

Module 7 — Agents Spécialisés

Jour 2 — Orchestration multi-agents

The Bearded Bear

Février 2026



Table des matières

| | |
|---|----------|
| Module 7 : Agents Specialises, BMAD et Docker (Claude Code 2.1.45 + Claude-Craft 7.13.0) | 3 |
| Objectifs | 3 |
| 1. Skills System (Claude Code 2.1.45) | 3 |
| Qu'est-ce qu'un Skill? | 3 |
| Invocation des Skills | 3 |
| Chargement automatique (context.yaml) | 3 |
| Skills disponibles (Claude-Craft 7.13.0) | 4 |
| 2. Panorama des 29 Agents (4 Catégories) | 4 |
| Common Agents (12) | 4 |
| Technology Reviewers (10) | 5 |
| Project Agents (2) | 6 |
| Docker Agents (5) | 6 |
| 3. BMAD Workflow in Action | 7 |
| Le Workflow BMAD v6 | 7 |
| Etape 1 : Initialisation du projet | 7 |
| Etape 2 : Generer le PRD | 7 |
| Etape 3 : Validation Quality Gate | 8 |
| Etape 4 : Concevoir l'architecture | 8 |
| Etape 5 : Sprint et implementation TDD | 8 |
| Etape 6 : Validation et tests d'acceptance | 9 |
| Utiliser les Project Agents | 9 |
| Enchaîner les commandes BMAD | 9 |
| 4. Docker Agents | 9 |
| Pourquoi des agents Docker dedies? | 9 |
| @docker-architect : Conception d'architecture | 10 |
| @docker-compose : Orchestration | 10 |
| @docker-debug : Diagnostic | 11 |
| @docker-dockerfile : Optimisation | 11 |
| @docker-cicd : Pipelines | 12 |
| 5. Agent YAML Frontmatter | 12 |
| Structure d'un agent | 12 |
| Proprietes du frontmatter | 12 |
| Exemple complet : Agent @tdd-coach | 13 |
| Exemple complet : Agent @docker-debug | 13 |
| Exemple complet : Agent @qa | 14 |
| Personnalisation : Creer son propre agent | 14 |
| 6. Skills vs Agents : Quand utiliser quoi? | 15 |
| Comparaison | 15 |
| Guide de selection | 15 |
| 7. Utilisation Pratique des Skills | 16 |
| Utiliser /security pour audit | 16 |

| | |
|---|----|
| Utiliser /git-workflow pour commits | 16 |
| Utiliser /testing pour TDD | 17 |
| 8. Combinaison Skills + Agents | 17 |
| Workflow multi-etapes | 17 |
| Workflow Docker complet | 18 |
| 9. Integration avec les Hooks | 18 |
| Charger un skill automatiquement via hook | 18 |
| Notification apres review | 19 |
| 10. Exercice Pratique | 19 |
| Objectif | 19 |
| Scenario | 19 |
| Etapas | 20 |
| Criteres de reussite | 20 |
| Points Cles a Retenir | 21 |

Module 7 : Agents Specialises, BMAD et Docker (Claude Code 2.1.45 + Claude-Craft 7.13.0)

Objectifs

A la fin de ce module, vous serez capable de : - Connaitre les 33 agents disponibles dans Claude-Craft 7.13.0, organisés en 4 catégories - Comprendre le système de Skills de Claude Code 2.1.45 - Maîtriser le workflow BMAD v6 avec ses commandes dédiées - Utiliser les 5 agents Docker pour l'infrastructure - Comprendre le format YAML frontmatter des agents - Savoir invoquer le bon skill/agent selon le contexte - Combiner skills et agents efficacement

1. Skills System (Claude Code 2.1.45)

Qu'est-ce qu'un Skill ?

Un **Skill** est un ensemble de guidelines et bonnes pratiques que Claude peut charger à la demande. C'est le système officiel de Claude Code pour étendre ses capacités.

