



Visualizing Data in PostgreSQL With Grafana

Preetam Jinka

PostgresConf NYC March 2019

About Me

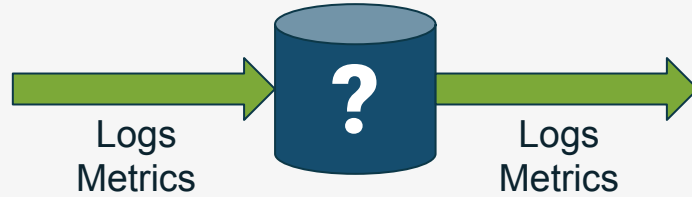
- Runtime + Code Science Infrastructure at ShiftLeft
- Twitter: @PreetamJinka

What is this talk about?

This is about making visualizations from data that **already exists in your database for another purpose.**

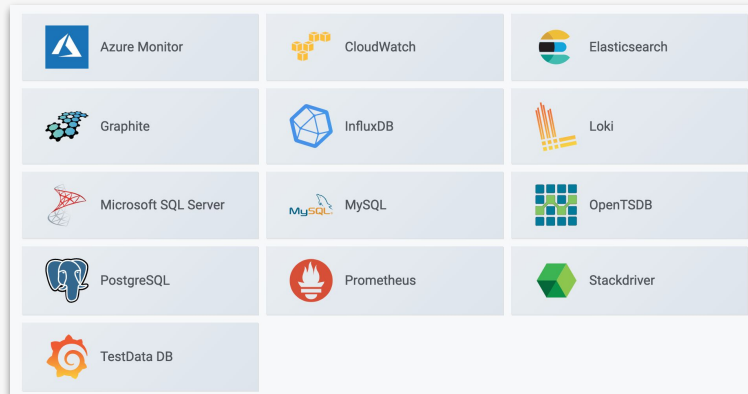
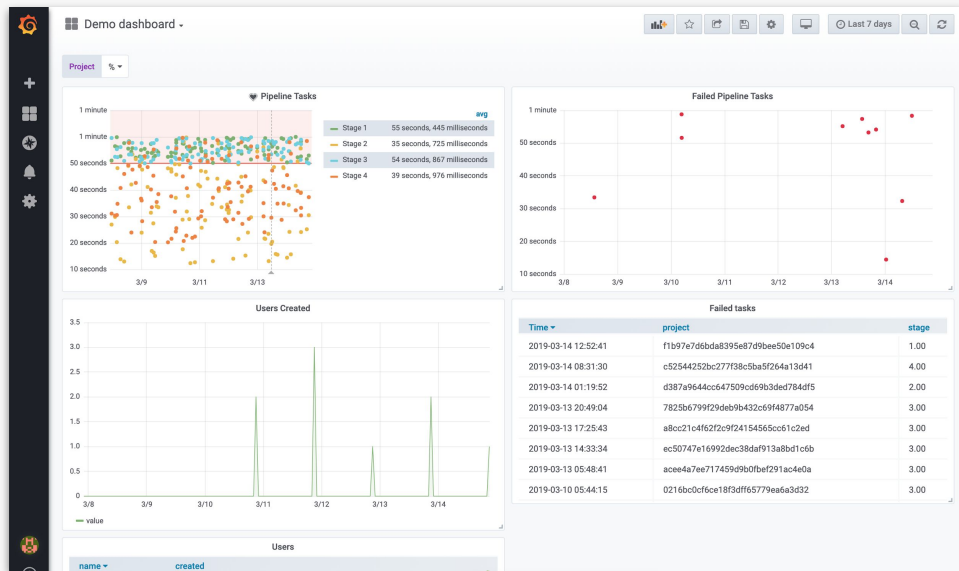
Why you should use Grafana with PostgreSQL

- Get a different perspective for monitoring



- Understand what your data looks like
- Build visualizations and reports without writing code

Grafana is a visualization platform.



Structure of this talk

1. Getting started
2. Simple visualizations
3. Alerts
4. Making dashboards interactive
5. Examples of Grafana+PG "wins" at ShiftLeft

Getting Started: Docker

Docker Compose:

<https://github.com/Preetam/compose-postgresql-grafana>

1. Run `docker-compose up --build`
2. Go to <http://localhost:3000/> and login using admin/admin.

Getting Started: Manual

- Use a read-only user
- Take advantage of per-table permissions for sensitive data.

