Release Engineering Quick-Start

November 4, 2025

Contents

1 Why Release Engineering Matters

Release Engineering (*RelEng*) turns source code into dependable, reproducible artifacts and makes delivery routine. Core outcomes:

- A **versioning model** (usually SemVer) that maps change scope to version numbers.
- **Tagged source**, signed builds, and **immutable artifacts**.
- **Automations** that build, test, package, sign, scan, and publish on events (merge, tag, or manual).
- **Auditability**: release notes, provenance, SBOMs, checksums, signatures.

2 Versioning Strategy (SemVer + Conventional Commits)

2.1 Semantic Versioning

- `MAJOR`: breaking changes (`v2.0.0`)
- `MINOR`: new backward-compatible features (`v1.1.0`)
- `PATCH`: bug fixes (`v1.0.1`)

2.2 Conventional Commits (supports automation)

Use commit types to drive changelogs and version bumps (e.g., `feat:`, `fix:`, `perf:`, `docs:`, `refactor:`). Mark breaking changes with `!` or `BREAKING CHANGE:`.

```
# Examples
git commit -m "feat(auth): add OIDC login flow"
git commit -m "fix(router): handle trailing slash redirects"
git commit -m "feat(api)!: drop deprecated /v1 endpoints"
```

3 Source Tagging & GitHub Releases

3.1 Tag and Release (UI + CLI)

GitHub UI

1. Repo → **Releases** → **Draft a new release**.
2. Create tag (e.g., `v1.0.0`) targeting `main` (or your release branch).
3. Generate notes; attach binaries if applicable; **Publish**.

GitHub CLI

```
gh auth login
gh release create v1.1.0 --generate-notes
gh release list
```

3.2 Auto-Generate Release Notes

GitHub can generate notes from merged PRs and commits. Keep PR titles crisp and use labels to improve sections.

4 Packaging Paths

Choose the artifact type(s) that fit your distribution model.

4.1 Node Package (GitHub Packages or npmjs)

package.json essentials

```
{
  "name": "@<owner>/<repo>",
  "version": "1.1.1",
  "main": "dist/index.js",
  "files": ["dist"],
  "repository": {"type":"git","url":"https://github.com/<owner>/<repo>.git"},
  "publishConfig": {"registry":"https://npm.pkg.github.com"}
}
```

.npmrc for owner scope

```
@<owner>:registry=https://npm.pkg.github.com
```

Local publish (PAT with packages:write)

```
npm ci
npm publish --access public   # or private if using GitHub Packages
```

GitHub Actions: publish on Release

```yaml
name: Publish package
on:
  release:
    types: [published]
permissions:
  contents: read
  packages: write
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          registry-url: https://npm.pkg.github.com
          scope: "@<owner>"
      - run: npm ci
      - run: npm publish
        env:
          NODE_AUTH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

4.2 Python Package (PyPI or GitHub Packages)

Project metadata (pyproject.toml)

```toml
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "your-package"
version = "1.0.0"
description = "Awesome thing"
requires-python = ">=3.9"
```

Publish with Twine

```bash
python -m build
python -m twine upload dist/*      # uses ~/.pypirc
```

GitHub Actions: build & publish on tag

```yaml
name: PyPI publish
on:
  push:
    tags: ["v*.*.*"]
jobs:
  build-publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: "3.11" }
      - run: pip install build twine
      - run: python -m build
      - run: python -m twine upload dist/*
        env:
          TWINE_USERNAME: __token__
          TWINE_PASSWORD: ${{ secrets.PYPI_TOKEN }}
```

4.3 Java (Maven) to GitHub Packages

pom.xml distributionManagement

```xml
<distributionManagement>
  <repository>
    <id>github</id>
    <name>GitHub Packages</name>
    <url>https://maven.pkg.github.com/&lt;owner&gt;/&lt;repo&gt;</url>
  </repository>
</distributionManagement>
```

Settings (GITHUB_TOKEN)

```xml
<servers>
  <server>
    <id>github</id>
    <username>${env.GITHUB_ACTOR}</username>
    <password>${env.GITHUB_TOKEN}</password>
  </server>
</servers>
```

Workflow

```yaml
name: Maven publish
on:
  release:
    types: [published]
permissions:
  contents: read
  packages: write
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Java
        uses: actions/setup-java@v4
        with:
          distribution: temurin
          java-version: "21"
          cache: maven
          server-id: github
          server-username: GITHUB_ACTOR
          server-password: GITHUB_TOKEN
      - run: mvn -B -DskipTests deploy
```

5 Containers to GHCR (GitHub Container Registry)

5.1 Dockerfile (example)