Invocation des Skills

```
# Charger un skill manuellement
/testing                # TDD/BDD principes
/security               # OWASP guidelines
/git-workflow           # Git best practices
/documentation          # Documentation standards
/solid-principles       # SOLID patterns
/kiss-dry-yagni         # Code simplicity

# Lister les skills disponibles
/skills
```

Chargement automatique (context.yaml)

Les skills peuvent être chargés automatiquement selon le contexte :

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

Skills disponibles (Claude-Craft 7.13.0)

| Skill | Trigger | Contenu |
|-------------------|-----------------------|--------------------------|
| testing | test, TDD, spec | TDD/BDD, couverture 80%+ |
| security | security, auth, OWASP | OWASP Top 10, audit |
| git-workflow | commit, branch, PR | Conventional Commits |
| documentation | docs, README, ADR | Standards documentation |
| workflow-analysis | feature, analyse | Workflow obligatoire |
| solid-principles | SOLID, refactor | SRP, OCP, LSP, ISP, DIP |
| kiss-dry-yagni | simple, duplicate | Principes de simplicité |

2. Panorama des 29 Agents (4 Catégories)

Claude-Craft 7.13.0 propose **33 agents** repartis en **4 catégories** distinctes. Chaque agent est un expert dans son domaine, invocable via la syntaxe `@agent-name`.

Optimisation des modèles : Les agents utilisent un modèle adapté à leur rôle. Les reviewers et auditors utilisent **haiku** (économique pour les tâches de vérification), tandis que les engineers et architects utilisent **sonnet** (puissant pour les tâches de conception et implémentation).

Common Agents (12)

Les agents transversaux utilisables quel que soit le stack technologique.

| 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 |

| Agent | Expertise |
|-----------------------|-------------------------------|
| @accessibility-expert | WCAG 2.2 AAA compliance |
| @research-assistant | Technical research |
| @ralph-conductor | Continuous loop orchestration |

Exemples d'utilisation :

Design d'API REST

@api-designer "Conçois l'API REST pour la gestion d'un panier e-commerce"

Audit de performance

@performance-auditor "Analyse les goulots d'etirement de ce service"

Refactoring guide

@refactoring-specialist "Refactore cette classe en appliquant le pattern
↪ Strategy"

Accessibilité

@accessibility-expert "Vérifie la conformité WCAG 2.2 AAA de ce formulaire"

Technology Reviewers (10)

Un reviewer dédié pour chaque stack technologique supportée.

| Agent | Technology |
|-----------------------|--------------|
| @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 |

Exemples d'utilisation :

Review Symfony

```
@symfony-reviewer "Revois ce controller et verifie le respect de Clean
↳ Architecture"

# Review React
@react-reviewer "Verifie ce composant pour les bonnes pratiques React 19"

# Review Python
@python-reviewer "Analyse ce service FastAPI pour les conventions Python
↳ 3.13"

# Review Flutter
@flutter-reviewer "Valide ce widget BLoC pour les patterns Flutter 3.38"
```

Project Agents (2)

Les agents projet pour la gestion technique et produit.

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

Note BMAD v6 : Les roles BMAD (bmad-master, pm, ba, architect, po, sm, dev, qa, qa-recette, ux) sont integres dans les commandes /workflow:* et /sprint:*, pas en agents autonomes. Cette integration permet un workflow plus fluide ou chaque commande active le role BMAD adapte.

Docker Agents (5)

Les agents specialises dans l'écosysteme Docker et l'infrastructure.

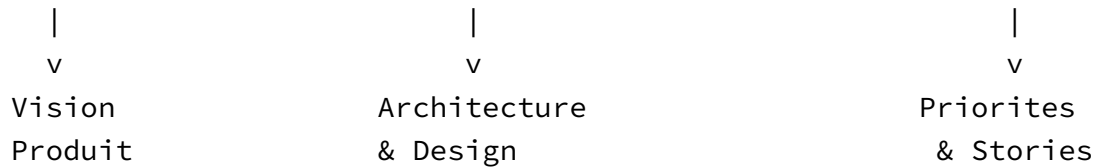
| Agent | Expertise |
|--------------------|---------------------------|
| @docker-dockerfile | Dockerfile optimization |
| @docker-compose | Compose orchestration |
| @docker-debug | Container troubleshooting |
| @docker-cicd | CI/CD pipelines |
| @docker-architect | Docker architecture |

3. BMAD Workflow in Action

Le Workflow BMAD v6

BMAD v6 utilise des **commandes dédiées** qui activent le rôle BMAD adapté à chaque phase du cycle de développement :

```
/workflow:plan (PRD) --> /workflow:design (Tech Spec) --> /sprint:next-story (Backlog)
```



```
--> /workflow:implement (Code) --> /qa:recette (Tests)
```



Etape 1 : Initialisation du projet

```
# Initialiser le workflow (auto-detect track)
/workflow:init "Implement user authentication with OAuth2, magic links, and
  ↪ 2FA"
```

```
# Claude configure la structure BMAD :
# - Cree le dossier .bmad/
# - Initialise les templates
# - Configure les quality gates
# - Recommande le track adapte (Quick Flow, Standard ou Enterprise)
```

