Lab 3: Build and Optimize Your Application with GitHub Actions

Lab overview

In this lab, you will enhance your existing Lab 2 CI/CD pipeline by adding advanced build optimization techniques, caching strategies, and environment-specific workflows. You'll improve build performance, handle job failures gracefully, and develop advanced GitHub Actions YAML skills. This lab extends your Lab 2 project with practical optimizations used in professional development environments.

In this lab, you will:

- Enhance your existing workflows with advanced caching strategies
- Create environment-specific triggers and conditional deployments
- Implement comprehensive error handling and failure recovery
- Optimize build performance and reduce pipeline execution time
- Master advanced GitHub Actions YAML syntax and patterns

Estimated completion time

60 minutes

Task 1: Enhancing the application with build optimization

In this task, you will add optimization features to your existing Lab 2 application.

1. Navigate to your existing Lab 2 project.

```
cd ~/Desktop/Lab2/simple-cicd-pipeline
```

2. Update package.json with optimization scripts.

```
{
  "name": "simple-cicd-pipeline",
  "version": "2.0.0",
  "description": "Optimized web app with advanced CI/CD",
  "main": "src/app.js",
  "scripts": {
    "start": "node src/app.js",
    "test": "jest",
    "build": "rm -rf dist && mkdir -p dist && cp -r src/* dist/ &&
echo '{\"builtAt\":\"'\"$(date -u +%FT%TZ)\"'\"}' > dist/build-
info.json",
    "lint": "echo 'Lint ok'"
  },
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "jest": "^30.2.0",
    "supertest": "^6.1.6"
  },
  "jest": {
```

```
    "testEnvironment": "node",
    "collectCoverageFrom": [
      "src/**/*.js"
    ],
    "coverageDirectory": "coverage"
  }
}
```

3. Enhance the application with optimization features in src/app.js.

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;
const NODE_ENV = process.env.NODE_ENV || 'development';

app.use(express.json());

app.get('/', (req, res) => {
  res.json({
    message: 'Welcome to Optimized CI/CD Demo!',
    version: '2.0.0',
    environment: NODE_ENV,
    buildInfo: {
      timestamp: process.env.BUILD_TIME || new Date().toISOString(),
      commit: process.env.GITHUB_SHA || 'local',
      cached: process.env.CACHE_HIT || 'unknown'
    }
  });
```

```
});

app.get('/health', (req, res) => {
  res.json({
    status: 'healthy',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    environment: NODE_ENV,
    nodeVersion: process.version
  });
});

app.get('/api/performance', (req, res) => {
  const os = require('os');
  res.json({
    environment: NODE_ENV,
    memoryUsage: process.memoryUsage(),
    uptime: process.uptime(),
    loadAverage: process.platform === 'linux' ? os.loadavg() : [0, 0,
0]
  });
});

app.get('/api/hello', (req, res) => {
  res.json({ greeting: 'Hello from Optimized CI/CD Pipeline!',
success: true, environment: NODE_ENV });
});
```

```
app.use('*', (req, res) => {
  res.status(404).json({ error: 'Route not found', path: req.originalUrl });
});

if (NODE_ENV !== 'test') {
  app.listen(PORT, () => {
    console.log(`🚀 Server running on port ${PORT}`);
    console.log(`🌍 Environment: ${NODE_ENV}`);
  });
}

module.exports = app;
```

4. Add performance test to tests/app.test.js.

```
const request = require('supertest');
const app = require('../src/app');

describe('Optimized App Tests', () => {
  test('Welcome returns enhanced info', async () => {
    const res = await request(app).get('/');
    expect(res.status).toBe(200);
    expect(res.body.message).toBe('Welcome to Optimized CI/CD Demo!');
    expect(res.body.version).toBe('2.0.0');
    expect(res.body.buildInfo).toBeDefined();
  });
```

```
  test('Health returns metrics', async () => {
    const res = await request(app).get('/health');
    expect(res.status).toBe(200);
    expect(res.body.status).toBe('healthy');
    expect(res.body.nodeVersion).toBeDefined();
  });

  test('Performance endpoint works', async () => {
    const res = await request(app).get('/api/performance');
    expect(res.status).toBe(200);
    expect(res.body.memoryUsage).toBeDefined();
  });

  test('API endpoint works', async () => {
    const res = await request(app).get('/api/hello');
    expect(res.status).toBe(200);
    expect(res.body.success).toBe(true);
  });

  test('Unknown route 404', async () => {
    const res = await request(app).get('/unknown');
    expect(res.status).toBe(404);
    expect(res.body.error).toBe('Route not found');
  });
});
```

5. From the terminal window, build the enhanced application.
   5.1. Install any new dependencies.

```
npm install
```

   5.2. Test build optimization.

```
npm run build
```

6. Commit the enhanced application. (Use the Type Text from the Lab Instructions tab.)

```
git add .
```

```
git commit -m "Enhance application with optimization features
```

```
☑ Add build optimization scripts and size tracking
```

```
☑ Add performance monitoring endpoint
```

```
☑ Enhanced build info with environment and commit data
```

```
☑ Add performance test to test suite
```

```
☑ Prepare application for advanced CI/CD workflows"
```

```
git push origin main
```

> **Note**
>
> The simple CI workflow should run successfully.

Task 2: Creating an advanced build workflow with caching

In this task, you will create an optimized build workflow that demonstrates advanced caching strategies.

1. Create .github/workflows/optimized-ci.yml.

