SITE RELIABILITY ENGINEERING

A practical guide

Abstract

This document intends to provide the concepts and guide to implement Site Reliability Engineering in software projects

Varun Kumar

varkum@microsoft.com

Table of Content

Part 1: Understanding SRE

What is SRE?

Site reliability engineering (SRE) creates a bridge between development and operations by applying a software engineering mindset to infrastructure and operations problems.

"Fundamentally, it's what happens when you ask a software engineer to design an operations function."

--Ben Treynor Sloss, VP Google Engineering, founder of Google SRE

In general, a Site Reliability Engineer is responsible for the capacity planning, availability, performance, monitoring, latency, change management, and emergency response for the platforms and services.

SRE Culture

1. Reduce Organizations Silos
   a. SRE shares ownership/responsibilities of production with developers.
   b. Use same tooling to make sure everyone has same views and same approach working with production.
2. Accept Failure as Normal
   a. Accept Failure as Normal by introducing the concept of Error budget, that how much a system can go out of specs.
   b. SRE has postmortems to make sure the failures happen in production systems don't happen the exact same way more than once.
3. Implement Gradual Change
   a. Roll out things to a small percentage of fleet before moving it for all users
4. Leverage Tooling and Automation
   a. Try to eliminate or at least reduce manual work (toil) as much as possible.
5. Measure Everything
   a. Measuring toil and the metrics of the system that are important for business to run as expected.

The Six TTX metrices

Time to Detect (TTD)

The time from the impact occurred to the time it is detected or reported. This metric determines the quality and accuracy of the monitoring ins place. The TTD helps in establishing the right balance between the monitoring sensitivity (to find all of the customer issues quickly and accurately) and response to incident (for issues that actually impact customers).

TTA (Time to Acknowledge)

The time taken to acknowledge the problem once it is detected.

Time to Engage (TTE)
The time from detection until the appropriate engineer is engaged. This metric is important to understand the effectiveness of mobilizing of response. It includes the triage time (determining severity and ownership) as well as the time to escalate and onboard the engineer to fix.

Time to Mitigate (TTM)
The time taken by the engineer to identify and mitigate the issue say, a work around for business continuity.

Time to Resolve (TTR)
The time taken by the team to do the postmortem and complete all the action items for permanent fix of the issue.

Time to Failure (TTF)
The time for which the service worked successfully before the occurrence of another failure i.e. the time in between the failures.

What is Toil?
Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows
SRE propose to keep operational work below 50% of time and other 50% of time should be spent on engineering project work such as adding new service features. If the toil is not reduced, it becomes difficult to achieve this goal. Moreover, repetitive manual work increases TTD and TTR and increase the manual fault rate.

Some examples of toil could be:

- Manually running a script (even if the script executes some automated tasks).
- Manual scaling based on parameters such as traffic, volume or user count.
- Work performed repeatedly.
- Tasks that could be completed by a machine with no human intervention required
- Actions that does not permanently improve service.

The steps to measure and reduce toil is available in the implementation section of document.

Measuring Service behaviors
To manage a service accurately, it is of paramount importance to understand the behaviors of the service that are critical for the business and how to measure and assess those behaviors. It is important to choose appropriate metrics to measure the service.

Service Level Indicators (SLIs)

Metric that provides the measure of the level of service provided

Service level indicator is a carefully defined quantitative measure of some aspect of the service. The measurements are often collected over a measurement window and aggregated into a rate, average, or percentile.

$$\text{SLI} = \left[\frac{\text{Number of Good events}}{\text{Total number of Events}} \right]$$

Defining a SLI

1. Chose application and services for which SLO needs to be defined.
2. Identify the features, activities and processes critical for the business.
3. Identity and classify target set of users to the critical components.
4. Measure and define the aspects of the application important to users.