Etape 2 : Generer le PRD

```
/workflow:plan
```

```
# Output :
# --- PRD : User Authentication System ---
#
# 1. Vision
#   Systeme d'authentification moderne, securise et frictionless
#
# 2. Objectifs
#   - Support OAuth2 (Google, GitHub, Microsoft)
#   - Magic links par email
#   - 2FA via TOTP (Google Authenticator)
```



```
#  
# 3. User Stories  
#   US-001: En tant qu'utilisateur, je veux me connecter via Google  
#   US-002: En tant qu'utilisateur, je veux recevoir un magic link  
#   US-003: En tant qu'utilisateur, je veux activer la 2FA  
#   ...
```

Etape 3 : Validation Quality Gate

```
# Valider le PRD avant de passer a l'architecture  
/gate:validate-prd
```

```
# Output :  
# --- PRD Quality Gate ---  
# Score : 87% (seuil : 80%)  
#  
# [PASS] Vision claire et mesurable  
# [PASS] User stories au format standard  
# [PASS] Criteres d'acceptation definis  
# [WARN] Metriques de succes a preciser  
# [PASS] Contraintes techniques identifiees  
#  
# Resultat : PASSE - Pret pour la phase Design
```

Etape 4 : Concevoir l'architecture

```
/workflow:design
```

```
# Output :  
# --- Tech Spec : Authentication Architecture ---  
#  
# 1. Architecture  
#   Clean Architecture avec 4 couches  
#  
# 2. Composants  
#   - AuthenticationService (Domain)  
#   - OAuth2Provider (Infrastructure)  
#   - MagicLinkGenerator (Application)  
#   - TwoFactorValidator (Domain)  
#  
# 3. Diagramme de sequence  
#   ...
```

Etape 5 : Sprint et implementation TDD

```
# Recuperer la prochaine story prete  
/sprint:next-story
```

```
# Implementer en TDD
/workflow:implement

# Le workflow suit le cycle TDD :
# Phase RED : Ecrit les tests qui echouent
# Phase GREEN : Implemente le minimum pour passer les tests
# Phase REFACTOR : Ameliore le code
```

Etape 6 : Validation et tests d'acceptance

```
# Valider la Definition of Done
/gate:validate-story US-001

# Tests d'acceptation automatisés via Chrome
/qa:recette --scope=story --id=US-001
```

Utiliser les Project Agents

```
# Product Owner pour la gestion produit
@product-owner "Prioritize the authentication backlog for Sprint 1"

# Tech Lead pour les decisions techniques
@tech-lead "Review the authentication architecture decisions"
```

Enchaîner les commandes BMAD

```
# Workflow complet en une session
/workflow:init "Notification system"
/workflow:plan
/gate:validate-prd
/workflow:design
/sprint:next-story
/workflow:implement
/gate:validate-story
```

4. Docker Agents

Pourquoi des agents Docker dédiés ?

Claude-Craft 7.13.0 inclut 5 agents spécialisés Docker car l'infrastructure as code est un pilier du développement moderne. Chaque agent couvre un aspect spécifique de l'écosystème Docker.

@docker-architect : Conception d'architecture

```
@docker-architect "Design Docker architecture for a microservices e-commerce  
  ↳ platform"
```

```
# Output :  
# --- Docker Architecture : E-Commerce Microservices ---  
#  
# Services :  
# - api-gateway (Nginx/Traefik)  
# - auth-service (Node.js)  
# - catalog-service (Python/FastAPI)  
# - order-service (Symfony)  
# - payment-service (.NET)  
# - notification-service (Go)  
#  
# Infrastructure :  
# - PostgreSQL (primary DB)  
# - Redis (cache + sessions)  
# - RabbitMQ (message broker)  
# - MinIO (object storage)  
#  
# Reseau :  
# - frontend-network (gateway + services publics)  
# - backend-network (services internes + BDD)  
# - monitoring-network (Prometheus + Grafana)
```

@docker-compose : Orchestration

```
# Generer un docker-compose complet  
/docker:compose-setup
```

```
# Ou via l'agent directement
```

```
@docker-compose "Create docker-compose.yml for Symfony + PostgreSQL + Redis  
  ↳ + Mailpit"
```

```
# Output :  
# Genere docker-compose.yml avec :  
# - Service PHP-FPM 8.5 (Symfony)  
# - Nginx reverse proxy  
# - PostgreSQL 17  
# - Redis 7.4  
# - Mailpit pour les emails  
# - Volumes pour la persistance  
# - Healthchecks pour chaque service
```

