KubeCon

CloudNativeCon

North America 2024

Navigating the cgroup Transition: Bridging the Gap Between Kubernetes and User Expectations

Sohan Kunkerkar, Red Hat Inc.

About the Speaker

Sohan Kunkerkar

Senior Software Engineer - Red Hat

- CRI-O maintainer
- Member of SIG-Node
- Love playing the flute
- Enjoy trekking and outdoor activities

Agenda

cgroup and Migration

- Introduction to cgroup
- Transition Path from v1 to v2
- cgroup in Kubernetes
  - Demo
  - Benefits of cgroup v2
- Best Practices for Migration

Impact and Future

- Real-World Experiences
  - Industry Adoption
  - Language/Workload Compatibility
- Impact on Kubernetes Ecosystem
  - Stakeholders Involved
  - Challenges
- Future Outlook
- Conclusion and Q&A

Introduction to cgroup

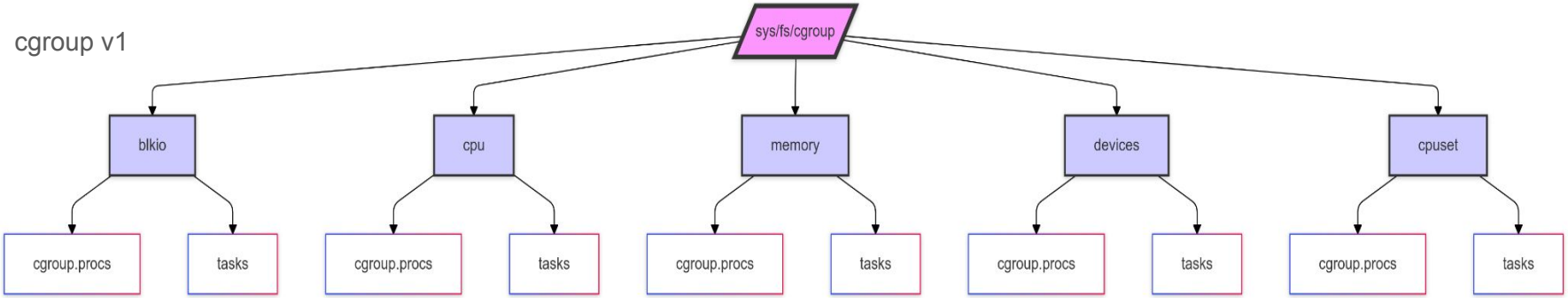
- A Linux kernel feature for managing system resources.
- Controls CPU, memory, disk I/O, and network bandwidth for processes.

- Impose limits on resource usage.
- Monitor the performance of grouped resources and control their scheduling and prioritization.

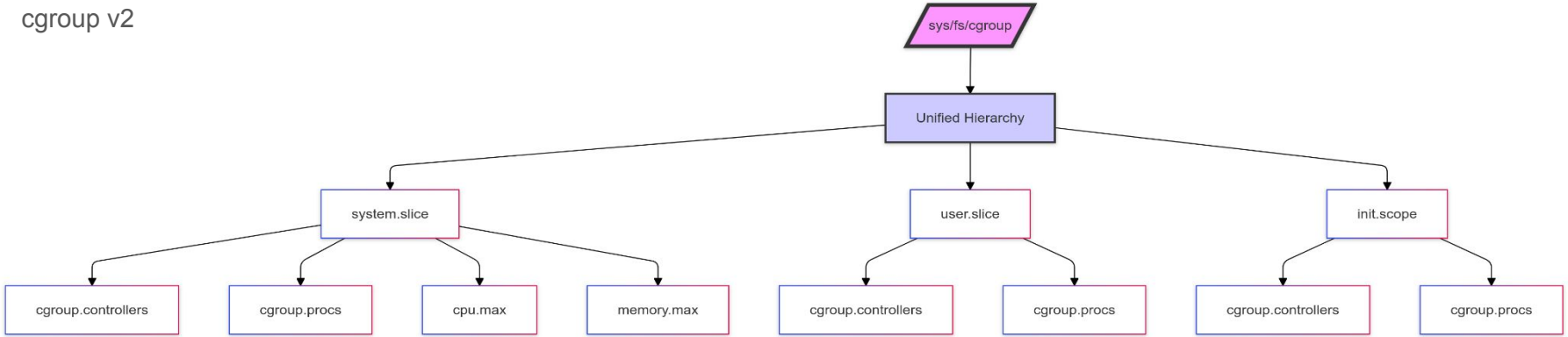
- Prevents any single process from monopolizing resources.
- Critical for process isolation, security and performance optimization, especially in multi-tenant environments such as cloud computing and container-based deployments.

