**1**

**The Meaning of Solution Architecture**

This book will be your first step in the solution architecture world, acting as a comprehensive guide to learn all about solution architecture, and allowing you to become a professional solution architect. In this chapter, we will explore the meaning of solution architecture, and how it is the foundation of solution development in an organization. A robust solution architecture design helps successful software application development in a complex organization, covering all aspects, from IT infrastructure, application security, and reliability, to operational aspects in production.

For successful application development, defining the solution architecture should be the first step, which then lays out the foundations and the robust building blocks of implementation. Solution architecture handles critical, non-functional requirements such as scalability, high availability, maintainability, performance, and security while keeping business requirements in mind.

A solution architect is a person who is responsible for designing solution architecture by collaborating with and across stakeholders. The solution architect analyzes the functional requirements and defines non-functional requirements in order to cover all aspects of the solution—and avoid any surprises. Each solution has multiple constraints, such as cost, budget, timeline, and regulatory constraints, so the solution architect must consider them while selecting technology during the application design process to solve a given business problem.

The solution architect develops a proof of concept and prototype in order to evaluate various technology platforms, and then chooses the best strategy for solution implementation.

They mentor the team throughout solution development and provide post-launch guidance to maintain and scale the final product.

In this chapter, you will learn about the following topics:

* What is solution architecture?
* The evolution of solution architecture
* Why is solution architecture important?
* The benefits of solution architecture
* Solution architecture in the public cloud

By the end of this chapter, you will have learned about the benefits of solution architecture to every aspect of an enterprise application. You will evaluate solution architecture in the public cloud and develop a cloud-native approach to architecture design.

**What is solution architecture?**

Asking this question to a variety of professionals may lead to ten different answers for the definition of solution architecture. In fact, they may all be correct, within the context of a given organization's structure. Each organization may see solution architecture from a different perspective, based on their business needs, organizational hierarchy, and solution complexity.

In a nutshell, solution architecture can be described as defining and foreseeing multiple aspects of a business solution, from both strategic and transactional perspectives. "Strategic" means that a solution architect defines a long-term vision for a software application to ensure it stays relevant, regardless of future changes, with possible extensions to address increasing user workload and additional feature demand. "Transactional" means an application should handle the current customer workload and address daily business challenges without any issues.

Solution architecture is not just about providing a software solution. It covers all aspects of a system, which includes, but is not limited to, system infrastructure, networking, security, compliance requirement, system operation, cost, and reliability. As can be seen in *Figure 1.1*, there are many aspects that a solution architect may need to address.

