CNCF Case Study - SNOW Corp.

Scaling GenAI for 200M Users: How SNOW Corp. Orchestrates 1000+ GPUs to Handle 700% Viral Traffic Spikes

Last update: 2025.12.18 (THU) KST

Topics

Introduction

Organization Scale & Workload Challenges

- SNOW Corp., the subsidiary of NAVER (The “Google of Korea”), manage a fleet of **1,000+ GPUs** to serve GenAI features for **200M+ global users.**
  - **Dominant Market Position**: #1 Camera/Photo app in South Korea, Japan, and Vietnam with **1.5B+** cumulative downloads.
  - **Market Leadership**: Three apps (SNOW, EPIK, B612) ranked in the "**Top 50 Gen AI Mobile Apps**" **by a16z** (Sept 2025).
- This introduced the following workload challenges for SNOW Corp.:
  - **Continuous Service Evolution**: Retaining market leadership requires the continuous release of novel GenAI filters. This demands an agile infrastructure capable of rapid model deployment and frequent updates without service interruption.
  - **Extreme Traffic Volatility:** The viral nature of AI trends (e.g., "Ghibli Filter") triggers unpredictable traffic surges (up to 700%), rendering static capacity planning impossible.
  - **Heterogeneous Inference Workflows:** Continuous release of diverse AI filters creates a complex mix of compute-heavy and memory-intensive workloads, making standard "one-size-fits-all" resource allocation inefficient.

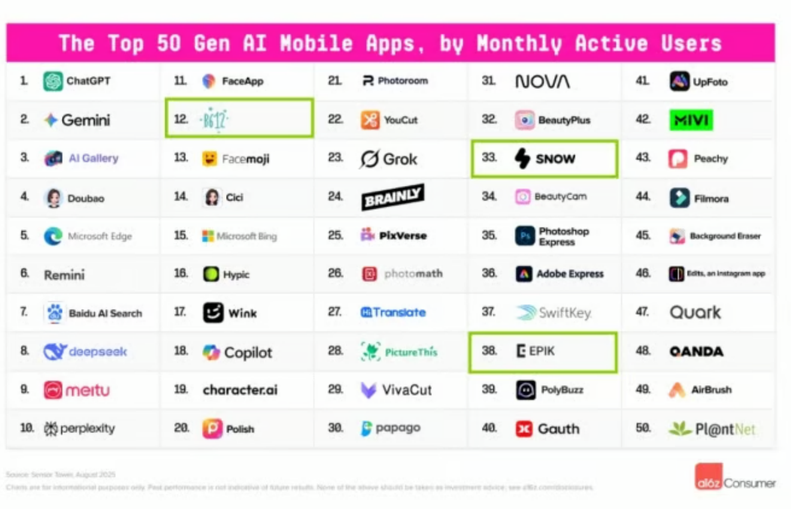

Figure 1. A16z report on the top 50 Gen AI Mobile Apps 2025.

