6.1 What are the advantages of using services as the fundamental component in a distributed software system?

A distributed database is a collection of databases that can be stored at different computer network sites. Each database can be managed by a different database management system and have different architectures that facilitate the distribution of transaction execution.

Objective:

1. To control the management of a distributed database (DDB) in such a way that it appears to the user as a centralized database.
2. Free object naming
3. Providing the appearance of a centralized database system.

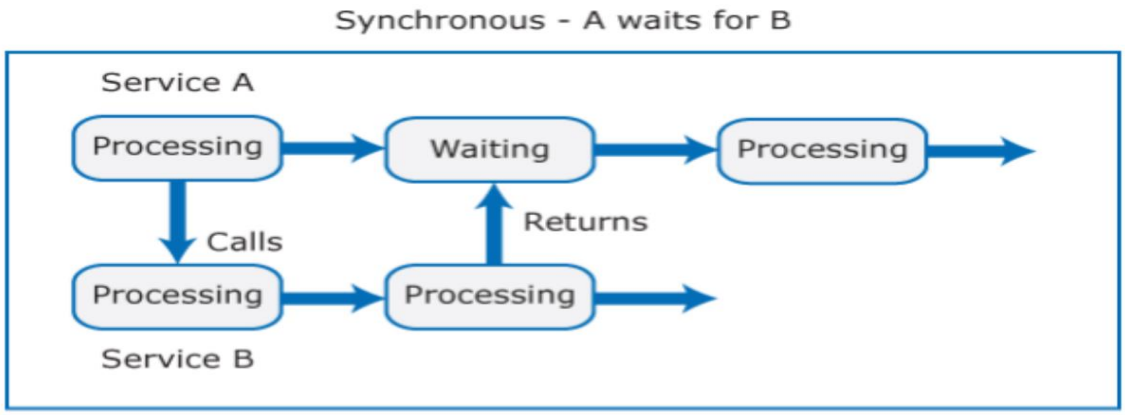
Advantages

- A software service is a software component that can be accessed from remote computers over the internet. Given an input, a service produces a corresponding output, without side effects.
- The service is accessed through its published interface and all details of the service implementation are hidden. Services do not maintain any internal state. As there is no local state, services can be dynamically reallocated from one virtual server to another.
- Service is related to a single business function, instead of relying on a shared database and other services in the system. This type of service is known as a microservice. Microservices are small–scale, stateless services that have a single responsibility.
- The software services support high availability, consistency, and scalability with ease. Services provide resiliency, fault tolerance, precision, and accuracy.
- The software services provide security, flexibility, and reliability for distributed software and applications.
- Users would be able to access the main functionality of the software and in turn, the system, even if some system's components have malfunctioned or are down.

6.5 Explain the differences between synchronous and asynchronous microservices interactions.

Synchronous communication:

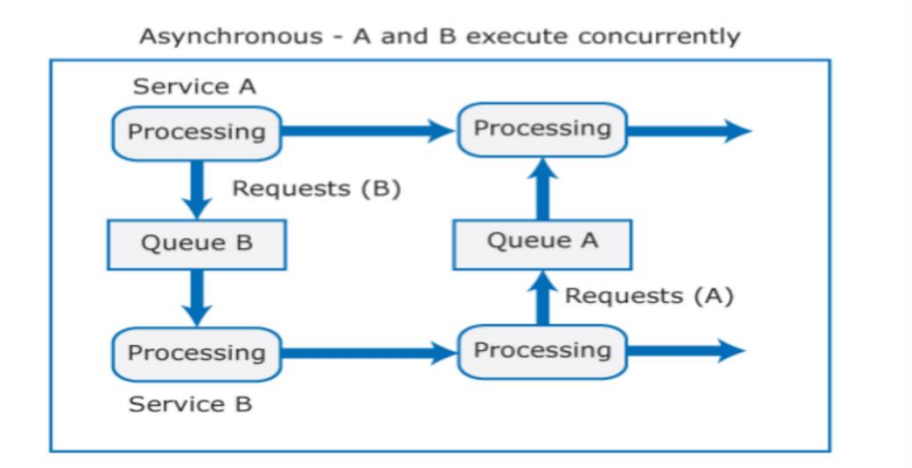
- Synchronous communication is the **most straightforward solution** when trying to make services communicate. Like a phone call, the client sends a request and waits for a response to come back. A synchronous request is considered blocking: the response is needed for the process to continue. If you don't answer the phone, the person calling you will not be able to continue.
- Most synchronous communication technologies are built around HTTP, including examples like gRPC, REST, or GraphQL.



