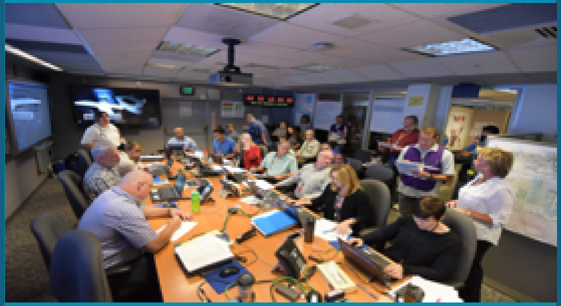
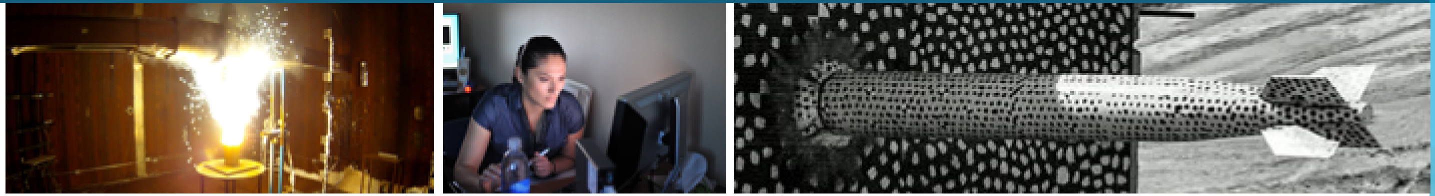
SAND2021-13623C

Sandia National Laboratories

Analysis and Visualization with Ovis Web Services (OWS) Tutorial





LDMSCON - 10/28/2021

Ben Schwaller & Nick Tucker

U.S. DEPARTMENT OF ENERGY NNSA

Presentation Outline

OWS Part 1:

- Goals and Overview of the OWS Infrastructure
- Basic Grafana Usage
  - Basic Navigation
  - Dashboard Time Selection
  - Creating Grafana Dashboards
  - Creating Queries and Calling Analyses
- Visualization Best Practices
  - Drill-down Dashboards
  - Performance Considerations
  - Security Considerations

Ben’s talk about OWS at Sandia

- What things has Sandia found useful to show?

OWS Part 2:

- Analysis Backend
  - Django Application Design Choices
  - Grafana Query Handling
  - Settings for Customized Site Usage
  - Calling Analysis Modules
  - Formatting DataFrames for Grafana
- Analysis Modules
  - Python Class Instantiation
  - DSOS Python API
  - Performance Considerations
  - Parallelization Today
  - Parallelization Tomorrow

OWS Part 1: Goals and Overview

Challenges in HPC Monitoring Analysis

HPC centers can produce TBs of complex data each day

- A single 1500-node HPC cluster at Sandia produces 1.5 TB of time-series data each day
  - Sampled at 1Hz to give appropriate granularity
- Data is high-dimensional (~5000 metrics)

Running even basic analyses across all data for all time can rapidly create high computational and storage needs

OWS infrastructure needs to support a variety of use cases ranging from system administrators who want to see high-level HPC center-centric views to developers who want fine-grained details about performance characteristics of their job

Goals for HPC Monitoring Analysis and Visualization Tools

To meet the challenges in HPC monitoring and provide usability we had several goals:

- Efficiently handle large and complex data format of HPC monitoring data
- Enable drill-down from center-level to function-level
- Allow for users to easily create and share their own analyses and visualizations

We accomplished these goals by:

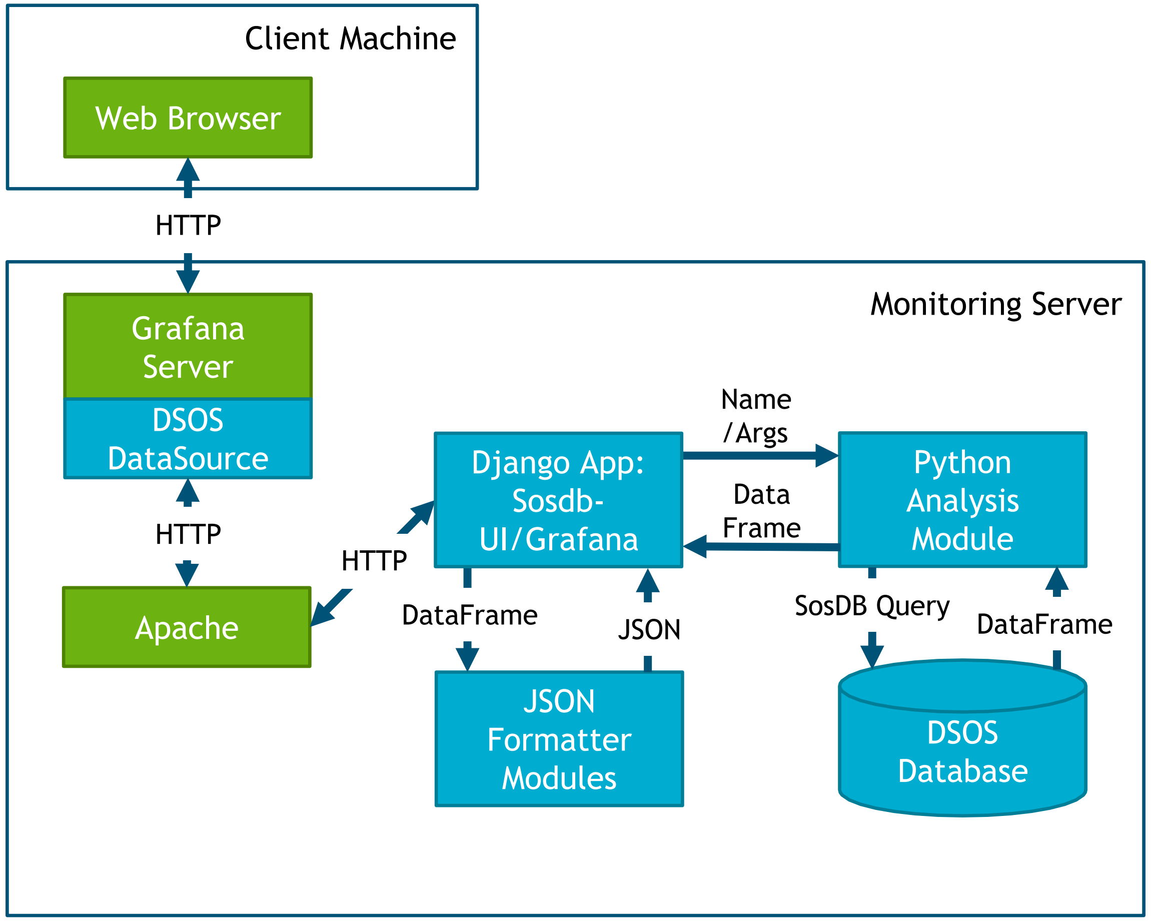
- Creating a system for in-query analyses so only user-requested analyses happen rather than always analyzing HPC data and storing results to a separate database
  - Saves significant compute and storage resources at the cost of increased query times
  - The "always analyze" use case is still available but is not the default
  - Allows for analyses to be easily changed without needing to recompute and store results
  - Users can easily add new analyses to be used in-query
- Using open-source front-end visualization which enables easy visualization creation and sharing
- Using DSOS database to support flexible querying and data management of HPC monitoring data

OWS Infrastructure Overview

Queries pass through several software objects to return derived metrics from a DSOS database to a web server

Note that a python module can be called along the pipeline

Part 2 will cover this in more detail



OWS Part 1: Basic Grafana Usage

Why Grafana?

Grafana is a widely used open source GitHub project designed to enable querying, visualization, and alerting on a wide variety of data sources.

