

# GitHub Actions Advanced Features

Building Powerful CI/CD Pipelines

55 Minutes • 5 Topics • 10 Demos

# Agenda

---

- **Matrix Builds** (10 min) - Parallelize across versions & platforms
- **Reusable Workflows** (10 min) - DRY principle for workflows
- **Composite Actions** (10 min) - Build custom actions
- **Environment Protection Rules** (10 min) - Control deployments
- **Caching & Artifacts** (10 min) - Speed up workflows
- **Q&A** (5 min)

# 1. Matrix Builds

---

## What are Matrix Builds?

- Run jobs across multiple versions, platforms, or configurations
- Automatically creates job for each combination
- Perfect for testing compatibility
- Reduces workflow duplication

# Matrix Builds: Simple Demo

## DEMO 1: Basic Matrix

```
strategy:
  matrix:
    os: [ubuntu-latest, windows-latest, macos-latest]
    node: [16, 18, 20]

runs-on: ${{ matrix.os }}
steps:
  - uses: actions/setup-node@v3
    with:
      node-version: ${{ matrix.node }}
```

**Result:** Creates 9 jobs (3 OS × 3 Node versions)

# Matrix Builds: Advanced Demo

## DEMO 2: Matrix with Include/Exclude

```
strategy:
  matrix:
    os: [ubuntu-latest, windows-latest]
    node: [16, 18, 20]
    include:
      - os: ubuntu-latest
        node: 20
        experimental: true
    exclude:
      - os: windows-latest
        node: 16
```

**Advanced Features:** `include` adds extra jobs, `exclude` removes combinations

## 2. Reusable Workflows

---

### Why Reusable Workflows?

- Eliminate workflow duplication across repositories
- Centralize CI/CD logic in one place
- Pass inputs and secrets to workflows
- Return outputs from workflows
- Easier maintenance and updates

# Reusable Workflows: Simple Demo

## DEMO 3: Basic Reusable Workflow

### Called Workflow:

```
on:
  workflow_call:
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Reusable workflow!"
```

### Caller Workflow:

```
jobs:
  call-workflow:
    uses: ../.github/workflows/reusable-simple.yml
```

# Reusable Workflows: Advanced Demo

## DEMO 4: Workflow with Inputs & Outputs

```
on:
  workflow_call:
    inputs:
      environment:
        required: true
        type: string
    outputs:
      build-id:
        value: ${{ jobs.build.outputs.id }}
    secrets:
      deploy-token:
        required: true
```

# 3. Composite Actions

---

## What are Composite Actions?

- Bundle multiple steps into a single reusable action
- Defined with `action.yml` file
- Can use other actions within them
- Support inputs, outputs, and environment variables
- Different from reusable workflows (actions vs workflows)

# Composite Actions: Simple Demo

## DEMO 5: Basic Composite Action

### action.yml:

```
name: 'Setup Node with Cache'
description: 'Install Node and cache dependencies'
runs:
  using: 'composite'
  steps:
    - uses: actions/setup-node@v3
    - run: npm ci
      shell: bash
```

### Usage:

```
- uses: ../github/actions/setup-node
```

# Reusable Workflows vs Composite Actions

## 1. Where they live & how you invoke them

- **Composite Action:** Defined in `action.yml`, invoked as a *step* inside a job ( `uses: ./path/to/action` )
- **Reusable Workflow:** Normal workflow YAML file, invoked at the *job level* ( `uses: .github/workflows/file.yml` with `workflow_call` )

## 2. Scope: steps vs jobs

- **Composite Action = steps only:** Packages steps into one reusable component; runs on the same runner as the caller job
- **Reusable Workflow = full workflow with jobs:** A complete workflow you can run from other workflows;

defines multiple jobs with their own runners and dependencies

# Reusable Workflows vs Composite Actions (cont.)

## 3. Inputs & secrets

- **Reusable Workflow:** Inputs + secrets defined on `workflow_call`; callers pass them via `with:` and `secrets:`
- **Composite Action:** Inputs/outputs in `action.yml`; secrets must be passed explicitly from caller
- **Limitation:** Environment secrets must be configured in the reusable workflow's environment; they cannot be passed from the caller via `workflow_call`

