业界 Serverless 架构调研

李海峰

December 29, 2022

本调研旨在根据公开资料搜集业界 Serverless 实现架构，以了解其实现方式、遇到的困难等，为我们设计实现 Serverless 平台提供参考。

Contents

1 概述

1.1 主要的 Serverless 产品

自 2014 年 AWS 发布 Lambda[11] 以来已经过了近 10 年时间，各个产商都相继推出了自己的 Serverless 产品，Serverless 相关技术和框架也逐渐趋于成熟。从服务模式来看，Serverless 主要有两方面的应用：一类是主要面向公有云市场的 Serverless 产品，还有一类是面向企业内部应用的私有云 Serverless 平台。

Serverless 根据其实现方式而言又可以分为以下几类 [18]:

- **函数即服务（FaaS）**: 这类产品的特点是用户只需要编写函数代码即可，如 table 1所示
- **后端即服务（BaaS）**：例如 AWS S3、DynamoDB 等 [26][1]
- **容器即服务（CaaS）**: 用户可以自行打包镜像运行，这类产品如 table 2所示

1.2 Serverless 产商服务能力排名

根据评测机构 Forrester 的评测，可以大致看出各个 Serverless 产商的竞争力排名，如 fig. 1所示。其中，Amazon、Microsoft、阿里巴巴几家公司处于领先的地位。

1.3 Serverless 技术实现的核心问题

Serverless 技术在实现上，一方面需要解决 Serverless 理念自身的技术要求所带来的挑战，另一方面，在落地实际业务时，也会收到实际业务影响而带来一些其他的问题。总结来看，在 Serverless 技术实现上，需要解决的核心问题有：

[1]此类应用与我们的实际需求差别较大，因此不在本文研究范围之内

产商	产品	发布时间	类型
Amazon	AWS Lambda	2014/03	公有云
Google	Cloud Functions	2017	公有云
Microsoft	Azure Functions	2016/03	公有云
IBM	IBM Cloud Functions	2016/02	公有云
Oracle	Oracle Functions	2019/08	公有云
Nimbella	Nimbella Platform	N/A	公有云
阿里巴巴	函数计算	2017/04	公有云
华为	函数工作流 FunctionGraph	2021/07	公有云
腾讯	云函数 SCF	2020/03	公有云
百度	函数计算 CFC	2017/11	公有云
字节跳动	ByteFaaS	2019/?	私有云
美团	Nest	2019/?	私有云

Table 1: 主要的 FaaS 提供商和产品

- **弹性伸缩**：弹性伸缩是 Serverless 的基础，如何能够响应近乎“无限”的请求、根据需要自动伸缩，是 Serverless 平台最核心的要素之一。它又可以拆解为两个子问题：基础架构和伸缩算法。
- **冷启动优化**：由于 Serverless 一般按照时长计费，且需要支持极度的伸缩，因此，决定其伸缩效率的关键因素是启动时间。如何降低启动时间是一个需要解决的问题，该问题一般又可以等价于冷启动优化问题。
- **高可用保障**：保障平台在突发流量增长时能够从容应对、容忍临时软硬件故障
- **应用部署**：
- **安全性**：FaaS 需要运行用户代码，如何进行安全隔离、保证函数调用之间互不影响，是非常重要的问题

产商	产品	发布时间	类型
Amazon	AWS Fargate	2019	公有云
Google	Cloud Run	2019	公有云
Microsoft	Azure Container Apps	2019	公有云
IBM	Code Engine	2016/02	公有云
阿里巴巴	Serverless 应用引擎 SAE	2017/04	公有云

Table 2: 一些 CaaS 应用产品

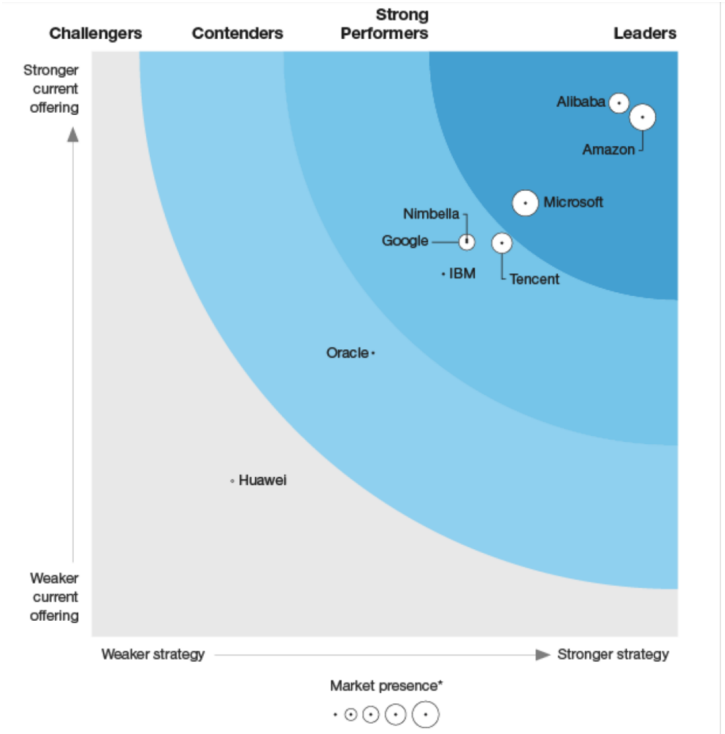

Figure 1: 《Forrester Wave™：函数即服务 (FaaS) 平台，2021 年第一季度》评估结果 [20]

