🚀 CopadoCON Bangalore — June 2026

Hackathon: Copado Headless

The Future of Salesforce DevOps Has No Browser Tab

🎯 Vision

> "What if a developer never had to open Copado's UI again?"

The Copado Headless Hackathon is a focused, multi-track challenge built around a single conviction: the best developer experience is the one that lives where developers already are — inside their terminal, their IDE, and their AI-powered coding environment.

Copado's full suite — CI/CD (Agentia Pro), Robotic Testing (Agentia Testing), and AI (Agentia AI Context Hub with its 5 specialist agents—the Orchestrate Agent is out of scope for this hackathon) — exposes rich, programmable APIs. This hackathon asks: what happens when you wrap all of that into a single, composable CLI that any developer or any AI agent can drive?

The Core Challenge Statement

Participants are challenged to build an open-source utility, extension, framework, or agentic workflow that enables Salesforce developers and release engineers to execute end-to-end Copado DevOps operations completely headless.

The ultimate goal is to eliminate the friction of context-switching to a browser-based UI. Every operation—from pulling user stories and committing metadata to running robotic tests and promoting to production—should happen natively where developers already live (e.g., the terminal, IDEs, git configurations, or autonomous AI streams).

Three Ways to Compete

The headless vision is intentionally open-ended. We've defined two reference tracks to give teams a concrete starting point — but any approach that achieves the headless goal is a valid submission.

Track	The Idea	Builds On

🔵 **Track A — The Headless CLI**	Build copado-hx, a unified terminal CLI wrapping the full Copado API surface.	—
🟣 **Track B — The Agentic Orchestrator**	Extend Track A with SKILL.md so any AI agent can drive copado-hx autonomously.	Track A
🟠 **Track C — Your Headless Idea**	Build anything headless: a VS Code extension, Git hooks, a Slack bot, a Raycast extension, a native MCP server — if it eliminates the browser tab using the Copado API surface, it qualifies.	Independent

Track A and Track B are inspiration, not constraints. Teams are free to interpret the headless challenge in any direction. Track C submissions will be judged on the same criteria as A and B: working demo, open-source code, and measurable reduction of UI dependency.

🔵 Track A — copado-hx: The Headless Developer CLI

The Concept: Focuses on creating an exceptional local developer experience (DX) via direct terminal execution or IDE integration.

Build copado-hx — a unified, open-source command-line interface that wraps the entire Copado API surface (CI/CD + CRT + AI) into a single, ergonomic tool. A pro developer working in VS Code or Cursor should be able to manage their entire Salesforce DevOps lifecycle — from user story to production — without ever logging into Copado.

This is the Copado Headless Developer Experience.

How copado-hx may look like? Here are some inspirational examples:

🔐 **Authentication**

Bash

```
None
copado-hx auth login                          # OAuth flow — opens
browser once, stores token
copado-hx auth login --token <api-token>      # Token-based auth for
CI environments
```

```
copado-hx auth status                          # Show current
authenticated org and user
copado-hx auth logout
```

📋 User Story Management (Copado CI/CD API)

Bash

```
None
copado-hx story list                           # List user stories
assigned to me
copado-hx story list --pipeline <id> --status "In Progress"
copado-hx story set --id US-1234               # Set working context
(like git checkout)
copado-hx story show                           # Show current user
story details + metadata scope
copado-hx story create --title "Add lead scoring" --pipeline <id>
```

⚙️ CI/CD Pipeline Operations (Copado Actions REST API)

Bash

```
None
copado-hx commit                               # Commit changes from
current user story context
copado-hx commit --message "feat: lead scoring logic" --us US-1234
copado-hx promote --env UAT                    # Promote current user
story to next environment
copado-hx promote --us US-1234 --env UAT --validate   # Promote +
run validation only
copado-hx deploy --env PROD                    # Deploy (requires
approval gate confirmation)
copado-hx status                               # Show pipeline status:
promotions, deployments, quality gates
copado-hx status --watch                       # Live-polling terminal
dashboard
```

🧪 Test Execution (CRT Open API)