Some examples of SLIs:

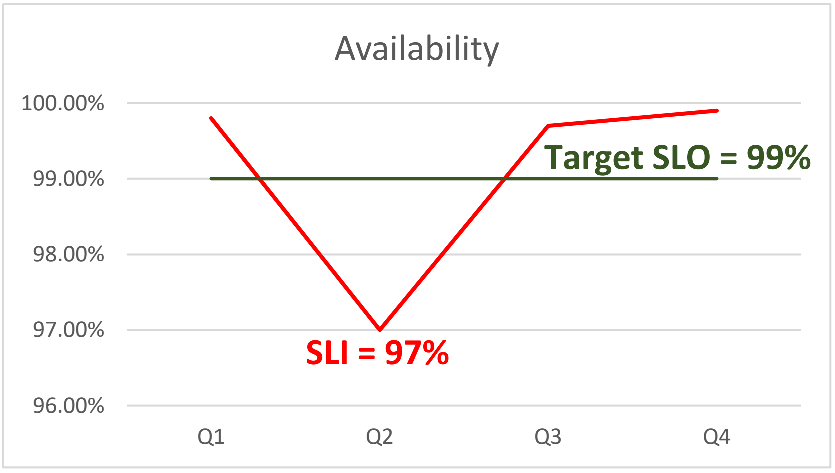
1. Success Rate: Number of successful requests / total number of requests
2. Error Rate: Number of failed requests / total number of requests
3. Request Latency: Number of requests completed successfully in < X ms / total number of requests.
4. Correctness: Number of requests with correct data/ total number of requests
5. Availability: Time for which service is usable / total time window of monitoring

Service Level Objectives (SLOs)

The target value for a service level and the performance of SLI against it over a period

Service level objective is a target value or range for a service level that is measured by an SLI. SLO helps to make data-driven decisions and define what work to be prioritized- new features or improving reliability.

SLI ≤ target, or lower bound ≤ SLI ≤ upper bound



It is important to understand the 100% is a wrong target as it's not possible to achieve. Therefore, it is wise to target near-100% which is practically achievable. Such as availability, it is not a practical approach to target 100% availability. The system could be down for some maintenance, failure or due to some external factors beyond control. So how can we define a SLO? What would be the characteristics of a good SLO?

Good SLO

A good SLO is one that

- Meets the service reliability goals

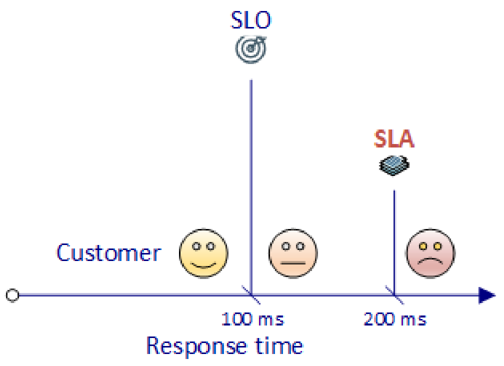
- Is derived from the user needs.
- Has considered the current performance of the service.
- Is approved by all the stakeholders in the organization.
- Is ambitious but achievable under normal circumstances.
- Can be achieved consistently over a period of time.
- Has a process in place for review and redefine?

SLO defining is an Iterative process

Running a service with an SLO is an adaptive and iterative process The SLI and SLO defined initially may or may not be correct or may not be valid after a period. It could be because of some new feature or change in existing or could be because the pre-mature SLO defined initially that need re-visiting and many more. Therefore, it is important to set up a review process to improve.

Service Level Agreements (SLAs)

Service level agreements are the contract with the customer that specify the service level targets and the consequences of missing those targets. SLAs are tied to the business decisions and SRE doesn't typically take part in constructing SLAs. However, SRE does helps to avoid consequences of missed targets.



Error Budget

Error budget is the mechanism to quantify allowed unreliability

An error budget is the inverse of reliability. It defines how unreliable the service is allowed to be. This unavailability can a result of some planned/unplanned maintenance, hardware or infrastructure failure, bad release rollouts etc. If the error budget is spent in full, the service freezes changes (except for emergency releases)
For example: If SLO is defined as 99.9 % request success per quarter, then error budget states that maximum 0.1 percent of requests can fail in the given quarter.