@docker-debug : Diagnostic

```
# Diagnostic automatisé
/docker:debug
```

```
# Ou via l'agent pour un problème précis
```

```
@docker-debug "Le container PHP retourne une erreur 502 Bad Gateway"
```

```
# Output :
# --- DIAGNOSTIC DOCKER ---
#
# Analyse en cours...
#
# Problème identifié :
# Le container php-fpm n'est pas accessible depuis nginx
#
# Causes possibles :
# 1. php-fpm n'écoute pas sur le bon socket/port
# 2. Les containers ne sont pas sur le même réseau
# 3. Healthcheck php-fpm en échec
#
# Solution recommandée :
# 1. Vérifier fastcgi_pass dans nginx.conf
# 2. S'assurer que les deux services partagent le même réseau
# 3. Vérifier les logs : docker compose logs php
```

@docker-dockerfile : Optimisation

```
@docker-dockerfile "Optimize this Dockerfile for production PHP 8.5"
```

```
# Output :
# --- DOCKERFILE OPTIMIZATION ---
#
# Problèmes détectés :
# 1. Image de base trop volumineuse (1.2 GB)
# 2. Pas de multi-stage build
# 3. Cache layers non optimisé
# 4. Secrets en dur dans le build
#
# Dockerfile optimisé :
# - Multi-stage : builder + runtime
# - Image alpine (120 MB)
# - OPcache configuré
# - Non-root user
# - Healthcheck intégré
```

@docker-cicd : Pipelines

```
# Generer un pipeline CI/CD
/docker:cicd-pipeline
```

```
@docker-cicd "Create GitHub Actions pipeline with Docker build, test, and
  ↳ deploy"
```

```
# Output :
# Genere .github/workflows/ci.yml avec :
# - Build multi-arch (amd64/arm64)
# - Tests dans containers isolés
# - Scan de securite (Trivy)
# - Push vers registry
# - Deploy staging/production
```

5. Agent YAML Frontmatter

Structure d'un agent

Chaque agent dans Claude-Craft 7.13.0 est defini par un fichier Markdown avec un **frontmatter YAML** qui configure son comportement :

```
---
name: my-agent
description: Expert in X
model: sonnet
tools: [Read, Write, Bash, Glob, Grep]
disallowedTools: [WebSearch]
skills: [testing, security]
---
```

Proprietes du frontmatter

| Propriete | Type | Description |
|-----------------|--------|--|
| name | string | Identifiant unique de l'agent |
| description | string | Description de l'expertise |
| model | string | Modele Claude a utiliser (sonnet, opus, haiku) |
| tools | list | Outils autorises pour l'agent |
| disallowedTools | list | Outils explicitement interdits |
| skills | list | Skills charges automatiquement |

| Propriete | Type | Description |
|----------------|--------|--|
| permissionMode | string | Mode de permissions specifique a l'agent |

Exemple complet : Agent @tdd-coach

```
---
name: tdd-coach
description: Expert TDD/BDD. Guides developers through Red-Green-Refactor
  ↪ cycle. Use when writing tests or implementing test-first.
model: sonnet
tools: [Read, Write, Bash, Glob, Grep]
disallowedTools: [WebSearch]
skills: [testing, solid-principles]
---
```

TDD Coach

Methodology

- Always start with a failing test (RED)
- Write the minimum code to pass (GREEN)
- Refactor while keeping tests green (REFACTOR)

Test Structure

- Arrange / Act / Assert pattern
- One assertion per test
- Descriptive test names

Coverage Targets

- Unit tests: 80%+
- Integration tests: key paths
- E2E tests: critical workflows

Exemple complet : Agent @docker-debug

```
---
name: docker-debug
description: Container troubleshooting specialist. Diagnoses Docker issues
  ↪ including networking, volumes, permissions, and performance.
model: sonnet
tools: [Read, Bash, Glob, Grep]
disallowedTools: [Write, WebSearch]
skills: []
---
```

Docker Debug Agent

Diagnostic Process

1. Check container status and health
2. Inspect logs for errors
3. Verify network connectivity
4. Check volume mounts and permissions
5. Analyze resource usage