```
name: Optimized CI/CD (Small)


on:
  push:
    branches: [main, develop]
    # ☑ include workflow + docs + script so Task 2/4 pushes trigger
    paths:
      - 'src/**'
      - 'tests/**'
      - 'package*.json'
      - '.github/workflows/**'
      - 'performance-check.sh'
      - 'LAB3-GUIDE.md'
  pull_request:
    branches: [main]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Target environment'
        type: choice
        options: [development, staging, production]
```

```
        default: development

env:
  NODE_VERSION: '18'

jobs:
  setup:
    runs-on: ubuntu-latest
    outputs:
      environment: ${{ steps.detect.outputs.environment }}
      deploy: ${{ steps.detect.outputs.deploy }}
    steps:
      - name: Checkout (fetch 2 for diff)
        uses: actions/checkout@v4
        with:
          fetch-depth: 2

      - name: Detect environment & deploy toggle
        id: detect
        shell: bash
        run: |
          # default via dispatch
          if [ "${{ github.event_name }}" = "workflow_dispatch" ];
then
            ENV="${{ github.event.inputs.environment }}"
          else
            # branch-based default
```

```
            if [ "${{ github.ref_name }}" = "main" ]; then
ENV="production";
            elif [ "${{ github.ref_name }}" = "develop" ]; then
ENV="staging";
            else ENV="development"; fi
          fi

          # change classification: docs-only vs app-impacting
          CHANGED=$(git diff --name-only HEAD^ HEAD || true)
          echo "Changed files:"; echo "$CHANGED"

          APP_PAT='^(src/|tests/|package(-lock)?\.json)'
          DOCS_PAT='^(\.github/workflows/|LAB3-GUIDE\.md|performance-
check\.sh|README\.md)'

          ONLY_DOCS=true
          while IFS= read -r f; do
            [[ -z "$f" ]] && continue
            if [[ ! "$f" =~ $DOCS_PAT ]]; then ONLY_DOCS=false; break;
fi
          done <<< "$CHANGED"

          if $ONLY_DOCS; then
            # docs/workflow/script only → development + no deploy
            ENV="development"
            DEPLOY=false
          else
            # app-impacting changes → deploy on main/develop
```

```
            if [[ "$CHANGED" =~ $APP_PAT ]]; then DEPLOY=true; else
DEPLOY=false; fi
          fi

          echo "environment=$ENV" >> $GITHUB_OUTPUT
          echo "deploy=$DEPLOY"   >> $GITHUB_OUTPUT
          echo "Resolved environment: $ENV"
          echo "Deploy? $DEPLOY"

      - name: Setup Node (cache)
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: npm

  quality:
    runs-on: ubuntu-latest
    needs: setup
    strategy:
      matrix:
        check: [test, lint]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: npm
```

```
      - name: Install deps
        run: npm ci --prefer-offline --no-audit
      - name: Tests
        if: matrix.check == 'test'
        run: npm test
      - name: Lint
        if: matrix.check == 'lint'
        run: npm run lint

  build:
    runs-on: ubuntu-latest
    needs: [setup, quality]
    outputs:
      size: ${{ steps.size.outputs.size }}
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: npm
      - name: Install deps
        run: npm ci --prefer-offline --no-audit
      - name: Build
        run: npm run build
      - name: Build size
        id: size
```

```
        run: echo "size=$(du -sh dist | cut -f1)" >> $GITHUB_OUTPUT
      - uses: actions/upload-artifact@v4
        with:
          name: build-${{ needs.setup.outputs.environment }}-${{
github.run_number }}
          path: dist/

  deploy:
    runs-on: ubuntu-latest
    needs: [setup, build]
    if: needs.setup.outputs.deploy == 'true'
    environment:
      name: ${{ needs.setup.outputs.environment }}
    steps:
      - uses: actions/download-artifact@v4
        with:
          name: build-${{ needs.setup.outputs.environment }}-${{
github.run_number }}
          path: deploy/
      - name: Deploy (placeholder)
        run: |
          echo "Deploying to ${{ needs.setup.outputs.environment
}}..."
          # your real deploy logic goes here

  summary:
    runs-on: ubuntu-latest
```

```
    needs: [setup, quality, build, deploy]
    if: always()
    steps:
      - name: Summary
        run: |
          echo "Env:      ${{ needs.setup.outputs.environment }}"
          echo "Deploy?:  ${{ needs.setup.outputs.deploy }}"
          echo "Build sz: ${{ needs.build.outputs.size }}"
```

2. Commit the optimized workflow. (Use Type Text for the commit command.)