- Open-source allows us to create custom data sources, visualizations, and backend interface as needed
- Also provides a rich assortment of database and visualization plugins

Grafana is being used across the HPC monitoring community

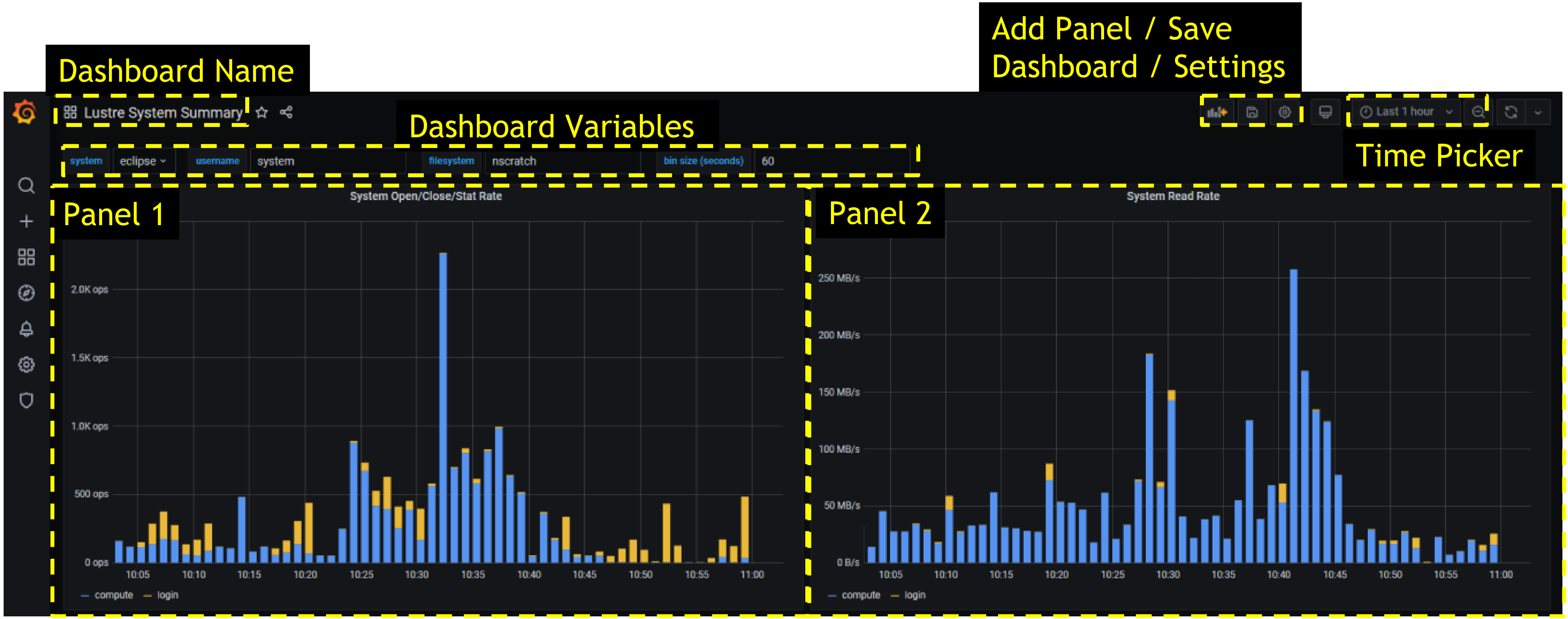
- Creates a common interface for collaboration

Focuses on filtering by time which is useful for HPC monitoring time-series data

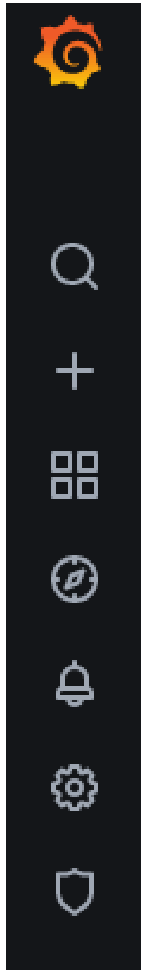
Grafana Dashboard Makeup

Grafana is made of dashboards with panels

- Each dashboards contains some set of panels which can share common parts of a query, such as the time range



Grafana Navigation



Left side bar of Grafana provides several options for navigation. Some have several sub options but their main purposes are:

- Grafana logo: navigate to the Home page
- Magnifying glass: search for dashboards by name
- Plus sign: add a new dashboard or dashboard folder
- Compass: explore database through queries
- Bell: manage alerts
- Cog: control settings for data sources, plugins, and users
- Shield: server administration tools which show security settings, usage statistics, and more

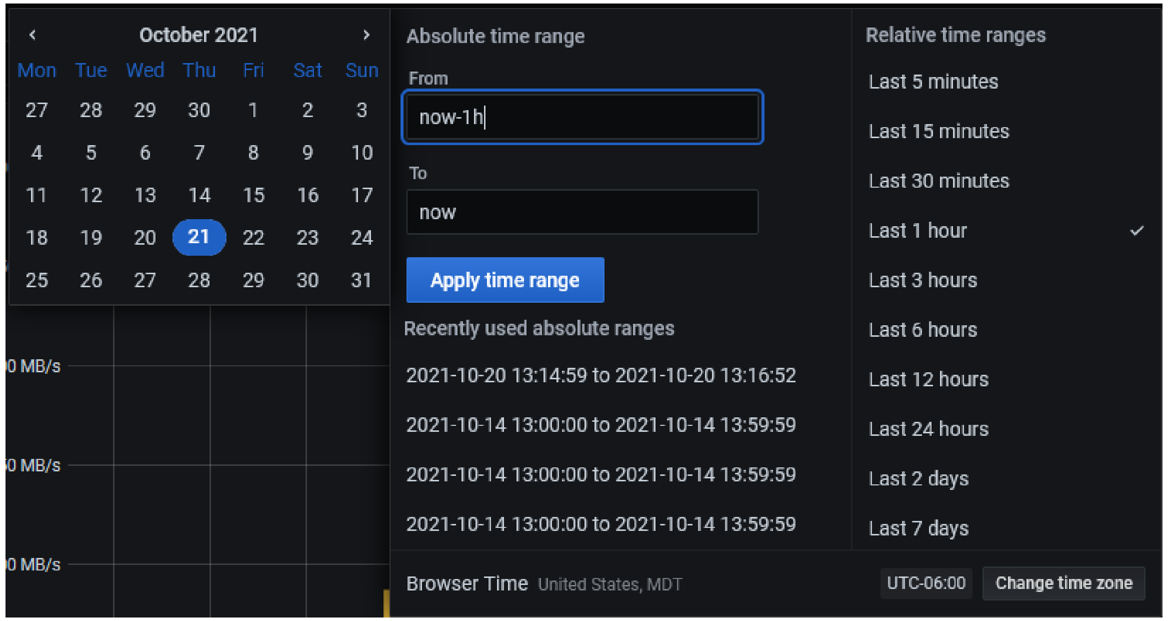
Grafana Time Selection

Grafana’s time picker is at the top of every dashboard besides Home

The time range set is automatically included the queries for all panels in the dashboard

The time range can be picked using a relative time range or an absolute time range

Users can also select a time range by dragging their mouse across a time-series plot



October 2021

Mon	Tue	Wed	Thu	Fri	Sat	Sun
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Absolute time range

