{JS}
MASTERY

YAML & GitHub Actions CI/CD Cheatsheet

Created by **JS Mastery**
Visit ***jsmastery.pro*** for more

What's in the guide?

If you've ever wrestled with YAML syntax, stared at a broken GitHub Action, or just wanted a quick reminder of how CI/CD pipelines fit together, this guide is for you.

This cheat sheet keeps things practical.

It covers the essentials of YAML, walks through GitHub Actions from the ground up, and gives you ready-to-use examples for real pipelines.

No fluff, no jargon, just what you need to get things working.

Use it as a quick refresher when you forget the difference between `|` and `>`, need to set up a matrix build, or want to remember how to trigger a workflow only on main.

Think of it as your go-to reference while coding, debugging, or setting up automation.

...before you go

While the YAML Cheatsheet gives you quick wins for writing cleaner configs and smoother CI/CD pipelines, imagine taking those skills further by actually building and deploying full backend systems.

If you're eager to dive deep into backend development from APIs and databases to authentication and scaling our upcoming **Backend Course** has got you covered.

It's the perfect combination of strong foundations, taking you from core networking concepts all the way to building and deploying APIs with full DevOps and AWS integration.

Join now, from the link above 👆

YAML Basics

Key-Value Pairs

Basic YAML syntax for storing configuration values. Space after colon is mandatory.

```yaml
name: "My App"
version: 1.0
debug: true
port: 3000
```

Scalars

Just values like strings, integers, floats, booleans, etc.

```yaml
# String
title: "Hello World"

# Number
port: 8080

# Boolean
debug: false

# Null
value: null   # or just ~
```

YAML Basics

Comments

Use # for documentation. Comments are ignored by YAML parser.

```yaml
# This is a comment
name: value  # Inline comment
```

Multi-line Strings

| preserves line breaks, > folds lines into single string with spaces.

```yaml
# Literal block (preserves newlines)
script: |
  echo "Line 1"
  echo "Line 2"

# Folded block (joins lines)
description: >
  This is a long
  description that
  spans multiple lines
```

YAML Basics

Arrays (Lists)

Two ways to define lists. Block style is more readable for multiple items.

```yaml
# Method 1: Block style
fruits:
  - apple
  - banana
  - orange

# Method 2: Flow style
colors: [red, green, blue]
```

Objects (Maps)

Nested key-value pairs for complex configurations. Use consistent 2-space indentation.

```yaml
database:
  host: localhost
  port: 5432
  credentials:
    username: admin
    password: secret123
```

YAML Basics

Anchor (&)

Lets you define a reusable block of config once.

```yaml
# Define an anchor with "&"
defaults: &defaults
  image: node:18
  cache: true
  retries: 3
```

Alias (*)

Lets you reference and reuse that block anywhere else in the YAML.

```yaml
# Use an alias with "*"
build:
  <<: *defaults
  script: npm run build
```

🚨 Common Mistakes

Missing space after colon:

YAML parser requires space between key and value.

```
❌ name:value
✅ name: value
```

Mixed indentation:

Consistent indentation is critical. Stick to 2 spaces throughout.

```
❌ items:
     - first     # 5 spaces
    - second     # 3 spaces

✅ items:
  - first       # 2 spaces
  - second      # 2 spaces
```

🚨 Common Mistakes

Unquoted special characters:

Apostrophes and special characters need quotes to avoid parsing errors.

```
❌ message: It's working!
✅ message: "It's working!"
```

Tabs instead of spaces:

YAML requires spaces for indentation. Tabs will cause syntax errors.

```
❌ name:→→value     # Tabs
✅ name:  value     # Spaces
```

GitHub Actions Keywords

`name`

Workflow/job title

`on`

Defines triggers (push, PR, schedule)

`jobs`

Defines jobs, their OS, and steps

`steps`

Sequential commands or actions

`run`

Shell commands to execute

`uses`

Use prebuilt actions

GitHub Actions Keywords

`with`

Pass params to actions

`env`

Set environment variables

`needs`

Make one job depend on another

`if`

Specifies a conditional expression

`outputs`

Defines output parameters for jobs

`concurrency`

Manages the number of concurrent workflow or job runs

GitHub Actions Structure

Basic Workflow Template

Minimal workflow structure. Every workflow needs name, trigger, jobs, and steps.

```yaml
name: My Workflow
on: [push]
jobs:
  my-job:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo "Hello World"
```

Workflow Name

Optional display name shown in GitHub Actions tab. Defaults to filename if omitted.

```yaml
name: "CI Pipeline"
```

GA → *Event Triggers*

Push events:

Triggers workflow on every push to any branch.

```
on: push
```

Multiple events:

Run workflow on multiple event types. Most common combination.

```
on: [push, pull_request]
```

Manual trigger:

Adds "Run workflow" button in GitHub UI for manual execution.

```
on: workflow_dispatch
```

GA → *Event Triggers*

Branch-specific:

Only trigger on pushes to specific branches. Saves CI resources.

```yaml
on:
  push:
    branches: [main, develop]
```

Path filtering:

Only run when specific files or directories change. Perfect for monorepos.

```yaml
on:
  push:
    paths:
      - 'src/**'
      - '**.js'
```

GA → *Event Triggers*

Tag releases:

Trigger on version tags like v1.0.0. Common for release workflows.

```yaml
on:
  push:
    tags: [v*]
```

Pull request events:

Specific PR events. synchronize means new commits pushed to PR.

```yaml
on:
  pull_request:
    types: [opened, synchronize, reopened]
```

GA → *Event Triggers*

Manual with inputs:

Manual trigger with form inputs. Great for deployment workflows.

```yaml
on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Deployment environment'
        required: true
        default: 'staging'
        type: choice
        options: ['staging', 'production']
```

Scheduled runs:

Cron-based scheduling. Format: minute hour day month weekday.

```yaml
on:
  schedule:
    - cron: '0 9 * * 1-5'  # 9 AM weekdays
```

GA → *Jobs Structure*

Basic job:

Single job with one step. Jobs run in parallel by default.

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - run: npm test
```

Multiple jobs:

Sequential execution using needs. Build waits for test to complete.

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - run: npm test

  build:
    runs-on: ubuntu-latest
    needs: test  # Runs after test
    steps:
      - run: npm run build
```

GA → *Jobs Structure*

Job dependencies:

Complex dependencies with conditions. Deploy only runs on main branch after both jobs pass.

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    steps: [...]

  deploy:
    runs-on: ubuntu-latest
    needs: [test, build]  # Wait for both
    if: github.ref == 'refs/heads/main'
    steps: [...]
```

GitHub Actions Structure

Runner Types

GitHub-hosted vs self-hosted runners. Latest versions get security updates.

```yaml
runs-on: ubuntu-latest    # Linux
runs-on: ubuntu-22.04     # Specific version
runs-on: windows-latest   # Windows
runs-on: macos-latest     # macOS
runs-on: self-hosted      # Your own runner
```

Job Settings

Job-level configuration. Timeout prevents hanging jobs, continue-on-error allows optional jobs.

```yaml
jobs:
  test:
    runs-on: ubuntu-latest
    timeout-minutes: 30         # Max runtime
    continue-on-error: true     # Don't fail workflow
    if: github.event_name == 'push'  # Conditional
```

GA → Matrix

Matrix with exclusions:

Skip specific combinations. Useful when certain versions don't work together.

```yaml
strategy:
  matrix:
    os: [ubuntu-latest, windows-latest]
    node: [16, 18]
    exclude:
      - os: windows-latest
        node: 16
```

Matrix Strategy

Runs job for each combination (3×2=6 jobs). Great for testing compatibility.

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

GA → Steps Types

Using actions

Pre-built actions from marketplace. with passes parameters to action.

```yaml
- name: Checkout code
  uses: actions/checkout@v4
  with:
    fetch-depth: 0
```

Multi-line scripts:

Use | for multi-line scripts. Each line executes in same shell session.

```yaml
- name: Build and test
  run: |
    npm run build
    npm test
    echo "Done!"
```

GA → Steps Types

Running commands:

Shell commands. Default shell is bash on Linux/macOS, PowerShell on Windows.

```
- name: Install deps
  run: npm install
```

Working directory:

Change execution directory for specific steps. Useful for monorepos.

```
- name: Run in subfolder
  run: npm install
  working-directory: ./frontend
```

GA → Steps Types

Custom shell:

Override default shell. Options: bash, pwsh, python, sh, cmd, powershell.

```
- name: PowerShell script
  run: Get-Date
  shell: powershell
```

GA → Environment Var

Workflow level:

Global variables available to all jobs. Set once, use everywhere.

```yaml
env:
  NODE_ENV: production

jobs:
  test:
    steps: [...]
```

Job level:

Job-specific variables. Override global variables if needed.

```yaml
jobs:
  test:
    env:
      DATABASE_URL: postgres://...
    steps: [...]
```

GA → Environment Var

Step level:

Step-specific variables. Highest precedence, overrides job and workflow variables.

```yaml
- name: Deploy
  run: echo $API_URL
  env:
    API_URL: https://api.example.com
```

Secrets & Variables

Variables are public, secrets are encrypted. GITHUB_TOKEN is automatically provided.

```yaml
- name: Deploy
  run: |
    echo "Public var: ${{ vars.API_URL }}"
    echo "Secret: ${{ secrets.API_KEY }}"
  env:
    TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Reusable Components

Reusable Workflow

Reusable workflows reduce duplication across repositories. Define once, call many times.

```yaml
# .github/workflows/reusable-build.yml
name: Reusable Build
on:
  workflow_call:
    inputs:
      node-version:
        required: true
        type: string
    outputs:
      build-id:
        value: ${{ jobs.build.outputs.build-id }}

jobs:
  build:
    runs-on: ubuntu-latest
    outputs:
      build-id: ${{ steps.build.outputs.id }}
    steps:
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ inputs.node-version }}
```

Reusable Components

Using reusable workflow:

Call reusable workflow like any other job. Pass inputs with with parameter.

```yaml
jobs:
  call-build:
    uses: ./.github/workflows/reusable-build.yml
    with:
      node-version: '18'
```

Reusable Components

Composite Actions

Custom actions for repetitive step sequences. Store in .github/actions/ directory.

```yaml
# .github/actions/setup-app/action.yml
name: 'Setup App'
description: 'Setup Node.js and install dependencies'
inputs:
  node-version:
    description: 'Node.js version'
    required: true
    default: '18'
runs:
  using: 'composite'
  steps:
    - uses: actions/setup-node@v4
      with:
        node-version: ${{ inputs.node-version }}
    - run: npm ci
      shell: bash
```

Reusable Components

Using composite action:

Use custom actions like marketplace actions. Great for standardizing setup steps.

```
- uses: ./.github/actions/setup-app
  with:
    node-version: '20'
```

Security & Permissions

Workflow Permissions

Limit workflow permissions to minimum required. Follows principle of least privilege.

```yaml
permissions:
  contents: read        # Clone repo
  pull-requests: write # Comment on PRs
  issues: write         # Create issues
  actions: read         # Read workflow runs
```

Environment Protection

Environment protection rules require manual approval before deployment. Configure in repo settings.

```yaml
jobs:
  deploy:
    environment: production  # Requires approval
    runs-on: ubuntu-latest
```

Security & Permissions

OIDC Authentication

Modern authentication without long-lived secrets. More secure than access keys.

```yaml
permissions:
  id-token: write
  contents: read

steps:
  - uses: aws-actions/configure-aws-credentials@v4
    with:
      role-to-assume: arn:aws:iam::12345:role/GitHubActions
      aws-region: us-east-1
```

CI/CD Pipeline Examples

Node.js/React Pipeline

Complete Node.js pipeline with linting, testing, and coverage. Matrix tests multiple Node versions.

```yaml
name: Node.js CI/CD
on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [16, 18, 20]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node-version }}
          cache: 'npm'
      - run: npm ci
      - run: npm run lint
      - run: npm test -- --coverage
      - uses: codecov/codecov-action@v3
```

CI/CD Pipeline Examples

Docker Build Pipeline

```yaml
name: Docker
on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to registry
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and push
        uses: docker/build-push-action@v5
        with:
          push: true
          tags: ghcr.io/${{ github.repository }}:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

Artifacts & Caching

Upload Artifacts

Store build outputs between jobs or for download. Artifacts expire after retention period.

```yaml
- name: Upload build files
  uses: actions/upload-artifact@v3
  with:
    name: build-files
    path: dist/
    retention-days: 30
```

Download Artifacts

Retrieve artifacts from previous jobs or workflow runs. Useful for deployment jobs.

```yaml
- name: Download build
  uses: actions/download-artifact@v3
  with:
    name: build-files
    path: ./build
```

Artifacts & Caching

Cache Dependencies

Cache dependencies to speed up builds. Key based on lock file hash for cache invalidation.

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

Cache Multiple Paths

Cache multiple dependency managers simultaneously. Key updates when any lock file changes.

```
- uses: actions/cache@v3
  with:
    path: |
      ~/.npm
      ~/.cache/pip
    key: ${{ runner.os }}-deps-${{ hashFiles('**/package-
lock.json', '**/requirements.txt') }}
```

Deployment Examples

Deploy to Vercel

Deploy to Vercel platform. Great for Next.js and React applications.

```yaml
- name: Deploy to Vercel
  uses: amondnet/vercel-action@v25
  with:
    vercel-token: ${{ secrets.VERCEL_TOKEN }}
    vercel-org-id: ${{ secrets.ORG_ID }}
    vercel-project-id: ${{ secrets.PROJECT_ID }}
    vercel-args: '--prod'
```

Deployment Examples

Deploy to Netlify

Deploy static sites to Netlify. Creates preview deploys for PRs, production for main branch.

```yaml
- name: Deploy to Netlify
  uses: nwtgck/actions-netlify@v2
  with:
    publish-dir: './dist'
    production-branch: main
    deploy-message: "Deploy from GitHub Actions"
  env:
    NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
    NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
```

Deployment Examples

Deploy to AWS S3

Deploy static site to S3 with CloudFront cache invalidation. --delete removes old files.

```
- name: Configure AWS
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{
secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: us-east-1

- name: Deploy to S3
  run: |
    aws s3 sync ./dist s3://${{ secrets.S3_BUCKET }} --
delete
    aws cloudfront create-invalidation --distribution-id
${{ secrets.CLOUDFRONT_ID }} --paths "/*"
```

Deployment Examples

Deploy to Azure

Deploy web applications to Azure App Service. Supports multiple runtime stacks.

```
- name: Azure Login
  uses: azure/login@v1
  with:
    creds: ${{ secrets.AZURE_CREDENTIALS }}

- name: Deploy to Azure Web App
  uses: azure/webapps-deploy@v2
  with:
    app-name: 'my-web-app'
    package: './dist'
```

Deployment Examples

Deploy to Google Cloud

Deploy containerized apps to Google Cloud Run. Serverless container platform.

```
- name: Setup GCloud
  uses: google-github-actions/setup-gcloud@v1
  with:
    service_account_key: ${{ secrets.GCP_SA_KEY }}
    project_id: ${{ secrets.GCP_PROJECT_ID }}

- name: Deploy to Cloud Run
  run: |
    gcloud run deploy my-service \
      --image gcr.io/${{ secrets.GCP_PROJECT_ID }}/my-
app:latest \
      --region us-central1 \
      --platform managed
```

Monitoring & Notifications

Status Checks

Verify deployment success with health checks. Non-zero exit codes fail the step.

```
- name: Health check
  run: curl -f http://localhost:3000/health || exit 1
```

Slack Notifications

Send Slack messages on workflow events. if: failure() only runs when previous steps fail.

```
- name: Slack notification
  if: failure()
  uses: rtCamp/action-slack-notify@v2
  env:
    SLACK_WEBHOOK: ${{ secrets.SLACK_WEBHOOK }}
    SLACK_MESSAGE: 'Build failed!'
```

Monitoring & Notifications

Email Notifications

Email notifications for critical failures. Configure SMTP settings in secrets.

```yaml
- name: Send email
  if: failure()
  uses: dawidd6/action-send-mail@v3
  with:
    server_address: smtp.gmail.com
    server_port: 587
    username: ${{ secrets.EMAIL_USERNAME }}
    password: ${{ secrets.EMAIL_PASSWORD }}
    subject: Build Failed
    body: The build has failed for ${{ github.repository }}
    to: developer@company.com
```

Monitoring & Notifications

Create Issues on Failure

Automatically create GitHub issues when builds fail. Great for tracking recurring problems.

```yaml
- name: Create issue
  if: failure()
  uses: actions/github-script@v6
  with:
    script: |
      github.rest.issues.create({
        owner: context.repo.owner,
        repo: context.repo.repo,
        title: 'Build failed',
        body: 'Build failed for commit ${{ github.sha }}'
      })
```

Debugging Tips

Debug Logging

Enable verbose logging for troubleshooting. Set as repository secrets.

```
# Enable in repository secrets
ACTIONS_STEP_DEBUG: true
ACTIONS_RUNNER_DEBUG: true
```

Print Context

Print all available context variables for debugging. Use toJSON() to see structure.

```
- name: Dump context
  run: |
    echo "GitHub context:"
    echo '${{ toJSON(github) }}'
    echo "Runner context:"
    echo '${{ toJSON(runner) }}'
```

Debugging Tips

SSH Debug Access

Get SSH access to failed runner for interactive debugging. Use sparingly for security.

```yaml
- name: Setup tmate session
  if: failure()
  uses: mxschmitt/action-tmate@v3
```

Security Checklist

- ✅ Use environment protection for production
- ✅ Limit workflow permissions
- ✅ Pin action versions (use full SHA)
- ✅ Review third-party actions
- ✅ Use OIDC instead of long-lived secrets
- ✅ Validate inputs in reusable workflows
- ✅ Use **`GITHUB_TOKEN`** for GitHub API calls

💡 Pro tip

Start with simple workflows and gradually add complexity. Test locally with **`act`** or **`github-cli`** when possible!

The End

Remember: CI/CD is a Journey, Not a Destination

Every great deployment pipeline started with a simple workflow. Don't try to implement everything at once, build incrementally, learn from failures, and always prioritize security over convenience.

Your future self (and your team) will thank you for the time invested in proper automation.

Happy automating! 🚀

P.S. If this cheatsheet got you excited about building smarter systems, you'll love our upcoming [Ultimate Backend Course]. Join the waitlist to get early access.