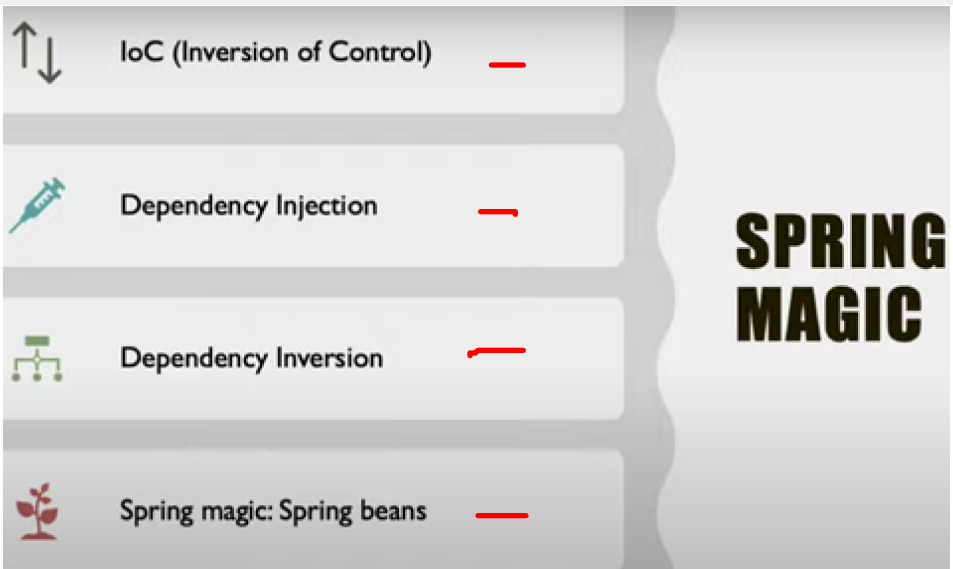
- Spring (Framework) focusses on creating enterprise applications with Java
- Lots of boilerplate code no longer necessary
- Offers IoC (Inversion of Control), transactions
- Spring has very many different parts

- Spring Boot is built on top of spring framework
- Auto configured
- Custom configuration happens with annotations (@Annotation), XML configuration no longer necessary
- Easy to set up
- Great for stand alone APIs



- A bean is simply:

an instance of a class managed by the Spring container

The Spring container is the core of the Spring Framework. It manages the lifecycle, configuration, and wiring of Spring beans. Types of Spring containers:

1. **BeanFactory**: Basic container, lazy initialization.
   - Example: `XmlBeanFactory`
2. **ApplicationContext**: Advanced container, eager initialization, supports more features.
   - Example: `ClassPathXmlApplicationContext`

Initialization:

```java
ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml")
```

Bean Retrieval:

```java
MyBean myBean = context.getBean("myBean", MyBean.class);
```

No, "Spring container" is not a single class but a concept. It's implemented through various interfaces and classes in the Spring Framework. For example, `BeanFactory` and `ApplicationContext` are interfaces that define the contract for a Spring container. Classes like `ClassPathXmlApplicationContext` are concrete implementations of these interfaces.

Spring Bean Scopes define the lifecycle and visibility of a bean
Here are the main types:

1. **Singleton**: One instance per Spring container.
   - Example: Database connection pool
2. **Prototype**: New instance each time bean is requested.
   - Example: User session objects
3. **Request**: One instance per HTTP request.
   - Example: User authentication object in a web app
4. **Session**: One instance per HTTP session.
   - Example: Shopping cart in an e-commerce site
5. **Application**: One instance per `ServletContext`.
   - Example: Application-wide settings
6. **Websocket**: One instance per WebSocket session.
   - Example: Chat application user session

Using injection over manual instantiation offers several advanta

1. **Decoupling**: Easier to switch implementations.
   - With injection: Change bean definition.
   - Without: Change every `new` statement.
2. **Testability**: Easier to mock dependencies for unit tests.
   - With injection: Inject mock object.
   - Without: Hard to replace manually created instance.
3. **Maintainability**: Centralized configuration.
   - With injection: Change in one place (bean config).
   - Without: Change in multiple places.
4. **Lifecycle Management**: Container handles it.
   - With injection: Spring manages object lifecycle.
   - Without: You manage it, risking resource leaks.

1. **Without IoC**:
   - You manually create and manage objects.

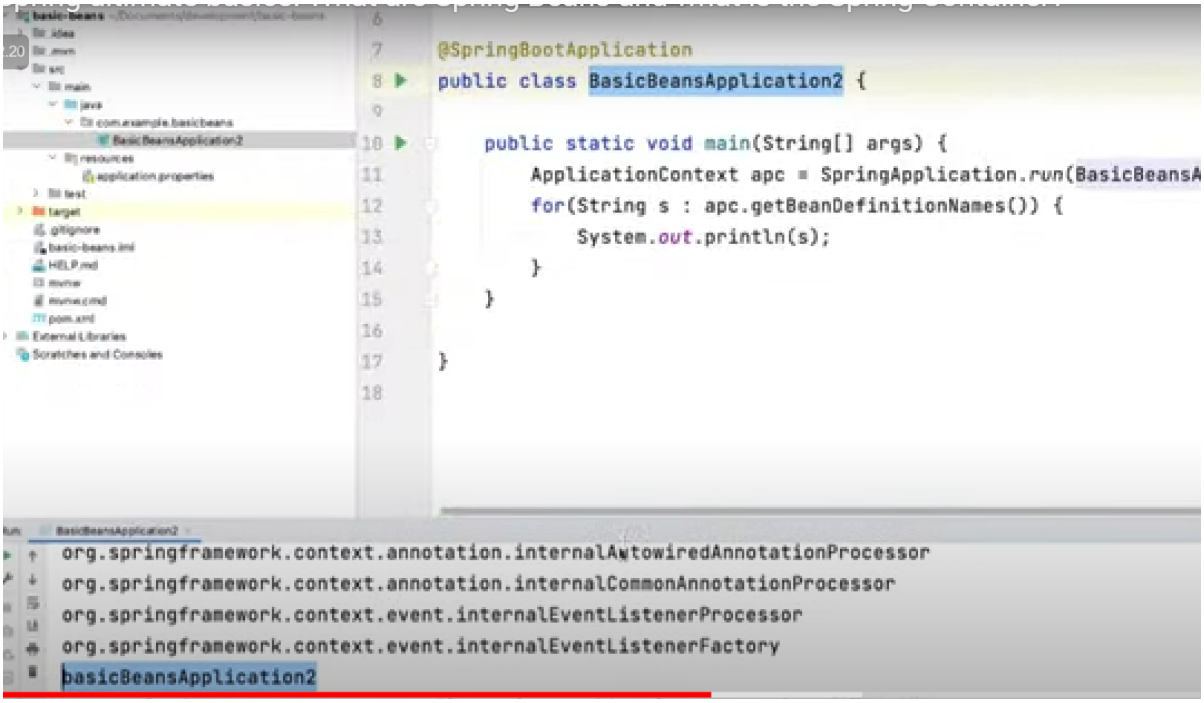
```cpp
MyClass obj = new MyClass();
```

2. **With IoC**:
   - The container creates and manages objects

```java
@Autowired
MyClass obj;
```

DI (Dependency Injection):

- **Concept**: A specific form of IoC where dependencies are "injected" into an object by the container.




```java
@Component
public class Customer {
    private String name;
    private Address address;

    public Customer(String name, Addr
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public void setName(String name)
        this.name = name;
    }

    public Address getAddress() {
```

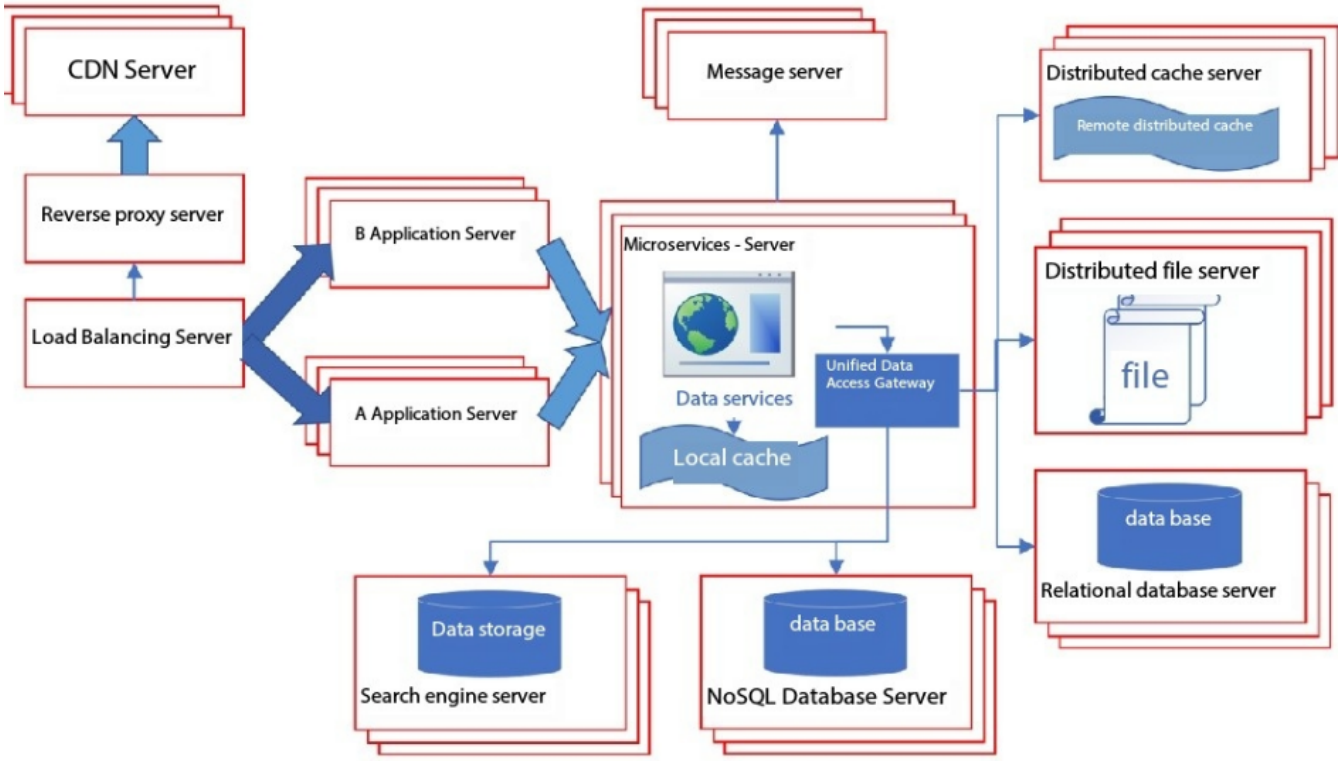
```
Parameter 0 of constructor in com.example.basicbeans.Customer required a bean of type 'java.lang.String'

Action:

Consider defining a bean of type 'java.lang.String' in your configuration.
```

```java
@Bean
public String getName() {
    return "Maaike";
}
```

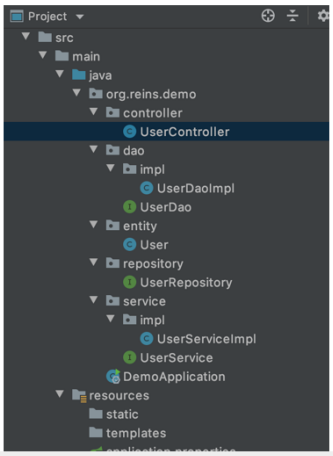
```java
@Component
public class Address {
}
```



Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring `ApplicationContext`.
session	Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`.
application	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`.

- HTTP is a stateless protocol
  - A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.

Default Scope - singleton



UserController.java

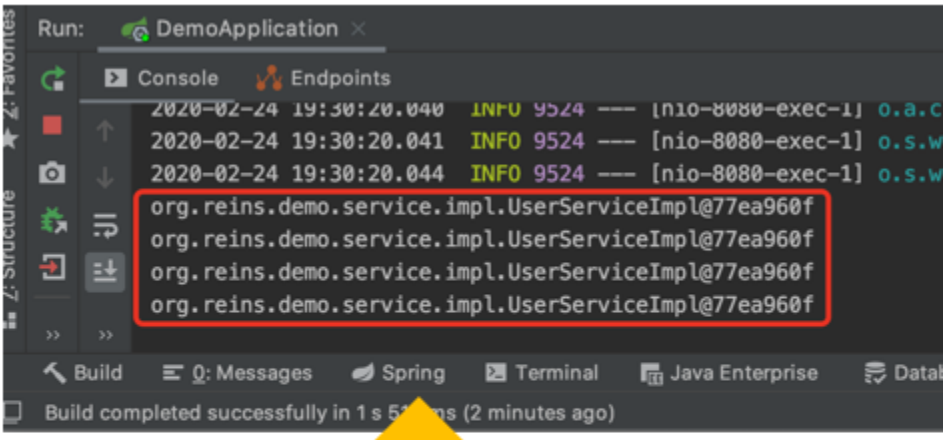
```
@RestController
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping(value = "/findUser/{id}")
    public User findOne(@PathVariable("id") String id) {
        System.out.println(userService);
        return userService.findUserById(Integer.valueOf(id));
    };
}
```

UserServiceImpl.java

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;
```



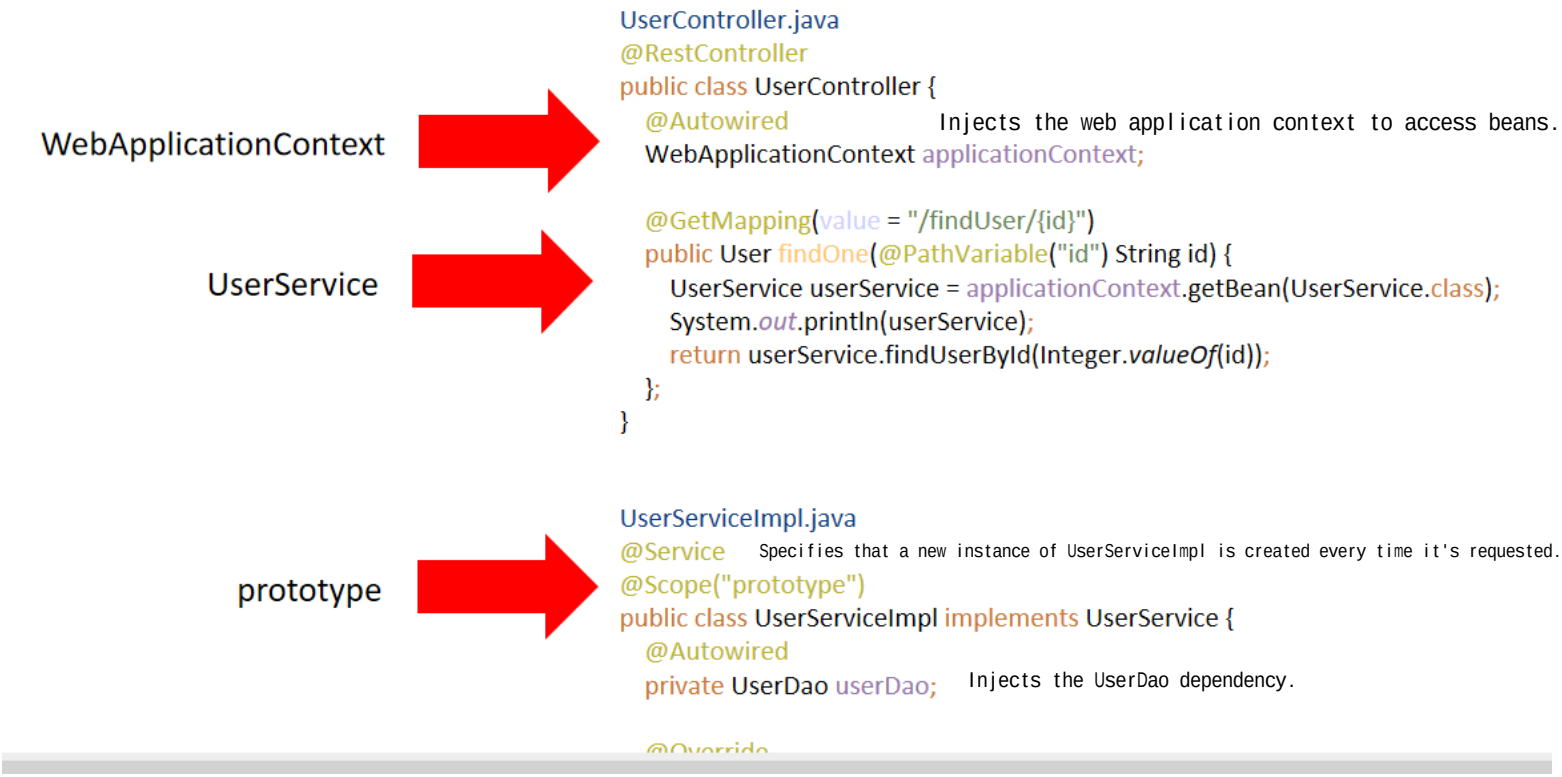
Single Service instance

prototype

UserServiceImpl.java

```
@Service
@Scope("prototype")
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;
```

