

# session-10

---

## Session 10

- session-10
  - 01:00 recap
  - 02:00
  - 01:00 servers creation
  - 09:00 config of all servers
  - 57:00 what is a queue
  - 01:00:00 asynchronous
    - what is rabbitmq and explain realworld usecase scenarios for retail system like wallmart or kroger in supply chain and instore operations like stocking inventory etc
  - RabbitMQ: The Speedy Delivery Rabbit for Your Retail Operations

## 01:00 recap

why Rdbms are heavy applications and why do we need **t3.medium**

## 02:00

mongo db

shipping

my sql these are t3.medium

### **t3.micro 6 servers**

best approach is to setup from right to left to create servers . Because if we have backend then we can enable server reference one time in the front end, rather than creating server and updating server reference everytime

**if manual process is clear then automation is very easy**

## 01:00 servers creation

## 09:00 config of all servers

mongodb configuration

servers configuration is completed 32:00 Quiz and quiz ends at 52:30

shipping module configuration start

## 57:00 what is a queue

rabbitmq synchronous vs asynchronous

Absolutely! Let's delve into the world of synchronous and asynchronous communication and operations.

### **Synchronous Communication**

- **"Wait for me!":** Synchronous communication is like a phone call or live video conference. All participants are present at the same time and interact in real-time. There's a continuous, immediate exchange of information.
- **Orderly and Structured:** Tasks are executed in a strict sequence. One task must finish before the next one can begin. It's like following a recipe step-by-step—you can't mix the ingredients until the oven is preheated.
- **Potential Bottlenecks:** If one participant or process is slow, it can hold up the entire chain of events. Imagine one person taking forever to answer a question in a live meeting, delaying everyone else.

## Asynchronous Communication

- **"I'll get back to you":** Asynchronous communication is like sending an email or a text message. You send your message, and the recipient can respond at their own convenience. There's no expectation of an immediate back-and-forth.
- **Flexibility and Independence:** Tasks can happen independently and even out of order. It's like having multiple tasks on your to-do list—you can pick and choose what to work on and when.
- **No Real-Time Pressure:** Participants can take time to think, research, or gather information before responding. It's less like a live debate and more like a written essay exchange.

## Examples

### Synchronous:

- Live webinars
- Phone calls
- Real-time chat rooms
- In-person meetings
- Code execution in a single-threaded computer program

### Asynchronous:

- Email
- Text messages
- Discussion forums
- Project management tools (e.g., Trello, Asana)
- Pre-recorded videos or lectures

### When to Choose Which:

- **Synchronous:** When you need immediate feedback, collaboration, or resolution. It's great for brainstorming, urgent problem-solving, and building relationships.
- **Asynchronous:** When you need flexibility, time for reflection, participants in different time zones, or documentation of the communication. It's excellent for ongoing discussions and tasks that don't require immediate attention.

## Technical Considerations

In computer programming, synchronous and asynchronous concepts are crucial:

- **Synchronous Operations:** These operations block the program's execution until they're complete. For example, a function that reads data from a file synchronously will halt the program until the entire file is

read.

- **Asynchronous Operations:** These operations run in the background without blocking the main program. The program can continue doing other things while the operation progresses. Using callbacks or promises is a common way to handle the results of asynchronous operations when they are ready.

**Let me know if you'd like to explore a specific area like how these concepts apply to web development, or any other technical aspect!**

Excellent question! Here's how synchronous and asynchronous concepts are fundamental to web servers and scaling:

### Synchronous Limitations for Servers

- **Traditional Web Servers:** Older web servers often used synchronous models, with each client request being handled by a dedicated thread or process.
  - **The Bottleneck:** If one request takes a long time (e.g., a heavy database query), it blocks that thread, preventing it from serving other clients while it waits. This leads to slowdowns and website unresponsiveness under load.

