

# Scaling Metrics Maturity in a Cloud Native World



# Contents

03	<b>Introduction</b>
04	<b>Lifecycle of Metrics Growth</b>
05	<b>Organization Maturity Ladder</b>
06	<b>— Stage 1: Foundations</b> <ul style="list-style-type: none"><li>  Scale of Data</li><li>  Breadth of Queries</li><li>  Technical and Operational Considerations</li></ul>
08	<b>— Stage 2: Growth &amp; Evangelism</b> <ul style="list-style-type: none"><li>  Scale of Data</li><li>  Breadth of Queries</li><li>  Technical and Operational Considerations</li></ul>
10	<b>— Stage 3: Complete Organizational Intelligence</b> <ul style="list-style-type: none"><li>  Scale of Data</li><li>  Breadth of Queries</li><li>  Technical and Operational Considerations</li></ul>
13	<b>Introducing Levitate</b> <ul style="list-style-type: none"><li>Tiers</li><li>Policies &amp; Governance</li><li>How is this all woven together?</li></ul>
20	<b>Conclusion</b>





# Introduction

Today, technology-focused companies across all industries — SaaS, Financial Services, Media, and E-commerce — rely on cloud infrastructure and micro-services to deliver value to customers. And, by extension, profits for the business.

The benefits of a performant infrastructure must be very obvious, and so must their degradations. The measurement and attribution of performance in a complex software environment is called Observability.

Observability is built on the storage and retrieval of logs, traces, and metrics. Logs are a mature problem and have significant innovations and available tools. Traces and Time series metrics, on the other hand, are relatively newer domains. We're in the generation of ongoing innovations in tracing and metrics data. And this is why the time is right to introduce a new paradigm to keep up with their increasing variety, volume, and velocity.

In this paper, we introduce [Levitate](#) — a fully managed, OpenMetrics, Prometheus, and OpenTelemetry-compatible time series database for your metrics.



# Lifecycle of Metric Growth

The lifecycle of metric growth is loosely represented as this.

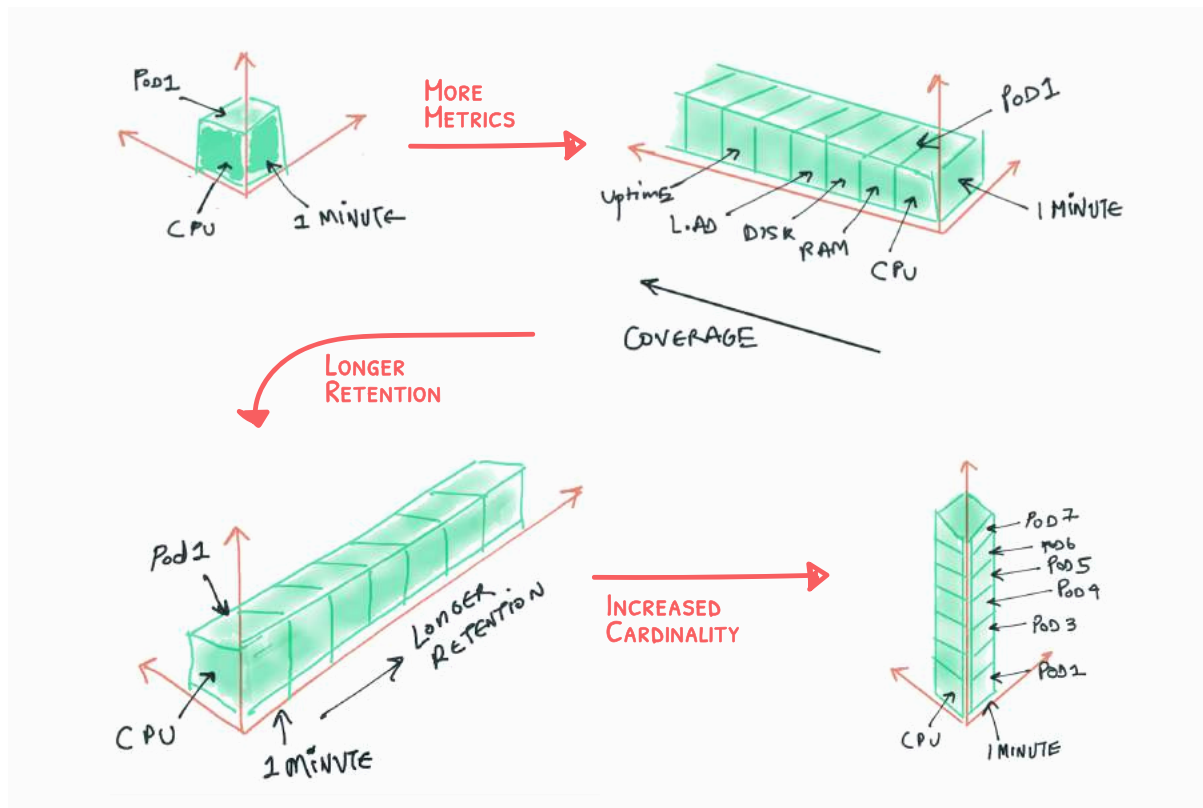


Fig 1: Diagrams showing the beginning of metric growth across the dimensions of [1] retention, [2] unique metric types, and [3] cardinality or instances

Modern Time Series systems don't have to grow along a single axis of Cardinality, Coverage, or Retention alone. Instead, the rate of ingestion and exploration warrants an expansion on all three axes.

