**Official Documentation:**

* AWS API Gateway WebSocket documentation
* Next.js documentation on WebSocket integration
* Nest.js WebSocket gateway documentation

**Architecture Comparison Articles to Research:**

* "WebSocket vs Socket.io vs Native Implementation" comparisons
* "Scaling WebSocket Connections in AWS" guides
* "WebSocket Connection Management Patterns" articles

**Key Evaluation Criteria to Research:**

1. **Scalability Patterns**
   * How each approach handles connection pooling
   * Load balancing WebSocket connections
   * Horizontal scaling considerations
2. **Error Handling & Reconnection**
   * Automatic reconnection strategies
   * Connection state management
   * Fallback mechanisms
3. **AWS Integration Specifics**
   * API Gateway WebSocket costs
   * Lambda WebSocket limitations
   * ALB vs API Gateway for WebSocket
4. **Framework Compatibility**
   * Next.js + WebSocket patterns
   * Nest.js WebSocket decorators & gateways
   * State management with WebSocket

**Research Questions by Approach**

**1. Native WebSocket Implementation**

* How do you handle connection state in a serverless environment?
* What's the reconnection strategy for dropped connections?
* How do you implement heartbeat/keep-alive?
* Can you use AWS ALB for WebSocket load balancing?
* How do you broadcast to multiple connections?

**2. Socket.io**

* Does Socket.io adapter work with AWS infrastructure?
* How does Socket.io handle serverless cold starts?
* What's the overhead of Socket.io protocol vs native WS?
* Can you use Redis adapter for scaling across multiple instances?
* How well does it integrate with Nest.js WebSocket gateway?

**3. AWS API Gateway WebSocket**

* What are the connection duration limits?
* How do you manage connection IDs?
* What's the pricing model for concurrent connections?
* How do you implement broadcasting to multiple clients?
* Can you integrate with Lambda for message processing?
* How do you handle connection state persistence?

**Architecture-Specific Questions**

**For Your Tech Stack:**

1. **Next.js Frontend**
   * Where do WebSocket connections originate? (API routes vs client-side)
   * How do you manage connection state in React components?
   * What's the reconnection UX pattern?
2. **Nest.js Backend**
   * Should you use Nest's @WebSocketGateway()?
   * How do you integrate WS with your existing REST architecture?
   * Where does WS gateway fit in your modular structure?
3. **AWS Lambda**
   * Can you maintain WebSocket connections in Lambda?
   * Do you need a separate always-on service for WS?
   * How do you handle Lambda timeout limits?

**For Your Use Case:**

1. **Task Status Updates**
   * How many concurrent users/connections expected?
   * What's the message frequency per connection?
   * Do you need bi-directional communication?
2. **Notification Patterns**
   * Should missed updates be queued?
   * How do you handle offline clients?
   * Do you need delivery confirmation?

**WebSocket Architecture Decision Matrix (Revised)**

**Evaluation Criteria & Weight**

| **Criteria** | **Weight** | **Description** | **What "Good" Means** |
| --- | --- | --- | --- |
| Scalability | High | Ability to handle growing connections | Can scale to thousands of connections |
| AWS Integration | High | Native AWS service compatibility | Works seamlessly with AWS services |
| Ease of Implementation | Medium | Development simplicity | Quick to implement, easy to understand |
| Cost Efficiency | Medium | Infrastructure and operational costs | Lower costs for expected usage |
| Framework Compatibility | High | Next.js/Nest.js integration | Native support, good documentation |
| Reliability | High | Connection stability & recovery | Auto-reconnection, handles failures well |
| Time to Market | Medium | Speed to implement MVP | Can build and deploy quickly |

**Comparison Matrix**

| **Solution** | **Scalability** | **AWS Integration** | **Ease of Implementation** | **Cost Efficiency** | **Framework Compatibility** | **Reliability** | **Time to Market** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Native WebSocket** | Poor | Poor | Easy | Excellent | Fair | Poor | Fair |
| **Socket.io** | Good | Poor | Very Easy | Good | Excellent | Excellent | Excellent |
| **AWS API Gateway WS** | Excellent | Excellent | Moderate | Fair | Fair | Good | Fair |

**Rating Scale:**

* **Excellent**: Best option for this criterion
* **Good**: Strong performance, minor limitations
* **Fair**: Adequate, some trade-offs
* **Poor**: Significant limitations
* **Very Easy/Easy/Moderate**: For implementation complexity specifically

**Detailed Reasoning:**

**Native WebSocket**

* **Scalability (Poor)**: Hard to scale without additional infrastructure
* **AWS Integration (Poor)**: Requires custom implementation for AWS services
* **Ease of Implementation (Easy)**: Simple API, but lacks features
* **Reliability (Poor)**: No built-in reconnection or fallbacks

**Socket.io**

* **Scalability (Good)**: Built-in adapters for scaling (Redis)
* **Framework Compatibility (Excellent)**: Great Nest.js support
* **Reliability (Excellent)**: Auto-reconnection, transport fallbacks
* **Time to Market (Excellent)**: Many features out-of-the-box

**AWS API Gateway WebSocket**

* **Scalability (Excellent)**: Managed by AWS, auto-scales
* **AWS Integration (Excellent)**: Native service, works with Lambda
* **Ease of Implementation (Moderate)**: Requires understanding AWS patterns
* **Cost Efficiency (Fair)**: Can be expensive at scale