Bash

```
None
copado-hx test list                            # List available test
suites and jobs
```

```
copado-hx test run --suite <suite-id>       # Note: suite resolves
to a jobId in the CRT API
copado-hx test run --job <job-id>           # Trigger a specific
CRT test job
copado-hx test status --execution <exec-id> # Poll execution status
copado-hx test results --execution <exec-id> # Retrieve results
(JUnit-compatible output)
copado-hx test results --execution <exec-id> --format json
copado-hx test results --execution <exec-id> --format pdf
```

> *Note: `--suite` is a convenience alias — in the CRT API, both suites and individual jobs are addressed by a `jobId`. Your team's hackathon credentials will include the relevant job IDs.*

🤖 AI Agent Conversations (Copado AI Platform API)

Bash

```
None
# Invoke any of the 5 specialist agents directly from the terminal
copado-hx ai ask --agent plan "Refine user story US-1234 and check
for metadata conflicts"
copado-hx ai ask --agent build "Generate Apex class for lead
scoring based on my org metadata"
copado-hx ai ask --agent test "Generate a CRT QWord test script for
the LeadScoring class"
copado-hx ai ask --agent release "What is the deployment status of
US-1234? Any blockers?"
copado-hx ai ask --agent operate "Create a change management plan
for the Spring 2026 release"

# Persistent dialogue sessions (multi-turn conversations)
copado-hx ai chat --agent build               # Opens an
interactive REPL with the Build Agent
copado-hx ai chat --agent release --us US-1234 # Scoped to a
specific user story context
```

The 5 Copado AI Agents — CLI Mapping

The CLI could expose all 5 specialist agents from the Copado AI Platform. Each agent is purpose-built for a specific DevOps lifecycle stage and could be invokable via `--agent <id>`:

Agent	--agent value	Primary CLI Use Cases
Plan Agent	plan	Refine user stories, detect metadata conflicts, sprint planning, backlog management
Build Agent	build	Generate/review Apex code, analyze org metadata, improve test coverage, troubleshoot issues
Test Agent	test	Generate CRT QWord test scripts, review test coverage, advise on automation best practices
Release Agent	release	Trigger commits/promotions/deployments, analyze job errors, generate release notes
Operate Agent	operate	Post-release docs, change management plans, training materials, troubleshooting guides

⚠️ **Important:** The Release Agent's commit, promote, and deploy capabilities are only available for Source Format Pipelines. The CLI should handle this gracefully and surface a clear error for Metadata Pipeline users.

An inspirational Demo for Track A

The live demo could show a developer completing this full flow without opening a browser:

```
None
copado-hx auth login → authenticated
copado-hx story set --id US-1234 → working context set
copado-hx ai ask --agent build "What metadata should I commit for
US-1234?" → AI-guided commit scope
copado-hx commit --message "feat: lead scoring" → commit triggered
copado-hx promote --env UAT --validate → validation deployment
triggered
```

```
copado-hx test run --suite <id> → CRT suite triggered (resolves to
a jobId in the CRT API)
copado-hx test status --execution <id> → check status until finish
(Success/Fail)
copado-hx test results --execution <id> → results displayed inline
copado-hx deploy --env PROD → deployment triggered (with human
approval confirmation prompt)
copado-hx ai ask --agent release "Generate release notes for
US-1234" → release notes output to terminal
```

Technical Tips

- **Language:** Node.js (Oclif or Commander.js), Python (Typer or Click), Go (Cobra), or Rust — team's choice
- **Auth storage:** Secure credential storage (OS keychain or encrypted local file) — no plaintext tokens in config files
- **Output formats:** Human-readable by default; --json flag for machine-readable output on all commands
- **Error handling:** Meaningful error messages that surface Copado API error details, not raw HTTP responses
- **Exit codes:** Standard POSIX exit codes (0 = success, 1 = error) for CI/CD pipeline compatibility
- **Config file:** .copado-hx.json in project root for pipeline/org defaults

---

🟣 Track B: The Agentic Orchestrator (The Autonomous AI Approach)

The Concept: Focuses on enabling autonomous coding assistants (like Cursor, Claude Code, or custom AI agents) to interact with Copado pipelines on behalf of the user.