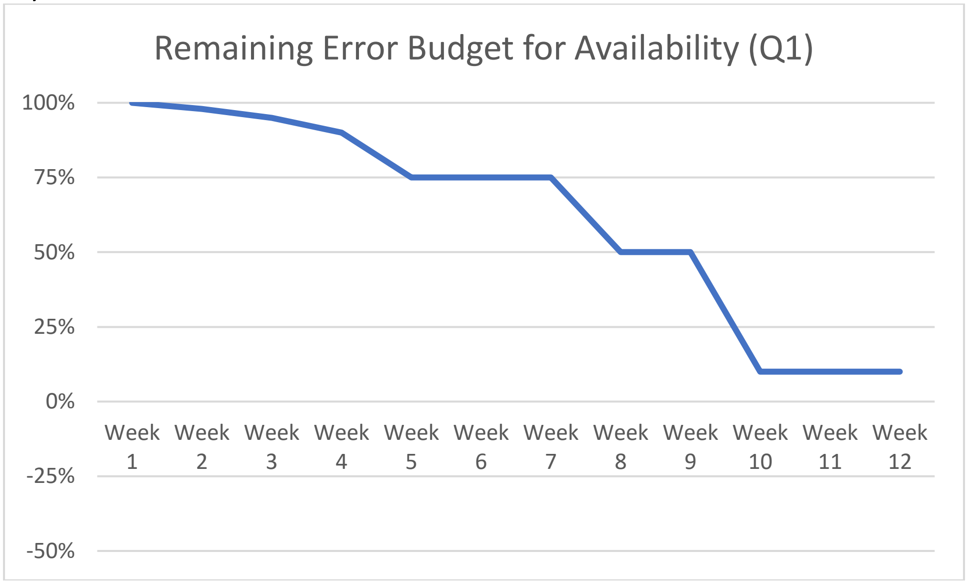
0.1% Unavailability means

0.1 / 100	(Percentage)
X 91 days	(Quarter)
X 24 hours	(day)
X 60 mins	(hour)

131.04 minutes of downtime per quarter.

which is just enough time for a monitoring system to surface an issue, and support team to investigate and resolve. And that too for not more than single incident per month.

It's important to keep a track of the error budget spent to prevent overspending as well as to plan releases as per the availability.



The graph shows that 90% error budget for availability in Quarter Q1 has been exhausted. That suggests that the system was unavailable for ~118 mins in the quarter.
The error budget is defined to be consumed. It doesn't matter what percentage of error budget we consume as long as it does not exceed 100%. However, It does help to plan the releases so that we don't break the availability further.

Cost Factor

Higher reliability is exponentially proportional cost. Higher stability limits and velocity of feature release increases cost dramatically. It is said that adding each 9 to the SLO increase cost to almost 10 times. That means from 99.9% to 99.99% will require X10 of the cost. The factors that affects cost are:

The cost of redundant machine/compute resources

The cost associated with redundancy for high availability of the system for load and guarantee delivery in case of region down or some unfortunate mass outage.

The opportunity cost

The cost borne by an organization for engineering resources to build systems or features

Cost of unreliability

Service failures can have many potential effects, including loss of revenue, financial penalty against SLA, user dissatisfaction, harm, or loss of trust; direct or indirect revenue loss; brand or reputational impact.

Postmortem Culture

Postmortem culture is a truly blameless process of learning from the incidents and preventing repeat outages because of the same cause. In other words, it revolves around the processes and tools rather than the people to put blame on. Introducing postmortems into an organization is as much a cultural change as it is a technical one.

A postmortem is a written record of an incident, its impact, the actions taken to mitigate or resolve it, the root cause(s), and the follow-up actions to prevent the incident from recurring.

How postmortems are different from the Root Cause Analysis (RCA)?

