Playbook 1

Orquestração Cognitiva

Guia Oficial de Implementação de Agentes Cognitivos
Integrações, Automação e IA Aplicada

CENC.Ai — Plataforma Cognitiva

Sumário

1. Introdução à Orquestração Cognitiva

1.1. O ponto de virada

Durante a primeira década da Inteligência Artificial corporativa, acreditava-se que o caminho era colocar IA dentro do sistema — um chatbot no ERP, um modelo de previsão no CRM, uma automação no portal. Essa abordagem trouxe ganhos rápidos, porém limitados:

- Cada módulo passou a usar uma IA diferente e isolada.
- A manutenção tornou-se cara e complexa.
- Faltava governança central e aprendizado compartilhado.
- O valor da IA ficou restrito ao seu contexto local.

A nova geração de IA corporativa inverte essa lógica: agora os sistemas passam a existir **dentro da inteligência**, e não o contrário.

1.2. O que é Orquestração Cognitiva

A orquestração cognitiva é a coordenação inteligente de múltiplos componentes de IA e automação dentro de um mesmo fluxo governado. Ela permite que modelos, APIs, bancos de dados e humanos cooperem de forma estruturada, auditável e expansível.

Dica

Em vez de "IA dentro do ERP", temos o ERP como uma Tool dentro de uma Plataforma Cognitiva capaz de decidir, aprender e agir.

1.3. Exemplo intuitivo

Uma empresa precisa responder diariamente a pedidos de orçamento por e-mail.

- **Modelo tradicional:** script fixo que gera PDF e envia e-mail; cada automação é reescrita.
- **Orquestração cognitiva:** um Agent interpreta o pedido, uma Task planeja as etapas e Tools executam (ERP, PDF, e-mail).

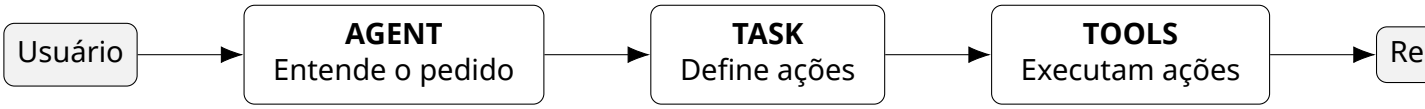

Figura 1: Fluxo conceitual da orquestração cognitiva.

Essência: o **Agent** pensa, a **Task** planeja e a **Tool** executa.

1.4. O novo papel da IA corporativa

A IA deixa de ser assistente e passa a ser o sistema de decisão central, capaz de:

- Entender intenções (NLP).
- Planejar etapas (planejamento cognitivo).
- Executar ou delegar (orquestração de Tools).
- Aprender com os resultados (feedback loop).

Essa camada de inteligência fica acima da infraestrutura existente, integrando sistemas legados e ampliando suas capacidades.

1.5. O que a Orquestração substitui

Tabela 1: Comparativo entre modelo antigo e Orquestração Cognitiva

Modelo antigo	Limitação	Solução via Orquestração
Chatbots isolados	Respostas limitadas	Agents com intenções múltiplas e histórico global.
Automações hardcoded	Manutenção difícil	Tasks modulares com versionamento.
APIs fixas	Falta de adaptação	Tools dinâmicas e plugáveis.
Falta de contexto	Respostas incoerentes	Memória e raciocínio interconectado.
Baixa escalabilidade	Reescrita por cliente	Fluxos reusáveis e licenciáveis.

1.6. Benefícios estratégicos

- Governança unificada: inteligência e automação sob uma mesma arquitetura.
- Reutilização de agentes e fluxos cognitivos.
- Expansão modular com novas tools plugáveis.
- Auditoria e rastreabilidade centralizadas.
- Economia cognitiva: menos tempo, mais valor.

1.7. Estrutura da Plataforma Cognitiva

Tabela 2: Camadas da Plataforma Cognitiva

Camada	Função	Exemplo
Agents	Entendem intenções e roteiam fluxos	Agente Financeiro, Agente de RH
Tasks	Planejam ações e verificam parâmetros	Gerar relatório, Atualizar registro
Tools	Executam operações concretas	APIs, SQL, LLMs, scripts

Nota

A arquitetura segue o princípio: “Pensar $\rightarrow$ Planejar $\rightarrow$ Executar $\rightarrow$ Aprender”.

1.8. Do ponto de vista técnico

A orquestração cognitiva é um **sistema arquitetural**, não apenas um framework. Divide-se em três camadas:

- **Orquestrador (Core Cognitivo):** processa intenções e contexto.
- **Camada de Execução (Tools):** conecta o core à infraestrutura.
- **Camada de Governança:** garante rastreabilidade, reputação e versionamento.

1.9. Analogias práticas

Tabela 3: Analogias entre o mundo real e o mundo cognitivo

Mundo real	Mundo cognitivo
Maestro	Agent
Partitura	Task
Instrumentos	Tools
Música	Execução orquestrada

Dica

A IA é o maestro da automação, não mais um músico isolado.

1.10. Por que isso muda tudo

Ao padronizar a estrutura, qualquer empresa pode:

- Transformar sistemas legados em componentes cognitivos reutilizáveis.
- Criar produtos licenciáveis baseados em fluxos inteligentes.
- Gerar receita recorrente sem reescrever automações.

1.11. O que vem a seguir

No próximo capítulo, exploraremos o modelo Agents $\rightarrow$ Tasks $\rightarrow$ Tools em profundidade: responsabilidades, fluxo de dados e versionamento.

Elemento visual sugerido

Nota

Resumo do Capítulo 1: a orquestração cognitiva é o novo paradigma da IA corporativa, permitindo que empresas coloquem seus sistemas dentro da inteligência — com governança, versionamento e potencial de monetização.

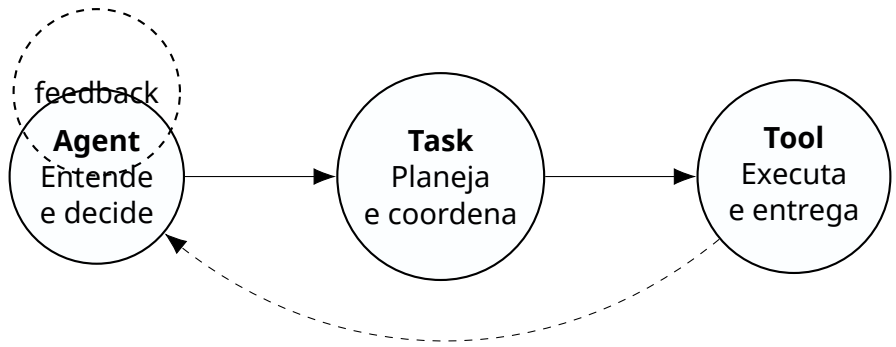

Figura 2: Fluxo cognitivo: Agent → Task → Tool com ciclo de aprendizado.

2. O Modelo Agents - Tasks - Tools

2.1. A espinha dorsal da Plataforma Cognitiva

O modelo Agents → Tasks → Tools funciona como o esqueleto lógico da orquestração cognitiva. Cada camada possui responsabilidade distinta — quando combinadas, permitem raciocinar, planejar e executar de forma hierárquica e auditável.

Dica

Resumo rápido: **Agents** interpretam intenções, **Tasks** planejam ações e **Tools** executam operações concretas.

2.2. Visão geral do fluxo

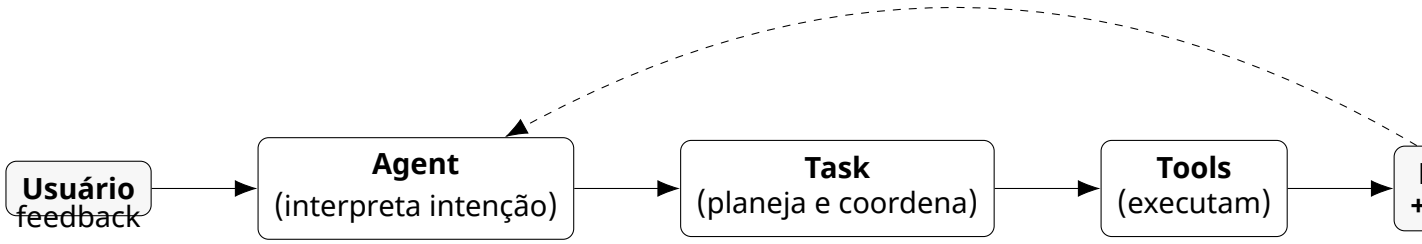

Figura 3: Fluxo lógico Agent → Task → Tool com retorno de feedback.

O fluxo é fundamentalmente bidirecional: a decisão flui de cima para baixo e o feedback sobe — alimentando histórico, aprendizado e possíveis replanejamentos.

2.3. Camada 1 — AGENTS (Interpretação e Intenção)

Os Agents são os pontos de entrada cognitivos da plataforma. Representam entidades que compreendem o pedido do usuário e decidem qual caminho a cadeia de orquestração deve seguir.

2.3.1. Funções principais

- Interpretar intenção (NLP / NLU).
- Gerar contexto cognitivo (quem pediu, o quê, por quê).
- Selecionar Tasks apropriadas.
- Armazenar histórico de interações.

2.3.2. Estrutura conceitual de um Agent

Listing 1: Exemplo JSON simplificado de um Agent

```
{
  "id": "ag_financeiro",
  "nome": "Agente Financeiro",
  "intencoes": ["gerar_relatorio_vendas","consultar_faturamento","prever_fluxo_caixa"],
  "policy": { "fallback": "human_in_the_loop", "retry_limit": 2 },
  "metadata": { "owner": "FinanceDept", "version": "1.0.3" }
}
```

2.3.3. Tipos de Agents

Tabela 4: Tipos comuns de Agents e suas responsabilidades

Tipo	Descrição / Exemplo
Agente de Domínio	Representa área de negócio (ex.: Agente Financeiro, Agente de RH).
Agente de Suporte	Executa tarefas transversais (ex.: Agente de Consulta de API).
Agente Orquestrador	Centraliza decisões e roteia entre agentes (ex.: Agente Raiz).

2.3.4. Boas práticas

- Nomear agents com propósito claro.
- Limitar intenções a 5–10 por agente (clareza > volume).
- Registrar contexto e histórico (memória curta e longa).
- Usar fallback humano quando confiança for menor que o limiar.

2.4. Camada 2 — TASKS (Planejamento e Coordenação)

As Tasks representam o planejamento lógico de uma ação. Criadas pelos Agents, são compostas por uma ou mais sub-tarefas executáveis e definem a ordem de invocação das Tools.

2.4.1. Funções principais

- Definir etapas de execução (steps).
- Validar parâmetros obrigatórios.

- Resolver dependências entre tools.
- Gerar logs e controlar erros.

2.4.2. Estrutura conceitual de uma Task

Listing 2: Exemplo JSON simplificado de uma Task

```
{
  "id": "task_gerar_relatorio",
  "descricao": "Gerar relatório de vendas mensal",
  "agent_ref": "ag_financeiro",
  "steps": [
    {"ordem":1, "tool":"tool_consultar_erp", "input":{"tabela":"vendas","mes":"outubro"}},

    {"ordem":2, "tool":"tool_gerar_pdf", "input":{"modelo":"financeiro"}},
    {"ordem":3, "tool":"tool_enviar_email", "input":{"destinatario":"financeiro@empresa.
    com"}}
  ],
  "regras": {"timeout":"60s","retry":1},
  "status": "pendente"
}
```

2.4.3. Tipos de Tasks

- **Sequencial:** executa em ordem fixa (Buscar dados $\rightarrow$ gerar relatório).
- **Condicional:** caminho varia conforme resultados (Se erro $\rightarrow$ alerta).
- **Paralela:** etapas simultâneas (coletar dados de 3 APIs).
- **Recursiva:** cria subtarefas até atingir objetivo (processar múltiplos clientes).

2.4.4. Boas práticas

- Sempre definir timeout máximo para a task.
- Manter steps atômicos (uma Tool = uma ação).
- Nomear tasks de forma declarativa (ex.: `task_atualizar_cliente`).
- Registrar parâmetros de entrada e saída para rastreabilidade.

2.5. Camada 3 — TOOLS (Execução e Integração)

As Tools são os mecanismos reais de execução — wrappers padronizados que conectam a plataforma a serviços, scripts ou modelos.

2.5.1. Funções principais

- Executar chamadas externas (API, DB, LLM).
- Retornar outputs padronizados.

- Lidar com erros, tempos limite e logs.
- Fornecer interfaces seguras e auditáveis.

