**Claude Code Templates: Deployment Requirements & Asset Checklist**

**Comprehensive Guide to Infrastructure, Tools, and Platforms Required for Successful Deployment**

**Executive Summary**

**Overview**

Successfully deploying Claude Code Templates requires a modern development environment with specific tools, platforms, and infrastructure. This document provides a complete asset checklist to assess your organization's readiness and identify any gaps before implementation.

**Asset Categories**

The templates require assets across six major categories:

1. **Version Control & CI/CD** - Git, GitHub/GitLab, automation pipelines
2. **Development Stack** - Languages, frameworks, databases, authentication
3. **Cloud Infrastructure** - Hosting, deployment, serverless functions
4. **Project Management** - Task tracking, documentation, collaboration
5. **Security & Compliance** - Secrets management, monitoring, audit logging
6. **Testing & Quality** - Test frameworks, coverage tools, linting

**Minimum Viable Deployment (Quick Start)**

To use the templates at a basic level, you need:

* ✅ **Git repository** (GitHub, GitLab, or Bitbucket)
* ✅ **One programming language** (TypeScript, Python, Java, etc.)
* ✅ **One framework** (Next.js, Django, Express, etc.)
* ✅ **Package manager** (npm, pip, Maven, etc.)
* ✅ **Test framework** (Jest, Pytest, JUnit, etc.)
* ✅ **Text editor/IDE** (VS Code, IntelliJ, PyCharm, etc.)
* ✅ **Claude Code CLI** installed and configured

**Estimated Setup Time**: 10-15 minutes

**Recommended Full Deployment**

For maximum value and all features:

* ✅ All minimum requirements above
* ✅ **Task management system** (Linear, Jira, GitHub Issues)
* ✅ **Database system** (PostgreSQL, MongoDB, MySQL)
* ✅ **Authentication provider** (Auth0, Supabase, Firebase)
* ✅ **CI/CD pipeline** (GitHub Actions, GitLab CI, Jenkins)
* ✅ **Cloud platform** (AWS, Vercel, Google Cloud, Azure)
* ✅ **Monitoring/logging** (DataDog, LogRocket, Sentry)
* ✅ **API documentation tool** (Swagger/OpenAPI, Postman)

**Estimated Setup Time**: 1-2 hours (if infrastructure exists), 1-2 days (from scratch)

**Quick Readiness Assessment**

Check your readiness level:

| **Readiness Level** | **Requirements Met** | **Templates Available** | **Business Impact** |
| --- | --- | --- | --- |
| **Basic** | Git + Language + Framework | 40% (8/21 templates) | 20-30% efficiency gain |
| **Standard** | Basic + Database + Tests | 70% (15/21 templates) | 40-50% efficiency gain |
| **Advanced** | Standard + CI/CD + Task Mgmt | 85% (18/21 templates) | 50-60% efficiency gain |
| **Enterprise** | All recommended assets | 100% (21/21 templates) | 60-70% efficiency gain |

**Part 1: Core Infrastructure Requirements**

**1.1 Version Control Systems**

**Required for**: All templates

**Options**:

* **GitHub** (Recommended) - Best Claude Code integration, Actions for CI/CD
* **GitLab** - Built-in CI/CD, self-hosted option
* **Bitbucket** - Atlassian ecosystem integration
* **Azure DevOps** - Microsoft ecosystem

**Required Features**:

* Git repository hosting
* Pull request/merge request workflows
* Branch protection rules
* Code review capabilities

**Used By**:

* All agent templates for code analysis
* Start Task Command for branch management
* Reset Branch Command for git operations
* Code Reviewer Agent for diff analysis

**Example Configuration**:

# GitHub repository

https://github.com/your-org/your-project

# Branch structure

main (protected)

├── develop

├── feature/\*

├── bugfix/\*

└── hotfix/\*

**1.2 CI/CD Platforms**

**Required for**: Automation commands, quality assurance agents

**Options**:

* **GitHub Actions** (Recommended for GitHub users)
* **GitLab CI/CD** (Recommended for GitLab users)
* **Jenkins** (Self-hosted, highly customizable)
* **CircleCI** (Cloud-based, fast builds)
* **Travis CI** (Open source friendly)

**Required Features**:

* Automated testing on commits/PRs
* Code quality checks (linting, type checking)
* Security scanning
* Deployment automation

**Used By**:

* API Test Command (test automation)
* Security Audit Command (automated scans)
* Code Reviewer Agent (pre-commit hooks)

**Example Configuration**:

# .github/workflows/ci.yml

name: CI

on: [push, pull\_request]

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- run: npm install

- run: npm test

- run: npm run lint

**1.3 Cloud & Deployment Platforms**

**Required for**: Backend agents, security auditor, deployment commands

**Options**:

* **Vercel** - Next.js optimized, serverless, edge functions
* **AWS** - Comprehensive services, Lambda, EC2, RDS
* **Google Cloud** - Cloud Run, Firebase, App Engine
* **Azure** - Microsoft ecosystem, Azure Functions
* **Heroku** - Simple deployment, Postgres add-ons
* **Netlify** - JAMstack, serverless functions
* **Railway** - Simple deployment, databases included

**Required Features**:

* Application hosting
* Environment variable management
* SSL/TLS certificates
* Logging and monitoring

**Used By**:

* Backend Architect Agent (deployment patterns)
* Security Auditor Agent (platform security)
* Database Admin Agent (managed databases)

**Example Stack**:

Frontend: Vercel (Next.js)

Backend API: AWS Lambda + API Gateway

Database: AWS RDS (PostgreSQL)

Storage: AWS S3

CDN: CloudFlare

**1.4 Monitoring & Logging**

**Required for**: Production deployments, debugging agents

**Options**:

* **Sentry** - Error tracking and performance monitoring
* **DataDog** - Infrastructure and APM monitoring
* **LogRocket** - Session replay and error tracking
* **New Relic** - Full-stack observability
* **CloudWatch** - AWS-native monitoring
* **Grafana + Prometheus** - Self-hosted monitoring

**Required Features**:

* Error tracking and alerting
* Performance monitoring
* Log aggregation
* User session tracking

**Used By**:

* Debugger Agent (error analysis)
* Backend Architect Agent (performance patterns)
* Database Optimizer Agent (query performance)

**1.5 Secrets Management**

**Required for**: All production deployments, security agents

**Options**:

* **Environment Variables** (Basic) - Platform-provided (Vercel, Heroku)
* **AWS Secrets Manager** - AWS-native secrets
* **HashiCorp Vault** - Enterprise secrets management
* **Azure Key Vault** - Azure-native secrets
* **1Password Secrets** - Team password manager integration
* **Doppler** - Developer-friendly secrets management

**Required Features**:

* Secure storage of API keys and credentials
* Environment-based secret management
* Rotation capabilities
* Audit logging

**Used By**:

* Security Auditor Agent (secrets scanning)
* Backend Architect Agent (secure patterns)

**Part 2: Development Stack Requirements**

**2.1 Programming Languages & Runtimes**

**Required**: At least one language

**Supported Languages**:

| **Language** | **Version** | **Package Manager** | **Use Cases** |
| --- | --- | --- | --- |
| **TypeScript/JavaScript** | Node.js 18+ | npm, yarn, pnpm | Full-stack web, APIs, serverless |
| **Python** | 3.9+ | pip, poetry, conda | Backend, data, ML, APIs |
| **Java** | 11+ | Maven, Gradle | Enterprise backend, microservices |
| **Go** | 1.19+ | go modules | High-performance APIs, microservices |
| **Ruby** | 3.0+ | bundler | Web apps, APIs (Rails) |
| **C#/.NET** | .NET 6+ | NuGet | Enterprise apps, Azure |
| **PHP** | 8.0+ | Composer | Web apps (Laravel, Symfony) |
| **Rust** | 1.65+ | Cargo | Systems programming, performance |

**Used By**: All development-focused agents and commands

**2.2 Web Frameworks**

**Required**: At least one framework for web projects

**Frontend Frameworks**:

* **Next.js** (React) - SSR, SSG, API routes, recommended
* **React** - SPA, component-based
* **Vue.js** - Progressive framework
* **Angular** - Enterprise SPA framework
* **Svelte/SvelteKit** - Compiler-based framework
* **Remix** - React framework with web fundamentals

**Backend Frameworks**:

* **Express.js** (Node.js) - Minimal, flexible
* **Fastify** (Node.js) - High performance
* **Django** (Python) - Batteries-included
* **FastAPI** (Python) - Modern, async, type-safe
* **Flask** (Python) - Lightweight, flexible
* **Spring Boot** (Java) - Enterprise Java
* **Ruby on Rails** - Convention over configuration
* **Laravel** (PHP) - Modern PHP framework
* **.NET Core** - Cross-platform .NET

**Used By**:

* JavaScript Pro Agent
* Backend Architect Agent
* API Documenter Agent
* Security Auditor Agent

**Example**: Next.js 14 with App Router

**2.3 Databases & Data Storage**

**Required**: For data-driven applications

**Relational Databases**:

* **PostgreSQL** (Recommended) - Open source, feature-rich
* **MySQL/MariaDB** - Popular, widely supported
* **SQLite** - Embedded, serverless
* **SQL Server** - Microsoft ecosystem
* **Oracle** - Enterprise databases

**NoSQL Databases**:

* **MongoDB** - Document database
* **Redis** - In-memory cache/database
* **DynamoDB** - AWS serverless database
* **Firebase Firestore** - Real-time database
* **Cassandra** - Distributed database

**Database-as-a-Service**:

* **Supabase** (Recommended) - PostgreSQL + Auth + Storage
* **PlanetScale** - MySQL-compatible, serverless
* **MongoDB Atlas** - Managed MongoDB
* **Firebase** - Google's mobile/web platform
* **Fauna** - Serverless document-relational

**Used By**:

* Database Admin Agent
* Database Optimizer Agent
* Backend Architect Agent
* Security Auditor Agent (RLS policies)

**Example Stack**: Supabase (PostgreSQL + Auth + Storage + Edge Functions)

**2.4 ORM/Query Builders**

**Required**: For type-safe database access

**Options**:

* **Prisma** (TypeScript) - Type-safe, migrations, recommended
* **Drizzle** (TypeScript) - Lightweight, SQL-like
* **TypeORM** (TypeScript) - Mature ORM
* **Sequelize** (JavaScript) - Promise-based ORM
* **SQLAlchemy** (Python) - Python SQL toolkit
* **Django ORM** (Python) - Built into Django
* **Hibernate** (Java) - Java persistence
* **Entity Framework** (.NET) - .NET ORM

**Used By**:

* Backend Architect Agent (query patterns)
* Database Admin Agent (schema design)
* Database Optimizer Agent (query optimization)

**2.5 Authentication & Authorization**

**Required**: For applications with user management

**Options**:

* **Supabase Auth** - Built-in with Supabase, OAuth, magic links
* **Auth0** - Enterprise auth, SSO, MFA
* **Firebase Auth** - Google ecosystem, easy setup
* **AWS Cognito** - AWS-native user management
* **NextAuth.js** - Next.js authentication
* **Passport.js** - Node.js authentication middleware
* **Clerk** - Modern authentication for React
* **Custom JWT** - Roll your own (not recommended)

**Required Features**:

* User registration/login
* Password reset flows
* Session management
* OAuth providers (Google, GitHub, etc.)
* Role-based access control (RBAC)

**Used By**:

* Backend Architect Agent (auth patterns)
* Security Auditor Agent (auth security)
* API Test Command (auth mocking)

**Example**: Supabase Auth with Google OAuth + Magic Links

**2.6 Testing Frameworks**

**Required**: For API Test Command and quality assurance

**JavaScript/TypeScript**:

* **Jest** (Recommended) - Full-featured, widely used
* **Vitest** - Fast, Vite-native
* **Mocha + Chai** - Flexible, assertion library
* **Playwright** - E2E testing
* **Cypress** - E2E testing

**Python**:

* **Pytest** (Recommended) - Feature-rich, plugins
* **unittest** - Built-in Python testing
* **Selenium** - E2E testing

**Java**:

* **JUnit** - Standard Java testing
* **TestNG** - Advanced testing framework
* **Mockito** - Mocking framework

**Other Languages**:

* **RSpec** (Ruby) - Behavior-driven testing
* **PHPUnit** (PHP) - PHP unit testing
* **xUnit** (.NET) - .NET testing framework

**Used By**:

* API Test Command (primary user)
* Code Reviewer Agent (coverage checks)
* Debugger Agent (test analysis)

**Example**: Jest with TypeScript for API testing

**2.7 API Documentation Tools**

**Required**: For API Documenter Agent

**Options**:

* **OpenAPI/Swagger** (Recommended) - Standard API specs
* **Postman** - API development platform
* **Insomnia** - API client and design
* **Redoc** - OpenAPI documentation
* **API Blueprint** - Markdown-based API design
* **GraphQL Playground** - GraphQL APIs
* **RapiDoc** - Web component for OpenAPI

**Used By**:

* API Documenter Agent (generates docs)
* Backend Architect Agent (API design)

**Example**: OpenAPI 3.0 spec with Swagger UI

**Part 3: Project Management & Collaboration Tools**

**3.1 Task Management Systems**

**Required**: For Product Manager, Project Manager, Scrum Master agents, Start Task Command

**Options**:

| **Platform** | **Best For** | **Integration** | **Pricing** |
| --- | --- | --- | --- |
| **Linear** | Modern teams, fast workflow | API, GitHub sync | $8/user/month |
| **Jira** | Enterprise, complex workflows | Extensive integrations | $7.75/user/month |
| **GitHub Issues** | Open source, simple workflow | Native GitHub | Free with GitHub |
| **Asana** | General project management | Many integrations | $10.99/user/month |
| **ClickUp** | All-in-one work management | Extensive integrations | $7/user/month |
| [**Monday.com**](http://monday.com/) | Visual project tracking | 200+ integrations | $8/seat/month |
| **Azure Boards** | Microsoft ecosystem | Azure DevOps | Included with Azure |

**Required Features**:

* Issue/ticket creation and tracking
* Project organization
* Status workflows
* API access for automation
* GitHub/GitLab integration

**Used By**:

* **Product Manager Agent** - Creates Linear projects and stories
* **Project Manager Agent** - Task coordination
* **Scrum Master Agent** - Sprint planning
* **Start Task Command** - Task initialization

**Example**: Linear with GitHub sync

**API Requirements**:

// Linear API for Product Manager Agent

{

createProject: (name, description, teamId) => Project

createIssue: (title, description, projectId, labels) => Issue

updateIssue: (issueId, updates) => Issue

getTeams: () => Team[]

}

**3.2 Documentation Platforms**

**Required**: For maintaining project knowledge

**Options**:

* **Notion** - All-in-one workspace, wikis
* **Confluence** - Atlassian knowledge base
* **GitBook** - Modern documentation
* **ReadMe** - API documentation
* **Docusaurus** - Static site generator
* **VitePress** - Vue-powered static site
* **MkDocs** - Python documentation

**Used By**:

* API Documenter Agent
* Codebase Explorer Agent
* All agents ([CLAUDE.md](http://claude.md/) reference)

**Example**: Notion for team wiki, GitBook for public docs

**3.3 Design & Mockup Tools**

**Required**: For Product Manager Agent (screenshot analysis)

**Options**:

* **Figma** (Recommended) - Collaborative design
* **Sketch** - Mac design tool
* **Adobe XD** - Adobe ecosystem
* **InVision** - Prototyping and collaboration
* **Balsamiq** - Wireframing
* **Miro** - Whiteboarding and planning

**Used By**:

* Product Manager Agent (analyzes mockups/screenshots)

**Example**: Figma designs exported as screenshots for analysis

**3.4 Communication Tools**

**Optional**: For team coordination

**Options**:

* **Slack** - Team messaging, integrations
* **Microsoft Teams** - Microsoft ecosystem
* **Discord** - Developer communities
* **Mattermost** - Self-hosted Slack alternative

**Used By**: Team coordination around template outputs

**Part 4: Per-Template Detailed Requirements**

**Template Category: Development Agents**

**1. JavaScript Pro Agent**

**Technical Requirements**:

* **Required**:
  + Node.js 18+ runtime
  + TypeScript 5.0+ or JavaScript ES2022+
  + Package manager: npm, yarn, or pnpm
  + Text editor/IDE with TypeScript support
* **Optional**:
  + React/Vue/Angular framework
  + ESLint for linting
  + Prettier for formatting
  + TypeScript strict mode enabled

**Platform Requirements**:

* Git repository
* Node.js environment

**Configuration Needs**:

* tsconfig.json or jsconfig.json
* package.json with scripts
* ESLint/Prettier configs

**Example Setup**:

// package.json

{

"name": "my-project",

"scripts": {

"dev": "next dev",

"build": "next build",

"test": "jest",

"lint": "eslint ."

},

"dependencies": {

"next": "^14.0.0",

"react": "^18.0.0",

"typescript": "^5.0.0"

}

}

**2. Backend Architect Agent**

**Technical Requirements**:

* **Required**:
  + Backend framework (Express, FastAPI, Django, etc.)
  + Database system (PostgreSQL, MongoDB, MySQL)
  + API design patterns (REST, GraphQL, gRPC)
  + Package manager for dependencies
* **Optional**:
  + ORM/ODM (Prisma, SQLAlchemy, TypeORM)
  + API documentation (OpenAPI/Swagger)
  + Rate limiting library
  + Queue system (BullMQ, Celery, SQS)

**Platform Requirements**:

* Database hosting (local or cloud)
* API hosting platform
* Environment variable management

**Configuration Needs**:

* Database connection string
* API route structure
* Authentication system configuration
* Logging library setup

**Placeholder Customization**:

// Template placeholders to fill:

[BACKEND\_FRAMEWORK] = "Express.js"

[DATABASE\_SYSTEM] = "PostgreSQL"

[AUTH\_SYSTEM] = "Auth0"

[ORM\_TOOL] = "Prisma"

[API\_STRUCTURE] = "/api/v1"

[RATE\_LIMIT\_LIB] = "express-rate-limit"

[QUEUE\_SYSTEM] = "BullMQ"

**3. Code Reviewer Agent**

**Technical Requirements**:

* **Required**:
  + Git repository
  + Programming language (any supported)
  + Code formatting tool (Prettier, Black, gofmt)
  + Linting tool (ESLint, Pylint, etc.)
* **Optional**:
  + CI/CD for automated reviews
  + Code coverage tools
  + Static analysis tools (SonarQube)

**Platform Requirements**:

* Git with diff support
* Pull request workflow

**Configuration Needs**:

* Linting rules configuration
* Code style guidelines
* Maximum complexity thresholds
* Coverage requirements

**Placeholder Customization**:

[PROJECT\_NAME] = "MyApp"

[PRIMARY\_FRAMEWORK] = "Next.js 14"

[LANGUAGE\_RUNTIME] = "TypeScript"

[MAX\_FUNCTION\_LINES] = 30

[MAX\_FILE\_LINES] = 200

[MIN\_CODE\_COVERAGE] = 80

**4. Debugger Agent**

**Technical Requirements**:

* **Required**:
  + Runtime with debugging support
  + Error tracking/logging system
  + Source maps (for compiled languages)
* **Optional**:
  + Sentry or similar error tracking
  + Performance monitoring (DataDog, New Relic)
  + Logging aggregation (Splunk, ELK)

**Platform Requirements**:

* Development environment
* Production logging access

**Configuration Needs**:

* Error tracking API keys
* Log levels configuration
* Source map generation

**5. Database Admin Agent**

**Technical Requirements**:

* **Required**:
  + Database system (PostgreSQL, MySQL, MongoDB, etc.)
  + Migration tool (Prisma Migrate, Alembic, Flyway)
  + Database client/ORM
* **Optional**:
  + Database GUI (pgAdmin, MongoDB Compass)
  + Backup solution
  + Replication setup

**Platform Requirements**:

* Database hosting (local, RDS, managed service)
* Migration environment

**Configuration Needs**:

* Database connection credentials
* Migration directory structure
* Schema naming conventions

**Example Setup**:

# Prisma setup

DATABASE\_URL="postgresql://user:pass@localhost:5432/mydb"

# Migrations directory

prisma/migrations/

**6. Database Optimizer Agent**

**Technical Requirements**:

* **Required**:
  + Same as Database Admin Agent
  + Query analysis tools
  + Performance monitoring
* **Optional**:
  + Query profiling tools (EXPLAIN, pg\_stat\_statements)
  + Index advisor tools
  + Connection pooling (PgBouncer)

**Platform Requirements**:

* Database with slow query logging
* Performance metrics access

**Configuration Needs**:

* Slow query threshold
* Index strategy
* Query optimization tools

**7. Security Auditor Agent**

**Technical Requirements**:

* **Required**:
  + Full application stack access
  + Authentication system
  + Database with security policies
* **Optional**:
  + Dependency scanning (Snyk, npm audit)
  + SAST tools (SonarQube, Checkmarx)
  + Secrets scanning (GitGuardian, TruffleHog)
  + Penetration testing tools

**Platform Requirements**:

* Production-like environment for testing
* Access to all security configurations

**Configuration Needs**:

* Authentication provider details
* API security configurations
* Database security policies (RLS)
* CORS settings
* Environment variable patterns

**Placeholder Customization**:

{PROJECT\_NAME} = "MyApp"

{AUTHENTICATION\_METHOD} = "Supabase Auth"

{API\_KEY\_TYPE} = "SDK keys"

{DATABASE\_SYSTEM} = "Supabase (PostgreSQL)"

{SENSITIVE\_DATA\_TYPES} = "user feedback, PII"

{INTEGRATION\_SYSTEMS} = "Stripe, SendGrid"

{COMPLIANCE\_REQUIREMENTS} = "GDPR, SOC2"

{WIDGET\_OR\_SDK} = "feedback widget"

**Template Category: Data & Analytics Agents**

**8. Data Engineer Agent**

**Technical Requirements**:

* **Required**:
  + Python 3.9+ or similar data processing language
  + ETL framework (Apache Airflow, Luigi, Dagster)
  + Data warehouse (PostgreSQL, BigQuery, Snowflake)
  + Data transformation tool (dbt, SQL)
* **Optional**:
  + Message queue (Kafka, RabbitMQ)
  + Data quality tools (Great Expectations)
  + Orchestration platform (Airflow, Prefect)

**Platform Requirements**:

* Data warehouse hosting
* ETL pipeline infrastructure
* Scheduling system

**Configuration Needs**:

* Data source connections
* Transformation logic
* Pipeline schedules

**9. Data Scientist Agent**

**Technical Requirements**:

* **Required**:
  + Python 3.9+ with data science libraries
  + Jupyter notebooks or similar
  + Data visualization (Matplotlib, Plotly)
  + ML libraries (scikit-learn, TensorFlow, PyTorch)
* **Optional**:
  + ML platform (SageMaker, Vertex AI)
  + Experiment tracking (MLflow, Weights & Biases)
  + Model deployment (Seldon, KFServing)

**Platform Requirements**:

* Data access
* Compute resources for training
* Model serving infrastructure

**Configuration Needs**:

* Data warehouse credentials
* Model registry
* Feature store (optional)

**Template Category: Project Management Agents**

**10. Product Manager Agent**

**Technical Requirements**:

* **Required**:
  + Task management system with API (Linear, Jira)
  + Image analysis capability (Claude's vision)
  + Markdown support
* **Optional**:
  + Figma/design tool integration
  + User analytics platform
  + Customer feedback system

**Platform Requirements**:

* Linear/Jira account with API access
* Design tool with export capability

**Configuration Needs**:

* Linear/Jira API token
* Team ID
* Project templates

**Placeholder Customization**:

[PRODUCT\_NAME] = "MyApp"

[CORE\_MISSION] = "Help B2B SaaS collect customer feedback"

[KEY\_FEATURES] = "Feedback widget, SDK integration, CRM sync"

[USER\_PERSONAS] = "Product managers, customer success teams"

[TEAM\_ID] = "your-linear-team-id"

[BUSINESS\_DOMAIN] = "SaaS analytics"

**API Access Example**:

// Linear API setup

LINEAR\_API\_KEY="lin\_api\_xxx"

LINEAR\_TEAM\_ID="team-id-123"

**11. Project Manager Agent**

**Technical Requirements**:

* **Required**:
  + Task management system (Linear, Jira, GitHub Issues)
  + Git repository
  + Team communication tool
* **Optional**:
  + Time tracking (Harvest, Toggl)
  + Resource planning (Float, Forecast)

**Platform Requirements**:

* Task management platform access
* Team calendar/scheduling

**Configuration Needs**:

* Task management API credentials
* Project structure templates
* Team member assignments

**12. Scrum Master Agent**

**Technical Requirements**:

* **Required**:
  + Same as Project Manager Agent
  + Sprint planning capability in task system
  + Retrospective tools
* **Optional**:
  + Burndown chart generation
  + Velocity tracking
  + Team health metrics

**Platform Requirements**:

* Agile-compatible task management (Jira, Azure Boards)

**Configuration Needs**:

* Sprint duration
* Team capacity
* Story point scale

**Template Category: Documentation Agents**

**13. API Documenter Agent**

**Technical Requirements**:

* **Required**:
  + API framework with introspection
  + Documentation format (OpenAPI, GraphQL schema)
  + Markdown support
* **Optional**:
  + API documentation hosting (ReadMe, Swagger UI)
  + API testing tools (Postman, Insomnia)
  + SDK generation tools

**Platform Requirements**:

* Documentation hosting
* API specification storage

**Configuration Needs**:

* API base URL
* Authentication schemes
* Response schemas

**Example Output**:

# OpenAPI 3.0 specification

openapi: 3.0.0

info:

title: My API

version: 1.0.0

paths:

/api/users:

get:

summary: List users

responses:

200:

description: Success

**14. Codebase Explorer Agent**

**Technical Requirements**:

* **Required**:
  + Git repository access
  + Programming language support
  + Code analysis tools
* **Optional**:
  + Dependency graph generation
  + Architecture visualization
  + Documentation generator

**Platform Requirements**:

* Repository access
* File system access

**Configuration Needs**:

* Repository structure
* Entry points
* Key modules

**15. Architect Review Agent**

**Technical Requirements**:

* **Required**:
  + Full codebase access
  + Architecture documentation
  + Design patterns knowledge
* **Optional**:
  + Architecture diagramming tools (C4, PlantUML)
  + Performance testing tools
  + Load testing tools (k6, JMeter)

**Platform Requirements**:

* Codebase access
* Documentation access

**Configuration Needs**:

* Current architecture documentation
* Target architecture goals
* Performance requirements

**Template Category: Automation Commands**

**16. API Test Command**

**Technical Requirements**:

* **Required**:
  + Test framework (Jest, Pytest, etc.)
  + Mocking library (jest.mock, unittest.mock)
  + HTTP request testing (supertest, requests)
  + Coverage tool (Istanbul, [coverage.py](http://coverage.py/))
* **Optional**:
  + Integration test environment
  + Test database
  + CI/CD integration

**Platform Requirements**:

* Test execution environment
* Database for integration tests

**Configuration Needs**:

* Test file patterns
* Coverage thresholds
* Mock configurations

**Placeholder Customization**:

[PROJECT\_NAME] = "MyApp"

[FRAMEWORK] = "Next.js 15"

[TEST\_FRAMEWORK] = "Jest"

[PACKAGE\_MANAGER] = "pnpm"

[TEST\_PATH] = "\_\_tests\_\_/api/"

[API\_BASE\_PATH] = "app/api/"

[DATABASE\_CLIENT] = "Supabase"

[AUTH\_PROVIDER] = "Supabase Auth"

[TABLE\_NAME] = "users"

[MIN\_COVERAGE] = 90

**Required Commands**:

# Must work in your project

pnpm test:coverage

pnpm test:api -- path/to/test

**17. Security Audit Command**

**Technical Requirements**:

* **Required**:
  + Full application stack
  + Security scanning tools
  + Vulnerability database access
* **Optional**:
  + SAST tools (Snyk, SonarQube)
  + DAST tools (OWASP ZAP, Burp Suite)
  + Dependency scanning (npm audit, safety)
  + Secrets scanning (GitGuardian)

**Platform Requirements**:

* Application environment access
* Network access for scanning

**Configuration Needs**:

* Security compliance requirements
* Vulnerability severity thresholds
* Audit log location

**Placeholder Customization**:

[FRAMEWORK] = "Next.js"

[BACKEND\_SERVICE] = "Supabase"

[DATABASE\_SERVICE] = "PostgreSQL"

[DEPLOYMENT\_PLATFORM] = "Vercel"

[AUTH\_PROVIDER] = "Auth0"

[MIDDLEWARE\_FILE] = "middleware.ts"

**Output**:

security\_audit\_log\_2024\_01\_15\_14\_30\_00/

├── authentication/

│ ├── findings.md

│ └── remediation.md

├── api-routes/

│ ├── findings.md

│ └── remediation.md

└── summary.md

**18. Start Task Command**

**Technical Requirements**:

* **Required**:
  + Task management system API
  + Git repository
  + Development environment
* **Optional**:
  + Multiple agent integration
  + Automated code review
  + Test generation

**Platform Requirements**:

* Task management (Linear, Jira, GitHub Issues)
* Git hosting

**Configuration Needs**:

* Task management API credentials
* Task ID prefix format
* Branch naming convention
* Workflow steps

**Placeholder Customization**:

[TASK\_MANAGEMENT\_SYSTEM] = "Linear"

[TASK\_PREFIX] = "PROJ"

[TASK\_TRACKING\_PATH] = ".claude/tasks/"

[TEAM\_KEY] = "proj"

[PACKAGE\_MANAGER] = "pnpm"

**Required API Access**:

// Linear API

{

getIssue: (issueId) => Issue

updateIssue: (issueId, updates) => Issue

addComment: (issueId, comment) => Comment

}

**19. Reset Branch Command**

**Technical Requirements**:

* **Required**:
  + Git repository
  + Git CLI access
  + Branch protection awareness
* **Optional**:
  + Backup creation
  + Stash management

**Platform Requirements**:

* Git repository with remote

**Configuration Needs**:

* Default branch name (main, master, develop)
* Branch protection rules
* Remote name (origin)

**Placeholder Customization**:

[DEFAULT\_BRANCH] = "main"

**Required Git Access**:

git status

git branch

git reset --hard

git clean -fd

git fetch origin

**Template Category: Configuration Templates**

**20.**[**CLAUDE.md**](http://claude.md/)**Template**

**Technical Requirements**:

* **Required**:
  + Text file in project root
  + Markdown support
  + Project documentation access
* **Optional**:
  + Integration with other docs

**Platform Requirements**:

* None (file-based)

**Configuration Needs**:

* Complete project overview
* All technology stack details
* Command reference
* Coding conventions

**Placeholder Customization** (50+ placeholders):

[PROJECT\_NAME] = "MyApp"

[LANGUAGE] = "TypeScript"

[FRAMEWORK] = "Next.js 14"

[DATABASE] = "PostgreSQL"

[PACKAGE\_MANAGER] = "pnpm"

[DEV\_COMMAND] = "dev"

[BUILD\_COMMAND] = "build"

[TEST\_COMMAND] = "test"

[DEPLOYMENT\_PLATFORM] = "Vercel"

... (50+ more placeholders)

**Required Information**:

* All essential commands
* Directory structure
* Naming conventions
* Critical business rules
* Security requirements
* Performance targets

**Part 5: Platform-Specific Integration Requirements**

**Integration: Linear (Task Management)**

**Required By**:

* Product Manager Agent
* Start Task Command

**Setup Steps**:

1. Create Linear account and workspace
2. Generate API key: Settings → API → Personal API keys
3. Get Team ID: Settings → Teams → Copy team identifier
4. Configure webhook (optional): Settings → Webhooks

**API Capabilities Needed**:

# Linear GraphQL API

query {

teams { id, name }

projects { id, name, teamId }

issues { id, title, description, state }

}

mutation {

projectCreate(input: {...})

issueCreate(input: {...})

issueUpdate(id: "...", input: {...})

}

**Environment Variables**:

LINEAR\_API\_KEY="lin\_api\_xxx"

LINEAR\_TEAM\_ID="team-id-123"

LINEAR\_WORKSPACE\_ID="workspace-id"

**Pricing**: $8/user/month (Standard plan)

**Integration: Jira (Task Management)**

**Required By**:

* Product Manager Agent
* Project Manager Agent
* Scrum Master Agent
* Start Task Command

**Setup Steps**:

1. Jira Cloud account
2. Create API token: Account Settings → Security → API tokens
3. Get Project key: Project Settings → Details
4. Configure automation (optional)

**API Capabilities Needed**:

// Jira REST API v3

{

getIssue: (issueKey) => {}

createIssue: (fields) => {}

updateIssue: (issueKey, fields) => {}

transitionIssue: (issueKey, transition) => {}

addComment: (issueKey, comment) => {}

}

**Environment Variables**:

JIRA\_HOST="your-domain.atlassian.net"

JIRA\_USER\_EMAIL="you@company.com"

JIRA\_API\_TOKEN="your-api-token"

JIRA\_PROJECT\_KEY="PROJ"

**Pricing**: $7.75/user/month (Standard plan)

**Integration: GitHub (Version Control + Issues)**

**Required By**:

* All agents (code access)
* Start Task Command (issues)
* Reset Branch Command

**Setup Steps**:

1. GitHub repository
2. Personal access token: Settings → Developer → Personal access tokens
3. Configure branch protection: Repository → Settings → Branches
4. Set up GitHub Actions (optional)

**API Capabilities Needed**:

// GitHub REST API

{

repos: { get, listBranches, getBranch }

pulls: { create, list, get, merge }

issues: { create, update, addLabels, createComment }

git: { getRef, updateRef }

}

**Environment Variables**:

GITHUB\_TOKEN="ghp\_xxx"

GITHUB\_OWNER="your-org"

GITHUB\_REPO="your-repo"

**Pricing**: Free for public repos, $4/user/month for private (Team plan)

**Integration: Supabase (Database + Auth + Storage)**

**Required By**:

* Backend Architect Agent
* Database Admin Agent
* Database Optimizer Agent
* Security Auditor Agent
* API Test Command

**Setup Steps**:

1. Supabase project: [supabase.com](http://supabase.com/) → New Project
2. Get credentials: Settings → API → Project URL and keys
3. Set up Row Level Security policies
4. Configure Auth providers: Authentication → Providers

**API Capabilities Needed**:

// Supabase Client

{

auth: { signUp, signIn, signOut, getUser }

from: (table) => { select, insert, update, delete, upsert }

rpc: (functionName, params) => {}

storage: { from: (bucket) => { upload, download, remove } }

}

**Environment Variables**:

NEXT\_PUBLIC\_SUPABASE\_URL="https://xxx.supabase.co"

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY="eyJxxx"

SUPABASE\_SERVICE\_ROLE\_KEY="eyJxxx" # Server-side only

**Pricing**: Free tier available, Pro $25/month

**Integration: Vercel (Deployment)**

**Required By**:

* Backend Architect Agent (deployment patterns)
* Security Auditor Agent (platform security)

**Setup Steps**:

1. Vercel account: [vercel.com](http://vercel.com/)
2. Connect GitHub repository
3. Configure environment variables: Project → Settings → Environment Variables
4. Set up deployment hooks (optional)

**API Capabilities Needed**:

// Vercel API

{

deployments: { create, get, list }

projects: { get, update }

env: { create, delete }

}

**Environment Variables**:

VERCEL\_TOKEN="xxx"

VERCEL\_ORG\_ID="team\_xxx"

VERCEL\_PROJECT\_ID="prj\_xxx"

**Pricing**: Free for hobby, Pro $20/month, Enterprise custom

**Part 6: Minimum Viable Setup by Use Case**

**Use Case 1: Frontend-Only Project**

**Stack**: Next.js + React + TypeScript

**Required Assets**:

* ✅ Node.js 18+, npm/yarn/pnpm
* ✅ Next.js 14+
* ✅ TypeScript
* ✅ Git + GitHub
* ✅ Jest for testing
* ✅ ESLint + Prettier

**Available Templates**: 12/21 (57%)

* JavaScript Pro Agent ✅
* Code Reviewer Agent ✅
* Debugger Agent ✅
* Codebase Explorer Agent ✅
* Architect Review Agent ✅
* API Documenter Agent ✅ (for API routes)
* API Test Command ✅ (for API routes)
* Reset Branch Command ✅
* [CLAUDE.md](http://claude.md/) Template ✅
* Start Task Command ⚠️ (requires task management)
* Security Audit Command ⚠️ (limited without backend)
* Product Manager Agent ⚠️ (requires Linear/Jira)

**Setup Time**: 10 minutes  
**Efficiency Gain**: 30-40%

**Use Case 2: Full-Stack Web Application**

**Stack**: Next.js + PostgreSQL + Supabase Auth + Vercel

**Required Assets**:

* ✅ All Frontend requirements
* ✅ PostgreSQL database (Supabase)
* ✅ Supabase Auth
* ✅ Vercel deployment
* ✅ Linear or Jira for tasks

**Available Templates**: 18/21 (86%)

* All frontend templates ✅
* Backend Architect Agent ✅
* Database Admin Agent ✅
* Database Optimizer Agent ✅
* Security Auditor Agent ✅
* Product Manager Agent ✅
* Project Manager Agent ✅
* Start Task Command ✅
* Security Audit Command ✅
* Data Engineer Agent ⚠️ (if ETL needed)
* Data Scientist Agent ⚠️ (if analytics needed)
* Scrum Master Agent ⚠️ (if using agile)

**Setup Time**: 1-2 hours  
**Efficiency Gain**: 50-60%

**Use Case 3: Enterprise Application**

**Stack**: Java Spring Boot + PostgreSQL + Azure + Jira

**Required Assets**:

* ✅ Java 11+, Maven/Gradle
* ✅ Spring Boot
* ✅ PostgreSQL (Azure Database)
* ✅ Azure deployment
* ✅ Jira with Agile boards
* ✅ JUnit for testing
* ✅ Azure DevOps for CI/CD

**Available Templates**: 20/21 (95%)

* All applicable agents ✅
* JavaScript Pro Agent ❌ (use Java equivalent)
* Backend Architect Agent ✅
* Database Admin/Optimizer ✅
* Security Auditor Agent ✅
* All project management agents ✅
* All commands ✅

**Setup Time**: 2-4 hours  
**Efficiency Gain**: 60-70%

**Use Case 4: Data Platform**

**Stack**: Python + Airflow + PostgreSQL + dbt + Snowflake

**Required Assets**:

* ✅ Python 3.9+, pip/poetry
* ✅ Apache Airflow
* ✅ PostgreSQL + Snowflake
* ✅ dbt for transformations
* ✅ Pytest for testing
* ✅ Git + GitHub

**Available Templates**: 14/21 (67%)

* Data Engineer Agent ✅
* Data Scientist Agent ✅
* Database Admin Agent ✅
* Database Optimizer Agent ✅
* Code Reviewer Agent ✅
* Debugger Agent ✅
* Security Auditor Agent ✅
* Codebase Explorer Agent ✅
* API Test Command ⚠️ (if APIs exist)
* JavaScript Pro Agent ❌
* Product Manager Agent ⚠️ (requires Linear/Jira)

**Setup Time**: 1-2 hours  
**Efficiency Gain**: 40-50%

**Part 7: Quick Start Checklist**

**Phase 1: Assessment (15 minutes)**

* Identify your primary programming language
* Identify your web framework (if applicable)
* Identify your database system (if applicable)
* Check if you have a task management system
* Verify Git repository access
* Confirm test framework availability
* Check CI/CD setup status

**Phase 2: Template Selection (10 minutes)**

* Review all 21 templates
* Mark templates that match your stack
* Identify templates requiring additional setup
* Prioritize templates by business value
* Create implementation order

**Phase 3: Environment Setup (30-60 minutes)**

* Install Claude Code CLI
* Create .claude/agents/ directory
* Create .claude/commands/ directory
* Copy selected templates
* Set up API keys for integrations (Linear, Jira, etc.)
* Configure environment variables

**Phase 4: Customization (15-30 minutes)**

**Option A: Automatic** (Recommended)

* Run automatic customization prompt in Claude Code
* Review generated configurations
* Validate placeholders were replaced correctly

**Option B: Manual**

* Open each template
* Replace all [PLACEHOLDER] values
* Validate syntax and paths
* Test with simple examples

**Phase 5: Validation (15 minutes)**

* Test JavaScript Pro/Backend Agent with simple code
* Test Code Reviewer Agent with a sample commit
* Test API Test Command on one endpoint (if applicable)
* Verify task management integration (if applicable)
* Run Security Audit Command (if applicable)

**Phase 6: Team Rollout (1-2 hours)**

* Document your specific configuration
* Create team training session
* Share example workflows
* Establish best practices
* Monitor adoption and gather feedback

**Part 8: Troubleshooting Common Setup Issues**

**Issue: "Template placeholders not working"**

**Cause**: Placeholders not replaced with actual values

**Solution**:

1. Search for [ in template files
2. Replace all [PLACEHOLDER] with actual values
3. Use automatic customization prompt

**Issue: "API Test Command fails"**

**Cause**: Missing test dependencies or incorrect paths

**Solution**:

# Check test framework installed

npm list jest # or pytest, etc.

# Verify test paths in template

[TEST\_PATH] = "\_\_tests\_\_/" # Must match actual directory

[API\_BASE\_PATH] = "app/api/" # Must match actual directory

# Check test command works

npm test # or pnpm test, etc.

**Issue: "Product Manager Agent can't create issues"**

**Cause**: Missing or incorrect API credentials

**Solution**:

# Linear

export LINEAR\_API\_KEY="lin\_api\_xxx"

export LINEAR\_TEAM\_ID="team-xxx"

# Jira

export JIRA\_HOST="your-domain.atlassian.net"

export JIRA\_USER\_EMAIL="you@company.com"

export JIRA\_API\_TOKEN="token-xxx"

# Test API access

curl -H "Authorization: Bearer $LINEAR\_API\_KEY" \

https://api.linear.app/graphql

**Issue: "Security Audit finds false positives"**

**Cause**: Template not customized for your stack

**Solution**:

1. Review security template placeholders
2. Remove sections that don't apply
3. Customize for your specific auth/database system
4. Add exceptions for known patterns

**Summary: Readiness Matrix**

| **Asset Category** | **Minimum Setup** | **Standard Setup** | **Advanced Setup** | **Enterprise Setup** |
| --- | --- | --- | --- | --- |
| **Version Control** | Git | Git + GitHub | Git + GitHub + Actions | Git + GitHub + Advanced Actions |
| **Language** | 1 language | 1-2 languages | 2+ languages | Multiple languages |
| **Framework** | 1 framework | 1-2 frameworks | Multiple frameworks | Enterprise frameworks |
| **Database** | Optional | 1 database | 1+ databases | Multiple databases |
| **Auth** | Optional | Basic auth | OAuth + SSO | Enterprise SSO |
| **Task Management** | Optional | GitHub Issues | Linear/Jira | Jira with Agile |
| **CI/CD** | Optional | Basic CI | Full CI/CD | Enterprise CI/CD |
| **Monitoring** | Optional | Basic logging | Full monitoring | Enterprise observability |
| **Testing** | Basic | 70%+ coverage | 90%+ coverage | 95%+ coverage |
| **Documentation** | README | Basic docs | Full documentation | Enterprise docs |
| **Templates Available** | 8-10/21 | 15-17/21 | 18-20/21 | 21/21 |
| **Setup Time** | 15 minutes | 1-2 hours | 2-4 hours | 1-2 days |
| **Efficiency Gain** | 20-30% | 40-50% | 50-60% | 60-70% |
| **Investment** | Minimal | Low | Medium | High |
| **ROI Timeline** | 2-4 weeks | 1-2 weeks | 1 week | Immediate |

**Conclusion**

Claude Code Templates are designed to work with modern development stacks and can be deployed incrementally. Start with the templates that match your current infrastructure, then expand as you add capabilities.

**Key Takeaways**:

1. **Minimum 6 assets needed** for basic deployment (Git, language, framework, package manager, test framework, IDE)
2. **12-15 assets recommended** for standard deployment (add database, auth, task management)
3. **20+ assets optimal** for enterprise deployment (add CI/CD, monitoring, full integrations)
4. **Templates are modular** - use what you need, add more later
5. **Automatic customization** makes setup quick (5-15 minutes)
6. **ROI starts immediately** even with partial deployment

For questions or issues, consult individual template documentation files or use Claude Code's built-in help.