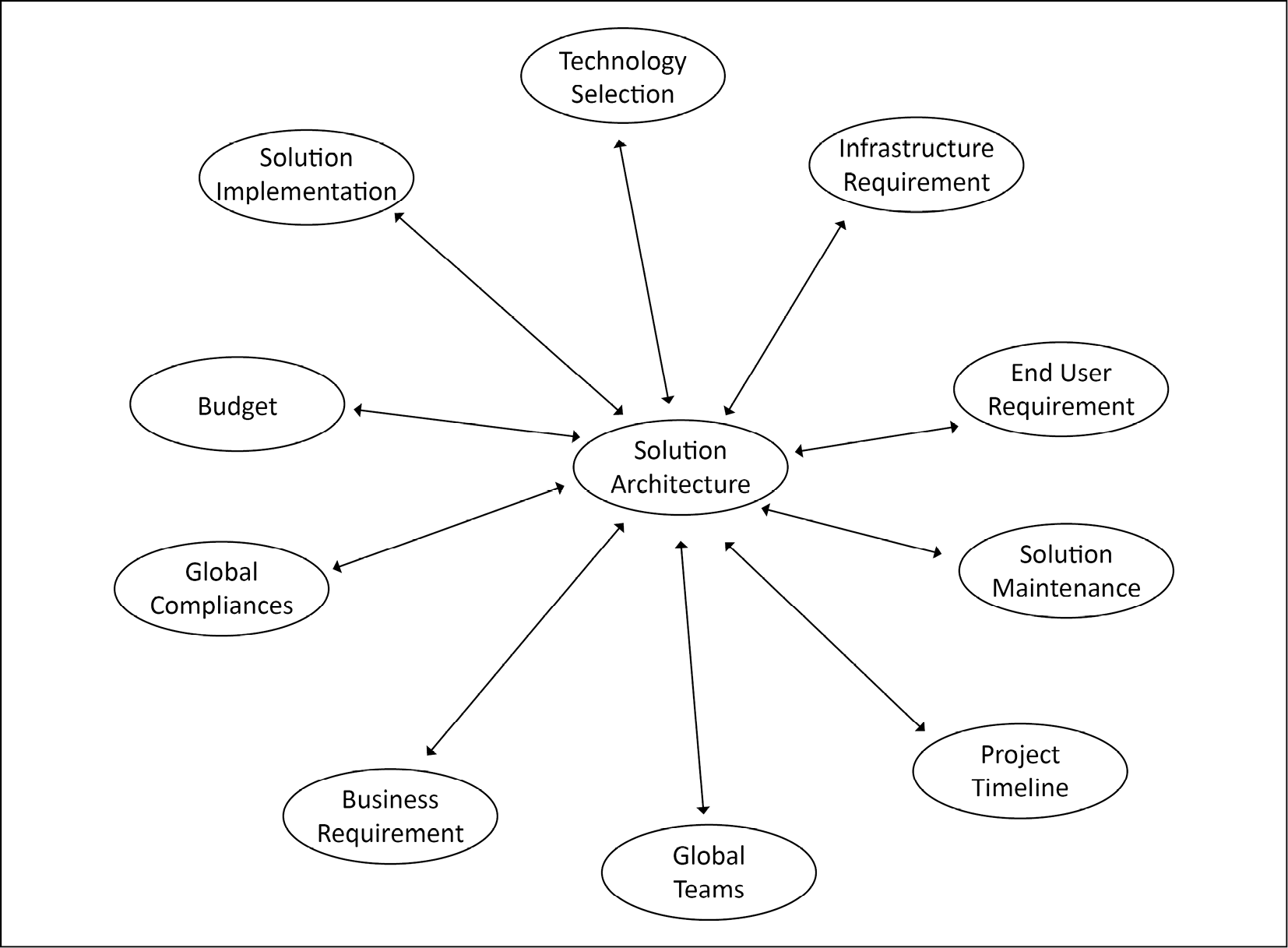


Figure 1.1: Circle of solution architecture

A good solution architect addresses the most common aspects of the software solution in an organization:

* **Globally distributed teams**: In this age of globalization, almost every product has users distributed across the globe, and stakeholder groups to take care of customer needs. Often, the software development team has an onshore-offshore model, where a team works across different time zones to increase productivity and optimize project costs. Solution design needs to consider a globally distributed team structure. This means solution development and operation should not be people-dependent but utilize tools to scale and collaborate regardless of team members' work locations and time zones.
* **Global compliance requirement**: When you are deploying your solution globally, each country and region has its laws and compliance regulations, which your solution must adhere to. Some examples are as follows:
  + The **Federal Risk and Authorization Management Program** (**FedRAMP**) and **Department of Defense Cloud Computing Security Requirements Guide** (**DoDSRG**) for the USA
  + The **General Data Protection Regulation** (**GDPR**) for Europe
  + The **Information Security Registered Assessors Program** (**IRAP**) for Australia
  + The **Center for Financial Industry Information Systems** (**FISC**) for Japan
  + The **Multi-Tier Cloud Security** (**MTCS**) standard for Singapore
  + The **G-Cloud** for the UK
  + The **IT-Grundschutz** for Germany
  + The **Multi-Level Protection Scheme** (**MLPS**) Level 3 for China
* Compliance requirements are different between industries; for example, the **International Organization for Standardization** (**ISO**) **9001** (which is primarily for healthcare, life sciences, medical devices, and the automotive and aerospace industries), the **Payment Card Industry Data Security Standard** (**PCI DSS**) for finance, and the **Health Insurance Portability and Accountability Act** (**HIPAA**) for healthcare. Solution architecture needs to consider any compliance adherence in the design phase. You will learn more about compliance in *Chapter 8*, *Security Considerations*.
* **Cost and budget**: Solution architecture gives a good estimation of the overall cost of the project, which helps to define a budget. This includes **capital expenditure** (**CapEx**), which is the upfront cost, and **operational expenditure** (**OpEx**), an ongoing cost. It helps management to create an overall budget for human resources, infrastructure resources, and other licensing-related costs.
* **Solution implementation component**: Solution architecture provides a high-level overview of different implementation components of the product beforehand, which helps to plan execution.
* **Business requirements**: Solution architecture considers all business requirements, both functional and non-functional. A functional requirement addresses the application features that an end user will directly interact with. Non-functional requirements are not directly related to customer-facing feature enhancement, but impact the overall application in terms of critical factors, including performance, scalability, and availability. It ensures that business requirements are compatible, allowing them to be converted into the technical implementation stage, and strikes a balance between stakeholders.
* **IT infrastructure requirements**: Solution architecture determines what kind of IT infrastructure is required to execute the project; this includes computing, storage, network, and considerations, and helps to plan the IT resources more effectively.
* **Technology selection**: During solution design, a solution architect creates a prototype, which considers the corporate requirements, and then recommends the right technology and tools for implementation. Solution architecture aims to build in-house rather than third-party tool sourcing while defining software standards across the organization.
* **End user requirements**: Solution architecture pays special attention to the requirements of the end user who will be the actual consumer of the product. It helps to discover the hidden requirements that a product owner may not be able to capture. During implementation and launch, the solution architect provides a standard document and standard language structure in order to make sure that all of the requirements have been met to satisfy the user's needs.
* **Solution maintenance**: Solution architecture is not just about solution design and implementation, but also takes care of post-launch activities, such as solution scalability, disaster recovery, and operational excellence.
* **Project timeline**: Solution architecture designs the layout details of each component with their complexity, which further helps to define the project milestones and timeline by providing resource estimation and associated risks.

