Microsoft

Scaling Performance Analysis Tools to Hundreds of Lustre Filesystems

Ellis Wilson - Microsoft

Today’s Talk

- The Need for Advanced Logging, Metrics, and Alerting
  - Problem statement for Azure Managed Lustre Filesystem (AMLFS)
- Making Sense of Logs from Thousands of Nodes
  - Collection and Aggregation of node syslog and other log output
  - Azure Monitor Log Curation, Search, and Analytics
- Node and Cluster Performance Metrics
  - What we collect, how, and why
  - Overview of the Azure Monitor Metrics Interface(s)
- Health Monitoring and Alerting
  - Automated problem detection (before the customer notices)
  - Azure Monitor Health Interface

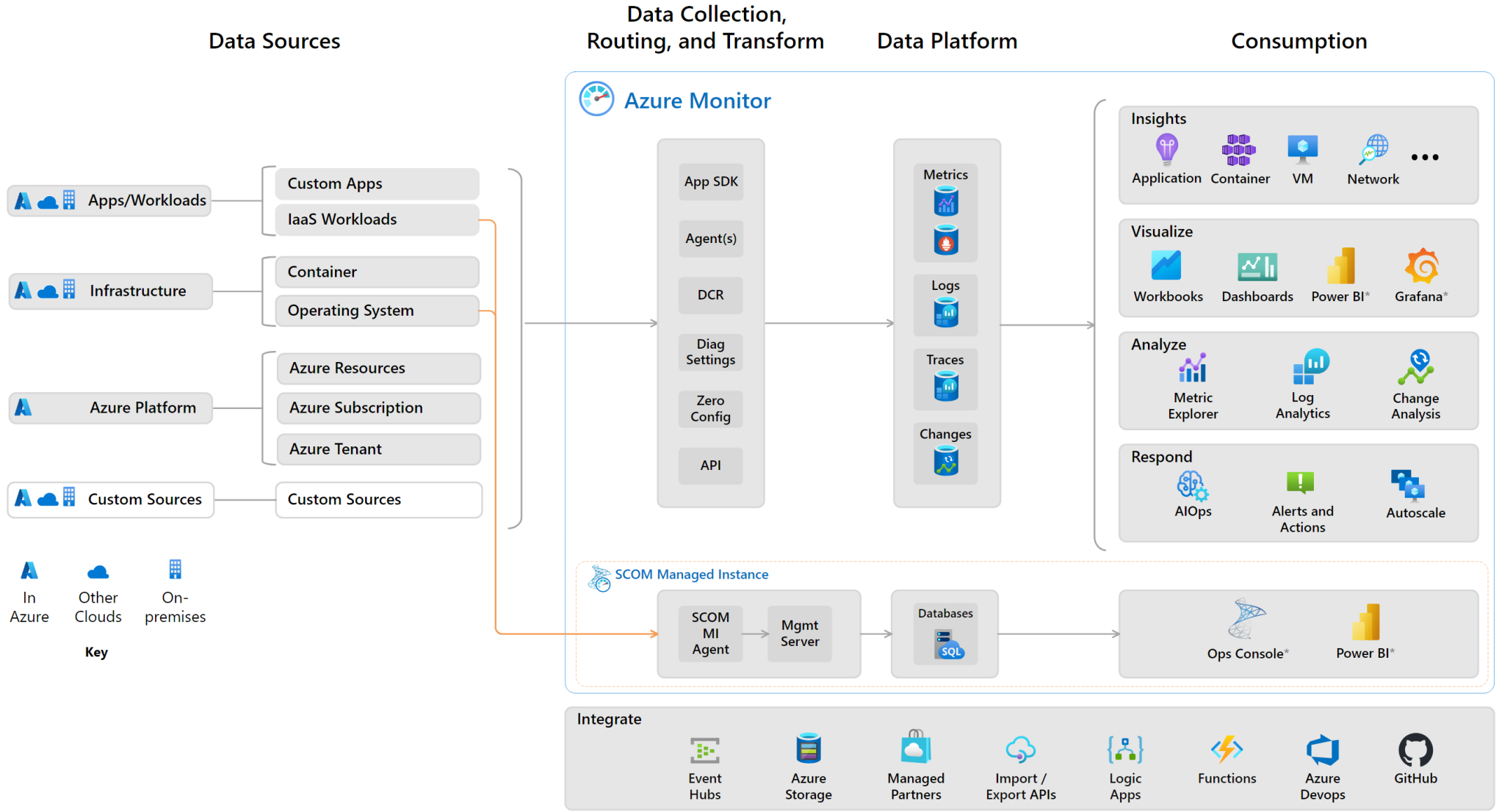
The Need for Observability in Lustre

- Typical end-user feedback tends to be … sparse:
  - “It’s not working”
  - “It’s stuck”
  - “It’s slow”
  - “I get an error”
- Lustre stats are useful, but only available at the CLI on-box
- A lot more to monitor:
  - Networking health and performance
  - CPU load
  - Memory utilization
  - Disk throughput, latency, load, capacities
  - Client connectivity
  - Cluster-wide health (e.g., heartbeats)
  - System infrastructure logs
  - Userspace crashes and core files
  - Kernel panics and vmcores
  - Lustre-related application health
  - Various versions (distro, Lustre, software)
  - …

Now Scale that to Thousands of Clusters

- Azure Managed Lustre (AMLFS) needed to design logging, metrics, and health telemetry to support tens of thousands of Lustre nodes
  - One admin per cluster is not even remotely viable
- Getting on-box to support customers is a non-starter
  - Ignoring scaling issues, privacy and security requirements preclude this
  - Cluster may be gone – transient/job-based usage
  - Must export (solely non-sensitive) logs, metrics, and health information to centralized service
- Need support for powerful querying
  - Downloading logs from ten thousand nodes and grepping won't cut it
- Visualize performance and health metrics
  - No sane way to find the one problematic OSS in a cluster without this
- Internal and Customer-facing Alerting

