Perfect 👍 Let's move on.

---

# 19. Serverless with Node.js

Serverless computing means you run code **without managing servers**. The cloud provider handles scaling, infrastructure, and availability. You only pay for execution time.

Common providers: **AWS Lambda**, **Google Cloud Functions (GCF)**, **Azure Functions**.

---

## 1. Why Serverless?

☑ Auto-scaling (no manual infra mgmt).
☑ Pay-per-use (no idle costs).
☑ Quick deployment & faster prototyping.
❌ Cold start latency.
❌ Limited execution time (e.g., AWS Lambda max ~15 mins).
❌ Debugging is harder than traditional servers.

---

## 2. Node.js in AWS Lambda

📌 Lambda executes a handler function.

```
exports.handler = async (event) => {
  console.log("Event:", event);

  return {
    statusCode: 200,
    body: JSON.stringify({ message: "Hello from Lambda!" }),
  };
};
```

- Deploy via AWS Console or CLI.
- Trigger with API Gateway, S3 events, DynamoDB streams, etc.

---

## 3. Node.js in Google Cloud Functions

```
exports.helloWorld = (req, res) => {
  res.send("Hello from Google Cloud Functions!");
};
```

- Deploy with `gcloud functions deploy helloWorld --runtime nodejs20 --trigger-http`.

## 4. Node.js in Azure Functions

```
module.exports = async function (context, req) {
  context.log("HTTP trigger function processed a request.");
  const name = req.query.name || (req.body && req.body.name);
  context.res = {
    body: "Hello " + (name || "World"),
  };
};
```

- Deploy via Azure CLI or VS Code Azure Extension.

---

## 5. Serverless Frameworks

Instead of manually deploying, use frameworks:

- **Serverless Framework** → Multi-cloud deployment (AWS, GCP, Azure).
- **SAM (Serverless Application Model)** → AWS-specific.
- **Architect, Claudia.js** → Simplify Lambda functions.

📌 Example (Serverless Framework `serverless.yml`):

```
service: hello-service

provider:
  name: aws
  runtime: nodejs20.x
  region: us-east-1

functions:
  hello:
    handler: handler.hello
    events:
      - http:
          path: hello
          method: get
```

---

## 6. Use Cases

- REST APIs (with API Gateway).
- Scheduled Jobs (cron-like).
- Event-driven tasks (file upload → process → store).
- Real-time notification services.

---

## 7. Interview Tips

- **Q:** Difference between serverless & containers?
  **A:** Serverless → no infra management, auto-scaled, short-lived tasks. Containers → full control, long-running services.
- **Q:** Cold start in Lambda?
  **A:** Delay when spinning up a new container for first request. Mitigate via provisioned concurrency.
- **Q:** When NOT to use serverless?
  **A:** Long-running tasks, heavy compute, apps requiring persistent connections (like WebSockets without API Gateway).

---

☑ Next is the **final one** → **20. Interview-Focused Questions**.
Want me to cover it like a **Q&A sheet** (straightforward for revision) or like **expanded explanations** (deep dive)?