An industry-standard and well-defined solution architecture addresses all business requirements within a technical solution and makes sure to deliver the desired result in order to satisfy the stakeholders, as per their expectations in terms of the quality, availability, maintainability, and scalability of the solution.

The initial design of a solution architecture may be conceived at a very early stage during the pre-sales cycle, such as the **request for proposal** (**RFP**) or the **request for information** (**RFI**), and is followed by the creation of a prototype or proof of concept, in order to discover any solution risk. The solution architect also identifies whether to build a solution or to source it. It helps to identify the appropriate technology, while also keeping an organization's critical security and compliance requirements in mind.

There are two primary situations for creating a solution architecture:

* First, enhancing technology for an existing application, which may include hardware refresh or software re-architecting.
* Second, creating a new solution from the ground up, where you get more flexibility to choose the best fit of technology to address a business requirement.

However, while re-architecting an existing solution, you need to consider the minimal impact on the current environment. Solution architects may decide to completely rebuild if re-architecting the existing solution is not worth it, and a better solution can be provided through a full rebuild approach.

Put simply, solution architecture is about looking at all the aspects of the system in order to generate a technical vision, which provides steps to implement the business requirements. Solution architecture can define an implementation for a project, or a group of projects in a complex environment, by putting together all of the different pieces that are related to data, infrastructure, networking, and software applications. A good solution architecture not only satisfies the functional and non-functional requirements but also addresses system scalabilities and maintenance in the long run.

We have now briefly covered the role of solution architecture and its different aspects. In the next section, we will look at the evolution of solution architecture.

**The evolution of solution architecture**

Solution architecture has evolved with technological modernization. Today, solution architecture design has changed drastically compared to a couple of decades ago, due to the increasing use of the internet, the availability of high-bandwidth networks, the low cost of storage, and compute availability.

Back in the days before the era of the internet, most solution designs focused on providing a thick desktop client that was capable of operating with low bandwidth and working offline when a system could not connect to the internet.

This technology has evolved over the two decades. **Service-oriented architecture** (**SOA**) started taking shape for distributed design, and applications started moving from monolithic to *modern n-tier architecture*, where the frontend server, application server, and database were live in their own compute and the storage layer. These SOAs are mostly achieved by an XML-based messaging protocol, called **Simple Object Access Protocol** (**SOAP**). A major component of this is its ability to follow a client-server model in order to create services.

In this age of digitization, you will see microservice-based solution design becoming increasingly popular, which is based on **JavaScript Object Notation** (**JSON**) messaging and the **Representational State Transfer** (**REST**) service. These are web APIs, which do not require XML-based web service protocols (SOAPs) to support their interfaces. They rely on web-based HTTP protocols such as POST, GET, UPDATE, DELETE, and so on. You will learn more about different architecture patterns in great detail in *Chapter 6*, *Solution Architecture Design Patterns*.

The microservice architecture addresses the need for changing requirements in an agile environment, where any solution changes need to be accommodated and deployed rapidly. Organizations have to be agile to stay ahead of the competition, which forces solution architecture to be flexible compared to the waterfall model, where you have a long cycle before project release.

The web-based microservice architecture is fueled by an almost unlimited resource capability, which is available from cloud providers, and can scale in minutes or seconds. It's becoming easier to innovate, experiment, and change as solution architects and developers can risk failing without impacting business functions.

**Why is solution architecture important?**

Solution architecture is a component of the foundation of an overall enterprise software solution that addresses specific problems and requirements. As the project size increases, the team becomes distributed globally. It is required to have solution architecture in place for long-term sustainability and a solid foundation.

Solution architecture addresses various solution needs, keeping the business context intact. It specifies and documents technology platforms, application components, data requirements, resource requirements, and many important non-functional requirements, such as scalability, reliability, performance, throughput, availability, security, and maintainability.

Solution architecture is vital for any industry to solve business problems using software applications. In the absence of solution architecture, there is a chance that software development could fail: projects can get delayed, go over budget, and not deliver enough in the form of functionalities. This scenario can be drastically improved by creating a solution architecture and applying experience and knowledge—all of which are provided by a solution architect. It helps to keep stakeholders from all areas, from non-technical business functions through to technical development, on the same page, which avoids confusion, keeps the project on track, within schedule and on time, and helps to derive maximum **return on investment** (**ROI**).

Often, the solution architect requires customer collaboration in order to understand specifications. In a solution architect's role, the architect needs to call on multiple skillsets, from technical leaders and experts to business analysts and project management. We will learn more about the solution architect's role in *Chapter 2*, *Solution Architects in an Organization*.

A good solution architecture puts specifications in place with a well-defined solution, which helps us to deliver and accomplish the final product, along with smooth product operability after launch.

A single problem can have multiple solutions, and each solution has its constraints. Solution architecture considers all the solutions and finds the best one by creating a hands-on proof of concept that accommodates all of the business and technical limitations.

