momento

A guide to unlocking serverless at enterprise scale

Ship faster, run leaner, and reduce total cost of ownership

Benjamen Pyle

Contents

Enterprises are always looking for ways to ship software with greater speed, higher quality, and fewer resources—while also maintaining feature agility.

While these concerns and obstacles aren't new to veterans in the technology sector, the modern cloud approach to software delivery—including serverless technologies—gives leaders and builders new ways to combat these competing forces. But what exactly is serverless? How can it help solve modern problems? And finally, is my enterprise ready and able to harness its power?

1. What is serverless?

So just what is serverless and what are the benefits that it can provide an enterprise organization? At its simplest, serverless is a categorization describing a software component. These components provide the developer with a contract around price, consumption, scale, and reliability. Something is said to be serverless if it displays these attributes:

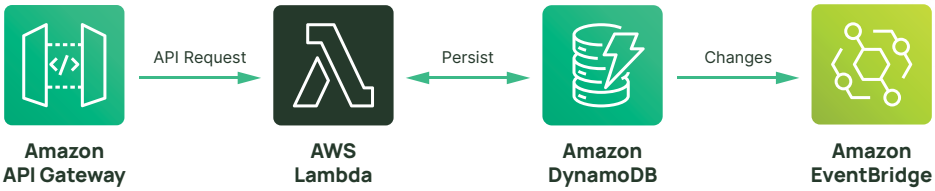
- **Nothing to provision, nothing to manage.** Serverless doesn't mean the lack of infrastructure, just that the provisioning and maintenance of that infrastructure are abstracted away from the builder.
- **Cost scales predictably with usage.** This is the scale-to-zero principle. Serverless means a usage-based pricing model with no minimums.
- **No planned downtime.** With no infrastructure to maintain, serverless means no maintenance windows, which means higher availability.
- **Ready with a single API call.** There can be some variation here from putting a file in S3 to launching a container in AWS Fargate, but serverless services are ready to be consumed on a moment's notice.

1.1 Serverless is more than compute

If so far this sounds strangely like AWS Lambda or Microsoft Azure Functions, that's because those two services are examples of serverless compute. But when leveraging serverless technologies there is a great deal more at an engineer's disposal than just compute.

Serverless components exist at all levels of an application's architecture. There are services for data, event management, workflow coordination, object storage, and API gateways just to list a few.

To further illustrate this point, the below diagram represents a web-based request that is designed to handle input, persist the output, and then propagate that change via change data capture. Every component here is serverless.



Serverless has been at the heart of cloud systems for over 15 years. In fact, Amazon SQS and S3 first shipped in 2006—and they're serverless. AWS has continued to build upon that success with Amazon SNS, Amazon DynamoDB, and Lambda. These single-purpose serverless components embody all the attributes listed above. They are assembled like building blocks to deliver robust architectures that perform from ultralight to massive scale.

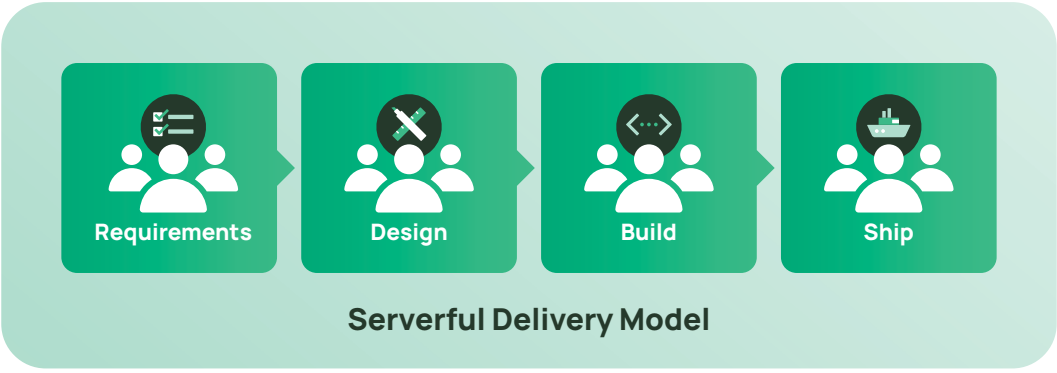
But if components are single-purposed, how does a developer or architect design around these capabilities? This is where the enterprise needs to be prepared to invest in its people.

