SOAFEE 아키텍처가 중요도가 혼재된 오토모티브 시스템에 클라우드 네이티브를 적용시키는 방법

맷 스펜서(Matt Spencer), Arm 수석 소프트웨어 아키텍트

백서

첨단 운전자 지원 시스템(ADAS), 자율 주행, 향상된 차량용 인포테인먼트(IVI) 등의 기능들이 도입되면서 최신 오토모티브 플랫폼에서 구현되는 기능 및 성능이 기하급수적으로 증가함에 따라, 자동차 제조업체들은 소프트웨어 정의 차량으로의 전환을 모색하고 있다.

이러한 전환은 미래 시장의 핵심 요소로서, 새로운 수익 창출의 기회뿐만 아니라 비용 절감을 통해 이윤을 높일 수 있는 광범위한 기회를 제공한다.

개발 및 통합 비용의 증가는 차량 내 기능 블록을 통합함으로써 관리할 수 있다. 또한 차량 모델과 세대 간 코드 재사용은 소프트웨어에 대한 초기 투자 비용 상쇄에 도움이 될 것이다.

오토모티브 도메인을 비롯한 임베디드 시스템용 코드 개발의 고질적인 문제는 소프트웨어가 선택된 특정 프로세서와 함께 제공되는 BSP에서만 사용 가능한 API로 작성되었다는 것이다.

소프트웨어 개발의 근원이 되는 BSP의 특정 API에 대한 종속성으로 인해, 한 프로세서에서 다른 프로세서로 애플리케이션 코드의 이식성(portability)이 보장되지 않는다.

이 백서는 소프트웨어 이식성(portability) 및 결합성(composability) 문제에 대해 Arm과 오토모티브 업계 최고의 기술 파트너들이 함께 구현하고 있는 솔루션을 소개한다.

개요

수많은 기능 요소로 구성된 복잡한 소프트웨어 솔루션을 안전하고 관리 가능하도록 제공하는 방법으로, 클라우드에서 실행되는 대규모 애플리케이션을 떠올릴 수 있다. 그동안 인프라 시장과 클라우드 서비스 제공업체(CSP)는 소프트웨어 개발에서 '클라우드 네이티브' 모범 사례를 채택하고, 복잡성 제한과 품질 향상을 지원하는 워크플로우 및 툴링(tooling) 구축을 통해 복잡한 소프트웨어 배포 문제를 해결해왔다.

클라우드 네이티브는 채택해야 하는 다양한 기술, 워크플로우와 설계 패턴을 정의함으로써, 프로덕션 환경에서 애플리케이션을 개발, 배포 및 업데이트하는데 수반되는 복잡성을 완화한다.

SOAFEE 프로젝트의 목표는 '기능 안전(FuSa)'과 빠르고 정확한 '실시간 제어'와 같은 오토모티브 도메인만의 과제와 제약을 해결하기 위해 클라우드 네이티브 개발 환경의 이점을 구현하는 것이다.

클라우드 네이티브의 기본 요구 사항 중 하나는 소프트웨어를 하드웨어에서 분리할 수 있어야 한다는 것이다. 기본 소프트웨어를 근본적으로 다시 설계할 필요 없이, 워크로드를 다른 하드웨어에 쉽게 배포할 수 있어야 한다. 이상적인 솔루션은 애플리케이션 코드를 재컴파일할 필요가 없는 바이너리 이식성을 구현하는 것이다.

클라우드 네이티브를 오토모티브에 적용하는 방법

'클라우드 네이티브 컴퓨팅 재단(CNCF)'은 클라우드 네이티브 배포에 사용되는 여러 툴의 사양과 구현을 관리하는 오픈 소스 재단이다. CNCF 회원들이 소유하고 동의한 클라우드 네이티브의 정의는 다음과 같다.

"클라우드 네이티브 기술은 조직이 퍼블릭, 프라이빗, 하이브리드 클라우드와 같은 현대적이고 동적인 환경에서 확장 가능한 애플리케이션을 구축 및 실행할 수 있도록 한다. 이러한 접근법의 예시로는 컨테이너, 서비스 메시(mesh), 마이크로서비스, 불변(immutable) 인프라, 선언형(declarative) API가 있다.

이러한 기술은 탄력적이고, 관리 및 관찰이 가능하도록 느슨하게 결합된 시스템을 가능하게 한다. 이를 강력한 자동화와 함께 활용하면, 엔지니어는 최소한의 노력으로도 거대한 영향을 미치는 변화를 자주, 또 예측 가능하게 수행할 수 있게 된다.

CNCF는 벤더 중립적인 오픈소스 프로젝트 에코시스템을 육성하고 유지함으로써 이 패러다임의 도입을 추진하고자 한다. CNCF는 모든 사람이 이러한 혁신에 접근할 수 있도록 최첨단 패턴을 대중화한다."

이 정의를 보면 클라우드 네이티브 솔루션을 반드시 클라우드에 구축해야 할 의무는 없다는 것을 알 수 있다. 대신, CNCF는 최첨단 설계 패턴을 기반으로 하는 컨테이너, 마이크로서비스, 선언형 API와 같은 기술의 사용을 장려하고 있다.

이러한 목표는 오토모티브 영역에서 Arm이 지향하는 바와 매우 유사하다. 이어지는 섹션에서는 일부 관련 기술들을 자세히 소개하고, 이러한 기술이 SOAFEE의 목표와 어떤 관련이 있는지 설명한다.

SOAFEE는 OCI 호환 컨테이너를 사용한다

Google이 규정한 컨테이너의 정의는 다음과 같다: *"컨테이너는 소프트웨어 서비스를 실행하는 데 필요한 특정 버전의 프로그래밍 언어 런타임 및 라이브러리와 같은 종속 항목과 애플리케이션 코드를 함께 포함하는 경량 패키지다."*

따라서, 기본적으로 컨테이너는 애플리케이션을 패키징하고 배포하는 편리한 방법이다. 컨테이너 환경은 오픈 컨테이너 이니셔티브(OCI)에 의해 정의되며, 두 개의 주요 부분과 컨테이너 레지스트리(예: hub.docker.com)와 통신하기 위한 제3의 표현 표준(expressing standards)으로 구성된다.

- 컨테이너 런타임 사양(Container Runtime Specification)
- 컨테이너 이미지 사양(Container Image Specification)
- 컨테이너 배포 사양(Container Distribution Specification)

런타임 사양은 컨테이너가 시스템에서 성공적으로 작동하기 위해 필요한 시스템 요구사항 및 인터페이스를 포함한다. 이미지 사양의 경우 컨테이너 런타임에 사용할 수 있도록 이미지를 구성하는 방법을 정의한다.

컨테이너 에코시스템에 대한 이러한 표준 기반 접근법의 장점은 컨테이너 런타임 구현의 혁신을 통해 특정 배포(deployment)의 도메인별 요구사항을 충족하는 솔루션을 만들도록 장려한다는 것이다. OCI의 경우 runc를 사용하는 컨테이너 런타임의 레퍼런스 구현을 가지고 있는 반면, 다른 도메인에 적용되는 다양한 컨테이너 런타임들도 존재한다.

다음은 이용 가능한 컨테이너 런타임의 일부 목록이다.

런타임	특성
runc	Go-lang으로 구현된 OCI의 레퍼런스 런타임
crun	C로 구현된 소규모 풋프린트 경량 런타임
gvisor	시스템 API 액세스를 제한하여 보안이 강화된 샌드박스 런타임
kata	보안 및 분리 강화를 위해 KVM을 활용하는 가상화된 런타임
runx	Xen에 구축된 가상화된 런타임
...	다른 컨테이너 런타임 이용 가능

여기서 중요한 점은 어떤 컨테이너 런타임을 사용해야 하는지에 있어서 모든 것을 만족시키는 단일 솔루션은 없다는 것이다. 그러나 확실한 것은 OCI 컨테이너 사양에 맞게 구축된 컨테이너는 플랫폼에서 수정 없이 작동된다.

SOAFEE는 이러한 애플리케이션과 런타임 간의 강력한 분리를 활용함으로써, 오토모티브 도메인의 까다로운 배포 특성을 충족하는 컨테이너 런타임을 이용 가능하게 하고, 필요한 경우 OCI 내의 업스트림 표준이 이러한 요구 사항을 표현할 수 있도록 보장하는 것을 목표로 한다.

마이크로서비스(Microservice)

마이크로서비스 아키텍처(Microservice Architecture)는 소프트웨어 디자인 패턴으로, 느슨하게 결합된 협업 서비스를 만들도록 함으로써 서비스를 함께 구성해 기능적 솔루션을 생성할 수 있도록 한다. 이러한 서비스에는 명확하게 정의된 인바운드 및 아웃바운드 인터페이스가 포함되어, 서비스가 시스템의 다른 구성 요소와 맺는 계약을 생성한다.

클라우드 네이티브 배포에서 마이크로서비스는 컨테이너 안에 캡슐화된다. 이를 통해 정의된 컨테이너 런타임 환경에서 실행할 수 있으며, 오케스트레이터가 배포를 관리 및 모니터링할 수 있다. 자세한 내용은 추후 설명할 예정이다.

인바운드 및 아웃바운드 API의 계약에서 정의된 예상 동작이 지켜지는 한, 특정 서비스에 대한 변경이 시스템 내 다른 서비스 성능에 영향을 미치지 않아야 하기 때문에, 마이크로서비스는 느슨하게 결합되도록 정의된다. 또한 이러한 특성은 마이크로서비스가 시스템의 나머지 부분과 분리되어 테스트될 수 있음을 의미하고, 완전히 구성된 시스템의 통합 테스팅을 진행하기 전, 크고 복잡한 시스템을 개별 서비스에 대한 단위 테스팅으로 분리할 수 있게 된다.

오케스트레이터(Orchestrator)

오케스트레이터(Orchestrator)는 클라우드 네이티브 시스템의 핵심 구성 요소로, 마이크로서비스 기반 솔루션의 구성, 배포 및 모니터링을 관리한다. 오케스트레이터 자체는 여러 표준 인터페이스로 구성된다.

인터페이스	설명
CRI	오케스트레이터와 컨테이너 런타임 간의 인터페이스
CNI	컨테이너 네트워크 인터페이스(Container Network Interface): 네트워크, 방화벽 등을 구성하고 제어하기 위한 표준 메커니즘
CSI	컨테이너 스토리지 인터페이스(Container Storage Interface): 컨테이너 인스턴스에서 사용할 수 있도록 스토리지를 노출하는 방법
디바이스 플러그인	/dev/video0과 같은 컨테이너 내의 시스템 리소스에 대한 관리 액세스 활성화
...	적용 가능한 다른 표준이 있다

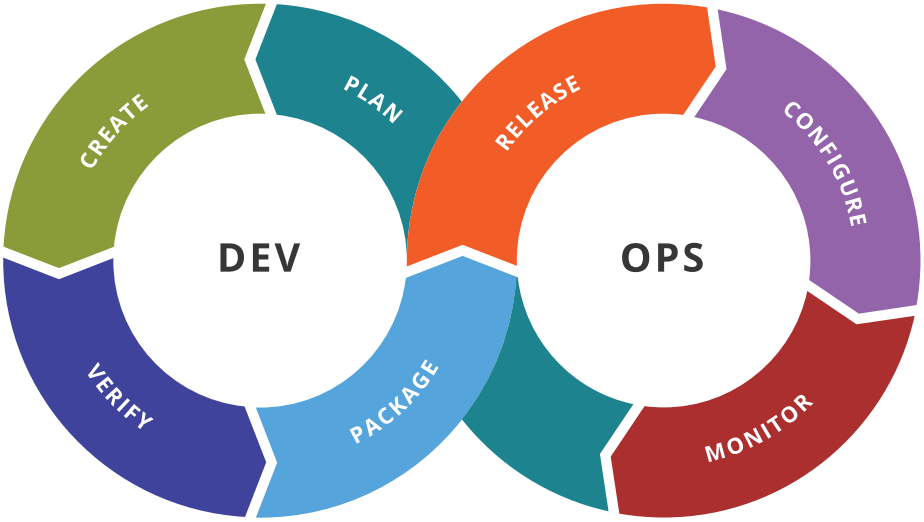
클라우드 네이티브 에코시스템의 모든 특징과 마찬가지로, 사용 사례별 요구 사항을 충족하는 이러한 표준 인터페이스의 여러 가지 구현이 존재한다. 컨테이너 런타임을 오케스트레이터로 관리하려면 CRI 인터페이스를 구현해야 하며, 이 백서의 앞부분에서 설명한 모든 런타임이 해당 요구 사항을 준수해야 한다.

오케스트레이터의 제어 하에서 이러한 인터페이스가 합쳐지면, 마이크로서비스 간의 통신과 올바른 작동에 필요한 데이터 소스에 액세스할 수 있도록 네트워크를 구성함으로써 복잡한 애플리케이션 배포를 관리할 수 있다.

오케스트레이터에는 여러 가지 옵션이 있으며, 기본값은 쿠버네티스(kubernetes, 줄여서 k8s라고도 한다)이지만 임베디드 및 리소스 제약 환경에 더 적합한 k3s 와 같이 더 작은 풋프린트로 구현하는 것도 가능하다.

데브옵스(DevOps)

클라우드 네이티브의 워크플로우 측면을 보통 데브옵스(DevOps) 프로세스라고 한다.



출처: commons.wikimedia.org/wiki/File:Devops-toolchain.svg

워크플로우는 두 가지 주요 부분으로 나뉜다. Dev(개발)는 개발 워크플로우를 다루고 Ops(운영)는 배포의 운영 측면을 다룬다. 이 두 분야를 명확하게 정의 및 관리되는 방식으로 통합함으로써 이 워크플로우에서 관리되는 애플리케이션의 개발, 배포를 간소화하고 지속적으로 개선할 수 있다.

위 그림에서 보이는 바와 같이, 배포 시 컨테이너 작업 모니터링의 출력이 다음 단계 개발 주기로 다시 이어지는 지속적인 프로세스라는 것을 확인할 수 있다.

이는 전달된 워크로드의 품질이 지속적으로 개선되고 있음을 보여준다. 이 프로세스를 통해, 시간이 지남에 따라 시장 출시 기간과 전체 비용을 줄이는 동시에 전반적인 품질을 높일 수 있다.

SOAFEE의 클라우드 네이티브 확장 방법

SOAFEE 프로젝트는 사용되는 모범 사례와 표준의 이점을 얻기 위해 클라우드 네이티브 프레임워크를 활용하는데, 문제는 오토모티브 솔루션으로 작업 시에는 추가적인 요구 사항과 제약 조건이 수반된다는 것이다. 여기에는 다양한 액셀러레이터를 사용할 수 있는 실시간(real time) 프로세서와 애플리케이션 프로세서가 결합된 이기종 컴퓨팅 아키텍처에 워크로드를 배포하는 기능이 포함된다.

'SOAFEE 워킹 그룹(Working Groups)'을 통해, SOAFEE는 클라우드 네이티브 구현에 있어 현재 존재하는 격차를 이해하고 관련 표준화 기관 내에서 업스트림 작업을 하고자 한다. 이러한 격차를 좁히기 위해 협력함으로써 클라우드 네이티브 솔루션을 오토모티브 및 안전 관련된 영역에도 적용할 수 있게 될 것이다.

오케스트레이터 개선 - 안전 및 실시간(Real time) 요구사항

오케스트레이터 스케줄링 프레임워크는 아래 열거된 표준 기술을 사용해 이러한 추가적인 요구 사항을 충족시킬 수 있는 특성들을 갖추고 있다:

- 노드 어피니티(node affinity)를 통해 워크로드가 배포될 위치 제한
- 테인트(Taints)와 톨러레이션(Tolerations)을 통해 특정 노드가 서로를 끌어당기거나 밀어내는 방법 설명
- 파드 오버헤드(pod overhead)를 통해 특정 워크로드에서 소비될 시스템 리소스의 양 계산

그러나 현재의 시스템 구현과 요구사항 간에는 격차가 존재하므로, SOAFEE는 표준 커뮤니티와 업스트림 작업을 통해 실시간 및 안전 요구 사항을 설명하는 언어를 표준화할 계획이다.

예시 - 실시간 요구 사항

- 필요한 I/O 대역폭
- 실행 시간 보장
- 캐시 정책

예시 - 안전 요구 사항

- FFI(Freedom from interference)
- 스플릿-락(Split-lock) 코어
- 가용성(Availability)

각 SOAFEE 기술 워킹 그룹은 이러한 확장 영역에 대한 작업을 진행해 오케스트레이터의 목적을 전달하는 데 사용할 수 있는 공통 언어를 고안해 낼 것이다. 이를 통해, 기본 시스템에서 적합한 부분에 워크로드를 배포하고 각 파드(Pod)에 대해 하위 레벨 아키텍처 기능을 구성해 이러한 요구 사항을 충족할 수 있게 될 것이다.

컨테이너 런타임 개선

이제 오케스트레이터가 워크로드의 추가 런타임 요구 사항을 적절하게 표현할 수 있으므로, 이러한 요구 사항을 충족할 수 있도록 컨테이너 런타임을 향상시키는 작업이 수행된다. 주요 경로 제안은 앞서 언급한 runx와 같은 가상화 된 컨테이너 런타임을 사용해 런타임 자체와 협력하여 VMM을 활성화함으로써, VMM을 통해 특권(privileged) 시스템 리소스의 제어를 컨테이너 런타임에서 하위 권한(lower privileged)의 실행 환경과 분리하는 것이다.

SOAFEE 워킹 그룹은 올바른 초기 실행 환경을 선택한 다음, 수정이 필요할 때 OCI 표준화 기관과 함께 업스트림 작업을 할 뿐만 아니라 향상된 기능을 구현하기 위해 선택한 컨테이너 런타임 작업도 담당하게 된다.

이식 가능한(portable) 워크로드

SOAFEE가 제안하는 핵심 가치는 워크로드의 재사용이다. 최종 애플리케이션을 구성하는 특정 마이크로서비스가 수정 없이 다양한 제품 라인과 솔루션에서 재사용될 수 있도록 함으로써, 이처럼 복잡한 소프트웨어 솔루션의 배포 비용을 줄일 수 있다.

이를 달성하기 위해서는, 액셀러레이터 또는 IO 디바이스를 특정 아키텍처별로 구현할 시, 재컴파일할 필요 없이 워크로드에 대한 일관된 액세스를 액셀러레이터와 고대역폭 IO 디바이스에 부여할 수 있는 방법을 이해해야 한다. 이 개념을 이해하기 위해서는 기능 안전 및 실시간과 관련된 도메인별 요구 사항을 고려해야 한다.

예를 들어, VirtIO라는 업계 표준이 있다. VirtIO는 액셀러레이터에 대한 반가상화(para-virtualised) 액세스를 제공해 액셀러레이터의 워크로드 뷰(workloads view)를 효과적으로 표준화하는 동시에 백엔드 액셀러레이터로의 효율적인 오프로드를 가능하게 한다.

표면적으로 VirtIO는 이식성 문제에 대한 완벽한 솔루션처럼 보이지만, 현재 출시된 VirtIO에는 몇 가지 문제가 있다. 첫째, VirtIO는 기능 안전이나 실시간 워크로드를 염두에 두고 설계되지 않았다. 둘째, 인터페이스가 오토모티브 도메인의 모든 요구 사항을 다루고 있지 않다. 예를 들어, TVM과 같은 사용자 공간을 통한 머신러닝 가속을 위한 VirtIO 인터페이스가 없으며 CAN 버스와 같은 오토모티브 전용 IO 디바이스에 대한 표준 인터페이스도 없다.

SOAFEE는 에코시스템 단편화를 야기할 수 있는 새로운 표준을 만들기 위함이 아닌, 기존 표준의 채택을 우선시하고 기능적으로 안전한 도메인에서 목적에 맞게 조정하므로, 문제를 해결하기 위해 VirtIO 표준화 기구 내에서 업스트림 작업을 진행할 것이다. 여기에는 Virtqueue에 대한 실시간 제약을 표현할 수 있게 하는 작업 항목과, 누락된 인터페이스에 대한 적합한 VirtIO 구현을 정의하기 위한 업계의 협력이 포함된다.

테스트 및 검증

마이크로서비스 기반 솔루션을 구축함으로써 얻을 수 있는 한 가지 좋은 기회는 구성요소와 시스템의 수준 테스트 및 검증에 CI/CD용 툴링(tooling)이 가능하다는 것이다. 예를 들어 SOAFEE의 워크로드 이식성 기능을 사용해 클라우드에서 워크로드 훈련 및 테스팅을 실행함으로써 워크로드 성능에 대한 일차적 신뢰를 줄 수 있다. 그다음 동일한 워크로드를 연구실 기반 인프라에 배포해 루프 검증(loop validation)에서 소프트웨어와 하드웨어를 모두 적용할 수 있게 된다.

앞서 소개한 표준 데브옵스 워크플로우는 클라우드 네이티브 배포가 복잡한 워크로드의 품질을 관리하는 방법을 제시하고, SOAFEE는 이 워크플로우의 채택과 강화를 가능하게 한다. SOAFEE 프로젝트는 이 기능을 실현하기 위해 시스템 통합 업체(SI), 툴 벤더 및 CSP와 협력하고 있다.

이렇듯 매우 중요한 SOAFEE의 역할에 대한 자세한 내용이 곧 공개될 예정이다.

오픈 소스 레퍼런스 구현

이러한 모든 노력은 프로젝트의 클라우드 네이티브 비전을 실현하는 데 필요한 모든 구성 요소를 포함하는 SOAFEE 요구 사항의 오픈 소스 레퍼런스 구현의 형태로 통합될 것이다. 레퍼런스는 기본 플랫폼을 대체 하드웨어로 포팅(porting) 할 수 있는 Yocto 레시피의 형태로 제공된다.

또한 이 레퍼런스 구현은 Autoware, AGL 등의 업스트림 중심 소프트웨어 스택에서 사용될 수 있으며, SOAFEE 아키텍처를 구현하는 모든 플랫폼에서 실행되는, 이식성이 뛰어나고 컨테이너화된 마이크로서비스 기반 구현을 가능하게 한다.

현재 SOAFEE 스택을 구축하는 방법과 하드웨어 지원에 대한 내용은 SOAFEE 프로젝트 페이지에서 확인할 수 있다. 자세한 내용은 gitlab.arm.com/soafee 에서 확인할 수 있다.

결론

SOAFEE 프로젝트는 소프트웨어 정의 차량을 위한 새로운 개방형 표준 기반 아키텍처를 정의하기 위해 자동차 제조업체, 반도체 및 클라우드 기술의 선도 업체를 한데 모았다. 이를 통해 컨테이너 오케스트레이션과 같은 클라우드 개념을 처음으로 오토모티브의 기능 안전과 결합할 수 있는 레퍼런스 구현을 제공한다. 이 프로젝트는 Arm 아키텍처에 대한 표준 부팅 및 보안 요구 사항을 정의하는 Arm 이니셔티브인 Project Cassini의 성공을 기반으로 구축되었다.

클라우드 네이티브 기술을 도입함으로써, Arm은 루프(loop) 내 소프트웨어 및 하드웨어와 같은 고급 CI/CD 기술과 교육 및 검증을 위한 클라우드 기반 인프라 사용을 가능하게 했다.

SOAFEE는 소프트웨어 정의 차량의 복잡성을 줄이는 동시에 개발 및 배포 비용을 줄일 수 있게 해준다. 또한 재통합할 필요 없이 기존 워크로드를 새로운 아키텍처에 배포할 수 있게 함으로써 소프트웨어에 대한 에코시스템의 투자를 최대한 재사용할 수 있다. 보다 자세한 내용은 Arm 웹사이트에서 확인할 수 있다.

용어 사전

용어	의미
AD	자율주행(Autonomous Drive)
ADAS	첨단 운전자 지원 시스템(Advanced Driver-Assistance Systems)
AGL	오토모티브 그레이드 리눅스(Automotive Grade Linux)
API	애플리케이션 프로그래밍 인터페이스(Application Programming Interface)
BSP	보드 서포트 패키지(Board Support Package)
CI/CD	지속적 통합(Continual Integration)/지속적 배포(Continual Deployment)
CNCF	클라우드 네이티브 컴퓨팅 파운데이션(Cloud-Native Computing Foundation)
CSP	클라우드 서비스 제공업체(Cloud Service Providers)
IVI	차량용 인포테인먼트(In Vehicle Infotainment)
OCI	오픈 컨테이너 이니셔티브(Open Container Initiative)

arm