Kubernetes Monitoring

Monitoring Kubernetes at scale requires a multi-layered approach, addressing not only resource consumption but also the intricate dependencies between workloads, infrastructure, and application performance. In this guide, we explore **advanced monitoring methodologies** beyond basic metrics, diving into proactive anomaly detection, predictive scaling, cross-layer correlation, and observability best practices.

Why Advanced Monitoring is Critical for Kubernetes

Kubernetes' dynamic nature makes traditional monitoring insufficient. Advanced monitoring provides:

- **Proactive Detection:** Identifying anomalies before they become critical.
- **Predictive Insights:** Enabling preemptive scaling and failure prevention.
- **Cross-Layer Correlation:** Linking application errors to infrastructure metrics and logs.
- **End-to-End Observability:** Covering metrics, traces, logs, and user experience.

Advanced Monitoring Aspects

1 Infrastructure Monitoring

Key Objectives:

- Ensure node health and resource sufficiency.
- Detect hardware bottlenecks and failures.

Advanced Metrics to Monitor:

- **CPU Steal Time:**
  Indicates hypervisor-level contention in virtualized environments.

  ```
  rate(node_cpu_seconds_total{mode="steal"}[5m])
  ```

- **Node Pressure Conditions:**
  Detect disk, memory, or network pressure.

  ```
  kube_node_status_condition{condition=~"MemoryPressure|DiskPressure"}
  ```

- **Node Eviction Thresholds:**

```
node_memory_MemAvailable_bytes < node_memory_MemTotal_bytes * 0.10
```

Tools:

- **Node Exporter:** For hardware metrics.
- **Prometheus Thanos:** For long-term storage and query scalability.

---

2 Application Performance Monitoring (APM)

Key Objectives:

- Measure latency, throughput, and error rates for microservices.
- Correlate user experience with backend performance.

Advanced Techniques:

1. **Distributed Tracing with OpenTelemetry:**
   Track requests across services to pinpoint bottlenecks.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: opentelemetry-agent
spec:
  containers:
  - name: otel-agent
    image: otel/opentelemetry-agent:latest
```

2. **Service-Level Objectives (SLOs):**
   Define and monitor SLOs to ensure compliance with SLAs.

```yaml
slo_target:
 latency: "95th percentile < 200ms"
 availability: "99.9%"
```

3. **Error Budget Burn Rate Alerts:**
   Monitor SLO violations to prevent customer dissatisfaction.

