Complete 3-Pillar GenAI Development Environment Setup Guide

This comprehensive guide walks you through setting up a powerful GenAI development environment using Claude Code for primary development, Gemini CLI for code review, and Claude Desktop with MCP for CI/CD coordination, all integrated with GitHub source control.

Overview: The 3-Pillar Architecture

The three-pillar approach creates a robust AI-assisted development workflow where each tool excels in its specialized role:

1. **Claude Code (Pro/Max)** - Primary development, architecture, refactoring
2. **Gemini CLI (Free)** - Code review, testing, validation, second opinions
3. **Claude Desktop + MCP** - CI/CD coordination, GitHub integration, deployment orchestration

This setup provides redundancy, specialized expertise, and comprehensive coverage of the development lifecycle while maintaining cost efficiency through Gemini's generous free tier.

Prerequisites and System Requirements

Common Requirements

- **Operating System**: macOS 10.15+, Windows 10/11, Claude AI Hub Claudiai or Ubuntu 20.04+ Codecademy +2
- **Node.js**: Version 18 or higher GitHub github
- **Git**: Latest version
- **Internet Connection**: Required for all AI model access
- **GitHub Account**: With repository access

Tool-Specific Requirements

- **Claude Code**: WSL required for Windows, 8GB RAM minimum Codecademy ITECS
- **Gemini CLI**: Cross-platform native support
- **Claude Desktop**: macOS/Windows only (no Linux support currently) Modelcontextprotocol +2

Pillar 1: Claude Code Setup (Primary Development)

Subscription and Pricing

Claude Pro: $20/month - Suitable for individual developers Anthropic Anthropic

- ~45 messages per 5-hour session Anthropic
- ~216 messages per day maximum 16x

- 200K token context window Amazon LobeHub

Claude Max (5x): $100/month - For heavy usage Apidog Anthropic

- ~225 messages per 5-hour session Anthropic
- 5x Pro usage limits

Claude Max (20x): $200/month - For development teams Apidog Anthropic

- ~900 messages per 5-hour session Anthropic
- 20x Pro usage limits

Installation Process

bash

```bash
# 1. Install Claude Code globally
npm install -g @anthropic-ai/claude-code

# 2. Verify installation
claude --version

# 3. Navigate to your project directory
cd your-project-directory

# 4. Launch Claude Code
claude
# Complete OAuth authentication when prompted

# 5. Initialize project context
# In Claude Code terminal, run:
/init
```

Essential Configuration

The `/init` command creates a **CLAUDE.md** file that provides persistent project context:

markdown

```
# CLAUDE.md Example
## Project Overview
This is a React TypeScript application with Node.js backend...

## Architecture
- Frontend: React 18 with TypeScript
- Backend: Node.js with Express
- Database: PostgreSQL
- Authentication: JWT

## Development Guidelines
- Use TypeScript strict mode
- Follow ESLint configuration
- Maintain 90%+ test coverage
```

GitHub Integration Setup

bash

```
# Install GitHub CLI if not already installed
# macOS
brew install gh

# Windows
winget install GitHub.cli

# Ubuntu
sudo apt install gh

# Authenticate with GitHub
gh auth login

# Clone repositories directly in Claude Code
git clone https://github.com/username/repository.git
cd repository
claude
/init
```

Best Practices for Primary Development

Natural Language Commands:

"Refactor the authentication module to use TypeScript interfaces"
"Add comprehensive error handling to the API routes"
"Create unit tests for the user service functions"
"Generate API documentation for the REST endpoints"

File Management:

- Use @filename to reference specific files
- Let Claude understand the entire codebase through CLAUDE.md Annjose
- Commit CLAUDE.md to version control for team sharing

Pillar 2: Gemini CLI Setup (Code Review & Validation)

Installation Options

Option 1: NPX (Recommended)

```bash
# Run directly without installation
npx https://github.com/google-gemini/gemini-cli

# Or from main branch for latest features
npx github:google-gemini/gemini-cli
```

Option 2: Global Installation

```bash
# Install globally
npm install -g @google/gemini-cli

# Launch from anywhere
gemini
```

Option 3: Docker (Sandbox Mode)

```bash
# Run in isolated container
docker run --rm -it us-docker.pkg.dev/gemini-code-dev/gemini-cli/sandbox:0.1.1
```

Authentication Setup

Google Account (Free Tier - Recommended)

```bash
# Launch Gemini CLI
gemini

# Select "Login with Google" when prompted
# Complete browser authentication
# Automatic token management
```

Free Tier Limits (2025):

- 60 model requests per minute Google Simonwillison
- 1,000 requests per day Google Simonwillison
- Gemini 2.5 Pro with 1M token context Entelligence Blog +2
- Completely free with personal Google account Google Simonwillison

GitHub Integration for Code Review

Install GitHub App:

1. Go to GitHub Marketplace
2. Search for "Gemini Code Assist" GitHub
3. Install on repositories (free for public and private repos) GitHub
4. Configure repository access

Setup Configuration:

bash

```
# Create .gemini directory in your project
mkdir .gemini

# Create config file
cat > .gemini/config.yaml << EOF
have_fun: true
code_review:
  disable: false
  comment_severity_threshold: MEDIUM
  max_review_comments: -1
pull_request_opened:
  help: false
  summary: true
  code_review: true
ignore_patterns: []
EOF

# Add custom style guide
cat > .gemini/styleguide.md << EOF
# Team Coding Standards
- Use TypeScript strict mode
- Follow conventional commits
- Maintain test coverage above 80%
- Use ESLint and Prettier configurations
EOF
```

Code Review Workflow Examples

Project Analysis:

bash

```
cd my-project
gemini

# Analyze entire codebase
> @src/ Review this codebase for security vulnerabilities and code quality issues

# Review specific files
> @src/auth.js @src/database.js Check these modules for potential race conditions

# Generate test coverage report
> Analyze test coverage and suggest additional test cases for untested code paths
```

Git Integration:

```bash
# Review recent changes
> Give me a summary of all changes that went in yesterday

# Analyze pull request
> Review the changes in the current git diff and identify potential issues

# Generate commit messages
> Generate a conventional commit message for the current staged changes
```

Pillar 3: Claude Desktop with MCP Setup (CI/CD Coordination)

Claude Desktop Installation

1. **Download**: Visit claude.ai/download Claude AI Hub Claudiai
2. **Install**: Run the installer for your platform Claude AI Hub
3. **Sign In**: Use your Claude account credentials Claude AI Hub
4. **Verify**: Check Claude menu > Check for Updates Modelcontextprotocol

MCP Configuration for GitHub Operations

Locate Configuration File:

- **macOS**: ~/Library/Application Support/Claude/claude_desktop_config.json :antCitation[]{citations="cfb2e021-c6c1-4adb-8998-f38903d64f30,21a4bf9a-ed6a-4197-8785-03041c674eb5"} Modelcontextprotocol
- **Windows**: %APPDATA%\Claude\claude_desktop_config.json Modelcontextprotocol

Basic GitHub MCP Setup:

json

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_xxxxxxxxxxxx"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/username/Projects"
      ]
    }
  }
}
```

Advanced Configuration with Full Toolsets:

json

```
{
  "mcpServers": {
    "github-full": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_xxxxxxxxxxxx",
        "GITHUB_TOOLSETS": "repos,issues,pull_requests,actions,code_security"
      }
    },
    "github-actions": {
      "command": "npx",
      "args": ["-y", "github-actions-mcp-server"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_xxxxxxxxxxxx"
      }
    }
  }
}
```

GitHub Token Setup

Create Personal Access Token:

1. Go to GitHub Settings > Developer settings > Personal access tokens All Things How GitHub
2. Generate new token (fine-grained recommended) GitHub
3. Select required scopes:
   - Repository access (read/write)
   - Actions (read/write)
   - Issues (read/write)
   - Pull requests (read/write)
   - Metadata (read)

Store Token Securely:

```bash
# macOS/Linux
export GITHUB_TOKEN="ghp_xxxxxxxxxxxx"

# Windows
set GITHUB_TOKEN=ghp_xxxxxxxxxxxx

# Add to your shell profile for persistence
echo 'export GITHUB_TOKEN="ghp_xxxxxxxxxxxx"' >> ~/.bashrc
```

CI/CD Coordination Examples

Workflow Management:

```
"List all GitHub Actions workflows in my main repository"
"Trigger the deployment workflow for the staging environment"
"Check the status of the latest build and show me any failures"
"Create a new release with automated changelog generation"
```

Repository Operations:

```
"Create a new feature branch called 'ai-integration' from main"
"List all open pull requests and their review status"
"Merge the approved PR #123 with squash commit"
"Create an issue for the bug we discussed earlier"
```

Security Configuration and Best Practices

Authentication Security

GitHub App vs Personal Access Tokens:

- **GitHub Apps**: Recommended for production (higher rate limits, better security) Stack Overflow
- **Fine-grained PATs**: Better than classic PATs, repository-specific access GitHub
- **Classic PATs**: Use only when necessary, broad access scope GitHub

Rate Limiting Considerations:

- **Unauthenticated**: 60 requests/hour
- **Personal Access Tokens**: 5,000 requests/hour
- **GitHub Apps**: 5,000 base + scaling (max 15,000/hour)

Security Best Practices

Token Management:

```json
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${GITHUB_TOKEN}"
      }
    }
  }
}
```

Repository Access Control:

- Use least-privilege principle
- Regular permission audits
- Environment-specific secrets
- Branch protection rules

Secret Scanning Setup:

yaml

```yaml
# .github/workflows/security.yml
name: Security Scan
on: [push, pull_request]
jobs:
  security:
    runs-on: ubuntu-latest
    permissions:
      security-events: write
      contents: read
    steps:
      - uses: actions/checkout@v4
      - name: Initialize CodeQL
        uses: github/codeql-action/init@v3
        with:
          languages: javascript, python
      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v3
```

Workflow Integration Between the Three Pillars

Development Workflow Pattern

1. Feature Development (Claude Code)

bash

```bash
# Start new feature
cd project-directory
claude

# Natural language development
> "Create a new user authentication module with JWT tokens"
> "Add input validation and error handling"
> "Generate comprehensive unit tests"
```

2. Code Review (Gemini CLI)

bash

```
# Switch to Gemini CLI for review
gemini

# Review the changes
> @src/auth/ Review this authentication module for security vulnerabilities
> Generate a summary of code quality issues and suggestions
> Create additional test cases for edge cases not covered
```

3. CI/CD Coordination (Claude Desktop + MCP)

```
# In Claude Desktop
"Create a pull request for the authentication feature"
"Set up automated testing workflow for this feature"
"Monitor the CI/CD pipeline and report on build status"
"Deploy to staging environment after tests pass"
```

Integrated Workflow Example

Complete Feature Development Cycle:

bash

```
# 1. DEVELOPMENT PHASE (Claude Code)
cd my-app
claude
> "Implement user registration API with email verification"
> "Add rate limiting to prevent abuse"
> "Create comprehensive error handling"

# 2. REVIEW PHASE (Gemini CLI)
gemini
> @src/api/auth/ Review this registration API for security issues
> Check rate limiting implementation for effectiveness
> Suggest improvements for error handling

# 3. DEPLOYMENT PHASE (Claude Desktop + MCP)
# In Claude Desktop:
"Create feature branch and push changes"
"Open pull request with generated description"
"Trigger automated testing pipeline"
"Monitor test results and deployment status"
"Merge to main branch after approval"
"Deploy to production with monitoring"
```

Advanced Configuration and Optimization

Performance Optimization

Claude Code Optimization:

```bash
# Optimize for large codebases
claude --include-files --context-limit 150000

# Use project-specific instructions
# Add to CLAUDE.md:
## AI Instructions
- Focus on TypeScript best practices
- Prioritize performance and security
- Generate comprehensive documentation
- Maintain consistent code style
```

Gemini CLI Optimization:

```bash
# Create reusable review templates
cat > .gemini/review-template.md << EOF
## Review Checklist
- [ ] Security vulnerabilities
- [ ] Performance issues
- [ ] Code maintainability
- [ ] Test coverage
- [ ] Documentation quality
EOF

# Use project-specific context
> @.gemini/review-template.md Use this checklist to review @src/
```

Multi-Repository Management

Centralized Configuration:

json

```
{
  "mcpServers": {
    "multi-repo-github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "ghp_xxxxxxxxxxxx",
        "GITHUB_ORGANIZATIONS": "myorg,myteam"
      }
    }
  }
}
```

Cross-Repository Operations:

```
"List all repositories in my organization with pending security alerts"
"Create standardized GitHub Actions workflows across all my repos"
"Generate deployment status report for all production services"
"Coordinate releases across multiple microservices"
```

Troubleshooting Common Issues

Claude Code Issues

Problem: High usage warnings **Solution**:

- Upgrade to Claude Max for higher limits
- Use more focused prompts
- Break large tasks into smaller sessions

Problem: Windows WSL issues **Solution**:

bash

```
# Ensure WSL2 is properly configured
wsl --set-default-version 2
wsl --update

# Install in WSL environment
wsl
npm install -g @anthropic-ai/claude-code
```

Gemini CLI Issues

Problem: Authentication failures **Solution**:

```bash
# Clear authentication cache
rm -rf ~/.config/gemini-cli

# Re-authenticate
gemini
# Select "Login with Google" again
```

Problem: Rate limiting **Solution**:

- Monitor request usage with `/stats` command
- Implement request caching
- Use API keys for higher limits if needed

Claude Desktop + MCP Issues

Problem: MCP servers not connecting **Solution**:

1. Verify Node.js version (18+) `Modelcontextprotocol` `Proflead`
2. Check configuration file syntax
3. Restart Claude Desktop after configuration changes `Modelcontextprotocol`
4. Verify token permissions

Problem: GitHub operations failing **Solution**:

```bash
# Test token manually
curl -H "Authorization: token $GITHUB_TOKEN" \
  https://api.github.com/user

# Check rate limits
curl -H "Authorization: token $GITHUB_TOKEN" \
  https://api.github.com/rate_limit
```

Network and Connectivity

Common Network Issues:

- Corporate firewalls blocking AI API access
- VPN interference with authentication
- DNS resolution issues

Solutions:

```bash
# Test connectivity
curl -I https://api.anthropic.com
curl -I https://api.openai.com
curl -I https://api.github.com

# Configure proxy if needed
export HTTP_PROXY=http://proxy.company.com:8080
export HTTPS_PROXY=http://proxy.company.com:8080
```

Cost Analysis and ROI Considerations

Monthly Cost Breakdown

Minimal Setup:

- Claude Pro: $20/month Anthropic Tech
- Gemini CLI: Free
- **Total: $20/month**

Power User Setup:

- Claude Max (5x): $100/month Anthropic VentureBeat
- Gemini CLI: Free
- **Total: $100/month**

Team Setup:

- Claude Max (20x): $200/month Anthropic VentureBeat
- Gemini CLI: Free (per developer)
- **Total: $200/month per team**

ROI Calculations

Productivity Gains:

- 30-50% faster code development
- 60-80% reduction in code review time
- 40-60% faster debugging and troubleshooting
- 70-90% reduction in documentation time

Cost Savings:

- Reduced debugging time: $500-1000/month per developer
- Faster feature delivery: $1000-2000/month per project
- Improved code quality: $500-1500/month in reduced bugs

Break-even Analysis: Most teams see ROI within 2-4 weeks of implementation through productivity gains alone.

Conclusion and Next Steps

This 3-pillar GenAI development environment provides a comprehensive, cost-effective solution for modern software development. The combination of Claude Code's development expertise, Gemini CLI's review capabilities, and Claude Desktop's integration power creates a robust AI-assisted workflow. Anthropic

Immediate Next Steps:

1. Set up Claude Pro subscription and install Claude Code
2. Configure Gemini CLI for your primary development machine
3. Install Claude Desktop and configure basic MCP integration
4. Test the workflow with a sample project
5. Gradually expand MCP capabilities as needed

Long-term Optimization:

- Monitor usage patterns and adjust subscriptions accordingly
- Expand MCP server integrations for additional tools
- Implement team-wide standardization
- Develop custom workflow automation patterns

The investment in this setup typically pays for itself within weeks through increased productivity, improved code quality, and reduced development cycle times. Start with the minimal setup and scale based on your team's needs and usage patterns.