```
org.reins.demo.service.impl.UserServiceImpl@145b081e
org.reins.demo.controller.UserController@6f6c8d45
org.reins.demo.service.impl.UserServiceImpl@35fcaedc
org.reins.demo.controller.UserController@6f6c8d45
org.reins.demo.service.impl.UserServiceImpl@6b9405d
org.reins.demo.controller.UserController@6f6c8d45
org.reins.demo.service.impl.UserServiceImpl@7dd48753
org.reins.demo.controller.UserController@6f6c8d45
```



2 Browsers(2 Sessions), 4 Requests(2 times / session)

		Controller	
		Prototype	session
Service	prototype	4 service instances 4 controller instances	4 service instances 2 controller instances
	session	2 service instances 4 controller instances	2 service instances 2 controller instances

A container in software engineering is an abstraction layer that packages code and its dependencies together so that applications can run uniformly and consistently across different computing environments. There are different types of containers:

1. OS-Level Containers:

- **Example**: Docker
- **Function**: Packages an application and its dependencies, libraries, and binaries into a single container image.

2. Language Runtime Containers:

- **Example**: Java Virtual Machine (JVM)
- **Function**: Executes bytecode, manages memory and resources.

3. Application Containers:

- **Example**: Spring Container, Servlet Container (Tomcat)
- **Function**: Manages the lifecycle, configuration, and runtime environment of application-level components like Spring Beans or Servlets.

4. Component Containers:

- **Example**: React Container Components
- **Function**: Manages data and logic, often delegates rendering to presentational components.

Java EE:

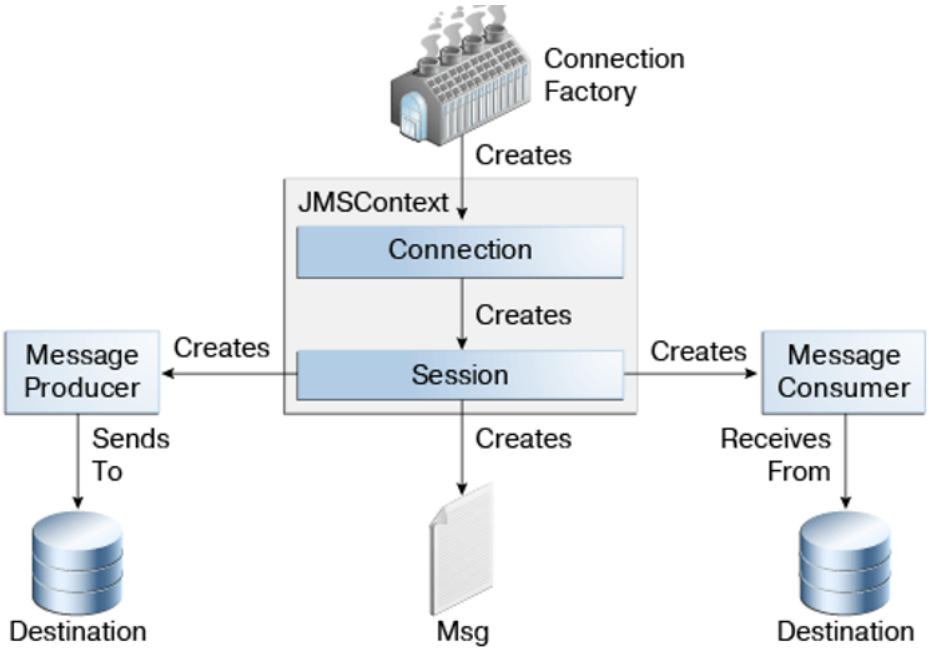
1. **Standardized**: Part of the official Java platform.
2. **Container-Managed**: EJB (Enterprise JavaBeans) for business logic, managed by the application server.
3. **Configuration**: Primarily XML-based, though annotations are supported.
4. **Libraries**: Comes with a set of standard APIs like JPA, JMS, and JAX-RS.
5. **Servers**: Requires a Java EE application server like GlassFish, WebLogic, or JBoss.

Java EE (Enterprise Edition) and Spring Framework are both used for building enterprise-level applications in Java, but they have different approaches and features.

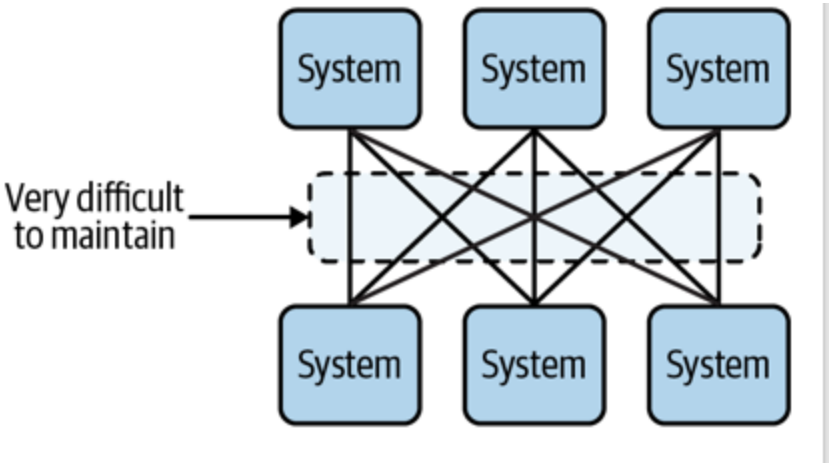
What is JMS API?

Java Message Service (JMS) API is a Java-based messaging standard that allows application components to create, send, receive, and read messages in a loosely coupled, reliable, and asynchronous way. It's often used in Java EE applications for point-to-point and publish-

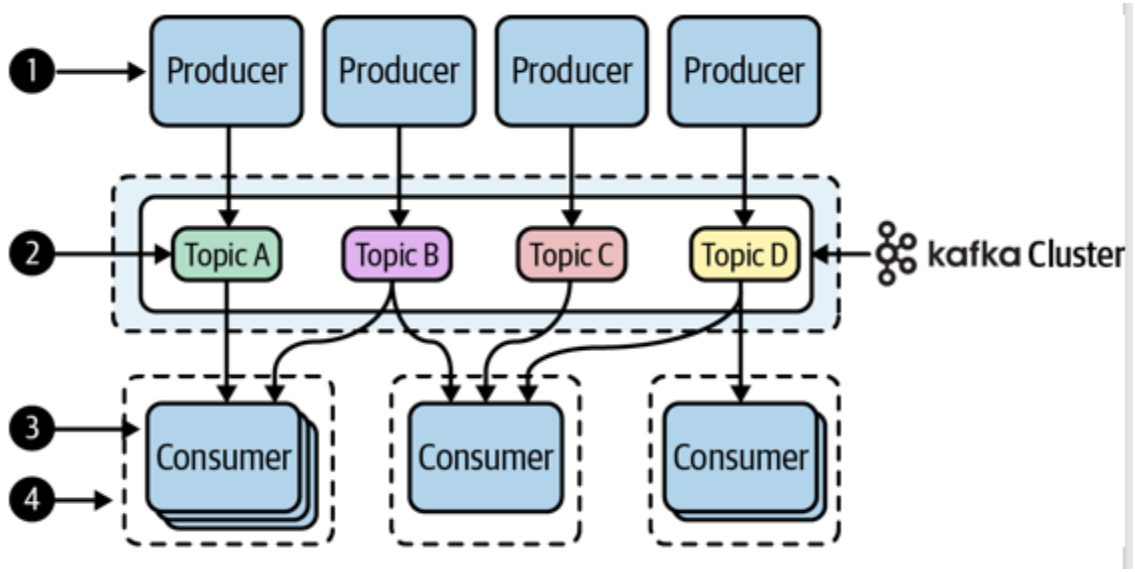
```
// Create a connection
Connection connection = connectionFactory.createConnection();
// Create a session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
// Create a message producer
MessageProducer producer = session.createProducer(queue);
// Create and send a message
TextMessage message = session.createTextMessage("Hello, World!");
producer.send(message);
```



Synchronous



Asynchronous

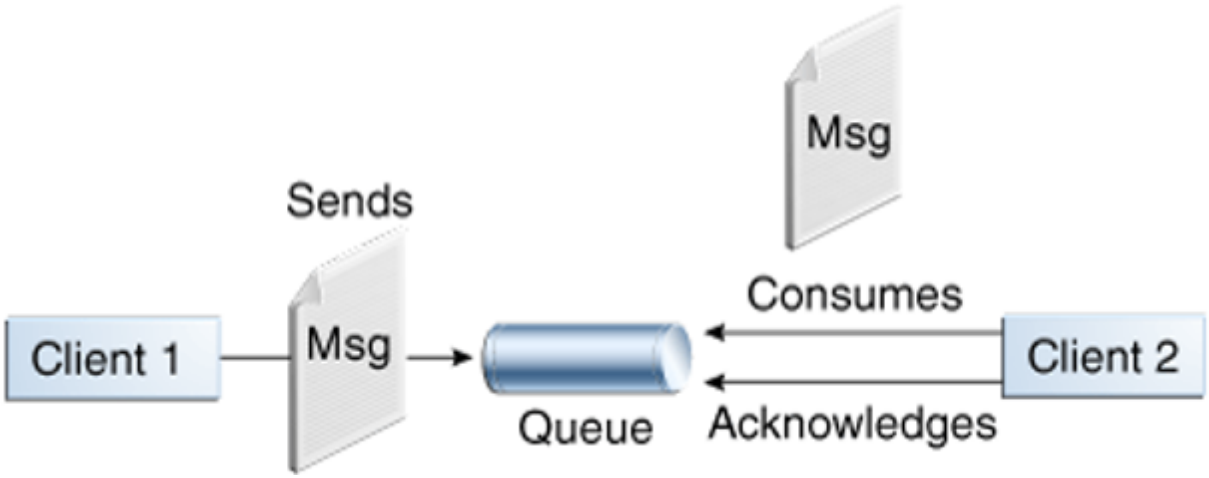


The JMS API in the Java EE platform has the following features.

- Application clients, Enterprise JavaBeans (EJB) components, and web components can send or synchronously receive a JMS message. Application clients can in addition set a message listener that allows JMS messages to be delivered to it asynchronously by being notified when a message is available.
- Message-driven beans, which are a kind of enterprise bean, enable the asynchronous consumption of messages in the EJB container. An application server typically pools message-driven beans to implement concurrent processing of messages.
- Message send and receive operations can participate in Java Transaction API (JTA) transactions, which allow JMS operations and database accesses to take place within a single transaction.

1)

- A **point-to-point** (PTP) product or application is built on the concept of message **queues**, senders, and receivers.
  - Each message has only one consumer.
  - The receiver can fetch the message whether or not it was running when the client sent the message.



2)

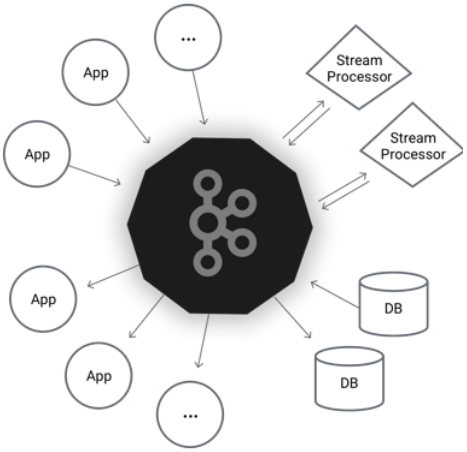
In a **publish/subscribe** (pub/sub) product or application, clients address messages to a **topic**, which functions somewhat like a bulletin board. Publishers and subscribers can dynamically publish or subscribe to the topic.

- Each message can have multiple consumers.
- A client that subscribes to a topic can consume only messages sent *after* the client has created a subscription, and the consumer must continue to be active in order for it to consume messages.

Apache Kafka

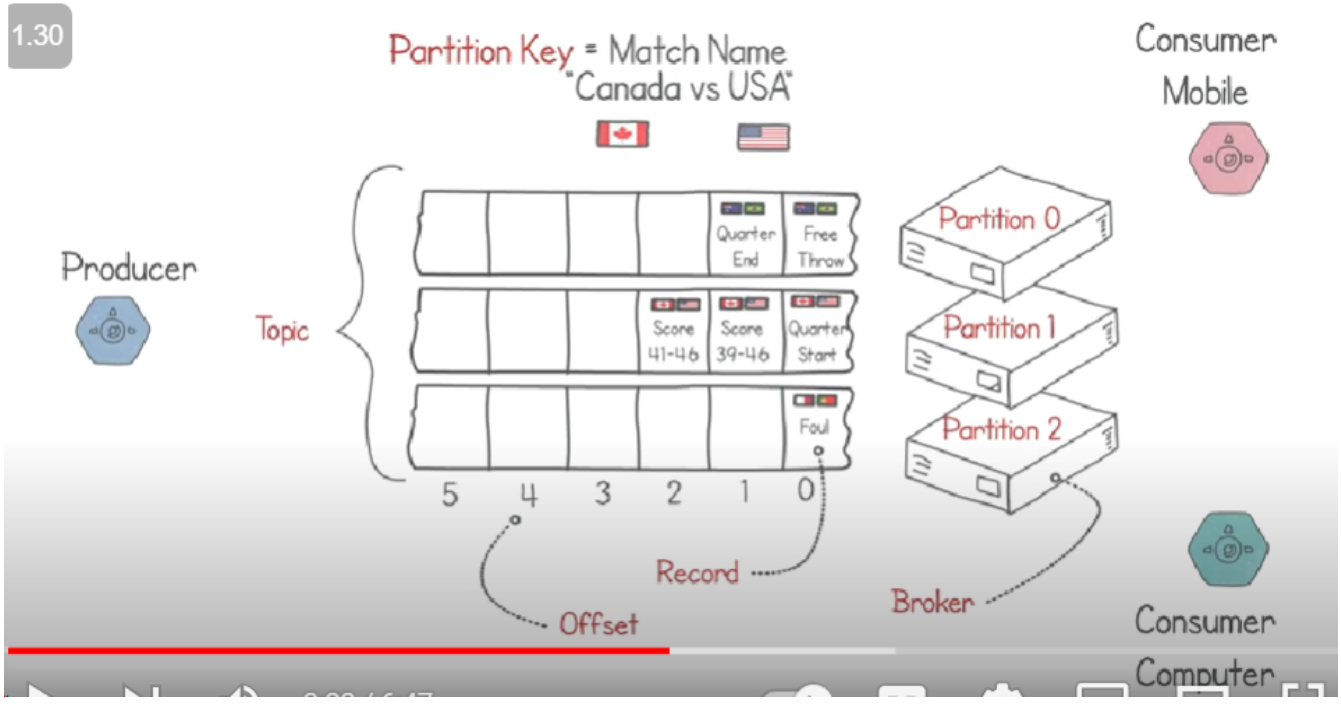
- is an open-source distributed event streaming platform.

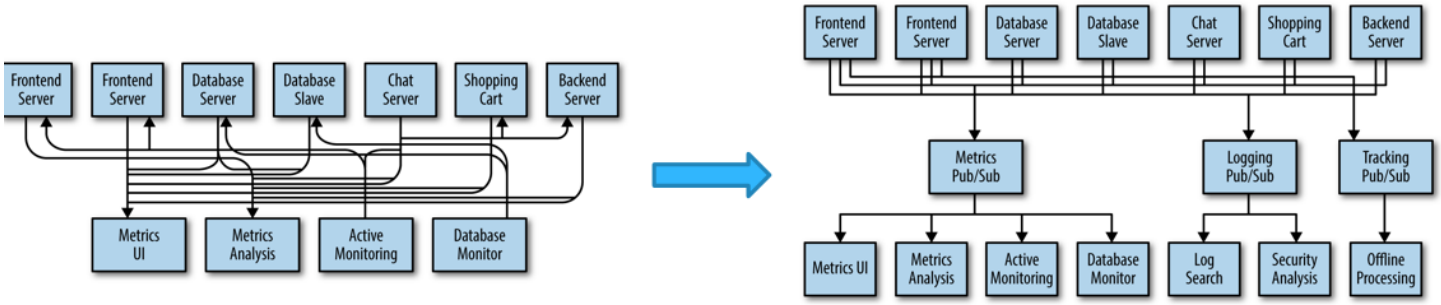
- To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
- To **store** streams of events durably and reliably for as long as you want.
- To **process** streams of events as they occur or retrospectively.



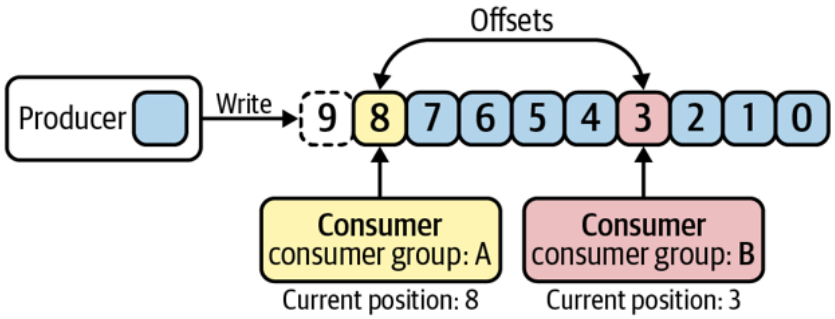
Kafka is a distributed streaming platform used for building real-time data pipelines and streaming applications. It's designed to handle high throughput with low latency.

1. **Producer**: Sends messages to Kafka topics.
   - Example: IoT sensors sending data.
2. **Consumer**: Reads messages from topics.
   - Example: Analytics application reading sensor data.
3. **Broker**: Kafka server that stores data and serves clients.
   - Example: A Kafka cluster can have multiple brokers.
4. **Topic**: Message category or feed name.
   - Example: `user_signup`, `page_views`.
5. **Partition**: Topics are divided into partitions for scalability.
   - Example: Partition 0, 1, 2 for `user_signup` topic.
6. **Zookeeper**: Manages broker metadata.
   - Example: Keeping track of brokers, partitions.

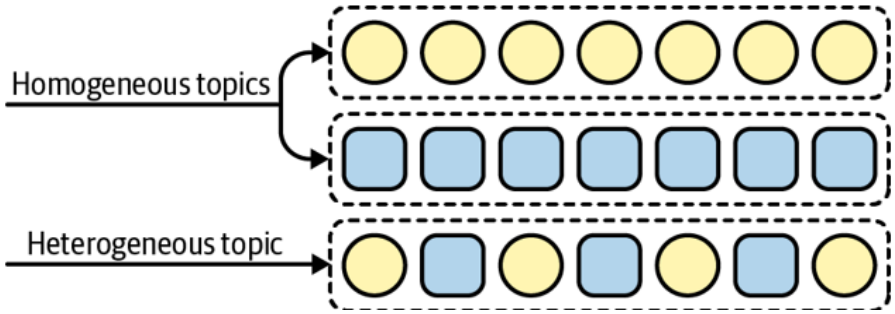




multiple consumer groups can each read from the same log, and maintain their own positions in the log/stream they are reading from.

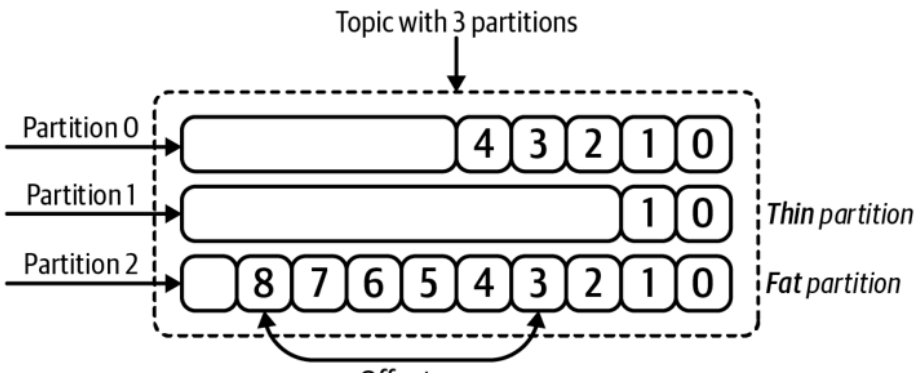


- *homogeneous topics* that contain only one type of data, or *heterogeneous topics* that contain multiple types of data



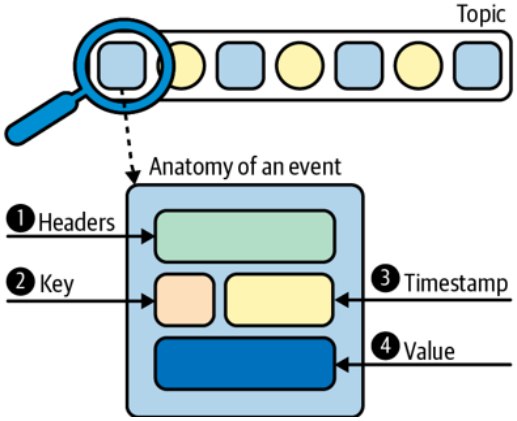
Kafka topics are broken into smaller units called *partitions*.

- Partitions are individual logs where data is produced and consumed from
- The number of partitions for a given topic is configurable, and having more partitions in a topic generally translates to more parallelism and throughput



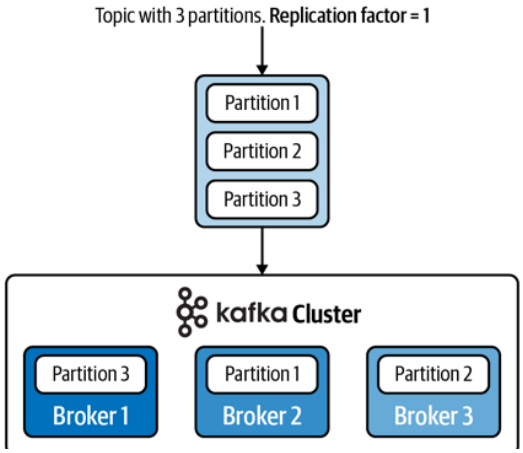
An event is a timestamped key-value pair that records *something that happened.*

- Application-level headers contain optional metadata about an event.
- Keys are also optional, but play an important role in how data is distributed across partitions.
- Each event is associated with a timestamp.
- The value contains the actual message contents, encoded as a byte array.

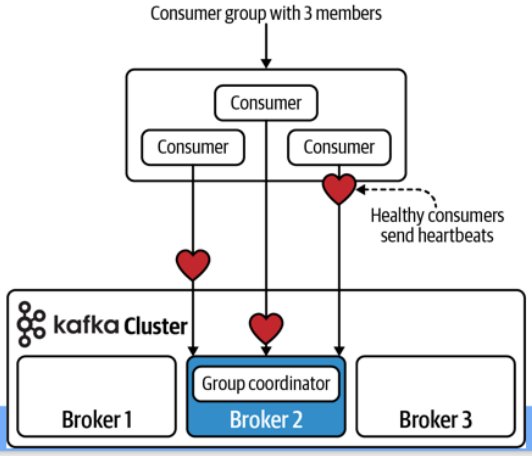


Kafka operates as a cluster, and multiple machines, called *brokers*, are involved in the storage and retrieval of data.

- Kafka clusters can be quite large, and can even span multiple data centers and geo- graphic regions.



- Consumer groups are made up of multiple cooperating consumers, and the member- ship of these groups can change over time.
  - every consumer group is assigned to a special broker called the *group coordinator*, which is responsible for receiving heartbeats from the consumers, and triggering a *rebalance* of work whenever a consumer is marked as dead.



Install Apache Kafka on Windows PC
https://www.youtube.com/watch?v=BwYFuhVhshI

- WebSocket is an application protocol that provides full-duplex communications between two peers over the TCP protocol.
  - In the traditional request-response model used in HTTP, the client requests resources and the server provides responses.
  - The exchange is always initiated by the client; the server cannot send any data without the client requesting it first.
  - The WebSocket protocol provides a full-duplex communication channel between the client and the server.
  - Combined with other client technologies, such as JavaScript and HTML5, WebSocket enables web applications to deliver a richer user experience.

The client initiates the handshake by sending a request to a WebSocket endpoint using its URI.

- The handshake is compatible with existing HTTP-based infrastructure:
  - web servers interpret it as an HTTP connection upgrade request.
- An example handshake from a client looks like this:

- EchoEndpoint