The Azure Monitor Data Platform



https://learn.microsoft.com/en-us/azure/azure-monitor/overview

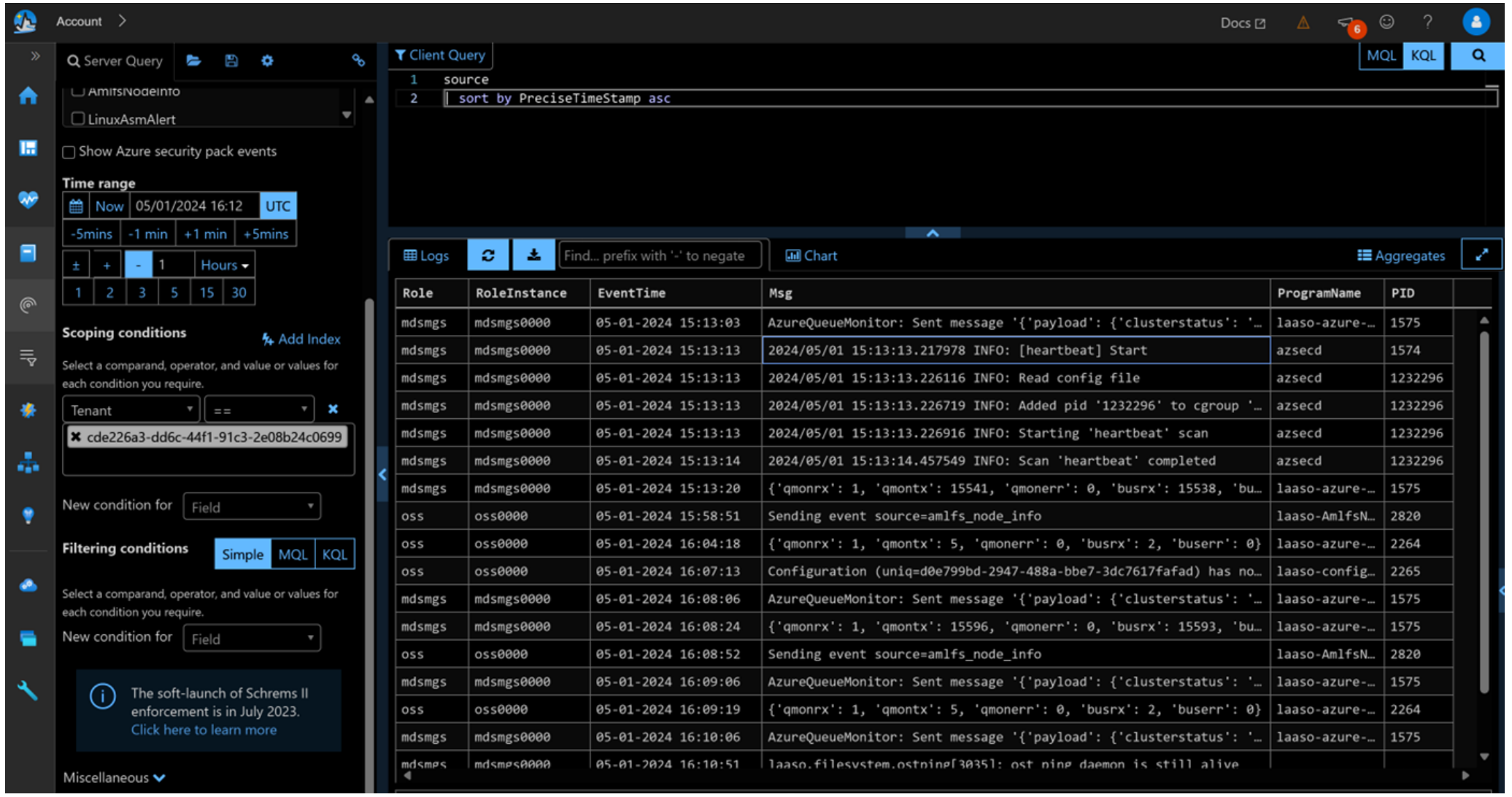
AMLFS Logging Design Criteria

- Ingest system logs from every AMLFS-side cluster node
  - MGS, MDSes, OSSes, and HSM Agents
  - All syslog traffic
- New log messages uploaded and available within low-digit minutes
- No personally-identifiable information (PII) exfiltrated
  - E.G.: File names or data
- Support powerful querying available in Azure Monitor
  - Syslog messages quantized into
    - Time, PID, ProgramName, Facility, Severity
  - Additional metadata associated with each log
    - Region, Cluster ID, Hostname, Role, RoleInstance
- Enable non-syslog log ingest for metric-like data

Azure Monitor Log Querying Interface: Dgrep

- Dgrep supports server and client-side queries
  - Typical debugging pattern:
    - Server-side selection of a cluster or a nodes logs over a time period
    - Client-side filtration down to the program, PID, or message content of interest
- Much more powerful queries become possible:
  - Example: Search all OSSes in the fleet for a specific Lustre error message over the last 3 days
  - Example: Gather logs from copytools on primary agents in a specific region
  - Example: Gather logs for a specific agent over the last 30 days at or above warning severity
- Deeper log analysis becomes possible with aggregates
  - Average, Count, Max, Min, and Sum across all previously mentioned dimensions
  - Example: Count by RoleInstance in a large cluster to find a misbehaving node
  - Example: Count by Msg to find log spam
  - Example: Sum Severity by Time and sort to find times with lots of warnings/errors

Dgrep User Interface: Simple Search by Time/Cluster



Metrics via Logs

- Client export stats are also collected as logs as they don't fit well into our normal metrics infrastructure
  - Great for locating problem clients or doing deep analysis on why "It's running slowly today"

