
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022
TiKV
TiKV
HELM

Tamland

How GitLab.com uses long-term monitoring data for capacity forecasting

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

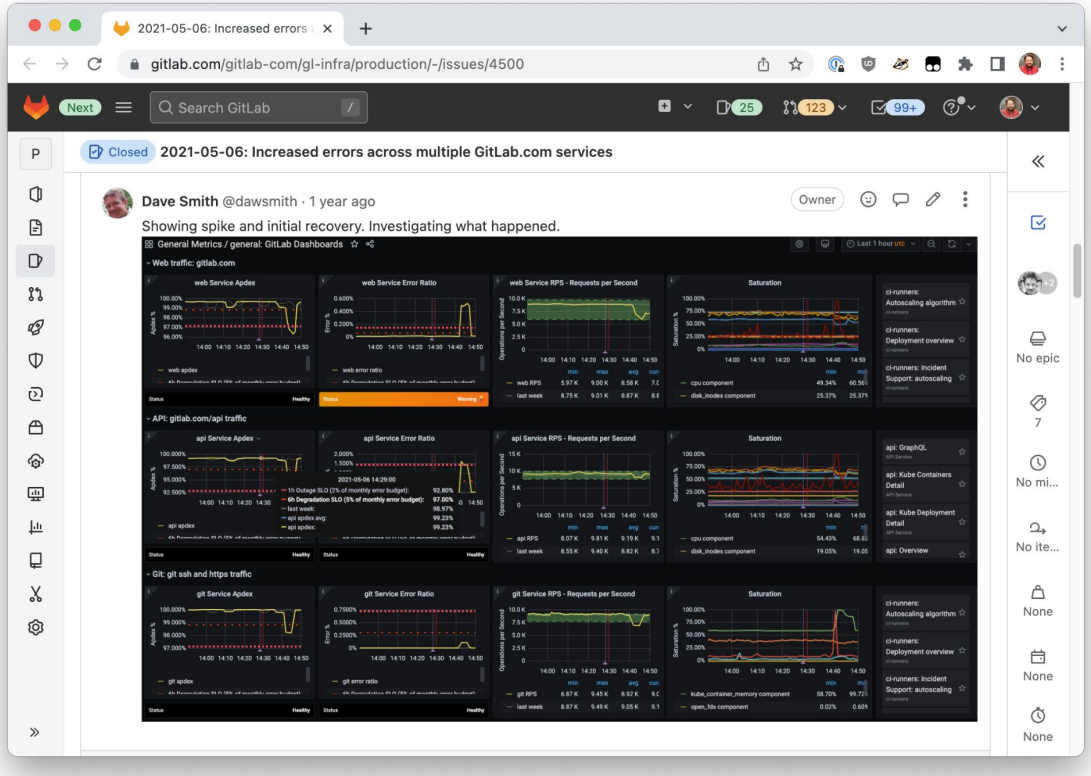
October 24-28, 2022

Andrew Newdigate
Distinguished Engineer, Infrastructure
GitLab, Inc.
twitter.com/suprememoocow

GitLab

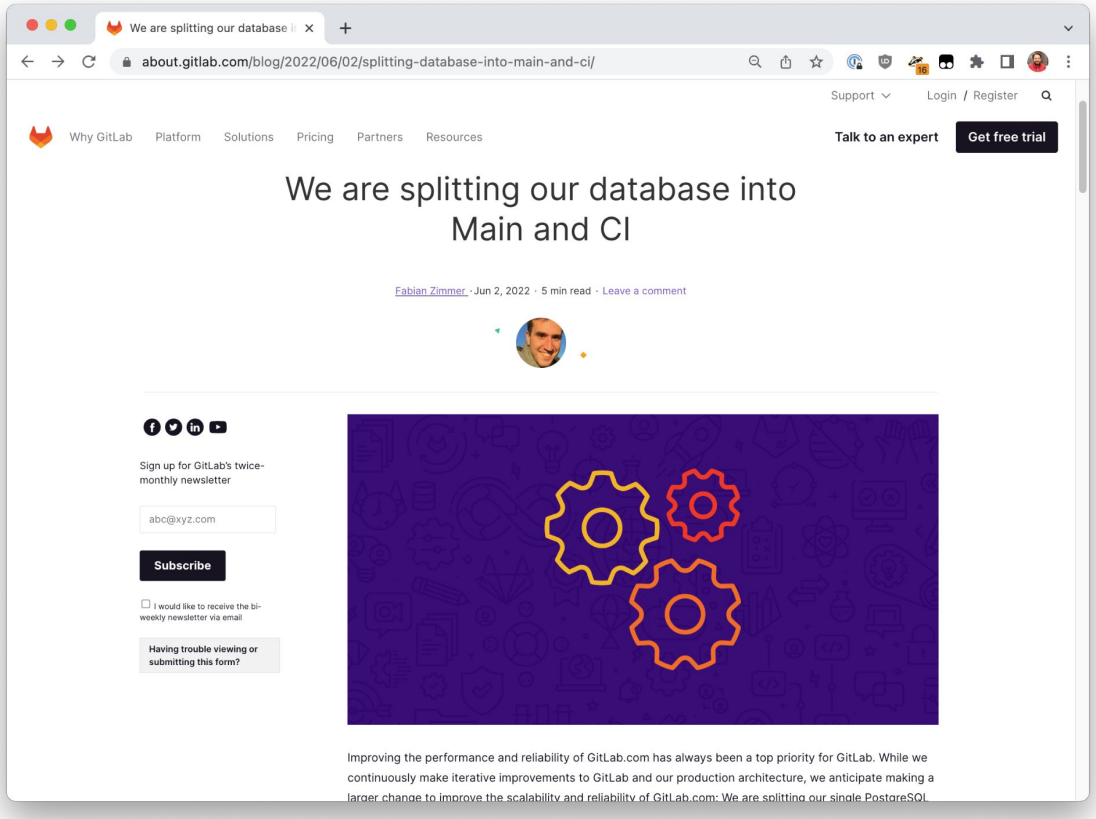
A series of Postgres saturation events..

- In **mid-2021**, GitLab.com experienced a series of database incidents, related to high resource utilization on our primary Postgres database.
- Inside the company, concerns were raised about growth rate of GitLab.com and **the ability for our primary database to support this growth**.
- There were concerns that **the database would hit maximum capacity**, leading to outages due to further saturation.



It’s probably time we break up, Postgres

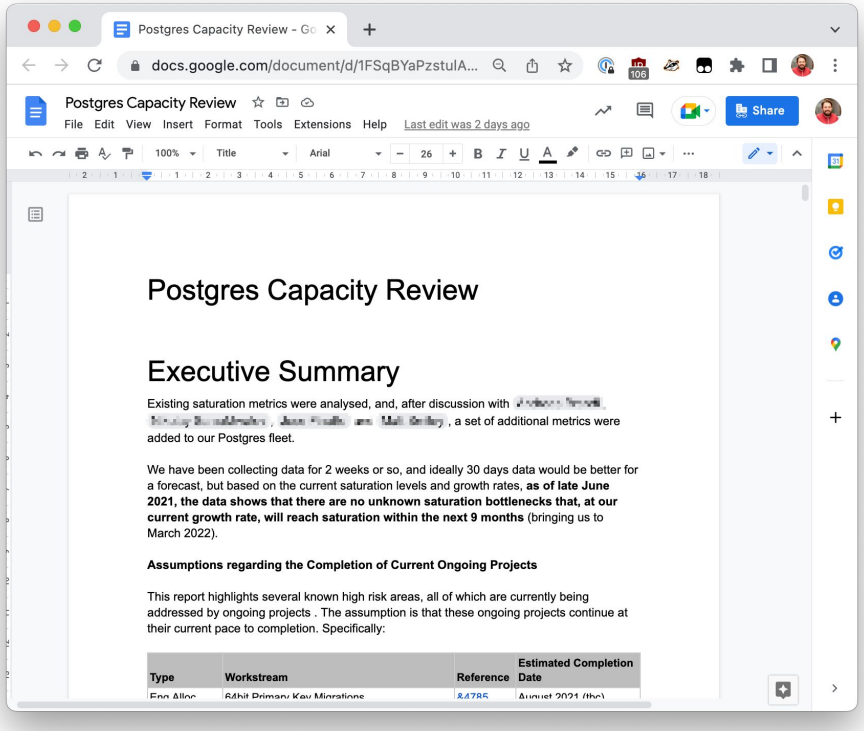
- Path forward would be split our main Postgres cluster into two separate clusters, through a process known as **functional decomposition**.
- This would be a **complicated software operation**, which would need to be prepared and performed carefully to avoid data-loss or outages.
- **Rushing the migration would be risky**.
- At the same time, **taking too long would be disastrous** if the Postgres server hit capacity and was nzswwo longer able to deal with further growth in traffic.



about.gitlab.com/blog/

Is there enough runway?

- It was important to know whether the **timeframes for the functional decomposition project were within the available capacity timeframes** for the main Postgres cluster.
- To answer this, a **capacity planning review** for Postgres was performed.
- This was done with Tamland, **GitLab.com's capacity planning tool**.
- The review confirmed that we had enough runway left, allowing the team to focus on the migration.



GitLab

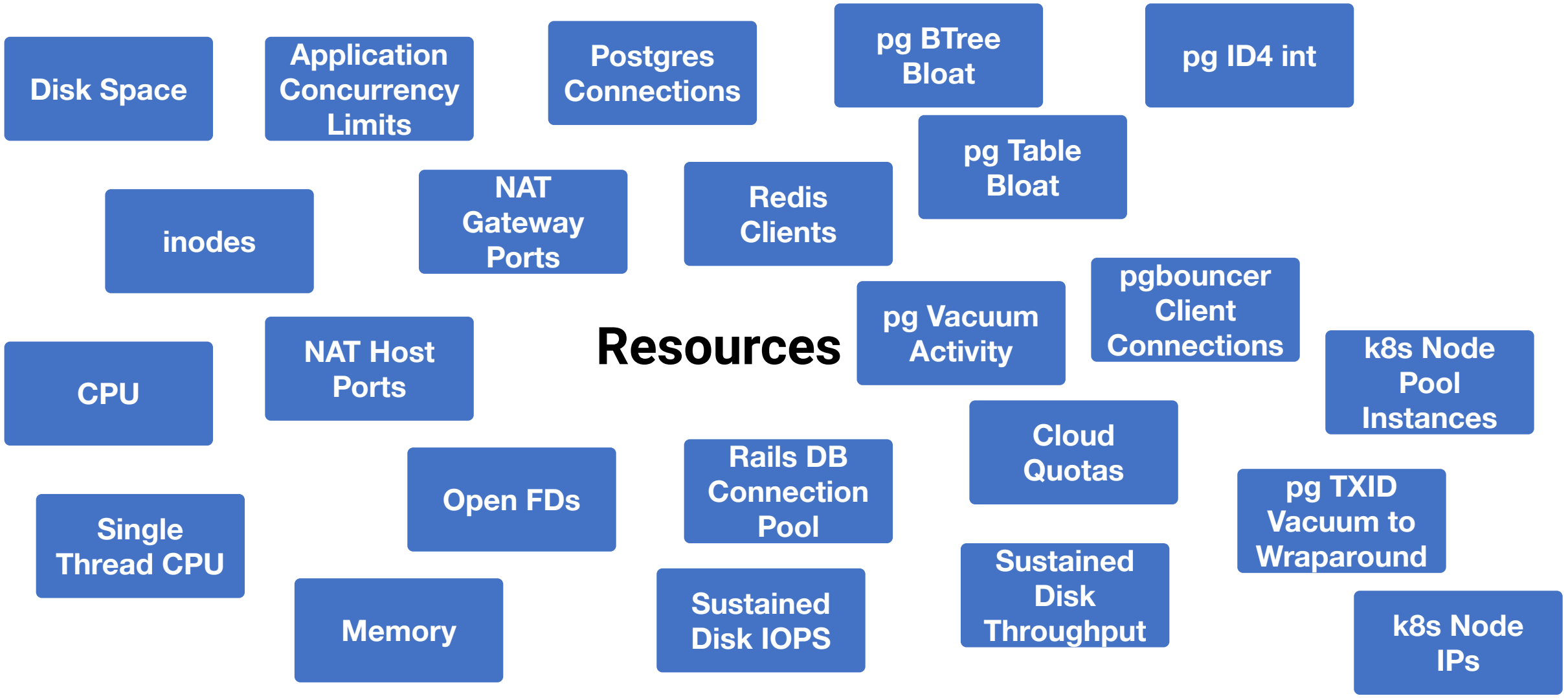
*"**The goal of capacity management is controlling uncertainty**. In the midst of the unknown, the service must be available now and continue to run in the future. A challenging but rewarding and delicate balance of tradeoffs is in play: efficiency vs. reliability, accuracy vs. complexity, and effort vs. benefit."*