### Asynchronous Power for Scaling

- **Modern Web Servers (Node.js, Nginx, etc.)** These are largely built on asynchronous models using techniques like event loops and non-blocking I/O:
  - **Multi-tasking Wizards:** A single thread can handle a large number of client requests concurrently. When one request is waiting (e.g., for database results), the server isn't idle; it moves on to serve other requests.
  - **Resource Efficient:** This leads to way better use of server resources, allowing a single server to handle a much higher volume of traffic.

### Scaling with Asynchronous Design

- **Horizontal Scaling:** Adding more servers to a cluster becomes very effective with asynchronous architecture. Each server can independently handle a huge number of requests, leading to more linear increases in capacity as you add servers.
- **Microservices:** Asynchronous communication is often the backbone of microservice architectures, where individual services can run independently and communicate via asynchronous messaging. This allows individual services to be scaled or updated without affecting the entire system.

### Examples in Action

- **Node.js:** A server environment built from the ground up for asynchronous patterns. It excels at handling many simultaneous connections with low overhead.
- **Reverse Proxies** Like Nginx, they often act as load balancers, asynchronously distributing requests across a pool of web servers.
- **WebSockets:** They provide bi-directional, real-time communication channels, but their underlying implementation is asynchronous.

### Caveat: Not Everything is Asynchronous

Even in modern setups, there will be synchronous components:

- **Heavy Database Operations:** These may sometimes still operate synchronously, especially if they use older database drivers.
- **CPU-bound Computations:** Intensive calculations might not easily fit into asynchronous models.

### Key Takeaways

- Asynchronous models are the reason modern web applications can handle massive traffic that would bring old-style servers to their knees.
- Understanding the balance of synchronous and asynchronous operations is essential when designing scalable web systems.

### Do you want to dive into a particular use case scenario or explore how asynchronous scaling works in real-world architectures?

How this works in real world architecture

Absolutely! Let's look at a practical use case to illustrate how asynchronous scaling enhances real-world web applications.

#### Scenario: A High-Traffic E-commerce Website

Imagine a popular online store. It needs to handle:

- Thousands of product page requests per second.
- User logins and profile updates.
- Shopping cart operations.
- Complex order processing, with inventory checks, payment gateway communication, etc.

#### Synchronous Bottleneck Nightmare

A synchronously designed system would struggle here:

- Slow database queries might stall product page rendering.
- Order processing could create huge backlogs, making the entire site feel sluggish.
- Scaling by adding more servers would be inefficient due to each server's limitations in handling concurrent requests.

#### Asynchronous Scaling to the Rescue

1. **Asynchronous Web Server:** A core server like Node.js or one using event-driven frameworks (e.g., Python's Tornado) avoids blocking on individual requests. This ensures swift responses to basic product information requests even if other operations are underway.
2. **Message Queues:** Heavy tasks like order processing are offloaded to asynchronous message queues (like RabbitMQ, Kafka, or Amazon SQS).
  - **Decoupling:** The web server doesn't wait for the entire order to complete. It sends a message to the queue and moves on to serve other requests.
  - **Worker Processes:** Specialized workers pick up tasks from the queue. They might handle payment interactions, update inventory, and communicate with external systems—all running independently in the background.

3. **NoSQL Databases:** For fast reads and writes of large amounts of non-relational data (like user profiles and product catalogs), a NoSQL database (e.g., MongoDB, Cassandra) can be a scalable, asynchronous fit.
4. **Caching:** Using solutions like Redis or Memcached to cache frequently accessed data reduces the load on both web servers and the database.

