**Explore and document different reliability requirements (e.g Autoscaling, load**

**Balancing, Cloud-based and Serverless deployments, fault-tolerance, recoverability):**

**Reliability Requirements:**

Reliability is an important non-functional requirement for most software products so a software requirements specification (SRS) should contain a reliability requirement, and most do. But, one of our indicators of the quality of a ‘good’ requirement is that it is testable, so it is reasonable to ask whether the reliability requirements in a SRS are testable as written. Moreover, they are typically part of a technical specifications document. They can be requirements that a company sets for its product and its own engineers or what it reports as its reliability to its customers. They can also be requirements set for suppliers or subcontractors.

Essential elements of a reliability requirement:

* Measurable:

Reliability metrics are best stated as probability statements that are measurable by test or analysis during the product development time frame.

* Customer usage and operating environment:

The demonstrated reliability goal has to take into account the customer usage and operating environment. The combined customer usage and operating environment conditions must be adequately defined in product requirements.

* Time:

Time could mean hours, years, cycles, mileage, shots, actuations, trips, etc. It is whatever is associated with the aging of the product.

* Failure definition:

The requirements should include a clear definition of product failure.

* Confidence:

A reliability requirement statement should be specified with a confidence level, which allows for consideration of the variability of data being compared to the specification.

Example: Web Hosting

Suppose, a web hosting company is claiming that they will provide at least 10,000 monthly visits and 100 GB storage for a website like t4tutorials.com. We will try it for one month and if their claim is true, and the hosting really supports the mentioned numbers, then we can say that the hosting is reliable hosting. Otherwise, we can say that hosting is not reliable.

Example: The RQ Website probability of failure on demand (POFOD) shall be 0.0001 (1 out of 10000 plays) when a student requests to play a course video.

**Autoscaling:**

Auto scaling is a cloud computing technique for dynamically allocating computational resources. Depending on the load to a server farm or pool, the number of servers that are active will typically vary automatically as user needs fluctuate. Core autoscaling features also allow lower cost, reliable performance by seamlessly increasing and decreasing new instances as demand spikes and drops. As such, autoscaling provides consistency despite the dynamic and, at times, unpredictable demand for applications. The overall benefit of auto scaling is that it eliminates the need to respond manually in real-time to traffic spikes that merit new resources and instances by automatically changing the active number of servers. Each of these servers requires configuration, monitoring and decommissioning, which is the core of autoscaling.

For instance, when such a spike is driven by a distributed denial of service (DDoS) attack, it can be difficult to recognize. More efficient monitoring of auto scaling metrics and better auto scaling policies can sometimes help a system respond quickly to this issue. Similarly, an auto scaling database automatically scales capacity up or down, starts up, or shuts down based on the needs of an application.

Relation between Auto Scaling and LoadBalance:  
Auto scaling and load balancing are related because an application typically scales based on load balancing serving capacity. In other words, the serving capacity of the [load balancer](https://avinetworks.wpengine.com/software-load-balancer/) is one of several metrics (including cloud monitoring metrics and CPU utilization) that shapes the auto scaling policy.

Some cloud vendors that provide auto scaling capabilities include:

* Amazon Web Services (AWS). AWS has multiple auto scaling services, including AWS Auto Scaling and Amazon EC2 Auto Scaling. [AWS Auto Scaling](https://www.techtarget.com/searchaws/definition/AWS-Auto-Scaling) is a service for users who need to scale resources across multiple AWS services. In contrast, the Amazon EC2 Auto Scaling service is focused on providing auto scaling capabilities for Amazon EC2 instances that provide virtual compute resources.
* Google Compute Engine (GCE). GCE provides auto scaling capabilities as a feature for its cloud users running managed instance groups of virtual machine (VM) instances. Managed instance groups are an optional deployment approach on [Google Compute Engine](https://www.techtarget.com/searchaws/definition/Google-Compute-Engine), where groupings of identical virtual machines are deployed across GCE in a managed approach to enable higher availability.
* IBM Cloud. IBM has a module known as cluster-autoscaler that can be deployed on IBM Cloud workloads. This autoscaler can increase or decrease the number of nodes in a cluster based on the sizing needed as defined by scheduled workload policies.
* Microsoft Azure. For users of the Microsoft Azure cloud platform, the Azure AutoScale service enables automatically scaling resources. Azure AutoScale can be implemented with VM, mobile and website deployments.
* Oracle Cloud Infrastructure. Oracle has multiple auto scaling services across its Oracle Cloud Infrastructure platform, including Compute Autoscaling and a flexible load balancer that enables elastic load balancing for network traffic.