Track B builds on Track A. The starting point is a working copado-hx CLI, which is then extended with a SKILL.md — a machine-readable, agent-consumable instruction file that teaches any AI coding agent (Cursor, Claude, OpenAI, Agentforce, or any MCP-compatible system) exactly how to use copado-hx as a native skill within their development workflow.

Teams who want to build an agentic Copado integration without the CLI foundation should consider Track C.

The result: a developer working in Cursor says *"promote my user story to UAT and run the smoke tests"* — and the AI agent knows exactly which copado-hx commands to run, in what order, with what parameters, and how to interpret the results.

copado-hx becomes a Copado skill that any agent can learn.

What is SKILL.md?

SKILL.md is a structured Markdown file — placed at the root of the copado-hx repository — that serves as the agent instruction manual for the CLI. It is the bridge between the CLI's capabilities and the reasoning layer of any AI agent system.

Think of it as:

- A system prompt for agents that need to use copado-hx
- A tool manifest that describes every command, its inputs, outputs, and when to use it
- A workflow playbook that teaches agents how to chain commands across the DevOps lifecycle
- A guardrails document that tells agents what they must never do autonomously (e.g., deploy to PROD without human confirmation)

SKILL.md is consumed by agents via:

- **Cursor:** Added to .cursor/rules/ or referenced in the system prompt
- **Claude / OpenAI:** Injected as a system prompt or tool description
- **Agentforce:** Packaged as an Agent Action instruction set
- **MCP servers:** Exposed as a tool schema via the Model Context Protocol

SKILL.md — Required Sections with sample content

1. Identity

Who this skill is, what it does, and what systems it connects to.

Markdown

```
None
## Identity
You have access to `copado-hx`, a CLI that gives you full control
over the Copado DevOps platform for Salesforce. Through this skill
you can manage user stories, trigger CI/CD pipeline actions
(commit, promote, validate, deploy), execute Copado Robotic Testing
(CRT) test suites, and converse with Copado's 5 specialist AI
agents (Plan, Build, Test, Release, Operate) — all without opening
a browser.
```

2. Prerequisites

What must be true before the agent can use any command.

Markdown

```
None
## Prerequisites
- `copado-hx auth status` must return an authenticated session
before any other command.
- If not authenticated, instruct the user to run `copado-hx auth
login` and pause.
- A working user story context must be set with `copado-hx story
set` before commit, promote, or deploy operations.
- Never infer or fabricate pipeline IDs, environment names, or user
story IDs. Always retrieve them from `copado-hx story list` or
`copado-hx status`.
```

3. Commands Reference

A structured, machine-parseable description of every command group: purpose, syntax, key flags, output schema, and example.

Markdown

```
None
## Commands Reference

### `copado-hx commit`
**Purpose:** Commits metadata changes from the current user story
to Git and updates the Copado user story record.
**When to use:** After the developer has made local code/config
changes and wants to push them to the feature branch.
**Syntax:** `copado-hx commit [--message <msg>] [--us <id>]`
**Output:** JSON with `{ commitId, status, filesCommitted[] }`
**Example:** `copado-hx commit --message "feat: add lead scoring"`
**Do not use if:** No user story context is set. Run `copado-hx
story set` first.

### `copado-hx promote`
**Purpose:** Promotes a user story to the next environment in the
pipeline.
**Flags:**
  - `--validate` : Run a validation-only deployment (no actual
deploy)
  - `--env <name>` : Target environment (e.g., UAT, SIT, PROD)
**Output:** JSON with `{ promotionId, status, jobExecutionId }`
**Poll for completion:** Use `copado-hx status --job
<jobExecutionId> --watch`
```

```
### `copado-hx test run`
**Purpose:** Triggers a CRT test suite or job execution.
**Output:** JSON with `{ executionId, status, projectId, jobId }`
**Poll for results:** Use `copado-hx test status --execution <id>`
until status is `Succeeded` or `Failed`. Then call `copado-hx test
results`.

### `copado-hx ai ask`
**Purpose:** Sends a prompt to one of the 5 Copado AI specialist
agents.
**Agents:** plan | build | test | release | operate
**Output:** Streaming text response from the agent.
**When to use each agent:**
  - `plan`: User story refinement, conflict detection, sprint
planning
  - `build`: Code generation, metadata analysis, coverage
improvement
  - `test`: QWord test script generation, automation advice
  - `release`: Deployment coordination, job error analysis, release
notes
  - `operate`: Post-release docs, change management,
troubleshooting guides
```