```
@ServerEndpoint("/echo")
public class EchoEndpoint {
 @OnMessage
 public void onMessage(Session session, String msg) {
   try {
     session.getBasicRemote().sendText(msg);
   } catch (IOException e) { ... }
 }
}
```

Annotation	Event	Example
OnOpen	Connection opened.	@OnOpen public void open(Session session, EndpointConfig conf) { }
OnMessage	Message received.	@OnMessage public void message (Session session, String msg) { }
OnError	Connection error.	@OnError public void error(Session session, Throwable error) { }
OnClose	Connection closed.	@OnClose public void close(Session session, CloseReason reason) { }

```
function connect() {
    wsocket = new WebSocket
            ("ws://localhost:8080/WebSocketSamples/dukeetf");
    wsocket.onmessage = onMessage;
}
```

The Java API for WebSocket provides

- support for converting between WebSocket messages and custom Java types using encoders and decoders.
- An encoder takes a Java object and produces a representation that can be transmitted as a WebSocket message;
  - for example, encoders typically produce JSON, XML, or binary representations.
- A decoder performs the reverse function: it reads a WebSocket message and creates a Java object.
- This mechanism simplifies WebSocket applications, because it decouples the business logic from the serialization and deserialization of objects.

Encoders

- Implement one of the following interfaces:
  - `Encoder.Text<T>` for text messages
  - `Encoder.Binary<T>` for binary messages

```
public class MessageATextEncoder implements Encoder.Text<MessageA> {
 @Override
 public void init(EndpointConfig ec) { }

 @Override
 public void destroy() { }

 @Override
 public String encode(MessageA msgA) throws EncodeException {
   // Access msgA's properties and convert to JSON text...
   return msgAJsonString;
 }
}
```

Decoders

- Implement one of the following interfaces:
  - `Decoder.Text<T>` for text messages
  - `Decoder.Binary<T>` for binary messages

```
public class MessageTextDecoder implements Decoder.Text<Message> {
 @Override
 public void init(EndpointConfig ec) { }
 @Override
 public void destroy() { }
 @Override
 public Message decode(String string) throws DecodeException {
   // Read message...
   if ( /* message is an A message */ ) return new MessageA(...);
   else if ( /* message is a B message */ ) return new MessageB(...);
 }
 @Override
 public boolean willDecode(String string) {
  // Determine if the message can be converted into either a
  // MessageA object or a MessageB object...
  return canDecode;
 }
}
```

```java
public class BankListener {

    @Autowired
    private WebSocketServer ws;

    @KafkaListener(topics = "topic1", groupId = "group_topic_test")
    public void topic1Listener(ConsumerRecord<String, String> record) {
        String[] value = record.value().split(",");
        bankService.transfer(value[0], value[1], Integer.valueOf(value[2]));
        kafkaTemplate.send("topic2",  "key", "Done");
    }

    @KafkaListener(topics = "topic2", groupId = "group_topic_test")
    public void topic2Listener(ConsumerRecord<String, String> record) {
        String value = record.value();
        System.out.println(value);
        ws.sendMessageToUser("Tom","Done");
    }
}
```

```javascript
function openSocket() {
  if (typeof (WebSocket) == "undefined") {
    alert("您的浏览器不支持WebSocket");
  } else {

    if (socket != null) {
      return;
    }


    var userId = document.getElementById('name').value;
;   var socketUrl = "ws://localhost:8080/websocket/transfer/" + userId
    console.log(socketUrl);
    setConnected(true);

    socket = new WebSocket(socketUrl);
    //打开事件
   socket.onopen = function () {
      console.log("websocket已打开");
      //socket.send("这是来自客户端的消息" + location.href + new Date());
    };
```

- What Is a Transaction?
- Container-Managed Transactions
- Isolation and Database Locking
- Updating Multiple Databases

- In a Java EE application, a transaction
  - is a series of actions that must all complete successfully, or else all the changes in each action are backed out. Transactions end in either a commit or a rollback.

- The Java Transaction API (JTA) allows applications
  - to access transactions in a manner that is independent of specific implementations

- The JTA defines the `UserTransaction` interface that applications use to start, commit, or roll back transactions.
  - Application components get a `UserTransaction` object through a JNDI lookup by using the name `java:comp/UserTransaction` or by requesting injection of a `UserTransaction` object.

```
begin transaction
 debit checking account
 credit savings account
 update history log
commit transaction
```

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
Required	T1	T1
RequiresNew	None	T2
RequiresNew	T1	T2
Mandatory	None	Error
Mandatory	T1	T1
NotSupported	None	None
NotSupported	T1	None
Supports	None	None
Supports	T1	T1
Never	None	None
Never	T1	Error

- Transaction attributes are specified by
  - decorating the enterprise bean class or method with a `javax.ejb.TransactionAttribute` annotation
  - and setting it to one of the `javax.ejb.TransactionAttributeType` constants.

Transaction Attribute	TransactionAttributeType Constant
Required	TransactionAttributeType.REQUIRED
RequiresNew	TransactionAttributeType.REQUIRES_NEW
Mandatory	TransactionAttributeType.MANDATORY
NotSupported	TransactionAttributeType.NOT_SUPPORTED
Supports	TransactionAttributeType.SUPPORTS
Never	TransactionAttributeType.NEVER

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
 ...
 @TransactionAttribute(REQUIRES_NEW)
 public void firstMethod() {...}

 @TransactionAttribute(REQUIRED)
 public void secondMethod() {...}
 public void thirdMethod() {...}
 public void fourthMethod() {...}
}
```

  - If the bean throws an application exception, the rollback is not automatic but can be initiated by a call to `setRollbackOnly`.

WHY DAO? !!

So, in your current setup, `BankRepository` (presumed to be a Spring Data Repository) is essentially being wrapped by another Spring component (`BankDaoImpl`), which is unnecessary unless you have a very specific reason to add an additional layer of abstraction.

In simpler terms, either:

- Use just `BankRepository` if Spring Data JPA's functionalities are sufficient.
- Or use a `BankDao` implementation without involving Spring Data JPA's repository.

Example using only `BankRepository`:

java Copy code

```java
@Repository
public class BankService {
    @Autowired
    private BankRepository bankRepository;

    @Transactional
    public void withdraw(String from, int amount) {
        Bank out = bankRepository.findById(from).get();
        out.setBalance(out.getBalance() - amount);
    }

    @Transactional
    public void deposit(String to, int amount) {
        Bank in = bankRepository.findById(to).get();
```

Stop gener

- BankService.java

```java
public interface BankService {
    public void transfer(String from, String to, int amount);
}
```

- BankServiceImpl.java

```java
@Service
public class BankServiceImpl implements BankService{

    @Autowired
    private BankDao bankDao;

    @Override
    @Transactional
    public void transfer(String from, String to, int amount){
        System.out.println(from + " " + to + amount);
        bankDao.withdraw(from, amount);
//      int result = 10 / 0;
        try {
            bankDao.deposit(to, amount);
        }catch (Exception e){
            e.printStackTrace();
        }
//      int result = 10 / 0;
    };
}
```

- BankDao.java

```java
public interface BankDao {
    public void withdraw(String from, int amount);
    public void deposit(String to, int amount);
}
```

- BankDaoImpl.java

```java
@Repository
public class BankDaoImpl implements BankDao {
    @Autowired
    private BankRepository bankRepository;

    @Transactional
    public void withdraw(String from, int amount) {
        Bank out = bankRepository.findById(from).get();
        out.setBalance(out.getBalance() - amount);
//      int result = 10 / 0;
    };

    @Transactional (propagation = Propagation.REQUIRES_NEW)
    public void deposit(String to, int amount) {
        Bank in = bankRepository.findById(to).get();
        in.setBalance(in.getBalance() + amount);
//      int result = 10 / 0;
    };
}
```

	transfer	withdraw	deposit	result
1	正常	正常	正常	Tom - 80; Jerry + 80
2	result = 10 / 0	正常	正常	整个事务回滚
3	正常	result = 10 / 0	正常	整个事务回滚
4	正常	正常	result = 10 / 0	整个事务回滚
5	正常	正常	正常 ***REQUIRES_NEW***	正常
6	result = 10 / 0，在deposit之前	正常	正常 ***REQUIRES_NEW***	整个事务回滚
7	result = 10 / 0，在deposit之后	正常	正常 ***REQUIRES_NEW***	deposit成功，withdraw和transfer事务回滚
8	正常	result = 10 / 0	正常 ***REQUIRES_NEW***	整个事务回滚
9	正常	正常	result = 10 / 0 ***REQUIRES_NEW***	withdraw和transfer成功，deposit事务回滚

imagining. Transactional systems like relational databases typically use one of the two methods to manage rollbacks:

1. **Write-Ahead Logging (WAL)**: Before making changes, the database logs the original data ("old value"). If a rollback is needed, the database consults this log and restores the old values.
2. **Two-Phase Commit**: Changes are first staged and not committed immediately. If any part of the transaction fails, the staged changes are discarded, effectively reverting to the state before the transaction.

Transaction Propagation

Transaction propagation defines how transactions relate to each other when methods are called within a transactional context. Types include:

1. **REQUIRED**: Use existing transaction if present; otherwise, create new.
2. **REQUIRES_NEW**: Always create a new transaction.
3. **SUPPORTS**: Use existing transaction if available; otherwise, non-transactional.
4. **NOT_SUPPORTED**: Always non-transactional.
5. **MANDATORY**: Must run within existing transaction; throws exception otherwise.
6. **NEVER**: Must not run within existing transaction; throws exception otherwise.
7. **NESTED**: Nested transaction if existing transaction exists.

```java
@Service
public class OrderService {

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private OrderItemRepository orderItemRepository;

    @Transactional(propagation = Propagation.REQUIRED)
    public void addOrder(Order order, List<OrderItem> orderItems) {
        // Add order
        Order savedOrder = orderRepository.save(order);

        // Add order items
        for (OrderItem item : orderItems) {
            item.setOrderId(savedOrder.getId()); // Set FK
        }
        orderItemRepository.saveAll(orderItems);
    }
}
```

```java
@Transactional(propagation = Propagation.REQUIRED)
public void methodA() {
    methodB();  // Will use methodA's transaction
    methodC();  // Will use methodA's transaction
}

public void methodB() {
    // Non-transactional but participates in methodA's transaction
}

@Transactional(propagation = Propagation.REQUIRES_NEW)
public void methodC() {
    // Will start a new separate transaction
}
```

In this example:

- `methodB` is non-transactional but will participate in `methodA`'s transaction because `methodA` is set to `REQUIRED`.
- `methodC` will start a new transaction because it's set to `REQUIRES_NEW`, separate from `methodA`'s transaction.

So, the `REQUIRED` setting in `methodA` won't force `methodC` to use the same transaction; `methodC`'s own transaction settings will take precedence.

2. **If `methodB` fails**:
   - `methodA`'s transaction rolls back, including all changes in `methodB`.
   - `methodC` is unaffected if it had already committed; otherwise, its own transaction rolls back.
3. **If `methodC` fails**:
   - `methodC`'s own separate transaction rolls back.
   - `methodA` and `methodB` are unaffected, unless `methodA` explicitly checks `methodC`'s status and decides to roll back.

Database Read Phenomena

1. **Dirty Reads**: Reading uncommitted changes from another transaction.
   - **Example**: Tx1 modifies a row. Tx2 reads the row before Tx1 commits. If Tx1 rolls back, Tx2 read invalid data.
2. **Non-Repeatable Reads**: A row read twice within the same transaction returns different results due to another transaction's modification.
   - **Example**: Tx1 reads a row. Tx2 updates that row and commits. Tx1 reads the row again, sees different data.
3. **Phantom Reads**: A query returns a different set of rows when repeated in the same transaction due to insert or delete operations from another transaction.
   - **Example**: Tx1 queries rows and gets 5 results. Tx2 inserts a new row that matches Tx1's query criteria and commits. Tx1 re-queries and gets 6 results.

Transaction Isolation Levels

1. **Read Uncommitted**: Allows dirty reads.
2. **Read Committed**: Prevents dirty reads but allows non-repeatable and phantom reads.
3. **Repeatable Read**: Prevents dirty and non-repeatable reads but allows phantom reads.
4. **Serializable**: Prevents all: dirty, non-repeatable, and phantom reads.

```sql
-- Set isolation level to Serializable to prevent all read phenomena
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;

-- Your queries here

COMMIT;
```

- In spring:

```
@Transactional(isolation = Isolation.READ_UNCOMMITTED)
@Transactional(isolation = Isolation.READ_COMMITTED)
@Transactional(isolation = Isolation.REPEATABLE_READ)
@Transactional(isolation = Isolation.SERIALIZABLE)
```

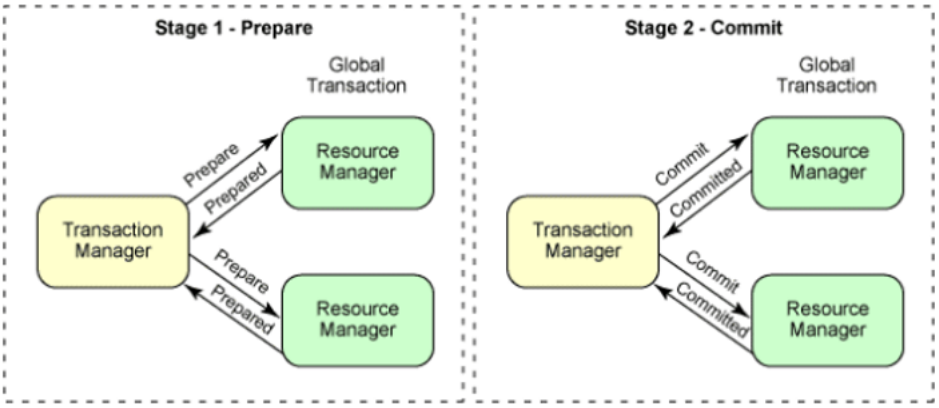
1. SQL Transaction

```sql
BEGIN TRANSACTION;
INSERT INTO orders (id, amount) VALUES (1, 100);
INSERT INTO order_items (order_id, product_id) VALUES (1, 101);
COMMIT;
```

2. Java Spring Transaction

```java
@Transactional
public void createOrder(Order order, List<OrderItem> items) {
    orderRepo.save(order);
    itemRepo.saveAll(items);
}
```

Two-phase commit



Two-Phase Commit (2PC)

Two-Phase Commit is a distributed transaction protocol to ensure consistency across multiple systems (e.g., databases, message queues). It involves two phases:

1. **Prepare Phase**: Each participant is asked to prepare to commit. They lock resources but don't commit yet.
2. **Commit/Rollback Phase**: If all participants report they can commit, the coordinator sends a commit command. Otherwise, it sends a rollback command.

The Optimistic Offline Lock pattern allows multiple transactions to access the same data without locking it. Instead, each transaction verifies the data hasn't changed before committing changes.

How it Works:

1. **Read**: Read data and store its version.
2. **Modify**: Perform operations offline, without locking data.
3. **Commit**: Before committing, check the stored version against the current version. If they match, commit the changes and update the version.

Pessimistic Offline Lock

1. **Read**: Acquire a lock when reading the data.
2. **Modify**: Perform offline operations. The data is locked for other transactions.
3. **Commit**: Commit changes and release the lock.

Pros:

- Less chance of conflict during the commit stage.
- Simple to understand and implement.

Cons:

- Resource-intensive due to locking.
- Potential for deadlocks.
- Reduced system throughput and responsiveness.

Each thread is associated with an instance of the class Thread.

- There are two basic strategies for using Thread objects to create a concurrent application.
- To directly control thread creation and management, simply instantiate Thread each time the application needs to initiate an asynchronous task.
- To abstract thread management from the rest of your application, pass the application's tasks to an *executor*.

Provide a Runnable object.

- The Runnable interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is passed to the Thread constructor, as in the HelloRunnable example:

```
public class HelloRunnable implements Runnable {
  public void run() {
    System.out.println("Hello from a thread!");
  }
  public static void main(String args[]) {
    (new Thread(new HelloRunnable())).start();
  }
}
```

Subclass Thread.

