



---

## Defining Services

Philipp Maier  
Course Developer, Google Cloud

In this module we focus on defining services.

A new development begins with the planning and design phases. These require information gathering, starting with business requirements. Once the requirements are defined, it is important to measure that they are providing business value. In this module, we will look at gathering requirements and then techniques for measuring the impact of the solutions.

Let's take a closer look at what we will cover.

# Learning objectives

---

- Describe users in terms of roles and personas.
- Write qualitative requirements with user stories.
- Write quantitative requirements using key performance indicators (KPIs).
- Use SMART criteria to evaluate your service requirements.
- Determine appropriate SLOs and SLIs for your services.

In this module, you will learn to describe users of a system in terms of the roles and personas they take.

These users will then help define and refine the qualitative requirements, which will be captured in the form of user stories. These provide a context for the architectural design and subsequent technical decisions you will make as a cloud architect. Examples of business requirements include accelerating the pace of software development, reducing capital expenditure, and reducing time to recover incidents. The technical requirements of a system are both the functional and non-functional features required.

To help identify the most important requirements and measure their impact, you will learn how to measure success using Key performance indicators, or KPIs.

We will also discuss the importance of using SMART criteria when defining KPIs.

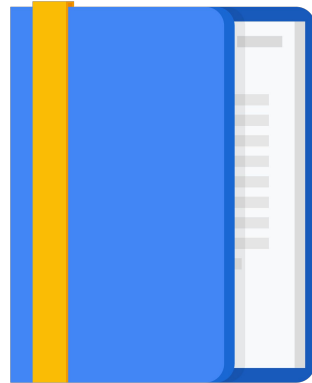
We'll finish by considering the most suitable service level objectives (or SLOs) and service level indicators (or SLIs), and from these the service level agreements (or SLAs).

# Agenda

---

Requirements, Analysis, and  
Design

SLOs, SLIs, and SLAs



Let's start talking about requirements, analysis, and design.

## Qualitative requirements define systems from the user's point of view

|      |  |
|------|--|
| Who  | Who are the users?<br>Who are the developers?<br>Who are the stakeholders?                 |
| What | What does the system do?<br>What are the main features?                                    |
| Why  | Why is the system needed?  |
| When | When do the users need and/or want the solution?<br>When can the developers be done?       |
| How  | How will the system work?<br>How many users will there be?<br>How much data will there be? |

Useful questions to ask as a cloud architect to help build the requirements are: Who? What? Why? When? And how?

- The “who” is about determining not only the users of the system, but also the developers and stakeholders. The aim is to build a full picture of who the system will affect, both directly and indirectly.
- The “what” is both simple and difficult. We need to establish the main areas of functionality required, but in a clear, unambiguous manner.
- “Why” the system is needed is a really important question. What is the problem the proposed system aims to address or solve? Without a clear understanding of the need, it is likely that ‘extra’ requirements will be added. The *why* will also potentially help in defining KPIs and SLOs, SLAs, etc.
- “When” helps determine a realistic timeline and can help contain the scope.
- “How” helps to determine a lot of the non-functional requirements. These could be, for instance, how many users the system needs to support concurrently, what is the average payload size of service requests, are there latency requirements, etc. They could be that the users will be located across the world or in a particular region only.

All of these requirements are vital to capture because they impact the potential solution you, as a cloud architect, will provide.

## Roles represent the goal of a user at some point

| Roles are not people or job titles  | Roles should describe a users objective  | Examples of Roles   |
|---|--|---|
| <ul style="list-style-type: none"><li>• People can play multiple roles</li><li>• A single role can be played by multiple people</li></ul> | <ul style="list-style-type: none"><li>• What does the user want to do?</li><li>• "User" is not a good role (<i>everyone is a user</i>)</li></ul> | <ul style="list-style-type: none"><li>• Shopper</li><li>• Account holder</li><li>• Customer</li><li>• Administrator</li><li>• Manager</li></ul> |

In the previous design activity you defined user roles for your application. Roles represent the goal of a user at some point, and they enable the analysis of a requirement in a particular context. It is important to note that a role is not necessarily a person. It is an actor on the system and could be another system such as a microservice client that is accessing another microservice.