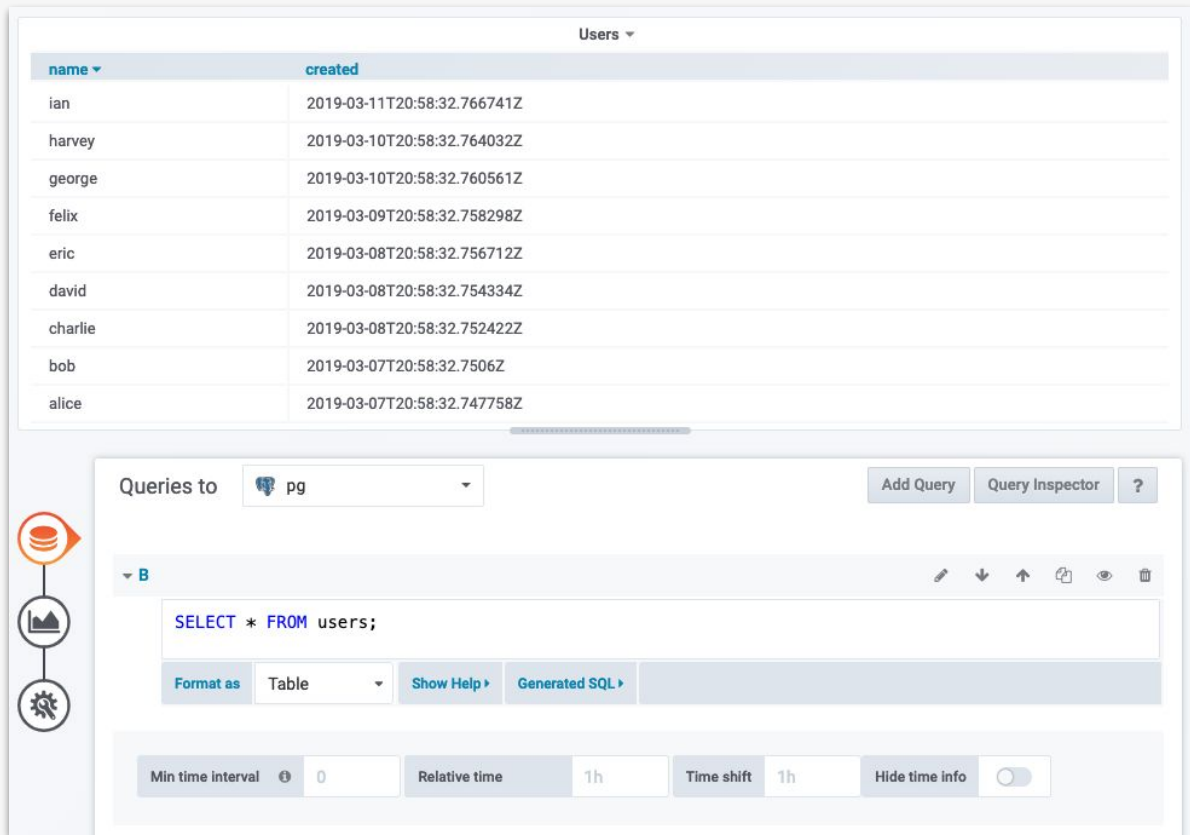
The screenshot shows a configuration window for a PostgreSQL connection. At the top, the connection is named 'pg' and is set as the 'Default' connection, indicated by a toggle switch. The 'PostgreSQL Connection' section includes fields for 'Host' (postgres:5432), 'Database' (postgres), 'User' (postgres), and 'Password' (password). The 'SSL Mode' is set to 'disable'. Below this, the 'Connection limits' section shows 'Max open' as 'unlimited', 'Max idle' as '2', and 'Max lifetime' as '14400'. The 'PostgreSQL details' section shows 'Version' as '11', 'TimescaleDB' as disabled with a 'Help >' link, and 'Min time interval' as '1m'.

Field	Value
Name	pg
Default	Default <input checked="" type="checkbox"/>
PostgreSQL Connection	
Host	postgres:5432
Database	postgres
User	postgres
Password	password
SSL Mode	disable
Connection limits	
Max open	unlimited
Max idle	2
Max lifetime	14400
PostgreSQL details	
Version	11
TimescaleDB	<input type="checkbox"/> Help >
Min time interval	1m

Suppose we had a "users" table for our application.

```
postgres=# select * from users;
 name |                created
-----+-----
alice | 2019-03-17 02:51:49.709148+00
bob   | 2019-03-17 02:51:49.727934+00
charlie | 2019-03-18 02:51:49.730087+00
david | 2019-03-18 02:51:49.731717+00
eric  | 2019-03-18 02:51:49.733585+00
felix | 2019-03-19 02:51:49.734678+00
george | 2019-03-20 02:51:49.735819+00
harvey | 2019-03-20 02:51:49.737792+00
ian   | 2019-03-21 02:51:49.738908+00
(9 rows)
```

The Simplest Visualization: SELECT * FROM table



The screenshot displays a data visualization interface. At the top, a table titled "Users" shows a list of users with their names and creation timestamps. Below the table, a query editor is visible, showing the SQL query "SELECT * FROM users;". The interface includes a sidebar with navigation icons, a "Queries to" dropdown set to "pg", and various controls for query execution and visualization.