Common Issues

- Port conflicts
- Volume permission denied
- Network isolation problems
- OOM kills
- Build cache corruption

Exemple complet : Agent @qa

```
---
name: qa
description: QA Engineer for BMAD v6. Validates stories against acceptance
  ↳ criteria, runs test suites, and ensures Definition of Done compliance.
model: sonnet
tools: [Read, Bash, Glob, Grep]
disallowedTools: [Write, WebSearch]
skills: [testing]
---
```

QA Engineer

Validation Process

1. Read acceptance criteria from story
2. Verify each criterion independently
3. Run automated test suite
4. Check edge cases and error handling
5. Validate DoD compliance

Quality Gates

- All acceptance criteria met
- Test coverage >= 80%
- No critical/high security issues
- Performance within SLA

Personnalisation : Creer son propre agent

```
# Creer un agent metier personnalise
# .claude/agents/order-specialist.md
```

```
---
name: order-specialist
```

description: Expert in order lifecycle management. Handles order state
 ↳ machines, payment flows, and fulfillment rules.

model: sonnet

tools: [Read, Write, Bash, Glob, Grep]

disallowedTools: [WebSearch]

skills: [testing, security]

Order Specialist

Domain Rules

- Order statuses: draft -> pending -> confirmed -> shipped -> delivered
- Cancellation only allowed before shipping
- Refund rules depend on delivery status

Validation

- Total must match sum of line items
- Currency consistency check
- Stock availability verification

6. Skills vs Agents : Quand utiliser quoi ?

Comparaison

| Aspect | Skills | Agents |
|---------------|----------------------------|------------------------------|
| Invocation | /skill-name | @agent-name ou prompt |
| Nature | Guidelines, principes | Expertise, dialogue, actions |
| Chargement | On-demand ou automatique | A la demande |
| Persistence | Durant la session | Conversation |
| Configuration | Triggers dans context.yaml | YAML frontmatter |
| Usage typique | Appliquer des regles | Resoudre un probleme |

Guide de selection

| | | |
|---|--------------|-------------|
| +-----+ SELECTION SKILL / AGENT +-----+ | | |
| | | |
| | | |
| | Besoin de... | -> Utiliser |
| | ----- | ----- |
| | | |

| | | |
|--------------------------|---------------------------------|--|
| Guidelines TDD | -> /testing | |
| Ecriture de tests guidée | -> @tdd-coach | |
| Guidelines securite | -> /security | |
| Audit securite complet | -> /symfony:check-security | |
| Design API | -> @api-designer | |
| Review de code | -> @symfony-reviewer (ou autre) | |
| Architecture systeme | -> @tech-lead | |
| Optimisation BDD | -> @database-architect | |
| Conventions git | -> /git-workflow | |
| Design UI/UX | -> @uiux-orchestrator | |
| Accessibilite | -> @accessibility-expert | |
| Infra Docker | -> @docker-architect | |
| Debug container | -> @docker-debug | |
| Pipeline CI/CD | -> @docker-cicd | |
| Gestion de projet | -> @product-owner | |
| Tests navigateur | -> /qa:recette | |
| Sprint autonome | -> @ralph-conductor | |
| | | |
| +-----+-----+-----+ | | |

7. Utilisation Pratique des Skills

Utiliser /security pour audit

```
# Charger le skill security
/security

# Puis demander l'audit
"Analyse ce code de paiement"

# Claude repond avec le contexte OWASP :
# AUDIT SECURITE
#
# Vulnerabilites detectees :
# 1. [A03:2021] Injection - Donnees non validees
# 2. [A07:2021] XSS - Pas d'echappement
# ...
```

Utiliser /git-workflow pour commits

```
# Charger le skill
/git-workflow
```

```
# Demander aide pour commit
"J'ai ajoute une feature de discount"

# Claude repond avec Conventional Commits :
# Message suggere :
# feat(order): add discount calculation
#
# Details :
# - Support des codes promo
# - Calcul progressif selon montant
```

Utiliser /testing pour TDD

```
# Charger le skill testing
/testing

# Claude repond maintenant avec le contexte TDD :
"Je veux creer un service de notification"

# Claude:
# "D'apres les principes TDD, commencons par ecrire les tests :
#
# ```php
# class NotificationServiceTest extends TestCase
# {
#     public function testSendsEmailOnOrderCreated(): void
#     {
#         // Arrange
#         ...
#     }
# }
# ```
```