## Result

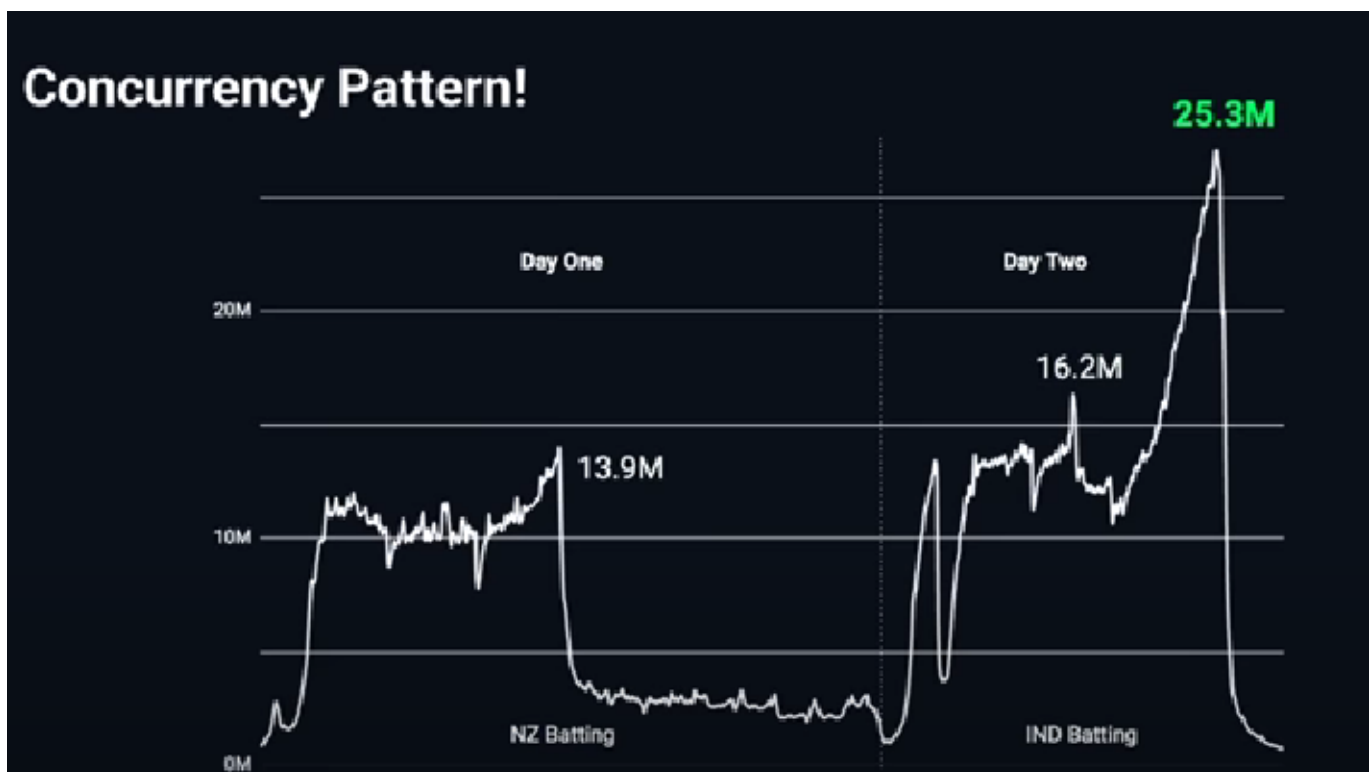
- **Enhanced User Experience:** The website remains highly responsive under high load. Visitors can browse and add items to their carts seamlessly.
- **Distributed Scalability:** You can easily add more web servers or worker nodes as needed to handle increased traffic. Each instance contributes effectively to the overall capacity.
- **Resilience:** Slowdowns in one part of the system (e.g., a temporary payment gateway issue) don't bring down the entire website.

## Important Notes

- **Complexity:** Asynchronous systems are often more complex to design and manage compared to purely synchronous setups.
- **Monitoring:** Good monitoring tools are crucial to understand the flow of events and identify bottlenecks in an asynchronous architecture.

**Let me know if you'd like to explore another specific scenario or ask about a particular component in this asynchronous architecture!**

How hotstar might have scaled



Excellent question! Let's break down how synchronous and asynchronous architectures handle sudden traffic spikes, using examples from streaming giants like Hotstar, Netflix, and Jio.