From: now-1h

To: now

Apply time range

Recently used absolute ranges

2021-10-20 13:14:59 to 2021-10-20 13:16:52

2021-10-14 13:00:00 to 2021-10-14 13:59:59

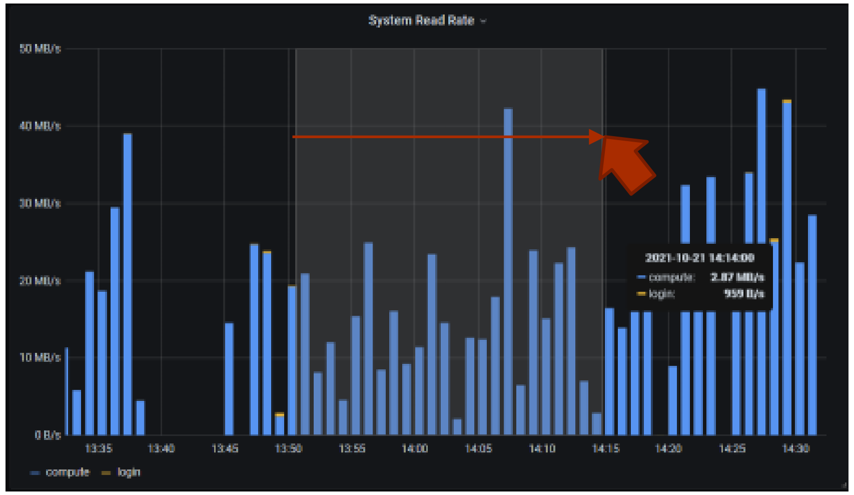
2021-10-14 13:00:00 to 2021-10-14 13:59:59

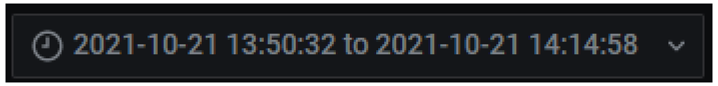
2021-10-14 13:00:00 to 2021-10-14 13:59:59

Relative time ranges

- Last 5 minutes
- Last 15 minutes
- Last 30 minutes
- Last 1 hour ✓
- Last 3 hours
- Last 6 hours
- Last 12 hours
- Last 24 hours
- Last 2 days
- Last 7 days

Browser Time United States, MDT UTC-06:00 Change time zone





2021-10-21 13:50:32 to 2021-10-21 14:14:58

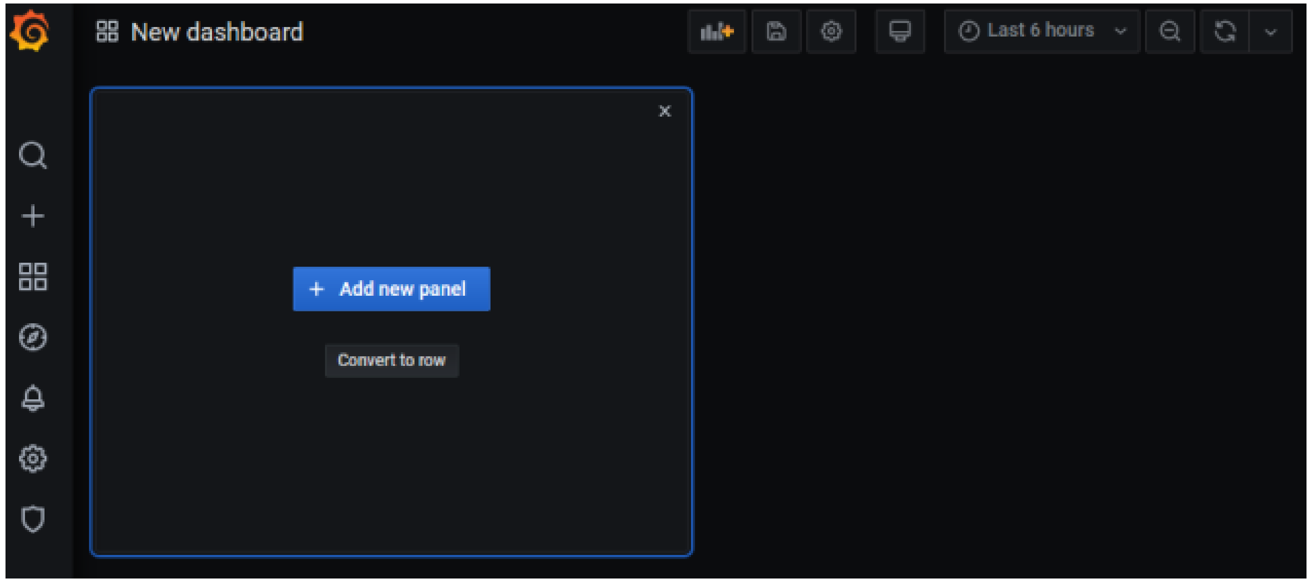
Creating Grafana Dashboards

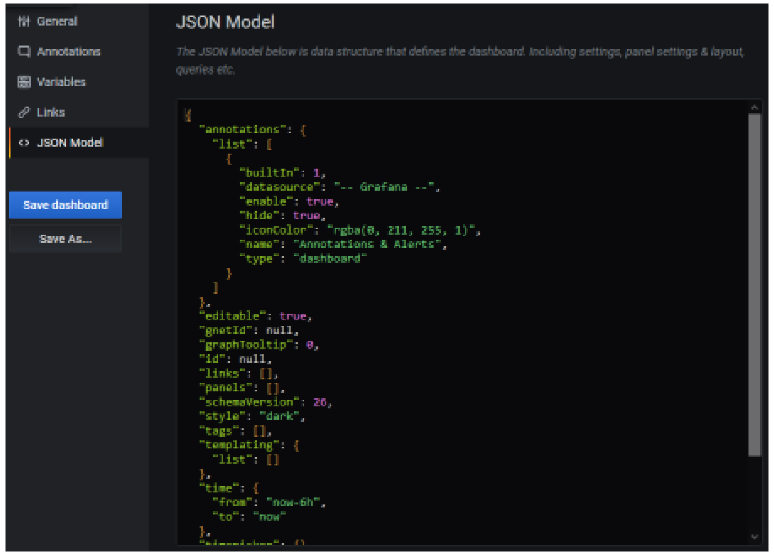
Dashboards are created using the plus tool on the left side navigation bar

Panels can be added using the add panel button at the top right

Dashboards can also be imported from an open-source grafana.com ID, URL, or from an existing dashboard's JSON

- A dashboard's JSON can be found in the dashboard settings
- The dashboard ID must be changed if copying from an existing dashboard

