Improving system reliability and reducing downtime is a critical objective for organisations across industries. It involves implementing strategies and technologies to ensure that systems operate consistently, efficiently, and without interruption.

We start by determining system criticality through the introduction of Service Level Objectives (SLOs), Service Level Indicators (SLIs), and Service Level Agreements (SLAs).

SLA is a contract between a service provider and a customer that defines the level of service expected from the service provider. It outlines the metrics used to measure service performance, as well as the responsibilities of each party.

SLO emphasis on a particular metric or performance standard of a system to meet the SLA

SLI is the actual indicator of how the system has been performing.

In a simplistic view, let's say we have an identity provider system (SSO) that manages single sign on for app applications. Given that it is a tier 0 application that needs to be up all the time. That would be the SLA.

Then the SLO would be 99.99% or 99.999% percentage of uptime. How well the system performs is SLI.

Next we measure system reliability through MTBR (Mean Time Between Repairs)and MTBF (Mean Time Between Failures). MTBR is how frequently repairs are needed. MTBF on the other hand is how long the system keeps operating before it falls over.

Once we have defined this, we can proceed to analyse and define the core metrics that are critical of reducing MTBF, then slowly expand out to cover external dependencies.

# Alerting and monitoring

**Core Application health**

Application health is an important metric that allows us to determine if the application is working well. In this case, it would be memory and cpu. So the alerting and auto scaling would be based on these to start off with. For example, if the cpu exceeds 70% then we would need to start scaling out another instance.

**System Dependencies**

Then we expand this view a little outwards, by looking at the system dependencies and this could be a database and external cache. This also means our metrics would cover

**Database**

* Connection number
* Connection success / errors
* Database availability
* Storage used percentage

Cache

* Memory current allocated vs used

**External Dependencies**

Given that it is an Identity Provider, it often integrates with external identity services such as Gigya or Auth0. We can build metrics around how many percentages of requests that are successful and those that failed. We can proceed to create an alert when there’s 5% of failure over a 5 minute interval. This is considered a high severity incident because it may require manual intervention from service providers to start investigations or to increase rate limits.

In terms of tooling, we can use AWS cloudwatch for monitoring and alerts. Monitoring hare involves creating a dashboard for metrics defined above.

# Reliability

### Autoscaling

We can also focus more on determining when an action will be taken if metrics exceed certain thresholds. These actions can be automated for example, when there’s too many requests coming into the server, we can turn on auto scaling which automatically scales and helps with load distributions. Depending on what type of resources, an example of this would be AWS EC2 autoscaling group or dynamodb autoscaling.

### Availability zones

A key consideration for reliability is configuring services to support availability zones. Resources in the cloud do go down and the system needs to be able to be fully operational.

### Backup

Backups are also important to ensure system availability. It can be crucial when we would like to recover our system to a specific point in time, In the case of crowdstrike issue - EC2 windows instances can be restored to a point before the patch occurred.

### Playbook

It is good practices to have a playbook which is sort of like a framework for handling specific scenarios, incorporating multiple runbooks and outlining roles, responsibilities, and communication plans

### Application resilience

Resilient applications that design for fault tolerance and reliability are also crucial. System can be highly available and resilient but if the application is not able to take advantage of this, then it might not be ideal.

CI/CD

It is important to have proper CI/CD in place so we can automate deployment process especially when we need to rollout hot fixes.

In short, when architecting systems, we need to determine the importance of this system and ensure that systems are configured to be able to autoscale, support availability zones, and have proper backup configured. On top of that, applications need to be resilient and proper monitoring/alerts need to be in place.