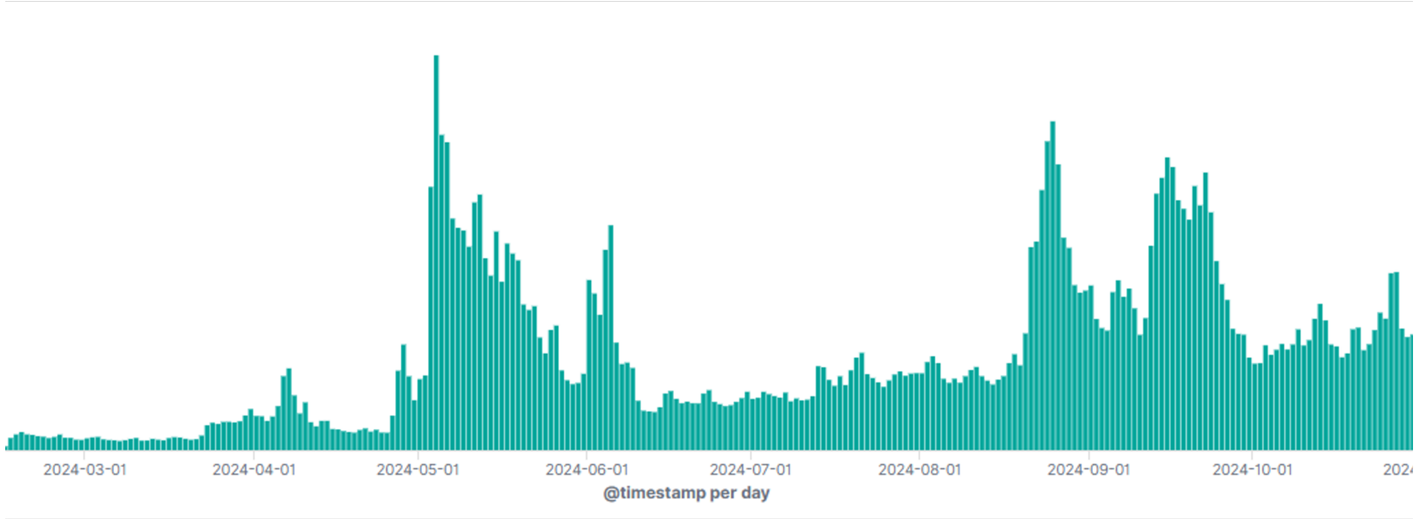

Figure 2. Usage of SNOW’s AI Image Filter in 2024.

Unifying 1000+ GPUs: From Static Docker to Orchestrated Kubernetes

- Supporting SNOW Corp.'s rapidly scaled AI services, the infrastructure managed **1000+ A100 GPUs** and **1,200+ workflows**.
- The legacy system, relying on manual Docker-based provisioning, could not handle this complexity and volume. This resulted in three critical issues:
  - **System Instability**: The lack of a centralized monitoring and recovery system caused serving instability and **negatively impacted the user experience**.
  - **Inefficient Resource Utilization**: Static allocation caused fragmentation and low utilization, causing **resource waste and unpredictable performance**.
  - **Operational Overload**: Instability, the lack of automatic recovery, and resource wastes required heavy reliance on manual intervention, driving up **staff workload and operational costs**.

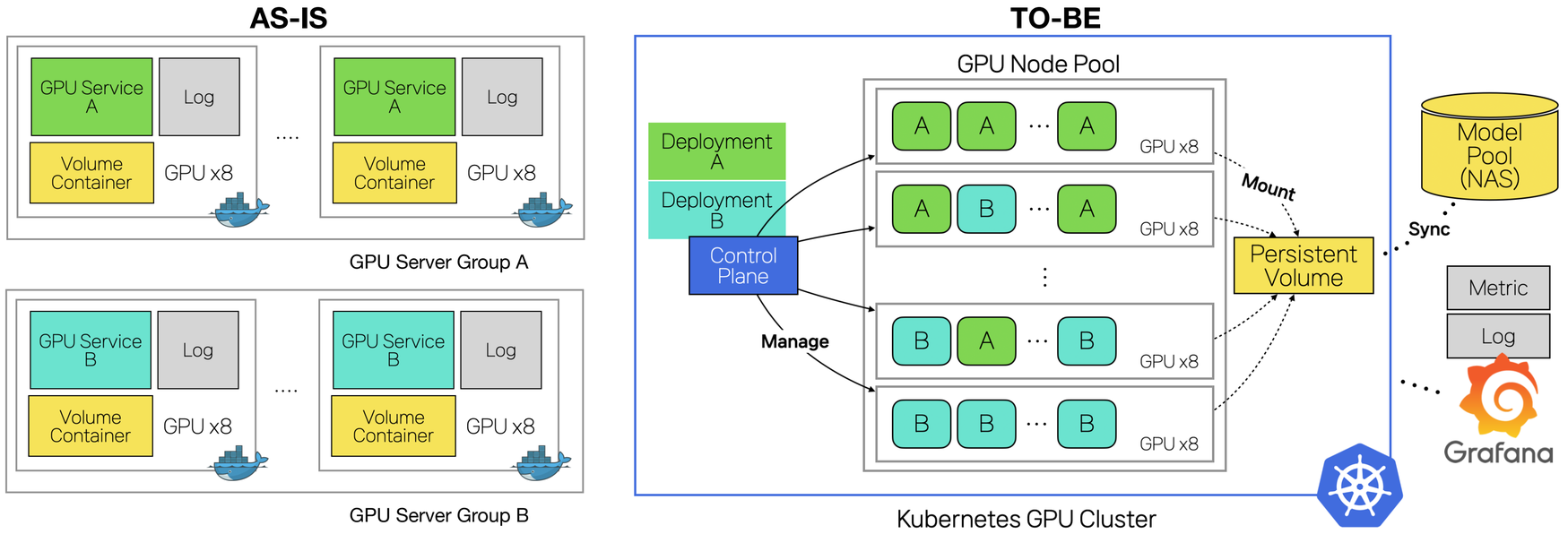

Figure 3. Infrastructure Evolution. The legacy topology (left) relied on isolated Docker hosts with manual GPU binding and local volume containers. The new Kubernetes architecture (right) unifies these into a centralized Control Plane with dynamic Node Pools and shared storage (NAS/PV), enabling automated scheduling and resource sharing.