SRE Best Practices for Capacity Management, Luis Quesada Torres and Doug Colish

https://research.google/pubs/pub50108/

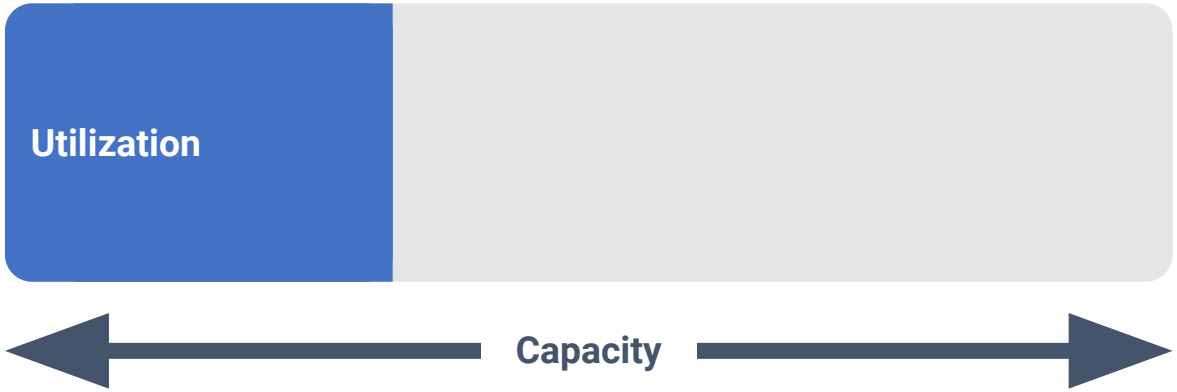
First, Some Definitions

Definitions: Resources



Resource definitions for GitLab.com: https://gitlab.com/gitlab-com/runbooks/-/tree/master/libsonnet/saturation-monitoring

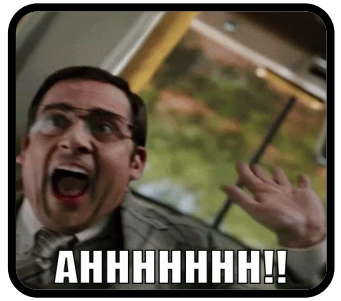
Definitions



SATURATION

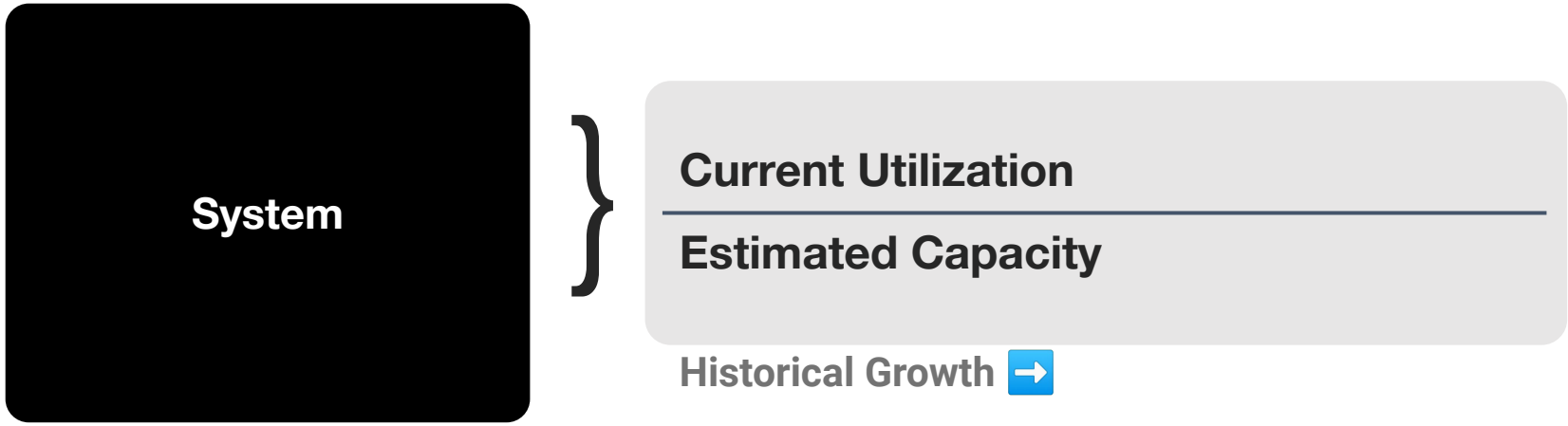
Saturation Events vs. Mitigation Time

Resource	Saturation Failure Mode	Migitation	Mitigation Time
CPU	Queuing, Increased application latency	Increase maximum CPU capacity in cluster.	Minutes
Memory	OOM kills, transient errors	Increase maximum memory capacity in cluster.	Minutes
Inodes / Disk Space	Write failures, corruption	Increase disk capacity	Potentially hours
Postgres Transaction ID Wraparound	Database shutdown	Postgres Recovery	Hours, **days**
Redis Single-Threaded CPU	Increased application latency, errors	Refactor application to shard Redis, migrate to Redis cluster.	**Weeks**, **months**
Postgres 32 bit ID Wraparound	Database shutdown	Migrate from 32 bit IDs to 64 bit IDs	**Months**
Postgres Primary CPU, no further vertical scaling possible	Increased application latency, errors	Postgres Functional Decomposition	**Months**



GitLab.com's approach to capacity planning...