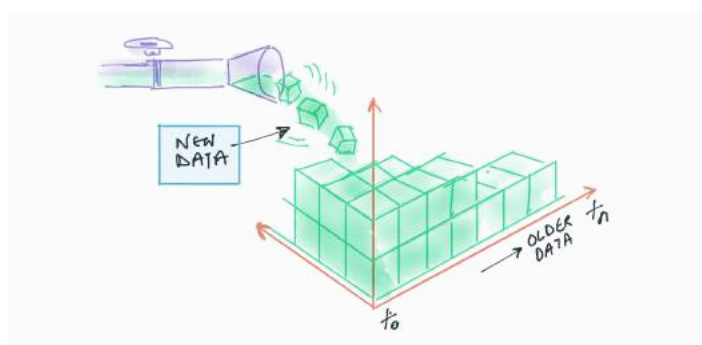


Fig 2: New product surfaces, larger scale, and longer retention cause growth in every dimension.



# Organization Maturity Ladder

The outcomes people want to achieve with Observability differ for different company stages. We spoke with many organizations and could broadly classify the stages of the organization based on one of the three dimensions:

- Scale of Data
- Breadth of Queries
- Technical and Operational Considerations

↓ Stage	No of Engineers	No of Customers	No of Services
Foundations	5	10	15
Operational Readiness	30	50	20
Operational Intelligence	100+	1000+	100+

Table 1: Typical company profiles for different stages



## STAGE 1

# Foundation

As a technology company deploys its first infrastructure components and services, the early customers generate the first time series associated with the product through their interactions and requests.

Stage	No of Engineers	No of Customers	No of Services
Foundations	5	10	15

Table 2: Typical company profile for Stage 1

## Scale of Data

In this initial stage, the scale of data is easily manageable. Architectural complexity and number of instances or tenants are at their lowest, meaning unique time series are low in number.

Data retention needs are also at their lowest at this stage, which is informed by the breadth of questions discussed below.

## Breadth of Queries

Small teams, low complexity, and low business maturity mean that essential incident detection is the focus rather than deeper intelligence over extended periods.

At this stage, teams set up fundamental alerts, such as checks on the availability of critical components and whether ingress volume is below static thresholds. Alerts will be added in response to new incidents rather than proactive analysis.



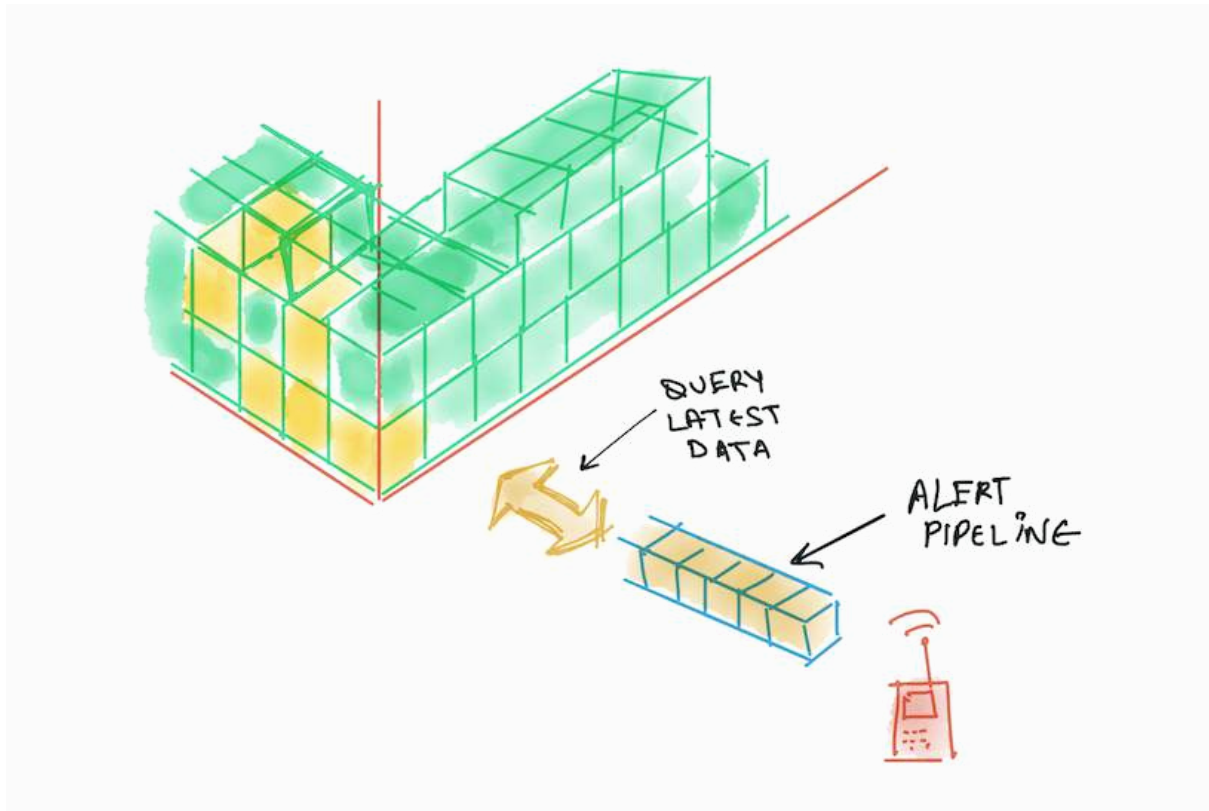


Fig 3: The most recent data is harnessed to build your primary alert pipeline

## — Technical and Operational Considerations

At this stage, a simple Prometheus set-up and a cloud provider's monitoring tool kit (e.g. AWS CloudWatch) are sufficient to meet a team's needs. There's no pressure to do better centralization, operationalize scaling of your resources, or formally **democratize access to time series data**.



