Варианты использования C/C++ кода в JS-приложении

kaspersky

Варианты использования C/C++ кода в JS-приложении

Сравниваем решения на N-API и WASM на примере реальных кейсов.

Евгений Карпов, архитектор

Игорь Карпинский, разработчик

Архитектура

Концептуальная архитектура решения

Agenda

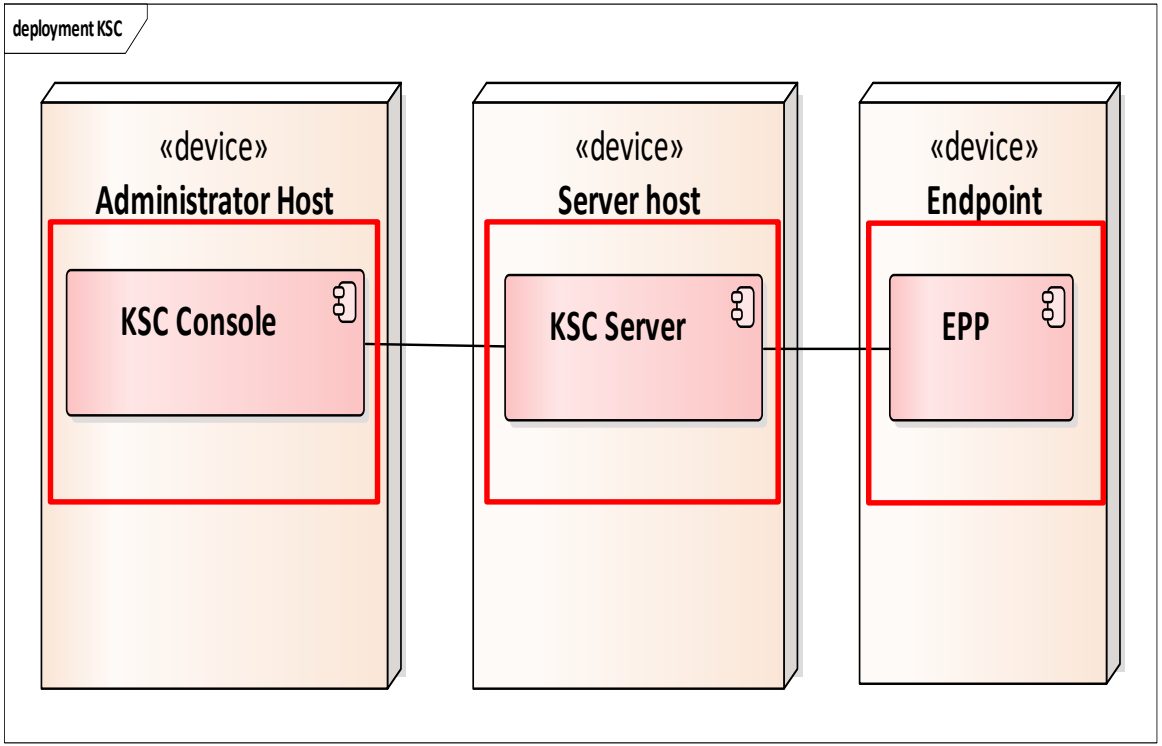
Концептуальная архитектура

Архитектура консоли управления

Архитектура консоли управления (web)

Архитектура плагина (web)