- The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the HelloThread example:

```
public class HelloThread extends Thread {
  public void run() {
    System.out.println("Hello from a thread!");
  }
  public static void main(String args[]) {
    (new HelloThread()).start();
  }
}
```

```
//Pause for 4 seconds
Thread.sleep(4000);
```

- A thread sends an interrupt by invoking interrupt on the Thread object for the thread to be interrupted.

```
for (int i = 0; i < importantInfo.length; i++) {
 // Pause for 4 seconds
 try {
  Thread.sleep(4000);
 } catch (InterruptedException e) {
  // We've been interrupted: no more messages.
  return;
 }
 // Print a message
 System.out.println(importantInfo[i]);
}
```

What if a thread goes a long time without invoking a method that throws InterruptedException?

- Then it must periodically invoke Thread.interrupted, which returns true if an interrupt has been received. For example:

```
for (int i = 0; i < inputs.length; i++) {
 heavyCrunch(inputs[i]);
 if (Thread.interrupted()) {
   // We've been interrupted: no more crunching.
   return;
 }
}
```

This form of communication is extremely efficient, but makes two kinds of errors possible: *thread interference* and *memory consistency errors*. The tool needed to prevent these errors is *synchronization*.

However, synchronization can introduce *thread contention*, which occurs when two or more threads try to access the same resource simultaneously *and* cause the Java runtime to execute one or more threads more slowly, or even suspend their execution. Starvation and livelock are forms of thread contention.

If the initial value of c is 0, their interleaved actions might follow this sequence:

- Thread A: Retrieve c.
- Thread B: Retrieve c.
- Thread A: Increment retrieved value; result is 1.
- Thread B: Decrement retrieved value; result is -1.
- Thread A: Store result in c; c is now 1.
- Thread B: Store result in c; c is now -1.

Thread A's result is lost, overwritten by Thread B.

Memory consistency errors occur when different threads have inconsistent views of what should be the same data.

The key to avoiding memory consistency errors is understanding the *happens-before* relationship.

- This relationship is simply a guarantee that memory writes by one specific statement are visible to another specific statement.

because there's no guarantee that thread A's change to counter will be visible to thread B — unless the programmer has established a happens-before relationship between these two statements.

```
public class SynchronizedCounter {
 private int c = 0;
 public synchronized void increment() { c++; }
 public synchronized void decrement() { c--; }
 public synchronized int value() { return c; }
}
```

Synchronization is built around an internal entity known as the *intrinsic lock* or *monitor lock*.

- Intrinsic locks play a role in both aspects of synchronization: enforcing exclusive access to an object's state and establishing happens-before relationships that are essential to visibility.

Every object has an intrinsic lock associated with it.

- By convention, a thread that needs exclusive and consistent access to an object's fields has to *acquire* the object's intrinsic lock before accessing them, and then *release* the intrinsic lock when it's done with them.
- A thread is said to *own* the intrinsic lock between the time it has acquired the lock and released the lock. As long as a thread owns an intrinsic lock, no other thread can acquire the same lock. The other thread will block when it attempts to acquire the lock.

When a thread releases an intrinsic lock,

- a happens-before relationship is established between that action and any subsequent acquisition of the same lock.

- **Locks In Synchronized Methods**

- When a thread invokes a synchronized method,
  - it automatically acquires the intrinsic lock for that method's object and releases it when the method returns.
  - The lock release occurs even if the return was caused by an uncaught exception.

- You might wonder what happens when a static synchronized method is invoked,
  - since a static method is associated with a class, not an object.
  - In this case, the thread acquires the intrinsic lock for the Class object associated with the class. Thus access to class's static fields is controlled by a lock that's distinct from the lock for any instance of the class.

Intrinsic Locks and Synchronization

- Synchronized statements are also useful for improving concurrency with fine-grained synchronization.

```
public class MsLunch {
  private long c1 = 0;
  private long c2 = 0;

  private Object lock1 = new Object();
  private Object lock2 = new Object();

  public void inc1() {
    synchronized(lock1)
     { c1++; }
  }
  public void inc2() {
    synchronized(lock2)
     { c2++; }
  }
}
```

- Use this idiom with extreme care. You must be absolutely sure that it really is safe to interleave access of the affected fields.

Allowing a thread to acquire the same lock more than once enables *reentrant synchronization*.

- This describes a situation where synchronized code, directly or indirectly, invokes a method that also contains synchronized code, and both sets of code use the same lock.
- Without reentrant synchronization, synchronized code would have to take many additional precautions to avoid having a thread cause itself to block.

There are actions you can specify that are atomic:

- Reads and writes are atomic for reference variables and for most primitive variables (all types except long and double).
- Reads and writes are atomic for *all* variables declared volatile (*including* long and double variables).

- Atomic actions cannot be interleaved, so they can be used without fear of thread interference.
  - However, this does not eliminate all need to synchronize atomic actions, because memory consistency errors are still possible.

- Using volatile variables reduces the risk of memory consistency errors,
  - because any write to a volatile variable establishes a happens-before relationship with subsequent reads of that same variable

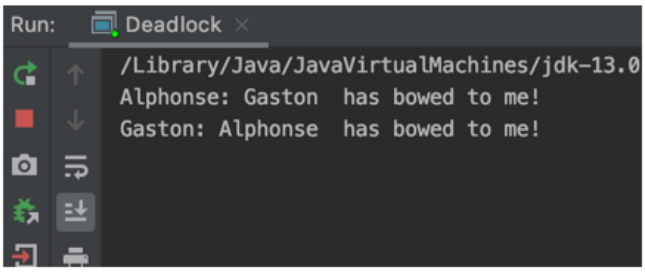
- A concurrent application's ability to execute in a timely manner is known as its *liveness*.

- **Deadlock**
  - *Deadlock* describes a situation where two or more threads are blocked forever, waiting for each other.
  - Here's an example.
    - Alphonse and Gaston are friends, and great believers in courtesy.
    - A strict rule of courtesy is that when you bow to a friend, you must remain bowed until your friend has a chance to return the bow.
    - Unfortunately, this rule does not account for the possibility that two friends might bow to each other at the same time.

```
package org.reins;

public class Deadlock {
   static class Friend {
      private final String name;
      public Friend(String name) {
         this.name = name;
      }
      public String getName() {
         return this.name;
      }
      public synchronized void bow(Friend bower) {
         System.out.format("%s: %s"
                  + "  has bowed to me!%n",
               this.name, bower.getName());
         bower.bowBack(this);
      }
      public synchronized void bowBack(Friend bower) {
         System.out.format("%s: %s"
                  + " has bowed back to me!%n",
               this.name, bower.getName());
      }
   }

   public static void main(String[] args) {
      final Friend alphonse =
            new Friend("Alphonse");
      final Friend gaston =
            new Friend("Gaston");
      new Thread(new Runnable() {
         public void run() { alphonse.bow(gaston); }
      }).start();
      new Thread(new Runnable() {
         public void run() { gaston.bow(alphonse); }
      }).start();
   }
}
```



- **Starvation**
  - *Starvation* describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads.

- **Livelock**
  - A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then *livelock* may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are simply too busy responding to each other to resume work.

- The most common coordination idiom is the *guarded block*.
- Such a block begins by polling a condition that must be true before the block can proceed.

Suppose,

- for example guardedJoy is a method that must not proceed until a shared variable joy has been set by another thread.

```
public void guardedJoy() {
  // Simple loop guard. Wastes
  // processor time. Don't do this!
  while(!joy) { }
  System.out.println("Joy has been achieved!");
}
```

```
public synchronized void guardedJoy() {
  // This guard only loops once for each special event, which may not
  // be the event we're waiting for.
  while(!joy) {
    try { wait(); }
    catch (InterruptedException e) {}
  }
  System.out.println("Joy and efficiency have been achieved!");
}
```

Let's use guarded blocks to create a *Producer-Consumer* application.

- This kind of application shares data between two threads:
- the *producer*, that creates the data, and the *consumer*, that does something with it.
- The two threads communicate using a shared object.
- Coordination is essential:
  - the consumer thread must not attempt to retrieve the data before the producer thread has delivered it,
  - and the producer thread must not attempt to deliver new data if the consumer hasn't retrieved the old data.

```
public class Drop {
  // Message sent from producer to consumer.
  private String message;
  // True if consumer should wait for producer to send message,
  // false if producer should wait for consumer to retrieve message.
  private boolean empty = true;
  public synchronized String take() {
    // Wait until message is available.
    while (empty) {
      try {
        wait();
      } catch (InterruptedException e) {}
    }
    // Toggle status.
    empty = true;
    // Notify producer that status has changed.
    notifyAll();
    return message;
  }
```

```
public synchronized void put(String message) {
  // Wait until message has been retrieved.
  while (!empty) {
    try {
      wait();
    } catch (InterruptedException e) {}
  }
  // Toggle status.
  empty = false;
  // Store message.
  this.message = message;
  // Notify consumer that status has changed.
  notifyAll();
}
```

```
import java.util.Random;
public class Producer implements Runnable {
  private Drop drop;
  public Producer(Drop drop) { this.drop = drop; }

  public void run() {
    String importantInfo[] = { "Mares eat oats", "Does eat oats",
          "Little lambs eat ivy", "A kid will eat ivy too" };
    Random random = new Random();
    for (int i = 0; i < importantInfo.length; i++) {
     drop.put(importantInfo[i]);
     try {
        Thread.sleep(random.nextInt(5000));
     } catch (InterruptedException e) {}
    }
    drop.put("DONE");
  }
}
```

```
import java.util.Random;
public class Consumer implements Runnable {
  private Drop drop;
  public Consumer(Drop drop) { this.drop = drop; }

  public void run() {
    Random random = new Random();
    for (String message = drop.take();
          ! message.equals("DONE"); message = drop.take()) {
      System.out.format("MESSAGE RECEIVED: %s%n", message);
      try { Thread.sleep(random.nextInt(5000)); }
      catch (InterruptedException e) {}
    }
  }
}
```



```
public class ProducerConsumerExample {
  public static void main(String[] args) {
    Drop drop = new Drop();
    (new Thread(new Producer(drop))).start();
    (new Thread(new Consumer(drop))).start();
  }
}
```

An object is considered *immutable* if its state cannot change after it is constructed.

- Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple reliable code.

Immutable objects are particularly useful in concurrent applications.

- Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.

- Suppose, for example, a thread executes the following code:

```
SynchronizedRGB color = new SynchronizedRGB(0, 0, 0, "Pitch Black");
...
int myColorInt = color.getRGB();          //Statement 1
String myColorName = color.getName(); //Statement 2
```

- If another thread invokes color.set after Statement 1 but before Statement 2, the value of myColorInt won't match the value of myColorName. To avoid this outcome, the two statements must be bound together:

```
synchronized (color) {
  int myColorInt = color.getRGB();
  String myColorName = color.getName();
}
```

- This kind of inconsistency is only possible for mutable objects — it will not be an issue for the immutable version of SynchronizedRGB.

The following rules define a simple strategy for creating immutable objects.

- Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
- Make all fields final and private.
- Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.
- If the instance fields include references to mutable objects, don't allow those objects to be changed:
  - Don't provide methods that modify the mutable objects.
  - Don't share references to the mutable objects.

We'll look at some of the high-level concurrency. Most of these features are implemented in the new java.util.concurrent packages.

- Lock objects support locking idioms that simplify many concurrent applications.
- Executors define a high-level API for launching and managing threads. Executor implementations provided by java.util.concurrent provide thread pool management suitable for large-scale applications.
- Concurrent collections make it easier to manage large collections of data, and can greatly reduce the need for synchronization.
- Atomic variables have features that minimize synchronization and help avoid memory consistency errors.
- ThreadLocalRandom provides efficient generation of pseudorandom numbers from multiple threads.

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Counter {
    private int count = 0;
    private final Lock lock = new ReentrantLock();

    public void increment() {
        lock.lock();
        try {
            System.out.println(Thread.currentThread().getName() + " 
            count++;
            System.out.println(Thread.currentThread().getName() + " 
        } finally {
            lock.unlock();
            System.out.println(Thread.currentThread().getName() + " 
        }
    }

    public int getCount() {
        return count;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter();

        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                counter.increment();
            }
        }, "Thread-1");

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                counter.increment();
            }
        }, "Thread-2");

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Final Count is: " + counter.getCount());
    }
}
```

The `join()` method in Java is used to ensure that a thread completes its execution before the main thread or any other thread can proceed. Essentially, calling `join()` on a thread instance will make the calling thread wait until the joined thread completes its task.

- This works well for small applications, but in large-scale applications, it makes sense to separate thread management and creation from the rest of the application.

Objects that encapsulate these functions are known as *executors*.

- Executor Interfaces define the three executor object types.
- Thread Pools are the most common kind of executor implementation.
- Fork/Join is a framework for taking advantage of multiple processors.

The Executor Interface

- The Executor interface provides a single method, execute, designed to be a drop-in replacement for a common thread-creation idiom. If r is a Runnable object, and e is an Executor object you can replace
- (new Thread(r)).start();
- With
- e.execute(r);

- The ExecutorService Interface
  - The ExecutorService interface supplements execute with a similar, but more versatile submit method.
  - Like execute, submit accepts Runnable objects, but also accepts Callable objects, which allow the task to return a value.
  - The submit method returns a Future object, which is used to retrieve the Callable return value and to manage the status of both Callable and Runnable tasks.

Most of the executor implementations in java.util.concurrent use *thread pools*, which consist of *worker threads*.

- This kind of thread exists separately from the Runnable and Callable tasks it executes and is often used to execute multiple tasks.
- Using worker threads minimizes the overhead due to thread creation
- One common type of thread pool is the *fixed thread pool*. This type of pool always has a specified number of threads running.
- An important advantage of the fixed thread pool is that applications using it *degrade gracefully*.

Certainly! Java's Executor framework provides a way to manage threads more efficiently, and thread pools are a significant part of this framework. Using thread pools can improve performance by reusing threads instead of creating a new thread for every task, thereby reducing the overhead associated with thread creation.

1. **Basic Executor**: For simpler use-cases, an Executor allows you to run tasks in a single worker thread.
2. **Thread Pool**: A thread pool reuses a fixed number of threads to execute tasks. `newFixedThreadPool` gives you a fixed-size thread pool.
3. **Scheduled Thread Pool**: Scheduled thread pools allow you to execute tasks at regular intervals or after a certain delay.

```
if (my portion of the work is small enough)
  do the work directly
else
  split my work into two pieces
  invoke the two pieces and wait for the results
```

```
protected static int sThreshold = 100000;

protected void compute() {
 if (mLength < sThreshold) {
   computeDirectly();
   return;
 }

 int split = mLength / 2;

 invokeAll(
       new ForkBlur(mSource, mStart, split, mDestination),
       new ForkBlur(mSource, mStart + split, mLength - split, mDestination));
}
```

- Create a task that represents all of the work to be done.

```
// source image pixels are in src
// destination image pixels are in dst
ForkBlur fb = new ForkBlur(src, 0, src.length, dst);
```

- Create the ForkJoinPool that will run the task.

```
ForkJoinPool pool = new ForkJoinPool();
```

- Run the task.

```
pool.invoke(fb);
```

Replacing the int field with an AtomicInteger allows us to prevent thread interference without resorting to synchronization, as in AtomicCounter:

```
import java.util.concurrent.atomic.AtomicInteger;
class AtomicCounter {
 private AtomicInteger c = new AtomicInteger(0);
 public void increment() {
  c.incrementAndGet();
 }
 public void decrement() {
  c.decrementAndGet();
 }
 public int value() {
  return c.get();
 }
}
```

- What is a Virtual Thread?
  - Like a platform thread, a virtual thread is also an instance of java.lang.Thread.
  - However, a virtual thread isn't tied to a specific OS thread.
  - A virtual thread still runs code on an OS thread.
  - However, when code running in a virtual thread calls a blocking I/O operation, the Java runtime suspends the virtual thread until it can be resumed. The OS thread associated with the suspended virtual thread is now free to perform operations for other virtual threads.
- Unlike platform threads,
  - virtual threads typically have a shallow call stack, performing as few as a single HTTP client call or a single JDBC query.
  - Virtual threads are suitable for running tasks that spend most of the time blocked, often waiting for I/O operations to complete. However, they aren't intended for long-running CPU-intensive operations.

- Memcached
  - Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

- Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

- Memcached is simple yet powerful.
  - Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches.
  - Its API is available for most popular languages.

- At heart it is a simple Key/Value store.