- **整合现有系统**:

1.4 AWS Lambda 的演进

各个 Serverless 平台都是在不断完善和改进的，AWS Lambda 在 Serverless 领域耕耘最久、积累最丰富，因此 Lambda 的功能、架构演进十分具有参考价值。AWS Lambda 从发布之初到现在也经历过许多变化，包含新功能的支持、架构的调整等，如 fig. 2所示。

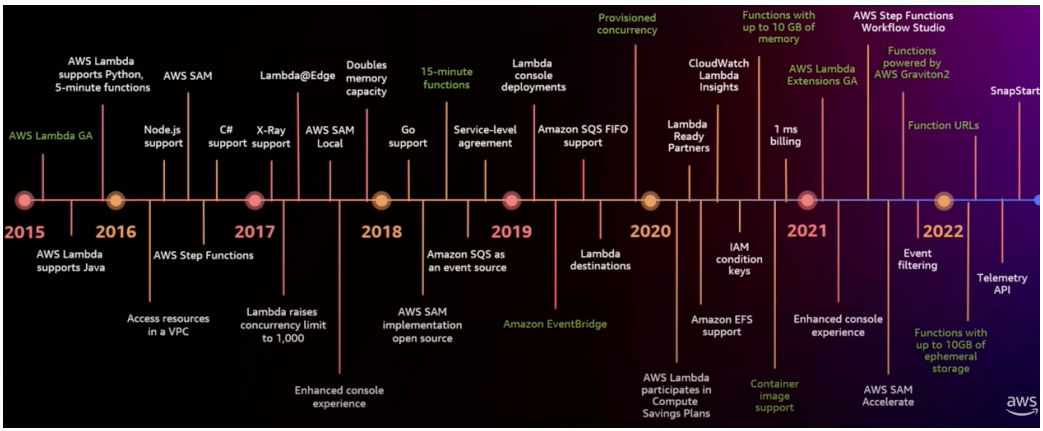

Figure 2: AWS Lambda 演进历史 [4]

2 Serverless 运行基础

2.1 底层虚拟化

基础设施是指 Serverless 底层的运行基础，需要支持虚拟化/资源隔离/扩缩容等，部分产品的架构选型如 table 3所示。

2.1.1 基于 Kubernetes

Kubernetes 已经是事实上的云原生标准，因此使用 Kubernetes 作为底层运行时是一个最常规的选择，且其本身自带了容器编排、服务治理等功能，生态较好。Knative 是谷歌开源的基于 Kubernetes 的 Serverless 运行环境。因此有不少 Serverless 实现是使用 Kubernetes 作为运行时，或者在此基础之上的 Knative，例如美团、字节以及谷歌等。

基础架构	产品
Kubernetes	美团 Nest[26, 27]、字节 ByteFaaS[21]、Nimbella[15]、京东 Serverless[24, 23]
Docker	阿里函数计算 [31][1]、Oracle Functions[17]
Knative	Google Cloud Run[12]、工商银行 Serverless[22][1]、滴滴 Serverless[19]
Firecracker/microVM	AWS Lambda、AWS Fargate[8]、腾讯云函数 [28, 30][2]
Apache OpenWhisk	IBM Cloud Functions[9]
EasyFaaS	百度函数计算 CFC[25]、工商银行 Serverless[22]

[1] 阿里函数计算第二代架构已经改为用神龙裸金属 + 安全容器实现 [31]

[2] 腾讯云使用自研的轻量级虚拟化技术，不确定是否与 AWS microVM 相同

[3] 工商银行函数计算 1.0 使用 Knative，2.0 改用百度函数计算产品

Table 3: 一些 Serverless 平台的基础设施

2.1.2 基于裸金属 + 虚拟化技术

使用裸金属服务器结合虚拟化技术（如安全容器或者 MicroVM），例如 AWS Lambda 和阿里函数计算等。相对于 Kubernetes 而言，这种方式弱化了集群管理的功能，而将工作负载的管理交由 Serverless 平台处理，更直接一些。如 AWS Lambda 使用 Firecraker/microVM 实现工作负载，如 fig. 3所示。

除了 AWS，阿里、腾讯等均采取了裸金属服务器 + 轻量级虚拟化技术的方式，其中虚拟化技术又分为 microVM 或者安全容器两类。除此之外，还有基于普通 Docker 实现的，例如基于 Docker 的 FaaS 平台（如 Fn Project）。

2.1.3 混合使用 Kubernetes 及裸金属服务器

除了上述的做法外，还有一些框架也有支持多种运行时的，例如 Apache OpenWhisk、百度开源的 EasyFaaS 等，采用框架封装支持多种运行时，可以混合使用 Kubernetes 以及 Docker 等。