PreciseTimeStamp	Role	RoleInstance	target	op_name	samples	min	max	sum
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.0.8@tcp.stats	statfs	11	1	71	133
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.16.100@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.16.101@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.16.102@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.5@tcp.stats	quotactl	60	1	4037	332
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.6@tcp.stats	write_bytes	2	1	4194304	6667
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.6@tcp.stats	write	2	4	652866	7114
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.6@tcp.stats	set_info	5	0	57	39
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.6@tcp.stats	quotactl	54	1	93	324
05-01-2024 16:52:03	oss	oss0001	obdfilter.lustrefs-OST0001.exports.10.17.32.7@tcp.stats	quotactl	12	1	3275	157
05-01-2024 16:52:03	mdsmgs	mdsmgs0000	mdt.lustrefs-MDT0000.exports.10.17.0.4@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	mdsmgs	mdsmgs0000	mdt.lustrefs-MDT0000.exports.10.17.0.5@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	mdsmgs	mdsmgs0000	mdt.lustrefs-MDT0000.exports.10.17.16.100@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	mdsmgs	mdsmgs0000	mdt.lustrefs-MDT0000.exports.10.17.16.101@tcp.stats	no-op-data	0	0	0	0
05-01-2024 16:52:03	mdsmgs	mdsmgs0000	mdt.lustrefs-MDT0000.exports.10.17.16.102@tcp.stats	close	1	3	285576	22

AMLFS Metrics Design

- Metrics are the second pillar of our approach to observability
  - Used for both performance analysis and cluster health triage
- Daemon metrics process on every node in every cluster collects, processes, and sends metrics to Azure Monitor Metrics
- Metrics collected at varying intervals, and via different means
  - Some metrics are gathered by running utilities (e.g., lctl, iostat)
  - Others gathered more directly via Python libraries (e.g., psutil)
- Similar time between upload and visibility to Logs (minutes)
- Two interfaces available for visualizing metrics
  - Jarvis and Grafana

