# 阅读笔记

## Towards Reference Architecture for a Multi-layer Controlled Self-adaptive Microservice System

将自适应应用于微服务的文章。

同时这篇文章也是一个模型建模，分为了两层处理：（可供轩轩参考）

* infrastructure-controlled layer
* application-controlled layer

阐述了目前微服务面临的困境：

* 微服务由于其自身分布式的特点需要自适应。
* 由于需求和上下文的改变，需要微服务系统处理。
* DevOp的需求。

感知与分析面临的新的要求：

* 需要对外部环境进行监控，系统底层资源等，目前能感知，但是无法精细控制。
* 系统上下文与系统资源耦合。

本文感知与分析需要监控的数据：

* 当微服务业务需求改变时，监控微服务本身的状态以及微服务与微服务之间的依赖关系。
* infrastructure layer包含一个MAPE，**感知外部环境的上下文及微服务系统层面上下文**。
* Application layer也包含一个MAPE，**感知微服务环境**，例如topology和需求的改变。并设立了一个adaptation strategy.

因为需求和场景的变化微服务系统在不断进化，系统必须能处理这些挑战。 同时，敏捷开发和DevOps方法要求系统能不停机运行，且持续集成。面对动态的环境无法仅靠人工来实现重新配置和调整，因此引入自适应系统。

该框架的创新点在于，将自主计算的MAPE控制循环构建一个多层控制循环。

概念图如下所示，自适应微服务系统引入了两个变量—reqiurements和external context。

其中external context，它是外部世界的一部分，包含硬件资源，OS和其他系统。

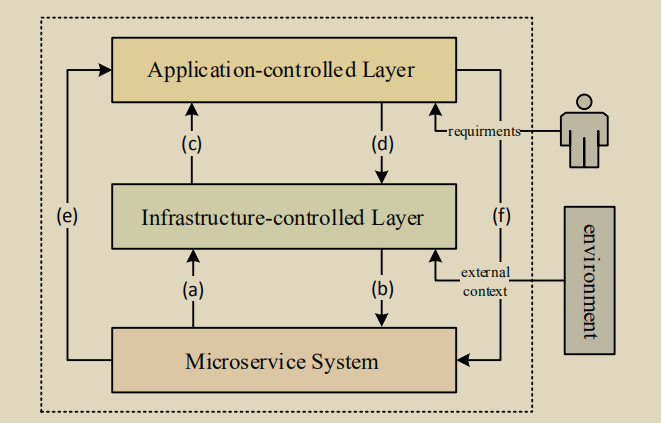


Figure 1: image-20211019204807207

挑战，系统通过重新配置微服务实例和重建微服务拓扑关系来实现自适应，因此将调整设施（adaption facility）隔离在A~和I~两层。两层分别管理不同的adaption。

I层通过MAPE控制循环来进行调整，感知的是**外部环境的context，微服务实例的系统context和监控数据，分析系统相关的信息，得出系统级别的决策**。A层也是同样的MAPE循环，不过对**象变为应用拓扑关系和用户修改的需求**，后续步骤也是同一层层面。

### 总结

简单分为两层，监控的数据不一样，没有建立起数据的联系关系。定制化的感知也未体现。

## Data-driven Adaptation in Microservice-based IoT Architectures

仍然是自适应应用于微服务的情况，在不同的层采取不同的自适应：(可以推荐给轩轩)

* 设备层：监控设备的QoS数据。
* 服务层：监控服务的QoS数据。
* 应用层：基于目标进行自适应。

应用机器学习处理设备层到服务层的数据，来判断QoS。

**很简单的分层，处理每个分层的数据**，参考的意义较小。

## Kuksa Self-Adaptive Microservices in Automotive Systems

采用MAPE-K。解决**特定**的领域的问题，仍然是物联网方向。与上文的思路基本一致。

## Gru: an Approach to Introduce Decentralized Autonomic Behavior in Microservices Architectures

去中心化的思想十分简单，**每个agent管理一部分容器，单独工作，获取邻居的信息**（这不就是路由表协议？）

监控的数据仍然十分**简单**， CPU、memory usage。

## Improving microservice-based applications with runtime placement adaptation

去中心化的监控，每一个服务使用其自由的监控协议栈。仍然是MAPE-K。监控如下的数据：

* 资源使用
* 应用日志
* 微服务通信之间的消息信息。

处理了监控数据的融合。使用了**model-runtime工具**。说白了也是新建一层，作为proxy在应用与系统之间。

去中心化与中心化的结合。与传统的MAPE-K不同的是加入了一个Model层。

**中心化负载处理每个去中心化的数据整合，半去中心化**。

*μ*Apps基于microservice构建，rely on message passing for communication and to decouple each microservice, allowing the logic in each service to scale independently.

