

Module 2 — Framework Claude-Craft

Jour 1 — Architecture et configuration

The Bearded Bear

Février 2026



Table des matières

Module 2 : Le Framework Claude-Craft	2
Objectifs	2
1. Pourquoi Claude-Craft?	2
Le problème	2
La solution : Claude-Craft 7.13.0	2
Bénéfices	2
2. TCL : Tiered Context Loading	3
Le concept révolutionnaire	3
Les trois niveaux	3
Comparaison des versions	4
Comment ça fonctionne?	4
3. Architecture Claude-Craft 7.13.0	4
Structure TCL	4
Technologies supportées (10 stacks)	5
4. Installation via NPX	5
Méthode moderne (recommandée)	5
Installation directe	5
Options disponibles	6
5. Système de Skills	6
Format officiel Claude Code	6
Invocation des skills	7
Skills disponibles (communs)	7
Nouvelles propriétés Skills v7.13.0	7
6. Commands et Agents	8
Commands (Actions rapides)	8
Agents (Personas IA)	9
Différence Skills vs Commands	10
Plugin Manifest	11
7. Configuration du contexte	11
CLAUDE.md minimal (v7.13.0)	11
INDEX.md (Table des matières)	12
context.yaml (Triggers automatiques)	12
Améliorations settings.json (v7.13.0)	13
8. Workflow avec TCL	13
Exemple pratique	13
Exercice Pratique	14
Objectifs	14
Points Clés à Retenir	14

Module 2 : Le Framework Claude-Craft

Objectifs

À la fin de ce module, vous serez capable de : - Expliquer l'architecture TCL (Tiered Context Loading) et son économie de tokens - Installer Claude-Craft via npx sur un projet - Comprendre les différents composants (skills, commands, agents, references) - Configurer le contexte projet avec la structure 7.13.0 - Comprendre le framework BMAD v6 et les 33 agents disponibles

1. Pourquoi Claude-Craft ?

Le problème

Claude Code est puissant, mais : - Pas de guidelines par défaut - Pas de patterns imposés - Qualité variable selon les prompts - Consommation de tokens non optimisée - Pas de gestion de projet intégrée - Pas d'automatisation des sprints

La solution : Claude-Craft 7.13.0

Claude-Craft ajoute une **couche de best practices optimisée** avec gestion de projet et automatisation :

CLAUDE CODE 2.1.45 (Moteur IA + Extended Thinking + MCP)
CLAUDE-CRAFT 7.13.0 TCL + Skills + References + Agents BMAD v6 + Ralph + QA Recette
VOTRE PROJET (Code source + contexte)

Bénéfices

Aspect	Sans Claude-Craft	Avec Claude-Craft 7.13.0
Tokens	~70,000 auto-chargés	~3,500 (économie 95%)

