Creating your CUJ (critical User Journey) Lo-Fi

What does your application does and how does it deliver customer value

These are your customers / users

Who are your customers?

other apps

Other teams

end users

Defining the outlines of your CUJ

what are the entry and exit points of your app?

entry

exit

Defining the boundaries and patterns of your CUJ

what are your dependencies

dependency on you to deliver

dependency on someone else to deliver to you

Defining the SLIs

what are you delivering to your users

Defining the SLOs

What are we committing to with those deliverables

Defining the failure modes

How could things go wrong?

eg. innability to validate if an account has enough funds to complete a payment request

Who we are and how can we work with you?

We are Payments Observability

In the pursuit of a metrics-driven approach, we're partnering with payments teams to **accelerate your DevSecOps maturity** by establishing a **metrics-driven observability model**.

Our goal is for all payments teams to have **aligned alerts based on clear Service Level Indicators (SLIs) and Service Level Objectives (SLOs)**.

This ensures our monitoring is targeted, proactive, and directly supports the reliable delivery of critical payment services.

clearer on what is their starting point

show the example dashboard

show a step by step plan they can follow

pager duty is configurable

its made up by the support groups

there is no coding, its all configuration

rolling the data in to SLIs and SLOs is also all configuration

its quick and Iteratible

how do we start customer matrix in to measurable metrics?

is this a separate piece of work separate to the technical aspect

is this in addition to our technical scope?

is what we are doing in this CUJ more about the customer scope

we are starting with the customer side but we can definetly support on the technical journey if that is where you are on

we are trying to move away frmo logs and moving towards basing observability on metrics

and have the ability to consolidate the metric in to something that can show something tangible about what the application is performing

products have multiple application metrics that are going on

how do we measure the metrics with respedct of the products when the products have lots of applications involved

we are trying to pivot away from what the memory usage is

we are trying to move towards how do we move in to delivering value

if an application is gennerwtic metrics and the metrics are going in to obstack we can draw the broader flows

🔍 Critical User Journey (CUJ) Guide

This guide helps you define at a high level how your application delivers value to its users by mapping out key interactions and dependencies.

Define Your Customers/Users

Start by identifying who your application serves, this could be other applications, Internal teams or End users.

Outline your failure modes

Identify potential risks: What could go wrong in this journey? What does failure look like in terms of delivering to your users?

Define the outlines of a Journey

Based on a key user/s that you outlined earlier - Map the flow/journey from start to finish:

Entry point: Where does the user or system begin interacting with your app?
Exit point: What marks the completion of the journey?

Define SLIs (Service Level Indicators)

Determine what you'll measure to track performance: Choose indicators that reflect the health of the CUJ (e.g., latency, error rate, availability).

Boundaries & Dependencies

Once you've outlined a complete end-to-end flow that your application participates in, identify and define the segments where your application operates independently - those parts that are not reliant on external systems or integrations, and are solely handled within your application's boundaries

Define SLOs (Service Level Objectives)

Set performance targets: What level of service is acceptable? Use these to guide reliability goals and team priorities.

Agenda fro intro sessions

KEY INFO

intro

Who are we?

We are the payments observability squad, we exist inside of COE with our primary focus being on supporting the payments teams.

What is our mission?

what are we asking for

At a high level what's the ask on you?

How can we support?

Our sole responsibility is enabling your team to develop your DevSecOps maturity.

Based on your understanding of the observability initiative, what level would you say your teams are at?

shifting from relative to proactive monitoring

Shifting from logs to Metrics

Logs VS metrics approach

many teams already have a Loggs approach to observability

logs are really more of a diagnostic tool but they are not ideal for indicating when something is going wrong.

Moving forward, you will need to adopt a metrics based approach as metrics provide real time insights in to your applications health

we need to think about how to reflect probers and vs metrics for instrumentation

the central team is pushing an approach that everyone's base line needs to be prober based, this is not necessarily what we recommend / think

the approach vs an approach

Approach

we are here to help you we need you to ask for help at any of these steps.

Start

Generate Metrics (Push/Pull Instrumentation)

Do these metrics support my defined observability questions?

