CI/CD Cheatsheet

CI/CD Cheatsheet

A practical quick-reference guide for the two most popular CI/CD platforms. Covers workflow syntax, triggers, secrets, artifacts, caching, matrix builds, and real-world deployment patterns for both GitHub Actions and GitLab CI.

GitHub Actions

GitHub Actions is the built-in CI/CD platform for GitHub repositories. Workflows are defined in YAML files under `.github/workflows/` and triggered by repository events, schedules, or manual dispatch.

Workflow Structure

Every workflow lives in `.github/workflows/<name>.yml` and requires a `name`, `on` (trigger), and at least one `job` with `steps`.

```
name: CI Pipeline

# Triggers — what starts this workflow
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  schedule:
    - cron: "0 6 * * 1"  # Every Monday at 06:00 UTC
  workflow_dispatch:       # Manual trigger from GitHub UI

# Global environment variables
env:
  NODE_VERSION: "20"
  REGISTRY: ghcr.io

jobs:
  build-and-test:
    name: Build & Test
    runs-on: ubuntu-latest
    timeout-minutes: 15

    # Job-level environment variables
```

```
    env:
      APP_DIR: ./apps/web

    # Job-level permissions
    permissions:
      contents: read
      packages: write

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: npm

      - name: Install dependencies
        run: npm ci

      - name: Run linter
        run: npm run lint

      - name: Run tests
        run: npm run test -- --coverage

      - name: Build application
        run: npm run build
```

Triggers

Control when workflows run with fine-grained event filters.

```
on:
  # Push to specific branches or tags
  push:
    branches:
      - main
      - "release/**"
    tags:
      - "v*"

  # Pull request targeting specific branches
  pull_request:
    types: [opened, synchronize, reopened]
    branches: [main]

  # Run on schedule (cron syntax, UTC)
  schedule:
    - cron: "0 0 * * *"   # Daily at midnight
    - cron: "0 6 * * 1"   # Every Monday at 06:00

  # Manual trigger with inputs
  workflow_dispatch:
    inputs:
      environment:
        description: "Deployment environment"
```

```
        required: true
        default: staging
        type: choice
        options:
          - staging
          - production
      debug_mode:
        description: "Enable debug logging"
        required: false
        type: boolean
        default: false

  # Trigger on workflow completion
  workflow_run:
    workflows: ["CI Pipeline"]
    types: [completed]
    branches: [main]

  # Trigger on issues or other events
  issues:
    types: [opened, labeled]
  release:
    types: [published]
```

Using Actions from the Marketplace

Reference community or first-party actions with `uses`. Pin by tag or SHA (SHA is recommended for security).

```
steps:
  # First-party GitHub Actions
  - uses: actions/checkout@v4              # Clone repo
  - uses: actions/setup-node@v4            # Install Node.js
  - uses: actions/setup-python@v5          # Install Python
  - uses: actions/setup-java@v4            # Install JDK
  - uses: actions/setup-go@v5              # Install Go
  - uses: actions/cache@v4                 # Cache dependencies
  - uses: actions/upload-artifact@v4       # Upload build artifacts
  - uses: actions/download-artifact@v4     # Download artifacts
  - uses: actions/labeler@v5               # Auto-label PRs
  - uses: actions/create-release@v1        # Create GitHub release
  - uses: actions/configure-pages@v5       # Setup GitHub Pages
  - uses: actions/deploy-pages@v4          # Deploy to Pages

  # Third-party (pin by SHA for security)
  - uses: docker/login-action@v3           # Log in to container registry
    with:
      registry: ghcr.io
      username: ${{ github.actor }}
      password: ${{ secrets.GITHUB_TOKEN }}

  - uses: docker/build-push-action@v5      # Build and push Docker image
    with:
      context: .
      push: true
      tags: ${{ env.REGISTRY }}/${{ github.repository }}:${{ github.sha }}

  - uses: docker/setup-buildx-action@v3    # Setup BuildKit
```

```
  # Reference a local action in the same repo
  - uses: ./.github/actions/custom-action
```