## STAGE 2

# Growth & Evangelism

Stage	No of Engineers	No of Customers	No of Services
Operational Readiness	30	50	20

Table 3: Typical company profile for Stage 2

Tolerance towards failures diminishes. There's more focus towards feature velocity. This results in reducing the focus towards stability, performance measurement, data hygiene etc... It's typically where you see an explosion of infra and monitoring bills. At this stage, orgs have a constant conflict between managing stability vs features.

As engineering and product teams ship more features, they need more data to measure the efficacy of their changes. The most common methodologies used at this point are charts and dashboards; which are widely shared across teams.

## Scale of Data

The volume of data is continually increasing and seldom cleaned.

As engineers scale, complexity of software design increases. This is when tribal knowledge gets embedded to pockets of this data. At any given point of time, it's hard to tell what data is important and what's not.

To deliver quality customer experiences, teams aspire to find problems before their customers. As a result, data retention needs are increased, coupled by ad hoc querying in the times of degradations, or constant measurements via dashboarding.

## Breadth of Queries

Reliability is mostly powered by alerts. Leading indicators, like System Saturation, Error Rate, and Queue Lags indicative of system strain, are aggressively used for this alerting.





Operational dashboards become numerous, and the queries get increasingly complex — just another day at work for your time series database.

There are queries for real-time troubleshooting and queries for historical data to analyze customer patterns and behavioral trends. And... there's no way to tell which one is important.

Querying needles in a haystack is expensive, and this slows down the database and the teams, and the tools that depend on it. The end result means friction between business, engineering, and infrastructure teams.

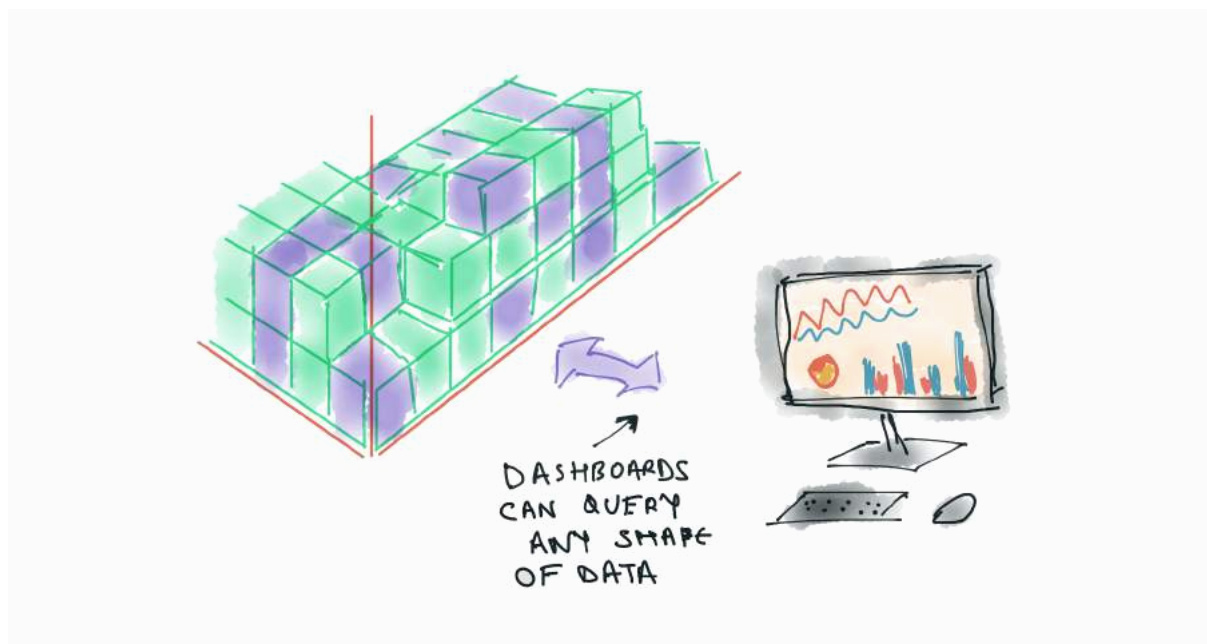


Fig 4: Greater dashboarding and ad hoc query needs can require all parts of your data

## — Technical and Operational Considerations

To keep up with the ever increasing load on your database, you throw more infrastructure and people at the problem. The uptimes, scalability, and the cost of the metrics database becomes a top priority, resulting in dedicated DevOps teams.

While teams chase engineers to send more metrics, set up alerts and dashboards; stakeholders grapple with the increasing cost and effort that goes into managing this data. To keep efforts and cost in check, time must be spent auditing what metrics are used, how much data retention is necessary, and prioritization of dashboards.



## STAGE 3

# Complete Organizational Intelligence

Stage	No of Engineers	No of Customers	No of Services
Operational Intelligence	100+	1000+	100+

Table 4: Typical company profile for Stage 3

As growth continues, metric data in a software company is no longer a regular requirement of just the engineering team. Different business teams — across product, finance, support, and customer success — will need to regularly access this data to properly run a business.

## Scale of Data

Growth explodes across all dimensions as new product lines, multi-region deployments, and customer scale lead to a massive increase in metric data.

## Breadth of Queries

Engineering teams must report on the historical performance of Service Level Objectives to demonstrate improvement and consistency of customer experience. As alerting is adjusted after new incidents, tests are conducted on historical data to understand the impact of the changes. More advanced teams may implement MLOps or AIOps approaches to incident detection and diagnosis, which require more historical data to find meaningful patterns.