name	created
ian	2019-03-11T20:58:32.766741Z
harvey	2019-03-10T20:58:32.764032Z
george	2019-03-10T20:58:32.760561Z
felix	2019-03-09T20:58:32.758298Z
eric	2019-03-08T20:58:32.756712Z
david	2019-03-08T20:58:32.754334Z
charlie	2019-03-08T20:58:32.752422Z
bob	2019-03-07T20:58:32.7506Z
alice	2019-03-07T20:58:32.747758Z

Queries to: pg

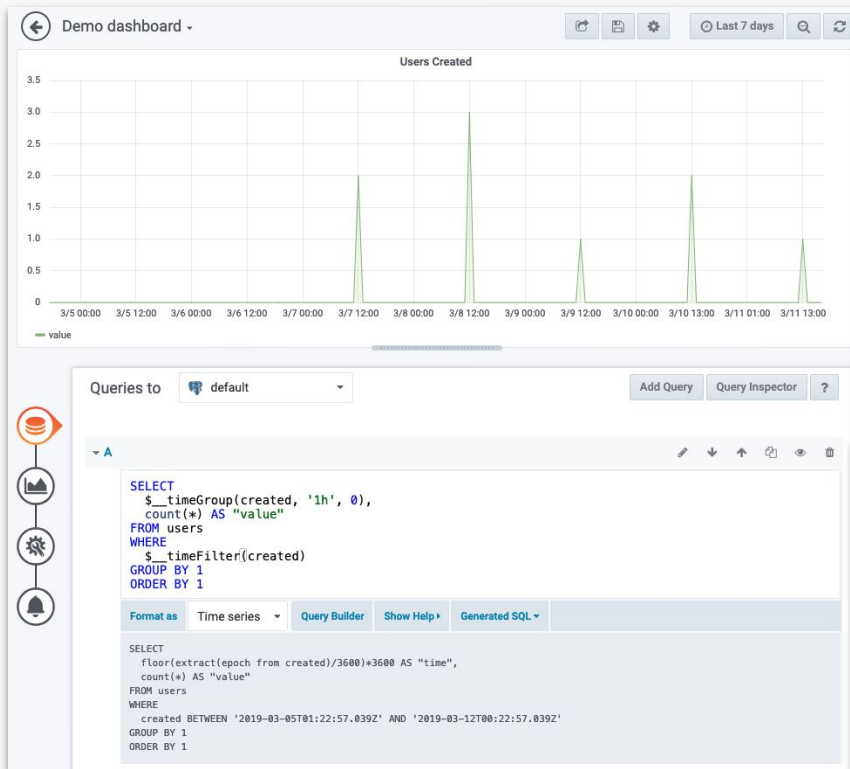
Add Query Query Inspector ?

SELECT * FROM users;

Format as: Table Show Help Generated SQL

Min time interval: 0 Relative time: 1h Time shift: 1h Hide time info:

Time Series and Charts



```
SELECT
  $__timeGroup(created, '1h', 0),
  count(*) AS "value"
FROM users
WHERE
  $__timeFilter(created)
GROUP BY 1
ORDER BY 1
```

```
SELECT
  floor(extract(epoch from created)/3600)*3600 AS "time",
  count(*) AS "value"
FROM users
WHERE
  created BETWEEN '2019-03-05T01:22:57.039Z' AND '2019-03-12T00:22:57.039Z'
GROUP BY 1
ORDER BY 1
```

