WHITE PAPER

True Scale-to-Zero with the Intelligent Wake-Up Proxy: Event-Driven Serverless Kubernetes Powered by KEDA

Cutting Kubernetes TCO by turning always-on, idle containers into on-demand workloads with true scale-to-zero and intelligent wake-up.

About the solution
A Kubernetes-native pattern that combines true KEDA scale-to-zero, a dedicated API-driven wake-up proxy, sub-5-second cold starts, and a generic web-based activation catalogue, with zero vendor lock-in. Public references of similar end-to-end setups are extremely rare, and no off-the-shelf product currently implements this architecture.

Jan Affolter
Product Owner & Digital Solution Design
Crissier, Vaud, Switzerland
December 2025
contact@proxy-keda.com

Table of contents

1 EXECUTIVE SUMMARY

Many Kubernetes clusters waste a significant share of their compute budget keeping rarely used microservices running idle, even though they generate little or no business value between requests. This white paper presents a production-ready architecture for achieving true scale-to-zero in Kubernetes while preserving rapid service availability, full autoscaling capabilities, and a user experience that remains within psychologically acceptable wait times.

The solution combines KEDA (Kubernetes Event-Driven Autoscaling), a custom API-driven wake-up proxy, and intelligent activation mechanisms to shut down idle workloads completely, then bring them back online on demand. It is designed to integrate seamlessly into existing Kubernetes estates, portals, and ITSM tools, without introducing vendor lock-in or requiring a new serverless platform.

Key result: the reference implementation reduces memory usage by up to 100% for idle services while consistently achieving **1–5 second cold starts** through targeted caching and orchestrated pod initialization. Wake-up can be triggered either via a simple API call – enabling direct integration into portals and service catalogs such as ServiceNow – or via the built-in web-based application activation catalogue, ensuring that on-demand activation fits naturally into both automated workflows and human-driven service journeys.

2 The Problem: Idle Kubernetes Costs

2.1 The Idle Resource Paradox

Modern Kubernetes platforms make it easy to deploy hundreds of microservices, but most organizations still run them “always on”, even when traffic is sparse or unpredictable. This leads to clusters where a large share of memory and CPU is permanently allocated to idle containers that deliver no business value between requests.

Teams know this and sometimes try manual scale-down to zero, but that breaks autoscaling, introduces operational risk, and requires human intervention to bring services back online. Native HPA-only approaches are not enough either, because a minimum replica count of at least one still means paying for idle resources 24/7.

2.2 Resource Waste in Enterprise Environments

Across many enterprises, Kubernetes clusters run with a large proportion of their capacity permanently allocated but rarely used. Industry analyses of real-world environments report average clusters running at only **13–25% CPU utilization and 18–35% memory utilization**, meaning **65–80% of provisioned compute and memory** often sits idle.

Studies and FinOps case studies further show that most Kubernetes estates **waste between 30% and 65% of paid resources** due to over-provisioned requests and always-on workloads, with some clusters overspending by **$50,000–$500,000 per year**, depending on size and cloud pricing. In practice, this waste is concentrated in microservices and batch workloads that are kept “warm” 24/7 for convenience or fear of cold starts, even when their actual usage is sporadic.

For CxOs, this means a significant share of the Kubernetes budget is funding idle containers—capacity that consumes power and hardware on-premises, or inflates cloud bills, without delivering commensurate business value. Eliminating idle consumption for rarely used services, while keeping the ability to bring them online in seconds when needed, is therefore one of the highest-leverage cost optimization opportunities in modern Kubernetes estates.

2.3 Why Current Options Fall Short

- **Always-on microservices with HPA**
    - Simple and cloud-native, but minimum replica counts keep at least one pod running per service, locking in idle memory and CPU for every application, regardless of actual demand.
- **Cloud functions / FaaS**
    - Offer true scale-to-zero but introduce provider lock-in, pay-per-invocation pricing, and cold-start behaviours that can range from a few seconds to well over ten seconds depending on runtime and platform.