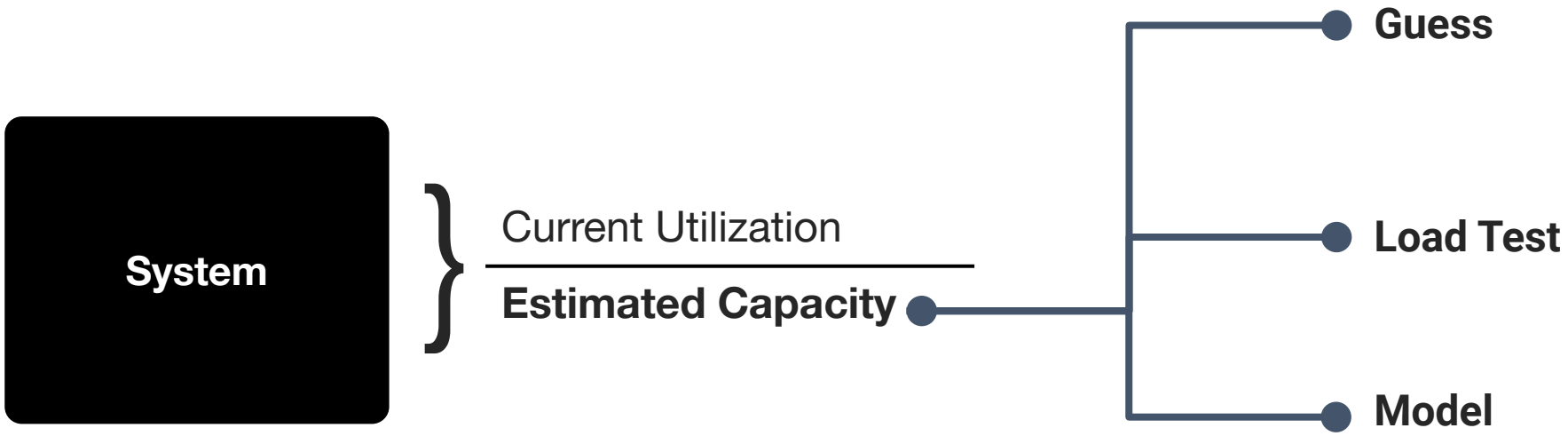
System Capacity Planning



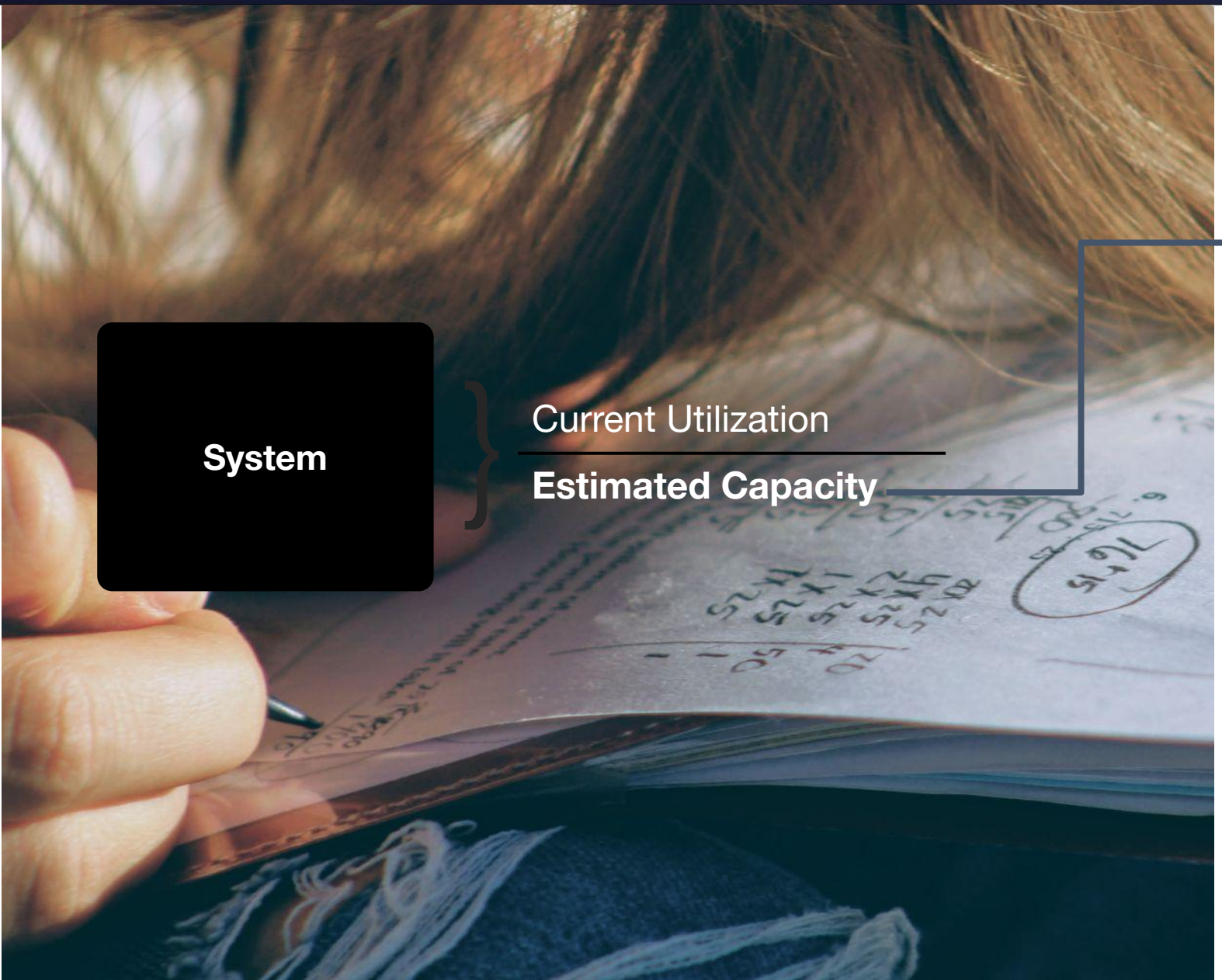
Demand Signals

- Monthly Active Users
- Requests per Second
- Concurrent Users
- etc

Estimating System Capacity is Hard



Guessing Maximum Capacity

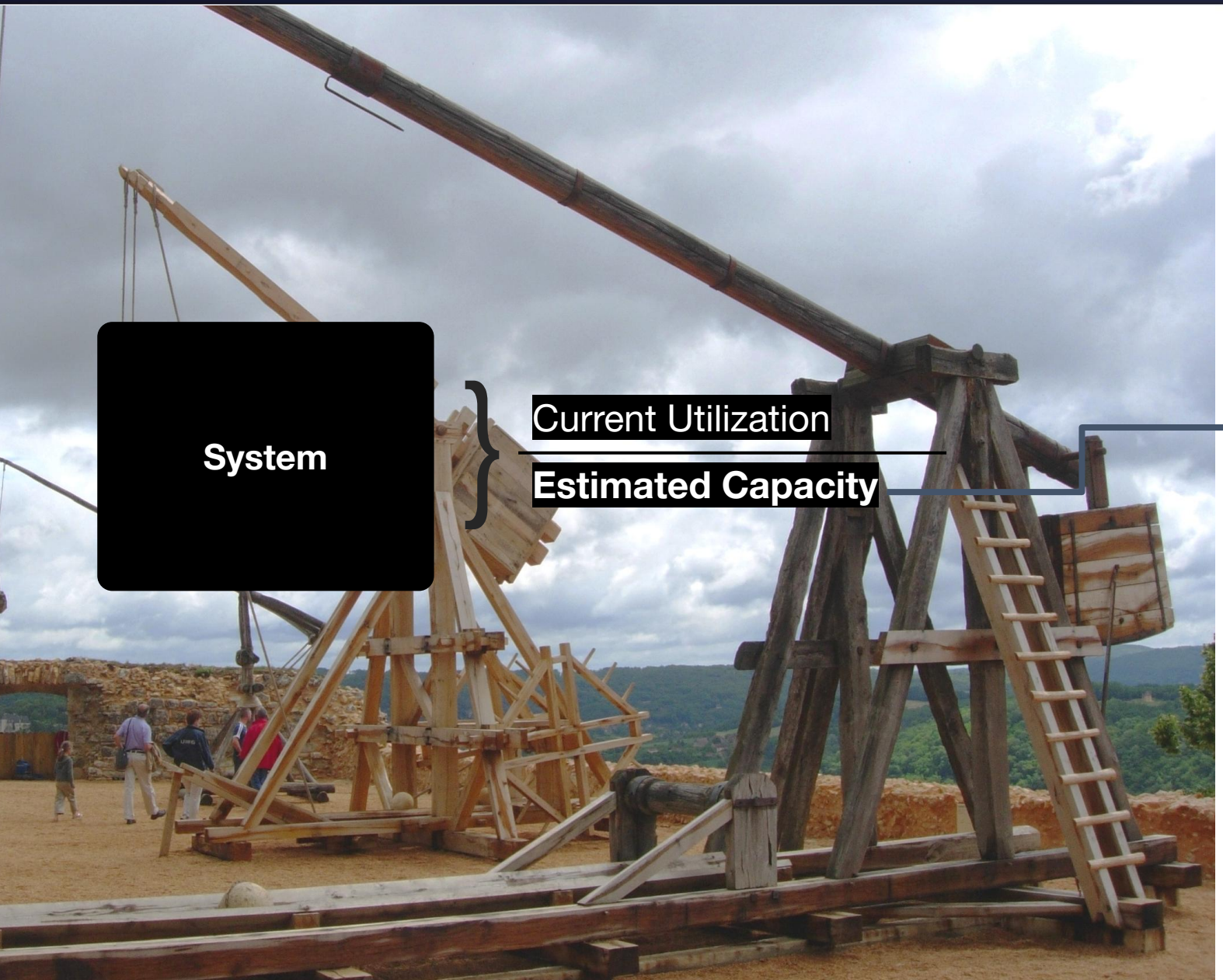


Guess

- aka, the "Back of the Envelope" method
- Likely inaccurate
- Not scalable
- Not automatable

Photo by Joshua Hoehne on Unsplash

Load Testing Maximum Capacity

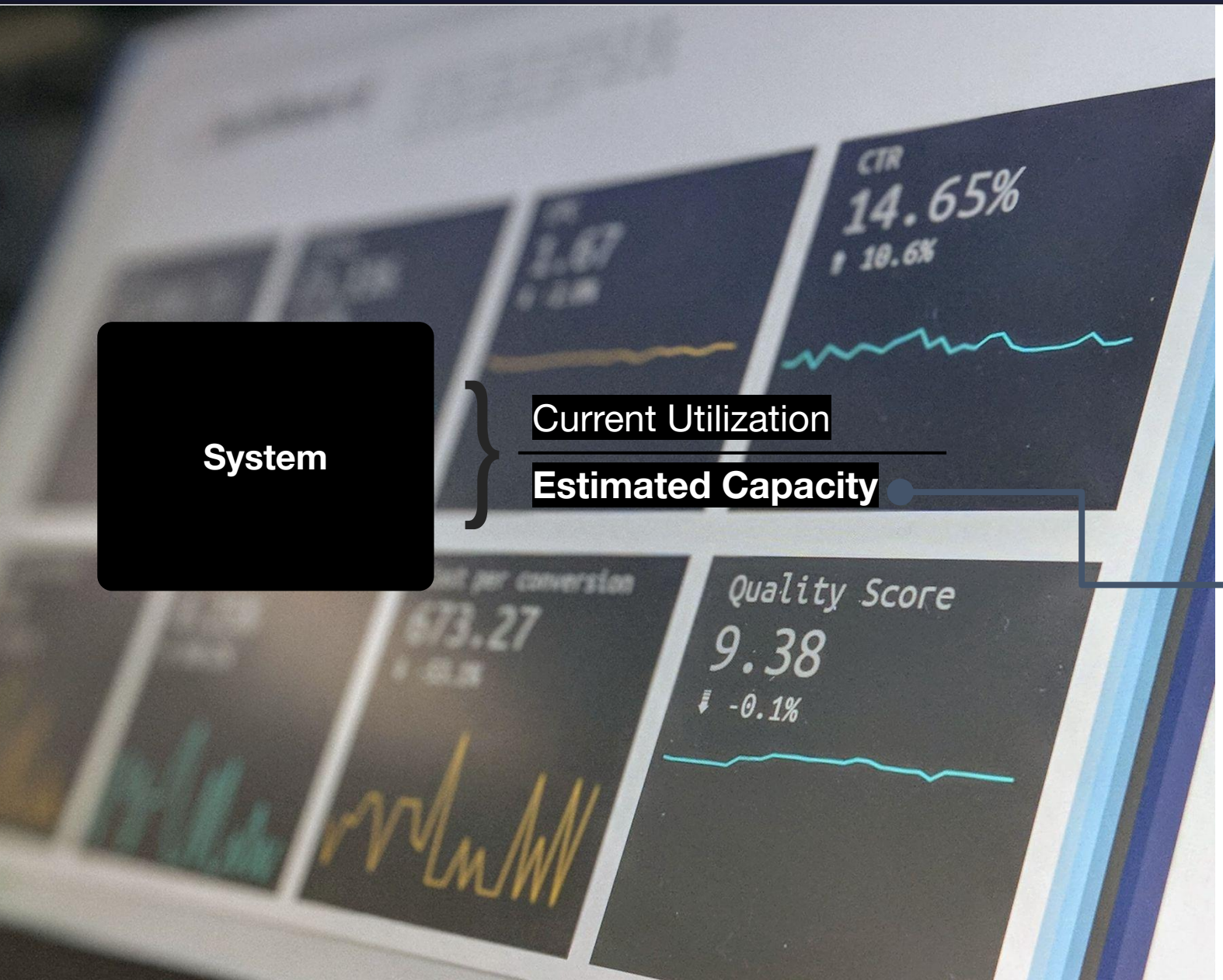


Load Test

- Setting up a equivalent testbed can be time-consuming and expensive.
- May miss important failure modes
- Difficult to guess correct workloads
- New features require that testing be redone.

Photo by Luc Viatour on Lucnix.be

Modelling Maximum Capacity



System

Current Utilization

Estimated Capacity

Model

- Complicated to build
- Difficult to confirm
- Requires updating as application and production environment evolve.

Photo by Stephen Dawson on Unsplash

Estimating System Capacity is Hard

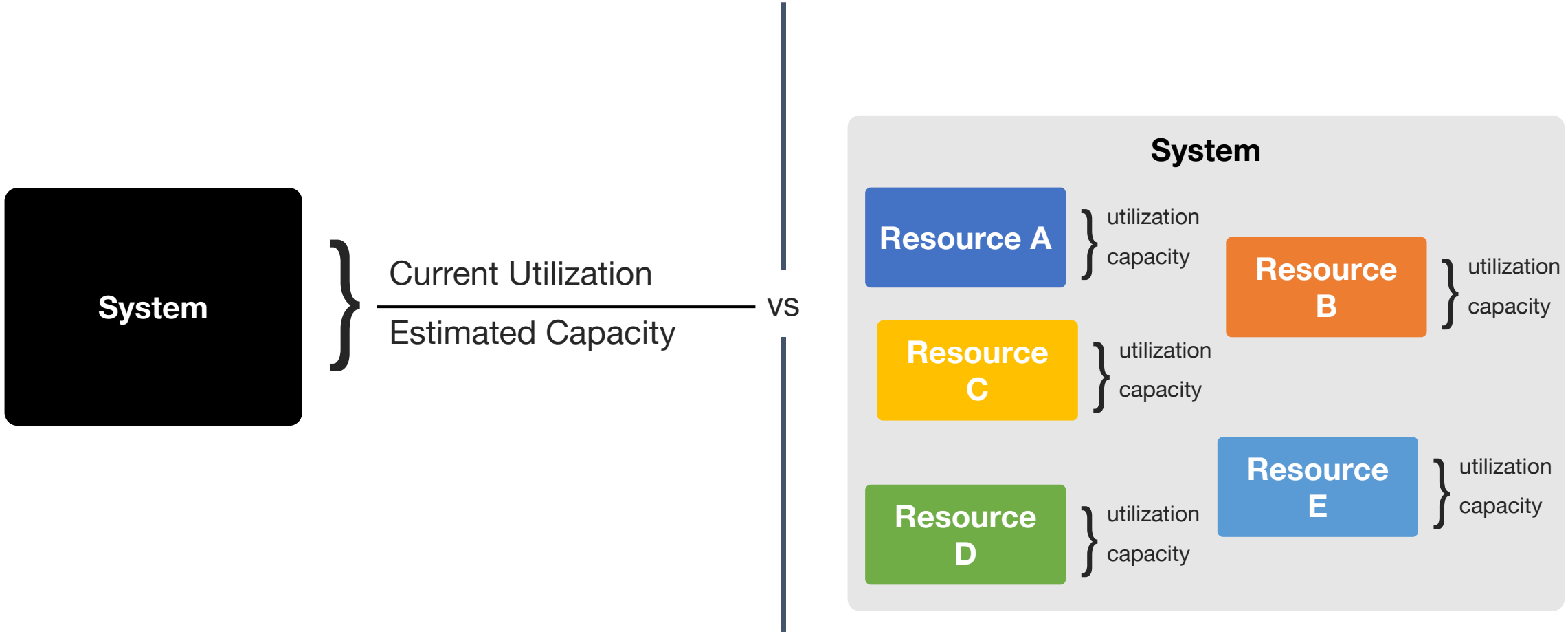
System

$$\frac{\text{Current Utilization}}{\textbf{Estimated Capacity ???}}$$

Photo by Jon Tyson on Unsplash

Resource Capacity, not System Capacity

Instead of treating system capacity as a blackbox, break the system down into the individual resources within the system, and measure, track and forecast each one individually (aka whitebox).



Resource/Service Matrix

	Service A	Service B	Service C	Service D	Service E	Service F
cgroup memory		✅	✅	✅		✅
CPU	✅		✅	✅	✅	✅
Disk Space	✅	✅			✅	✅
GCP Quota Limits	✅					
Disk inodes	✅	✅	✅	✅		✅
PGBouncer Connections Primary		✅				
Single Threaded CPU	✅		✅			
Private Runners			✅			
Public Runners			✅			
Disk IOPS		✅		✅		
Disk Read Throughput		✅		✅		
Disk Write Throughput		✅		✅		
NAT Gateway Port Allocation	✅					
NAT Host Port Allocation	✅					

Building a Prometheus Model for Monitoring Utilization…

Measuring Utilization Consistently

Using Recording Rules

```
gitlab_resource_utilization:ratio{service="postgres", resource="primary_cpu"}

gitlab_resource_utilization:ratio{service="postgres", resource="pgbouncer_connection_pool"}

gitlab_resource_utilization:ratio{service="redis", resource="redis_memory"}

gitlab_resource_utilization:ratio{service="gitaly", resource="total_disk_capacity"}

gitlab_resource_utilization:ratio{service="web", resource="rails_db_connection_pool"}

gitlab_resource_utilization:ratio{service="nat", resource="nat_host_port_allocation"}
```

Prometheus Recording Rules


```yaml
- record: gitlab_component_saturation:ratio
  labels:
    resource: gcp_quota_limit
  expr: |
    gcp_quota_usage
    /
    gcp_quota_limit
```

1

```yaml
232   - record: gitlab_component_saturation:ratio
233     labels:
234       component: open_fds
235     expr: |
236       process_open_fds
237       /
238       process_max_fds
```

