Microservices: The Evolution of Building Modern Applications

June 2016

Table of Contents

Introduction

As enterprises work to replicate the development agility of internet companies and innovate in highly competitive markets, application development has grown increasingly complex. The large, monolithic codebases that traditionally power enterprise applications make it difficult to quickly launch new services. Siloed and potentially distributed development and operations teams present organizational alignment problems. On top of this, users are more demanding than ever – enterprises need to scale effectively and monitor deployments to ensure customers are provided with high performance and a consistent experience. Of course, all this needs to be done while providing always-on service availability.

Due to these trends, there is demand for a software architecture pattern that can handle the requirements of the modern age. Monolithic architectures have been the traditional approach, but limitations with scaling, difficulties in maintaining a large codebase, high risk upgrades, and large upfront setup costs have compelled enterprises to explore different approaches.

In the last few years, microservices have come to the forefront of the conversation. They have been rapidly adopted, due to their ability to provide modularity, scalability, high availability, as well as facilitate organizational alignment.

This whitepaper discusses the background behind microservices, their advantages, and how new generations of technologies, such as MongoDB, enable them.

The Monolith

Before microservices, a common approach to design an application was to use a monolithic architecture. In this mode of development, the application is developed, tested, packaged, and deployed as a single unit. Codebases are compiled together, and the application is deployed as one entity. Scaling required copying instances of the application binaries and the required libraries to different servers, and the application code typically ran as a single process. Continuous delivery — an approach that involves fast, iterative software development and safe updates to the deployed application — was challenging since the full monolithic application stack needed to be recompiled,

relinked, and tested for even the smallest incremental release.

What are Microservices?

Microservices is a software architecture where applications are broken down into small autonomous services. Services are typically focused on a specific, discrete objective or function and decoupled along business boundaries. Separating services by business boundaries allows teams to focus on the right goals and also ensures autonomy between services. Each service is developed, tested, and deployed independently, and services are usually separated as independent processes that communicate over a network via agreed APIs, although in some cases that network may be local to the machine.

Figure 1: Online Store Monolithic Architecture

To illustrate the concept further, consider a common business service such as a shopping cart for an online store. When displaying a product page, the mobile application would need to interact with four different services: User Account, Product Catalog, Inventory, and Shopping Cart. In **Figure 1**, these services are aggregated into a monolithic application. Services are not decoupled but instead the web pages, classes, and configuration files are all packaged together in a single WAR (Web application ARchive) file. Any small code change will require regression testing on all features, not to mention recompiling and relinking the whole application.

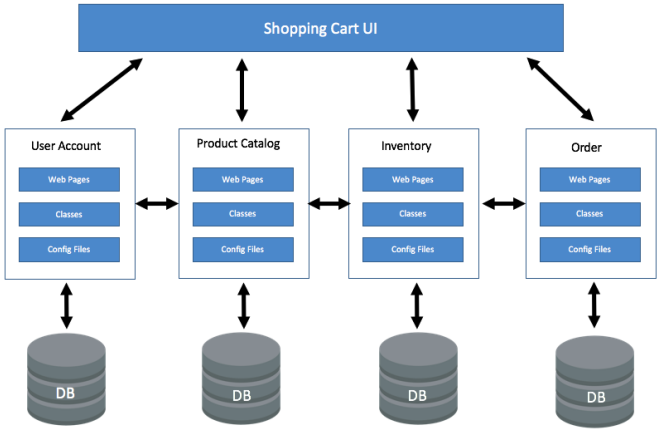

Figure 2: Online Store Microservices Architecture

As illustrated in **Figure 2**, in a microservices architecture, the User Account, Product Catalog, Inventory, and Shopping Cart functions are decoupled into four separate services (i.e separate WAR files), and organized by business boundaries, which makes it clear where the code resides for specific functionality. Web pages, classes, and configuration files specific to a service are grouped together. This allows the different services to evolve and iterate independently as each team can develop, test, and deploy code separately.

Each service also connects to its own database. The reason is that in a shared database environment, a catastrophic failure with the database infrastructure has the potential to affect multiple microservices and result in a substantial outage. Thus, it is recommended to decouple any shared databases so that each microservice has its own database.