- While synchronous communication is **ideal for querying data** (returning a result without modifying the system's state), it can have some drawbacks for commands (changing the system's state with possible side effects).
- Indeed, when a command **involves multiple services performing an action** or other services reacting to it (think of a distributed transaction on an e-commerce website that involves the payment, inventory & order services), it can become really hard to keep track of all services to call using synchronous communication. If one call fails, we don't want the whole order to be canceled.
- **Less Complexity**
- Synchronous programs are **easier** to write and understand.
- **Fewer difficult-to-find** bug

Asynchronous communication

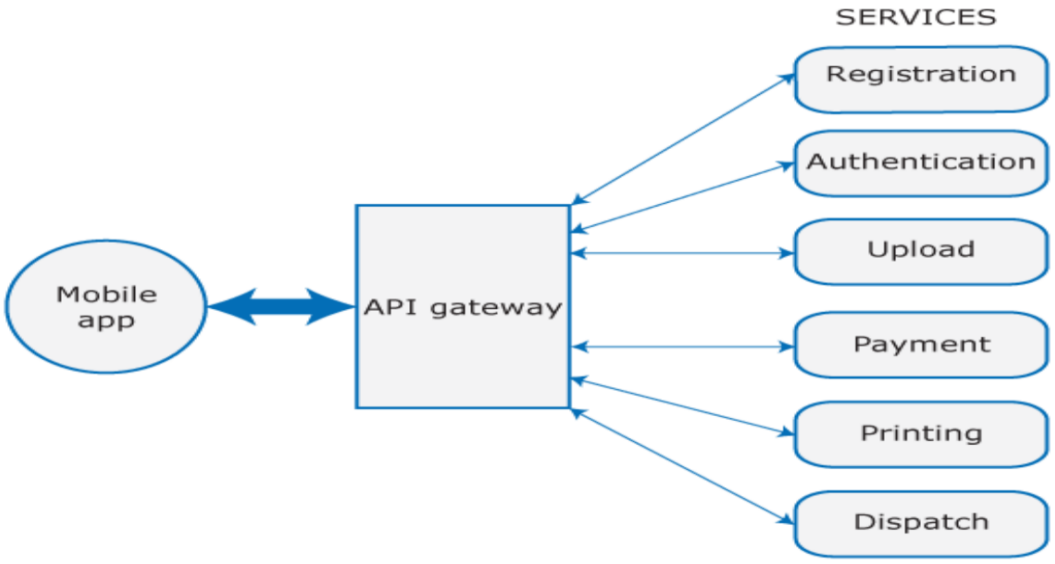
With asynchronous communication, a middleman is added to our infrastructure between services. Each interaction between **services acts as a text message** we would receive on our phones.



- It is best to **avoid temporal coupling** (if service A depends on service B, service B being unavailable at the exact moment service A needs it is not an issue)
- It acts as a buffe**r to mitigate spikes** (if there're more emails to be sent than usual, they will be sent when the email service can handle them, as opposed to not being able to send them immediately)

- There are two fundamentally different ways to work with asynchronous communication: queues & logs.
- Asynchronous is **more efficien**t than synchronous
- Services that **interact asynchronously are loosely coupled**, so making changes to these services should be easier.
- If service developers don't have much experience in concurrent programming, however, it usually **takes longer to develop** a reliable asynchronous system.

6.9 Consider the Upload service for photographs to be printed as shown in Figure 6.5 (0). Suggest how this might be implemented and then design a RESTful interface for this service, explaining the function of each of the HTTP verbs. For each operation, identify its input and output.



The API gateway is an important component that insulates the user app from the system's microservices. The gateway is a single point of contact and translates service requests from the app into calls to the microservices used in the system.
Using a gateway also means it is possible to change the service decomposition by splitting or combining services without affecting the client app.
REST stands for Representational State Transfer and API stands for Application Program Interface. REST is a software architectural style that defines the set of rules to be used for creating web services. Web services that follow the REST architectural style are known as RESTful web services. It allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. Interaction in REST-based systems happens through Internet's Hypertext Transfer Protocol (HTTP). :
HTTP verbs: Some of the common HTTP methods/verbs are described below:

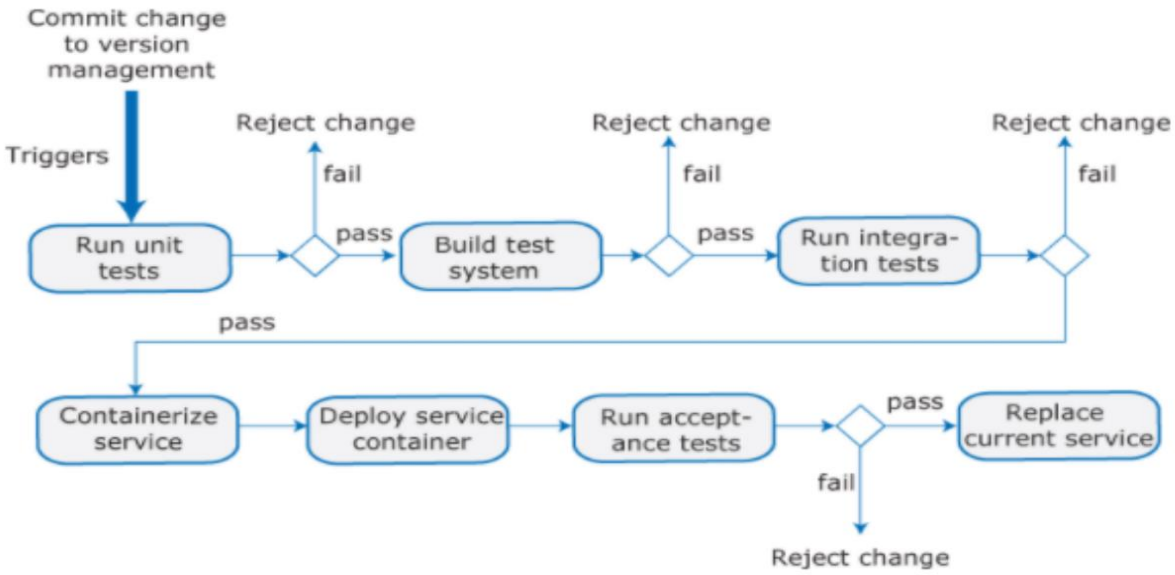
- **GET:** Retrieves one or more resources identified by the request URI and it can cache the information received.

- **POST:** Create a resource from the submission of a request and the response is not cacheable in this case. This method is unsafe if no security is applied to the endpoint as it would allow anyone to create a random resource by submission.
- **PUT:** Update an existing resource on the server specified by the request URI.
- **DELETE:** Delete an existing resource on the server specified by the request URI. It always returns an appropriate HTTP status for every request.

Registration**: Input**: POST-Where user will submit the required field by giving inputs? **Output:** GET- it caches the information received.
Authentication: **Input**: POST: Where the user will give the authentication verification. **Output:** PUT**:** Update an existing verification on the server specified by the request and process it.
Upload: **Input:** POST: Where the user will update a photo. **Output:** GET- it caches the information received
Payment: **Input:** POST: Where the user will POST the payment **Output:** GET- it caches the information received
Printing: **Input: :** POST: Where the service will Print the photo **Output:** GET- where the user will cache the printed photo

6.10 Why should you use continuous deployment in a microservices architecture? Briefly explain each of the stages in the continuous deployment pipeline.

- A **microservices architecture** is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices architecture diagrams and services independently. Microservices architecture is an architectural style in which the system is constructed from communicating microservices. It is well suited to cloud-based systems where each microservice can run in its own container.
- **Continuous deployment** is a strategy in software development where code changes to an application are released automatically into the production environment. This automation is driven by a series of predefined tests. Once new updates pass those tests, the system pushes the updates directly to the software's users.
- It is a completely automated process that relies on automated testing to check that the new version is of production quality. The value of continuous delivery lies in the fact that the code is ready to always deploy QA automatically tests every build in each environment, and if it passes, then the code is ready to deploy.
- Implementing continuous delivery effectively increases quality and velocity and makes the team more efficient. Faster release cycles are one of the major advantages of microservices architectures. But agility can't be achieved without a good CI/CD process. If continuous deployment is used, you may need to maintain multiple versions of deployed services so that you can switch to an older version if problems are discovered in a newly deployed service.



Stages in the continuous deployment pipeline:

A continuous delivery pipeline consists of five main phases—build/develop, commit, test, stage, and deploy.

Build/Develop: A build/develop process performs the following:
Pulls source code from a public or private repository.
Establishes links to relevant modules, dependencies, and libraries.
Builds (compiles) all components into a binary artifact.
Depending on the programming language and the integrated development environment (IDE), the build process can include various tools. The IDE may offer build capabilities or require integration with a separate tool. Additional tools include scripts and a virtual machine (VM) or a Docker container.

