0x06tudio
Progetto didattico di Ingegneria del Software
Università degli Studi di Padova

0x06tudio@gmail.com

Architettura di Integrazione, Tooling e Dashboarding

Studio Tecnologico – Capitolato C9: SpecCheck (Zucchetti S.p.A.)

Informazioni sul documento

Versione:	0.1.0
Stato:	Approvato
Data ultimo aggiornamento:	2026/11/16
Uso:	Interno
Destinatari:	0x06tudio

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
0.1.0	2026/11/15	Stesura iniziale del documento	Pietro Brocadello	Catalin Chirilov

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Riferimenti . . . 3
1.2.1 Riferimenti normativi . . . 3
1.2.2 Riferimenti informativi . . . 3

2 Contesto e Vincoli del Capitolato C9 **4**
2.1 Visione d'insieme dell'architettura . . . 5

3 Plugin e Sidecar Executables per Agenti CLI **5**
3.1 Punti di estensione degli agenti CLI . . . 5
3.2 Progettazione di un plugin . . . 6
3.3 Il pattern Sidecar . . . 7
3.4 Modalità di comunicazione plugin–sidecar . . . 8
3.5 Contratto dati e versionamento . . . 8
3.6 Ciclo di vita, packaging e prestazioni . . . 9
3.7 Testing e qualità . . . 9
3.8 Rischi e mitigazioni . . . 10

4 Interfacce di Presentazione: TUI e Bridge verso Dashboard Web **10**
4.1 Due livelli con peso diverso . . . 10
4.2 Terminal User Interface (TUI) . . . 10
4.3 Bridge di comunicazione verso la dashboard web/HTML . . . 12
4.4 Sicurezza del bridge locale . . . 12
4.5 Modello dati condiviso e sviluppo della dashboard . . . 13

5 Automazione del Deployment: Script Unificati Bash/PowerShell **13**
5.1 Obiettivo . . . 13
5.2 Perché due linguaggi di scripting . . . 14
5.3 Passi tipici dello script di installazione . . . 14
5.4 Buone pratiche per gli script . . . 14
5.5 Distribuzione e sicurezza della supply chain . . . 15
5.6 Testing degli script e rischi . . . 15

6 Valutazione e Confronto **16**
6.1 Vantaggi e svantaggi dell'architettura plugin + sidecar . . . 16
6.2 Confronto tra le soluzioni di integrazione . . . 16
6.3 Confronto tra le soluzioni di presentazione . . . 17

7 Conclusioni e Raccomandazioni **17**
7.1 Collegamento con i requisiti del capitolato . . . 18

1 Introduzione

1.1 Scopo del documento

Il presente studio tecnologico approfondisce le tecnologie candidate per la parte di integrazione, tooling e presentazione dei risultati del progetto SpecCheck (Capitolato C9 proposto da Zucchetti S.p.A.). Le aree trattate sono tre:

- **Plugin e sidecar executables:** progettazione di estensioni ed eseguibili di supporto per agenti a riga di comando (CLI) come OpenCode;
- **Interfacce di presentazione:** interfacce utente da terminale (TUI_G) e bridge di comunicazione verso dashboard grafiche web/HTML;
- **Automazione del deployment:** script unificati Bash/PowerShell per l'installazione, la configurazione e la distribuzione automatizzata del toolchain.

Per ciascuna area vengono descritti il problema, le opzioni disponibili, i criteri di scelta, i rischi e una proposta di utilizzo. Le scelte indicate sono proposte motivate, da validare con il gruppo e con il proponente. I dettagli sulle API e sui formati di configurazione di OpenCode vanno verificati sulla documentazione ufficiale aggiornata, perché lo strumento evolve rapidamente.

1.2 Riferimenti

1.2.1 Riferimenti normativi

- Capitolato d'appalto C9: *SpecCheck – Sviluppo di Skills per OpenCode per la verifica della tracciabilità end-to-end da casi d'uso a requisiti e codice sorgente* (Zucchetti S.p.A., A.A. 2026/2027);
- Norme di Progetto del gruppo 0x06tudio (in fase di redazione).