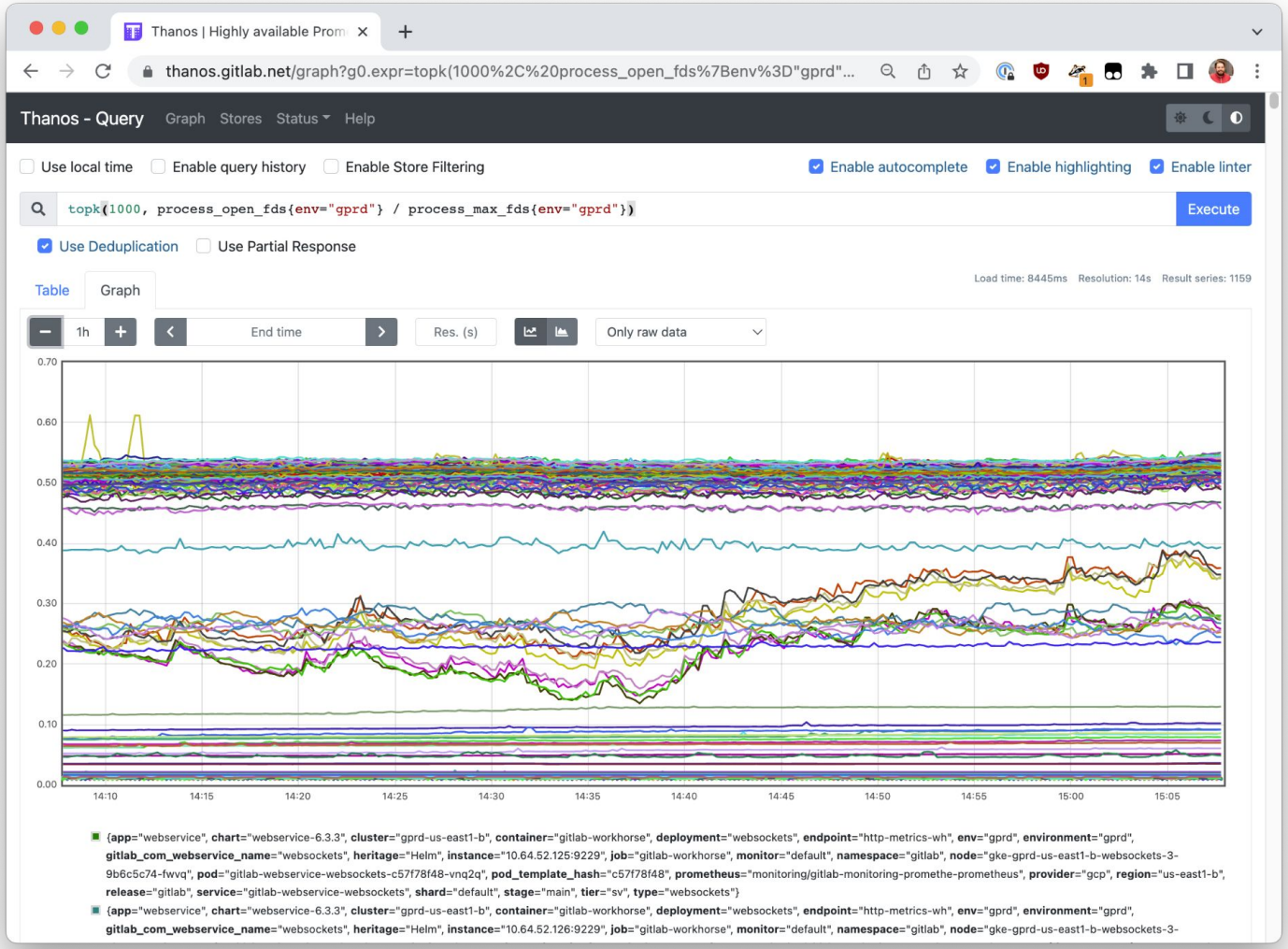
2

Robust Perception: [Alerting on approaching open file limits]

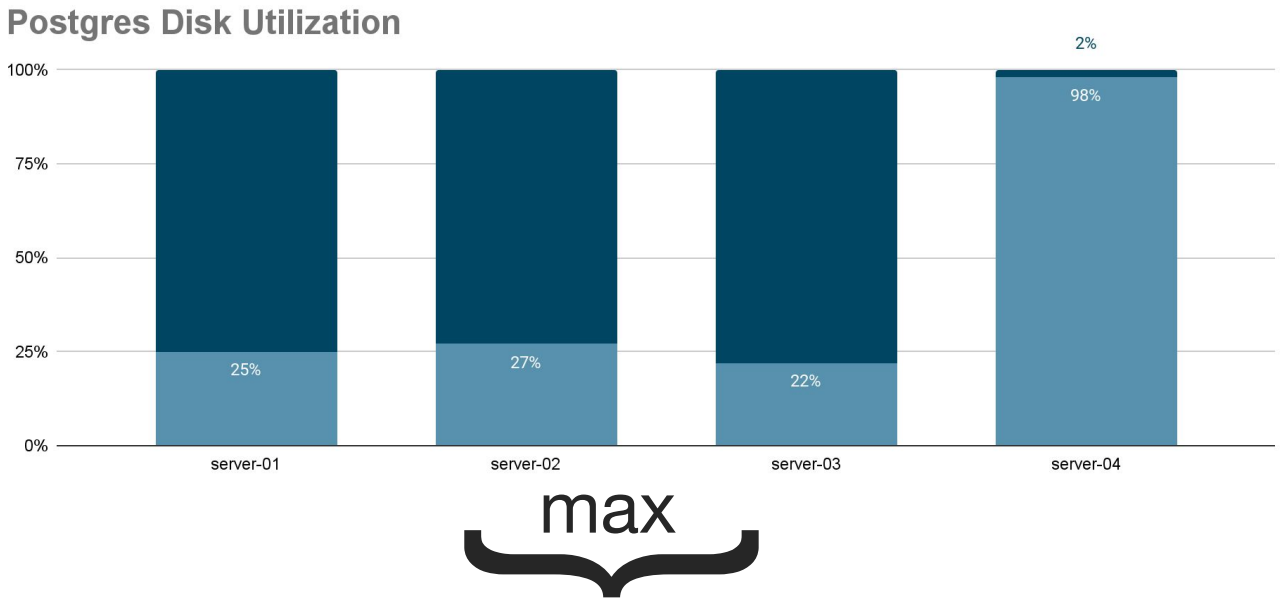
```yaml
- record: gitlab_component_saturation:ratio
  labels:
    component: kube_persistent_volume_claim_disk_space
  expr: |
    kubelet_volume_stats_used_bytes
    /
    kubelet_volume_stats_capacity_bytes
```

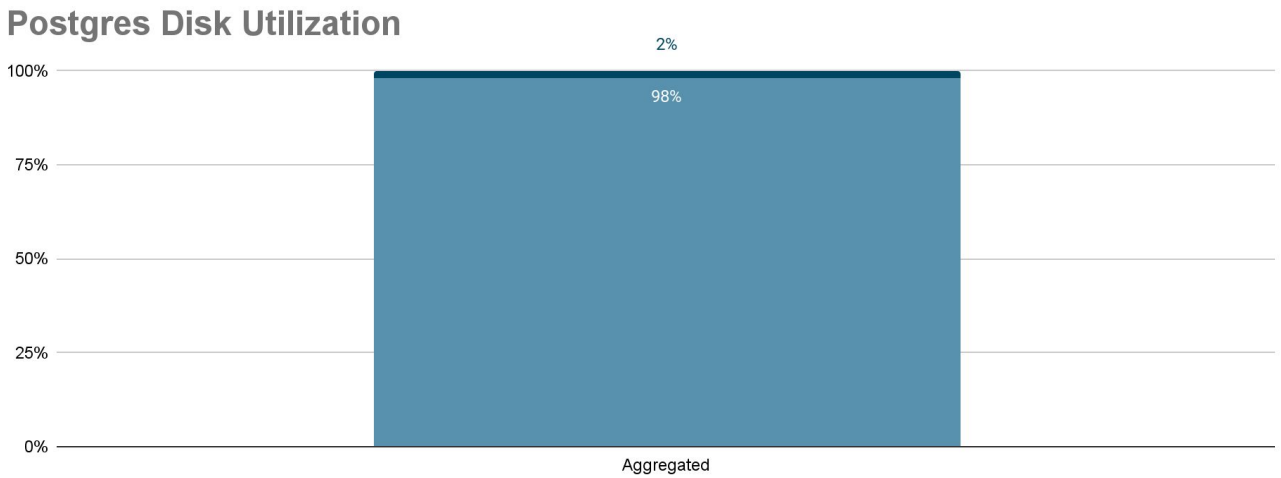
3

Aggregating to Reduce Data



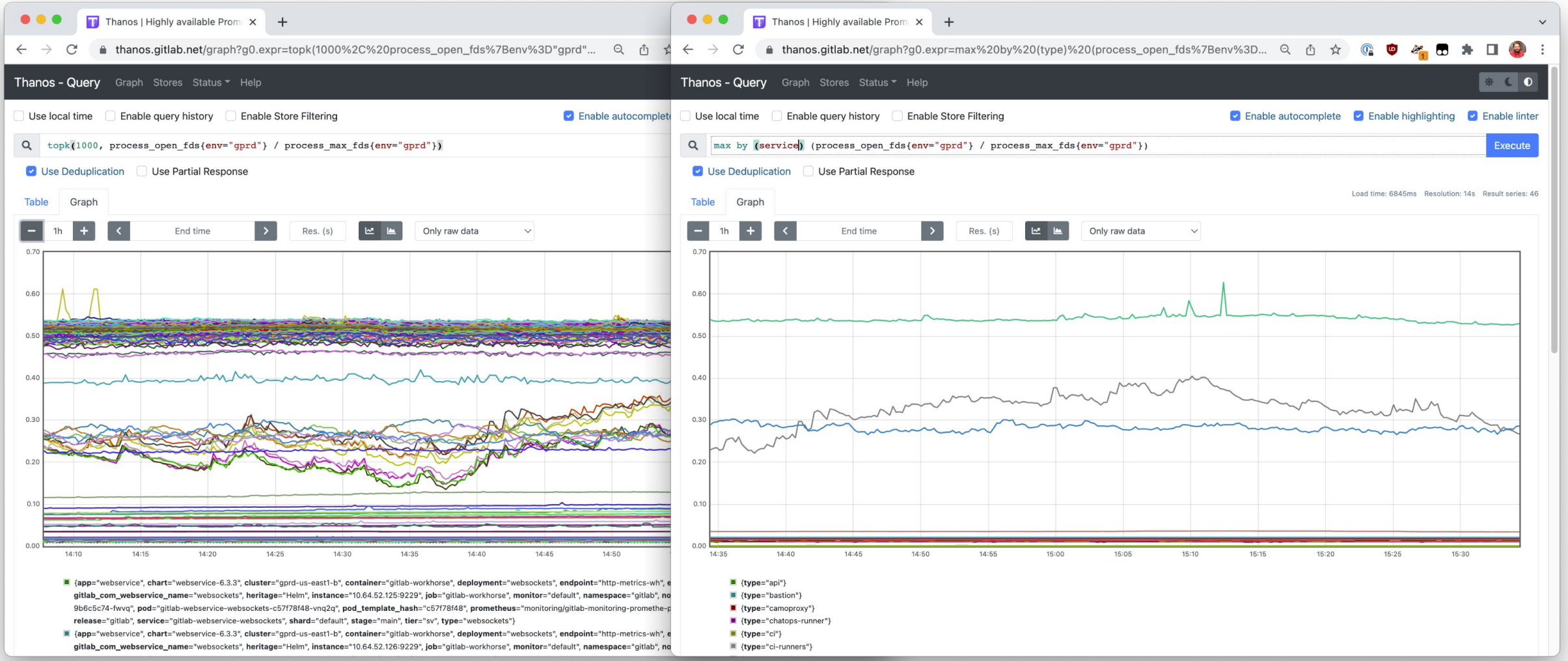
Aggregating Utilization Across Resources





