China Summary Translation: 'Adopt a Mesh App and Service Architecture to Power Your Digital Business'

Published: 16 September 2019 **ID:** G00450677

Analyst(s): Sandy Shen, Anne Thomas

Digital business imperatives are driving application leaders to modernize their application portfolios to enable better experiences and greater agility. MASA provides three architectural capabilities to power your digital business: multiexperience, API mediation and multigrained services.

中国语境

最近国内对“中台”的讨论非常多。这个概念是由阿里巴巴率先提出的，由于定义不够明确，引发了一些困惑。有很多厂商借着这个概念推销产品，也有很多中国客户想要清晰了解中台的概念和搭建中台的最佳实践。中台是中国市场独有的概念，这种说法 Gartner 并没有在其他市场发现过，也没有在自己的报告中使用过。相比而言，Gartner 建议客户使用网格应用和服务架构（MASA）框架规划数字业务的应用架构。围绕这个话题，Gartner 根据客户问询以及公开信息，总结出以下要点：

1. MASA 是一种综合性企业应用架构，而中台只涉及其中的服务层。MASA 有三个层次：多样化体验层（针对特定用户、特定场景的终端应用）、应用编程接口（API）解析层，以及多粒度服务层（管理应用的业务逻辑与数据，并根据业务需求开放不同颗粒度的接口）。中台只涉及服务层，而且提供的是打包应用模块，比如订单管理、产品管理和客户管理等，并不支持多粒度服务的概念。

2. 实现数字业务应用并不是要把每个应用都改造成微服务架构。各项应用接口的颗粒度应由业务需求决定，即根据业务来确定是否针对整体应用、功能模块还是具体特征开放 API。部分后台服务适合用微服务架构来搭建，但并不需要把全部应用都分解成微服务。过度分解会浪费大量资金和精力，而且还会增加应用实施和治理的难度。产品经理应该与应用架构师一起对服务颗粒度进行规划和扩展，使其符合业务需求和技术需求（见第 4 点）。

3. 数据和应用都是数字业务能力的重要组成部分，应在统一的框架下同时处理。很多中国客户关心如何建立数据中台和应用中台，认为这是两个独立的平台。数据中台可以将多种数据源整合到中央数据仓库进行关联和分析，对诸如客户、产品、订单等业务实体实现统一视图，便于业务用户访问和分析数据。然而，数据中台的主要目的是数据分析，并不能支持业务应用和前端应用实时、高性能的访问。将业务与数据分开处理，会大幅增加部署所需的资源和成本。Gartner 的混合集成平台（HIP）框架可以协调应用、数据和流程的集成与治理。技术能力较为先进、IT 治理较为成熟的大型企业机构，或安全方面要求较高的企业机构，可以考虑通过数字集成中枢（DIH）来集成传统应用的数据，并为前端应用提供实时、高性能的数据访问。要详细了解这方面信息，请参阅“Innovation

Insight for Hybrid Integration Platforms”以及“Innovation Insight: The Digital Integration Hub Turbocharges Your API Strategy”。

4. MASA 是一种以业务为导向的架构，需要采用新的治理模式和开发流程。MASA 架构要求不是由开发团队决定应该提供哪些服务，而是需要根据业务需求来开发服务和 API。这就对产品管理实践提出了更高的要求，产品经理需要根据最紧迫的业务需求来制定应用服务和 API 路线图。Gartner 2019 年的 CIO 调研显示，目前已经有 48%的中国企业机构在使用或建立以产品为中心的开发模式，但这种做法才刚刚起步，需要一段时间来完善。实施 MASA 的理想团队应该是由多部门合作，并且拥有丰富的开发技能，包括用户体验（UX）/用户界面（UI）设计与开发、API 设计、应用集成、服务架构与开发等能力。

5. MASA 并不能依靠购买厂商解决方案来实现。通过 MASA 的规划、管理和建设，企业机构可以享受到更好的灵活性和敏捷性。中台也致力于提升灵活性和敏捷性，其方法是提供一些可独立部署的模块，主要服务于电子商务和销售类场景。企业机构可以利用厂商方案获取一部分 MASA 所需的技术能力，如集成平台即服务（iPaaS）、API 网关、API 生命周期管理和多体验开发平台（MXDP）等，但没有哪家厂商的解决方案能覆盖一个企业机构所需的全部技术和业务场景。无论企业机构采用的是传统应用、商业应用、定制开发应用，无论它们是以本地、托管还是软件即服务（SaaS）的方式在管理，也无论应用数据是来自内部还是外部，企业机构都需要负责其自身架构的设计和管理工作。企业机构对厂商所谓的“中台”方案一定要警惕，确保厂商不是以“中台”的名义推销其商业应用。

