🔹 Day 78 – AWS + Web3 (Part 3 – Building a Smart Contract API with AWS Lambda)

Section 1: Setup & Architecture

Q1. What are we building today?
A serverless API that reads/writes data to an Ethereum smart contract using AWS Lambda.
Tip: No servers, just Lambda + Web3.js + API Gateway.

Q2. Why use AWS Lambda for Web3?
It executes smart contract functions on demand with minimal cost.
Tip: Perfect for event-driven DApps.

Q3. What's the role of API Gateway here?
Acts as a public entry point for DApp frontend → Lambda backend.
Tip: Add rate limiting for protection.

Q4. What languages can we use?
Node.js (with Web3.js or Ethers.js) works best for blockchain integration.
Tip: Python also works using web3.py.

Q5. What environment variables do we need?
Store PRIVATE_KEY, INFURA_URL, and CONTRACT_ADDRESS securely in Lambda.
Tip: Never hardcode private keys.

Section 2: Secure Key Handling

Q6. How to store private keys safely?
Use AWS KMS or Secrets Manager to encrypt and retrieve keys.
Tip: Never log sensitive data.

Q7. What's the purpose of AWS IAM here?
IAM controls who can trigger the Lambda and access resources.
Tip: Assign least privilege roles only.

Q8. How to sign transactions in Lambda?
Use Ethers.js wallet.signTransaction() or Web3.js sendTransaction().
Tip: Use async calls for better performance.

Section 3: DApp Integration

Q9. How does the frontend call this API?
Using Axios or Fetch to hit API Gateway endpoints.
Tip: Add authentication headers via Cognito or JWTs.

Q10. How to fetch data from blockchain?
Use contract methods like contract.methods.balanceOf(wallet).call().
Tip: For static data, cache results with ElastiCache.

Q11. How to handle large blockchain queries?
Offload them to Step Functions or SQS for async processing.
Tip: Keep Lambda light and fast.

Section 4: Database & Storage

Q12. Why use DynamoDB here?
To store user wallets, transaction logs, or metadata.
Tip: Store TX hash, not private keys.

Q13. What about NFT metadata?
Use S3 for images, DynamoDB for JSON, and link them in your contract.
Tip: Always backup on IPFS.

Q14. Can we use RDS for Web3 apps?
Yes, for transactional off-chain data or DeFi records.

Tip: Use RDS + Lambda for financial accuracy.

Section 5: Event Handling

Q15. How can smart contract events trigger AWS?
Set up a listener → push events to EventBridge → invoke Lambda.
Tip: Ideal for real-time NFT updates or wallet alerts.

Q16. How to automate contract monitoring?
Use CloudWatch logs + Lambda scheduled triggers (CRON jobs).
Tip: Great for DeFi oracle data updates.

Section 6: Scaling & Optimization

Q17. How to scale smart contract APIs?
Use API Gateway + Lambda concurrency + caching layer.
Tip: Scale reads easily, writes carefully.

Q18. How to reduce latency?
Use CloudFront + regional Lambda@Edge functions.
Tip: Always choose the region nearest your users.

Q19. How to handle blockchain node overload?
Use Infura or Alchemy for high-performance RPCs.
Tip: Never rely on one node provider.

Section 7: Monitoring & Debugging

Q20. How do you monitor Lambda performance?
Use CloudWatch and X-Ray for latency and invocation errors.
Tip: Add custom metrics for failed TXs.

Q21. How to debug failed contract calls?
Log receipt.status and error.message from Web3.js.

Tip: Always wrap calls in try-catch.

Section 8: Security & Best Practices

Q22. What’s the top security rule?
Never expose private keys in frontend or public APIs.
Tip: Use backend-only signing logic.

Q23. How to prevent DDoS attacks?
Enable WAF + Shield on API Gateway.
Tip: Add request throttling (max 10 req/sec).

Q24. Can we integrate Cognito with Web3 wallets?
Yes — hybrid login: wallet for signing, Cognito for user sessions.
Tip: Great UX + secure authentication combo.

Section 9: Deployment & CI/CD

Q25. How to deploy the DApp backend?
Use AWS SAM or Serverless Framework.
Tip: Automate Lambda packaging via GitHub Actions.

Q26. What if I want to test locally?
Use AWS SAM CLI or LocalStack with Ganache testnet.
Tip: Keep test RPC URLs in .env file.

Q27. How to integrate with Polygon or BSC?
Change the RPC URL — architecture stays the same.
Tip: Multi-chain ready design = future-proof.

Section 10: Future & Wrap-Up

Q28. What’s next after this?
Integrate AWS Amplify frontend with your Lambda API.

Tip: You get full-stack Web3 power.

Q29. Real-world use case?
NFT minting site, DAO dashboard, or on-chain voting system.

Q30. Key takeaway?
AWS + Web3 = scalability + decentralization → production-ready DApps.
Tip: Learn to connect both worlds — that’s where innovation lives.