Концептуальная архитектура



Трехзвенная архитектура

- Управляемый хост
- Сервер администрирования
- Консоль администрирования

Agenda

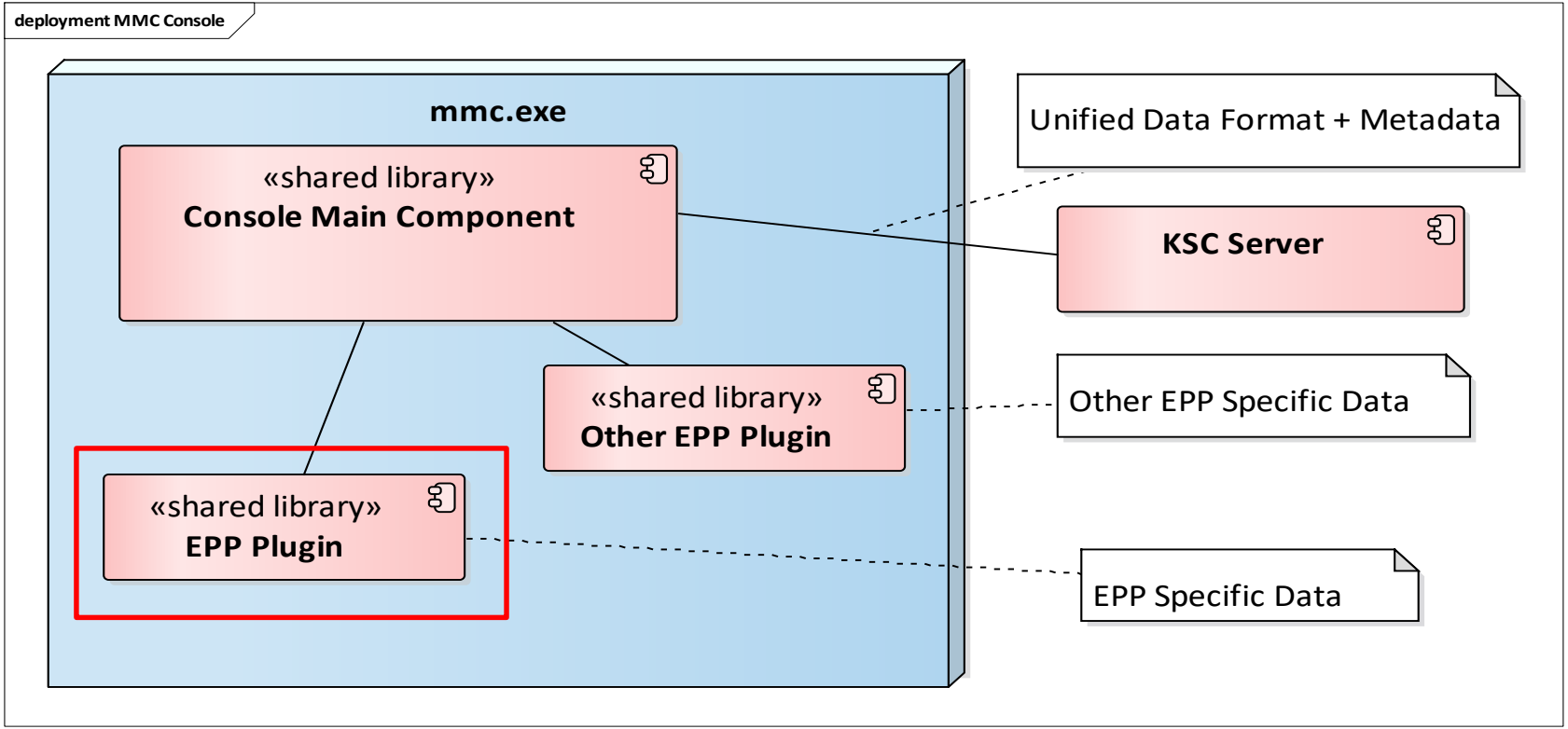
Концептуальная архитектура

Архитектура консоли управления

Архитектура консоли управления (web)

Архитектура плагина (web)

Архитектура консоли управления



Agenda

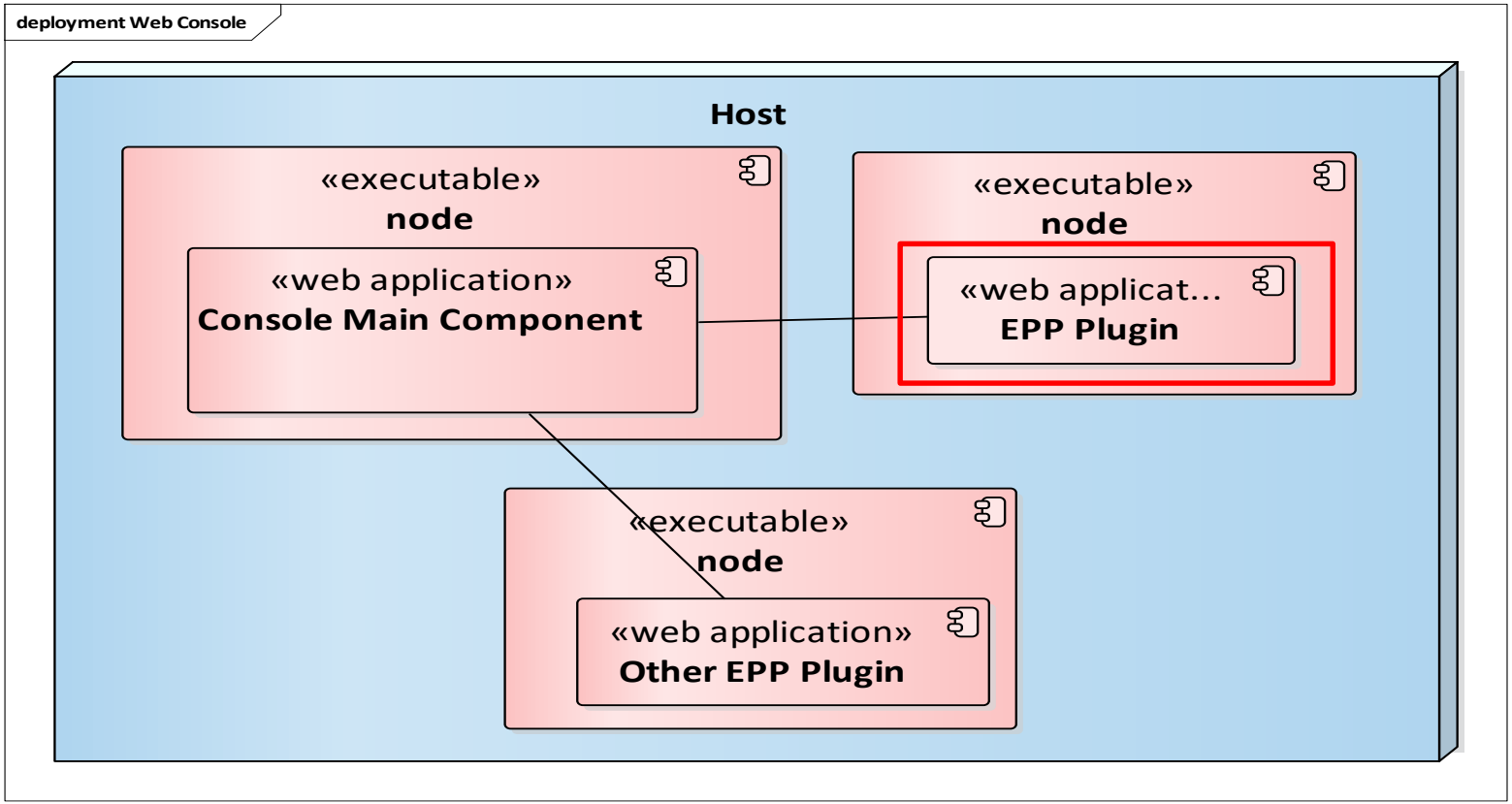
Концептуальная архитектура

Архитектура консоли управления

Архитектура консоли управления (web)

Строение плагина

Архитектура консоли управления (web)



Agenda

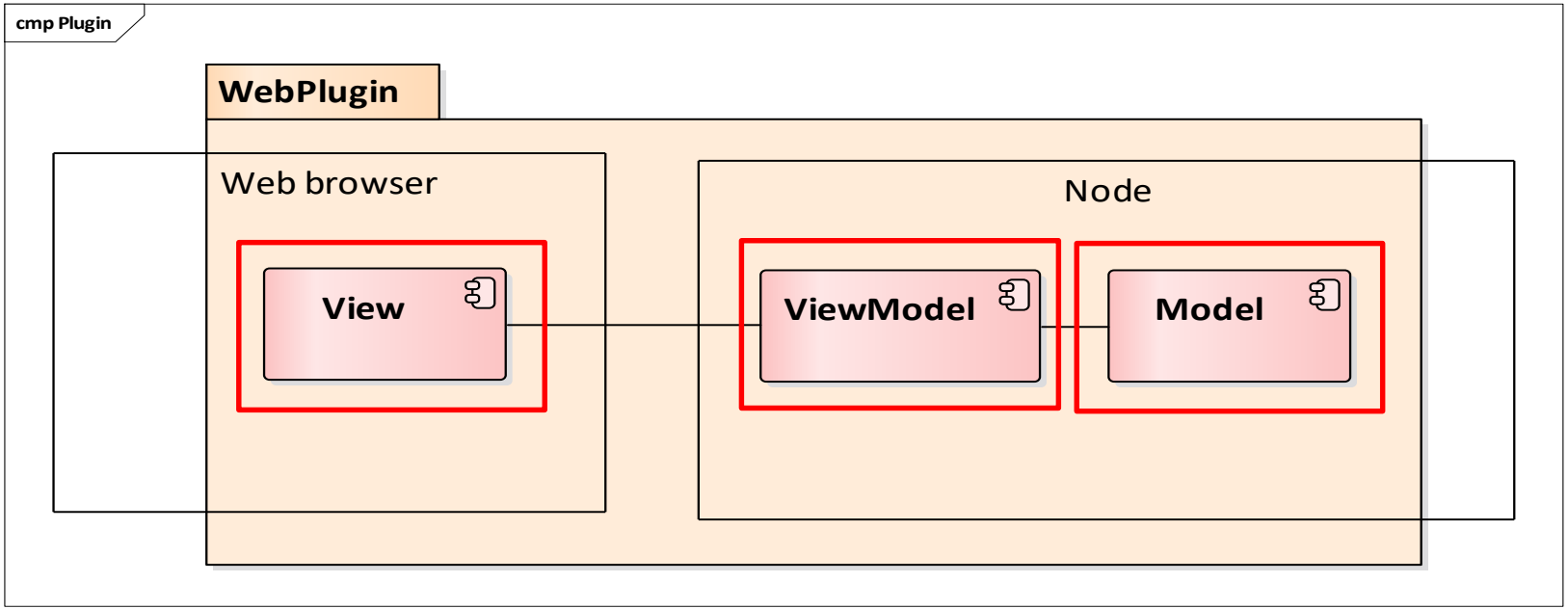
Концептуальная архитектура

Архитектура консоли управления

Архитектура консоли управления (web)

Архитектура плагина

Архитектура плагина



Задача

Постановка задачи по миграции существующего функционала на web технологии

Agenda

Постановка задачи

Решение

Почему не Wasm?

Результаты

Дано:

- Плагин на основе C++
- Функционал работы с продуктовыми крипто-контейнерами
- Бинарная сериализация

Задача:

- Адаптировать функционал для web
- Сохранить переносимость между платформами

Agenda

Постановка задачи

Решение

Почему не Wasm?

Результаты

C++ Addons (N-API)

Мостик между C++ и JS
Нет необходимости перекомпилировать существующий код на C\C++

Простой способ интеграции
require(addon.node)

Реализация
- Простой сценарный JS интерфейс
- Передача объектов между вызовами

Использование в JS коде


```
const addon = require('my_node_addon.node');
const container = addon.DeserializeChallenge(requestId, response);
const data = addon.ExtractEncryptedChallengeData(requestId, container.dataBlob);


const responseContainer = addon.CreateContainerPacket(requestId, data);
const response = addon.SerializeResponse(requestId, responseContainer);
```

Сборка

Windows:

- ОС Windows 7 и выше
- Runtime MS Visual Studio 14
- Динамическая линковка

Linux:

- Статическая линковка с c++\c++abi
- gcc 7.3.0
- glibc 2.11

https://en.wikipedia.org/wiki/Runtime_library

Agenda

Постановка задачи

Решение

Почему не Wasm?

Результаты

Отдельная сборка

Несколько динамических библиотек
Необходима сборка под новый toolset (emscripten)

Несколько команд разработки
Необходима координация сборки для нескольких команд

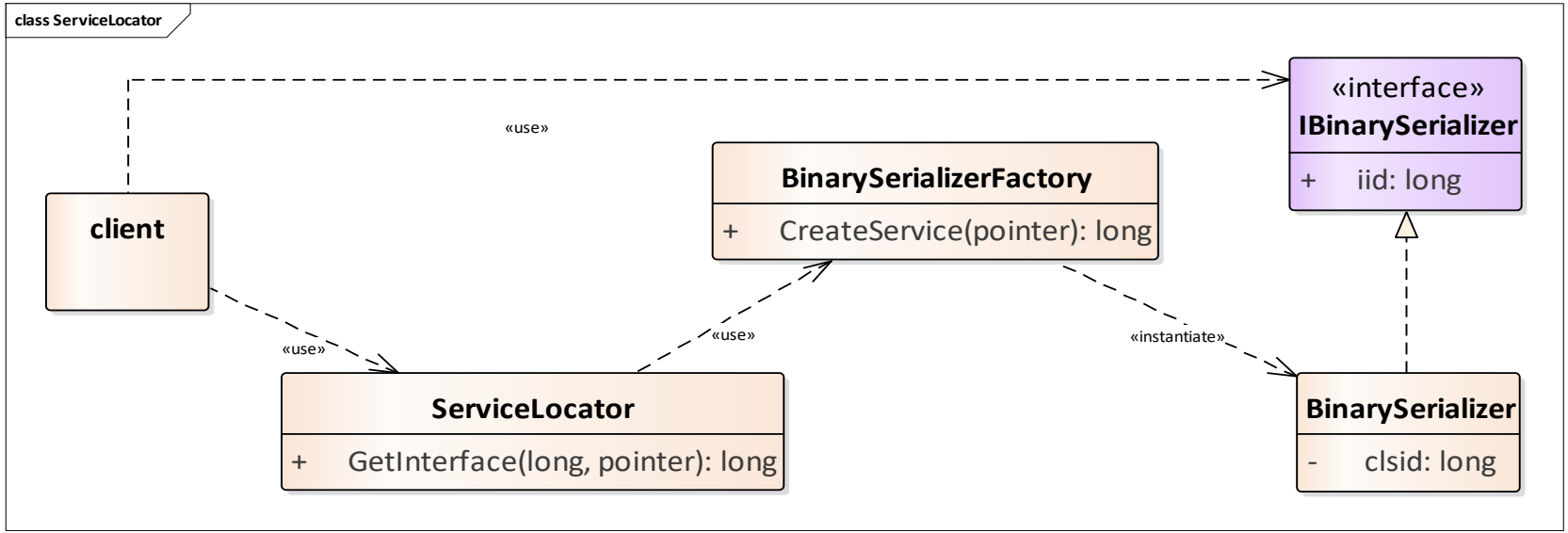
Динамическая загрузка

Модули загружаются независимо
Библиотеки не имеют явных зависимости

DI контейнер
Основа всего - интерфейс

https://github.com/WebAssembly/tool-conventions/blob/master/DynamicLinking.md
https://github.com/WebAssembly/interface-types/blob/master/proposals/interface-types/Explainer.md

Динамическая загрузка



Сроки

Необходимо исследование
Новая технология. Необходимо время на изучение.

Недостаточно сложных примеров
В наличии достаточно простые примеры

Быстрые релизы
Только адаптация функционала

Agenda

Постановка задачи

Решение

Почему не Wasm?

Результаты

- Переиспользование существующего кода
- Поддержка ОС Linux
  - Ограниченный набор ОС
  - Сборка минимум для двух toolset-ов
  - Переключаемся между платформами в runtime
- Размеры модуля
  - Несем с собой все необходимое для разных платформ
- Сбор диагностической информации

Windows

11 Мб

12 динамических библиотек

Linux

253 Мб

7 динамических библиотек

Let’s talk?

Evgeny.A.Karpov@kaspersky.com, Igor.Karpinsky@kaspersky.com

kaspersky