Microservices grew from **Service Oriented Architecture (SOA)**, which gained popularity in the early 2000s and emerged as a way to combat large monolithic applications. Key differences between SOA and microservices are:

- SOAs are stateful, while microservices are stateless
- SOAs tend to use a enterprise service bus for communication, while microservices use a less elaborate and simple messaging system

- SOAs may have hundreds or thousands of lines of code, while microservices could have less than one hundred lines
- SOAs put a greater emphasis on reusability (i.e. runtime code, databases), whereas microservices focus on decoupling as much as possible
- A systematic change in a SOA requires modifying the monolith, whereas a systematic change in a microservice is to create a new service
- SOAs use traditional relational databases more often, while microservices gravitate more towards modern, non-relational databases. Further sections will cover the advantages of non-relational databases over relational databases in a microservices architecture

Many architects found that SOAs suffered problems with communication protocols and lacked sufficient guidelines on effectively separating services, which laid the foundation for microservices to emerge as a best practice method to implement a truly SOA.

New Technologies Enable Microservices

The downsides of deploying and provisioning hundreds and potentially thousands of services did not outweigh the benefits gained with a microservices architecture (faster development, scalability).

The emergence of technologies such as containers (Docker, LXC) and orchestration frameworks (Kubernetes, Mesos) mitigate many of the problems that prevented using microservices architectures in the past.

Containers are lightweight run-time environments that provide isolation and scalability with minimal impact to performance and capacity. Packaging is simplified as the same environment can simultaneously host development, support, test, and production versions of the application, so that going from dev to test to QA to production is easier. Containers work very well in a microservices environment as they isolate services to an individual container. Updating a service becomes a simple process to automate and manage, and changing one service will not impact other services, provided that APIs are maintained.

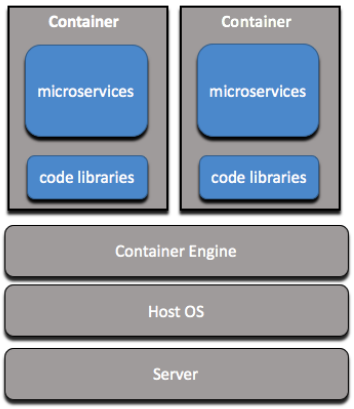

Figure 3: Microservices in a container environment

When organizations start running containers at scale, many look to orchestration frameworks to help manage the increased complexity. Orchestration frameworks help deploy and manage containers: provision hosts, instantiate containers, handle failures, and provide automated scaling. Kubernetes and Mesos are popular orchestration frameworks that make it easier to deploy containers at massive scale in a microservice environment.

To learn more about building microservices architectures with containers and MongoDB, download our guide: Enabling Microservices: Containers and Orchestration Explained

Benefits of Microservices

Many organizations can better meet the needs of modern application development by implementing microservices. The benefits include:

Faster Time To Market: In a monolithic application, any small change in the application will require redeploying the entire application stack, which carries higher risk and complexity. This results in longer release cycles, as changes may be batched together and not released until reaching a minimum threshold. With microservices, a small change to a service can be committed, tested, and deployed immediately since changes are isolated from the rest of the system.

Continuous integration — a software practice of integrating and testing developer changes to the main code branch multiple times a day — is much simpler and faster as there are fewer functions to test. This results in a more iterative release cadence as less code needs to be compiled and retested. Orchestration tools such as Kubernetes facilitate faster time to market by automating the on-line, rolling upgrade of containers, and providing the ability to roll back any changes should they be necessary.

Flexibility and Scalability: Monolithic applications require all components of the system to scale together. If one service requires extra performance, the only option is to scale all the services rather than the individual service that needs additional capacity. With microservices, only the services that require extra performance need to be scaled. Scaling is achieved by deploying more containers, enabling more effective capacity planning, less software licensing costs, and lower TCO as the service and hardware can be matched more appropriately.