4. Workflow Playbooks

Step-by-step multi-command workflows the agent must follow for common scenarios. These are the agent's "recipes."

Markdown

```
None
## Workflow Playbooks

### Playbook: Full Story Delivery (Commit → UAT → Test → PROD)
Use this when the developer says: "ship my user story", "promote to
prod", "deploy US-1234 end to end", or similar.

**Steps:**
1. Verify auth: `copado-hx auth status`
2. Set context: `copado-hx story set --id <us-id>`
3. Ask Build Agent for commit guidance:
```

```
   `copado-hx ai ask --agent build "What metadata should I commit
for <us-id>?"`
4. Commit: `copado-hx commit --message "<generated message>"`
5. Promote + validate to UAT: `copado-hx promote --env UAT
--validate`
6. Poll until complete: `copado-hx status --watch`
7. Run CRT smoke tests: `copado-hx test run --suite
<smoke-suite-id>`
*(Note: --suite is a convenience alias for a CRT jobId — retrieve
it from `copado-hx test list`)*
8. Poll test results: `copado-hx test status --execution <id>
--watch`
9. **STOP. Ask the human:** "Tests passed. Shall I proceed to
deploy to PROD?"
10. Only on explicit human approval: `copado-hx deploy --env PROD`
11. Generate release notes: `copado-hx ai ask --agent release
"Generate release notes for <us-id>"`

### Playbook: Investigate a Failed Deployment
Use this when the developer says: "why did my deployment fail?",
"fix my pipeline error".

**Steps:**
1. `copado-hx status` → retrieve the failed job execution ID
2. `copado-hx ai ask --agent release "Analyze the job execution
error for <jobExecutionId>"`
3. Present the root cause and suggested fix to the developer.
4. If a code fix is needed: `copado-hx ai ask --agent build "Fix
the issue: <error summary>"`

### Playbook: Generate and Run a Test
Use this when the developer says: "write a test for my class",
"test this feature".

**Steps:**
1. `copado-hx ai ask --agent test "Generate a CRT QWord test script
for <class/feature>"`
2. Present the generated script to the developer for review.
3. **STOP. Ask the human:** "Shall I trigger this test suite?"
4. On approval: `copado-hx test run --suite <id>`
```

```
*(Note: --suite is a convenience alias for a CRT jobId — retrieve
it from `copado-hx test list`)*
5. `copado-hx test results --execution <id>`
```

5. Guardrails — What Agents Must Never Do

The non-negotiable safety rules.

Markdown

```
None
## Guardrails — What Agents Must Never Do

🚫 **Never deploy to a PROD or production environment without
explicit human confirmation.** Always pause and ask: "I'm about to
deploy to PROD. Please confirm."

🚫 **Never fabricate or guess IDs** (user story IDs, pipeline IDs,
environment names, suite IDs). Always retrieve them from the CLI
first.

🚫 **Never run `copado-hx deploy` immediately after `copado-hx
promote`** without checking test results and receiving human
approval.

🚫 **Never store or log API tokens** in any output, file, or
message.

🚫 **Never chain more than 3 destructive actions** (commit,
promote, deploy) without a human checkpoint between each stage.

⚠️ **Always surface test failures to the human** before proceeding
to the next pipeline stage. Do not auto-retry failed tests.
```

6. Output Parsing Guide

How the agent should interpret CLI output to make decisions.