Postmortem, unlike RCA doesn't focus only on the root cause and resolution of the particular incident, instead, it involves deep diving of the problem and actions to prevent repetition: Some of the aspects it involves are:

- The Impact of the Incident in terms of business and value.
- What triggered the incident?
- How was the incident detected or wasn't detected at all?
- Details logs of the steps taken to resolve or mitigate the Incident.
- What were the assumptions made?
- What were the expectations that didn't met?
- The observation during Incident mitigation or resolution.
- The plan of action to prevent this incident from re-curing and better detection.

Postmortem criteria

The postmortem indeed is a costlier process in terms both time and effort, So it is important to define in what type of incidents it must be executed. Some of the postmortem triggers may include:

- Degradation beyond a certain threshold
- Production Data loss
- Steps performed for mitigation and not resolution,
- Longer resolution or mitigation time
- A detection/monitoring failure.

Part 2: Implementing SRE

Implementing SLI

1. **Identify and document right SLIs**
   Not every metric in the monitoring system can be SLI. It is important to understand the metrices from the business point of view. Choosing too many SLIs creates noise and diverts attention from the indicators which are important.

2. **Collect Indicators**
   Collect metrices defined by the SLI from all the sources such has monitoring system on server or client-side instrumentation to get the thorough insights.

3. **Aggregate Indicators**
   Aggregate the metrices for a defined time period (measurement window) for better understanding.

4. **Standardize Indicators**
   A standard definition for the SLI should be followed so that they are not left for individual interpretations.

Implementing SLO

1. **Defining and document SLO**
   SLOs should specify how they're measured and the conditions under which they're valid. It may be appropriate to define separate objectives for each class of workload:

   - 90% of Type A request will be completed in 1 ms.
   - 99% of Type A request will be completed in 10 ms.
   - 99% of Type B request will be completed in 20 ms.

   Critical Dependencies
   A service cannot be more available than the intersection of all its critical dependencies. If the aim for the service is to offer 99.99 percent availability, then all the critical dependencies must be significantly more than 99.99 percent available.

   Frequency, detection time, and recovery time
   A service cannot be more available than its incident frequency multiplied by its detection and recovery time. For example, three complete outages per year that last 20 minutes each result in a total of 60 minutes of outages. Even if the service worked perfectly the rest of the year, 99.99 percent availability (no more than 53 minutes of downtime per year) would not be feasible.

2. **Measure SLO**
   Monitor and measure the SLIs and compare it with the SLOs to keep a track.

3. **Respond**
   If the intermediate thresholds or SLO is breached, take necessary actions required to bring the back to normal.

4. **Validate and Improve SLO**
   Validate the SLOs based on the business feedback, postmortems and SRE feedbacks and re-define if required.

Error Budget expenditure

1. **Define ad document measuring window**
   A measuring window need to be defined to measure and track error budget spent so that decisions can be made for reliability or feature development. This window can be defined as weekly, monthly, quarterly, yearly etc. It depends upon the system size and criticality, team structure, maturity of SRE etc.

2. **Define and document Error budget policy**
   It is very important to define an Error budget policy to track the consumption of error budget and to take actions at various level of consumptions to mitigate the risk of exhaustion and/or plan releases accordingly. A typical error budget policy contains:
   a. The trigger for the error budget policy i.e. when it takes effect for example spending X% of error budget in just a week or on a single outage.
   b. The various thresholds of budget spent.
   c. The actions to be taken on reaching the thresholds such as re-prioritization of work.
   d. Consequences of breaching threshold consecutively in multiple measuring windows such as giving control back to dev team to improve reliability.
   e. Decision makers for change in scenarios as defined in the policy.
   f. Acceptance of policy to be followed by all the teams: Dev, SRE, managers and leadership.

   Example of an Error budget policy

   Threshold 1: Automated alerts sent to team for the SLO's in risk
   Threshold 2: SREs include dev team.
   Threshold 3: The 30-day error budget is exhausted and the root
   cause has not been found; feature releases blocked, dev team
   dedicates more resources
   Threshold 4: The 90-day error budget is exhausted and the root
   cause has not been found; SRE escalates to executive leadership
   to obtain more engineering time for reliability work

3. **Reduce critical dependencies**
   Critical dependencies should be reduced to save error budget expenses. It's not practically feasible to get rid of all critical dependencies in the large systems. However, some best practices around system design can be followed to optimize reliability. The most common strategies to reduce critical dependencies is to eliminate Single points of failures, redundancy, automatic failover and fallback, fast and reliable rollback.

