Scalability and Flexibility in Document Management Systems

Comparing Server-Based and Serverless Architectures

A FormKiQ White Paper - updated April 3rd, 2024 (v3.0)

FormKiQ

Flexible Document Management

Learn more about our features and pricing at **formkiq.com**

Background

As digitization and process improvement continues, boosted by globalization, remote work, and the reduction of paper-based and manual processes in general, the reliance on document and information management systems increases in step.

With the recent general availability of artificial intelligence for enterprises, this process change has now accelerated, as has the opportunity for efficiency gains.

As cloud-based applications and storage increase, both from general digitization and from the integration of AI, there is an opportunity to leverage cloud-native services to create document management and information management systems that can keep pace with both the growth and process innovation of any organization.

What is scalability and why is it essential?

Scalability is the ability for a software solution to maintain functionality no matter the size of your data or the amount of processing required.

This is not only important for large data sets and complex processes; it's essential that smaller data sets and simpler processes are handled by the solution in the most efficient manner possible.

In other words, a small business and a large business should both be able to make effective use of the solution, with no minimum level of size or complexity required for a scalable solution to be economical.

For document management systems, scalability matters for multiple reasons:

- unlimited capacity is available for document and metadata storage, with high availability and redundancy often included in the services
- system performance, i.e., the organization and retrieval of documents, does not degrade with the growth in the number of documents that is expected over time
- creating a replica of the live application for development or testing can be done at low cost
- the system can not only scale up with growth but can also scale down using the same automated processes if the organization contracts, such as in the case of seasonal variation
- also related to growth and contraction, migration and integration processes often include initial or periodic times of high usage, a temporary scaling up followed by a scaling down to regular usage levels
- scalability in architecture allows for the easy addition of new regions, languages, and compliance with local regulations and data residency requirements without a major overhaul of the existing infrastructure
- scalable systems are often more resilient to failures, as they can redistribute workloads to healthy resources in the event of a component failure; additionally, scalability plays a crucial role in disaster recovery strategies, enabling quick scaling up of resources to replace those affected by catastrophic events

How is scalability achieved?

There are two main methods to achieve scalability:

1. **Increase the capacity of the system**, often through replicating the system, i.e., adding new database instances and/or processing instances

2. **Optimize the performance of the system**, allowing more input, output, and processing within the same system capacity

While optimization is an important part of all software development, the most effective way to achieve scalability consistently is by increasing capacity, and that is often achieved by removing any barriers that prevent or restrict that capacity increase.

This is commonly done through cloud-based software, avoiding two major logistical barriers, the need for employing a hardware infrastructure team and the need to source compute and networking hardware for capacity increases.

In addition, greater efficiency in scaling capacity can be found by utilizing serverless computing in your solution design.

(For more information on scalability itself, see Foundations of Scalable Systems by Ian Gorton at https://www.oreilly.com/library/view/foundations-of-scalable/9781098106058/ch01.html)

What is flexibility and why is it essential?

Flexibility in software is the ability of a system to adapt to changing conditions and requirements without extensive redesign or redevelopment. This includes the capacity to accommodate new features, integrate with other systems, adjust to varying workloads, and allow for user-specific customizations, all while maintaining performance, security, and compliance standards.

Flexibility ensures that the software can evolve over time in response to new business strategies, technological advancements, and user feedback, thereby extending its usefulness and preventing obsolescence.

As with scalability, flexibility is important for any size organization; for instance, a startup may choose to innovate on existing processes, so a reliance on less flexible tools may cause unnecessary overhead that could risk the overall viability of the business model.

For document management systems, flexibility matters for multiple reasons:

- as organizations grow and change over time and their document management needs shift, flexibility allows the system to adapt to these evolving requirements without needing a complete overhaul
- different users and departments within an organization may have unique needs and preferences for how they manage and interact with documents
- processes may need to change depending on document types, document content, or external factors
- integrations with other systems is generally required, with future integration needs not yet known
- security adaptations may be required at any time, often with short timeframes for implementation
- systems that enable the use of only a subset of existing modules can provide substantial cost efficiencies over static architectures
- as with scalability, flexibility in architecture allows for the easy addition of new regions, languages, and compliance with local regulations and data residency requirements without a major overhaul of the existing application
- as with scalability, flexible systems are often more resilient to failures, as they can adjust workloads in the event of a component failure

How is flexibility achieved?

There are two main methods to achieve flexibility:

1. **Utilize configuration settings**, whether through an initial onboarding with configuration files, or with a configuration administration system for real-time changes

2. **Change functionality to meet customer specifications**, providing a bespoke solution when needed, but usually by diverging from the standard codebase and creating an additional and ongoing maintenance burden

While configuration is almost always required for adapting a system to existing process, even if it's just a settings page, most organizations require more robust customization, and variances in the codebase can be found in many enterprise products.

This code-level customization can increase risk on both sides.

For the vendor, it can lead to codebase fragmentation, making future updates and maintenance more complex and error-prone.

For the customer, this bespoke approach may result in compatibility issues, increased dependency on the vendor for support, and challenges in adapting to new features or system upgrades, potentially leading to higher long-term costs and operational disruptions.

How newer software systems approach flexibility

Several innovations can be utilized by newer systems to enable flexibility while reducing the risk of codebase fragmentation and providing alternatives to large configuration files, including:

- modular design, including microservices and containerization as needed, is leveraged to provide more robust customization than configuration alone can achieve, without needing to enable one-off, divergent changes to the codebase
- increasing automation made available for both testing and deployment, often as a foundational component of the modular systems in use; this includes infrastructure-as-code, to ensure that the servers and managed resources are orchestrated in a consistent and verifiable way
- no-code features such as workflow and ruleset designers, and more intuitive configuration administration, combined with robust configuration versioning, enable more granular control over flexibility via configuration settings
- API-first design is utilized to ensure that all functionality is accessible programmatically, enabling not only integrations and migrations, but also allowing flexibility through custom modules or third-party tools to fill functionality gaps
- event-driven architecture for highly decoupled, asynchronous interactions between different parts of the system based on events, allowing for more dynamic and responsive processes and components
- serverless computing abstracts server management and infrastructure concerns, enables automatic scaling, high availability, a pay-for-use billing model, and provides flexibility in managing workload demands without the complexity of traditional infrastructure

Most of these innovations are well-known, but a key one, serverless computing, is still not utilized extensively by most organizations.

What is serverless computing?

Serverless computing is a cloud-native architecture that allows developers to build and run applications without having to manage servers.

The term "serverless computing" can be considered imprecise, as the software does still run on servers; the term relates to the fact that the servers are abstracted from the software, and that any configuration and maintenance, and even the number and types of servers, are managed by the cloud provider to meet real-time demand. This abstraction not only simplifies the infrastructure management required for the functioning of the software, but also provides a real-time cost model for computer processing, where the customer is only charged for the amount of time the code is being executed.

How does serverless compare to server-based?

In a traditional server-based architecture, the software provider configures and maintains a specific number of always-on servers according to predetermined specifications. If demand increases, either more servers are required to be added to the pool of available servers, or the servers themselves need to be upgraded by the addition of more memory, additional CPUs, an increase of storage space, or a combination of all three. If demand wanes, whether through long-term contraction or just due to an overnight, weekend, or holiday period, the servers would need to be reduced in number or specifically downgraded in order to prevent wasting computing power.

While it's possible to use automation to scale servers, and while cloud providers can allow autoscaling by providing additional servers on demand, there are limitations to the minimum and maximum size of servers, and it's ultimately the customer's responsibility to ensure that they have mechanisms in place to scale up and down as needed.

In addition, traditional server-based architecture often requires routine tasks, such as managing the operating system and file system, keeping up with security patches, and setting up and maintaining logging and monitoring. Cloud providers can abstract some of these tasks, such as Amazon EC2 providing logging through CloudWatch or file handling using Elastic File Storage, but some tasks will always be the responsibility of the customer.

How serverless is being used

Many software solutions combine server-based and serverless components; AWS customers often make use of S3 to store files, AWS Lambda functions to run tasks, DynamoDB for a NoSQL database, and CloudFront as a content delivery network. All of these components are part of AWS' serverless offerings, where they fulfill workflows without any requirement to maintain and configure servers or to set up scaling automation, i.e., autoscaling.

It's common to see static websites or JavaScript-based client applications that are stored in S3 and served by CloudFront; in many cases, these sites or applications may have some limited back-end functionality handled by AWS Lambda and API Gateway, for instance when handling a contact form.

For document management systems, expanding the scope and responsibilities of serverless components allows for better scalability, by reducing the components that require server configuration and scaling mechanisms. While some functionality may not be possible with serverless components, such as providing full text search using Amazon OpenSearch, using serverless whenever possible allows the restrictions on scaling to be reduced significantly.

How does serverless affect cost?

Serverless is not free, though in the case of smaller workloads, serverless can often be run almost entirely within the free tier provided by the major cloud providers. For some workflows, on-demand serverless can be more expensive than a stable server-based workflow, particularly when no cost optimizations have been performed. In the case of AWS, there are upfront cost commitments that can be made for processing workflows, such as DynamoDB Provisioned Capacity and Compute Savings Plans. There are also optimizations available for storage, specifically storage tiers (and intelligent tiering, when available) for products such as S3 and DynamoDB.

Where serverless excels is in the dynamic scaling, which is exactly how it achieves its scalability. In the case of storage, this scalability means that your S3 or DynamoDB storage will never run out of space, but at the cost of your cost growing as your storage increases. In the case of compute workflows such as AWS Fargate or AWS Lambda, it's possible to run hundreds or even thousands of tasks concurrently. As Fargate tasks can use 4 virtual CPUs and up to 30GB of memory each, this level of compute concurrency should be able to meet upwards of 99.999% of workloads.

Where server-based excels is when your workloads use the majority of your server-based capacity. As this theoretical study from AWS indicates when comparing EC2 to Fargate for AWS ECS — **https://aws.amazon.com/blogs/containers/theoretical-cost-optimization-by-amazon-ecs-launch-type-fargate-vs-ec2/** — if your workload is able to remain at near-full utilization of your provisioned EC2 servers, you will see some savings over serverless Fargate; in the case of this paper, they estimated the savings on an ECS cluster of fully-utilized EC2s as 20% over the same cluster using AWS Fargate. But in cases where the ECS cluster has little to no utilization, Fargate can be up to 87% cheaper than EC2.

Flexibility: considering serverless despite a higher usage cost

The key component of systems that would benefit from a serverless model is a requirement for flexibility in usage. Some workloads are consistent enough that it can be more cost-effective to provision server-based infrastructure; for example, if a web application has a consistent level of traffic, with little to no variance, the per-request cost of a server-based architecture will likely be lower than a serverless architecture.

However, even in cases like this, there are some advantages to serverless that may outweigh the moderate increase in cost:

- even with relative consistency in workload for a server-based system, any large spike in capacity needs has a higher risk of failure
- some server-based components still grow in size despite a consistent level of usage, such as log files, databases, or cache stores, and when hard limits are reached, the risk of failure increases
- it can be cost-prohibitive to replicate server-based systems for non-prod environments, depending on the components required
- each server-based component requires networking and security configurations that are generally more complex than connecting together various serverless components and managed services

Looking at serverless for specific document management system tasks

While there are benefits and drawbacks to serverless depending on the system being designed, there are specific tasks within the document management system workflows that can leverage serverless components or managed services for a lower total cost of ownership over a server-based architecture.

Authentication and authorization

As document management systems often include documents of varying confidentiality and differing ownership, it's essential for a document management system to include authentication and authorization functionality.

In a server-based architecture, authentication often utilizes a database instance as well as the compute instance; in cases of federated logins, the database instance can be replaced by a reliance on an existing identity provider such as Microsoft Active Directory or Google Workspace.

By leveraging a managed authentication service like Amazon Cognito, a document management system can use the built-in authentication or a federated authentication, with no requirement to store user information within a specific database instance.

For authorization, a server-based architecture often implements a module within the application code. This adds some processing overhead to the application and the server instances that host it. A serverless API management service, like Amazon API Gateway, can be used to offload most of the authorization processing, whether through combining that service with a managed authentication service like Cognito, or by relying on internal cloud-based identity and access management such as AWS IAM.

Document storage

In a server-based architecture, the storage of documents can be handled through the use of a file server, or even by storing on a single application server, but scaling in either of these models can be challenging.

Using a managed object storage service like Amazon S3 removes any scaling challenges, and most include storage tiers for better cost efficiency for long-term infrequent-access storage.

Amazon S3 is known to be more reliable than a local file server, due to its high fault tolerance, reliability, and availability. Because S3 has no minimum storage requirements and offers intelligent tiering, archives, and lifecycle policies to remove expired content, the cost will generally be lower than the overhead of a local file server.

Document import

A server-based architecture may include a mail server for receiving documents via email, and may also include an application module to receive documents via an API. Scaling can be an important consideration for both of these methods for importing.

It's possible to mitigate those scaling concerns through the use of a managed email service like Amazon SES and an API management service like Amazon API Gateway. In addition, the object storage service (e.g., Amazon S3) can also provide functionality to assist in importing objects, such as signed URLs for secure uploads, and a command-line interface (CLI) for uploading objects directly from a workstation or file server.

As the workloads for both a mail server and an API will be variable, the total cost will likely be lower for managed services vs. configuring and running an email and application server for import tasks.

Optical character recognition (OCR)

A server-based architecture can include an OCR module, which would need to be designed to work with inconsistent workloads possibly with queuing functionality.

A managed service like Amazon Textract allows for offloading of OCR processing, with other managed services helping for the queueing and orchestration, such as Amazon SQS. It's also possible to use a well-configured OCR module that runs using serverless compute with an open source OCR library such as Tesseract, which may be possible at a

lower cost than by using Amazon Textract, though results and functionality will vary between the two OCR engines. While Tesseract provides a lower cost in many cases, Textract may be a better choice when table or form data needs to be extracted with its structure maintained.

Artificial Intelligence (AI), both for intelligent document classification (IDP) and content generation (GenAI)

With Amazon Bedrock, it's possible to choose from several large language models (LLMs) in order to provide the analysis and generative functionality required for information management workflows.

The benefit to using Bedrock is that since the models are made available within the AWS account that also stores the documents and metadata, no personal or proprietary information needs to be transmitted externally.

But as with OCR, it's possible to use an open source solution for many AI tasks. For instance, intelligent document processing may be achievable with comparable results using an open source model such as those available from Mistral, running on a GPU-based ECS Fargate container, assuming it includes a mechanism for only running the container when needed.

Document search

Document management systems require the storage of metadata for each document to assist in classification and search. For server-based systems, this would usually include a database instance.

A serverless architecture could involve a managed NoSQL database service like Amazon DynamoDB, which can store metadata in a flexible key-value model, which not only enables easier scaling than a server-based database cluster, but can remove the need for data migration on system updates.

For more robust search, such as Fulltext Search, it's possible to experiment with a managed serverless relational database, like Amazon Aurora Serverless, or to interact with a server-based Fulltext Search system like Elasticsearch or Amazon OpenSearch. Serverless architecture can easily leverage these server-based features as required.

Client Interface

A server-based system may include a full-stack monolithic application that includes the application controller layer and the presentation of visual information, or it may split these responsibilities between an API/middleware and one or more front-end clients. This could involve one or more application server clusters, and the use of auto-scaling configurations could prevent most cases of failure due to an overload of requests.

A serverless system would likely include separation between the API and the client, though that is not guaranteed, and could make use of a managed API service like Amazon API Gateway for the API, while using a managed object storage like Amazon S3 and a CDN like Amazon CloudFront to service a static front-end client. This client could use a JavaScript client framework like React or Angular to interact with the API, without requiring an application server instance.

Conclusion

While there is no reason why a server-based architecture cannot be used for a document management system, the importance of scalability for a DMS, as well as the specific use-cases of a DMS that are well-suited to managed services and serverless components, makes a server-less architecture a clear competitor, and in the case of a document management system hosted in a cloud provider like AWS, a low-risk and high-value choice.

Learn more about FormKiQ's Serverless and API-First Document Management options at https://formkiq.com