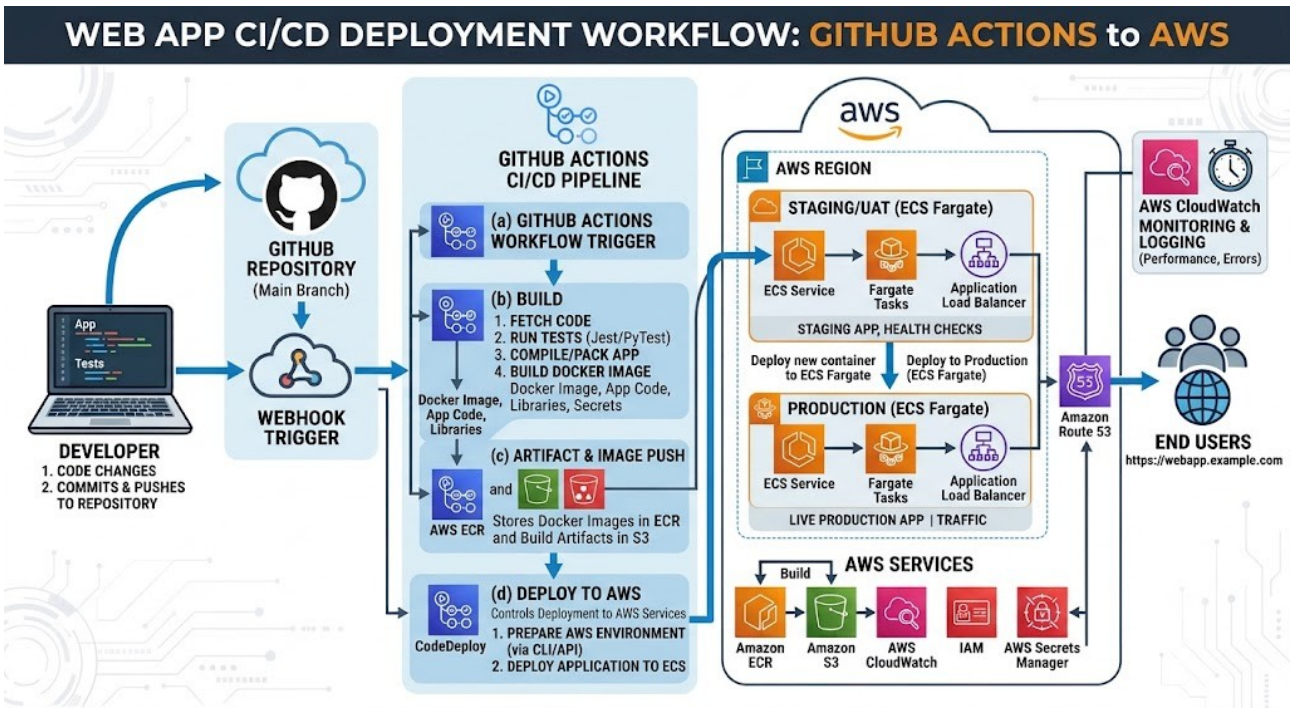


For a web application hosted on **AWS, with source code maintained in GitHub**, each phase of the software delivery lifecycle (Dev, Build, Stage, and Prod) produces and consumes different artifacts. These artifacts include source code, configuration files, dependency packages, compiled application binaries, container images, deployment packages, infrastructure templates, test reports, and release documentation.

A useful way to design this is to establish an **artifact lifecycle and promotion strategy**, where the same tested and versioned build artifact moves from Build → Stage → Production rather than rebuilding the application separately for each environment.

1. Overview of artifacts across Dev, Build, Stage and Prod

Phase	Primary purpose	Key artifacts	Typical formats
Dev	Application development	Source code, configuration, unit tests, API specifications	.js, .ts, .py, .java, .json, .yaml, .yml, .md
Build	Compile, package and validate	Build packages, dependencies, container images, test reports, SBOM	.zip, .jar, .war, .tar.gz, Docker image, .json, .xml
Stage	Pre-production validation	Deployment packages, infrastructure templates, test reports, release candidates	Docker image, .zip, .yaml, .json, .xml, .html
Prod	Production deployment and operations	Release packages, deployment manifests, configuration, monitoring and audit records	Docker image, .yaml, .json, .log, .csv, .zip

2. Detailed artifact inventory by phase

A. DEV – Development phase

This phase focuses on application development, source control, developer testing, and code quality.