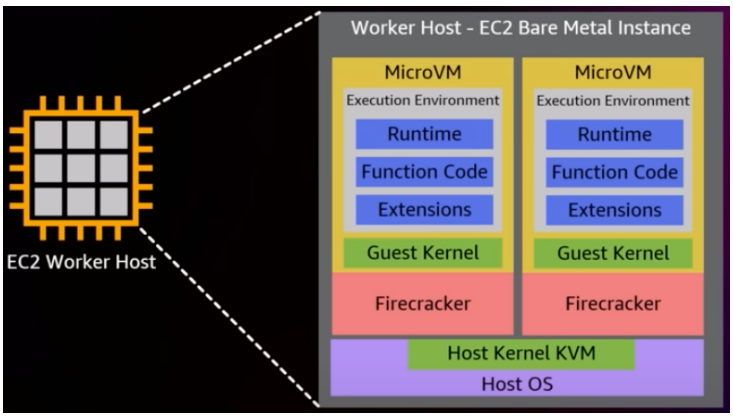

Figure 3: AWS Lambda 的工作负载使用 Firecracker 虚拟化 [4]

2.2 隔离及安全

2.2.1 隔离策略

FaaS 系统中，资源隔离是必须的。从资源利用的角度来说，需要隔离不同的函数运行实例，使其相互之间不受影响，例如 AWS 运行 Lambda 最小的实例内存为 128Mb，而典型的宿主机实例则大至 384Gb 内存。另一方面，从安全的角度来说，用户代码是不可信代码，如何防止攻击、保证底层系统安全，也需要对函数运行进行隔离。

常规的 Docker 实现依赖 Linux 内核提供的 groups、namespaces 等机制实现隔离，AWS 基于 Firecracker/microVM 实现了自己的多层安全策略，在性能/安全之间进行了较好的平衡，如 fig. 4所示 [1, 3]。这种模式的一个附加的好处是，可以更灵活地进行监控。

一些 Serverless 平台即支持 FaaS，也支持 CaaS，这两种模式的最终运行方式有所区别，因此有将这两种模式的运行时进行隔离的做法，例如字节的 FaaS 平台架构就采取了不同的 K8s 集群运行函数负载和镜像负载，如 fig. 5所示。

2.2.2 安全容器

由于容器技术是操作系统级的虚拟化技术，它的安全性比虚拟机要差，因此有一些技术用来增强容器的安全性，也称之为安全容器技术，例如 Kata Container

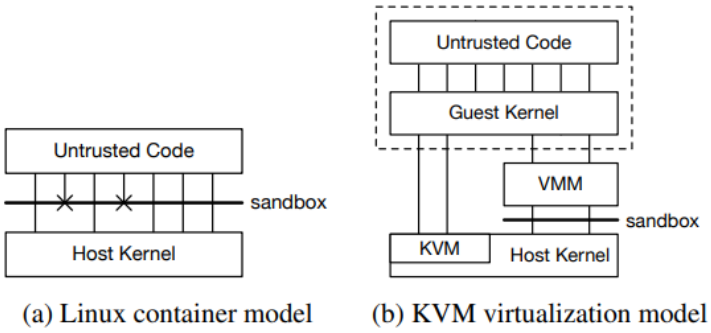

Figure 4: Docker 与 Firecracker 的隔离方式对比 [1]

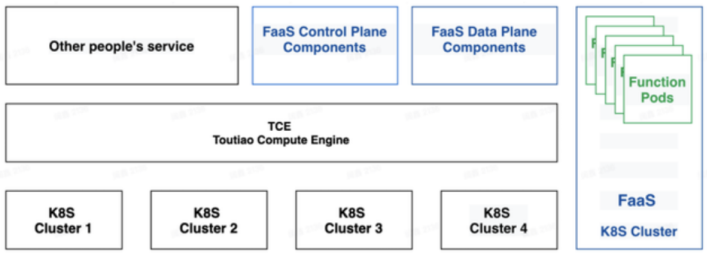

Figure 5: 字节跳动 FaaS 平台架构 [21]

和 Firecracker [2], 以及在 Kata 基础上优化的 RunD[13]。

3 平台整体架构

3.1 开发语言

Kubernetes 生态的主流语言是 Go，在 Serverless 平台的开发中，关于开发语言的资料相对较少，但根据现有资料，可以了解和推测：

- 基于 Kubernetes、Knative 等 Kubernetes 生态的系统以 Go 为主；包括 EasyFaaS、firecracker-containerd 等也是使用 Go 开发

[2]严格意义上 Firecracker 是一种轻量级虚拟技术（MicroVM）。

- 美团在建设 Nest 平台时没有选择 Go 而选择了 Java 作为主要语言，原因是因为美团内部 Java 占主导地位 [26]；另外 Apache OpenWhisk 主要是 Scala 实现
- Rust 有较多的应用，例如 Firecracker 使用 Rust 开发，字节使用了 Rust/WebAssembly[21]

3.2 Serverless 平台架构设计

3.2.1 调用模式和方式

典型的 Serverless 架构是基于事件驱动的模式，根据调用模型又可以分为同步模式和异步模式两种。同步模式下，客户端调用后等待函数运行完成得到结果；异步模式下，客户端触发调用后即完成，通常采用事件队列的方式来实现异步调用 [4]。