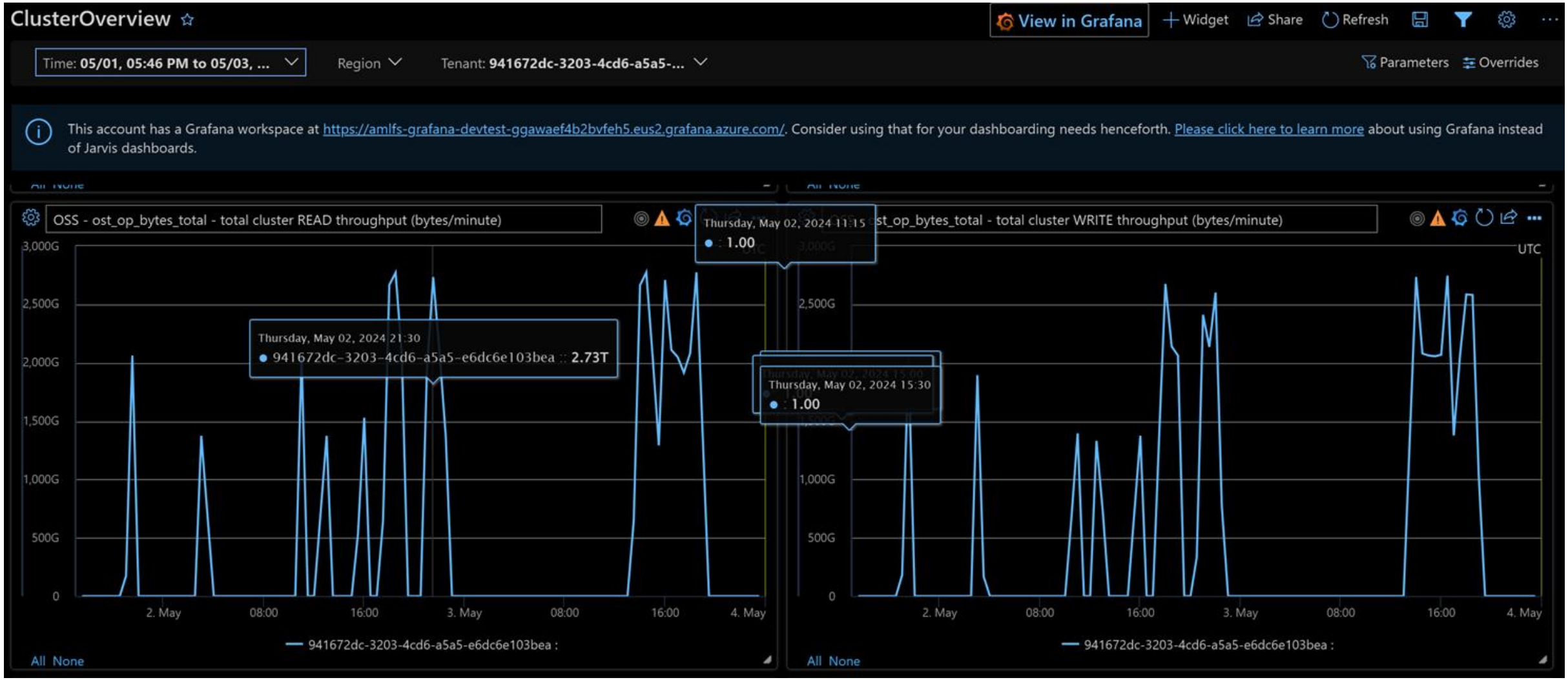
Component Metrics

- CPU
  - Overall percentages broken down akin to top (busy, idle, iowait, etc)
  - Per-core percentages
- Memory
  - Capacity in various states (total, free, available, cached, slab, etc)
- Networking
  - Total packets in/out
  - Error counts
- Disk Performance
  - Throughput, utilization, merges, iops, request sizes
- OS/Crash/Log Disk Capacities
  - Fullness by bytes (total, used, free, %)
- Data Disk Capacities
  - Fullness by bytes (total, used, free, %)
  - Fullness by inodes (total, used, free, %)