```dockerfile
FROM node:20-alpine
LABEL org.opencontainers.image.source="https://github.com/<owner>/<repo>"
WORKDIR /app
COPY . /app
RUN npm ci --omit=dev
EXPOSE 3000
CMD ["npm","start"]
```

5.2 Build & push (CLI)

```bash
export IMAGE=ghcr.io/<owner>/<repo>:v1.0.0
echo $GITHUB_TOKEN | docker login ghcr.io -u <owner> --password-stdin
docker build -t $IMAGE .
docker push $IMAGE
```

5.3 Workflow: docker/build-push-action

```yaml
name: Build & Push Image
on:
  push:
    tags: ["v*.*.*"]
permissions:
  contents: read
  packages: write
jobs:
  image:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          tags: ghcr.io/${{ github.repository }}:${{ github.ref_name }}
```

6 Containers to AWS (ECR, ECS Fargate, EKS)

6.1 Amazon ECR: Build, Tag, Push

Create (or use) a repository

```
aws ecr create-repository --repository-name <name> --region <region> \
  --image-scanning-configuration scanOnPush=true
```

Authenticate (OIDC or static credentials)

```
aws ecr get-login-password --region <region> \
| docker login --username AWS --password-stdin <account>.dkr.ecr.<region>.amazonaws.com
```

Build, tag, push

```
IMAGE=<account>.dkr.ecr.<region>.amazonaws.com/<name>:v1.0.0
docker build -t "$IMAGE" .
docker push "$IMAGE"
```

6.2 GitHub Actions → ECR (OIDC)

Set up AWS IAM OIDC trust (once) Create an IAM role trusted by `token.actions.githubusercontent.co` (replace placeholders).

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"Federated":
    ↪  "arn:aws:iam::<account>:oidc-provider/token.actions.githubusercontent.com"},
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
      },
      "StringLike": {
        "token.actions.githubusercontent.com:sub":
        ↪  "repo:<owner>/<repo>:ref:refs/heads/main"
      }
    }
  }]
}
```

Attach a policy granting ECR push (minimal example).

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "ecr:GetAuthorizationToken",
      "ecr:BatchCheckLayerAvailability",
      "ecr:CompleteLayerUpload",
      "ecr:DescribeRepositories",
      "ecr:InitiateLayerUpload",
      "ecr:PutImage",
      "ecr:UploadLayerPart"
    ],
    "Resource": "*"
  }]
}
```

Workflow: build-push-to-ECR (with cache)

```yaml
name: Build and Push (ECR)
on:
  push:
    tags: ["v*.*.*"]
permissions:
  id-token: write
  contents: read
jobs:
  build-push:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Configure AWS credentials (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::<account>:role/<role-name>
          aws-region: <region>

      - name: Login to Amazon ECR
        uses: aws-actions/amazon-ecr-login@v2

      - name: Build and push
        uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          tags: |
            <account>.dkr.ecr.<region>.amazonaws.com/<name>:${{ github.ref_name }}
            <account>.dkr.ecr.<region>.amazonaws.com/<name>:${{ github.sha }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

6.3 ECS Fargate: Deploy

Task definition (snippet)

```
{
  "family": "my-service",
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024",
  "requiresCompatibilities": ["FARGATE"],
  "executionRoleArn": "arn:aws:iam::<account>:role/ecsTaskExecutionRole",
  "taskRoleArn": "arn:aws:iam::<account>:role/appTaskRole",
  "containerDefinitions": [{
    "name": "web",
    "image": "<account>.dkr.ecr.<region>.amazonaws.com/<name>:v1.0.0",
    "portMappings": [{"containerPort": 3000, "protocol": "tcp"}],
    "essential": true
  }]
}
```

Workflow: update ECS service (rolling)

```yaml
name: Deploy to ECS
on:
  workflow_dispatch: {}
permissions:
  id-token: write
  contents: read
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::<account>:role/<role-name>
          aws-region: <region>

      - uses: aws-actions/amazon-ecr-login@v2

      - name: Build and push image
        uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          tags: <account>.dkr.ecr.<region>.amazonaws.com/<name>:${{ github.sha }}

      - name: Render task definition
        uses: aws-actions/amazon-ecs-render-task-definition@v1
        with:
          task-definition: ecs-task-def.json
          container-name: web
          image: <account>.dkr.ecr.<region>.amazonaws.com/<name>:${{ github.sha }}

      - name: Deploy service
        uses: aws-actions/amazon-ecs-deploy-task-definition@v1
        with:
          task-definition: ${{ steps.render.outputs.task-definition }}
          service: my-service
          cluster: my-cluster
          wait-for-service-stability: true
```

Blue/Green (optional) Use ECS + CodeDeploy for canary/blue-green; wire the service to a CodeDeploy application and deployment group, then trigger a deployment with a new task definition revision.

6.4 EKS: Deploy

Configure `kubeconfig` via OIDC role Grant the role `eks:DescribeCluster`. Map the role to a Kubernetes RBAC subject with cluster-admin or a bound role.

```
aws eks update-kubeconfig --name <cluster> --region <region> --role-arn
  ↪ arn:aws:iam::<account>:role/<role-name>
```

Minimal Deployment (snippet)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 2
  selector: { matchLabels: { app: web } }
  template:
    metadata: { labels: { app: web } }
    spec:
      containers:
      - name: web
        image: <account>.dkr.ecr.<region>.amazonaws.com/<name>:v1.0.0
        ports: [{containerPort: 3000}]
```

Workflow: kubectl rollout

```yaml
name: Deploy to EKS
on: { workflow_dispatch: {} }
permissions:
  id-token: write
  contents: read
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::<account>:role/<role-name>
          aws-region: <region>
      - name: Get kubeconfig
        run: aws eks update-kubeconfig --name <cluster> --region <region>
      - name: Update image
        run: |
          kubectl set image deploy/web
          ↪ web=<account>.dkr.ecr.<region>.amazonaws.com/<name>:${{ github.sha }}
          kubectl rollout status deploy/web
```

6.5 Tips & Gotchas

- Enable **scanOnPush** in ECR; fail builds on critical vulns (e.g., pre-deploy gate with Trivy/Grype).
- Use `cache-to: type=gha,mode=max` to speed Docker builds.
- Prefer **multi-arch** images when you have arm64/amd64 hosts (`platforms: linux/amd64,linux/arm64`).
- Grant the OIDC role least-privilege: ECR push, ECS deploy, or EKS describe; use separate roles per workflow if needed.

7 Security: Checksums, Signatures, SBOM, Provenance

7.1 Checksums

Generate `SHA256` for assets and publish alongside releases.

```
shasum -a 256 artifact.tar.gz > artifact.tar.gz.sha256
```

7.2 Signing (cosign) and SBOM (syft)

```
# Image signature
cosign sign ghcr.io/<owner>/<repo>:v1.0.0

# SBOM
syft ghcr.io/<owner>/<repo>:v1.0.0 -o spdx-json > sbom.spdx.json
```

7.3 Supply-chain provenance (SLSA generator gist)

Record build provenance and attach to the release (or publish as attestation).

8 Release Trains, Channels, and Promotion

- **Channels**: `-alpha`, `-beta`, `-rc`, `-stable`.
- **Promotion**: build once, promote by retagging (no rebuild).
- **Canary → Staged → Global** rollouts with metrics and fast rollback.

9 Branching & Protection

- Protect `main`: require PR reviews, status checks, linear history.
- Optional maintenance branches (`release/1.x`) for patch backports.
- Tag from a green build of `main` (or from a release branch commit).

10 Automated Changelog

10.1 Keep a Changelog (example sections)

- Added, Changed, Deprecated, Removed, Fixed, Security.

10.2 Action example (conventional commits)

```yaml
name: Changelog
on:
  push:
    tags: ["v*.*.*"]
jobs:
  changelog:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Generate changelog
        uses: metcalfc/changelog-generator@v4.3.1
        with:
          mytoken: ${{ secrets.GITHUB_TOKEN }}
```

11 Monorepos (optional)

- Workspaces (npm/pnpm/yarn) or tools like Lerna/Changesets.
- Independent versioning per package vs. fixed version for all.
- Filtered workflows: build/test only what changed.

12 Troubleshooting Cheatsheet

Symptom	**Likely Fix**
403 on package publish	Missing scopes: add `packages:write`; ensure registry/scope in `.npmrc`.
Image push denied	Workflow permissions: enable "Read and write" for GITHUB_TOKEN; login to GHCR.
Wrong version	Ensure tag matches version file; automate version bump via CI.
SBOM empty	Build stage excluded deps; generate SBOM from final artifact/image.