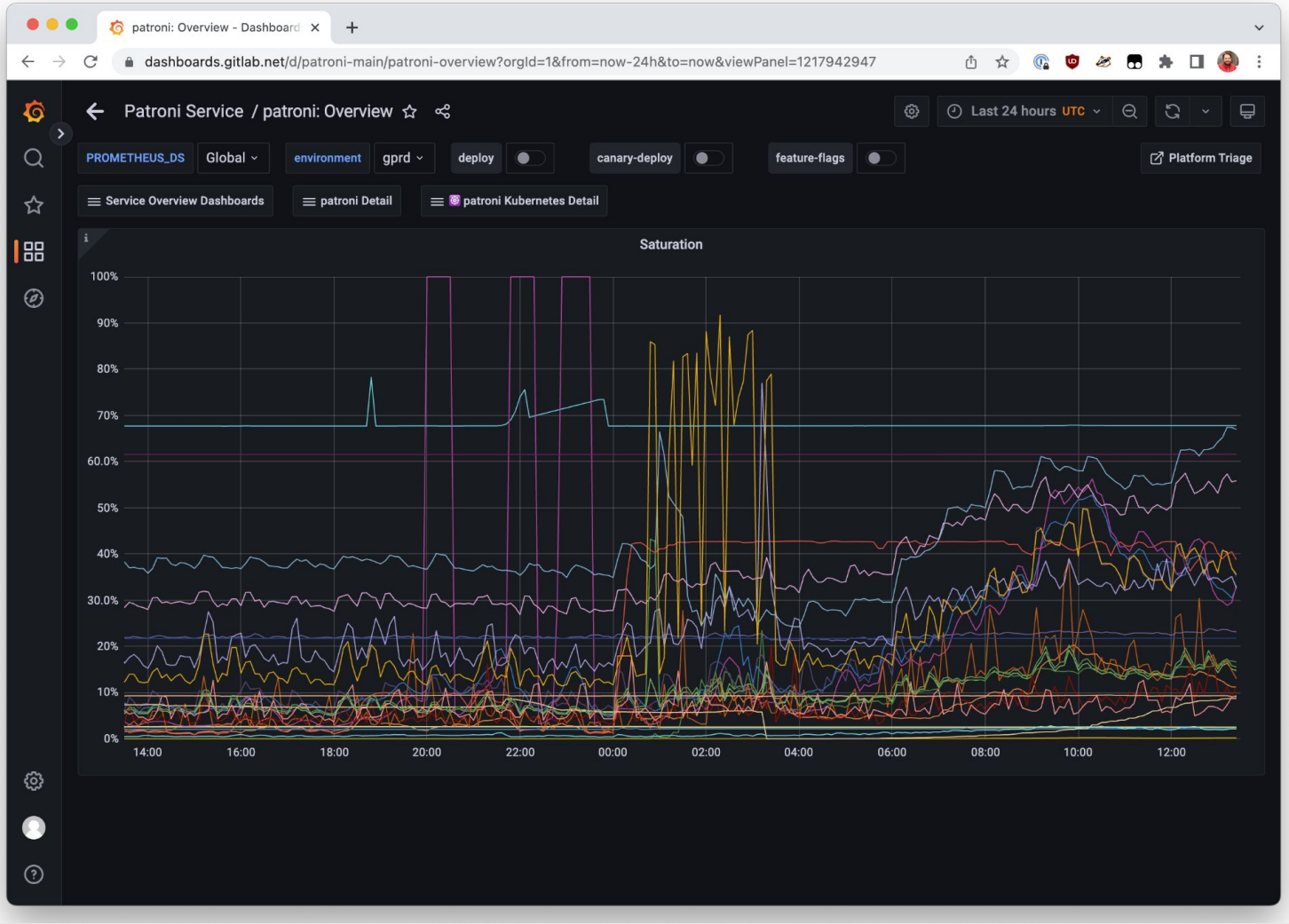

```
max by(service) (
  1 -
  node_filesystem_avail_bytes
  /
  node_filesystem_size_bytes
)
```

Aggregating to Reduce Data

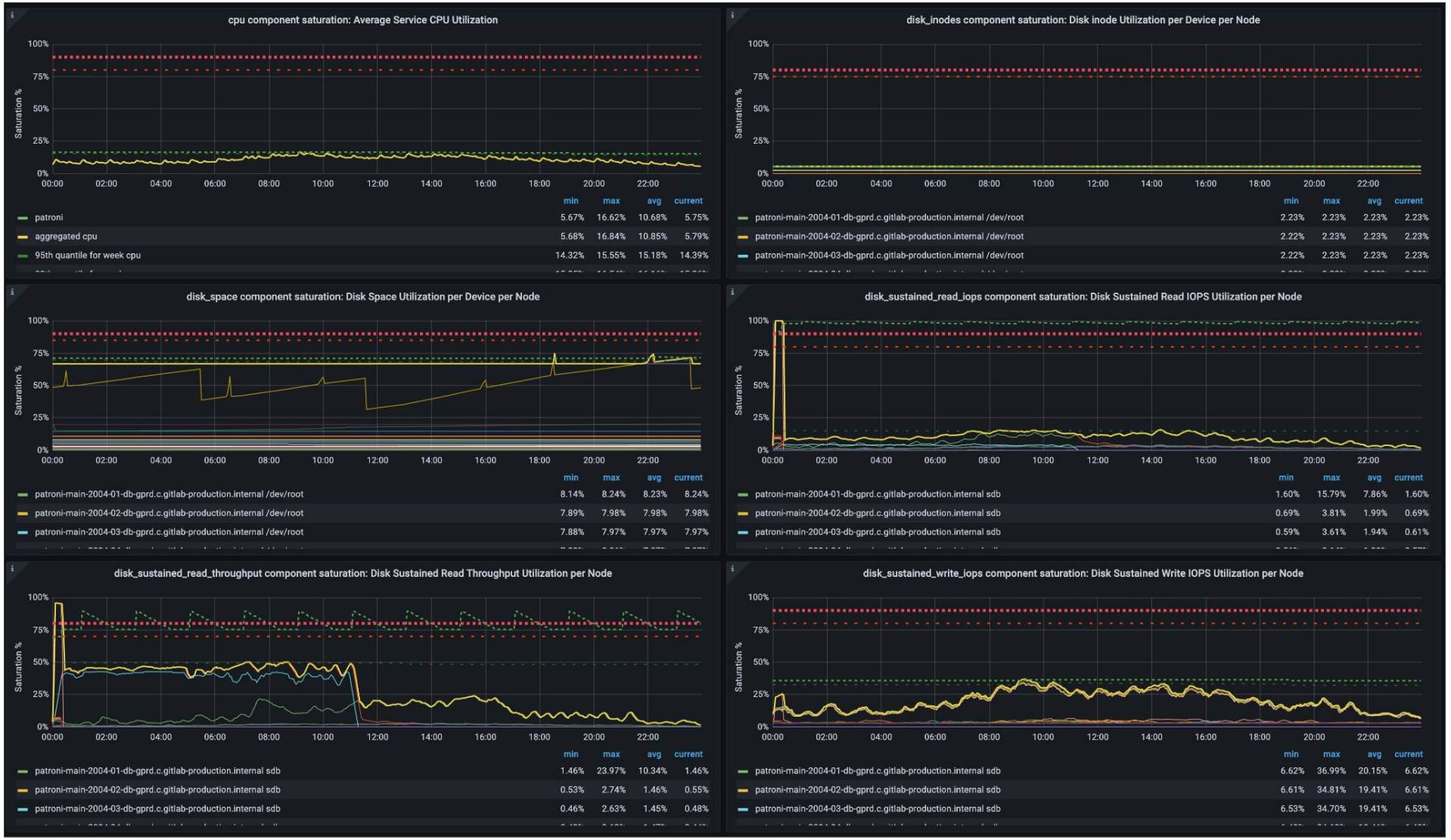


```
{
  disk_space: resourceSaturationPoint({
    title: 'Disk Space Utilization per Device per Node',
    horizontallyScalable: true,
    description: |||
      Disk space utilization per device per node.
    |||,
    grafana_dashboard_uid: 'sat_disk_space',
    resourceLabels: ['device'],
    // We filter on `fqdn=""` to filter out any nameless workers. This is done mostly for the ci-runner fleet
    query: |||
      1 - (
        node_filesystem_avail_bytes{%(selector)s}
        /
        node_filesystem_size_bytes{%(selector)s}
      )
    |||,
    slos: {
      soft: 0.85,
      hard: 0.90,
      alertTriggerDuration: '15m',
    },
  }),
}
```

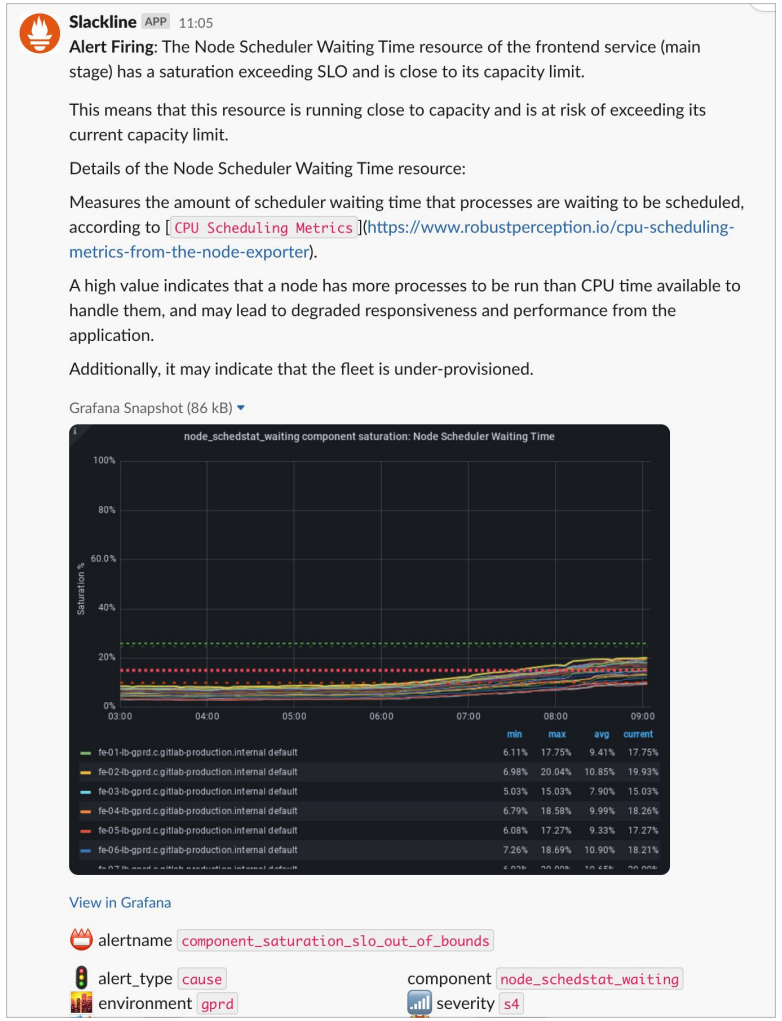
Real-time Utilization Monitoring



Real-time Utilization Monitoring (Detail)