2.5.2. Estrutura conceitual de uma Tool

Listing 3: Exemplo JSON simplificado de uma Tool

```
{
  "id": "tool_consultar_erp",
  "tipo": "API",
  "nome": "Consulta ERP",
  "descricao": "Recupera dados de vendas via ERP central",
  "input_schema": { "tabela":"string","mes":"string" },
  "output_schema": { "resultado":"json","status":"string" },
  "execucao": { "endpoint":"https://erp.exemplo.com/api/vendas","method":"POST","auth":"
    oauth2" }
}
```

2.5.3. Tipos comuns de Tools

Tabela 5: Tipos comuns de Tools

Tipo	Função / Exemplo
API REST	Chamadas HTTP externas (ERP, CRM, Hotmart).
Database	Consultas SQL ou NoSQL (`tool_query_sql`).
Script Local	Execução de scripts Python/JS (`tool_generate_pdf`).
LLM	Interpretação ou geração via modelos (`tool_prompt_gpt`).
Workflow Externo	Ponte para orquestradores externos (`tool_zapier_bridge`).

2.5.4. Boas práticas

- Usar schemas claros de `input`/`output`.
- Registrar logs de execução (correlation id, timestamps).
- Evitar lógica de negócio dentro da Tool — mantenha comportamento determinístico.
- Implementar retries controlados e timeouts.

2.6. Comunicação entre camadas

A comunicação é feita por mensagens padronizadas (tipicamente JSON), garantindo rastreabilidade e modularidade. Um exemplo de envelope de mensagem:

Listing 4: Estrutura padrão de mensagem entre camadas

```
{
  "context": { "user":"leedjr", "session_id":"abc123", "intent":"gerar_relatorio_vendas"
    },
  "task": { "id":"task_gerar_relatorio", "parameters":{ "mes":"outubro" } },
```

```
  "response": { "status":"success", "output":"Relatório PDF gerado com sucesso" }
}
```

Cada camada adiciona metadados, logs e um trace-id, formando um histórico completo da execução cognitiva.

2.7. Versionamento e rastreabilidade

Recomendamos que cada entidade (Agent, Task, Tool) contenha:

- `version` para controle de atualização;
- `author` e `date` para origem;
- `log_id` vinculado ao histórico de execução;
- `checksum` para garantir integridade dos artefatos.

Atenção

Esses metadados alimentam governança: sem eles, auditoria e conformidade tornam-se improváveis.

2.8. Governança e segurança entre camadas

Tabela 6: Mecanismos de governança e sua função

Mecanismo	Função / Aplicação
Autenticação	Garante que somente agentes autorizados executem tasks (API keys, tokens).
Escopos de execução	Limita o que cada Tool pode acessar (sandbox, RBAC).
Auditoria cruzada	Cada camada loga sua execução (logs distribuídos e imutáveis).
Reputação	Mede desempenho e confiabilidade de agentes (opcional: RI - Reputation Index).

2.9. Diagrama conceitual resumido

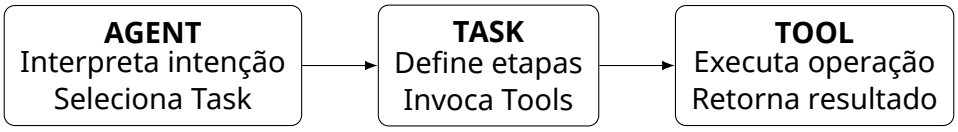

Figura 4: Resumo conceitual das três camadas principais.

2.10. Conclusão do capítulo

A estrutura Agents → Tasks → Tools é o alicerce para orquestração cognitiva: separa raciocínio (agents), planejamento (tasks) e execução (tools), entregando modularidade, rastreabili-

dade e reuso.

Dica

Com essa arquitetura, uma empresa transforma automações em produtos cognitivos licensiáveis, governados centralmente e atualizáveis sem reescrever a base de código.

Apêndice: Exemplos e Templates

Template JSON resumido

```
{
  "agent": { "id":"ag_exemplo", "version":"1.0" },
  "task": { "id":"task_exemplo", "steps":[] },
  "tools": [ {"id":"tool_exemplo"} ]
}
```

Observações sobre figuras e arquivos auxiliares

Caso deseje figuras finais, insira os arquivos em `figs/` e atualize os `\includegraphics` nas legendas correspondentes. Ex.: `figs/fluxo-agents.png`.

3. Capítulo 3 — Papel do NLP e do Planejamento Cognitivo

3.1. Introdução

Todo fluxo cognitivo começa com uma intenção humana — uma frase, um comando, uma pergunta ou um evento. Essa intenção precisa ser compreendida, estruturada e traduzida para que a plataforma saiba o que fazer.

Essa tradução ocorre em duas etapas principais, conforme a tabela 7.

Tabela 7: Mecanismos de interpretação e planejamento

Mecanismo	Função	Etapa
NLP (Natural Language Processing)	Compreende linguagem e extrai intenção e parâmetros	Interpretação
Planejamento Cognitivo	Constrói o plano de execução (Tasks + Tools) com base na intenção	Planejamento

3.2. Entendendo a intenção (NLP)

O NLP transforma linguagem natural em estrutura semântica interpretável. Envolve três subetapas fundamentais:

- **Classificação da intenção:** o que o usuário quer fazer.
- **Extração de entidades:** com quem / o que está envolvido.
- **Contextualização:** onde e quando a ação ocorre.

3.2.1. Exemplo prático

Listing 5: Estrutura semântica derivada de NLP

```
{
  "intent": "enviar_relatorio_vendas",
  "entities": {
    "unidade": "matriz",
    "destinatario": "financeiro"
  },
  "confidence": 0.93,
  "language": "pt-BR"
}
```

Nota

O Agent entende o que precisa ser feito, mas ainda não sabe como — essa é a função do Planejamento Cognitivo.

3.3. O Planejamento Cognitivo

O **Planejador Cognitivo (Cognitive Planner)** é a camada de raciocínio que transforma intenções em planos de ação — ou seja, responde à pergunta: "Quais etapas preciso executar para cumprir essa intenção?"

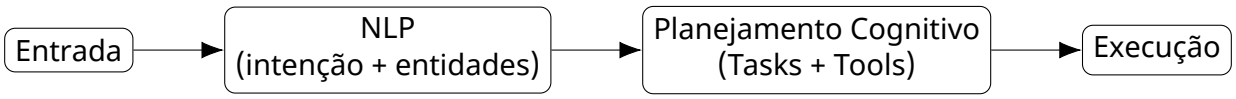

Figura 5: Pipeline simplificado de interpretação e execução cognitiva.

3.4. O ciclo interno de decisão do Agent

Cada Agent segue um ciclo RACI — Recognize → Analyze → Create → Implement — representado abaixo:

1. Receber input (linguagem natural, evento ou API call).
2. Interpretar intenção via NLP + contexto.
3. Resolver parâmetros com base em entidades e memória.
4. Criar plano (lista de Tasks).
5. Delegar execução às Tools.
6. Avaliar resultado e gerar feedback.

3.5. Estrutura lógica da decisão cognitiva

O Agent balanceia três dimensões principais:

Tabela 8: Dimensões cognitivas de decisão

Dimensão	Função	Exemplo
Semântica	O que o usuário quis dizer	"Gerar relatório" → ação = gerar
Contextual	O que está em andamento	Sessão ativa, empresa = matriz
Operacional	O que é possível executar	Tool `report_generator` disponível

A interseção dessas três dimensões define o plano cognitivo ótimo.

3.6. Exemplo completo — Do prompt à execução

Listing 6: Fluxo completo do raciocínio cognitivo

```
Entrada: "Quero o relatório financeiro de outubro em PDF."

NLP:
{
  "intent": "gerar_relatorio_financeiro",
  "entities": {"mes": "outubro", "formato": "pdf"},
  "confidence": 0.96
```

```
}

Planejamento cognitivo:
{
  "task_plan": [
    {"task": "buscar_dados_financeiros", "params": {"mes": "outubro"}},
    {"task": "gerar_relatorio_pdf", "params": {"template": "financeiro"}},
    {"task": "enviar_relatorio", "params": {"destino": "usuário"}}
  ]
}

Execução:
Tools chamadas: query_db, generate_pdf, send_email.
Resultado: Relatório financeiro enviado.
```

3.7. Resolução de parâmetros

Nem sempre o usuário fornece todas as informações. O Agent deve então:

- Deduzir do contexto.
- Perguntar de volta (loop de diálogo).
- Abort ar com mensagem clara, se necessário.

Listing 7: Fluxo de resolução de parâmetros

```
Usuário → "Atualize o cadastro do cliente João"↓

Agent → intenção = "atualizar_cliente"↓

Verifica parâmetros:
  - cliente: "João"
  - campo: ? ↓

Pergunta: "Qual campo deseja atualizar?"↓

Usuário → "O endereço"↓

Executa: atualizar_endereco_cliente("João")
```

Nota

Esse processo é conhecido como Human-in-the-loop Cognitivo — essencial para garantir confiança e transparência na execução.

3.8. Feedback cognitivo e aprendizado

Cada execução gera feedbacks que alimentam o aprendizado da plataforma:

- Se uma Task falhar → o Agent aprende qual Tool é menos confiável.
- Se o usuário corrigir algo → o modelo refina o mapeamento de intenção.
- Se houver ambiguidades → adicionam-se novos sinônimos e padrões.

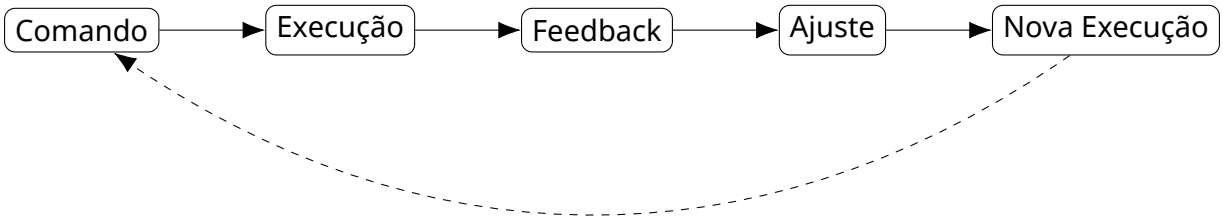

Figura 6: Ciclo de Aprendizado Cognitivo (CAC).

3.8.1. Evolução da confiança ao longo das execuções

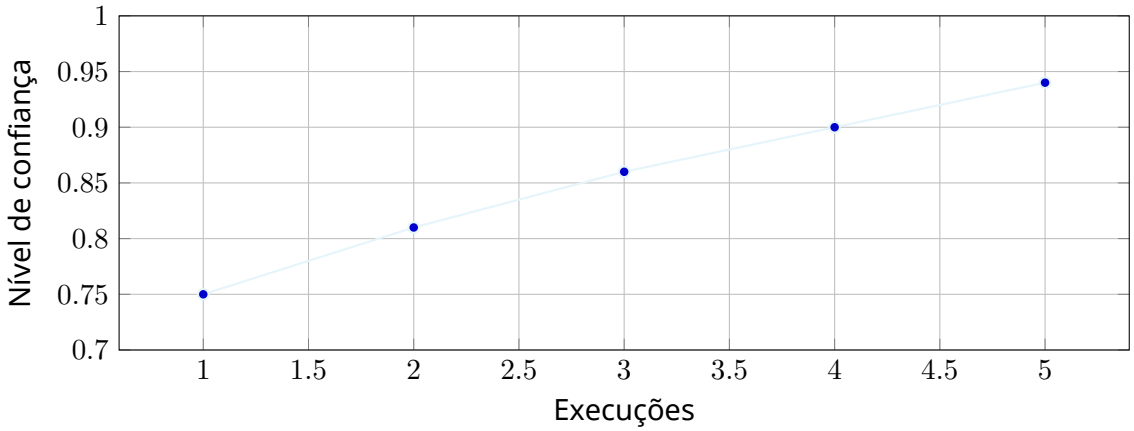

Figura 7: Aprimoramento do nível de confiança após sucessivos feedbacks.