Method

Building a Cloud-Native Foundation with CNCF Ecosystem

- **Goal:** to enhance serving stability by migrating services to Kubernetes, ensuring automatic recovery and monitoring capabilities across the entire cluster.
  - Transitioned from legacy Docker-based provisioning on Bare Metal/VMs to a **containerized orchestration platform**.
    - Architected multi-region on-premise HA Clusters across multiple physical data centers.
    - Decoupled External ETCD topology to maximize service survivability for production-grade services.
  - Established centralized monitoring using **Prometheus**, **Loki** and **Grafana**, integrated with alert systems.
  - Orchestrated with CNCF Ecosystem including the following projects: **Cilium** for high-performance CNI, **Helm** for code-based service configuration, **Traefik** for dynamic ingress/reverse proxy, **HAMi** for efficient GPU sharing, and **KEDA** for autoscaling system.

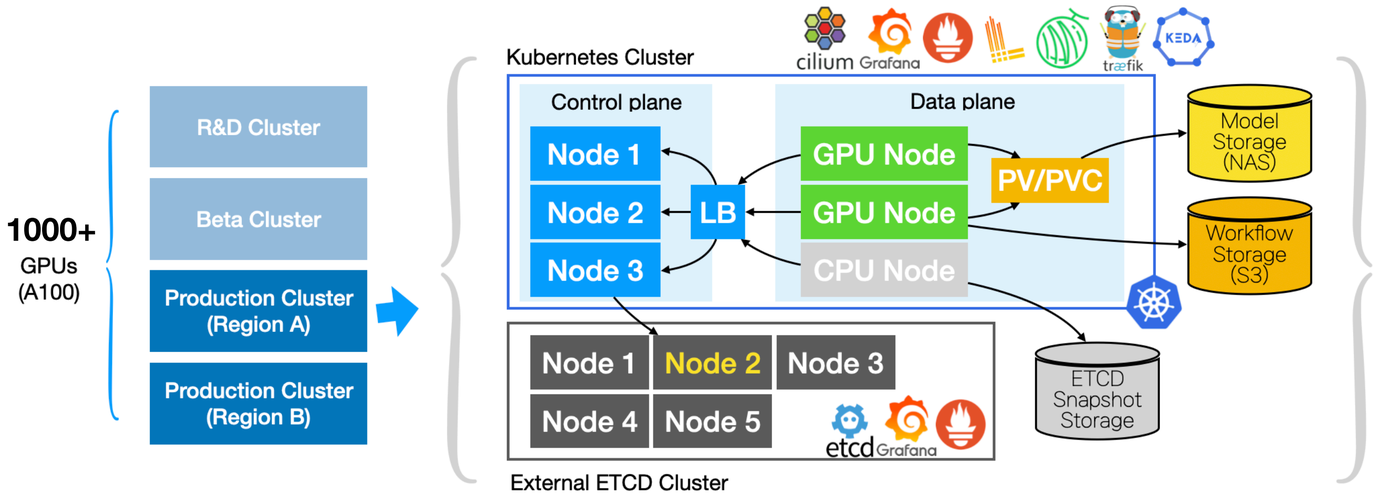

Figure 5. Implementing high availability (HA) Kubernetes clusters by leveraging CNCF projects.

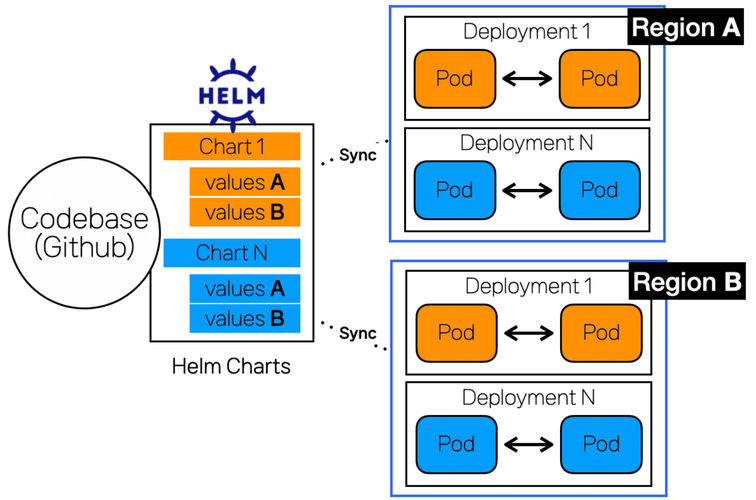

Figure 6. Standardized Service Deployment Workflow using Helm Charts. Note that the synchronization of Helm charts and the clusters is performed by CICD pipeline via GitHub Actions.