Markdown

```
None
## Output Parsing Guide
```

```
All `copado-hx` commands support `--json` for structured output.
Always use `--json` when parsing output programmatically.

| Field | Meaning | Agent Action |
|---|---|---|
| `status: "Completed Successfully"` | Action succeeded | Proceed
to next step |
| `status: "Completed with Errors"` | Partial failure | Stop,
surface errors to human |
| `status: "In Progress"` | Still running | Poll again in 10
seconds |
| `status: "Failed"` | Hard failure | Stop, invoke Release Agent
for analysis |
| `testResult: "Succeeded"` | All tests passed | Safe to proceed |
| `testResult: "Failed"` | Tests failed | Stop, surface failures,
do not deploy |
```

7. Agent Persona Routing

When to switch between the 5 Copado AI agents based on developer intent.

Markdown

```
None
## Agent Persona Routing

When the developer's request maps to a DevOps lifecycle stage,
route to the appropriate Copado AI agent using `copado-hx ai ask
--agent <id>`:

| Developer Says | Route to Agent |
|---|---|
| "Write a user story", "plan this feature", "check for conflicts"
| `plan` |
| "Write the code", "generate Apex", "review my class", "fix this
bug" | `build` |
| "Write a test", "generate test script", "improve coverage" |
`test` |
| "Deploy this", "promote to UAT", "why did it fail?", "release
notes" | `release` |
| "Write docs", "create training material", "change management
plan" | `operate` |
```

Inspirational Demo for Track B

The live demo could show an AI agent (Cursor, Claude, or equivalent) receiving a single natural-language instruction from a developer and autonomously executing a multi-step copado-hx workflow — pausing at the right moments for human approval — using only the SKILL.md as its instruction set.

Example demo scenario:

Developer types in Cursor chat: *"My lead scoring feature is ready. Run the tests and if they pass, deploy to UAT."*

The agent could do but not limited to:

1. Read SKILL.md to understand the workflow.
2. Execute copado-hx story show → copado-hx test run → poll results.
3. Surface test results to the developer.
4. Ask for explicit approval before promoting to UAT.
5. Execute copado-hx promote --env UAT --validate on approval.
6. Report the outcome inline in the IDE.

No Copado UI tab should be opened at any point during the demo.

Track B Bonus Challenges

- **MCP Server:** Package copado-hx as a Model Context Protocol (MCP) server so it can be natively discovered and invoked by any MCP-compatible agent (Cursor, Claude Desktop, etc.) without manual SKILL.md injection.
- **Agentforce Action:** Wrap copado-hx commands as Agentforce Agent Actions callable from a Salesforce Agentforce agent.
- **Multi-agent handoff:** Demonstrate the Build Agent and Test Agent being invoked in sequence by the external AI agent, with context passed between them via the CLI's --json output.

🟠 Track C: Your Headless Idea (The Open Track)

The Concept: Build any open-source tool, integration, or workflow that eliminates the Copado browser UI from a developer's daily flow — using the Copado API surface — in a way not covered by Track A or Track B.

Track C exists because the best headless ideas might not look like a CLI. If your team has a creative approach that achieves the vision, this is your track.

Example directions (not exhaustive):

- A VS Code extension with a Copado sidebar: user stories, pipeline status, one-click commit/promote, inline AI agent chat.
- A Git hooks framework: pre-push triggers a CRT smoke test; post-merge auto-promotes to the next environment.
- A Slack / Teams bot: approve deployments, query pipeline status, and invoke AI agents from a chat thread.
- A Raycast extension: surface Copado user stories and pipeline actions from the macOS launcher.
- A native MCP server: expose the full Copado API surface as MCP tools, discoverable by any MCP-compatible agent without a CLI intermediary.
- A GitHub Actions / GitLab CI composite action: a reusable pipeline step that wraps Copado commit + promote + test into a single declarative workflow block.

Track C Requirements:

- Must use at least two of the three Copado API surfaces (CI/CD, CRT, AI).
- Must demonstrably eliminate at least one class of browser UI interaction.
- All standard submission requirements apply (GitHub repo, README, demo video, live demo, slide deck).
- No SKILL.md required, but bonus points if you include one.

Track C is judged on the same criteria as A and B. Creativity and real-world usefulness carry extra weight for this track.

