Shopware AI Agent

Autonomous Plugin Development with LLMs

SCUC 2026 · @jankal · nuonic Digital

🧠 Why?

- 🔁 Plugin development is **repetitive**
  - boilerplate: `composer.json`, Plugin class, `services.xml`...
  - same patterns over and over
- 🔀 Context switching is **costly**
  - docs, Admin API, CLI, code, browser
- 🤖 LLMs are great at code generation
  - but they lack **execution context**

🧠 The Gap

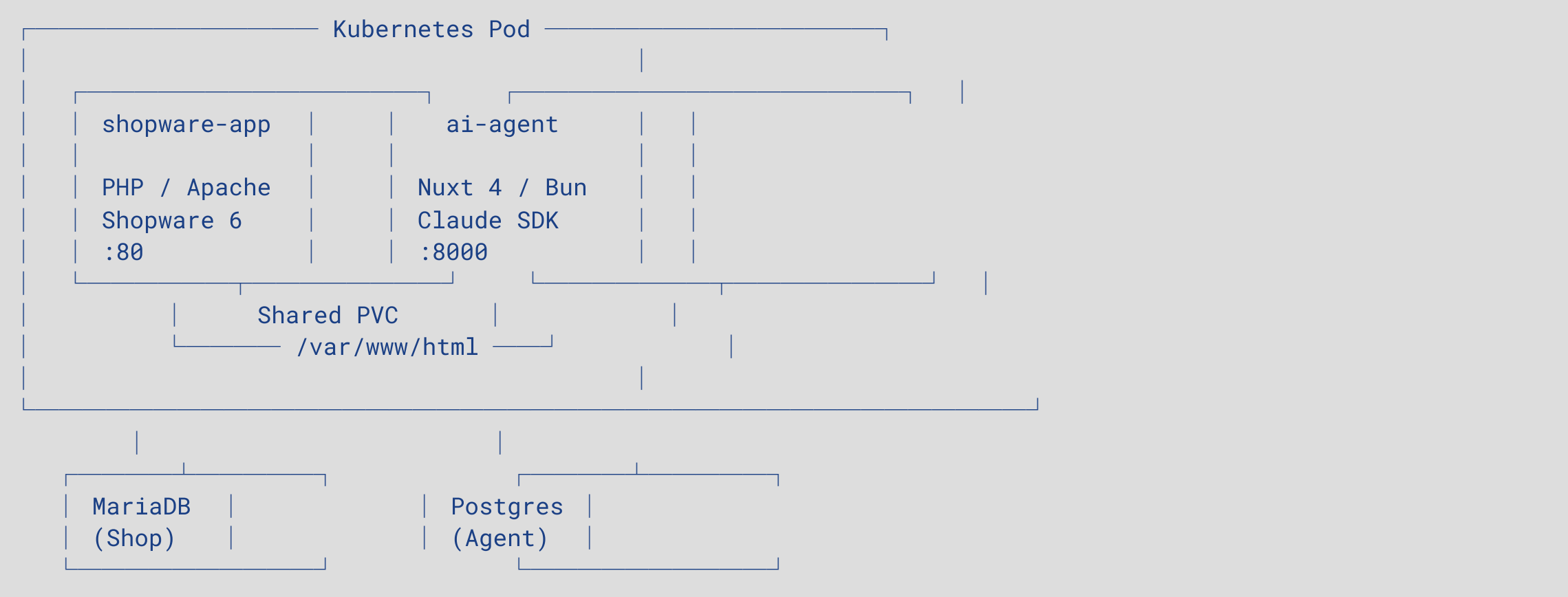
An LLM that can only *generate* code is a **fancy autocomplete**.

An LLM that can **write**, **execute**, **validate**, and **see** the result is an **autonomous agent**.

→ We gave Claude a Shopware instance to play with.

 What?

Architecture: Sidecar Pattern



Why a Sidecar?

- Shared Volume = **zero-latency** file I/O
  - Agent writes PHP → instantly visible to Shopware
- `ReadWriteOnce` PVC is sufficient (same Pod)
  - no expensive RWX network storage needed
- **K8s Exec API** for command execution
  - no SSH, no network hops
- Independent lifecycle
  - Shopware crashes? Agent still runs.

Tech Stack

Component	Technology
Agent Runtime	**Nuxt 4** + **Bun**
LLM	Claude Opus 4 (Anthropic SDK)
Tool Bridge	**MCP** (Model Context Protocol)
K8s Integration	`@kubernetes/client-node`
Conversations	PostgreSQL + **Drizzle ORM**
Validation	`shopware-cli` (PHPStan, CS)
Screenshots	**Playwright** (Chromium)
Frontend	Nuxt UI + Tailwind CSS

⚙ How?