3.9. Estrutura JSON completa de raciocínio do Agent

Listing 8: Exemplo completo de raciocínio cognitivo

```
{
  "agent": "ag_financeiro",
  "input": "Quero o relatório de outubro",
  "nlp": {
    "intent": "gerar_relatorio",
    "entities": { "mes": "outubro" },
    "confidence": 0.91
  },
  "planner": {
    "task_plan": [
      { "task": "buscar_dados", "params": { "mes": "outubro" } },
      { "task": "gerar_pdf", "params": { "modelo": "financeiro" } }
    ]
```

```
  },
  "execution": {
    "status": "success",
    "tools": [ "tool_query_erp", "tool_pdf_generator" ]
  },
  "feedback": {
    "latency": "1.8s",
    "accuracy": "ok",
    "user_satisfaction": 1
  }
}
```

3.10. Boas práticas no design de raciocínio cognitivo

- Armazene sempre o input original e a intenção resolvida.
- Implemente controle de confiança (`confidence score`).
- Prefira pedir confirmação a agir por suposição.
- Registre logs detalhados do planejador.
- Mantenha separação entre interpretação, planejamento e execução.

3.11. Diagrama conceitual do fluxo cognitivo

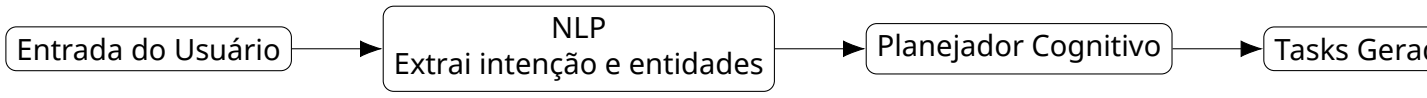

Figura 8: Fluxo cognitivo completo do NLP à execução.

3.12. Conclusão do capítulo

O NLP é a porta de entrada da orquestração; o planejador cognitivo é o cérebro que transforma intenção em plano. Juntos, conectam linguagem humana e execução técnica, tornando sistemas corporativos mais autônomos e adaptáveis.

Dica

No próximo capítulo — **Design de Fluxos Cognitivos** — veremos como representar graficamente as Tasks e Tools em modelos reutilizáveis e versionáveis.

4. Capítulo 4 — Design de Fluxos Cognitivos

4.1. O que é um fluxo cognitivo

Um fluxo cognitivo é a representação formal e visual de um raciocínio de IA aplicado a um processo de negócio.

Ele mostra como um Agent interpreta uma intenção, cria Tasks, executa Tools e gera resultados mensuráveis.

Dica

Imagine uma linha de produção automatizada — mas, em vez de peças, ela processa inteligência: **Intenção** → **Interpretação** → **Planejamento** → **Execução** → **Resultado**.

Cada etapa tem insumos, regras e responsáveis (Agents, Tasks, Tools).

4.2. Estrutura básica de um fluxo cognitivo

Um fluxo cognitivo completo é composto por 5 blocos principais:

Tabela 9: Blocos principais de um fluxo cognitivo

Bloco	Função	Exemplo
Trigger (Gatilho)	Evento inicial que dispara o fluxo.	Comando do usuário, webhook, agenda.
Agent	Interpreta a intenção e cria plano.	ag_financeiro
Tasks	Etapas planejadas de execução.	buscar_dados, gerar_pdf, enviar_email
Tools	Ações concretas executadas.	tool_query_db, tool_send_mail
Output	Resultado final e feedback.	“Relatório enviado.”

(Trigger) ↓ [AGENT] → entende intenção ↓ [TASK 1] → define o que fazer ↓ [TOOL 1] → executa (consulta ERP) ↓ [TASK 2] → próxima etapa ↓ [TOOL 2] → executa (gera PDF) ↓ [OUTPUT] → resultado e logs

4.3. Tipos de fluxos cognitivos

4.4. Etapas do design de um fluxo cognitivo

1. Mapear a intenção principal.
2. Identificar o gatilho inicial.
3. Definir subintenções (Tasks).
4. Selecionar Tools disponíveis.
5. Definir parâmetros obrigatórios.

Tabela 10: Tipos de fluxos cognitivos

Tipo	Característica	Exemplo prático
Sequencial	Execução linear, previsível.	Gerar relatório → enviar PDF.
Condicional	Caminhos dependem de resultados.	Se erro → notificar suporte.
Paralelo	Tasks simultâneas, sincronizadas.	Buscar dados em 3 sistemas.
Recursivo	Subtasks geradas dinamicamente.	Processar múltiplos clientes.
Evento-Reativo	Disparado por eventos externos.	Ao criar cliente → enviar boas-vindas.

6. Projetar o fluxo visualmente (Drawflow, Mermaid, Miro).
7. Definir métricas e logs.

4.5. Exemplo de blueprint visual

Fluxo: Gerar e enviar relatório de vendas.

```
(Trigger: comando do usuário)
   ↓
[Agent: ag_financeiro]
   ↓
[TASK: validar_parâmetros]
      Tool: NLPValidator
   ↓
[TASK: buscar_dados]
      Tool: ERPQuery
   ↓
[TASK: gerar_pdf]
      Tool: PDFBuilder
   ↓
[TASK: enviar_email]
      Tool: MailerAPI
   ↓
[OUTPUT: sucesso / erro / feedback]
```

Listing 9: Estrutura JSON correspondente

```
{
  "flow_id": "flow_relatorio_vendas_v1",
  "intent": "gerar_relatorio_vendas",
  "agent": "ag_financeiro",
  "tasks": [
    { "id": "task_validar", "tool": "tool_nlp_validator", "input": { "campos": ["mês", "
    destinatário"] } },
    { "id": "task_buscar_dados", "tool": "tool_erp_query", "input": { "tabela": "vendas" }
```

```
    },
    { "id": "task_gerar_pdf", "tool": "tool_pdf_builder", "input": { "modelo": "
    financeiro" } },
    { "id": "task_enviar_email", "tool": "tool_mailer_api", "input": { "para": "
    financeiro@empresa.com" } }
  ],
  "output": { "mensagem": "Relatório gerado e enviado com sucesso", "status": "success" }
}
```

4.6. Encadeamento de múltiplos fluxos

Em plataformas maiores, um Agent pode acionar outros Agents, criando uma rede de fluxos.

```
[Agente Financeiro]
      Task: Gerar Relatório
           [Tool: ERP]
      Task: Validar Resultado
           [Agente Auditor]
                 [Tool: AI_Validator]
```

4.7. Boas práticas de design de fluxo

- Modularize cada Task para ser reutilizável.
- Nomeie semanticamente as Tasks.
- Padronize entradas e saídas.
- Inclua logs e métricas desde o início.
- Desenhe antes de codificar.
- Defina fallback e exceções.

4.8. Camada visual e documentação

Cada fluxo deve ter sua versão visual documentada:

- Nó de Agent (azul) — início da decisão.
- Nó de Task (verde) — etapa de planejamento.
- Nó de Tool (laranja) — ação executável.
- Nó de Output (cinza) — resultados e logs.

Dica

Sugestão visual: usar um diagrama SVG com animação mostrando o fluxo de dados e as mensagens JSON entre cada nó.

4.9. Estrutura de versionamento do fluxo

Listing 10: Metadados obrigatórios de um fluxo

```
{
  "flow_id": "flow_relatorio_vendas_v1",
  "version": "1.0.0",
  "author": "Leed Jr",
  "last_updated": "2025-11-09",
  "tags": ["financeiro", "relatorio", "erp"],
  "description": "Fluxo completo de geração e envio de relatórios de vendas.",
  "dependencies": ["ag_financeiro", "tool_pdf_builder"]
}
```

4.10. Modelo de documentação de fluxo (template)

Tabela 11: Template de documentação de fluxo

Campo	Descrição	Exemplo
Flow ID	Identificador único	flow_finance_report_v2
Intenção Principal	Ação cognitiva principal	“Gerar relatório financeiro mensal”
Agente Responsável	Quem interpreta e executa	ag_financeiro
Tasks Incluídas	Etapas do fluxo	validar_dados, consultar_db, gerar_pdf, enviar_email
Ferramentas Usadas	Dependências externas	ERP API, PDFBuilder, Mailer
Parâmetros Necessários	Entradas esperadas	mês, formato, destinatário
Outputs Esperados	Saída do fluxo	PDF + confirmação por e-mail
Fallback	Ação alternativa em caso de erro	Reagendar tentativa + notificar suporte
Autor / Versão / Data	Controle de governança	Leed Jr, v1.2, 2025-11-09

4.11. Fluxos como ativos licenciáveis

Cada fluxo documentado pode ser:

- Exportado como JSON.
- Versionado em repositórios.
- Implantado em diferentes instâncias de cliente.
- Licenciado como ativo cognitivo.

Nota

Fluxos bem projetados não são apenas automações — são produtos de inteligência prontos para gerar receita.

4.12. Conclusão do capítulo

O design de fluxos cognitivos é a arte de traduzir raciocínio em arquitetura. Ele define o esqueleto do produto, a rastreabilidade da IA e a base para monetização.

Dica

No próximo capítulo, exploraremos a Padronização e Governança dos Fluxos, incluindo versionamento, auditoria e repositórios cognitivos.

5. Capítulo 5 — Padronização e Governança dos Fluxos

6. Padronização e Governança dos Fluxos

6.1. O propósito da governança cognitiva

A governança cognitiva é o conjunto de políticas, padrões e mecanismos que garantem que cada fluxo, agente e tool da plataforma:

- seja compreensível;
- tenha rastreamento completo;
- e possa ser atualizado, auditado e licenciado com segurança.

Sem governança, uma plataforma cognitiva vira um conjunto de scripts. Com governança, ela se torna uma infraestrutura corporativa de inteligência.

6.2. Os quatro pilares da governança

Tabela 12: Os quatro pilares da governança cognitiva

Pilar	Descrição	Benefício
Padronização	Todos os fluxos seguem o mesmo formato, nomenclatura e versionamento.	Facilita manutenção e reuso.
Auditoria	Cada execução gera logs rastreáveis.	Aumenta confiabilidade e conformidade.
Controle de versão	Mudanças são registradas e reversíveis.	Garante estabilidade e histórico.
Catálogo Cognitivo	Todos os fluxos são indexados e documentados.	Permite descoberta e licenciamento.

6.3. Estrutura de versionamento semântico

Cada elemento (Agent, Task, Tool ou Flow) deve possuir um identificador de versão semântica, seguindo o padrão **MAJOR.MINOR.PATCH**.

Tabela 13: Componentes da versão semântica

Parte	Significado	Exemplo
MAJOR	Mudança estrutural (quebra compatibilidade).	2
MINOR	Adição de funcionalidade sem quebra.	1
PATCH	Correção de bug ou melhoria leve.	3

Exemplo: `v2.1.3` → Segunda geração do fluxo, com novas tasks e pequenas correções.

Listing 11: Estrutura recomendada de metadados

```
{
```

```
  "version": "1.2.0",
  "created_at": "2025-11-09",
  "updated_at": "2025-11-09",
  "author": "Leed Jr",
  "status": "stable",
  "changelog": [
    "v1.2.0 - Adicionado fallback de envio de e-mail.",
    "v1.1.0 - Otimizada consulta ERP.",
    "v1.0.0 - Versão inicial do fluxo."
  ]
}
```

6.4. Regras de padronização de nomenclatura

Tabela 14: Regras de nomenclatura padronizada

Tipo	Formato recomendado	Exemplo
Flow ID	flow_<area>_<ação>_v<versão>	flow_financeiro_relatorio_v1
Agent ID	ag_<domínio>	ag_financeiro
Task ID	task_<ação>	task_gerar_pdf
Tool ID	tool_<função>	tool_send_mail
Logs	log_<data>_<hash>	log_2025_11_09_af23c

Dica

Use nomes descritivos e evite abreviações obscuras — clareza é parte da governança.