Yes

Onboard to Obstack (Push/Pull, Dashboards)

Assess the metrics & dashboards

Define CUJ/SLOs/SLIs

Do these metrics support my defined observability questions?

Yes

Update CUJs/SLOs with new SLOs/SLIs

Configure Alerts & Test Alerting thresholds

Do the SLOs/SLIs support my CUJs?

Yes

Monitor, Analyse & Evaluate (Target State)

No

Customisation /Extension

No

Revisit Instrumentation Method

No

Refine and Iterate

Do you have any questions so far

Self assessment

Next steps

Revisit the approach diagram so that you fully understand the iterative nature of this process

Use the self assessment tool to define the current state of your Observability journey

Continue your applications observability journey

Reach out to us if you need help kicking off.

Resources

What resources do we have to support you?

Our resources:
- Observability program & strategy - CUJ's & How to create them
- Technical Resources & Implementation

CRE resources:
- Observability and SRE
- Observability guidance

Capability measure	Level 0	Awareness	Level 1	Enablement	Level 2	Embedding	Level 3
What type of observability data exists for your technical service and how easily is it available?	The data to support diagnosis of typical failures is not readily collected or available.	Existing logging, metrics and traces exist to support diagnosing failures, but may require manual digging or building of ad hoc tools.	Documentation exists noting the logs, metrics and traces needed to support can levels and response to incidents	All observability data for the service is available via pre-built dashboards and is sufficient to diagnose a production issue and recover quickly.			
How does your squad find out that your service is broken?	Failures of the service do not generate real-time alerts.	Failures of the service generate alerts but those alerts are delivered to a central team (e.g. FlightDeck) not the engineering team	Alerts capturing failures are well defined and documented for the failed service who can fix it	Alerts are received before customer complaints about a degraded/unavailable service. The squad actively maintains these alerts.			
How does your squad measure the reliability of your service?	The reliability of the service is not measured, or is inferred from for e.g. incident reports or customer complaints.	A Service Level Indicator (SLI) measuring the service is published with long term data collected	Reliability of the service is measured by executing at least 1 core business function/journey. Service Level Objective (SLO) target established	Service Level Objective (SLO) have coverage of most critical business functions with SLOs that reflect customer and business requirements. The business is regularly providing feedback in SLO governance			

FAQs/Common Talking Points

How do I check my current state of observability?

Measure against the guide.

Most teams will be starting with logs being onboarded to Observe. (Level 1)

Observe vs Obstack

Observe Inc is an enterprise logging tool for centralising diagnostic log information that allow the owners of a system to inspect and diagnose and / reason about its behaviours in order to debug or fix or optimise it.

Obstack is the Group's metrics store and alerting tool, for SLI & SLO alerts derived from metrics.

Starting Point

You can start where works best for your squad.

How is the CUJ analysis related to the generation of metrics?

How hard is it to set up dashboards and alerts?

We have a template available that when utilised can give you something fairly similar to this [link]

As a PO why would I care about this?

How do I get access to the Dashboard?

All metrics hosted on metrics.cba grafana/prometheus instance - is open to anyone at CBA.

My application is very small is it still worth generating metrics?

Absolutely. Even small services can be critical components within a larger system. Having metrics in place ensures you can monitor performance, catch issues early, and provide valuable insights that help on your service.

Where should I start if I have many applications?

We encourage teams to prioritize applications that serve as critical dependencies within key systems , especially those tied to customer-facing deliverables.

Creating your CUJ (critical User Journey) Lo-Fi

What does your application does and how does it deliver customer value

Define your customers / users

Who are your customers?

other apps

Other teams

end users

Defining the outlines of your CUj

what are the entry and exit points of your app?

entry

exit

What user will you outline as the key customer that your CUj is relating to?

Defining the boundaries and patterns of your CUj

what are your dependencies

dependency on you to deliver

dependency on someone else to deliver to you

Defining the failure modes

How could things go wrong?

Defining the SLIs

what are you delivering to your users

Defining the SLOs

What are we committing to with those deliverables

Each level is a different pass of the iterative process. Level 3 demands constant evaluation of the alerts, SLIs, SLOs and continuous improvement to make sure everything aligns with the business requirements.