1.2.2 Riferimenti informativi

- Studio tecnologico interno: *Tracciabilità Semantica: Casi d'Uso vs Specifiche*, 0x06tudio, versione 0.1.0 (pipeline ibrida con core deterministico in Python);
- OpenCode (agente di harness): `https://opencode.ai/`;
- Documentazione di OpenCode: `https://opencode.ai/docs/`;
- OpenSpec (comandi, skill ed eseguibile di appoggio): `https://openspec.dev/`;
- GitHub Spec-Kit: `https://github.com/github/spec-kit`;
- Model Context Protocol: `https://modelcontextprotocol.io/`;
- Librerie TUI:
  - Bubble Tea (Go): `https://github.com/charmbracelet/bubbletea`;
  - Ratatui (Rust): `https://ratatui.rs/`;
  - Textual (Python): `https://textual.textualize.io/`;
  - FTXUI (C++): `https://github.com/ArthurSonzogni/FTXUI`;
- Strumenti per script e test:

- ShellCheck: https://www.shellcheck.net/;
- Bats: https://github.com/bats-core/bats-core;
- Pester: https://pester.dev/;
- PSScriptAnalyzer: https://github.com/PowerShell/PSScriptAnalyzer.

2 Contesto e Vincoli del Capitolato C9

Il capitolato C9 chiede di realizzare estensioni di OpenCode (skill, tool, comandi e almeno un plug-in) che verifichino la tracciabilità da casi d'uso a requisiti e a codice sorgente. Il prodotto finale deve essere un pacchetto unico, corredato di un sistema di installazione. La tabella seguente riassume i vincoli che orientano le scelte di questo documento.

Indicazione del capitolato	**Conseguenza per questo studio**
Estendere OpenCode con skill, tool, comandi e almeno un plug-in, come prodotto unico	Architettura a pacchetto: skill e comandi orchestrano, tool e sidecar verificano, il plug-in ospita la dashboard
Il confronto tra specifiche e codice non può essere affidato al solo LLM: va realizzato come plug-in o come applicativo esterno richiamato da skill e comandi (esempio citato: OpenSpec)	Giustifica il $sidecar_G$ e il contratto dati (Sezione 3)
Dashboard: il requisito obbligatorio ammette una visualizzazione "a caratteri"; la dashboard grafica, anche solo una pagina HTML, è opzionale	La TUI ha priorità; report HTML e bridge sono un'estensione (Sezione 4)
Sistema di installazione unitario, anche solo uno script Bash o PowerShell	Sezione 5
Test automatici soprattutto dall'MVP; modelli grandi con tetto di costo e modelli piccoli (Gemma 4 26B, Qwen 3.8 27B) senza limiti	Il sidecar deve essere testabile senza LLM; i test di integrazione vanno ripetuti anche con modelli piccoli
Pubblicazione su GitHub come titolo preferenziale; manuale d'uso e manuale per chi estende skill, comandi e tool	Struttura del repository, licenza open source e documentazione di estensione
Fasi: PoC (algoritmi di confronto, lettura dei diagrammi), poi MVP (requisiti obbligatori), poi opzionali	Il PoC proposto nella Sezione 7 è coerente con questa sequenza

2.1 Visione d'insieme dell'architettura

L'architettura ipotizzata ha quattro componenti principali, in corrispondenza con lo schema di massima del capitolato (skill, tool, command e plug-in per dashboard attorno a OpenCode):

1. **Agente CLI (OpenCode):** ospita il modello linguistico e il ciclo di interazione con lo sviluppatore;
2. **Skill e comandi:** definiscono il flusso di verifica e le regole di giudizio per l'LLM;
3. **Sidecar:** eseguibile separato che svolge la parte deterministica (parsing, indicizzazione, confronto, generazione dei report) e comunica con il plug-in tramite un protocollo definito;
4. **Livelli di presentazione:** una TUI per l'uso interattivo da terminale e, come requisito opzionale, una dashboard web/HTML per la consultazione e la condivisione dei risultati.

Il flusso tipico è il seguente: lo sviluppatore invoca un comando o una skill di SpecCheck nell'agente; il plug-in inoltra la richiesta al sidecar; il sidecar produce risultati strutturati (JSON); questi vengono mostrati nella TUI e, se richiesto, pubblicati verso la dashboard tramite un bridge locale.

Questa impostazione è coerente con lo studio *Tracciabilità Semantica*, che propone un core deterministico in Python (CLI) richiamato da una skill OpenCode. Il sidecar descritto qui ne rappresenta il livello di integrazione e presentazione.

3 Plugin e Sidecar Executables per Agenti CLI

3.1 Punti di estensione degli agenti CLI

Gli agenti a riga di comando basati su LLM (OpenCode, Claude Code, Gemini CLI e simili) condividono un'architettura simile: un ciclo che alterna ragionamento del modello e invocazione di strumenti. L'estendibilità avviene attraverso più meccanismi, con diverso grado di accoppiamento.

