iRODS and Observability

Arcot (Raja) Rajasekar
rajasekar@unc.edu
The University of North Carolina
at Chapel Hill

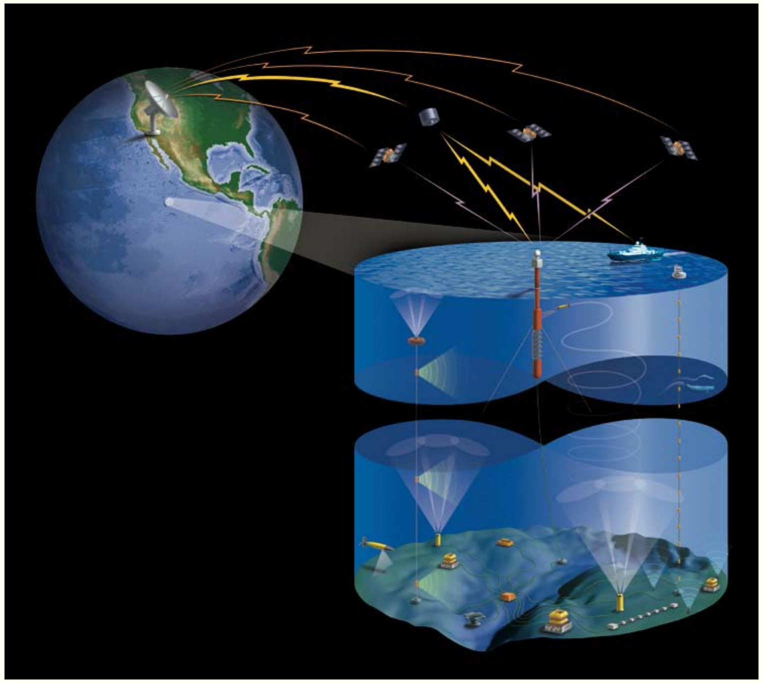
THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

D·I·C·E

UNC SCHOOL OF INFORMATION AND LIBRARY SCIENCE

Outline

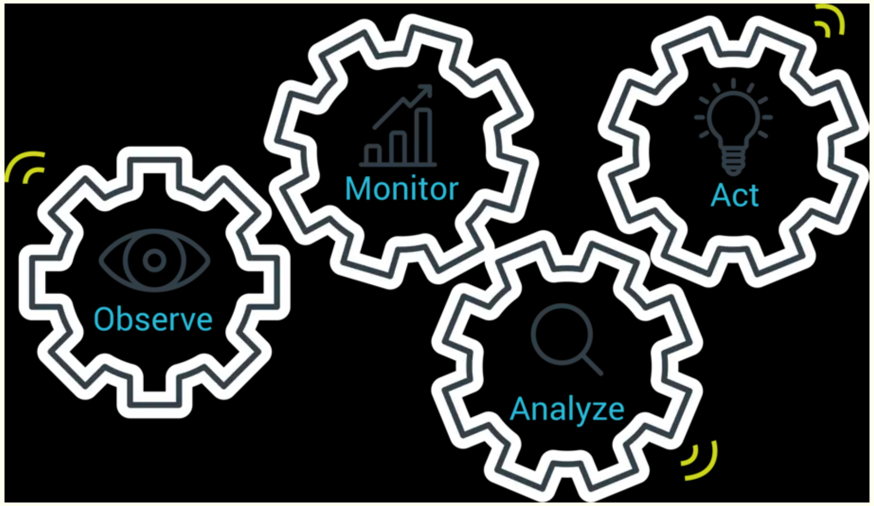
- Observability
- Current Tracking in iRODS
- Realizing Observability through iRODS
- Q&A



Observability

What is Observability?

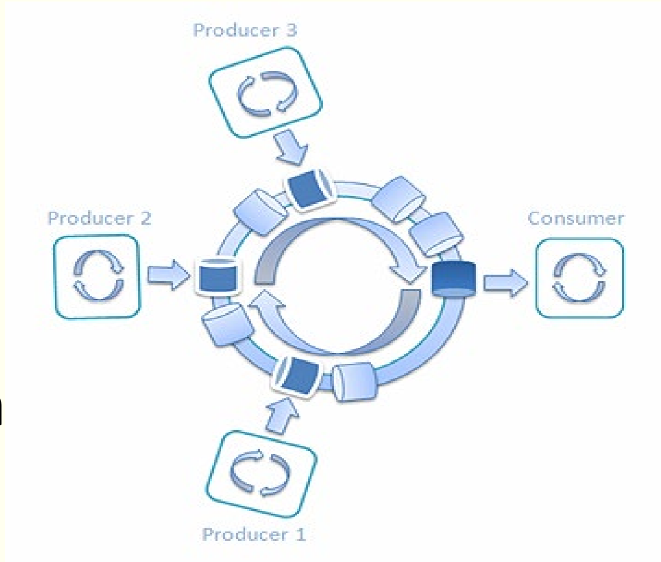
- Observability is the ability to understand what is happening inside of a system from the knowledge of its external outputs.
- It originates from engineering, particularly from control theory
- In a dynamic system one can estimate the state of a system from monitoring the observables.
- To Observe is to Control



From https://www.devo.com/

Observability in Software System

- Observability helps understand and answer specific questions about what's happening in highly distributed systems
- Observability empowers cross-functional teams (IT Admins, system developers, application engineers, managers) to identify problems before they even manifest or become unmanageable
- Observability enables you to realize what is slow or broken, and to quickly figure out what needs to be done to improve performance
- Observability is a measurement that can pinpoint bottle necks, degrading performance, improvable usage patterns, and predict failure.
- Observability helps increase performance, availability, resiliency and user satisfaction