Migration Hurdle: GPU Sharing for Sequential Pipelines

- **Problem:** Kubernetes' strict GPU resource isolation blocked the migration of our legacy architecture.
  - A core product feature relies on a sequential **"Train-to-Inference" pipeline**.
    - While seamless in Docker, the default Kubernetes scheduler cannot share a single GPU across multiple containers, preventing two distinct engines from co-existing on the same device.
  - This threatened to force:
    - **2x GPU usage** for existing pipelines, which severely degrades the GPU utilization.
    - **A massive code rewrite** just to accommodate the infrastructure change.
- **Solution**: HAMi is an open-source extended scheduler that virtualizes GPU resources (vGPU) in a Kubernetes environment
  - **Device sharing**: enables multiple containers in the same pod to share the same GPU concurrently.
  - **Flexibility**: requires zero changes to existing programs. Just install using Helm and assigning GPUs in the chart.
  - **Kubernetes-native Integration**: can be used in parallel with the existing kube-scheduler and does not conflict with cluster standard configurations or the autoscaling ecosystem (KEDA, HPA, etc.).
- By adopting HAMi, a flexible GPU scheduling mechanism—comparable to the level of Docker—was achieved within the Kubernetes environment, securing both enhanced GPU resource utilization and stability during migration, without **requiring additional code modification**.

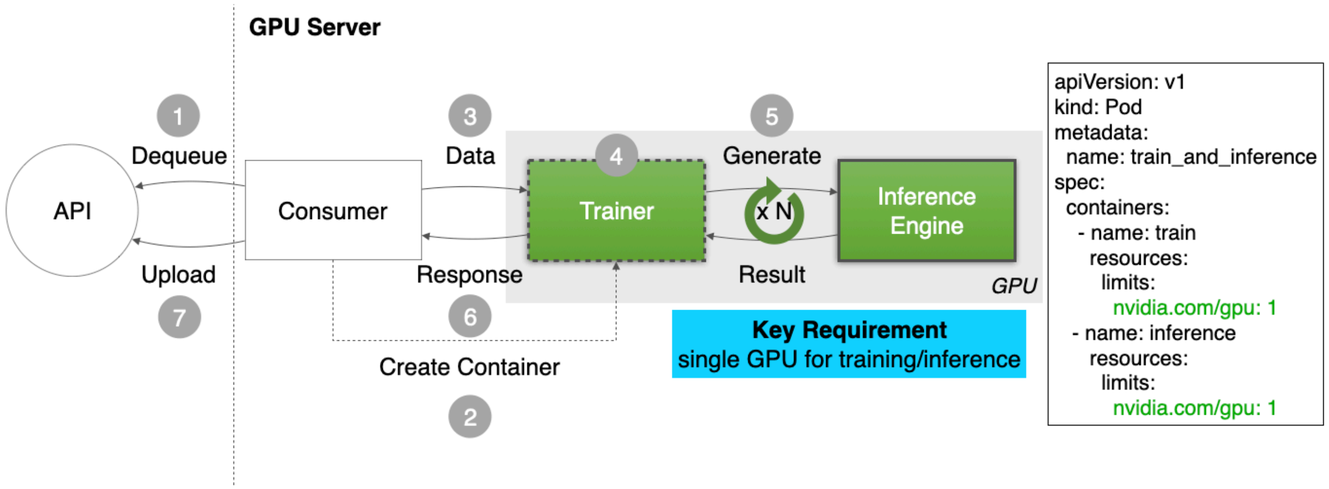

Figure 7. Sequential Train-to-Inference Pipeline Workflow.