Let's learn about the various benefits of solution architecture in detail.

**The benefits of solution architecture**

Now that we have detailed the importance of solution architecture, we will now provide more details on the benefits of solution architecture in various aspects of an organization; *Figure 1.2* is a breakdown of the potential benefits bestowed upon an organization when employing the role of solution architect in the business.

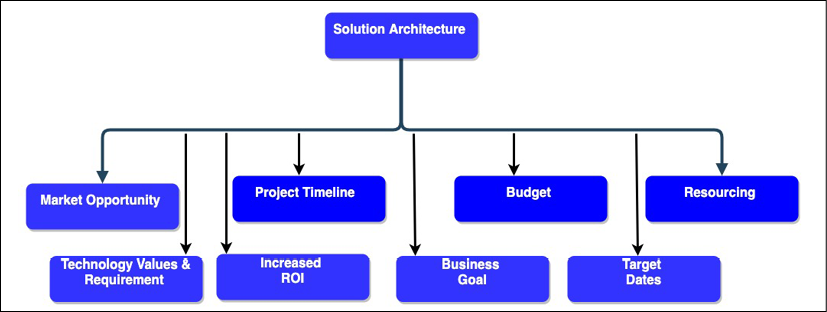


Figure 1.2: A solution architecture's beneficial attributes

The preceding diagram highlights the following attributes of a good solution architecture:

* **Technology values and requirements**: Solution architecture determines the ROI, which solution can be obtained by a particular technology selection, and the market trends. The solution architect evaluates which technology an organization or project should adopt in order to achieve long-term sustainability, maintainability, and team comfort.
* **Business goal**: The primary responsibility of a solution architecture design is to accommodate the needs of the stakeholders and adapt it to their requirements. Solution architecture converts business goals into a technical vision by analyzing market trends and implementing best practices. Solution architecture needs to be flexible enough to meet new, challenging, demanding, and rapidly changing business requirements.
* **Target dates**: A solution architect continuously works with all stakeholders, including the business team, customers, and the development team. A solution architect defines the process standard and provides guidelines for solution development. They make sure that the overall solution is in alignment with the business objective and launch timeline, to ensure minimal chances of target date slippage.
* **Increased ROI**: Solution architecture determines the ROI and helps to measure the success of the project. Solution architecture forces a business to think about how to reduce costs and remove process wastage by applying automation in order to improve the overall ROI.
* **Market opportunity**: Solution architecture involves the process of analyzing and continuously evaluating the latest trends in the market. It also helps with backing up and promoting new products.
* **Budget and resourcing**: For a better budget, it is always recommended to invest well in estimation. A well-defined solution architecture helps us to understand the amount of resources that are required for project completion. This helps in the formulation of better budget forecasting and resource planning.
* **Project timeline**: Defining an accurate project timeline is critical for solution implementation. A solution architect determines the resources and effort that will be required during the design phase, which should help define the schedule.

Now you have had a high-level overview of solution architecture and its benefits, Let's investigate more closely the everyday aspects of solution architecture.

**Addressing the business needs and quality of delivery**

In the life cycle of product development, the most challenging phase is to establish the nature of the requirements, especially when multiple elements are competing to be addressed as high priority and are evolving rapidly. This is even more challenging when there are different views of the same requirement from various stakeholders. For example, a business user analyzes the page design from a user point of view, while a developer is looking at it from the perspective of implementation feasibility and load latency. This can cause conflicts and misunderstandings of the requirements between functional and technical members. In such cases, solution architecture helps to bridge the gap and define a standard that all members can understand.

Functional requirements are product features to accommodate user requirements and address the primary need of a given business problem. When users interact with software applications, they interact with functional requirements directly. For example, in an e-commerce application, examples of functional requirements are that users see their order history, search for merchandise, add them into the cart, and make payment using their preferred payment method. While the primary responsibility for the collection of functional requirements resides with the product owner, a solution architect makes sure of their design and implementation in such a way that it can scale as per user demand and accommodate any future extension.

Solution architecture defines standard documentation that explains the technical aspects to non-technical stakeholders and updates them regularly. As a solution architecture's design spans across the organization and different teams, it can help to discover hidden requirements. The solution architect makes sure that the development team knows about the requirements, and also maintains the cycle of progress.

A good solution architecture defines not only the solution design but also the success criteria in the form of qualitative and quantitative output, in order to ensure the quality of delivery. The qualitative output can be collected from user feedback, such as through sentiment analysis, while quantitative output may include latency, performance, load time on the technical side, and sales numbers on the business side. Taking continuous feedback and adapting to it is the key to high-quality delivery, which should adhere to all the phases of solution design and development.

**Selecting the best technology platform**

In a rapid and competitive market, the biggest challenge is maintaining the use of the best technologies. Today, when you have multiple resources all around the world, a specific technology should be chosen very carefully. The solution is the architecture design process, which can effectively tackle this problem.