## Synchronous Architecture: The Struggle

- **Hitting the Ceiling:** Systems with primarily synchronous components will have hard limits on how much concurrent traffic they can handle. Imagine each visitor's stream using a dedicated server thread – there are only so many threads!
- **The Domino Effect:** When a spike exceeds capacity:
  - Requests get queued, increasing response times dramatically.
  - Users might see errors or endlessly loading screens.
  - Cascading failures can occur when dependent systems get overwhelmed by delayed responses.

## Asynchronous Architecture: Built to Flex

- **Buffering with Queues:** When a spike hits, asynchronous systems can absorb it by placing incoming requests into queues. This prevents immediate overload on the core streaming servers.
- **Elastic Scaling:** They are primed for rapid scaling. Here's how streaming platforms likely do it:
  - **Monitoring is Key:** Traffic patterns are closely watched, and automated systems can spin up additional servers or containers based on predefined thresholds.
  - **Pre-Warmed Instances:** They often keep some "reserve" capacity ready to go, rather than starting from scratch every time a spike hits.
- **Resilience by Design:** Since services are often decoupled (communicating asynchronously), a slowdown in processing payments (for example) doesn't mean the streaming itself stops for everyone.

## Real-World Considerations (Inspired by Hotstar, Netflix, Jio)

- **Content Delivery Networks (CDNs):** Edge servers geographically distributed to cache video content close to users. This dramatically reduces load on the central streaming infrastructure.
- **Adaptive Bitrate:** The video quality can be dynamically adjusted per user based on their connection strength. This prevents bottlenecks during periods of congestion.
- **Staggered Releases:** Big-ticket events might be released at slightly different times in various regions to spread out the peak traffic.
- **Partnerships:** Cloud providers play a huge role, allowing streaming platforms to quickly scale their infrastructure on-demand.

## Caveats

- **Cost:** Asynchronous, highly scalable systems are more complex to build and can be more expensive to operate.
- **Not Everything is Equal:** Live streaming presents different challenges than on-demand content, often requiring specialized real-time streaming technologies.

## Key Takeaways

- Synchronous architectures hit a hard wall when facing sudden traffic spikes.
- Asynchronous architectures with queuing, elastic scaling, and intelligent design choices are essential for platforms like Hotstar, Netflix, and Jio to handle the unpredictable viewership surges.

**Would you like to explore the role of specific technologies like load balancers, containerization (like Kubernetes), or real-time streaming protocols in this context?**

01:00:00 asynchronous

client --> messaging broker --> server

sivakumar --> ramesh

there will be a queue for ramesh

sivakumar --> ramesh queue

ramesh whatsapp --> subscribed to ramesh queue

if online --> message will be delivered

when ramesh gets online --> ramesh whatsapp --> ramesh queue

Here's a breakdown of asynchronous communication, tailored to your scenario:

### What is Asynchronous Communication?

- **No immediate response required:** In asynchronous communication, the sender and receiver don't need to be actively engaged in the conversation at the exact same moment. This allows for flexibility and time for thoughtful responses.
- **Example:** Email is a classic example. You write the email, send it, and the recipient can read and respond on their own schedule.

### How it Works in Your Scenario

1. **Client (Sivakumar) sends a message:** Sivakumar wants to send a message to Ramesh. He interacts with the "client" application (potentially WhatsApp or a similar platform).
2. **Messaging Broker:** The message isn't sent directly to Ramesh. Instead, it's routed through a messaging broker. This broker acts as a central hub responsible for handling, storing, and routing messages.
3. **Ramesh's Queue:** The messaging broker places Sivakumar's message in a queue specifically designated for Ramesh. This queue holds messages until Ramesh is ready to receive them.
4. **Subscription (Ramesh's WhatsApp):** Ramesh's WhatsApp application has "subscribed" to his queue on the messaging broker. This means it's set up to monitor the queue and retrieve new messages.
5. **Two Scenarios:**
  - **Ramesh is Online:** If Ramesh has WhatsApp open and is actively online, the message is delivered immediately from the queue to his WhatsApp.
  - **Ramesh is Offline:** If Ramesh isn't currently online, the message remains in the queue. As soon as Ramesh's WhatsApp reconnects (when he gets online), the message is delivered from the queue to his WhatsApp.