6. 实施 MASA 需要很长时间，但并不需要等其完全实现后才能收获业务价值。企业机构需要进行持续管理和迭代，也需要较长的时间来让各种应用达到合理的颗粒度组合。如果能选择一些只要投入较小精力就能看到产出的应用，企业机构就可以较快享受到 MASA 的灵活性与敏捷性，比如在不影响用户体验、业务逻辑或数据模型的基础上，对服务进行快速发布、修改和扩展。

企业架构师和应用领导者可通过本文了解 MASA 的概念，以此来引导数字业务应用架构的搭建，并通过文末推荐的报告了解 MASA 实施过程中的最佳实践，从而实现应用架构的升级。

主要挑战

- 数字化竞争压力需要企业机构进行新的开发和大范围的应用现代化，从而改善用户体验、实现业务敏捷性和确保安全性。
- 套件式的应用、复杂的架构和大量的技术遗留问题都给开发团队带来了负担，限制了团队支持数字业务转型的能力。
- 有限的资金意味着应用领导者无法替换或重构全部传统应用。

建议

负责应用开发和平台的应用领导者应：

- 投资适用于特定用户和特定场景的终端应用，以改善用户体验，支持多样化体验。
- 通过后台应用服务支持业务功能来提高业务敏捷性，在提供核心业务能力时以务实的方法拆分应用，构建应用编程接口。

- 通过构建 API 解析层来实现敏捷性并确保安全性。

以下是该文件的英文版全文

Adopt a Mesh App and Service Architecture to Power Your Digital Business

Anne Thomas, Aashish Gupta

Digital business imperatives are driving application leaders to modernize their application portfolios to enable better experiences and greater agility. MASA provides three architectural capabilities to power your digital business: multiexperience, API mediation and multigrained services.

Key Challenges

- Competitive pressures for digitalization demand new development and extensive application modernization to improve user experiences, enable business agility and ensure security.
- Monolithic applications, architectural complexity and extensive technical debt place burdens on development teams and limit their ability to support digital business transformation.
- Limited funding means that application leaders can't replace or refactor all of their legacy applications.

Recommendations

Application leaders responsible for application development and platforms should:

- Improve user experiences and support multiexperience by investing in fit-for-purpose apps.
- Increase business agility by service-enabling business capabilities and take a pragmatic approach when decomposing applications and building APIs to surface core capabilities.
- Enable flexibility and ensure security by building an API mediation layer.

Introduction

Digital business depends on a flexible application portfolio that enables delightful digital experiences, that supports new ways of doing business, and that makes the most of new technologies.

Unfortunately, the traditional three-tier application architecture (comprising a browser, a monolithic application deployed in an application server, and a database) is woefully inadequate to support these digital business demands. Inflexible monolithic applications, architectural complexity, and technical debt burden development teams, impede agility, and frustrate users. Application leaders