*μ*Apps的工作流无法指定，The *μ*App communication must be monitored to infer the underlying workflow.缺点。

然而，微服务的管理很复杂，现存的tools不能进行管理操作，基于实时执行数据来进行微服务替换。

通过自动管理任务来收集运行时数据，以此提升*μ*Apps的管理，挑战： \* 联合监控 \* 找到一个high-performing placement \* Migrating microservices：

提出一个REMaP调整机制，解决方案是使用MAPE-K的adaption manager来在运行时自治地改变placement of *μ*Apps，这和现存的依赖静态信息的管理工具不一样。

运行在adaption manager的管理下提供一个集群和*μ*Apps的联合视图，

***MAPE-K***

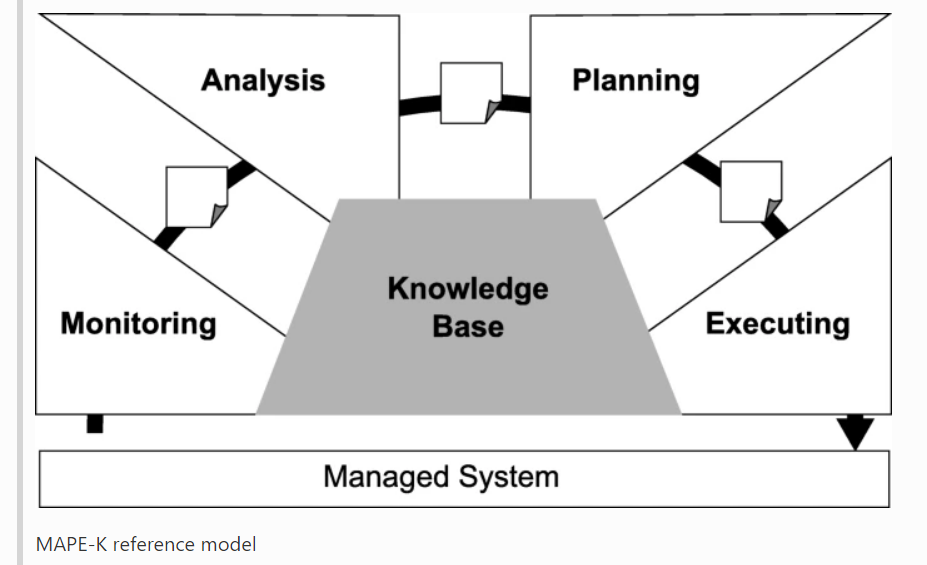


Figure 2: image-20211020162730743

## Self-managing cloud-native applications: Design, implementation, and experience

1、负载均衡

2、处理暂时的失效、在不可靠的infra上实现可靠性

3、监控云原生软件以自适应的应对局部失效…

本文提出了一种云原生架构具有可扩展可靠性

这篇好像讲的是一个实验，“实验”使用了

#### 监控

通过一些关键指标代表整个系统的可用性：**响应时间、吞吐量**。

通过一段时间的数据采集，得到了系统状态的一个点，一段时间内一个请求耗时的分布大致上反映了系统的性能。

所有服务都伴随有监控功能， 把监控得到的数据聚合到etcd集中式的存储中心上，提供给集群来做自适应的扩缩容；

背景：现在的管理功能都是由基础设施或者第三方服务来提供，外在于应用部署的。这样将管理功能和应用分开使得无法自然的scaling,要求额外的管理code和人为干涉。使用基础设施提供的管理服务也会导致vendor lock-in,降低工作效率。

主要注重的方面： **Cloud-native applications**的Resilience（处理系统资源和第三方服务的failure和fluctuation ）和Elasticity（动态调整capacity ）

当前的监控和健康管理以及伸缩主要有几种选择：1. 云服务商提供的监控服务 2. 第三方的服务 3. 用组件来实现海量实时的监控(ad-hoc solution).