Native Tools

The agent has **6 built-in tools** for direct Shopware interaction:

Tool	What it does
`write_file`	Write to shared volume (with diff)
`read_file`	Read from shared volume
`list_directory`	Browse the filesystem
`search_files`	Glob + content search
`exec_command`	Shell commands via K8s Exec API
`validate_extension`	PHPStan, code style via `shopware-cli`

K8s Exec API

```
// k8s-exec.ts (simplified)
export async function execInContainer(command: string) {
  const kc = new k8s.KubeConfig()
  kc.loadFromCluster()

  const exec = new k8s.Exec(kc)

  await exec.exec(
    NAMESPACE,            // from Downward API
    POD_NAME,             // from Downward API
    'shopware-app',       // target container
    ['/bin/sh', '-c',
     `cd /var/www/html && su -s /bin/sh www-data -c '${command}'`],
    stdout, stderr,
    null,                 // no stdin
    false,                // no tty
    (status) => resolve(status)
  )
}
```

No SSH. No network hops. Just the **Kubernetes API**.

MCP Server Tools

Three **MCP servers** extend the agent's capabilities:

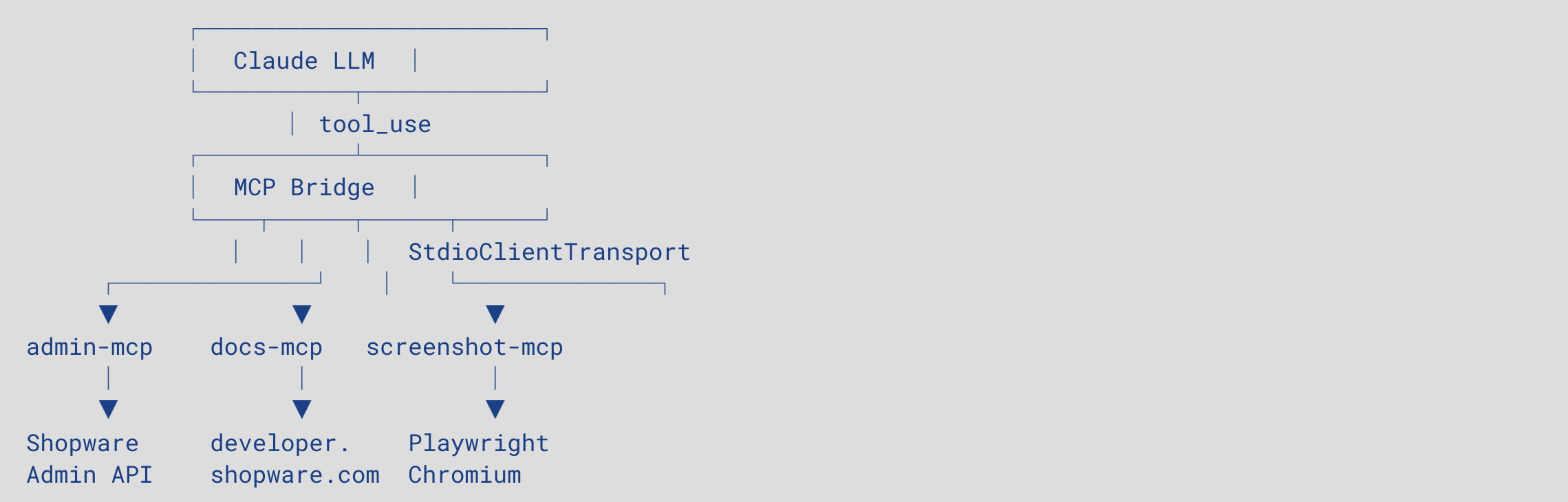
admin-mcp (`@shopware-ag/admin-mcp`)
→ Products, categories, orders, themes, media via Admin API

docs-mcp (custom, Algolia)
→ Search developer.shopware.com documentation

screenshot-mcp (custom, Playwright)
→ Take screenshots, click elements, fill forms