Meccanismo	**Descrizione e uso in SpecCheck**	**Accoppiamento**
Skill	Istruzioni in file che guidano un compito e vengono attivate dall'harness quando serve; definiscono il flusso di verifica	Basso
Comandi personalizzati	Prompt predefiniti richiamabili con un nome (es. `/speccheck`); avvio guidato di un'analisi	Basso
Tool personalizzati	Funzioni richiamabili dal modello con parametri tipizzati; espongono all'LLM le funzioni del sidecar	Medio
Plugin	Moduli caricati dall'agente che si agganciano a eventi del ciclo di vita; raccolta di contesto e dashboard	Medio-alto
Server MCP	Processi esterni che espongono tool e risorse con un protocollo standard; alternativa standardizzata al sidecar	Basso
SDK / API client-server	Controllo programmatico dell'agente da un'applicazione esterna	Basso

Per OpenCode, a quanto risulta dalla documentazione consultabile, i plugin sono moduli JavaScript/TypeScript caricati da directory di progetto o globali (o da pacchetti npm indicati nel file di configurazione) e possono registrare hook su eventi come l'esecuzione di un tool (prima e dopo), la gestione dei file e della sessione. Esistono inoltre tool personalizzati, agenti configurabili, comandi personalizzati e il supporto a server MCP. OpenCode adotta anche un'architettura client-server: la TUI è un client di un server locale che espone un'API HTTP, per la quale esiste un SDK. Questo permette sia di estendere l'agente dall'interno (plugin) sia di pilotarlo dall'esterno (SDK). Tali dettagli vanno verificati sulla documentazione aggiornata prima di fissare le scelte.

3.2 Progettazione di un plugin

Un plugin ben progettato deve essere **sottile**: contiene solo la logica di integrazione, non la logica di dominio. Le responsabilità consigliate sono:

- **Registrazione dei tool:** definire uno o più tool, ad esempio `speccheck_analyze` e `speccheck_report`, con uno schema esplicito dei parametri, così che il modello sappia quando e come invocarli;
- **Hook sul ciclo di vita:** reagire a eventi (ad esempio dopo la modifica di un file) per segnalare che i risultati precedenti sono obsoleti;
- **Gestione della configurazione:** leggere le impostazioni di SpecCheck (percorso del sidecar, file dei requisiti, soglie) da un file dedicato nel repository;

- **Delega al sidecar:** avviare o raggiungere il sidecar, inoltrare la richiesta e convertire la risposta in un riassunto compatto per il modello e in un formato strutturato per la dashboard.

Buone pratiche di progettazione:

- **Separazione dei compiti:** il plugin non conosce i dettagli dell'analisi, il sidecar non conosce l'agente;
- **Output limitato:** al modello si restituiscono riassunti compatti, perché ogni token consumato riduce il contesto disponibile; i dettagli completi vanno salvati su file e referenziati per percorso;
- **Gestione esplicita degli errori:** un fallimento del sidecar diventa un messaggio chiaro per l'agente e per l'utente, non un'eccezione che interrompe la sessione;
- **Determinismo e *idempotenza*$_G$:** due invocazioni con gli stessi input producono gli stessi artefatti, così i risultati sono confrontabili e testabili;
- **Sicurezza:** il plugin esegue codice con i permessi dell'utente. Vanno evitati comandi costruiti concatenando stringhe provenienti dal modello (rischio di command injection e prompt injection); si usano argomenti passati come lista e validazione degli input.

3.3 Il pattern Sidecar

Il *sidecar*$_G$ è un eseguibile separato che affianca l'applicazione principale (qui l'agente CLI) e le fornisce funzionalità aggiuntive. È la forma di "applicativo esterno richiamato da skill e comandi" indicata dal capitolato, ed è l'architettura di OpenSpec (comandi, skill ed eseguibile di appoggio).

Perché un sidecar e non solo un plugin:

- **Indipendenza dal linguaggio:** il plugin è vincolato all'ecosistema dell'agente (per OpenCode, JavaScript/TypeScript), mentre il sidecar può essere scritto nel linguaggio più adatto all'analisi (Python per il parsing e le librerie di elaborazione, Go o Rust per un binario unico);
- **Isolamento:** un crash o un consumo eccessivo di memoria del sidecar non abbatte l'agente;
- **Riuso:** lo stesso sidecar può essere richiamato da altri agenti, da script CI o dalla dashboard;
- **Testabilità:** si può testare il sidecar in modo autonomo, senza avviare l'agente né un LLM;
- **Ciclo di rilascio separato:** il sidecar può essere aggiornato senza modificare il plugin, purché il contratto resti compatibile.