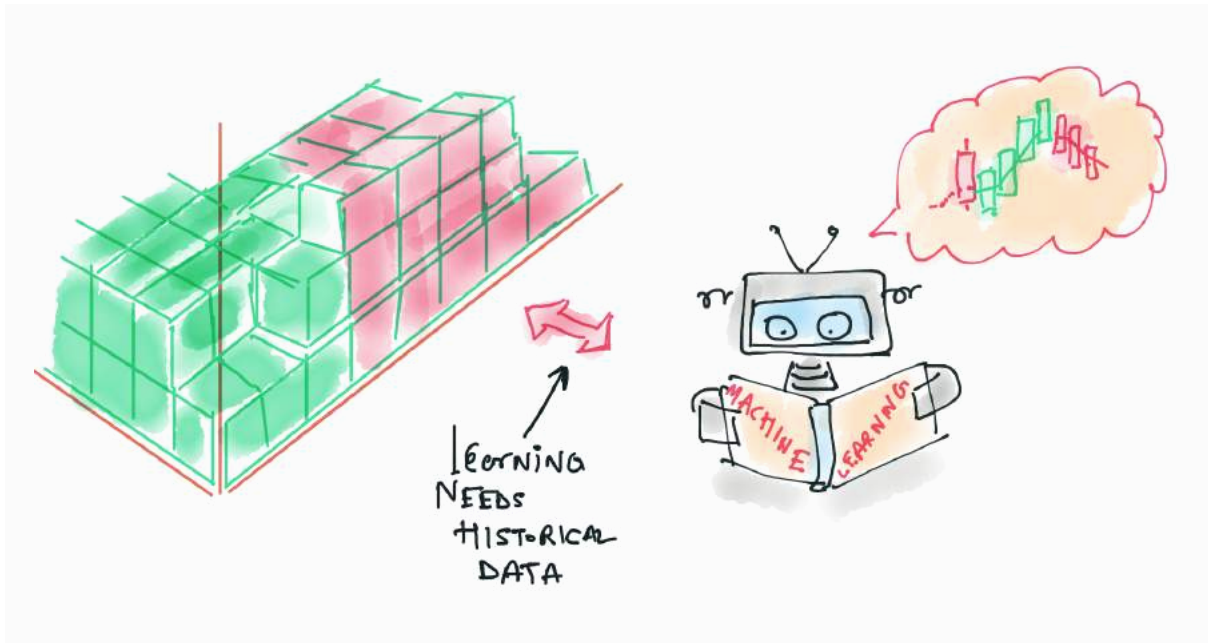


Fig 5: MLOps and AIOps primarily use historical data for hunting patterns.

Other business units will establish regular reporting that depends on time series data from the engineering team. Product teams will want to understand improvements and degradations in the customer experience and the impact on the business. Finance teams must report on customer lifetime value and margins to meet regulatory needs. Customer success and support operations need access to data and alerts to provide world-class customer service, proactive support, and fast debugging.

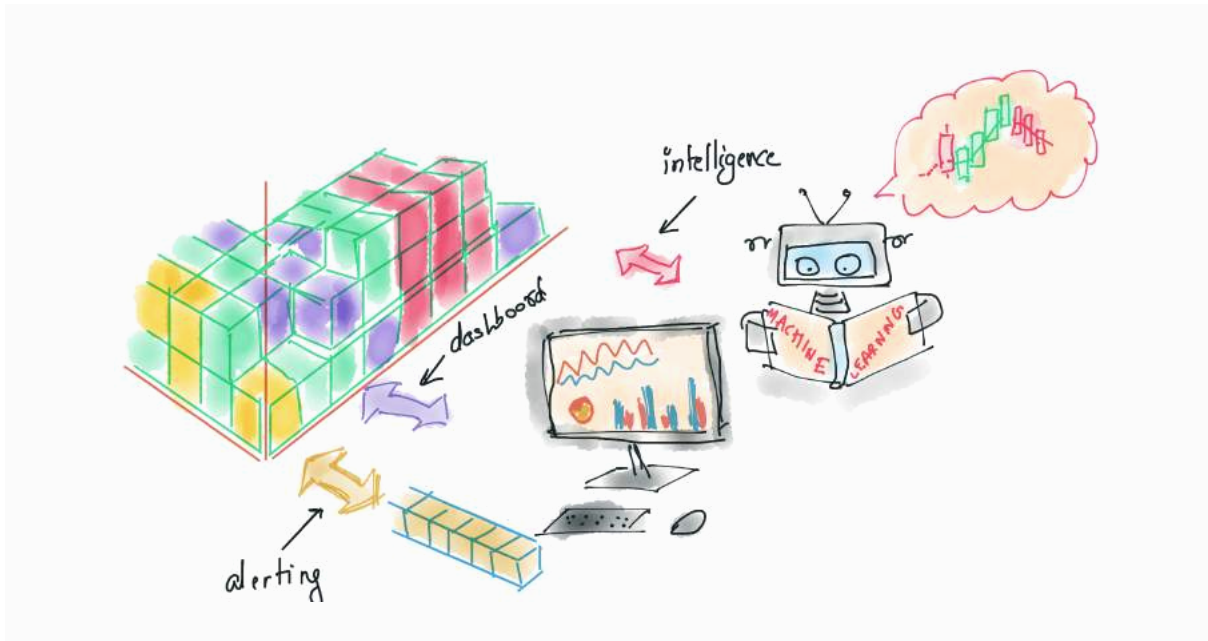


Fig 6: The Time Series system is now the backbone of your Operational Intelligence



# — Technical and Operational Considerations

Enterprises must dedicate full-time engineering resources to manage their time series database at this stage.

Automation to scale with rapid changes in data shape and recovery from downtime must be implemented and maintained.