文章的主要解决方案：对于Self-managing (micro)services，使用现代的分布式算法和组件来建立service，使用分布式的内存键值对数据库存储solution。

数据库用来存储每个管理功能的state ,利用共识算法来实现管理功能节点的选举和赋值。管理功能节点部署一个无状态的应用来作为原子服务。

## MONAD: Self-adaptive Micro-service Infrastructure for Heterogeneous Scientific Workflows

介绍将微服务应用于科学计算等领域、如深度学习等方面（？有这个必要吗）

就是很常规的内容

分为几个层次：监控收集、可视化展示、预警系统、

为了达到非侵入式的监控、使用消息队列的方式？

然后预警系统的作用是根据收集的到的监控数据、结合一定的算法、如果一些数据指标超过了阈值、就会触发自动的调整会根据反馈分配新的资源给计算单元，这个部分分为两个模块：

* 系统的识别：根据系统的监控数据判断某个计算节点监控指标是否符合要求；
* 控制器的设计：根据一定的策略处理监控指标超过阈值的情况；

## DKEM: A Distributed Knowledge Based Evolution Model for Service Ecosystem(基于知识的分布式进化模型)

听名字好像很贴合？，

讲的好像是推荐服务? 然后语义分析、docker 镜像推荐打标签那一套？

然后相似服务的冲突消解？类似于知识图谱中的冲突消解？

以微服务系统为例子：系统会自动的替换失效的服务(稳定性不行)，替换成具有相同的服务(可能由不同的服务提供商提供，因此服务之间怎么判断等效性？)

分布式的微服务系统中某个服务发生故障、就会自适应的从服务知识库(service knnowledge)中找到相似功能的服务，来替代稳定性不行，出错了的服务。

还有一个bridge rules：**负责从不同的知识库中组合匹配知识**，

构建了不同层次的知识库、如果发现不是的微服务系统中的某个服务出现了故障、或者因为稳定性差等原因，从某个知识库中匹配出和原有服务功能相似的服务，替代原有服务：这个叫Local evolution；

而对于某个服务的失效，单个的知识库不能提供功能相似的服务的话、需要多个知识库之间协作、匹配出能替代原有服务的新的服务，替代原有服务：这个叫做Global Evolution；

## 

## 参考文献

[1] Liu P , Mao X , Zhang S , et al. Towards Reference Architecture for a Multi-layer Controlled Self-adaptive Microservice System[C]// The 30th International Conference on Software Engineering and Knowledge Engineering. 2018.

[2] MD Sanctis, Muccini H , Vaidhyanathan K . Data-driven Adaptation in Microservice-based IoT Architectures[C]// International Conference on Software Architecture (ICSA) 2020. 2020.

[3] Banijamali A , Kuvaja P , Oivo M , et al. Kuksa\*: Self-Adaptive Microservices in Automotive Systems[J]. 2021.

[4] Florio L , Nitto E D . Gru: An Approach to Introduce Decentralized Autonomic Behavior in Microservices Architectures[C]// 2016 IEEE International Conference on Autonomic Computing (ICAC). IEEE, 2016.

[5] Sampaio A R , Rubin J , Beschastnikh I , et al. Improving microservice-based applications with runtime placement adaptation[J]. Journal of Internet Services and Applications, 2019, 10(1).

[6] Toffetti G, Brunner S, Blöchlinger M, et al. Self-managing cloud-native applications: Design, implementation, and experience[J]. Future Generation Computer Systems, 2017, 72: 165-179.

[7] Nguyen P, Nahrstedt K. Monad: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows[C]//2017 IEEE International Conference on Autonomic Computing (ICAC). IEEE, 2017: 187-196.

[8] Wang X, Feng Z, Chen S, et al. Dkem: A distributed knowledge based evolution model for service ecosystem[C]//2018 IEEE International Conference on Web Services (ICWS). IEEE, 2018: 1-8.