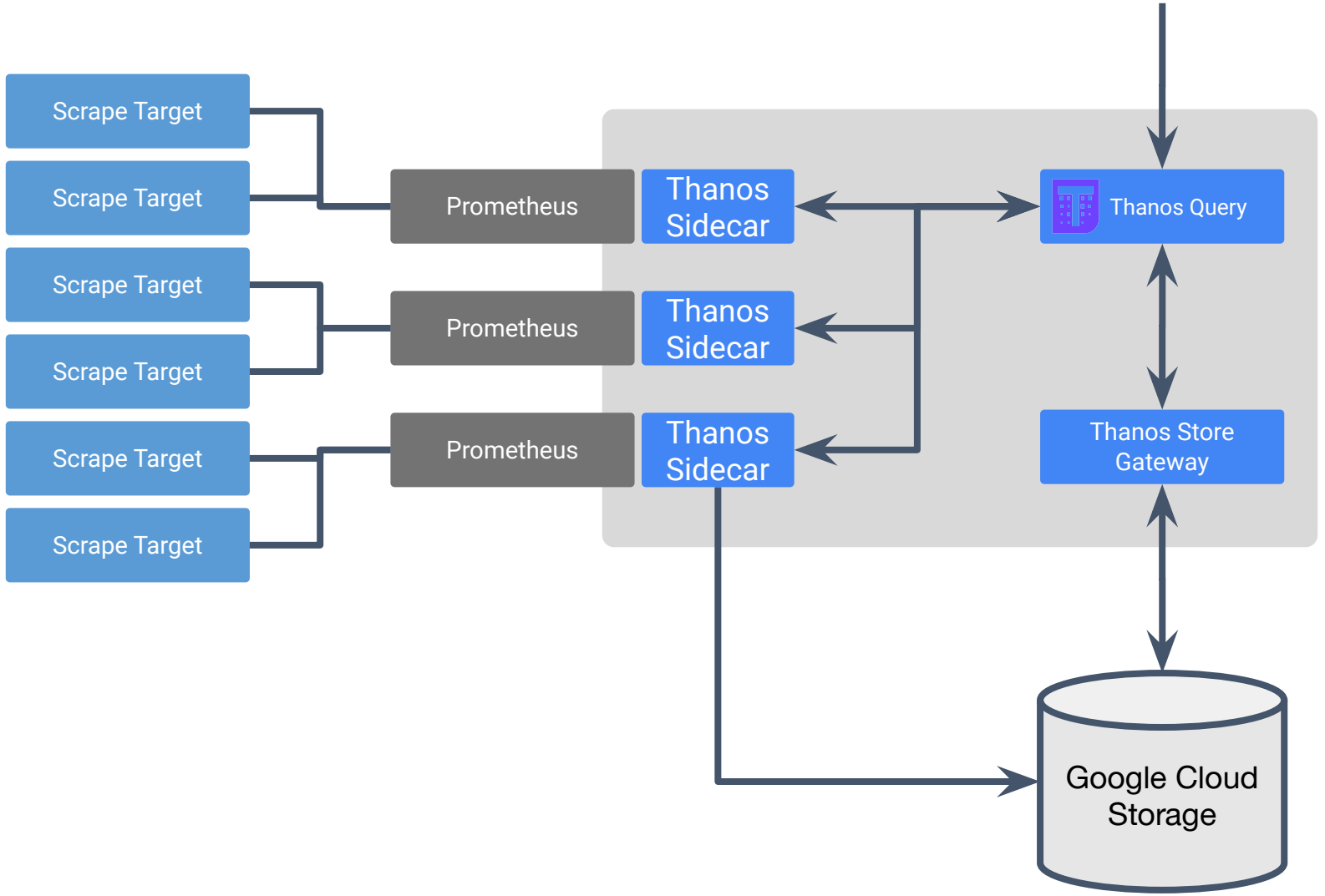
Alerting



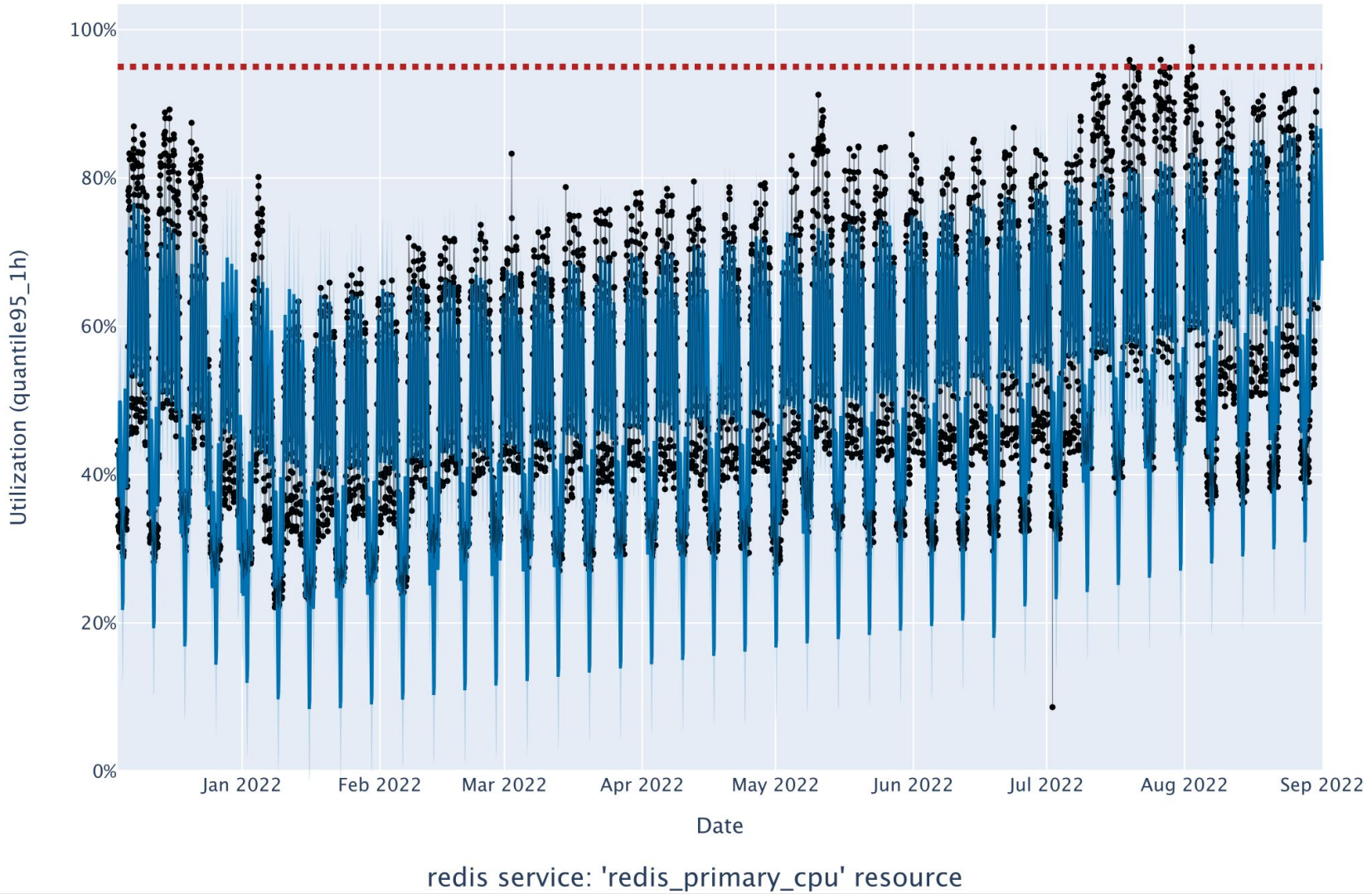
- **Alerting thresholds are usually less than 100%.**
- Alerting threshold depends on the nature of the resource being monitored, severity of saturation, etc.
- **Some can be very low** (this example has a threshold of only 20%)
- When an alert fires, Slackline tool will **generate a snapshot of the saturation detail graph** and post it to Slack (example left)

Tamland, Forecaster

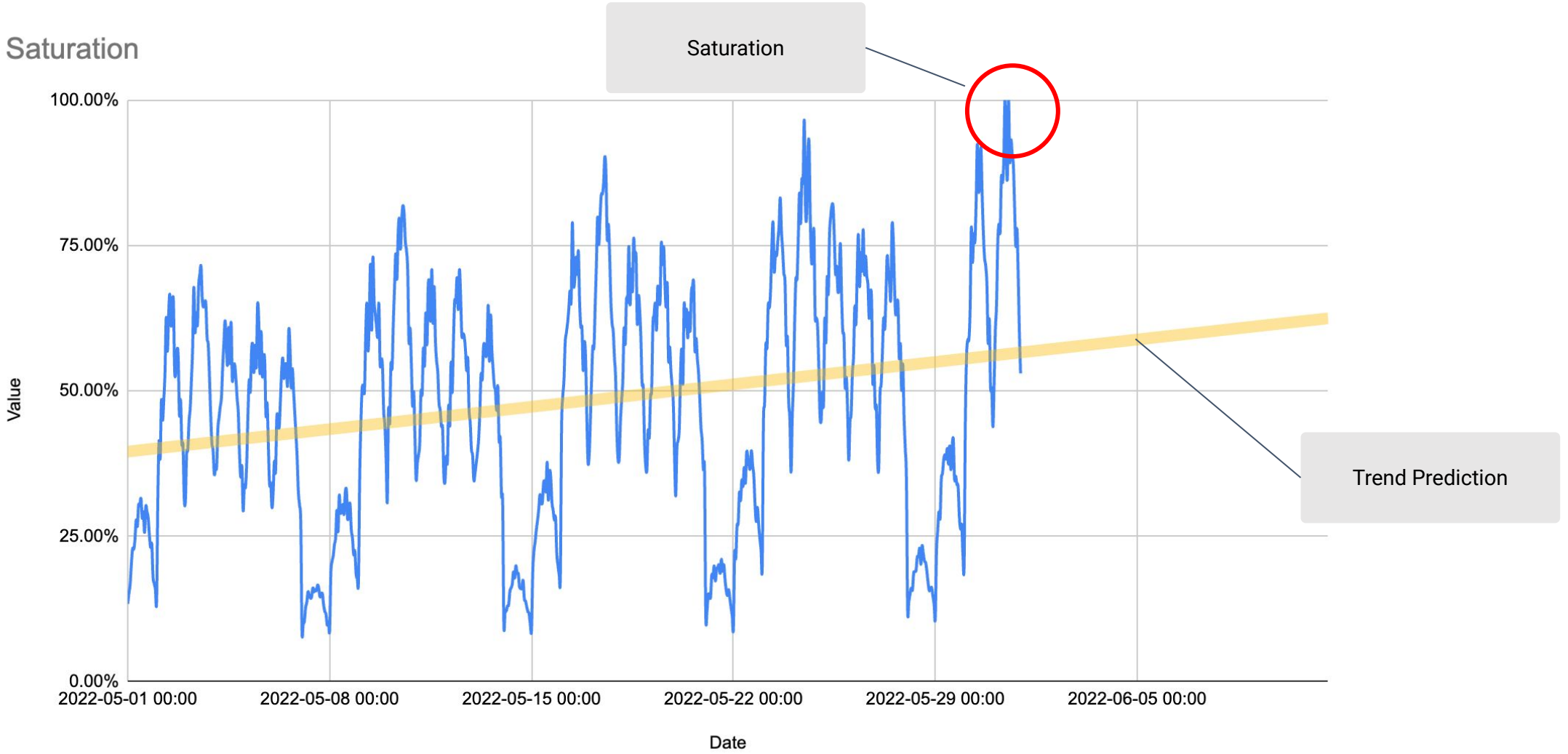
Using Thanos for Long-Term Metrics Storage