```
burn_rate > 1.0
```

Tooling Stack:

- **Jaeger/Zipkin:** For tracing.
- **Prometheus and Grafana:** To track SLOs.

---

3 Network Monitoring

Key Objectives:

- Measure connectivity, packet loss, and traffic anomalies.
- Monitor inter-service communications for latency and errors.

Advanced Techniques:

1. **eBPF-Based Monitoring (Cilium):**
   Monitor service-to-service connections at the kernel level.

   ```
   cilium monitor --type drop
   ```

2. **Ingress Controller Monitoring:**
   Track request success rates and latency.

   ```
   rate(nginx_ingress_controller_requests[5m])
   ```

3. **DNS Query Latency:**
   Analyze CoreDNS performance.

   ```
   histogram_quantile(0.95, rate(coredns_dns_request_duration_seconds_bucket[5m]))
   ```

Tools:

- **Istio/Kiali:** For service mesh visualization and telemetry.
- **Cilium/Hubble:** For advanced network observability.

---

4 Storage Monitoring

Key Objectives:

- Avoid storage saturation and IOPS bottlenecks.
- Monitor Persistent Volume Claim (PVC) performance.

Advanced Metrics:

1. **PVC Latency and IOPS:**
   Measure storage performance per workload.

   kube_persistentvolumeclaim_resource_requests_storage_bytes

2. **Filesystem Performance:**
   Monitor disk utilization and inode usage.

   node_filesystem_avail_bytes / node_filesystem_size_bytes

Scenarios to Watch:

- **Read/Write Latency Spikes:** Indicative of degraded storage backends.
- **Excessive Inode Consumption:** Common with small file workloads.

Tools:

- **Longhorn/OpenEBS:** For containerized storage monitoring.
- **Prometheus Alerting Rules:**

  expr: kube_persistentvolumeclaim_resource_requests_storage_bytes > 80%

---

5 Predictive and Proactive Monitoring

Key Objectives:

- Identify and resolve issues before they impact production.
- Leverage ML-based anomaly detection and forecasting.

Advanced Techniques:

1. **Anomaly Detection with Prometheus:**
   Use Holt-Winters or Rate-of-Change queries.

   predict_linear(node_filesystem_free_bytes[1h], 3600) < 1

2. **Forecasting Workload Trends:**
   Predict when workloads will breach resource limits.

   avg_over_time(container_cpu_usage_seconds_total[24h])

3. **Chaos Engineering:**
   Test resilience by introducing controlled failures.

   ```
   kubectl create chaosengine -f chaos-experiment.yaml
   ```

Tooling:

- **Grafana ML Plugins:** For anomaly visualization.
- **Litmus Chaos/Gremlin:** For fault injection.

---

6 Security Monitoring

Key Objectives:

- Detect unusual activity, privilege escalations, and intrusions.
- Ensure workloads comply with security policies.

Advanced Techniques:

1. **Runtime Security Policies (Falco):**
   Detect suspicious pod activity.

   ```
   rule: "Unauthorized Shell Access"
   condition: container.name=/nginx/ and evt.type=execve
   action: alert
   ```

2. **Network Traffic Inspection:**
   Detect lateral movement or unauthorized connections.

   ```
   kubectl logs -n kube-system calico-node
   ```

Tools:

- **Sysdig Falco:** For runtime security.
- **OPA/Gatekeeper:** For policy enforcement.

---

7 Correlation Across Observability Pillars

Key Objective: Unify **metrics**, **traces**, and **logs** for holistic monitoring.

Implementation:

1. **Link Logs with Traces:**
   Include trace IDs in log entries.

   ```
   logging:
     traceId: otel_trace_id()
   ```

2. **Centralized Dashboarding:**
   Use Grafana's Unified Observability for traces and logs.
3. **Kubernetes Event Correlation:**
   Combine metrics with Kubernetes events.

   ```
   kubectl get events --sort-by='.lastTimestamp'
   ```

---

8 Scaling Observability for Large Clusters

Challenges:

- High cardinality metrics (e.g., metrics with many labels).
- Retaining historical data for forensic analysis.

Solutions:

1. **Downsample Metrics with Thanos/Cortex:**
   Retain high-resolution data for short periods and low-resolution data long-term.
2. **Implement Query Limits:**
   Prevent runaway queries in Prometheus.

   ```
   query_log_file: /var/log/prometheus_query.log
   ```

3. **Use Data Aggregators:**
   Aggregate metrics across multiple Prometheus instances.

Key Metrics for Advanced Monitoring

Below is a comprehensive summary of advanced Kubernetes monitoring metrics:

Category	Metric Name / Query	Description	Use Case
Infrastructure	rate(node_cpu_seconds_total{mode="steal"}[5m])	CPU steal time on nodes	Detect hypervisor-level

Category	Metric Name / Query	Description	Use Case
			contention
	` kube_node_status_condition{condition=~"MemoryPressure	DiskPressure "}`	Node pressure conditions
Application	histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))	95th percentile latency for HTTP requests	Measure user experience
	rate(app_errors_total[5m])	Error rate over 5 minutes	Identify application failures
Network	rate(container_network_receive_bytes_total[5m])	Network traffic received	Diagnose network saturation
	coredns_dns_request_duration_seconds_bucket	CoreDNS request latency	Optimize DNS resolution times
Storage	kube_persistentvolumeclaim_resource_requests_storage_bytes	Total storage requested by PVCs	Prevent storage saturation
Predictive	predict_linear(node_filesystem_free_bytes[1h], 3600)	Predict available disk space	Prevent disk saturation
	avg_over_time(container_cpu_usage_seconds_total[24h])	Average CPU usage over 24 hours	Forecast scaling needs
Security	falco_unexpected_shell_access	Unauthorized shell access alerts	Detect potential intrusions

Category	Metric Name / Query	Description	Use Case
Cluster State	kube_node_status_condition{condition="Ready", status="true"}	Nodes in Ready state	Ensure cluster stability

---

Real-World Insights

Scenario: Reducing API Server Latency

- **Problem:** API server request latencies exceed 500ms during scaling events.
- **Solution:**
  - Enable horizontal scaling of the API server.
  - Optimize `etcd` storage backend for faster read/write operations.

Scenario: Predicting Storage Exhaustion

- **Problem:** Disk usage spikes unpredictably.
- **Solution:**
  - Use predictive queries in Prometheus to estimate when storage will run out.
  - Migrate workloads to faster storage classes.

Scenario: Debugging Network Packet Loss

- **Problem:** Intermittent application timeouts.
- **Solution:**
  - Use eBPF-based tools (Cilium/Hubble) to trace dropped packets.
  - Optimize pod-to-pod network policies.

Best Practices for Advanced Kubernetes Monitoring

1. **Implement Multi-Tenancy Observability:**
   Provide namespace-level isolation for monitoring dashboards and alerts.
2. **Reduce High Cardinality Metrics:**
   Remove unnecessary labels or aggregate metrics where possible.
3. **Enable Multi-Cluster Observability:**
   Use tools like Prometheus Federation or Thanos to monitor multi-cluster setups.
4. **Audit and Harden Observability Systems:**
   Protect metrics and logs from unauthorized access or tampering.

5. **Continuously Optimize SLOs:**
   Use real-world performance data to adjust service-level objectives dynamically.

Conclusion

Advanced Kubernetes monitoring goes beyond basic observability by incorporating predictive insights, anomaly detection, and multi-layer correlation. Leveraging tools like **Prometheus**, **Grafana**, and **EFK** alongside modern techniques like distributed tracing and eBPF provides a robust framework for operating Kubernetes at scale.