The selection of the technology stack plays a significant role in efficient solution implementation by the team. In solution architecture, we should use different strategies to adopt various platforms, technologies, and tools. A solution architect should validate all of the needs carefully, and then evaluate and investigate the result using multiple parameters in order to find the best-fit solution for the product development by creating a working model of the product in the form of a prototype.

Good solution architecture addresses the depth of different tools and technologies by investigating all possible architectural strategies, based on the mixed use case, techniques, tools, and code reuse, which come as part of years of experience. The best platform simplifies the implementation process; however, the right technology selection is critical. This can be achieved by building a prototype according to the business requirement assessment, and the agility, speed, and security of the application.

**Addressing solution constraints and issues**

Any solution can be limited by various constraints and may encounter issues due to further complexities or unforeseen risks. Solution architecture needs to balance multiple constraints, such as resources, technologies, cost, quality, time to market, and frequently changing requirements.

Each project has its own specific goal, requirement, budget, and timeline. Solution architecture evaluates all of the possible critical paths and shares best practices to achieve a project goal in a given timeframe and budget. This is a systematic approach, where all tasks are interdependent of prior tasks; in order to achieve success in the project, all tasks need to be executed in sequence. A delay in one task can impact the project timeline and can result in the organization missing the market window to launch the product.

If there is an issue in the project development process, the probability of a project getting delayed is high. Sometimes, you encounter problems that are limitations of the adopted technology, or even of the solution environment. If you have a well-thought-out solution architecture, the most common issues are related to the non-functional requirements: resources and budgeting can mitigate problems encountered in the product development life cycle.

A solution architect helps to drive the project by diving deep into each component of it. They think of an out-of-the-box idea to save the project from unforeseen issues, such as those covered in disaster recovery, and will prepare a backup plan in the event that things do not work out with the main one. They evaluate the best possible way to execute the project by choosing the best practice and balancing constraints.

**Helping in resource and cost management**

There are always risks and uncertainties involved during solution implementation; it can become very tedious should a developer need to spend time on fixing a bug, for example. A good solution architecture controls the cost and budget and reduces uncertainty by providing developers with the required guidance in terms of priority, different communication services, and the details of each component.

Solution architecture also creates documentation that will be used to keep the system up to date, along with a deployment diagram, software patches, and a software release version, and enforces the runbook to tackle frequent issues and business continuation processes. It also addresses the indirect impacts of the cost of building a solution by considering extensibility, scalability, and other external factors that matter to the development environment.

**Managing solution delivery and project life cycle**

Lots of planning is involved in the inception stage of solution architecture. Solution architecture starts with a strategic view and provides more technical implementation input as you move forward with the solution implementation.

Solution architecture ensures an end-to-end solution delivery and impacts the overall project life cycle. It defines a process standard for different phases of the project life cycle and makes sure that it is applied across the organization so that other dependencies can be addressed as the implementation moves forward.

Solution architecture considers a holistic view of the project. It keeps syncing other dependent groups, such as security, compliance, infrastructure, project management, and support, in order to keep them engaged in different phases of the project life cycle as required.

**Addressing non-functional requirements**

Often, a solution architect has to deal with the **non-functional requirements** (**NFRs**) in an application. For project success, it is essential to address them, as they have a broader impact on the overall project and solution. These NFRs can make or break your user base, and address critical aspects of a solution, such as security, availability, latency concerns, maintenance, logging, masking confidential information, performance concerns, reliability, maintainability, scalability, and usability. If these are not considered on time, it can impact your project delivery. *Figure 1.3* shows some of the most common NFRs.