### Key Advantages of Asynchronous Communication in this Setup

- **Reliability:** Even if Ramesh isn't online, the messaging broker ensures the safekeeping of the message until it can be delivered successfully.
- **Decoupling:** Sivakumar doesn't need to know whether Ramesh is online or not to send a message. This separation allows for flexibility and avoids disruptions to the sender.
- **Scalability:** The messaging broker can handle multiple senders and recipients efficiently, ensuring that messages destined for different users are correctly routed and stored in their respective queues.

**Let me know if you'd like more detail on messaging brokers, the concept of queues, or have any other questions!**

what is rabbitmq and explain realworld usecase scenarios for retail system like walmart or kroger in supply chain and instore operations like stocking inventory etc

01:05:00 rabbit mq installation

## RabbitMQ: The Speedy Delivery Rabbit for Your Retail Operations

RabbitMQ is an open-source message broker that excels at routing messages between applications. It acts as a central hub, ensuring messages get delivered reliably, efficiently, and in the right order, even if the receiving application is unavailable for a short time.

Here's how RabbitMQ can be a game-changer for retail giants like Walmart and Kroger in their supply chain and in-store operations:

### Supply Chain Management:

- **Order Routing:** Orders from various channels (online, mobile app, in-store) can be placed into queues. RabbitMQ routes them to the appropriate fulfillment center or warehouse for processing, ensuring efficient order fulfillment.
- **Inventory Updates:** As items are shipped from warehouses, RabbitMQ can transmit real-time inventory updates to a central database. This improves stock visibility across the entire supply chain and prevents stockouts.
- **Delivery Notifications:** Messages can be sent to trigger notifications for successful deliveries, allowing for faster customer service and order tracking updates.

### In-Store Operations:

- **Low Stock Alerts:** Inventory sensors on shelves can send messages to RabbitMQ when stock levels dip below a certain threshold. This triggers alerts for store staff to restock items, preventing empty shelves and lost sales.
- **Price Change Management:** Price updates can be sent to RabbitMQ and then distributed to electronic shelf labels throughout the store, ensuring all prices are accurate and reflect promotions.
- **Cashier Communication:** Scanned items at checkout could trigger messages to RabbitMQ to verify product information, pricing, and promotions, streamlining the checkout process.

### Real-World Benefits

- **Improved Efficiency:** Streamlined communication between various systems reduces manual tasks and errors.
- **Enhanced Visibility:** Real-time data flow across the supply chain and in-store operations provides better control and decision-making.
- **Scalability:** RabbitMQ can handle massive volumes of messages, making it suitable for large retail operations.
- **Resilience:** The asynchronous nature of message queuing ensures deliveries even if individual systems are temporarily unavailable.

### Beyond Walmart and Kroger



RabbitMQ isn't limited to retail giants. Any business with complex workflows and communication needs can benefit from its functionalities. Here are some additional use cases:

- **Manufacturing:** Routing production orders, managing quality control checks, and coordinating deliveries with suppliers.
- **Healthcare:** Sending appointment reminders, lab test results, and prescription updates securely.
- **Financial Services:** Processing transactions, updating account balances, and triggering fraud alerts.

**In essence, RabbitMQ acts as the reliable backbone for communication between various parts of your system, ensuring a smooth flow of information and a more responsive and efficient operation.**

01:07:00 payment application

**configuration of payment application is completed 01:12:00 01:12:30**

dispatch is a different application you can try installing it **01:13:00**

end of class