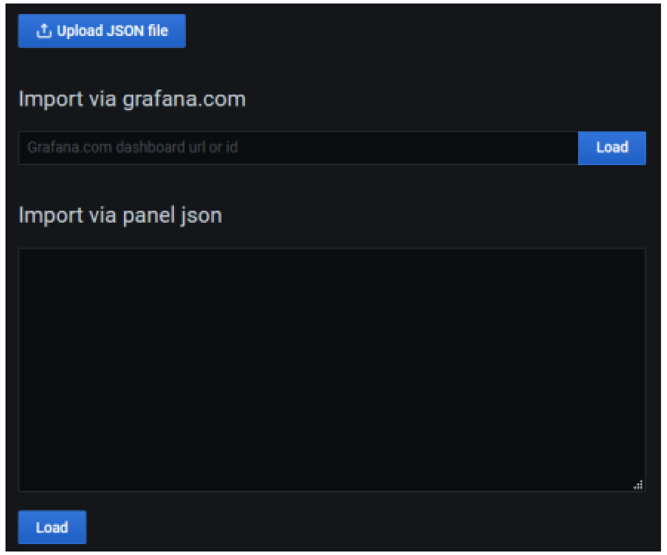




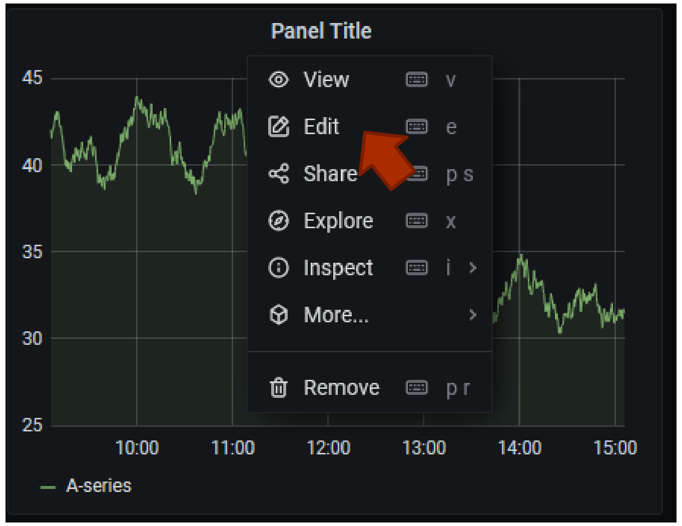
JSON Model

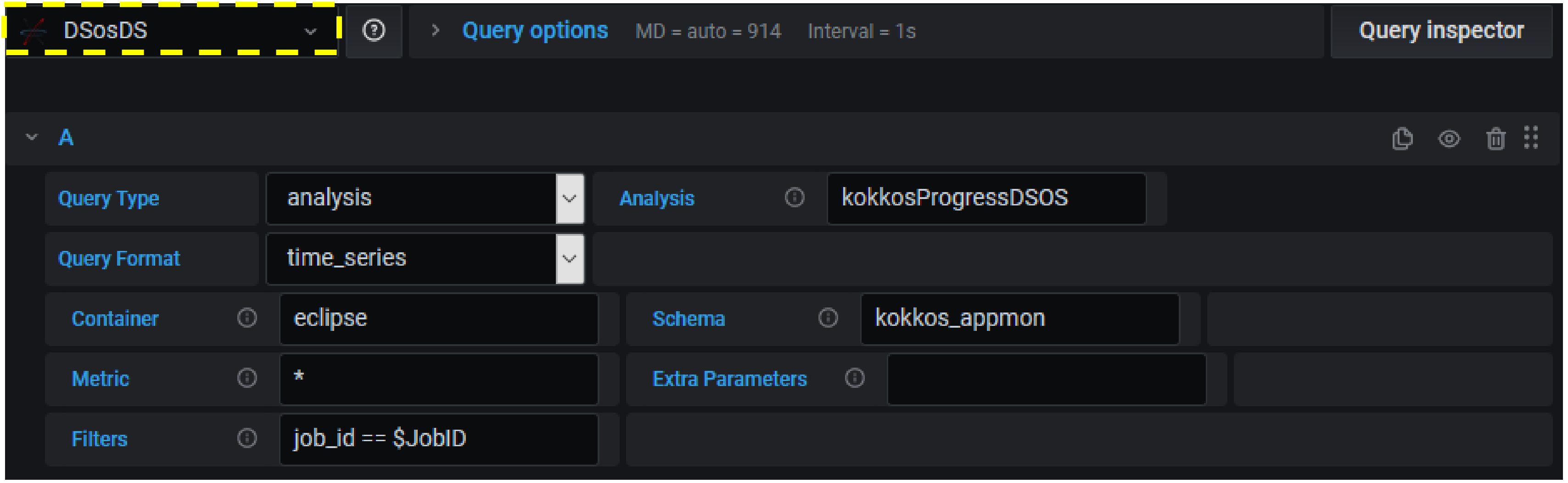
The JSON Model below is data structure that defines the dashboard. Including settings, panel settings & layout, queries etc.

