WHITEPAPER

ACHIEVE A SUCCESSFUL MICROSERVICES ARCHITECTURE

BENEFITS

- Easier development and maintenance
- Faster and independent deployment
- Independent scalability of application components
- Freedom from long-term technology commitment

facebook.com/redhatinc
@redhatnews
linkedin.com/company/red-hat

redhat.com

EXECUTIVE SUMMARY

Microservices architecture is a new architectural style for creating loosely coupled but autonomous services. Emerging trends in technology—such as DevOps, Platform-as-a-Service (PaaS), containers, and continuous integration and delivery (CI/CD) methods—let organizations create and manage these modular systems on an unprecedented scale that exceeds earlier approaches like service-oriented architecture (SOA). But organizations that refactor monolithic applications into microservices experience widely varying degrees of success. The key to using microservices effectively is a solid understanding of how and why organizations should use microservices to build applications.

IMPROVING SERVICE-ORIENTED ARCHITECTURE

Service-oriented architecture (SOA) is commonly defined as application components that communicate to provide services to other components over a network. The goal of SOA was to create resilient distributed applications without complex centralized components.

However, SOA was strongly coupled to organizational structures and applied to support new internal structures. Therefore, its success was highly dependent on the resulting, restructured organizational capabilities and the structure of the teams designing the architecture. Instead of creating loosely coupled but autonomous systems, SOA created highly fragile systems that required complex infrastructure. In addition, early SOA implementations created vendor lock-in, because proprietary middleware often focused on centralized logic, persistence, governance, and administration.

Microservices architectures are starting to deliver on the promises of SOA at every step of building applications, from development to deployment to operations. Microservices architecture focuses on simplifying technology to build complex systems with streamlined components. Centralized logic and integration infrastructure—based on heavyweight, non-standardized platforms—is replaced by communication via simple, standardized pipes, based on asynchronous HTTP or messaging protocols. SOAP, XML, and other heavyweight protocols and data formats are replaced by lightweight JSON over HTTP-based REST. Each microservice has its own data store, and centralized governance and persistence are not required.

Microservices use continuous integration (CI) and continuous delivery (CD) methodologies and practices, as well as several critical components that were not as common in SOA, such as:

- Polyglot programming and persistence.
- Containers or immutable virtual machines (VMs).
- Elastic, programmable Infrastructure-as-a-Service (IaaS) and PaaS.

ARE YOU READY FOR MICROSERVICES?

Has your organization:

- Built a well-structured monolithic application?
- Determined what need microservices will meet?
- Realigned teams around microservices?
- Adopted best practices in DevOps and CI/CD?
- Identified business boundaries within the application?
- Implemented microservices orchestration and management tools and processes?

AN INNOVATIVE SOLUTION FOR FLEXIBLE, RESPONSIVE I.T.

FASTER DEPLOYMENT

Microservices are smaller in scope, which is determined by a focus on domain boundaries and consistent domain modeling, and require less code. Deployment strategies including focused, self-contained archives—often packaged as Linux containers and CI/CD—lead to faster, more reliable deployments. As a result, the software development lifecycle becomes faster in general. New features and bug fixes, as well as fully tested security patches, are released more rapidly.

MODULAR CONTROL

With microservices, each service can scale independently to meet temporary or seasonal traffic increases, complete batch processing, and meet other business needs. Improved fault isolation restricts service issues, such as memory leaks or open database connections, to only affect that specific service. The scalability of microservices complements the flexibility of cloud services, letting you improve service while also handling more customers simultaneously without interrupting service.

MORE CHOICE

Open source technology solutions and organizational methods are leading the microservices market. As a result, microservices reduce vendor lock-in and eliminate long-term technology commitment, letting you choose the tools you need to meet IT and business goals.

BUILDING A SOLID FOUNDATION FOR MICROSERVICES

To achieve success with microservices, organizations must first create a solid foundation for their monolithic architecture. Modularity, domain boundaries, and fundamental distributed systems theory must be considered and established to achieve the full benefits of microservices.

In addition, microservices create the most benefit for more complex systems. While each service is fully independent, there are operational requirements that must be met, including capabilities such as:

- DevOps.
- PaaS.
- Containers or immutable VMs.
- Service replication, registry, and discovery.
- Proactive monitoring and alerts.