The role should describe the user's objective when using the system. For example, the role of a shopper on an e-commerce application clearly defines what the user wants to do. There are many ways to determine the roles for the requirement you are working on. One process that works particularly well is:

- First brainstorm an initial set of roles: Write as many roles as you can think of, with each role being a single user.
- Now organize this initial set: Here you can identify overlapping roles and related roles and group these together.
- With the set of roles now grouped, consolidate the roles: The aim here is to consolidate and condense the roles to remove duplication.
- Finally, refine the roles, including internal and external roles, and the different usage patterns. Here extra information can be provided such as the user's level of expertise in the domain, or the frequency of use of the proposed software.

Following a simple process like this provides structure and brings focus to the task.

## Personas describe a typical person who plays a role

- In a real-world application, go find your users and talk to them.
- Personas tell a story of who they are.
- Personas are not a list of job functions.
- For each role, there could be many personas.

### Example persona:

*Jocelyn is a busy working mom who wants to access MegaCorp Bank to check her account balances and make sure that there are enough funds to pay for her kids' music and sport lessons. She also uses the website to automate payment of bills and view her credit account balances. Jocelyn wants to save time and money, and she wants a credit card that gives her cash back.*

Identifying user roles is a useful technique as part of the requirements-gathering process. An additional technique, in particular for more important roles, can be to create a persona for the role. A persona is an imaginary representation of a user role. The aim of the persona is to help the architect and developers think about the characteristics of users by personalizing them. Often a role has multiple personas.

Consider the example of requirements for a banking application. We can think in terms of users of the system, and many requirements can be gathered this way. Using personas can provide further insights. For example, Jocelyn is a persona who is a busy working mom. Jocelyn wants to save time and money as well as perform the standard banking operations online and receive benefits such as cash back. Using a persona helps build a fuller picture of the requirements. For instance, Jocelyn's wanting to save time indicates that the tasks to be performed should possibly be automated, which affects latency and service design.

In this example, when a question arises from the architect or maybe a developer, they can often better answer that question by thinking, "What would Jocelyn want here?"

## User stories describe a feature from the user's point of view

- Give each story a title that describes its purpose.
- Write a short, one sentence description.
- Specify the user role, what they want to do, and why.
- Use the template: As a [role], I want to..., so that I can...

### Example user story:

#### Balance Inquiry

*As an account holder, I want to check my available balance at any time of day so I am sure not to overdraw my account.*

Now, user stories describe one thing a user wants the system to do. They are written in a structured way typically using the form:

**As a** [type of user] **I want to** [do something] **so that I can** [get some benefit]

Another commonly used form is:





**Given** [some context] **When I** [do something] **Then** [this should happen]

So when writing stories, give each story a title that describes its purpose as a starting point. Follow this with a concise one-sentence description of the story that follows one of the forms just described. This form describes the user role, what they want to do, and why they want to do it. As an example, consider a banking system and a story to determine the available balance of a bank account.

The title of the story could be Balance Inquiry. Then following the template we describe the story **As an** account holder, **I want to** check my available balance at any time of day, **so I am** sure not to overdraw my account.

This explains the role, what they want to do, and why they want to do it.

## Evaluate user stories with the INVEST criteria

- Independent 
- Negotiable 
- Valuable 
- Estimatable 
- Small 
- Testable 

User stories provide a clear and simple way of agreeing to requirements with a customer/end user. The INVEST criteria can be used to evaluate good user stories. Let me go through each letter of these criteria.