Environment Variables and Secrets

GitHub provides context expressions, repository secrets, and environment variables at multiple scopes.

```
on: push

env:
  # Global for all jobs
  APP_NAME: myapp
  NODE_ENV: production

jobs:
  deploy:
    runs-on: ubuntu-latest
    environment: production  # Required environment with protection rules

    env:
      # Job-level variable
      DEPLOY_PATH: /var/www/${{ env.APP_NAME }}

    steps:
      # Built-in environment variables (always available)
      - name: Show built-in vars
        run: |
          echo "Home: $HOME"
          echo "Runner OS: $RUNNER_OS"
          echo "Runner Arch: $RUNNER_ARCH"
          echo "Workspace: $GITHUB_WORKSPACE"
          echo "Event: $GITHUB_EVENT_NAME"
          echo "SHA: $GITHUB_SHA"
          echo "Ref: $GITHUB_REF"

      # Use repository secrets (set in Settings > Secrets)
      - name: Deploy with credentials
        env:
          DB_PASSWORD: ${{ secrets.DB_PASSWORD }}
          API_KEY: ${{ secrets.API_KEY }}
          # GITHUB_TOKEN is automatically available
          GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          echo "Deploying to $DEPLOY_PATH"
          # Use variables safely — never echo secrets
          deploy.sh --password "$DB_PASSWORD"

      # Use context expressions
      - name: Use context data
        run: |
          echo "Actor: ${{ github.actor }}"
          echo "Repo: ${{ github.repository }}"
          echo "Branch: ${{ github.ref_name }}"
          echo "Event: ${{ github.event_name }}"

      # Use environment-specific secrets
      - name: Use environment secret
        env:
```

```
          PROD_API_KEY: ${{ secrets.PROD_API_KEY }}
        run: echo "Secret length: ${#PROD_API_KEY}"

      # Multi-line environment variable
      - name: Set multi-line variable
        env:
          SSH_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
        run: |
          mkdir -p ~/.ssh
          echo "$SSH_KEY" > ~/.ssh/deploy_key
          chmod 600 ~/.ssh/deploy_key
```

Artifacts and Caching

Share data between jobs and speed up workflows by caching dependencies.

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

      # Cache npm dependencies
      - uses: actions/cache@v4
        with:
          path: ~/.npm
          key: ${{ runner.os }}-npm-${{ hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-npm-

      - run: npm ci
      - run: npm run build

      # Upload build output as artifact
      - uses: actions/upload-artifact@v4
        with:
          name: build-output
          path: dist/
          retention-days: 14
          if-no-files-found: error

      # Upload test results
      - uses: actions/upload-artifact@v4
        with:
          name: test-results
          path: coverage/
          retention-days: 7

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
      # Download artifact from build job
      - uses: actions/download-artifact@v4
        with:
          name: build-output
          path: dist/
```

```
      - name: Deploy
        run: deploy.sh dist/

      # Upload deployment logs as artifact
      - uses: actions/upload-artifact@v4
        if: always()  # Upload even if previous steps fail
        with:
          name: deploy-logs
          path: logs/
```

Matrix Builds

Run jobs across multiple versions, platforms, or configurations in parallel.

```
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      # Don't cancel in-progress jobs if one fails
      fail-fast: false

      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        node-version: [18, 20, 22]
        # Exclude specific combinations
        exclude:
          - os: windows-latest
            node-version: 18
        # Include additional combinations
        include:
          - os: ubuntu-latest
            node-version: 22
            experimental: true
            label: "Node 22 (experimental)"

    name: Test on Node ${{ matrix.node-version }} (${{ matrix.os }})
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node-version }}
      - run: npm ci
      - run: npm test
      - run: npm run test:integration
        if: matrix.experimental
```

Reusable Workflows

Call workflows from other workflows to share CI logic across repositories.

