# FaaS における共通コンテナイメージを考慮したプロビジョニング手法

蓮原裕太[1]　菅谷みどり[1]

**概要**：FaaS はインフラ管理を抽象化と，イベント駆動による高いコスト効率を提供する．しかし，リクエスト後に実行環境を起動するコールドスタートが深刻な遅延の原因となる．特にトラフィック急増時には，コンテナイメージの大量プルがボトルネックとなり，応答性がさらに低下する．従来のスケジューリングやプリヒーティングなどの対策は，課金対象となり FaaS の従量課金の利点を損なうか，従来モデルの CaaS と比較して，FaaS に特化したものではない．本研究は，この課題をサーバレスの範囲内で克服するため，FaaS の特性である共通コンテナイメージを活用したプロビジョニング手法を提案する．具体的には，関数毎ではなく，コンテナイメージごとのプリウォーミングと，関数実行後のアイドル時間を短縮により，メモリ効率を維持しつつ，関数間でコンテナイメージのキャッシュを可能にする．これにより，コールドスタートによる遅延を，コンテナへの関数配置のみに限定し，パフォーマンスとリソース効率の両立を図る．

**キーワード**：FaaS, Provisioning, Container, AWS Lambda, OpenWhisk, Pre-warming

## 1. はじめに

サーバレスコンピューティングの登場により、開発者は従来のインフラ管理から解放され，アプリケーションの実装やビジネスロジックに専念可能となった．

これは，クラウドプロバイダが，ハードウェアリソースや OS，ミドルウェア等のインフラ管理をすべて代行するためである．

サーバレスで提供されるサービスの中でも，特に注目されているのが Function as a Service (FaaS) である．HTTP リクエストに代表されるイベントをトリガーとしてアプリケーション(関数)を実行するイベント駆動と，OS やミドルウェア，ランタイムの管理といったインフラ管理の抽象化といった特徴がある．

イベント駆動による予測不可能な負荷に対するリソースの弾力性と開発の容易さを理由に FaaS の人気は高まっており，すでに商用クラウド(AWS Lambda, Azure Functions, GCP Cloud Run functions, Cloudflare Workers) やオープンソース (OpenWhisk, Knative, OpenFaaS) が広く普及している．

## 2. FaaS の概要

FaaS (参考:AWS Lambda, OpenWhisk, Azure Functions) の挙動について説明し，そこから発生する FaaS の課題について説明する

### 2.1 FaaS におけるランタイムのライフサイクル

#### 2.1.1 ライフサイクルの概要

図 1 に示すように, Control Plane はエンドユーザからの関数呼び出しリクエスト(イベント)を受けると，FaaS Runtime にコンテナの初期化と関数の配置といったランタイムの初期化を指示する．

関数の実行準備が完了すると，割り当てられたイベントを受け取り，関数を実行する．実行後はアイドル時間が設けられており，コントロールプレーンがシャットダウンを決定するまではランタイムの初期化なしに関数を実行可能である．

#### 2.1.2 FaaS の課題

FaaS はイベント駆動型ゆえに課題が存在する．実際に関数が実行されるにはコンテナの初期化と関数の初期化(コールドスタート)が必要となり遅延が発生する．

また, FaaS ワークロードはバーストトラフィックを含み，非常に動的であり[2]，FaaS プラットフォームはリクエストの急増に対応するために，多くのランタイムを新たにプロビジョニングする．

結果として，バーストトラフィックによって，同じコンテナイメージがコンテナレジストリから大量にプルされることで，ネットワークとストレージがボトルネックとなり応答性が低下する[3].

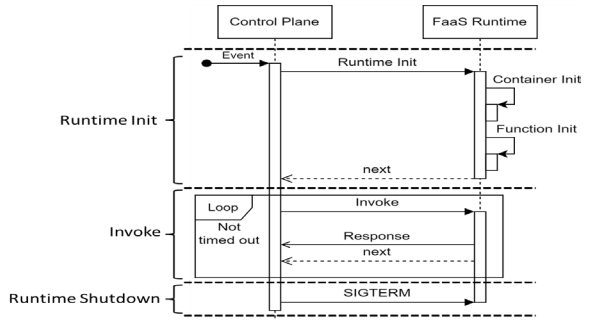


図1.　関数実行におけるランタイムのライフサイクル概要

## 3. 先行研究と本研究における課題

ここでは，先行研究を取り上げ，コールドスタート削減手法の方向性を説明し，その後本研究の課題について述べる．

### 3.1 先行研究の方向性

先行研究では, FaaS のコールドスタートを削減するため，主に以下の手法が用いられている．

1 芝浦工業大学
Shibaura Institute of Technology, Koto, Tokyo 3-7-5, Japan

- 局所性を考慮したスケジューリング:一度実行された関数を同じノードで実行し，キャッシュを再利用することで遅延を削減する[4].
- プリベーキング:コンテナ起動後，関数実行直前までの状態をスナップショットとして保存し，そこから復元することで，言語ランタイムの起動遅延などを削減する[5].
- プリウォーミング:イベント到着前にコンテナと言語ランタイムを事前に起動して準備しておく手法[6]．一部のプラットフォームでは課金対象として提供される．