Create valuable alerts with the simplest charts



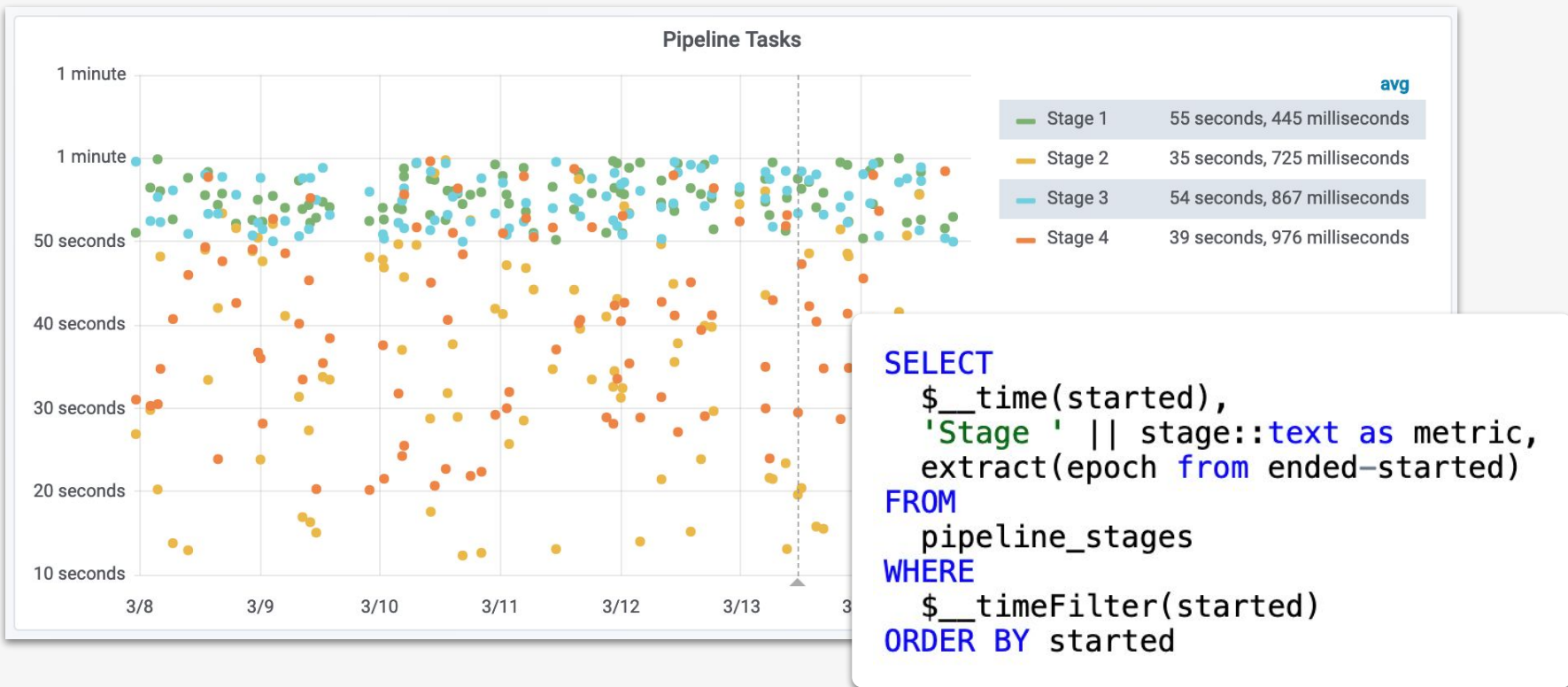
More complicated example: Pipeline Metadata



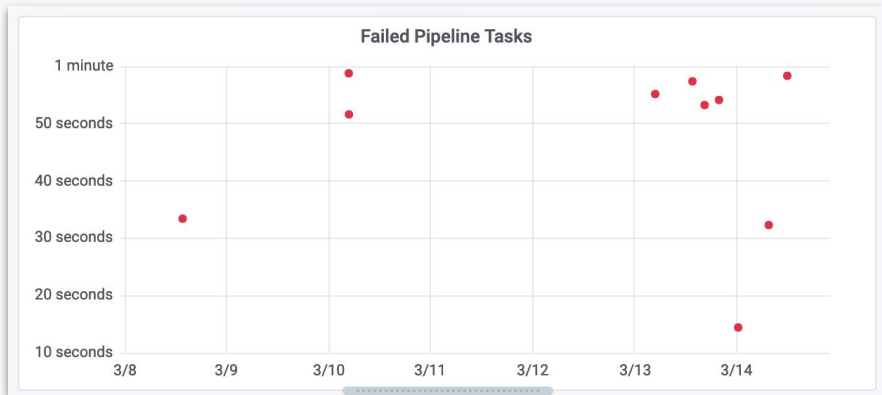
Column	Type
project	text
stage	integer
started	timestamp with time zone
ended	timestamp with time zone
failed	boolean

project	stage	started	ended	failed
b89c1251cbe068eb1f65da1e10b2d329	1	2019-03-12 16:04:08.737775+00	2019-03-12 16:05:06.068346+00	f
b89c1251cbe068eb1f65da1e10b2d329	2	2019-03-12 16:05:08.737775+00	2019-03-12 16:05:30.245483+00	f
b89c1251cbe068eb1f65da1e10b2d329	3	2019-03-12 16:06:08.737775+00	2019-03-12 16:06:59.070184+00	f
b89c1251cbe068eb1f65da1e10b2d329	4	2019-03-12 16:07:08.737775+00	2019-03-12 16:07:51.519871+00	f
c687373127df94197bef9c7d9bb6b94b	1	2019-03-13 20:00:24.504441+00	2019-03-13 20:01:20.839139+00	f
c687373127df94197bef9c7d9bb6b94b	2	2019-03-13 20:01:24.504441+00	2019-03-13 20:01:44.961685+00	f
c687373127df94197bef9c7d9bb6b94b	3	2019-03-13 20:02:24.504441+00	2019-03-13 20:03:22.930905+00	f
c687373127df94197bef9c7d9bb6b94b	4	2019-03-13 20:03:24.504441+00	2019-03-13 20:04:11.817668+00	f
ec50747e16992dec38daf913a8bd1c6b	1	2019-03-13 21:31:34.474025+00	2019-03-13 21:32:31.656488+00	f
ec50747e16992dec38daf913a8bd1c6b	2	2019-03-13 21:32:34.474025+00	2019-03-13 21:33:23.075038+00	f