Serverless 调用触发一般包含 HTTP 请求和事件，由客户端主动发起调用；除此之外，还有定时运行的场景，如周期性报表 [26]。字节跳动实现了 gRPC、Thrift 等方式的调用 [21]。

3.2.2 基本的结构

尽管 Serverless 平台技术实现、业务功能都存在诸多的差异，但由于都是 Serverless 理念，其基本的运行方式、技术架构上都存在共通之处，我们可以按照功能把整体 Serverless 的架构分为如下几个部分：

- **调用触发**：作为 Serverless 的调用入口，负责响应用户请求
- **执行引擎**：负责最终执行负载（函数或者自定义镜像）
- **应用治理**：负责用户的函数管理，包括部署、配置、测试等
- **计费**：提供函数调用的计费信息以便计算成本
- **辅助服务**：除此之外的其他组件都可以归并为辅助工作负载执行或者事件触发的范畴之中，例如管理工作负载的生命周期、镜像拉取等各种功能

美团的 Nest 平台架构比较有代表性，如 fig. 6所示。

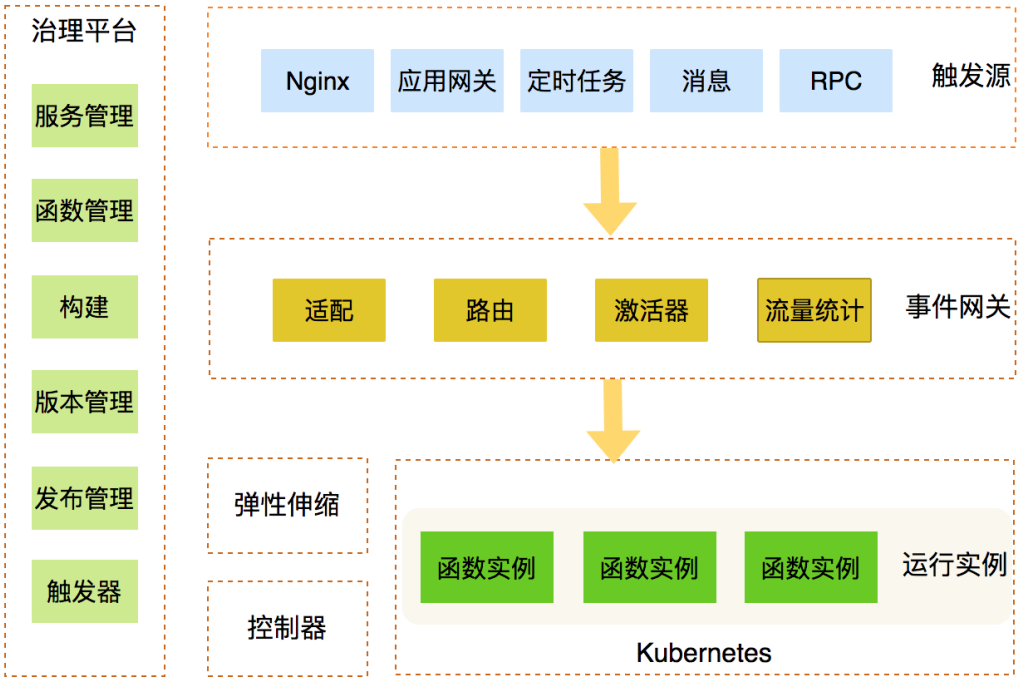

Figure 6: 美团 Nest 架构图 [26]

3.3 AWS Lambda 的整体架构

AWS Lambda 平台使用类似微服务的架构，划分了不同的子系统，每个子系统负责不同的职责，其架构如 fig. 7所示。

Lambda 分别使用不同的服务来支持同步异步调用。同步调用时，各个系统间的协作关系如 fig. 8所示，其中：

- Frontend Invoke Service 作为调用入口
- Counting Service 负责计费信息追踪，每次 Lambda 调用均会调用此服务，同时还有并发控制信息
- Assignment Service 负责为函数调用执行分配一个工作单元
- Placement Service 负责维护 Worker 的状态，提供基于时间的租约，供 Assignment Service 获取可用节点信息

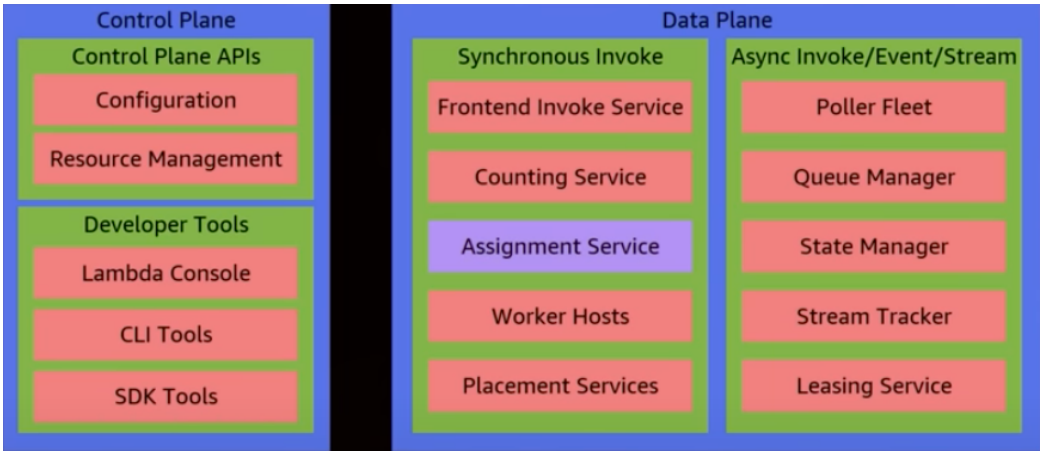

Figure 7: AWS Lambda 的整体服务划分 [4]