cgroup Versions

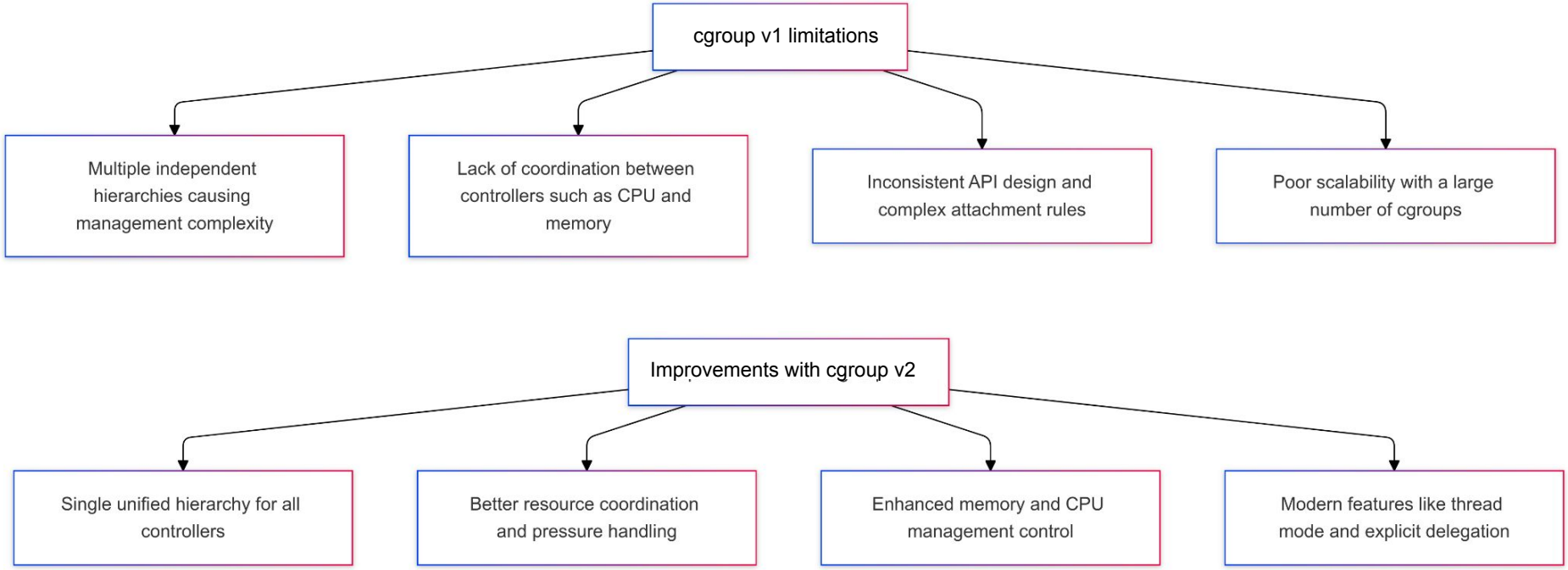
cgroup v1



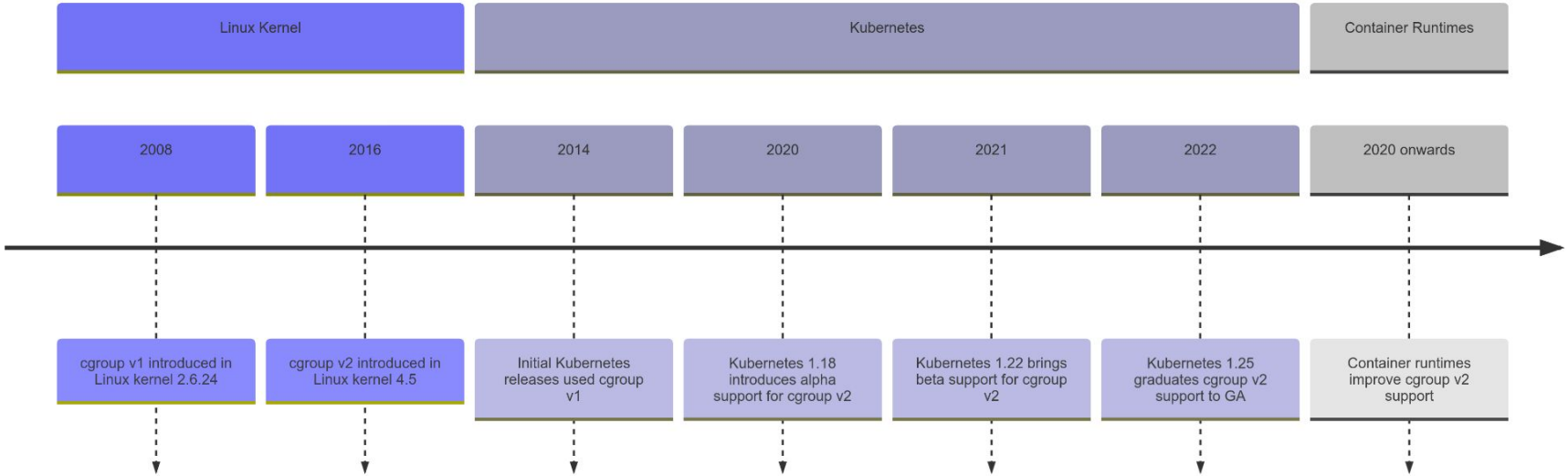
cgroup v2



Transition Path from cgroup v1 to v2



cgroup in Kubernetes



cgroup in Kubernetes

Resource Allocation

Isolation

Monitoring

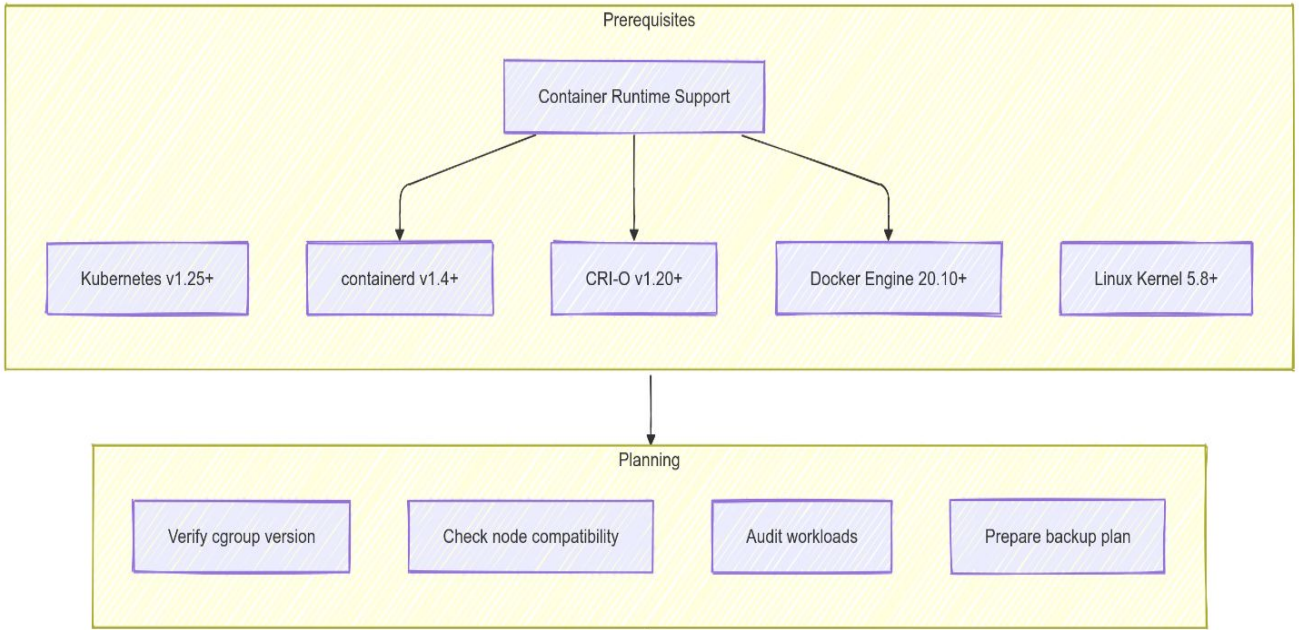
```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: webserver
  name: webserver
spec:
  containers:
  - image: nginx
    name: webserver
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Demo

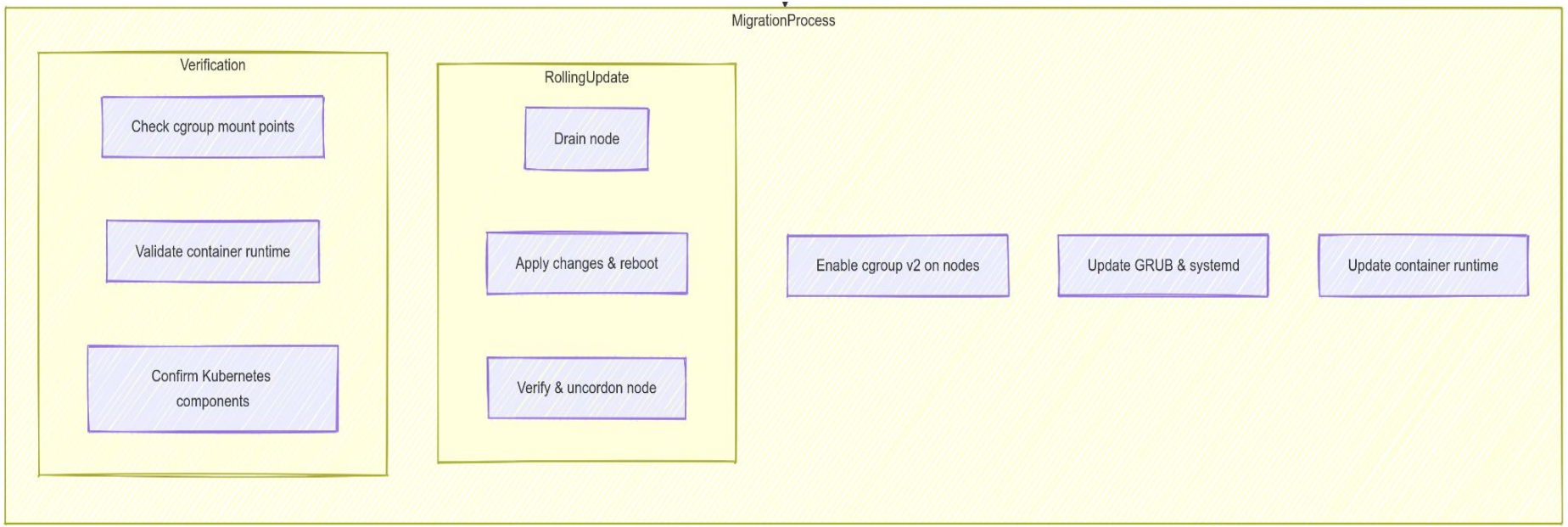
Benefits of cgroup v2 in Kubernetes

- **Memory QoS**: Enables fine-grained memory allocation to ensure critical workloads maintain performance.
- **Swap Support**: Allows effective use of swap space to handle memory overcommitment without crashes.
- **CPU Load Protection**: Protects critical processes from CPU overcommitment during high-load scenarios.
- **Pressure Stall Information (PSI)**: Provides real-time metrics on resource pressure for informed scheduling decisions.
- **eBPF-based Resource Management**: Facilitates dynamic and efficient resource monitoring and control.
- **Nested Containers**: Supports better isolation and management in complex applications requiring multiple container layers.