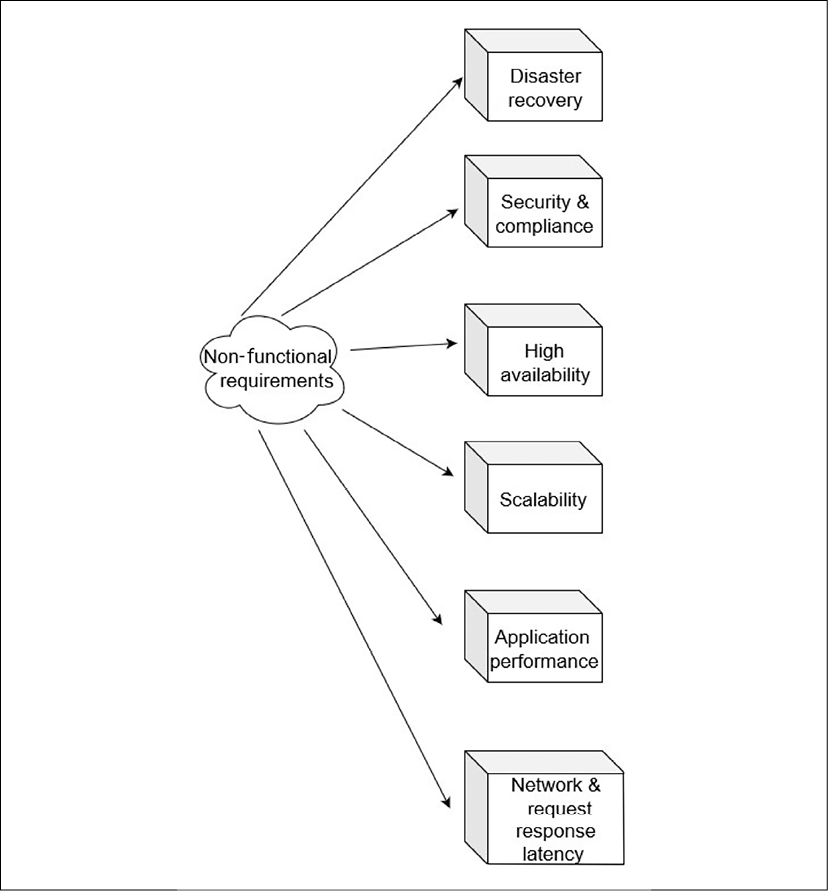


Figure 1.3: Non-functional attributes of solution architecture

As shown, NFRs include the following attributes of solution architecture. However, there can be more NFRs, depending on the type of project:

* **Disaster recovery**: To make sure the solution is up and running in case of any unforeseen events.
* **Security and compliance**: Put a safety net in place for a solution to save it from an external attack, such as a virus, malware, and so on. Also make sure that the solution complies with local and industry laws by meeting compliance requirements.
* **High availability**: To make sure the solution is always up and running.
* **Scalability**: To make sure the solution can handle the additional load in case of increasing demands.
* **Application performance**: To make sure the application is loading as per user expectation, and without much delay.
* **Network request and response latency**: Any activity performed on the application should be completed within an appropriate time, and should not be allowed to time out.

You will learn more details of the preceding attributes in *Chapter 3*, *Attributes of the Solution Architecture*. The solution architecture defines an initial framework for product development and the building blocks of the solution. While establishing a solution architecture, quality and customer satisfaction are always the main focus. Solution architecture needs to be built continuously by working on a proof of concept and exploring and testing until the desired quality is reached.

**Solution architecture in the public cloud**

Solution architecture in the cloud has become increasingly important these days and is becoming the "new normal" as more enterprises choose to migrate their workload to it. The public cloud has been a critical factor fueling start-up organizations' growth, as they do not need huge upfront investment. It provides flexibility to organizations to be run as an experiment, to be agile and innovative.

The great thing about cloud computing architecture is that you have an end-to-end view of all architecture components, including the frontend platforms, the application development platform, servers, storage, database, automation, delivery, and the networks that are required to manage the entire solution landscape.

Before jumping into solution architecture in the cloud, let's understand more about the public cloud and how it is becoming an essential and driving technology platform for businesses.

**What is the public cloud?**

The public cloud is based on the standard computing model in which a service provider makes resources, such as virtual machines, applications, and storage, available to their customers over the internet. Public cloud services offer a *pay-as-you-go* model.

In the cloud computing model, a public cloud vendor provides on-demand availability of IT resources, such as the server, database, network, and storage, which organizations can use with secure web-based interfaces, or through application programs over the internet. In most cases, the customer only pays for the services that they are using for the duration of utilization, which saves costs for them by optimizing IT resources to reduce idle time.

You can think of the public cloud in terms of an electric power supply model, where you switch on the light and pay only for the amount of electricity you use in units. As soon as you switch it off, you are not paying for it. It abstracts you from the complexity of power generation using turbines, resources to maintain the facility, a large infrastructure setup, and you use the entire service in a simplified way.

In addition to cost benefits, major public cloud providers, such as **Amazon Web Services** (**AWS**), **Google Cloud Platform** (**GCP**), and **Microsoft Azure** help to bring innovation by extending their technology platform through the cloud. These public cloud providers have mastered the scalability and future-looking architecture with comprehensive machine learning and analytics. With the public cloud, you get access to these cutting-edge technologies and the option of considering them to advance your architecture.

**Public clouds, private clouds, and hybrid clouds**

Here, you will get a high-level overview of the different types of cloud computing deployment models. You will learn more about the details in *Chapter 5*, *Cloud Migration and Hybrid Cloud Architecture Design*.

A **private cloud**, or **on-premises**, is registered to a single organization that owns and accesses it. Private clouds act as a replicate or extension of the company's existing data center. Often, a **public cloud** has a shared tenancy, which means virtual servers from multiple customers share the same physical server; however, they offer dedicated physical servers to customers if the customer wants it for a license or compliance need. A public cloud, such as AWS, Microsoft Azure, or GCP, utilizes massive IT infrastructure that can be accessed over the internet through a pay-as-you-go model.

A third model is the **hybrid cloud**, used by large enterprises who are moving their workload from on-premises to a cloud, where they still have a legacy application that cannot move to the cloud directly, or maybe they have a licensed application that needs to stay on-premises—or sometimes, due to compliance reasons, they need to secure data on-premises. In such a situation, the hybrid model helps when the enterprise has to maintain a partial environment on-premises and move other applications to the public cloud. Sometimes an organization moves to test and develop the environment to the public cloud and keep production environments on-premises. A hybrid model can vary depending upon the organization's cloud strategy.

