**SBOM = Software Bill of Materials**

**An SBOM is a list of all software components, dependencies, and libraries that make up your application — including their versions, licenses, and sources.**

**🧁 Real-World Analogy: The Cake Example**

If you're baking a cake, the **recipe** might look like:

* 2 cups flour
* 1 cup sugar
* 2 eggs
* 1 tsp vanilla

This recipe is your **bill of materials** (BOM) for the cake.

Now apply that to software:

If your app uses:

* Python 3.11
* Flask 2.3.2
* NumPy 1.24.2
* A logging library from GitHub

Then your **SBOM is a software recipe** showing exactly these ingredients.

**💡 Why is SBOM Important?**

Imagine:

* A vulnerability (e.g., **log4j**) is discovered.
* You don’t know which of your apps use it.
* You can’t even be sure if you’re affected.

With an **SBOM**, you can:

✅ Search all your SBOMs for log4j  
✅ Know **where it is used**  
✅ Patch **only affected apps**

Without SBOM, you're **blind**.

**As a DevOps/Security Engineer — What’s Your Role in SBOM?**

Your responsibility isn’t just to **generate the SBOM**.

**You need to:**

**✅ 1. Generate the SBOM**

For every application (Java, Python, .NET, Node.js):

trivy fs . --format cyclonedx --output sbom.json

**✅ 2. Store the SBOM**

Store SBOMs in:

* Git repo (/sboms/)
* S3 bucket
* Artifactory/Nexus
* Inside the container image (via OCI annotations)
* Signed and versioned if needed

**✅ 3. Scan the SBOM for Vulnerabilities**

Once SBOM is created, run:

trivy sbom sbom.json

This tells you:

* Which components have **known vulnerabilities (CVEs)**
* Their severity (HIGH, CRITICAL)
* What you need to fix

**✅ 4. Act on the Findings**

| **CVE Found?** | **What You Do** |
| --- | --- |
| Yes | Alert Dev Team, Update Dependency, Fix it |
| No | You're secure, move on |

**✅ 5. (Optional but Powerful) Sign the SBOM**

Use tools like **Cosign** or **Sigstore** to cryptographically sign your SBOM.

This proves:

* It hasn’t been tampered with
* It matches the software you built

cosign attest --predicate sbom.json --type cyclonedx <your-docker-image>

**✅ 6. Feed SBOM into Dashboards or Trackers**

For long-term analysis, feed SBOM into tools like:

* **DependencyTrack**
* **Anchore**
* **JFrog Xray**
* **CycloneDX CLI**

These tools:

* Monitor all your apps
* Automatically alert when a CVE is discovered later

**📦 What Does an SBOM Actually Contain?**

A good SBOM contains:

{

"name": "flask",

"version": "2.3.2",

"license": "BSD-3-Clause",

"purl": "pkg:pypi/flask@2.3.2",

"dependencies": [

"werkzeug",

"itsdangerous",

"jinja2"

]

}

This tells us:

* What component
* Which version
* Where it came from
* What depends on what

**✅ Summary — Your Job in SBOM Lifecycle**

| **Step** | **Description** |
| --- | --- |
| 📄 Generate | From source code or container |
| 🔐 Sign (optional) | Using cosign/sigstore |
| 🧠 Scan | With Trivy, Grype, Snyk |
| 📬 Store | Upload to Git, S3, Artifact repo |
| 🚨 Alert/Act | Fix CVEs or notify |
| 📊 Feed | To dashboards or vulnerability trackers |

**Tools You Use**

| **Purpose** | **Tool** |
| --- | --- |
| Generate SBOM | Trivy, Syft |
| Scan SBOM | Trivy, Grype |
| Sign SBOM | Cosign, Sigstore |
| Store/Track | Nexus, JFrog, GitHub |
| Monitor CVEs | DependencyTrack, Anchore, GitHub Security |

**Final Thought: Why Is SBOM a Big Deal?**

Because software supply chain attacks (like **SolarWinds**, **Log4Shell**) became so frequent, even the **US government mandates SBOMs** for any software sold to federal agencies.

**SBOM = visibility + traceability + security**

**Formats**

* **CycloneDX**:  
  **Cyclone Dependency eXchange**  
  (A lightweight SBOM standard focused on security, originally developed by OWASP)
* **SPDX**:  
  **Software Package Data Exchange**  
  (A standardized SBOM format by the Linux Foundation to describe software components, licenses, and metadata)

**🔥 Key Differences (CycloneDX vs SPDX)**

| **Feature** | **CycloneDX** | **SPDX** |
| --- | --- | --- |
| 🧠 Created By | OWASP | Linux Foundation |
| 🔁 Main Use | **Security & vulnerability tracking** | **License & compliance tracking** |
| 🧰 Tooling Ecosystem | Syft, Trivy, CycloneDX CLI, etc. | SPDX tools, License scanners, etc. |
| 📦 Focus | Security metadata (CVEs, hashes) | Legal metadata (licenses, notices) |
| 📄 Output Format Options | XML, JSON | Tag-Value, JSON, YAML, RDF, etc. |
| 🔗 Component Model | Package URLs (PURLs), hashes | SPDX IDs, file-level license refs |
| ✅ Easier to Read | ✅ Yes (JSON is clean) | ❌ Tag-value format is messy |
| 🎯 Ideal Use Case | DevSecOps pipelines | Legal SBOMs for compliance |