Capability measure	Level 0	Awareness | Level 1	Enablement | Level 2	Embedding | Level 3
What tools are you using to measure your applications health? (e.g. metrics, how can you visualise this?)	No visibility	Logs going to Observe.	Baseline metrics generated and going to Obstack with Dashboard Logs going to Observe.	Detailed dashboards with dependency's shown Well defined metrics being generated and onboarded to Obstack Logs going to Observe.
What does your current incident response process look like? How do you know if something is broken?	No alerts, relying on user complaints	Alerts going to FlightDeck	Alerts going to PagerDuty and respective team members	Different levels of Alerts with critical ones going to PagerDuty Team is using the dashboards to check predicted issues to help mitigate potential issues before they impact customers.
How do you define and measure your SLIs/SLOs to ensure they are aligned with the business expectations?	No level of measure	Having some sort of indicator that defines when an alert is triggered.	A first pass defined CUj Mix of generalised SLI/SLOs and defined SLI/SLOS	Well defined SLI/SLOs based on CUjs. Defined and iterated on CUjs Constant revision of alert thresholds, SLIs,SLOs

 Observability maturity self assessment

Use this self assessment tool to help you define the observability maturity of your application. Doing this will provide clarity over where to continue your applications observability journey.

Name of the people working on this

Name of your squad

NOTE
Each level is a different pass of the iterative process. Level 3 demands constant evaluation of the alerts, SLIs, SLOs and continuous improvement to make sure everything aligns with the business requirements.

New grid

Capability measure	Current state assessment	Level 0	Current state assessment	Level 1	Current state assessment	Level 2	Current state assessment	Level 3	Current state assessment
What type of observability data exists for your technical service and how easily is it available?	What tools are you using to measure your applications health? Logs or metrics, how can you visualize this? Add details...	The data to support diagnosis of typical failures is not readily collected or available. E.g. No metrics or logs being generated or captured	Add details...	Existing logging, metrics and traces exist to support diagnosing failures, but may require digging or building ad hoc tools. E.g. Logs going to Observe	Add details...	Documentation exists noting the logs, metrics and traces needed to support run books and response to incidents. E.g. Baseline metrics generated and going to Obstack with Dashboard. Logs going to Observe.	Add details...	All observability data for the service is available on real-time dashboards and is sufficient to diagnose a production issue and recover quickly. E.g. Detailed dashboards with dependencies shown. Well defined metrics being generated and onboarded to ObStack. Logs going to Observe.	Add details...
How does your squad find out that your service is broken?	What does your current incident response process look like? How do you know if something is broken? Add details...	Failures of the service do not generate real-time alerts. E.g. No alerts, relying on user complaints.	Add details...	Failures of the service generate alerts but those alerts are delivered to a central team (e.g. Pagerduty), not the engineering team. E.g. Alerts going to FlightDeck	Add details...	Alerts capturing failures are sent to the squad responsible for the failed service who can fix it E.g. Alerts going through PagerDuty based on SLO's.	Add details...	Alerts are received before customer complaints about a degraded/unavailable service. The squad actively maintains these alerts E.g. Different levels of Alerts with critical ones going to PagerDuty Team is using the dashboards to check predicted issues to help mitigate potential issues before they impact customers.	Add details...
How does your squad measure the reliability of your service?	How do you define and measure your applications deliverables against the business expectations? Add details...	The reliability of the service is not measured, or is inferred from for e.g. incident reports or customer complaints E.g. No level of measure	Add details...	A Service Level Indicator (SLI) measuring the service is published with long term data collected E.g. Having some sort of indicator that defines when an alert is triggered.	Add details...	Reliability of the service is measured by exercising at least 1 core business function with an Service Level Objective (SLO) target established E.g. A first pass defined CUJ. Mix of generalised SLI/SLOs and defined SLI/SLOS	Add details...	Service Level Objective (SLO) have coverage of most critical business functions with SLOs target governance in line with business requirements. The Business is regularly providing feedback in SLO governance E.g. Well defined SLI/SLOs based on CUJs. Defined and iterate on CUJs Constant revision of alert thresholds, SLIs,SLOs	Add details...

