



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

CRI-O: Faster Pulls, Better Metrics, and a Future Beyond Images

Sohan Kunkerkar & Ayato Tokubi, Red Hat



Overview



KubeCon



CloudNativeCon

Europe 2026

- Introduction
- CRI-O updates
 - CRI List Streaming
 - CRI Stats
 - Read-Only Artifact Store
 - Security Updates
- Support MemoryQoS with Cgroup v2 (KEP-2570)
- P2P Image Distribution
- Demo
- Future work
- Closing Remarks



Introduction



KubeCon



CloudNativeCon

Barcamp 2020

CRI Implementation

Implementation of the Kubernetes **C**ontainer **R**untime **I**nterface - compliant with the **O**pen Container

- Compliant with the Open Container Initiative (OCI)
- Supports OCI based container images, runtimes, and registries

Key Principles

- Balance stability and features
- Focus on security



cri-o

CNCF Graduated Project – widely adopted in Kubernetes & OpenShift

CRI List Streaming: KEP-5825

- CRI list operations use **unary RPCs** — entire response in a single gRPC message
- gRPC per-message size limit: **16 MB** (defined in kubelet)
 - Already raised twice: 4 MB → 8 MB → 16 MB
- At ~10,000+ containers/pods, the response exceeds the limit → complete RPC failure
- Solution: **server-side streaming** RPCs
 - StreamContainers, StreamPodSandboxes, StreamImages, and stats streams
- One chunk per gRPC message, same filter parameters as unary RPCs
- Does **NOT** reduce memory usage — kubelet still collects all items into a list
- Backwards-compatible: falls back to unary if runtime returns **UNIMPLEMENTED**
- **Alpha** in Kubernetes 1.36 (**CRIListStreaming** feature gate), CRI-O 1.36

CRI Stats: KEP-2371

- Container and pod stats collected **via CRI** instead of cAdvisor
- Kubelet and Container Runtime collects same information
 - => Kubelet queries the container runtime directly — eliminates duplication
- Container runtimes implemented required metrics for graduation
- Was supposed to graduate to beta in 1.36, remains **alpha**
- Performance tests revealed no performance improvement without cAdvisor
 - Kubelet still runs cAdvisor code path even with CRI stats enabled
 - Paying the cost of both systems
 - cAdvisor code path removal from kubelet is planned

Additional Artifact Stores

- OCI artifacts previously hardcoded to `<graphroot>/artifacts/`
- New config: `additional_artifact_stores` in `[crio.runtime]`
- **Read-only** stores, checked in order before the default location
- No merging or overlay — simple lookup chain
- Use cases:
 - Large ML models on dedicated storage (NVMe, network-attached)
 - Pre-populated caches baked into node images (air-gapped / immutable)
 - Shared read-only storage across nodes (NFS, cloud filestore)
- **Experimental**, subject to change — shipped in CRI-O v1.35

Security Updates

Configurable TLS (since 1.35)

- `tls_min_version` and `tls_cipher_suites` in `[crio.api]`
- Applies to both streaming server and metrics server
- Compliance: FedRAMP, PCI-DSS, internal security policies
- PQC-ready: enable post-quantum cipher suites as they land in Go

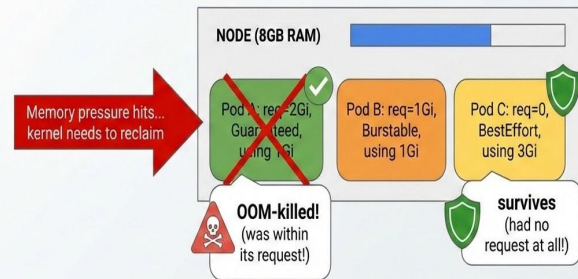


OpenVEX Reports

- Generate OpenVEX vulnerability reports via **govulncheck**
- Standardized vulnerability status for supply chain security workflows

MemoryQoS: The Problem

- NODE (8GB RAM)
 - Pod A: req=2Gi, Guaranteed, using 1Gi
 - Pod B: req=1Gi, Burstable, using 1Gi
 - Pod C: req=0, BestEffort, using 3Gi
- Memory pressure hits... kernel needs to reclaim
 - Pod A: OOM-killed! (was within its request!)
 - Pod C: survives (had no request at all!)
- requests.memory is invisible to the kernel
- It's a scheduling hint only -- no kernel enforcement



! requests.memory is invisible to the kernel.
It's a scheduling hint only -- no kernel enforcement.