6.5. Estrutura padrão de documentação

Listing 12: Exemplo de documentação padrão de fluxo

```
{
  "flow_id": "flow_relatorio_financeiro_v2",
  "title": "Geração e Envio de Relatórios Financeiros",
  "author": "Leed Jr",
  "description": "Fluxo cognitivo que gera e envia relatórios financeiros mensais via ERP
     e e-mail.",
  "version": "2.0.1",
  "tags": ["financeiro", "erp", "relatorio", "pdf"],
  "agents": ["ag_financeiro"],
  "tasks": ["task_buscar_dados", "task_gerar_pdf", "task_enviar_email"],
  "tools": ["tool_erp_query", "tool_pdf_builder", "tool_mailer"],
  "dependencies": ["tool_pdf_builder >= 1.0.0"],
  "last_run": "2025-11-05T14:23:10Z",
  "metrics": {
    "avg_runtime": "2.3s",
    "success_rate": "98.4%",
```

```
    "failure_rate": "1.6%"
  }
}
```

6.6. Estrutura de logs e rastreabilidade

Cada execução (run) deve gerar um registro auditável completo:

Tabela 15: Campos mínimos de um log de execução

Campo	Descrição	Exemplo
run_id	Identificador único	run_2025_11_09_abc123
flow_id	Fluxo executado	flow_relatorio_financeiro_v2
agent_id	Agente responsável	ag_financeiro
input	Entrada original	“Gerar relatório de outubro”
parameters	Parâmetros resolvidos	{ ”mes”: ”outubro”}
tools_executadas	Tools usadas	[”tool_erp_query”, ”tool_pdf_builder”]
output	Resultado final	“Relatório PDF enviado.”
status	Sucesso ou falha	success
tempo_execucao	Duração total	2.41s
erro_log	Descrição de erro	SMTP timeout

6.7. Auditoria cruzada

Listing 13: Exemplo de auditoria cruzada com trace_id

```
{
  "trace_id": "abc-123",
  "agent_log": "Gerar relatório de vendas",
  "task_log": ["buscar_dados", "gerar_pdf"],
  "tool_log": ["ERPQuery=ok", "PDFBuilder=ok"]
}
```

6.8. Métricas e observabilidade cognitiva

Tabela 16: Métricas essenciais de observabilidade cognitiva

Métrica	Descrição	Unidade
Success Rate	% de execuções bem-sucedidas	%
Mean Execution Time (MET)	Tempo médio por fluxo	segundos
Error Rate	% de falhas	%
Tool Latency	Tempo médio por Tool	ms
Custo Cognitivo	Tokens + compute time + API calls	$
User Satisfaction Index (USI)	Feedback do usuário (0–1)	índice

6.9. O Catálogo de Fluxos Cognitivos

O Catálogo Cognitivo é o repositório central que registra todos os fluxos, agentes e tools.

Listing 14: Exemplo de entrada de catálogo

```
{
  "id": "flow_relatorio_financeiro_v2",
  "nome": "Geração de Relatórios Financeiros",
  "descricao": "Fluxo completo de consulta, geração e envio de relatórios financeiros.",
  "autor": "Leed Jr",
  "tags": ["financeiro", "erp", "relatorio"],
  "licenca": "Enterprise",
  "versao": "2.0.1",
  "status": "ativo"
}
```

6.10. Políticas de versionamento e ciclo de vida

Tabela 17: Estados de ciclo de vida dos fluxos

Estado	Descrição	Ação recomendada
Draft	Em desenvolvimento	Não disponível para produção.
Stable	Testado e validado	Pode ser publicado e licenciado.
Deprecated	Substituído por nova versão	Avisar usuários e migrar.
Archived	Fora de uso	Mantido para auditoria.

6.11. Regras de auditoria e conformidade

- Todas as execuções devem ter `trace_id` e `timestamp`.
- Nenhum fluxo pode ser executado sem versão registrada.
- Alterações só por usuários com papel `Maintainer`.
- Logs devem ser imutáveis (apenas append).
- Backups de catálogo e logs devem ser diários.

Essas regras garantem conformidade com **LGPD, GDPR** e o **AI Act** europeu.

6.12. Exemplo de painel de governança

Elementos visuais recomendados:

- Tabela de fluxos ativos (nome, versão, status, sucesso, tempo médio).
- Gráfico de latência por Tool.
- Mapa de relacionamentos (Agent $\leftrightarrow$ Task $\leftrightarrow$ Tool).
- Linha do tempo de execuções (trace view).

6.13. Checklist de governança cognitiva

Tabela 18: Checklist de governança cognitiva

Item	Verificado
Cada fluxo possui metadados completos (autor, versão, changelog)?	☐
Existe catálogo atualizado de fluxos e tools?	☐
Logs possuem trace_id e timestamps precisos?	☐
Métricas de execução são coletadas?	☐
Há política de backup e auditoria imutável?	☐
Há controle de acesso e perfis de usuário?	☐

6.14. Conclusão do capítulo

Governança é o que transforma automação em patrimônio. Um fluxo cognitivo bem governado é rastreável, confiável, versionável e vendável. Essa camada é o "cérebro administrativo" da plataforma — base para compliance, segurança e monetização.

Dica

No Capítulo 6 — Construção do Agente Raiz (Orchestrator), veremos como o Agent Orquestrador gerencia e roteia fluxos de forma inteligente.

7. Capítulo 6 — O Agente Raiz (Orchestrator)

7.1. Conceito e propósito

O **Agente Raiz** (Orchestrator) é o núcleo inteligente da plataforma cognitiva. Suas responsabilidades principais incluem:

- Receber inputs (texto, eventos, API calls, webhooks);
- Interpretar a intenção (via NLP rápido ou meta-agent);
- Escolher qual Agent de domínio deve atuar;
- Solicitar e validar planos cognitivos;
- Gerenciar a execução paralela de Tasks e Tools;
- Aplicar políticas de segurança, custo e reputação;
- Registrar logs, métricas e auditoria completa de cada decisão.

O Orchestrator é o "sistema nervoso central": pensa globalmente e executa localmente.

7.2. Arquitetura em camadas

Tabela 19: Camadas da arquitetura do Orchestrator

Camada	Função	Exemplo / API
Input Layer	Recebe e normaliza triggers.	Texto, eventos, cron jobs, APIs.
Routing Layer	Decide qual agente deve atuar.	`routeIntent()`
Planning Layer	Solicita e valida planos cognitivos.	`requestPlan()`
Execution Layer	Executa Tasks/Tools conforme plano.	`ExecutionEngine`
Governance Layer	Aplica políticas, quotas e logging.	`agent_policy.json`

7.3. Diagrama conceitual do fluxo

7.4. Regras de decisão e roteamento

Tabela 20: Regras de roteamento por `confidence` e condições

Critério	Ação
`confidence` ≥ 0.9	Executa diretamente com Agent correspondente.
$0.6 \leq$ `confidence` < 0.9	Executa sob supervisão (modo seguro).
`confidence` < 0.6	Retorna ao usuário para confirmação (Human-in-the-Loop).
`agent_overloaded = true`	Redireciona para Agent secundário.
`policy.budget_exceeded = true`	Solicita autorização adicional ou adia execução.

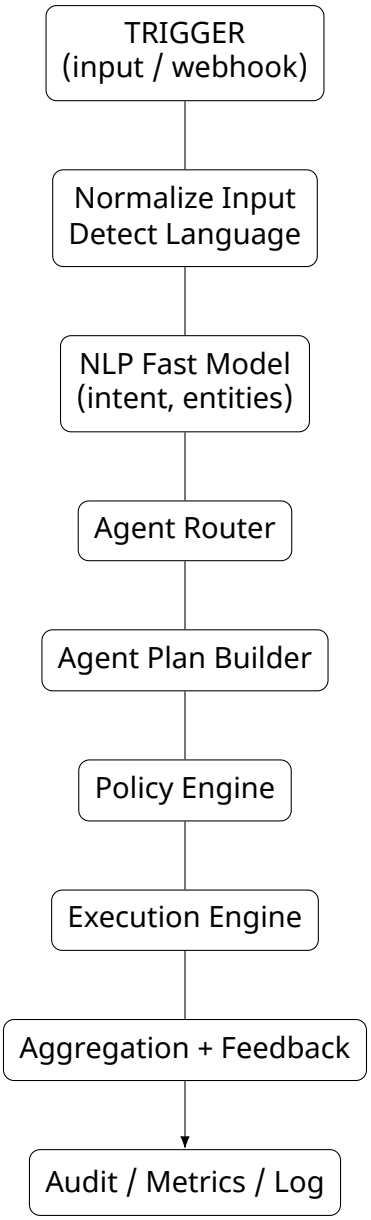

Figura 9: Fluxo conceitual do Orchestrator — do trigger ao audit.

7.5. Estrutura de mensagens padrão

Trigger → Orchestrator

Listing 15: Trigger enviado ao Orchestrator

```
{
  "trace_id": "trace_2025_11_09_0001",
  "session_id": "sess_abc123",
  "user_id": "usr_fin_01",
  "input": "Gerar relatório de vendas de outubro",
  "locale": "pt-BR",
  "channel": "portal_web",
  "metadata": { "tenant": "cliente_A", "priority": "normal" }
}
```

Orchestrator → Agent (request_plan)

Listing 16: Request plan para Agent

```
{
  "trace_id": "trace_2025_11_09_0001",
  "agent_id": "ag_financeiro",
  "intent": "gerar_relatorio_vendas",
  "entities": { "mes": "outubro" },
  "context": { "tenant": "cliente_A" },
  "confidence": 0.92
}
```

Agent → Orchestrator (plan)

Listing 17: Plano retornado pelo Agent

```
{
  "agent_id": "ag_financeiro",
  "plan": {
    "tasks": [
      { "id": "task_validar_parametros" },
      { "id": "task_buscar_dados" },
      { "id": "task_gerar_pdf" },
      { "id": "task_enviar_email" }
    ],
    "estimated_runtime": 3.5,
    "estimated_cost": 0.02,
    "fallback": "notify_operator"
  }
}
```

7.6. Pseudocódigo do Orchestrator

Listing 18: Pseudocódigo: fluxo principal do Orchestrator

```
async function orchestratorTrigger(trigger) {
  const traceId = trigger.trace_id || uuid();
  log.info(traceId, "Trigger recebido", trigger);

  // Normalização e NLP
  const normalized = normalizeInput(trigger.input);
  const nlp = await fastNLP(normalized, trigger.locale);

  // Escolha de Agent
  const agent = await routeIntent(nlp.intent, nlp.confidence, trigger.metadata);
  if (!agent) return askClarification(traceId, trigger);

  // Solicitar plano
```

```
  const plan = await requestPlan(agent, nlp, trigger);
  if (!plan) return handleNoPlan(traceId, agent);

  // Validação de políticas
  if (!checkPolicies(agent, plan, trigger.metadata)) {
    return denyExecution(traceId, plan);
  }

  // Execução
  const runId = `run_${Date.now()}`;
  await executionEngine.runPlan(runId, traceId, plan, trigger.session_id);

  // Feedback final
  return { traceId, status: "executando", runId };
}
```

7.7. Execution Engine — estrutura e ciclo

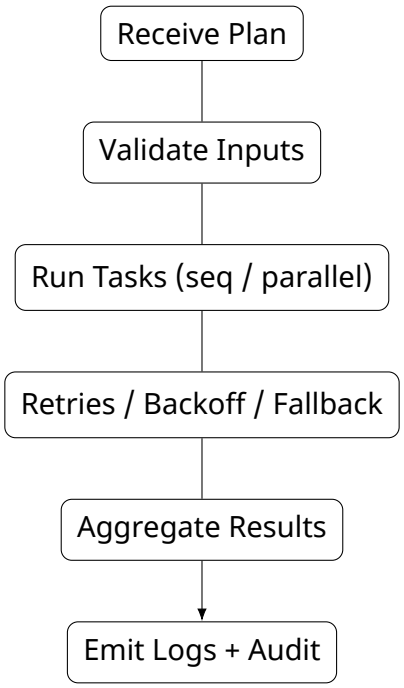

Figura 10: Ciclo do Execution Engine

Listing 19: Trecho do ExecutionEngine (pseudocódigo)

```
class ExecutionEngine {
  async runPlan(runId, traceId, plan, sessionId) {
    for (const task of plan.tasks) {
      const policy = await loadTaskPolicy(task.id);
      try {
        if (policy.parallel) {
          await Promise.all(task.tools.map(t => execToolWithPolicy(t, traceId)));
        } else {
```

```
          for (const tool of task.tools) {
            const result = await execToolWithPolicy(tool, traceId);
            if (result.status === "error") throw result.error;
          }
        }
      } catch (err) {
        log.error(traceId, "Erro na task", { taskId: task.id, err: err.message });
        const recovered = await this.applyFallback(task, err, traceId);
        if (!recovered) {
          await this.persistStatus(runId, "failed", err.message);
          return;
        }
      }
    }
    await this.persistStatus(runId, "success");
    log.info(traceId, "Execução concluída", { runId });
  }

  async applyFallback(task, error, traceId) {
    const policy = await loadTaskPolicy(task.id);
    if (policy.fallback === "retry_once") {
      log.warn(traceId, "Retry após falha", { taskId: task.id });
      return this.runTask(task);
    } else if (policy.fallback === "notify_operator") {
      notifyHumanOperator(task, error);
      return false;
    }
    return false;
  }
}
```

7.8. Políticas de execução (exemplo)

Listing 20: task_policy.json (exemplo)

```
{
  "task_id": "task_buscar_dados",
  "version": "1.1.0",
  "timeout_ms": 20000,
  "retries": 2,
  "backoff": { "type": "exponential", "base_ms": 500 },
  "parallel": false,
  "fallback": "notify_operator",
  "on_failure": "abort_plan",
  "dependencies": ["tool_erp_query"]
}
```

7.9. Políticas do agente (exemplo)

Listing 21: agent_policy.json (exemplo)

```
{
  "agent_id": "ag_financeiro",
  "version": "1.2.0",
  "allowed_intents": ["gerar_relatorio_vendas", "consultar_faturamento"],
  "max_cost_per_run": 0.05,
  "min_confidence": 0.7,
  "budget_per_day": 10.0,
  "max_parallel_tasks": 3,
  "allowed_tools": ["tool_erp_query", "tool_pdf_builder", "tool_mailer"],
  "fallback": { "type": "escalar", "action": "notificar_suporte" },
  "security": {
    "required_scopes": ["financeiro:read", "financeiro:report"],
    "tenant_restricted": true
  },
  "logging": { "level": "info", "audit": true }
}
```

7.10. Paralelismo, sincronização e priorização

- **Sequencial:** execução linear — ex.: consultar ERP → gerar PDF.
- **Paralela:** tarefas independentes em simultâneo — ex.: consultar 3 filiais.
- **Condicional:** caminho baseado em resultados anteriores.