Svantaggi e costi:

- più artefatti da distribuire e versionare;
- gestione del ciclo di vita del processo (avvio, attesa di prontezza, arresto, riavvio);
- necessità di un protocollo di comunicazione ben definito;

- difficoltà aggiuntive su più sistemi operativi (percorsi, permessi, antivirus su Windows).

3.4 Modalità di comunicazione plugin–sidecar

Modalità	**Pro**	**Contro**
Esecuzione one-shot (argomenti, lettura di stdout)	Semplicissima, nessuno stato	Costo di avvio a ogni chiamata
Processo persistente su stdin/stdout (es. JSON-RPC 2.0)	Nessuna porta di rete, avvio una sola volta	Gestione dei flussi, debug meno immediato
Server HTTP locale su `127.0.0.1`	Facile da testare, riusabile dalla dashboard	Gestione delle porte e dell'autenticazione locale
Server MCP_G	Standard, riusabile da più agenti	Maggiore complessità iniziale
Socket Unix / named pipe	Efficiente, non esposto in rete	Differenze tra Unix e Windows

Per SpecCheck si propone un **sidecar con interfaccia a riga di comando one-shot** per le operazioni semplici e, solo se servirà, una **modalità server locale** per sessioni interattive e dashboard, mantenendo un unico contratto dati. Se in futuro interessasse l'interoperabilità con più agenti, le stesse funzioni possono essere esposte anche come server MCP.

3.5 Contratto dati e versionamento

Il contratto tra i componenti è la parte più importante e va trattato come un'interfaccia pubblica:

- **schema esplicito** dei messaggi (JSON Schema o tipi condivisi) per richieste, risposte, errori e avanzamento;

- **versione del protocollo** in ogni messaggio e verifica all'avvio (handshake): se plugin e sidecar non sono compatibili, il sistema segnala l'errore;
- **codici di errore stabili** e messaggi leggibili;
- **formati di report standard** quando possibile, ad esempio $SARIF_G$ per i risultati di analisi, oppure un JSON proprietario documentato;
- **versionamento semantico** (SemVer) di plugin, sidecar e protocollo.

Gli esiti di verifica possono riutilizzare la classificazione dello studio sulla tracciabilità semantica (`COVERED`, `PARTIAL`, `CONTRADICTION`, `MISSING`), così che sidecar e skill parlino lo stesso linguaggio.

```
{ "protocol": "1.0", "id": "42", "method": "analyze",
  "params": { "path": "src/", "requirements": "docs/requisiti.md"
     } }

{ "protocol": "1.0", "id": "42", "status": "ok",
  "result": { "covered": 18, "partial": 2, "missing": 1,
              "reportPath": ".speccheck/report.json" } }
```

Listing 1: Esempio indicativo di richiesta e risposta del sidecar

3.6 Ciclo di vita, packaging e prestazioni

- **Avvio e prontezza:** il plugin avvia il sidecar in modo pigro (alla prima richiesta) e attende un segnale di prontezza;
- **Arresto pulito:** alla chiusura della sessione il plugin termina il sidecar, con timeout e terminazione forzata per evitare processi orfani;
- **Streaming dei progressi:** per analisi lunghe il sidecar invia eventi di avanzamento, così TUI e dashboard possono mostrare lo stato;
- **Packaging:** preferire un binario autonomo per piattaforma (Windows, Linux, macOS; x86-64 e ARM64) per ridurre le dipendenze sul sistema dell'utente;
- **Cancellazione e timeout:** ogni operazione deve poter essere interrotta.

3.7 Testing e qualità

- test unitari del sidecar sulle funzioni di dominio;
- test di contratto che verificano che plugin e sidecar rispettino lo stesso schema;
- test di integrazione end-to-end con un agente reale o simulato (mock);
- test di robustezza: sidecar assente, versione incompatibile, risposta malformata, timeout;
- ripetizione dei casi d'uso principali con modelli piccoli (Gemma 4 26B, Qwen 3.8 27B), come richiesto dal capitolato per verificare la solidità delle realizzazioni.

