SLOs and SLIs in the Real World: A Deep Dive

Elisa Binette @elisabPDX
Matthew Flaming @mflaming
<table>
<thead>
<tr>
<th>Service Level Indicator</th>
<th>Service Level Objective</th>
<th>Service Level Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>X should be true…</td>
<td>Y proportion of the time</td>
<td>or else…</td>
</tr>
<tr>
<td>“10 key takeaways about SLIs delivered in 20 minutes”</td>
<td>99.9% of the time</td>
<td>We’ll mail you a Wookie</td>
</tr>
</tbody>
</table>
“Data is being collected properly, and customers can login to the system and view their data 99.9% of the time”
System Boundaries

Login Service:
Authenticate user credentials
A System of Systems

UI/API Tier

Login Service

Data Storage/Query Tier

Legacy Data Tier

Data Ingest & Routing
SLIs + SLOs: A Simple Recipe

1. Identify system boundaries
2. Define capabilities exposed by each system
3. Plain-English definition of “available” for each capability
4. Define corresponding technical SLIs
5. Start measuring to get a baseline
6. Define SLO targets (per SLI or per capability)
7. Iterate and tune

@elisabPDX   @mflaming
Data Ingest Tier
Multiple capabilities
- Data ingested
- Data routed
One (or more) SLIs per Capability

Data Ingest SLI
Percent of well-formed payloads accepted

Data Ingest SLO
99.9%

Data Routing SLI
Time to deliver message to correct destination

Data Routing SLO
99.5% of messages in under 5 sec

@elisabPDX  @mflaming
Choosing SLO targets

SLO numbers need to be:

- What the team *actually* commits to supporting
- What the org *actually* commits to supporting
- Reflective of technical reality

SLOs represent an ongoing commitment!

When in doubt, measure first
SLIs Act as Broad Proxies for Availability

Horizontally Scaled Data Tier

Single Capability
  • Query data

Multiple SLIs
  • Latency
  • Correctness/error rate
SELECT percentage(count(*), where error = 'true') FROM Query SINCE 10 minutes ago

Since 10 minutes ago

0

Percentage
Compound SLOs

99.95% of well formed queries will receive well formed responses

99.9% of queries will be answered in less than 1000ms
SLIs/SLOs for Hard Sharded Systems

Hard Sharded Legacy DBs

Single Capability
- Query performance

Multiple SLIs
- Latency
- Correctness
- Freshness

ACME MONITORING PRODUCT
Sharded vs. Horizontally Scaled SLOs

**Horizontally Scaled**
- SLO: 66%

**Hard Sharded**
- SLO: 0%
- SLO: 100%
- SLO: 100%
Defining SLIs/SLOs for Core Infrastructure

Container Orchestration/Runtime

Network
Ask the Customer!

What do you use this system for?

What kinds of guarantees would you like to see?

What assumptions do you make in your code?
Hard Dependencies Require Higher SLOs

Network
Multiple Capabilities
- Load balancing
- Intra-AZ routing
- Inter-AZ routing

Multiple SLIs
- Load balancer endpoint uptime
- Intra-AZ latency/packet loss
- Inter-AZ latency/packet loss

One SLO per capability
- 99.99% goal
Capabilities Clarify Contracts

Container scheduler/cluster
Single Capability
• Run jobs with expected resources

Multiple SLIs??

“Accepted jobs will run with the quota of requested resources available to them 99.99% of the time.”

Container Orchestration/Runtime
“Accepted jobs will run with the quota of requested resources available to them 99.99% of the time.”

**Expected resources available to jobs**
- CPU and memory quotas FTW
- Network saturation is possible
“Accepted jobs will run with the quota of requested resources available to them 99.99% of the time.”

Jobs in runnable state 99.99% of time
- Potential uptime vs. job correctness

```c
/*
 * Describes possible task states. IMPORTANT: Mesos assumes tasks that
 * enter terminal states (see below) imply the task is no longer
 * running and thus clean up any thing associated with the task
 * (ultimately offering any resources being consumed by that task to
 * another task).
 */

enum TaskState {
    TASK_STAGING = 6; // Initial state. Framework status updates should not use.
    TASK_STARTING = 0; // The task is being launched by the executor.
    TASK_RUNNING = 1;
    TASK_KILLING = 8; // The task is being killed by the executor.
    TASK_FINISHED = 2; // The task finished successfully on its own without external interference.
    TASK_FAILED = 3; // TERMINAL: The task failed to finish successfully.
    TASK_KILLED = 4; // TERMINAL: The task was killed by the executor.
    TASK_ERROR = 7; // TERMINAL: The task description contains an error.
    TASK_LOST = 5; // The task failed but can be rescheduled.
    TASK_DROPPED = 9; // TERMINAL: the task failed to launch because of a transient error.
    TASK_UNREACHABLE = 10; // The task was running on an agent that has lost contact with the master
    TASK_GONE = 11; // TERMINAL. This can occur if the agent has been terminated along with all of its tasks
    TASK_GONE_BY_OPERATOR = 12; // The task was running on an agent that the master cannot contact
    TASK_UNKNOWN = 13; // The master has no knowledge of the task.
};
```
Don’t Forget the Big Picture

Overall
Multiple Capabilities
• Collect data
• Login
• View data

Single dumb SLI
• Does sample workflow succeed

Allows us to sanity check individual system SLIs/SLOs
Customer Specific SLOs
Recap

1. Worry about SLIs more than SLOs
2. Start with plain English descriptions of availability, not with technical underpinnings
3. Define SLIs and SLOs for specific capabilities at system boundaries
4. Each logical instance of a system (e.g., hard shard) gets its own SLO
5. SLIs are not the same as alerts, and are not a replacement for thorough alerting
6. "AND" together SLIs for a given capability into a single SLO for that capability
7. Write down your SLI/SLO contracts and share them
8. Key customers may need their own SLOs/SLIs
9. Assume SLOs and SLIs will both evolve over time
10. SLOs represent an ongoing commitment
<table>
<thead>
<tr>
<th>Service Level Indicator</th>
<th>Service Level Objective</th>
<th>Service Level Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>“10 key takeaways about SLIs delivered in 20 minutes”</td>
<td>99.9% of the time</td>
<td>We’ll mail you a Wookie</td>
</tr>
</tbody>
</table>
Thank you

Elisa Binette
@elisabPDX

Matthew Flaming
@mflaming