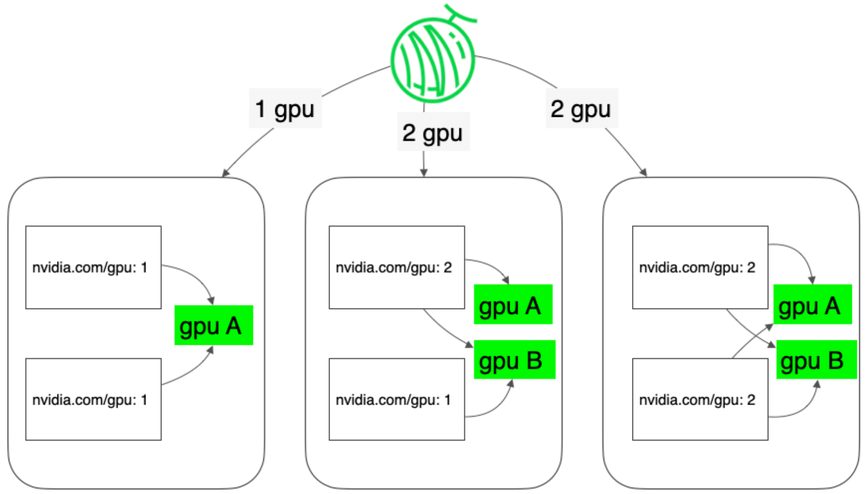

Figure 8. HAMI's flexible GPU allocation Feature.

Proactive GPU Orchestration Based on Real-time User Traffic

- **Problem:** conventional metrics fail to capture true service saturation.
  - **Inaccurate Saturation Signal:** Standard metrics (CPU/RAM) fail to reflect actual service load. Even GPU-specific metrics (like DCGM utilization) are unreliable because our workloads are heterogeneous: varying workflow intensities mean that high GPU utilization does not necessarily correlate with service saturation, and vice-versa.
  - **Lagging Indicator**: While KEDA's built-in RabbitMQ scaler supports queue-length based scaling, it functions as a reactive lagging indicator. Given our ~1 minute model warm-up time, scaling triggered after a queue backlog forms is too late, causing requests to be throttled before new workloads are ready to serve.
- **Solution**: we utilized KEDA's Metrics API and introduced a custom metric server to adapt KEDA scaler.
  - **Custom KEDA Adapter**: Developed a lightweight Metric Server to expose real-time RabbitMQ consumer states directly to the Kubernetes HPA.
  - **Proactive Thresholding**: Introduced a "Consumer Saturation" metric (Unacked Messages / Active Consumers). By setting the scale-out threshold, the cluster proactively provisions new GPUs before the current pool is fully saturated, effectively creating a buffer that absorbs the 60s warm-up latency.

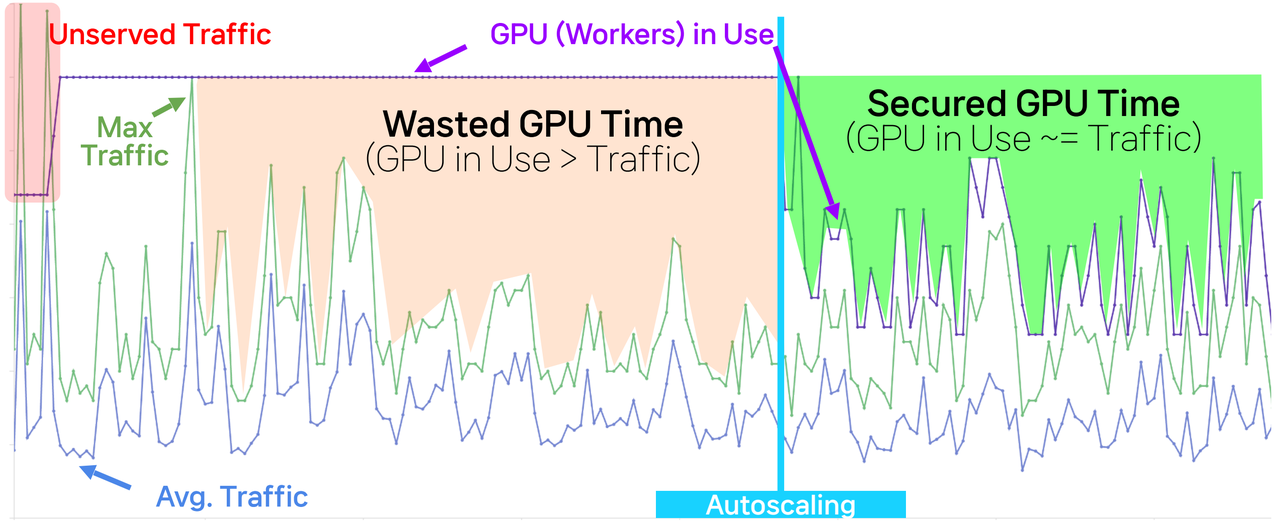

Figure 9. Impact of Proactive Autoscaling on GPU Efficiency. The chart highlights the shift from static provisioning—plagued by 'Wasted GPU Time' (Orange) and unserved spikes (Red)—to dynamic scaling (Green), where GPU allocation tightly synchronizes with real-time user traffic.