- **Knative and similar serverless layers**
    - Bring request-driven autoscaling and can reduce cold starts, but add networking and control-plane complexity, often require additional meshes or gateways, and still show variable cold-start latencies in real-world setups.

As a result, many enterprises either accept high levels of idle Kubernetes spend or adopt serverless products that fragment their platform and introduce new forms of lock-in. This leaves a clear gap for a Kubernetes-native pattern that delivers **true scale-to-zero, predictable low cold-start times, and no vendor lock-in**, while remaining simple enough to operate at scale.

2.4 Scope of the Solution

Proxy-KEDA2 addresses these idle-cost and cold-start challenges by introducing a Kubernetes-native pattern that combines true KEDA scale-to-zero with an API-driven wake-up proxy and a lightweight activation catalogue. It enables workloads to shut down completely when idle, then reliably wake in 1–5 seconds, without changing the underlying applications or abandoning existing Kubernetes investments.

The approach remains fully portable, relying only on standard Kubernetes and open-source KEDA, with no dependence on proprietary serverless platforms or per-invocation pricing models. Applications can be activated either via a simple API call—integrated into portals and ITSM tools such as ServiceNow—or via the built-in web-based catalogue, making on-demand environments accessible to both users and automation.

3 Business Objectives

3.1 Cost and TCO goals

The primary objective is to eliminate idle resource spend for non-critical and infrequently used services by scaling their workloads all the way to zero when they are not in use. The solution targets up to 100% reduction in memory usage for dormant services, while preserving agreed SLAs for availability and response time so that cost savings do not come at the expense of user experience.

On-premises, this means reducing power and hardware utilization by allowing single-node or edge clusters to shed idle workloads; in hybrid and cloud environments, it means avoiding compute and memory charges for services that are not actively serving traffic, rather than keeping them warm 24/7.

3.2 Performance and user-experience constraints

The solution is designed to keep applications "feel instant" for end-users and internal consumers, even when they have been scaled to zero. Target behaviour is 1–5 second cold starts from the moment a wake-up is triggered until the first request can be served, with wake-up latency on the control path (API call) well below typical application SLA thresholds.

Autoscaling capabilities must remain fully intact during active periods: once a service is awake, standard Kubernetes and KEDA scaling should handle traffic spikes and load variations without special handling or manual intervention.

3.3 Non-functional goals

The pattern must remain Kubernetes-native and portable, relying only on upstream Kubernetes and open-source KEDA, with no dependence on proprietary serverless platforms or per-invocation pricing models that introduce vendor lock-in.