- Worker 是最终负责执行计算的单元

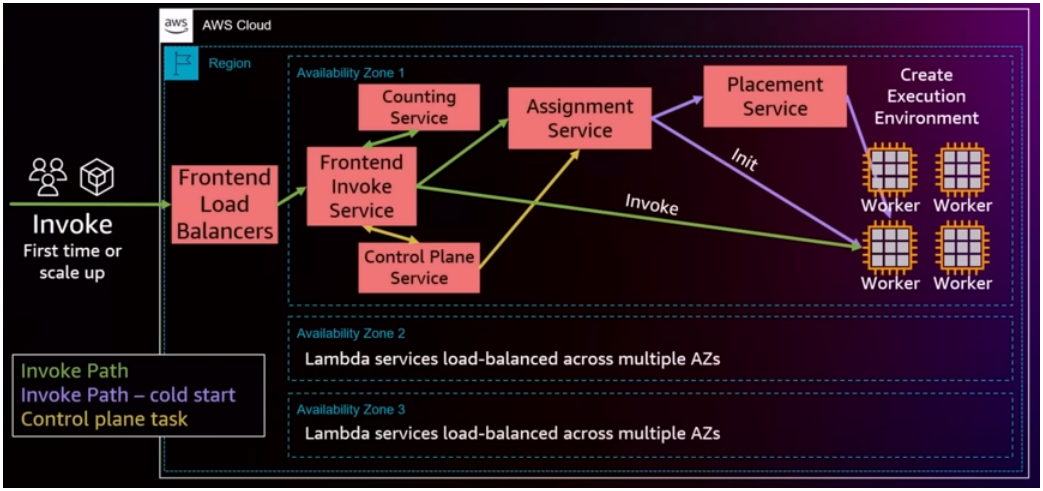

Figure 8: AWS Lambda 中的同步调用 [4]

而执行异步任务时，会首先将其缓存到队列中，并由 Poller 负责消费队列中的事件，然后转化为同步调用，其过程如 fig. 9所示。

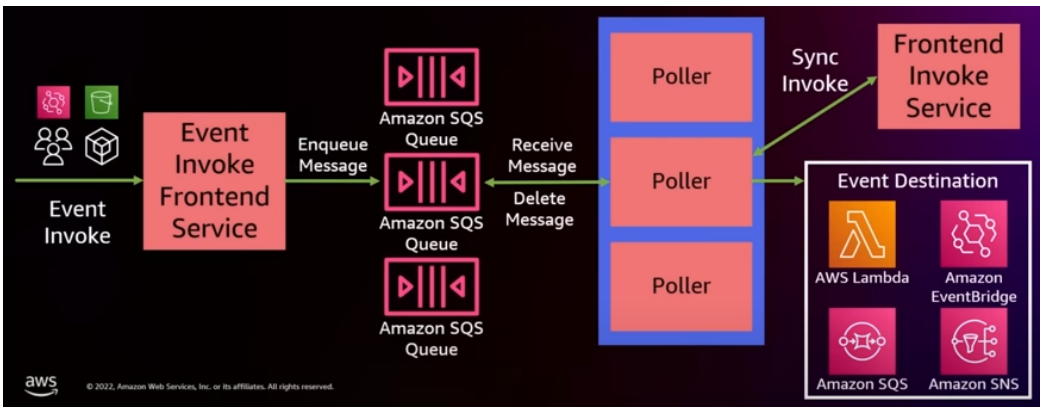

Figure 9: AWS Lambda 中的异步调用 [4]

4 冷启动问题

通常 Serverless 平台中，用户部署代码后，平台需要分配资源、下载代码（或者镜像）、初始化、启动实例等环节，这些步骤都是比较耗时的操作。而只有这些操作完成后，实例才具有服务的能力，这种延时称之为冷启动问题，它决定了 Serverless 平台弹性伸缩的能力，如 fig. 10所示。

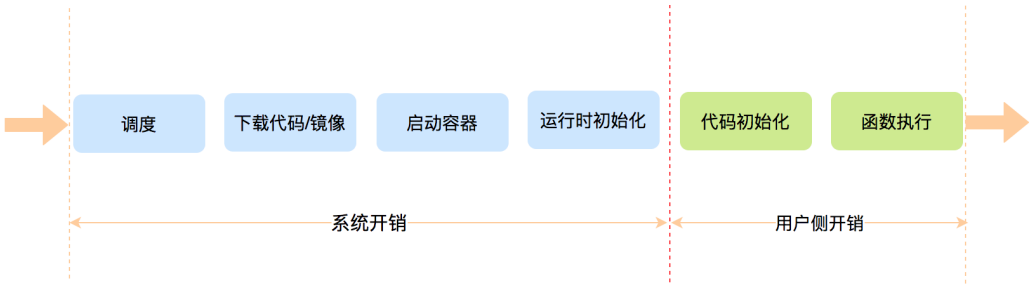

Figure 10: 冷启动的各个阶段 [26]