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "id": null,
  "links": [],
  "panels": [],
  "schemaVersion": 26,
  "style": "dark",
  "tags": [],
  "templating": {
    "list": []
  },
  "time": {
    "from": "now-6h",
    "to": "now"
  },
```



Creating Queries within Panels





Query Type	analysis	Analysis	kokkosProgressDSOS
Query Format	time_series		
Container	eclipse	Schema	kokkos_appmon
Metric	*	Extra Parameters	
Filters	job_id == $JobID		

Right clicking on a panel and hitting edit will take you to the panel’s query editor

Here we can create queries for a DSOS database and, if desired, run an analysis module in-line

Select DSosDS (DSOS data source) from the available data sources to start

DSOS Data Source Query Parameters

Query Type: either ‘metrics’ or ‘analysis’
- Metrics will perform a raw query on the DSOS database using the given Grafana parameters
- Analysis will call the chosen analysis module in the next parameter

Analysis: the python analysis module to be called

Query Format: time_series, table, or heatmap
- Specifies the visualization JSON format for Grafana compatibility
- NOTE: time_series expects timestamp to be in the returned data

Container: the path to the DSOS container
- Simple changes can be made on the backend so the full path is not needed

Schema: LDMS schema to query

Metric: comma-separated list of metrics within the chosen schema to query

Extra Parameters: other arguments to be sent to the analysis module

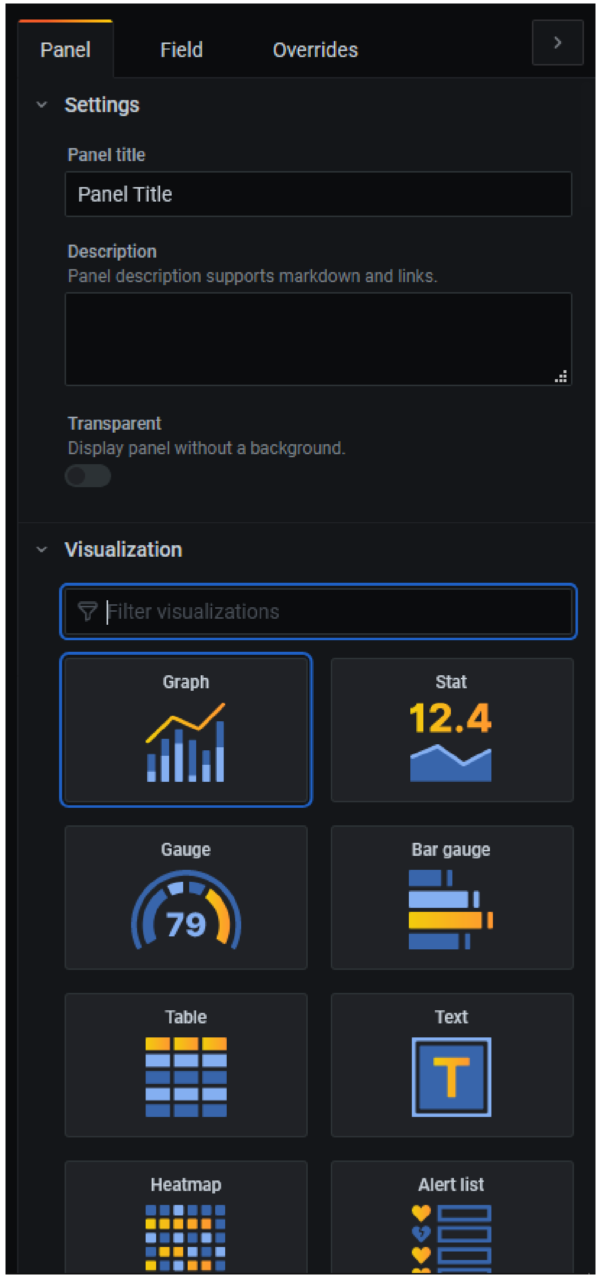
Filters: comma-separated list of additional parameters to be included in the DSOS query
- E.g. job_id > 0,component_id == 3

Changing Panel Visualizations

Panel visualizations can be modified while editing the panel using the options panel on the right side

The panel name, visualization type, visualization options, and overrides can all be adjusted in this view

- Visualization types and options are too numerous to cover entirely here although the Grafana documentation on these is pretty good
  - https://grafana.com/docs/grafana/latest/visualizations/
- Field overrides are useful for adjusting the names of columns in tables or adding data links to certain cells



Dashboard Variables

We can add variables that can be altered by dashboards users and used by Grafana queries

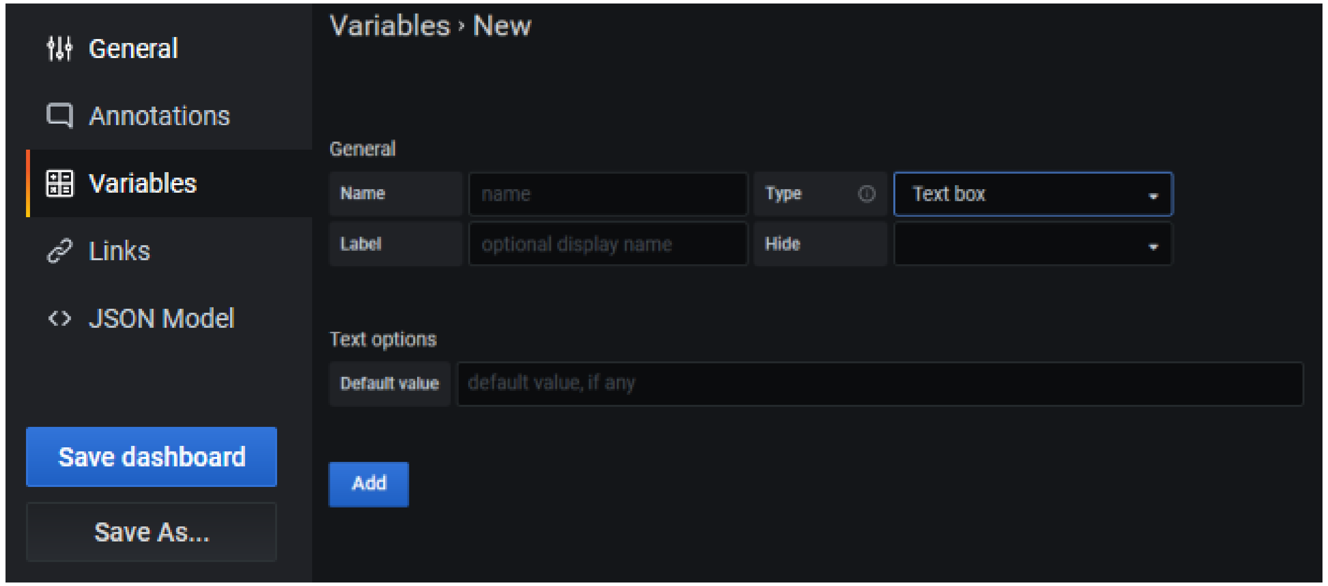
- Go to dashboard settings and select variables
- Add a variable

Wide variety of variable types. The most commonly used ones in Sandia's infrastructure are:

- Text box (allows users to type in a value)
- Custom (allows creator to set a variety of options using a comma-separated list)

Variables can be referenced using $NAME in queries

Additional variables are always available such as $to and $from which specify the time range



Variables › New

General

Name	name	Type	Text box
Label	optional display name	Hide	

Text options

Default value	default value, if any

OWS Part 1: Visualization Best Practices

Invest in the Home Dashboard

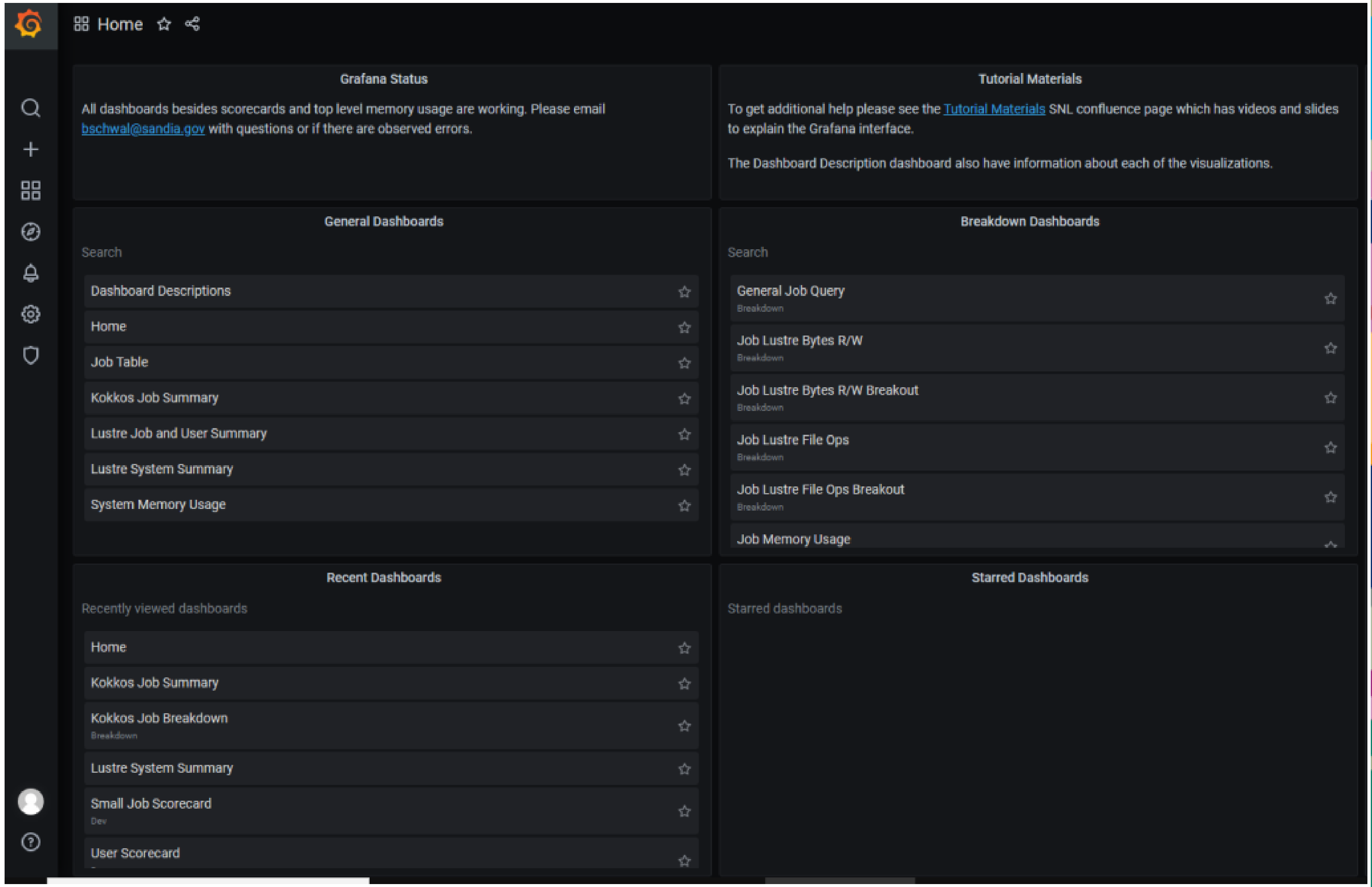
Users first view of the Grafana dashboard will be the Home dashboard

- Users will also commonly navigate back to Home using the Grafana logo

Configuring the Home dashboard with the already available panels of recent and starred allow users to see what they care about most

Providing commonly used sets of dashboards is also useful

MOTD-esque text panels can be an easy way to disseminate information to users



Grafana Status

All dashboards besides scorecards and top level memory usage are working. Please email bschwal@sandia.gov with questions or if there are observed errors.

Tutorial Materials

To get additional help please see the Tutorial Materials SNL confluence page which has videos and slides to explain the Grafana interface.

The Dashboard Description dashboard also have information about each of the visualizations.

General Dashboards

Search

- Dashboard Descriptions
- Home
- Job Table
- Kokkos Job Summary
- Lustre Job and User Summary
- Lustre System Summary
- System Memory Usage

Breakdown Dashboards

Search