Commit: The commit phase checks and sends the latest source code changes to the repository. Every check-in creates a new instance of the deployment pipeline. Once the first stage passes, a release candidate is created. The goal is to eliminate any builds unsuitable for production and quickly inform developers of broken applications.
Commit tasks typically run as a set of jobs, including:

- Compile the source code
- Run the relevant commit tests
- Create binaries for later phases
- Perform code analysis to verify health
- Prepare artifacts like test databases for later phases

Test: During the test phase, the completed build undergoes comprehensive dynamic testing. It occurs after the source code has undergone static testing. Dynamic tests commonly include:
Unit or functional testing—helps verify new features and functions work as intended.

Regression testing—helps ensure new additions and changes do not break previously working features.

Stage: The staging phase involves extensive testing for all code changes to verify they work as intended, using a staging environment, a replica of the production (live) environment. It is the last phase before deploying changes to the live environment.

The staging environment mimics the real production setting, including hardware, software, configuration, architecture, and scale. You can deploy a staging environment as part of the release cycle and remove it after deployment in production.

Deploy: The deployment phase occurs after the build passes all testing and becomes a candidate for deployment in production. A continuous delivery pipeline sends the candidate to human teams for approval and deployment. A continuous deployment pipeline deploys the build automatically after it passes testing.

7.2. Explain in your own words what you understand by an SQL injection attack. Explain how you can use data validation to avoid such attacks.

- SQL injections are typically performed via web page or application input. These input forms are often found in features like search boxes, form fields, and URL parameters.
- SQL injection is one of the most common web attack mechanisms utilized by attackers to steal sensitive data from organizations. While SQL Injection can affect any data-driven application that uses a SQL database.
- SQL Injection is a code injection technique that hackers can use to insert malicious SQL statements into input fields for execution by the underlying SQL database. This technique is made possible because of improper coding of vulnerable web applications.
- To perform an SQL injection attack, bad actors need to identify vulnerabilities within a web page or application. After locating a target, attackers create malicious payloads and send their input content to execute malicious commands.

1. Validate User Inputs: A common first step to preventing SQL injection attacks is validating user inputs. First, identify the essential SQL statements and establish a whitelist for all valid SQL statements, leaving unvalidated statements out of the query. This process is known as input validation or query redesign.

2. Sanitize Data By Limiting Special Characters: Another component of safeguarding against SQL injection attacks is mitigating inadequate data sanitization. Because SQLi attackers can use unique character sequences to take advantage of a database, sanitizing data not to allow string concatenation is critical.

3. Enforce Prepared Statements and Parameterization: Input validation and data sanitization aren't fix-all's. It's critical organizations also use prepared statements with parameterized queries, also known as variable binding, for writing all database queries. By defining all SQL code involved with queries, or parameterization, you can distinguish between user input and code.

4. Use Stored Procedures in The Database: Like parameterization, using stored procedures also requires variable binding. Unlike the prepared statements approach to mitigating SQLi, stored procedures reside in the database and are called from the web application. Stored procedures are also not immune to vulnerabilities if dynamic SQL generation is used.

5. Actively Manage Patches and Updates: Vulnerabilities in applications and databases that are exploitable using SQL injection are regularly discovered and publicly identified. Like so many cybersecurity threats, it's vital organizations stay in tune with the most recent news and apply patches and updates as soon as practical.

6. Raise Virtual or Physical Firewalls: We strongly recommend using a software or appliance-based web application firewall (WAF) to help filter out malicious data. Firewalls today, including NGFW and FWaaS offerings, have both a comprehensive set of default rules and the ease to change configurations as needed. If a patch or update has yet to be released, WAFs can be handy.

7. Harden Your OS And Applications: This step goes beyond mitigating SQL injection attacks in ensuring your entire physical and virtual framework is working intentionally. Adopting application vendor security guidelines can enhance an organization's defensive posture and help identify and disable unnecessary applications and servers.

8. Reduce Your Attack Surface: In cybersecurity, an attack surface refers to the array of potential entry points for attackers. So in the context of SQLi attacks, this means disposing of any database functionalities that you don't need or further safeguarding them.

9. Establish Appropriate Privileges And Strict Access: Given the power, SQL database holds for an organization, it's imperative to enforce least privilege access policies with strict rules. If a website only requires the use of SELECT statements for a database, there's no reason it should have additional INSERT, UPDATE, or DELETE privileges

10. Limit Read-Access: Connected to the principle of least privilege for SQL injection protection is configuring read-access to the database. If your organization only requires active users employing read access, it's undoubtedly easier to adopt. Nevertheless, this added step is imperative for stopping attackers from altering stored information.

11. Encryption: Keep Your Secrets Secret: It's best to assume internet-connected applications are not secure. Therefore encryption and hashing passwords, confidential data, and connection strings are of the utmost importance.