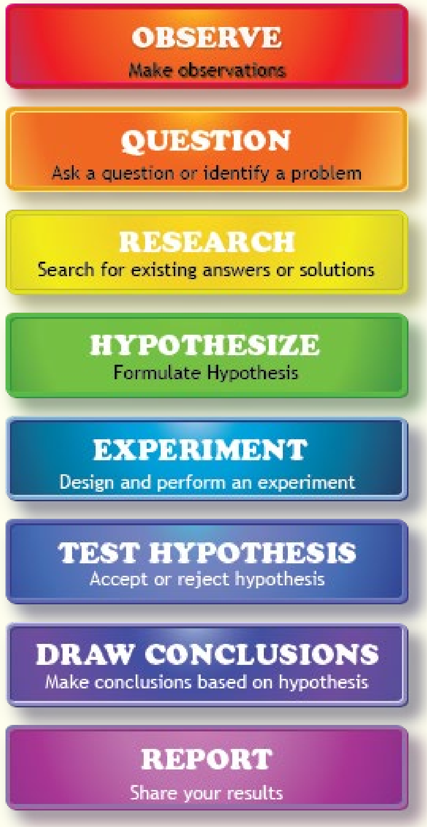


Observability in Cyber Infrastructure

- With increased use of
  - Chaining of Micro-services & Web services
  - Multi-party software
  - Agile programming
  - Automatic updates, bug fixes and service releases
  - Containers
  - Dynamic Libraries and packages
  - Multi-lingual scripts
  - Cloud services
  - Large networks and diverse hardware
  - Distributed computing and storage
  - Complex security structures
  - Virtual Machines
- Its no longer your grandpa's slide rule and calculator
- A simple 'click' or call can span a large complex software and hardware conglomerate in milliseconds to deliver the result
- An innocuous update to an obscure package can have a cascading effect
- Finding problems and correcting them can be a nightmare
- Going beyond that, predicting failure and degradation of services can be highly challenging
- Observability is the name of the game

Observability in Software System

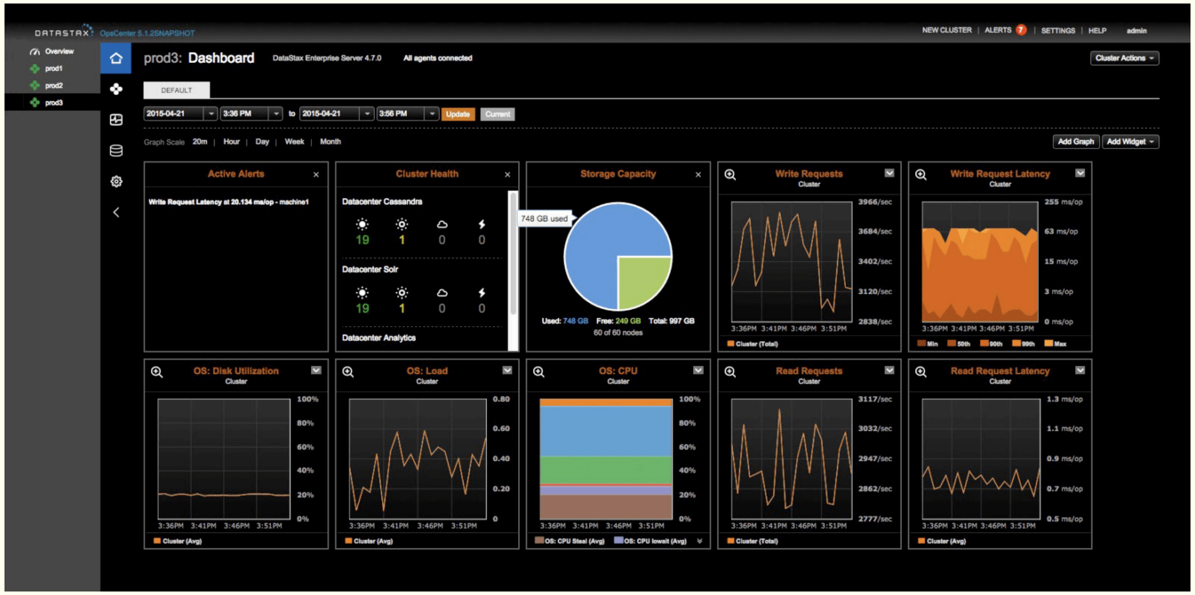
- Observability is the practice of achieving actionable insights
- The aim is to understand
  - When an event or issue happened
  - Why it happened
  - Where it happened
  - Who or What is responsible
  - How to recover
- Hopefully before it happened
  - Predict Vs Diagnose
- Monitoring: Data generated by well-instrumented software systems provide the clues
- Machine Learning & Data Analytics are part of the solution


OBSERVE
Make observations
QUESTION
Ask a question or identify a problem
RESEARCH
Search for existing answers or solutions
HYPOTHESIZE
Formulate Hypothesis
EXPERIMENT
Design and perform an experiment
TEST HYPOTHESIS
Accept or reject hypothesis
DRAW CONCLUSIONS
Make conclusions based on hypothesis
REPORT
Share your results

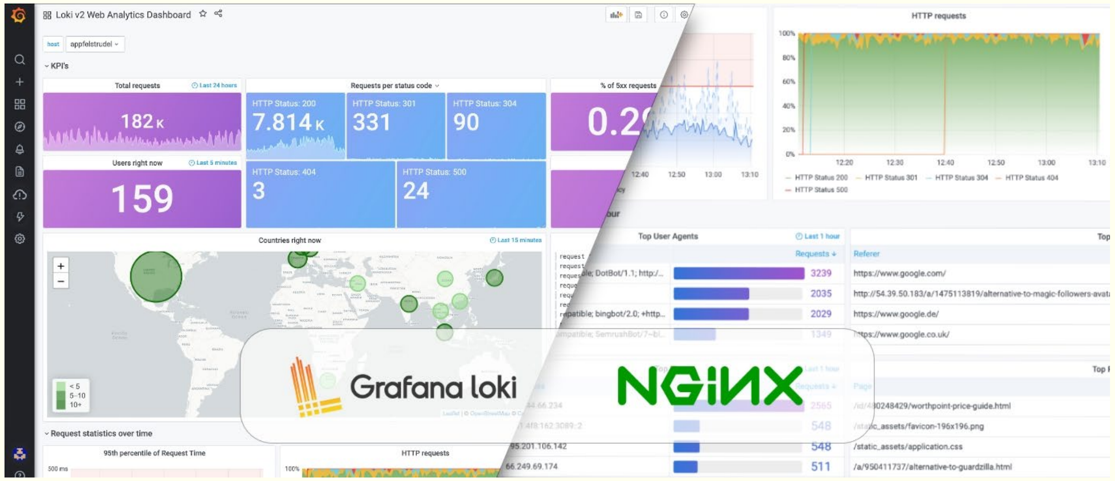
From: https://sciencenotes.org/steps-scientific-method/

Example Observability Systems

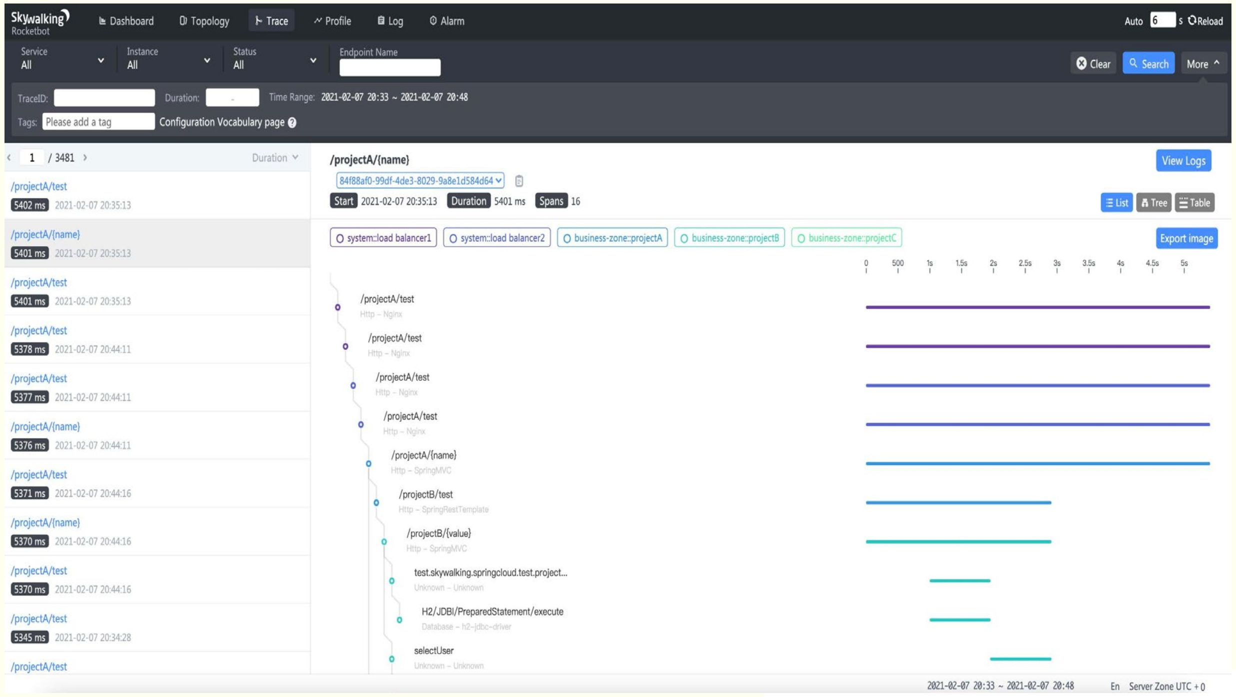
- **DataStax**



- **Grafna Dashboard**



Example Observability Systems



Skywalking Rocketbot — Dashboard · Topology · Trace · Profile · Log · Alarm — Auto 6 s Reload

Service: All · Instance: All · Status: All · Endpoint Name: — Clear · Search · More

TraceID: · Duration: · Time Range: 2021-02-07 20:33 ~ 2021-02-07 20:48

Tags: Please add a tag · Configuration Vocabulary page

1 / 3481 · Duration

```
/projectA/test      5402 ms  2021-02-07 20:35:13
/projectA/{name}    5401 ms  2021-02-07 20:35:13
/projectA/test      5401 ms  2021-02-07 20:35:13
/projectA/test      5378 ms  2021-02-07 20:44:11
/projectA/test      5377 ms  2021-02-07 20:44:11
/projectA/{name}    5376 ms  2021-02-07 20:44:11
/projectA/test      5371 ms  2021-02-07 20:44:16
/projectA/{name}    5370 ms  2021-02-07 20:44:16
/projectA/test      5370 ms  2021-02-07 20:44:16
/projectA/test      5345 ms  2021-02-07 20:34:28
/projectA/test
```

/projectA/{name} — View Logs

84f88af0-99df-4de3-8029-9a8e1d584d64

Start 2021-02-07 20:35:13 · Duration 5401 ms · Spans 16 — List · Tree · Table

system::load balancer1 · system::load balancer2 · business-zone::projectA · business-zone::projectB · business-zone::projectC — Export image

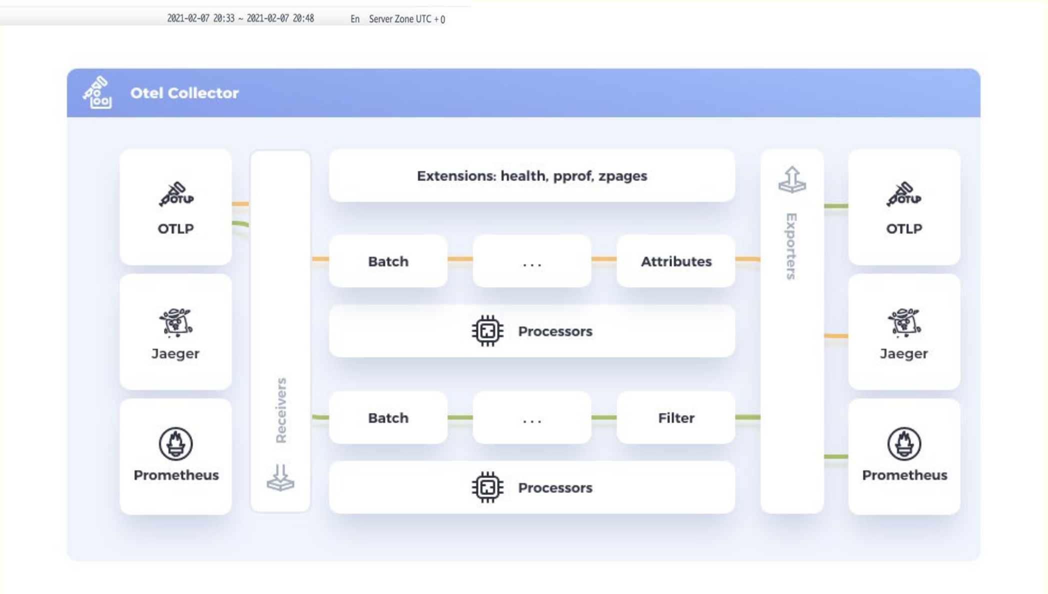
0 · 500 · 1s · 1.5s · 2s · 2.5s · 3s · 3.5s · 4s · 4.5s · 5s

```
/projectA/test
  Http – Nginx
 /projectA/test
   Http – Nginx
  /projectA/test
    Http – Nginx
   /projectA/test
     Http – Nginx
    /projectA/{name}
      Http – SpringMVC
     /projectB/test
       Http – SpringRestTemplate
      /projectB/{value}
        Http – SpringMVC
       test.skywalking.springcloud.test.project...
         Unknown – Unknown
        H2/JDBI/PreparedStatement/execute
          Database – h2-jdbc-driver
       selectUser
         Unknown – Unknown
```

2021-02-07 20:33 ~ 2021-02-07 20:48 · En · Server Zone UTC + 0

← Apache Skywalker

Open Telemetry →



Three Pillars of Observability

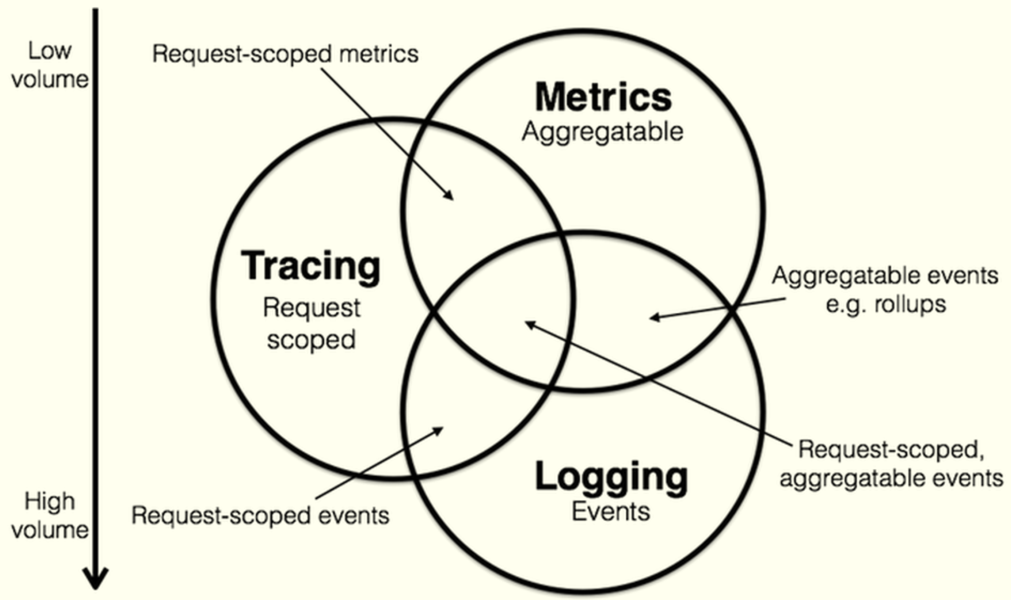
- Logging: collects information about events happening in the system and helps find unexpected behavior
- Tracing: collects information to create an end-to-end view of how transactions are executed in a distributed system. Tracing can recognize a problem through comparing and contrasting.
- Metrics: provide a real-time indication of how the system is running. Metrics can be leveraged to build alerts, allowing proactive reaction to unexpected values

From: https://www.humio.com/

Two More Pillars:

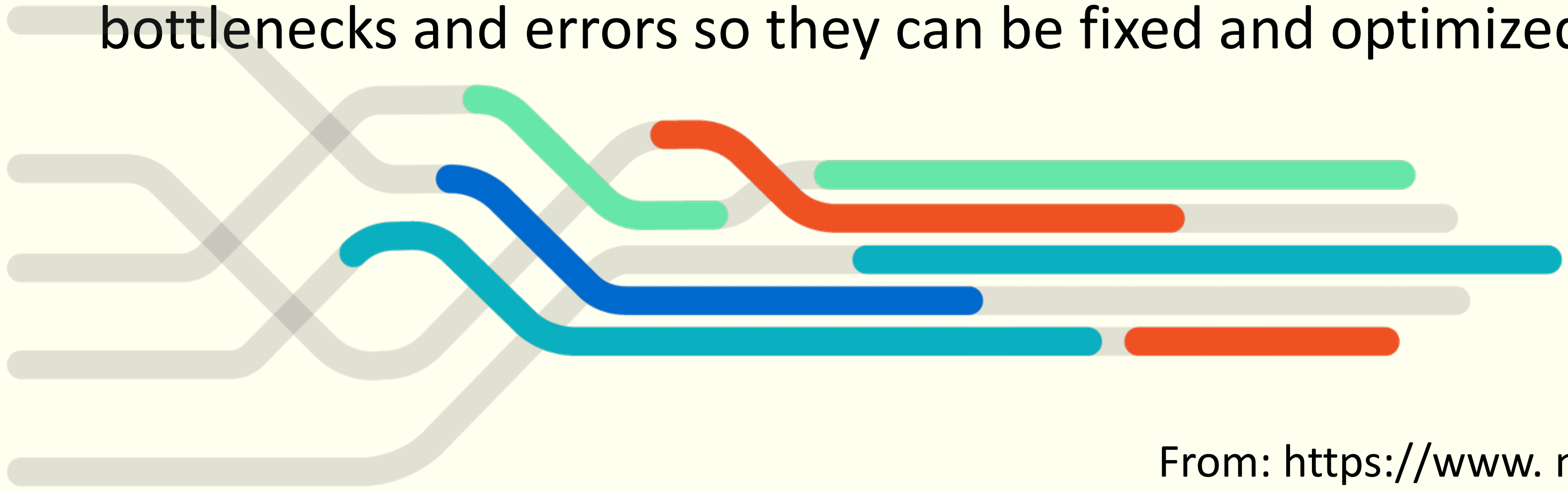
Visualization: Visual Cues for abnormalities

Analytics: Deep analytics to predict faults, failures and service degradation



Journey: A User Experience

- Tracings create an end-to-end view of how transactions are executed in a distributed system. They also capture end-to-end and inter-service latencies of individual calls in a distributed journey
- Journey: The sum total of all activities a user performs during a session. A journey can have multiple sub-journeys. Each journey can be made of several paths which can be parallel in a distributed system.
- A journey captures timings, possibly call and return expressions, status code and anything else that an Observer deems to be necessary.
- Journey can be abstracted into templates and help find bottlenecks and errors so they can be fixed and optimized.

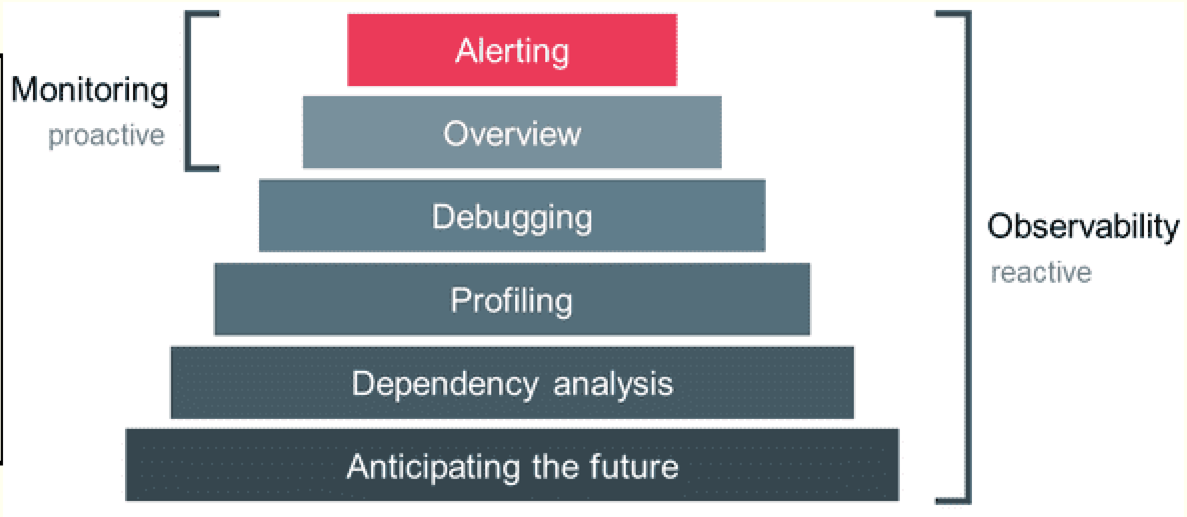


From: https://www. newrelic.com/

Observability in iRODS: Current Status

- Server Logs: collects information about system events and error messages happening in the system. Can be used to find unexpected behavior (distributed)
- Audit Trails: collects user-defined information on triggered action. Can be used to recreate traces that are executed across distributed iRODS servers (centralized).
- Status Metadata: Can store persistent information that can help for further metrics (centralized)

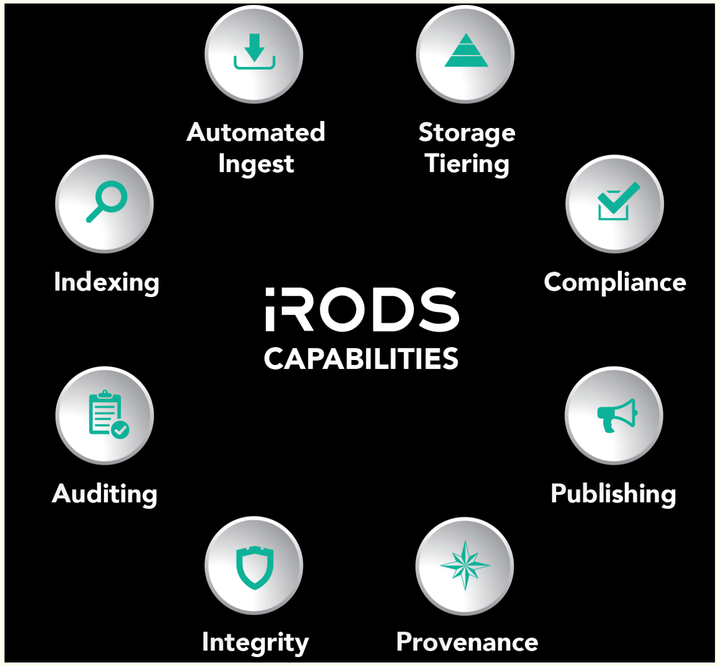
iRODS is currently supportive more towards Monitoring activities than towards Observability.