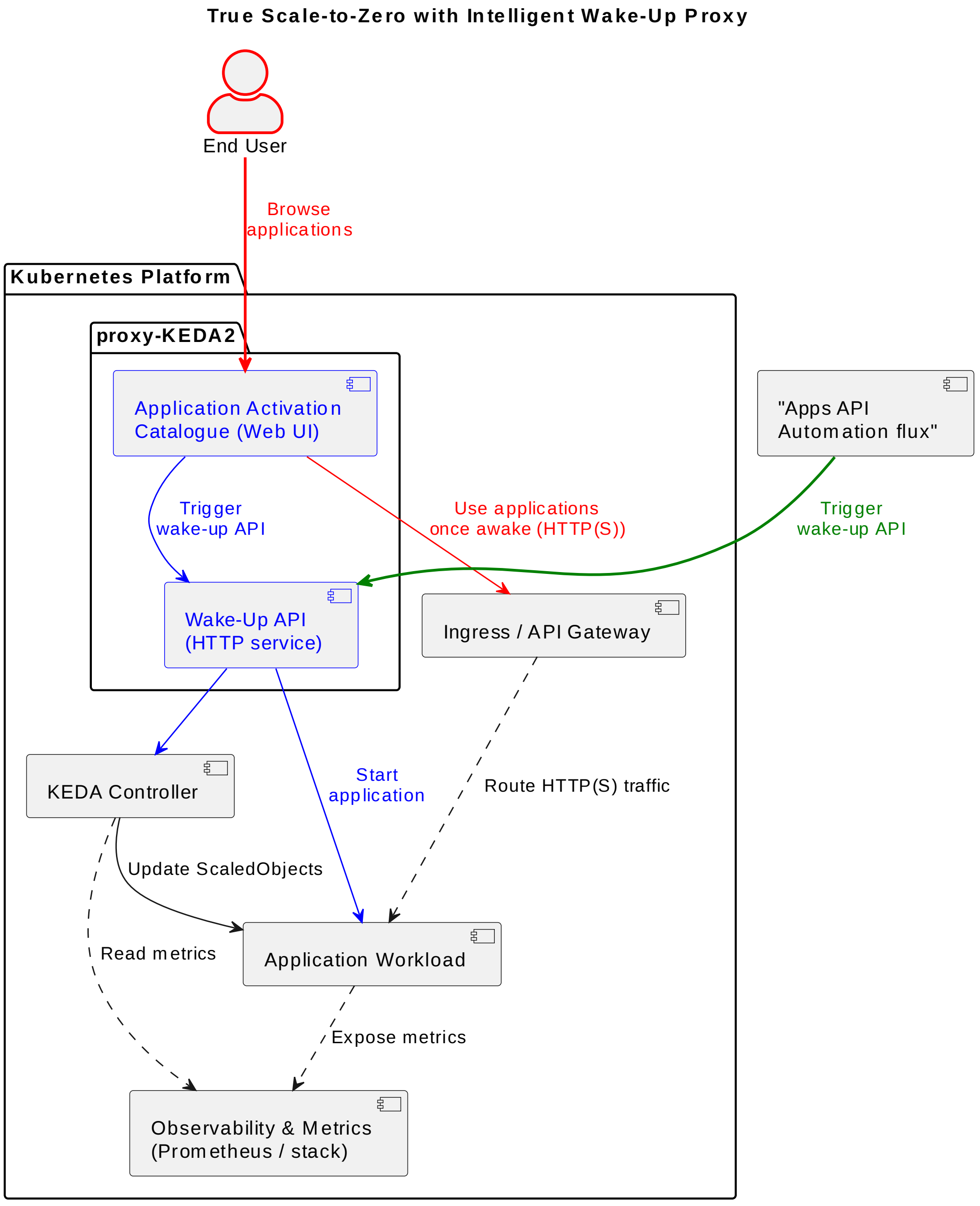
Operationally, the goal is simplicity: wake-up is performed via a simple HTTP/API endpoint that can be integrated into portals and ITSM/service catalogs such as ServiceNow, as well as through a built-in activation catalogue for human users. The approach should reuse existing Kubernetes and KEDA skills, avoid introducing new heavy frameworks, and require no manual intervention during normal operation

4 Solution Overview

4.1 High-level architecture

The solution introduces a Kubernetes-native pattern that sits entirely inside the existing cluster and leverages standard components: Kubernetes deployments and services, KEDA for event-driven autoscaling and scale-to-zero, and a lightweight API-driven wake-up proxy with a web-based activation catalogue. At a high level, applications scale down to zero pods when idle, and are brought back online in seconds when a user or system triggers a wake-up.

A single architecture diagram is sufficient at this level: external consumers (users, portals, ITSM) call either the activation catalogue or a wake-up API; the proxy then coordinates with KEDA and Kubernetes to raise replicas from 0 to 1 and hand traffic back through the existing ingress or API gateway.



4.2 Key building blocks

- **KEDA scale-to-zero**
  KEDA’s ScaledObjects drive true scale-to-zero by reducing replicas to 0 when triggers are idle, then restoring them when demand returns, integrating natively with the Horizontal Pod Autoscaler.