## 4. Orchestration & features

- **Reusable Workflow:** Defines the entire job; can enforce runner labels; supports full workflow features

- **Composite Action:** Defines a list of steps; gives flexibility to run before/after other steps

# Choosing Between Them

## Use a Composite Action when...

- You need **reusable step sequences** (setup toolchain, cache, install, standard logging wrapper)
- The logic fits naturally inside one job and should share the caller job's environment and runner

## Use a Reusable Workflow when...

- You need to **standardize an entire pipeline pattern** across repos (build+test → security scanning → deploy jobs)
- You need strongly typed inputs and secrets, and want to keep workflows centrally maintained
- You need **job-level structure** (multiple jobs, job dependencies, distinct runners)

# Composite Actions: Advanced Demo

## DEMO 6: Action with Inputs & Outputs

```
inputs:
  node-version:
    description: 'Node version'
    required: true
    default: '18'
outputs:
  cache-hit:
    description: 'Whether cache was hit'
    value: ${{ steps.cache.outputs.cache-hit }}
runs:
  using: 'composite'
  steps:
    - id: cache
      uses: actions/cache@v3
```

# 4. Environment Protection Rules

---

## Why Environment Protection?

- Control deployments to sensitive environments
- Require manual approvals before deployment
- Restrict which branches can deploy
- Set wait timers before deployment
- Limit deployment reviewers

# Environment Protection: Simple Demo

## DEMO 7: Basic Environment

```
jobs:
  deploy-staging:
    runs-on: ubuntu-latest
    environment:
      name: staging
      url: https://staging.example.com
    steps:
      - run: echo "Deploying to staging"
```

**Environment configuration:** Set up in Settings → Environments → New environment

# Environment Protection: Advanced Demo

## DEMO 8: Production with Approvals

```
jobs:
  deploy-production:
    runs-on: ubuntu-latest
    environment:
      name: production
    steps:
      - run: echo "Deploying to production"
```

### Protection rules (configured in Settings):

- ✓ Required reviewers (up to 6 reviewers)
- ✓ Wait timer (0-43,200 minutes)
- ✓ Deployment branches (selected branches only)

# 5. Caching & Artifacts

---

## Caching vs Artifacts

### Caching

- Speed up workflow runs
- Store dependencies
- Shared across workflow runs
- Automatically evicted after 7 days

### Artifacts

- Share data between jobs
- Store build outputs
- Available for download
- Retained for 90 days (default)

# Caching & Artifacts: Simple Demo

## DEMO 9: Basic Caching

```
- uses: actions/cache@v3
  with:
    path: ~/.npm
    key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
    restore-keys: |
      ${{ runner.os }}-node-

- run: npm ci
```

**Key generation:** Include dependencies in key to invalidate when changed

# Caching & Artifacts: Advanced Demo

## DEMO 10: Multi-stage with Artifacts

```
jobs:
  build:
    steps:
      - uses: actions/cache@v3
        # ... cache setup
      - run: npm run build
      - uses: actions/upload-artifact@v3
        with:
          name: build-output
          path: dist/

  deploy:
    needs: build
    steps:
      - uses: actions/download-artifact@v3
        with:
          name: build-output
```

# Best Practices

---

- **Matrix Builds:** Use fail-fast: false for CI, max-parallel to control costs
- **Reusable Workflows:** Version them, document inputs/outputs thoroughly
- **Composite Actions:** Keep them focused and single-purpose
- **Environments:** Always use for production, consider for staging
- **Caching:** Include dependency lock files in cache key
- **Artifacts:** Set appropriate retention periods to manage storage

# Questions & Answers

Thank you!

All demos are in the repository:

`github.com/seandorsett/github-actions-advanced-features`

# Additional Resources

---

→ GitHub Actions Documentation:

`docs.github.com/actions`

→ Actions Marketplace: `github.com/marketplace?`

`type=actions`

→ GitHub Actions Community: `github.community`

→ This Repository: All demos runnable and documented

**Next Steps:** Try combining these features in your own projects. Start small, iterate, and build powerful CI/CD pipelines!

GitHub Actions Advanced Features • 2026 • All demos available in repository