Overcoming Ongoing Digital Transformational Challenges with a Microservices Architecture

IT organizations must look beyond yesterday's monolithic Web applications and embrace microservices, whose loosely-coupled architectures speed development, testing and deployment, accommodating today's and tomorrow's digital requirements.

Executive Summary

The accelerating digitization of business is forcing many organizations to rethink their operating models to meet the ever-rising expectations of technology-savvy customers. At a fundamental level, companies across all industries need to ensure their services are available through digital channels and are competitive with technological innovations offered by rivals far and wide.

To enable such digital transformation, IT organizations need to overcome several key challenges:

- **The cost of change.** Customers want significantly reduced technology costs and are unwilling to fund technology changes that do not result in direct customer benefits.

- **The pace of change.** Quarterly or even monthly release cycles no longer meet user needs. Customers expect more frequent rollouts, even multiple times a day, to keep pace with their informational and service requirements.

- **The quality of change.** Decision-makers want to improve the scalability of applications and deliver a rich, interactive and dynamic user experience on a wide variety of devices, while minimizing the risk of change.

This cannot be done with traditional multi-tier Web applications — the de facto standard for building applications that enable today’s era of Internet applications — as they are too difficult to develop, test and maintain. Multi-tier Web apps typically consist of a complex code base, as the result of enormous functionality built into a single Web application. Further, a single change can affect multiple sub-units, creating a much larger and more complicated testing effort, requiring testers to understand various code interdependencies or test the entire application for each change. Lastly, because changing even a single aspect of these monolithic applications affects the entire code base, it slows down the testing process and makes it error-prone.

To meet growing multi-dimensional challenges, IT organizations must shift from monolithic Web applications that serve HTML to desktop browsers, toward an applications architecture that enables architectural extensibility.
This white paper describes an alternative architecture that is based on microservices. Applications designed with a microservices architecture consist of a set of narrowly focused, independently deployable services. We believe that microservices are the basis for a next-generation architecture that can meet the operational flexibility and functional needs of today’s burgeoning, dynamic digital economy. This said, much complexity exists at a higher level with this approach, in terms of managing services and orchestrating business processes throughout. This paper details some of the best practices for implementing a microservices architecture.

Defining a Microservices Architecture

The microservices-based architectural style is an approach to developing a single application as a suite of small, narrowly focused, independently deployable services. Each microservice runs in its own process and communicates with a lightweight mechanism, often an HTTP resource API. These services are built around business capabilities and are independently deployable by fully automated deployment machinery (see Figure 1). Although a microservices-based architecture is not a particularly new idea, the concept has achieved popularity among thought leaders due to the recent digital technology surge.

Microservices architectures can offer businesses numerous significant benefits, including:

- Systems built with a microservices-based architecture are inherently loosely coupled, which allows them to be easily built, changed and selectively scaled, which is critical for new digital applications. Because systems are broken into multiple services, it allows for frequent changes without affecting the entire system.
- The services are very simple and focused on a single business capability.
- Multiple developers and teams can deliver independently under this model, increasing the pace of change.
- Continuous delivery is enabled, allowing for frequent releases while keeping the rest of the system available and stable.
- Because each service is operationally encapsulated, development teams are free to change implementation details without worrying about inadvertent side effects that impact other services due to a shared infrastructure.

For all these reasons, microservices enable businesses with a unique opportunity to use IT as a tool to easily make functional changes and measure results.

One of the goals of microservices architectures is to build more robust systems by thinking about the operational aspects of architecture. In fact, microservices can be considered as the first post-DevOps architecture.

---

Microservices: An Anatomical View

**A monolithic architecture** puts all its functionality into a single process and scales by replicating monoliths on multiple levels.

**A microservice architecture** puts each element of functionality into a separate service and scales by distributing these services across servers, replicating as needed.

---

Figure 1
DevOps is an approach to software development that is focused on streamlined communication, collaboration, integration and automation (of both testing and coding), as well as measurement of cooperation between software developers and other IT functions. The term was popularized through a series of “DevOps Days,” starting in 2009 in Belgium. Since then, DevOps Days conferences have been held in many countries worldwide. (For more, read our white papers “How DevOps Drives Real Business Growth” and “DevOps Best Practices Combine Coding with Collaboration.”)

To succeed with microservices, development efforts must mesh with DevOps, making operational concerns a first-class citizen in architectural design. Microservices architectures assume continuous evolutionary change, making them less expensive and error prone.

Figure 2 offers a vertically partitioned reference architecture for building microservices.

**Best Practices for Designing a Microservices Architecture**

A microservices architecture can be expensive to implement due its distributed nature and operational complexity. Succeeding with microservices depends on the existence of certain architectural, infrastructure and process elements, including continuous integration, testing, logging and tracing, messaging, service registration and discovery, monitoring, etc.

