Adopting SLIs and SLOs for internal cloud platform

The What?

As part of our tech goals for the last four months, we aimed to establish SLOs/SLIs for our internal compute platform that uses Amazon EKS for the backend. We call it *Infinity* for the reason that it scales automatically with respect to the incoming traffic without limitations so our users don't have to care about the resources allocation. Infinity runs 100s of different microservice applications for our product engineering teams. It enables our builders to run their applications reliably and securely at scale, without worrying about the complexities of cloud infrastructure setup of AWS. As a platform engineering team, our main focus is to increase developer productivity and improve the developer experience. Applications/services deployed to Infinity get default configuration out of the box. For example, efficient traffic load balancing, human readable DNS records, centralized log events, application monitoring, tracing and many more.

If you are unfamiliar with the concept of SLO/SLI/SLAs here is a brief overview of how they are defined:

SLIs	SLOs	SLAs
Service Level indicators represent the measurement to determine the availability of the systems/service/application	Service Level Objectives informed by SLIs are the goals we set for how much target availability we expect of a system/service/application	Service Level Agreement are the legal contracts that we promise to our user that explains what happens if the system doesn't meet its target SLO
X should be true..	Y times..	Or else..

The Why?

At Scout24, one of our core engineering values is having reliable systems that promises our users (internal product engineering teams) to have trust in the products provided by the application-platform team. Adopting SLIs and SLOs for all application-platform products and making it transparent to our builders shows our commitments to build reliable products without sacrificing the developing velocity and is an indicative of the product's health. Making these promises available to our users also helps in reducing the cognitive overload on platform teams whether to focus on a feature request(development) or work on operation stability of the system. Aannndddd, it's cool too :)

About application-platform.

> We, at Scout24, provide useful and usable products and guidance around internal tooling like compute platform, CI/CD, observability, persistence, storage, traffic, and resource management that enables Scout24 product engineering teams to efficiently build, deploy, and run secure & reliable applications at scale.

Tech stack and languages we use—AWS, EKS, CDK, Jenkins, Aurora Postgres and MySQL, Datadog, Kibana, Typescript, Golang, python,

If this excites you to work for us, please have a look at our open positions here and apply right away!

When we started on this journey of defining the **service levels** for our internal users, it was a bit of an effort on the 'how-to' part as we couldn't find any help in the market who could guide us in defining SLOs for internal compute platform. Questions like what does a service mean for our users? How much availability can we strive for? How to derive the availability of systems that we are dependent on were challenging?

Infinity runs ~700 microservice applications for more than 50 teams (at the time of writing this blog) and is the recommended compute solution at Scout24. As mentioned earlier, Scout24's best practices are baked inside infinity and our product engineering team's leverage those out of the box. For example, best practices of standard way of stage naming, monitoring dashboards, alerting, SLO dashboards, and ways of logging at Scout24. With all the batteries included, Infinity becomes a business 'critical' service which in case of any incident will have major customer impact and potential loss of business revenue for Scout24. Hence, it is imperative for application-platform team to keep infinity's operations flawless and promise our users with highest safety and reliability.

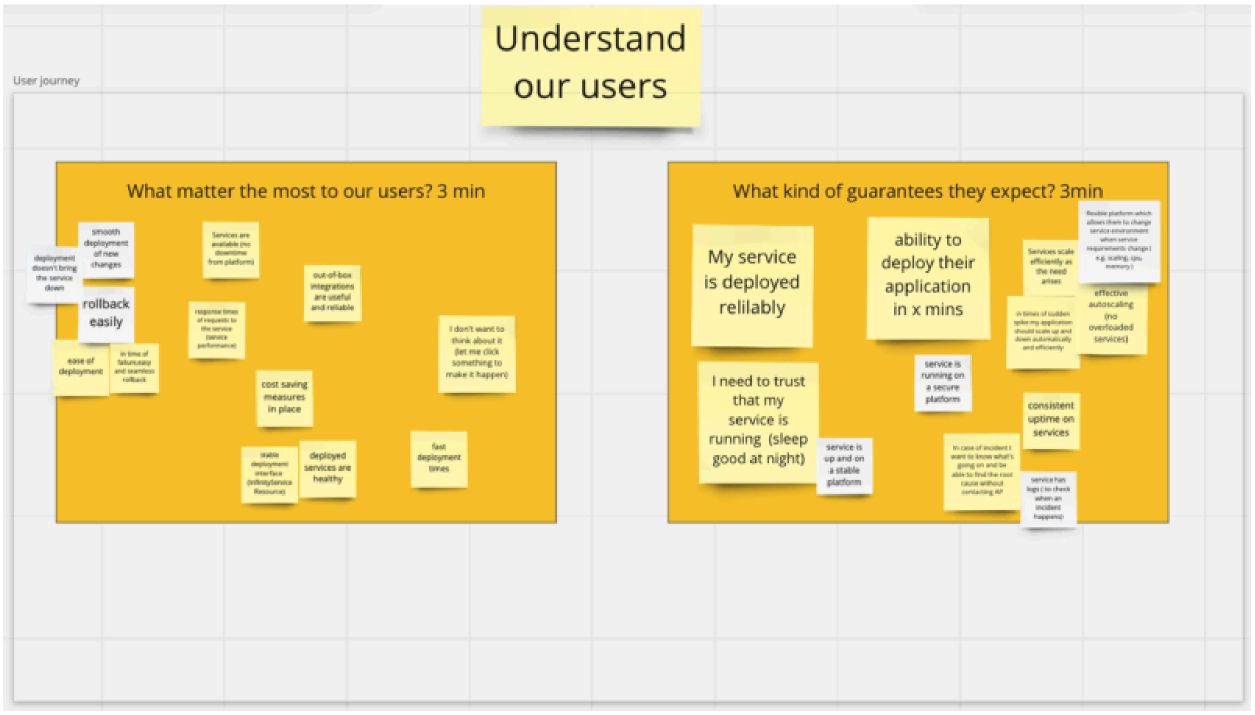
The How?