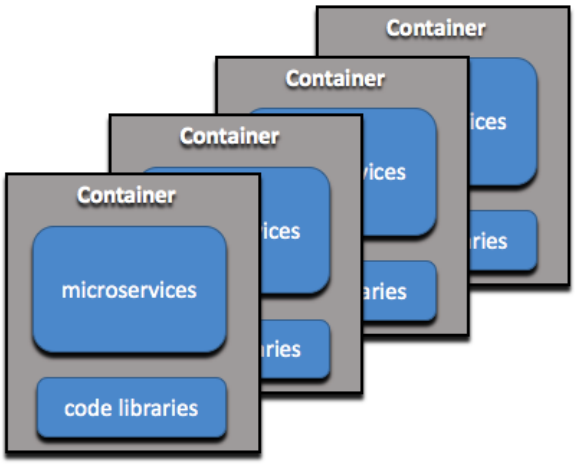

Figure 4: Scaling of microservices and containers

Resiliency: A major issue with monolithic applications is that if a service fails, the whole application may be compromised. In microservices, service boundaries serve as natural isolation barriers to prevent cascading failures from bringing down the whole system. If using containers, orchestration frameworks can provide added resiliency: when one container fails, a new one is started, restoring full redundancy and capacity.

Alignment With Organization: Microservices enable better alignment of the architecture to the organization, as team sizes can be optimally defined to match the required tasks. Teams can be broken down into smaller groups and focus on a single component of the application. This is especially useful for distributed teams. For example, if a team in Singapore handles three services, while a team in San Francisco handles five services, each team can release and deploy features and functionalities independently. This helps break down silos between teams and fosters better collaboration as cross discipline teams (Ops, Dev, QA) collectively own the services. This also ensures that the communication between teams matches the communication through the services' APIs. **Essentially, the APIs between services define a contract between development teams on what each service should provide to others.**

Reduction in Cost: By using containers, applications and environments (design, test, production, support) can better share the same infrastructure, resulting in increased hardware utilization and reduced costs due to administrative simplification. In addition, microservices also help reduce technical debt. With a monolithic application, there are costs (time, resources) associated with refactoring code for a large application. By breaking the application into API accessible microservices, code refactoring can be done service by service, resulting in less time maintaining and updating code.

How MongoDB Enables Microservices

There are some fundamental technology principles that are required to ensure companies can reap the advantages of microservices, specifically around a flexible data model, redundancy, automation, and scalability.

Flexible Data Model: MongoDB's dynamic schema is ideal for handling the requirements of microservices and continuous delivery. When rolling out a new feature that changes the data model, there's no requirement to update all of the existing records, something that can take weeks for a relational database. Developers can quickly iterate

and model data against an ever changing environment, resulting in faster time to market and greater agility.

Redundancy: Due to the distributed nature of microservices, there are more potential failure points, such as more network links, and thus, microservices need to be designed with redundancy in mind. MongoDB is well suited for this requirement, as it provides built-in redundancy through MongoDB replica sets. Replica sets not only provide greater resilience to failure, but also provide disaster recovery with multi-data center deployments and the ability to isolate operational workloads from analytical reporting in a single database cluster.

Monitoring and Automation: With a small number of services, it is not difficult to manage tasks manually. As the number of services grow, productivity can stall if there is not an automated process in place to handle the growing complexity. Choosing technology that handles monitoring and automation is key to ensuring devops teams can remain productive, especially as the environment becomes more complex.

MongoDB Ops Manager (also available as the hosted Cloud Manager service) features visualization, custom dashboards, and automated alerting to help manage a complex environment. Ops Manager tracks 100+ key database and systems health metrics including operations counters, CPU utilization, replication status, and any node status. The metrics are securely reported to Ops Manager where they are processed and visualized.

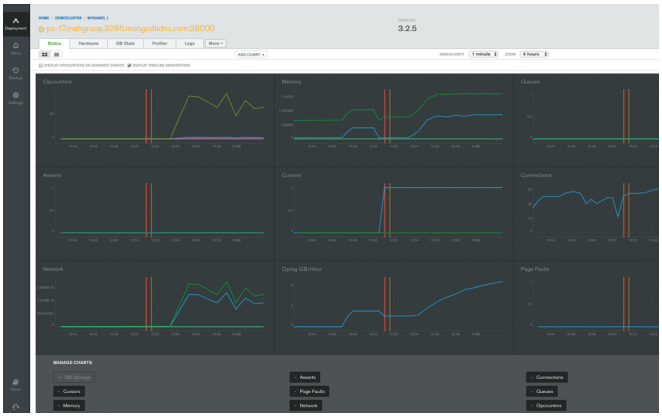

Figure 5: Ops Manager provides real time & historic visibility into the MongoDB deployment

Integration with existing monitoring tools is also straightforward via the Ops Manager RESTful API, and with packaged integrations to leading Application Performance Management (APM) platforms such as New Relic. This integration allows MongoDB status to be consolidated and monitored alongside the rest of your application infrastructure, all from a single pane of glass.