**Design for Business Capabilities**

A microservices architecture typically requires dedicated teams delivering specific business capabilities; for example, payments, accounts, loan applications and disputes are all different business capabilities with different requirements that can be best addressed when teams are focused on building a set of capabilities.

Organizations need to avoid designing over technology layers, such as data access services, and should shift from a project-specific design mindset to a product-specific design approach. Services should be designed as an independent product delivering a business capability that is well documented, easy to use and responsible for one and only one business capability; this is known as the single responsibility principle (SRP). A good rule of thumb is that the microservice should be as small as possible but as big as necessary to represent the domain concept or business capability it represents.

**Microservices: An Illustrative View**

Figure 2
Design for Orchestration Layers

Considering the growing number of device types and the fact that more companies’ business strategies are dependent on providing value to customers on numerous devices, it is no longer optimal to have granular resource-based application programming interfaces (APIs) that closely represent the data model. Instead, APIs must be optimized for each of the targeted devices (see Figure 3).

Use Events

With microservices, everything is about services, and focused on request/response activities. By default, developers implement synchronous integration between the services because the requirements are typically explained to them in that manner.

The synchronous calls between services pose many challenges, especially the need to implement business transactions involving several services. Teams struggle to continuously deliver changes in an efficient and safe manner, which often results in a “monolithic” release process in which all services are deployed simultaneously, alongside the client application.

These issues can be avoided by using a model in which an upstream service publishes events, while downstream services subscribe to those events.

Implementing this model limits the boundaries of transactions and promotes decoupling among the services.

Embrace DevOps Practices

While a microservices architecture is a cleaner approach, it is also a more complicated one with more moving parts and heterogeneous languages, frameworks and service protocols. Organizations need to shift to a DevOps model, which recognizes that developers are in the best place to operationalize (write, build, deploy) their own code.

Use 12-Factor Apps

The 12-factor app is a methodology for building software-as-a-service apps. The methodology can be applied to apps written in any programming language, and used in any combination of backing services (database, queue, memory cache, etc.).

Monitor Everything

Because microservices involve more moving parts, it is important to measure everything efficiently and easily through mechanisms such as response time graphs, service failure graphs and dashboards.

Monitoring is about knowledge, control and reaction. Being able to understand the system’s...
behavior when it becomes a highly-distributed functionality requires a more structured approach. Key monitoring strategies that should be implemented with a microservices architecture include:

- **Use logs.** Log analysis tools are available, such as Kibana, a visual interface for interacting with large amounts of aggregated log data, or Splunk, a fully integrated solution for log management data collection, storage and visualization.
- **Use monitoring tools.** Examples include Datadog and AppDynamics.

**Design for Fail-over Scenarios**

Organizations should consider possible disruptions (i.e., network latency, data center shutdown) and test fail-over scenarios by documenting all testing steps and recovery execution.

This can be done by executing Gameday exercises, in which failures are purposefully injected into critical systems to help identify problems. This helps in fixing flaws and, over a period of time, improving the overall stability of the system.

**Service Documentation**

A service is only as good as its documentation. Documents should be easy to find, publicly accessible and able to show examples of complete request/response cycles.

Organizations should follow approaches to defining services using JSON, or a markdown language, which allow developers to describe the service interfaces and underlying data models in a way that enables more precise communication with others. Such approaches also help developers generate mock or production services, interactive documentation, code samples and potentially other tooling that will help in service/API initiatives.

**Use Circuit Breaker as a Design Pattern**

The Circuit Breaker design pattern is useful for point-to-point connections. This design uses a service client that talks to a service; it also has configurable time-outs and number of failures that can be tolerated. Circuit Breaker can prevent an application from repeatedly trying to execute an operation that is likely to fail, allowing it to continue without waiting for the fault to be rectified or wasting CPU cycles while it determines that the fault is long-lasting.

**Service Design Review**

By conducting a service design review, organizations can avoid duplicate services, and ensure service flexibility, service reusability and the service data model. A service governance board typically conducts these reviews. This review should also ensure that services do not deviate from approved design patterns.
Business Situation
A U.S.-based bank holding company specializing in credit cards, home loans, auto loans, banking and savings products wanted to enable innovation, improve speed-to-market and simplify middleware delivery to support new channels and evolving business needs. As part of its IT strategy, this large bank wanted to enable the broader Internet/partner development community to provide customer offerings based on the services exposed by the bank to support its loan volume growth goals (both auto and home).

Challenges
The IT landscape for home and auto loans consisted of about 150 services, addressing multiple applications used by internal and external clients. Existing SOA development and security approaches provided limited access to internal networks and did not scale effectively to solve certain use cases.