From: https://ish-ar.io/observability/

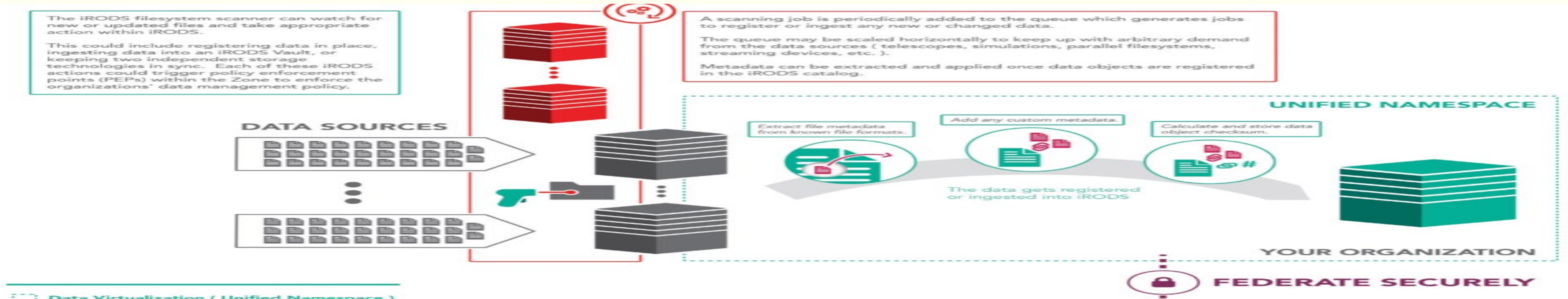
Observability in iRODS

- Towards better performance with proactive metrics & analysis:
  - Help iRODS become better and more pro-active in maintaining performance
  - Help systems that use iRODS to apply iRODS observability metrics to become better and pro-active in maintaining performance
- Server Logs, Audit Trails and Status Metadata in iRODS provide a strong and stable foundation for performing Observability.
- Use of policies, rules and microservices provide one more level for gaining information to perform observability
- Missing: Metrics, Journeys, Visualization and Analytics



iRODS Observability: Metrics

- Application Performance Monitoring (APM): To check whether the system satisfies the SLA contracts, meets performance standards, identify bugs and potential issues, and provide flawless user experiences via close monitoring of IT resources.
- Reduce MTTR (Mean Time To Resolution)
- Continuous Monitoring towards Proactive Remediation
- Alerts and Simple Analysis
- Metrics: What can we monitor in iRODS (not a comprehensive list)
    - CPU/Memory Usage
    - Network Traffic
    - Database Load
    - Error Types/Rates
    - Request rates
    - Response times (mean, max, min)
    - Bandwidth/Throughput
    - Concurrent Connections
    - Number of instances/threads
    - Microservice/function usage/time
    - Uptime, Restarts & Availability
    - User Experience (happy faces)
    - Other Software KPIs


DATA SOURCES
Extract file metadata from known file formats.
Add any custom metadata.
Calculate and store data object checksum.
The data gets registered or ingested into iRODS
UNIFIED NAMESPACE
YOUR ORGANIZATION
FEDERATE SECURELY
Data Virtualization (Unified Namespace)

The iRODS filesystem scanner can watch for new or updated files and take appropriate action within iRODS.

This could include registering data in place, ingesting data into an iRODS Vault, or keeping two independent storage technologies in sync. Each of these iRODS actions could trigger policy enforcement points (PEPs) within the Zone to enforce the organizations' data management policy.

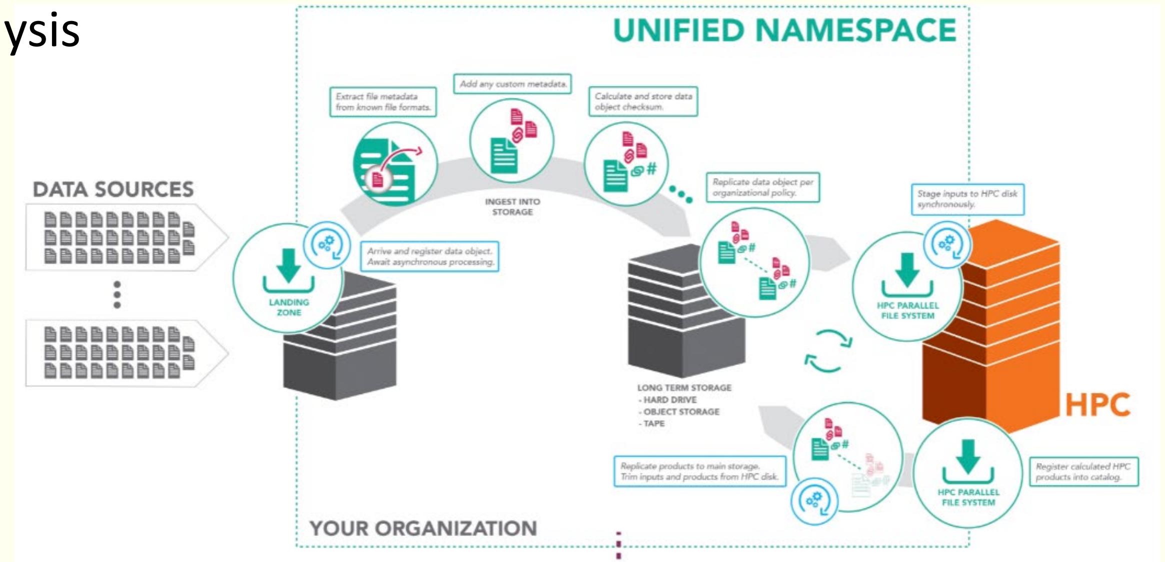
A scanning job is periodically added to the queue which generates jobs to register or ingest any new or changed data.