- General Job Query (Breakdown)
- Job Lustre Bytes R/W (Breakdown)
- Job Lustre Bytes R/W Breakout (Breakdown)
- Job Lustre File Ops (Breakdown)
- Job Lustre File Ops Breakout (Breakdown)
- Job Memory Usage

Recent Dashboards

Recently viewed dashboards

- Home
- Kokkos Job Summary
- Kokkos Job Breakdown (Breakdown)
- Lustre System Summary
- Small Job Scorecard (Dev)
- User Scorecard

Starred Dashboards

Starred dashboards

High-Level vs Low-Level

To incorporate users and admins alike, Sandia separated the dashboards into high-level (General) and low-level (Breakdown) folders

- Also added panels with links to these dashboards in the Home dashboard

Idea is to allow admins to start at center or cluster level views and drill-down into job-level views whereas users can go immediately to their job-level view

More about specific examples in Ben’s Sandia OWS talk!

Drill-down Dashboards

Configure high-level dashboards to be able to flow to low-level dashboards using links

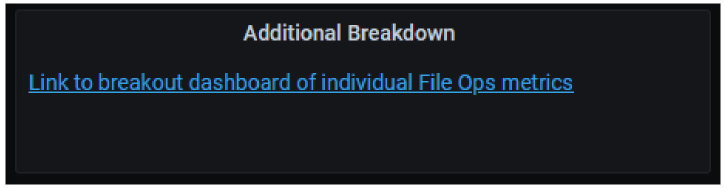
- Create links either in table cell values or in text panels

Create links that incorporate the current dashboard's time range and custom variables

- https://mygrafana.com/d/<DASHBOARD_ID>/<DASHBOARD_NAME>?orgId=1**&to=${__to}&from=${__from}**

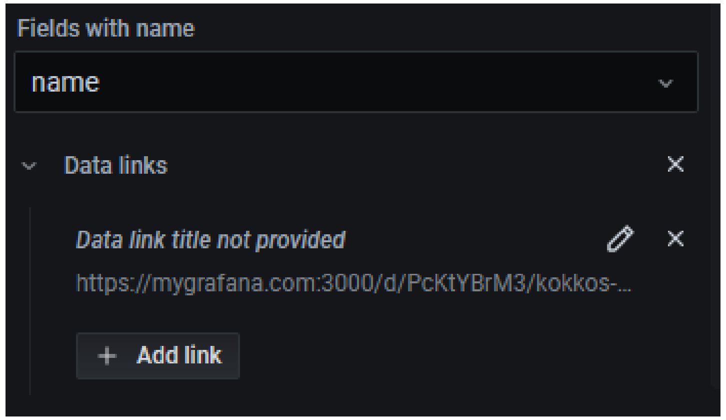
Can pass table cell values and dashboard values through links too

- https://mygrafana.com/d/<DASHBOARD_ID>/<DASHBOARD_NAME>?orgId=1**&var-name=${__value.raw}&var-JobID=$JobID**



Additional Breakdown

Link to breakout dashboard of individual File Ops metrics



Fields with name

name

Data links

Data link title not provided

https://mygrafana.com:3000/d/PcKtYBrM3/kokkos-...

Add link

Performance Considerations

Make the purpose of each dashboard specific to limit the total number of panels on a dashboard

- More panels makes the dashboard busier and creates more query overhead
- E.g. a comprehensive Lustre dashboard that shows all clusters and job performances on the same table could be burdensome

Make high-level analyses simplistic as they will likely be run over a larger time-range or a larger span of cluster nodes

Make default time-ranges on dashboards reasonable for the accompanying analyses

Security Considerations and Future Security Work

Grafana supports a wide-variety of authentication options including LDAP

Can restrict sets of dashboards by Grafana "role"

- 3 role levels of admin, editor, and viewer
- Using LDAP, can set user roles by metagroup
- This is done in the ldap.toml file which controls the Grafana server's LDAP configuration

Current infrastructure does not support restricting users to only be able to view some data

- DSOS user permission capabilities will enable this use case

Questions?

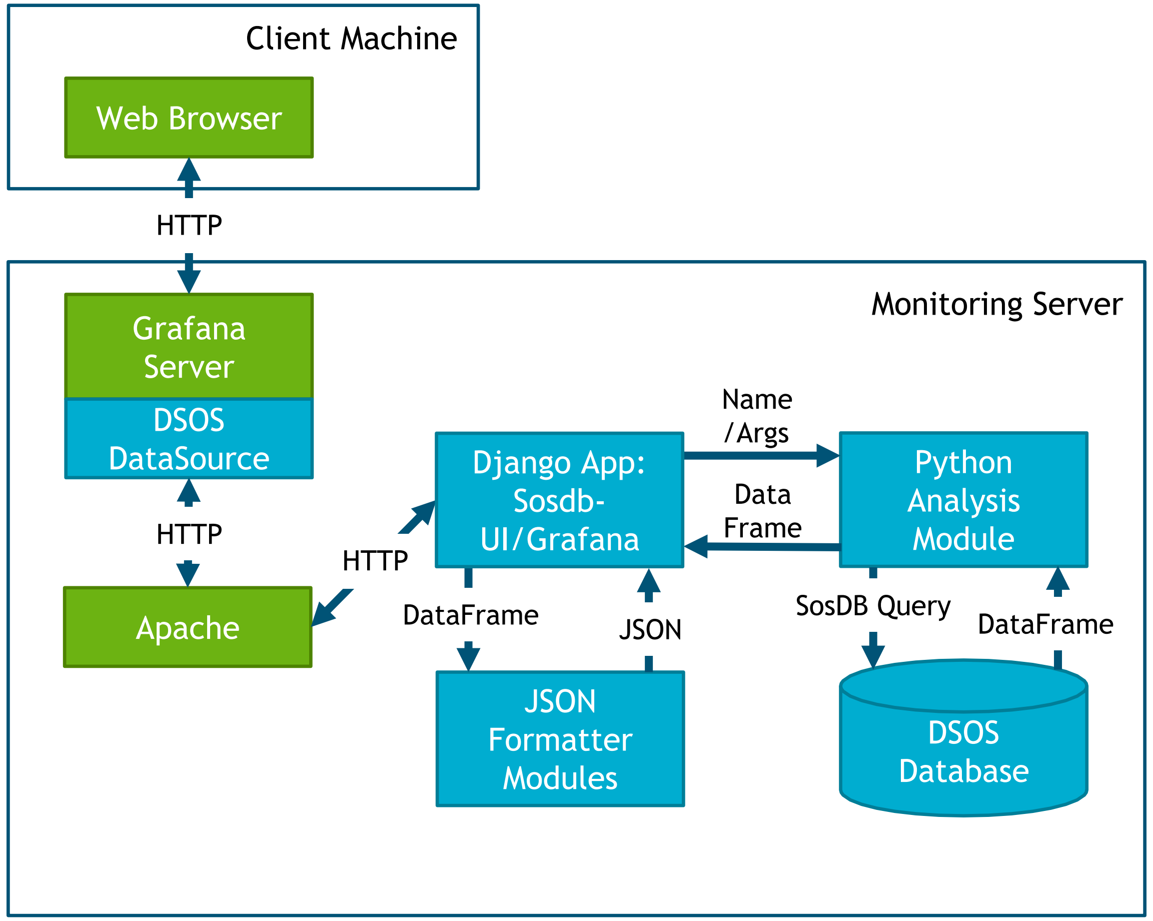
OWS Part 2: Analysis Backend

In-depth Architecture Overview

Ovis Web Services is comprised of several different applications

- sosdb-ui
  - The original OWS web app. Raw metric data and filtering along with LDMS monitoring. Mostly deprecated, and planned to collapse sosdb-grafana into sosdb-ui
- sosdb-grafana
  - Provides Grafana query handling for D/SOS
- numsos
  - Provides the JSON formatters used by sosdb-grafana
  - Provides statistical analysis functionality to perform on DataFrames as well as pluggable analysis modules for Grafana
- Distributed Scalable Object Store (DSOS)
  - Highly indexed database that provides fast query times across large databases
- Apache
  - Hosts the Django framework

OWS does not currently support any other database's aside from DSOS



Django Application Design Choices

The Ovis Web Services suite follows the fundamental goal of Django's stack

Loose coupling

- The various layers of OWS don't "know" about each other unless absolutely necessary

Less code

- OWS follows the DRY principle, and has base classes for requests that use redundant code, e.g. the analysis modules

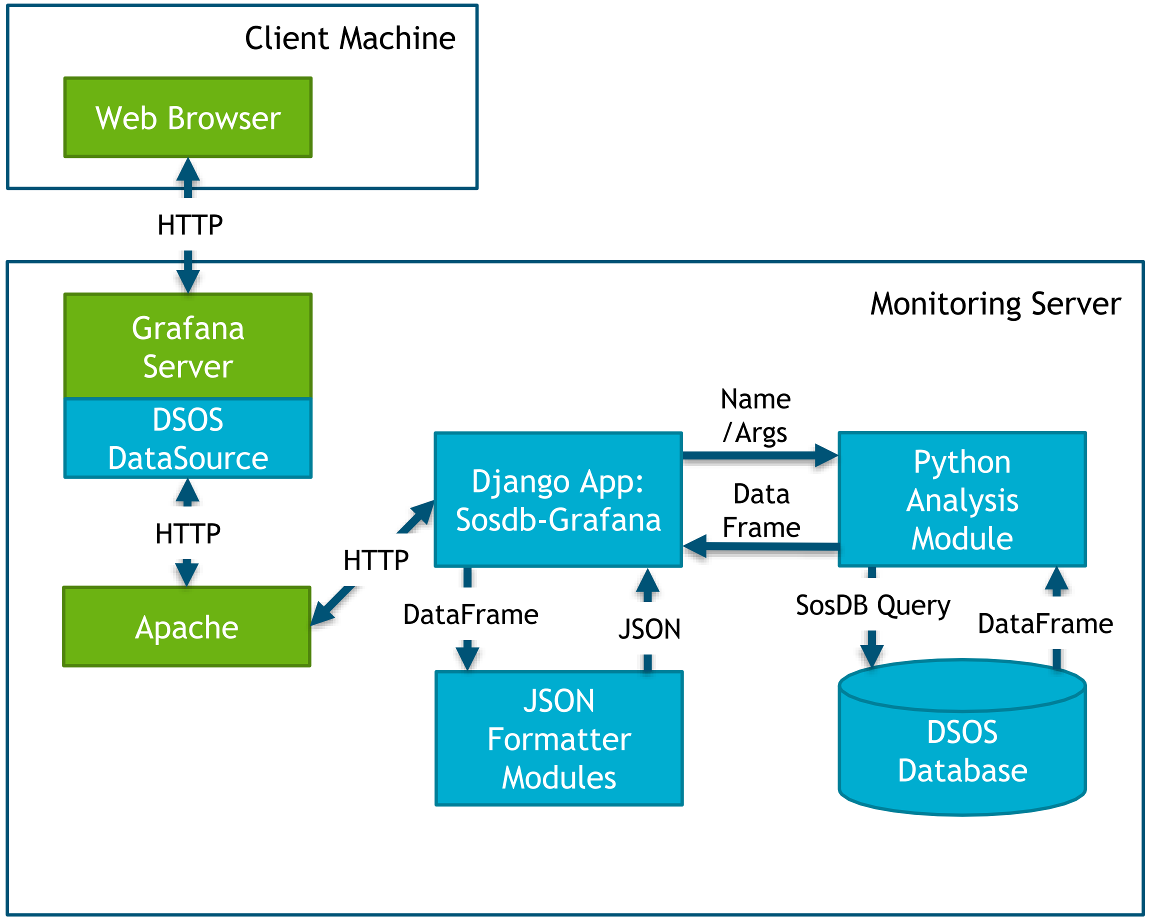
Quick Development

- Large clusters today are inherently unique in their infrastructure, and admins must be able to finely tune custom analysis to their own unique specifications.