- **API-driven wake-up proxy (Proxy-KEDA2)**
  A dedicated control-plane service exposes a simple HTTP API (`/wake/<app>`), patches KEDA and deployment settings to wake applications on demand, waits for readiness, and then hands control back to normal autoscaling.

- **Activation catalogue (web UI)**
  A lightweight web portal lists onboarded applications with their status and lets users trigger wake-up and then open the application, without needing kubectl or direct cluster access.

- **Kubernetes-native integration**
  All scaling and wake-up behaviour is implemented using the Kubernetes API, standard workloads, and KEDA CRDs, so the pattern runs on any CNCF-conformant cluster without proprietary dependencies.

4.3 Integration points

The wake-up API is designed to integrate cleanly with existing enterprise tools:

- **Portals and intranets** can embed “Start application” buttons that call the wake-up API before redirecting users to the application URL.
- **ITSM and service catalogs (e.g. ServiceNow)** can register catalog items that invoke the API as part of a request or fulfillment workflow, leveraging existing Kubernetes spokes or generic REST integrations.
- **CI/CD and automation pipelines** can use the same API to bring environments online for tests, demos, or batch windows, and let KEDA return them to zero when no longer needed.

5 Operational Behaviour (Conceptual)

5.1 Idle state

In the idle state, selected microservices are fully deployed but scaled down to zero running pods. KEDA monitors the chosen triggers and, when there is no activity, reduces the replicas to zero so that the applications consume no CPU or memory while they are not in use. From a user perspective, the application still exists in the catalogue or portal, but it is in a "sleeping" state rather than burning resources in the background.

5.2 Wake-up

When a user, portal, or ITSM workflow needs a dormant application, it calls the wake-up API or uses the activation catalogue to start it. The wake-up proxy coordinates with the autoscaling layer so that the application is brought back from zero to a running instance, and then made reachable again through the existing ingress or API gateway. End-users experience a short one-time delay of 1–5 seconds as the application wakes, after which it behaves like any other online service.

5.3 Active

Once awake, the application runs as a normal Kubernetes workload. Standard autoscaling behaviour applies: KEDA and the native Horizontal Pod Autoscaler can increase or decrease the number of replicas based on demand, ensuring that performance and SLAs are maintained during peak usage without manual intervention. For the business, this means that scale-to-zero does not limit elasticity; it only eliminates cost when there is no traffic.

5.4 Auto-sleep

After a configurable period of inactivity, the same event-driven logic that woke the service allows it to return to the idle state. When no triggers are active and no requests are observed for a defined cooling period, the application is again scaled back to zero replicas, freeing all associated compute and memory resources. This creates a continuous cycle of Idle → Wake-up → Active → Auto-sleep that aligns resource usage with real demand while keeping the operational model simple and predictable.

6 Performance and Cost Impact

6.1 Cold-start behaviour

In the reference implementation, applications that have been fully scaled to zero can typically be brought back online in **1–5 seconds** from the moment a wake-up is triggered until the first request is served. This range is achieved by combining image caching, streamlined initialization paths, and tight coordination between the wake-up proxy and the autoscaling layer. For end-users, this translates into a brief, predictable delay on first access, after which the service behaves like any always-on microservice.

This cold-start profile compares favourably with many serverless and scale-to-zero patterns, where cold starts of several seconds to minutes are not uncommon for heavier workloads, especially when images must be pulled or GPU resources are involved.

6.2 Idle resource elimination

By allowing selected services to scale all the way down to **zero replicas** when not in use, the pattern eliminates idle CPU and memory consumption for those workloads. In cluster-level terms, this can translate into **up to 100% reduction in memory usage for dormant services**, and a meaningful drop in overall cluster utilization, particularly in estates where many internal tools and low-traffic microservices are kept permanently warm today.

For organizations that currently waste 30–65% of their Kubernetes resources due to over-provisioning and always-on workloads, removing idle pods is one of the most direct ways to cut that waste without compromising performance when the services are actually needed.