```
public class SimpleBookRepository implements
BookRepository {

    @Override
    @Cacheable("books")
    public Book getByIsbn(String isbn) {
       simulateSlowService();
       return new Book(isbn, "Some book");
    }

    // Don't do this at home
    private void simulateSlowService() {
       try {
          long time = 3000L;
          Thread.sleep(time);
       } catch (InterruptedException e) {
          throw new IllegalStateException(e);
       }
    }
}
```

- Se33537SpringcacheApplication.java

```
@SpringBootApplication
@EnableCaching
public class Se33537SpringcacheApplication {
    public static void main(String[] args) {
        SpringApplication.run(Se33537SpringcacheApplication.class, args);
    }
}
```

- pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

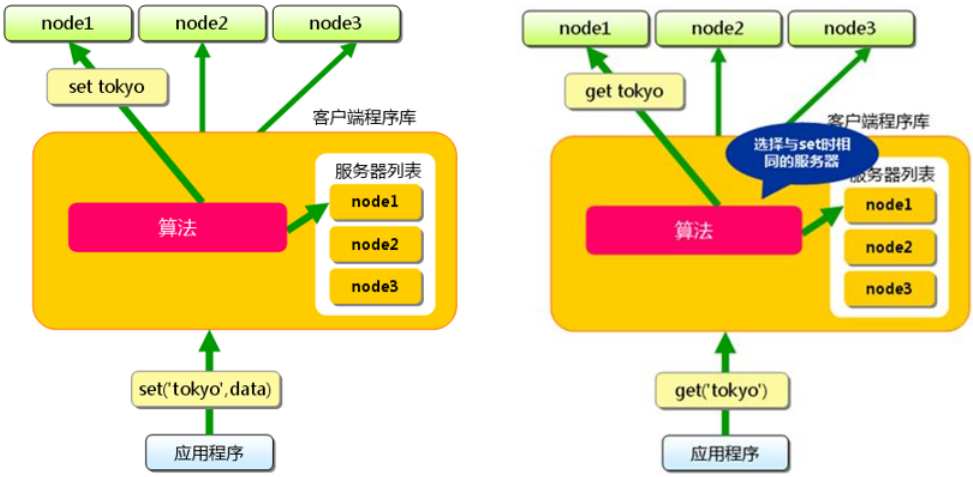
```java
@Override
public void run(String... args) throws Exception {
    logger.info(".... Fetching books");
    logger.info("isbn-1234 -->" + bookRepository.getByIsbn("isbn-1234"));
    logger.info("isbn-4567 -->" + bookRepository.getByIsbn("isbn-4567"));
    logger.info("isbn-1234 -->" + bookRepository.getByIsbn("isbn-1234"));
    logger.info("isbn-4567 -->" + bookRepository.getByIsbn("isbn-4567"));
    logger.info("isbn-1234 -->" + bookRepository.getByIsbn("isbn-1234"));
    logger.info("isbn-1234 -->" + bookRepository.getByIsbn("isbn-1234"));
}
}
```

- With Caching

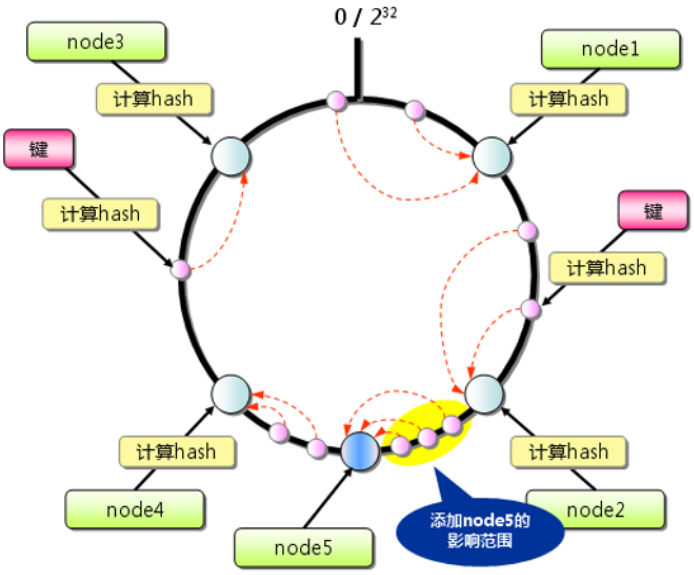
```
T15:56:14.670+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : .... Fetching books
T15:56:17.676+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T15:56:20.680+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-4567 -->Book{isbn='isbn-4567', title='Some book'}
T15:56:20.682+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T15:56:20.682+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-4567 -->Book{isbn='isbn-4567', title='Some book'}
T15:56:20.682+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T15:56:20.682+08:00  INFO 61868 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
```

- Without Caching

```
T16:31:28.147+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : .... Fetching books
T16:31:31.154+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T16:31:34.158+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-4567 -->Book{isbn='isbn-4567', title='Some book'}
T16:31:37.160+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T16:31:40.164+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-4567 -->Book{isbn='isbn-4567', title='Some book'}
T16:31:43.166+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
T16:31:46.171+08:00  INFO 63894 --- [           main] o.r.s.se3353_7_springcache.AppRunner     : isbn-1234 -->Book{isbn='isbn-1234', title='Some book'}
```



Distribution



- Redis is what is called a key-value store,
  - often referred to as a NoSQL database.
  - The essence of a key-value store is the ability to store some data, called a value, inside a key.

- Redis is an open source, BSD licensed, advanced key-value store.
  - It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

- In order to achieve its outstanding performance, Redis works with an in-memory dataset.
  - Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log.

- Redis also supports trivial-to-setup master-slave replication,
  - with very fast non-blocking first synchronization, auto-reconnection on net split and so forth.

Name	Type	Data storage options	Query types	Additional features
Redis	In-memory non-relational database	Strings, lists, sets, hashes, sorted sets	Commands for each data type for common access patterns, with bulk operations, and partial transaction support	Publish/Subscribe, master/slave replication, disk persistence, scripting (stored procedures)
memcached	In-memory key-value cache	Mapping of keys to values	Commands for create, read, update, delete, and a few others	Multithreaded server for additional performance
MySQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions	SELECT, INSERT, UPDATE, DELETE, functions, stored procedures	ACID compliant (with InnoDB), master/slave and master/master replication
PostgreSQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions, customizable types	SELECT, INSERT, UPDATE, DELETE, built-in functions, custom stored procedures	ACID compliant, master/slave replication, multi-master replication (third party)
MongoDB	On-disk non-relational document store	Databases of tables of schema-less BSON documents	Commands for create, read, update, delete, conditional queries, and more	Supports map-reduce operations, master/slave replication, sharding, spatial indexes

```
# Redis数据库索引（默认为0）
spring.data.redis.database=0
# Redis服务器地址
spring.data.redis.host=localhost
# Redis服务器连接端口
spring.data.redis.port=6379
# Redis服务器连接密码（默认为空
spring.data.redis.password=
# 连接超时时间（毫秒）
spring.data.redis.timeout=300
```

- PersonDaoImpl.java

```
@Repository
public class PersonDaoImpl implements PersonDao {
  @Autowired
  private PersonRepository personRepository;
  @Autowired
  private RedisTemplate redisTemplate;

  @Override
  public Person findOne(Integer id) {
    Person person = null;
    String p = (String)redisTemplate.opsForValue().get("user" + id);
    if (p == null) {
          person = personRepository.getOne(id);
          redisTemplate.opsForValue().set("user" + id, JSON.toJSONString(person));
    } else {
          person = JSON.parseObject(p, Person.class);
          System.out.println("Person: " + id + " is in Redis");
    }
    return person;
  }
}
```


- pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

- Lucene is a high performance, scalable Information Retrieval (IR) library.
  - It lets you add indexing and searching capabilities to your applications.
  - Lucene is a mature, free, open-source project implemented in Java.
  - it's a member of the popular Apache Jakarta family of projects, licensed under the liberal Apache Software License.

- Lucene provides a simple yet powerful core API
  - that requires minimal understanding of full-text indexing and searching.

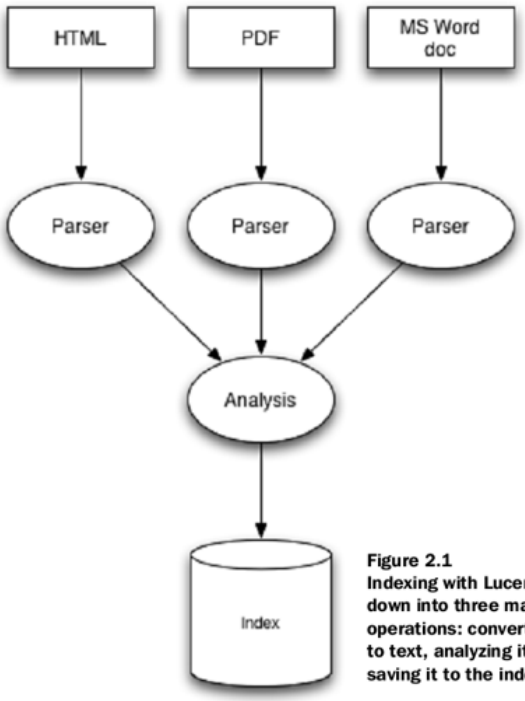


Figure 2.1
Indexing with Lucene breaks down into three main operations: converting data to text, analyzing it, and saving it to the index.

- At the heart of all search engines is the concept of indexing:
  - processing the original data into a highly efficient cross-reference lookup in order to facilitate rapid searching.

- To search large amounts of text quickly, you must first index that text and convert it into a format that will let you search it rapidly, eliminating the slow sequential scanning process.
- This conversion process is called indexing, and its output is called an index.
- You can think of an index as a data structure that allows fast random access to words stored inside it.

- Searching is the process of looking up words in an index to find documents where they appear.

- The quality of a search is typically described using precision and recall metrics.
  - Recall measures how well the search system finds relevant documents, whereas precision measures how well the system filters out the irrelevant documents.

- A number of other factors
  - speed and the ability to quickly search large quantities of text.
  - Support for single and multi term queries, phrase queries, wildcards, result ranking, and sorting are also important, as is a friendly syntax for entering those queries.

- To use Lucene, an application should:Create Documents by adding Fields;
  1. Create an IndexWriter and add documents to it with addDocument();
  2. Call QueryParser.parse() to build a query from a string; and
  3. Create an IndexSearcher and pass the query to its search() method.

- The fundamental concepts in Lucene are index, document, field and term.

- An index contains a sequence of documents.
  - A document is a sequence of fields.
  - A field is a named sequence of terms.
  - A term is a sequence of bytes.

- The same sequence of bytes in two different fields is considered a different term.
  - Thus terms are represented as a pair: the string naming the field, and the bytes within the field.

- Scoring
  - Scoring is very much dependent on the way documents are indexed, so it is important to understand indexing.
  - Be sure to use the useful IndexSearcher.explain(Query, doc) to understand how the score for a certain matching document was computed.
  - Generally, the Query determines which documents match (a binary decision), while the Similarity determines how to assign scores to the matching documents.

- Fields and Documents
  - In Lucene, the objects we are scoring are Documents.
  - A Document is a collection of Fields.
  - It is important to note that Lucene scoring works on Fields and then combines the results to return Documents.
  - This is important because two Documents with the exact same content, but one having the content in two Fields and the other in one Field may return different scores for the same query due to length normalization.

- Score Boosting
  - Lucene allows influencing the score contribution of various parts of the query by wrapping with BoostQuery.

21-35 for exam

- Contents
  - WS Overview
  - SOAP WS
  - RESTful WS

Certainly! SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language) are both standards used in web services, which allow different applications to communicate with each other over the internet. Let's break down each one:

WSDL (Web Services Description Language)

WSDL is an XML-based interface definition language that is used for describing the functionality offered by a web service. It specifies the operations (methods) and messages used during the interaction with the web service.

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-services</artifact
  </dependency>
  <!-- tag::springws[] -->
  <dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
  </dependency>
  <!-- end::springws[] -->
</dependencies>
```

SOAP web service

JAX-WS Web Service

SOAP-based Web Services
- Coupling with the message format
- Coupling with the encoding of WS
- Parse and assemble SOAP
- Need a WSDL to describe the details of WS
- Need a proxy generated from WSDL

It is a time-cost way to implement Web Service with SOAP
- We should find a new way to implement WS

REpresentational State Transfer

- Representational:
  - All data are resources. Representation for client.
  - Each resource can have different representations
  - Each resource has its own unique identity(URI)

- State:
  - It refers to state of client. Server is stateless.
  - The representation of resource is a state of client.

- Transfer:
  - Client's representation will be transferred when client access different resources by different URI.
  - It means the states of client are also transferred.
  - That is Representation State Transfer

- Microservice architectures are the 'new normal'.
  - Building small, self-contained, ready to run applications can bring great flexibility and added resilience to your code.
  - Spring Boot's many purpose-built features make it easy to build and run your microservices in production at scale.
  - And don't forget, no microservice architecture is complete without Spring Cloud – easing administration and boosting your fault-tolerance.

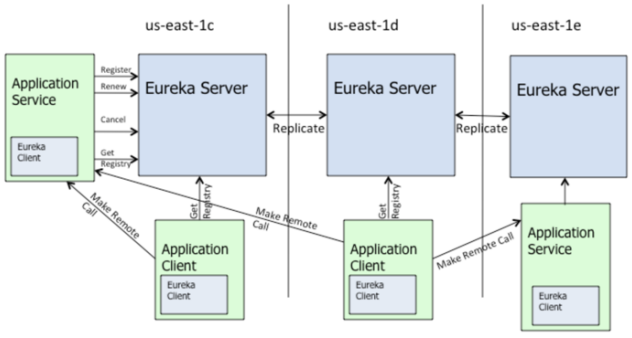
- What are microservices?
  - Microservices are a modern approach to software whereby application code is delivered in small, manageable pieces, independent of others.
- Why build microservices?
  - Their small scale and relative isolation can lead to many additional benefits, such as easier maintenance, improved productivity, greater fault tolerance, better business alignment, and more.

- The distributed nature of microservices brings challenges.
  - service discovery, load-balancing, circuit-breaking, distributed tracing, and monitoring

- In the cloud,
  - applications can't always know the exact location of other services.
  - A service registry, such as Netflix Eureka, or a sidecar solution, such as HashiCorp Consul, can help.
  - Spring Cloud provides DiscoveryClient implementations for popular registries such as Eureka, Consul, Zookeeper, and even Kubernetes' built-in system.
  - There's also a Spring Cloud Load Balancer to help you distribute the load carefully among your service instances.

- Eureka is
  - a REST (Representational State Transfer) based service that is primarily used in the AWS cloud fo locating services for the purpose of load balancing and failover of middle-tier servers.
  - We call this service, the **Eureka Server**.
  - Eureka also comes with a Java-based client component, the **Eureka Client**, which makes interactions with the service much easier.
  - The client also has a built-in load balancer that does basic round-robin load balancing.
  - At Netflix, a much more sophisticated load balancer wraps Eureka to provide weighted load balancing based on several factors like traffic, resource usage, error conditions etc. to provide superior resiliency.

- There is **one** eureka cluster per **region** which knows only about instances in its region. There is the least **one** eureka server per **zone** to handle zone failures.



Service Application

pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
</dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
@EnableEurekaServer
@SpringBootApplication
public class ServiceRegistrationAndDiscoveryServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(
      ServiceRegistrationAndDiscoveryServiceApplication.class, args);
  }
}
```

```
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

Client Application

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
</dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
@RequestMapping("/service-instances/{applicationName}")
public List<ServiceInstance> serviceInstancesByApplicationName(
    @PathVariable String applicationName) {
  return this.discoveryClient.getInstances(applicationName);
}
```

resources/application.properties

```
spring.application.name=a-bootiful-client
```

Spring Cloud Gateway

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
    <exclusions>
      <exclusion>
        <artifactId>spring-boot-starter-web</artifactId>
        <groupId>org.springframework.boot</groupId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

```
<!-- https://mvnrepository.com/artifact/io.netty/netty-resolver-dns-native-macos -->
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-resolver-dns-native-macos</artifactId>
  <version>4.1.97.Final</version>
  <classifier>osx-aarch_64</classifier>
</dependency>
```

- Main Java Class

```
@RestController
@RequestMapping("/books")
@SpringBootApplication
public class RoutingAndFilteringBookApplication {

  @RequestMapping(value = "/available")
  public String available() {
    return "Spring in Action";
  }

  @RequestMapping(value = "/checked-out")
  public String checkedOut() {
    return "Spring Boot in Action";
  }

  public static void main(String[] args) {
    SpringApplication.run(RoutingAndFilteringBookApplication.class, args);
  }
}
```

```
spring.application.name=book

server.port=8090
```

- Serverless applications
  - take advantage of modern cloud computing capabilities and abstractions to let you focus on logic rather than on infrastructure.
  - In a serverless environment, you can concentrate on writing application code while the underlying platform takes care of scaling, runtimes, resource allocation, security, and other "server" specifics.

    - Serverless workloads are "event-driven workloads that aren't concerned with aspects normally handled by server infrastructure."
    - Concerns like "how many instances to run" and "what operating system to use" are all managed by a Function as a Service platform (or FaaS), leaving developers free to focus on business logic.

    Serverless characteristics?
    - Serverless applications have a number of specific characteristics, including:
      - Event-driven code execution with triggers
      - Platform handles all the starting, stopping, and scaling chores
      - Scales to zero, with low to no cost when idle
      - Stateless

Spring Cloud Function
- provides capabilities that lets Spring developers take advantage of serverless or FaaS platforms.

Spring Cloud Function
- provides adaptors so that you can run your functions on the most common FaaS services including :
- Amazon Lambda, Apache OpenWhisk, Microsoft Azure, and Project Riff.

- application.yaml

```
spring:
  application:
    name: book-service

eureka:
  instance:
    prefer-ip-address: true
    ip-address: localhost
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://localhost:8040/eureka

server:
  port: 11130
```

Database performance depends on several factors at the database level,

- such as tables, queries, and configuration settings.

Advanced users look for opportunities to improve the MySQL software itself,

- or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work?
- Are the right indexes in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use?.
- Does each table use an appropriate row format?
- Does the application use an appropriate locking strategy?
- Are all memory areas used for caching sized correctly?

System bottlenecks typically arise from these sources:

- Disk seeks.
- Disk reading and writing.
- CPU cycles.
- Memory bandwidth.

The best way to improve the performance of SELECT operations is

- to create indexes on one or more of the columns that are tested in the query.
- The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the WHERE clause, and retrieve the other column values for those rows.
- All MySQL data types can be indexed.

Although it can be tempting to create an index for every possible column used in a query,

- unnecessary indexes waste space and waste time for MySQL to determine which indexes to use.
- Indexes also add to the cost of inserts, updates, and deletes because each index must be updated.
- You must find the right balance to achieve fast queries using the optimal set of indexes.

- Indexes are used to find rows with specific column values quickly.
  - Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows.
  - The larger the table, the more this costs.
  - If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data.
  - This is much faster than reading every row sequentially.

- Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. Exceptions:
  - Indexes on spatial data types use R-trees;
  - MEMORY tables also support hash indexes;
  - InnoDB uses inverted lists for FULLTEXT indexes.

MySQL uses indexes for these operations:

- To find the rows matching a WHERE clause quickly.
- To eliminate rows from consideration.
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows.
- To retrieve rows from other tables when performing joins.
- To find the MIN() or MAX() value for a specific indexed column *key_col*.
- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable index (for example, ORDER BY *key_part1*, *key_part2*).
- In some cases, a query can be optimized to retrieve values without consulting the data rows.

Indexes are less important for queries on small tables,

- or big tables where report queries process most or all of the rows.
- When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for tl query.

Primary Key Optimization

- The primary key for a table represents
  - the column or set of columns that you use in your most vital queries.
  - It has an associated index, for fast query performance.
  - Query performance benefits from the NOT NULL optimization, because it cannot include any NULL values.
  - With the InnoDB storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

- If your table is big and important,
  - but does not have an obvious column or set of columns to use as a primary key,
  - you might create a separate column with auto-increment values to use as the primary key.
  - These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

SPATIAL Index Optimization

- MySQL permits creation of SPATIAL indexes on NOT NULL geometry-valued columns.
  - For comparisons to work properly, each column in a SPATIAL index must be SRID-restricted.
  - That is, the column definition must include an explicit SRID attribute, and all column values must have the same SRID.

- The optimizer considers SPATIAL indexes only for SRID-restricted columns:
  - Indexes on columns restricted to a Cartesian SRID enable Cartesian bounding box computations.
  - Indexes on columns restricted to a geographic SRID enable geographic bounding box computations.

Foreign Key Optimization

- If a table has many columns, and you query many different combinations of columns,
  - it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table.
  - That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation.
  - Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk.
  - (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

Column Indexes

- **Index Prefixes**
  - With *col_name*(*N*) syntax in an index specification for a string column, you can create an index that uses only the first *N* characters of the column.
    - Indexing only a prefix of column values in this way can make the index file much smaller. When you index a BLOB or TEXT column, you *must* specify a prefix length for the index. For example:
      CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  - Prefixes can be up to 767 bytes long for InnoDB tables that use the REDUNDANT or COMPACT row format.
  - The prefix length limit is 3072 bytes for InnoDB tables that use the DYNAMIC or COMPRESSED row format.
  - For MyISAM tables, the prefix length limit is 1000 bytes.
  - If a search term exceeds the index prefix length, the index is used to exclude non-matching rows, and the remaining rows are examined for possible matches.

- **Spatial Indexes**
  - You can create indexes on spatial data types.
  - MyISAM and InnoDB support R-tree indexes on spatial types.
  - Other storage engines use B-trees for indexing spatial types (except for ARCHIVE, which does not support spatial type indexing).

- **Indexes in the MEMORY Storage Engine**
  - The MEMORY storage engine uses HASH indexes by default, but also supports BTREE indexes.

Multiple-Column Indexes

- MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index.

- Suppose that you have the SELECT statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

- If an index exists on (col1, col2, col3),
  - only the first two queries use the index.
  - The third and fourth queries do involve indexed columns, but do not use an index to perform lookups because (col2) and (col2, col3) are not leftmost prefixes of (col1, col2, col3).

- For example, the following SELECT statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

- The following SELECT statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

Comparison of B-Tree and Hash Indexes

- **Hash Index Characteristics**
  - They are used only for equality comparisons that use the = or <=> operators (but are *very* fast). They are not used for comparison operators such as < that find a range of values.
  - Systems that rely on this type of single-value lookup are known as "key-value stores"; to use MySQL for such applications, use hash indexes wherever possible.
  - The optimizer cannot use a hash index to speed up ORDER BY operations. (This type of index cannot be used to search for the next entry in order.)
  - MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use).
  - Only whole keys can be used to search for a row.

B-tree Index

Creating a B-tree index is usually the default in SQL databases, and you can create one like this:

sql Copy code