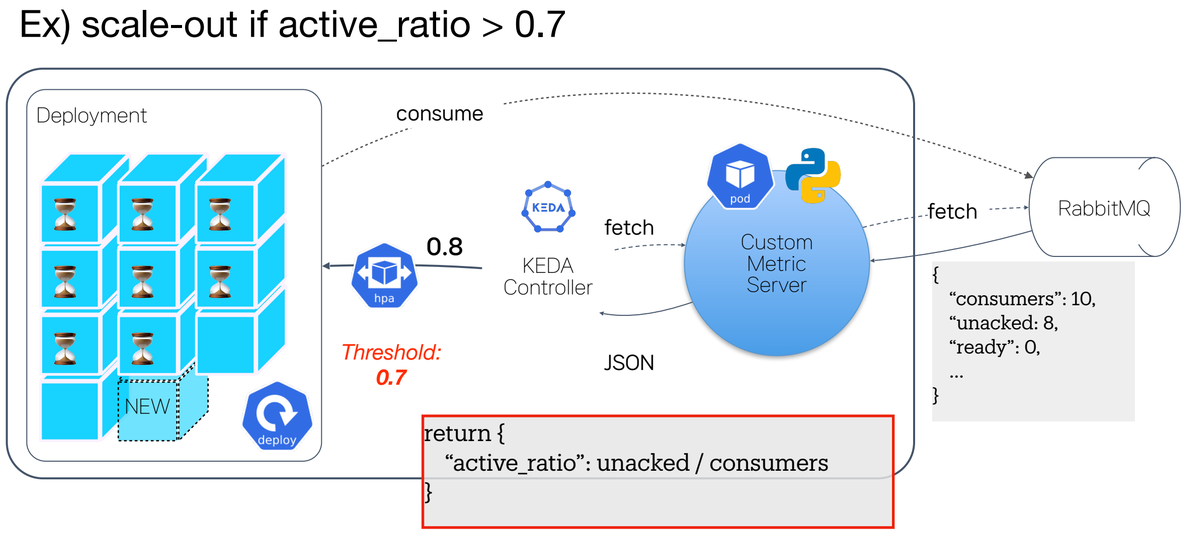

Figure 10. Custom Autoscaling Logic Example. The system calculates saturation (active_ratio) by dividing unacknowledged messages (representing the number of busy workers) by the count of active consumers. In this example, the deployment scaled out since the active ratio (0.8) exceeds the threshold (0.7).