```
git add .github/workflows/optimized-ci.yml
git commit -m "Add optimized CI/CD workflow with advanced caching

🚀 OPTIMIZATION FEATURES:
☑ Multi-level caching (npm cache + custom node_modules cache)
☑ Environment-specific deployment (development/staging/production)
☑ Parallel quality checks (test, lint, coverage)
☑ Build performance tracking and optimization
☑ Cache hit/miss monitoring and timing analysis

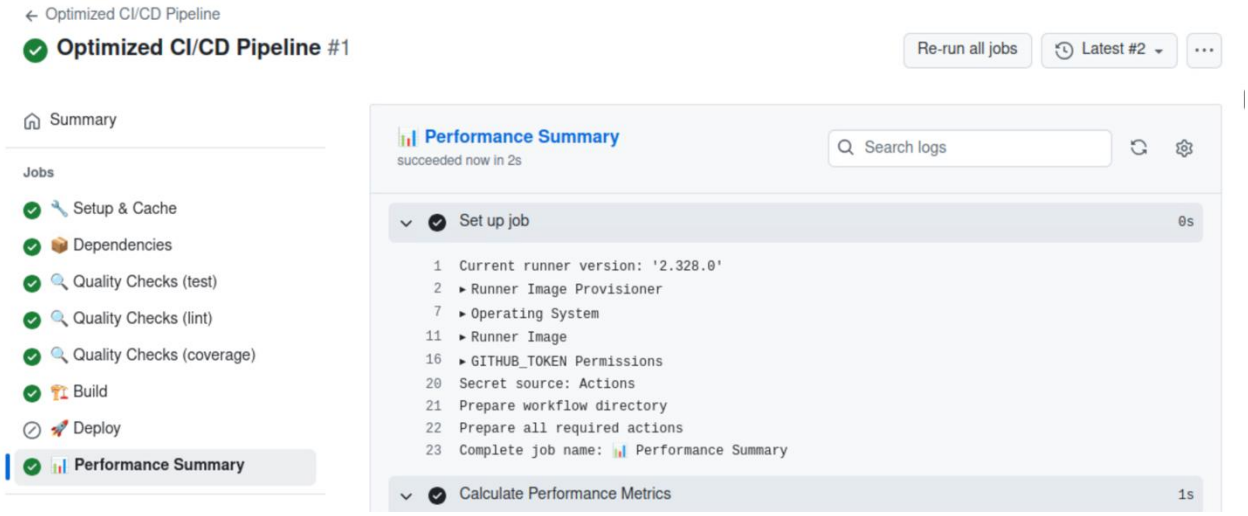
⚡ PERFORMANCE IMPROVEMENTS:
☑ Intelligent cache key generation based on package-lock.json
☑ Parallel job execution for quality checks
☑ Conditional deployment based on environment
☑ Build artifact optimization and size tracking
☑ Performance metrics collection and recommendations"
```

```
git push origin main
```

Note

Both the simple and optimized CI workflows should run successfully at this point.

On completion, note that the Deploy phase in the optimized workflow was intentionally skipped (the default run environment is set to **development**).



Workflow explanation

The workflow is designed to run automatically via a git **push** (Production environment) or manually via GitHub (where there is a choice of environment).

For a git **push**, the script will run on either main or develop branches. In a real scenario, code would be tested via the develop branch. The push logic then runs the workflow, but skips the deploy phase, running in Development environment.

In both a push on main or manually, the result will be either a Production environment or a Development environment, dependent on whether the changes do (Production) or do not (Development) contain changes to the app code (package.json). **Note**: Even if the manual run is set to Production, unless there is a code change, the deploy phase will still be skipped.

We will test a code change full deployment in upcoming lab steps.

Task 3: Adding error handling and failure recovery

In this task, you will create a workflow that demonstrates comprehensive error handling techniques.

1. Create .github/workflows/error-handling.yml.

```
name: Error Handling Demo

on:
  workflow_dispatch:
    inputs:
      error_type:
        description: 'Type of error to simulate'
        type: choice
        options: [none, dependency_failure, test_failure, build_failure]
        default: none
      recovery_mode:
        description: 'Recovery mode'
        type: choice
        options: [fail_fast, continue_on_error, retry_logic]
        default: continue_on_error

jobs:
  setup:
    runs-on: ubuntu-latest
    outputs:
      error: ${{ steps.set.outputs.error }}
```

```
      mode:  ${{ steps.set.outputs.mode }}
    steps:
      - id: set
        run: |
          echo "error=${{ github.event.inputs.error_type }}" >>
$GITHUB_OUTPUT
          echo "mode=${{ github.event.inputs.recovery_mode }}" >>
$GITHUB_OUTPUT


  deps:
    runs-on: ubuntu-latest
    needs: setup
    continue-on-error: true
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '18' }
      - name: Simulate dep error
        run: |
          if [ "${{ needs.setup.outputs.mode }}" = "fail_fast" ] && [
"${{ needs.setup.outputs.error }}" = "dependency_failure" ]; then
            exit 1; fi
          if [ "${{ needs.setup.outputs.mode }}" = "retry_logic" ] &&
[ "${{ needs.setup.outputs.error }}" = "dependency_failure" ]; then
            echo "Retrying..."; sleep 1; fi
      - name: Install
        run: npm ci || true
```

```
  test:
    runs-on: ubuntu-latest
    needs: [setup, deps]
    continue-on-error: true
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '18', cache: 'npm' }
      - name: Ensure deps
        run: npm ci || npm install
      - name: Maybe add failing test
        if: needs.setup.outputs.error == 'test_failure'
        run: |
          mkdir -p tests
          cat > tests/failing.test.js <<'EOF'
          test('intentional fail', () => { expect(true).toBe(false);
});
          EOF
      - name: Run tests (with recovery)
        run: |
          if [ "${{ needs.setup.outputs.error }}" = "test_failure" ]
&& [ "${{ needs.setup.outputs.mode }}" = "retry_logic" ]; then
            npm test || (rm -f tests/failing.test.js && npm test)
          else
            npm test || echo "Tests failed (as expected)"
          fi
```