- **Pod-Level Resource:** Enables setting CPU and memory requests/limits at the pod level, which applies to the aggregate of all containers within the pod.

cgroup v2 Migration: Best Practices



cgroup v2 Migration: Best Practices



Infrastructure Adoption

Default Support Optional Support

Operating Systems

- RHEL 10
- RHEL 9, Ubuntu 22.04 LTS
- SLES 15 SP3+, Amazon Linux 2023
- RHEL 8, Ubuntu 20.04 LTS
- Fedora 31+
- Fedora CoreOS

Kubernetes Distributions

- OpenShift 4.14+
- EKS 1.25+ (with AL2023 nodes)
- GKE 1.26+ (with COS/Ubuntu nodes)
- OpenShift (Pre-4.14)

Other Requirements

- Linux Kernel 5.8+ (Minimum)
- systemd 237+
- containerd 1.4+
- CRI-O 1.20+
- Docker 20.10+

Adoption in Kubernetes Ecosystem

containerd

cilium

cAdvisor

systemd

Prometheus

gVisor

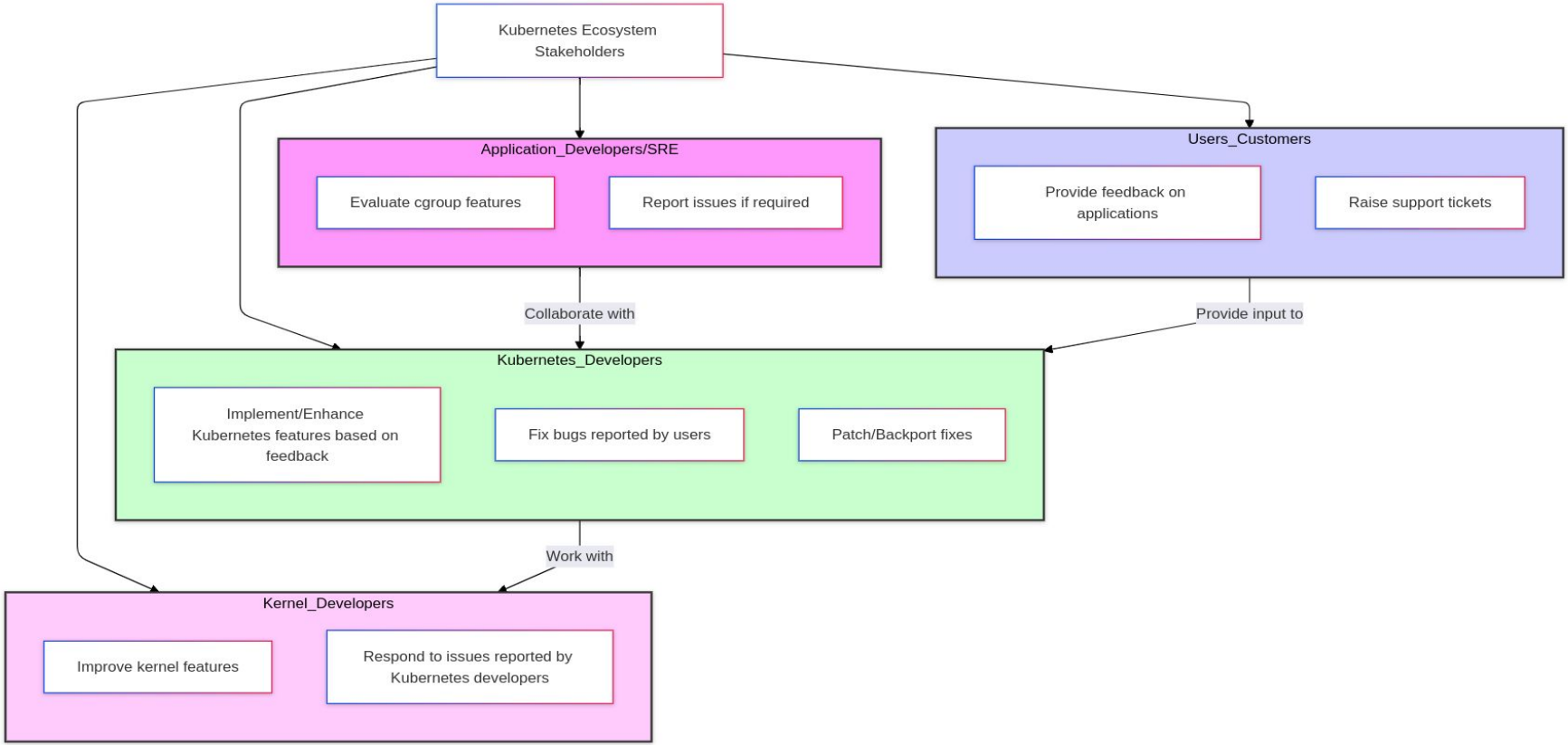
Language Compatibility

Language	Version Requirements	Configuration Needs	Specific Considerations
Java	• JDK 8u392+ • JDK 11.0.16+ • JDK 17.0.4+ • JDK 19+	Configuration Required	• -XX:+UseContainerSupport • -XX:+UseZGC or -XX:+UseG1GC recommended • Memory limits need verification
Node.js	• 14.x: Limited support • 16.x+: Full support	Native Support	• Automatic memory limit detection • V8 heap configuration recommended