We started with a simple recipe laid out by our observability team in defining right SLIs and SLOs.

Here is the link to the template that you can use to run a small workshop with your team (Put a link to the miro board as a template to define SLI/SLO)

Step -1: Start with understanding your user. What do they use your service for? What matters most to them? What kind of guarantees do they expect? Decide clearly who the "users" are for your service. These are the people whose happiness you are optimizing.

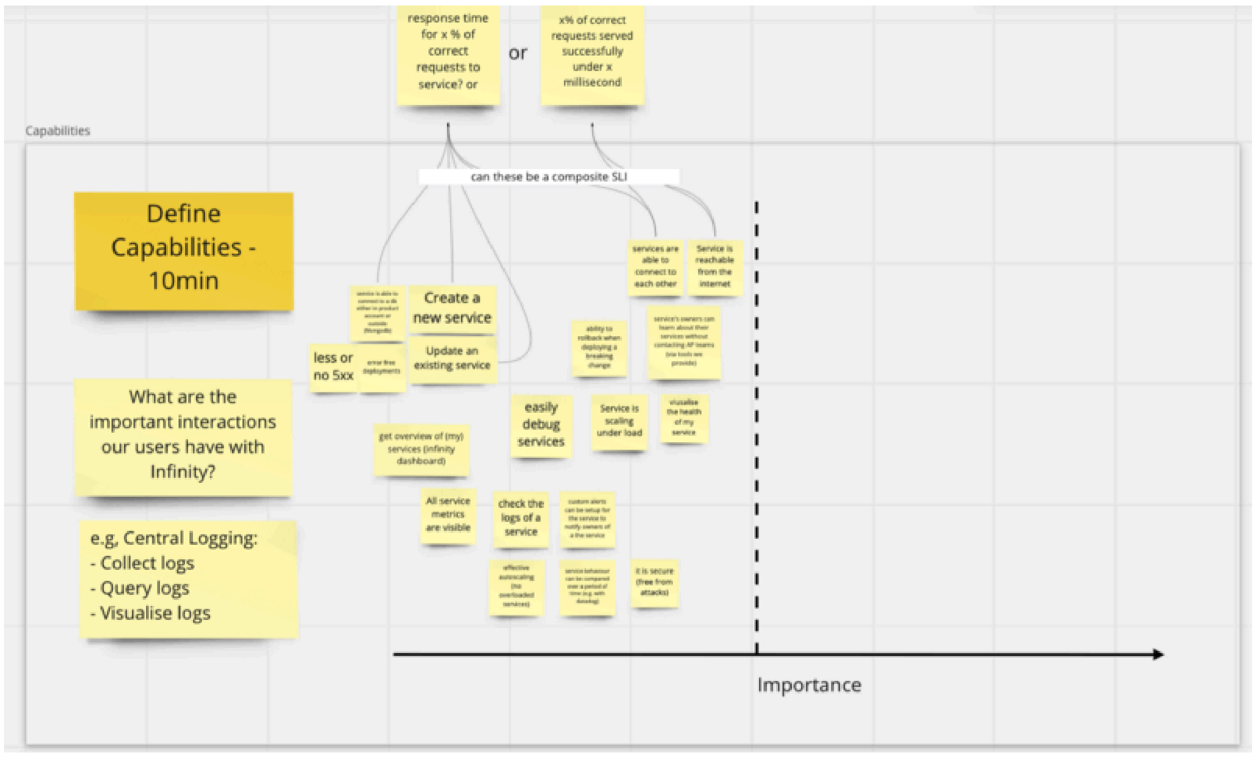
Some examples for Infinity in this case would be, *ease of deployments, any updates to the service should not bring any downtime.* In terms of guarantees we think our users expect *the ability to deploy a service in x minutes, their service is running on a stable and secure platform(a good night sleep and no alarms for on-call), service has logs to check when an incident happens and so on.*