3.8 Rischi e mitigazioni

Rischio	Mitigazione
Le API dell'agente cambiano tra versioni	Fissare la versione supportata; isolare l'uso dell'API del plugin in un solo modulo; test di compatibilità
Sidecar non trovato o non eseguibile	Verifica all'avvio con messaggio guidato; script di installazione (Sezione 5)
Esecuzione di codice non attendibile proveniente dal modello	Validazione degli input, nessuna interpolazione di stringhe nei comandi, permessi minimi
Sovraccarico del contesto del modello	Riassunti compatti, dettagli su file
Differenze tra sistemi operativi	Matrice di test su Windows, Linux e macOS

4 Interfacce di Presentazione: TUI e Bridge verso Dashboard Web

4.1 Due livelli con peso diverso

Gli sviluppatori lavorano nel terminale e vogliono un riscontro immediato senza cambiare contesto: la TUI risponde a questa esigenza. La dashboard web/HTML serve a scopi diversi: panoramica, grafici, navigazione dei risultati, condivisione con chi non usa il terminale e archiviazione dei report. I due livelli devono alimentarsi dalla stessa fonte dati per evitare incoerenze.

Nel capitolato i due livelli hanno peso diverso. Il requisito obbligatorio chiede di riportare lo stato di realizzazione, evidenziando quali casi d'uso sono rilevati nelle specifiche e quali specifiche nel codice, e ogni discrepanza deve essere facilmente percepibile dal programmatore. Una vista testuale è accettata, perché OpenCode privilegia il terminale. La dashboard grafica (anche una pagina HTML) è un requisito opzionale. La TUI è quindi il primo livello da realizzare; il bridge verso HTML è il primo passo opzionale.

4.2 Terminal User Interface (TUI)

Una TUI è un'interfaccia testuale interattiva che usa il terminale come superficie di disegno: finestre, tabelle, liste navigabili, barre di avanzamento, scorciatoie da tastiera. Tecnicamente si basa su sequenze di escape ANSI e sul controllo della modalità del terminale (modalità raw, schermo alternativo, gestione del ridimensionamento).

Linguaggio	**Libreria**	**Note**
Go	Bubble Tea (con Lip Gloss e Bubbles)	Architettura Model–Update–View, molto usata
Rust	Ratatui	Rendering a immediate mode, prestazioni elevate
Python	Textual, Rich	Textual offre widget e styling in stile CSS; Rich per output formattato semplice
JavaScript/TypeScript	Ink, blessed	Naturale se il plugin è già in TypeScript
C/C++	ncurses, FTXUI	Controllo massimo; FTXUI è un'opzione moderna in C++

Se il sidecar è scritto in Python, come suggerisce lo studio sulla tracciabilità semantica, Textual (o Rich per una vista più semplice) è la scelta più coerente. Per un sidecar in C++, per cui il gruppo ha competenze maturate nei corsi di programmazione a oggetti, FTXUI è un'alternativa naturale.

Principi di progettazione di una TUI efficace:

- **separare stato e vista:** l'architettura Model–Update–View rende la TUI testabile senza un terminale;
- **operazioni non bloccanti:** l'analisi gira in un altro thread o processo e la TUI mostra i progressi;
- **navigazione da tastiera** coerente e documentata (frecce, tab, `/` per cercare, `q` per uscire, `?` per l'aiuto);
- **degradazione elegante:** supportare terminali senza colori o con larghezza ridotta, rispettare la variabile `NO_COLOR`, offrire una modalità non interattiva quando l'output è rediretto (rilevamento TTY);
- **accessibilità:** non affidare il significato solo al colore (usare anche simboli o testo) e prevedere alternative ASCII a Unicode;
- **compatibilità multipiattaforma:** Windows Terminal, PowerShell, cmd e terminali Unix si comportano in modo diverso.

Contenuti tipici della TUI di SpecCheck:

- elenco dei casi d'uso e delle specifiche con stato (coperto, parziale, in contraddizione, non coperto);
- dettaglio di una specifica con i frammenti di codice collegati e la motivazione dell'esito;
- barra di avanzamento e log dell'analisi in corso;
- riepilogo finale con indicatori sintetici e comando per aprire la dashboard HTML, se presente.

Integrazione con OpenCode: poiché OpenCode ha una propria TUI, SpecCheck può (a) restituire testo formattato nei risultati dei tool, (b) offrire un comando separato (`speccheck tui`) che apre la propria interfaccia a schermo intero, oppure (c) collegarsi al server dell'agente tramite SDK per mostrare lo stato della sessione. La soluzione (a) è la più semplice e va realizzata comunque; la (b) è la meno invasiva per una vista interattiva.