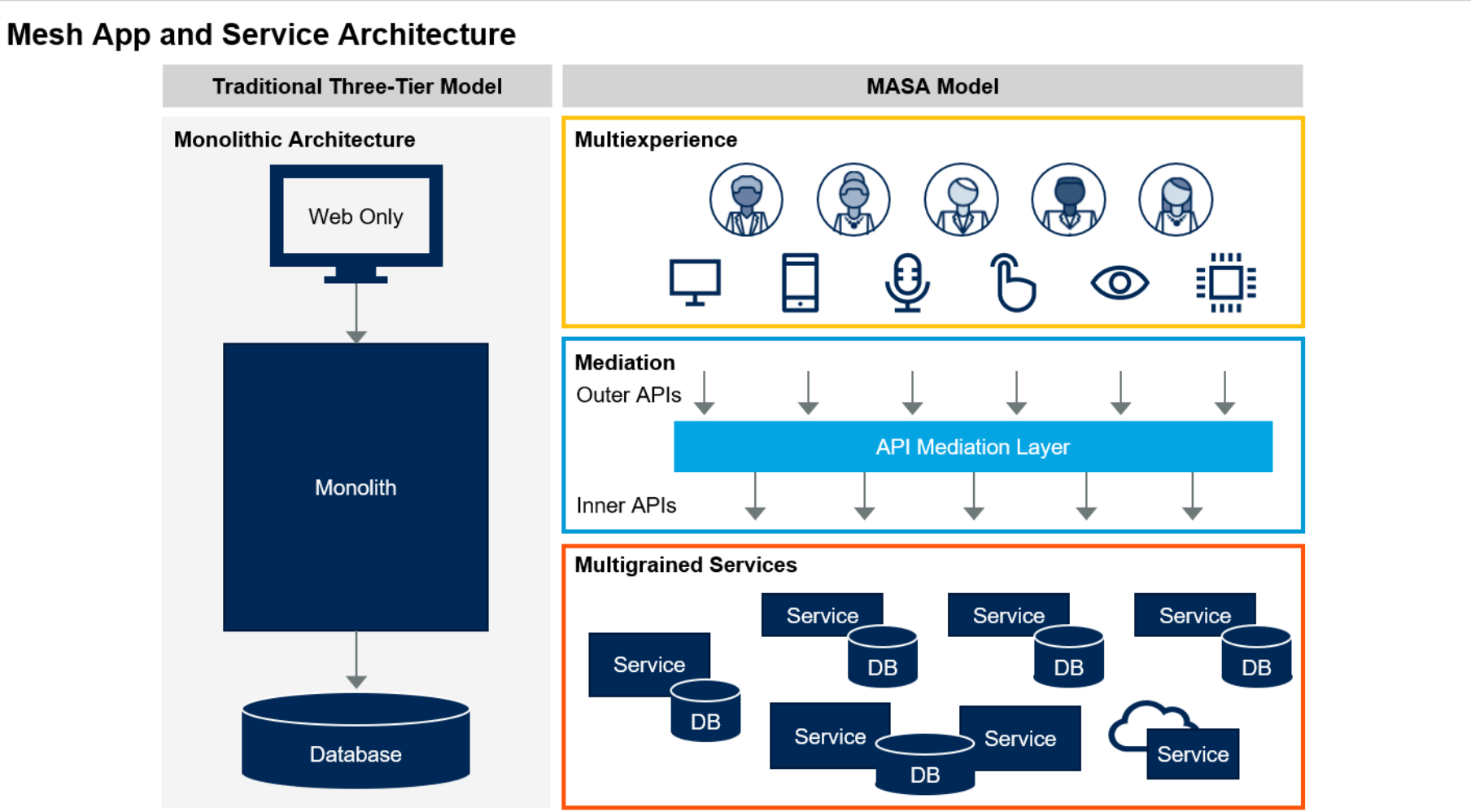
must modernize their application portfolios to facilitate digital business transformation. This modernization effort starts with a modern application architecture.

So, what does modern application architecture look like?

A mesh app and services architecture (MASA) provides the fundamental architectural capabilities that enable applications to support multiple experiences and respond rapidly to digital business demands. MASA provides the architecture for individual applications as well as a strategy for modernizing the entire application portfolio. It provides an evolutionary approach that enables development teams to iteratively modernize their applications in direct response to business priorities.

This research describes the three fundamental components of MASA (multiexperience, mediation and multigrained services — as seen in Figure 1) and the benefits they bring to your business.

Figure 1. Mesh App and Service Architecture



Analysis

Deliver Multiexperience UX by Investing in Fit-for-Purpose Apps

In the digital age, customers, partners and employees have come to expect a great user experience (UX) that extends beyond web and mobile interfaces to support voice, touch, wearables and immersive technologies.

Optimized experiences should support the specific needs of the different personas that use the application as well as the different capabilities afforded by the devices they are using. At the same time, an optimized experience should ensure that a particular persona has a consistent experience as they move from one device to another.