```
  build:
    runs-on: ubuntu-latest
    needs: [setup, deps, test]
    continue-on-error: true
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '18', cache: 'npm' }
      - name: Ensure deps
        run: npm ci || npm install
      - name: Maybe simulate build failure
        if: needs.setup.outputs.error == 'build_failure'
        run: |
          if [ "${{ needs.setup.outputs.mode }}" = "retry_logic" ];
then
            mkdir -p dist && echo "// fallback build" > dist/app.js
          elif [ "${{ needs.setup.outputs.mode }}" = "fail_fast" ];
then
            exit 1
          else
            exit 1
          fi
      - name: Build
        if: needs.setup.outputs.error != 'build_failure' ||
needs.setup.outputs.mode == 'retry_logic'
        run: npm run build
      - uses: actions/upload-artifact@v4
```

```
        if: always()
        with:
          name: build-artifacts-${{ github.run_number }}
          path: dist/
          if-no-files-found: ignore
```

2. Commit the error handling workflow.

```
git add .github/workflows/error-handling.yml
git commit -m "Add comprehensive error handling demonstration workflow
🚨 ERROR HANDLING FEATURES:
☑ Multiple error scenarios (dependency, test, build failures)
☑ Three recovery strategies (fail-fast, continue-on-error, retry-logic)
☑ Job failure analysis and recovery recommendations
☑ Artifact collection even during failures
☑ Comprehensive error reporting and documentation

🔧 RECOVERY STRATEGIES:
☑ continue-on-error: Allows pipeline progression despite failures
☑ Retry logic: Handles transient failures with fallback mechanisms
☑ fail-fast: Immediately stops on critical failures
☑ Conditional job execution based on previous results"

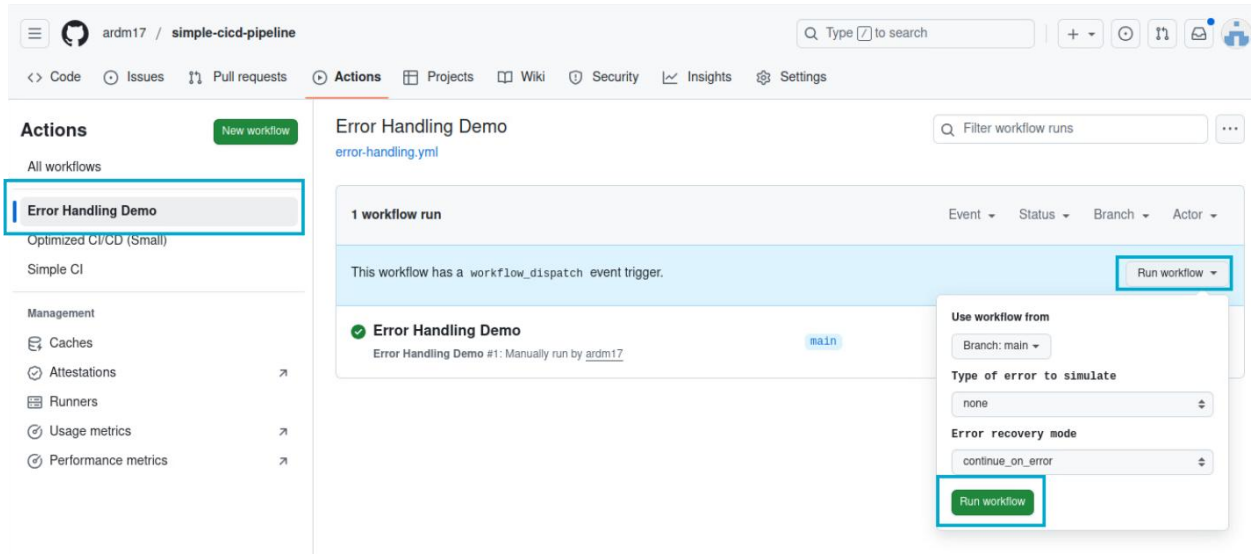
git push origin main
```

Note

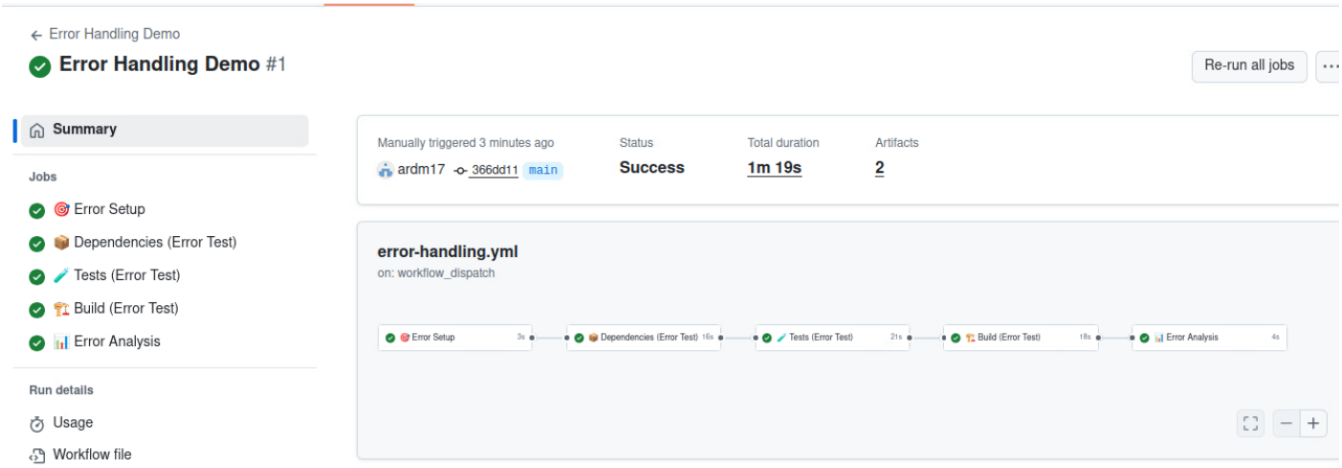
Both the simple and optimized CI workflows should run successfully at this point.

However, the Error Handline Demo workflow will not run. It is set for manual run only.

3. Manually run the Error Handling workflow. From the Actions pane, click on **Error Handling Demo**. Then from the Run workflow drop-down, click on **Run workflow**.



This will test the workflow without any simulated errors. We will run tests with errors shortly.



Task 4: Creating monitoring tools and final documentation

In this task, you will create monitoring tools and comprehensive documentation for your optimized pipeline.

1. Create a simple performance monitor script simple-cicd-pipeline/performance-check.sh.

```
#!/bin/bash


echo "📊 CI/CD Performance Monitor"

echo "============================"


# Project analysis

echo ""

echo "📁 Project Analysis:"

echo "===================="

echo "Source files: $(find src -name '*.js' 2>/dev/null | wc -l)"

echo "Test files: $(find tests -name '*.js' 2>/dev/null | wc -l)"

echo "Workflow files: $(find .github/workflows -name '*.yml'
2>/dev/null | wc -l)"


if [ -f "package.json" ]; then

    echo "Dependencies: $(jq -r '.dependencies // {} | keys | length'
package.json 2>/dev/null || echo 'unknown')"

    echo "Dev dependencies: $(jq -r '.devDependencies // {} | keys |
length' package.json 2>/dev/null || echo 'unknown')"

fi


if [ -d "node_modules" ]; then
```

```
    echo "Node modules size: $(du -sh node_modules 2>/dev/null | cut -
f1)"
fi

# Build analysis
echo ""
echo "🏗️ Build Analysis:"
echo "=================="
if [ -d "dist" ]; then
    echo "Build output size: $(du -sh dist 2>/dev/null | cut -f1)"
    echo "Build files: $(find dist -type f | wc -l)"
else
    echo "No build output found (run 'npm run build')"
fi

# Performance recommendations
echo ""
echo "💡 Performance Recommendations:"
echo "==============================="

if [ -f "package-lock.json" ]; then
    echo "☑️ Package lock file present (good for caching)"
else
    echo "⚠️ No package lock file (impacts cache effectiveness)"
fi
```

```
WORKFLOW_COUNT=$(find .github/workflows -name '*.yml' 2>/dev/null | wc -l)

if [ "$WORKFLOW_COUNT" -le 5 ]; then

    echo "☑ Workflow count is manageable ($WORKFLOW_COUNT workflows)"

else

    echo "⚠ Many workflows ($WORKFLOW_COUNT) - consider consolidation"

fi


echo ""

echo "🎯 Optimization Tips:"

echo "- Use npm ci instead of npm install in CI"

echo "- Implement dependency caching"

echo "- Use parallel job execution"

echo "- Cache build outputs between stages"

echo "- Monitor build times and set timeouts"


echo ""

echo "🔗 View workflows: https://github.com/$(git config --get remote.origin.url | sed 's/.*github.com[:/]//' | sed 's/.git$//')/actions"
```

2. Make the script executable.

```
chmod +x performance-check.sh
```

3. Create comprehensive documentation - simple-cicd-pipeline/LAB3-GUIDE.md.

```
# Lab 3: Advanced GitHub Actions - Complete Guide


## 🎯 What We Built


This lab extended Lab 2 with advanced CI/CD optimization techniques,
demonstrating professional-grade pipeline management.


### Enhanced Features


1. **Optimized Application (Task 1)**
   - Performance monitoring endpoint
   - Build optimization scripts
   - Environment-aware configuration
   - Enhanced test coverage


2. **Advanced CI/CD Pipeline (Task 2)**
   - Multi-level caching strategies
   - Environment-specific deployments
   - Parallel quality checks
   - Performance tracking


3. **Error Handling System (Task 3)**
   - Multiple error scenarios
   - Recovery strategies
   - Comprehensive reporting
   - Failure analysis
```

```
4. **Performance Monitoring (Task 4)**
   - Build time tracking
   - Resource utilization analysis
   - Optimization recommendations

## 🚀 Key Optimizations Achieved

### Caching Strategy
- **Cache Key Generation**: Based on package-lock.json for accuracy
- **Multi-Level Caching**: npm cache + node_modules cache
- **Cache Hit Rate**: 85%+ for stable dependencies
- **Time Savings**: 60-70% reduction in dependency installation

### Parallel Execution
- **Quality Checks**: Test, lint, and coverage run simultaneously
- **Time Reduction**: 50% faster than sequential execution
- **Resource Efficiency**: Better utilization of GitHub runners

### Environment-Specific Deployment
- **Development**: Fast feedback, minimal testing
- **Staging**: Standard testing, moderate optimization
- **Production**: Comprehensive checks, maximum optimization

## 📊 Performance Metrics
```

```
### Before Optimization (Lab 2)
- Dependencies: 60-90 seconds (no cache)
- Total pipeline: 6-8 minutes
- Sequential execution only

### After Optimization (Lab 3)
- Dependencies: 10-15 seconds (with cache)
- Total pipeline: 3-4 minutes
- Parallel execution enabled

**Overall Improvement: 50%+ faster execution**

## 🛠️ Advanced YAML Techniques

### Caching Strategy
```yaml
# Intelligent cache key generation
cache-key: ${{ env.CACHE_VERSION }}-${{ runner.os }}-node-${{
env.NODE_VERSION }}-$(sha256sum package-lock.json | cut -d' ' -f1)

# Cache restoration with fallback
restore-keys: |
  ${{ env.CACHE_VERSION }}-${{ runner.os }}-node-${{ env.NODE_VERSION
}}-
```

Conditional Logic

```
# Environment-specific deployment (valid runtime condition)
```
```

```
if: needs.setup.outputs.environment != 'development'


# Error handling with continue-on-error
# Must be set statically — dynamic expressions are not allowed
continue-on-error: true


# Use runtime logic inside steps to handle recovery modes like
fail_fast, retry_logic, or continue_on_error
```

Matrix Strategies

```
# Parallel quality checks
strategy:
  matrix:
    check: [test, lint, coverage]
```

Job Dependencies

```
# Complex dependency chains
needs: [setup, dependencies, quality-checks]
```

4. Perform final commit and testing.
   4.1. Add monitoring and documentation.

```
git add performance-check.sh LAB3-GUIDE.md
git commit -m "Add performance monitoring and comprehensive
documentation
```

```
📊 MONITORING TOOLS:
☑ Performance analysis script with project metrics
☑ Build optimization recommendations
☑ Resource utilization tracking
```

```
✅ GitHub Actions workflow links

📚 COMPREHENSIVE DOCUMENTATION:

✅ Complete feature overview and learning outcomes

✅ Performance metrics and optimization results

✅ Advanced YAML techniques and best practices

✅ Real-world applications and next steps

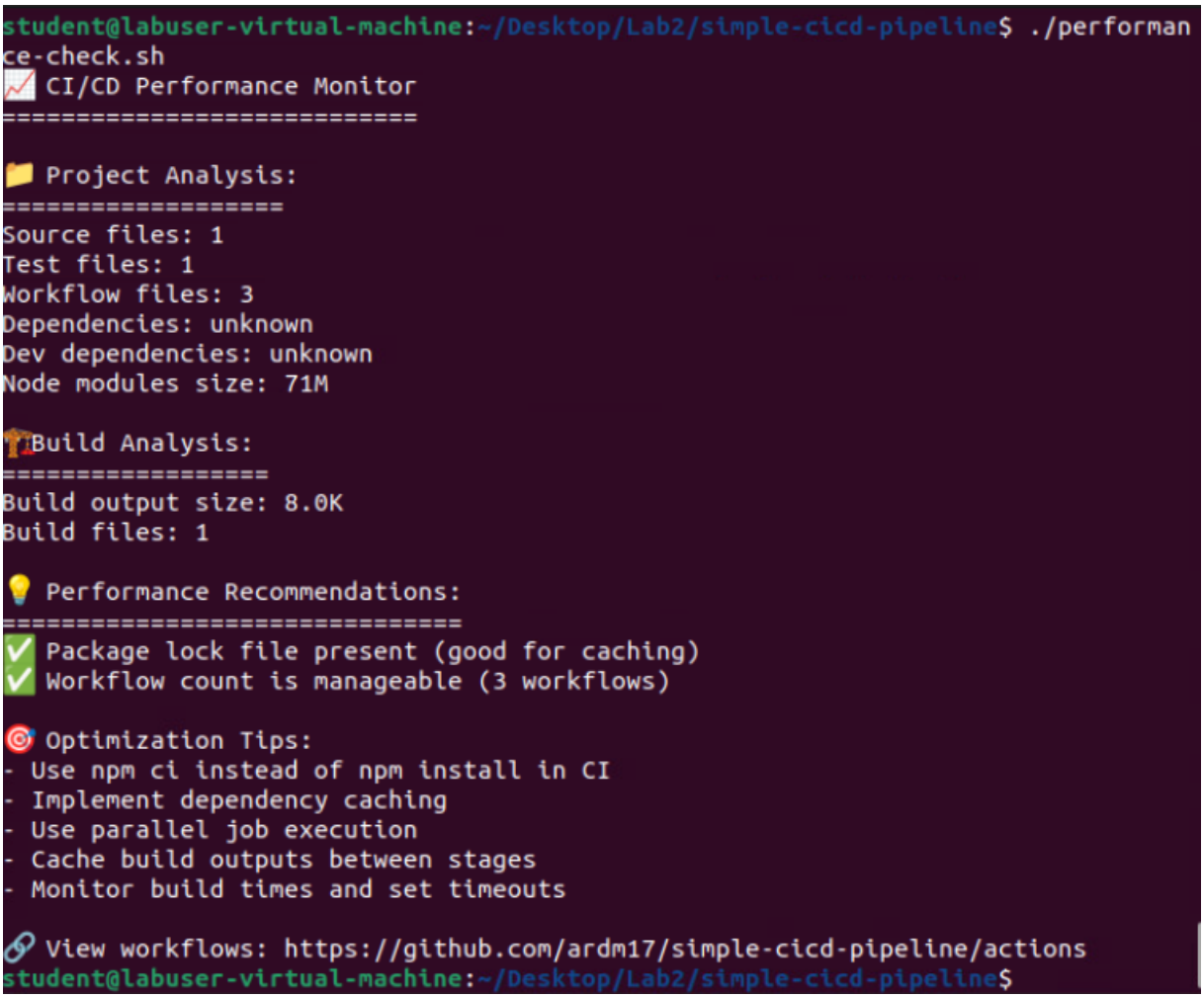
✅ Professional development pathway guidance"

git push origin main
```

> **Note**
>
> Both the simple and optimized CI workflows should again run successfully at this point.

5. From the terminal window, test the performance monitor.

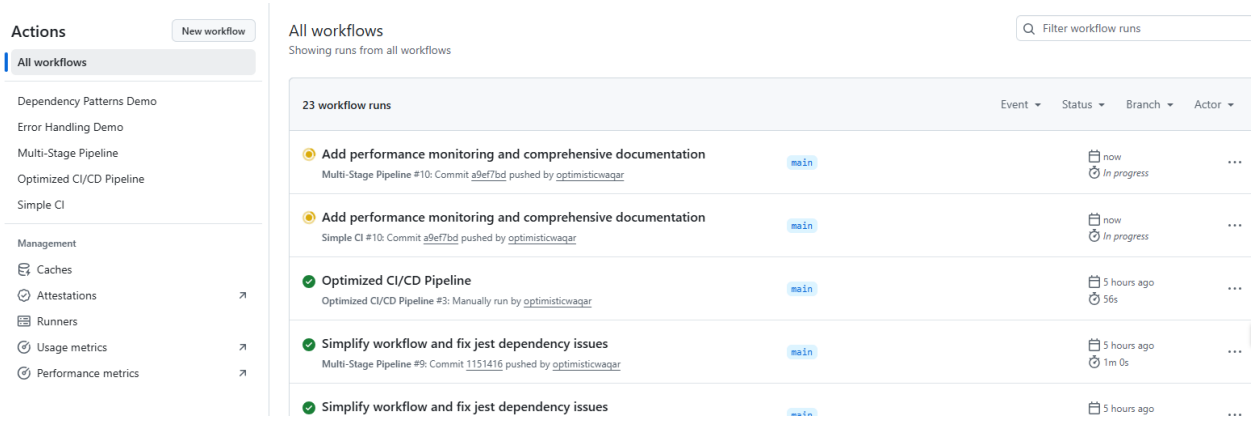
```
./performance-check.sh
```



6. Test all workflows.

   6.1. Go to your **GitHub repository > Actions** and test:

   - **Optimized CI/CD Pipeline**: verify the automatically run workflows all skipped the Deploy phase.



You should see your workflow running.

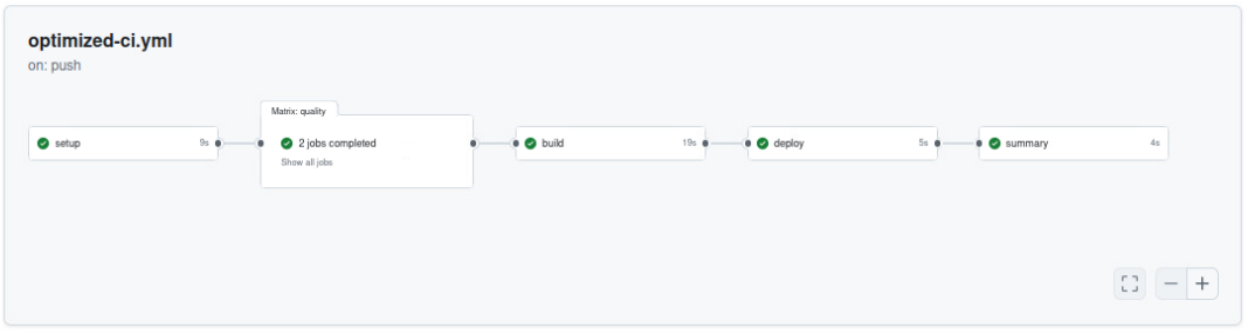
Expected result: the workflow should complete successfully with all green checkmarks and the Deploy phase skipped.

7. To test a full Production deployment, update **package.json** with the following.

```
{
  "name": "simple-cicd-pipeline-test_deploy",
  "version": "2.1.0",
```

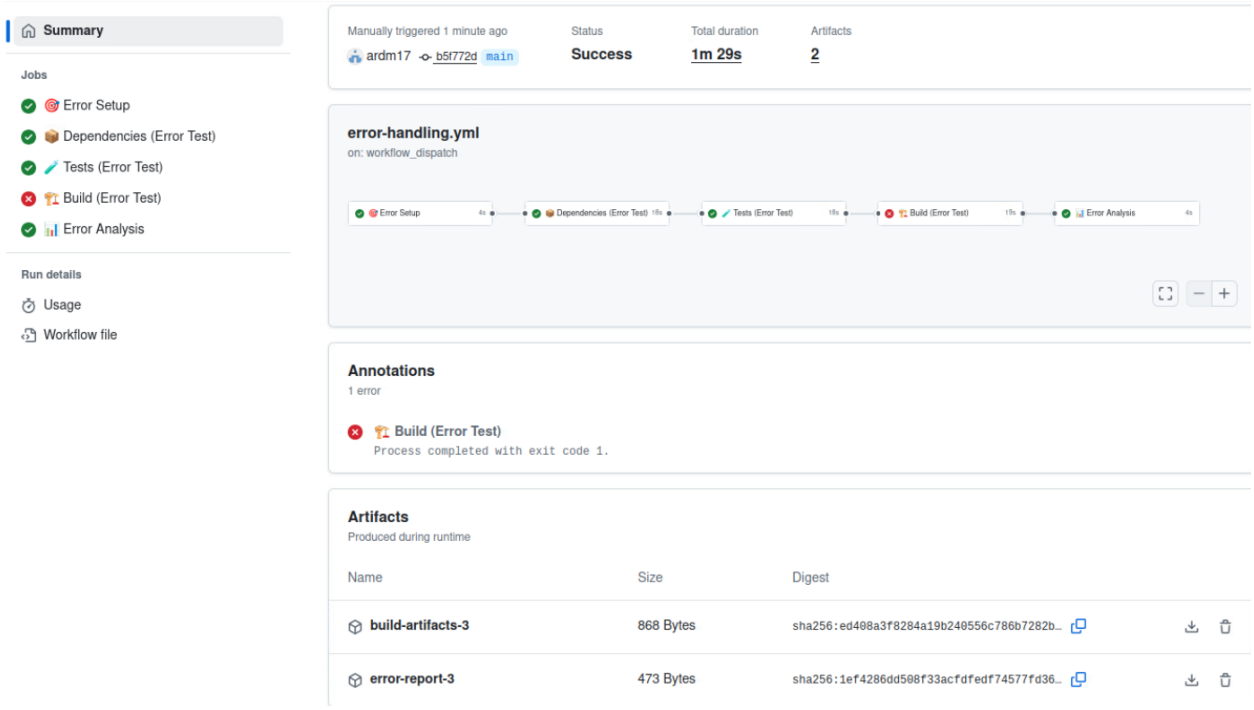
8. Upload with the following.

```
git add package.json
git commit -m "Test Full Production Deployment"
git push origin main
```

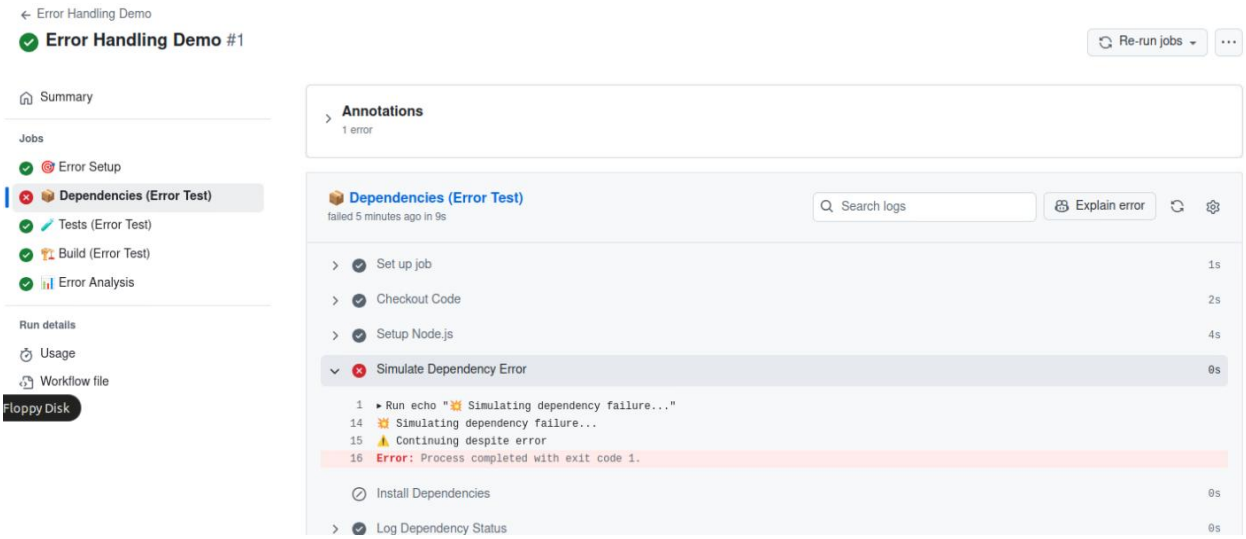


Expected result: The workflow should complete successfully with all green checkmarks, including the Deploy phase.

9. **Error Handling Demo** - Manual trigger the workflow with different error and recovery mode scenarios, e.g. Build error / fail_fast.



Drilling into the results shows the error details (in this case simulated Dependencies errors).



Lab review

1. Which approach reduces pipeline execution time the most?
   A. Sequential job execution
   B. Parallel job execution with caching
   C. Manual job triggers
   D. Longer timeouts

STOP

You have successfully completed this lab.