**Lektion 7.**

Först skapade jag bara enkel Node.js Express applikation för att kunna utföra uppgifterna i lektionen.

Efter det skapade jag en Dockerfile.

* **Minimal basimage**: node:18-alpine för att minska attackytan.
* **Icke-root användare**: Skapade en användare appuser för att undvika att köra som root.

För att optimera bygget och minska säkerhetsrisker exkluderade jag onödiga filer genom att skapa en .dockerignore-fil med innehållet: node\_modules och npm-debug.log

Nu kunde jag bygga Docker-imagen: docker build -t dockerapp:1.0 .

Sedan skulle jag köra containern. Men där stötte jag på problem med upptagen port. Lösningen visade sig vara att port 3000 i ställetagen så i stället använde jag port 4000 lokalt.  
docker run -p 4000:3000 dockerapp:1.0

**Köra Trivy:**

För att identifiera sårbarheter installerade jag Trivy och scannade min Docker-imge.

* Resultatet var:  
  **0 kritiska sårbarheter**
* **1 hög sårbarhet** hittades i cross-spawn:
  + **CVE-2024-21538** – Regular Expression Denial of Service (ReDoS)
  + Fixas genom att uppdatera cross-spawn till version 7.0.5 eller 6.0.6.

En bild som visar text, skärmbild, Teckensnitt, linje

AI-genererat innehåll kan vara felaktigt.

**Fix:**

Lösningen för denna sårbarhet var att installera en nyare version av cross-spawn.

npm install cross-spawn@7.0.5 –save

Efter det byggde jag om Docker-imagen med: docker build -t dockerapp:1.1 .

**Säker drift & molnimplementation:**  
Genom att skapa container görs applikationen mer säker eftersom den körs i en egen avgränsad miljö. Det i sig minskar risken att påverkas av andra program på servern. Dessutom gör det att applikationen alltid fungerar likadant, oavsett var den körs. T.ex i en molnlösning där man använder AWS ECS/EKS eller Azure Kubernetes Service (AKS). Det tillsammans med automatiserad säkerhetsscanning och principen om minsta möjliga behörighet för att skydda systemet görs den säkrare.