4.3 Bridge di comunicazione verso la dashboard web/HTML

Il bridge trasferisce i dati dal processo locale (sidecar o plugin) alla dashboard. Sono possibili tre approcci, dal più semplice al più sofisticato.

Approccio A – report statico. Il sidecar genera un file HTML autocontenuto che incorpora i dati (JSON inline) e il codice JavaScript necessario, apribile con un doppio clic. Non richiede server, è archiviabile e allegabile alla documentazione e funziona offline. Non offre aggiornamenti in tempo reale e cresce con i dati.

Approccio B – server locale con aggiornamento in tempo reale. Il sidecar avvia un server HTTP su `127.0.0.1` che serve la dashboard e un'API JSON; gli aggiornamenti arrivano al browser tramite Server-Sent Events (SSE_G) o WebSocket.

Approccio C – dashboard come client dell'agente. La dashboard usa l'API del server dell'agente (se esposta) per leggere gli eventi di sessione. È potente, ma lega la dashboard all'API dell'agente e ne eredita le variazioni.

Tecnologia	Direzione	Caratteristiche
Polling HTTP	Client → server	Semplice, ma inefficiente e con latenza
SSE	Server → client	Unidirezionale, ricollegamento automatico, ideale per progressi e log
WebSocket	Bidirezionale	Più flessibile (comandi dalla dashboard), più complesso

Per una dashboard prevalentemente di consultazione SSE è in genere sufficiente. Una soluzione progressiva per il progetto è partire dall'approccio A (valore immediato, rischio minimo) e aggiungere l'approccio B come estensione.

4.4 Sicurezza del bridge locale

Un server locale è comunque una superficie d'attacco, anche su `localhost`:

- **binding solo su 127.0.0.1**, mai su `0.0.0.0` per impostazione predefinita;
- **token di sessione** casuale generato all'avvio e richiesto dalle richieste, per impedire che altre pagine web o processi usino l'API;
- **controllo di Origin e CORS** restrittivo e verifica dell'intestazione `Host` contro il DNS rebinding;
- **nessun dato sensibile** (chiavi API, segreti) nei payload; attenzione a non pubblicare per errore codice sorgente riservato;
- **Content Security Policy** e sanificazione di ogni contenuto proveniente dal codice analizzato o dal modello (rischio XSS): frammenti di codice e motivazioni generate dall'LLM sono input non fidato;
- porta configurabile, con scelta automatica di una porta libera.

4.5 Modello dati condiviso e sviluppo della dashboard

TUI e dashboard devono leggere lo stesso modello. Si propone un documento di risultato (ad esempio `report.json`) con sezioni per metadati dell'esecuzione, casi d'uso, specifiche, evidenze e metriche, validato da uno schema. In questo modo:

- TUI e dashboard sono semplici viste dello stesso dato;
- il report può essere archiviato, confrontato tra versioni e consumato da altri strumenti (CI);
- i test possono usare file di risultato di esempio (fixture) per verificare le interfacce senza eseguire l'analisi.

La dashboard HTML può essere realizzata con tecnologie web standard, con o senza framework. Senza framework (JavaScript puro, Web Components) si hanno meno dipendenze e un file singolo facilmente distribuibile, ideale per l'approccio A. Con un framework (React, Vue, Svelte) e uno strumento di build si ottiene una struttura migliore per interfacce ricche, a costo di una fase di build e di più dipendenze. Per i grafici di copertura e le matrici di tracciabilità si possono usare librerie come Chart.js o D3. I criteri di qualità sono: responsività, prestazioni, funzionamento offline e compatibilità con i browser principali.

5 Automazione del Deployment: Script Unificati Bash/PowerShell

5.1 Obiettivo

L'utente deve poter ottenere un toolchain funzionante (plugin, sidecar, skill, comandi, configurazione, eventuali dipendenze) con un solo comando e in modo ripetibile. Il capitolato richiede che l'insieme degli artefatti sia considerato un prodotto unico, con un sistema di installazione che può essere un semplice script Bash o PowerShell. Gli script riducono gli errori manuali e rendono riproducibili gli ambienti di sviluppo, test e dimostrazione.

5.2 Perché due linguaggi di scripting

Bash (o POSIX `sh`) è lo standard su Linux e macOS; su Windows è disponibile tramite WSL o Git Bash, ma non di default. PowerShell è presente su Windows (Windows PowerShell 5.1) ed è disponibile in versione multipiattaforma (PowerShell 7+). Per una distribuzione davvero multipiattaforma si forniscono due script con comportamento equivalente (`install.sh` e `install.ps1`) e un unico punto di verità per versioni, URL e checksum. "Unificato" significa interfaccia e semantica comuni: stessi parametri, stessi passi, stessi messaggi e stessi codici di uscita.