### 3.2 従来コンピューティングモデルとの相違点

先行研究では，様々な手法で FaaS が持つコールドスタートによる遅延が改善されているが，コールドスタートニヨル遅延の多くは，コンテナを用いた従来のコンピューティングモデルである CaaS (Container as a Service) における課題と同じである．

CaaS は開発者が OS やミドルウェア，ランタイム，コンテナに割り当てられるハードウェアリソースをあらかじめ決定し，その上でアプリケーションを動作させる．このとき，コンテナイメージはアプリケーション専用にビルドされ，レジストリに保存される．そのため，アプリケーションごとのイメージが大量に保存されることとなる．

そのため，CaaS においてもコンテナイメージのプルによる遅延といったコールドスタートの課題を抱えている[7].

一方で，プラットフォームに依存するが CaaS はアプリケーションごとにコンテナイメージを作成するのに対して，FaaS はあらかじめプロバイダが用意したコンテナイメージを共有し，アプリケーション(関数)を動作させるという違いがある．

### 3.3 本研究における課題

GCP Cloud Run や Knative, OpenFaaS といった FaaS プラットフォームは，その中で kubernetes を使用しており，FaaS がコンテナを使用するうえで，CaaS から進化してきたことがわかる．先行研究も，FaaS と CaaS が持つ差異について注目できておらず，あくまでもコンテナのプロビジョニングは，CaaS と同じようにアプリケーションごとである．

また，プリベーキングやプロウォーミングといった手法は，プラットフォームによっては用意されているが，課金対象であり，サーバレスの文脈から逸脱する．

このことから，既存プラットフォームと先行研究においては，CaaS では不可能であり FaaS に特化した方法や，サーバレスの範疇でのコールドスタートという課題解決がなされていない．

## 4. 提案

本研究はサーバレスの範疇で FaaS 特性に基づいたコールドスタートの改善を目的として，共通コンテナイメージを考慮したプロビジョニング手法を提案する．

### 4.1 提案概要

本研究では，関数間で共通のコンテナイメージを使用する場合に，コンテナイメージごとのプリヒーティングを実施することで，従来よりも一つの関数当たりのメモリフットプリントを削減するとともに，コンテナを起動していることによる，関数間にわたったコンテナイメージのキャッシュを可能にする．

メモリフットプリントの削減と遅延削減を両立させるために，単にコンテナを冗長化するのではなく，関数実行後のアイドル時間を削減する．これにより，メモリフットプリントを抑えながらコールドスタートをプリウォーミング済みのコンテナに関数を配置するだけに留め，遅延削減を達成する．

また，本提案手法が OpenWhisk のように最大同時実行数が小さく，イベントをキューに待機させるプラットフォームと，AWS Lambda のように実質的に最大同時実行数を持たず積極的にコンテナをプロビジョニングするプラットフォームのどちらに適用可能かについても調査する．OpenWhisk がリソースの限られた環境も想定されるのに対し, Lambda が潤沢なクラウド環境を想定していることから，これら二つのプラットフォームをシミュレートすることで，現実的な環境への適用可能性を議論する．

## 5. 結論

本研究では，FaaS の問題を改善するために，共通コンテナイメージを考慮したプロビジョニング手法を提示した．今後はこれらの実装・評価を行う．

## 参考文献


[1] Liang Wang, et al., “Peeking Behind the Curtains of Serverless Platforms.” In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 133–146,

[2] Mohammad Shahrad, et al., “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider,” in 2020 USENIX Annual Technical Conference (USENIX ATC 20), pages. 205-218, 2020.

[3] M. Littley *et al*., "Bolt: Towards a Scalable Docker Registry via Hyperconvergence," *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, 2019, pp. 358-366,

[4] Alexander Fuerst, et al.. Locality-aware Load-Balancing For Serverless Clusters. HPDC '22: The 31st International Symposium on High-Performance Parallel and Distributed Computing. pp. 227-239, 27 June 2022

[5] Paulo Silva, et al., “Prebaking Functions to Warm the Serverless Cold Start,” Middleware '20: Proceedings of the 21st International Middleware Conference, pages. 1-13, 2020

[6] T. Y. Htet, et al., "Pre-warming: Alleviating Cold Start Occurrences on Cloud-based Serverless Platforms," *2024 IEEE 10th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, Shanghai, China, 2024, pp. 66-72,

[7] R. Kolodziejczyk, et al., "Containers on the Move: An Experimental Analysis of Container Migration in Kubernetes," *ICC 2025 - IEEE International Conference on Communications*, Montreal, QC, Canada, 2025, pp. 1-7,