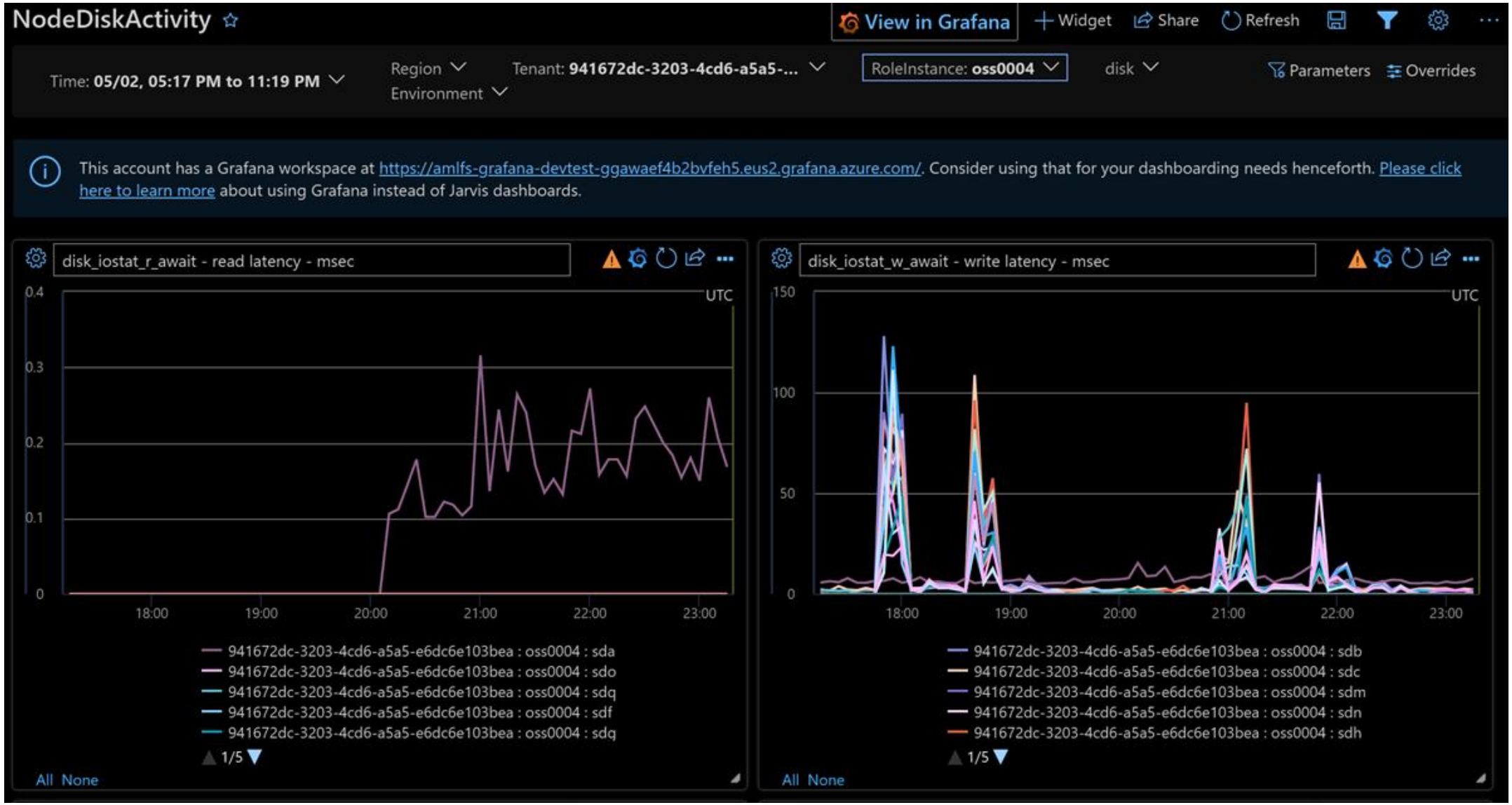
Lustre-specific Metrics

- OSS:
  - Request statistics by request type
    - Total requests
    - Total bytes moved in requests
    - Min/max op size since restart
    - Min/max op latency since restart
  - Total evicted clients
  - Total connected clients
- All Cluster Nodes
  - Event Alert (e.g., node restart)
  - AMLFS Heartbeat
  - AMLFS Version
- MGS/MDS:
  - Same as OST, plus:
  - HSM request count by type (restore, remove, archive)
  - HSM current/completed/errored requests
  - HSM registered agents
  - Changelog unread events
  - Changelog size
  - LDLM MGS Timeouts