Open source Projects:

Remit/AutoScaling Stimulator:

* Multiverse is an autoscaling simulator and the core of the toolbox. It simulates auto scaling both on the application and the VM cluster level. Multiverse attempts to accurately simulate the application as a network of services with buffers.
* Stethoscope is a simulation data visualization tool. The tool leverages Python's matplotlib package to produce two categories of plots. The first category represents the quality of an autoscaling policy as experienced both by the user and the application owner. The second category characterizes the auto scaling behavior, i.e. an impact of an autoscaling policy on the internal application state.
* Cruncher is an auto scaling simulation automation tool that leverages the simulation capabilities of Multiverse and the visualization options offered by Stethoscope. This automation tool arranges concrete simulations to run based on the alternative configuration files provided to it.
* Praxiteles is a tool that generates sets of configuration files for autoscaling simulations based on meta-configuration files (so-called recipes), cloud provider traces, and aspectual models such as application topology model.
* Training Ground is a tool that helps to get the justified results for the short simulations that use online learning-based models in their autoscaling policy.

**Load Balancing:**

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool. Modern high‑traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost‑effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers. A [load balancer](https://www.nginx.com/solutions/adc) acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

* Distributes client requests or network load efficiently across multiple servers
* Ensures high availability and reliability by sending requests only to servers that are online
* Provides the flexibility to add or subtract servers as demand dictates

Open Source:

* [Traefik](https://github.com/containous/traefik) bills itself as the “cloud native edge router.” It’s a modern microservices-focused application load balancer and reverse proxy written in Golang. With its emphasis on support for several modern container orchestration platforms, batteries-included logging, and several popular metric formats, Traefik is a top choice for container-based microservices architectures.
* [Nginx](https://github.com/nginx/nginx) is a name that should be instantly recognizable to anyone involved in web application engineering. This tool offers load balancing capabilities via its [ngx\_http\_upstream\_module](http://nginx.org/en/docs/http/ngx_http_upstream_module.html). A well-established, widely supported option, Nginx offers highly scalable performance out of the box and can be extended with additional modules like Lua.
* [Seesaw](https://github.com/google/seesaw) is another open-source load balancer written in Golang. It was originally created by Google SREs to provide a robust solution for load balancing internal Google infrastructure traffic. When choosing Seesaw, you’re getting the collective engineering acumen of Google’s powerful SRE cohort in an open-source ecosystem.
* [HAProxy](https://github.com/haproxy/) is another common name in the web ecosystem. HAProxy offers reverse proxying and load balancing of TCP and HTTP traffic. When you choose HAProxy, you’re choosing a high-performance, well-established solution.
* A relatively lesser-known offering, [Neutrino](https://github.com/eBay/Neutrino) is a Scala-based software load balancer originally developed by eBay. Neutrino’s strength lies in the broad compatibility of its runtime environment, the JVM.

**Cloud-based and Serverless deployments:**

You can consider serverless computing the next evolutionary step to make things easier for developers. Data centers, virtual machines, Elastic Compute Cloud (EC2), Simple Storage Service, and so on – these were the steps that brought us to today’s notion of serverless architecture. Whenever we refer to [serverless](https://en.wikipedia.org/wiki/Serverless_computing) or serverless architecture, we are talking about relying on a third-party service. So your provider is in charge of managing and assigning computer resources for your code to execute.

Opensource:

* [Apache OpenWhisk](https://openwhisk.apache.org/) is a serverless, open source cloud platform that allows you to execute code in response to events at any scale. It’s written in the Scala language. The framework processes the inputs from triggers like HTTP requests and later fires a snippet of code on either JavaScript or Swift.
* [Fission](https://github.com/fission/fission) is a serverless computing framework that enables developers to build functions using Kubernetes. It allows coders to write short-lived functions in any programming language and map them with any event triggers, such as HTTP requests.
* [IronFunctions](https://github.com/iron-io/functions) is a serverless computing framework that offers a cohesive microservices platform by integrating its existing services and embracing Docker. Developers write the functions in Go language.
* [Fn Project](https://fnproject.io/) is an open source container-native serverless platform that you can run anywhere—on any cloud or on-premise. It’s easy to use, supports every programming language, and is extensible and performant.
* [OpenLambda](https://github.com/open-lambda/open-lambda) is an Apache-licensed serverless computing project, written in Go and based on Linux containers. The primary goal of OpenLambda is to enable exploration of new approaches to serverless computing.