- **I**ndependent: A story should be independent to prevent problems with prioritization and planning.
- **N**egotiable: They are not written contracts, but are used to stimulate discussion between customer and developers until there is a clear agreement. They aid collaboration.
- **V**aluable: Stories should provide value to users. Think about outcomes and impact, not outputs and deliverables.
- **E**stimatable: The story must be estimatable. If it is not, it often indicates missing details or the story is too large.
- **S**mall: Good stories should be small. This helps keep scope small and therefore less ambiguous and supports fast feedback from users.
- **T**estable: Stories must be testable so that developers can verify that the story has been implemented correctly and validate when the requirement has been met/is done.



Refer to your Design and Process Workbook.

Refer to your Design and Process Workbook.

- Refine the roles you listed in Activity 1.
- Write personas for each role.
- Write user stories for the main features of your case study.



In this design activity, you are going to work on activities 2a and 2b of the design workbook. In the first activity, you defined roles. For an online store, examples of roles might be account holder, shopper, customer, administrator, and seller. Roles are played by people, and different people playing the same role might be significantly different.

In activity 2a, you will write some user personas. Personas are stories that describe typical people playing some role while using your system. It's important to understand your users when designing a system, so it's important to write some personas. In a real project, you should find some users and interview them. For this course, feel free to use your imagination.

In activity 2b, you will write some user stories. User stories are short, 1-sentence descriptions of your application's features. In the first activity you listed some features. Now, turn those features into user stories. Write your user stories in the form "As a (fill in the role), I want to (tell me your goal), so (why is this important to you)."

For example, an online store might have a feature to search for products. The user story might be, “As a shopper, I want to be able to quickly search for products by name, keyword or category, so I can quickly find information about products I want to purchase.”

*Jocelyn is a busy working mom who wants to access MegaCorp Bank to check her account balances and make sure that there are enough funds to pay for her kids' music and sport lessons. She also uses the web site to automate payment of bills and see her credit account balances. Jocelyn wants to save time and money, and she wants a credit card that gives her cash back.*

Here is an example persona.

“Jocelyn is a busy working mom who wants to access MegaCorp Bank to check her account balances and make sure that there are enough funds to pay for her kids' music and sport lessons. She also uses the web site to automate payment of bills and see her credit account balances. Jocelyn wants to save time and money, and she wants a credit card that gives her cash back.”

*Balance Inquiry*

***As a** checking account holder, **I want to** check my available balance at any time of day **so that** I am sure not to overdraw my account.*

Here is an example user story for a feature: Balance Inquiry

“As a checking account holder, I want to check my available balance at any time of day, so that I am sure not to overdraw my account.”

## Review Activity 2: Analyzing your case study

- Refine the roles you listed in Activity 1.
- Write personas for each role.
- Write user stories for the main features of your case study.



In this second activity you were asked to write personas and user stories for your case study.

**Karen:**

Karen is a busy businesswoman who likes to take luxury weekend breaks, often booked at the last minute. A typical booking comprises a hotel and flight. Recommendations play a major role in the choice Karen makes, as does customer feedback. Karen likes to perform all operations from her phone.

**Andrew:**

Andrew is a student who likes to travel home to visit parents and also takes vacations twice yearly. His primary concern is cost, and he will always book the lowest price travel regardless of convenience. Andrew has no loyalty and will use whichever retailer can provide the best deal.

Here are a couple of examples of personas for our online travel portal.

Karen is a busy businesswoman who likes to take luxury weekend breaks, often booked at the last minute. A typical booking comprises a hotel and flight. Recommendations play a major role in the choice Karen makes, as does customer feedback. Karen likes to perform all operations from her phone.

Andrew is a student who likes to travel home to visit parents and also takes vacations twice yearly. His primary concern is cost, and he will always book the lowest price travel regardless of convenience. Andrew has no loyalty and will use whichever retailer can provide the best deal.

### **Search for Flight and Hotel**

**As a** traveler, **I want to** search for a flight-hotel combination to a destination on dates of my choice **so that** I can find the best price.