- OWS was designed with this in mind, with plug-ability as a primary objective.
- OWS provides raw metric querying, basic analysis modules, as well as a framework of basic templates for custom python3 analysis module development.
- With this approach users can create exactly the analysis needed for their unique situation in an easy to read language.

Grafana Query Handling

Query steps:

- A client makes a request on the Grafana client server
- The request is sent to the apache server running OWS
- The apache server handles the request and makes the appropriate queries to the relevant containers or analysis modules
- If an analysis module is specified, the module is dynamically loaded in the Django application at the time of the query, following one the fundamental Django design goals of "loose coupling"
- The analysis module (or raw metric query) then queries the specified container on the back-end and returned in a pandas DataFrame object
- The data is then formatted into a JSON object by Django, and returned to the client



OWS Configuration

settings.py is the chief OWS configuration file

- Its template is installed into /<install_dir>/sosgui/settings.py.example by default
- Ensure to change file name to settings.py before starting apache

A default configuration file is included on installation with many of the relevant variables preconfigured.

Key settings for your system:

- ALLOWED_HOSTS = [‘localhost’,‘10.10.0.1’]
  - Determines the hosts that Django will allow to connect
- DSOS_CONF = “<path_to_dsos.conf>”
  - This is the path to the dsos.conf file that defines your DSOS cluster for OWS
- DSOS_ROOT = “<path_to_dsos_container>”
  - The location of the physical container. With DSOS, this container can be on any node(s) in the cluster
- LOG_FILE = “<path_to_install_dir/log_file.txt>”
- ODS_LOG_FILE = “<path_to_install_dir/ods_log.txt>”
  - This is a log file for D/SOS

OWS Setup

Create users for the Django application

cd /<install_dir>

python3 manage.py migrate

python3 manage.py migrate –run-syncdb

python3 manage.py createsuperuser

Follow prompt to create admin user

Can create other users as well, but single admin user is all that's needed for OWS to function

Once you have a user database create, your apache configuration in order, and your sosgui/settings.py file correctly configured, you can start apache.

Apache will begin serving requests from Grafana at the configured address <apache>:<port>/grafana/

Calling Analysis Modules

Analysis modules are imported on a by request basis

Analysis modules by design are part of the numsos module.

- They are installed into <numsos_install_path>/lib/python3.6/site-packages/graf_analysis
- To use custom analysis modules, simply drop your custom module into the installation directory listed above.
- You will then be able to reference your analysis module from grafana by it’s module name
  - e.g. if python file is compMinMeanMax.py, Analysis Module would be “compMinMeanMax”
- Analysis modules can also be used outside of Grafana, and be fed to any application needed for your use case that accepts pandas DataFrames.

Formatting DataFrames for Grafana

Numsos currently has three formatters for Grafana

Time_series

- Returns time series data in the format:
- [metric_value, posix_time_stamp]

Heatmap

- Bins heatmap data on the server side so that Grafana doesn’t spend a large amount of time parsing the returned time_series data into bins

Table

- Simply returns the data in a JSON format without adding or changing values

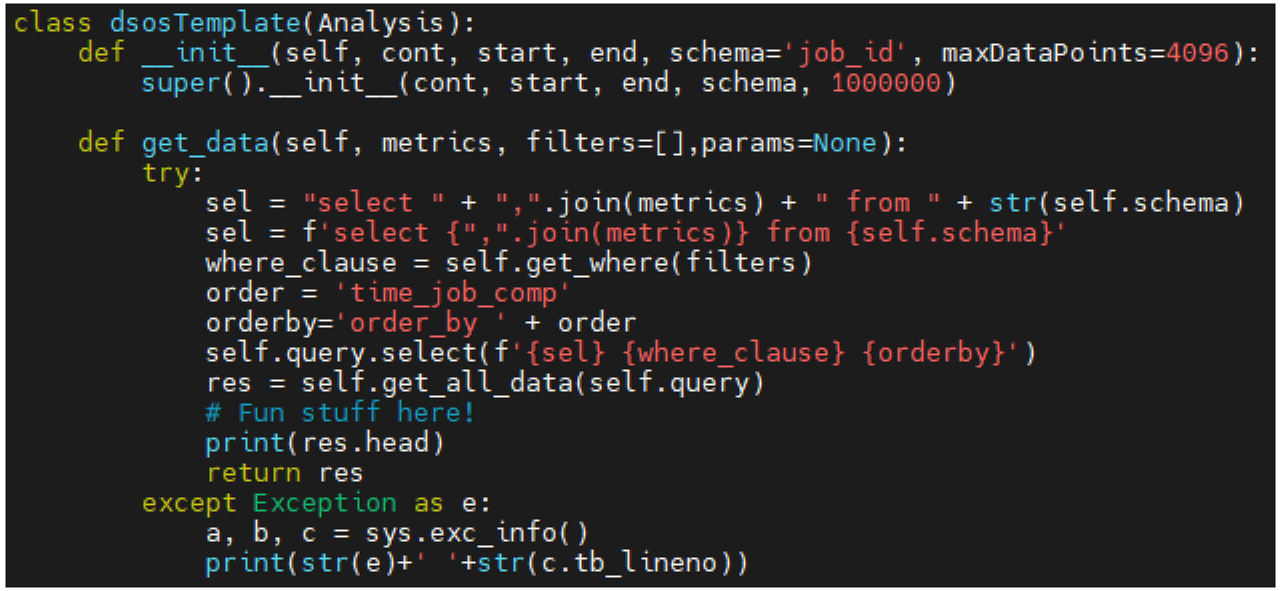
OWS Part 2: Analysis Module

Python Class Instantiation

Init sets up class parameters with a super call, most importantly setting up self.query

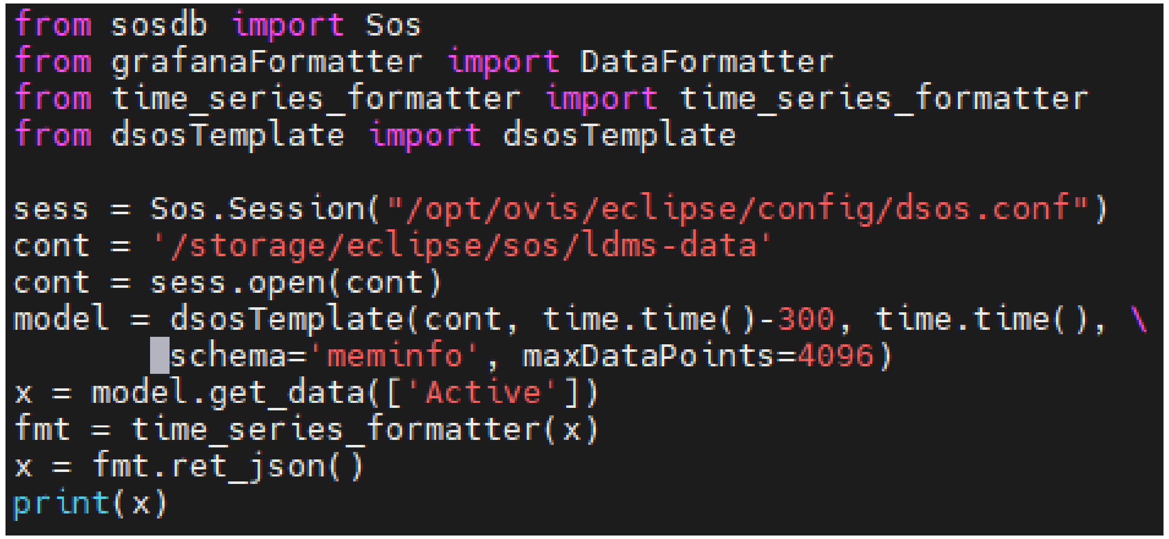
- Self.query is set in the Analysis class template as Sos.SqlQuery(cont, 10000000)

maxDataPoints is a Grafana variable sent with every query to indicate the maximum amount of points the current screen resolution allows for in the panel