Understand our users
User journey
What matter the most to our users? 3 min
smooth deployment of new changes
deployment doesn't bring the service down
rollback easily
ease of deployment
in time of failure,easy and seamless rollback
Services are available (no downtime from platform)
response times of requests to the service (service performance)
out-of-box integrations are useful and reliable
I don't want to think about it (let me click something to make it happen)
cost saving measures in place
stable deployment interface (InfinityService Resource)
deployed services are healthy
fast deployment times
What kind of guarantees they expect? 3min
My service is deployed relilably
ability to deploy their application in x mins
Services scale efficiently as the need arises
flexible platform which allows them to change service environment when service requirements change (e.g. scaling, cpu, memory)
in times of sudden spike my application should scale up and down automatically and efficiently
effective autoscaling (no overloaded services)
service is running on a secure platform
I need to trust that my service is running (sleep good at night)
service is up and on a stable platform
consistent uptime on services
In case of incident I want to know what's going on and be able to find the root cause without contacting AP
service has logs (to check when an incident happens)

It is highly recommended that your users also participate in this workshop when you are defining what your "service" means to your users.

Step -2: Use plain/simple English to define what your service is for your user, skip the technical details.

Step -3: Define Capabilities of your service. These are the *abilities* that your service is capable of doing for your user. Consider, what are different ways/activities your users interact with your service/system to get their job done. For infinity(our internal compute platform), these capabilities would be defined as, *ability to create/update a service, a user is able to deploy their applications without errors, different services running on infinity are able to connect to each other, the services deployed are secure and free from risk of attacks from internet, service is able to scale up and down based on the incoming traffic.*



response time for x % of correct requests to service? or
or
x% of correct requests served successfully under x millisecond
Capabilities
can these be a composite SLI
Define Capabilities - 10min
What are the important interactions our users have with Infinity?
e.g, Central Logging:
- Collect logs
- Query logs
- Visualise logs
Create a new service
Update an existing service
less or no 5xx
error free deployments
services are able to connect to each other
Service is reachable from the internet
ability to rollback when deploying a breaking change
service's owners can learn about their services without contacting AP teams (via tools we provide)
easily debug services
Service is scaling under load
visualise the health of my service
get overview of (my) services (infinity dashboard)
All service metrics are visible
check the logs of a service
custom alerts can be setup for the service to notify owners of a the service
effective autoscaling (no overloaded services)
service behaviour can be compared over a period of time (e.g. with datadog)
it is secure (free from attacks)
Importance

Step -4: Abstract SLIs assuming perfect metrics are existing. In order to get a good start on defining the SLI, you can formulate your capabilities under following categories -

1. Request/Response.
- Availability : the proportion of *valid* requests served *successfully*
- Latency: the proportion of *valid* requests served *faster* than a *threshold*
- Quality: the proportion of *valid* requests served without *degrading service quality*

2. Data processing

- **Coverage**: the proportion of *valid* data processed *successfully*
- **Freshness**: the proportion of *valid* data updated *more recently* than a *threshold*
- **Correctness**: the proportion of *valid* data producing *correct output*

3. Storage

- **Durability**: the proportion of *written data* that can be *successfully re-read*

From Infinity's capabilities and the expectations from users, we concluded that only 'availability' and 'latency' for request/response types SLOs could be abstracted.

Our abstracted SLIs for Availability and Latency are defined as follows -

Availability—

1. *ratio of healthy containers (*which are being able to service traffic)*to the desired containers*(needed by the application, determined by the autoscaling group)*.* This provides a good snapshot of availability of a service/application running on infinity, i.e. are services up and able to serve traffic.

Metrics used are

a= kubernetes_state.deployment.replicas_desired per cluster as sum, and b= aws.applicationelb.healthy_host_count per cluster as sum.

resulting equation = a / b *100 yielded the desired result.