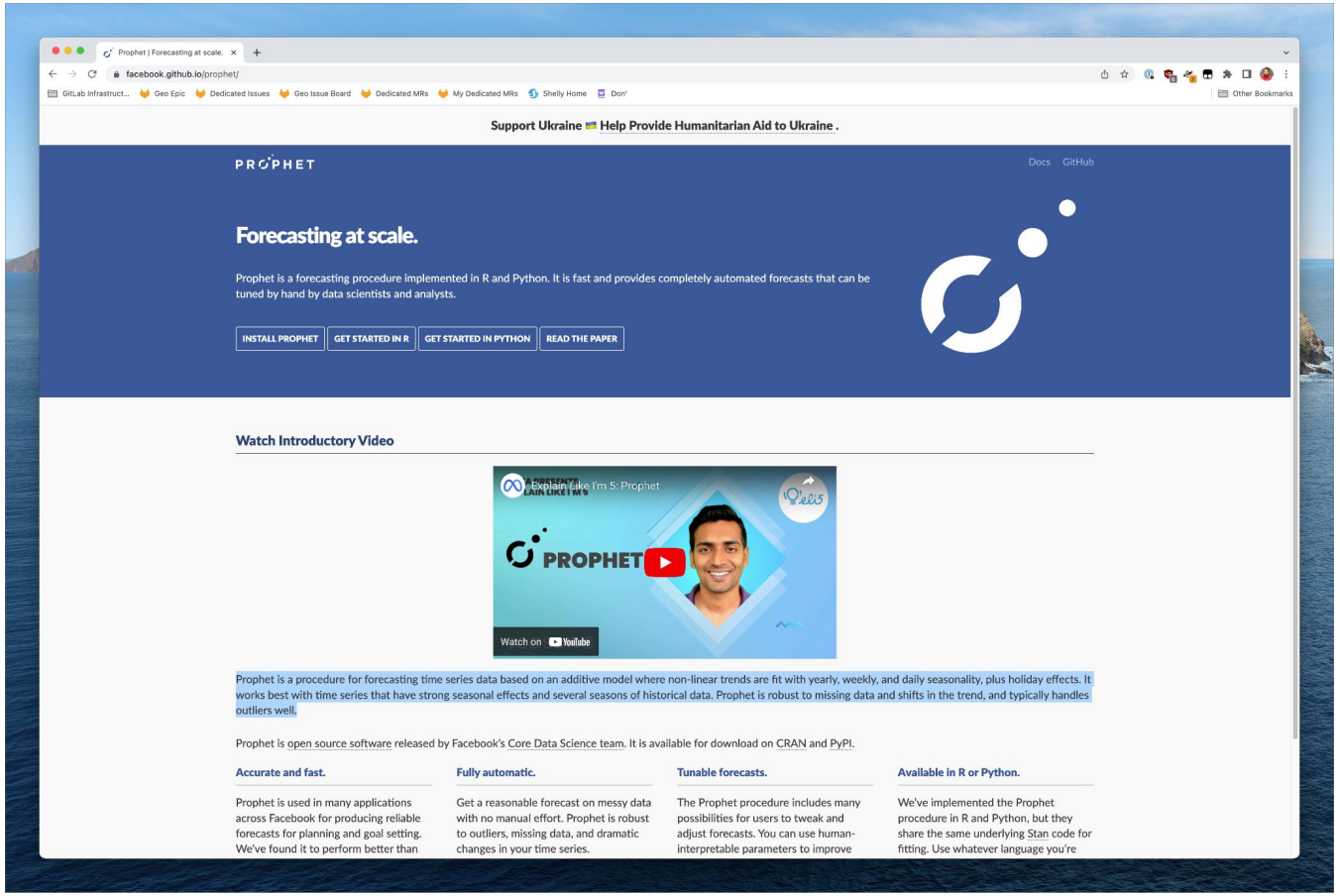
Forecasting



First Attempt at Forecasting: Linear Regression



Second Attempt: Prophet

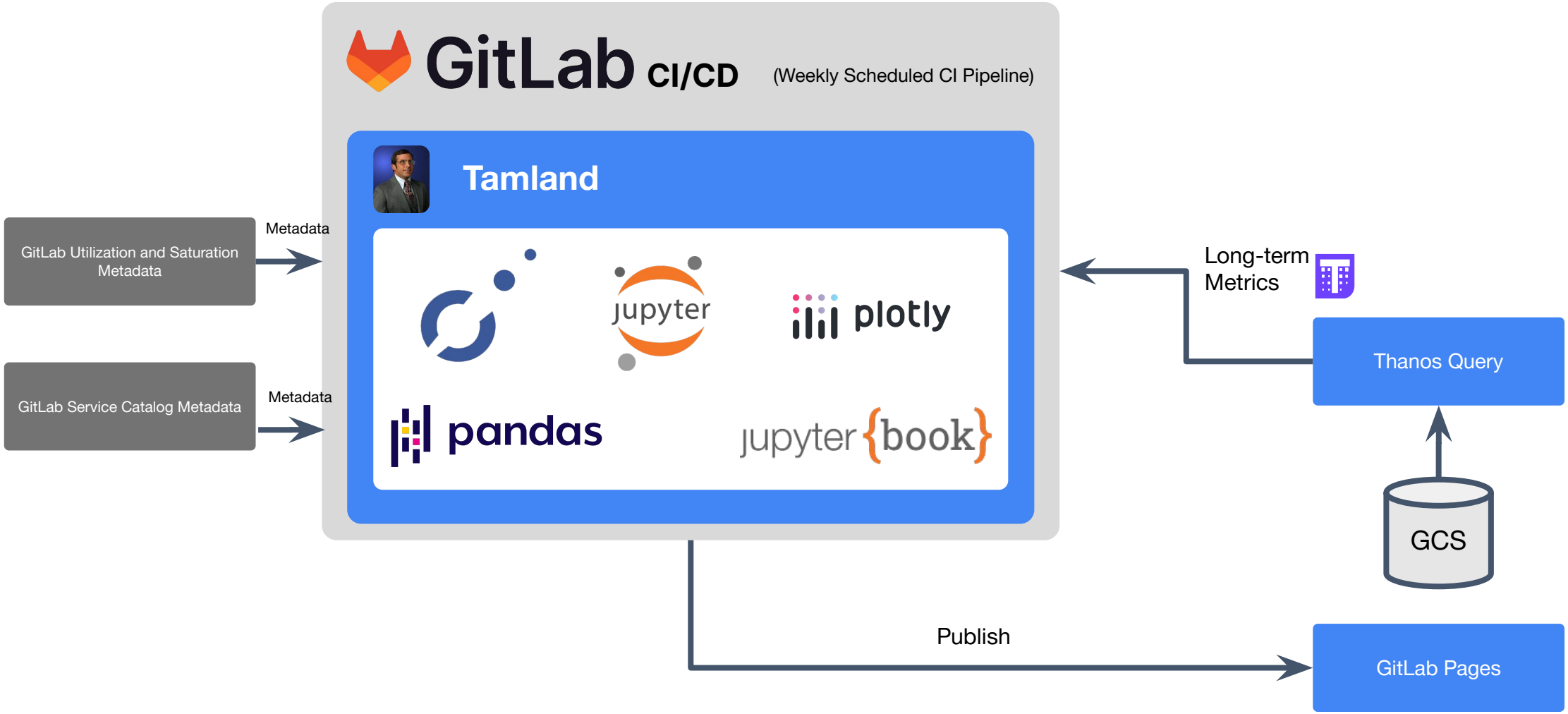


facebook.github.io/prophet

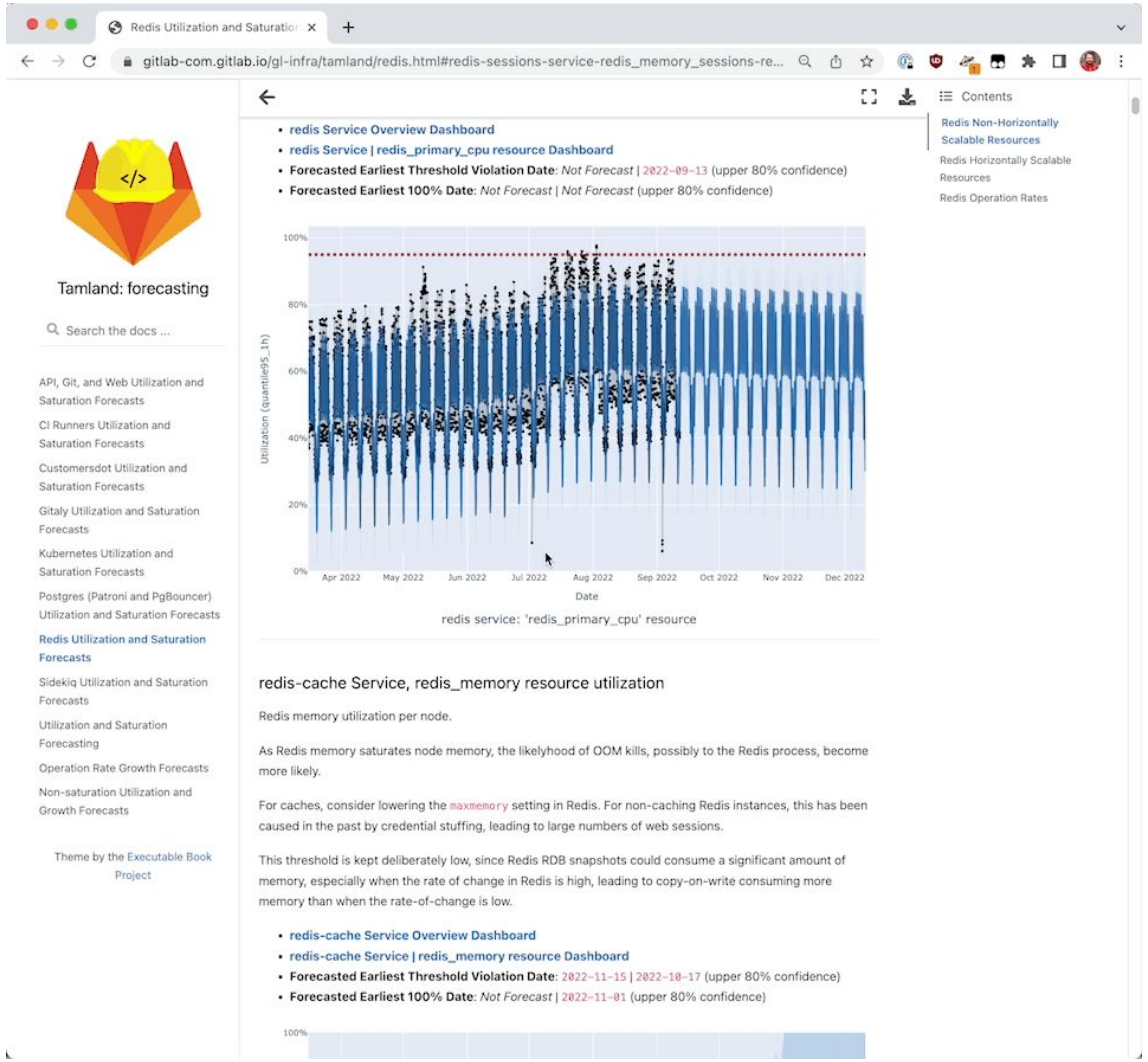
Why Prophet?

- Building forecasting models is a **specialist data-science skill** and is difficult to do.
- **Prophet is designed for ease-of-use**, for people without a statistics or data-science major.
- **Scalable**: no need to tune each forecast independently. Prophet is able to work well, with some generic tuning, to forecast across 400 different utilization series for GitLab.com.
- Works well with data with multiple seasonalities - in our case, our data is very strongly seasonal on **hourly and weekly seasons**.
- **Handles outliers** (eg, during an incident) well.
- **Handles missing data** (eg, during a monitoring outage) well.
- **Automatic changepoint detection**. Can tell when the trend has changed and adjust the forecast.

Thanos Long Term Metrics + Facebook Prophet + Jupyter = Tamland



Tamland Static Site



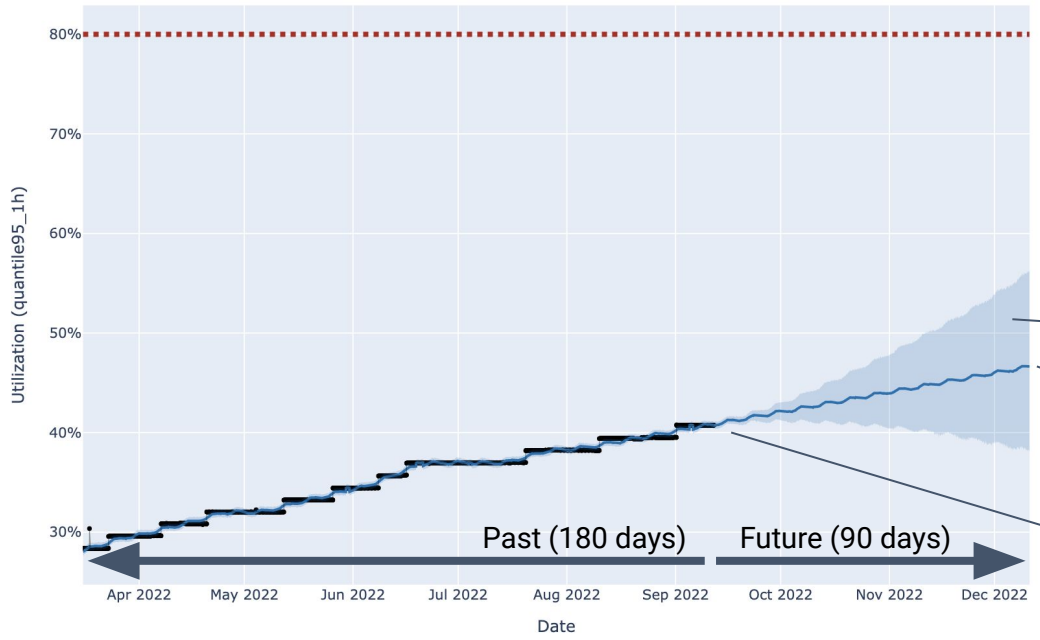
Tamland Forecasting

praefect Service, disk_inodes resource utilization

Disk inode utilization per device per node.

If this is too high, its possible that a directory is filling up with files. Consider logging in an checking temp directories for large numbers of files

- praefect Service Overview Dashboard
- praefect Service | disk_inodes resource Dashboard
- **Forecasted Earliest Threshold Violation Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)
- **Forecasted Earliest 100% Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)



praefect service: 'disk_inodes' resource

Title

Resource Description

Grafana Dashboard Links for Further Analysis

Forecast Violation Dates

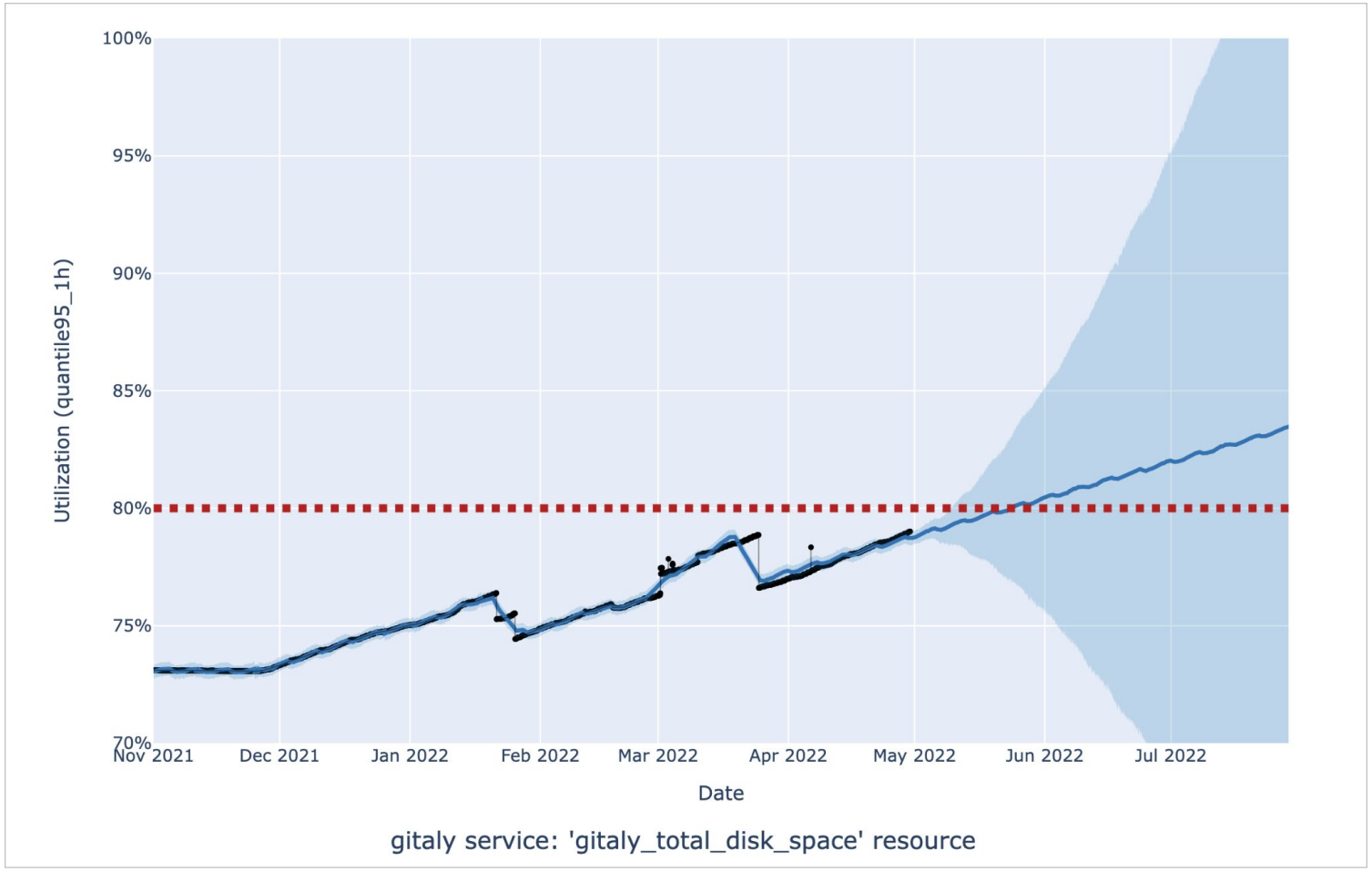
Saturation Alerting Threshold for this Resource