To support needs across the business, teams create multiple database instances to handle concurrency and implement query sharding to improve performance. They will also implement a solution like Thanos or Cortex to enable long-term storage of metric data. These implementations will cost significant engineering time to create and maintain and make things operationally confusing for developers sending and querying metrics.

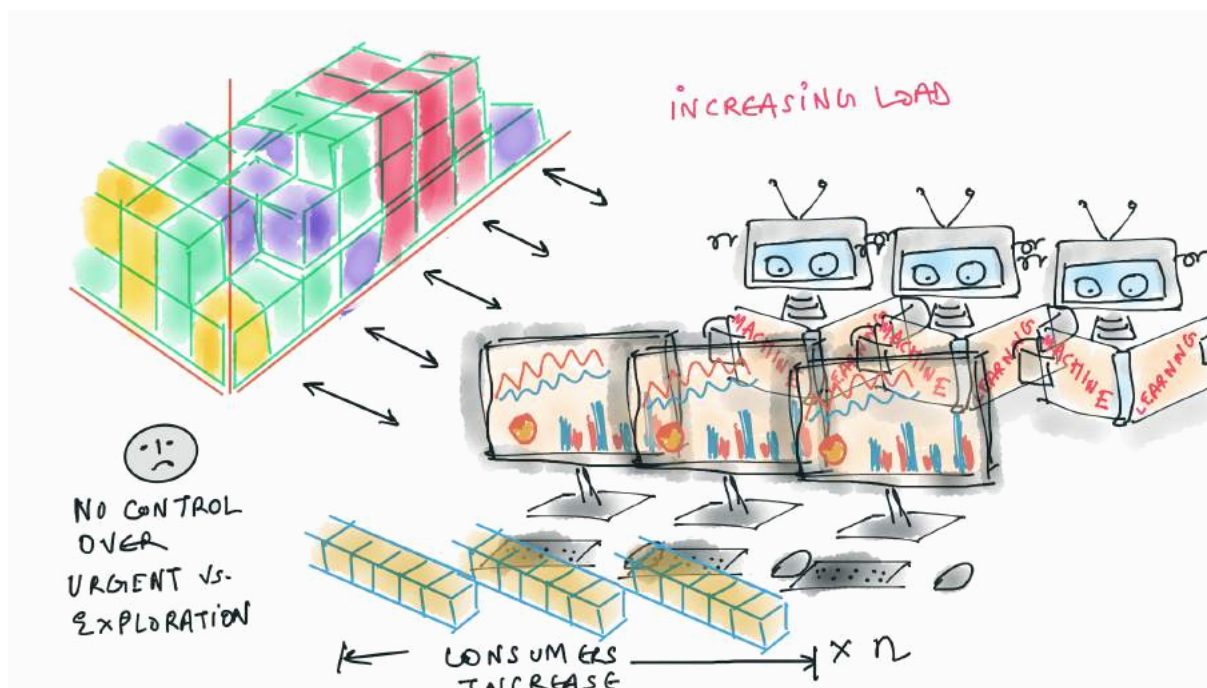


Fig 7: The explosion of query surface area and use cases makes utilizing your database highly competitive



INTRODUCING...

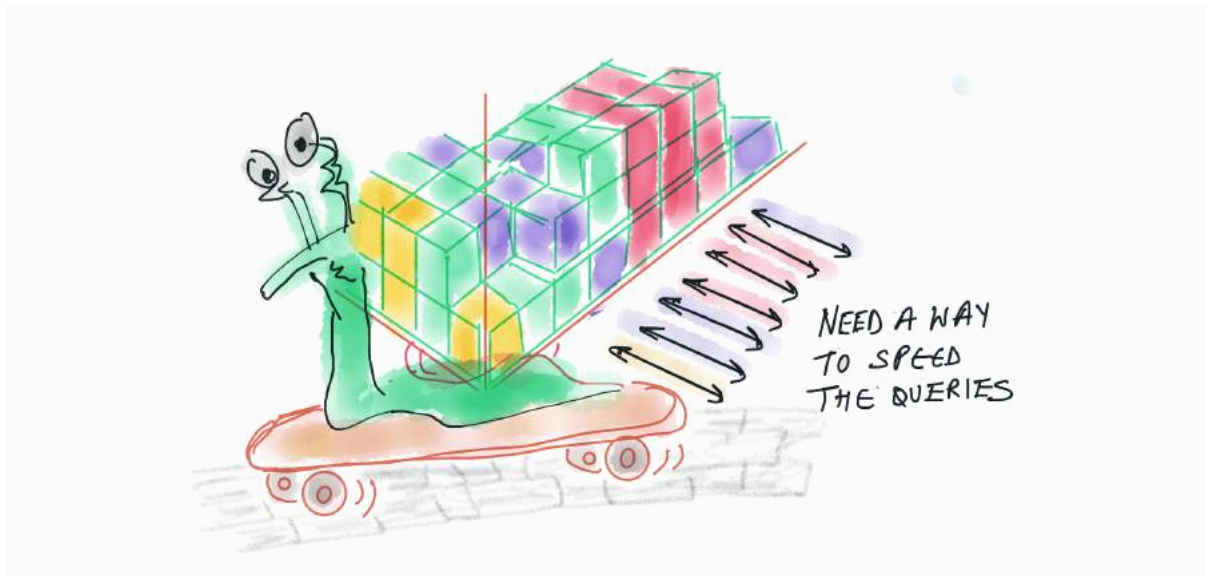


Fig 8: Levitate gives you wheels.

Into this world of scaling time series data needs, we introduce a mission-critical time series database that allows control over queries, and storage to build a cost-effective and toil-free foundation for operational readiness.