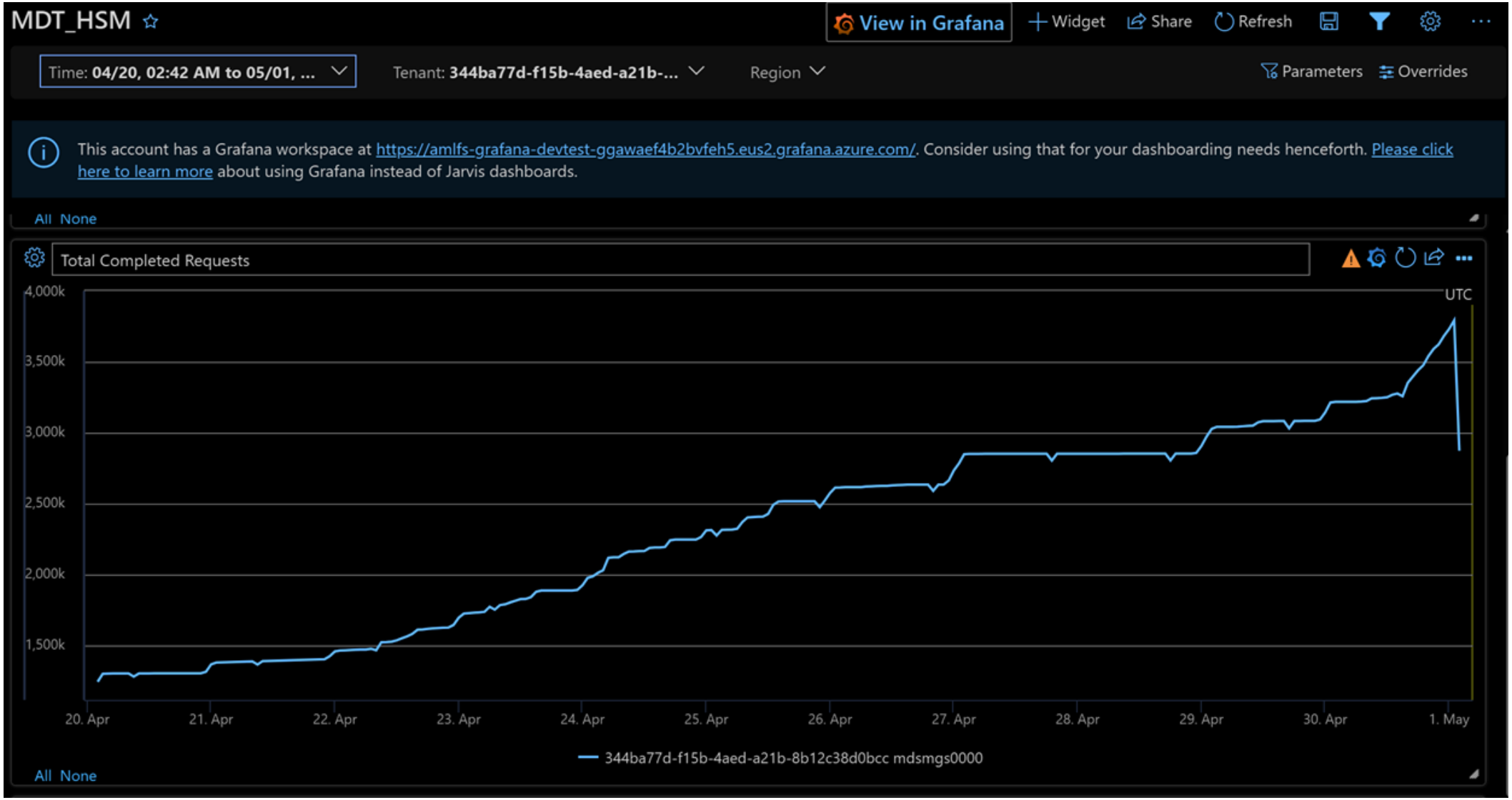
Geneva Dashboard Interface: Cluster Diagnostics