```sql
CREATE INDEX index_name ON table_name (column_name);
```

Hash Index

For databases that support hash indexing directly, like PostgreSQL, you can create a hash index like this:

sql Copy code

```sql
CREATE INDEX index_name ON table_name USING HASH (column_name);
```

EXAMPLE

sql Copy code

```sql
SELECT * FROM employees WHERE id = 5;
```

Hash Index: In this case, a hash index on the `id` column would be beneficial. The database

sql Copy code

```sql
SELECT * FROM employees WHERE salary BETWEEN 50000 AND 70000 ORDER BY salar
```

3-tree Index: A B-tree index on the `salary` column would excel here. It would make the

Descending Indexes

- MySQL supports descending indexes:
  - DESC in an index definition is no longer ignored but causes storage of key values in descending order.
  - Previously, indexes could be scanned in reverse order but at a performance penalty.
  - A descending index can be scanned in forward order, which is more efficient.
  - Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others.

Optimizing Data Size

- Design your tables to minimize their space on the disk.
  - This can result in huge improvements by reducing the amount of data written to and read from disk.
  - Smaller tables normally require less main memory while their contents are being actively processed during query execution.
  - Any space reduction for table data also results in smaller indexes that can be processed faster.

- You can get better performance for a table and minimize storage space by using the techniques listed here:
  - Table Columns
  - Row Format
  - Indexes
  - Joins
  - Normalization

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory.
  - For example, use the smaller integer types if possible to get smaller tables. MEDIUMINT is often a better choice than INT because a MEDIUMINT column uses 25% less space.
- Declare columns to be NOT NULL if possible.
  - It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is NULL.
  - You also save some storage space, one bit per column.
  - If you really need NULL values in your tables, use them. Just avoid the default setting that allows NULL values in every column.

- **Row Format**
  - InnoDB tables are created using the DYNAMIC row format by default.
    - To use a row format other than DYNAMIC, configure innodb_default_row_format, or specify the ROW_FORMAT option explicitly in a CREATE TABLE or ALTER TABLE statement.
    - The compact family of row formats, which includes COMPACT, DYNAMIC, and COMPRESSED, decreases row storage space at the cost of increasing CPU use for some operations.
    - The compact family of row formats also optimizes CHAR column storage when using a variable-length character set such as utf8mb3 or utf8mb4.
  - With ROW_FORMAT=REDUNDANT, CHAR(*N*) occupies *N* × the maximum byte length of the character set.
    - Many languages can be written primarily using single-byte utf8 characters, so a fixed storage length often wastes space.
  - With the compact family of rows formats, InnoDB allocates a variable amount of storage in the range of *N* to *N* × the maximum byte length of the character set for these columns by stripping trailing spaces.
    - The minimum storage length is *N* bytes to facilitate in-place updates in typical cases.

- **Indexes**
  - The primary index of a table should be as short as possible.
    - This makes identification of each row easy and efficient.
    - For InnoDB tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.
  - Create only the indexes that you need to improve query performance.
    - Indexes are good for retrieval, but slow down insert and update operations.
    - If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used.
    - If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.

Indexes

- If it is very likely that a long string column has a unique prefix on the first number of characters,
  - it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column.
  - Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks.

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often.
  - This is especially true if it is a dynamic-format table
  - and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.
- Declare columns with identical information in different tables with identical data types,
  - to speed up joins based on the corresponding columns.
- Keep column names simple, so that you can use the same name across different tables and simplify join queries.
  - For example, in a table named customer, use a column name of name instead of customer_name.
  - To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as third normal form).
  - Instead of repeating lengthy values such as names and addresses,
  - assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.
- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data,
  - for example in a business intelligence scenario where you analyze all the data from large tables,
  - you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers,
  - prefer numeric columns to string columns.
  - Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.
- If you are using numeric data,
  - it is faster in many cases to access information from a database (using a live connection) than to access a text file.
  - Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses.
  - You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

Optimizing for Character and String Types

- Use binary collation order for fast comparison and sort operations,
  - when you do not need language-specific collation features. You can use the BINARY operator to use binary collation within a particular query.
- When comparing values from different columns,
  - declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.
- For column values less than 8KB in size, use binary VARCHAR instead of BLOB.
  - The GROUP BY and ORDER BY clauses can generate temporary tables, and these temporary tables can use the MEMORY storage engine if the original table does not contain any BLOB columns.

Optimizing for Character and String Types

- If a table contains string columns such as name and address, but many queries do not retrieve those columns,
  - consider splitting the string columns into a separate table and using join queries with a foreign key when necessary.
  - When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block.
  - Such compact tables reduce disk I/O and memory usage for common queries.
- When you use a randomly generated value as a primary key in an InnoDB table,
  - prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, InnoDB can insert and retrieve them faster.

Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first.
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column,
  - consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- You could put the BLOB-specific table on a different storage device or even a separate database instance.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries.
  - Since hash functions can produce duplicate results for different inputs, you still include a clause AND *blob_column = long_string_value* in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

How MySQL Opens and Closes Tables

- When you execute a **mysqladmin status** command, you should see something like this:

  Uptime: 426 Running threads: 1 Questions: 11082

  Reloads: 1 Open tables: 12
- The Open tables value of 12 can be somewhat puzzling if you have fewer than 12 tables.
- MySQL is multithreaded, so there may be many clients issuing queries for a given table simultaneously.
  - To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session.
  - This uses additional memory but normally increases performance.
  - With MyISAM tables, one extra file descriptor is required for the data file for each client that has the table open.

low MySQL Opens and Closes Tables

- MySQL closes an unused table and removes it from the table cache under the following circumstances:
  - When the cache is full and a thread tries to open a table that is not in the cache.
  - When the cache contains more than table_open_cache entries and a table in the cache is no longer being used by any threads.
  - When a table-flushing operation occurs. This happens when someone issues a FLUSH TABLES statement or executes a **mysqladmin flush-tables** or **mysqladmin refresh** command.

If you have many MyISAM tables in the same database directory,

- open, close, and create operations are slow.

If you execute SELECT statements on many different tables,

- there is a little overhead when the table cache is full,
- because for every table that has to be opened, another must be closed.
- You can reduce this overhead by increasing the number of entries permitted in the table cache.

MySQL has no limit on the number of databases.

- The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of tables.

- The underlying file system may have a limit on the number of files that represent tables.
- Individual storage engines may impose engine-specific constraints. InnoDB permits up to 4 billion tables.

- The effective maximum table size for MySQL databases is
  - usually determined by operating system constraints on file sizes, not by MySQL internal limits.
  - For up-to-date information operating system file size limits, refer to the documentation specific to your operating system.
  - Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

If you encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.
- You are using InnoDB tables and have run out of room in an InnoDB tablespace file.
  - The maximum tablespace size is also the maximum size for a table.
  - Generally, partitioning of tables into multiple tablespace files is recommended for tables larger than 1TB in size.
- You have hit an operating system file size limit.
  - For example, you are using MyISAM tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a MyISAM table and the space required for the table exceeds what is permitted by the internal pointer size.
  - MyISAM permits data and index files to grow up to 256TB by default, but this limit can be changed up to the maximum permissible size of 65,536TB ($256^7 - 1$ bytes).

Column Count Limits

- MySQL has hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact column limit depends on several factors:
  - The maximum row size for a table constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size.
  - The storage requirements of individual columns constrain the number of columns that fit within a given maximum row size.
  - Storage engines may impose additional restrictions that limit table column count. For example, InnoDB has a limit of 1017 columns per table.
  - Functional key parts are implemented as hidden virtual generated stored columns, so each functional key part in a table index counts against the table total column limit.

Row Size Limits

- The maximum row size for a given table is determined by several factors:
  - The internal representation of a MySQL table has a maximum row size limit of 65,535 bytes, even if the storage engine is capable of supporting larger rows. BLOB and TEXT columns only contribute 9 to 12 bytes toward the row size limit because their contents are stored separately from the rest of the row.
  - The maximum row size for an InnoDB table, which applies to data stored locally within a database page, is slightly less than half a page for 4KB, 8KB, 16KB, and 32KB innodb_page_size settings.
  - Different storage formats use different amounts of page header and trailer data, which affects the amount of storage available for rows.

- In InnoDB, having a long PRIMARY KEY (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space.
  - Create an AUTO_INCREMENT column as the primary key if your primary key is long, or index a prefix of a long VARCHAR column instead of the entire column.
- Use the VARCHAR data type instead of CHAR to store variable-length strings or for columns with many NULL values.
- For tables that are big, or contain lots of repetitive text or numeric data, consider using COMPRESSED row format.

Optimizing InnoDB Transaction Management

- To minimize the chance of this issue occurring:
  - Increase the size of the buffer pool so that all the data change changes can be cached rather than immediately written to disk.
  - Set innodb_change_buffering=all so that update and delete operations are buffered in addition to inserts.
  - Consider issuing COMMIT statements periodically during the big data change operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.
- A long-running transaction can prevent InnoDB from purging data that was changed by a different transaction.
  - When rows are modified or deleted within a long-running transaction, other transactions using the READ COMMITTED and REPEATABLE READ isolation levels have to do more work to reconstruct the older data if they read those same rows.
  - When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the covering index technique.

Backup and Recovery

- It is important to back up your databases
  - so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake.
  - Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replica servers.
- Several backup and recovery topics with which you should be familiar:
  - Types of backups: Logical versus physical, full versus incremental, and so forth.
  - Methods for creating backups.
  - Recovery methods, including point-in-time recovery.
  - Backup scheduling, compression, and encryption.
  - Table maintenance, to enable recovery of corrupt tables.

- Physical (Raw) Versus Logical Backups
  - Physical backups consist of raw copies of the directories and files that store database contents.
    - This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.
  - Logical backups save information represented as logical database structure (CREATE DATABASE, CREATE TABLE statements) and content (INSERT statements or delimited-text files).
    - This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

- Online Versus Offline Backups
  - Online backups take place while the MySQL server is running so that the database information can be obtained from the server.
  - Offline backups take place while the server is stopped.
  - This distinction can also be described as "hot" versus "cold" backups;
    - a "warm" backup is one where the server remains running but locked against modifying data while you access database files externally.
  - Online backup methods have these characteristics:
    - The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
    - Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.
  - Offline backup methods have these characteristics:
    - Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replica that can be taken offline without harming availability.
    - The backup procedure is simpler because there is no possibility of interference from client activity.
  - A similar distinction between online and offline applies for recovery operations, and similar characteristics apply.
    - However, it is more likely for clients to be affected by online recovery than by online backup because recovery requires stronger locking.
    - During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

- A local backup is performed on the same host where the MySQL server runs,
  - whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server host.
  - **mysqldump** can connect to local or remote servers.
    - For SQL output (CREATE and INSERT statements), local or remote dumps can be done and generate output on the client.
    - For delimited-text output (with the --tab option), data files are created on the server host.
  - SELECT ... INTO OUTFILE can be initiated from a local or remote client host, but the output file is created on the server host.
  - Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

- Snapshot Backups
  - Some file system implementations enable "snapshots" to be taken.
  - These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system.
    - (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.)
  - MySQL itself does not provide the capability for taking file system snapshots.
    - It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

- A full backup includes all data managed by a MySQL server at a given point in time.
- An incremental backup consists of the changes made to the data during a given time span (from one point in time to another).
- MySQL has different ways to perform full backups.
- Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Backup Scheduling, Compression, and Encryption

- Backup scheduling is valuable for automating backup procedures.
- Compression of backup output reduces space requirements, and
- encryption of the output provides better security against unauthorized access of backed-up data.
- MySQL itself does not provide these capabilities.
  - The MySQL Enterprise Backup product can compress InnoDB backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

Making a Hot Backup with MySQL Enterprise Backup

- Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product to do physical backups of entire instances or selected databases, tables, or both.
- This product includes features for incremental and compressed backups.
- Backing up the physical database files makes restore much faster than logical techniques such as the mysqldump command.
- InnoDB tables are copied using a hot backup mechanism.
  - (Ideally, the InnoDB tables should represent a substantial majority of the data.)
- Tables from other storage engines are copied using a warm backup mechanism.

Making Delimited-Text File Backups

- To create a text file containing a table's data, you can use SELECT * INTO OUTFILE '*file_name*' FROM *tbl_name*.
- The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk.
- This method works for any kind of data file, but saves only table data, not the table structure.
- Another way to create text data files (along with files containing CREATE TABLE statements for the backed up tables) is to use **mysqldump** with the --tab option.
- To reload a delimited-text data file, use LOAD DATA or **mysqlimport**.

Making Incremental Backups by Enabling the Binary Log

- MySQL supports incremental backups using the binary log.
- The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup.
- At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using FLUSH LOGS.
- The next time you do a full backup, you should also rotate the binary log using FLUSH LOGS or **mysqldump --flush-logs**.

- Making Backups Using Replicas
  - If you have performance problems with a server while making backups, one strategy that can help is to set up replication and perform backups on the replica rather than on the source.
  - If you are backing up a replica, you should back up its connection metadata repository and applier metadata repository when you back up the replica's databases, regardless of the backup method you choose.
  - This information is always needed to resume replication after you restore the replica's data.
    - If your replica is replicating LOAD DATA statements, you should also back up any SQL_LOAD-* files that exist in the directory that the replica uses for this purpose.
    - The replica needs these files to resume replication of any interrupted LOAD DATA operations.

- Recovering Corrupt Tables
  - If you have to restore MyISAM tables that have become corrupt, try to recover them using REPAIR TABLE or **myisamchk -r** first. That should work in 99.9% of all cases.
- Making Backups Using a File System Snapshot
  - If you are using a Veritas file system, you can make a backup like this:
    - From a client program, execute FLUSH TABLES WITH READ LOCK.
    - From another shell, execute mount vxfs snapshot.
    - From the first client, execute UNLOCK TABLES.
    - Copy files from the snapshot.
    - Unmount the snapshot.
  - Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

Using mysqldump for Backups

- Consider using the MySQL Shell dump utilities, which provide
  - parallel dumping with multiple threads, file compression, and progress information display, as well as cloud features such as Oracle Cloud Infrastructure Object Storage streaming, and MySQL Database Service compatibility checks and modifications.
  - Dumps can be easily imported into a MySQL Server instance or a MySQL Database Service DB System using the MySQL Shell load dump utilities.
- A dump file can be used in several ways:
  - As a backup to enable data recovery in case of data loss.
  - As a source of data for setting up replicas.
  - As a source of data for experimentation:
    - To make a copy of a database that you can use without changing the original data.
    - To test potential upgrade incompatibilities.

- Partitioning takes this notion a step further,
  - by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed.
  - In effect, different portions of a table are stored as separate tables in different locations.
  - The user-selected rule by which the division of data is accomplished is known as a partitioning function, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function.
  - The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression.
  - This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given WHERE clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search.
- In addition, MySQL supports explicit partition selection for queries. For example, SELECT * FROM t PARTITION (p0,p1) WHERE c < 5 selects only those rows in partitions p0 and p1 that match the WHERE condition.
  - In this case, MySQL does not check any other partitions of table t; this can greatly speed up queries when you already know which partition or partitions you wish to examine.
  - Partition selection is also supported for the data modification statements DELETE, INSERT, REPLACE, UPDATE, and LOAD DATA, LOAD XML.

The types of partitioning which are available in MySQL 8.0 are listed here:

- **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range.
- **LIST partitioning.** Similar to partitioning by RANGE, except that the partition is selected based on columns matching one of a set of discrete values.
- **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value.
- **KEY partitioning.** This type of partitioning is similar to partitioning by HASH, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type.

- A very common use of database partitioning is to segregate data by date.

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL, lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL, email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

- Other partitioning types require a partitioning expression that yields an integer value or NULL.

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL, lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL, email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960), PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980), PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Basic concept of MongoDB

REin
REliable, INtelligent & Scalable Systems

- A document is the basic unit of data for MongoDB
- A collection can be thought of as the schema-free equivalent of a table, the documents in the same collection could have different shapes or types.
- Every document has a special key "_id", it is unique across the document's collection
- Mongo DB groups collections into database and each database has its own permission and be stored in separate disks

- Embedded document

  - Embedded document

    embedded documents are entire Mongo DB documents that are used as the values for a key in another document,