Results

Hybrid Cloud Bursting: Handling 700% Traffic Spikes with Multi-Cluster Scaling

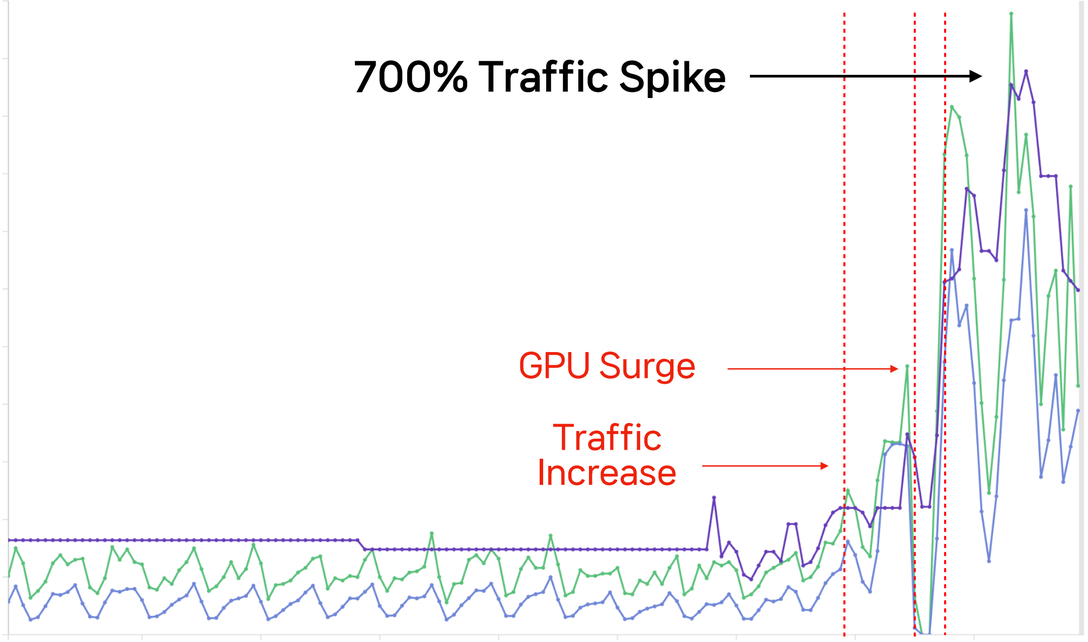

Figure 11. Real-world validation during the 'Ghibli Filter' viral event. The graph captures a sudden 700% traffic surge (Green/Blue lines). The autoscaling system and hybrid cloud infrastructure rapidly provisioned additional GPU capacity (Purple line) to match this demand, successfully serving the 7x peak load without service interruption.

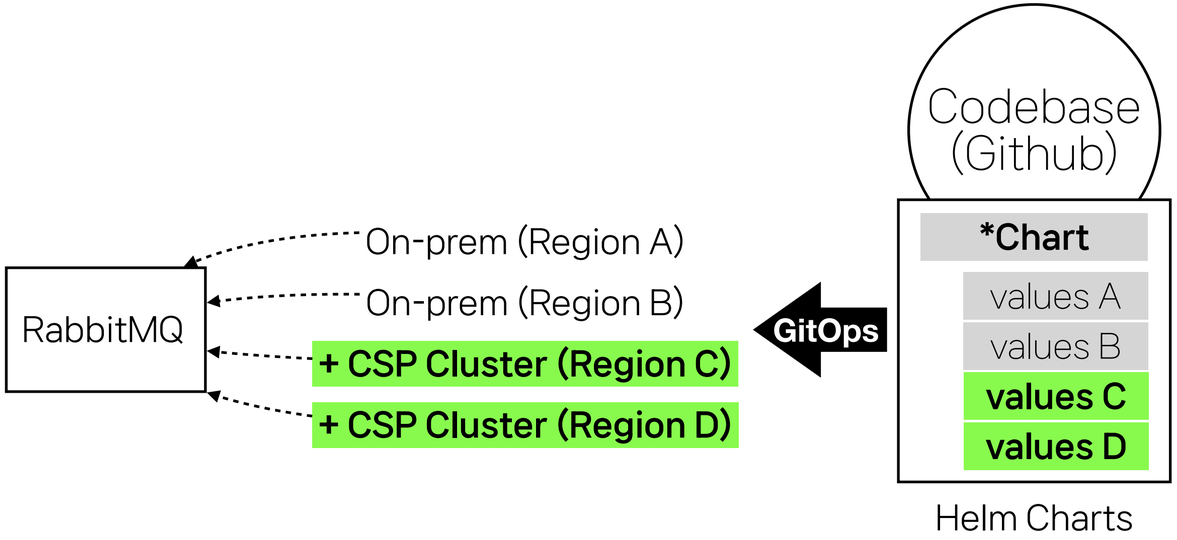

Figure 12. Hybrid Cloud Bursting Architecture. To overcome on-premise capacity limits, the system expanded into Cloud Service Provider (CSP) regions. A unified GitOps pipeline deployed identical Helm charts across all clusters, while CSP worker nodes consumed tasks from the central RabbitMQ via secured connection.

- **Incident:** Viral "Ghibli style" trend triggered massive demand for SNOW's Ghibli filter, tripling traffic in 3 hours on a low-staff Saturday morning (KST).
- **Initial State:** Autoscaling initially held, but throttling ensued due to GPU surge, urgently demanding a strategic pivot to resolve GPU saturation.
- **Challenge:** Overcome capacity limits immediately to consume the maximum possible traffic volume before the viral trend faded.
- **Achievement**: Successfully multiplied consumed traffic to 7x peak levels, securing the business opportunity through autonomous, zero-interruption scaling.

Quantitative Results

- **System Resilience Enhancement**
  - MTTR reduced by **91%** (from approx. ~2 hrs to ~10 min.)
  - GPU Surge-related user errors dropped by **85%** during peak traffic due to proactive scaling. (See Figure 13)
- **Resource Efficiency Maximization through KEDA-based Autoscaling**
  - Average GPU time for the production cluster was reduced by **55%**.
  - Internal batch process time decreased by **81%** (from approx. 6 hrs to less than 1 hr)
- **Operational Autonomy**
  - Operational cost savings equivalent to **10.8 Man-months** were achieved.
  - Release period is reduced from 1~2 months to **days**.
- **Zero-downtime Scaling**
  - Successfully handled **700% traffic spikes** during viral events (e.g., 'Ghibli Trend') without service interruption, securing critical business opportunities.