As there are multiple public cloud providers in the market, you may start seeing the trends of **multi-cloud**. Enterprises choose to distribute their workload between different public cloud vendors to get the most out of each cloud technology or provide options to their team depending on their skill set.

**The public cloud architecture**

A typical definition of the public cloud is that it is a fully virtualized environment, which is accessible both over the internet or through a private network. However, in recent times, public cloud vendors have also started offering an on-premises physical infrastructure for better hybrid cloud adoption. The public cloud provides a multi-tenancy model, where IT infrastructure, such as storage and computational power, are shared between multiple customers; however, they are isolated at the software and logical network levels and do not interfere with each other's workload. In the public cloud, by creating network-level isolation, organizations can have their virtual private cloud, which is equivalent to the logical data center. Looking at organizations' regulatory needs, the public cloud also provides dedicated physical instances, however, those are also accessible over the web, but this is a less common option.

Public cloud storage achieves high durability and availability by creating a redundancy model using multiple data centers and robust data replication. This makes them achieve architecture resiliency and easy scalability. There are three major types of cloud computing models, as shown in *Figure 1.4*.

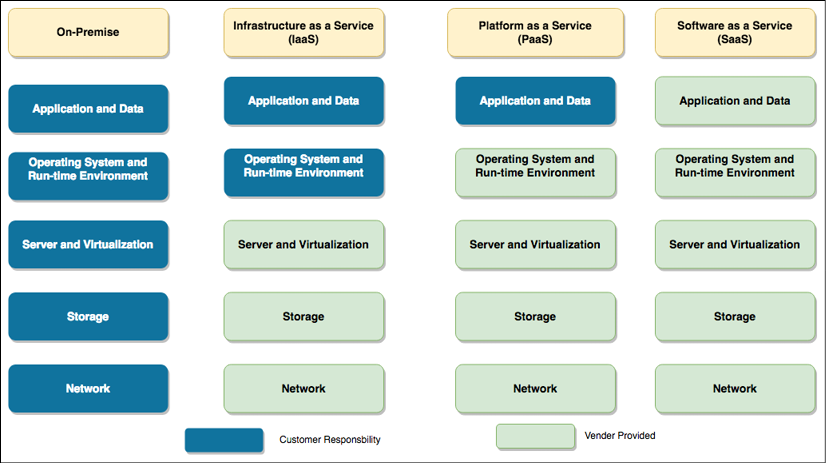


Figure 1.4: Types of cloud computing models

In *Figure 1.4* you can see a comparison between customer responsibilities in the on-premises environment with the cloud computing service model. In the on-premises environment, the customer has to manage everything, while in the cloud computing model, customers can offload responsibilities to the vendor and focus on their business needs. The following points are high-level details of services that are provided under different cloud computing models:

* **Infrastructure as a Service** (**IaaS**): Here, a cloud vendor provides infrastructure resources, such as a compute server, networking components, and data storage space, as managed services. It helps customers to use IT resources without worrying about handling data center overheads, such as heating and cooling, racking and stacking, physical security, and so on.
* **Platform as a Service** (**PaaS**): The PaaS model adds a layer of service where the cloud vendor takes care of the resources that are required for your development platform, such as the operating system, software maintenance, and patching, along with infrastructure resources. The PaaS model facilitates your team's focus on writing business logic and handling data by taking care of all of the burdens of platform maintenance for you.
* **Software as a Service** (**SaaS**): The SaaS model adds one more layer of abstraction on top of the PaaS and IaaS models, wherein the cloud or software vendor provides ready-to-use software, and you pay for the service. For example, you use email services such as Gmail, Yahoo! Mail, AOL, and so on, where you get your own space for emails as a service, and you don't have to worry about underlying applications or infrastructures.

The fourth emerging model is the **Function as a Service** (**FaaS**) model, which is becoming popular in the building of serverless architecture through using services including AWS Lambda. You will learn more details about serverless architecture in *Chapter 6*, *Solution Architecture Design Patterns*.

As the public cloud functionality and the cost model are very different, let's learn how to develop a cloud-native approach to architecture design.

**Thinking cloud-native architecture**

With the increasing adoption of the cloud, cloud-native architecture is an upcoming trend that optimizes system architectures for cloud capabilities. Typical on-premises architecture is normally built for a fixed infrastructure, as adding new IT resources such as servers and computing power adds a considerable amount of time, cost, and effort. However, the cloud is charged based on usage, and provides ease through automation, such as in the scaling of servers up and down, on-demand, without worrying about a long procurement cycle. Cloud-native architecture primarily focuses on achieving on-demand scale, distributed design, and replacing failed components rather than fixing them.