6.3 Illustrative cost scenarios

- **On-premises clusters**
    - Scaling non-critical services to zero during idle periods reduces power draw and frees capacity on existing hardware, delaying or avoiding node expansions and new server purchases.
- **Cloud and hybrid environments**
    - In managed Kubernetes services, every idle vCPU and GB of RAM still incurs cost; by removing pods entirely instead of keeping them warm, organizations can reclaim a significant share of their node spend, especially for large clusters running many low-duty-cycle services.

Depending on the size of the estate and the proportion of workloads that can adopt this pattern, enterprises can expect meaningful reductions in their Kubernetes infrastructure costs while preserving, and in some cases improving, overall utilization efficiency.

7 Positioning vs Alternatives

7.1 When this pattern is preferable

For enterprises that already standardized on Kubernetes, this pattern offers **true scale-to-zero without leaving the Kubernetes ecosystem**, making it more attractive than “always-on” microservices, Knative, or standalone FaaS in several situations. Always-on microservices with HPA remain simple but keep at least one replica running per service, locking in idle memory and CPU even when traffic is sporadic.

Knative and cloud functions are well suited for net-new serverless use cases, but they introduce additional control planes, networking layers, and provider-specific runtimes, with associated lock-in and variable cold-start behaviour. The proposed pattern is preferable when organizations want Kubernetes-native scale-to-zero, predictable 1–5 second wake-ups, and integration into existing clusters without adopting a new platform.

7.2 User experience and perceived waiting time

The solution is explicitly designed for **on-demand activation triggered by humans or human-centric flows**: service catalogs, ITSM portals, internal applications, and automation that runs in front of a user or business process. In these contexts, the key question is not just “Can we wake from zero?” but “Does the wake-up delay feel acceptable to people waiting to use the service?”.

UX research identifies three important thresholds: around **0.1 seconds** feels instantaneous, **about 1 second** keeps the user’s flow of thought uninterrupted, and **around 10 seconds** is the upper limit before attention drifts and frustration rises sharply. By keeping cold-start times in the **1–5 second range**, the proposed pattern stays within a psychologically acceptable window: users notice that the system is “doing something”, but the delay is short enough that they typically do not abandon the task or perceive the application as broken.

7.3 How it complements Kubernetes and FinOps

The solution complements existing Kubernetes and FinOps practices by **turning idle workloads into on-demand workloads**, rather than replacing current autoscaling or cost-management tools. It works alongside HPA/KEDA, node rightsizing, and cluster-level optimizations, focusing specifically on eliminating the cost of rarely used services that today are simply left running.

For FinOps teams, this adds another lever to the workload-optimization toolkit: instead of only tuning requests and limits, they can classify services by duty cycle and apply scale-to-zero for low-duty-cycle applications, improving utilization KPIs and reducing waste without restructuring the entire platform.

8 Risk, Security and Governance (High Level)

8.1 Fit with security and governance

The solution is designed to plug into existing Kubernetes security and governance models rather than bypass them. All actions (wake-up, scale-to-zero, status checks) go through the Kubernetes API and are governed by standard RBAC policies, so proxy-KEDA2 and the activation catalogue can be given tightly scoped service accounts with least-privilege access. This allows organizations to align the pattern with their existing controls for environment separation, change management, and auditability, including policy-as-code and admission control where in place.

From a governance standpoint, scale-to-zero is an additional operating mode for selected workloads, not a separate platform: it can be brought under the same change, risk, and compliance processes already defined for Kubernetes deployments and autoscaling policies. Kubernetes governance guidance emphasizes aligning platform behaviour with business objectives and applying consistent guardrails, both of which this pattern supports by using standard resources and controllers.

8.2 Key assurances

The architecture is intentionally Kubernetes-native and portable, relying on upstream Kubernetes constructs and the open-source KEDA project, which is designed to run on any conformant Kubernetes cluster. There is no dependency on proprietary serverless runtimes or per-invocation billing models, which reduces vendor lock-in risk and keeps exit options open between on-prem, private cloud, and managed Kubernetes services.