Reducing toil

1. **Identify and Measure Toil**
   Follow a data-driven approach to identify and evaluate sources of toil, make objective remedial decisions, and quantify the time saved by toil reduction projects.

2. **Cost benefit analysis**
   It's important to analyze return on investment i.e. cost versus benefit to confirm that the time saved by reducing toil will be more than the time invested in development and maintenance of automated solution.

3. **Assess Risk**
   Automation can save a lot of human efforts but may cause side effects under wrong circumstances such as automation script failure, system restart etc. It's important to carefully access those risks before implementing automation.

4. **Reject the Toil**
   If the Cost of responding to toil or effort required to reduce toil is more than the business outcome from the tasks, then the toil-intensive task should be rejected.

5. **SLO driven toil reduction**
   If ignoring toil doesn't consume or exceed the service's error budget, then It would be better to use the engineering effort on other productive activities.

6. **Reduce toil**
   The toil can be reduced or eliminated at two levels:
   6.1 At Source: Identity the source where the toil is generated and figure out if it's feasible to change the system to reduce or eliminate the toil.
   6.2 During Operations: Automating the tasks that are performed manually by the operations teams.

   Approaches to reduce toil

   - **Baby steps**
     For complex system, implement partially automation and incrementally move toward full automation. Engineers in this approach may still handle some of the resulting operations until it is completely automated.

   - **Self-Service**
     Provide an option to users, wherever possible, to resolve the most common issues without the need to contacting the operations team.

   - **Automate Toil Response**
     Once the process is thoroughly documented, try to break down the manual work into components that can be implemented separately and used to create a composable software library that other automation projects can reuse later.

   - **Use Open Source and Third-Party Tools**
     Do not re-invent the wheel. It's not necessary to develop everything from scratch. Look for opportunities to use or extend third-party or open source libraries to reduce development costs.

   - **Feature development**, which is focused on improving reliability, performance, or utilization, which often reduces toil as a second-order effect.

7. **Use Feedback to Improve**
   Feedback is an important part of improvement of any process. Seek productive feedback from people interact with the automation tools, scripts, documents etc and try to optimize based on those.

Postmortem Template

Date: postmortem Date

Authors: The author of the postmortem report

Status: Current Status (Eg: Complete, action items in progress)

Summary: The summary of what happened. (Eg: some Cririal service down for x no of hours, effecting y users, because of z release)

Impact: Business and technical impact (Eg: Data lost, revenue loss etc)

Root Causes: Why it happened, This include the technical cause of the issue as well the business cause defining we lag proactiveness.

Trigger: What triggered the incident (eg: sudden in Increased traffic, hardware/infra failure, some edge case)

Resolution: What measures and take to resolve the issue (Redirected traffic to warm standby instance, Added capacity to existing instance etc)

Detection: Monitoring team detected high volume of request failures.

Action Items:

Action Item	**Type**	**Owner**	**ADO Work item ID**
To update playbook with instructions for responding to cascading failure	mitigate	someone	
Setting Alerts on the particular server for the specific case	prevent	someone	
Freeze new releases on production until x date due to error budget exhaustion	other	someone	

Lessons Learned

What went well

- Monitoring quickly alerted us to high rate (reaching ~100%) of HTTP 500s

What went wrong

- We're out of practice in responding to cascading failure
- We exceeded our availability error budget (by several orders of magnitude) due to the exceptional surge of traffic that essentially all resulted in failures

Timeline

2015-10-21 **(all times UTC)**

1. 14:53 Traffic increases by 100x after post ….
2. 14:54 **OUTAGE BEGINS** — the service stopped responding to requests
3. 14:57 All traffic to service failed: [Monitor URL]
4. 14:58 XYZ started investigating.
5. 15:01 **INCIDENT BEGINS** XYZ created incident #123 due to cascading failure
6. All steps taken to investigate
7. 15:36 **OUTAGE MITIGATED**, initiated the disaster recovery server
8. All steps taken to mitigate

Supporting information:

- Monitoring dashboard URL

How to improve Reliability?