Artifact	Description	Format / Type	Typical location
Application source code	Frontend and backend source code	.js, .ts, .jsx, .tsx, .py, .java	GitHub repository
Application configuration	Application-specific settings	.json, .yaml, .env.example	GitHub
Dependency definitions	Libraries and packages required	package.json, requirements.txt, pom.xml	
Dependency lock files	Exact dependency versions	package-lock.json, yarn.lock, poetry.lock	GitHub
API specifications	REST API definitions	OpenAPI .yaml / .json	GitHub
Database scripts	Database schema and migrations	.sql, migration files	GitHub
Unit test scripts	Automated developer tests	.js, .ts, .py, .java	GitHub
Test data	Sample data for testing	.json, .csv, .sql	GitHub or test storage
Code quality configuration	Linting and static analysis rules	.json, .yaml, .properties	GitHub
Dockerfile	Instructions to build container images	Dockerfile	GitHub
Infrastructure as Code	AWS infrastructure definitions	Terraform .tf, CloudFormation .yaml / .json	GitHub

Typical outputs from Dev:

- Developer commits and pull requests.
- Feature branches.
- Reviewed and approved source code.
- Unit test scripts.
- Updated application and infrastructure configurations.

B. BUILD – Build and packaging phase

This phase converts the approved source code into deployable application artifacts. Builds are typically triggered by GitHub pull request merges or commits to designated branches.

Artifact	Description	Format / Type	Typical storage
Frontend build	Compiled frontend application	.js, .css, .html, static assets	S3 / GitHub Actions
Backend build	Compiled backend application	.jar, .war, .zip, .dll	S3 / GitHub Actions
Container image	Packaged application with runtime dependencies	Docker / OCI image	Amazon ECR
Build configuration	Instructions for compiling and packaging	buildspec.yml, pom.xml, package.json	GitHub
Build logs	Build execution information	.log, .txt	GitHub Actions / CloudWatch
Unit test results	Results of automated unit tests	JUnit .xml, JSON	GitHub Actions
Code coverage	Percentage of code covered by tests	Cobertura .xml, LCOV .info	GitHub / quality platform
Static code analysis	Code quality and vulnerability findings	JSON, XML, HTML	SonarQube / GitHub
Dependency scan report	Vulnerabilities in third-party libraries	JSON, SARIF	GitHub / security platform
Software Bill of Materials (SBOM)	Inventory of application components and dependencies	SPDX JSON, CycloneDX JSON	GitHub / S3
Build metadata	Build number, commit ID, timestamp, version	JSON / TXT	GitHub / S3
Artifact checksum	Integrity verification of build output	SHA-256 .txt	S3 / release repository

Example build outputs

For a React frontend:

```
frontend-build/
  index.html
  assets/
    app.js
    app.css
    images/
```

For a Java backend:

```
application-1.0.0.jar
```

For a containerised application:

```
my-web-app:1.0.0
```

The container image would be pushed to **Amazon Elastic Container Registry (ECR)**.

For a typical GitHub Actions workflow, the build configuration would be:

```
.github/
  workflows/
    build.yml
```

The workflow can produce and upload build artifacts using GitHub Actions artifact functionality.

C. STAGE – Pre-production deployment and validation phase

The Stage environment is used to validate the application before production deployment.

The objective is to verify that the application, infrastructure, integrations, security, and performance are working as expected.

Artifact	Description	Format / Type	Typical storage
Release candidate	Version of application approved for pre-production testing	Docker image / .zip / .jar	ECR / S3
Deployment manifest	Defines application deployment configuration	.yaml, .json	GitHub
Infrastructure templates	AWS infrastructure definitions	Terraform .tf, CloudFormation .yaml	GitHub
Environment configuration	Stage-specific application settings	.yaml, .json	AWS Systems Manager
Database migration scripts	Database schema changes	.sql	GitHub
Integration test reports	Integration test execution results	XML, JSON, HTML	GitHub / S3
Functional test reports	Application functionality validation	XML, HTML, JSON	Test management platform
Regression test reports	Results of regression testing	XML, HTML	GitHub / S3
Performance test reports	Load and stress test results	HTML, JSON, CSV	S3
Security test reports	Vulnerability and penetration testing findings	SARIF, JSON, HTML	Security platform
UAT sign-off	Business acceptance approval	PDF, DOCX, Jira approval record	Document repository / Jira
Deployment logs	Deployment execution and validation logs	.log, JSON	CloudWatch
Smoke test results	Basic application health verification	JSON, XML, HTML	GitHub / S3

Important Stage artifacts

Consider the following particularly important:

1. Release Candidate (RC)

This is the actual application package intended for production.