Copado CI/CD Instance (Agentia Pro) — Actions REST API

Endpoint	Description
POST /actions/commit	Commit metadata from a user story
POST /actions/promote	Promote a user story to the next environment
POST /actions/validate	Run a validation-only deployment

POST /actions/deploy	Execute a deployment
GET /user-stories	List and filter user stories
GET /user-stories/{id}	Get user story details and metadata scope
GET /environments	List pipeline environments
GET /job-executions/{id}	Poll job execution status and logs

Subject to confirmation with the provided API credentials.

Agentia Testing Instance (CRT) — Open API

Endpoint	**Description**
POST /pace/v4/projects/{projectId}/jobs/{jobId}/builds	Trigger a test job execution
GET /pace/v4/projects/{projectId}/jobs/{jobId}/builds/{buildId}	Poll execution status
GET /pace/v4/projects/{projectId}/jobs/{jobId}/builds/{buildId}/results	Retrieve test results
GET /pace/v4/projects/{projectId}/jobs	List available test jobs

- **Auth:** Personal Access Key (PAK) — generated per team from the CRT profile page.

Agentia AI Context Hub (Copado AI Platform) — Dialogue API

Endpoint	Description
POST /dialogues	Start a new dialogue with a specific agent
POST /dialogues/{id}/messages	Send a message and receive a response
GET /dialogues/{id}	Retrieve dialogue history
GET /organizations/{orgId}/workspaces	List available workspaces

Subject to confirmation with the provided API credentials.

- **Agent IDs:** plan · build · test · release · operate
- **Base URL (US):** https://copadogpt-api.robotic.copado.com
- **Auth Header:** X-Authorization: <your-api-key>

📋 Submission Requirements

- **GitHub Repository** — All source code, open-sourced (MIT or Apache 2.0)
- **README.md** — Setup, architecture diagram, API usage, and install instructions
- **SKILL.md** — (Track B required · Track C optional) — The agent instruction file, at the repo root
- **Demo Video (max 5 min)** — Recorded walkthrough of the working solution
- **Live Demo (10 min)** — Presented to judges at CopadoCON Bangalore
- **Slide Deck (max 8 slides)** — Problem, solution, architecture, future vision

🧰 Recommended Tech Stack

Teams are free to use any technology. These are starting points only.

Layer	Options
CLI Framework	Node.js + Oclif or Commander.js · Python + Typer or Click · Go + Cobra
Terminal UI / Watch mode	Ink (React for CLI) · Rich (Python) · Bubble Tea (Go)
MCP Server (Track B bonus)	@modelcontextprotocol/sdk (Node.js) · mcp (Python)
Auth / Token storage	keytar (Node.js) · keyring (Python) · OS keychain
HTTP Client	axios · httpx · Go net/http
Output formatting	chalk · rich · lipgloss

💡 Inspiration Prompts

- *"What if copado-hx commit was as natural as git commit?"*
- *"What if Cursor could say 'I see you changed LeadScoring.cls — want me to commit it to US-1234 and run the CRT smoke tests?'"*
- *"What if the Build Agent could review your Apex diff before you commit, right in the terminal?"*
- *"What if SKILL.md was the new package.json — the file every Copado project ships with so any AI agent knows how to work with it?"*
- *"What if a failed deployment at 2am could be diagnosed and fixed by an agent, with a Slack message asking you to approve the fix?"*

📌 Rules

1. All code must be written during the hackathon — no pre-built solutions.
2. Teams may use any open-source libraries and frameworks.

3. All Copado API calls must use the provided hackathon credentials.
4. API credentials must never be committed to source code or logged to output.
5. Teams of 1–5 participants.
6. The Copado UI may be used for setup/configuration only — not as part of the demo flow.
7. The Orchestrate Agent and Agentia Studio are out of scope — use only the 5 specialist agents: Plan, Build, Test, Release, Operate.
8. Track C teams must clearly state in their README which Copado API surfaces they use and which UI interactions their solution eliminates.

CopadoCON Bangalore | *June 2026* | *Copado Headless Hackathon*

"Developers can do everything from the CLI." — The Copado Creed 🚀