**18**

**Solution Architecture Document**

In previous chapters, you learned about various aspects of solution architecture design and optimization. As the solution architect works on the design, it is essential to have consistent communication with other stakeholders for successful application delivery. The solution architect needs to communicate a solution design to all technical and non-technical stakeholders.

The **Solution Architecture Document** (**SAD**) provides an end-to-end view of the application and helps everyone be on the same page. In this chapter, you will learn about various aspects of the SAD, which addresses the needs of all stakeholders associated with the development of the application.

You will learn about the structure of the SAD and other types of documents of which the solution architect needs to be aware, such as the request for proposal, where the solution architect needs to provide input to make strategic decisions. We will cover the following topics to gain a deeper understanding of the documentation involved in solution architecture:

* Purpose of the SAD
* Views of the SAD
* Structure of the SAD
* IT procurement documentation for a solution architecture

By the end of this chapter, you will know about the SAD, its structure, and the various details that need to be accommodated in the documentation.

You will learn about various IT procurement documentation such as the request for proposal, the request for information, and the request for quotation, in which a solution architect participates to provide feedback.

**Purpose of the SAD**

The need for architecture documentation often gets ignored, and teams start working on implementation without understanding the overall architecture. A SAD provides a broad view of the overall solution design to keep all stakeholders informed.

The SAD helps to achieve the following goals:

* Communicate the end-to-end application solution to all stakeholders
* Provide high-level architecture and different views of the application design to address the application's service-quality requirements such as reliability, security, performance, and scalability
* Provide traceability of the solution back to business requirements and look at how the application will meet all functional and **non-functional requirements** (**NFRs**)
* Provide all views of the solution required for design, building, testing, and implementation
* Define the impacts of the solution for estimation, planning, and delivery purposes
* Define the business process, continuation, and operations needed for a solution to work uninterrupted after the production launch

SADs define the purpose and goal of the solution and address critical components such as solution constraints, assumptions, and risks that often get overlooked by the implementation team. The solution architect must make sure they create the document in an easy language that business users can understand and relate business context with technical design. Documentation helps to retain knowledge due to resource attrition and makes the overall design process a people-independent one.

For existing applications where modernization effort is needed, a SAD presents an abstract view of current and future architecture, along with a transition plan. The solution architect understands the existing system dependencies and documents them to uncover any potential risk in advance. The migration plan helps businesses understand the tools and technology required to handle the new system and plan resources accordingly.

The solution architect conducts various assessments during solution design by building a **proof of concept** (**POC**) or through market research. A SAD should list all architecture assessments and their impact, along with the choice of technology. A SAD presents a conceptual view of the current and target state of the solution design and maintains a record of change. Let's understand various aspects of a SAD in the next section.

**Views of the SAD**

The solution architect needs to create a SAD that is understandable by both business users and technical users. A SAD bridges the communication gap between the business user and the development team to understand the function of the overall application. The best way to capture all stakeholders' input is by putting yourself in their situation and looking at problems from the stakeholders' perspectives. The solution architect evaluates both the business and technical aspects of architecture design to take cognizance of all technical and non-technical users' requirements.

As illustrated in the following diagram, the holistic view of the SAD comprises various views derived from business requirements to cover different aspects:

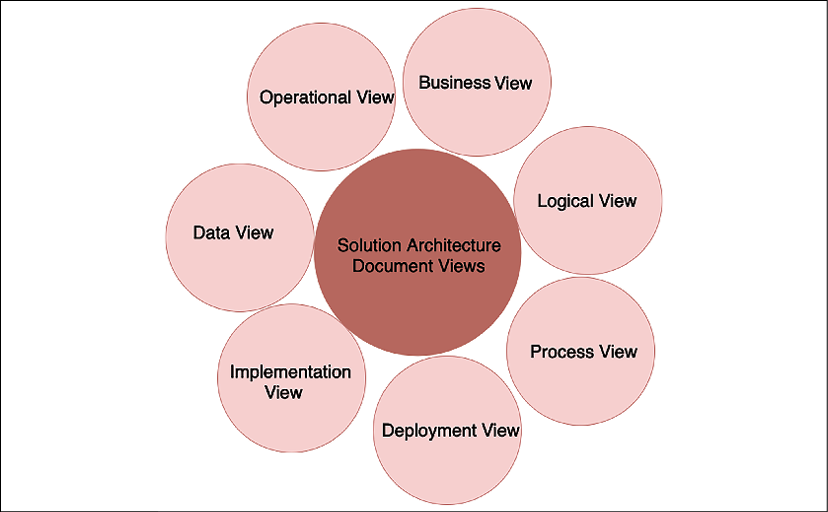


Figure 18.1: SAD views

Solution architects can choose standard diagrams such as a **Unified Modeling Language** (**UML**) diagram or a block diagram from **Microsoft Visio** to represent various views. Overall, the diagram should be easy to read and understandable by all business and technical stakeholders. A SAD should include the following views, wherever possible, to address everyone's needs:

* **Business View**: Architecture design is all about addressing business concerns and solving business purposes. The business view shows the value proposition of the overall solution and product. To simplify, the solution architect may choose to detect high-level scenarios related to business and present these as a use case diagram. The business view also describes stakeholders and the required resources to execute the project. You can define the business view as a use case view as well.
* **Logical View**: This presents various packages on the system so that business users and designers can understand the various logical components of the system. The logical view offers a chronicled order of the system in which it should build. It shows how the multiple packages of the system are connected and how the user can interact with them. For example, in a banking application, the user first needs to authenticate and authorize using a security package, log in to the account using the account package, apply for a loan using a loan package, and so on. Here, each package represents a different module and can be built as a microservice.
* **Process View**: This presents more details, showing how the key processes of the system work together. It can be reflected using a state diagram. The solution architect can create a sequence diagram if they want to show more details. In a banking application, a process view can present the approval of a loan or account.
* **Deployment View**: This presents how the application is going to work in the production environment. It shows how different system components (such as the network firewall, load balancer, application servers, and database) are connected. The solution architect should create a simple block diagram that business users can understand. You can add more details to the UML deployment diagram to show various node components and their dependencies for technical users, such as the development and DevOps teams. The deployment view represents the physical layout of the system.
* **Implementation View**: This is the core of the SAD and represents architectural and technology choices. The solution architect needs to put the architecture diagram here—for example, if it is 3-tier, *N-tier*, or event-driven architecture—along with the reasoning behind it.

You also need to detail technology choices—for example, using Java versus Node.js, along with the pros and cons of using them. You want to justify the resources and skills required to execute the project in the implementation view. The development team uses an implementation view to create a detailed design such as a class diagram, but that doesn't need to be part of the SAD.

* **Data View**: As most applications are data-driven, this makes the data view important. The data view represents how data will flow between the different components and how it will be stored. It can also be used to explain data security and data integrity. The solution architect can use the entity-relationship diagram to show the relationship between different tables and schemas in the database. You will learn more about the entity-relationship diagram in the *Data architecture* section. The data view also explains the reports and analytics needed.
* **Operational View**: This explains how the system is going to be maintained post-launch. Often, you define **service-level agreements** (**SLAs**), alert and monitoring functionality, a disaster recovery plan, and a support plan for the system. The operational view also provides details of how system maintenance will be carried out, such as by deployment of a bug fix, patching, backup and recovery, and handling security incidents.

All the views listed make sure the SAD covers all aspects of the system and stakeholders. You may choose to include additional views—such as a physical architecture view, a network architecture view, or a security (controls) architecture view—as per the stakeholders' requirements. As a solution architect, you need to provide a comprehensive view of system functioning and understanding. Let's explore the structure of the SAD in more detail in the next section.

**Structure of the SAD**

The structure of the SAD can differ from project to project as per stakeholder requirements and the nature of the project. Your project could be creating a new product from the ground up, modernizing a legacy application, or moving the entire system to the cloud.