12. Deny Extended URLs: Another tactic by SQLi attackers is sending excessively long URLs causing the server to fail at logging the complete request.

13. Don't Divulge More Than Necessary in Error Messages: SQL injection attackers can learn a great deal about database architecture from error messages, ensuring that they display minimal information. Use of the "Remote Only" custom Errors mode (or equivalent) can display verbose error messages on the local machine while ensuring that an external attacker gets nothing more than the fact that his or her actions resulted in an unhandled error.

14. No Shared Databases or User Accounts: Shared databases by multiple websites or applications can be a recipe for disaster. And the same is true for user accounts that have access to various web applications. This shared access might provide flexibility for the managing organization or administrator, but it also unnecessarily poses a more significant risk

15. Enforce Best Practices For Account And Password Policies: While it might go without saying, organizations must follow the best account and password policies for foolproof security. Default and built-in passwords should be changed upon receipt and before usage, with regularly scheduled password updates. Suitable passwords in length and character complexity are essential for all SQL server administrators, users, and machine accounts.

16. Continuous Monitoring of SQL Statements: Organizations or third-party vendors should continually monitor all SQL statements of database-connected applications for an application, including documenting all database accounts, prepared statements, and stored procedures.
17. Perform Regular Auditing and Penetration Testing: Regular audits of your database and application security are becoming increasingly necessary, including auditing logs for suspicious activity, group and role membership privileges, and variable binding terms.

7.3. What do you think are the advantages and disadvantages of using a special-purpose device rather than a mobile phone in two-factor authentication? (Hint: Think about the problems of using a mobile phone as an authentication device.

Authentication is the process of determining whether someone or something is, in fact, who or what it says it is. Authentication technology provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server. In doing this, authentication assures secure systems, secure processes, and enterprise information security.
There are several authentication types. For purposes of user identity, users are typically identified with a user ID, and authentication occurs when the user provides credentials such as a password that matches their user ID. The practice of requiring a user ID and password is known as *single-factor authentication* (SFA). In recent years, companies have strengthened authentication by asking for additional authentication factors, such as a unique code that is provided to a user over a mobile device when a sign-on is attempted or a biometric signature, like a facial scan or thumbprint. This is known as *two-factor authentication* (2FA).
Advantages:

- It doesn't rely on the use of a mobile or other internet-connected device.
- It's portable and the user can keep it in their pocket, purse, or briefcase.
- The token refreshes after a set amount of time – no permanent passcode to steal.
- No need to carry a token generator, as the process uses your mobile device.
- The passcodes are generated on the fly, making them more secure than static passwords.
- A maximum number of passcode entries can be set, limiting the risk of passcode guessing.
- The process is user-friendly.
- No need to connect to a cellular network or the internet, because all the necessary data are already stored on the device.
- Ease of use – just connect the token to the USB port and press a single button when requested.
- There are Yubikey token models with two slots, which allows using them to access two websites instead of one.
- Contactless device, protected from any possibility of malware injection.
- One-time password is generated by the device itself, which reduces the possibility of intercept to a minimum.
- Does not require a network connection of any kind.

- Built-in power source is enough for years of independent operation.

Disadvantages:

- It's easily lost or stolen from your pocket, purse, or briefcase.
- It could be vulnerable to man-in-the-middle attacks
- Low prevalence, because the standard is still quite new. Currently, U2F tokens are supported by Gmail, Google Accounts, GitHub, Dropbox, LastPass, and WordPress.
- U2F USB tokens, now, are compatible only with the Chrome browser since version 38. This restriction is applied not by Yubikey tokens, but the U2F standard itself.
- Many companies block operations with USB ports on corporate computers.
- U2F devices are created to access 1 or 2 specific resources; an active Internet user will need to store a bundle of devices for the access to different websites.
- U2F devices are relatively costly: prices for the cheapest models start from $20.
- The token is easily forgot inserted into computer when going away from a workstation. Some people also leave them inserted all the time for the convenience, which undermines the whole concept of 2FA.
- It is desirable to buy tokens in pairs in case of loss.
- If a token is compromised, then you only need to order a new one.
- If you stop using the service, then the money was wasted, there is no possibility to use it with another service.
- If you need to protect several accounts (or accounts at different resources), you will need a separate token for each
- The secret key in such tokens is pre-flashed at the factory, then passed to the supplier, then transferred to the website owner. Of course, keys are transmitted in encrypted form, but there remains a tiny possibility that at some stage an unscrupulous employee or a hacker will leak secret keys. In this respect, the Protectimus Slim NFC wins unconditionally. It is flashed by the user from their personal smartphone with a secret key that is known only to them.