2*. ratio of successful cloudformation(cfn)* ***response*** *to valid cloudformation(cfn)* ***requests.*** Our users only create services using AWS cloudformation custom::resource. We can determine failing cloudformation *requests* by the status of the cloudformation stack and what error message is generated. This helps us differentiate between errors caused by Infinity and errors caused by AWS.

At Scout24, we use datadog as our monitoring tool and alert for non AWS metrics. All metrics mentioned in this blog are datadog metrics. For sake of simplicity, the jargon in the following metrics are simplified. For example, "infinity" is renamed from "i6y" as the actual metric we push from infinity(our compute platform) to datadog.

If you use other monitoring tools, your metrics might look different.

Metrics used are -

a= infinity.cloudformation_response.count and sum it for all EKS clusters and response_type:success

b= infinity.cloudformation_response.count and sum it for all EKS clusters and response_type:user_error

c =infinity.cloudformation_request.count and sum it for all EKS clusters and valid:true

resulting equation = [(a + b)/c x 100] yielded the desired result.

Latency—

1. *Proportion of services created and updated successfully.* In order to learn about service latency, we separated the deploymentDuration during each operation: creation, updating, and deleting a service.

Metrics used are -

a= sum:infinity.cfn-interface-service_latency from{resource_status:ready, !service_name:inf*} and sum by aws_account_name

b =sum:infinity.cfn-interface service_latency from {!service_name:inf*} and sum by aws_account_name

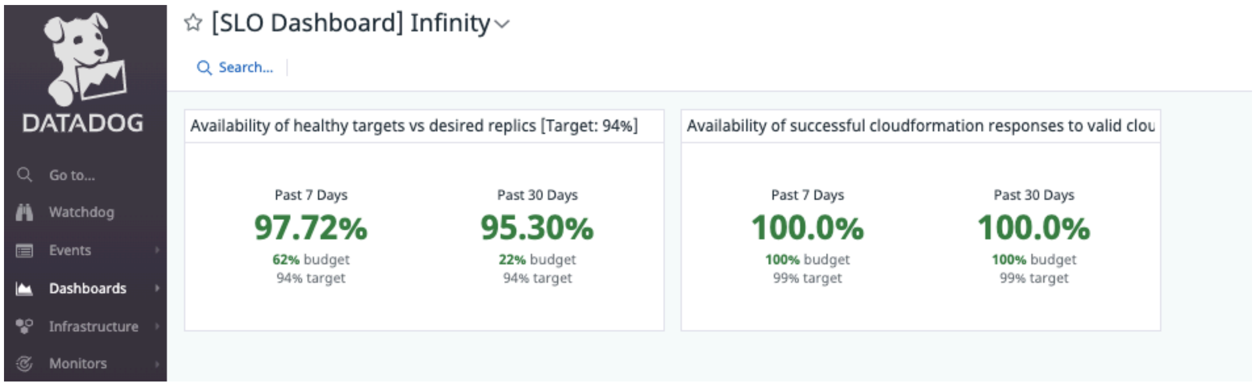
resulting equation = [(a / b) * 100] yielded the desired result.

Step -5: List down any dependencies your service/application/system is dependent on

Start by listing each of your dependencies and their SLOs. If one of your dependencies has a certain SLO, you can't guarantee anything higher than that without building fault tolerance to handle failures in that dependency. In our case, our only dependency was AWS' availability and at the time of writing, we couldn't abstract the guarantees from them. In such a case, where your dependency doesn't have any SLO, our suggestion is to track and monitor that dependency availability yourself and what you expect from it. This will help you understand how available it is.

Before, we show you how the final metrics look like, a note on how the SLOs are generated and where the metrics are stored.

In Scout24, we utilize something called service catalog, a centralized inventory of metadata and information about services and products. It provides easy access to information related to who owns what inside Scout24 by centralizing and standardizing information about the different services and organizations. Once you define the metrics and how to calculate, we put the calculation metrics inside a yaml file which is stored in our service-catalog which then automatically creates a SLO dashboard out of the box as shown below.



