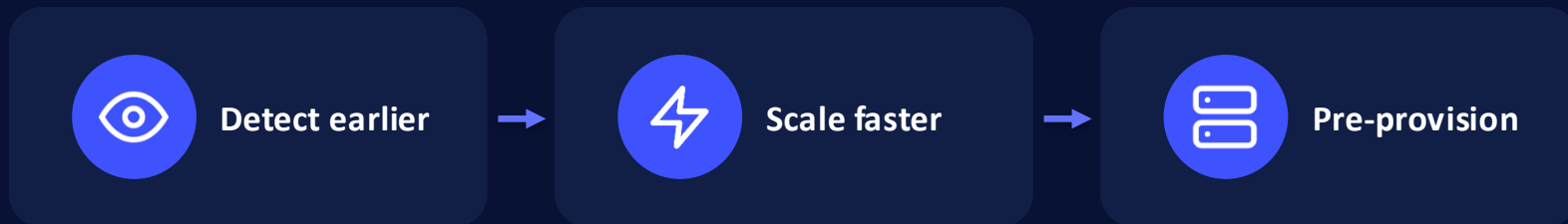
Tune queries and indexes



Let the CDN absorb spikes

Tune your CDN caching - a higher cache-hit ratio keeps
artificial spikes off the origin.

Bringing It Together



compound into

**Enough runway for a ~3.4-min median startup
(and the slow tail) to finish in time**

Key Takeaways



ADOBE takes care of scaling signals & mechanisms



YOU take care of startup timing and application that's fit for traffic

Call to Action



...in a perfect world

This Week

Measure baseline,
break down by phase

Next 2 Weeks

Move async, lazy-load,
remove external calls

Ongoing

Monitor P95/P99,
celebrate every second saved

Realistically? Longer. But every second you shave still shrinks the window — so start anyway.

Questions?

Let's discuss your elasticity challenges