4.1 优化镜像加载时间

镜像的加载是一个必须要考虑的问题，常规的执行方式是在实例上下载代码，或者需要下载用户自定义的镜像。然而这些自定义的镜像可能非常大，例如，AWS Lambda 支持最大 250Mb 的代码（Zip 包），或者大至 10G 的镜像。优化初始化时

间的常规做法有：

- 裁减镜像或者使用轻量级虚拟化技术，提高镜像的启动速度 [26, 28]，
- 对代码进行缓存以避免重复下载 [28]
- 优化运行时的加载速度，例如 Java 使用 JIT[6]
- 优化 VPC 网络以提高下载速度 [29]
- 使用高效的压缩算法，例如使用 Zstd 压缩算法 [26]

AWS Lambda 通过优化镜像的底层存储，更进一步提升了镜像的加载速度，其核心思想是优化镜像在分布式文件系统中的布局，最大程度进行复用，如 fig. 11所示。

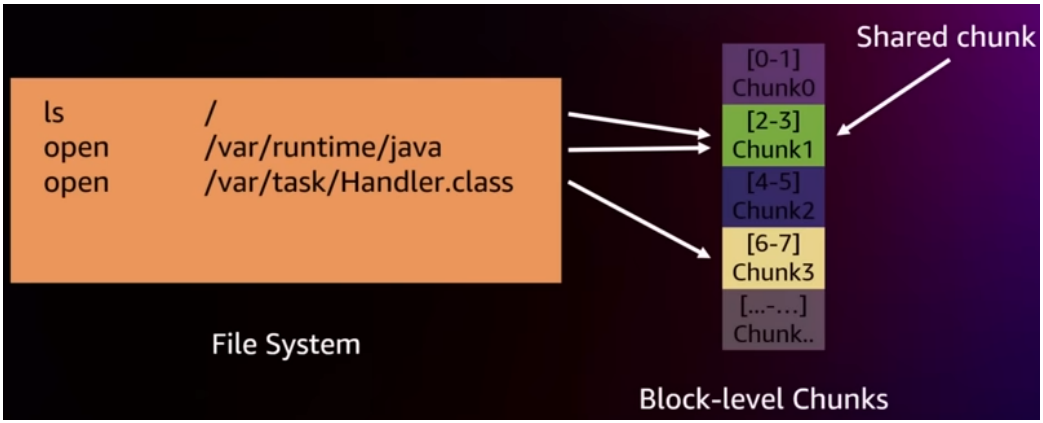

Figure 11: AWS Lambda 的镜像存储在分布式文件系统中的布局优化 [4]

4.2 池化工作实例

另一个解决冷启动的方式是提前创建工作实例，阿里云、腾讯云等均支持预留实例，允许用户预留一些实例[3] 根据流量情况进行实时预测，利用一些算法进行预测（例如机器学习），并按需进行自动扩缩容是普遍的实现 [28, 4, 26]。

工作实例创建完后，如果流量下降，一般也不会立即回收，而是会采取一定的

[3]这并不是一个符合 Serverless 设计理念的做法，并可能因此产生额外的费用。

策略进行回收，例如 Azure Functions 每 20 分钟回收一次 [5]。这样维护一个工作实例的池，有助于降低平均的响应延迟。

4.3 其他前沿的优化方案

除了以上的一些常规策略外，目前还有一些较为前沿的优化方案，例如镜像快照、热容器等 [6, 16]。AWS Lambda 已经上线了镜像快照，将 Lambda 运行镜像的构建流程从函数调用触发前移到发布过程，大幅降低了冷启动延迟，如 fig. 12所示。

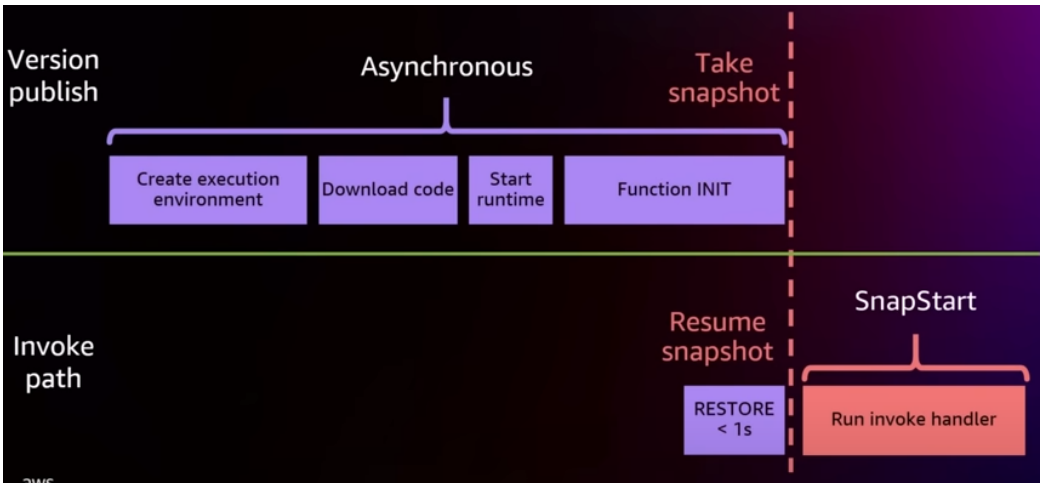

Figure 12: AWS Lambda 的镜像快照 [4]