5.3 Passi tipici dello script di installazione

1. **Rilevamento della piattaforma:** sistema operativo e architettura (x86-64, ARM64), per scegliere l'artefatto corretto;
2. **Verifica dei prerequisiti:** presenza e versione minima di OpenCode, Node.js (se richiesto dal plugin), Git ed eventuale runtime, con messaggi chiari su come risolvere;
3. **Scaricamento degli artefatti** dal canale di distribuzione (ad esempio GitHub Releases) in una directory temporanea;
4. **Verifica di integrità:** confronto del checksum SHA-256 e, se possibile, della firma; interruzione in caso di mancata corrispondenza;
5. **Installazione:** copia del sidecar in una directory dell'utente (senza privilegi di amministratore, se possibile), copia di skill e comandi e installazione del plugin nella posizione riconosciuta dall'agente;
6. **Configurazione:** creazione o aggiornamento non distruttivo dei file di configurazione, con backup dell'originale;
7. **Aggiornamento del PATH** o creazione di un collegamento, con indicazione all'utente se serve riaprire il terminale;
8. **Verifica finale (smoke test):** esecuzione di `speccheck -version` e di un controllo diagnostico;
9. **Riepilogo** di cosa è stato installato e dove, con il comando per la disinstallazione.

5.4 Buone pratiche per gli script

Bash:

- usare `set -euo pipefail` e gestire le uscite con `trap` per ripulire i file temporanei;
- citare sempre le variabili e verificare i comandi con `command -v`;
- analizzare il codice con ShellCheck.

PowerShell:

- impostare `$ErrorActionPreference = 'Stop'` e usare `Set-StrictMode`;
- gestire la Execution Policy, spesso restrittiva, con istruzioni chiare o con un'esecuzione limitata al processo;

- usare `Join-Path` per i percorsi e prestare attenzione alle differenze tra PowerShell 5.1 e 7;
- analizzare il codice con PSScriptAnalyzer e testarlo con Pester.

Comuni a entrambi:

- **idempotenza:** rieseguire lo script non deve rovinare un'installazione esistente;
- **parametri:** versione da installare, directory di destinazione, modalità non interattiva, simulazione (`-dry-run`), disinstallazione;
- **modalità silenziosa e verbosa** con log su file per la diagnostica;
- **codici di uscita** coerenti e messaggi d'errore azionabili;
- **nessun segreto** nello script o nei log;
- **versioni pinnate** degli artefatti per installazioni riproducibili.

5.5 Distribuzione e sicurezza della supply chain

Il modello `curl ... | sh` (o `irm ... | iex` in PowerShell) è comodo ma comporta il rischio di eseguire codice non verificato. Le mitigazioni proposte sono:

- offrire anche la strada "scarica, ispeziona, esegui" e documentarla;
- servire gli script solo via HTTPS da una fonte controllata (release del repository);
- pubblicare checksum e, se possibile, firme degli artefatti, e verificarli prima di installare;
- fissare le versioni delle dipendenze e minimizzarne il numero;
- generare gli artefatti con una pipeline CI/CD riproducibile (ad esempio GitHub Actions), con build per tutte le piattaforme e pubblicazione automatica delle release;
- valutare in prospettiva un elenco dei componenti (SBOM).

Per chi preferisce i gestori di pacchetti sono possibili un pacchetto npm per il plugin, Homebrew, Scoop o winget per il sidecar, oppure un'immagine di container per ambienti isolati; lo script resta il percorso comune di base. La pubblicazione su GitHub, titolo preferenziale nel capitolato, rende naturale l'uso di GitHub Releases come canale.

5.6 Testing degli script e rischi

- esecuzione automatica in CI su una matrice di sistemi (Ubuntu, macOS, Windows) e, dove utile, in container puliti per verificare le installazioni da zero;
- test di idempotenza (doppia esecuzione) e di aggiornamento da versione precedente;
- test dei percorsi di errore: rete assente, checksum errato, prerequisito mancante, permessi insufficienti;
- analisi statica (ShellCheck, PSScriptAnalyzer) come controllo obbligatorio nella pipeline;
- framework di test: Bats per Bash e Pester per PowerShell.