For each project, the SAD document may differ, but, overall, it should consider various stakeholders' views and consider the necessary sections, as shown in the following screenshot:

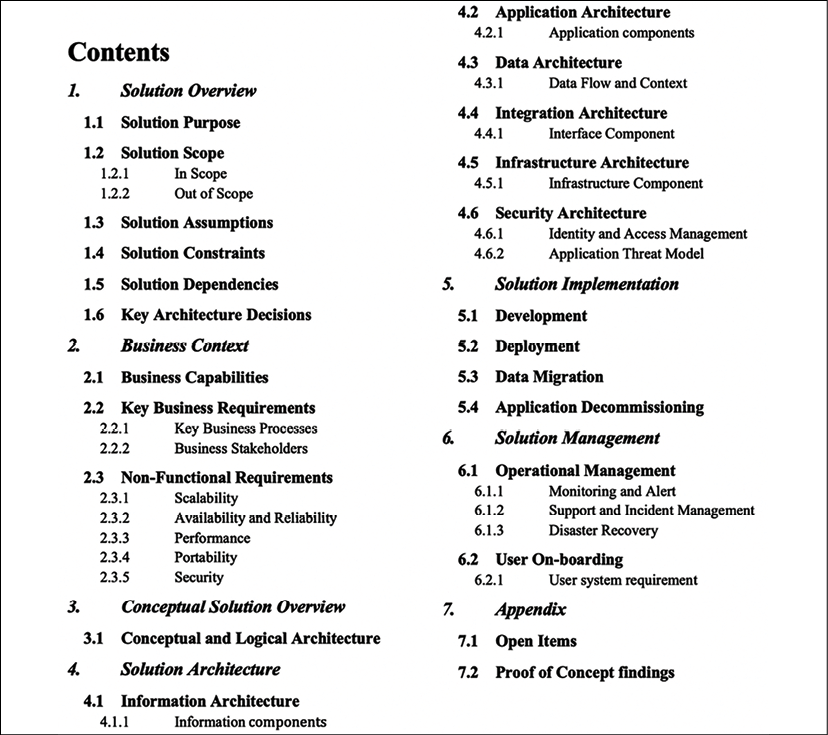


Figure 18.2: Structure of a SAD

In the preceding SAD structure, you can see different sections covering multiple solution architecture and design aspects. The solution architect may choose to add additional subsections or remove some sections as per the project requirements. For example, you can add another introduction section to talk about the document's purpose, with a summary. For a transition project, you may add a subsection to present the existing architecture and compare it with the target architecture, and so on. Let's look into the details of each section.

**Solution overview**

In the solution overview section, you need to briefly introduce the solution in a couple of paragraphs, describing the functioning of the solution and its different components at a very high level. It's nice to add a high-level block diagram showing various components in one place. The following diagram illustrates the solution overview of an e-commerce platform:

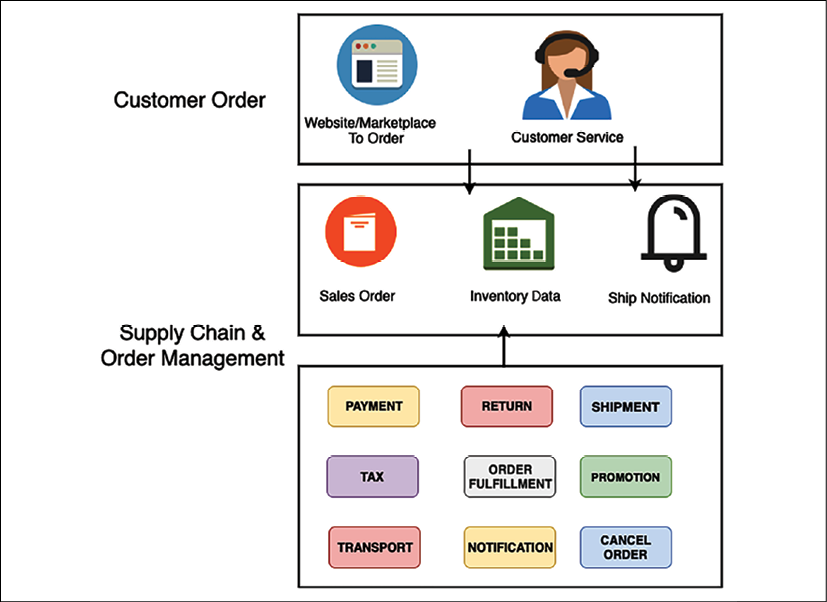


Figure 18.3: Solution overview of an e-commerce platform

You need to provide a brief about each component in simplified language so that the business user can understand the overall working of the solution. Major subsections include:

* **Solution purpose**: This provides a brief about a business concern that the solution is solving and the justification to build a given solution.
* **Solution scope**: This states the business scope that the proposed solution will address. **Clearly** Describes out-of-scope items that the solution will not accommodate.
* **Solution assumptions**: List down all the assumptions based on which solution architect came up with the solution—for example, minimum network bandwidth availability.
* **Solution constraints**: List all technical, business, and resource constraints. Often, constraints come from industry and government compliances, which need to be listed in this section. You can also highlight the risk and mitigation plan.
* **Solution dependencies**: List all upstream and downstream dependencies. For example, an e-commerce website needs to communicate with a shipping system such as UPS or FedEx to ship a package to customers.
* **Key architecture decisions**: List major problem statements and the corresponding proposed solution options. Describe the pros and cons of each option, why a particular decision was made, and the rationale behind it.

After giving a solution overview, you want to relate it to the business context. Let's look at the business context view in more detail in the next section.

**Business context**

In the business context section, the solution architect needs to provide a high-level overview of the business capabilities and requirements that the solution will address. This section only contains an abstract view of requirements. Detailed requirements need to be a part of a separate requirements document. However, the external link of the requirements document can be provided here. You should include the following primary subsections:

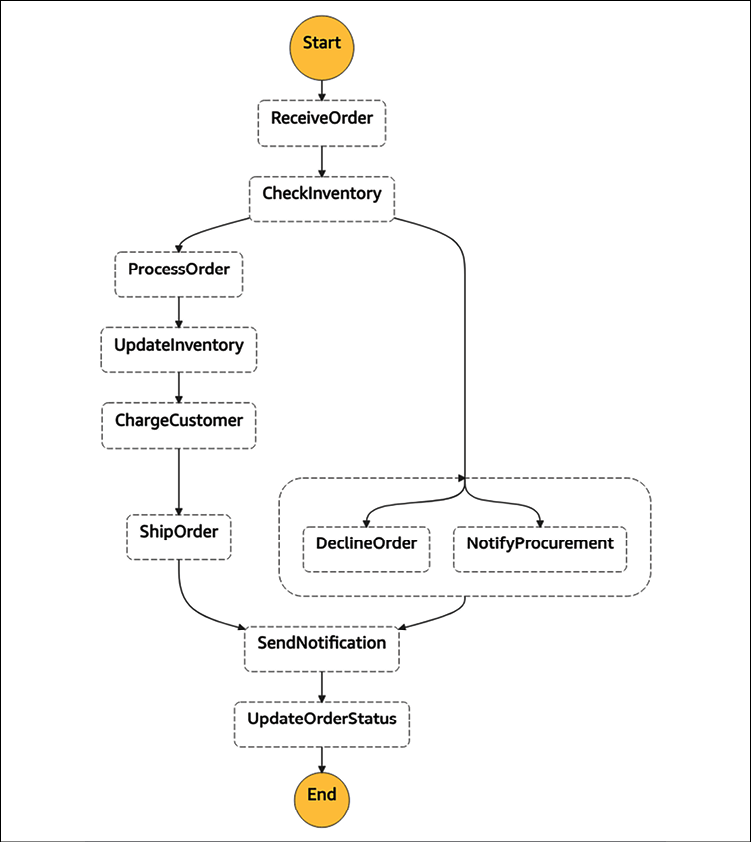
* **Business capabilities**: Provide a brief description of the business capabilities for which the solution is being designed. Make sure to include the benefits of capabilities and how they will address customer needs.
* **Key business requirements**: List all key business concerns that the solution is going to address. Provide a high-level view of key requirements and add a reference to the detailed requirements document.
* **Key business processes**: Solution architects should show key processes with a business process document. The following diagram illustrates a simplified view of an e-commerce application business process model:

Figure 18.4: Business process diagram of an e-commerce platform

* **Business stakeholders**: List stakeholders who are directly or indirectly impacted by the project. This includes sponsors, developers, end users, vendors, and partners.
* **NFRs** (**Non-Functional Requirements**): Solution architects need to focus more on NFRs as these often get missed by the business user and development team. At a high level, an NFR should include:
  + **Scalability**: How can the application scale as workloads fluctuate? (For example, scale from 1,000 transactions per second to 10,000 transactions per second in a given day or month.)
  + **Availability and reliability**: What is the acceptable downtime for system availability? (For example, 99.99% availability or 45 minutes' downtime per month.)
  + **Performance**: What is the performance requirement? Where can the system handle the load increase without impacting the end user experience? (For example, the catalog page needs to load within 3 seconds.)
  + **Portability**: Can the application run on multiple platforms without any additional work? (For example, the mobile app needs to run in the iOS and Android operating systems.)
  + **Capacity**: What is the maximum workload that the application can handle? (For example, the maximum number of users, the number of requests, the expected response time, and the expected application load.)

The conceptual view of architecture is a sweet spot that provides a good system overview for both business and technical stakeholders. Let's learn more about the conceptual view in more detail.

**Conceptual solution overview**

The conceptual solution overview section provides an abstract-level diagram that captures a big-picture view of the whole solution, including business and technical aspects. It provides a basis for analyses and trade-off studies to help refine and optimize the solution architecture in sufficient detail, to support solution design and implementation. The following diagram illustrates a conceptual architecture diagram of an e-commerce platform:

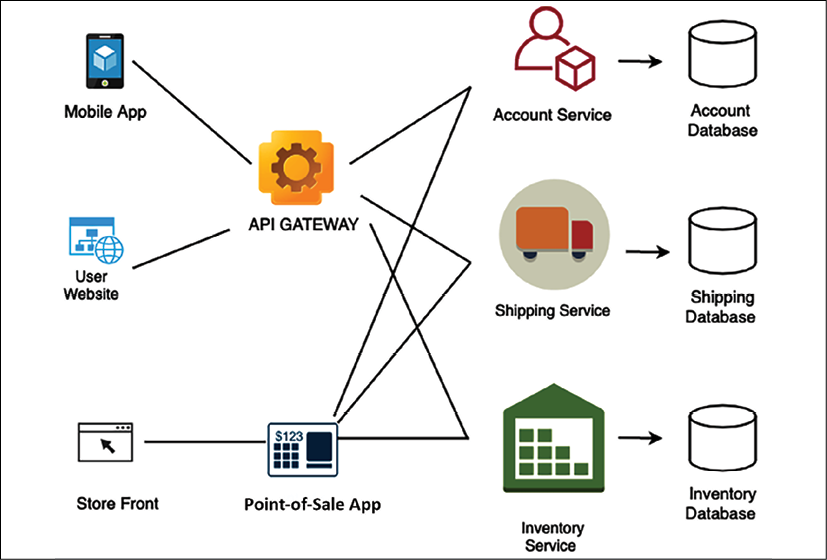


Figure 18.5: Conceptual architecture diagram of an e-commerce platform

The preceding diagram shows an abstract view of significant modules and information flowing between them. The conceptual architecture provides a good understanding of the overall architecture for both business and technical users. However, technical users need further architectural depth. Let's dive deeper into the solution architecture in the next section.

**Solution architecture**

The solution architecture section dives deep into each part of the architecture. It provides different views that the technical team can use to create a detailed design and work on implementation. These views could target other user groups, such as developers, infrastructure engineers, DevOps engineers, security engineers, and **user experience** (**UX**) designers.

Let's get into the following major subsections to learn more details.

**Information architecture**

This section provides a user navigation flow to the application. At a high level, the solution architect needs to put in an application navigation structure. As shown in the following diagram, for an e-commerce website, it is taking three clicks for the user to navigate to the desired page:

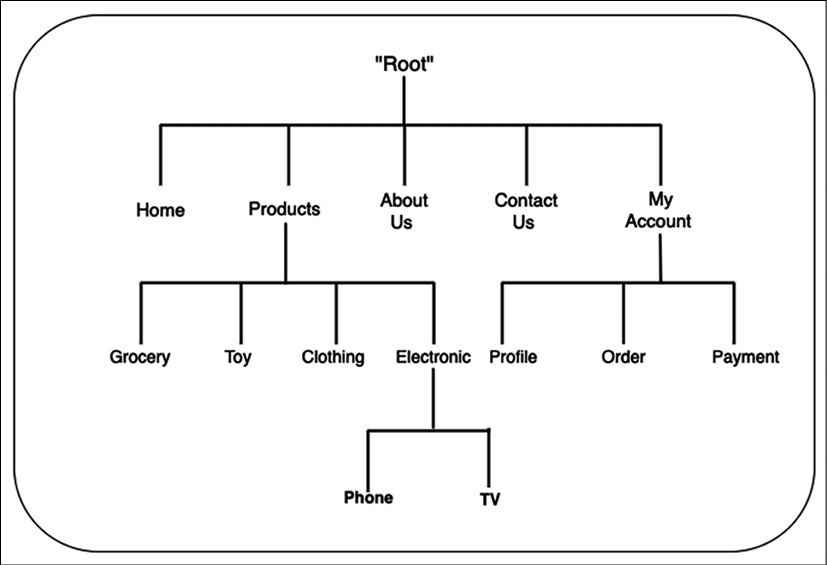


Figure 18.6: Informational architecture diagram of an e-commerce platform

Solution architects can add more details, such as website navigation, taxonomy, or a high-level wireframe that UX designers can use to generate a detailed wireframe.

**Application architecture**

This section targets the development team. It provides more implementation details upon which a software architect or development team can build a detailed design. The following diagram shows the application architecture for an e-commerce website, with technology building blocks such as caching, networking, content distribution, and data store:

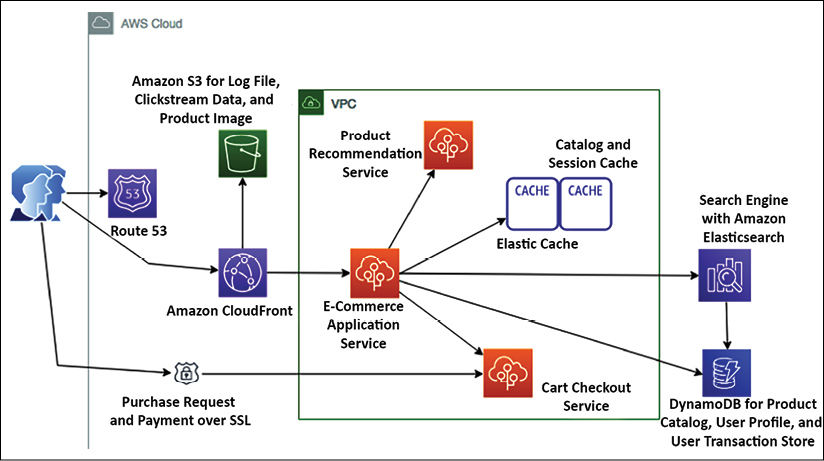


Figure 18.7: Application architecture diagram of an e-commerce platform

This section lists all application modules that need to be retired, retained, replatformed, and transformed for an application modernization architecture.

**Data architecture**

This section is primarily utilized by the database admin and development team to understand database schemas and how tables are related. This section often includes an **entity-relationship diagram** (**ERD**) showing the relationships of entity sets stored in a database, as shown in the following screenshot:

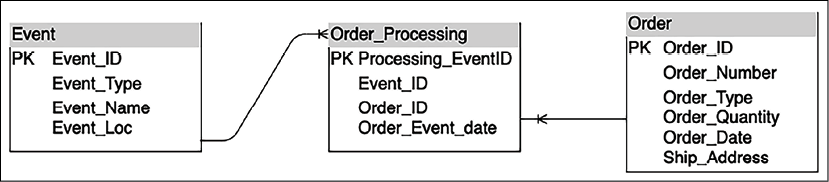


Figure 18.8: ERD of an e-commerce platform

The data architecture section lists all data objects that need to be considered during application development.

**Integration architecture**

This section mainly targets vendors, partners, and other teams. For example, the following diagram shows all integration points with other systems for an e-commerce application:

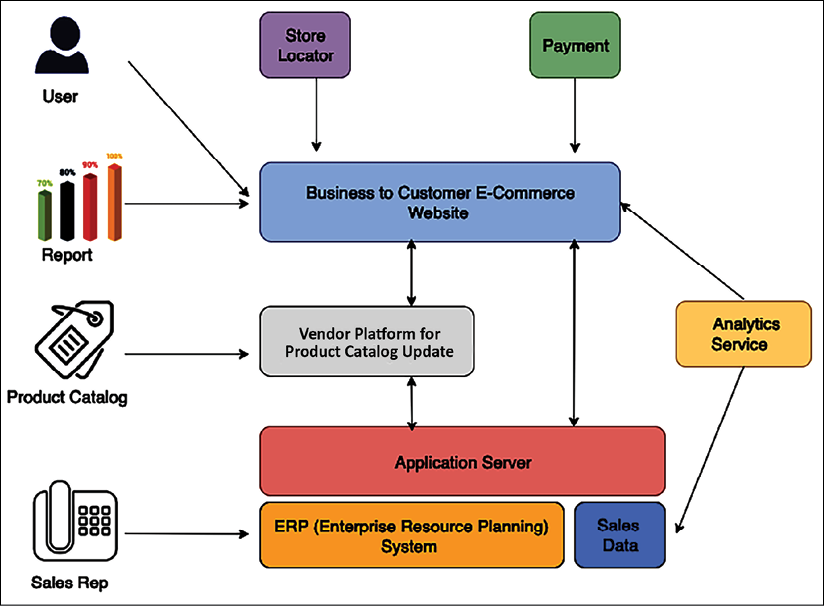


Figure 18.9: Integration architecture diagram of an e-commerce platform

The integration architecture section lists all upstream and downstream systems and their dependencies regarding your application.

**Infrastructure architecture**

This section is primarily targeted at the infrastructure team and system engineers. The solution architect needs to include the deployment diagram to view the logical server location and its dependencies.

For example, the following diagram illustrates the production deployment diagram for an e-commerce application. You can produce a separate diagram for other environments, such as dev, **quality assurance** (**QA**), and **User Acceptance Testing** (**UAT**) environments:

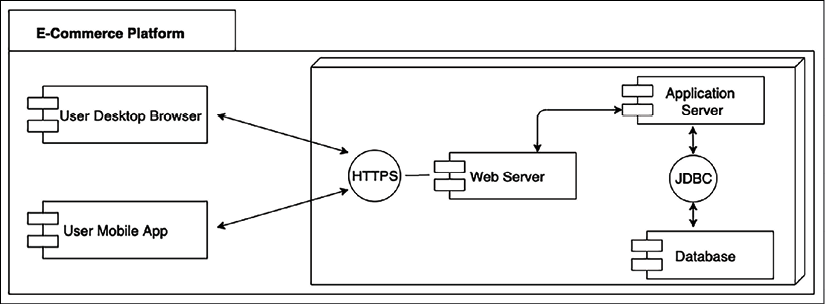


Figure 18.10: Deployment diagram of an e-commerce platform

This section lists all server configurations, databases, networks, and switches to deploy the application.

**Security architecture**

This section includes all the security and compliance aspects of the application, including:

* **Identity and Access Management** (**IAM**) such as **Active Directory** (**AD**), user authentication, and authorization management
* **Infrastructure security** such as the firewall configuration, **intrusion prevention system** (**IPS**)/**intrusion detection system** (**IDS**) required, and antivirus software
* **Application security** such as WAF and **Distributed Denial of Service** (**DDoS**) protection
* **Data security** at rest and in transit using **Secure Sockets Layer** (**SSL**), encryption algorithms, key management, and so on

Overall, the solution architect can include an application security threat model to identify any potential vulnerabilities, such as **cross-site scripting** (**XSS**) and **SQL injection** (**SQLi**), and plan to protect the application from any security threat.

**Solution implementation**

The solution delivery section includes essential considerations to develop and deploy a solution. It can consist of the following major subsections:

* **Development**: This section is essential for the development team. It talks about development tools, programming language, code repository, code versioning, and branching, with the rationale behind the choices.
* **Deployment**: This section mainly focuses on DevOps engineers and talks about the deployment approach, deployment tools, various deployment components, and deployment checklist, with the rationale behind the choices.
* **Data migration**: This section helps the team to understand data migration and the ingestion approach, the scope of data migration, various data objects, data ingestion tools used, sources of data and data formats, and so on.
* **Application decommissioning**: This section lists existing systems that need to be decommissioned and an exit strategy for the current system if the **return on investment** (**ROI**) is not being realized. The solution architect needs to provide an approach and timeline for decommissioning the old system and carry out an overall impact assessment.

The SAD includes a development approach and tools. However, it does not have an application-level detailed design, such as a class diagram or adding pseudocode. Such details need to be handled by the software architect or senior developer under the corresponding software application details design document. As a solution gets deployed, it needs to be managed in production. Let's learn about the details that go into the solution management section.

**Solution management**

The solution management section is focused on production support and ongoing system maintenance across other non-product environments. The solution management section is primarily targeted at the operations management team. This section addresses the following areas:

* Operational management such as system patching and upgrades of dev, test, staging, and prod environments
* Tools to manage application upgrades and new releases
* Tools to manage system infrastructure
* System monitoring and alerts; operations dashboard
* Production support, SLA, and incident management
* Disaster recovery and **Business Process Continuation** (**BPC**)

A solution architect needs to do research and collect data to validate the right solution during solution design. Such kinds of additional details can be put in the Appendix section. Let's learn more details of the Appendix section of a SAD.

**Appendix**

Like every business proposal document, a SAD also has a pretty open Appendix section for containing any data that supports your overall architecture and solution choices. In the Appendix section, the solution architect can include open issues and any research data, such as the outcome of the POC, tools comparison data, and vendors' and partners' data.

In this topic, you got a good overview of the SAD structure with different sections. A SAD should include the major sections mentioned previously; however, the solution architect may exclude some sections or include additional sections as per organization and project requirements. As with other documents, it's essential to continue to iterate upon SADs and look for an opportunity to improve. More robust SADs lead to well-defined implementation guidelines and reduce any risk of failure.

A SAD is a running document created during the initial stages and kept up to date over the years based on various changes throughout the application life cycle. In addition to the SAD, solution architecture often gets involved in a significant procurement proposal with a specific requirement known as a **request for x** (**RFx**) document. Let's become familiar with RFx documents.

**IT procurement documentation for a solution architecture**

IT procurement documents are popularly known as **RFx documents**. This is a term that includes different stages of the procurement process. When you refer to RFx, it references the formal requesting process. RFx documents are categorized as **request for proposal** (**RFP**), **request for information** (**RFI**), and **request for quotation** (**RFQ**) documents.

Solution architects are often involved in the procurement process to provide their input or lead them. These procurements may be related to outsourcing, contracting, procuring software such as a database or development tools, or buying SaaS solutions.

As these documents could be highly technical and have a broad, long-term impact, the solution architect needs to provide input or respond to any procurement requirement and prepare the invite. Let's understand the difference between different RFx documents, as follows:

* **RFI**: RFI comes early in the procurement process, where buyers invite information from different vendors to make an informed decision regarding their choice of procurement for a later stage. An RFI document collects information about the capabilities of the various suppliers, where the buyer can compare all suppliers in a similar parameter and proceed to the next proposal steps with shortlisted suppliers.
* **RFP**: In this process, shortlisted suppliers from the RFI process have more information about the project's outcome. An RFP document is more open than an RFI one, where suppliers can provide the best way to acquire solutions for the buyer. The supplier can include multiple choices, with the pros and cons of each approach.
* **RFQ**: In this process, buyers narrow down the requirement compared to the RFP and list down the exact requirements of work, equipment, and supplies. Suppliers need to provide a cost for the listed requirements, and the buyer can choose the best quotation among them to award the contract.

RFP is the most popular choice, as often, to speed up the process, the buyer's organization often chooses to ask for the RFP document only from potential vendors. In such a situation, the RFP document needs to have the structure in place so that that buyer can put a clear comparison between preferred vendors in terms of capabilities, solution approach, and cost to make a quick decision.

Due to the technicalities of procurement in IT organizations, solution architects play an essential role in evaluating vendors' capabilities and approaches from the buyer side and responding to RFP documents from the supplier side.

**Summary**

A SAD aims to keep all stakeholders on the same page and get formal agreement on solution design and requirements. As stakeholders comprise both business and technical users, you learned about various SAD views that the solution architect needs to consider. You need to include views for non-technical users, such as business, process, and logical views. For technical users, include views such as application, development, deployment, and operational views.

In this chapter, you learned about the detailed structure of the SAD, with major sections and subsections.

Various sections of the SAD include details such as an overview of the solution, business, and conceptual architecture. In the architecture diagram, you also learned about various architecture views, such as application, data, infrastructure, integration, and security. You learned about other sections for solution delivery consideration and operations management.

It was a long journey of learning. You are almost at the end of the book, but before closing, you need to learn some tips for becoming a solution architect and continuing to improve your knowledge.

In the next and final chapter, you will learn various soft skills such as communication style, ownership, critical thinking, and continuous learning techniques to become a better solution architect.