The services were designed per channel, providing the same business capability across multiple services, but they lacked governance. The services also lacked a bounded context concept, and the domain was spread across multiple services. There were no owners for some of the services as the developers had either left the organization or were engaged in other projects.

Solution
In this engagement, we started with a functional analysis of existing applications and services from a service design perspective and prioritized mobile application business requirements. A set of core services were selected based on business benefit and redesigned around the business capability. These services were exposed to external clients and partners adopting our API strategy. We implemented a microservices architecture with core services, composite services and API services (see Figure 4).
We used the following best practices to implement this microservices architecture:

- Defined an orchestration layer to handle channel-specific requirements.
- Documented the APIs using Swagger specifications to generate documentation in real time.
- Adopted DevOps practices to build and deploy the services, service configuration management and service monitoring using Splunk.
- Used IgniteXML as a repository for core services, as well as an API, and included service design review as part of the service/API delivery process.
- Used a developer portal for publishing the APIs for easy access by internal developers.
- Designed services as products, adopting the single responsibility principle; for example, a payment instructions service exposed functionality for payment instructions, including:
  - Create and delete a payment instruction with multiple payment options, such as weekly or monthly payment, one-time payment or a recurring payment.
  - Modify a payment instruction with options to modify all the instances of payments or only selected instances of payments.
  - Search for payments with multiple search options, such as search with payment state (historical, scheduled or pending or any combination), search with amount options (with less than or greater than amount), search with payment ID.
  - Execute a monthly, weekly or bi-monthly payment for an auto loan or home loan.
  - Modify or cancel an instance of a scheduled payment before it is executed.

Benefits

- The bank was able to improve customer reach and retention by exposing the informational and transactional services on multiple devices, such as mobiles and tablets.
- The project enabled an ecosystem for digital innovation by exposing services/APIs with external partners, thus contributing to volume growth goals. The bank simplified the home-buying experience for consumers by integrating real estate search capabilities with financing resources.
- The project delivered changes/enhancements to existing applications in a shorter amount of time. For example, the bank was able to integrate with cloud storage applications, such as Google Drive and Dropbox, so customers could access their loan documents from any device, in a very short period of time, since changes were required only to the loan application services portion of the application.
Looking Forward

Modern applications require a new foundation, built on scalable, resilient and adaptive services. Microservices offer an approach for delivering highly-scalable, cross-platform services that can be developed, deployed and scaled independently of one another to provide the modern application product team with the greatest amount of flexibility and control.

To successfully implement a microservices architecture, organizations must meet basic prerequisites that require an organizational shift, such as close collaboration between developers and operations. Areas to focus on include:

- **Rapid deployment:** With many services to manage, organizations need to be able to quickly deploy them, both to test and production environments.

- **Fast provisioning:** Along with continuous delivery, organizations must have automated mechanisms for machine provisioning, as well as for deployment, testing, etc.

- **Basic monitoring:** With more moving parts that must be coordinated in production, basic monitoring must be in place to detect problems quickly. This monitoring can include detecting technical issues such as service availability and business issues such as a sudden drop in transaction volume.

By moving to a microservices-based approach, organizations can address the business needs for operational flexibility, functional simplicity and continuous change that define today’s digital economy.

Footnotes


About Cognizant

Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting, and business process outsourcing services, dedicated to helping the world’s leading companies build stronger businesses. Headquartered in Teaneck, New Jersey (U.S.), Cognizant combines a passion for client satisfaction, technology innovation, deep industry and business process expertise, and a global, collaborative workforce that embodies the future of work. With over 100 development and delivery centers worldwide and approximately 219,300 employees as of September 30, 2015, Cognizant is a member of the NASDAQ-100, the S&P 500, the Forbes Global 2000, and the Fortune 500 and is ranked among the top performing and fastest growing companies in the world. Visit us online at www.cognizant.com or follow us on Twitter: Cognizant.

About the Authors

Mrudul Palvankar is a Senior Manager and Senior Architect in Cognizant’s Banking and Financial Services (BFS) Technology Architecture Office, where she heads the architecture and technology team. She has a post-graduate degree in business administration and has 15 years of software development experience across multiple industry segments. She is also TOGAF certified and is actively engaged in architecture transformation consulting and solution delivery for banking and financial services’ customers. Mrudul can be reached at Mrudul.Palvankar@cognizant.com.

Vivek Kant is Chief Architect and Technology Partner with Cognizant’s Banking & Financial Services business unit, where he leads technology strategy for clients across Europe and the APAC region. He has a post-graduate degree in software engineering and has over 16 years of experience spanning technology consulting, enterprise architecture and core software engineering. Vivek can be reached at Vivek.Kant@cognizant.com.