Braking down SLI and SLO creation

What makes a good SLO

What makes a good SLI

what is it measuring

what is the business function

what is the internal deliverable (not dependent on external factors)

Service Reliability:

- SLIs & SLOs
- Observations

April 2025

Commonwealth Bank

Service Reliability: SLIs & SLOs

SLIs: Service Level Indicators

"An SLI is a **service level indicator**, a carefully defined quantitative measure of some aspect of the level of service that is provided."

SLOs: Service Level Objectives

"An SLO is a **service level objective**, a target value or range of values for a service level that is measured by an SLI."

Source: *The Google SRE Book*

Commonwealth Bank

Service Reliability:

SLIs & SLOs

SLIs: Service Level Indicators =
Reliability Metrics

~~An SLI is a **service level indicator**: a carefully defined quantitative measure of some aspect of the level of service that is provided.~~

SLOs: Service Level Objectives =
Reliability Targets

~~An SLO is a **service level objective**: a target value or range of values for a service level that is measured by an SLI.~~

Commonwealth Bank

SLIs and SLOs at CBA (part 1)

Problem statement; why are we discussing these?

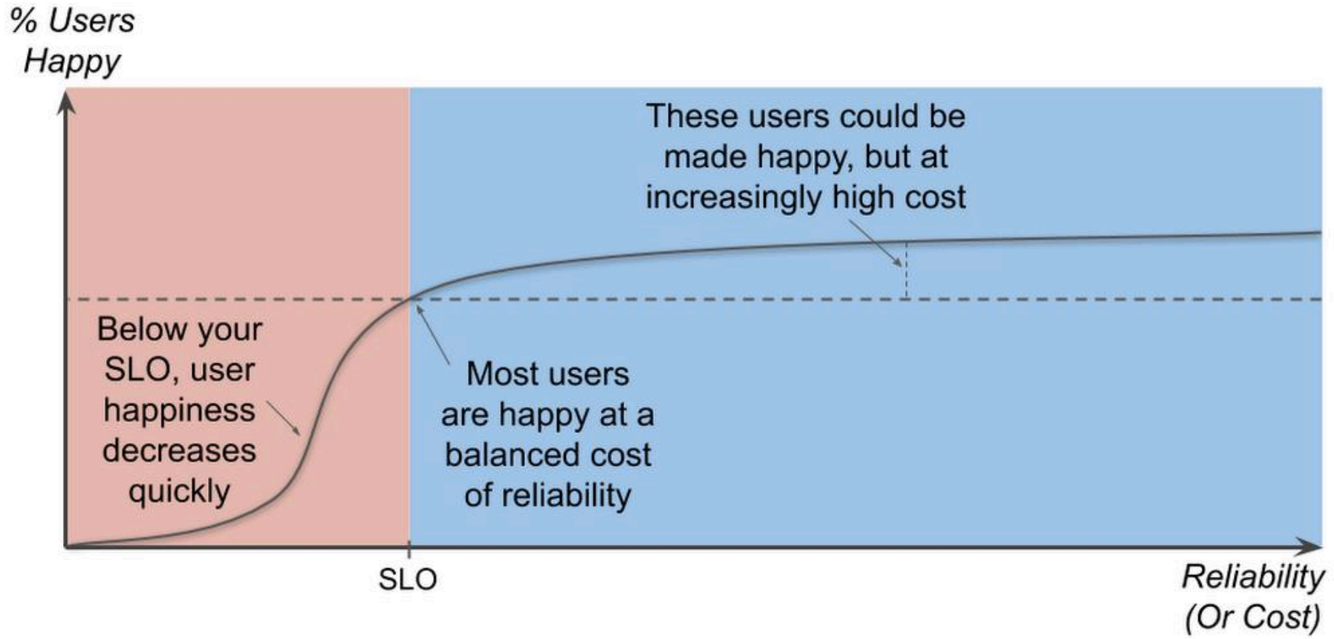
- *CBA requires a uniform way to define, measure and track the reliability of critical services*.
- *Service Level Objectives (SLOs) are a foundation of SRE and provide a standardised way to measure service reliability*.
- *CBA is broadly adopting SLOs as it transitions to a DevSecOps model*.