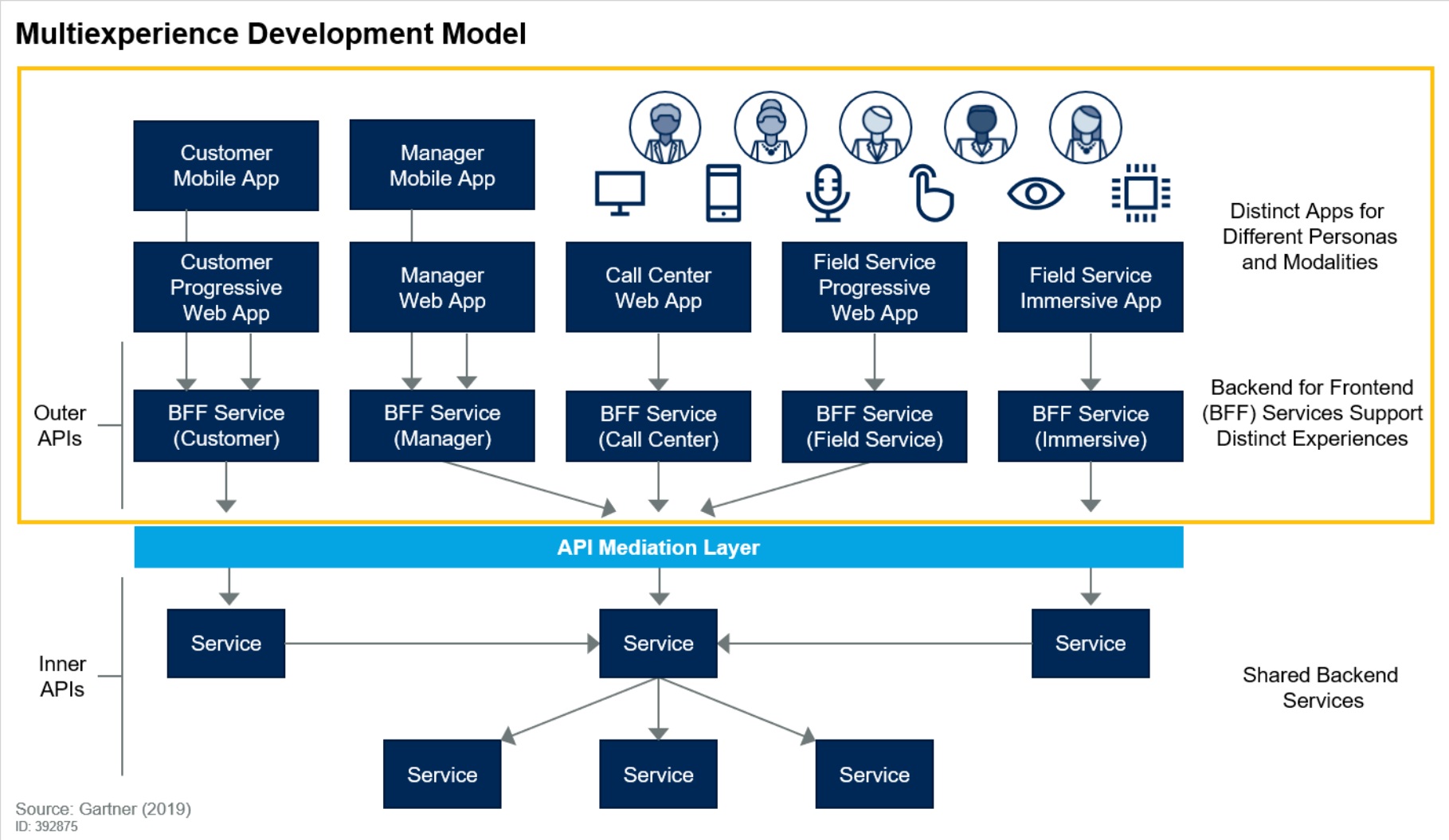
The best practice for providing an optimized experience is to build separate, fit-for-purpose apps that align the design, function and capabilities of the app to fit the workflows of individual personas and modalities. Unlike a one-size-fits-all client experience that attempts to support every feature that every type of user wants from the software, fit-for-purpose apps focus in on the workflows that users follow as they perform their specific tasks. This approach provides the opportunity to create smaller purpose-built apps that are easier to design, develop and deploy. This approach also allows the app to be focused on whatever channel the user prefers when performing tasks, whether it's a web-based app running on a laptop, a mobile app on a phone or a conversational interface.

Multiexperience design requires the backend to be flexible enough to support the different capabilities and workflows of every app in use. Because users don't always use the same devices, and often switch from one device to another during their working day, the backend must offer a continuous experience (see "Survey Analysis: Insights to Kick-Start an Enterprise Multiexperience Development Strategy" and "API Mediation Is the Key to Your Multiexperience Strategy"). Application leaders should consider adopting microapps to ensure consistency across modalities (see "Innovation Insight for Microapps").

MASA supports these requirements by implementing the backend functionality as a mesh of services that can be composed to support the specific needs of an optimized fit-for-purpose app. A popular design pattern called Backends for Frontends (also known as BFF) supports custom workflows for these optimized apps. Such services align with a specific UX. They offer customized APIs that support the specific workflow required by the UX and orchestrate invocations of the generic backend services that implement shared application functionality.

This model enables development teams to rapidly implement new frontends to support new personas or new devices without impacting other apps or services. Figure 2 illustrates this multiexperience development model.