### **Supply Hotel Inventory**

**As a** hotel operator, **I want to** bulk supply hotel inventory **so that** ClickTravel can sell it on my behalf.

### **Analyze Sales Performance**

**As a** ClickTravel manager, **I want to** analyze the sales performance data of all our suppliers **so that** I can identify poor performers and help them improve.

Here are a couple of examples of users stories for our online travel portal.

For the “Search for Flight and Hotel” feature I could write:

As a traveler, I want to search for a flight-hotel combination to a destination on dates of my choice, so that I can find the best price.

For the “Supply Hotel Inventory” feature I could write:

As a hotel operator, I want to bulk supply hotel inventory, so that ClickTravel can sell it on my behalf.

For the “Analyze Sales Performance” feature I could write:

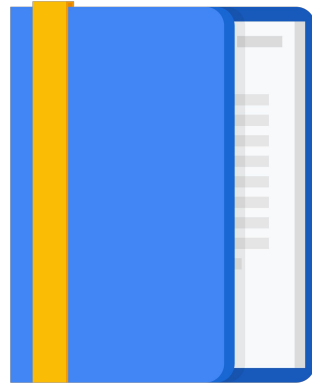
As a ClickTravel manager, I want to analyze the sales performance data of all our suppliers, so that I can identify poor performers and help them improve.

# Agenda

---

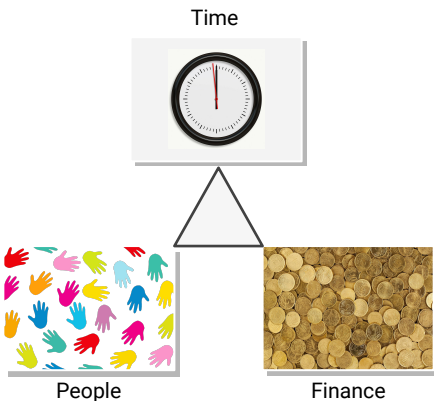
Requirements, Analysis, and Design

SLOs, SLIs, and SLAs



With a set of requirements in place, we will now move on to consider how to measure whether the technical and business requirements have been met.

## Quantitative requirements are things that are measurable



Given the constraints:

- Time
- Finance
- People

What can be achieved:

- How many users are there?
- How much data is there?
- What are the rewards and risks?
- Which features can be launched?

To manage a service well, it is important to understand which behaviors matter and how to measure and evaluate these behaviors. These must always be considered in the context of the constraints, which are usually time, funding, and people. Then we consider what can be achieved. The type of system being evaluated determines the data that can be measured.

For example, for user-facing systems, was a request responded to? (which refers to availability), how long did it take to respond? (which refers to latency), how many requests can be handled? (which refers to throughput).

For data storage systems, how long does it take to read and write data? (that's latency), is the data there when we need it? (that's availability), if there is a failure, do we lose any data (that's durability).

The key to all these items is that the questions can be answered with data gathered from the services.



## Key performance indicators (KPIs) are metrics that can be used to measure success

In business, common KPIs include:

- Return on investment (ROI)
- Earnings before interest and taxes (EBIT)
- Employee turnover
- Customer churn

In software, common KPIs include:

- Page views
- User registrations
- Clickthroughs
- Checkouts

Business decision makers want to measure the value of projects. This enables them to better support the most valuable projects and not waste resources on those that are not beneficial. A common way to measure success is to use KPIs. KPIs can be categorized as business KPIs and technical KPIs.

Business KPIs are a formal way of measuring what the business values, such as ROI, in relation to a project or service. Others include earnings before interest and taxes or impact on users such as customer churn, or maybe employee turnover.

Technical or software KPIs can consider aspects such as how effective the software is through page views, user registrations, and number of checkouts. These KPIs should also be closely aligned with business objectives.

As an architect, it is important that you understand how the business measures success of the systems that you design.