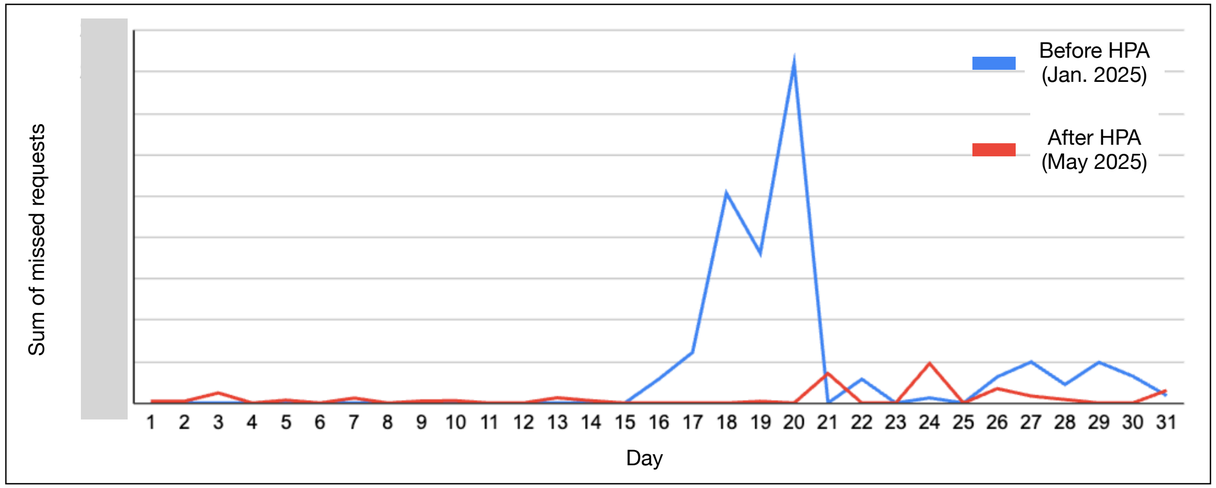

Figure 13. Error count comparison on GPU surges. Note that the KEDA-based GPU orchestration system was applied in May (red line).

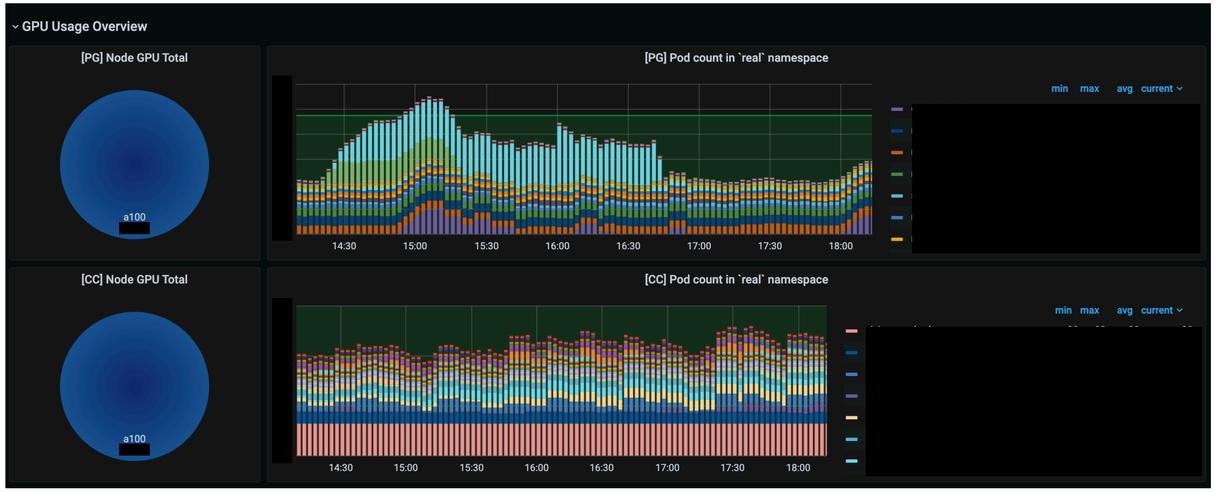

Figure 14. GPU Usage Monitoring Dashboard (Grafana).

End of Document