To improve reliability, TTX metrices need to be reduced to reduce the impact of the outage.

1. To improve the time-to-detect, monitoring system should be implemented to catch outages faster and send automated alerts instead of relying on people to notice abnormalities.
2. The monitoring system should SLO compliance along with the metrics like your SLIs, active users, performance counters etc.
3. To improve time-to-resolution, develop playbook to define actions need to be taken to respond to an outage instead of figure out everything from scratch by the team.
4. Task such as redirecting traffic to secondary nodes should be automated while investigating the incidents.
5. Reduce impact percentage by limiting the number of users impacted by a change. For example, rolling deployment, where a new feature is released in multiple stages or to incremental nodes instead of pushing entire release to all the users at once.
6. Impact can also be reduced by engineering your service to run in a degraded mode during a failure. For example, to allow read-only operations during outage. If most users perform read operations, then this may mitigate the impact of an outage.
7. Increasing time to failure such as running the service in multiple failure domains and automatically directing traffic away from a zone or region that has failed.

Tools

1. **Alerting and Monitoring**

   The monitoring system used by the SRE team should have following capabilities:

   - Option to select metrices

     The system should allow, what metrics need to be collected based on the critical services defined by the business

- Data Freshness
  The monitoring system should page metrices as soon as possible.

- Speed of retrieval
  It should allow to query and analyze vast volume of data quickly for the faster resolution,

- Retention of data
  The system should retain data for fairly longer period of time to provide long-term view of your data and allow analysis of trends with system growth.

- Dashboards
  It should allow you to concisely display time-series data in graphs, and to structure data in tables or a range of chart styles for better analysis.

- Alerts
  Monitoring system should allow to configure alerts on various metrices so that the team can be notified in case of some event happened that requires attention.

2. **Postmortems Repository**
   There should be a central postmortems repository to store postmortems for analysis and feedback. The repository should provide an efficient way of managing and tracking the Action items, update TTx metrices, timelines, lesson learnt, and other important information related to the Incident. The repository should also have an option to search and filter on various criteria to learn from the previous incidents and the approach taken.

3. **Runbooks**
   Create automated runbooks to perform the actions to detect, mitigate and/or resolve the issues wherever possible. If it's not possible to create automated or are costlier than its benefits, then create manual runbooks to help person performing the task(s) to follow a guided steps for a particular Incident or outage, instead of exploring everything from scratch.

Tool URLs

1. Postmortem ADO template
   https://dev.azure.com/CloudAvenue/CloudAvenueDemo/_workitems/edit/8

2. SRE Calculators
   https://srecalculator.azurewebsites.net/

References

1. What is ‘Site Reliability Engineering’? [video]
   https://landing.google.com/sre/interview/ben-treynor-sloss/

2. Site Reliability Engineering [book]
   Edited by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy
   https://landing.google.com/sre/books/

3. The Site Reliability Workbook [book]
   Edited by Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara and Stephen
   https://landing.google.com/sre/books/

4. The Calculus of Service Availability [whitepaper]
   https://queue.acm.org/detail.cfm?id=3096459

5. Business Monitoring: If You Can’t Measure It, You Can’t Improve It [blog]
   https://www.anodot.com/blog/business-monitoring-incidents-cycle/

6. awesome-sre [github]
   https://github.com/dastergon/awesome-sre

7. post-mortems [github]
   https://github.com/danluu/post-mortems#config-errors

8. Availability Calculator [github]
   https://github.com/dastergon/availability-calculator

9. Fundamentals of an error budget policy [video]
   https://www.coursera.org/lecture/site-reliability-engineering-slos/fundamentals-of-an-error-budget-policy-EMoEZ