Scalability: Scaling to meet extra demand is a requirement of any IT environment, and microservices are no exception. MongoDB provides a scalable solution that automatically partitions and distributes the database across nodes, which can easily serve IT infrastructures that require dynamic and high-performance capabilities. Additionally, MongoDB is ideally suited to scale-out on commodity hardware with auto-sharding, which, if needed, allows the service to be easily distributed across different geographic regions. This is better from the monolithic, scale up design of traditional RDBMS because scaling in MongoDB is automatic and transparent.

To capture more business benefit, many organizations are also shifting microservices to the cloud. The dynamic nature of the cloud allows enterprises to spin up and down instances, while providing continuous availability in case of any failures.

Considerations Before Moving to Microservices

Though microservices offer many advantages, they are not appropriate for all deployments. There are several considerations to keep in mind before implementing a microservices project:

Monitoring Challenges: One of the biggest challenges for microservices is effectively monitoring the overall system. Monitoring one or two services is relatively straightforward, but effectively monitoring many services can be very challenging. Not only are there are more servers to monitor, but there are also more log files to analyze, as well as additional opportunities for network partitions. Traditional approaches to monitoring stats, such as CPU, memory, and network latencies are important, but enterprises also need to expand ways to view metrics about the system and how it behaves over a long period of time. Automating the monitoring process can help mitigate

some of these challenges and reduce operational overhead.

High Developer Skillset: Microservices is implemented on distributed systems, which are necessarily more complex. Network latency, hardware failures, unreliable networks, asynchronicity, and fault tolerance need to be dealt with gracefully and appropriately. In order to handle the added complexity, developers need to have a strong operations and production background. Developers can no longer create the application and hand it off to the operations team; they need to understand the interdependencies between DevOps, testing, and release in order to properly design a service. Before implementing a microservices architecture, it is important to determine if your team has the right capabilities to handle the associated complexities.

More Operations Overhead: For a given monolithic application, it may require one application server cluster with a few processes, while a microservice application may comprise 50 services and 200 processes after adding in resiliency. Operating and monitoring all these new process can be a daunting task. Additionally, services need to be tested and quickly propagated through the continuous delivery pipeline, which requires proper tooling and skills.

Incorrect Service Boundaries: It is imperative to establish the proper service boundaries during the design phase. A common problem is to create services from internal components without considering the proper service boundaries. As more functionality gets added, there is a risk that the team ends up building a giant distributed monolith. Getting the service boundaries incorrect may result in higher costs, overcoupled services, and more testing complexity.

MongoDB Microservice Deployments

MongoDB is deployed by thousands of organizations around the world, including over half of all Fortune 100 companies. Many enterprises use MongoDB in a microservices architecture to achieve their business and deployment goals.

Backcountry.com is an online specialty retailer that sells outdoor clothing and gear for recreations such as skiing, mountaineering, rock climbing, fishing, etc. In 2014, Backcountry found itself naturally gravitating to a microservices architecture as teams became more distributed. Additionally, difficulties leveraging Backcountry's Oracle ATG platform propelled it to use MongoDB to accelerate the transformation. One of the major challenges was the schema management burden. As more and more developers joined and made contributions to the code, the schemas became convoluted and harder to maintain; contributing to 20% of the Scrum backlog. Taking advantage of MongoDB's flexible data model, Backcountry was able to iterate faster, reduce development time, and mitigate technical debt.

In order for Gap Inc to operate more efficiently, the architecture team converted its purchase order system from a monolithic application to a microservices architecture. Gap selected MongoDB to power its microservices and MongoDB's flexible data model was instrumental in helping Gap develop a new purchase order system in 75 days, a record for the company. Additionally, five months into the project, new requirements mandated adding new types of purchase orders. Leveraging the flexibility of MongoDB's dynamic schema, developers were able to add the new features in days, instead of months as it had previously taken.