The queue may be scaled horizontally to keep up with arbitrary demand from the data sources (telescopes, simulations, parallel filesystems, streaming devices, etc.).

Metadata can be extracted and applied once data objects are registered in the iRODS catalog.

iRODS Observability: Journeys

- Distributed Tracing (DT): Chaining of services and peer-to-peer connections across distributed systems makes it hard to trace the activities of a session but is critical for performance monitoring.
- DT helps identify bottlenecks across dynamic and heterogenous infrastructures
- Journeys: Session level performance analysis and monitoring
  - Distributed Transaction Monitoring and Analysis
  - Create User or Application Profiles
  - Define Patterns and Templates of Journeys and Sub-journeys
  - Latency optimization
  - Failure Models – Alternate Pathways
  - Service Dependency Analysis
  - Critical Path Analysis
  - Root Cause Analysis


UNIFIED NAMESPACE
DATA SOURCES
Extract file metadata from known file formats.
Add any custom metadata.
Calculate and store data object checksum.
INGEST INTO STORAGE
Replicate data object per organizational policy.
Stage inputs to HPC disk synchronously.
LANDING ZONE
Arrive and register data object. Await asynchronous processing.
HPC PARALLEL FILE SYSTEM
LONG TERM STORAGE
- HARD DRIVE
- OBJECT STORAGE
- TAPE
HPC
Replicate products to main storage. Trim inputs and products from HPC disk.
HPC PARALLEL FILE SYSTEM
Register calculated HPC products into catalog.
YOUR ORGANIZATION

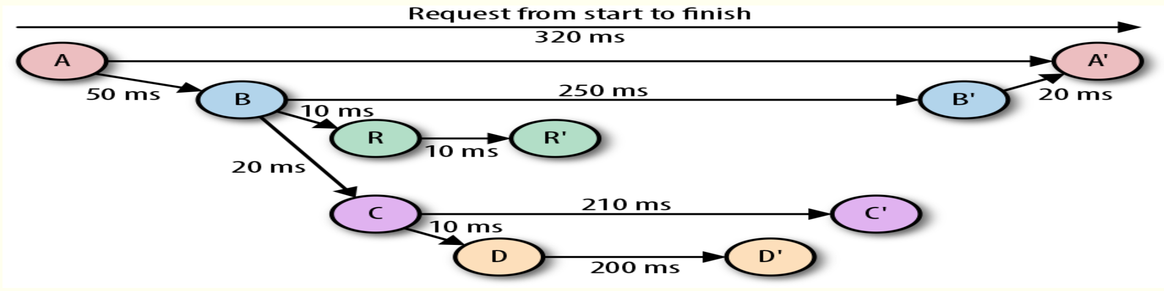
iRODS Observability: Analytics

- Predictive Analytics: What is likely to happen?
- Descriptive & Diagnostic Analytics: What happened and why it happened?
- Prescriptive Analytics: How can we avoid that happening?

Some Examples

- Statistical Analytics: Analyze metrics data for informative nuggets. Max, Min, Median, Mean, StdDev, etc. provide insights. Can be used to define norms, SLAs and expected outcomes and latencies
- Graph Analytics: Use traces and journeys to find patterns. Pattern analysis. Critical nodes and Most used nodes. Candidates for improvements. Pre-staging and pre-processing options.
- Text Analytics: Contextual data of journey to define dynamic slicing and define repeatable experiences.
- Machine Learning: Learn good and bad patterns. Successful journeys and failed journeys.
- …