Example:

web-app:v1.0.0-rc1

2. Deployment configuration

For example:

```
deployment/
   task-definition.json
   appspec.yaml
   deployment.yaml
```

The exact files depend on the AWS deployment service being used.

3. Test evidence

Examples:

- Functional testing report.
- API testing report.
- Security scanning report.
- Performance testing report.
- UAT approval.

4. Release approval

This is the formal evidence that the application has passed Stage validation and is authorised for production deployment.

D. PROD – Production deployment and operations phase

This phase involves deploying the approved release to the production AWS environment.

The key principle is that production should receive the exact application artifact that passed Stage testing.

Artifact	Description	Format / Type	Typical storage
Production release	Approved application version	Docker image / .zip / .jar	ECR / S3
Release manifest	Identifies the deployed application version	JSON / YAML	GitHub / S3
Deployment configuration	Production deployment settings	.yaml, .json	GitHub / AWS
Infrastructure configuration	Production infrastructure definition	Terraform .tf, CloudFormation .yaml	GitHub
Secrets and credentials	Database passwords, API keys, tokens	Secure secret values	AWS Secrets Manager
Application logs	Application execution logs	JSON / text	CloudWatch
Infrastructure logs	AWS infrastructure and service events	JSON	CloudWatch / CloudTrail
Monitoring dashboards	Application and infrastructure health	CloudWatch dashboard configuration	AWS CloudWatch
Alert configurations	Production monitoring and alert rules	JSON / YAML	AWS CloudWatch
Backup records	Database and application backup information	Snapshots / metadata	AWS Backup / S3
Rollback package	Previously approved application release	Docker image / .zip / .jar	ECR / S3
Release notes	Summary of production changes	Markdown / PDF	GitHub
Deployment audit records	Who deployed what and when	JSON / logs	GitHub / AWS

Important production artifacts

1. **Production release manifest:** Identifies the exact version deployed
2. **Deployment approval:** Records the authorised production release
3. **Monitoring configuration:** Defines application health monitoring
4. **Rollback reference:** Identifies the previous stable version
5. **Operational documentation:** Includes troubleshooting and recovery procedures.

3. Where these artifacts reside in GitHub and AWS

An important consideration is that not every artifact should be stored in GitHub. Recommended divisiion of artifact storage is as follows:

Repository / AWS service	**Artifacts stored**
GitHub repository	Source code, Dockerfiles, deployment scripts, IaC, test scripts, workflow definitions
GitHub Actions	Build outputs, test reports, workflow logs, release evidence
Amazon ECR	Docker/OCI container images
Amazon S3	Build packages, static application files, test reports, release packages, backups
AWS Systems Manager Parameter Store	Environment-specific configuration parameters
AWS Secrets Manager	Passwords, credentials, API keys and secrets
Amazon CloudWatch	Application logs, infrastructure logs, metrics and dashboards
AWS CloudFormation	Infrastructure deployment stacks and templates
Terraform state backend (typically S3)	Terraform infrastructure state
GitHub Releases	Versioned release notes and release metadata

AWS CodePipeline, for example, uses Amazon S3 to store and transfer pipeline input and output artifacts between stages.

4. Recommended GitHub repository structure

For a web application, I would recommend a structure similar to this:

```
web-application/
│
├── frontend/
│   ├── src/
│   ├── public/
│   ├── package.json
│   └── package-lock.json
│
├── backend/
│   ├── src/
│   ├── tests/
│   └── pom.xml
│
├── database/
│   └── migrations/
│
├── infrastructure/
│   ├── dev/
│   ├── stage/
│   └── prod/
│
├── deployment/
│   ├── Dockerfile
│   ├── appspec.yaml
│   └── scripts/
│
├── tests/
│   ├── unit/
│   ├── integration/
│   ├── functional/
│   └── performance/
│
├── .github/
│   └── workflows/
│       ├── build.yml
│       ├── deploy-dev.yml
│       ├── deploy-stage.yml
│       └── deploy-prod.yml
│
├── docs/
│   ├── architecture.md
│   ├── deployment-guide.md
│   └── release-notes.md
│
└── README.md
```

This provides a clear separation between application code, infrastructure, deployment instructions, testing, and CI/CD workflows.

