Technical reviewer (initial review)	Brian Singer
Editorial reviewer (final approval)	Daniel Ruby
If you are a peer or client reviewer, please click this link upon completing your review to notify your project manager	https://zpr.io/kASLVrLhUiZm

Keyword: service level objectives

URL: https://www.nobl9.com/it-incident-management/service-level-objectives

HTML Title: Service Level Objectives: Modern Best Practices

HTML Description/Blurb for main page introduction: Learn about the importance of service level objectives (SLOs) in monitoring and maintaining system reliability, including related concepts and best practices for effective implementation.

Web page template's pull-down menu title: Service Level Objectives

H1: Service Level Objectives: Modern Best Practices

Service level objectives (SLOs) help to measure system reliability. They define clear targets for system performance and a threshold below which the user experience will be considered degraded and the platform unstable.

This article will cover SLOs in-depth, along with related concepts like SLAs, SLIs, error budgets, burn rates, and time windows. We will also explore advanced topics like composite SLOs, SLO alerting strategies, common challenges, and some recommendations to help you optimize the use of SLOs.

Summary of service level objectives key concepts

The table below summarizes seven essential service level objective concepts this article will explore in detail.

Concept	Description

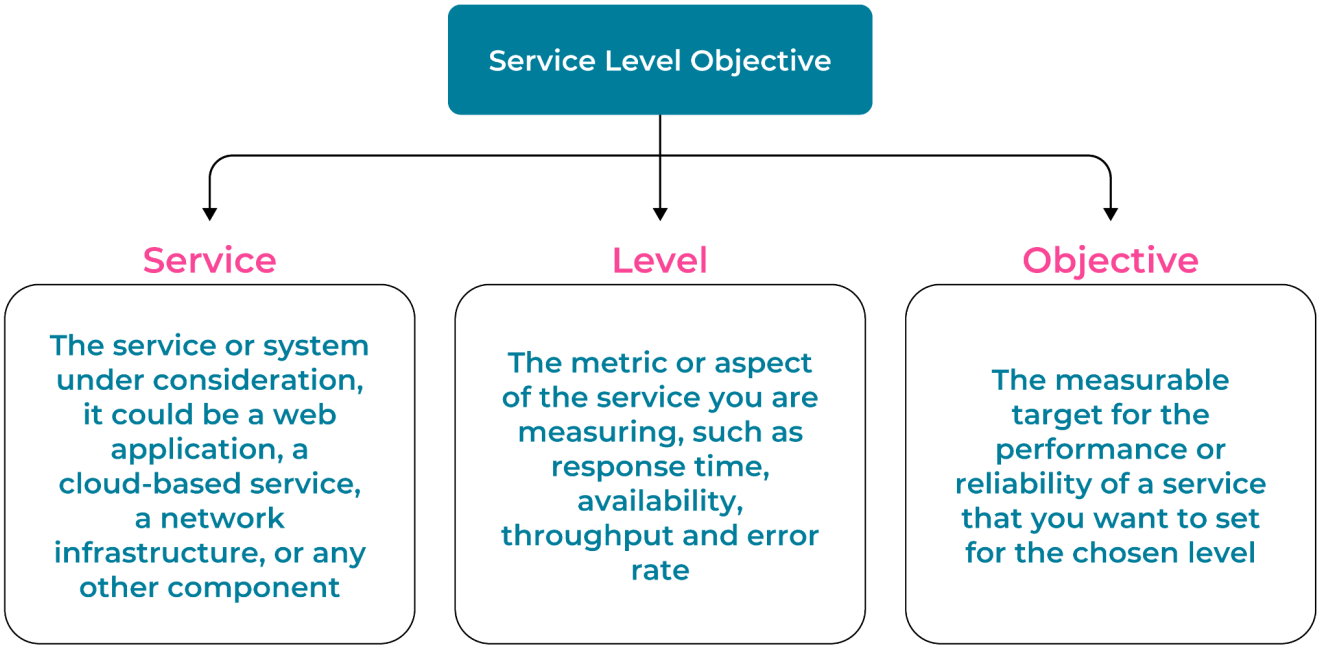
Service	A resource or functionality made available to users, whether humans, applications, or devices, to perform specific tasks or fulfill particular needs.
Service level objective	A measurable target for the performance or reliability of a service, such as uptime or response time that a provider aims to meet over a specified period. It helps teams monitor service health and make informed trade-offs between reliability and innovation.
Error Budget	The allowable amount of unreliability over a specific period of time.
Burn Rate	The rate at which the error budget is consumed. A 1x burn rate means that the error budget will last for the time window and be consumed exactly at the end of it.
SLO time window	The period of time during which the service level performance is measured and evaluated against its defined target.
Composite SLOs	Combines multiple SLOs into one error budget to provide a unified reliability target for services or applications that depend on multiple components.
Setting SLO-based alerts	Alerting strategies on SLOs, including alerting on the amount of error budget remaining or alerting on burn rates.
Alerting window	The period of time during which the metrics are measured, and the alert is evaluated against the defined threshold.
Burn rate alerting	Alerting on the rate at which the error budget is being consumed over a specific time window. The alert will be triggered if the consumption rate of the error budget exceeds a predefined rate.
Multi-Window Multi-Burn Alerting	A recommended alerting method that uses two alerting windows; a short window for alerting on spikes with high burn rates and a long window to alert on slow

	burn rates that will consume the error budget in the long term of the time window.
SLO challenges	Common challenges when setting SLOs include: ● Overly complex SLOs. ● Keeping teams involved. ● Choosing the right metrics. ● Ensuring an accurate measurement.
SLO Best Practices	Recommendations for creating effective SLOs include: ● Selecting metrics that matter to users ● Starting simple ● Setting realistic targets ● Regularly reviewing and improving SLOs based on historical data ● Aligning with different teams ● Selecting suitable time windows. ● Create SLAs based on SLOs.

Service level objective (SLO)

Service level objectives (SLOs) are a set of targets, often expressed in percentages, defined by service providers and internal teams. They help ensure the expected performance and reliability delivered to users of a service. They are crucial in monitoring the service quality you offer and indicating whether you are meeting user expectations or are at risk of breaching agreements with the stakeholders.

For example, a team can define an SLO that specifies that an application must be available 99.5% of the time over one year. Another SLO could state that 97% of web page load time from users should be less than 3 seconds within one month. These measurable targets help align teams on what represents an acceptable performance and provide a benchmark for tracking progress over time.



Service level objectives have three components: Service, Level, and Objective

Service level indicator (SLI)

SLIs are the foundation of service level objectives (SLOs), representing the metrics upon which they are built. Those metrics, such as availability, latency, or error rate, indicate your system's real-time performance.

While SLIs indicate the service's performance, the SLO defines the target for that performance.

To calculate the performance of a service or application at a point in time, you can use this formula:

Good Events / Total Events * 100

In this context, the SLI represents the metrics used in the queries to calculate "good" and "total" events. At the same time, the SLO defines the target of that calculation to determine whether the service meets the required performance.

Key elements to define SLI:

Three elements are important for defining SLIs: the type of service observed, the SLI metric, and the time window.

1. Type of services observed

Common service types include:

- **Request-driven**: Services where users send requests and expect responses, such as web browsers interacting with HTTP services or APIs for mobile apps.
- **Pipeline**: Processes that transform input data into a different output such as converting videos from one format to another or aggregating log files from multiple sources to create reports.
- **Storage**: Services that store data (e.g., files, records, or videos) for later retrieval.

2. SLI metrics

The table below shows the standard SLI metrics for each type of service, with an SLO example for each SLI type

Type of service	SLI Metric	Description	SLO Example
Request-driven	Availability	The percentage of time the service is available without failures.	The website should have at least 99.5% uptime over the year.
	Latency	The time taken by a service or a webpage to respond to a user request.	95% of web page loads should be completed within 2 seconds over a 30-day period
	Error rate	The percentage of requests for a service or website that result in error codes or failures over a period of time	The error rate should be less than 0.1% of all transactions made during a week.
	Throughput	The number of requests that an API or service can handle in a time period, which is often	The website should handle at least 1,000 requests per second on average during

		referred to as RPM (requests per minute)	peak hours over a 7-day period.
Pipeline	Freshness	This metric measures how recently the information accessed by the user has been updated.	95% of data presented in reports should be updated within 1 hour of the latest available information over a 30-day period.
	Correctness	The proportion of records coming into the pipeline that take in data and perform computations on it that result in the correct value being output.	99.8% of records processed by the data pipeline should produce correct outputs over a 7-day period.
	Coverage	It is the proportion of jobs or records that were successfully processed within a defined time window	A batch job should successfully process 99% of the jobs over the week
Storage	Durability	The proportion of stored data that remains readable and retrievable without being corrupted over a given period.	Successful data backup restoration of 99.5% of the stored data over the last month

3. Time windows

Time windows define the period during which the error budget is calculated for a given SLO. There are two types of time windows:

- **Rolling window:** A continuously moving time window that doesn't have fixed start and end dates. For example, for a rolling window of 30 days, we calculate the metric data from the past 30 days. It continuously evaluates the SLO over the last X days, weeks, or months.
  It is recommended that the rolling window be defined in weeks to ensure every

rolling window has the same number of weekends. Generally, a rolling window of 4 weeks would be an excellent general-purpose time window.

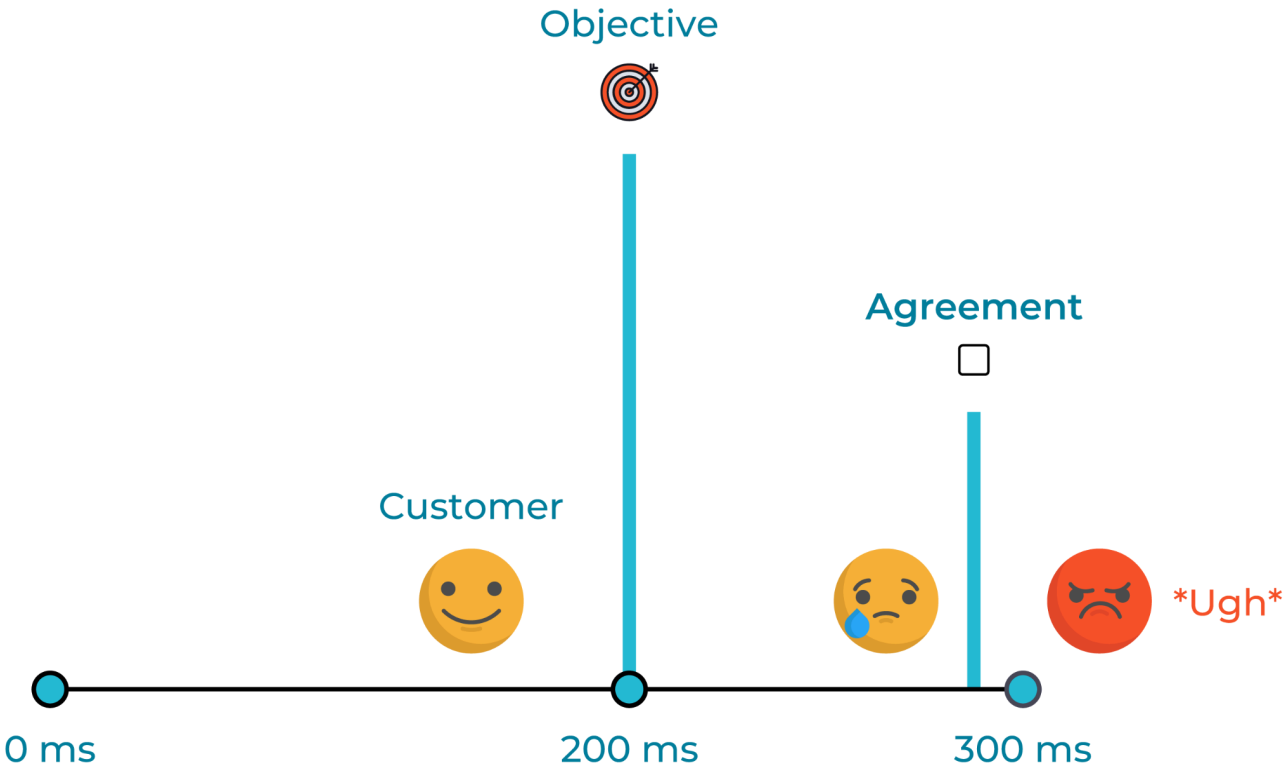
- **Calendar-aligned:** Calendar-aligned windows calculate the SLO in fixed periods, such as a month, a quarter, or a year. The error budget is restored at the beginning of each calendar window. For example, instead of a 30-day rolling window, you can calculate your metric for the first week of the last month.

Service level agreement (SLA)

A service level agreement (SLA) is a written contract between a service provider and a customer, created by the legal and business teams, that defines the expectations and commitments between both sides and the consequences for not meeting the agreed SLA.

While SLAs and SLOs are both reliability targets, they differ in purpose. SLAs typically have a legal remedy associated with them, such as monetary credits. Conversely, SLOs are internal targets that define measurable performance metrics. In another context, SLOs focus on technical performance goals, while SLAs provide the legal framework that formalizes those objectives.

In order to meet an SLA, teams should generally use an SLO target that is higher. For example, an SLA might include a term for 99.0% availability, while the service provider targets an SLO of 99.5% to provide a buffer before breaching the SLA. An easy way to tell the difference between an SLO and an SLA is to ask yourself, “What happens if the objective is unmet?” If there is no legal consequence, you are talking about an SLO.



SLA is a more relaxed target than SLO (Source)

Error budgets

Error budgets are the acceptable amount of non-compliance before the SLO is considered to be breached. Any downtime or errors will consume the error budget, and if the errors continue for a longer time, the error budget will be completely consumed, leading to violating the SLO. On the other hand, if the service operates perfectly, the error budget will remain intact.

Think of an error budget as your monthly household budget. You have money to cover essential expenses like rent, utilities, and groceries. This is similar to the reliability targets defined by your SLOs. If you're staying within your budget, you can take risks, like buying a TV or decorating a room, similar to developers focusing on new features or innovations. However, if unexpected expenses arise and you're close to overspending, you might need to skip the luxuries and focus on the critical needs; similar to the developers, when they are near consuming the error budget, they stop further development and apply code freezes to reduce any further risks.

The error budget is calculated with this formula:

Error Budget % = 100% - SLO %

For example, if your SLO is 99.5%, your error budget is 0.5% of the total duration or 0.5 % of the total expected events over a defined time period.

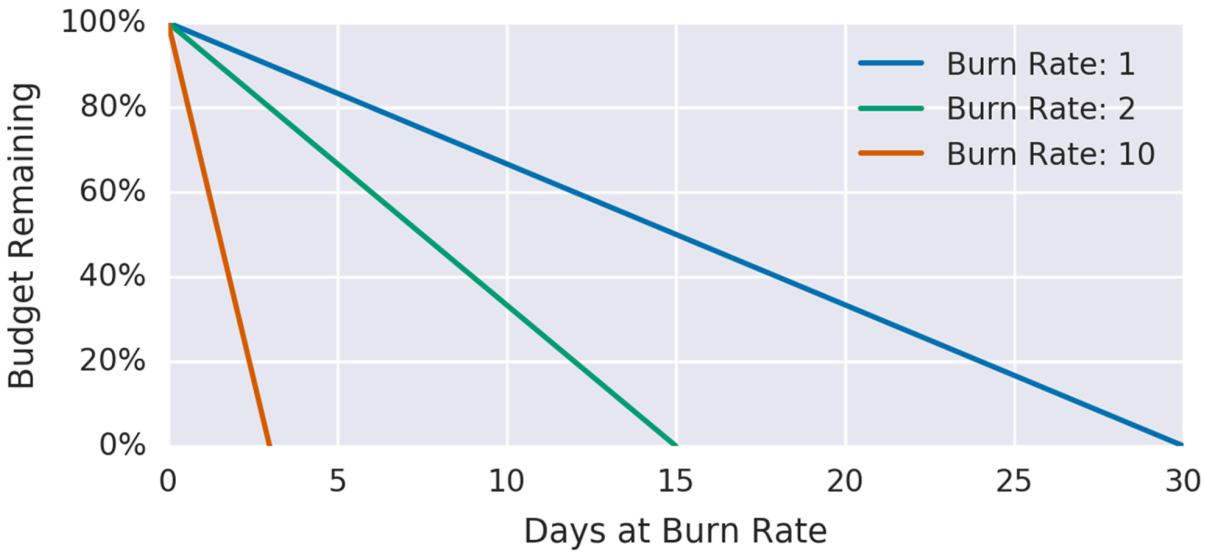
Error Budget = Error Budget % * Total time window or total number of events

The table of nines below might help you understand how an error budget is estimated from the SLO and that every extra nine of reliability you commit to in your SLO means a lower error budget value and, accordingly, a higher risk.

SLO %	**Error budget**		
	Per month	**Per quarter**	**Per year**
90%	3 days	9 days	36.5 days
95%	1.5 days	4.5 days	18.25 days
99%	7.2 hours	21.6 hours	3.65 days
99.5%	3.6 hours	10.8 hours	1.83 days
99.9%	43.2 minutes	2.16 hours	8.76 hours
99.95%	21.6 minutes	1.08 hours	4.38 hours
99.99%	4.32 minutes	12.96 minutes	52.6 minutes
99.999%	26.9 seconds	1.30 minutes	5.26 minutes

Burn rate

The burn rate is the rate at which the error budget is consumed. A 1x burn rate means that the error budget will last for the time window and be consumed at the end of it. A burn rate higher than 1x indicates that the error budget is burnt too quickly, which would result in breaching the SLO.



A burn rate higher than 1x will consume the error budget before the time window ends. (Source)

To calculate the burn rate, you need to be able to calculate the SLI's error rate during a specific time window.

Error rate = Number of Failed Events / Total Number of Events * 100

Imagine you have a web application with 50 failed login attempts from a total of 2000 attempts during the last week.

Error rate = 50 / 2000 * 100 = 2.5%

Once you have calculated the error rate, you will be able to calculate the burn rate of your SLO with this formula:

Burn rate = Error rate in a time window / (100 – SLO%)

For the previous example, if the availability SLO of the web application is 95%:

Burn rate = 2.5 / (100 – 95) = 0.5 %

Composite SLO

In complex applications, several components are needed to deliver the final service for the user. While each component may have its own Service Level Objective (SLO), determining the overall reliability of the application requires more than just looking at those individual SLOs.

For example, imagine an application composed of three microservices: a database, an API, and a front-end service. Each of these has its own SLO, and the system will only operate if the three microservices are available. How can we calculate the SLO of the application itself?

This is where a composite SLO comes in. A composite SLO combines multiple SLO targets into a single SLO that represents your service's overall reliability.

Weighting in composite SLOs

Earlier, we introduced the concept of composite SLOs, which combine multiple SLOs into a single target. But what happens when an application consists of several components with different criticality levels, each having a unique SLO target? How can the composite target be evaluated to account for these variances?

Here comes the importance of weighting in composite SLOs. as it helps to ensure that the critical components in the application have more influence on the target than the other services by assigning a weight to each component SLO.

Use case 1

Let’s consider an application with four services, A, B, C, and D, with SLOs 99%, 98%, 95%, and 97 $, respectively. Here, we assign equal weights with a weight of 1 for each service.

Normalized weight = Weight of service / Sum of all weights * 100 = 1 / 4 * 100 = 25 %

	SLO A	SLO B	SLO C	SLO D
Target	99%	98%	95%	97%
Weights	1	1	1	1

Normalized Weights	25%	25%	25%	25%

In this case, the composite SLO can be set as follows:

Composite SLO = (99% * 25% + 98% * 25% + 95% * 25% + 97% * 25%) * 100 = (0.2475 + 0.245 + 0.2375+ 0.2425) * 100 = 97.52%

Use case 2

For the same previous example, we will assign different weights for services A, B, C, and D as 8,6,4 and 2, respectively.

	SLO A	SLO B	SLO C	SLO D
Target	99%	98%	95%	97%
Weights	8	6	4	2
Normalized Weights	40%	30%	20%	10%

In this case, the composite SLO can be set as follows:

Composite SLO = (99% * 40% + 98% * 30% + 95% * 20% + 97% * 10%) * 100 = (0.396 + 0.294 + 0.19+ 0.097) * 100 = 97.7%

How to assign weights

There's no universal rule about weighting individual SLOs; the best approach will depend on your system's context and specific requirements. Here are some bases on which you can assign the weight:

Based on SLO Type

- Availability-Critical SLOs:
  In most cases, service availability is more important than service response time, as a slow but functional service is better than a completely unavailable service.

In such cases, it is recommended to assign a higher weight to the availability SLOs than the latency SLOs.

- Latency-Critical SLOs:
  In some specific services, such as online meetings and video conference tools, the response time will be as important as the service's availability. In such cases, it is recommended that equal weights be assigned for availability and latency SLOs.
- Based on business requirements:
  When creating a composite SLO for a service or application that includes a multi-step user journey, it's recommended to assign weights based on the relative business importance of each step. The steps that have more influence on the business should have higher weights.
  For example, an online grocery application includes user journey steps like browsing categories, order placement, payment checkout, and feedback giving. For example, the ordering and payment steps would be more critical than the others, so they will have higher weights assigned to them.

Alerting on SLOs

An alert on SLO measures and notifies how much deviation from the SLO has occurred, ensuring that notifications are triggered well before the error budget is exhausted. This helps teams take preventive actions rather than reacting after the SLO has been violated.

According to Google's SRE books, there are some key factors to evaluate an SLO alerting system:

- **Precision**: The proportion of failure events detected. A higher rate indicates that every alert corresponds to an event.
- **Recall**: The percentage of failure events were detected. A higher percentage ensures failure events are not missed.
- **Detection time**: How quickly an alert was triggered after an issue starts. Short detection times minimize the consumption of the error budget and enable faster response.
- **Reset time**: How quickly an alert triggers if there's an issue and how quickly an alert clears after the issue is resolved, which helps the operations team focus on new incidents.

Alerting on the remaining error budget

Error budget-based alerts track the amount of the error budget consumed and the amount remaining. This method helps clarify how much room remains before an SLO breach.

There are two methods to alert on the remaining error budget:

- **Alert on the remaining percentage**: For example, an alert is triggered when the service has only 20% of its error budget left.
- **Alert on the remaining duration**: An alert will trigger when only 90 minutes of allowable downtime remain in the SLO window.

To calculate the error budget remaining, you must first calculate the error budget consumed:

Consumed Error Budget = Number of failed events / Number of the allowed to-fail events (Error budget) * 100

Remaining Error Budget = 100 % - Consumed Error Budget

Let's think about an online store that promises an SLO of 99.5% availability over a 30-day time window. From the historical data, we can expect this online store to handle 100,000 order requests over the 30-day period, which translated into an error budget of 500 orders. During the month, 50 order requests failed to process.

Error Budget Consumed = 50 / 500 * 100 = 10 %

Remaining Error Budget = 100 % - 10 % = 90 %

Alerting on the burn rate

Alerting on burn rates helps to notify teams about failure incidents that cause an error budget to be consumed before the end of the SLO time window.
Some may think they can ignore alerting and check the burn rate from time to time, but that's not right; even the low and steady burn rates, which might go unnoticed, can significantly consume the error budget in the long term of the time window.

To calculate the burn rate, we use this formula:

Burn rate = (Budget consumed × Compliance period) / Alerting window

For example, if a team decides that consuming 10% of the error budget (20 out of 200 allowed failures) in one hour is critical and should be alerted on, they can calculate the burn rate for a 30-day compliance period as follows:

Burn rate = 10% × 720 hrs / 1 hr = 72

The challenge here would be:
This burn rate alert will never alert if the error rate is lower than the burn rate of 72. A burn rate of 25 will still consume all the budget in 8 hours.

Time to consume the budget = Error Budget / Burn rate per hour = 200 / 25 = 8 hours

Multiple burn rate alerts

Multiple burn rate alerts ensure that the low and steady burn rates that will consume the error budget in the long term are detected and notified.

For example, we can alert in these cases:

- If the burn rate over the last 1 hour consumes 2% of the budget
- if the burn rate for the last 6 hours consumes 5% of the budget
- if the burn rate for the last 3 days consumes 10% of the budget

The challenge would be:
The long reset time for the alert. For example, in the last alert, you would wait 3 days to know that 10% of your error budget had already been consumed. After the team applied their code changes, you would wait another 3 days to evaluate the result. It will also require some mechanism of alert suppression so as to avoid alerts firing multiple times for the same events.

Best practice: Multi-window, multiple-burn rate alerts

This is the recommended alerting method in which an alert is configured on two different alerting windows: a short one and a long one; the shorter time window. This type of alert is for sure more complex than the previous methods. Still, it ensures that the internal team is notified in both cases when there's a critical issue with a high burn

rate and also notified when there is a steady and slow burn rate that might exhaust the burn rate in the long term.

For example, we can alert in these cases:

- If the burn rate over the last 1 hour consumes 2% of the budget and the burn rate over the last 5 minutes consumes 2% of the budget
- if the burn rate for the last 6 hours consumes 5% of the budget and the burn rate for the last 30 minutes consumes 5% of the budget
- if the burn rate for the last 3 days consumes 10% of the budget and the burn rate for the last 6 hours consumes 10% of the budget

Alert fatigue

Alert fatigue happens when an excessive number of alerts, often including false positives, are triggered, overwhelming the operations team. This leads to critical alerts being missed or delayed. False positives are alerts that were flagged to be exhausting the SLO but, after further analysis, were found not to be.

Symptom-based alert over a caused-based alert

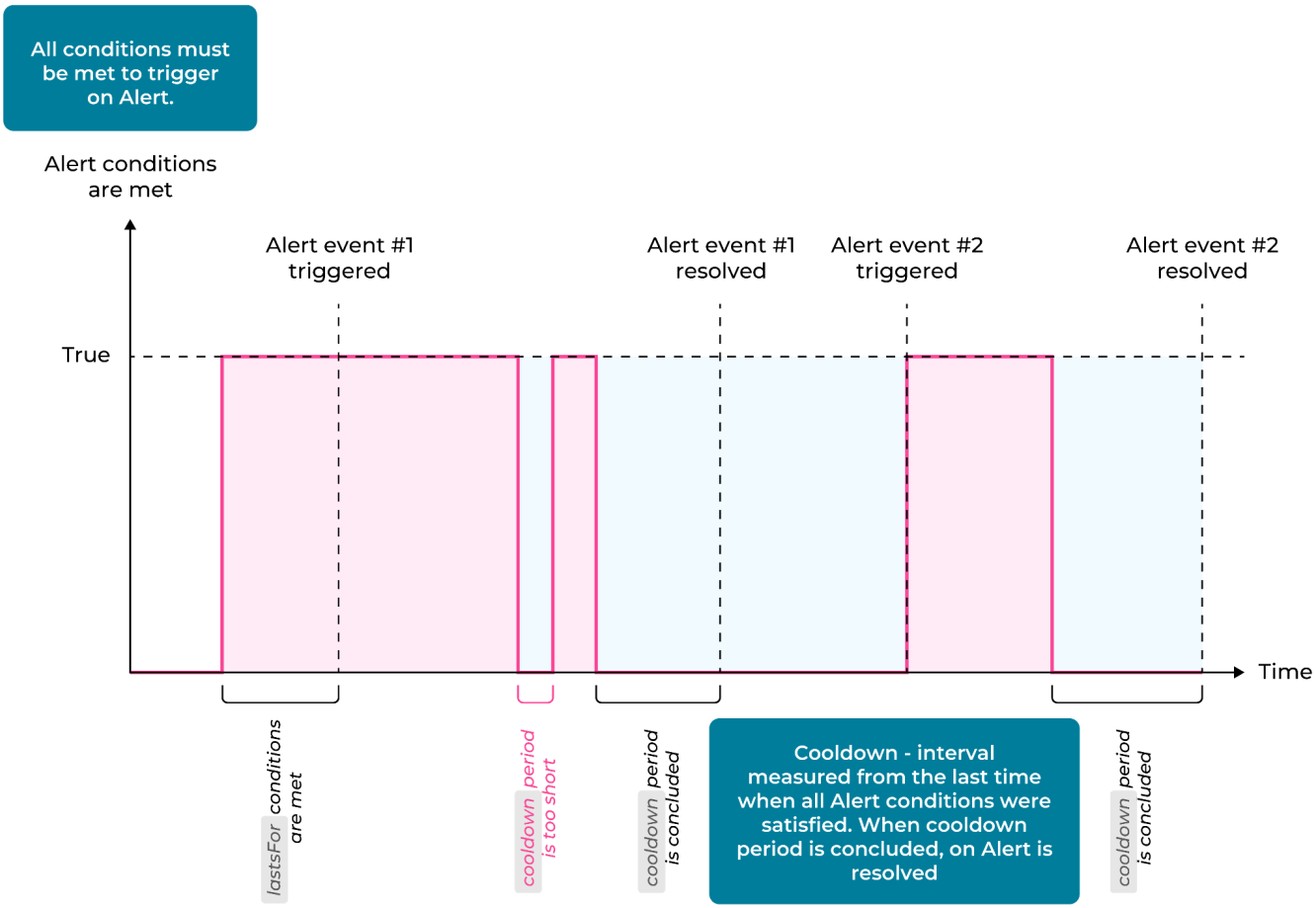
In addition to multi-windows and multi-burn alerts, there are two common types of alerts teams should consider: cause-based and symptom-based.

- **Cause-based alerts:** Focus on specific technical measurements. For example, you can alert your on-call team when the CPU usage over a database exceeds 90%, but that high CPU usage may not be meaningful to the user and might not affect your SLO budget at all. This will cause alert fatigue.
- **Symptom-based alerts:** Focus on specific symptoms, like latency. For example, you can alert your on-call team when 98% of the database queries take more than 500ms over the last 5 minutes. This is an incident that requires attention as it will breach the SLO.

Alert cooldown

Another way to overcome alert fatigue is the cooldown feature. Nobl9 provides this out-of-the-box feature to users to ensure that alerts are not repeatedly triggered for the same issue in a short period of time. Applying a cooldown period allows your team time to resolve the incident without being overwhelmed by duplicate alerts.

It is a parameter you define while creating an alert policy that establishes the amount of time that must pass to change the status of an alert from the okay status to an alert being triggered or from an active alert to the resolved status.



A lifecycle of an alert in Noble9 configured with a cooldown time. (Source)

Seven proven SLO best practices

The six service-level objective best practices below can help teams optimize their monitoring and alerting strategy.

1. Focus on the user experience

SLOs should focus on users' expectations and what matters to them. Most users care about the application or service functioning as expected; they won't care if your CPU

utilization is over 70%. So, always resist the temptation to add many SLOs for metrics that don't relate.

- **Define the user journey**: define the critical services of your application or website that the users interact with, such as the front-end service or the cart service
- **Base SLIs on user journeys**: Pick up the SLI metrics related to the journey you defined. For example, select metrics like uptime, latency, or error rate.
- **Align the SLO target with the customer's expectations:** set targets based on the user's satisfaction levels.

2. Use meaningful and measurable metrics

Creating an effective SLO requires meaningful and measurable SLI metrics that accurately reflect service performance and the user experience. Here are the best practices to follow when defining these metrics:

- **Define a clear SLI**: SLI metrics shouldn't use generic terms like "I want a reliable or fast service." Instead, teams should use specific metrics, such as a service with 99.5% availability up time or a service with no more than 3 seconds response time.
- **Keep it simple:** Limit the number of SLIs to the top-priority ones that impact the user's journey most.
- **Use a monitoring tool**: Accurate measurement and collection of data points requires a monitoring and logging system.

3. Set realistic targets

It’s essential to be realistic when setting an SLO to avoid setting expectations that your system can’t meet. Setting an achievable target ensures customer satisfaction and prevents SLO breaching. Here are three tactics to help teams get it right:

- **Avoid aggressive targets**: setting a 100% SLO for uptime or performance is unrealistic and leads to burnout and SLO violation. Instead, set a target that you can consistently achieve.
- **Take into account external dependencies**: If your service or application depends on a third-party service, this may also affect your SLO, so always leave room in your SLO for incidents you have no hand in.

4. Involve technical teams and stakeholders

Setting effective SLOs requires communication across different internal teams and involving the stakeholders to ensure that the expectations are achievable and satisfying for everyone.

- **Collaborate with cross-functional teams**: Engage developers, operations, product managers, and business stakeholders to ensure the expectations are achievable and satisfying for everyone.
- **Get mutual agreement and collect feedback**: Involve all the internal teams in the SLO setting process and collect feedback after that process. Also, ensure all agree on the expected targets to ensure alignment and avoid future conflicts.

5. Choosing the convenient time window

Choosing the right time window requires context and can significantly affect how service-level objectives influence business outcomes. Here is a breakdown of the benefits of short vs. long time windows:

- **Short windows allow teams to make decisions quickly.** If an SLO was violated in the previous week, immediate action, such as bug fixes, can help prevent further violations in the following weeks. Use short windows if you have an application that requires tight feedback loops. For example, in an e-commerce platform, daily SLOs should monitor operations to track critical metrics such as website uptime, page load times, and order processing success rate.
- **Long windows are useful for strategic decisions.** For example, imagine you are deciding whether to migrate your infrastructure to a new cloud provider. It must be a well-informed decision that requires a large amount of data and a long time window to evaluate effectively.

6. Create SLAs based on SLOs

It's important not to get mistaken between Service Level Objectives (SLOs) and Service Level Agreements (SLAs). A good SLA relies on a well-defined and accurate SLO. SLOs define the internal goals for service performance, while SLAs define the minimum accepted level of performance promised to the customer. Here are some recommendations when creating SLAs:

- **Guide the SLA with the SLO target**: Define the commitment target in the SLA to be more relaxed than the defined SLO target; for example, if your SLO specified an uptime of 99.5%, the SLA should be slightly lower, maybe 99%, to allow for unexpected incidents.
- **Define clear penalties**: The SLA contract must indicate the consequence of breaching the SLA target. This can be a refund, a penalty fee, or contract termination.
- **Use simple language**: Avoid using complex terminology to define the contract terms so that all parties can understand what they are agreeing to.

7. Revisit, review, and update regularly

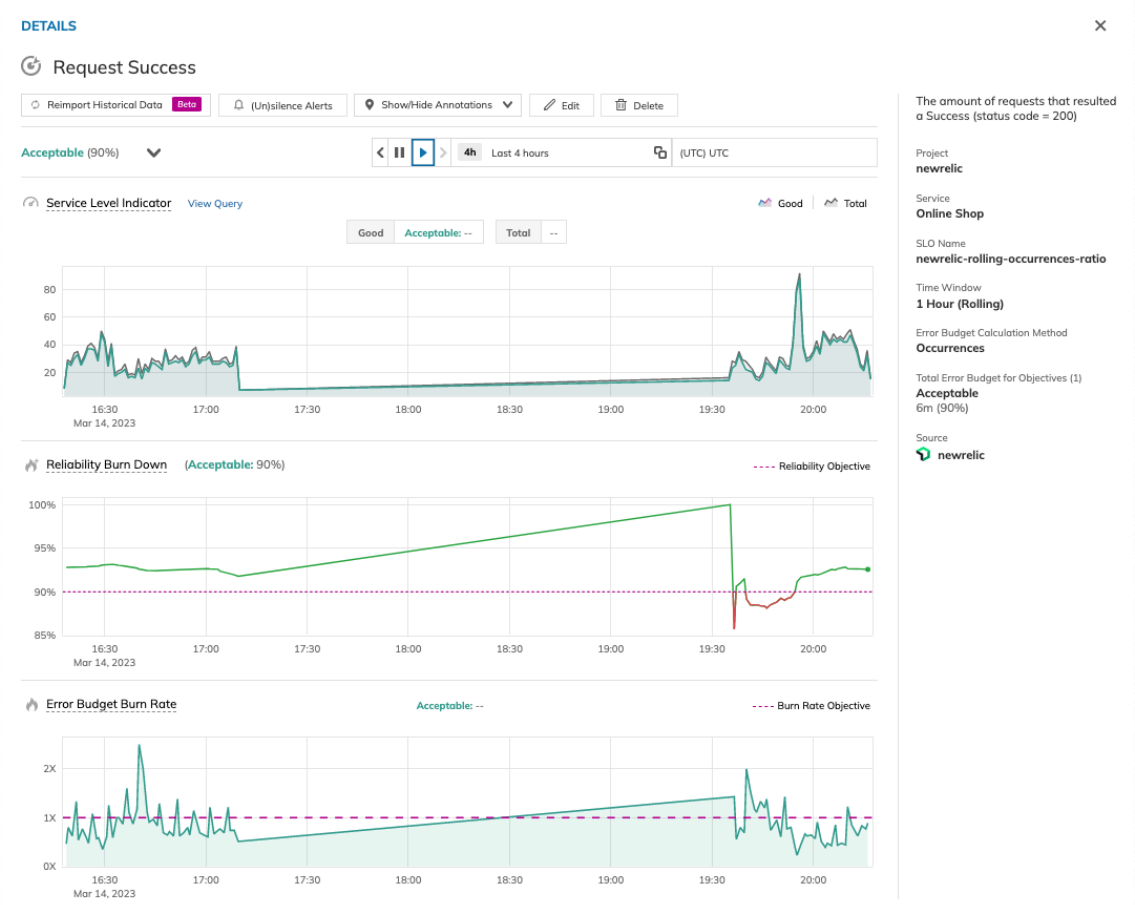
Feedback loops and continuous improvement are essential to a robust monitoring strategy. Teams should encourage empiricism and improvement with practices such as:

- **Consider SLOs as a dynamic process**: adjust and update periodically to align with the evolving users and business needs.
- **Use historical data**: Analyze the past and current SLI values to determine how an SLO target can be updated to be realistic and achievable.
- **Manage the factors contributing to error budgets**: External factors such as third-party outages and maintenance windows can skew error budgets. These anomalies can exaggerate failure rates and give an inaccurate representation of service reliability. For example, Nobl9 allows users to control the factors contributing to the error budget to circumvent this problem.
- **Tighten or loosen SLOs**: Based on historical performance data, check your SLOs. If they are consistently met, consider tightening them; if they are too challenging, try relaxing them to allow more error budget.

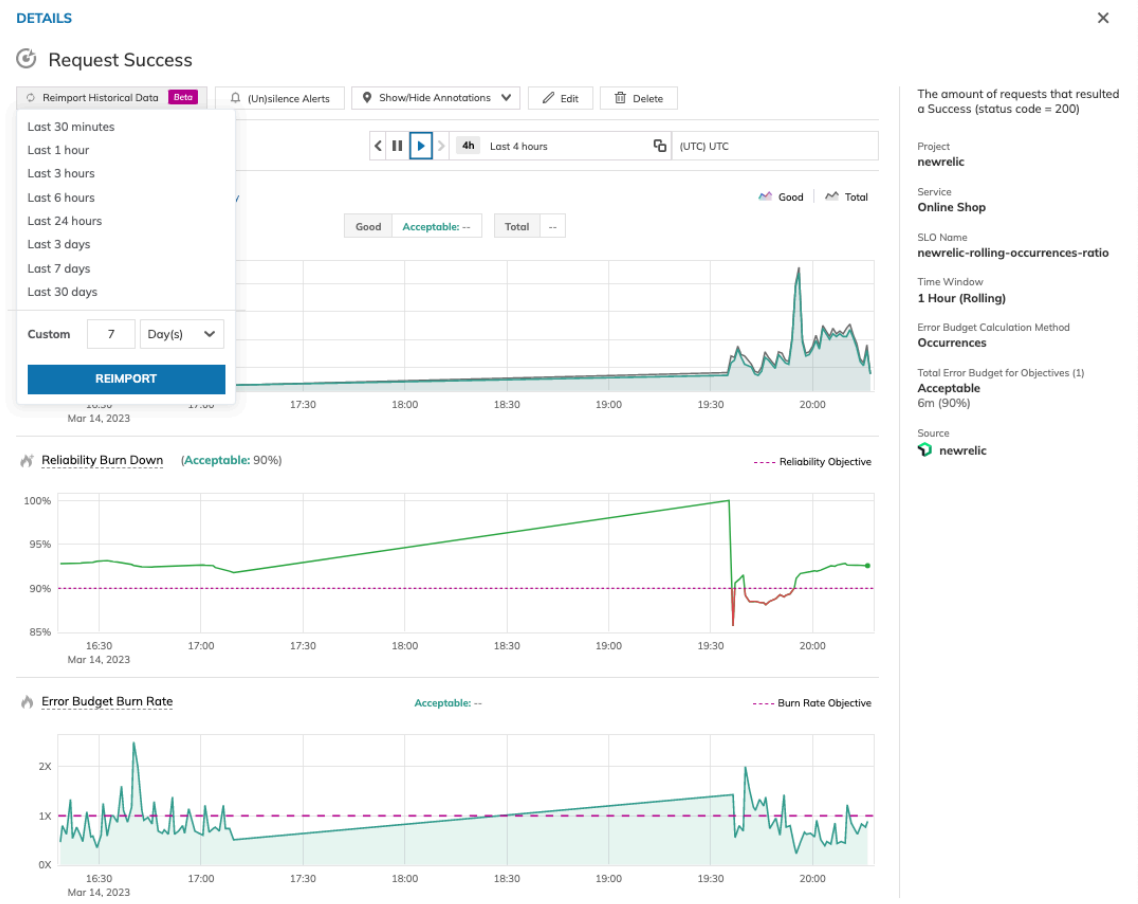
Recovering historical data with Nobl9 Replay

What happens if your SLI data is lost or your monitoring system corrupts?

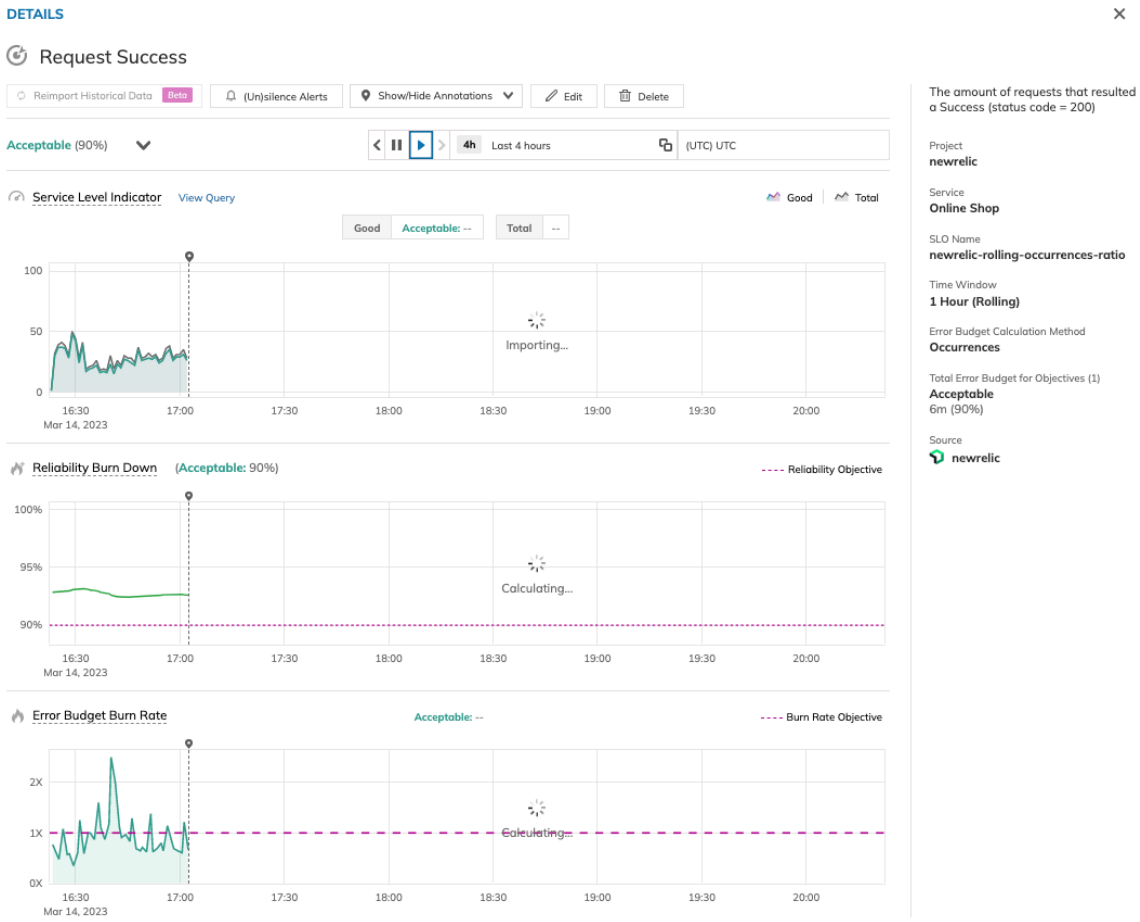
Nobl9 introduced the Replay feature to address this common monitoring problem. Replay allows users to retrieve historical SLI data and recalculate SLO error budgets without worrying about data loss or gaps in SLI data that will affect the calculation of the SLO. This is equivalent to fast-forwarding historical data through the SLO logic to evaluate the results. Replaying the stored historical data allows teams to assess different SLO configurations and the design of the ideal SLO without waiting weeks to observe the live SLI metrics and try different configuration permutations sequentially.



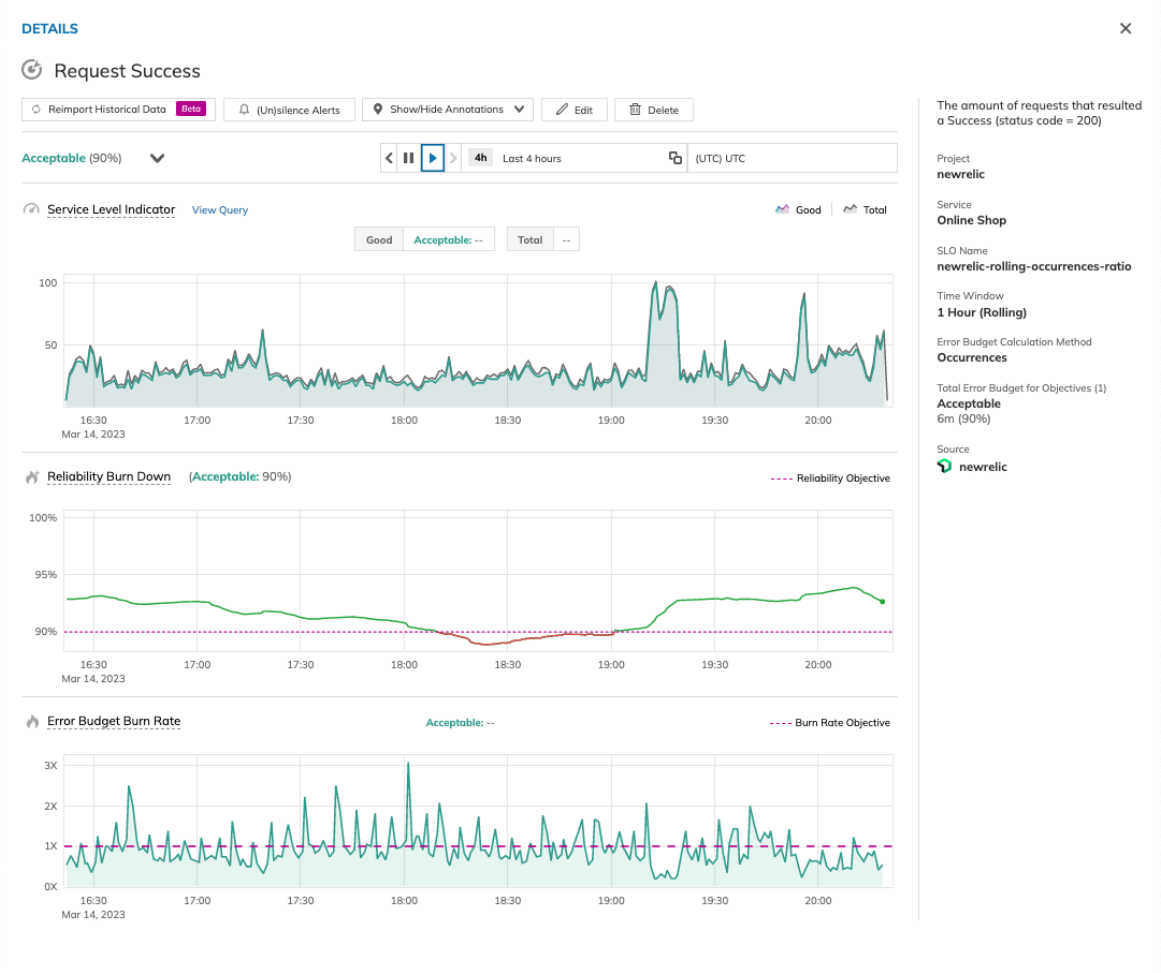
An affected SLO with data loss. (Source)



Reimporting historical data in the SLO details pane. (Source)



Reimporting and error budget calculation are in progress. (Source)



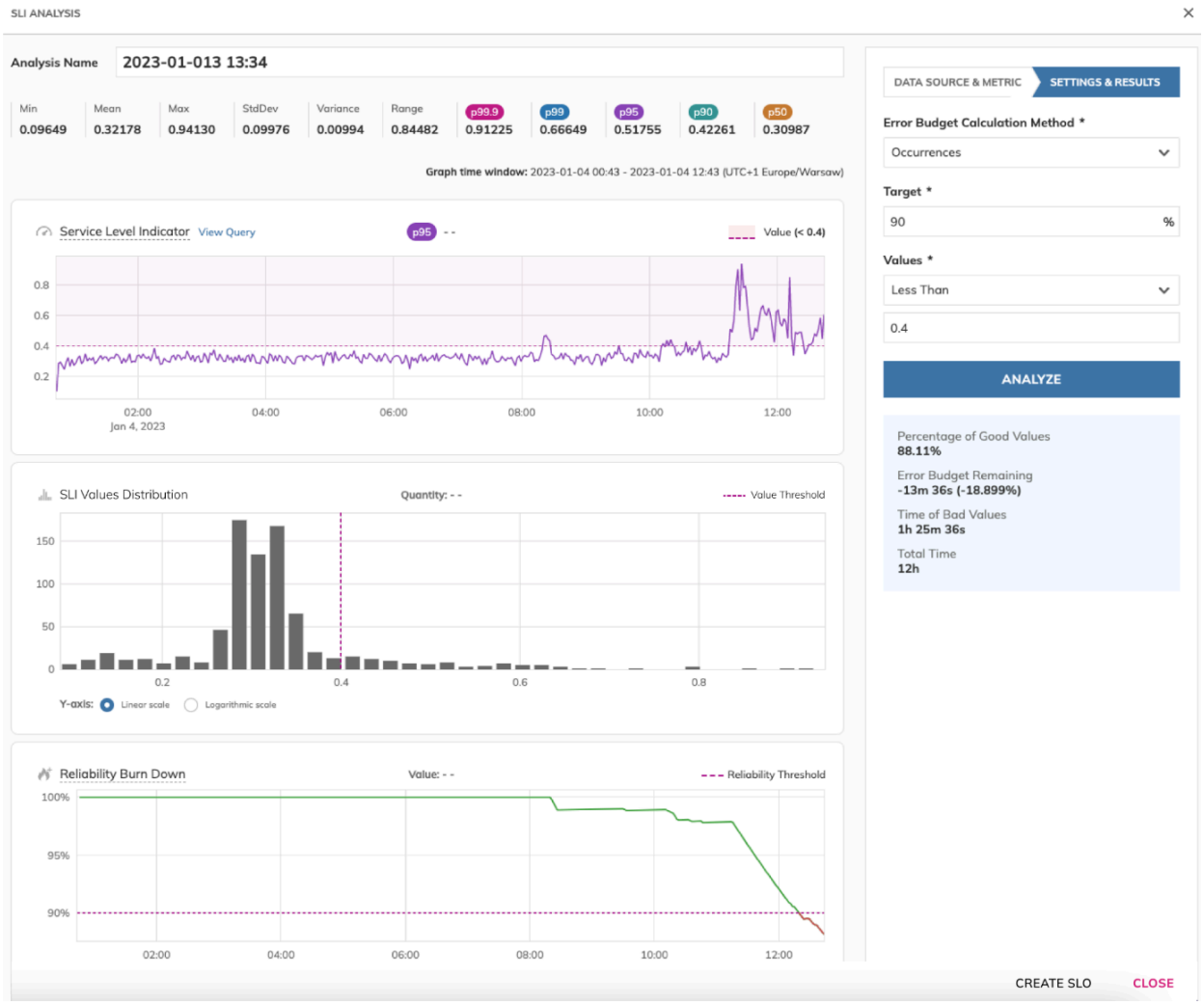
SLO with backfilled data. (Source)

Test and adjust the SLOs with the Nobl9 SLI Analyzer

To set a reliable and realistic target for your SLO, you will need to review your system's historical performance. Nobl9 introduced the SLI Analyzer feature, which processes up to 30 days of historical data and lets you review the outcome of your SLO before you implement it. This avoids the risk of unrealistic targets and saves the time you would spend on trial and error.

You can change the targets from the interface and tweak them to see the resulting error budgets and error budget burndown. Once your analysis is complete and you have determined your target, you can directly create a new SLO from the analysis

interface.



Analysis result from SLI Analyzer (source)

Last thoughts

SLOs, including composite SLOs, are essential components of an effective monitoring strategy. SLOs mainly focus on a single target, while composite SLOs combine multiple SLOs from different targets into one SLO; it's ideal for complex systems with multiple services. We have also discussed the alerting strategies for SLOs; you can alert when the service's performance crosses a predefined threshold (static) or alert on the burn rate, whether slow or fast. Using the multi-window, multi-burn-rate approach is always recommended to capture both short-term spikes in error rate and long-term consumption of the error budget.

By adopting SLO best practices, you can make sure that you have set up an effective SLO. Organizations should define SLOs based on critical metrics focused on user experience, set an achievable target based on historical data, and involve all teams and stakeholders.