ACME CORP. ENGINEERING

ACME React Plugin SDK

> Trasformare applicazioni in piattaforme estensibili grazie alla runtime composition.

ARCHITETTURA

Contract-first & Web Components

TECNOLOGIA

ESM Dynamic Import

PRESENTATO DA

ACME Engineering Team

01 / INTRODUZIONE

Il problema da risolvere

Le applicazioni enterprise tendono naturalmente all'entropia: crescono nel tempo trasformandosi in **monoliti** difficili da gestire e manutenere.

- Deploy monolitico & lento
- Forte accoppiamento tra moduli
- Impossibilità di rilasci indipendenti
- Ecosistema chiuso a terze parti

Il nodo centrale

Il problema non è React, ma il **modello di composizione**.

BUILD-TIME COMPOSITION

Bundling statico

↓

TRANSITION TO

GOAL

RUNTIME COMPOSITION

Loading on-demand

"Decidere cosa caricare mentre l'applicazione è già in esecuzione"

02 / VISION

Obiettivo della piattaforma plugin

Costruire un ecosistema aperto dove l'estendibilità è una feature nativa, non un ripensamento.

PRINCIPIO CHIAVE

L'host controlla quando caricare.

Il plugin controlla cosa renderizzare.

Separazione netta di responsabilità per massimizzare la stabilità.

Indipendenza Totale

I plugin sono compilati e rilasciati indipendentemente dall'host. L'host non conosce il plugin in fase di build.

`Decoupled Build`

Caricamento Dinamico

Discovery e caricamento avvengono a runtime tramite Manifest JSON e moduli standard ESM.

`ESM Imports`

Contratto Sicuro

L'integrazione è garantita da un contratto TypeScript condiviso che definisce input e capabilities.

`Type Safety`

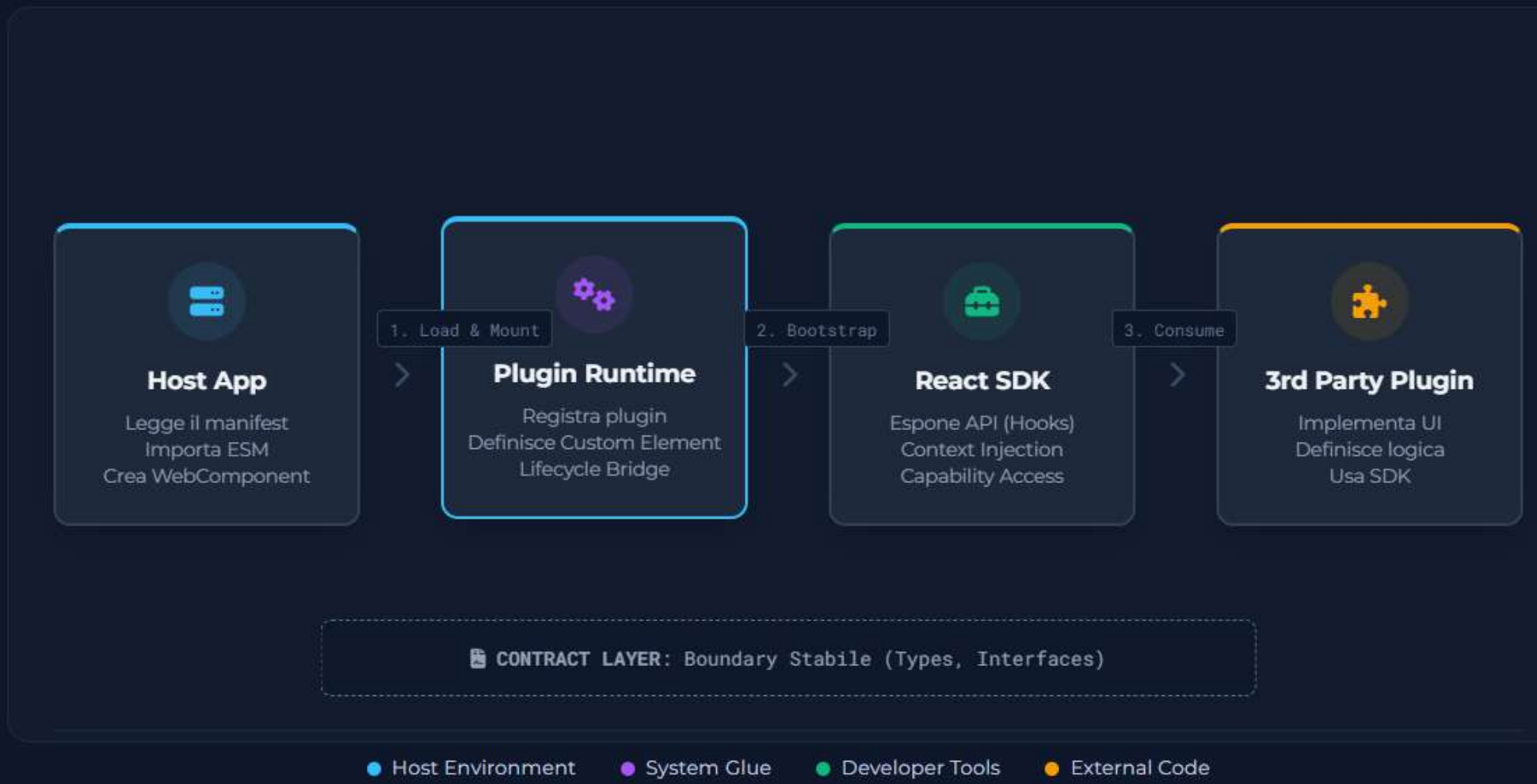
Rendering Isolato

L'interfaccia utente è incapsulata in Web Components per evitare collisioni CSS e conflitti DOM.

`Shadow DOM`

03 / ARCHITETTURA

Architettura ad alto livello



04 / CONTRATTO

Il cuore del sistema: contratto condiviso

types/contract.d.ts

```ts
export type PluginManifest<PT = string> = {
  type: PT;
  id: string;
  version: string;
  entry: string; // ESM entry URL
};

export type PluginContext = {
  user: HostUser;
  services: HostServices;
  manifest: PluginManifest;
};
```

Contract-first Architecture

Il contratto (le definizioni di tipo) è l'unico punto di contatto tra Host e Plugin. Questo elimina completamente le dipendenze dirette a build-time.

Confine Stabile

Le interfacce definiscono esattamente cosa il plugin può fare e quali dati riceve.

Evoluzione Indipendente

Host e plugin possono essere aggiornati separatamente fintanto che rispettano il contratto.

Zero Knowledge

L'host non sa come è fatto il plugin. Il plugin non sa chi è l'host.

HOST APP — CONTRACT — PLUGIN

05 / SDK REACT

Esperienza sviluppatore

Registrazione Plugin

src/index.tsx

```tsx
import { definePlugin } from "@acme/sdk";
import { PluginRoot } from "./Root";

// Entry point dichiarativo
export default definePlugin({
   type: "WIDGET",
   id: "acme.analytics-widget",
   version: "1.0.0",
   Root: PluginRoot,
});
```

Utilizzo Context

src/Root.tsx

```tsx
import { useUser, useServices } from "@acme/sdk";

// Accesso alle capacità dell'host
export function PluginRoot() {
   const user = useUser();
   const api = useServices();

   return (
      <div>Hello {user.name}</div>
   );
}
```

Modello Capability-based

INVERSION OF CONTROL

Il plugin non importa dipendenze dirette. Richiede **capacità** (API, Dati, Servizi) che l'host inietta a runtime attraverso il React Context, garantendo il disaccoppiamento totale.

07 / RUNTIME

Plugin Runtime: Bridge Web Components e React

runtime/PluginElement.ts

```
class PluginElement extends HTMLElement {
  connectedCallback() {
    // 1. Crea Shadow DOM
    const root = this.attachShadow({ mode: 'open' });
    // 2. Monta React root
    this._root = ReactDOM.createRoot(root);
    this._root.render(<PluginApp />);
  }

  disconnectedCallback() {
    // 3. Cleanup React tree
    if (this._root) {
      this._root.unmount();
    }
  }
}

customElements.define("plugin-widget", PluginElement);
```

Il Ponte tra due Mondi

Il runtime gestisce la transizione critica tra il DOM standard del browser (Web Components) e il Virtual DOM di React, garantendo che i due cicli di vita siano sincronizzati.

Standard Nativo

L'host vede solo un elemento HTML standard. Non ha bisogno di sapere che dentro c'è React.

Lifecycle Sync

Quando l'elemento viene rimosso dal DOM, React viene smontato correttamente per evitare memory leaks.

Framework Agnostic

Questo pattern permette di supportare plugin scritti in Angular, Vue o Svelte semplicemente cambiando l'implementazione interna.

DOM (BROWSER) — BRIDGE — VIRTUAL DOM (REACT)

08 / HOST APPLICATION

Caricamento dinamico

Processo di bootstrap del plugin a runtime in 4 fasi.

01

FETCH

```
const manifest = await
fetch(url)
.then(r => r.json());
```

L'host scarica il file JSON che descrive il plugin e i suoi entry point.

02

IMPORT

```
await import(
manifest.entry
);
```

Caricamento asincrono del modulo JavaScript via ESM standard.

03

CREATE

```
const el = document.
createElement(
"plugin-widget");
```

Istanziazione del Web Component registrato dal modulo.

04

INJECT

```
el.ctx = {
user, services
};
root.appendChild(el);
```

Passaggio del contesto (capabilities) e montaggio nel DOM.

09 / LIFECYCLE

Lifecycle completo del plugin

Sequenza end-to-end: dal caricamento del manifest alla distruzione.

01

MANIFEST

Host legge il contratto JSON.

```
fetch("/plugin.json")
```

02

IMPORT

Caricamento asincrono modulo ESM.

```
await import(entry)
```

03

REGISTER

Plugin si annuncia al runtime.

```
definePlugin(config)
```

04

CREATE

Host crea istanza DOM.

```
createElement(tag)
```

05

MOUNT

WebComponent attached to DOM.

```
connectedCallback()
```

06

ACTIVATE

Bootstrap di React/Framework.

```
createRoot().render()
```

07

RENDER

UI visibile e interattiva.

```
<PluginUI />
```

08

DESTROY

Cleanup e unmount.

```
disconnectedCallback()
```

Isolamento tramite Web Components

I Web Components offrono un meccanismo standard del browser per **incapsulare** logica e stile, garantendo che i plugin operino in un ambiente protetto e prevedibile.

Isolamento DOM & CSS
Shadow DOM previene collisioni di stile globali

Boundary Chiaro
Contratto esplicito tra Host e Plugin

Framework Agnostic
React è un dettaglio implementativo interno

Version Coexistence
Supporto multi-versione nello stesso runtime

11 / RUNTIME REGISTRY

Registry globale runtime

core/registry.ts

```ts
// 1. Inizializzazione Safe (Shared Singleton)
const REGISTRY_KEY = "__acme_plugin_registry__";
const globalScope = globalThis as any;

if (!globalScope[REGISTRY_KEY]) {
  globalScope[REGISTRY_KEY] = new Map();
}
const registry = globalScope[REGISTRY_KEY];

// 2. Registrazione (dal bundle del plugin)
registry.set("acme.analytics", {
  version: "1.2.0",
  factory: () => import("./analytics-plugin.js")
});

// 3. Risoluzione (dal runtime host)
const plugin = registry.get("acme.analytics");
```

Il Problema dei Bundle Separati

Poiché Host e Plugin vengono caricati come moduli ESM indipendenti, non condividono lo scope locale. Serve un punto di incontro universale.

Single Source of Truth

Utilizzando `globalThis`, garantiamo che tutti i moduli vedano esattamente la stessa istanza della registry.

Visibilità Cross-Bundle

Evita il problema comune dove un plugin "caricato" è invisibile al codice host a causa di istanze di React/Store duplicate.

Risoluzione Sicura

Il runtime può verificare la presenza e la versione del plugin prima di tentare il mount.

PLUGIN BUNDLE — GLOBAL SCOPE — HOST RUNTIME

12 / DISTRIBUZIONE

Due modelli di distribuzione

Self-contained

Bundle autonomo con dipendenze

VANTAGGI

- ✓ **Deploy Semplice:** Nessuna configurazione complessa lato build o runtime.
- ✓ **Isolamento Totale:** Può usare una versione di React diversa dall'host (grazie allo Shadow DOM).

SVANTAGGI

- ✗ **Bundle Size:** Include React (~40kb gzip) per ogni singolo plugin caricato.
- ✗ **Duplicazione:** Possibile spreco di memoria e rischio di "Multiple React Instances".

VS

Peer Dependency

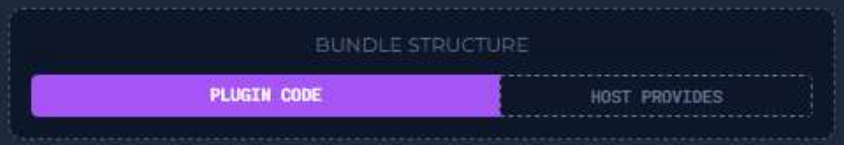
React fornito dall'Host a runtime

VANTAGGI

- ✓ **Performance:** Bundle estremamente compatto, scarica solo la logica UI.
- ✓ **Runtime Unificato:** React Context e Hooks funzionano nativamente tra host e plugin.

SVANTAGGI

- ✗ **Configurazione:** Richiede setup Webpack/Vite avanzato (Externals o Module Federation).
- ✗ **Vincoli:** Il plugin deve essere compatibile con la versione React dell'host.



13 / BENEFICI

Vantaggi Principali

L'architettura plugin-based trasforma non solo il codice, ma l'intero **ciclo di vita** del software, abilitando nuove capacità operative.

Deploy Indipendente
Rilasci di plugin istantanei senza rebuild o restart dell'host.

Estensibilità Sicura
Apertura a terze parti senza esporre il codice sorgente core.

Isolamento Runtime
Un errore nel plugin non compromette l'intera applicazione.

Scalabilità Architetturale
Sistema modulare che cresce orizzontalmente senza complessità esponenziale.

Evoluzione Disaccoppiata
Host e Plugin evolvono a velocità diverse grazie al contratto stabile.

Impatto sul Business

Velocity	High
Risk	Low
Flexibility	High

Da Monolite a Ecosistema Aperto

L'architettura abilita team distribuiti a lavorare in parallelo senza blocchi.

14 / DEMO

Demo architetturale — flow sintetico

Registry Check

Il runtime verifica la `globalThis.registry` durante la fase di **register** per evitare collisioni di ID tra plugin caricati.

DOM Isolation

Creazione del custom element e uso di `attachShadow` per garantire un boundary CSS/DOM solido tra host e plugin.

Context Prop

Il context tipizzato (User, Services) attraversa il confine React/DOM e viene iniettato nel root tramite il `PluginProvider`.

15 / USE CASES

Applicazioni Reali

Questa architettura non è solo teoria. Il modello **Runtime Composition** abilita scenari applicativi concreti impossibili con il modello statico tradizionale.

Dashboard Estensibili
Iniezione di widget sviluppati da terze parti senza ricompilare l'host.

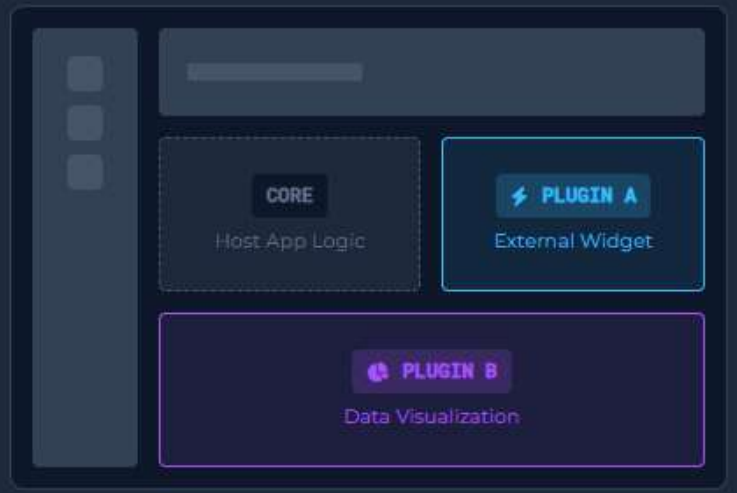
Marketplace Plugin
Discovery e installazione dinamica di estensioni (stile VS Code).

Enterprise Customization
Logiche verticali, policy e branding specifici per singolo tenant.

Feature Flags Dinamiche
Abilitazione granulare di intere sezioni di UI per utenti specifici.

Microfrontend Runtime
Scomposizione di grandi app in domini autonomi senza coupling a build-time.

Anatomia di una Dashboard Composta



L'host fornisce il layout (shell), i plugin riempiono i contenuti a runtime in base al contesto utente.

16 / ROADMAP

Evoluzioni future

Piano di sviluppo per consolidare l'ecosistema e migliorare la DX.

FASE 1 — Q3 2026

Fondamenta & DX

- **CLI Plugin Generator:** Tooling per scaffolding rapido di nuovi plugin.
- **Manifest Validation:** Controllo rigoroso dello schema e integrità a build-time.

FASE 2 — Q4 2026

Ecosistema & Trust

- **Registry Remoto:** Marketplace centralizzato per discovery e distribuzione.
- **Plugin Signing:** Firma digitale per garantire l'origine del codice.
- **Auto-Versioning:** Semantic versioning basato su analisi del contratto.

FASE 3 — 2027+

Piattaforma Estesa

- **Multi-Framework:** Supporto nativo per wrapper Angular e Vue.
- **Sandbox Tooling:** Hot-reload isolato e ambiente di test simulato.

17 / CONCLUSIONE

Takeaway

Da Applicazione a Piattaforma.

> Abbiamo ridefinito il confine architetturale. Non integriamo più codice statico, ma componiamo capacità a runtime.

Separazione dei Ruoli

L'Host diventa l'infrastruttura (Piattaforma), il Plugin diventa l'unità di valore (Estensione).

Sistema Dinamico

Un'architettura estensibile, scalabile e indipendente dal deploy centrale.

Contract-First

Il contratto tipizzato è il fondamento che garantisce un'evoluzione sicura e disaccoppiata.

18 / VISIONE FINALE

Messaggio Chiave

~~Non è una libreria.~~

È una Entità Runtime.

> Questa singola distinzione cambia completamente il modello architetturale. Non stai più costruendo un'applicazione monolitica, ma orchestrando una piattaforma vivente.

Codice Vivo

Il plugin non è un asset statico. È codice che viene caricato, eseguito e composto dinamicamente nell'ambiente dell'utente.

Composizione vs Integrazione

Passiamo dal modello "build-time integration" (rigido) al modello "runtime composition" (flessibile e scalabile).

Da App a Piattaforma

L'Host smette di essere il proprietario di tutto il codice e diventa il garante delle regole e delle capacità.