- In Q1/25, all BU/SUs have an OKR to create fully end-to-end SLOs for at least one of their significant services. The expectation is to build on this experience to develop SLOs for all significant services.
- Central SRE has published four levels (0 - 3) of SLO maturity:
  > CBA has a requirement for all '*TCF Foundational and Very High Critical services*' to reach and sustain SLO maturity level 2.
- CBA's Central SRE team is providing guidance on adopting SLOs, with tooling for defining, tracking and visualising them.

Service reliability

We don’t define reliability; critical user journeys do!

- It’s important to recognise that it is our users that define what’s an acceptable level of service ‘reliability’.
- A level of reliability that’s too low won’t meet user expectations and ‘happiness’ diminishes rapidly.
- A level of reliability that’s too high is expensive, and the marginal gains in user ‘happiness’ don’t justify the extra cost.
- *Well defined SLOs are a balance between meeting user expectations and the costs to implement and sustain a level of reliability*.



Source: *Google Cloud Blog*

Reliability levels

Availability is usually expressed as a percentage of uptime in a given period

Reliability Level	Permitted Total Downtime *Per year*	*Per quarter*	*Per 30 days*
90%	36.5 days	9 days	3 days
95%	18.25 days	4.5 days	1.5 days
99%	3.65 days	21.6 hours	7.2 hours
99.5%	1.83 days	10.8 hours	3.6 hours
99.9%	8.76 hours	2.16 hours	43.2 minutes
99.95%	4.38 hours	1.08 hours	21.6 minutes
99.99%	52.6 minutes	12.96 minutes	4.32 minutes
99.999%	5.26 minutes	1.30 minutes	25.9 seconds

← 863ms per day!

Source: *The Google SRE Book*

But it’s not quite that simple

Reliability Level	Allowed Unreliability Window		
	Per year	*Per quarter*	*Per 30 days*
90%	36.5 days	9 days	3 days
95%	18.25 days	4.5 days	1.5 days
99%	3.65 days	21.6 hours	7.2 hours
99.5%	1.83 days	10.8 hours	3.6 hours
99.9%	8.76 hours	2.16 hours	43.2 minutes
99.95%	4.38 hours	1.08 hours	21.6 minutes
99.99%	52.6 minutes	12.96 minutes	4.32 minutes
99.999%	5.26 minutes	1.30 minutes	25.9 seconds

Error Rate	Allowed Duration
100%	21.6 minutes
10%	3.6 hours
1%	36 hours
0.1%	15 days
< 0.05%	All month

Source: *The Google SRE Book*

“100% is the wrong reliability target for basically everything.”

Benjamin Treynor Sloss, Vice President of 24x7 Engineering, Google

- 100% reliability is prohibitively expensive – Google’s ‘Enterprise Roadmap to SRE’ suggests that *every extra ‘9’ in reliability will incur a 10x increase in cost*.

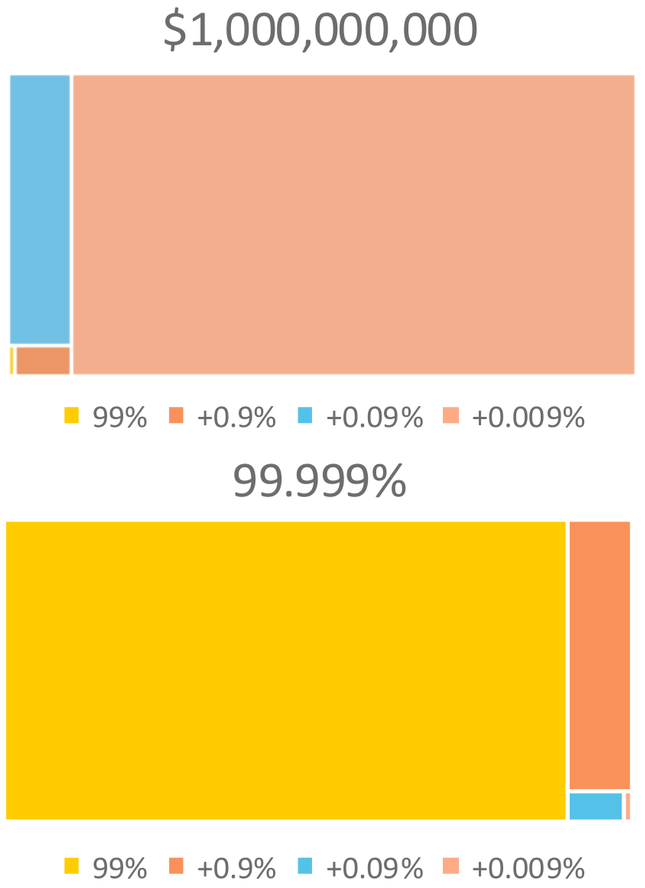
  99% @ $1,000,000 → 99.9% @ $10,000,000 → 99.99% @ $100,000,000 → 99.999% @ $1,000,000,000