```
{
    "name" : "John Doe",
    "address" : {
                "street" : "123 Park Street",
                "city" : "Anytown",
                "state" : "NY"
    }
}
```

- A collection is a group of documents
  - If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.
- Collections are *schema-free*.
  - This means that the documents within a single collection can have any number of different "shapes."
  - {"greeting" : "Hello, world!"}
  - {"foo" : 5}

- In addition to grouping documents by collection, MongoDB groups collections into *databases*.
  - A database has its own permissions, and each database is stored in separate files on disk.
  - A good rule of thumb is to store all data for a single application in the same database.
- There are also several reserved database names, which you can access directly but have special semantics. These are as follows:
  - *admin:* This is the "root" database, in terms of authentication.
  - *local:* This database will never be replicated and can be used to store any collections that should be local to a single server.
  - *config:* When Mongo is being used in a sharded setup, the *config* database is used internally to store information about the shards.

- **The Labeled Property Graph Model**
- A *labeled property graph* is made up of *nodes*, *relationships*, *properties*, and *labels*.
  - Nodes contain properties. Think of nodes as documents that store properties in the form of arbitrary key-value pairs. In Neo4j, the keys are strings and the values are the Java string and primitive data types, plus arrays of these types.
  - Nodes can be tagged with one or more labels. Labels group nodes together, and indicate the roles they play within the dataset.
  - Relationships connect nodes and structure the graph. A relationship always has a direction, a single name, and a *start node* and an *end node*—there are no dangling relationships. Together, a relationship's direction and name add semantic clarity to the structuring of nodes.

What is Data Lake?

- A data lake is a system or repository of data stored in its natural/raw format, usually object blobs or files.
  - A data lake is usually a single store of data including raw copies of source system data, sensor dat social data etc., and transformed data used for tasks such as reporting, visualization, advanced analytics and machine learning.
  - A data lake can include structured data from relational databases (rows and columns), semi-structured data (CSV, logs, XML, JSON), unstructured data (emails, documents, PDFs) and binary data (images, audio, video).
  - A data lake can be established "on premises" (within an organization's data centers) or "in the cloud" (using cloud services from vendors such as Amazon, Microsoft, Oracle Cloud, or Google).

A data lake is a centralized repository designed to store, process, and secure large amounts of structured, semistructured, and unstructured data.

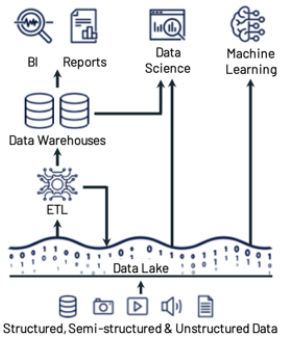
- It can store data in its native format and process any variety of it, ignoring size limits.

	Data Lake	Data Warehouse
Data Structure	Raw	Processed
Purpose of Data	Not yet determined	Currently in use
Users	Data scientists	Business professionals
Accessibility	Highly accessible and quick to update	More complicated and costly to make changes

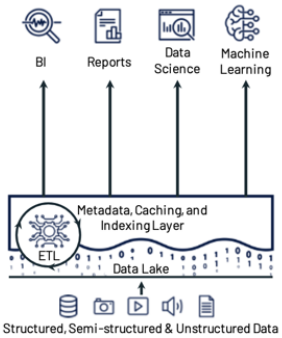
Using HDFS as the core storage and MapReduce as the basic computing model

- Evolution of data platform architectures to today's two-tier model (a-b) and the new Lakehouse model (c)

(a) First-generation platforms.



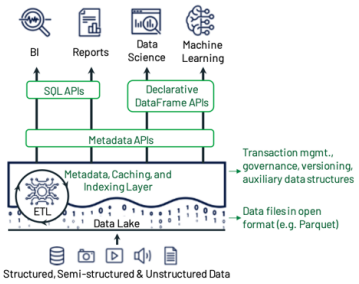
(b) Current two-tier architectures.



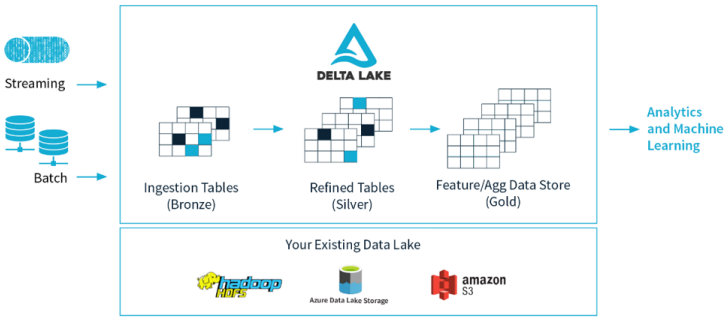
(c) Lakehouse platforms.

Extract, transform, and load (ETL) is the process of combining data from multiple sources into a large, central repository called a data warehouse. ETL uses a set of business rules to clean and organize raw data and prepare it for storage, data analytics, and machine learning (ML).

- The system centers around a meta- data layer such as Delta Lake that adds transactions, versioning, and auxiliary data structures over files in an open for- mat, and can be queried with diverse APIs and engines.



- Delta Lake



Clustering

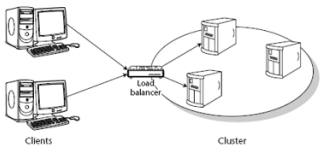
- Clustering addresses many of the issues faced by large-scale systems at the same time.
- A cluster is a loosely coupled group of servers that provide unified services to their clients.
- The client's view of the cluster is a single, simple system, not a group of collaborating servers. This is referred to as a single-system view or single-system image.
- Computers in a cluster are called nodes.

- The main principle behind clustering is that of redundancy. ավելորդություն
  - Reliability
    - Remove single points of failure
  - Availability
    - Overall availability is $1-(1-f\%)^n$
  - Serviceability
    - More complex than a single application server
    - But we could get ability for hot upgrade
  - Scalability
    - It is cheaper to build a cluster using standard hardware than to rely on multiprocessor machines.
    - Extending a cluster by adding extra servers can be done during operation and hence is less disruptive than plugging in another CPU board.

the state of being not or no longer needed or useful.

Load balancing and Failover

- Load balancing means distributing the requests among cluster nodes to optimize the performance of the whole system.
  - The algorithm that the load balancer uses to decide which target node to pick for a request can be systematic or random.
  - Alternatively, the load balancer could try to monitor the load on the different nodes in the cluster and pick node that appears less loaded than others.
- An important feature for Web load balancers is session stickiness, which means that all requests in a client's session are directed to the same server.



The concept of idempotence

- An idempotent method is one that can be called repeatedly with the same arguments and achieves the same results each time.
  - HTTP GET
  - Generally, any methods that alter a persistent store based on its current state are not idempotent, since two invocations of the same method will alter the persistent store twice.

nginx

- nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev.
  - For a long time, it has been running on many heavily loaded Russian sites including
    - Yandex, Mail.Ru, VKontakte, and Rambler.
- According to Netcraft nginx served or proxied 25.68% busiest sites in February 2020.
  - Here are some of the success stories:
    - Dropbox, Netflix, Wordpress.com, FastMail.FM.

- Setting Up a Simple Proxy Server
  - The configuration of a proxy server will look like this:

```
server {
 location / {
  proxy_pass http://localhost:8080/;
 }
 location ~ \.(gif|jpg|png)$ {
  root /data/images;
 }
}
```

  - This server will filter requests ending with .gif, .jpg, or .png and map them to the `/data/images` directory (by adding URI to the root directive's parameter) and pass all other requests to the proxied server configured above.

- Load balancing methods
  - The following load balancing mechanisms (or methods) are supported in nginx:
    - round-robin — requests to the application servers are distributed in a round-robin fashion,
    - least-connected — next request is assigned to the server with the least number of active connections,
    - ip-hash — a hash-function is used to determine what server should be selected for the next request (based on the client's IP address).

- Default load balancing configuration
  - The simplest configuration for load balancing with nginx may look like the following:

```
http {
 upstream myapp1 {
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

  - Reverse proxy implementation in nginx includes load balancing for HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached.

  - If there is the need to tie a client to a particular application server
    - in other words, make the client's session "sticky" or "persistent" in terms of always trying to select a particular server — the ip-hash load balancing mechanism can be used.

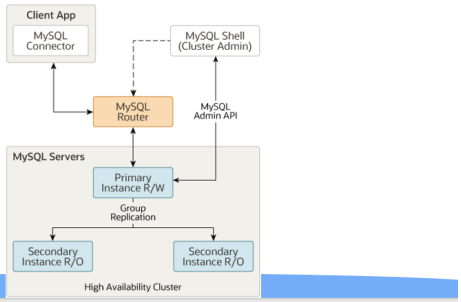
```
http {
 upstream myapp1 {
  ip_hash;
  server srv1.example.com;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
```

Weighted load balancing

```
http {
 upstream myapp1 {
  server srv1.example.com weight=3;
  server srv2.example.com;
  server srv3.example.com;
 }
 server {
  listen 80;
  location / {
   proxy_pass http://myapp1;
  }
 }
}
```

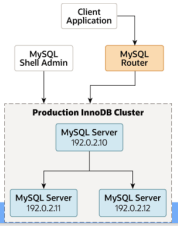
MySQL InnoDB Cluster

- MySQL InnoDB Cluster provides a complete high availability solution for MySQL.



When working in a production environment,

- the MySQL server instances which make up an InnoDB Cluster run on multiple host machines as part of a network rather than on single machine.



What is cloud computing?

- There is little consensus on how to define the Cloud
  - A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.
    - In "*Cloud Computing and Grid Computing 360-Degree Compared*"
  - Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud.
    - In "*Above the Clouds: A Berkeley View of Cloud Computing*"

Cloud in the real world

- Cloud as reality, as told by industry partners

SaaS	Software-as-a-Service	Google Apps, Microsoft "Software+Services"
PaaS	Platform-as-a-Service	IBM IT Factory, Google AppEngine, Force.com
IaaS	Infrastructure-as-a-Service	Amazon EC2, IBM Blue Cloud, Sun Grid
dSaaS	data-Storage-as-a-Service	Nirvanix SDN, Amazon S3, Cleversafe dsNet

amazon web services

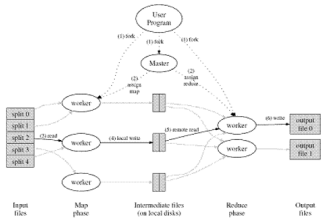
Windows Azure
Microsoft's Cloud Services Platform

salesforce

Google app engine

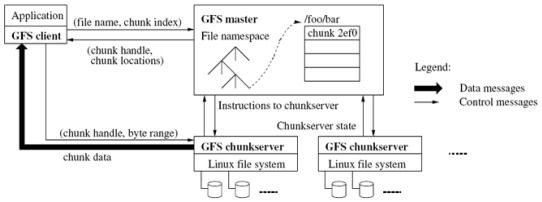
Core techniques of cloud computing

- Google
- MapReduce
  - parallelizes the computation, distributes the data, and handles failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

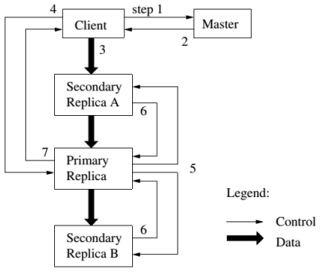


Core techniques of cloud computing

- Google
- Distributed Google File System
  - Google File System(GFS) to meet the rapidly growing demands of Google's data processing needs.



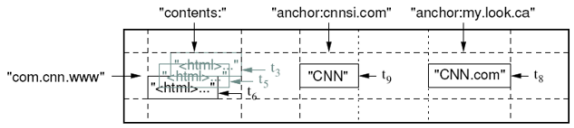
Write Control and Data Flow



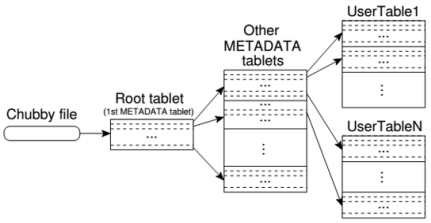
Decoupling Data and Control Flow

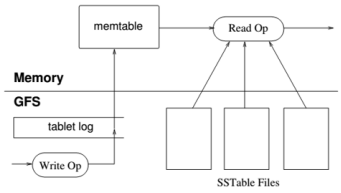
GFS separates the flow of data and control to achieve scalability and fault tolerance. The control flow is managed by the master server, which handles metadata operations and maintains the overall system state. On the other hand, the data flow occurs directly between the clients and the chunk servers, allowing for parallel data access and reducing the load on the master server.

- Google
- Bigtable
  - Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.



- A Bigtable is a sparse, distributed, persistent multidimensional sorted map.
  - The map is indexed by a row key , column key , and a timestamp; each value in the map is an uninterpreted array of bytes.



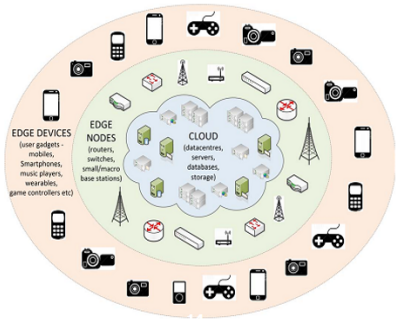


- hadoop

- The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. Hadoop includes these subprojects:
  - **Hadoop Common**: The common utilities that support the other Hadoop subprojects.
  - **HDFS**: A distributed file system that provides high throughput access to application data.
  - **MapReduce**: A software framework for distributed processing of large data sets on compute clusters.

- IBM, Amazon, Yahoo
  - Base stone

- **Edge computing**
  - is a method of optimizing cloud computing systems by performing data processing at the edge of the network, near the source of the data.

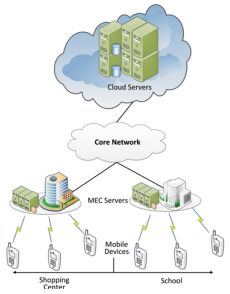


- Cloud offloading
  - In the cloud computing paradigm, most of the computations happen in the cloud, which means data and requests are processed in the centralized cloud.
  - In the edge computing paradigm, not only data but also operations applied on the data should be cached at the edge.
  - Challenges:
    - The data at the edge node should be synchronized with the cloud
    - Another issue involves the collaboration of multiple edges when a user moves from one edge node to another
  - This reduces the communications bandwidth needed between sensors and the central data center by performing analytics and knowledge generation at or near the source of the data.
  - This approach requires leveraging resources that may not be continuously connected to a network such as laptops, smartphones, tablets and sensors

- Video Analytics
  - Cloud computing is no longer suitable for applications that requires video analytics due to the long data transmission latency and privacy concerns.
  - With the edge computing paradigm, the request can be generated from the cloud and pushed to all the things in a target area. Each thing, for example, a smart phone, can perform the request and search its local camera data and only report the result back to the cloud.
  - In this paradigm, it is possible to leverage the data and computing power on every thing and get the result much faster compared with solitary cloud computing.

- Smart Home
  - Smart home would report an impressive amount of data and for the consideration of data transportation pressure and privacy protection, this data should be mostly consumed in the home.
  - This feature makes the cloud computing paradigm unsuitable for a smart home.
  - Edge computing is considered perfect for building a smart home:
    - with an edge gateway running a specialized edge operating system (edgeOS) in the home,
    - the things can be connected and managed easily in the home,
    - the data can be processed locally to release the burdens for Internet bandwidth,
    - and the service can also be deployed on the edgeOS for better management and delivery.

- The Rise of Cloud-Terminal Fusion Computing
  - Cloud Servers
  - Mobile Edge Computing Servers
  - Mobile Devices (Edge Devices)



GraphQL

REIn
REliable, INtelligent & Scalable Systems

- A query language for your API
  - GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.
  - GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.
  - https://graphql.org/
  - GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data.
  - A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type.

```
type Query {
  me: User
}
type User {
  id: ID
  name: String
}
```

```
function Query_me(request) {
  return request.auth.user
}
function User_name(user) {
  return user.getName()
}
```

- Fields
  - At its simplest, GraphQL is about asking for specific fields on objects.

```
{
  hero {
    name
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2"
    }
  }
}
```

Fields
- GraphQL queries can traverse related objects and their fields, letting clients fetch lots of related data in one request, instead of making several roundtrips as one would need in a classic REST architecture.

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

Arguments
- in GraphQL, every field and nested object can get its own set of arguments, making GraphQL a complete replacement for making multiple API fetches.

```
{
  human(id: "1000") {
    name
    height
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 1.72
    }
  }
}
```

```
{
  human(id: "1000") {
    name
    height(unit: FOOT)
  }
}
```

```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```

Aliases
- The two hero fields would have conflicted, but since we can alias them to different names, we can get both results in one request.

```
{
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```

```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```

Fragments
- Fragments let you construct sets of fields, and then include them in queries where you need to.

```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        },
        {
          "name": "C-3PO"
        },
        {
          "name": "R2-D2"
        }
      ]
    },
    "rightComparison": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

Variables

- GraphQL has a first-class way to factor dynamic values out of the query, and pass them as a separate dictionary. These values are called *variables*.

```
query HeroNameAndFriends($episode: Episode) {
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}

VARIABLES

{
  "episode": "JEDI"
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
```

Mutations

- GraphQL is similar - technically any query could be implemented to cause a data write.
- Note how createReview field returns the stars and commentary fields of the newly created review.

```
mutation CreateReviewForEpisode($ep: Episode
  createReview(episode: $ep, review: $revies
    stars
    commentary
  }
}

VARIABLES

{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

Inline Fragments

- If you are querying a field that returns an interface or a union type, you will need to use inline fragments to access data on the underlying concrete type.

```
query HeroForEpisode($ep: Episode!) {
  hero(episode: $ep) {
    name
    ... on Droid {
      primaryFunction
    }
    ... on Human {
      height
    }
  }
}