80% Confidence Interval

Median Confidence Line

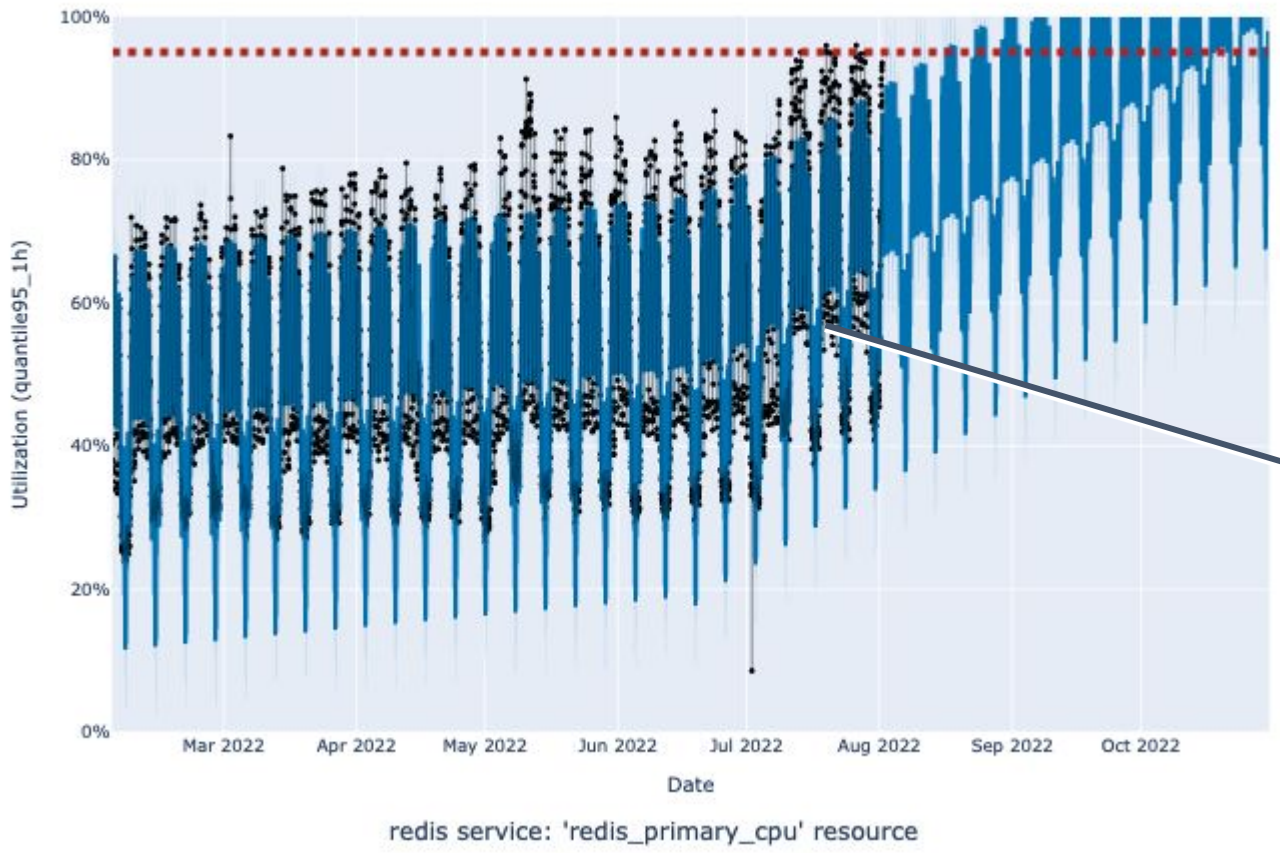
Current Date

Planning the Rollout of Extra Gitaly Nodes for Git Storage



Changepoints

- **Forecasted Earliest Threshold Violation Date**: 2022-08-16 | 2022-08-02 (upper 80% confidence)
- **Forecasted Earliest 100% Date**: 2022-08-30 | 2022-08-09 (upper 80% confidence)



Changepoint

Runaway Session State Bug

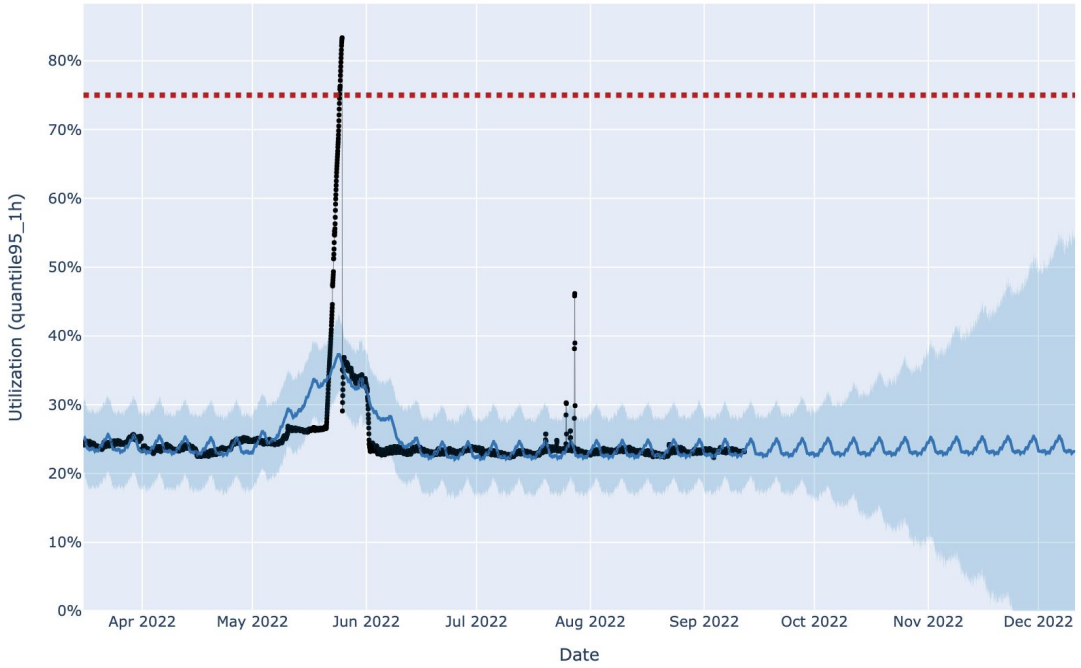
KubeCon CloudNativeCon North America 2022

redis-sessions Service, redis_memory_sessions resource utilization

Redis maxmemory utilization per node

On the sessions Redis we have maxmemory and an eviction policy as a safety-valve, but do not want or expect to reach that limit under normal circumstances; if we start evicting we will start logging out users slightly early (although only the longest inactive sessions), so we want to be alerted some time before that happens.

- **redis-sessions Service Overview Dashboard**
- **redis-sessions Service | redis_memory_sessions resource Dashboard**
- **Forecasted Earliest Threshold Violation Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)
- **Forecasted Earliest 100% Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)

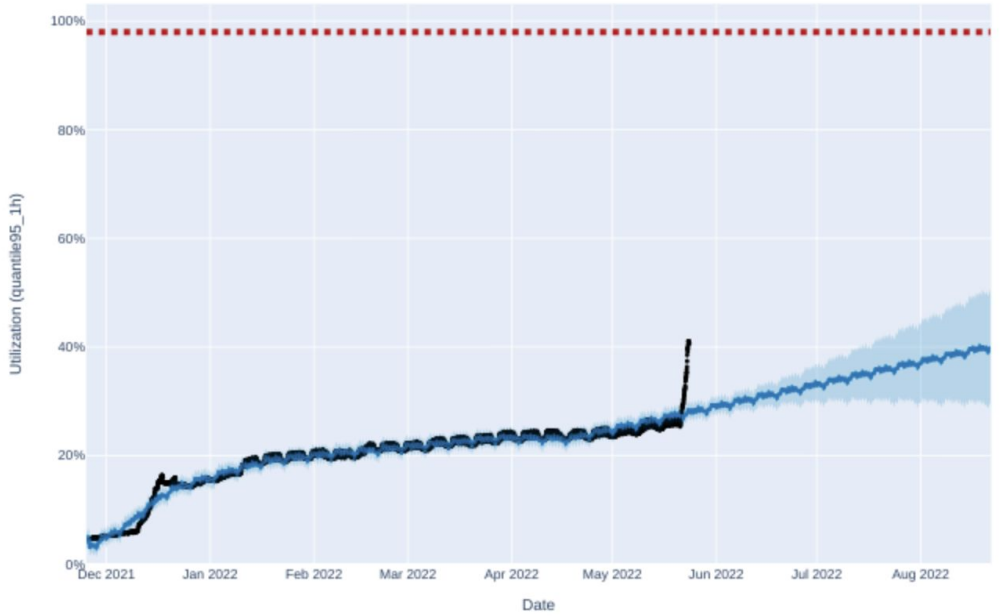


redis-sessions service: 'redis_memory_sessions' resource

Bob Van Landuyt @reprazent · 3 months ago Maintainer

redis-sessions Service, memory resource utilization

- **redis-sessions Service Overview Dashboard**
- **redis-sessions Service | memory resource Dashboard**
- **Forecasted Earliest Threshold Violation Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)
- **Forecasted Earliest 100% Date**: *Not Forecast* | *Not Forecast* (upper 80% confidence)



redis-sessions service: 'memory' resource

This doesn't look like it's close to saturation just yet, though the behaviour is highly uncharacteristic. We should probably figure out what happened here. It looks like something might have changed on 2022-05-22 that made us store more

Though `redis_memory_sessions` looks worse, this takes `maxmemory` into account.

Tamland Today

- **Weekly** forecasting for **363 service/resource combinations.**
- **366 Capacity Planning issues opened**/investigated (of which 262 have been closed).
- **Tamland report generation job takes around ~90 minutes** to run (with caching of metrics) on a dedicated `n1-standard-8` runner.

Future Plans

- **Further decouple** Tamland from GitLab.com Infrastructure, allowing more widespread usage.
- **Improve accuracy**: Allow further customization of forecasting methods and alerting parameters for different resource classes.
- **Better ownership** - allow teams to receive notifications for the services they manage.
- **Experiment with other forecasting libraries**, such as NeuralProphet or LinkedIn's Greykite.

Neural Prophet

In Summary

The GitLab.com capacity planning process:

- **Avoid estimating total system capacity**
- Use a **consistent resource utilization monitoring** strategy across all your services.
- Use a **scalable forecasting library** - Prophet
- **Setup solid engineering workflows** to review and escalate potential issues.

You Stay Classy KubeCon NA!

Tamland Source Code:
https://gitlab.com/gitlab-com/gl-infra/tamland

GitLab Runbooks Saturation Definitions:
https://gitlab.com/gitlab-com/runbooks/-/tree/master/libsonnet/saturation-monitoring

Interesting in this sort of thing? Join us!

https://about.gitlab.com/jobs/all-jobs/#Engineering

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Please scan the QR Code above to
leave feedback on this session