```
class dsosTemplate(Analysis):
    def __init__(self, cont, start, end, schema='job_id', maxDataPoints=4096):
        super().__init__(cont, start, end, schema, 1000000)

    def get_data(self, metrics, filters=[],params=None):
        try:
            sel = "select " + ",".join(metrics) + " from " + str(self.schema)
            sel = f'select {",".join(metrics)} from {self.schema}'
            where_clause = self.get_where(filters)
            order = 'time_job_comp'
            orderby='order_by ' + order
            self.query.select(f'{sel} {where_clause} {orderby}')
            res = self.get_all_data(self.query)
            # Fun stuff here!
            print(res.head)
            return res
        except Exception as e:
            a, b, c = sys.exc_info()
            print(str(e)+' '+str(c.tb_lineno))
```



```
from sosdb import Sos
from grafanaFormatter import DataFormatter
from time_series_formatter import time_series_formatter
from dsosTemplate import dsosTemplate

sess = Sos.Session("/opt/ovis/eclipse/config/dsos.conf")
cont = '/storage/eclipse/sos/ldms-data'
cont = sess.open(cont)
model = dsosTemplate(cont, time.time()-300, time.time(), \
        schema='meminfo', maxDataPoints=4096)
x = model.get_data(['Active'])
fmt = time_series_formatter(x)
x = fmt.ret_json()
print(x)
```

DSOS Python API

Sos.SqlQuery is an object for instantiating DSOS queries

- Object takes two variables for instantiation:
  - Container path
  - Query size
  - E.g. query = Sos.SqlQuery('/home/me/database', 1000)

Sos.SqlQuery.select() sets the filter parameters for the query

- Uses Sql-like syntax described in DSOS tutorial
- E.g. query.select('select Active from meminfo where job_id == 101')

Created an analysis class function to parse the filters and add time range filters called get_where

- E.g where_clause = self.get_where(filters)

Sos.SqlSQuery.next() will get up to <query size> records that match the query and returns a pandas DataFrame

- E.g. df = query.next()

Created an analysis class function to get all data matching the query called get_all_data

- Provide the query object as a parameter
- E.g. df = self.get_all_data(query)

Let’s make some analyses!

1. Show a time-series of a single metric

2. Show a time-series of a single metric averaged over all nodes

2. Show a time-series of the rate of a single metric cumulated over all nodes

3. Show a table of the top 5 jobs in terms of the average value of a metric

Performance Considerations

Based on experience, the query and data formatting takes more time than the analysis

- Moving to clustering, machine learning models, higher level statistics would like change this balance
- Data rollover and partitioning will likely help with query performance

Be judicious in what data to query and filter as much as possible where applicable

- Choosing the best indices for the query is **critical**
  - I.e. searching for a specific job's data is fastest using an index based primarily on job_id
  - Non-job or component specific queries should use an index based primarily on timestamp

Parallelization Today / Tomorrow

The OWS infrastructure is currently parallelized through DSOS queries

- We suggest distributing the LDMS data across multiple sources
- Python API calls to DSOS database distributed the query across the containers DSOS encapsulates and performs independent parallel queries

Future parallelization techniques could include:

- Creating a load balancing Apache server to distribute multiple queries from users / single dashboard across analysis cluster
- Parallelizing JSON formatter modules
- Creating parallel python analysis code

Questions?