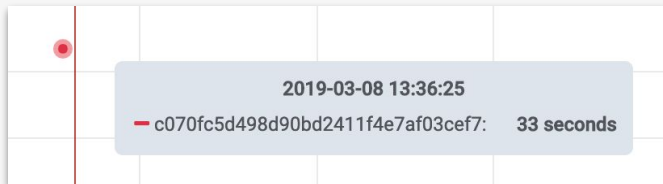
More complicated example: Pipeline Metadata



Create multiple charts using the same data

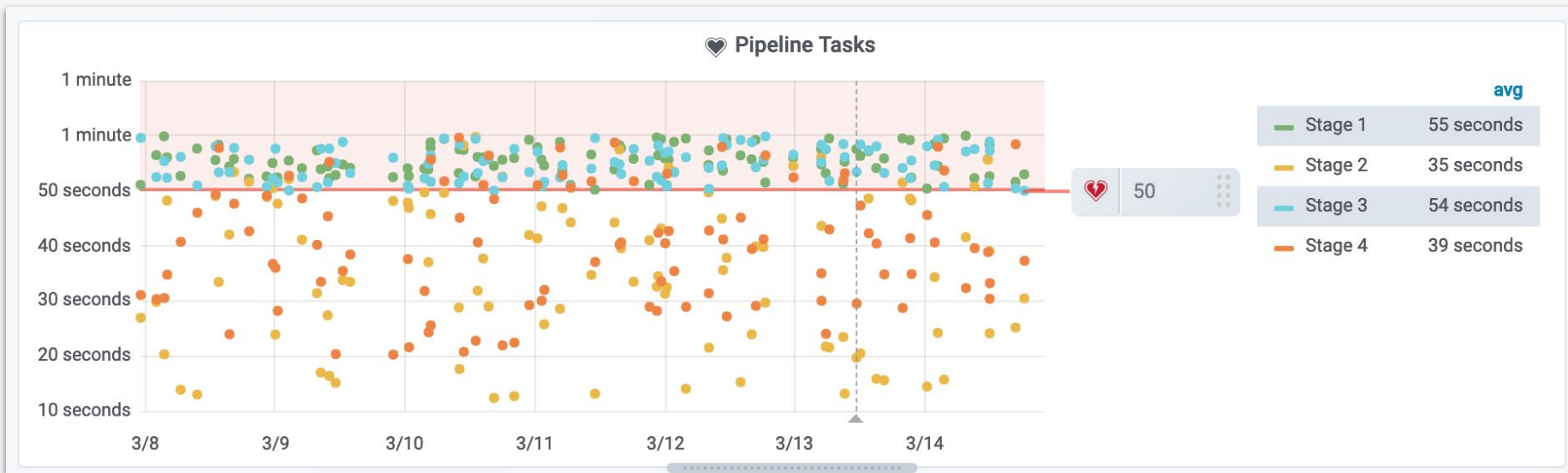


```
SELECT
  $__time(started),
  project as metric,
  extract(epoch from ended-started)
FROM
  pipeline_stages
WHERE
  $__timeFilter(started)
  AND failed = 't'
ORDER BY started
```



Time	project	stage
2019-03-14 12:52:41	f1b97e7d6bda8395e87d9bee50e109c4	1.00
2019-03-14 08:31:30	c52544252bc277f38c5ba5f264a13d41	4.00
2019-03-14 01:19:52	d387a9644cc647509cd69b3ded784df5	2.00
2019-03-13 20:49:04	7825b6799f29deb9b432c69f4877a054	3.00
2019-03-13 17:25:43	a8cc21c4f62f2c9f24154565cc61c2ed	3.00
2019-03-13 14:33:34	ec50747e16992dec38daf913a8bd1c6b	3.00
2019-03-13 05:48:41	acee4a7ee717459d9b0fbef291ac4e0a	3.00
2019-03-10 05:44:15	0216bc0cf6ce18f3dff65779ea6a3d32	3.00

And yes, create alerts for those too.



A note about alerts...

