Claude Code Agent Cheat Sheet: 7 Battle-Tested Builds for Teams

Turn Claude Code from Individual Tool to Team Superpower — Plugins, Agents, and Architectures That Save Hours Weekly

By Julian, codecoast labs GmbH — 500+ Hours Battle-Testing Claude Code

[Visual: Interconnected agent nodes with plugin icons]

Free for email subscribers. Unlock scalable setups without the chaos.

julian@codecoast.ch

The Team Chaos Problem

You've seen it before: One developer discovers Claude Code and becomes 3x more productive. But when you try to scale that across the team? Chaos.

Common Team Pains

- **Inconsistent setups** — Every dev has their own prompts, workflows, and "tricks"
- **No shared leverage** — Knowledge stays siloed; no compounding returns
- **Slow onboarding** — New hires take weeks to learn the unwritten rules
- **Security concerns** — No governance on what AI can access or modify
- **Context switching** — Tools don't talk to each other; Claude doesn't know your stack

My Story

After 500+ hours battle-testing Claude Code across consulting projects, I've distilled what works into 7 reusable builds. These aren't theoretical—they're production-tested architectures that engineering teams use daily.

What You'll Get

This cheat sheet encodes best practices for plugins, agents, and integrations. Each build includes:

- The specific pain it solves
- Architecture diagram
- Copy-paste code snippets
- ROI metrics so you can justify the investment

Ready for a Custom Audit?

Get team-specific recommendations and ROI projections

Book Your €997 Team Audit →

1 Multi-Agent Code Review System

Pain: Senior devs bogged down in PRs

Your senior engineers spend 40% of their time reviewing code. Most of that is catching the same issues: style violations, missing tests, security oversights. What if AI handled the repetitive checks?

SOLUTION

Deploy 5 parallel review agents, each specialized for a single concern:

🐛 Bug Hunter	🔒 Security	📐 Style	🧪 Tests	📝 Docs

Each agent runs in parallel on PR creation, posts findings as comments, and escalates only significant issues to human reviewers.

[Diagram: Flowchart showing PR → 5 parallel agents → Merged findings → Human review gate]

```
# Slash command to trigger review
/review-pr --parallel --agents="bugs,security,style,tests,docs"

# Example agent prompt (security)
"Analyze this diff for security issues: hardcoded secrets,
SQL injection, XSS vectors, insecure dependencies.
Output: JSON with severity, line number, recommendation."
```

ROI Saves 30-40% senior dev time → €2,400-3,200/month per senior (at €120/hr)

2 Context Loader with CLAUDE.md

Pain: Context-switching kills productivity

Every time Claude starts a new conversation, it forgets your architecture, naming conventions, and tech stack. You waste the first 5 minutes re-explaining the same context.

SOLUTION

Create a project-specific `CLAUDE.md` file at the repo root. Claude Code automatically loads this on every session, giving it persistent context about your project.

```
my-project/ ├── CLAUDE.md ← Auto-loaded context ├── src/ ├── tests/ ├──
package.json └── README.md
```

```
# CLAUDE.md

## Project Overview
E-commerce API built with Node.js + TypeScript + PostgreSQL.

## Architecture
- src/controllers/ → Route handlers
- src/services/ → Business logic
- src/models/ → Prisma schema

## Conventions
- Use kebab-case for files, camelCase for variables
- All endpoints return { data, error, meta } shape
- Tests use Vitest; run with `pnpm test`

## Commands
- `pnpm dev` → Start dev server (port 3000)
- `pnpm db:migrate` → Run migrations
```

Pro Tip

Add team-specific prompts like "Always check for N+1 queries" or "Use our custom logger, not console.log". The more specific, the better.

ROI New hires productive in days, not weeks. Zero context re-explanation.

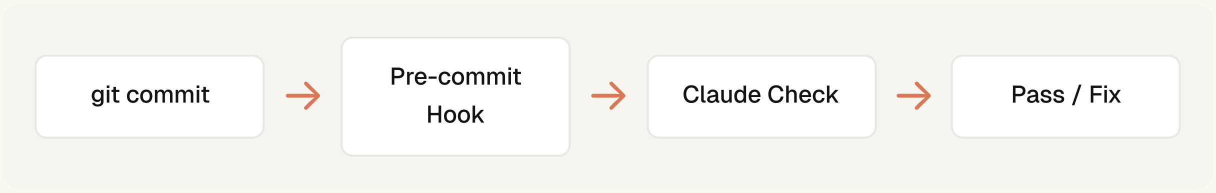
3 Standards Enforcer Hooks

Pain: Inconsistent code styles

"Can you rename this variable?" "We don't use that pattern here." Half your PR comments are style debates, not logic discussions.

SOLUTION

Pre-commit hooks that run Claude-powered checks before code ever reaches the PR. Catches issues at write-time, not review-time.



```
// .husky/pre-commit
#!/bin/sh

# Run Claude standards check on staged files
claude-code check-standards --staged \
  --rules="naming,imports,error-handling" \
  --fix-minor \
  --fail-on-major

# Exit codes:
# 0 = All good (or auto-fixed minor issues)
# 1 = Major issues found, commit blocked
```

```
# .claude/standards.yaml
naming:
  files: kebab-case
  variables: camelCase
  constants: SCREAMING_SNAKE_CASE

imports:
  order: [builtin, external, internal, relative]
  no-default-export: true

error-handling:
  require-try-catch-in: [controllers, services]
  custom-error-class: AppError
```

ROI Reduces PR style debates by 50%. Consistent codebase = faster reviews.

4 Auto-Documentation Agent

Pain: Outdated docs

Your README says "Run `npm start`" but you switched to pnpm six months ago. API docs are three versions behind. Nobody wants to write docs.

SOLUTION

Merge-triggered agent that automatically updates documentation when code changes. Monitors for structural changes and keeps docs in sync.

[Diagram: PR Merged → Webhook → Claude Doc Agent → Updated README/API Docs → Auto-commit]

```yaml
# .github/workflows/auto-docs.yml
name: Update Documentation
on:
  push:
    branches: [main]
    paths:
      - 'src/**'
      - 'package.json'

jobs:
  update-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run Claude Doc Agent
        run: |
          claude-code agent run doc-updater \
            --scope="README.md,docs/api.md" \
            --context="Recent changes in this commit" \
            --auto-commit
```

```
# Doc Agent Prompt Template
"Review the code changes in this commit. Update documentation:
1. README.md - Commands, setup instructions, dependencies
2. docs/api.md - Endpoint signatures, request/response shapes
3. CHANGELOG.md - Add entry for this change

Rules: Keep existing structure. Only update what changed.
Mark uncertain updates with [REVIEW NEEDED]."
```

ROI Keeps knowledge fresh. Saves 10+ hours/month on manual doc updates.

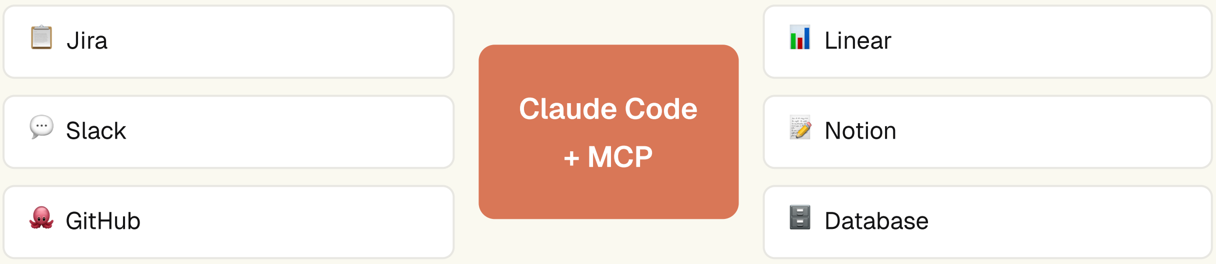
5 Project Sync with MCP Integrations

Pain: Tool-switching (Jira/Slack/GitHub)

"Let me check the ticket..." Tab switch. "I'll post in Slack..." Tab switch. "What was the PR number?" Tab switch. Context switching fragments your flow.

SOLUTION

MCP (Model Context Protocol) servers that connect Claude directly to your tools. Ask questions, update tickets, and post messages without leaving your editor.



```
// claude_config.json - MCP Server Setup
{
  "mcpServers": {
    "jira": {
      "command": "npx",
      "args": ["@anthropic/mcp-server-jira"],
      "env": {
        "JIRA_URL": "https://yourteam.atlassian.net",
        "JIRA_TOKEN": "${JIRA_API_TOKEN}"
      }
    },
    "slack": {
      "command": "npx",
      "args": ["@anthropic/mcp-server-slack"],
      "env": {
        "SLACK_TOKEN": "${SLACK_BOT_TOKEN}"
      }
    }
  }
}
```

```
# Now you can ask Claude:
"What's the status of PROJ-123?"
"Post a summary of my changes to #engineering"
"Create a ticket for this bug I just found"
```

ROI Cuts context switches by 60%. Stay in flow state longer.

6 Security Auditor Agent

Pain: Vulnerabilities in AI-assisted code

AI generates code fast. Sometimes too fast. Hardcoded API keys, SQL injection vectors, and XSS vulnerabilities slip through when you're moving quickly.

SOLUTION

Dedicated security agent that scans every AI-generated change for common vulnerabilities before it reaches production.

[Diagram: Code Change → Security Agent Scan → Risk Report → Gate: Pass/Block/Review]

```
# Security Audit Prompt Template
"Scan this code for security vulnerabilities:

CHECK LIST:
□ Hardcoded secrets (API keys, passwords, tokens)
□ SQL injection vectors (string concatenation in queries)
□ XSS vulnerabilities (unsanitized user input in HTML)
□ Path traversal (user input in file paths)
□ Insecure dependencies (known CVEs)
□ Missing authentication/authorization checks
□ Sensitive data exposure (PII in logs, error messages)

OUTPUT FORMAT:
{
  "severity": "critical|high|medium|low",
  "location": "file:line",
  "issue": "description",
  "fix": "recommendation"
}

If no issues: { "status": "clean", "confidence": 0.0-1.0 }"
```

⚠ **Important**

Never rely solely on AI for security. Use this as a first-pass filter, not a replacement for proper security audits and penetration testing.

ROI Prevents costly breaches. Compliance boost for SOC2/ISO audits.

7 Multi-Feature Execution Loop

Pain: Sequential planning limits parallel work

You have 5 features to ship this sprint. But Claude can only work on one thing at a time... or can it?

SOLUTION

A custom Claude skill that transforms a planning document into parallel execution. It breaks down your roadmap into independent tasks, assigns sub-agents, and loops until completion—enabling 3-5 features in parallel.

How It Works

1. **Input:** Upload a planning doc (feature specs, roadmap)
2. **Analysis:** Claude breaks work into independent tasks
3. **Delegation:** Sub-agents assigned to each task
4. **Execution Loop:** Tasks run in parallel, with iteration on failures
5. **Output:** Merged code branches ready for review

```
# Execution Loop Skill Prompt
"Analyze this plan: [PLAN_DOCUMENT]

PROCESS:
1. Break into independent tasks (no cross-dependencies)
2. For each task, create a sub-agent:
   - /code-gen for implementation
   - /test for test coverage
   - /doc for documentation
3. Execute all tasks in parallel
4. On failure: retry with modified approach (max 3 attempts)
5. On success: commit to feature branch
6. Loop until all tasks complete

OUTPUT: Summary of completed work + branch names for merging"
```

Git Worktree for Parallel Dev

Pair this with Git worktrees—each feature gets its own isolated checkout:

```
main (primary worktree)
   ../feature1-worktree → feature1-branch
   ../feature2-worktree → feature2-branch
```

```
../feature3-worktree → feature3-branch
```

```
# Setup worktrees for parallel features
git worktree add ../feature1-wt feature1-branch
git worktree add ../feature2-wt feature2-branch

# Each Claude agent works in its own worktree
# No stashing, no branch switching, no conflicts

# Merge when ready
git merge feature1-branch
git worktree remove ../feature1-wt
```

Build #7 Continued: Safety & Setup

⚠️ **Safety Constraints**

Guardrails you MUST implement:

- Token/cost caps in prompts (prevent runaway spending)
- Error-handling loops with max retry limits
- Human-review gates before any merge to main
- CI checks that block broken builds

Security Considerations:

- Use enterprise Claude accounts with data isolation
- Never put sensitive code (secrets, PII) in prompts
- Integrate with secure MCP for internal tools only
- Audit logs for all agent actions

Setup Steps:

1. Install Claude Code plugin for agents
2. Configure Git worktrees in repo
3. Test loop on a dummy feature first
4. Monitor via logs (output to Slack/Discord)

Limits

- Max 5 parallel tasks (more = diminishing returns + overload)
- Always `git worktree prune` after merges
- Enforce CI checks on all agent-generated code
- Review all merges—AI is a multiplier, not a replacement

ROI Handle 3-5 features in parallel. Saves weeks per sprint cycle.

Putting It All Together

Implementation Order

#2 CLAUDE.md → #3 Hooks → #1 Review → #6 Security → #5 MCP → #7 Loops → #4 Docs

Start with #2 (CLAUDE.md) — it's free, takes 30 minutes, and immediately improves every Claude interaction.

Best Practices:

- Version control your plugins and prompts
- Document what each agent does for the team
- Run training sessions when rolling out new builds
- Measure before/after metrics to prove ROI

Scale Your Team's Claude Code

You've now got 7 battle-tested builds to transform Claude Code from an individual productivity hack into a team-wide superpower.

€5,000 - €15,000

Estimated Monthly Savings (5-person engineering team)

What These Builds Unlock

- **Compounding returns** — Each build makes the others more effective
- **Consistent quality** — Standards enforced automatically, not by debate
- **Faster shipping** — Parallel work + automated reviews = shorter cycles
- **Lower risk** — Security checks catch issues before production
- **Better onboarding** — New hires productive in days with CLAUDE.md

Get Custom ROI Projections for Your Team

The €997 Team Audit includes:

- 90-minute deep-dive into your current workflow
- Prioritized build recommendations for your stack
- Custom ROI calculations based on your team size
- Implementation roadmap with timelines

Book Your Team Audit →

Bonus

Reply to your download email for a **free 15-minute plugin consultation**. I'll answer your specific questions about implementing any of these builds.

codecoast labs GmbH

julian@codecoast.ch | @codecoast

codecoast.ch