- If your SLO is aligned with customer expectations, 100% is simply not a reasonable goal.

- Even if *you* could achieve 100% reliability within *your* system, your customers would likely not experience *overall* 100% reliability.

- If you do manage to create an experience that is 100% reliable, and want to maintain that level of reliability, you can never update or improve your service.

- An SLO of 100% means you only have time to be reactive.



SLO Process Overview

How to SLO? Start simple, start small. Rinse and repeat.

1. Identify critical user journeys (CUJs) and order them by business impact.
2. For each CUJ, determine which metrics to use as service-level indicators (SLIs) to most accurately track the user experience.
3. Determine service-level objective (SLO) target goals and the SLO measurement period.
4. Create SLI and SLO consoles.
5. Create SLO based alerts.

Critical User Journeys (CUJs)

The first step in SLO creation is listing *critical user journeys* for your service.

- *A critical user journey (CUJ) describes a set of interactions a user/consumer has with a service to achieve some result.*

- Let’s imagine a web-based shopping site. A few of the actions our customers will be taking when they use the service:

  > Browse products for sale
  > Add items into their shopping cart
  > Check-out and purchase their selected items

- Each of these is a *critical user journey* for this service, and the reliability of each should be measured and tracked.

Service Level Indicators (SLIs)

Metrics that measure some aspect of CUJ service reliability

We recommend treating an SLI as the ratio of two numbers: *The number of good events divided by the total number of events*.

For example:

SLI Type	SLI Definition
Availability	Number of successful HTTP requests / total HTTP requests
Latency	Number of successful RPC requests that complete in < 100ms / total RPC requests
Quality	Number of search results that used the entire corpus / total number of search results, including those that degraded
Freshness	Number of “stock check count” requests that used stock data fresher than 10 minutes / total number of stock check requests

SLIs range from 0% to 100%, where 0% means nothing works, and 100% means nothing is broken.

System and infrastructure metrics are useful, but these are *not* SLIs and rarely have any direct bearing on user experience.

Service Level Objectives (SLOs)

The SLOs for your service are targets for one or more SLIs, aggregated over a specified time

	SLI definition	SLO over a four-week rolling window
Availability	The proportion of successful HTTP requests, as measured from the load balancer. Any HTTP status other than 500–599 is considered successful.	99%
Latency	The proportion of sufficiently fast requests, as measured from the load balancer. “Sufficiently fast” is defined as < 200ms, or < 1,000ms.	90% of requests < 200ms 99% of requests < 1,000ms
Correctness	The proportion of records injected into the state table by a prober that result in the correct data being read from the output table. A prober injects synthetic data, with known correct outcomes, and exports a success metric.	99.99% of records injected by the prober result in the correct output

Please do not be tempted to ever use the arithmetic mean in an SLI or SLO. The ‘average’ can be heavily skewed and will often misrepresent your user’s true experience. The 50th percentile, or median, is often a much better measurement.

Future Opportunities

Beyond SLIs and SLOs...

- Single pane of glass view of all critical CBA services and their reliability.
- Consumers of SLOs will include a wider audience than just service owners; SLOs help to reduce silos and provide a common language with which to discuss service reliability across the entire organization.
- SLIs and SLOs will inform data driven decision making: Monitoring, alerting and emergency response, demand forecasting and capacity planning, change management decisions, engineering prioritization.
- A good MTTR is directly related to how quickly you can detect and identify a problem's root cause (the mean time to detect, or MTTD). The longer it takes to identify a problem, the longer it will take you to restore the system to full operation. SLOs will reduce MTTR by shortening MTTD and focusing investigations on a particular CUJ: Targeted, automated health signals.
- Comprehensive SLIs and SLOs are an important foundation piece in the transition to broader DevSecOps & SRE best practices.