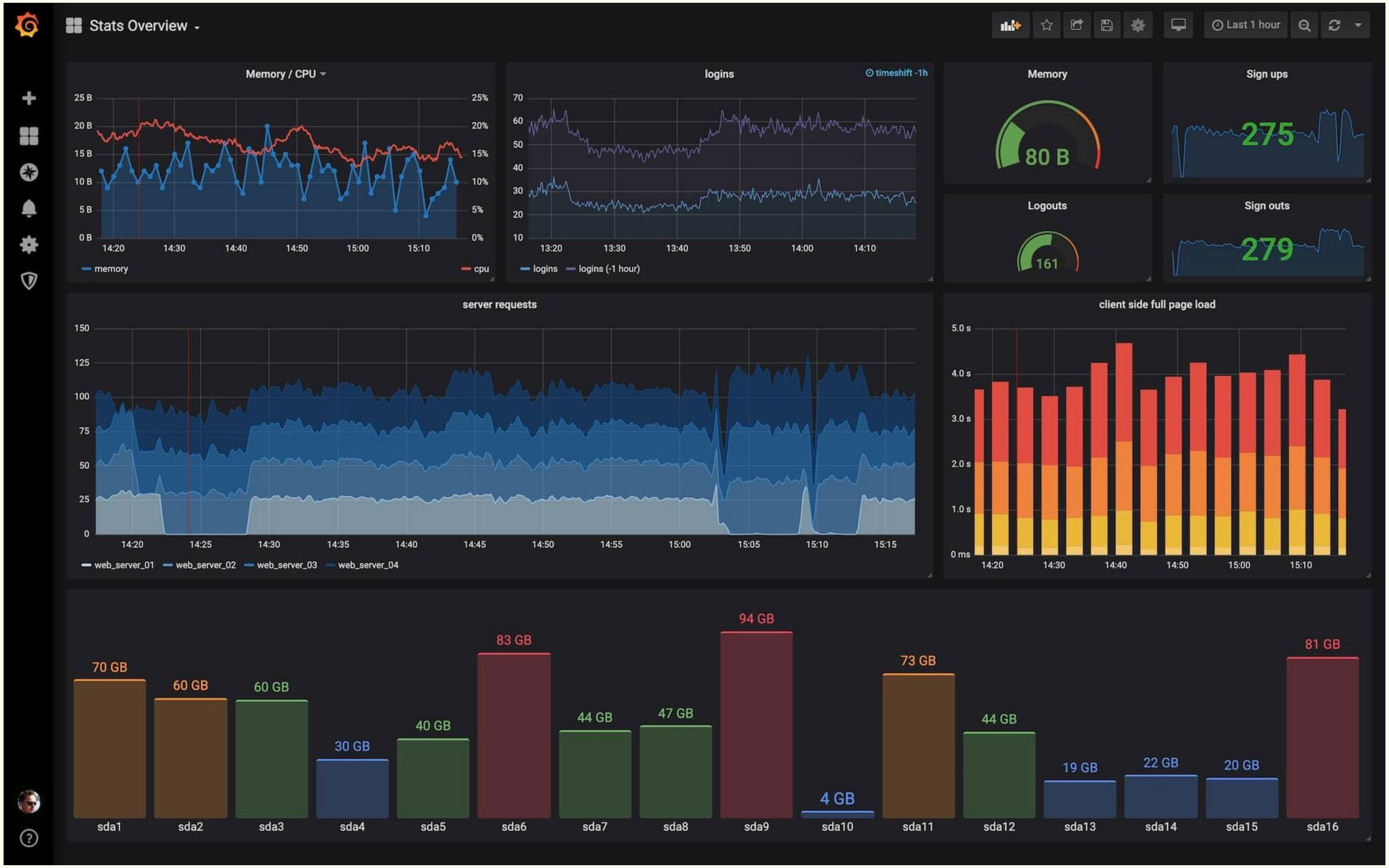
We are just scratching the surface



From: https://www.oreilly.com/

iRODS Observability: Visualization

A System Administrator’s Dream

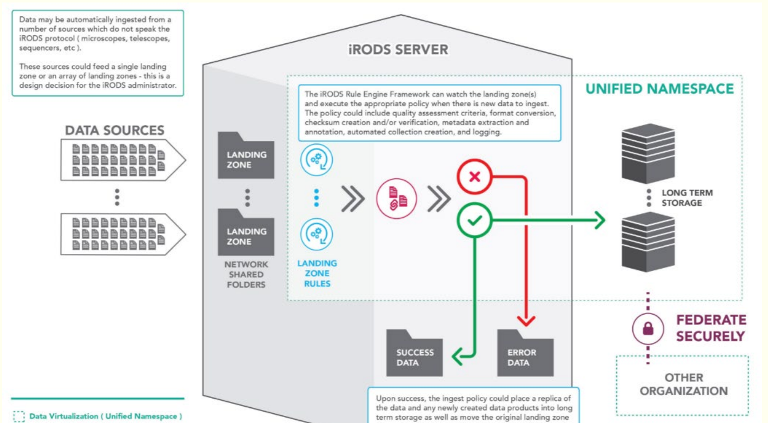


From: https://www.oreilly.com/

iRODS and Observability

- Observability is becoming important because of complexities of the applications as well as need for high availability and throughput by the user community
- Observability can be used as a means to monitor the system continuously and, if possible, correct them on the fly
- Observability can also provide insight to developers on how performance can be improved
- Observability in iRODS
  - Multiple assets already available in iRODS: server logs, audit trails, metadata
  - Other assets we haven't leveraged yet: policies, rules, micro-services
- There is a clear need for Observability in iRODS
  - Metrics can be improved
  - Journeys can help in making user experience better
  - Analytics can help find problems before they occur
  - Visualization can help developers and administrators with visual cues and human analytics
- Good idea to think about when we already do enterprise level applications

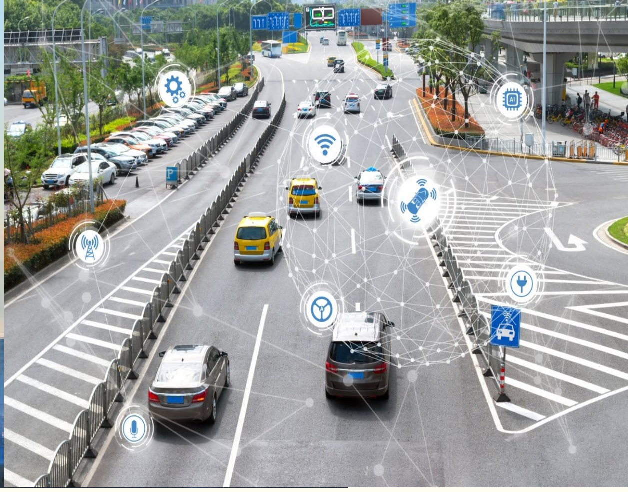
Scalable, reliable resilience needs better Managed Adaptive System Support


Data may be automatically ingested from a number of sources which do not speak the iRODS protocol (microscopes, telescopes, sequencers, etc).

These sources could feed a single landing zone or an array of landing zones - this is a design decision for the iRODS administrator.

The iRODS Rule Engine Framework can watch the landing zone(s) and execute the appropriate policy when there is new data to ingest. The policy could include quality assessment criteria, format conversion, checksum creation and/or verification, metadata extraction and annotation, automated collection creation, and logging.

Upon success, the ingest policy could place a replica of the data and any newly created data products into long term storage as well as move the original landing zone



Observability & iRODS

Q & A

rajasekar@unc.edu