The public cloud is not just about infrastructure, but most public cloud providers offer a broad range of managed services that allow the user to ignore underlying infrastructure and operation maintenance. For example, AWS provides Lambda, a serverless computing platform that can be used to run code without managing the server or runtime environment. Similarly, the Amazon DynamoDB database is highly scalable; tables can be created and data stored without managing a database server. Managed services make it easy to develop scalable applications rapidly.

In cloud-native architecture, you continually create automated operations for recovery, scalability, self-healing, and high availability using the cloud capabilities of continuous integration, deployment, and infrastructure automation. It encourages the continuous optimization of your application in terms of cost and performance, using new cloud capabilities that are released and improved upon every day.

You will learn more details about cloud-native architecture patterns in the next chapter.

**Public cloud providers and cloud service offerings**

There are several public cloud providers in the IT industry; among them, the key players are AWS, GCP, Microsoft Azure, and Alibaba Cloud. These providers offer an array of services, from computing, storage, networking, databases, and application development, to analytics and machine learning.

*Figure 1.5* is a screenshot from the AWS console; you can see the array of services on offer in multiple areas. The highlighted EC2 service, known as Amazon Elastic Compute Cloud, allows you to spin up a virtual machine in minutes in the AWS cloud.

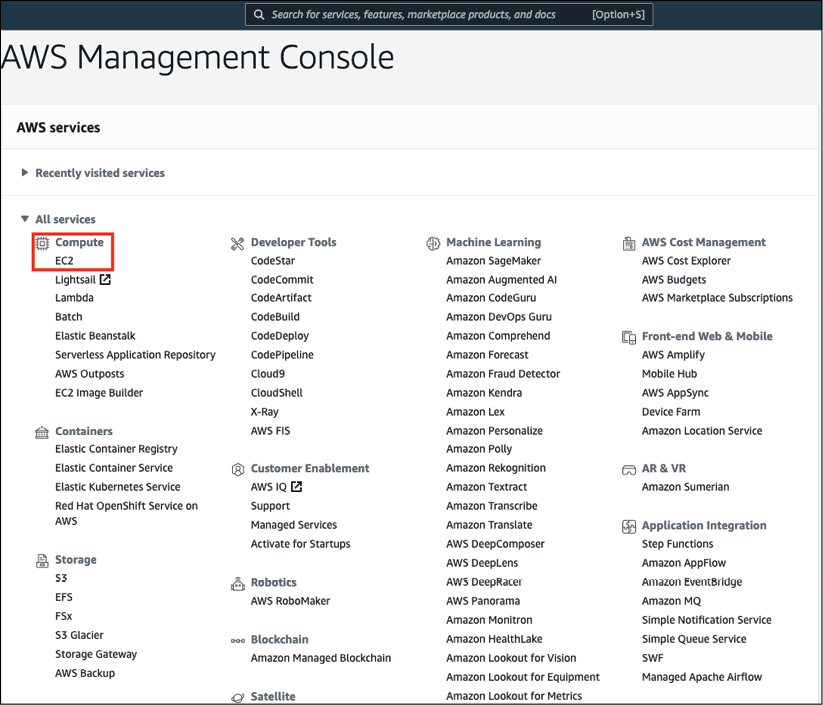


Figure 1.5: AWS console and service offerings

Public cloud vendors provide infrastructure and facilitate an array of services in various areas, such as analytics, machine learning, blockchain, robotics, application development, email, security, monitoring, and alerting. With the public cloud, different technical capabilities become more accessible to the development team, which helps drive innovation and reduce the time to market for the product launch.

Public cloud providers allow global infrastructure to spread across the world, which helps an application to be scaled globally near your user base. To encourage adoption, all cloud services provide a free-tier service, with lots of learning resources, so you can try your hand and develop your knowledge of them.

**Summary**

In this chapter, you have learned about the definition of solution architecture from industry standards in a simplified form. You learned about the importance of solution architecture, and how it can help an organization to achieve more significant results in maximizing the return on its investments. This chapter helped you to understand the benefits of having a solution architecture, and how it helps in different aspects of solution design and implementation.

In summary, solution architecture is a building block in a complex organization and is used to address all stakeholders' needs and establish a standard in order to fill the gap between business requirements and technical solutions. A good solution architect not only addresses functional requirements, but also puts long-term thought into, and takes care of, non-functional requirements, such as scalability, performance, resiliency, high availability, and disaster recovery. Solution architecture finds an optimal solution to accommodate the constraints of cost, resources, timelines, security, and compliance.

You have also explored the basics of cloud computing, solution architecture in the cloud environment, and the significant public cloud providers and their service offerings. This also aided in the gaining of a high-level overview of different cloud computing models, such as IaaS, PaaS, and SaaS, and the cloud computing deployment models in the public, private, and hybrid cloud. Finally, this chapter shed some light on the evolution of solution architecture design.

In the next chapter, you will learn all about the solution architect role itself—the different types of solution architect, the role's responsibilities with regards to solution architecture, and how these fit into an organizational structure and agile environment.

**Join our book's Discord space**

Join the book's Discord workspace to ask questions and interact with the authors and other solutions architecture professionals: <https://packt.link/SAHandbook>