MemoryQoS: Tiered Protection (KEP-2570, K8s 1.36)

- cgroup v2 memory protection -- different tiers per QoS class:



Guaranteed: memory.min (HARD LOCK)

- Kernel NEVER reclaims this memory
- ✓ Pod paid for it -- it's theirs



Burstable: memory.low (SOFT PROTECT) + memory.high (THROTTLE)

- Kernel prefers not to reclaim (soft)
- ~ memory.high = speed bump before OOM (factor 0.9)



BestEffort: no protection

- BestEffort: no protection: Reclaimed first under pressure
 - ✗ Reclaimed first under pressure
- Different QoS classes deserve different kernel protection levels



cgroup Hierarchy & Observability

- **kubepods/:** `memory.min` = sum of all requests
 - **Guaranteed:** `kubepods-guaranteed.slice/`
 - `memory.min` = Guaranteed total (e.g., container: 512Mi)
 - **Burstable:** `kubepods-burstable.slice/`
 - `memory.low` = Burstable total (e.g., container: 256Mi, `memory.high`: 486Mi)
 - **BestEffort:** `kubepods-besteffort.slice/` (no protection)
- **New Prometheus Metrics (Alpha):**
 - `kubelet_memory_qos_node_memory_min_bytes` (hard-protected total)
 - `kubelet_memory_qos_node_memory_low_bytes` (soft-protected total)
- **CRI-O Fix (PR #9820):** Uses cgroup v2 Unified map in `UpdateContainerResources()`, enabling live memory protection updates during in-place pod resize.

Why P2P Image Distribution? Why Now?

- **AI/ML is changing container image sizes:**
 - Traditional app image: ~200MB
 - PyTorch + CUDA image: ~8GB
 - LLM model image: ~20-50GB+
 - 100-node GPU cluster x 20GB = 2TB egress per deploy
- **OCI Image Volume (KEP-4639) in K8s:**
 - Mounts OCI artifacts as read-only volumes.
 - Solves "how to use," not "how to distribute."
 - Every node still pulls independently from the registry
- **Spegel solves distribution:**
 - One node pulls, others use P2P (bandwidth reduction)
 - Works with regular images and OCI artifacts
 - Offers no rate limiting, works air-gapped, and provides faster startup

Spegel + CRI-O: Architecture

- **Problem: Storage**

- CRI-O's fixed storage (shared with Podman/Buildah) decompresses layers, losing compressed blobs.

- **Solution: Blob Cache Layer**

- A TeeReader writes to overlay storage and a blob cache simultaneously
- This keeps compressed blobs (matching registry images)
- `metadata.json` tracks blobs and sources

- **Spegel Integration**

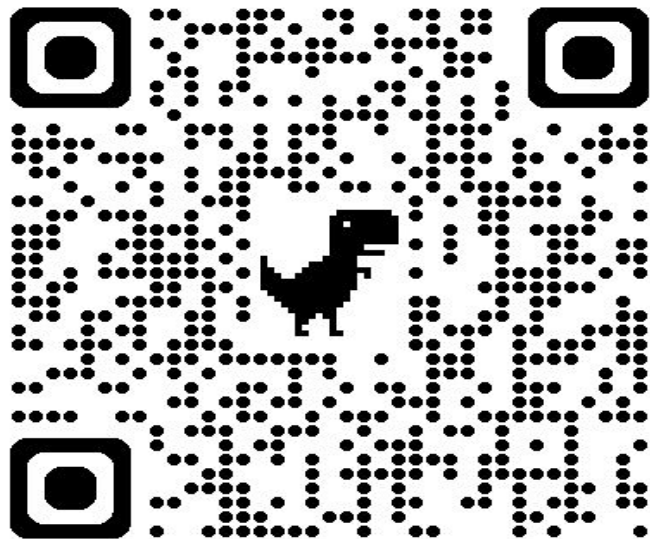
- CRIOClient monitors `metadata.json`
- It advertises available blobs via libp2p DHT and serves blobs P2P

- **Status: WIP**

- <https://github.com/cri-o/cri-o/pull/9628>
- <https://github.com/spegel-org/spegel/pull/1119>
- <https://github.com/containers/container-libs/pull/561>