5 高可用

5.1 异地多分区

高可用主要需要解决的问题是容忍分布式故障，基于 Kubernetes 的架构自身有一定的容忍能力，但单个 Kubernetes 集群存在伸缩瓶颈 [7]，且隔离程度有限，因此通常采取多集群的方式实现异地容灾、业务隔离和高可用，例如美团、字节均采取这种方式 [26, 21]。美团的异地多集群架构如 fig. 13所示。

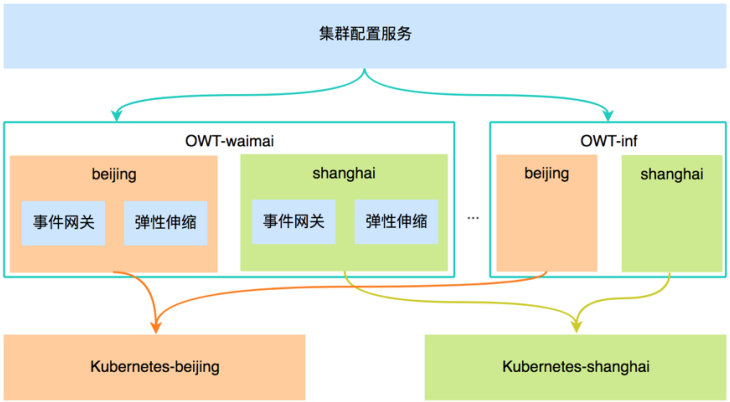

Figure 13: 美团 Nest 的部署架构

对于规模更大的公有云例如 AWS，采取的是分地域 + 可用区的方式，一方面实现隔离和高可用，另一方面可以就近服务而提升性能 [2]。

5.2 并发度限制

尽管理论上 Serverless 具有无限的扩容能力，但主流的 Serverless 平台都会进行并发限制, 通常有设置并发度的选项。AWS 默认每个区域为 1000QPS [4][10]；Google Cloud Run 单个容器默认限制为 80，最高可设置为 1000[14]；美团在事件网关上进行限流，并支持对后端函数进行降级和限流 [26]。

5.3 关键服务高可用

除了运行计算任务的集群本身需要进行分区外，从系统架构上还需要对关键的服务进行高可用设计，例如 AWS Lambda 的架构中，最开始是不同的分区有单独的 Worker Manager 服务负责管理这个分区下的任务分配，但由此带来的问题是一旦该服务出现问题会影响到整个分区，因此 Lambda 在其最新的架构中引入了高可用的 Assignment Service，采用分布式共识算法来解决这一问题，如 fig. 14所示。

[4]AWS 的并发限制是可以申请增加的，不设定上限，但需要提前申请。

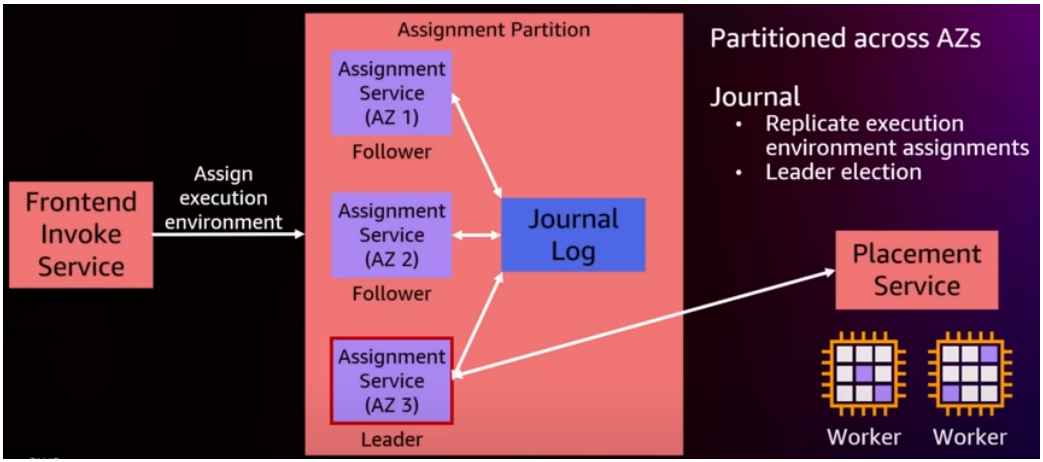

Figure 14: AWS Lambda 架构中 Assignment Service 的高可用设计 [4]

6 部署及开发工具支持

7 性能参考指标

References

[1] Alexandru Agache et al.
"Firecracker: Lightweight virtualization for serverless applications".
In: *NSDI 2020*. 2020.
URL: https://www.amazon.science/publications/firecracker-lightweight-virtualization-for-serverless-applications.