Python	• 3.9+: Full support • 3.7-3.8: Limited	Native Support	• cgroups module available • Memory tracking automatic
Go	• 1.16+: Full support • 1.19+: Enhanced features	Native Support	• GOMEMLIMIT awareness • Automatic resource detection • GOGC configuration optional
.NET	• .NET Core 3.1+ • .NET 5.0+: Enhanced	Version Dependent	• GC configuration recommended • Server GC considerations

Optimizing Workload Performance

Workload Type	Configuration	Key Considerations
Memory-Intensive	Better memory usage control with *memory.high*, PSI metrics	Monitor with PSI to detect memory pressure early; optimize memory.high and memory.swap.max.
CPU-Bound	Unified CPU control (*cpu.max*), better throttling management	Enhanced QoS adherence; adjust cpu.max and cpu.weight to prevent performance dips.
I/O Heavy	Improved I/O prioritization with *io.max, io.weight*	Use io.max to control I/O bandwidth; monitor for latency-sensitive apps.
ML/AI Workloads	Better hierarchical control over device access and prioritization	Ensure kernel, device compatibility; leverage NUMA-aware scheduling.

Impact on Kubernetes Ecosystem



Challenges

- **User-Specific Challenges:**
  - **Complex Dependencies**: Large applications depend on v1-specific behavior, making migration difficult.
  - **User Adoption Barriers**: Users remain on v1 due to familiarity; they prefer hybrid setups.
  - **Behavior Changes on Upgrade**:
    - Upgrading clusters to versions where cgroup v2 is default can alter behavior, especially in handling OOM kills compared to cgroup v1.
    - https://github.com/kubernetes/kubernetes/pull/126096
  - **Compatibility and Performance Challenges:** Applications not optimized for cgroup v2 may face unexpected behavior and performance issues.

- **Kubernetes Maintenance Challenges:**
  - **CI Coverage Requirements:** Maintaining equal coverage for cgroup v1 and v2 in Kubernetes CI jobs requires significant resources and investment.
  - **Legacy Maintenance**: Older Kubernetes versions (< v1.25) are still tied to cgroup v1, requiring constant patching for bugs and CVEs.

Future Outlook

- cgroup v1 Maintenance Mode in Kubernetes 1.31.
  - No new features
  - Security fixes will provided but no assurance on the bugs
- Plan to deprecate cgroup v1 sooner.
  - https://github.com/opencontainers/runtime-spec/issues/1251
  - https://github.com/systemd/systemd/issues/30852
- Identify stack changes to accelerate the shift.
- Publicize feedback from users transitioning to v2.

Conclusion

- Confident in continued tooling enhancements for cgroup v2.
- Collaboration across Kubernetes projects will continue to refine the integration.
- Expect refinements to boost workload compatibility and observability.

Source image: https://i.kym-cdn.com/entries/icons/original/000/036/770/cover1.jpg

Thank you!

References

- https://thenewstack.io/linux-cgroups-v2-brings-rootless-containers-superior-memory-management/
- https://docs.kernel.org/admin-guide/cgroup-v2.html
- https://blog.kintone.io/entry/2022/03/08/170206
- https://zouyee.medium.com/a-tragedy-caused-by-a-single-kubernetes-command-7b6126b06513
- https://kubernetes.io/blog/2024/08/14/kubernetes-1-31-moving-cgroup-v1-support-maintenance-mode/
- https://www.redhat.com/en/blog/world-domination-cgroups-rhel-8-welcome-cgroups-v2
- https://www.perfectscale.io/blog/cgroups-and-memoryqos-w-bottlerocket
- https://cloud.google.com/kubernetes-engine/docs/how-to/migrate-cgroupv2
- https://kubernetes.io/blog/2024/08/14/kubernetes-1-31-moving-cgroup-v1-support-maintenance-mode/
- https://www.youtube.com/watch?v=dWIeIczbZHc
- https://kubernetes.io/docs/concepts/architecture/cgroups/
- https://martinheinz.dev/blog/91


KubeCon
CloudNativeCon
North America 2024