Levitate is built for high query performance, high availability, and low latency writing. Levitate automatically scales your data ingestion, query speed, and concurrency needs.

**Levitate's write availability SLA is ~99.9%,  
with no work needed by your team!**

**That means your Prometheus agents can be lightweight transmitters.  
Buh-bye storage concerns, and no data loss!**



Levitate’s standard read availability SLA is 99.5%, giving your team confidence that your monitoring database won’t leave them blind during an incident.

Operation Type	Availability	Durability
Read	99.5%	Replication Factor 2
Write	99.9%	3 Availability Zones

Table 5: Levitate Service Level Agreements

## Tiers

Inspired by principles of Data Warehousing, where Data Tiering is a common phenomenon, Levitate introduces tiers for the Time Series storage.

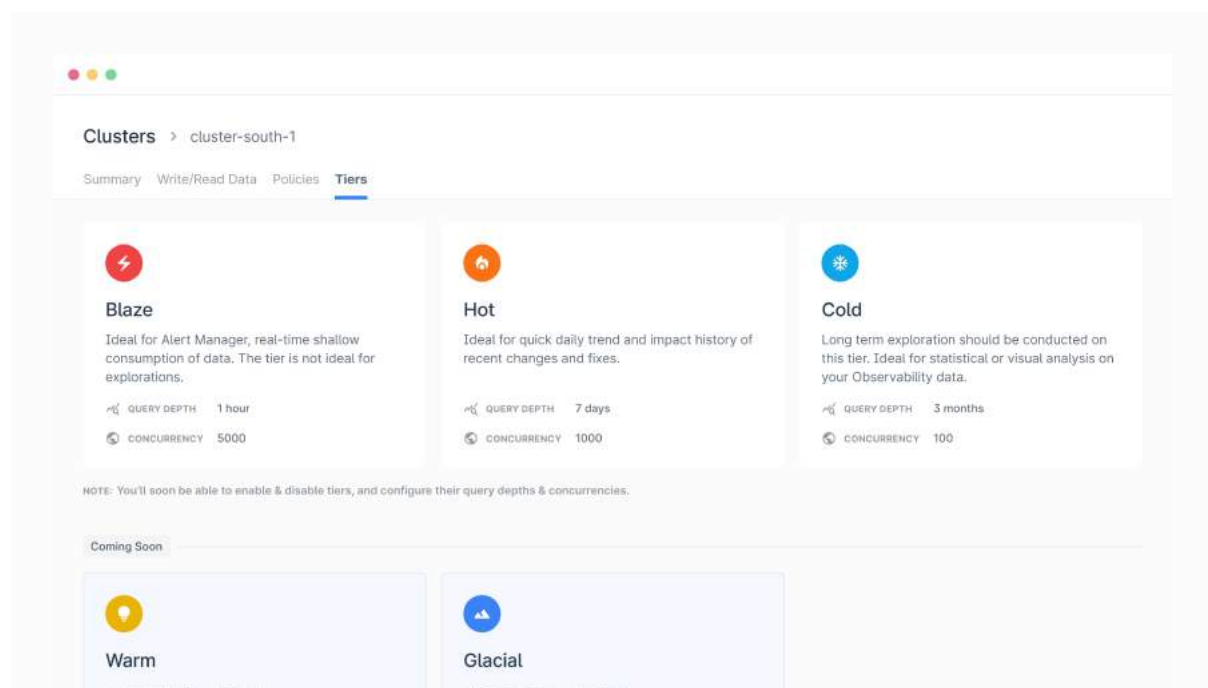
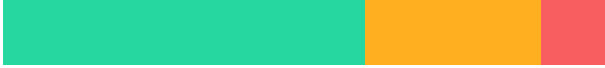


Fig 9: Example of Levitate Tiers

Teams only emit metrics to a single write endpoint from an integration perspective, and data reaches each of the respective tiers. All tiers have all the fresh data, but their data retention varies. Each tier functions as a completely independent time series database but serves different concurrencies, query latency, and depths.







To understand the power of tiers, imagine the current situation where all the metrics, used or unused, are saved in the database. Say there is an auto-refreshing dashboard panel that greedily aggregates 6-months of payment failures saved per transaction and renders it on a histogram chart.

Imagine five such dashboards, refreshing every 15 seconds. This is enough to bring an 8-core, 32GB RAM database to its knees. During this period, the ingestion, as well as critical alerting, gets impacted.

These are real on-the-floor problems with each team having 10+ dashboards per service, and the nuisance compounds when you realize multiple teams are querying for large data sets.

**You don't just pay extra for what you don't use;  
you pay extra because of what you don't use.**

Levitate can segregate data into virtual clusters, or Tiers, with distinct access control parameters like:

- **Retention Limit:** Limit the data that is available in the Tier i.e. x months/days/hours.
- **Concurrency Control:** Limit the active number of queries the Tier can handle simultaneously. This is the most trustworthy indicator of Performance.
- **Range Control:** Limit the number of days of data allowed to be looked up in a single query. It directly impacts the number of data points or series loaded into the memory.

## Policies and Governance

Additionally, Levitate offers powerful features to identify time series your team isn't using and trim data according to policies you create.

Policies are classified as Access Policies and Data Policies.



## Access Policies

Access Policies control how Tiers are engaged based on **Access Methods** .