[SLO Dashboard] Infinity

Availability of healthy targets vs desired replics [Target: 94%]

Past 7 Days 97.72% 62% budget 94% target

Past 30 Days 95.30% 22% budget 94% target

Availability of successful cloudformation responses to valid clou

Past 7 Days 100.0% 100% budget 99% target

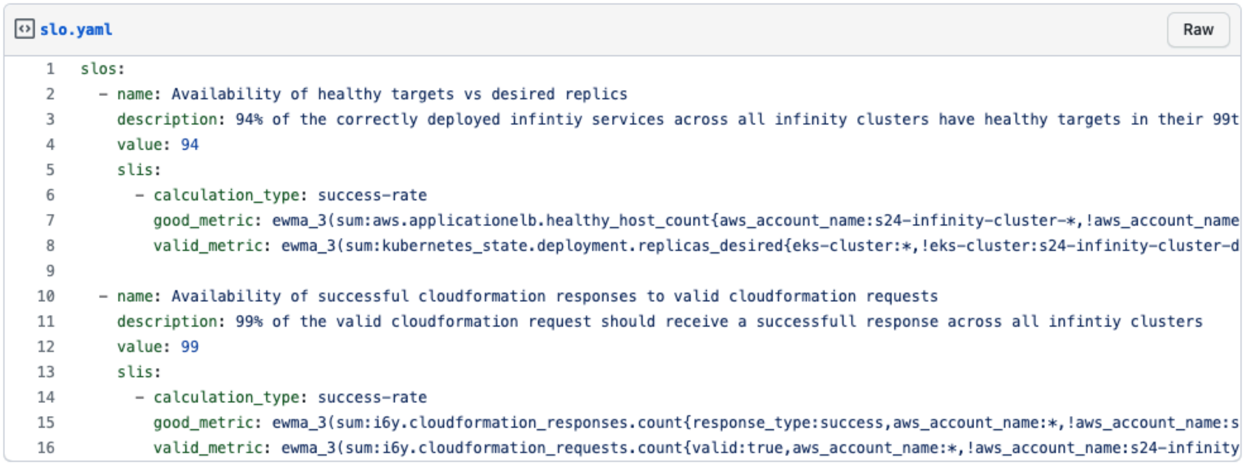
Past 30 Days 100.0% 100% budget 99% target

Step -6: Define SLOs

SLOs as stated earlier are the objective that you promise to your user about your service. Hence, the whole team needs to agree on these objectives before committing to the users. We also suggest not using the target based on current performance because in that case it might lead you to have a service that needs constant reliability work just to keep within its target. We observed the target objective for ~3 months to get the baseline.

> It's better to start with a lower target and increase it later than to loosen it later when you find the target is unattainable.

Our final SLOs defined and published looks like this -



```yaml
slos:
  - name: Availability of healthy targets vs desired replicas
    description: 94% of the correctly deployed infintiy services across all infinity clusters have healthy targets in their 99t
    value: 94
    slis:
      - calculation_type: success-rate
        good_metric: ewma_3(sum:aws.applicationelb.healthy_host_count{aws_account_name:s24-infinity-cluster-*,!aws_account_name
        valid_metric: ewma_3(sum:kubernetes_state.deployment.replicas_desired{eks-cluster:*,!eks-cluster:s24-infinity-cluster-d

  - name: Availability of successful cloudformation responses to valid cloudformation requests
    description: 99% of the valid cloudformation request should receive a successfull response across all infintiy clusters
    value: 99
    slis:
      - calculation_type: success-rate
        good_metric: ewma_3(sum:i6y.cloudformation_responses.count{response_type:success,aws_account_name:*,!aws_account_name:s
        valid_metric: ewma_3(sum:i6y.cloudformation_requests.count{valid:true,aws_account_name:*,!aws_account_name:s24-infinity
```

(I will put a Gist from github instead of this screenshot when publishing in medium)

At Scout24, we use datadog as our monitoring tool and alerts for non AWS metrics. All metrics mentioned above are datadog metrics. If you use other monitoring tools, your metrics might look different.

Step -7: Iterate and Tune

The first definition of the SLIs and SLOs aren't set in stone. They are meant to create a feedback loop with your users and monitor the health of your applications. They are a reliability commitment long term goals that are indicative of where you should invest your resources.

In case you breach your SLOs first time after defining, go back to the drawing board and see where you missed what and iterate over. ;)

If you have any feedback or questions, feel free to reach out to us or write to us here. Follow us for more updates as we will be posting more exciting things we do at application-platform.

Happy SLOing and stay tuned!

Further reading (in no particular order)—

SLO workshop—
https://www.usenix.org/sites/default/files/conference/protected-files/srecon18emea_slides_fong-jones.pdf

SLO overview from New relic—https://newrelic.com/blog/best-practices/best-practices-for-setting-slos-and-slis-for-modern-complex-systems

Implementing SLOs by Google—https://sre.google/workbook/implementing-slos/

A guide to service level objectives -https://www.circonus.com/2018/07/a-guide-to-service-level-objectives/