Figure 2. Multiexperience Development Model



Increase Business Agility by Implementing Backend Functionality Using a Pragmatic Multigrained Service Model

Modern applications must support multiple experiences, and they must be agile enough to enable developers to rapidly deliver new capabilities to capitalize on digital business opportunities. MASA ensures that an application's backend is agile and customizable. The backend consists of multiple independent modules (that is, services) that developers can create, update and replace rapidly and orchestrate to support distinct workflows. These backend services should be generic and agnostic to any that consumes them.

Each service encapsulates a business capability and exposes that capability to consumers through an API. The most common type of API used in MASA is a REST API, but MASA also supports other types of request/response APIs (such as SOAP and gRPC), as well as message- and event-based APIs.

MASA takes a very pragmatic approach to the backend service model. Services can be quite varied in terms of purpose, granularity, ownership and deployments. Not all services need rapid replacement and updates, and therefore MASA supports priority-driven refactoring and delivery cadence for these services.

A client app often interfaces to the backend via a BFF service. You may have a one-to-one mapping between frontend apps and BFF services, or a BFF service may support multiple apps with similar experiences. A BFF service invokes or triggers the appropriate set of services to accomplish its tasks.

Your development teams might implement some of those services specifically for this application. Other services could tap into functionality within other applications. You might get some of those services from partners or an industry ecosystem. Some might be SaaS services. They may be a mix of on-premises and cloud-hosted services.

MASA supports any type of service implementation model and granularity level. For example, your developers could implement the services using:

- A third-generation language (3GL) like Java or C#.
- A development platform such as a low-code or multiexperience development platform.
- An iPaaS or enterprise service bus (ESB).
- A data virtualization tool.
- A serverless function PaaS.

The point is that you can implement the individual services using whatever technologies are most appropriate for the application. The services can also have a broad range of granularities. Not every application needs to be decomposed into dozens or hundreds of microservices. Oftentimes, you don't have a business justification to rearchitect an application — and that's just fine. Development teams should be pragmatic when building new applications or modernizing legacy applications, and you need to let business drivers inspire decomposition strategies.

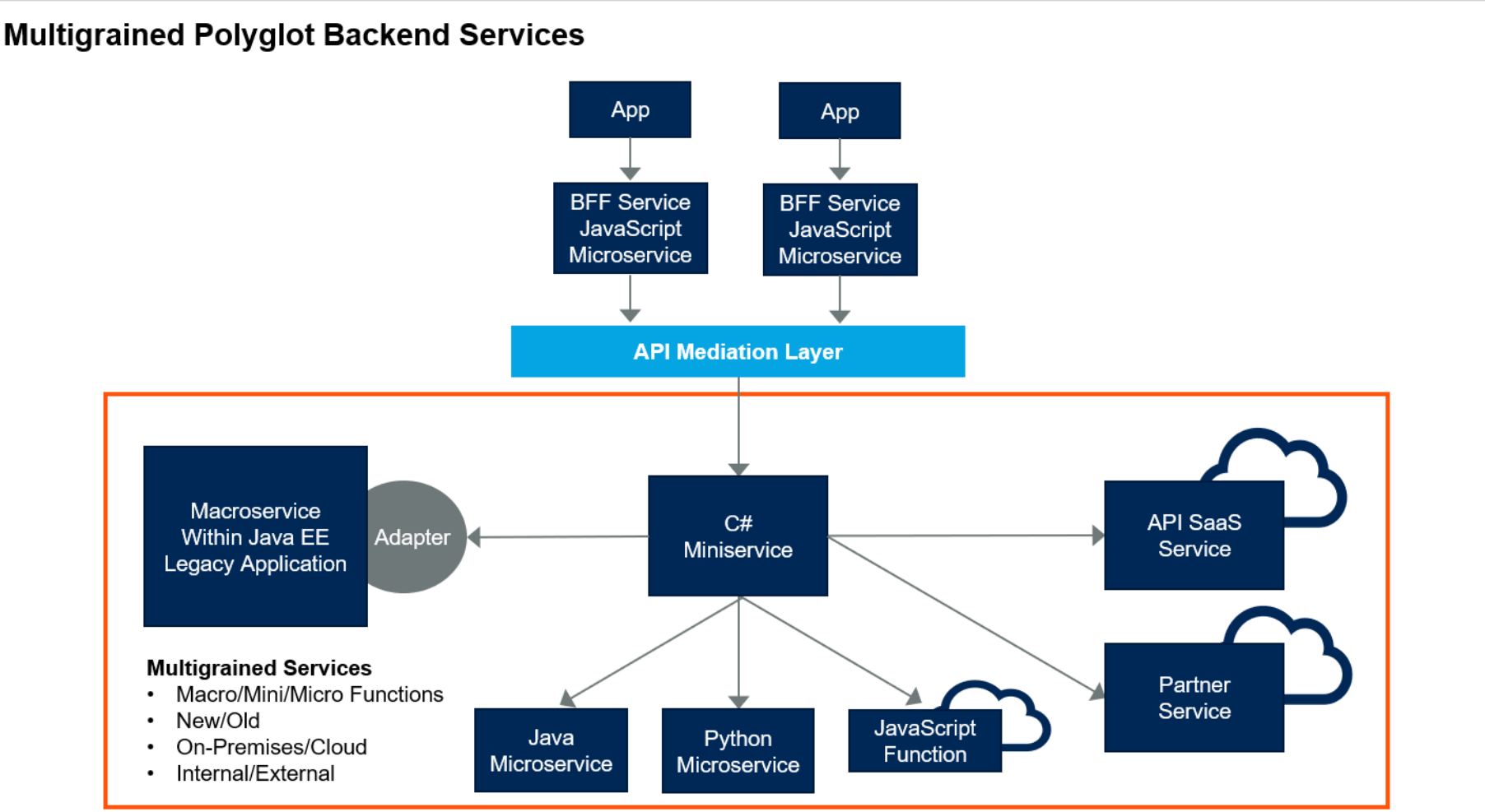
The spectrum of granularity includes:

- Large-grained **macroservices** that encapsulate and expose some or all of the capabilities of a monolithic application without refactoring.
- Coarse-grained, independently deployable **miniservices** that implement domain-level functionality.
- Fine-grained, independently deployable **microservices** that implement individual features.

When architecting or rearchitecting applications, development teams should determine service granularity based on change boundaries. Teams should typically start with coarse-grained services and decompose further only when application requirements demand it. You should also make sure that developers don't turn everything into a microservice. Too much granularity makes the application more complex than it needs to be, which can negatively impact the application's agility and stability.

Services should be cohesive. Things that change together should stay together (see "Innovation Insight for Microservices" and "Not Just Microservices: Choose the Right Service Granularity for Your Application" and Figure 3).

Figure 3. Multigrained Polyglot Backend Services



Source: Gartner (2019)
ID: 392875

Ensure Flexibility and Security Using an API Mediation Layer

A critical principle in MASA is ensuring that the individual parts of the application can change as needed without impacting other parts of the application. Because APIs can cause tight bindings between application components, all API interactions should be abstracted to ensure loose coupling and flexibility.

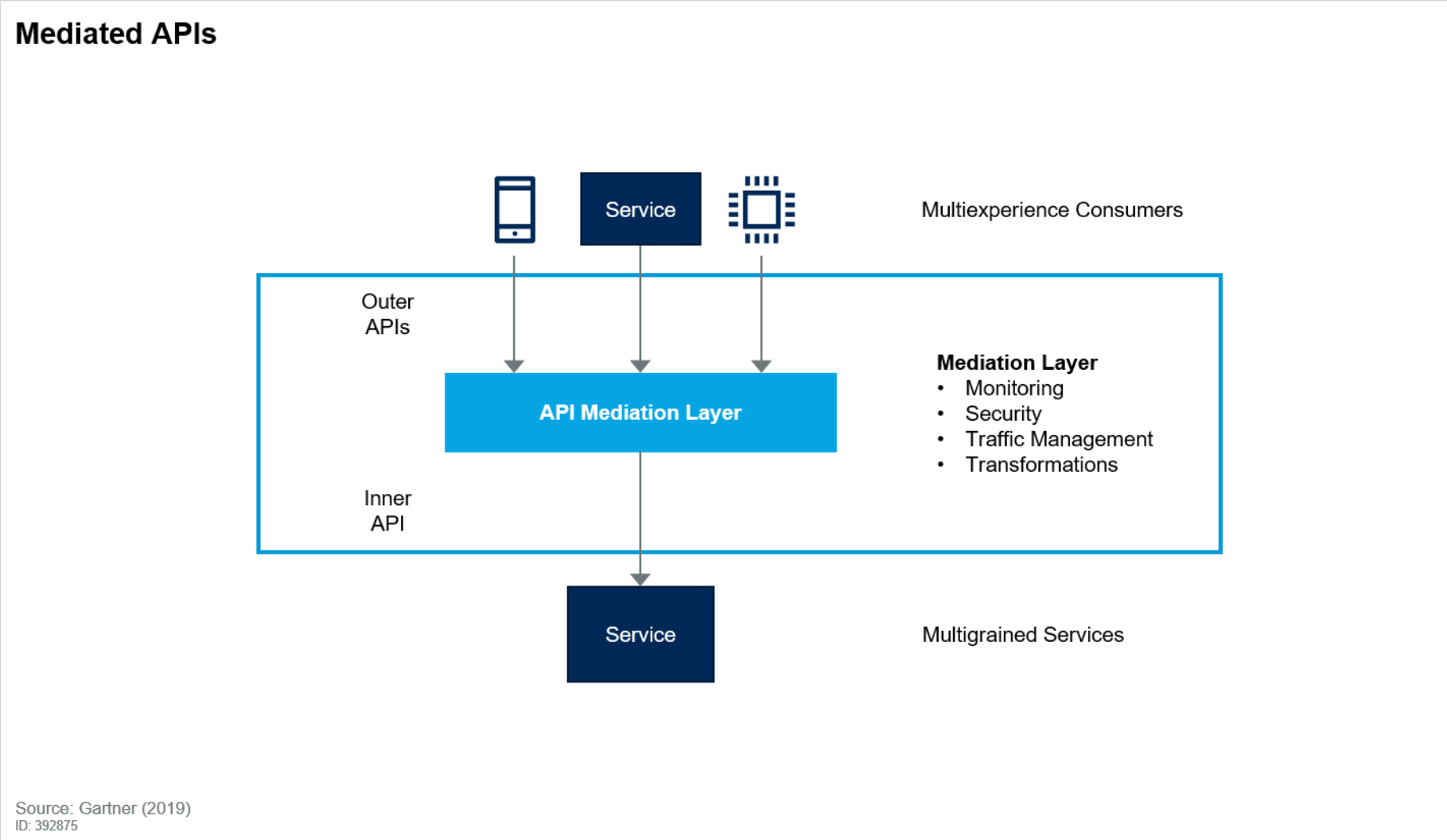
> APIs can expose sensitive information and functionality to any application that can access the API. You must properly secure your APIs to prevent data breaches and malfeasance.

An API mediation layer solves these and other concerns. The mediation layer allows a service to expose an "inner API" that directly reflects its domain model, and one or more "outer APIs" that are tailored to support specific client requirements. The API mediation layer intercepts communications between two application components (app-to-service or service-to-service) and enforces policies that apply to the communication (see Figure 4).

A mediator can inject additional capabilities that can enrich and protect the interaction. The model is, by definition, extensible, permitting any type of injected capability, including:

- **Monitoring interactions:** The API mediation can monitor interactions from both technical and business perspectives. The mediator can direct the instrumentation information to application performance management systems and to business analytics systems. It can track API utilization for monetization purposes.
- **Securing interactions:** API mediation can enforce authentication and authorization policies and prevent malware intrusions. It can also encrypt communications, manage the encryption keys, and validate signatures to ensure that the message content has not been changed.
- **Managing traffic flow:** API mediation can support request routing, load balancing, throttling, redirection, and auto-recovery of failed interactions.
- **Managing transformations:** API mediation can translate between different protocols. It can transform messages, thereby supporting custom formats for different consumers and mappings between different API versions. It can also filter sensitive information out of the messages.