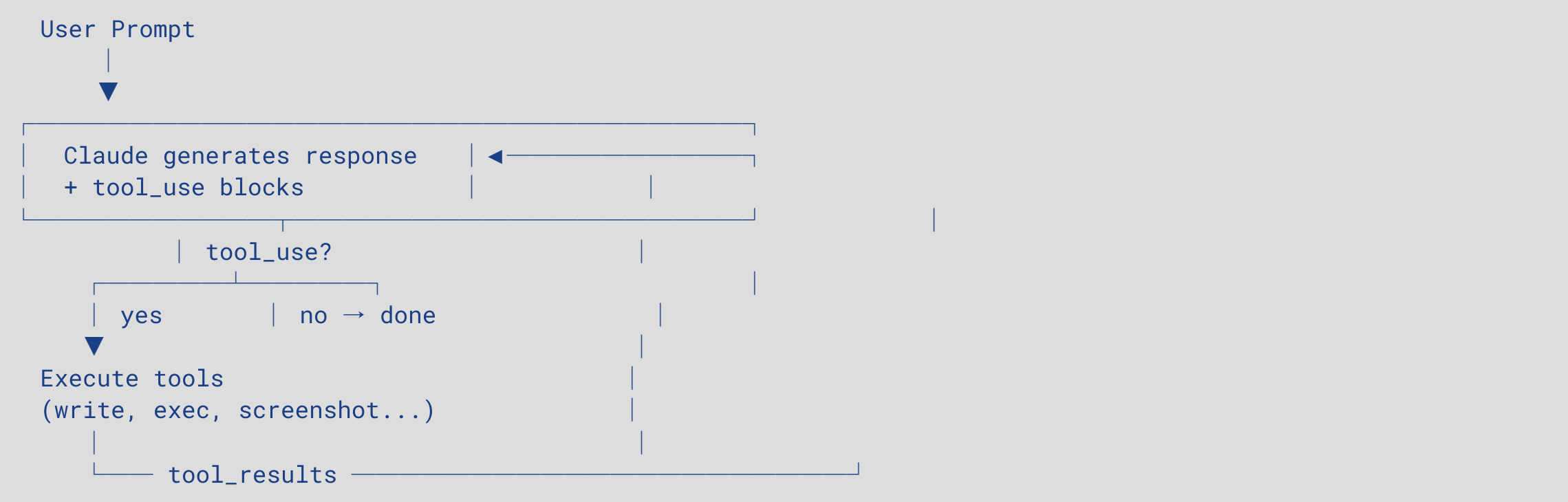
MCP = Model Context Protocol — open standard for LLM tool integration

MCP Bridge



Tools are **discovered dynamically** — the bridge merges all MCP tools into the LLM tool palette at runtime.

The Agentic Loop



Up to **25 rounds** per conversation turn.

SSE streaming for real-time UI updates.

Tool Tracking & Revert

Every `write_file` is tracked with full diffs:

```
// dispatchTool (simplified)
case 'write_file': {
  const result = await writeFileOp(input.path, input.content)
  // result.diff → unified diff
  // result.previousContent → for revert

  await db.insert(toolActions).values({
    conversationId,
    toolName: 'write_file',
    diff: result.diff,
    previousContent: result.previousContent,
    isReversible: true,
  })
}
```

→ Users can **revert any file write** from the UI.

System Prompt

The agent has a detailed **German system prompt** with:

- ✅ Shopware plugin directory structure
- ✅ Required `composer.json` fields
- ✅ CLI command reference (`plugin:refresh`, `cache:clear`, …)
- ✅ Best practices (`strict_types`, PSR-4, namespaces)
- ✅ Admin API tool documentation
- ✅ Validation workflow (validate → fix → repeat)
- ✅ Screenshot workflow (compile → screenshot → show)

Demo

"Create a plugin that adds a custom CMS element"

What Happens Behind the Scenes

1. 💬 User sends prompt via SSE stream
2. 🧠 Claude plans the plugin structure
3. 📝 `write_file` × 5–8 calls

   composer.json, Plugin.php, services.xml, CmsElement, …
4. ⚡ `exec_command` : `plugin:refresh && plugin:install`
5. ⚡ `exec_command` : `plugin:activate && cache:clear`
6. ✅ `validate_extension` : PHPStan + code style
7. 🔧 Auto-fix any validation errors
8. 📸 `screenshot` : verify result in storefront

Embedded in Shopware Admin

The agent UI runs as a **Shopware App** (not a plugin):

- `manifest.xml` with `<admin><module>`
- **Zero PHP code** — purely declarative
- iframe in Shopware Admin under Settings
- `postMessage('sw-app-loaded')` handshake
- Works even if Shopware is broken
  (agent runs in a separate container)

Security

Filesystem

- Path traversal prevention (`safePath()`)
- All paths relative to `/var/www/html`

Kubernetes

- Namespace-scoped RBAC (ServiceAccount)
- Only `pods/exec` permission in own namespace
- Resource limits (500m CPU, 512Mi RAM)

Agent

Lessons Learned

- 🎯 **Context is everything**
  Tools > more tokens. Let the LLM *do*, not just *say*.
- 🔄 **Validation loops are essential**
  LLMs make mistakes. Automated validation catches them.
- 📸 **Visual feedback closes the loop**
  Screenshots let the LLM self-correct UI issues.
- 🏗️ **MCP is the right abstraction**
  Swappable tool servers, standardized protocol.
- ⚡ **Sidecar > microservice**
  Shared volumes beat API calls for file I/O.

What's Next?

- 🌐 Multi-tenant SaaS (scale-to-zero with KEDA)
- 🔐 Auth between Shopware ↔️ Agent
- 📦 Git export to production repos
- 🧪 Automated test generation
- 🔌 More MCP servers (GitHub, CI/CD, monitoring)

Questions?

@jankal · nuonic Digital · SCUC 2026