SLOs @ CBA: Observations

Some Observability Misunderstandings

1	**Obstack is not ready for Production use.**
	Obstack has been GA for over twelve months! **It's *the* place to define your SLOs**.
2	**You cannot put *any* metrics into Obstack that are not related to CUJs and SLOs.**
	We *really* want you to monitor your services with SLOs, but we won't question up to 20,000 timeseries in total, per application CI. But be wary of cardinality.
3	**You cannot monitor your service using logs.**
	We *really* want you to monitor your services with SLOs, and logs should ideally not be your sole source of observability, but they do have merit for certain events: • High frequency, low cardinality events are best stored as *metrics*. • However, *low frequency, high cardinality events are best kept as logs*.
4	**You cannot onboard to PagerDuty without CUJs and SLOs.**
	We *really* want you to monitor your services with SLOs, but there is minimal scrutiny of PagerDuty onboarding PRs, and *none* based on the existence of SLOs.

Cardinality

Thanks to Google Gemini…

In the context of monitoring and observability, metric cardinality refers to the number of unique timeseries generated by a metric and its associated labels (dimensions). Essentially, it's the count of distinct combinations of metric name and labels. A metric with high cardinality has many unique combinations, while a metric with low cardinality has fewer.

Timeseries:

- A time series is a sequence of data points collected over time, with each point representing a specific combination of metric name, labels and values.

Example:

- Imagine a metric called **`http_requests_total`**. With just the label **`status_code`**, a metric with a limited number of status codes (e.g., **200**, **404**, **500**) would have low cardinality. However, if you add labels like **`user_id`**, **`region`**, and **`url`**, the number of unique combinations would increase dramatically, resulting in high cardinality.

Note to Self: Deep Breath

Monitor for *Symptoms*, not *Causes*

CPU is not an SLI

```
- alert: something-app-prod-ECS-HighCPU-alert
  expr: (aws_ecs_containerinsights_cpu_utilized_average{account_id="3602611xxxxx", dimension_ClusterName="p-s-something-app-ecscluster"} /
         aws_ecs_containerinsights_cpu_reserved_average{account_id="3602611xxxxx", dimension_ClusterName="p-s-something-app-ecscluster"}) * 100 > 70
  for: 10m
  labels:
    cmdb_id: CI0001xxxxx
    severity: warning
  annotations:
    description: Alert - High CPU utilization on Something App ECS.
    summary: High CPU Utilization Detected on Something App ECS.
    message: CPU utilization on ECS has exceeded 70%. Immediate investigation is needed to prevent service degradation or outage.
    runbook_url: https://commbank.atlassian.net/wiki/spaces/Something/pages/12345/Runbook+for+Something+App+Tier+Alerts
```

Monitor for *Symptoms*, not *Causes*

CPU still isn’t an SLI

```
- alert: CpuUtilizationMaximum_PrdEgress
  expr: |-
    aws_ec2_cpuutilization_maximum{account_id="586150xxxxxx"} > 60
  for: 15m
  labels:
    alerter: blackbox
    severity: critical
    cmdb_id: CI0137xxxxx
  annotations:
    message: PRD Egress CPU Utilization is High (Greater than 60 percent) in last 15 minutes.
    runbook_url: "https://commbank.atlassian.net/wiki/spaces/something/pages/921315964/Runbook+-+Workflow+Hosting"
    description: |-
      PRD Egress High CPU Utilization
```

Monitor for *Symptoms*, not *Causes*

Filesystem

```
- alert: Something-EC2-HighDisk-alert
  expr: (aws_cwagent_disk_used_percent_average{account_id=~”xxxxxxxxxx", region=~"ap-southeast-2"}) > 80
  for: 30m
  labels:
    cmdb_id: CI0001xxxxx
    severity: warning
  annotations:
    description: Alert - High Disk utilization on AWS EC2.
    summary: High Disk Utilization Detected on AWS EC2.
    message: Disk utilization on EC2 has exceeded 80%. Immediate investigation is needed to prevent service degradation or outage.
    runbook_url: https://commbank.atlassian.net/wiki/spaces/Something/pages/8117xxxxx/Runbook+Something+High+Memory+on+all+EC2+instances
```

Monitor for *Symptoms*, not *Causes*

- Monitoring and alerting on system/platform metrics rarely gives any indication as to service health.