- Watch out for expensive queries
- Add comments to your queries so you can differentiate them in query profilers.
- Maybe set:
`ALTER ROLE reader SET statement_timeout=30000`

Making dashboards interactive with variables

Variables > Edit

General

Name	project	Type	Constant
Label	Project	Hide	

Constant options

Value	%
-------	---

← Demo dashboard ▾

Project % ▾

Making dashboards interactive with variables

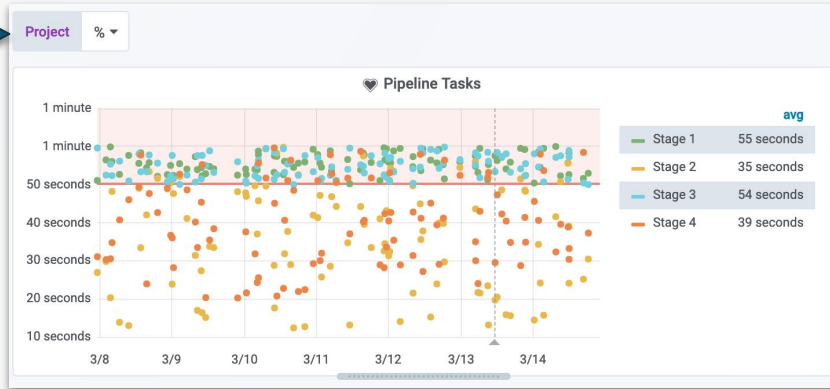
```
pipeline_stages
WHERE
  $__timeFilter(started)
  AND project LIKE '$Project'
ORDER BY started
```

Format as Time series Query Build

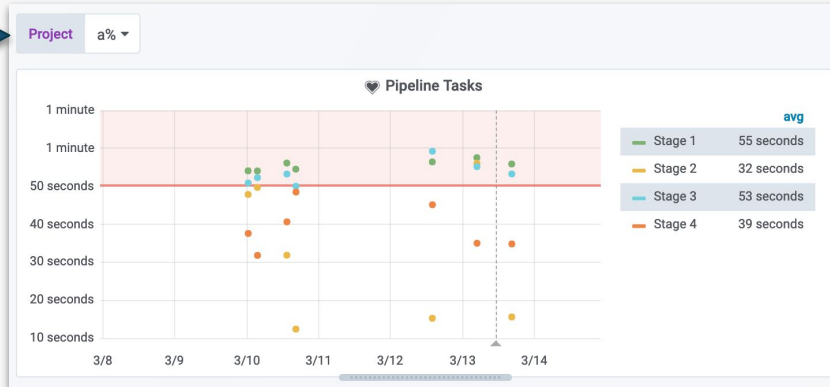
```
SELECT
  started AS "time",
  'Stage ' || stage::text as metric,
  extract(epoch from ended-started)
FROM
  pipeline_stages
WHERE
  started BETWEEN '2019-03-08T07:04:21
  AND project LIKE '%'
ORDER BY started
```

Making dashboards interactive with variables

Without Filter



With Filter



```
pipeline_stages
WHERE
  $__timeFilter(started)
  AND project LIKE '$Project'
ORDER BY started
```