[2] *AWS Global Infrastructure*. 2021.
URL: https://aws.amazon.com/about-aws/global-infrastructure/.

[3] *AWS re:Invent 2020: Deep dive into AWS Lambda security: Function isolation*.
2020. URL: https://www.youtube.com/watch?v=FTwsMYXWGB0.

[4] *AWS re:Invent 2022 - A closer look at AWS Lambda (SVS404-R)*. 2022.
URL: https://www.youtube.com/watch?v=0_jfH6qijVY.

[5] *Azure Functions: Cold Starts in Numbers*. 2018.
URL: https://mikhail.io/2018/04/azure-functions-cold-starts-in-numbers/.

[6] Joao Carreira et al.
"From warm to hot starts: Leveraging runtimes for the serverless era".
In: *Proceedings of the Workshop on Hot Topics in Operating Systems*. 2021,
pp. 58–64.

[7] *Considerations for large clusters*. 2022.
URL: https://kubernetes.io/docs/setup/best-practices/cluster-large/#:~:text=More%20specifically%2C%20Kubernetes%20is%20designed,No%20more%20than%205000%20nodes.

[8] *Firecracker - Secure and fast microVMs for serverless computing*. 2022.
URL: https://firecracker-microvm.github.io/.

[9] *How Cloud Functions works*. 2022.
URL: https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-about&locale=en.

[10] *How do I request a concurrency limit increase for my Lambda function?* 2021.
URL: https://aws.amazon.com/premiumsupport/knowledge-center/lambda-concurrency-limit-increase/?nc1=h_ls.

[11] *Introducing AWS Lambda*. 2014.
URL: https://aws.amazon.com/cn/about-aws/whats-new/2014/11/13/introducing-aws-lambda/.

[12] *Is Google Cloud Run really Knative?* 2020.
URL: https://ahmet.im/blog/cloud-run-is-a-knative/.

[13] Zijun Li et al. "RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing".
In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)*.
Carlsbad, CA: USENIX Association, July 2022, pp. 53–68.
ISBN: 978-1-939133-29-27.
URL: https://www.usenix.org/conference/atc22/presentation/li-zijun-rund.

[14] *Maximum concurrent requests per instance (services)*. 2022.
URL: https://cloud.google.com/run/docs/about-concurrency.

[15] *Meet Nimbella, newest Kubernetes serverless solution on the block*. 2019.
URL: https://techgenix.com/nimbella-kubernetes-serverless/.

[16] Anup Mohan et al. "Agile cold starts for scalable serverless".
In: *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*.
2019.

[17] *Overview of Functions*. 2022. URL: https://docs.oracle.com/en-us/iaas/Content/Functions/Concepts/functionsoverview.htm.

[18] 云原生行业研究短报告 *(七)*: 无服务器. 2021. URL: https://pdf.dfcfw.com/pdf/H3_AP202201141540404897_1.pdf?1642193074000.pdf.

[19] 从零到一，*Serverless* 平台在滴滴内部落地. 2020.
URL: https://cloud.tencent.com/developer/article/1664422.

[20] 如何评估 *Serverless* 服务能力？这份报告给出了 *40* 条标准. 2021.
URL: https://developer.aliyun.com/article/784359.

[21] 字节跳动函数计算大规模实践及 *Serverless* 展望. 2022.
URL: https://zhuanlan.zhihu.com/p/532458495.

[22] 工商银行 *Serverless* 函数计算落地实践. 2021.
URL: https://www.infoq.cn/article/plg82yfmwvioqvvl0hzi.

[23] 微服务低代码 *Serverless* 平台 *(*星链*)* 的应用实践. 2022.
URL: https://xie.infoq.cn/article/a6cc05adc879c31b0df66ebc1.

[24] 李道兵：京东云的云原生理念及 *Serverless* 最佳实践. 2019.
URL: https://www.infoq.cn/article/5vbvEcE2UJuxxE4xer2k.

[25] 百度 *Serverless* 架构揭秘与应用实践. 2021.
URL: https://xie.infoq.cn/article/429b59c0c6c91f9cf8b633021.

[26] 美团 *Serverless* 平台 *Nest* 的探索与实践. 2021. URL:
https://tech.meituan.com/2021/04/21/nest-serverless.html.

[27] 美团如何通过新平台落地 *Serverless* 的?. 2022.
URL: https://www.infoq.cn/article/ecrsxE0VtOOwXlFgCvgW?utm_source=rss&utm_medium=article.

[28] 腾讯云函数计算冷启动优化实践. 2019.
URL: https://cloud.tencent.com/developer/article/1461709.

[29] 腾讯云函数访问 *VPC* 网络架构优化. 2019.
URL: https://cloud.tencent.com/developer/article/1461707.

[30] 腾讯云原生战略及 *Serverless* 平台实践解析. 2019.
URL: https://www.infoq.cn/article/we9ubdr1p*tr6crsmyrv.

[31] 阿里云 *FaaS* 架构设计. 2021.
URL: https://developer.aliyun.com/article/819594?spm=a2c6h.12873581.technical-group.dArticle819594.46fb5e66m8eGjW.