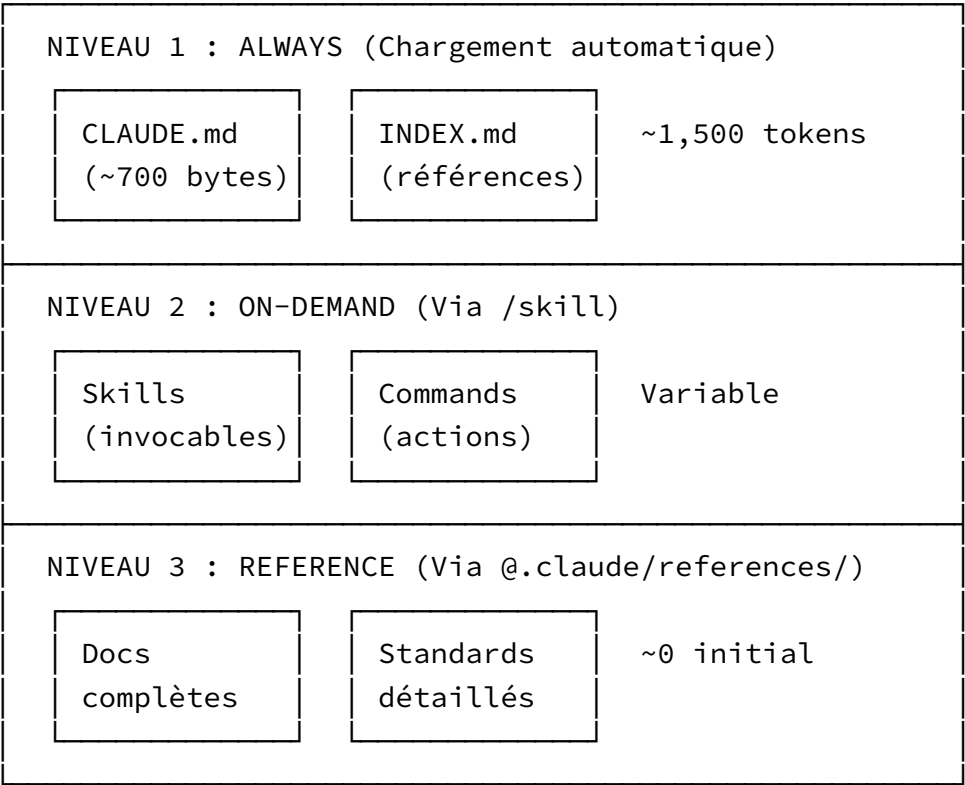
Aspect	Sans Claude-Craft	Avec Claude-Craft 7.13.0
Architecture	Variable	Clean Architecture imposée
Tests	Optionnels	TDD obligatoire
Sécurité	À la demande	OWASP intégré
Technologies	1 seule	10 stacks supportées
Gestion projet	Variable	BMAD v6 (5 quality gates)
Automatisation	Manuelle	Ralph (boucle IA continue)
Agents	0	33 agents (4 catégories)

2. TCL : Tiered Context Loading

Le concept révolutionnaire

Le **TCL (Tiered Context Loading)** est l'innovation majeure de Claude-Craft. Il charge le contexte en 3 niveaux pour économiser ~95% des tokens.

Les trois niveaux



Comparaison des versions

Version	Auto-chargé	À la demande	Économie
v3.x	~70,000 tokens	-	-
v7.x	~3,500 tokens	Reste	95%

Comment ça fonctionne ?

1. **Démarrage** : Seuls `CLAUDE.md` et `INDEX.md` sont chargés
2. **Contexte** : `context.yaml` définit les triggers automatiques
3. **À la demande** : Les skills sont chargés via `/skill`
4. **Références** : Documentation complète via `@.claude/references/`

3. Architecture Claude-Craft 7.13.0

Structure TCL

```
.claude/
├── CLAUDE.md           # Config minimale (~700 bytes) [ALWAYS]
├── INDEX.md           # Référence rapide, liens [ALWAYS]
├── context.yaml        # Triggers automatiques
├── references/         # Documentation complète [REFERENCE]
│   ├── base/          # Principes universels
│   │   ├── SOLID.md
│   │   ├── KISS-DRY-YAGNI.md
│   │   ├── testing.md
│   │   └── security.md
│   └── symfony/       # Spécifique technologie
│       ├── CLAUDE.md
│       ├── architecture.md
│       └── patterns.md
├── skills/            # Best practices [ON-DEMAND]
│   ├── testing.md
│   ├── security.md
│   └── git-workflow.md
├── agents/            # Agents IA spécialisés
│   ├── reviewer.md
│   └── tdd-coach.md
└── commands/          # Commandes slash
```

```
| | | common/
| | | symfony/
| | .bmad/           # BMAD v6 project management
| | .recette/        # QA Recette test plans & sessions
```

Technologies supportées (10 stacks)

Stack	Catégorie	Focus principal
symfony	Backend PHP	Clean Architecture + DDD
laravel	Backend PHP	Eloquent + API Resources
react	Frontend JS	Hooks + State Management
angular	Frontend TS	Standalone + Signals
vuejs	Frontend JS	Composition API
flutter	Mobile	BLoC/Riverpod
reactnative	Mobile	Native Modules
python	Backend	FastAPI/Django
php	Backend	Standards PSR
csharp	Backend	Clean Architecture + CQRS

4. Installation via NPX

Méthode moderne (recommandée)

```
# Installation interactive
npx @the-bearded-bear/claude-craft install

# L'assistant vous guide :
# 1. Sélection du répertoire cible
# 2. Choix de la technologie
# 3. Choix de la langue (fr, en, es, de, pt)
```

Installation directe

```
# Syntaxe complète
npx @the-bearded-bear/claude-craft install <chemin> --tech=<stack>
↳ --lang=<langue>
```

Exemples

```
npx @the-bearded-bear/claude-craft install ~/mon-projet --tech=symfony
  ↪ --lang=fr
npx @the-bearded-bear/claude-craft install . --tech=react --lang=en
npx @the-bearded-bear/claude-craft install ~/mobile-app --tech=flutter
  ↪ --lang=fr
```

Options disponibles

Voir l'aide

```
npx @the-bearded-bear/claude-craft --help
```

Technologies disponibles

```
--tech=symfony      # Symfony 8.0 / PHP 8.5
--tech=laravel      # Laravel 12 / PHP 8.5
--tech=react        # React 19 / TypeScript 5.7
--tech=angular      # Angular 19.x / TypeScript 5.7
--tech=vuejs        # Vue 3.5+ / TypeScript 5.7
--tech=flutter      # Flutter 3.38 / Dart 3.10
--tech=reactnative  # React Native 0.76+
--tech=python       # Python 3.13+ / FastAPI
--tech=php          # PHP 8.5 standards
--tech=csharp       # .NET 10 / C# 14
```

Langues

```
--lang=fr          # Français
--lang=en          # English
--lang=es          # Español
--lang=de          # Deutsch
--lang=pt          # Português
```

5. Système de Skills

Format officiel Claude Code

Les skills sont des fichiers markdown avec un frontmatter YAML définissant les métadonnées (name, description, model, tools, disallowedTools) :

name: testing

description: Testing - TDD/BDD principles. Use when writing tests.

model: claude-opus-4-6

tools:

- Read
- Grep
- Glob

disallowedTools:

- Write

Testing - TDD/BDD Principles

Quick Reference

- RED: Write failing test first
- GREEN: Implement minimum code
- REFACTOR: Clean up

See @.claude/references/base/testing.md for detailed documentation.

Invocation des skills

Dans Claude Code

```
/testing          # Charge le skill testing
/security         # Charge le skill security
/git-workflow     # Charge le skill git
```

Les skills sont aussi chargés automatiquement
selon le contexte (via context.yaml)

Skills disponibles (communs)

Skill	Trigger automatique
testing	Mention de “test”, “TDD”, “spec”
security	Mention de “security”, “auth”, “OWASP”
git-workflow	Mention de “commit”, “branch”, “PR”
documentation	Mention de “docs”, “README”, “API doc”
workflow-analysis	Début de feature, analyse

Nouvelles propriétés Skills v7.13.0

Le frontmatter YAML des skills a été enrichi avec de nouvelles propriétés :

Propriété	Description
allowed-tools	Restriction des outils disponibles pour ce skill
model	Modèle optimisé pour le skill (ex : haiku pour skills simples, économie de coûts)

Propriété	Description
triggers	Déclencheurs étendus : file patterns + keywords (pas seulement keywords)

```

---
name: testing
description: TDD/BDD principes
model: haiku          # Optimisation coûts pour skills simples
allowed-tools:
  - Read
  - Grep
  - Glob
triggers:
  keywords: ["test", "TDD", "spec"]
  file_patterns: ["*Test.php", "*.spec.ts", "*.test.js"]
---

```

6. Commands et Agents

Commands (Actions rapides)

```

# Commandes communes
/common:pre-commit-check          # Validation avant commit
/common:ralph-run                 # Boucle continue Claude
/common:setup-project-context     # Configurer le contexte
/common:add-technology            # Ajouter une stack

# Commandes Symfony
/symfony:check-compliance         # Audit complet
/symfony:check-architecture       # Validation archi
/symfony:check-code-quality       # PHPStan, CS-Fixer
/symfony:check-security           # OWASP Top 10
/symfony:check-testing            # Couverture tests

# Commandes C#/.NET
/csharp:generate-feature          # Générer feature CQRS
/csharp:check-compliance         # Audit complet
/csharp:check-architecture       # Validation Clean Archi

# Commandes BMAD v6
/workflow:init                   # Initialiser le workflow (auto-detect
  ↳ track)
/workflow:status                 # Statut du projet
/sprint:next-story               # Prochaine story prête

```

```
/sprint:transition           # Transition de statut
/gate:validate-prd          # Quality gate PRD (>=80%)
/gate:validate-story        # Validation DoD story
/project:run-sprint         # Exécuter un sprint complet

# Commandes Docker
/docker:compose-setup      # Générer docker-compose
/docker:architecture       # Designer l'architecture Docker
/docker:debug              # Diagnostiquer les problèmes
/docker:cicd-pipeline      # Générer pipeline CI/CD
/docker:optimize           # Optimiser le setup Docker

# Commandes QA Recette
/qa:recette                # Tests d'acceptation automatisés
/qa:status                 # Statut de la session recette
```

Agents (Personas IA)

Les agents sont des experts spécialisés avec un contexte et une personnalité. Claude-Craft 7.13.0 propose **33 agents** répartis en 4 catégories :

Agents communs (12)

Agent	Expertise
@api-designer	REST/GraphQL API design
@database-architect	Database optimization
@devops-engineer	CI/CD, Docker, deployment
@performance-auditor	Performance analysis
@refactoring-specialist	Safe code refactoring
@tdd-coach	Test-Driven Development
@uiux-orchestrator	UI/UX coordination
@ui-designer	Design systems, tokens
@ux-ergonome	User flows, cognitive ergonomics
@accessibility-expert	WCAG 2.2 AAA compliance
@research-assistant	Technical research
@ralph-conductor	Continuous loop orchestration

Reviewers technologie (10)

Agent	Technologie
@symfony-reviewer	Symfony/PHP
@flutter-reviewer	Flutter/Dart
@react-reviewer	React
@python-reviewer	Python
@angular-reviewer	Angular
@laravel-reviewer	Laravel
@vuejs-reviewer	Vue.js
@reactnative-reviewer	React Native
@csharp-reviewer	C#/.NET
@php-reviewer	PHP

Agents Docker (5)

Agent	Expertise
@docker-dockerfile	Dockerfile optimization
@docker-compose	Compose orchestration
@docker-debug	Container troubleshooting
@docker-cicd	CI/CD pipelines
@docker-architect	Docker architecture

Agents Projet (2)

Agent	Rôle
@product-owner	Product management (CSPO)
@tech-lead	Technical leadership

Note : Les rôles BMAD (bmad-master, pm, ba, architect, po, sm, dev, qa, qa-recette, ux) sont intégrés dans les commandes `/workflow:*` et `/sprint:*`, pas en agents autonomes.

Différence Skills vs Commands

Aspect	Skills	Commands
Invocation	/skill-name ou auto	/prefix:action
Contenu	Guidelines, principes	Actions structurées
Chargement	On-demand ou auto	À la demande
Output	Contextuel	Structuré, checklist

Plugin Manifest

Claude-Craft introduit un système de **Plugin Manifest** pour la distribution et la déclaration de capabilities :

```
.claude-plugin/  
├─ plugin.json          # Manifeste du plugin  
├─ commands/           # Commandes ajoutées  
├─ agents/             # Agents spécialisés  
├─ hooks/              # Scripts de hooks  
└─ skills/             # Skills du plugin
```

Structure plugin.json

```
{  
  "name": "mon-plugin",  
  "version": "1.0.0",  
  "description": "Plugin personnalisé",  
  "capabilities": {  
    "skills": ["commands/my-skill.md"],  
    "agents": ["agents/my-agent.md"],  
    "hooks": {  
      "PostToolUse": [{"matcher": "Write", "command": "hooks/post-write.sh"}]  
    },  
    "mcp": ["mcp-config.json"]  
  }  
}
```

Le manifeste déclare les capabilities du plugin (skills, agents, hooks, MCP) pour faciliter la distribution et l'installation via marketplace.

7. Configuration du contexte

CLAUDE.md minimal (v7.13.0)

Le fichier CLAUDE.md est maintenant minimaliste (~700 bytes) :

```
# MonProjet - Symfony 8.0

## Quick Reference
See '@.claude/INDEX.md' for patterns and commands.

## Technology
| Stack | Version |
|-----|-----|
| Symfony | 8.0 |
| PHP | 8.5 |
| Doctrine | 4.0 |

## Commands
- 'make dev' - Start development server
- 'make test' - Run test suite
- 'make lint' - Check code style
```

INDEX.md (Table des matières)

```
# Claude-Craft Quick Index

## Commands Available
- '/symfony:check-compliance' - Full audit
- '/symfony:check-architecture' - Architecture validation
- '/workflow:init' - Initialize BMAD framework
- '/workflow:status' - Project status

## References
- Architecture: '@.claude/references/symfony/architecture.md'
- Testing: '@.claude/references/base/testing.md'
- Security: '@.claude/references/base/security.md'

## Skills
- '/testing' - TDD/BDD principles
- '/security' - OWASP guidelines
```

context.yaml (Triggers automatiques)

```
# .claude/context.yaml
triggers:
  testing:
    keywords: ["test", "TDD", "spec", "PHPUnit"]
    auto_load: true
  security:
    keywords: ["security", "auth", "OWASP", "vulnerability"]
    auto_load: true
  architecture:
    keywords: ["architecture", "clean", "hexagonal", "DDD"]
```

```
auto_load: false # Chargement manuel uniquement
```

Améliorations settings.json (v7.13.0)

```
{
  "plansDirectory": ".claude/plans",
  "permissions": {
    "allow": [
      "Bash(npm *)", "Bash(pnpm *)", "Bash(yarn *)",
      "Bash/php *)", "Bash(flutter *)", "Bash(ng *)", "Bash(dotnet *)"
    ],
    "deny": [
      "Bash(chmod 777 *)",
      "Bash(*curl*|*sh*)",
      "Write(*credentials*)"
    ]
  }
}
```

- **plansDirectory** : répertoire dédié pour les plans générés par Claude
- **Permissions étendues** : support natif des gestionnaires de paquets multi-technologie
- **Deny rules** : blocage des opérations dangereuses (chmod 777, pipe curl vers sh, écriture credentials)

8. Workflow avec TCL

Exemple pratique

Vous: "Je veux créer une feature de gestion des commandes"

Claude: [Charge automatiquement via context.yaml]
→ Détecte "feature" → charge workflow-analysis

"Analysons d'abord les besoins..."

Vous: "Commençons par les tests"

Claude: [Charge automatiquement]
→ Détecte "tests" → charge testing skill

"Suivant les principes TDD, voici les tests..."

Vous: "@.claude/references/symfony/patterns.md"

Claude: [Charge la référence complète]
→ Accède aux patterns détaillés

"Selon le pattern Repository..."

Exercice Pratique

Voir `exercices/exercice-02-installation-claude-craft.md`

Objectifs

1. Installer Claude-Craft via npx
 2. Explorer la structure TCL
 3. Comprendre les 3 niveaux de chargement
 4. Tester les skills et commands
 5. Configurer le contexte projet
-

Points Clés à Retenir

1. **TCL** = Économie de 95% des tokens via chargement en 3 niveaux
 2. **CLAUDE.md** = Configuration minimale (~700 bytes)
 3. **INDEX.md** = Table des matières et liens rapides
 4. **Skills** = Chargés on-demand ou via triggers (format YAML frontmatter)
 5. **References** = Documentation complète via `@.claude/references/`
 6. **NPX** = Installation moderne en une commande
 7. **10 stacks** = Support multi-technologie
 8. **BMAD v6** = Gestion de projet intégrée avec 5 quality gates
 9. **33 agents** = Experts spécialisés en 4 catégories
 10. **160 commandes** = Couverture complète du workflow de développement (20 namespaces)
-

Durée estimée : 1h30 **Prochain module :** Workflow de Développement