KPIs indicate whether you are on track to achieve the goal



Goal: Increase turnover for an online store

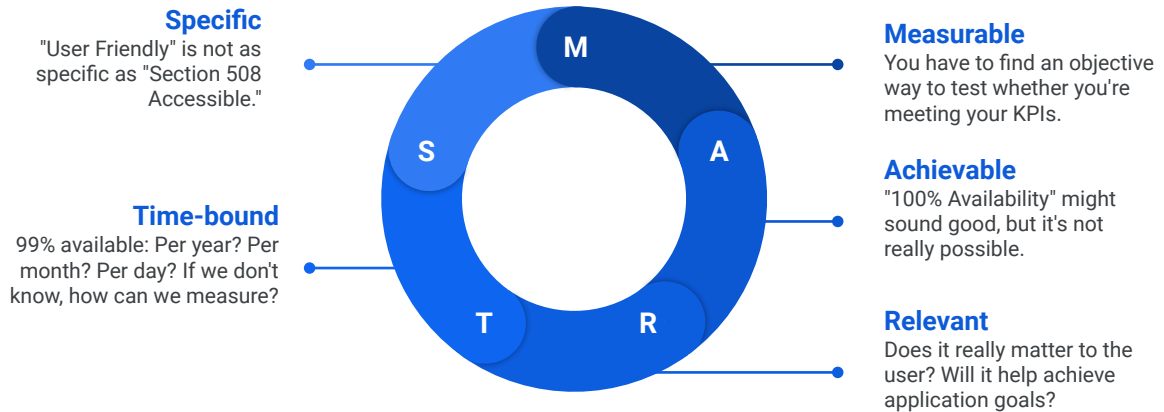
KPI: The percentage of conversions on the website

Now, a KPI is not the same thing as a goal or objective. The goal is the outcome or result you want to achieve. The KPI is a metric that indicates whether you are on track to achieve the goal.

To be the most effective, KPIs need an accompanying goal. This should be the starting point in defining KPIs. Then for each goal, define the KPIs that will allow you to monitor and measure progress. For each KPI, define targets for what success looks like. Monitoring KPIs against goals is important to achieving success and allows readjustment based on feedback.

As an example, a goal may be to increase turnover for an online store, and an associated KPI may be the percentage of conversions on the website.

## For KPIs to be effective, they must be SMART

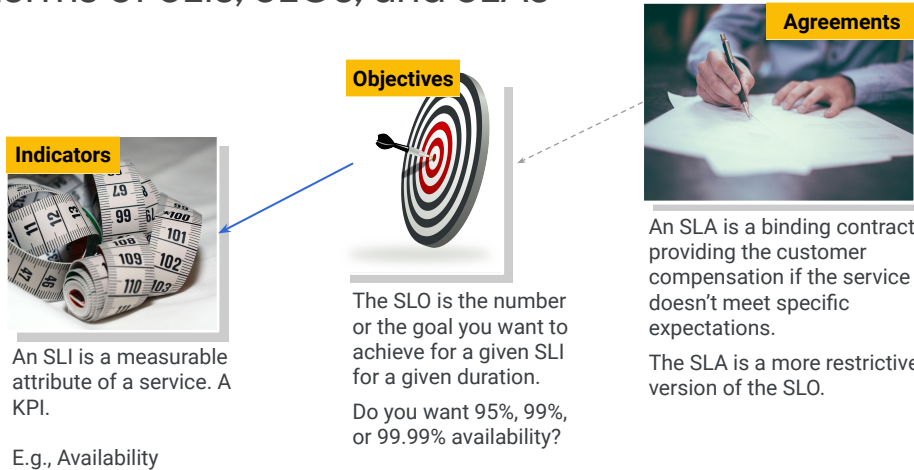


For KPIs to be effective, they must be SMART. That is:

- **Specific** rather than general. For example, "user friendly" is not specific; it's very subjective. "Section 508 accessible" is much more specific.
- **Measurable** is vital because monitoring the KPI indicates whether you are moving toward or away from your goal.
- Being **Achievable** is also important. For example, expecting 100% conversions on a website is not achievable.
- **Relevant** is absolutely vital. Without a relevant KPI, the goal probably will not be met. In our example of increasing turnover, if we are improving the conversion rate, a subsequent increase in turnover should be achievable—assuming a similar number of users.
- **Time-bound** helps with measuring the KPI. Some KPIs are more sensitive to time; for example, is "availability" per day, per month, or per year?