Rischio	**Mitigazione**
Divergenza di comportamento tra script Bash e PowerShell	Specifica comune, test equivalenti in CI, manutenzione congiunta
Modifiche distruttive a configurazioni esistenti	Backup, modalità `dry-run`, modifica minima e reversibile
Esecuzione di artefatti manomessi	Checksum e firma, HTTPS, release controllate
Problemi di permessi e antivirus su Windows	Installazione per utente, istruzioni dedicate, eventuale firma del binario
Dipendenze non presenti sul sistema	Verifica dei prerequisiti con messaggi guidati; binari autonomi

6 Valutazione e Confronto

6.1 Vantaggi e svantaggi dell'architettura plugin + sidecar

- **Pro:** separazione netta tra ragionamento dell'LLM e logica deterministica, come richiesto dal capitolato; sidecar testabile e riusabile senza LLM; indipendenza dal linguaggio; aggiornamenti separati;
- **Contro:** più artefatti da distribuire, gestione del ciclo di vita del processo, necessità di un contratto dati versionato, maggiore attenzione ai sistemi operativi diversi.

6.2 Confronto tra le soluzioni di integrazione

Criterio	**Solo plugin**	**Plugin + sidecar**	**Server MCP**
Indipendenza dal linguaggio	Bassa (ecosistema dell'agente)	Alta	Alta
Isolamento dei guasti	Bassa	Alta	Alta
Riuso con altri agenti	Bassa	Media (tramite CLI o HTTP)	Alta (protocollo standard)
Aderenza al capitolato	Ammessa	Alta (modello OpenSpec)	Non citata, ammessa come estensione
Complessità iniziale	Bassa	Media	Media-Alta

6.3 Confronto tra le soluzioni di presentazione

Criterio	**TUI / vista testuale**	**Report HTML statico**	**Server locale + SSE**
Stato nel capitolato	Obbligatorio	Opzionale	Opzionale
Tempo reale	Sì (nel terminale)	No	Sì
Condivisione e archiviazione	Bassa	Alta (file singolo)	Media
Superficie di attacco	Nessuna	Bassa	Media (da mitigare)
Complessità	Media	Bassa	Media-Alta

7 Conclusioni e Raccomandazioni

Lo studio indica un'architettura a pacchetto in cui skill e comandi orchestrano il flusso, un sidecar deterministico svolge la verifica e un plugin integra i risultati in OpenCode. Le scelte proposte sono riassunte nella tabella seguente.

Area	**Proposta**	**Motivazione**
Estensione dell'agente	Plugin sottile, tool personalizzati e comandi dedicati	Integrazione naturale con accoppiamento contenuto
Logica di dominio	Sidecar autonomo, CLI one-shot ed eventuale modalità server locale	Isolamento, riuso, testabilità, indipendenza dal linguaggio
Contratto dati	Schema JSON versionato (o SARIF per i risultati)	Interoperabilità e verificabilità
Presentazione da terminale	Vista testuale nei risultati dei tool e TUI separata (`speccheck tui`)	Requisito obbligatorio, feedback immediato
Presentazione grafica	Report HTML autocontenuto, poi server locale con SSE	Requisito opzionale, valore immediato con rischio minimo
Deployment	`install.sh` e `install.ps1` equivalenti, con verifica dei checksum e CI	Riproducibilità e bassa barriera d'ingresso

7.1 Collegamento con i requisiti del capitolato

Requisito del capitolato	**Dove incide questo studio**
Obbligatori 1–3: lettura di diagrammi dei casi d'uso (Mermaid, XMI, eventualmente immagini), documentazione aggiuntiva e specifiche	Tool esposti dal plugin e implementati nel sidecar
Obbligatori 4–8: confronto, verifica di rispondenza e verifica del comportamento del codice	Parte deterministica nel sidecar, con contratto dati che riporta esiti e motivazioni (Sezione 3)
Obbligatorio 9: dashboard dello stato di realizzazione	TUI o vista testuale (Sezione 4)
Opzionale: dashboard grafica HTML	Report HTML e bridge (Sezione 4)
Opzionale: copertura delle specifiche (codice non derivato da specifiche formali)	Il report deve prevedere anche le parti di codice senza specifica corrispondente
Opzionale: lettura diretta di file PDF	Estrazione di testo e immagini in un tool del sidecar
Pacchetto unico con installazione	Script Bash e PowerShell (Sezione 5)
Manuale d'uso e di estensione, test automatici, pubblicazione su GitHub	Piano di Qualifica e Norme di Progetto