Because meeting these requirements could be a significant investment without an immediate return, using microservices may not be cost-effective for every team or project. Evaluating a monolith-first approach ensures that applications are built following solid design principles and that domain boundaries are properly defined, letting you gradually transition to a microservices architecture if needed for scalability. For example, even a basic shopping cart app should contain:

- Separation of concerns.
- High cohesion and low coupling using well-defined application program interfaces (APIs).

Organizations achieve success with microservices because of their organizational structure, not their technology. Flexible teams with flat organizational structure, cross-functional abilities, and autonomy are key.

- Separate interfaces, APIs, and implementations, following the Law of Demeter, or principle of least knowledge.
- Domain-driven design that groups related objects.

Once a monolithic application that needs to be scaled has been built according to sound software architecture principles, it can be refactored into microservices. The most effective refactoring method consists of the following steps:

1. Identify the business boundaries and semantic differences in the application domain and begin decomposing each domain into its own microservice.
2. Find the component that undergoes the most requested changes—such as business rules updates associated with price calculations or regulatory changes—or is often patched to resolve security vulnerabilities.
3. After defining basic domain-based microservices, refine the APIs used to let services interact. Compose these APIs using aggregator, proxy, chained, branch, event-driven, and other design patterns.

ALIGNED, SKILLED TEAMS

Organizations achieve success with microservices because of their organizational structure, not their technology. Flexible teams with flat organizational structure, cross-functional abilities, and autonomy are key.

Achieving an effective, skilled team requires realigning staff around functionality rather than architecture—for example, Amazon's "two-pizza team" of 8 to 10 people who are responsible for the creation and maintenance of their service. Conway's Law dictates that organizations can only produce designs that mimic the organization's structure. Teams that are segmented—for example, into development, operations, quality assurance, and security—experience limitations to flexibility and delays in progress.

Establishing a DevOps practice before transitioning to microservices can mitigate or prevent these issues—and avoid the creation of a failed SOA instead of an effective microservices architecture—by determining communication strategies in advance.

EFFECTIVE MANAGEMENT TOOLS

In addition to well-designed software and an effective, organized team, achieving a highly scalable architecture requires tools to help you manage additional services and application components, including:

- Service registry and discovery tools, such as Kubernetes.
- Packaging standards for containerizing applications, such as Docker, and orchestration tools for replicating containers at scale, such as Kubernetes. OpenShift by Red Hat includes both of these proven open source technologies.
- CI environment creation tools, such as Jenkins or Shippable for Docker and Kubernetes.

- Dependency resolution tools, such as Nexus.
- Failover and resiliency tools, including libraries such as Hystrix and Ribbon.
- Service monitoring, alert, and event tools, such the ELK (ElasticSearch, LogStash, and Kibana) stack.

DATA MANAGEMENT

Another important consideration for transitioning to microservices is data management. Unlike SOA, microservices do not share data. Instead, each microservice has a separate physical data store and polyglot persistence that lets a variety of database engines run under each microservice. As a result, a data store can be chosen on a per-service basis, instead of storing all data in a corporate relational database management system (RDBMS).

However, maintaining multiple copies of an enterprise database can increase licensing costs and complexity. In addition, data stores may need to be aligned for consistency. Generic extract, transform, and load (ETL) or data virtualization tools can help with data normalization, and event sourcing is a well-known design pattern that helps align data stores to adjust to retroactive changes.

CONCLUSION

Microservices architecture can offer organizations many advantages, from independent scalability of diverse application components to faster, easier software development and maintenance. But microservices do not always benefit every team or project and could be a significant investment without an immediate return. Transitioning to microservices should be a gradual process, and refactoring only pieces of existing applications—without a full transition—can also bring benefits. To achieve success with microservices, organizations must first build a well-designed application according to existing platform standards, then refactor the application into a collection of microservices as necessary to meet business needs. With the right people, processes, and tools, microservices can deliver faster development and deployment, easier maintenance, improved scalability, and freedom from long-term technology commitment.

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@redhatnews
linkedin.com/company/red-hat

redhat.com
INC0336100_0216