So to summarize, KPIs are used to measure success or progress toward a goal.

## Quantitative requirements can be expressed in terms of SLIs, SLOs, and SLAs







Let's introduce service level terminology. To provide a given level of service to customers, it is important to define service level indicators (SLIs), objectives (SLOs), and agreements (SLAs). These are measurements that describe basic properties of the metrics to measure, the values those metrics should read, and how to react if the metrics cannot be met.

**Service level indicator** is a quantitative measure of some aspect of the level of service being provided. Examples include throughput, latency, and error rate.

**Service level objective** is an agreed-upon target or range of values for a service level that is measured by an SLI. It is normally stated in the form  $SLI \leq \text{target}$  OR  $\text{lower bound} \leq SLI \leq \text{upper bound}$ . An example of an SLO is that the average latency of HTTP requests for our service should be less than 100 milliseconds.

**Service level agreement** is an agreement between a service provider and consumer. They define responsibilities for delivering a service and consequences when these responsibilities are not met. The SLA is a more restrictive version of the SLO. We want to architect a solution and maintain an agreed SLO so that we provide ourselves spare capacity against the SLA.

## SLIs must be time-bound and measurable

-  Fast response time
-  HTTP GET requests respond within 400 ms aggregated per minute
-  Highly available
-  Percentage of successful requests over all requests aggregated per minute

Understanding what users want from a service will help inform the selection of indicators. The indicators must be measurable. For example, fast response time is not measurable, whereas HTTP GET requests that respond within 400ms aggregated per minute is clearly measurable. Similarly, highly available is not measurable, but percentage of successful requests over all requests aggregated per minute is measurable.

Not only must indicators be measurable, but the way they are aggregated needs careful consideration. For example, consider requests per second to a service. How is the value calculated: by measurements obtained once per second or by averaging requests over a minute? The once-per-second measurement may hide high request rates that occur in bursts of a few seconds.

For example, consider a service that receives 1000 requests per second on even numbered seconds and 0 requests on odd numbered seconds. The average requests per second could be reported over a minute as 500. However, the reality is that the load at times is twice as large as the average. Similar averages can mask user experience when used for metrics like latency. It can mask the requests that take a lot longer to respond than the average.

It is better to use percentiles for such metrics where a high order percentile, such as 99%, shows worst case values, while the 50th percentile will indicate a typical case.

## SLOs must be achievable and relevant

| SLI   | SLO     |   |
|---|---------|---|
| HTTP POST photo uploads complete within 100ms aggregated per minute               | 99%     | ✗ If our users are using mobile phones, maybe this is overkill. |
|   | 80%     | ✓ This might be good enough.                                    |
| Available as measured with an uptime check every 10 seconds aggregated per minute | 100%    | ✗ Sounds good, but not practical.                               |
|   | 99.999% | ✗ Possible, but maybe too expensive.                            |
|   | 99%     | ✓ Maybe good enough and easier and more cost-effective.         |

The relevancy of SLOs is vital. You want objectives that help or improve the user experience. It is easy to define SLOs based around what is easy to measure rather than what is useful. For clarity, SLOs should specify how they are measured and the conditions when they are valid.

Consider availability as measured with an uptime check over 10 seconds aggregated per minute. It is unrealistic as well as undesirable to have SLOs with a 100% target. Such a target results in expensive, overly conservative solutions that are still unlikely to reach the SLO. It is better to track the rate at which SLOs are missed and work to improve this. In many cases, 99% may be good enough availability and be far easier to achieve as well as engineer. It is also highly likely to be much more cost effective to run.

The use case needs to be considered also. For example, if a HTTP service for photo uploads requires 99% of uploads to complete within 100ms aggregated per minute, this may be unrealistic or overkill if the majority of users are using mobile phones. In such a case, an SLO of 80% is much more achievable and good enough.

It is often ok to specify multiple SLOs. Consider the following:

99% of HTTP GET calls will complete in less than 100ms

This is a valid SLO but it may be the case that the shape of the performance curve is important. In this case, the SLO could be written as follows:

90% of HTTP GET calls will complete in less than 50ms  
99% of HTTP GET calls will complete in less than 100ms  
99.9% of HTTP GET calls will complete in less than 500ms

## Tips for determining SLOs

- The goal isn't to make SLOs as high as possible; the goal is to make them as low as you can get away with while still making users happy. That's why it's important to understand your users.
- The higher you set the SLO, the higher the cost in compute resources (redundancy) and operations effort (people time).
- Applications should not significantly outperform their SLOs, because users come to expect the level of reliability you usually give them.

Selecting SLOs has both product and business implications. Often tradeoffs need to be made based on constraints such as staff, time to market, and funding. As the slide states, the aim is to keep users happy, not to have an SLO that requires heroic efforts to maintain. Let me give you some tips on selecting SLOs:

- Do not make them too high: It is better to have lower SLOs to begin with and tighten them over time as you learn about the system instead of defining those that are unattainable and require a significant effort and cost to try and achieve
- Keep them simple: More complex SLIs can obscure important changes in performance.
- Avoid absolute values: To have a SLO that states 100% availability is unrealistic. Such a SLO increases the time to build, complexity, and cost to operate, and in most cases is highly unlikely to be required.
- Minimize SLOs: A common mistake is to have too many SLOs. The recommendation is to have just enough SLOs to give coverage of the key system attributes.

In summary, good SLOs should reflect what the users care about. They work as a forcing function for development teams. A poor SLO will result in a significant amount of wasted work if it is too ambitious, or a poor product if it is too relaxed.



## An SLA is a business contract between the provider and the customer

The SLA stipulates that:

- A penalty will apply to the provider if the service does not maintain certain availability and/or performance thresholds.
- If the SLA is broken, the customer will receive compensation from the provider.

Not all services have an SLA, but all services should have an SLO.

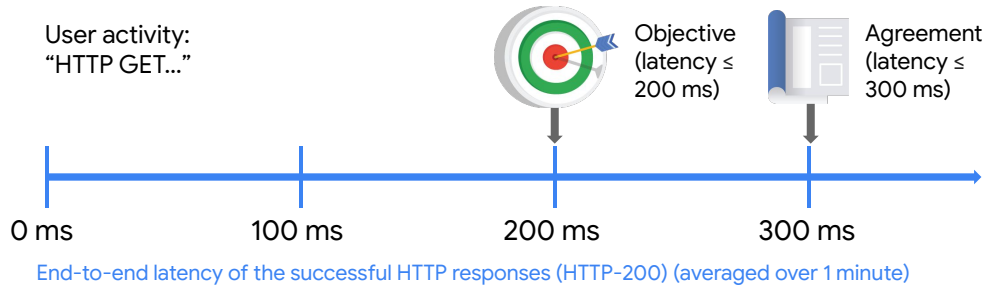
Your SLO thresholds should be stricter than your SLA.



An SLA is a business contract between the service provider and the customer. A penalty will apply if the service provider does not maintain the levels agreed on. Not every service has an SLA, but all services should have SLOs.

As with SLOs, it is better to be conservative with SLAs because it is too difficult to change or remove SLAs that offer little value or cause a large amount of work. In addition, because they can have a financial implication through compensation to the customer, setting them too high can result in unnecessary compensation being paid. To provide protection and some level of safety, an SLA should have a threshold that is lower than the SLO. This should always be the case.

## Example: SLI, SLO, and SLA



- SLI: The latency of successful HTTP responses (HTTP-200).
- SLO: The latency of 99% of the responses must be  $\leq 200$  ms.
- SLA: The user is compensated if 99th percentile latency exceeds 300 ms.

Let's consider an example of a service and SLI, SLO, and SLAs for the service. The service is an HTTP endpoint accessed using a HTTP GET.

The SLI is the end-to-end latency of successful HTTP responses, that is HTTP 200s. These are averaged over one minute. The SLO has been agreed that the latency of 99% of the responses must be less than or equal to two hundred milliseconds.

The SLA is set that the user is compensated if the 99th percentile latency exceeds 300ms. The SLA has clearly built a buffer over the SLO, which means that even if the SLO is exceeded, there is some capacity before the SLA is broken. This is the wanted position in the relationship between SLO and SLA.

## Activity 3: Defining SLIs and SLOs

Refer to your Design and Process Workbook.

- Write SLIs and SLOs for your case study features.



In this design activity, you will define SLIs and SLOs for your case study.

Let's say you wanted to write an SLI and SLO for an online shopping application related to availability. The SLO is what you want to measure. For example, you might measure "the fraction of successful vs unsuccessful HTTP responses from an API endpoint, aggregated per day."

The SLO is the target your are trying to achieve. For your online store's Search for Products service, that might be 99.95%.

| User Story             | SLO   | SLI   |
|------------------------|---|---|
| <i>Balance Inquiry</i> | <i>Available 99.95%</i>                               | <i>Fraction of 200 vs 500 HTTP responses from API endpoint measured per day</i>       |
| <i>Balance Inquiry</i> | <i>95% of requests will complete in under 300 ms.</i> | <i>Time to last byte GET requests measured every 10 seconds aggregated per minute</i> |
|                        |   |   |

Here are a couple more examples for an imaginary online bank. The SLI defines what you want to measure, like the fraction of 200 vs 500 HTTP responses from an API endpoint or the time to last byte GET requests measured every 10 seconds aggregated per minute.

The SLO defines the target you want to achieve, like 99.95% availability or that 95% of requests will complete in under 300 ms.

## Review Activity 3: Defining SLIs and SLOs

- Write SLIs and SLOs for your case study features.



In this third activity you were asked to write SLIs and SLOs for your case study.

| User Story                | SLO  | SLI  |
|---------------------------|--|--|
| Search Hotel and Flight   | Available 99.95%                               | Fraction of 200 vs 500 HTTP responses from API endpoint measured per month         |
| Search Hotel and Flight   | 95% of requests will complete in under 200 ms. | Time to last byte GET requests measured every 15 seconds aggregated per 5 minutes  |
| Supply Hotel Inventory    | Error rate of < 0.00001%                       | Upload errors measured as a percentage of bulk uploads per day by custom metric    |
| Supply Hotel Inventory    | Available 99.9%                                | Fraction of 200 vs 500 HTTP responses from API endpoint measured per month         |
| Analyze sales performance | 95% of queries will complete in under 10s      | Time to last byte GET requests measured every 60 seconds aggregated per 10 minutes |

Here are some example SLOs and SLIs for out travel portal application. Notice that the SLI describes what we are going to measure and how: for example, the “Fraction of 200 vs 500 HTTP responses from API endpoint measured per month.” This example is a way of measuring availability.

The SLO represents the goal we are trying to achieve for a given SLI. For example, “Available 99.95%” of the time.”

Feel free to pause the video to read through the other SLOs and SLIs for each user story.

# Review

---

## Defining Services

In this module we learned about qualitative and quantitative requirements. Qualitative requirements are things that the user cares about, like features. We can express qualitative requirements in the form of user stories. In order to understand our users better, we should write personas.

Quantitative requirements are things we can measure. We can express these as key performance indicators, or KPIs. KPIs in software are things like user signups, clicks per session, completed purchases, or customer retention. We can also express quantitative requirements as SLOs and SLIs. These are lower-level metrics, things like latency, availability, or response time.