O Execution Engine mantém um pool de workers limitado por `max_parallel_tasks` nas políticas.

Fórmula de priorização (PriorityScore)

$$\text{PriorityScore} = 0.5 \times \text{importance} + 0.3 \times \left(1 - \frac{\text{estimated\_runtime}}{\text{max\_runtime}}\right) + 0.2 \times \text{user\_tier}$$

Fluxos com maior `PriorityScore` sobem na fila de execução.

7.11. Tratamento de falhas e fallback

Tabela 21: Ações recomendadas para tipos de falha

Tipo de erro	Ação	Exemplo
Timeout (rede)	Retry com backoff	`tool_erp_query`
Falha permanente	Aciona fallback alternativo	`tool_cache_query`
Erro lógico (dados)	Abort plan	`task_validar_parametros`
Falha crítica	Escalonar humano	`notify_operator`

7.12. Métricas do Orchestrator

Métricas essenciais a serem expostas ao sistema de observabilidade:

- `runs_total` — número total de execuções;
- `runs_failed_total` — execuções com falha;
- `avg_runtime_seconds` — tempo médio por fluxo;
- `tool_latency_avg` — latência média das Tools;
- `agent_confidence_avg` — confiança média de roteamento;
- `cost_per_run` — custo médio estimado vs. real.

7.13. Checklist do Orchestrator

Tabela 22: Checklist mínimo de implementação

Item	Status
API `/trigger` implementada	☐
NLP rápido operacional	☐
Router de agentes por confiança	☐
Execution Engine (Tasks/Tools)	☐
Políticas (agent/task JSON) versionadas	☐
Logs estruturados (trace + audit)	☐
Métricas e observabilidade (Prometheus/OTel)	☐

7.14. Sugestão visual

Um diagrama circular animado pode facilitar o entendimento:
Trigger → Orchestrator → Agent → Tasks → Tools → Result → Feedback → Audit Com ênfase em: Pensar (Orchestrator), Planejar (Agent), Executar (Tools), Aprender (Governance Feedback).

7.15. Conclusão

O Agente Raiz (Orchestrator) é o núcleo orquestrador que aplica raciocínio, política e automação com rastreabilidade e segurança — transformando IA isolada em arquitetura viva, governável e auditável.

Dica

No Capítulo 7 — Execução com Ferramentas (Tools Layer) iremos detalhar adapters, padrões de integração e exemplos práticos de implementação.

8. Capítulo 7 — Execução com Ferramentas (Tools Layer)

8.1. Conceito e papel da camada de Tools

A **Tools Layer** é o nível físico de execução dentro da arquitetura cognitiva — a ponte entre o pensamento (Agents) e a ação (sistemas reais). Enquanto o Agent decide o que precisa ser feito e a Task planeja como, a Tool executa onde e com quem.

Cada Tool representa uma integração concreta capaz de interagir com: APIs (REST/GraphQL/-SOAP), LLMs, scripts/funções internas, sistemas locais (ERP/CRM/DB) e serviços de mensageria.

8.2. Estrutura lógica das Tools

Tabela 23: Camadas lógicas de uma Tool

Camada	Função / Exemplo
Tool Definition	Define metadados e políticas (ex.: `tool_pdf_builder.json`).
Tool Adapter	Traduz chamada genérica para execução específica (HTTP, LLM, Script).
Execution Policy	Regras de timeout, retry e fallback (policy.retry = 2).
Result Normalizer	Uniformiza saídas para o Orchestrator (JSON padrão).
Monitor/Audit Layer	Logs, métricas e traces (OpenTelemetry / Prometheus).

8.3. Estrutura padrão de uma Tool (JSON Definition)

Listing 22: Exemplo: definição de Tool (`tool_pdf_builder_v1`)

```
{
  "id": "tool_pdf_builder_v1",
  "name": "PDF Builder",
  "description": "Gera relatórios em PDF a partir de templates e payloads.",
  "type": "script",
  "exec": {
    "adapter": "script",
    "image": "registry.company/pdf-builder:1.4",
    "cmd": ["/app/build", "--template", "{{template}}", "--data", "{{payload}}"],
    "timeout_ms": 60000
  },
  "input_schema": { "type": "object", "required": ["template","payload"] },
  "output_schema": { "type": "object", "required": ["pdf_url"] },
  "policy": { "retries": 1, "backoff": { "type":"exponential","base_ms":500 }, "
    idempotent": true },
  "security": { "required_scopes": ["report:generate"] }
}
```

Tabela 24: Tipos de Tools e adapters recomendados

Tipo	Execução típica	Adapter recomendado
API REST/GraphQL	Requisição HTTP, JSON, OAuth2	`httpAdapter`
LLM (IA Generativa)	Prompt → resposta textual	`llmAdapter`
Script / Function	Execução containerizada local	`scriptAdapter`
SQL / DB Query	Consulta parametrizada segura	`sqlAdapter`
Mensageria / Eventos	Publish / Consume assíncrono	`queueAdapter`

8.4. Tipos de ferramentas mais comuns

8.5. Padrão Adapter — "Executar com abstração"

Para interoperabilidade, toda Tool usa um adapter que padroniza execução. Fluxo conceitual:

Task → `execToolWithPolicy(tool_id, input)` → Tool Catalog → Adapter → Exec/Monitor → Normalizec

Exemplo de interface TypeScript (conceitual):

Listing 23: Interface conceitual de Adapter

```
interface ToolAdapter {
  exec(execSpec: any, input: any, context: ExecContext): Promise<any>;
}
```

8.6. Adapters práticos (exemplos)

1) HTTP Adapter (pseudo)

Listing 24: HTTP Adapter (exemplo simplificado)

```
async function httpAdapter(execSpec, input, context) {
  const { endpoint, method, headers, auth } = execSpec;
  const token = await getToken(auth);
  const response = await fetch(endpoint, {
    method,
    headers: { "Authorization": `Bearer ${token}`, "Content-Type": "application/json" },
    body: JSON.stringify(input)
  });
  if (!response.ok) throw new Error(`HTTP ${response.status}`);
  return await response.json();
}
```

2) LLM Adapter (pseudo)

Listing 25: LLM Adapter (exemplo simplificado)

```
async function llmAdapter(execSpec, input, context) {
  const { provider, model, max_tokens, temperature } = execSpec;
  const client = getLLMClient(provider);
```

```
  const response = await client.generate({
    model,
    messages: [
      { role: "system", content: input.system || "Você é um assistente confiável." },
      { role: "user", content: input.prompt }
    ],
    max_tokens, temperature
  });
  return { output_text: response.text };
}
```

3) Script Adapter (pseudo)

Listing 26: Script Adapter (exemplo simplificado)

```
async function scriptAdapter(execSpec, input, context) {
  const { image, cmd, timeout_ms } = execSpec;
  const inputPath = `/tmp/input_${context.traceId}.json`;
  fs.writeFileSync(inputPath, JSON.stringify(input));
  const result = await runDocker({
    image,
    command: cmd.map(c => replaceTemplate(c, input)),
    volumes: { [inputPath]: "/app/input.json" },
    timeout: timeout_ms
  });
  return JSON.parse(fs.readFileSync(result.outputPath, "utf-8"));
}
```

8.7. Execução com política — `execToolWithPolicy`

Ponto de entrada genérico que aplica validação, autorização, retries e backoff.

Listing 27: Execução com políticas (esquemático)

```
async function execToolWithPolicy(toolId, input, runMeta) {
  const tool = catalog.getTool(toolId);
  const adapter = adapters[tool.exec.adapter];
  const policy = tool.policy;

  validateInput(tool.input_schema, input);
  authorizeScopes(runMeta.session, tool.security.required_scopes);

  let attempt = 0;
  while (attempt <= policy.retries) {
    attempt++;
    try {
      const result = await Promise.race([ adapter.exec(tool.exec, input, runMeta),
    timeoutPromise(policy.timeout_ms) ]);
      return result;
```

```
    } catch (err) {
      log.error(runMeta.traceId, "Erro na tool", { toolId, attempt, message: err.message
    });
      if (attempt > policy.retries) throw err;
      await wait(backoffTime(policy.backoff, attempt));
    }
  }
}
```

8.8. Tratamento de falhas e fallback

Tabela 25: Estratégias por tipo de erro

Tipo	Estratégia	Exemplo
Timeout	Retry com backoff	API instável
4xx (Client)	Corrigir input / abortar	Parâmetro inválido
5xx (Server)	Retry / fallback	ERP fora temporariamente
Falha crítica	Notificar operador / HITL	Script corrompido
Autorização	Refresh token / escalate	OAuth2 expirado

8.9. Segurança e isolamento

Boas práticas:

- Secrets: armazenados em Vault, nunca no payload.
- Execução sandbox: containers com limites de CPU/RAM.
- Tenant isolation: contextos segregados por cliente.
- Logging: evitar PII; redigir dados sensíveis.
- Scopes de acesso: validados antes da execução.

8.10. Métricas e observabilidade

Métricas recomendadas a exportar via OpenTelemetry:

- `tool_calls_total` — quantidade de execuções;
- `tool_latency_ms` — latência média por tool;
- `tool_failures_total` — falhas por tipo;
- `tool_cost_estimate` / `tool_cost_actual` — custo previsto/real;
- `tool_success_rate` — taxa de sucesso.

8.11. Integração com LLMs corporativos

Recomendações práticas:

- Versionar prompts (ex.: `prompt_v1.2.json`).

- Separar papéis `system` e `user`.
- Logar prompt/response para análise (sem PII).
- Substituir PII por tokens contextuais (`{CLIENTE_ID}`).
- Para tarefas determinísticas, usar `temperature = 0.0`.

8.12. Transações e compensações (Sagas)

Padrão para processos distribuídos: definir compensações para cada etapa.

Listing 28: Exemplo de saga (compensações)

```
{
  "task_id": "task_process_order",
  "steps": [
    { "tool": "tool_create_order", "compensate": "tool_cancel_order" },
    { "tool": "tool_charge_card", "compensate": "tool_refund_card" },
    { "tool": "tool_notify_warehouse" }
  ]
}
```

8.13. Exemplo completo: execução híbrida (API + LLM + Script)

Caso: G̈erar relatório de satisfação de clientes—¨ pipeline exemplo:

1. `tool_erp_query` (API) — buscar dados NPS;
2. `tool_llm_summary` (LLM) — gerar insights e resumo;
3. `tool_pdf_builder` (Script) — gerar PDF e armazenar;
4. `tool_mailer` (API) — enviar relatório por e-mail.

Fluxo JSON (resumo):

Listing 29: Plano de execução: relatório de satisfação

```
{
  "plan_id": "plan_satisfacao_cliente_v1",
  "tasks": [
    { "id": "fetch_nps", "tool": "tool_erp_query" },
    { "id": "analyze_nps", "tool": "tool_llm_summary" },
    { "id": "generate_pdf", "tool": "tool_pdf_builder" },
    { "id": "send_email", "tool": "tool_mailer" }
  ]
}
```

Tabela 26: Checklist mínimo para a Tools Layer

Item	Status
Catálogo de Tools (schema + políticas)	☐
Adapters (HTTP, LLM, Script, SQL)	☐
Execução com retries e timeout (`execToolWithPolicy`)	☐
Fallback configurável nas policies	☐
Idempotência e circuit breaker	☐
Logging + métricas com `traceId`	☐
Testes: unitários e integração (mocks)	☐

8.14. Checklist de implementação da Tools Layer

8.15. Sugestão visual

Fluxo modular horizontal:

[Task] → [Tool Catalog] → [Adapter (HTTP/LLM/Script)] → [Execution + Policy Engine] → [Result Normali

Use ícones coloridos para diferenciar APIs, LLMs e Scripts nos diagramas didáticos.