Comparethemarket.com is one of the UK's leading providers for price comparison services and uses MongoDB as the operational database behind its large microservice environment. Service uptime is critical, and MongoDB's distributed design is key to ensure that SLA's are always met. Comparethemarket.com's deployment consists of microservices deployed in AWS. Each microservice, or logical grouping of related microservices, is provisioned with its own MongoDB replica set running in Docker containers, and deployed across multiple AWS Availability Zones to provide resiliency and high availability. MongoDB Ops Manager is used to provide the operational automation that is essential to launch new features quickly: deploying replica sets, providing continuous backups, and performing zero downtime upgrades.

fuboTV is a streaming service in North America that streams sports content from soccer leagues all over the world and uses MongoDB as the core database for its

microservices architecture. The traffic profile of the fuboTV streaming service is extremely bursty with the site typically handling 100x normal traffic volumes ten minutes before a match. To keep pace with business growth and demanding software release schedule, fuboTV migrated its MongoDB database to Docker containers managed by the Kubernetes orchestration system on the Google Cloud Platform.

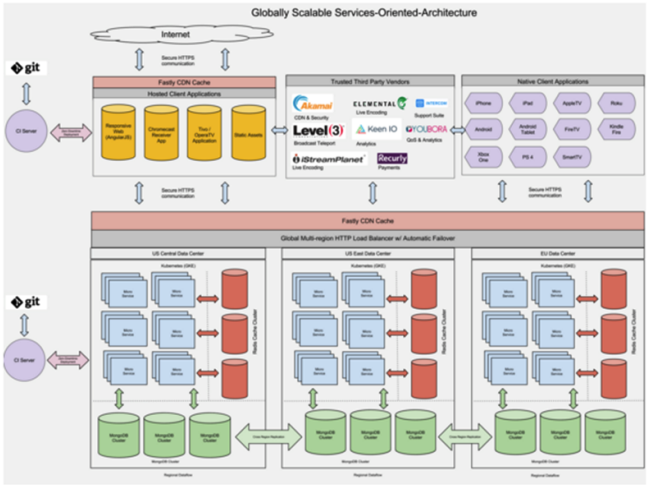

Figure 6: fuboTV Microservices Architecture

This brings high levels of flexibility, efficiency, and uptime to fuboTV. Using containers managed by Kubernetes, fuboTV can provision all of its environments – development, test, QA and production – to a single cluster of physical hosts. Kubernetes scheduler is used to precisely control resource allocation across all of its apps, enabling fuboTV to maximize utilization and reduce costs. Kubernetes replication controller automatically reschedules containers if an instance fails — enabling fault resiliency and continuous availability. Data redundancy is provided by MongoDB replication within the replica set. This enables fuboTV to have zero downtime as it deploys and upgrades its applications.

OTTO is top German retailer for fashion and lifestyle goods that has two million daily site visitors. OTTO's problem was that it had parallel teams spanning multiple business domains (business, project management, IT); these teams had various business problems but all needed to deliver results quickly. Independently, all the teams chose MongoDB as the best tool to quickly and easily achieve results. With loosely coupled teams, architecture, and operations, OTTO removed the bottleneck to deploy and test. Teams could quickly and iteratively correct errors and support continuous delivery. MongoDB was the driving force to enable OTTO's business, IT, and project management teams to deliver fast results, drive development agility, and allow teams to innovate risk-free.

Summary

A microservices architecture has many advantages over a monolithic architecture, but that does not imply microservices do not come without their own challenges. Proper planning and application decoupling is required to ensure that a microservices architecture will achieve the desired results. MongoDB is well suited for a microservices architecture with its ability to provide a flexible schema, redundancy, automation, and scalability. Together, MongoDB and microservices can help organizations align teams effectively, achieve faster innovation, and meet the challenges of a demanding new age in application development and delivery.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than a third of the Fortune 100.

To start using MongoDB 3.2 with microservices as quickly and efficiently as possible, bring in the experts. MongoDB's consulting engineers can deliver a private training on 3.2 features and how to deploy microservices with MongoDB. Learn more on the MongoDB 3.2 Upgrade Services.

We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can

scale up and down when you need to, with no downtime, full security, and high performance.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Professional helps you manage your deployment and keep it running smoothly. It includes support from MongoDB engineers, as well as access to MongoDB Cloud Manager.

Development Support helps you get up and running quickly. It gives you a complete package of software and services for the early stages of your project.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)

New York • Palo Alto • Washington, D.C. • London • Dublin • Barcelona • Sydney • Tel Aviv
US 866-237-8815 • INTL +1-650-440-4474 • info@mongodb.com