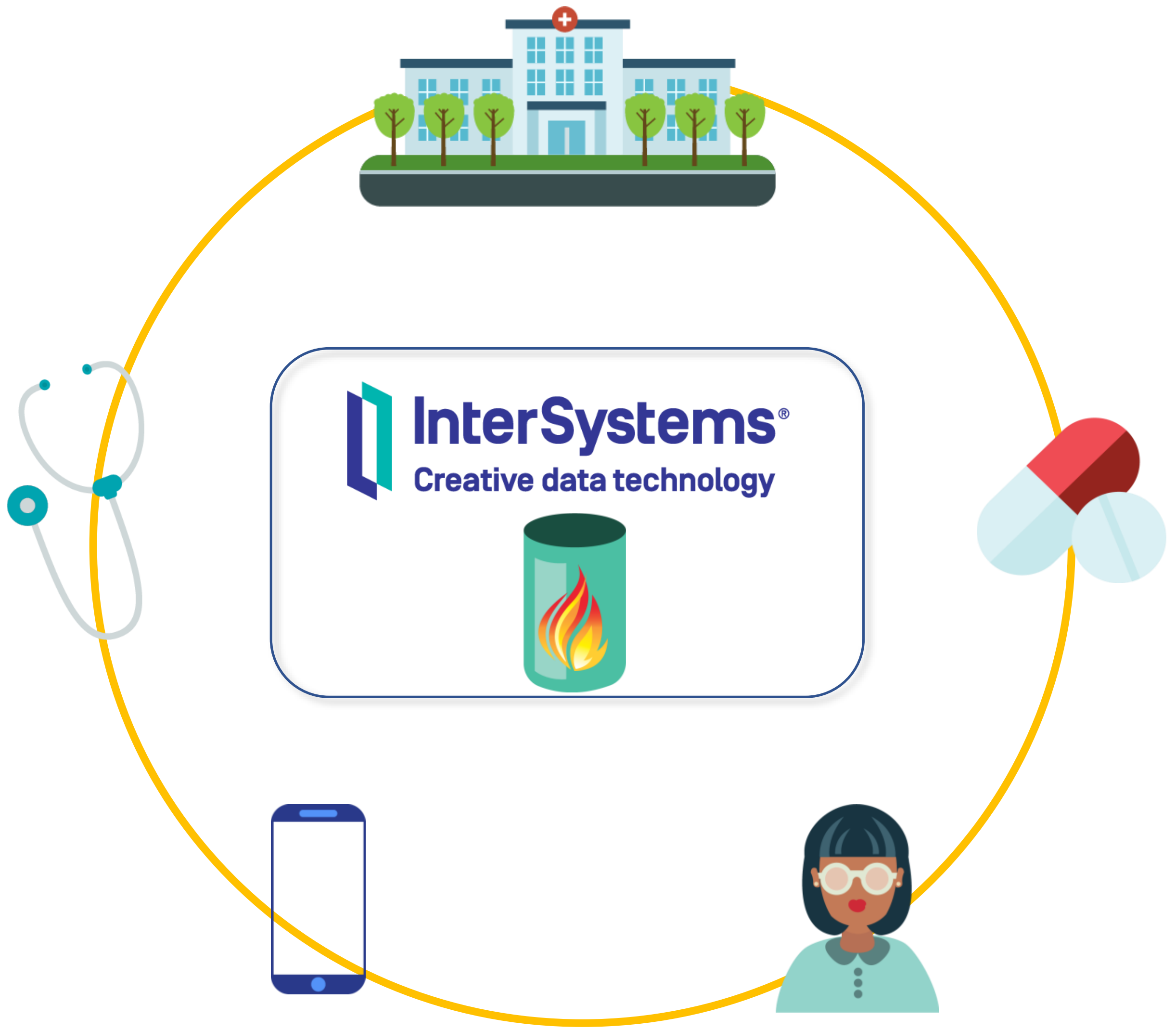
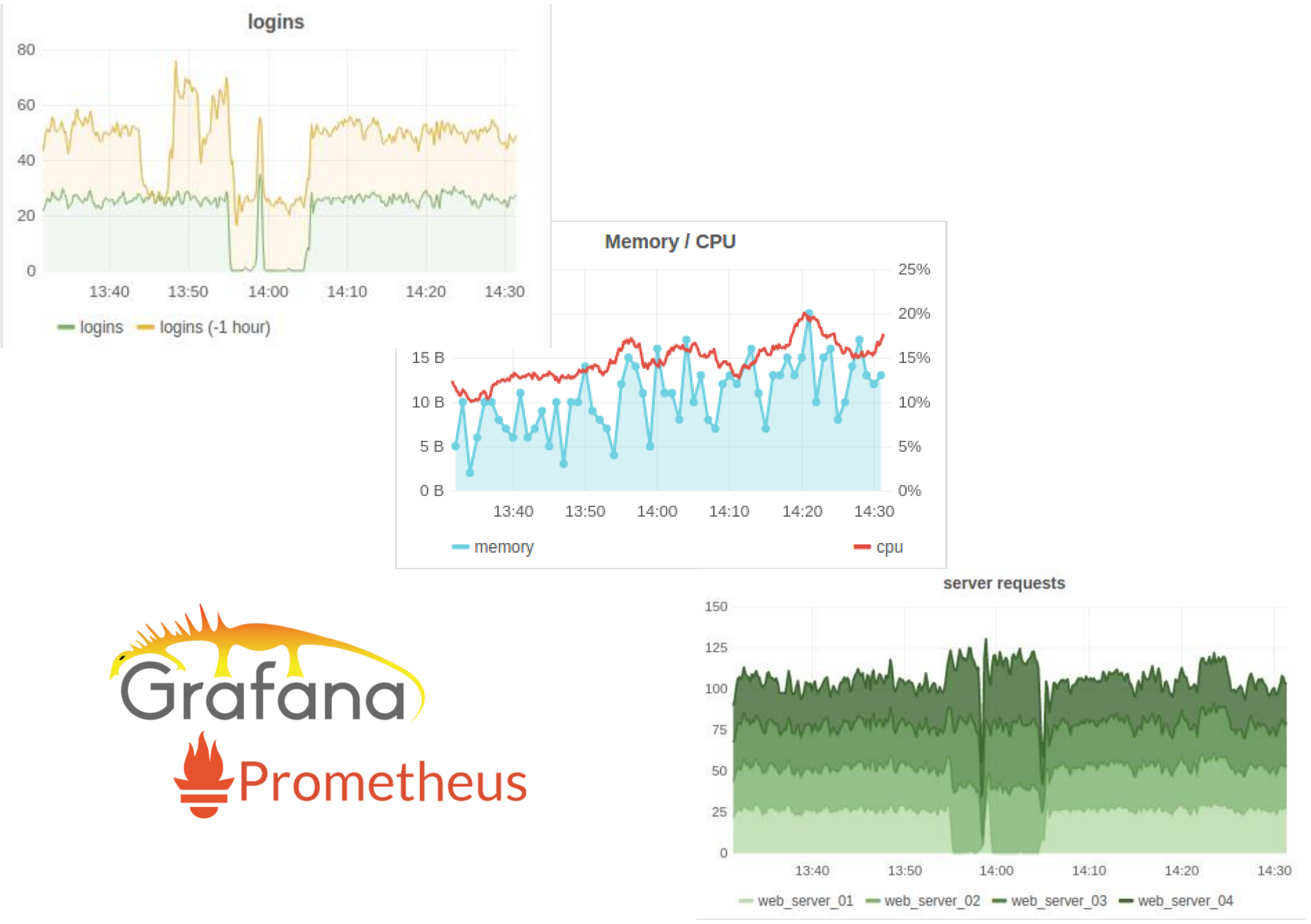
Observability, Audit, Monitoring

System Alerting and Monitoring (SAM)



Pierre-Yves Duquesnoy
Senior Sales Engineer



Grafana

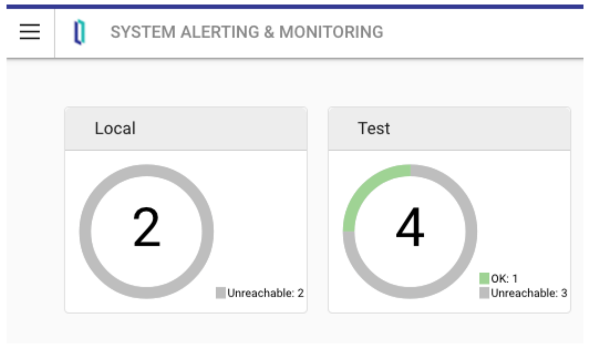
Prometheus

SAM

System Alerting & Monitoring

What is it?

- System Alerting and Monitoring (SAM) is a unified cluster monitoring tool



Benefits?

- A single pane for all your IRIS instances
- Groups instances in clusters
- Super easy implementation
- Upgradable Community Edition license for enterprise features
- Built-in metrics exporter in IRIS instances: nothing to install on Monitored Instances (2020.1+)

How?

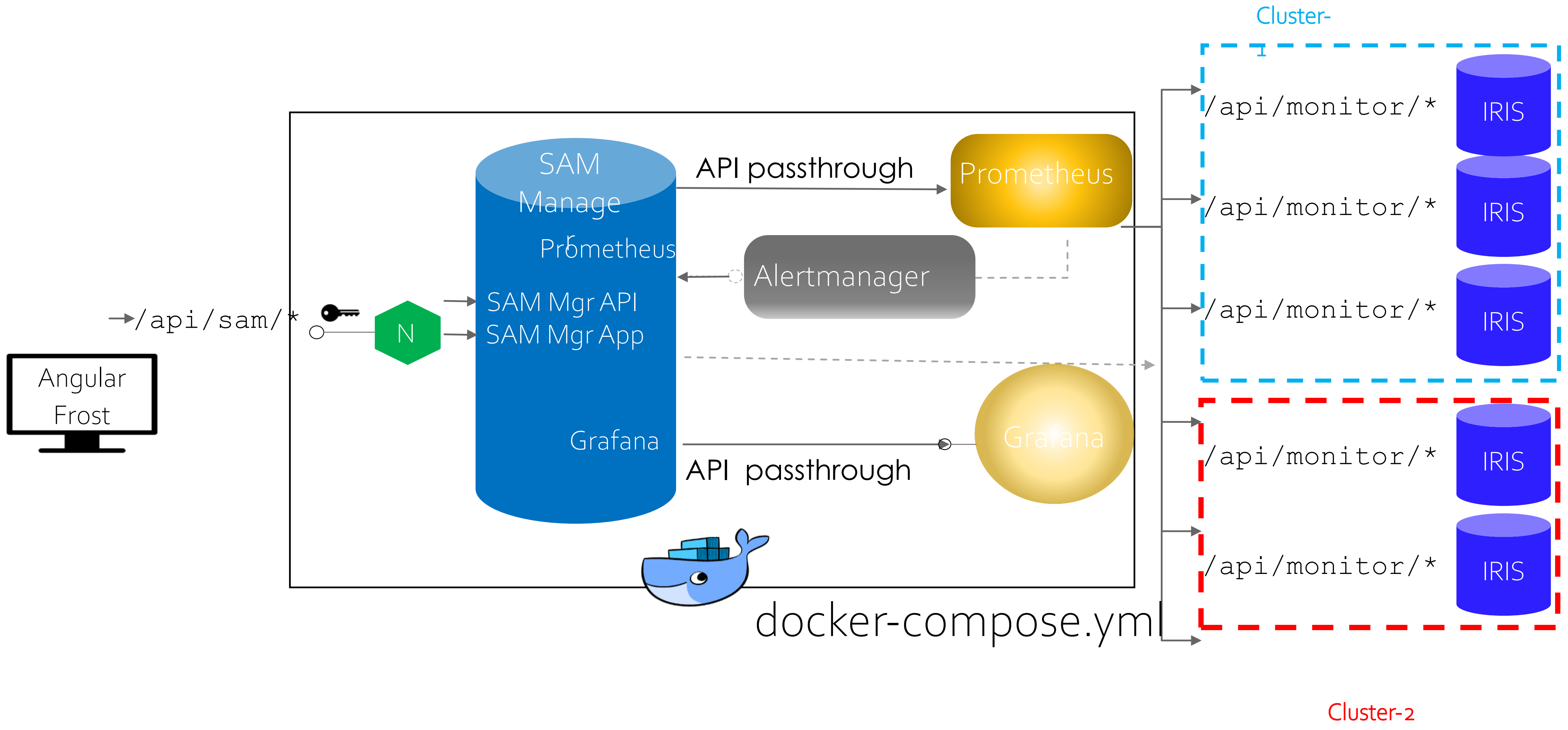
- Leveraging industry standard open source software
  - Prometheus - a CNCF graduated project
  - Grafana – the most widely used metrics visualizer
  - Bundled in a Docker compose to run

What is System Alerting and Monitoring?

A native and open solution to InterSytem IRIS monitoring, providing

- A Web-UI
  - Define clusters and instances
  - Monitor instances health status
  - View Alerts
  - Set new metric rules
  - Provides metrics visualization
- Leveraging open source known, documented and well maintained components
    - Alertmanager v0.20.0
    - Grafana v6.7.1
    - Nginx 1.17.9
    - Prometheus v2.17.1
    - SAM Manager 1.0
  - Extensible
  - With much more to come…

The SAM Solution – General Overview



SAM Server Side

- Included & Enabled in IRIS 2020.1+

- Built on top of existing Monitoring tools
  - System Monitor
    - On by default, Single system State value “OK”,”WARNING”,”CRITICAL” +System Metrics (CPU, Lock Table…)

- Prometheus exporter
  - REST API
    - /api/monitor/metrics
    - /api/monitor/alerts

System Monitor

- maintains a *single* system-health value
  - $System.Monitor.State()
  - ON by default
  - Samples system status and resource usage indicators (lck-table-%full, CPU-warning, ECP, etc.)
  - Generates notifications and alerts
    - System Alerts (severity 2) Warnings (severity 1), OK (severity 0)
      - uses a fixed set rules to evaluate collected values and identifies deviating metrics
      - writes to messages.log
    - Log Monitor (^MONITOR), ON by default
      - messages.log and
      - alerts.log

  - SYS.Monitor.SystemSensors
    - SYS.Monitor.DashboardSensors
      - SYS.Monitor.SAM.Sensors – to collect all available sensors without storing them; polled by SAM Mgr/client
        - GetSensors() every 30 seconds

SAM Server Side Metrics

- /api/monitor/metrics
  - Simple Key/values
    - Iris_cpu_usage, iris_glo_ref_per_sec, iris_db_latency …
  - Allows Definition of additional Application Metrics

SAM Virtual Appliance

- Set of docker containers started with docker-compose
- SAM Manager
  - And IRIS Instance, with the main Application
  - Maintains Data History
- Nginx
  - Web Server, Acces control,
- Prometheus
  - TS Database, scraper, visualizer, integrator
- Grafana
  - Visualization
- Alert Manager
  - Deduplication, routing of alerts

Initial Setup

- Get the tarball sam-<version>.tar.gz for WRC -> Components or GitHub
  - https://github.com/intersystems-community/sam
- Uncompress and untar with:
  - `tar zpxvf sam-<version>.tar.gz`
- Execute: `./start.sh`
  - First be sure that ./config folder and subfolders have RW permissions.
- To stop SAM just execute: `./stop.sh`

Define a Cluster

- A Cluster is a set of instances tom manage together
- From SAM portal, just define new cluster and start adding IS IRIS instances to it
  - By default, IRIS 2020.1+ are pre-configured to be automatically accesible from SAM through /api/monitor API
    - Just be sure Access is Unauthenticated to this API
- Configure basic settings
  - # of days (1 to 30) for SAM to store data

Define Alerts

- Prometheus Alert Rules to indicate SAM to fire an alert
  - Prometheus Query Language syntax
  - More info: https://prometheus.io/docs/prometheus/latest/querying/basics/
  - Basic syntax:

    ```
    metric_name{cluster="cluster_name",label(s)}>value
    ```
  - Allows Arithmetics and Logical Operators

```
iris_cpu_usage{cluster="test"}>90
(iris_db_size_mb{cluster="test",id="USER"}/iris_db_max_size_mb{cluster="test",id
="USER"})*100>90
iris_ecp_conn{cluster="production"}<1 or iris_ecp_conn{cluster="production"}>20
iris_system_alerts_new{cluster="test"}>=1 and
iris_system_monitor_health_state{cluster="test"}!=0
```

Add an Application Metric

- Create a SubClass of %SYS.Monitor.SAM.Abstract
- Define a “PRODUCT” to group metrics
- Implement GetSensors()
  - Use SetSensor(Sensor , Value , Item As %String = "")
    - Do ..SetSensor(“TubesReceived”,+$Get(^User.TubesReceivedD))
- Add Class to Configuration

```
%SYS>Do ##class(SYS.Monitor.SAM.Config).AddApplicationClass(ClassName,Namespace)
```

- **Add Permissions** to /api/monitor Web Application
  - To execute code and Access data in “Namespace”
- Test with http://ServerIP:Port/csp/monitor/metrics

A bit on SAM Licensing

SAM Community Edition

- 5 concurrent users
- 8 cores
- 10 GB Database

SAM Enterprise

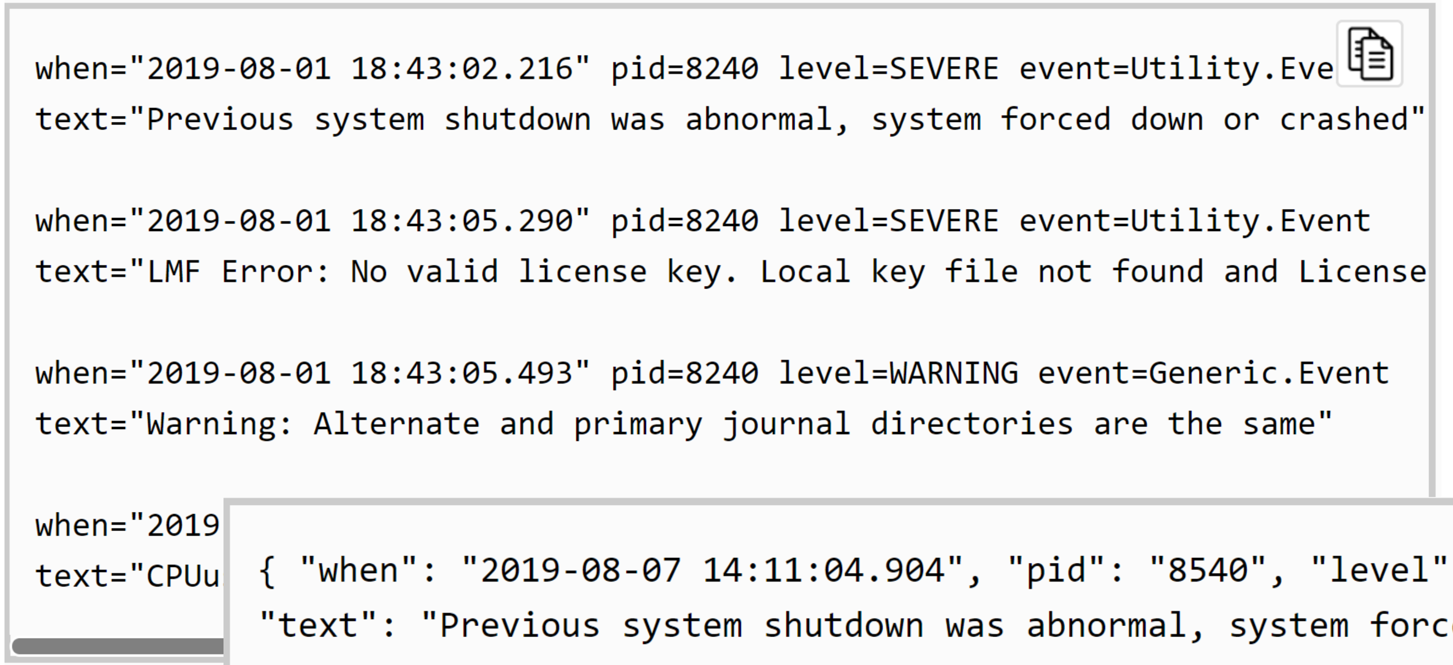
- Concurrent users and cores per licence, unlimited dabase size
- Leverages InterSystems Mirroring
- (no Sharding included)

SAM roadmap

This year	Next Year	Beyond
• IRIS SQL metrics	• Structured Logging support	• Leveraging ML
• IRIS Interoperability metrics	• IRIS backend thresholds	• Autonomic Computing
• Deployment by ICM & IKO	• System Performance, Technology Architects and WRC support	
• Documentation for APM	• Management Capabilities	

SAM Workshop

- Install SAM
- Start SAM with: `docker-compose up -d`
- Connect to Mgmt Portal and change the _SYSTEM password
- Connect to SAM
  - `http://127.0.0.1:8080/api/sam/app/index.csp`
- Define a Cluster, Add an IRIS Server_to_monitor
- Create an Alert in SAM
- Login to Server_to_Monitor and generate Load
- Login to Server_to_Monitor and generate LockTableFull
- Create Custom App Metric

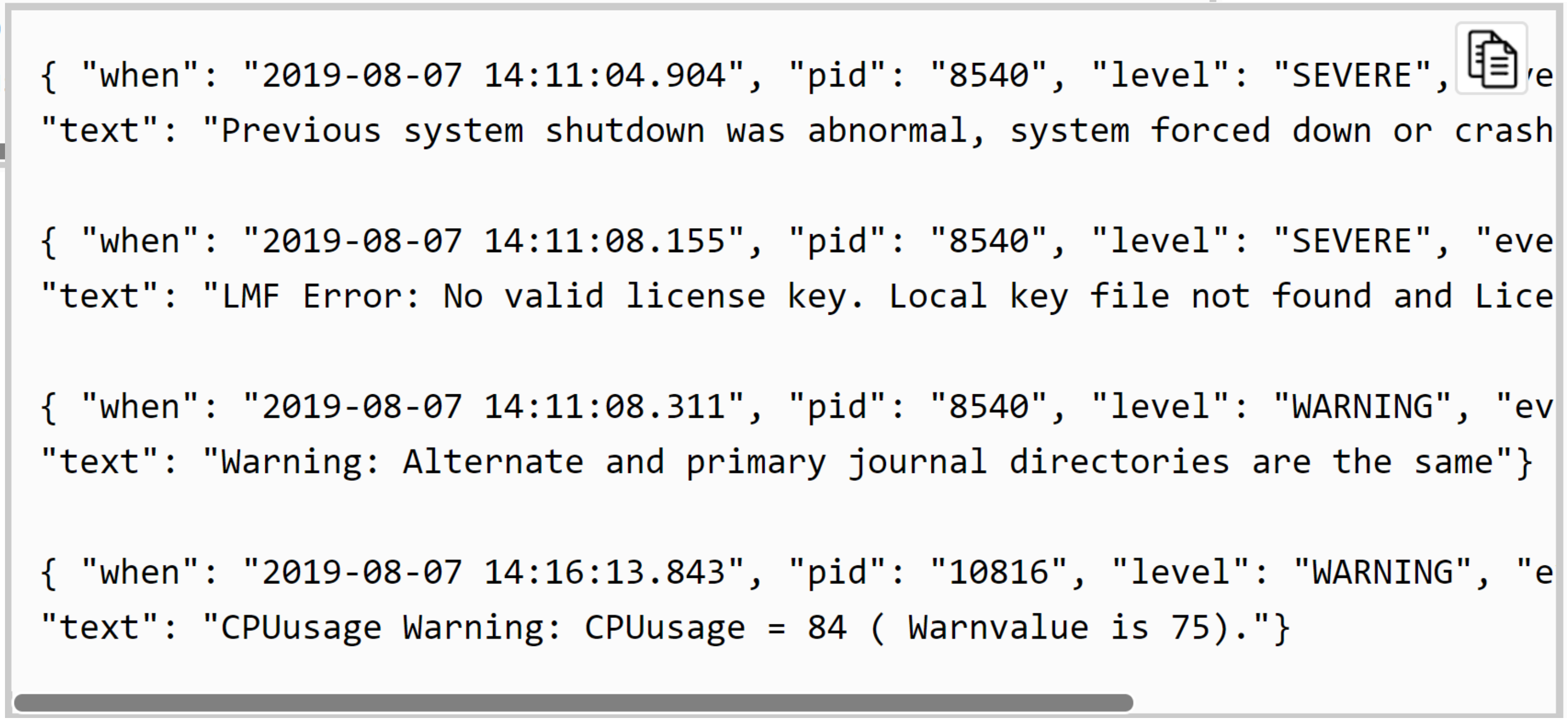


```
when="2019-08-01 18:43:02.216" pid=8240 level=SEVERE event=Utility.Eve
text="Previous system shutdown was abnormal, system forced down or crashed"

when="2019-08-01 18:43:05.290" pid=8240 level=SEVERE event=Utility.Event
text="LMF Error: No valid license key. Local key file not found and License

when="2019-08-01 18:43:05.493" pid=8240 level=WARNING event=Generic.Event
text="Warning: Alternate and primary journal directories are the same"

when="2019
text="CPUu
```



```
{ "when": "2019-08-07 14:11:04.904", "pid": "8540", "level": "SEVERE",
"text": "Previous system shutdown was abnormal, system forced down or crash

{ "when": "2019-08-07 14:11:08.155", "pid": "8540", "level": "SEVERE", "eve
"text": "LMF Error: No valid license key. Local key file not found and Lice

{ "when": "2019-08-07 14:11:08.311", "pid": "8540", "level": "WARNING", "ev
"text": "Warning: Alternate and primary journal directories are the same"}

{ "when": "2019-08-07 14:16:13.843", "pid": "10816", "level": "WARNING", "e
"text": "CPUusage Warning: CPUusage = 84 ( Warnvalue is 75)."}
```

Structured Logging

Structured Logging in a nutshell

- Supported in IRIS 2020.2
- Channels all the logs into single Machine Readable format
- Ideal for 3rd-party analysis tools
- Information from
  - Messages.log (like Caché cconsole.log)
  - Information from Audit database
- 2 Possible formats
  - Name/Value Pairs
  - JSON

Examples

- Name/Value Pairs

```
when="2019-08-01 18:46:10.493" pid=11948 level=WARNING
event=System.Monitor text="CPUusage Warning: CPUusage = 79 (
Warnvalue is 75)."
```

- JSON

```
{ "when": "2019-08-07 14:16:13.843", "pid": "10816", "level":
"WARNING", "event": "System.Monitor", "text": "CPUusage
Warning: CPUusage = 84 ( Warnvalue is 75)."}
```

Name/Value pairs

- Each Line contains Name=Value pairs separated by Spaces
- Some or all of following (in bold the required fields):
  - Host
  - Instance
  - **When**
  - **Pid**
  - **Level**
  - **Event**
  - **Text**
  - Source
  - Type
  - Group
  - namespace

Enabling Structured Logging

- Interactive:
  - %SYS> Do ^LOGDMN
- API
  - Class SYS.LogDmn

```
1) Enable logging
2) Disable logging
3) Display configuration
4)  Edit configuration
5) Set default configuration
6) Display logging status
7) Start logging
8) Stop logging
9) Restart logging
```

Configuration details

- Minimum Log Level
  - -2: detailed debug messages (hex dumps)
  - -1: less detailed debug messages
  - 0: informational messages, including all audit events
  - 1 (default): warnings [problems that may need attention]
  - 2: severe errors
  - 3 fatal errors [system failure]
- Pipe command:

```
irislogd -f c:/myfilename.log
```

- Format: NVP /JSON
- Interval (default is 10 seconds)

Irislogd

- Executable to generate the logs, called by IRIS
- Options

Argument	Purpose
-d	Diagnostic and error messages
-e errfilename	File for error and diagnostics
-f logfilename	File for los messages
-h hostname	Includes the gien hostname in structured log file
-i irisinstance	Includes the given instance name in strucutrd log file
-s	**Write log messages to the Unix syslog facility**
No -s no -f	Writes to stdout

Workshop

- Log into IRIS_to_monitor instance
- %SYS> Do ^LOGDMN
- Select output to shared volume/bind /shared
- Force an alert (lock table full)
- Check log content