8.16. Conclusão

A Tools Layer é onde o raciocínio cognitivo encontra o mundo real. Projetada com adapters, políticas e observabilidade, ela garante que ações sejam executadas com segurança, rastreabilidade e previsibilidade — transformando intenções em resultados concretos.

Dica

No Capítulo 8 — Segurança, Controle e Auditoria — detalharemos sandboxing, gestão de segredos, políticas de acesso e trilhas de auditoria para ambientes corporativos.

9. Capítulo 8 — Segurança, Controle e Auditoria

9.1. Propósito da camada de segurança

A camada de Segurança, Controle e Auditoria garante que toda ação cognitiva — desde o pensamento do Agent até a execução de uma Tool — seja segura, rastreável e conforme às normas corporativas e legais. Segurança cognitiva não é apenas criptografia: é a garantia de que a inteligência age dentro de limites verificáveis.

Esta camada envolve três pilares:

- **Proteção (Security)** — impedir acessos indevidos e vazamentos;
- **Controle (Governance)** — definir quem pode o quê e até onde;
- **Auditoria (Accountability)** — registrar, explicar e justificar cada decisão da IA.

9.2. Escopo técnico da segurança cognitiva

Tabela 27: Escopo técnico e responsabilidades

Camada	Responsabilidade	Exemplo
Identidade (Auth)	Garantir origem legítima	JWT, OAuth2, mTLS
Autorização (Access Control)	Controlar escopos e permissões	RBAC, ABAC, scope-based
Execução Segura (Sandbox)	Isolar processos e scripts	Containers, namespaces
Governança (Policies)	Definir orçamentos, riscos e reputação	agent_policy.json
Auditoria & Explainability	Registrar raciocínio e execução	Logs, traces, decisões cognitivas

9.3. Modelo de controle de acesso (RBAC + ABAC)

A Matrix Cognitiva recomenda um modelo híbrido:

Nota

RBAC para permissões estáveis por função; ABAC para decisões dinâmicas baseadas em atributos de contexto (tenant, horário, nível de risco).

Exemplo RBAC (JSON)

Listing 30: Exemplo: Role-Based Access Control

```
{
  "role": "finance_admin",
  "permissions": ["report:read", "report:generate", "tool:use:pdf_builder"]
}
```

Exemplo ABAC (regra)

Listing 31: Exemplo: Attribute-Based Access Control

```
{
  "rule": "permitir se tenant = contexto.tenant e horario_em_expediente = true"
}
```

9.4. Políticas de segurança por camada

Tabela 28: Políticas por camada

Camada	Tipos de políticas / arquivo
Agent Policy	custos, escopos, paralelismo, reputação — `agent_policy.json`
Task Policy	retries, timeout, fallback, isolamento — `task_policy.json`
Tool Policy	escopos, tempo máximo, rate-limit — `tool_definition.json`
Execution Policy	logs, auditoria, retenção, budget — `run_policy.json`

9.5. Autenticação e autorização

Autenticação (quem é o usuário?) Métodos recomendados: OAuth2/OIDC, API Keys (sistemas), JWT assinado (usuários), mTLS (conexões internas).

Autorização (o que pode fazer?) Validação por scope e tenant. Antes de executar uma Tool valide: `session.user.roles`, `session.scopes` e `tool.security.required_scopes`.

Listing 32: Exemplo: validação de autorização (pseudo)

```
authorize(session, tool) {
  const allowed = tool.security.required_scopes.every(s => session.scopes.includes(s));
  if (!allowed) throw new AuthorizationError("Acesso negado à Tool.");
}
```

9.6. Isolamento e sandboxing

Scripts e automações devem executar em ambientes isolados para evitar interferência entre tenants.

Tabela 29: Técnicas de isolamento

Técnica	Propósito	Exemplo
Container (Docker)	Isolamento de execução	Limite de CPU/RAM
Network sandbox	Evitar saída não autorizada	Bloquear DNS externos
Storage ephemeral	Evitar persistência sensível	`/tmp/session_id` autolimpável
Vault secrets	Gestão de credenciais	HashiCorp Vault, AWS KMS
IAM por tenant	Segregar identidades	`vault/tenant_A/db/erp_read`

9.7. Logs cognitivos e auditoria

Cada decisão, ação ou exceção deve gerar trilhas cognitives auditáveis e imutáveis (WORM).

Listing 33: Exemplo: log cognitivo estruturado

```
{
  "trace_id": "trace_2025_11_09_0001",
  "run_id": "run_000124",
  "agent": "ag_financeiro",
  "task": "task_buscar_dados",
  "tool": "tool_erp_query",
  "event": "executed",
  "timestamp": "2025-11-09T15:30:42Z",
  "meta": {
    "user_id": "usr_001",
    "tenant": "cliente_A",
    "cost": 0.012,
    "latency_ms": 1043,
    "confidence": 0.93
  }
}
```

Tipos de log

- **Audit:** registro permanente para compliance.
- **Trace:** debug e rastreamento técnico.
- **Decision:** explicações do raciocínio do Agent.
- **Security:** acessos, falhas e violações de escopo.

9.8. Explainability — o "porquê" das decisões

Além do o quê, é necessário registrar o porquê. Exemplo:

Listing 34: Log explicativo de decisão

```
{
  "trace_id": "trace_2025_11_09_0001",
  "agent_decision": {
    "intent": "gerar_relatorio_vendas",
    "confidence": 0.91,
    "alternatives": [
      { "intent": "consultar_faturamento", "confidence": 0.73 }
    ],
    "chosen_agent": "ag_financeiro",
    "reason": "Maior confiança e histórico positivo de 98% de sucesso."
  }
}
```

Esse tipo de explicação é essencial para auditorias e conformidade (ex.: EU AI Act).

9.9. Governança de custo e reputação

Cada Agent/Tool opera sob limites definidos por política:

- se `cost_per_run > max_cost` → adiar ou solicitar aprovação;
- se `confidence < min_confidence` → solicitar confirmação humana;
- se `error_rate > threshold` → desabilitar temporariamente o Agent.

9.10. Proteção de dados e privacidade

Boas práticas para PII e dados sensíveis:

- Anonimização: substituir PII por tokens contextuais (`{CLIENTE_ID}`).
- Consent tracking: registrar consentimentos para uso de dados.
- PII masking: ocultar dados sensíveis nos logs.
- Ciclo de vida de dados: deleção automática de contextos expirados.
- Localização de dados: armazenar conforme jurisdição (LGPD / GDPR).

9.11. Conformidade com normas e legislações

Tabela 30: Exemplos de conformidade

Regulamento	Aplicação prática
LGPD (Brasil)	Consentimento e rastreabilidade de dados pessoais.
GDPR (UE)	Direito ao esquecimento e anonimização.
AI Act (UE)	Logs de decisão e mitigação de riscos.
ISO 27001	Gestão da segurança da informação.
SOC 2	Boas práticas de auditoria e controle.

9.12. Painel de auditoria (Governance Dashboard)

Módulos recomendados:

- Visão geral de execuções por tenant (success/fail/custo).
- Monitor de reputação de Agents e Tools.
- Rastreio por `trace_id` (trace view completo).
- Alertas de conformidade (tentativas de acesso indevido, custos anômalos).

Métricas sugeridas: `audit_events_total`, `security_incidents_total`, `compliance_alerts_open`, `avg_decision_confidence`.

Tabela 31: Checklist mínimo de segurança e auditoria

Item	Status
Autenticação (OAuth2, JWT, mTLS)	□
Autorização (RBAC + ABAC + Scopes)	□
Vault de segredos configurado por tenant	□
Logs estruturados: Trace + Audit + Decision	□
Explainability: logs de decisão armazenados	□
Sandbox seguro: containers com limites de recursos	□
Compliance (LGPD/GDPR/AI Act) compatível	□
Dashboard de métricas e alertas ativo	□

9.13. Checklist de Segurança e Auditoria

9.14. Sugestão visual

Um gráfico radial com três círculos concêntricos:

- Núcleo: Proteção (auth, sandbox, vault).
- Anel intermédio: Governança (políticas, custo, reputação).
- Anel externo: Auditoria (logs, explainability, compliance).

Setas entre anéis representam o ciclo: Proteger → Controlar → Explicar → Melhorar.

9.15. Conclusão

A segurança na orquestração cognitiva assegura rastreabilidade ética, técnica e legal de cada decisão da inteligência. Sem segurança não há confiança; sem confiança não há governança; sem governança não há negócio cognitivo sustentável.

Dica

No Capítulo 9 — Padrões de Design Cognitivo discutiremos padrões práticos como Chain of Thought, Human-in-the-Loop, Retry, Parallel Execution e Fallback Tree, com exemplos aplicáveis a automações corporativas.

10. Capítulo 9 — Padrões de Design Cognitivo

10.1. Objetivo do capítulo

Os Padrões de Design Cognitivo são blocos reutilizáveis de raciocínio e execução que permitem estruturar fluxos inteligentes de forma previsível, auditável e expansível. Assim como padrões de software (Factory, Observer), padrões cognitivos aplicam-se ao comportamento de Agents, Tasks e Tools.

10.2. Visão geral dos padrões

Tabela 32: Padrões cognitivos — finalidade e quando usar

Padrão	Finalidade	Quando usar
Chain of Thought (CoT)	Encadear raciocínios e subetapas.	Planejamento complexo e explicabilidade.
Human-in-the-Loop (HITL)	Inserir validação humana.	Ações críticas / baixa confiança.
Retry Pattern	Repetir automaticamente em falhas temporárias.	APIs instáveis, rede.
Parallel Execution	Executar tasks simultâneas.	Coleta de dados de múltiplas fontes.
Fallback Tree	Redundância controlada.	Alta disponibilidade / resiliência.

10.3. Chain of Thought (CoT)

Conceito Divide um problema complexo em etapas lógicas, encadeadas e explicáveis — cada etapa pode ser monitorada e logada.

Exemplo de fluxo

```
Input: "Gerar relatório de faturamento trimestral"
Step 1: Identificar período → trimestre atual
Step 2: Buscar dados no ERP
Step 3: Consolidar resultados
Step 4: Gerar PDF e enviar e-mail
```

Listing 35: JSON: padrão Chain of Thought

```
{
  "pattern": "chain_of_thought",
  "steps": [
    "entender_intencao",
    "buscar_dados",
    "gerar_relatorio",
```

```
    "enviar_resultado"
  ]
}
```

Benefícios Explicabilidade, depuração facilitada e reuso de raciocínio.

10.4. Human-in-the-Loop (HITL)

Conceito Permite que humano interrompa, aprove ou corrija o fluxo antes de prosseguir.

Aplicações Ações financeiras críticas, casos de baixa confiança (confidence < 0.7) ou exceções detectadas por políticas.

Listing 36: JSON: padrão HITL

```
{
  "pattern": "human_in_the_loop",
  "trigger": "confidence_below_threshold",
  "action": "request_approval",
  "parameters": {
    "approver_role": "manager",
    "timeout_minutes": 30
  }
}
```

Benefícios Segurança operacional, transparência e registro humano para auditoria.

10.5. Retry Pattern

Conceito Controle de tentativas automáticas em falhas temporárias (rede, instabilidade).

Listing 37: JSON: Retry Pattern

```
{
  "pattern": "retry",
  "max_retries": 3,
  "strategy": "exponential_backoff",
  "base_ms": 500,
  "max_backoff_ms": 8000
}
```

Pseudocódigo

Listing 38: Pseudocódigo: retry simplificado

```
for (let attempt = 1; attempt <= maxRetries; attempt++) {
  try { return await exec(); }
  catch (err) {
    if (attempt === maxRetries) throw err;
    await wait(baseMs * 2 ** (attempt - 1));
```

```
  }
}
```

Benefícios Robustez sem intervenção humana, redução de falhas transitórias.

10.6. Parallel Execution

Conceito Executar Tasks/Tools em paralelo para reduzir latência total.

Listing 39: JSON: Parallel Execution

```
{
  "pattern": "parallel_execution",
  "tasks": [
    { "id": "task_buscar_filial_a" },
    { "id": "task_buscar_filial_b" },
    { "id": "task_buscar_filial_c" }
  ],
  "sync_point": "task_gerar_relatorio"
}
```

Recomendação Sempre definir `sync_point` e limites de paralelismo (`max_parallel`) para evitar sobrecarga.