**Fault tolerance:**

Fault Tolerance simply means a system’s ability to continue operating uninterrupted despite the failure of one or more of its components. This is true whether it is a computer system, a cloud cluster, a network, or something else. In other words, fault tolerance refers to how an operating system (OS) responds to and allows for software or hardware malfunctions and failures. An OS’s ability to recover and tolerate faults without failing can be handled by hardware, software, or a combined solution leveraging load balancers(see more below). Some computer systems use multiple duplicate fault tolerant systems to handle faults gracefully. This is called a fault tolerant network.

Open Source:

* [mesos](https://github.com/mesos)/[chronos](https://github.com/mesos/chronos): Fault tolerant job scheduler for Mesos which handles dependencies and ISO8601 based schedules
* Tencent?Tseer: A high available service discovery & registration & fault tolerance framework
* Ovh / metronome: It is a distributed and fault tolerant event scheduler

**Recoverability:**

It is a software testing technique which verifies software’s ability to recover from failures like software/hardware crashes, network failures etc. The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss. Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point. The recovery testing is significant if we are developing the application for a user who will decide the difference between success and failure for our organization. Therefore, we need to develop software, which has enough consistency and recoverability.

Open source:

### FreeRecover – Windows: FreeRecover is an easily installable open-source data recovery application that supports Windows OS.It has a simple search and preview deleted files process for the deleted data and generates previews and recovers deleted files from NTFS drives.Its features include providing integrity estimates of files found and batch recovery to a directory but lacks proper sorting and options for the data information we want to locate.

### Kickass Undelete – Windows: Kickass Undelete is a popular file recovery tool for Windows operating systems. The scanning and recovery process in Kickass Undelete is quick and stable. With Kickass Undelete, you have a good enough data recovery system with its source available to edit accordingly. Unfortunately, the software is no longer actively maintained.

### Scrounge NTFS – Windows: Scrounge NTFS is a free and open-source data recovery program that supports Windows and POSIX compliant.

### PhotoRec: PhotRec is a more text-based user interface and uses file carving techniques to deliver data recovery services. It’s a widely used free and open-source file recovery program to retrieve and restore deleted and misplaced files from any internal or external memory device. It ignores the database system for the underlying data. This way, you quickly recover the lost, damaged, or reformatted files through its tools.

### TestDisk: Distributed under GPL (GNU General Public License v2+), TestDisk is a powerful file recovery tool that makes fixing and recovery of partition table super easy.

**References:**

<https://www.hitechnectar.com/blogs/open-source-data-recovery-software/#FreeRecover>

<https://www.weibull.com/hotwire/issue80/relbasics80.htm#:~:text=Reliability%20requirements%20are%20typically%20part,set%20for%20suppliers%20or%20subcontractors>.

<https://accendoreliability.com/reliability-specifications-requirements/>

<https://t4tutorials.com/reliability-requirements/>

<https://argondigital.com/blog/product-management/reliability-requirements/>

<https://requirementsquest.com/nonfunctional-requirement-examples/>

<https://www.w3schools.com/whatis/whatis_aws_auto_scaling.asp>

<https://avinetworks.com/glossary/auto-scaling/>

<https://www.techtarget.com/searchcloudcomputing/definition/autoscaling>

<https://github.com/Remit/autoscaling-simulator>

<https://www.nginx.com/resources/glossary/load-balancing/>

<https://www.sentinelone.com/blog/serverless-deployment/>

<https://opensource.com/article/18/11/open-source-serverless-platforms>

<https://avinetworks.com/glossary/fault-tolerance/>

<https://www.guru99.com/recovery-testing.html#:~:text=In%20Software%20Engineering%2C%20Recoverability%20Testing,The%20number%20of%20restart%20points>

<https://www.javatpoint.com/recovery-testing>

<https://github.com/ovh/metronome>

https://github.com/Tencent/TSeer