Security-wise, KEDA and the wake-up proxy follow established patterns: KEDA uses standard CRDs and can authenticate to external systems via Kubernetes secrets and dedicated authentication resources, while RBAC is used to grant only the permissions required for scaling operations. This means organizations can adopt true scale-to-zero with fast wake-up while staying within their existing security, compliance, and governance frameworks.

8.3 Future Enhancements: Administrative Control

In a future iteration, the proxy can also expose controlled administration endpoints for de-provisioning or updating onboarded applications, guarded by dedicated security tokens that provide restricted access to proxy administration. These tokens can be used by either humans (platform teams) or automation (CI/CD, ITSM workflows) to safely trigger lifecycle actions without granting direct cluster-admin privileges.

9 Next Steps and Engagement Model

9.1 Suggested pilot scope

This pilot lets the executive team validate, under an existing non-disclosure agreement, that scale-to-zero can reduce infrastructure costs on non-critical Kubernetes services within a short, low-risk engagement before committing to broader rollout. The recommended pilot focuses on a small, well-defined set of low-duty-cycle services (for example 5–10 internal tools or portals) running on an existing Kubernetes cluster, with all activities conducted remotely using standard collaboration and access tools while local teams retain full control over the environment.

The customer team selects the cluster and candidate workloads; the engagement then concentrates on deploying and configuring proxy-KEDA2 and KEDA, wiring them to existing services, and enabling fast wake-up for those applications without changing their core code.

The goal is to validate cold-start behaviour and idle resource reduction on real workloads with minimal disruption and no travel overhead.

9.2 Engagement options

Engagement is structured as a remote collaboration from the outset and typically starts with a brief online assessment workshop to review the current Kubernetes estate, identify pilot candidates, and agree on success metrics.

The PoC phase then focuses on remote support for deploying and tuning proxy-KEDA2 and its associated features (wake-up API, activation catalogue, KEDA configuration) on the customer's existing cluster. Your team operates the environment; the support role is to provide reference manifests, configuration guidance, and troubleshooting to achieve target wake-up times on selected services. During this phase, KEDA is installed on the existing cluster, selected service manifests are updated to support scale-to-zero, and the proxy-KEDA2 component is deployed from a container registry, ideally GitHub Container Registry (GHCR) under a dedicated organization repository, using image pull secrets if required for private images.

To keep the initial pilot simple, the proxy-KEDA2 component is deployed and managed independently of the customer's existing CI/CD pipelines; further integration of proxy-KEDA2 and KEDA into CI/CD or GitOps workflows remains the responsibility of the customer's platform and application teams, with on-demand support available to help design and review that integration when they decide to industrialize the pattern.

If the pilot is successful, a follow-up engagement can expand the pattern to additional clusters or services and can include sharing more detailed reference implementation assets and best practices while still operating entirely remotely, always within the existing NDA framework. Commercial terms for the assessment and PoC are agreed separately based on scope and platform maturity, ensuring a time-boxed, predictable engagement without open-ended consulting costs.

9.3 Prerequisites for a pilot

To keep the pilot focused and effective, a few baseline conditions are required:

- **Kubernetes platform in place**
  At least one CNCF-conformant Kubernetes cluster (on-prem or managed cloud) with basic observability (metrics, logs) already available.

- **Candidate workloads identified**
  A small set of low-duty-cycle services (e.g. 5–10 internal tools or portals) that currently run 24/7 and are good candidates for scale-to-zero without impacting critical customer-facing SLAs.
- **Platform access and ownership**
  A platform / SRE team with permissions to deploy components into the cluster (namespaces, RBAC, Ingress) and to integrate with existing CI/CD or ITSM systems where needed.
- **Remote collaboration tools**
  Standard remote access and collaboration capabilities (screen-sharing, ticketing, shared documentation) to support a fully remote assessment and PoC