Figure 4. Mediated APIs

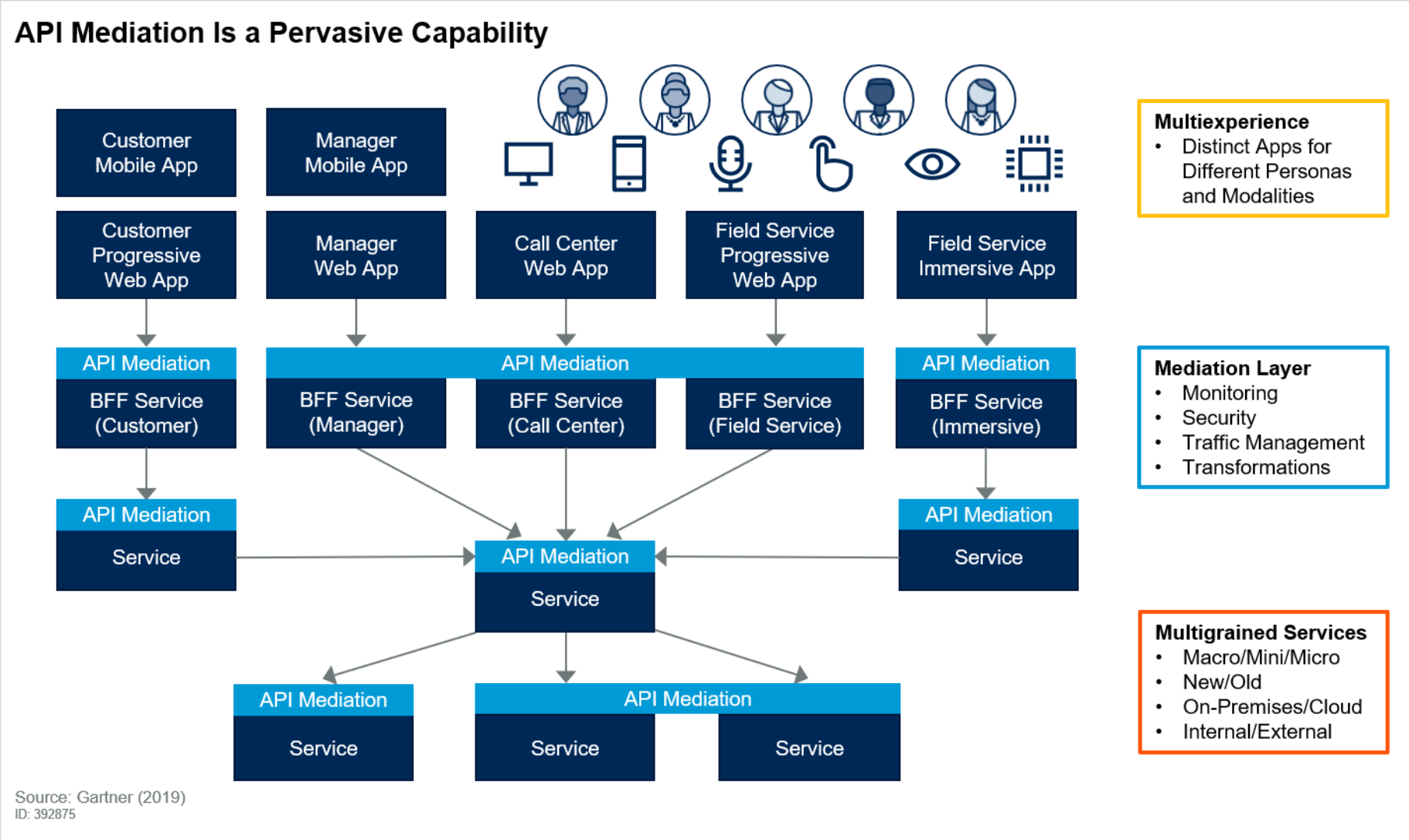


Teams typically start building a mediation layer by using an API gateway, but as the MASA ecosystem grows, teams can extend the mediation layer using a variety of complementary technologies. These could include service meshes, event brokers, network mediators, and homegrown, custom-built mediators.

Every API should be mediated, but you want to minimize the overhead imposed by the mediation layer. Therefore, you should use an appropriate mediator for different types of communication. For example, you typically use an enterprise gateway to mediate external inbound and outbound calls, and you would use a service mesh to mediate service-to-service calls within a managed container cluster (see “Mediated APIs: An Essential Application Architecture for Digital Business” and “How a Service Mesh Fits Into Your API Mediation Strategy”).

Figure 5 illustrates the three parts of MASA working together to enable a flexible, agile and secure architecture. It shows mediators protecting every API. In some cases, you can use shared mediators. In other cases, you may use a dedicated mediator (such as a sidecar proxy) for an individual service.

Figure 5. API Mediation Is a Pervasive Capability



Acronym Key and Glossary Terms

API	application programming interface
BFF	Backend for Frontend
MASA	mesh app and service architecture
UX	user experience

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

“How to Prepare for the Future of Applications”

“Building an Agile Application Architecture With Integrated Apps, APIs and Services”

“Survey Analysis: Insights to Kick-Start an Enterprise Multiexperience Development Strategy”

“Building UIs in an Agile Application Architecture”

“Mediated APIs: An Essential Application Architecture for Digital Business”

“Innovation Insight for Microservices”

“Not Just Microservices: Choose the Right Service Granularity for Your Application”

“Create the Role of Product Manager as Part of Treating APIs as Products”

“How to Build an Effective API Security Strategy”

Evidence

This architectural model is based on surveys, interviews, case studies, and technical blog posts from leading digital business firms.

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp