Serverless Architectures: The Evolution of Cloud Computing

December 2017

mongoDB

Table of Contents

Introduction

Since the advent of the computer, building software has been a complicated process. Over the past decade, new infrastructure approaches (IaaS and PaaS), software architectures (SOA and Microservices), and methodologies (Agile, Continuous Delivery and DevOps) have emerged to mitigate the complexity of application development. While microservices has been the hot trend over the past couple of years, serverless architectures have been gaining momentum by providing a new way to build scalable and cost effective applications. Serverless computing frees developers from the traditional cost of building applications by automatically provisioning servers and storage, maintaining infrastructure, upgrading software, and only charging for consumed resources.

This whitepaper discusses the background behind serverless computing, it's advantages, best practices, and key considerations when evaluating a database with your serverless environment.

The Evolution of Cloud Computing

In order to understand how serverless computing fits with modern application development, it is helpful to first understand the evolution of cloud computing.

Initially, enterprises leveraged data centers to abstract the physical hosting environment. The hardware was the unit of scale, which meant that in order to scale out your application you would allocate more compute, storage, and networking resources to handle the required load.

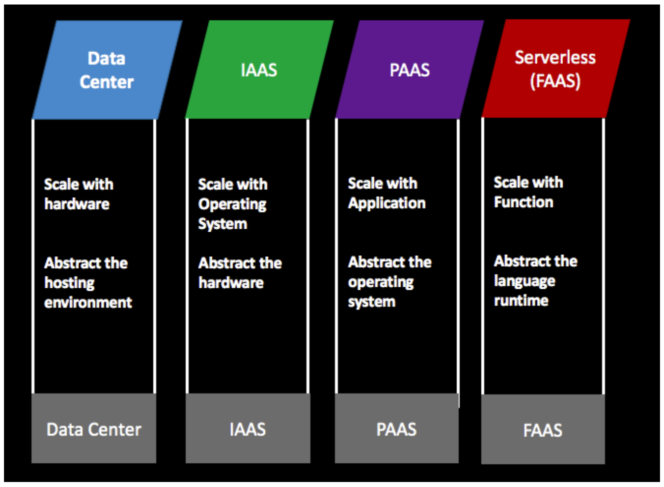


Figure 1: Evolution of Cloud Computing

As new technologies emerged, enterprises started moving to an Infrastructure-as-a-Service (IaaS) model, which abstracted away the physical hardware and made the operating system the unit of scale. Virtual machines (VMs), which are packaged operating systems that run an application, were used to emulate dedicated hardware resources. If an application required more resources, then a VM could be resized or other VMs provisioned to handle the load.

With Platform-as-a-Service (PaaS), the operating system was abstracted away, and the application became the unit of scale. There was no longer any need to deal with different operating system versions and middlewares, as the application would be allocated more resources if the load increased.

Now with serverless computing (also referred to as Function-as-a-Service), the language runtime is abstracted away and the function is the unit of scale. The cloud provider takes responsibility for the vast majority of the IT stack. If the function needs to be scaled, the cloud provider allocates the appropriate resources and creates copies of the function to handle the required load.

What is Serverless Computing?

Serverless computing is the next layer of abstraction in cloud computing. It does not mean that there are no servers. Instead, the underlying infrastructure (physical and virtual hosts, virtual machines, containers), as well as the operating system, are abstracted away from the developer. Applications are run in stateless compute containers that are event triggered (e.g. a user uploading a photo which triggers notifications to his/her followers). Developers create functions and depend on the infrastructure to allocate the proper resources to execute the function. If the load on the function grows, the infrastructure will create copies of the function and scale to meet demand.

Serverless computing supports multiple languages so developers can choose the tools they are most comfortable with. Users are only charged for runtime and resources (e.g. RAM) that the function consumes; thus there is no longer any concept of under or over provisioning. For example, if a function runs for 500ms and consumes 15 MB of RAM, the user will only be charged for 500 ms of runtime, and the cost to use 15 MB of RAM.

Serverless architectures are a natural extension of microservices. Similar to microservices, serverless architecture applications are broken down into specific core components. While microservices may group similar functionality into one service, serverless applications delineate functionality into finer grained components. Custom code is developed and executed as isolated, autonomous, granular functions that run in a stateless compute service.

To illustrate this point, let's look at a simple example of how a microservice and serverless architecture differ.

In Figure 2, a client interacts with a "User" microservice. A container is pre-provisioned, and all of the functionality of the "User" service resides in the container. The service consists of different functions (update_user, get_user, create_user, delete_user) and is scaled based on the overall load across the service. The service will consume hardware resources even while idle, and the user will still be charged for the underutilized resources.

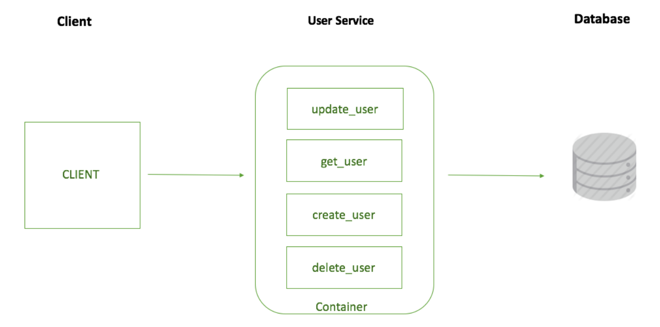


Figure 2: Microservices Architecture

For a serverless architecture, the "User" service would be separated into more granular functions. In Figure 3, each API endpoint corresponds to a specific function and file. When a "create user" request is initiated by the client, the entire codebase of the "User" service does not have to run; instead only create_user.js will execute. There is no need to pre-provision containers, as standalone functions only consume resources when needed, and users are only charged on the actual runtime of their functions.

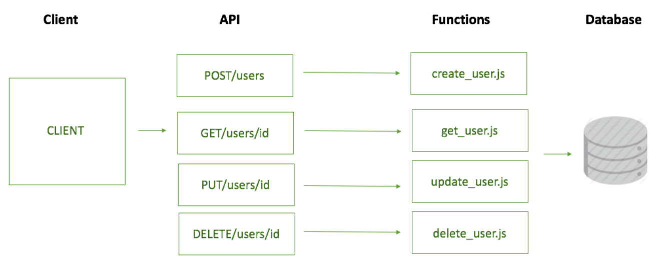


Figure 3: Serverless Architecture

Benefits of Serverless Computing

Costs Scale With Usage: One of the biggest benefits of serverless computing is that you only pay for the runtime of your function. There is no concept of "idle" resources as you are not charged if the function is not executed. This is especially helpful for applications that are only used a few times an hour, which means any dedicated hardware, VMs, or containers would be sitting idle for the majority of the time, and a user would be charged for underutilized resources. With serverless computing, enterprises could build out an entire infrastructure and not pay for any compute resources until customers start using the application.

Elastic Scalability: Elastic scalability is also simple with a serverless architecture. If a function needs to scale, the infrastructure will make copies of the function to handle the load. An example of this could be a chatbot that responds to weather requests. In a serverless architecture, a chatbot function would handle the response by retrieving the user's location and responding back with the temperature. For a few requests this is not a problem, but what happens if the chatbot service is flooded with thousands of request a second. For this scenario, the chatbot function would automatically scale by instantiating thousands of copies of the function. Once the requests have subsided, the environment would automatically terminate the idle instances and scale down, allowing costs to scale proportionally with user demand.

Rapid Development and Iteration: Serverless computing is ideal for companies that need to quickly develop, prototype, and iterate. Development is faster since there aren't any dependencies on IT Ops. Functions are typically single threaded, which makes debugging and deploying functions simpler. The build process is also broken down into smaller and more manageable chunks. This increases the number of changes that can be pushed through the Continuous Delivery pipeline, resulting in rapid deployment and more iterative feedback.

Less System Administration: Serverless doesn't mean that you completely obviate the operational element of your infrastructure, but it does mean that there is less system administration. There are no servers to manage, provision, and scale, as well as no patching and upgrading. Servers are automatically deployed in multiple availability zones to provide high availability. Support is also streamlined; if there is an issue in the middle of the night it is the responsibility of the cloud provider to resolve the problem.

Developer Productivity: By using a serverless architecture, developers can focus more on writing code than having to worry about managing the operational tasks of the application. This allows them to develop innovative features and focus on the core business logic that matters most to the business.

Serverless Computing Principles

Stateless Single-Purpose Functions: For a serverless architecture, it is good practice to design functions around the Single Responsibility Principle (SRP). A function that is developed to perform a specific task is easier to test, deploy, release, and is also more robust. Functions designed with well defined interfaces are also more likely to be re-used. From a deployment unit point of view, serverless functions are "stateless", which means local resources and processes will not be retained after the function executes. This allows the platform to quickly scale to handle an increasing number of events or requests.

More Front-End Functionality: In serverless architectures, functions that terminate faster are less expensive. Creating more functionality in the front-end helps to reduce costs by minimizing the number or complexity of function requests. This contrasts the typical application models where all requests flow through the back-end layer before being routed to the database. The emergence of rich client-side application frameworks, such as AngularJS, make it possible to seamlessly execute complex functionality within the browser. Back-end logic is decoupled from the server and moved to the front-end, allowing the front-end to communicate with more services. This reduces the number of hops required to access a resource and results in better performance and a richer user experience.

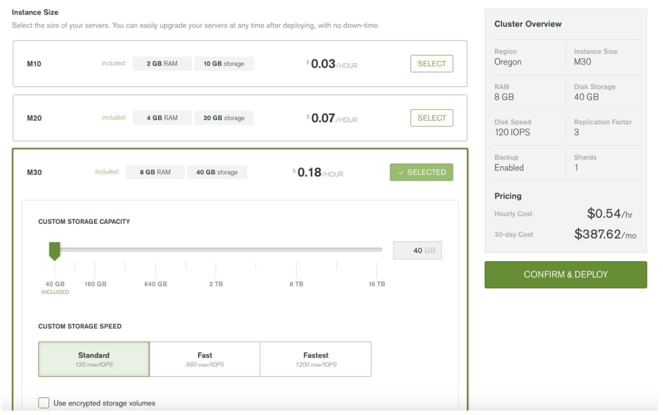
Design Event Driven Pipelines: Building event driven, push based pipelines is powerful and allows you to scale out architectures easily. An event driven pipeline is a chain reaction of events that propagate down the pipeline without any user input. For example, a user could upload a video file which kicks off a function that takes the file and video encodes it into multiple formats: 480p, 720p, 1080p. The newly encoded videos are saved to a storage service, which triggers another event. A function responds to the event and creates metadata about the videos that is pushed to a database. Metadata is saved to the database, which triggers another function that creates an email notification sent to the user confirming the video has been encoded to the different formats.These actions all happen asynchronously without any user input. With event driven pipelines, there is no need to manage servers or event queues, and applications can respond quickly to events and scale as needed.

Embrace Third Party Services: Serverless computing vendors may not comprise all the technologies that developers are familiar with or need. Leveraging third party services and applications allow developers to pick the right tools to help accomplish the task at hand. There are many services that developers can leverage from Auth0 for authentication to Braintree or Stripe for payment processing. It is much more useful for a developer to spend time solving a problem unique to their domain, than it is to recreate functionality that has already been built by someone else.

MongoDB Atlas and Serverless Computing

With MongoDB Atlas, users can leverage the rich functionality of MongoDB — expressive query language, flexible schema, always-on availability, distributed scale-out — from a serverless environment. MongoDB Atlas is a database as a service and provides all the features of MongoDB without the heavy lifting of operational tasks. Developers no longer need to worry about provisioning, configuration, patching, upgrades, backups, and failure recovery. Atlas offers elastic scalability, either by scaling up on a range of instance sizes, or scaling out with automatic sharding, all with no application downtime.

Setting up Atlas is simple.



Instance Size
Select the size of your servers. You can easily upgrade your servers at any time after deploying, with no down-time.

M10	included: 2 GB RAM	10 GB storage	$0.03/HOUR	SELECT
M20	included: 4 GB RAM	20 GB storage	$0.07/HOUR	SELECT
M30	included: 8 GB RAM	40 GB storage	$0.18/HOUR	✓ SELECTED

CUSTOM STORAGE CAPACITY: 40 GB

CUSTOM STORAGE SPEED: [✓ Standard 120 max/IOPS] Fast 660 max/IOPS Fastest 1200 max/IOPS

☐ Use encrypted storage volumes

Cluster Overview

Region	Oregon
Instance Size	M30
RAM	8 GB
Disk Storage	40 GB
Disk Speed	120 IOPS
Replication Factor	3
Backup	Enabled
Shards	1

Pricing

Hourly Cost	$0.54/hr
30-day Cost	$387.62/mo

Figure 4: Provisioning MongoDB Atlas cluster

Select the instance size that fits your application needs and click “CONFIRM & DEPLOY”.

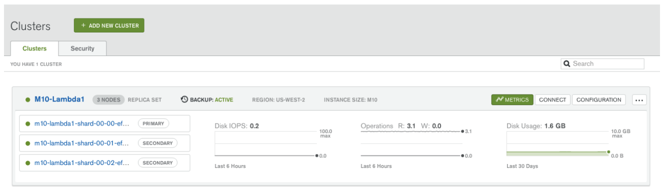


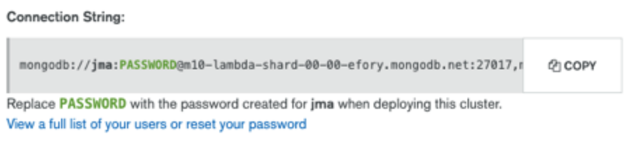
Figure 5: MongoDB Atlas Cluster Monitoring

MongoDB Atlas provides many benefits for those interested in building a serverless architecture:

Vendor Independence: Cloud providers typically only offer databases specific to that provider, which may not fit with what a developer needs. MongoDB Atlas provides independence from the underlying cloud provider and empowers developers to choose the appropriate tools for their needs. Developers can leverage the rich functionality of MongoDB's query language and flexible data model, without worrying about the operational tasks of managing the database. If you decide to shift to your own infrastructure, or another cloud provider, you won't have to repopulate your data on a different database. MongoDB Atlas is currently available only on AWS, with support for Microsoft Azure and Google Cloud Platform (GCP) coming soon.

MEAN Stack: Serverless architectures accelerate the trend of shifting business logic from the back-end to the front-end. This makes the choice of front-end framework much more important. AngularJS, is ideally suited for this requirement, and is a popular front-end for serverless architectures. AngularJS is a structural Javascript framework for dynamic web applications that provides interactive functions and AJAX — a technique for creating fast and dynamic web pages — rich components. Combined with NodeJS, ExpressJS, and MongoDB, these tools form the MEAN stack (MongoDB, ExpressJS, AngularJS, NodeJS). There are huge advantages to using JavaScript and JSON throughout your serverless stack. Someone working on the front-end can easily understand the function (back-end) code and database queries. Additionally, using the same syntax and objects through the whole stack frees your team from understanding multiple language best practices, as well as reduces the barrier to entry to understand the codebase, resulting in higher software performance and developer productivity.

Rapid Deployment: With MongoDB Atlas, a MongoDB cluster can be provisioned and deployed in minutes and even seconds. Developers no longer need to worry about configuring or managing servers. Integrating MongoDB Atlas into your serverless platform is as simple as passing the connection string into your serverless application.


Connection String:

```
mongodb://jma:PASSWORD@m10-lambda-shard-00-00-efory.mongodb.net:27017,r
```

Replace **PASSWORD** with the password created for **jma** when deploying this cluster.
View a full list of your users or reset your password

Figure 6: MongoDB Atlas Connection String

MongoDB Atlas features extensive capabilities to defend, detect, and control access to MongoDB, offering among the most complete security controls of any modern database:

- **User Rights Management:** Control access to sensitive data using industry standard mechanisms for authentication and authorization at the database level
- **Encryption:** Protect data in motion over the network and at rest in persistent storage

To ensure a secure system right out of the box, authentication and IP Address whitelisting are automatically enabled.

IP address whitelisting is a key MongoDB Atlas security feature, adding an extra layer to prevent 3rd parties from accessing your data. Clients are prevented from accessing the database unless their IP address has been added to the IP whitelist for your MongoDB Atlas group.

For AWS clusters, VPC Peering for MongoDB Atlas is available, offering a simple, robust solution. It allows the whitelisting of an entire AWS Security Group within the VPC containing your application servers.

Scalability: You should expect your serverless functions to scale out, thus downstream setups need to be architected to keep up and scale out with your functions. Relational databases tend to break down with this model. MongoDB Atlas is designed with scalability as a core principle. When your cluster hits a certain threshold, MongoDB Atlas will send you an alert and with one click you can provision new servers.

Flexible Schema: Because serverless architectures are event driven, many use cases revolve around around the Internet of Things (IoT) and mobile. MongoDB is ideal for these use cases and more as its flexible document model enables you to store and process data of any type: events, geospatial, time series, text, binary, and anything else. Adding new fields to the document structure is simple, making it easy to handle changing data generated by event driven applications. Developers spend less time modifying schemas and more time innovating.

MongoDB Stitch: Backend as a Service

MongoDB Stitch is a backend as a service (BaaS), giving developers a REST-like API to MongoDB, and composability with other services, backed by a robust system for configuring fine-grained data access controls. Stitch provides native SDKs for JavaScript, iOS, and Android.

Built-in integrations give your application frontend access to your favorite third party services: Twilio, AWS S3 & SES, Github, Facebook, Google, and more. For ultimate flexibility, you can add custom integrations using MongoDB Stitch's HTTP service.

Build complex logic and orchestrate data between clients, services, and MongoDB with server-side JavaScript functions. You can call functions from the Stitch SDKs or through Webhooks.

Unlike other BaaS offerings, MongoDB Stitch works with your existing as well as new MongoDB clusters, giving you access to the full power and scalability of the database. By defining appropriate data access rules, you can selectively expose your existing MongoDB data to other applications through MongoDB Stitch's API.

Take advantage of the free tier to get started; when you need more bandwidth, the usage-based pricing model ensures you only pay for what you consume. Learn more and try it out for yourself.

Important Considerations Before Going Serverless

Though serverless computing offers many advantages, there are several considerations to keep in mind before implementing a serverless project.

Understand Your Break Even Point: It is important to understand the break even point for your application. For an application that only handles tens of requests an hour, a serverless architecture is cost effective. On the other hand, for an application that handles thousands of requests per second it may be cheaper to provision more servers, rather than pay for millions of processes a month. Figure 5 is hypothetical and not based on real numbers, but illustrates that beyond a specific number of requests it may not be economical to use a serverless architecture.

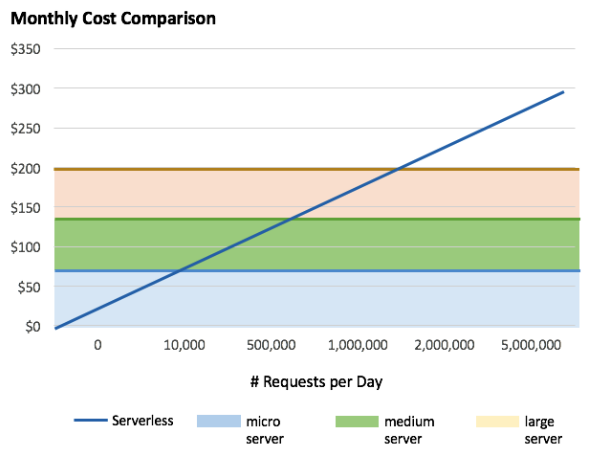


Figure 7: Server vs. Serverless Break Even Point

Cloud Provider Lock-In: Each cloud provider has their own implementation of serverless computing; thus it is likely that features will vary significantly from platform to platform. This makes the risk of cloud provider lock-in high, as porting functions between different providers is difficult and expensive. Not only will you need to update your operational tools (deployment, monitoring), but you will most likely also need to update your code to be compatible with the different interfaces. Users are more susceptible to changes in pricing, performance, and stability as there is less flexibility to shift to another provider.

High Startup Latency: For a function that's just been created, or hasn't been used recently, the startup latency will be high because the environment will have to spin up a new container before the function is executed. This can be problematic for certain applications that see intermittent traffic patterns or require low latencies (e.g. financial trading applications).

Complexity Managing Processes: In a serverless architecture, the time that developers spend on managing the application is not necessarily reduced, but allocated differently. Though developers spend less time managing the infrastructure, they spend more time managing processes. Handling hundreds of processes may be trivial, but handling millions or billions of processes can be overwhelming to inexperienced teams. Before implementing a serverless architecture, it is important to determine if your team has the right capabilities to handle the associated complexities.

Lack of Resources and Skillsets: Serverless computing is a nascent field, which means hiring knowledgeable developers or experts is difficult. Learning serverless computing requires a steep learning curve and resources are not easily accessible. There isn't a wealth of information on Stack Overflow or other forums. Books detailing best practices are also scarce, resulting in trial and error for many developers.

Summary

Serverless architectures are relatively new and build on the work started by microservices. MongoDB Atlas is a database as a service that is well suited for serverless architectures as it provides elastic scalability and all the features of the database without managing operational tasks. Though serverless architectures provide many benefits, there are many considerations to keep in mind when looking to implement a serverless architectures. Thus, diligent planning should be taken before embarking on that path.

We Can Help

We are the MongoDB experts. Over 4,300 organizations rely on our commercial products. 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 Stitch is a backend as a service (BaaS), giving developers full access to MongoDB, declarative read/write controls, and integration with their choice of services.

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)
MongoDB Stitch backend as a service (mongodb.com/cloud/stitch)