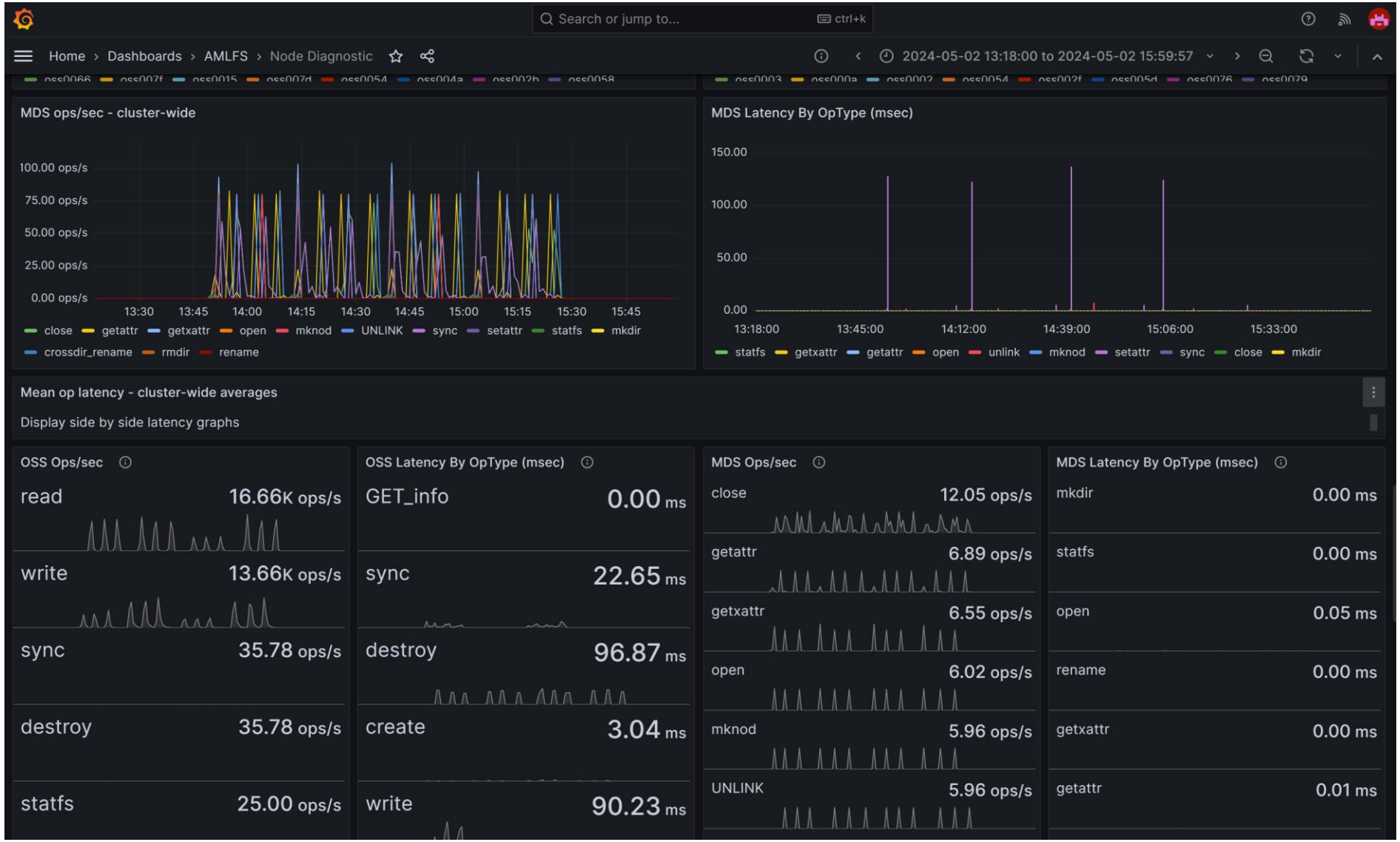
Geneva Dashboard Interface: Disk Stats for One Node



Geneva Dashboard Interface: HSM Metrics



Grafana Dashboard Interface: Per Node Stats



Cluster Health Monitoring and Alerts

- Logs and metrics are great – *when you know you have a problem*
  - But this tends to occur after an angry Lustre user has escalated a complaint
    - Which means it's guaranteed to be 3am
- Knowing there is a problem prior to or simultaneously with a customer being made aware is infinitely better
- Health Monitors and Alerts are the third pillar of observability
- Most of this leverages existing metrics discussed before
  - Take existing stats that we're already sending to Azure Monitoring, and set rules on them
  - If a stat is found to fail the test so many times in a row or for known duration, automatically create an incident ticket and autopopulate information about the problem
    - Engineers can jump right from the incident via links to logs and metrics around that time to triage/diagnose

AMLFS Monitors

- We have rules that raise alerts and/or create incidents of varying severities for the following:
  - Missing Heartbeats
  - Heartbeat indicates Degraded cluster
  - Unexpected reboot/shutdown events
  - MGT/MDT/OST capacity available too low
  - Non-data disk capacity available too low
  - Coredumps observed
  - Read and Write op latencies too long
  - Total CPU percentage too high
  - Free memory too low
  - *We continue to adjust and add more as customer issues find cases we'd like to detect early*
- Most of these only "pop" if they hit many times over some duration

Azure Monitor Health Interface: Cluster Getting Full

Wrapping Up

- *In AMLFS we had to design an observability infrastructure to support thousands of clusters, and hundreds of thousands of cluster nodes, including nodes that may not be around when you need to debug*
- Logs, Metrics, and Alerts comprise our three pillars of observability
  - Support Log aggregation, parsing, querying, and statistics via Azure Monitor Logs/Dgrep
  - Support Metric gathering and aggregation from various on-box utilities and visualization
  - Support Monitors/Alerts and automated incident creation when certain metrics go south
- In most cases, we can avoid getting on-box until we know exactly what we plan to do to fix the problem
  - Less time on-box, less room for user-error, far less exposure to customer data
- *Again, looking for feedback and others experiences in this area!*

Microsoft

Thanks! Questions?