Format as

Time series

Query Build

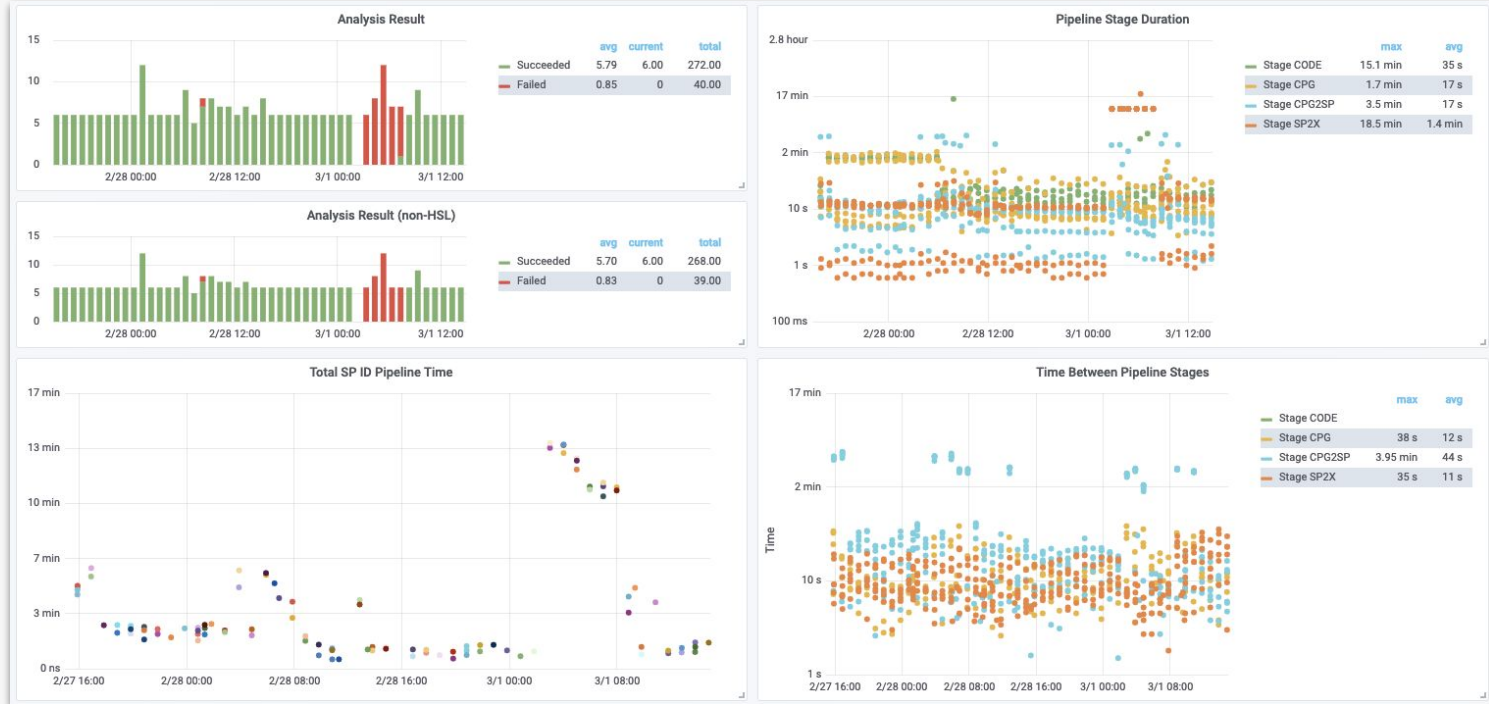
```
SELECT
  started AS "time",
  'Stage ' || stage::text as metric,
  extract(epoch from ended-started)
FROM
  pipeline_stages
WHERE
  started BETWEEN '2019-03-08T07:04:21'
  AND project LIKE '%'
ORDER BY started
```

Grafana+PostgreSQL "wins" at ShiftLeft

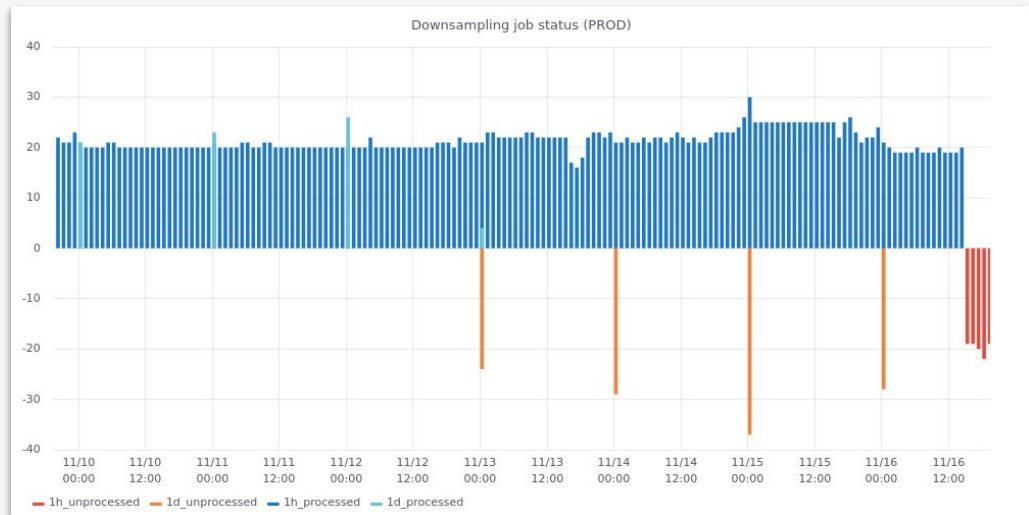
Overview

- **Before: problems required looking into the database**
 - One person with knowledge and credentials had to do it
 - Took a long time to format or interpret data
 - Depends on adhoc queries that weren't always documented
- **After: just look at Grafana**
 - Almost everyone has access to Grafana
 - Dashboards and visualizations are easy to interpret even across teams