10.7. Fallback Tree

Conceito Plano hierárquico de alternativas em caso de falha — redundância cognitiva.

Listing 40: JSON: Fallback Tree

```
{
  "pattern": "fallback_tree",
  "primary": "tool_erp_query",
  "alternatives": [
    { "tool": "tool_cache_query", "condition": "timeout" },
    { "tool": "tool_notify_operator", "condition": "failure" }
  ]
}
```

Benefícios Alta disponibilidade e resiliência; evita bloqueios.

10.8. Composição de padrões

Padrões são compostos com frequência:

```
Chain of Thought (planejamento)
  → Parallel Execution (coleta)
    → Retry (tolerância)
```

```
      → Human-in-the-Loop (aprov. final)
        → Fallback Tree (resiliência)
```

Listing 41: JSON: composição de padrões

```
{
  "patterns": [
    "chain_of_thought",
    "parallel_execution",
    "retry",
    "human_in_the_loop",
    "fallback_tree"
  ]
}
```

10.9. Padrões e políticas de execução

Cada Task declara os padrões aplicáveis nas suas policies.

Listing 42: JSON: padrões na task policy

```
{
  "task_id": "task_atualizar_cliente",
  "patterns": ["retry", "fallback_tree"],
  "retry": { "max_retries": 2, "strategy": "exponential" },
  "fallback_tree": { "primary": "tool_api_write", "backup": "tool_cache_store" }
}
```

10.10. Métricas cognitivas por padrão

Tabela 33: Métricas operacionais por padrão

Métrica	Descrição
avg_retries_per_task	Média de tentativas por execução.
avg_parallel_tasks	Número médio de tasks em paralelo.
human_interventions_total	Quantidade total de intervenções humanas.
fallback_usage_rate	Taxa de uso de rotas alternativas (%).
avg_thought_depth	Profundidade média de raciocínio (nº de passos).

10.11. Checklist de implementação de padrões

10.12. Sugestão visual

Fluxo modular colorido representando os cinco padrões:

$$[\text{CoT}] \rightarrow [\text{Parallel}] \rightarrow [\text{Retry}] \rightarrow [\text{HITL}] \rightarrow [\text{Fallback}]$$

Use ícones distintos (□, □, □, □, □) para cada módulo nos diagramas didáticos.

Tabela 34: Checklist de padrões cognitivos

Item	Status
Chain of Thought implementada	☐
HITL configurado (aprov. humana)	☐
Retry configurado por política	☐
Parallel Execution validada	☐
Fallback Tree definido	☐
Métricas cognitivas exportadas	☐

10.13. Conclusão

Padrões cognitivos são a linguagem arquitetural da inteligência aplicada. Ao padronizar CoT, HITL, Retry, Parallel Execution e Fallback Tree, transformamos fluxos automatizados em sistemas cognitivos previsíveis, auditáveis e governáveis.

Dica

No Capítulo 10 — Exemplo Prático: Orquestração ERP Inteligente veremos um caso completo integrando Agents, Tasks, Tools, Patterns e Policies, com fluxograma, pseudocódigo e JSON funcional.

11. Capítulo 10 — Exemplo Prático: Orquestração ERP Inteligente

11.1. Objetivo do capítulo

Este capítulo demonstra um caso completo de orquestração cognitiva integrando todos os componentes da arquitetura: **Agents → Tasks → Tools**, com uso de padrões cognitivos (Chain of Thought, Retry, Fallback e HITL). O objetivo é mostrar como transformar um processo empresarial tradicional — geração de relatório financeiro — em um fluxo cognitivo auditável e automatizado.

11.2. Contexto do caso de uso

Solicitação:

> "Envie o relatório de vendas da matriz para o setor financeiro."

Objetivo corporativo: automatizar a geração e o envio de relatórios financeiros mensais do ERP, com validação de parâmetros, geração de PDF e notificação final ao usuário.

11.3. Estrutura do fluxo cognitivo

Tabela 35: Camadas e elementos do fluxo

Camada	Descrição / Elementos
Agent	`ag_relatorio_financeiro` — interpreta intenção e cria plano.
Tasks	`task_validar_filtros`, `task_buscar_dados`, `task_gerar_pdf`, `task_enviar_email`.
Tools	`tool_nlp_validator`, `tool_erp_query`, `tool_pdf_builder`, `tool_mailer`.
Patterns	CoT, Retry, Fallback, HITL — garantem resiliência e explicabilidade.

11.4. Fluxo lógico (texto-diagrama)

```
[Usuário] "Envie o relatório de vendas da matriz"
     ↓
[Agent: ag_relatorio_financeiro]
     ↓
Identifica intenção → "gerar_relatorio_vendas"
     ↓
[Task: validar_filtros]
       Tool: tool_nlp_validator
     ↓
[Task: buscar_dados]
       Tool: tool_erp_query (Retry x3, Fallback: tool_cache_query)
     ↓
[Task: gerar_pdf]
       Tool: tool_pdf_builder (timeout 60s)
     ↓
```

```
[Task: enviar_email]
       Tool: tool_mailer
     ↓
[Orchestrator]
       Agrega resultado e audita execução
```

11.5. Plano cognitivo (JSON completo)

Listing 43: Plano cognitivo completo do fluxo ERP

```
{
  "flow_id": "flow_relatorio_vendas_v1",
  "agent": "ag_relatorio_financeiro",
  "intent": "gerar_relatorio_vendas",
  "context": { "tenant": "cliente_A", "mes": "outubro" },
  "patterns": ["chain_of_thought", "retry", "fallback_tree"],
  "tasks": [
    {
      "id": "task_validar_filtros",
      "tool": "tool_nlp_validator",
      "fallback": "ask_user",
      "policy": { "timeout_ms": 5000 }
    },
    {
      "id": "task_buscar_dados",
      "tool": "tool_erp_query",
      "policy": {
        "retries": 3,
        "backoff": { "type": "exponential", "base_ms": 500 },
        "timeout_ms": 15000
      },
      "fallback_tree": {
        "primary": "tool_erp_query",
        "alternatives": [{ "tool": "tool_cache_query", "condition": "timeout" }]
      }
    },
    {
      "id": "task_gerar_pdf",
      "tool": "tool_pdf_builder",
      "policy": { "timeout_ms": 60000, "idempotent": true }
    },
    {
      "id": "task_enviar_email",
      "tool": "tool_mailer",
      "policy": { "retries": 1, "timeout_ms": 8000 }
    }
  ]
}
```

11.6. Pseudocódigo completo do fluxo

Listing 44: Execução orquestrada do fluxo ERP

```
async function fluxoRelatorioFinanceiro(trigger) {
  const trace = startTrace("flow_relatorio_vendas_v1", trigger);

  const filters = await execToolWithPolicy("tool_nlp_validator", trigger);
  if (!filters.valid) return askUser("Faltam parâmetros obrigatórios.");

  let dados;
  try {
    dados = await execToolWithPolicy("tool_erp_query", filters);
  } catch (err) {
    log.warn(trace, "ERP indisponível, tentando cache...");
    dados = await execToolWithPolicy("tool_cache_query", filters);
  }

  const pdf = await execToolWithPolicy("tool_pdf_builder", {
    template: "relatorio_vendas",
    payload: dados
  });

  const envio = await execToolWithPolicy("tool_mailer", {
    to: "financeiro@empresa.com",
    subject: "Relatório de Vendas - Outubro",
    attachment: pdf.pdf_url
  });

  logAudit(trace, { status: "success", email: envio.to });
  return "Relatório enviado com sucesso!";
}
```

11.7. Aplicação de padrões cognitivos no exemplo

Tabela 36: Aplicação prática dos padrões cognitivos

Padrão	Aplicação no caso	Resultado
Chain of Thought	Definição sequencial das 4 tasks.	Clareza e explicabilidade.
Retry	Tool ERP tenta até 3x.	Robustez contra falhas temporárias.
Fallback Tree	Redireciona para cache se ERP falhar.	Continuidade garantida.
HITL	Solicita parâmetros se NLP falhar.	Interação humana segura.

Tabela 37: Resultados operacionais esperados

Métrica	Valor médio	Fonte
Tempo médio de execução	4.8s	Execution Engine
Custo médio	0.021 créditos	Agent Policy
Taxa de sucesso	99.2%	Audit Logs
Fallback ativado	6%	Tool Policy
Intervenções humanas	0.4%	HITL Tracker

11.8. Métricas e resultados esperados

11.9. Governança e rastreabilidade

Cada execução gera:

- `trace_id` único e `run_id` auditável;
- logs detalhados (Tool → Task → Agent);
- decisão explicável (por que o Agent foi escolhido);
- auditoria imutável (`run_status`: success / fail / fallback / hitl).

Esses registros suportam auditorias internas, relatórios de conformidade e análise cognitiva.

11.10. Trecho de log real (audit.json)

Listing 45: Exemplo de log auditável

```
{
  "trace_id": "trace_2025_11_09_0012",
  "run_id": "run_674be9",
  "agent": "ag_relatorio_financeiro",
  "status": "success",
  "steps": [
    { "task": "task_validar_filtros", "result": "ok" },
    { "task": "task_buscar_dados", "tool": "tool_erp_query", "retries": 1 },
    { "task": "task_gerar_pdf", "latency_ms": 1084 },
    { "task": "task_enviar_email", "latency_ms": 432 }
  ],
  "cost_total": 0.021,
  "timestamp": "2025-11-09T15:40:12Z"
}
```

11.11. Checklist do fluxo ERP cognitivo

11.12. Sugestão visual

Um diagrama linear animado:

 Usuário → Agent → Task1 → Task2 → Task3 → Task4 → Resultado + Log

Tabela 38: Checklist de implantação do fluxo ERP

Item	Status
Agent definido (`ag_relatorio_financeiro`)	☐
Tasks documentadas e versionadas	☐
Tools integradas (ERP, PDF, Mailer)	☐
Policies aplicadas (Retry, Timeout, Fallback)	☐
Logs e auditoria ativos (trace/run)	☐
Painel de métricas (Prometheus/Grafana)	☐

Cada bloco colorido representa um nível (Agent, Task, Tool) com setas de raciocínio (CoT) e fallback lateral opcional.

11.13. Conclusão

O caso de orquestração ERP demonstra como unir inteligência, automação e governança em um fluxo modular, seguro e replicável. Cada fluxo cognitivo é, em essência, um produto rastreável, vendável e evolutivo.

Dica

No Capítulo 11 — Checklist de Implantação veremos como validar todos os componentes antes do deploy, incluindo arquitetura, segurança, documentação e testes.

12. Capítulo 11 — Checklist de Implantação

12.1. Objetivo do capítulo

Fornecer um checklist completo e validável para garantir que um fluxo cognitivo (Agents → Tasks → Tools) esteja pronto para entrar em produção de forma segura, governável e sustentável. Este capítulo é a ponte entre design e operação — o momento em que a inteligência passa a gerar valor real para a organização.

12.2. Estrutura geral da implantação

O processo envolve quatro dimensões principais:

Tabela 39: Dimensões da implantação

Dimensão	Foco	Resultado esperado
Técnica	Infraestrutura e execução	Ambiente estável e escalável
Governança	Políticas, versionamento, papéis	Controle e rastreabilidade
Segurança	Acesso, isolamento e logs	Execução protegida
Comercial	Licenciamento e catálogo	Monetização e distribuição

12.3. Checklist técnico de implantação

Tabela 40: Itens técnicos essenciais

Etapa	Descrição	Ferramentas recomendadas
1. Infraestrutura base	Ambiente de execução (containers, filas, banco, API Gateway).	Docker / K8s / NATS / PostgreSQL
2. Orchestrator ativo	Endpoint `/trigger` documentado.	Node.js / Go / FastAPI
3. Engine de execução	Workers distribuídos rodando Tasks/Tools.	Redis Queue / Celery / RabbitMQ
4. Catálogo cognitivo	Repositório com Agents/Tasks/Tools versionados.	JSON/DB + UI web
5. Adapters	HTTP, LLM, Script, SQL implementados.	Axios / OpenAI SDK / Docker SDK
6. Context Store	Memória curta e longa configuradas.	Redis / MongoDB
7. Vault de segredos	Credenciais segregadas por tenant.	HashiCorp Vault / AWS KMS
8. Observabilidade	Logs estruturados e traces ativos.	Prometheus / Grafana / OTel
9. Testes de carga e falha	Simulação de 1000+ execuções paralelas.	k6 / Locust / ChaosMonkey

Tabela 41: Validação do design cognitivo

Item	Validação	Resultado esperado
Agents documentados	Cada Agent possui ficha técnica completa.	`agent_id`, intenções, políticas registradas.
Tasks associadas	Cada Task pertence a um Agent e tem plano definido.	Fluxo CoT mapeado.
Tools disponíveis	Tools declaradas com schema e política.	JSON validado por AJV.
Policies aplicadas	Timeouts, retries, fallbacks definidos.	Tolerância a falhas garantida.
Fluxo testado	Execução completa validada.	Happy path e erros simulados.
Explainability ativa	Logs cognitivos gerados.	Cada decisão explicável por `trace_id`.