8. Combinaison Skills + Agents

Workflow multi-etapes

```
# 1. Charger le skill d'architecture
/solid-principles

# 2. Design avec contexte SOLID
@api-designer "Concois l'API pour le module Paiement"

# 3. Charger le skill testing
/testing
```

```
# 4. Tests avec TDD
@bdd-coach "Ecris les tests pour PaymentService"

# 5. Implementation
"Implements PaymentService selon les tests"

# 6. Review avec le skill security charge
/security
@symfony-reviewer "Revois PaymentService"

# 7. Validation BMAD
/gate:validate-story
```

Workflow Docker complet

```
# 1. Architecture
@docker-architect "Design l'infrastructure pour un projet Symfony"

# 2. Generation du compose
/docker:compose-setup

# 3. Debug si nécessaire
@docker-debug "Le container PHP ne démarre pas"

# 4. Pipeline CI/CD
@docker-cicd "Configure le pipeline GitHub Actions"
```

9. Integration avec les Hooks

Charger un skill automatiquement via hook

```
// .claude/settings.json
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Write",
        "command": "echo 'Loading testing skill...'",
        "onlyIf": "*Test.php"
      }
    ]
  }
}
```

Notification apres review

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "if [[ $FILE_PATH == *.php ]]; then echo 'Consider
        ↪ running @symfony-reviewer'; fi"
      }
    ]
  }
}
```

10. Exercice Pratique**Objectif**

Realiser une code review complete avec skills, agents, BMAD et Docker.

Scenario

Vous avez un PaymentController a reviewer et a deployer :

```
class PaymentController extends AbstractController
{
    private $em;
    private $stripe;

    public function __construct($em, $stripe) {
        $this->em = $em;
        $this->stripe = $stripe;
    }

    #[Route('/pay', methods: ['POST'])]
    public function pay(Request $request)
    {
        $data = json_decode($request->getContent());
        $order = $this->em->find(Order::class, $data->order_id);

        $charge = $this->stripe->charges->create([
            'amount' => $order->getTotal() * 100,
            'currency' => 'eur',
            'source' => $data->token
        ]);
    }
}
```

```
$order->setStatus('paid');  
$order->setPaymentId($charge->id);  
$this->em->flush();  
  
return new JsonResponse(['status' => 'ok']);  
}  
}
```

Etapes

1. Charger les skills pertinents

```
/security  
/solid-principles
```

2. Code review avec Technology Reviewer

```
@symfony-reviewer "Revois ce PaymentController"
```

3. Audit securite

```
/symfony:check-security
```

4. Tests manquants avec TDD Coach

```
/testing  
@tdd-coach "Propose les tests necessaires pour PaymentController"
```

5. Proposition de refactoring

```
@refactoring-specialist "Refactore ce controller en appliquant Clean  
↪ Architecture"
```

6. Diagnostic Docker

```
@docker-debug "Le container Stripe webhook ne recoit pas les events en  
↪ local"
```

7. Validation finale BMAD

```
/gate:validate-story US-042
```

8. Tests d'acceptation via navigateur

```
/qa:recette --scope=story --id=US-042
```

Criteres de reussite

- ☐ Skills charges : /security, /solid-principles, /testing

- ☐ Issues identifiées par @symfony-reviewer
 - ☐ Vulnérabilités listées par /symfony :check-security
 - ☐ Tests proposés par @tdd-coach
 - ☐ Code refactoré par @refactoring-specialist
 - ☐ Problème Docker diagnostiqué par @docker-debug
 - ☐ Story validée par /gate :validate-story
 - ☐ Tests navigateur exécutés par /qa :recette
-

Points Clés à Retenir

1. **33 agents** organisés en **4 catégories** : Common (12), Technology Reviewers (10), Docker (5), Project (2)
 2. **Skills** = Guidelines chargées on-demand (/skill-name)
 3. **Agents** = Experts spécialisés pour dialogue et actions (@agent-name)
 4. **BMAD workflow** = /workflow :plan -> /workflow :design -> /sprint :next-story -> /workflow :implément avec quality gates
 5. **BMAD roles** = Intégrés dans les commandes /workflow :* et /sprint :*, pas en agents autonomes
 6. **Docker agents** = Architecture, Compose, Debug, CI/CD, Dockerfile
 7. **YAML frontmatter** = Configuration déclarative des agents (name, model, tools, skills)
 8. **Combinaison** = Skills + Agents + BMAD + Docker pour workflow complet
-

Durée estimée : 1h30 **Prochain module** : Outils Avancés et Productivité