```
# .github/workflows/reusable-build.yml (the reusable workflow)
name: Reusable Build

on:
  workflow_call:
    inputs:
```

```
      node-version:
        required: false
        type: string
        default: "20"
      build-command:
        required: false
        type: string
        default: "npm run build"
    secrets:
      registry-token:
        required: true

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ inputs.node-version }}
      - run: npm ci
      - run: ${{ inputs.build-command }}
```

```
# .github/workflows/ci.yml (the caller)
name: CI

on:
  push:
    branches: [main]

jobs:
  call-build:
    uses: ./.github/workflows/reusable-build.yml
    with:
      node-version: "22"
      build-command: "npm run build:prod"
    secrets:
      registry-token: ${{ secrets.GITHUB_TOKEN }}

  # Call from another repository
  call-external:
    uses: myorg/shared-actions/.github/workflows/deploy.yml@v2
    with:
      environment: staging
    secrets:
      deploy-key: ${{ secrets.STAGING_DEPLOY_KEY }}
```

Self-Hosted Runners

Run workflows on your own infrastructure instead of GitHub-hosted runners.

```
jobs:
  build-on-prem:
    runs-on: [self-hosted, linux, x64]
    # Or use a custom label
    # runs-on: self-hosted-gpu
```

```
    steps:
      - uses: actions/checkout@v4
      - run: |
          echo "Running on self-hosted runner"
          echo "Runner name: $RUNNER_NAME"
          echo "OS: $(uname -a)"

  # Use runner groups (Enterprise/Team)
  deploy-production:
    runs-on:
      - self-hosted
      - linux
      - production  # Custom label for prod runners
    environment: production
    steps:
      - uses: actions/checkout@v4
      - run: ./deploy.sh
```

Complete Real-World Example: Node.js CI/CD

```
name: Node.js CI/CD

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  contents: read
  packages: write
  id-token: write

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  lint:
    name: Lint
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: "20"
          cache: npm
      - run: npm ci
      - run: npm run lint
      - run: npm run typecheck

  test:
    name: Test
    needs: lint
```

```yaml
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        node-version: [18, 20, 22]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node-version }}
          cache: npm
      - run: npm ci
      - run: npm run test:unit -- --coverage
      - uses: actions/upload-artifact@v4
        if: matrix.node-version == 20
        with:
          name: coverage-report
          path: coverage/

  build:
    name: Build & Push Image
    needs: test
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    outputs:
      image-tag: ${{ steps.meta.outputs.version }}
    steps:
      - uses: actions/checkout@v4

      - name: Log in to Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=sha,prefix=
            type=ref,event=branch
            type=semver,pattern={{version}}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

  deploy-staging:
    name: Deploy to Staging
    needs: build
```

```
    runs-on: ubuntu-latest
    environment: staging
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to staging
        run: |
          echo "Deploying image tag: ${{ needs.build.outputs.image-tag }}"
          # kubectl set image deployment/web app=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{
          # kubectl rollout status deployment/web

  deploy-production:
    name: Deploy to Production
    needs: [build, deploy-staging]
    runs-on: ubuntu-latest
    environment: production
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to production
        run: |
          echo "Deploying image tag: ${{ needs.build.outputs.image-tag }}"
          # kubectl set image deployment/web app=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{
          # kubectl rollout status deployment/web
```

GitLab CI

GitLab CI is the built-in CI/CD system for GitLab. Pipelines are defined in `.gitlab-ci.yml` at the repository root and executed by GitLab runners. Jobs are organized into stages that run sequentially, with jobs within each stage running in parallel.

Pipeline Structure

The `.gitlab-ci.yml` file defines stages, jobs, and their execution order.

```
# Default settings applied to all jobs
default:
  image: node:20-alpine
  before_script:
    - npm ci --cache .npm
    - npm ci
  cache:
    key:
      files:
        - package-lock.json
    paths:
      - .npm/
  artifacts:
    expire_in: 7 days

# Global variables
variables:
  npm_config_cache: "$CI_PROJECT_DIR/.npm"
  DOCKER_DRIVER: overlay2

# Stages execute in order; jobs within a stage run in parallel
stages:
```

```
  - lint
  - test
  - build
  - deploy

lint:
  stage: lint
  script:
    - npm run lint
    - npm run typecheck

test:unit:
  stage: test
  script:
    - npm run test:unit -- --coverage
  coverage: '/All files[^|]*\|[^|]*\s+([\d.]+)/'
  artifacts:
    when: always
    paths:
      - coverage/

test:integration:
  stage: test
  script:
    - npm run test:integration
  needs: ["lint"]

build:
  stage: build
  script:
    - npm run build
  artifacts:
    paths:
      - dist/
    expire_in: 30 days

deploy:staging:
  stage: deploy
  script:
    - echo "Deploying to staging..."
  environment: staging
  only:
    - main

deploy:production:
  stage: deploy
  script:
    - echo "Deploying to production..."
  environment: production
  when: manual
  only:
    - main
```

Runners

GitLab runners execute jobs. Use shared runners, group runners, or specific project runners with tags.

```
# Use shared runners (available to all projects)
job-on-shared-runner:
  stage: test
  script:
    - echo "Running on shared runner"

# Use a specific runner by tag
job-on-specific-runner:
  stage: test
  tags:
    - docker
    - linux
  script:
    - echo "Running on tagged runner"

# Use a self-hosted runner with custom Docker image
job-on-custom-runner:
  stage: deploy
  tags:
    - production
    - kubernetes
  image: alpine:latest
  script:
    - kubectl apply -f k8s/
```

Variables and Secrets

Define variables at multiple scopes with different protection levels.

```
# Global variables (visible in pipeline logs — not for secrets)
variables:
  APP_NAME: myapp
  APP_ENV: production
  DEPLOY_REGION: us-east-1

# File-type variable (useful for certificates, config files)
variables:
  KUBE_CONFIG:  # Set as File type in GitLab UI
  SSL_CERT:     # Set as File type in GitLab UI

build:
  stage: build
  variables:
    # Job-level override
    NODE_ENV: production
  script:
    # CI/CD variables (set in Settings > CI/CD > Variables)
    # Protected: only available on protected branches/tags
    # Masked: hidden in job logs
    # Type: Variable or File
    - echo "Deploying $APP_NAME to $DEPLOY_REGION"
    - echo "DB host: $DB_HOST"         # Variable type
    - echo "Config at $APP_CONFIG"      # File type (path to temp file)
    - cat "$APP_CONFIG"                 # Read file-type variable
    - |
      # Use predefined CI/CD variables
      echo "Pipeline ID: $CI_PIPELINE_ID"
```

```
      echo "Commit SHA: $CI_COMMIT_SHA"
      echo "Branch: $CI_COMMIT_BRANCH"
      echo "Default branch: $CI_DEFAULT_BRANCH"
      echo "Project path: $CI_PROJECT_PATH"
      echo "Runner tags: $CI_RUNNER_TAGS"

deploy:
  stage: deploy
  script:
    # Access protected variables (only on protected branches)
    - echo "$PRODUCTION_DB_PASSWORD" | docker login -u admin --password-stdin
  environment:
    name: production
  only:
    - main  # Protected branch — can access protected variables
```

Artifacts and Cache

Persist files between jobs (artifacts) and speed up jobs by caching dependencies (cache).

```
# Define artifacts — files passed between stages
build:
  stage: build
  script:
    - npm run build
  artifacts:
    name: "build-$CI_COMMIT_SHORT_SHA"
    paths:
      - dist/
      - build-report.json
    exclude:
      - dist/**/*.map
    expire_in: 2 weeks
    when: on_success      # on_success (default), on_failure, always
    expose_as: "Build Output"  # Download link label in UI

test:
  stage: test
  needs: [build]  # Start as soon as build finishes (don't wait for stage order)
  script:
    - npm run test
  artifacts:
    when: always        # Upload even if tests fail
    paths:
      - test-results/
    reports:
      # JUnit test report (shows results in MR)
      junit: test-results/junit.xml
      # Code coverage report
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
    # Artifact downloading
  dependencies:
    - build  # Only download artifacts from build job

# Cache — speed up job execution
.cache-template: &cache-npm
```

```
  cache:
    key:
      files:
        - package-lock.json
      prefix: npm
    paths:
      - .npm/
    policy: pull-push  # Default: read and update cache

install:
  stage: .pre
  <<: *cache-npm
  script:
    - npm ci

test:unit:
  stage: test
  <<: *cache-npm
  cache:
    <<: *cache-npm
    policy: pull  # Only read cache, don't update
  script:
    - npm run test:unit

# Docker layer caching
docker-build:
  stage: build
  image: docker:24
  services:
    - docker:24-dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  cache:
    key: "$CI_JOB_NAME"
    paths:
      - .docker-cache/
  before_script:
    - docker load -i .docker-cache/build.tar || true
  script:
    - docker build --cache-from myapp:latest -t myapp:$CI_COMMIT_SHA .
    - docker save myapp:$CI_COMMIT_SHA -o .docker-cache/build.tar
```

Rules, Only/Except, and Workflow

Control when jobs run using rules (preferred) or the legacy only/except keywords.

```
# Modern rules syntax (preferred)
test:
  script: npm test
  rules:
    # Run on main branch
    - if: $CI_COMMIT_BRANCH == "main"
    # Run on merge request pipelines
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    # Run on scheduled pipelines
    - if: $CI_PIPELINE_SOURCE == "schedule"
      variables:
        RUN_SLOW_TESTS: "true"
```

```
    # Run when a specific file changes
    - changes:
        - src/**/*
        - package.json
    # Run on tags matching pattern
    - if: $CI_COMMIT_TAG =~ /^v\d+\.\d+\.\d+$/
    # Run manually
    - if: $CI_COMMIT_BRANCH == "main"
      when: manual
      allow_failure: true
    # Default fallback (don't run if none of the above match)
    - when: never

# Complex rules with conditions
deploy:
  script: ./deploy.sh
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
      exists:
        - Dockerfile
      changes:
        - src/**/*
      variables:
        DEPLOY_ENV: production
    - if: $CI_COMMIT_BRANCH == "develop"
      variables:
        DEPLOY_ENV: staging

# Legacy only/except (still supported)
legacy-job:
  script: echo "Legacy syntax"
  only:
    - main
    - develop
    - /^release\/.*$/
    - merge_requests
    - tags
    - schedules
    - api
    - web  # Manual trigger from UI
  except:
    - feature/*
    - /^hotfix\/.*/
```

Environments

Define deployment environments with tracking, URLs, and protection rules.

```
variables:
  KUBE_NAMESPACE: ""

deploy:staging:
  stage: deploy
  script:
    - echo "Deploying to staging..."
    - kubectl apply -f k8s/staging/
  environment:
    name: staging
```

```
    url: https://staging.example.com
    on_stop: stop:staging  # Trigger cleanup job
  only:
    - develop

# Stop/truncate an environment
stop:staging:
  stage: deploy
  script:
    - echo "Stopping staging environment..."
    - kubectl delete namespace staging
  environment:
    name: staging
    action: stop  # Marks environment as stopped
  when: manual

deploy:production:
  stage: deploy
  script:
    - echo "Deploying to production..."
    - kubectl apply -f k8s/production/
  environment:
    name: production
    url: https://example.com
    # Auto-stop after 30 days of inactivity (requires Premium+)
    auto_stop_in: 30 days
  only:
    - main

deploy:review:
  stage: deploy
  script:
    - echo "Deploying review app for $CI_MERGE_REQUEST_IID..."
    - kubectl create namespace review-$CI_MERGE_REQUEST_IID || true
    - kubectl apply -f k8s/review/ -n review-$CI_MERGE_REQUEST_IID
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    url: https://$CI_ENVIRONMENT_SLUG.example.com
    on_stop: stop:review
  only:
    - merge_requests

stop:review:
  stage: deploy
  script:
    - kubectl delete namespace review-$CI_MERGE_REQUEST_IID
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    action: stop
  when: manual
  only:
    - merge_requests
```

Manual Jobs

Require human intervention before execution. Useful for production deployments and destructive operations.

```
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script: npm run build
  artifacts:
    paths: [dist/]

test:
  stage: test
  script: npm test

# Manual job — blocked until a user clicks "Play"
deploy:production:
  stage: deploy
  script:
    - ./deploy.sh production
  environment:
    name: production
  when: manual
  # Allow pipeline to succeed even if this job is skipped
  allow_failure: false

# Manual job with optional execution
deploy:canary:
  stage: deploy
  script:
    - ./deploy-canary.sh
  when: manual
  allow_failure: true  # Pipeline passes even if skipped

# Manual job triggered only on main
cleanup:
  stage: deploy
  script:
    - ./cleanup.sh
  when: manual
  only:
    - main
  allow_failure: true
```

Include, Extend, and Templates

Reuse configuration with `include`, share settings with `extends`, and create job templates.

```
# Include external configuration files
include:
  # File in same repository
  - local: .gitlab/ci/rules.yml
  # File in another repository
  - project: myorg/shared-pipelines
    ref: main
    file: /templates/nodejs.yml
  # Remote URL
```

```yaml
  - remote: https://example.com/ci-templates/base.yml
  # Template from GitLab's CI/CD catalog (GitLab 16+)
  - component: gitlab.com/gitlab-org/components/dotnet/dotnet-build@1.2.0
    inputs:
      stage: build
      dotnet_version: "8.0.x"

# YAML anchors and aliases
.cache-npm: &cache-npm
  cache:
    key:
      files: [package-lock.json]
    paths: [.npm/]

.install-deps: &install-deps
  before_script:
    - npm ci --cache .npm

# Extend templates
.test-base:
  <<: [*cache-npm, *install-deps]
  stage: test
  image: node:20-alpine

test:unit:
  extends: .test-base
  script:
    - npm run test:unit

test:integration:
  extends: .test-base
  script:
    - npm run test:integration
  services:
    - name: postgres:15
      alias: db

# Hidden job templates (prefixed with .) don't run on their own
.docker-base:
  image: docker:24
  services:
    - docker:24-dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY

build:api:
  extends: .docker-base
  stage: build
  script:
    - docker build -t $CI_REGISTRY_IMAGE/api:$CI_COMMIT_SHA ./apps/api
    - docker push $CI_REGISTRY_IMAGE/api:$CI_COMMIT_SHA

build:web:
  extends: .docker-base
  stage: build
  script:
```

```
    - docker build -t $CI_REGISTRY_IMAGE/web:$CI_COMMIT_SHA ./apps/web
    - docker push $CI_REGISTRY_IMAGE/web:$CI_COMMIT_SHA
```

Merge Request Pipelines

Special pipeline behavior for merge requests including MR-specific variables and workflow rules.

```
# Control which pipelines run for MRs vs branch pushes
workflow:
  rules:
    # Run merge request pipelines for branches with open MRs
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    # Run branch pipelines for the default branch
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
    # Run tag pipelines
    - if: $CI_COMMIT_TAG
    # Run scheduled pipelines
    - if: $CI_PIPELINE_SOURCE == "schedule"
    # Run API/web-triggered pipelines
    - if: $CI_PIPELINE_SOURCE == "api"
    - if: $CI_PIPELINE_SOURCE == "web"
    # Don't run branch pipelines for feature branches (use MR pipeline instead)
    - when: never

variables:
  # Only build the MR diff in MR pipelines
  GET_SOURCES_ATTEMPT: "3"

test:
  stage: test
  script:
    - echo "MR Title: $CI_MERGE_REQUEST_TITLE"
    - echo "MR Source: $CI_MERGE_REQUEST_SOURCE_BRANCH_NAME"
    - echo "MR Target: $CI_MERGE_REQUEST_TARGET_BRANCH_NAME"
    - echo "MR IID: $CI_MERGE_REQUEST_IID"
    - npm run test
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

lint:
  stage: test
  script:
    - npm run lint
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      # Only lint changed files
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

# Run only on merge request pipelines, not branch pipelines
review-deploy:
  stage: deploy
  script:
    - echo "Deploying review app for MR !${CI_MERGE_REQUEST_IID}"
  environment:
    name: review/$CI_COMMIT_REF_SLUG
  rules:
```

```
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: manual
```

Complete Real-World Example: Multi-Service Docker Pipeline

```
# .gitlab-ci.yml — Multi-service Docker CI/CD

include:
  - local: .gitlab/ci/variables.yml

default:
  image: docker:24
  services:
    - docker:24-dind
  before_script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --password-stdin "$C

variables:
  DOCKER_TLS_CERTDIR: "/certs"

stages:
  - validate
  - test
  - build
  - deploy

# ─── Validate ──────────────────────────────────────────

lint:
  stage: validate
  image: node:20-alpine
  before_script:
    - npm ci
  script:
    - npm run lint
    - npm run typecheck
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

# ─── Test ──────────────────────────────────────────────

test:unit:
  stage: test
  image: node:20-alpine
  services:
    - name: postgres:15-alpine
      alias: postgres
  variables:
    POSTGRES_DB: testdb
    POSTGRES_USER: testuser
    POSTGRES_PASSWORD: testpass
    DATABASE_URL: postgresql://testuser:testpass@postgres:5432/testdb
  before_script:
    - npm ci
    - npx prisma migrate deploy
  script:
    - npm run test:unit -- --ci --coverage
```

```
  coverage: '/All files[^|]*\|[^|]*\s+([\d.]+)/'
  artifacts:
    when: always
    reports:
      junit: test-results/junit.xml
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

# ─── Build ──────────────────────────────────────────────

.docker-build:
  stage: build
  variables:
    IMAGE: $CI_REGISTRY_IMAGE/$SERVICE
  script:
    - docker pull $IMAGE:latest || true
    - docker build
        --cache-from $IMAGE:latest
        --tag $IMAGE:$CI_COMMIT_SHA
        --tag $IMAGE:latest
        -f apps/$SERVICE/Dockerfile
        apps/$SERVICE
    - docker push $IMAGE:$CI_COMMIT_SHA
    - docker push $IMAGE:latest

build:api:
  extends: .docker-build
  variables:
    SERVICE: api
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      changes: [apps/api/**/*]

build:web:
  extends: .docker-build
  variables:
    SERVICE: web
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      changes: [apps/web/**/*]

build:worker:
  extends: .docker-build
  variables:
    SERVICE: worker
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      changes: [apps/worker/**/*]

# ─── Deploy ─────────────────────────────────────────────

.deploy-k8s:
  stage: deploy
  image: bitnami/kubectl:latest
  script:
```

```
    - kubectl config use-context $KUBE_CONTEXT
    - kubectl set image deployment/$SERVICE $SERVICE=$CI_REGISTRY_IMAGE/$SERVICE:$CI_COMMIT_S
    - kubectl rollout status deployment/$SERVICE -n $KUBE_NAMESPACE --timeout=300s

deploy:staging:
  extends: .deploy-k8s
  variables:
    KUBE_CONTEXT: staging
    KUBE_NAMESPACE: staging
    SERVICE: api
  environment:
    name: staging
    url: https://staging-api.example.com
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

deploy:production:
  extends: .deploy-k8s
  variables:
    KUBE_CONTEXT: production
    KUBE_NAMESPACE: production
    SERVICE: api
  environment:
    name: production
    url: https://api.example.com
  when: manual
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Quick Comparison

Feature	GitHub Actions	GitLab CI
Config location	`.github/workflows/*.yml`	`.gitlab-ci.yml`
Trigger syntax	`on: push, pull_request, schedule`	`only, except, rules`
Secrets scope	Repository, Environment, Org	Project, Group, Instance, Env
Artifact retention	Configurable per artifact	`expire_in` per artifact
Cache	`actions/cache` action	Built-in `cache` keyword
Matrix builds	`strategy.matrix`	`matrix` (parallel) or `parallel: N`
Reusable workflows	`uses` + `workflow_call`	`include` + `extends`
Environments	`environment:` keyword	`environment:` keyword
Manual jobs	`workflow_dispatch` + `when: manual` not native	`when: manual`
Self-hosted runners	`runs-on: self-hosted`	`tags:` on runners
Container support	`container:` in job	`image:` + `services:`
Merge request pipelines	`pull_request` trigger	`merge_request_event` source

Next Steps

- Docker CLI Cheat Sheet — Build and manage container images
- Kubernetes kubectl Cheat Sheet — Orchestrate containers at scale
- Helm CLI Cheat Sheet — Package and deploy Kubernetes applications
- Terraform Cheat Sheet — Infrastructure as code reference