12.4. Checklist cognitivo e de fluxo

12.5. Checklist de segurança

Tabela 42: Validações de segurança

Categoria	Validação	Requisito
Autenticação	OAuth2 e JWT implementados.	Tokens verificados em cada chamada.
Autorização	RBAC e ABAC ativos.	Acesso por escopos e atributos.
Segredos	Vault por tenant.	Sem credenciais hardcoded.
Sandbox	Execução isolada.	Containers com limite CPU/RAM.
Logs de auditoria	Estruturados e imutáveis.	Logs com `trace_id` + `run_id`.
Compliance	LGPD/GDPR compatível.	Consentimento e PII masking ativos.

12.6. Checklist de governança

12.7. Checklist comercial e de licenciamento

12.8. Testes obrigatórios antes do deploy

12.9. Métricas de saúde operacional

12.10. Template de validação final (pré-deploy)

Listing 46: Template de validação final (pré-deploy)

```
{
  "deployment_id": "deploy_2025_11_09_01",
  "flow": "flow_relatorio_vendas_v1",
  "validation": {
    "technical": true,
```

Tabela 43: Validações de governança

Categoria	Validação	Resultado esperado
Versionamento	Agents/Tasks/Tools versionados (vX.Y.Z).	Histórico e rollback disponíveis.
Changelog	Últimas alterações registradas.	Transparência de evolução.
Controle de custos	Limites e cotas por tenant.	Evita abuso de recursos.
Reputação e score	Índice de confiabilidade calculado.	Incentivo à qualidade.
Auditoria completa	Relatórios exportáveis.	Conformidade legal e ética.

Tabela 44: Itens comerciais

Item	Descrição	Exemplo
Ficha Técnica de Agente	Documento padrão anexado ao catálogo.	`ag_financeiro_v1.2.json`
Template de proposta	Documento editável para clientes.	”Relatório Financeiro Cognitivo”
Planilha de precificação	Fórmula de custo + margem.	`pricing_tools.xlsx`
Contrato de licenciamento	Acordo de uso / revenda.	WhiteLabel / SaaS / SDK
Catálogo cognitivo interno	Listagem pública interna.	UI Web / API `/catalog`


```
    "security": true,
    "governance": true,
    "compliance": true,
    "commercial_ready": true
  },
  "signoff": {
    "tech_lead": "approved",
    "security_officer": "approved",
    "data_privacy": "approved",
    "operations": "approved"
  },
  "timestamp": "2025-11-09T16:00:00Z"
}
```


Observação: este documento deve ser assinado digitalmente e armazenado junto ao catálogo de deploys para rastreabilidade.

12.11. Checklist resumido (para campo)

12.12. Sugestão visual

Um painel Kanban com quatro colunas:

 Técnica Segurança Governança Comercial

Cada cartão representa um item do checklist com estados: Concluído, Em validação, Pendente.

Tabela 45: Suite mínima de testes

Tipo de teste	Descrição	Resultado esperado
Unitário	Cada Tool isoladamente.	Saída conforme schema.
Integração	Task completa com 2+ Tools.	Dados coerentes e válidos.
Carga (Stress)	Execuções simultâneas.	Estabilidade ≥ 95%.
Chaos Test	Falhas simuladas em Tools.	Recuperação via Fallback.
HITL Test	Intervenção humana simulada.	Aprovação dentro do SLA.
Compliance Test	Logs e anonimização.	100% compatível com LGPD.

Tabela 46: SLA e metas operacionais

Métrica	Descrição	Alvo
uptime_24h	Disponibilidade da orquestração.	$\geq$ 99,5%
avg_task_latency	Tempo médio por task.	$\leq$ 2s
tool_failure_rate	Falhas por Tool.	$\leq$ 1%
human_interventions	Casos de HITL.	$\leq$ 3%
trace_completeness	Logs completos por run.	100%
audit_export_success	Exportações de logs válidas.	$\geq$ 99%

12.13. Conclusão

O checklist de implantação consolida todos os pilares do design cognitivo — transformando o modelo em um sistema confiável, auditável e pronto para escala. Um fluxo cognitivo só é completo quando está documentado, testado e licenciado.

Dica

No Capítulo 12 — Apêndice: Template da Ficha Técnica de Agente apresentaremos o modelo oficial de documentação de cada Agent, incluindo campos técnicos, parâmetros, dependências e políticas — essencial para governança e licenciamento.

Tabela 47: Resumo rápido de verificação

Área	Itens-chave
Técnica	Orchestrator, Engine, Adapters, Logs
Segurança	Vault, RBAC, Sandbox, Logs
Governança	Policies, Versionamento, Métricas
Comercial	Licença, Pricing, Catálogo
Testes	Unitários, Integração, Carga, Compliance

13. Capítulo 12 — Apêndice: Template da Ficha Técnica de Agente

13.1. Objetivo do capítulo

A Ficha Técnica de Agente é o documento padronizado que descreve comportamento, estrutura e políticas de cada Agent na plataforma cognitiva — o contrato técnico entre desenvolvedores, auditores e clientes. Permite rastreabilidade (versão, autor, data), documentação padronizada, auditoria e base para licenciamento.

13.2. Estrutura geral da ficha

Tabela 48: Seções essenciais da Ficha Técnica

Seção	Descrição	Obrigatório
Identificação	Nome, ID e versão do agente.	□
Descrição funcional	O que o agente faz e seu propósito.	□
Intenções reconhecidas	Frases e contextos que entende.	□
Tasks associadas	Lista de tarefas e Tools usadas.	□
Políticas operacionais	Regras de execução, custo, paralelismo.	□
Parâmetros	Entradas e saídas esperadas.	□
Segurança e compliance	Escopos, limites e logs.	□
Histórico e autor	Data, versão, responsável.	□

13.3. Template oficial (JSON técnico)

Listing 47: Template JSON da Ficha Técnica do Agente

```
{
  "agent_id": "ag_relatorio_financeiro",
  "name": "Agente de Relatório Financeiro",
  "version": "1.3.0",
  "description": "Gera relatórios financeiros mensais integrando dados do ERP e envia
    automaticamente por e-mail.",
  "intent_examples": [
    "Gerar relatório de vendas",
    "Enviar relatório de faturamento mensal",
    "Criar relatório financeiro do último trimestre"
  ],
  "tasks": [
```

```
    {
      "id": "task_validar_filtros",
      "description": "Valida parâmetros de entrada e corrige via NLP.",
      "tool": "tool_nlp_validator"
    },
    {
      "id": "task_buscar_dados",
      "description": "Consulta o ERP e retorna dados de vendas.",
      "tool": "tool_erp_query"
    },
    {
      "id": "task_gerar_pdf",
      "description": "Gera o arquivo PDF do relatório.",
      "tool": "tool_pdf_builder"
    },
    {
      "id": "task_enviar_email",
      "description": "Envia o relatório por e-mail ao destinatário configurado.",
      "tool": "tool_mailer"
    }
  ],
  "patterns": ["chain_of_thought", "retry", "fallback_tree"],
  "policies": {
    "max_parallel_tasks": 3,
    "timeout_ms": 60000,
    "retry_policy": { "max_retries": 2, "strategy": "exponential_backoff" },
    "cost_limit_per_run": 0.05
  },
  "parameters": {
    "input": ["mês", "empresa", "email_destino"],
    "output": ["pdf_url", "status_envio"]
  },
  "security": {
    "required_scopes": ["financeiro:read", "financeiro:report"],
    "tenant_restricted": true,
    "audit_level": "full"
  },
  "governance": {
    "owner": "Equipe de Automação Cognitiva",
    "last_review": "2025-11-09",
    "approved_by": "CTO",
    "license_type": "Enterprise SaaS",
    "pricing_model": "por execução"
  },
  "metrics": {
    "avg_runtime": 4.8,
    "success_rate": 0.992,
    "fallback_rate": 0.06,
```

```
    "human_intervention_rate": 0.004
  },
  "changelog": [
    {
      "version": "1.3.0",
      "date": "2025-11-09",
      "changes": [
        "Adicionada integração com cache de ERP",
        "Melhorado fallback e logs de decisão"
      ]
    }
  ]
}
```

13.4. Template visual (Ficha para PDF/LaTeX)

```
FICHA TÉCNICA DO AGENTE COGNITIVO

Identificação
ID: ag_relatorio_financeiro
Nome: Agente de Relatório Financeiro
Versão: 1.3.0
Autor: Equipe de Automação Cognitiva
Última revisão: 09/11/2025

Descrição
Gera relatórios financeiros mensais integrando dados do ERP
e envia automaticamente por e-mail ao setor financeiro.

Intenções reconhecidas
- "Gerar relatório de vendas"
- "Enviar relatório de faturamento"
- "Relatório financeiro do último trimestre"

Tasks associadas
1. Validar Filtros → tool_nlp_validator
2. Buscar Dados → tool_erp_query
3. Gerar PDF → tool_pdf_builder
4. Enviar Email → tool_mailer

Políticas
- Execuções paralelas: 3
- Timeout padrão: 60s
```

```
  - Retries: 2 (exponential backoff)
  - Limite de custo: 0.05 créditos por run

  Segurança e Compliance
  - Scopes: financeiro:read, financeiro:report
  - Restrição por tenant: Ativa
  - Auditoria: Nível completo
  - Padrões: LGPD / AI Act / ISO 27001

  Métricas
  - Tempo médio: 4.8s
  - Taxa de sucesso: 99.2%
  - Taxa de fallback: 6%
  - Intervenções humanas: 0.4%

  Histórico
  - v1.3.0 (09/11/2025)
    → Adicionada integração com cache ERP
    → Melhoria nos logs cognitivos

  Licenciamento
  Tipo: Enterprise SaaS
  Modelo: Por execução
```

13.5. Integração da ficha no Catálogo Cognitivo

A ficha deve ser registrada no Catálogo Cognitivo e exposta via API:

Listing 48: Exemplos de endpoints do catálogo

```
GET /catalog/agent/:id   -> retorna ficha completa
GET /catalog/list        -> lista agentes (id + versão)
```

Resposta de exemplo (resumo):

Listing 49: Exemplo: resposta /catalog/list

```
{
  "agents": [
    { "id": "ag_relatorio_financeiro", "version": "1.3.0" },
    { "id": "ag_rh_recrutamento", "version": "1.0.5" },
    { "id": "ag_marketing_analytics", "version": "2.1.1" }
  ]
}
```

Tabela 49: Itens mínimos da ficha técnica

Item	Descrição	Status
Identificação completa	ID, nome, versão, autor	☐
Intenções mapeadas	Lista de frases e contextos	☐
Tasks e Tools listadas	Todas documentadas	☐
Políticas definidas	Timeout, retry, fallback	☐
Segurança aplicada	Scopes e nível de auditoria	☐
Governança	Dono, versão, licença	☐
Métricas registradas	Latência, sucesso, custo	☐
Histórico atualizado	Última versão documentada	☐

13.6. Checklist da Ficha Técnica

13.7. Sugestão visual

Um gráfico circular em camadas:

- Identificação (núcleo)
- Estrutura (Tasks & Tools)
- Policies (execução)
- Segurança (scopes & vault)
- Métricas (observability)
- Governança (licença & owner)

Cada anel representa maturidade do agente: do técnico ao comercial.

13.8. Conclusão

A Ficha Técnica é o DNA do agente cognitivo: documentação, governança e base comercial. Automatize a geração dessa ficha a partir do JSON oficial para garantir consistência entre catálogo, UI e contratos.

Referências e Recursos

- Documentação interna CENC.Ai — Playbooks e templates.
- Materiais de arquitetura: OpenTelemetry, Prometheus, HashiCorp Vault.
- Referências legais: LGPD, GDPR, AI Act (UE).

Licença e Direitos

Versão: Public Release 1.0 — Novembro/2025

Continue sua jornada prática com o curso completo **Plataforma Cognitiva** — disponível em ai.cenc.com.br/curso/cimaas?r=pt