- Naïve threshold-based metric alerting is fragile, noisy and misdirected.

- Instead, we need to continuously monitor service health and functionality:
  > Alert on the *symptoms* of unreliability.
  > Do not look for *causes* and assume unreliability.

- Platform and system level metrics have there uses in problem diagnosis, capacity planning and trend analysis.
- They are not, however, a good proxy for measuring *system reliability*.

Q: *WHY ARE WE STILL DOING THIS? I have opinions…*

If 100% is wrong, can we please have 99.999%?

Users may expect at least 99.999% of DNS queries shall be successfully resolved by vhdns.Method: Probe existing records periodically using Blackbox prober. Consider it as successful as long as response != SERVFAIL.

If 100% is wrong, can we please have 99.999%?

Users may expect at least 99.999% of DNS queries shall be successfully resolved by vhdns.Method: Probe existing records periodically using Blackbox prober. Consider it as successful as long as response != SERVFAIL.

- Be careful what you wish for!
  - > Recall that 99.999% only offers 0.86s of downtime per day; 5.26 total minutes per year.
  - > One failed query from 100,000 is incredibly brittle.

If 100% is wrong, can we please have 99.999%?

Users may expect at least 99.999% of DNS queries shall be successfully resolved by vhdns.Method: Probe existing records periodically using Blackbox prober. Consider it as successful as long as response != SERVFAIL.

- Be careful what you wish for!
  - > Recall that 99.999% only offers 0.86s of downtime per day; 5.26 total minutes per year.
  - > One failed query from 100,000 is incredibly brittle.

- It's not possible to measure 99.999% via a probe. At a minimum, the probe would need to execute every 0.43s to have sufficient samples to measure this (see next slide).
- See https://uptime.is/99.999
- Obstack and the Blackbox Exporter don't provide 99.999% availability, and neither does the platform and network that they run on!
- Assuming significant levels of DNS queries, exposing query counters at a loadbalancer or from the binary *may* give sufficient data.

44,100 Hz

https://en.wikipedia.org/wiki/Nyquist - Shannon_sampling_theorem

The Nyquist Shannon sampling theorem states that to accurately record an event of duration ‘n’ you need to sample at a rate of at least ‘n ÷ 2’.

> A 99.5% SLO over a day allows 432s of downtime, requiring a minimum sample rate of 216s. ✅
> A 99.9% SLO over a day allows 86.4s of downtime, requiring a minimum sample rate of 43.2s. ✅
> A 99.95% SLO over a day allows 43.2s of downtime, requiring a minimum sample rate of 21.6s. ❓
> A 99.99% SLO over a day allows 8.64s of downtime, requiring a minimum sample rate of 4.32s. ❌
> A 99.999% SLO over a day allows 0.864s of downtime, requiring a minimum sample rate of 0.432s. ❌

When setting your SLOs, be aware of the data granularity required to support it!

Unless you can probe exceedingly fast or have a service with high transactions per second and a means of exporting accurate statistics, you will very likely not have sufficient data to calculate your SLO adherence.

Or more simply, *you will miss events*.

My SLO only applies when my system is being used!

- SLOs, and the Sloth framework utilised within Obstack, are designed to track continuous 24-hour availability.

- Many CBA systems have historically only offered ‘business hours’ availability.
- Many more have well defined periods of scheduled, sustained unavailability (maintenance).

Q: *Should SLOs only apply during documented periods of expected, scheduled availability?*

- The SRE purist answer is that SLOs should be continuous and inclusive of maintenance events, etc.
- The pragmatic answer might be that today CBA does have systems that operate within timebound limits, and SLOs at CBA might need to accommodate these (for now)?

> SPOILER: Obstack and Sloth have *no* support for non-24-hour SLOs.

Thank-you!

Will Charles, Principal Engineer, Core SRE

will.charles@cba.com.au

http://edu.sre.cba/contact/contact-us and http://chat.sre.cba/

Commonwealth Bank