**🧰 Example Use Cases**

* Use **CycloneDX** if:
  + You're in **DevSecOps**
  + You want to integrate with **Trivy, Syft, Grype**
  + You care about **vulnerabilities, hashes, runtime components**
* Use **SPDX** if:
  + You're in **legal/compliance**
  + You need to submit SBOMs to customers, regulators, or for SPDX-specific audits
  + You care more about **licenses and provenance**

**🧩 STEP 0: Install All Required Tools**

**1. 🔧 Install Syft**

curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sudo sh -s -- -b /usr/local/bin

syft version

**2. 🔧 Install Grype**

curl -sSfL https://raw.githubusercontent.com/anchore/grype/main/install.sh | sudo sh -s -- -b /usr/local/bin

grype version

**3. 🔧 Install Trivy**

sudo apt-get install wget apt-transport-https gnupg lsb-release

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -

echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb\_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy

**4. 🔧 Install CycloneDX CLI**

wget https://github.com/CycloneDX/cyclonedx-cli/releases/latest/download/cyclonedx-linux-x64

chmod +x cyclonedx-linux-x64

sudo mv cyclonedx-linux-x64 /usr/local/bin/cyclonedx

cyclonedx version

**5. 🔐 Install Cosign (for signing SBOM)**

COSIGN\_VERSION=$(curl -s https://api.github.com/repos/sigstore/cosign/releases/latest | grep tag\_name | cut -d '"' -f 4)

curl -LO "https://github.com/sigstore/cosign/releases/download/${COSIGN\_VERSION}/cosign-linux-amd64"

**Step 2: Move to PATH and Make Executable**

chmod +x cosign-linux-amd64

sudo mv cosign-linux-amd64 /usr/local/bin/cosign

**🧪**

**📦 STEP 2: Generate SBOM (Syft)**

syft . -o cyclonedx-json -q > enriched-sbom.json

syft target/java-demoapp-null.jar -o cyclonedx-json > sbom-myapp.json

syft docker.io/library/nginx:latest -o cyclonedx-json > image-enriched-sbom.json

This SBOM includes:

* Package names
* Versions
* Package URLs (purl)
* Partial licenses

**🛡️ STEP 3: Scan SBOM for Vulnerabilities (Grype)**

grype sbom:enriched-sbom.json --output table

or save result:

grype sbom:enriched-sbom.json -o json > grype-report.json

**🧪 Optional: Vulnerability Scan with Trivy as double-check**

trivy fs . --format cyclonedx --output enriched-sbom.json

trivy sbom enriched-sbom.json --format table

trivy fs . --scanners vuln,license,secret

**🔏 STEP 4: Sign the SBOM (Cosign)**

First, generate a key-pair (just once):

cosign generate-key-pair

Then, sign the SBOM:

cosign attest \

--key cosign.key \ # 🔑 Private key used to sign the attestation

--predicate enriched-sbom.json \ # 📄 The JSON file to attach (e.g., SBOM)

--type cyclonedxcyclo \ # 🏷️ Metadata type (used for filtering/verification)

<artifact-name> # 📦 The artifact (usually an image)

ls

**✅ Verify a signed SBOM or artifact (attestation):**

cosign verify-attestation --key cosign.pub <artifact>

For example:

cosign verify-attestation --key cosign.pub nginx:latest

**You can also inspect the SBOM from the attestation:**

cosign verify-attestation --key cosign.pub nginx:latest \

--type cyclonedx

**🔁 STEP 5: Convert SBOM Between Formats (Optional)**

**CycloneDX → SPDX:**

cyclonedx convert --input-file enriched-sbom.json --output-format spdxjson --output-file sbom.spdx.json

**🧪 STEP 6: Validate the Final SBOM**

cyclonedx validate --input-file enriched-sbom.json

**2. Who Verifies These Signatures?**

**These roles typically perform verification:**

| **Role** | **Why They Verify** |
| --- | --- |
| **DevOps Engineers** | Ensure only signed images are deployed in CI/CD |
| **Security Teams** | Validate that SBOMs and images haven't been tampered with |
| **Cloud Native Platforms** (e.g., Tekton Chains, Kyverno, Gatekeeper) | Enforce admission policies to block unsigned artifacts |
| **Auditors / Compliance Engineers** | Verify provenance & supply chain security |
| **Package Consumers** | Ensure artifacts came from trusted publishers (esp. open-source SBOMs or tools) |

**3. When to Verify**

**Here are the critical points in a DevSecOps workflow:**

| **Stage** | **What Gets Verified** | **When** |
| --- | --- | --- |
| **CI/CD Deployments** | Docker image signature | Before deploying to staging/prod |
| **Admission Control** | SBOM attestation or signature | Kubernetes gatekeepers (OPA/Kyverno) |
| **Supply Chain Scanning** | SBOMs & provenance | During security scans |
| **Artifact Pull** | Image signatures | Before pulling from registries |
| **Incident Response** | Attestations | During post-incident RCA |

**🔐 Summary**

* You sign with your **private key**
* Others verify with your **public key**
* Verification happens before usage, during deploy, or at gates (CI/CD or K8s Admission)
* This ensures **trust**, **immutability**, and **compliance**

**📂 Output Files You'll Get**

| **File Name** | **Purpose** |
| --- | --- |
| base-sbom.json | Raw SBOM from Syft |
| enriched-sbom.json | Final SBOM with licenses, hash |
| grype-report.json | CVE vulnerability results |
| sbom.spdx.json | SPDX-compliant SBOM |
| .sig file | Signed attestation |