VARIABLES

{
  "ep": "JEDI"
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "primaryFunction": "Astromech"
    }
  }
}
```

Docker

REin
REliable, INtelligent & Scalable Systems

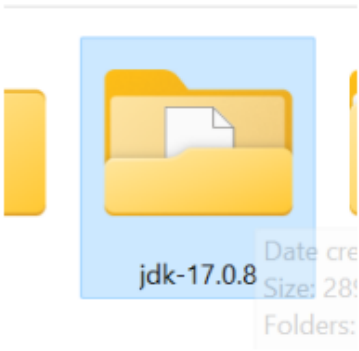
- **What is a Container?** https://www.docker.com
  - A container is a sandboxed process running on a host machine that is isolated from all other processes running on that host machine.
  - That isolation leverages kernel namespaces and cgroups, features that have been in Linux for a long time.
  - Docker makes these capabilities approachable and easy to use. To summarize, a container:
    - Is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.
    - Can be run on local machines, virtual machines, or deployed to the cloud.
    - Is portable (and can be run on any OS).
    - Is isolated from other containers and runs its own software, binaries, configurations, etc.
  - If you're familiar with chroot, then think of a container as an extended version of chroot.
    - The filesystem comes from the image. However, a container adds additional isolation not available when using chroot.

- **What is an Image?** https://www.docker.com
  - A running container uses an isolated filesystem.
  - This isolated filesystem is provided by an image, and the image must contain everything needed to run an application - all dependencies, configurations, scripts, binaries, etc.
  - The image also contains other configurations for the container, such as environment variables, a default command to run, and other metadata.

- Docker is written in the Go programming language and
  - takes advantage of several features of the Linux kernel to deliver its functionality.
- Docker uses a technology called namespaces to provide the isolated workspace called the *container*.
  - When you run a container, Docker creates a set of *namespaces* for that container.
  - These namespaces provide a layer of isolation.
  - Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

JDK > JRE > JVM

JDK(Java Development Kit) is a software package that includes the **JRE a**s well as tools and libraries for developing Java applications (debuggers, **compilers**).



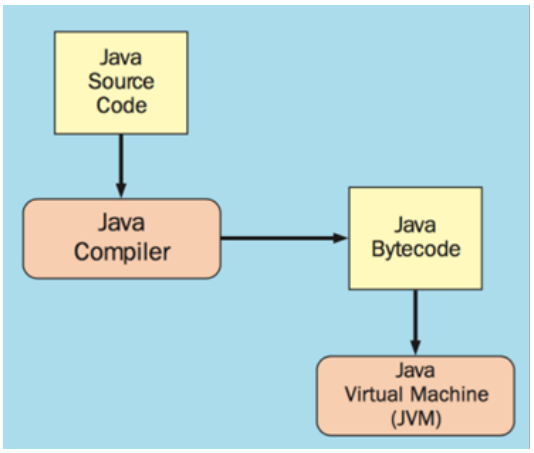
The JRE ("Java Runtime Environment") is a software package that allows you to run Java applications.
It includes a set of libraries, tools, and executables that are required to run Java programs.
For instance Java Virtual Machine (JVM), Class libraries, Java Plugin.

> To draw an analogy to Python, the JRE can be thought of as similar to the Python interpreter or the Python runtime environment, which allows you to run Python code. The JDK can be thought of as similar to a **Python development environment such as Anaconda or PyCharm,** which includes not only the Python interpreter but also tools for developing, testing, and debugging Python code.

JVM stands for Java Virtual Machine. It is an abstract machine that provides the runtime environment for executing Java programs. The JVM is responsible for interpreting and executing Java bytecode, which is generated by the Java compiler from the Java source code. The JVM has several components, including **the class loader, bytecode verifier, interpreter, Just-In-Time (JIT) compiler, and garbage collector.**

The JVM is designed to be platform-independent, which means that Java code can be written once and run on any platform that has a JVM installed, without the need for recompilation. This is because the **JVM provides a layer of abstraction between the Java code and the underlying hardware and operating system.**

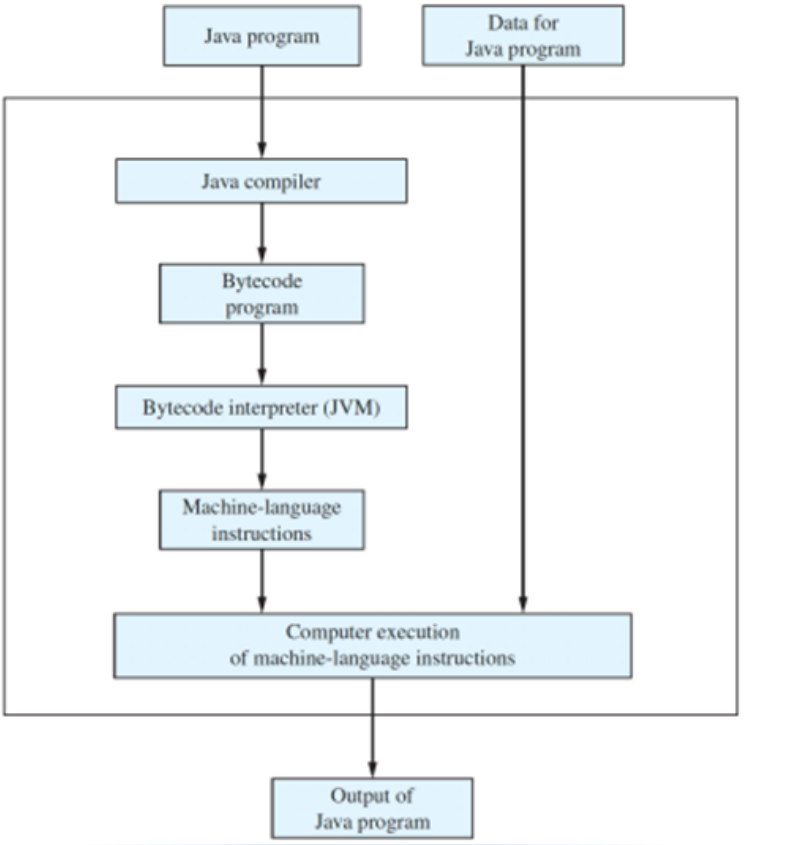
> The JVM is not only used for running Java programs, but also for other languages that can be **compiled into bytecode that the JVM can execute, s**uch as Kotlin, Groovy, and Scala.



- Bytecode
  - The Java compiler translates your Java program into a language called bytecode.
  - This bytecode is not the machine language for any particular computer, but it is similar to the machine language of most common computers.
  - easily translated into the machine language of a given computer.
  - Each type of computer will have its own translator—called an interpreter—that translates from bytecode instructions to machine-language instructions for that computer. (JVM)

Machine code is the lowest-level language that a computer can understand. It consists of a series of binary instructions that are executed directly by the computer's CPU. Machine code is specific to the CPU architecture of the computer, and it is often represented as hexadecimal values.

Bytecode, on the other hand, is an intermediate representation of a program that is used by interpreters and virtual machines. Bytecode is a set of instructions that are designed to be executed



Java SE (Standard Edition) provides the core Java language features and libraries for general-purpose application development. It includes basic APIs for tasks like file handling, data structures, and GUI development.

Java EE (Enterprise Edition) is built on top of Java SE and offers additional features for large-scale, distributed, and networked applications. It includes extended APIs for enterprise-level services like distributed computing, transaction management, and web services.

In summary, Java EE and Jakarta EE offer the same foundational technologies but differ in governance, licensing, and community involvement. Jakarta EE is the future path for the evolution of what was once Java EE.

Bean:

- **What**: A reusable component that encapsulates business logic.
- **Example**: A stateless bean for calculations.

java Copy code

```java
@Stateless
public class CalculatorBean {
    public int add(int a, int b) {
        return a + b;
    }
}
```

Container:

- **What**: Runtime environment that manages the lifecycle, transactions, and security of beans.
- **Example**: EJB container in an application server like GlassFish or JBoss.

In short, a Bean is the code you write, and the Container is what manages that code.

Servlet is a Java class that handles HTTP requests and responses on the server-side. Used in Java EE, now Jakarta EE.

Example:

java Copy code

```java
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello, World!");
  }
}
```

`javax.servlet` is from Java EE (Enterprise Edition).

Jakarta EE is the new name for Java EE, now under the Eclipse Foundation. Packages are transitioning to `jakarta.*`.

`jakarta.servlet` versus `javax.servlet`

> Any chance someone has a simple, factual answer as to why this still doesn't work?

Because:

- You are building against the Tomcat 10.0.10 servlet-api JAR file.
- Tomcat 10.0.x implements version 5.0 of the Servlet spec; see http://tomcat.apache.org/whichversion.html
- Servlet 5.0 is Jakarta EE, not Java EE; see https://en.wikipedia.org/wiki/Jakarta_Servlet#History
- In later versions of Jakarta EE, the package names for the servlet classes have changed from `javax.servlet` to `jakarta.servlet`.

And your code is trying to use the **old** package name.

Solutions:

1. Change your webapp code to import from the **new** `jakarta.servlet` package, OR
2. Roll back to a version of Tomcat that supports the older version of the Servlet spec; i.e. Tomcat 9.0.x or earlier.

You can run servlets without Tomcat using other servlet containers like Jetty, GlassFish, or JBoss. You can also embed a servlet container in a standalone Java application.

What is Tomcat?

Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation. It implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, providing a "pure Java" HTTP web server environment for Java code to run in.

Key Features:

1. **Servlet Container**: Executes Java servlets, which are server-side Java programs that handle client requests and produce responses.
2. **JSP Engine**: Executes JavaServer Pages, allowing you to mix regular HTML with Java code.
3. **Static File Serving**: Can serve static resources like HTML, CSS, and images.
4. **Configuration**: Highly configurable via XML-based configuration files.

Do You Use It in Spring Boot?

In a Spring Boot application, you often use an embedded version of Tomcat as the default web server. When you add the `**spring-boot-starter-web**` dependency to your `**pom.xml**` or `**build.gradle**` file, Spring Boot automatically includes Tomcat as a dependency and configures it as the default embedded server.

Example: `spring-boot-starter-web` Dependency

Here's how you include `**spring-boot-starter-web**` in your `**pom.xml**`:

Rege

Yes, Spring Boot includes an embedded Tomcat by default when you use the `**spring-boot-starter-web**` dependency.

In computer science, separation of concerns is a design principle for separating a computer program into distinct sections such that each section addresses a separate concern.

https://dev.to/tamerlang/separation-of-concerns-the-simple-way-4jp2

Well, because Separation of Concerns involves two processes: reduction of coupling and increasing cohesion.

Cohesion is the measure of how related a group of things is, for example in your kitchen you usually put the knives in the knife block, the spoons with spoons, the forks with forks, you get it. In computer science, it's how strong the relationship is between the methods and data of a class.

These two functions are essentially very cohesive or similar enough to be in the same module/class which is responsible for drawing, and to be fair it feels very natural to put them together. It will look something like this:

```
class Draw {

   public function drawCircle(){
    // draw a circle
   }

   public function drawRectangle(){
    // draw a rectangle
   }
}
```

Coupling is basically a measure of dependence between two or more classes, modules, or components. Tight coupling is bad, and loose coupling is good.

avoid tight coupling

Java Interface-based Programming
Dependency Injection in Spring Boot
Event-based Communication

```
class Logger {
    log(message) {
        console.log(message);
    }
}

class BusinessLogic {
    constructor() {
        this.logger = new Logger();  // Hardcoded dependency
    }

    doSomething() {
        this.logger.log("Doing something");
    }
}
```

The `BusinessLogic` class is tightly coupled with the `Logger` class because it creates an instance of `Logger` within its constructor. If you decide to change how logging is done or switch to a different logging library, you will have to modify the `BusinessLogic` class as well.

```jsx
const element = <h1>Hello, world!</h1>;
```

USX is typically transpiled to JavaScript using a tool like Babel.

JSX produces React “elements”.

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript.

Why JSX?

React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.

Instead of artificially separating *technologies* by putting markup and logic in separate files, React separates *concerns* with loosely coupled units called “components” that contain both. We

> After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

JSX Represents Objects

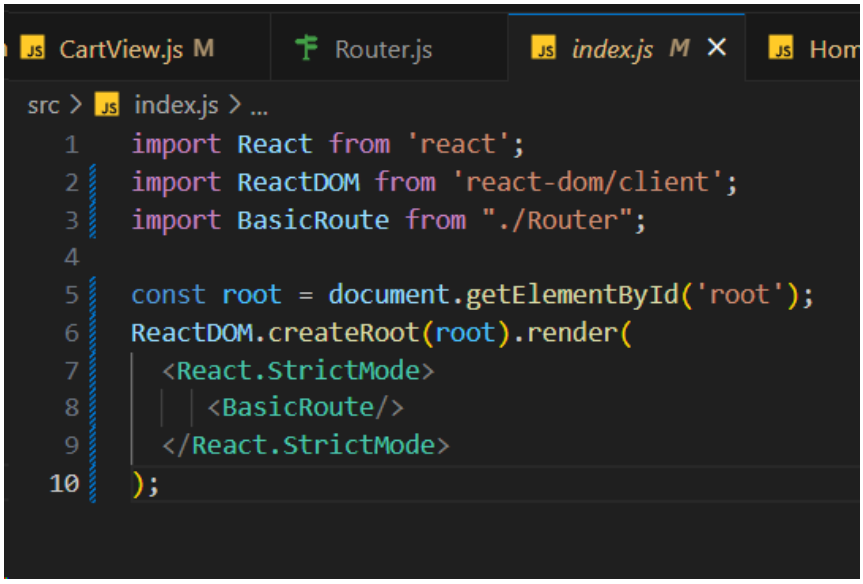
Babel compiles JSX down to `React.createElement()` calls.

These two examples are identical:

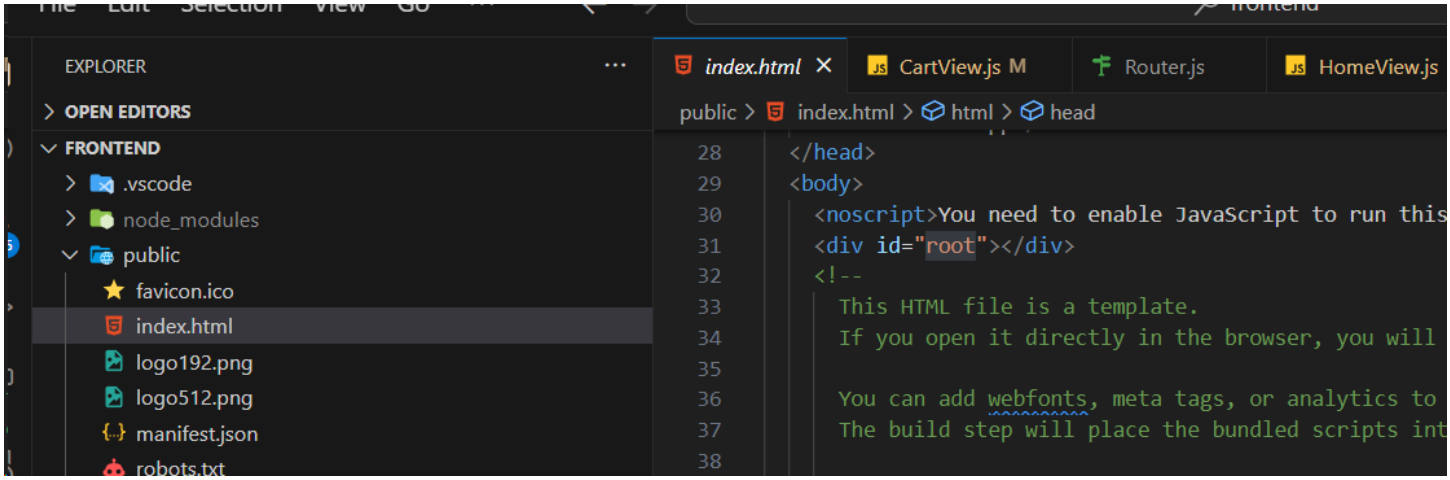
```jsx
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

These objects are called “React elements”. You can think of them as descriptions of what you want to see on the screen. React reads these objects and uses them to construct the DOM and keep it up to date.



The `document` object is a built-in object provided by web browsers that represents the loaded web page. It serves as an entry point to the web page's content, which is often an HTML document. When you use `document.getElementById('root')`, you're asking th browser to find and return the HTML element with the `id` attribute set to `'root'`.



The term "render" in the context of React refers to the process of converting React components into HTML elements that can be displayed in the browser. The rendering process involves the following steps:

React Only Updates What's Necessary

If you don't call a component's `render` method (in the case of class components) or don't return JSX from a functional component, React won't know what to display for that component and you'll likely get an empty output for that part of the UI.

Here's what it would mean in different scenarios:

1. **For Class Components**: If the `render` method isn't provided or doesn't return anything (null or undefined), that component won't produce any output when it's used within another component's JSX.

jsx Copy code

```jsx
class NoRender extends React.Component {
  // No render method
}

// Using NoRender in another component
function App() {
  return (
    <div>
      <NoRender />
      <p>Other Content</p>
    </div>
  );
}
```

is not HTML element without render

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

or directly render inside

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  root.render(element);
}
```

This function is a valid React component because it accepts a single "props" (which stands for properties) object argument with data and returns a React element. We call such components "function components" because they are literally JavaScript functions.

You can also use an ES6 class to define a component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

The above two components are equivalent from React's point of view.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

All React components must act like pure functions with respect to their props. => no prop change

use state

State is similar to props, but it is private and fully controlled by the component.

> if my component is not class i can not use state?

> As of React 16.8, you can use state in functional components as well, thanks to React Hooks. Prior to React 16.8, state was limited to class components. Now, however, functional components are more powerful and can do almost everything that class components can do.

What is a Hook? A Hook is a special function that lets you "hook into" React features. For example, `useState` is a Hook that lets you add React state to function components. We'll l

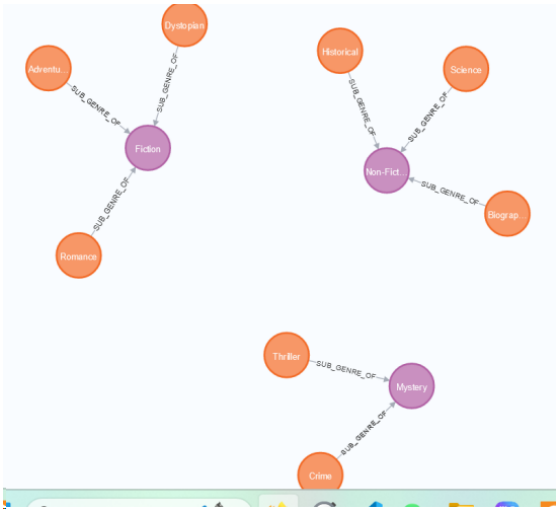
State: a component's memory

gets rendered on the screen. Whenever a state variable is updated, React re-renders the component to reflect the changes. This makes state variables fundamentally different from regular variables defined in a function or a method, as changing those does not cause the component to re-render.

```
 1:  import React, { useState } from 'react';
 2:
 3:  function Example() {
 4:    const [count, setCount] = useState(0);
 5:
 6:    return (
 7:      <div>
 8:        <p>You clicked {count} times</p>
 9:        <button onClick={() => setCount(count + 1)}>
10:         Click me
11:        </button>
12:      </div>
13:    );
14:  }
```

- To run Neo4j as a console application, use:
  <NEO4J_HOME>/bin/neo4j console

Romance




```
mvn clean package -DskipTests
```

```
C:\Users\vahag\mongodb-database-tools-windows-x86_64-100.9.4\bin>mongodump --db bookstore --out D:\edu\SJTU\2-2\web\book
store\backend\mongo-data
2024-01-05T16:08:17.477+0800    writing bookstore.bookCoverImages to D:\edu\SJTU\2-2\web\bookstore\backend\mongo-data\bo
okstore\bookCoverImages.bson
2024-01-05T16:08:17.487+0800    done dumping bookstore.bookCoverImages (12 documents)
```

```
D:\edu\SJTU\2-2\web\bookstore\backend\mongo-data>docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                               NAMES
7606758d9eed   spring-boot-app   "java -Djava.securit…"   40 seconds ago   Up 39 seconds   0.0.0.0:8080->8080/tcp              backend-app-1
22ca22271711   mysql:5.7        "docker-entrypoint.s…"   40 seconds ago   Up 39 seconds   33060/tcp, 0.0.0.0:3307->3306/tcp   backend-db-1
b7691734aa20   mongo:latest     "docker-entrypoint.s…"   40 seconds ago   Up 39 seconds   0.0.0.0:27017->27017/tcp            backend-mongodb-1

D:\edu\SJTU\2-2\web\bookstore\backend\mongo-data>docker cp bookstore backend-mongodb-1:/tmp/dump
Successfully copied 1.57MB to backend-mongodb-1:/tmp/dump
```

```
D:\edu\SJTU\2-2\web\bookstore\backend>docker exec -it backend-mongodb-1 mongorestore --username mongoadmin --password mongopassword /tmp/dump
2024-01-05T08:18:47.367+0000    WARNING: On some systems, a password provided directly using --password may be visible to system status programs such as `ps` that may be invoked by other users. Consider omitti
ng the password to provide it via stdin, or using the --config option to specify a configuration file with the password.
2024-01-05T08:18:47.377+0000    preparing collections to restore from
2024-01-05T08:18:47.377+0000    reading metadata for bookstore.bookCoverImages from /tmp/dump/bookstore/bookCoverImages.metadata.json
2024-01-05T08:18:47.394+0000    restoring bookstore.bookCoverImages from /tmp/dump/bookstore/bookCoverImages.bson
2024-01-05T08:18:47.408+0000    finished restoring bookstore.bookCoverImages (12 documents, 0 failures)
2024-01-05T08:18:47.408+0000    no indexes to restore for collection bookstore.bookCoverImages
2024-01-05T08:18:47.408+0000    12 document(s) restored successfully. 0 document(s) failed to restore.
```

```
vahagn@swiftx:~/wordcount$ ~/hadoop/hadoop-3.3.6/sbin/start-dfs.sh
```

```
hdfs dfs -mkdir /wordcount_input
```

```
vahagn@swiftx:~/wordcount$ hdfs dfs -put ./input/* /wordcount_input
```