Demo: Spegel + CRI-O P2P Pull



Food for Thought: P2P-Aware Scheduling

- Core Concept: The scheduler identifies which nodes already have the required container images cached.
- Benefit (Example): A Pod requiring a large image (e.g., `llama-70b-inference`, 45GB) starts near-instantly by avoiding the 45GB image pull.
- Mechanism: Spegel's Locality API (via DHT) tracks which nodes possess specific blobs (layers).
- Community Feedback Sought on

Future Work



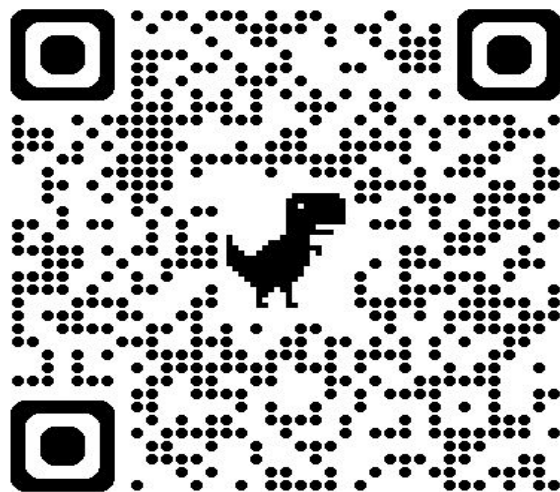
KubeCon



CloudNativeCon

Europe 2026

Explore and contribute to CRI-O's future roadmap



Reach out to us

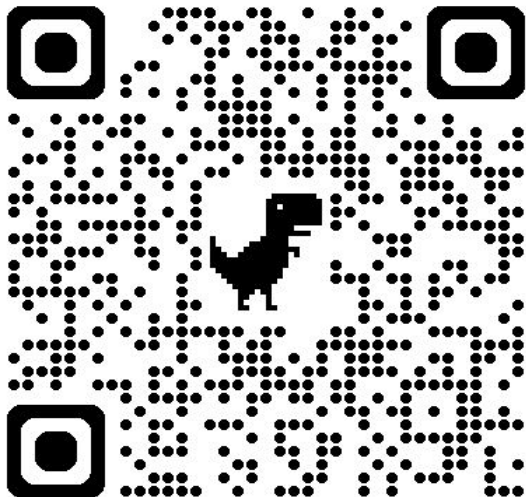


KubeCon

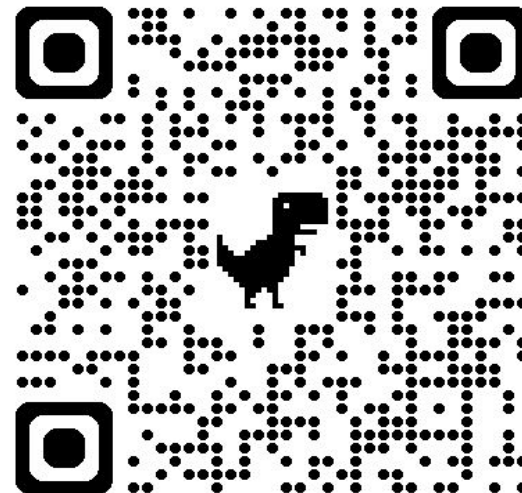


CloudNativeCon

Europe 2026



Github



Slack

Thanks!

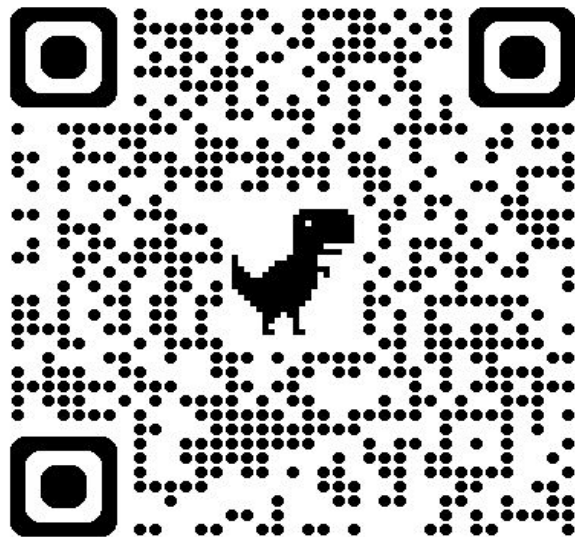


KubeCon



CloudNativeCon

Europe 2026





KubeCon



CloudNativeCon

Europe 2026