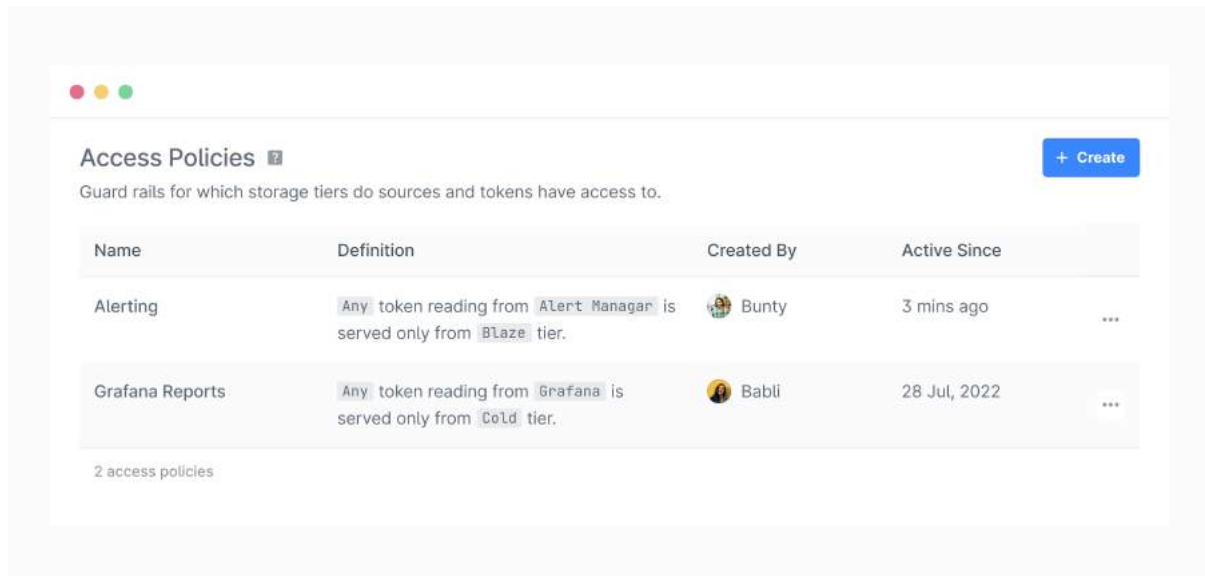


Fig 10: Access Policies

## Data Policies

Data Policies control how data moves in and out of Tiers based on Access **Frequency** and **Time** .

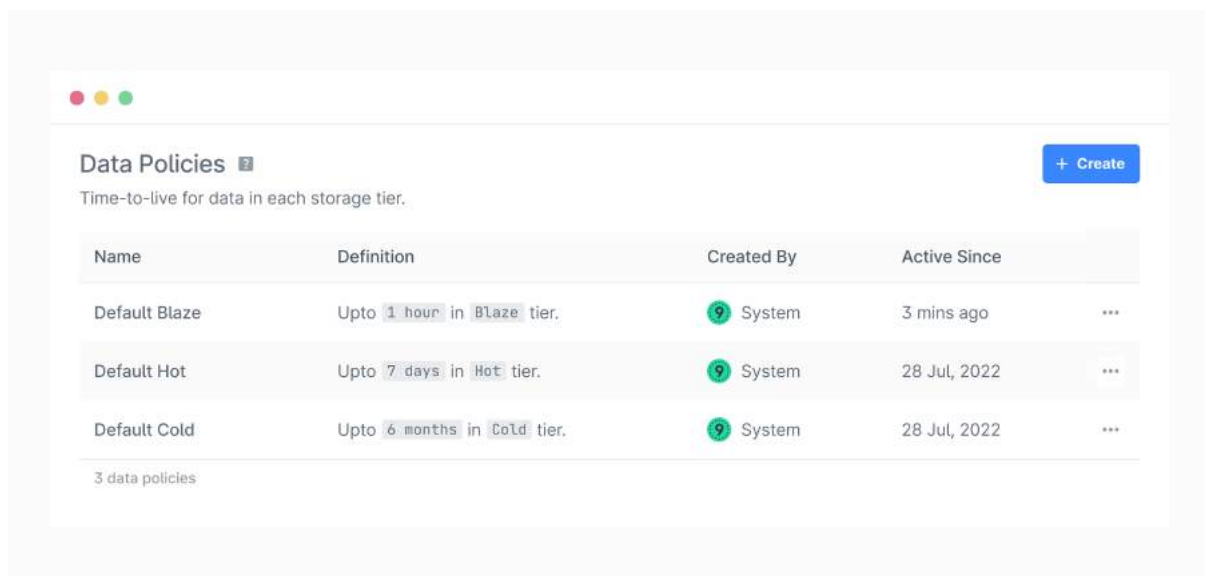


Fig 11: Data Policies





## Policy Samples

Policy	Implications
<b>Any</b> token from <b>Grafana</b> will only read from the <b>Cold</b> Tier.	Ensures that Grafana queries will NEVER interfere with other high-priority traffic reading from the <b>Blaze</b> Tier.
<b>Infra-team</b> token from <b>Grafana</b> will only read from the <b>Blaze</b> Tier.	Ensures that when a Grafana panel uses an <b>Infra</b> data source configured to use an <b>Infra-team</b> token, Levitate will serve all queries from that panel from the <b>Blaze</b> Tier, which is fast but restricted to ONLY the last 1 hour of data.  An attempt by this panel to read more data will only yield 1-hour of data.
<b>Science-Read</b> token from <b>Anywhere</b> will only read from the <b>Cold</b> Tier.	Mandates that whenever the token supplied is <b>Science-Read</b> , no matter where the request originates, it will ALWAYS be served from the <b>Cold</b> Tier.

Table 6: Example policy configurations

## Total Cost of Ownership (TCO) Reduction

There are four essential parts of Levitate (Policy, Query-Routing, Sync, and Consumption Engines) that work together to ensure we bring down your total cost of monitoring.