5. Recommended artifact promotion lifecycle

The following is the recommended approach for managing artifacts across the four phases.

Step	Activity	Output artifact
1. Dev	Developer commits code to GitHub	Source code commit
2. Build	GitHub Actions compiles and packages the application	Build package / Docker image
3. Build validation	Execute unit tests, security scans and quality checks	Test reports / SBOM
4. Deploy Dev	Deploy build into development AWS environment	Dev deployment
5. Deploy Stage	Promote the same build to Stage	Release candidate
6. Stage validation	Execute integration, regression, performance and UAT tests	Validation reports / approvals
7. Production approval	Obtain authorised release approval	Release approval record
8. Deploy Prod	Deploy the approved artifact	Production release
9. Operations	Monitor application and infrastructure	Logs, metrics, alerts and audit records

Important: Do not rebuild the application separately for Stage and Production. Promote the same immutable artifact, preferably identified by its container image digest (SHA-256), through each environment.

Only environment-specific configuration and secrets should change.

6. Important distinction: Artifacts versus configurations

This is particularly important when deploying applications across multiple AWS environments.

Consider the following example:

Component	Dev	Stage	Production
Application image	web-app@sha256:ABC123	Same image	Same image
Database endpoint	Dev DB	Stage DB	Production DB
API endpoint	Dev API	Stage API	Production API
Logging level	DEBUG	INFO	WARN/INFO
AWS account	Development account	Non-production account	Production account
Secrets	Dev secrets	Stage secrets	Production secrets

The application image remains unchanged. Only environment-specific configuration changes.

Never commit actual production credentials, passwords or API secrets into GitHub.

7. Recommended artifact governance and ownership

For enterprise-level governance, I would also maintain an artifact ownership matrix.

Artifact category	Responsible team	Approval / control
Source code	Development	Peer review / pull request
Build packages	DevOps	Automated build validation
Container images	DevOps / Platform	Vulnerability scanning
Infrastructure templates	Cloud / DevOps	Infrastructure code review
Test reports	QA	QA approval
Security reports	Security team	Security clearance
UAT reports	Business / Product Owner	Business sign-off
Release manifest	Release Management	Release approval
Production deployment	DevOps / Operations	Change approval
Monitoring configuration	Cloud Operations	Operational validation

8. Recommended minimum artifact checklist

While establishing an artifact management standard for the organisation, the following could be made mandatory.

No.	Artifact	Format	Mandatory phase
1	Source code	Git	Dev
2	Dependency manifest and lock file	JSON / XML / TXT	Dev
3	Build workflow	YAML	Build
4	Dockerfile	Dockerfile	Build
5	Compiled application package	JAR / ZIP / OCI image	Build
6	Unit test report	XML / HTML	Build
7	Code quality report	JSON / HTML	Build
8	Vulnerability scan report	SARIF / JSON	Build
9	SBOM	SPDX / CycloneDX JSON	Build
10	Infrastructure as Code	TF / YAML / JSON	Build
11	Deployment manifest	YAML / JSON	Stage
12	Integration and regression reports	XML / HTML	Stage
13	Performance test report	HTML / CSV	Stage
14	UAT sign-off	PDF / Jira record	Stage
15	Release manifest	JSON / YAML	Prod
16	Deployment approval	Jira / GitHub approval record	Prod
17	Rollback reference	Image digest / version	Prod
18	Monitoring and logging configuration	JSON / YAML	Prod
19	Production deployment logs	JSON / TXT	Prod
20	Release notes	Markdown / PDF	Prod

9. One important question about your AWS architecture

The exact deployment artifacts depend on which AWS hosting architecture you use. For example:

AWS hosting architecture	**Main deployment artifacts**
EC2	ZIP package, AppSpec YAML, deployment scripts
ECS with Fargate	Docker image, ECS task definition JSON, AppSpec YAML
EKS	Docker image, Kubernetes deployment YAML, Helm charts
Lambda	Deployment ZIP, Lambda configuration, SAM/CloudFormation templates
S3 + CloudFront	Frontend build files, S3 deployment package, CloudFront configuration

AWS CodeDeploy uses different deployment specifications depending on the target platform. For example, ECS uses an AppSpec file and task definition, whereas EC2 deployments use an application revision bundle.