Grafana+PostgreSQL "wins" at ShiftLeft: Reports

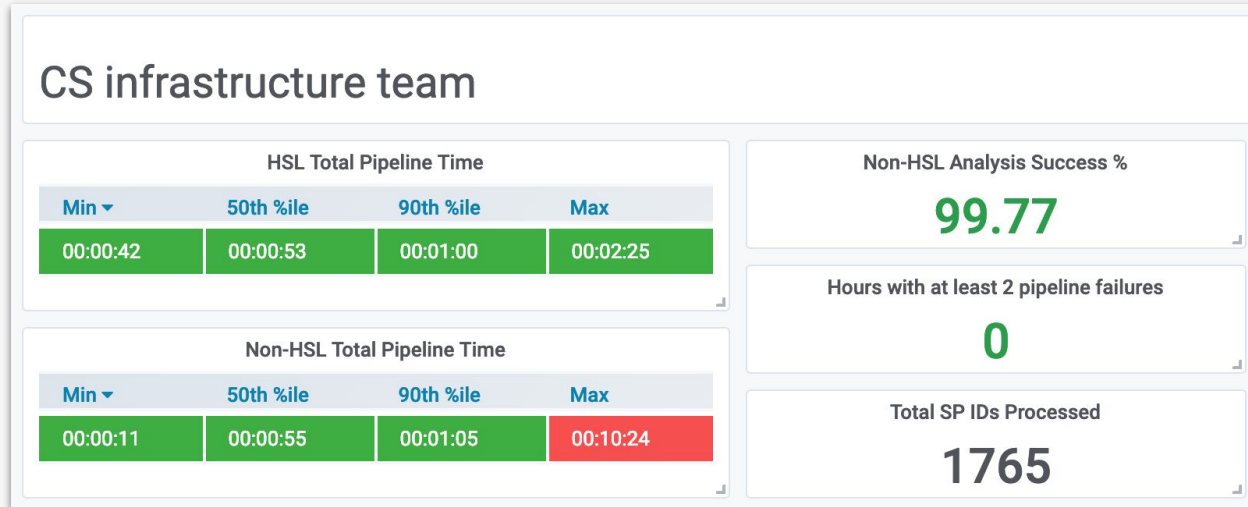


Grafana+PostgreSQL "wins" at ShiftLeft: Reports



```
SELECT
  $__time(time_bucket),
  granularity || 'unprocessed' AS metric,
  -SUM(CASE WHEN last_processed_time IS NULL THEN 1 ELSE 0 END)
  unprocessed
FROM
  metrics_downsampling_status
WHERE
  $__timeFilter(time_bucket)
GROUP BY
  time_bucket, granularity
ORDER BY time_bucket;
```

Grafana+PostgreSQL "wins" at ShiftLeft: Reports



Grafana+PostgreSQL "wins" at ShiftLeft: Data Exploration

team-cs-infra / CS Analysis Dashboard -

Org ID % ▾ Project name MAT ▾ SPID filter sl/ ▾ Limit 100 ▾

SP ID Overview

SP ID ▾	Created	Conclusions	Violations
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/9	2018-12-19T00:28:16.568336Z	15.00	217.00
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/8	2018-12-19T00:28:36.862711Z	15.00	217.00
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/7	2018-12-19T00:27:55.79153Z	15.00	217.00
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/1	2018-12-19T00:19:17.686596Z	15.00	217.00
sl/7ed1c663-726d-4560-afac-547583582841/hsl-01.21.2019-MAT/87d98b516be2731e7dd2eb898121a496917cb689/76144cd5a42de460b67545ccc5c21066469fbbd20e7b6a79af8acbfd06bba4f9/1	2019-01-21T23:30:40.52704Z	15.00	213.00
sl/7ed1c663-726d-4560-afac-547583582841/hsl-01.11.2019-MAT/87d98b516be2731e7dd2eb898121a496917cb689/76144cd5a42de460b67545ccc5c21066469fbbd20e7b6a79af8acbfd06bba4f9/1	2019-01-11T21:05:46.743161Z	15.00	213.00

Analysis History

SP ID	Created ▾
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/1	2018-12-18 16:19:17
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/7	2018-12-18 16:27:55
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/9	2018-12-18 16:28:16
sl/7ed1c663-726d-4560-afac-547583582841/hsl-12.18.2018-MAT/87d98b516be2731e7dd2eb898121a496917cb689/59ffae02da849df6ef210fcb3af01a75e1c995a8fc266d78e83455d307672e7/8	2018-12-18 16:28:36
sl/7ed1c663-726d-4560-afac-547583582841/hsl-01.11.2019-MAT/87d98b516be2731e7dd2eb898121a496917cb689/76144cd5a42de460b67545ccc5c21066469fbbd20e7b6a79af8acbfd06bba4f9/1	2019-01-11 13:05:46

Conclusions Breakdown

Conclusion ▾	Violations
attacker-to-file	4.00
database-to-log	1.00
env-data-leak	7.00
env-data-leak-to-net	1.00

Endpoints Breakdown

IO Type ▾	Endpoint	Routes
input	database	0
input	debug	1.00
input	environment	0
input	execution	0

Endpoints and Routes

io_type ▾	Endpoint	Route
input	http	/patients
input	http	/
input	http	/account/{accountId}/deposit
input	http	/admin/printSecrets

Questions?

Feel free to send me Grafana/PostgreSQL questions on Twitter:
@PreetamJinka