By tiering your data according to needs, improving dashboard speeds, trimming data explosion, reducing the overheads to manage, and scaling a time series database, Levitate is able to pass down obvious cost advantages to its customers.

Our existing customers have more than halved their storage costs with Levitate, excluding advantages over reduced engineering overheads and their management.



# How is this all woven together?

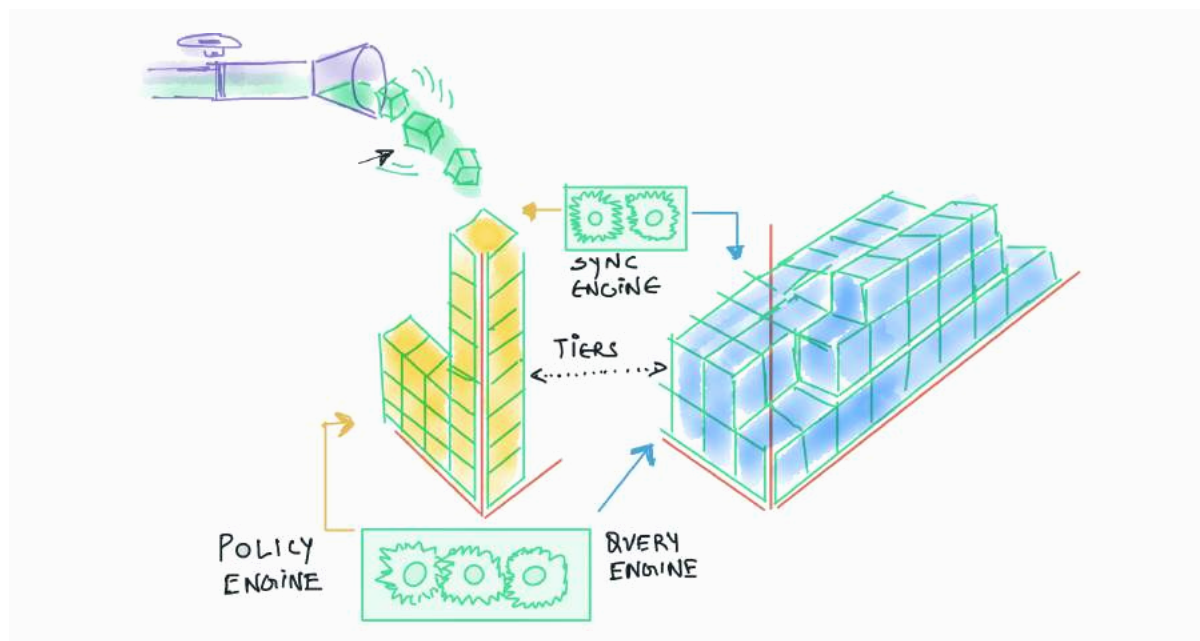


Fig 12: Levitate's routing layer creates a fast lane (yellow) for alerting queries and a slower lane (blue) to handle dashboarding.

There are four essential parts to Levitate.

01

## Policy Engine

Heavily inspired by Sven Marquardt, the Policy Engine provides other engines with the capability to express and evaluate data storage and access rules. These rules can get pretty complex and also powerful.

Any token from Grafana will only read from cold Tier

Anything that isn't used for seven consecutive days should be purged from the Hot tier





02

## Query-Routing Engine

Just before the big sale day, I wish to switch all Grafana dashboards to Cold Tier so that my alerting does NOT stop. At the same time, I would like the Infrastructure team's Grafana dashboards to be fast.

Based on tokens and source, a query routing engine can route the queries to their rightful tiers. The overhead of evaluation is minimal (single-digit ms).

03

## Sync Engine

Very often, the database we read from is also the same one we write to.

In situations like these, a handful of highly greedy queries on the database can also result in an outage and a blackout on ingestion.

Levitate separates the write and the read channels to overcome this, allowing guaranteed ingestion no matter how bad the read loads may be.

04

## Consumption Engine

The smaller the data, the faster your system. As simple as that.

In the most steady state, what you don't use is not retained. What you use more is readily available in fast tiers.

The fundamental pivot is **usage**. The consumption engine keeps track of metrics (later time-series) being consumed and makes them available to all other engines.





# Conclusion

Observability is fast becoming a first-class citizen in an organization, and once you set on it — **data volume**, **variety**, and **velocity** increase rapidly.

**The one thing that differentiates leaders and laggards is that leaders have questions.**

Leaders want answers now, and the tooling has to be able to support that. One can spend engineering on the features that customers love or on supporting infrastructure, but not do both well. But, when there are no good choices available, leaders are forced to rob Paul to pay Pete, slowing down the velocity at which engineering can focus on the core product.

## **The four recurring patterns Observability Leaders hear:**

- Dashboards are slow
- There is a constant need to vertically scale a Time Series Database
- Data is abundant, but it's not being used
- Teams have to be dedicated to managing Time Series lakes

If these recurring points crop up, you know it's time to take critical decisions on your data storage strategies. Talk to us if you're looking for a managed offering that can put your engineering focus back on features, rather than the necessary distractions of scale.

— **We Levitate This.** —





# About Us

Last9 helps businesses gain insights into the Ruben Goldberg of micro-services. With two products, Levitate & Compass, we help understand, track, and improve an organization's system dependencies. By reducing the toil for platform engineering teams, they can now focus on shipping reliable features & products.

Last9 recently raised a Series A round of \$11 million, backed by Sequoia Capital and prominent angel investors.

To know more about us, visit [last9.io](https://last9.io), or reach out to us on Twitter [@last9io](https://twitter.com/last9io).