1.2 Serverless is more than technology

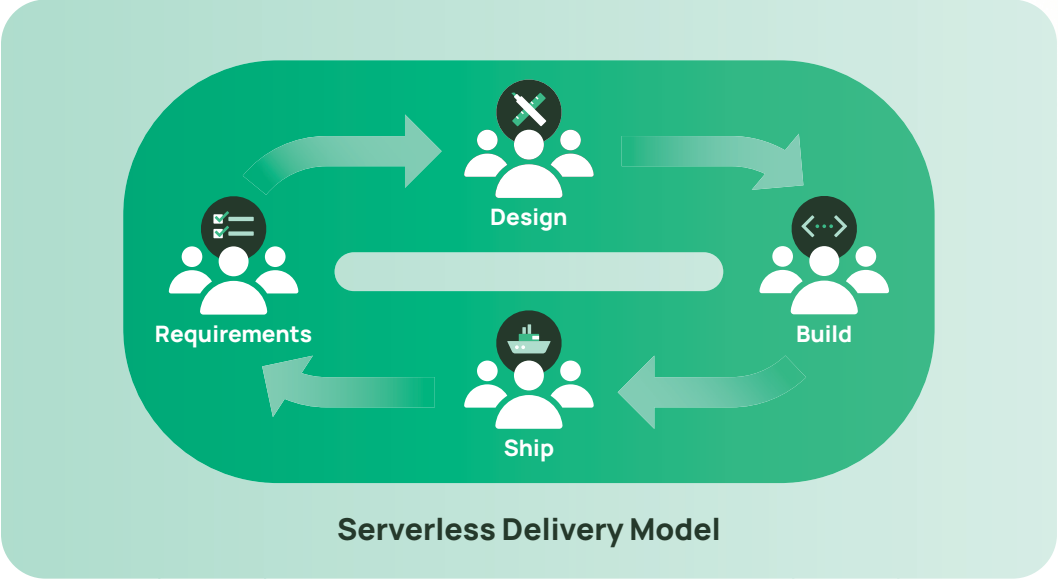
If serverless describes technology capabilities, then why does this involve people?

Because at the heart of every technology solution lies the people who design, build, and operate it. In a traditional "serverful" world, tasks to bring a software solution to market are often siloed and executed in a sequential or waterfall fashion. Enterprises often have substantial investments in their processes and people. Requirements are gathered, solutions are designed, and infrastructure is provisioned and ultimately deployed with the software solution installed.

In this scenario, people fill very specific roles in service of delivery. Engineers and operations teams work together but not always in harmony. In the case that a design doesn't match the needs of the user workflows, teams must work back through the process to remedy the issue.



Contrast that with a design that has invested in serverless, and a couple of things will begin to happen. The first outcome is that the job roles and functions needed to ship a product will start to compress. Product people will start to understand cost and scale at a much more intimate level. Operations engineers who in serverful delivery are focused on strictly running infrastructure will now be assisting in the selection and configuration of serverless components.



Serverless Delivery Model

The second outcome is that delivery teams who are working with autonomy to ship value to their customers will be able to experiment and pivot much faster. These teams won't be heavily invested in infrastructure or sunk costs, nor will they be constrained by a design that was built with a stream when all it needed was a queue. Serverless builds are adaptable, configurable, and modular, giving teams an advantage over competition working strictly with custom-built infrastructures.

When thinking about adopting or evaluating serverless in an organization, the impact on existing talent is great. People want their work to matter and make a difference. The greatest compliment a builder can receive is that someone using their software appreciates it and finds it useful. Serverless enables faster feedback loops. The design, development, release, and adaptation phases of software development are more compacted. These tighter loops allow the builder to more easily bind their purpose to the enterprise's purpose. Alignment equals engagement. And in the modern workforce, employee engagement matters.

2. Applying serverless

There is no one-size-fits-all approach to adopting serverless in an enterprise. However, there are several consistent considerations regardless of how much or how little of the serverless apple one takes a bite out of. The following are things an enterprise needs to decide as it moves forward.

2.1 Where to Begin

The age-old "where to begin" question strikes people first. The start sometimes paralyzes even the most experienced enterprises but that shouldn't be the case with serverless. Two natural starting points hold whether the enterprise is shipping consumer software, business-to-business software, or providing software as internal tools to support its operations.

2.1.1 New Features

Building new software is the perfect place to begin an enterprise's serverless journey. This endeavor doesn't have to be large either. When trying out a new technology, starting at either end of the difficulty spectrum can be a good choice. If the build is easy, an enterprise can see a quick win and gain confidence. If the build is difficult and highly demanding, the enterprise tackles something challenging, gains confidence, and proves or disproves critical assumptions about the technology. However, if the enterprise wants to start with something difficult, it might be useful to bootstrap the organization's capabilities by bringing in some seasoned experts. Serverless knowledge will spread like wildfire, so be prepared—in a positive way—for this engagement.

When introducing serverless on new builds, the following areas are generally good starting points:

- **API Development.** This will combine components like Amazon API Gateway, AWS Step Functions, Lambda, and DynamoDB.
- **Data Pipelines.** Data movement is a common challenge in any application. Serverless components can turn traditional batch-based operations into nimble real-time data mover powerhouses. Tools like Step Functions, DynamoDB, Amazon Kinesis, and Amazon SQS are all helpful in these workloads.

2.1.2 Existing Applications

Many enterprises don't always have new features rolling out where they can try brand new patterns. This is why starting small with an existing application can be extremely valuable.

A common scenario ripe for a serverless build is the extension of an existing application. One such use case is adding functionality to an API. By leveraging Lambda, API Gateway, or even an existing load balancer, new functionality can be added without disturbing a more monolithic component. Remember: Lambda and serverless are not synonymous, but using Lambda as an event-driven compute engine—as it was intended—is a very natural starting point.

The beauty of extending an API with Lambda is that an enterprise can bring its existing toolchain. Heavy on .NET Core? No problem, Lambda can do that. Does the enterprise lean into Java? The same as .NET, Lambda can do that. Lambda also comes prepared with Linux-based runtimes to launch Rust and Go code. Or perhaps there's already a heavy investment in containers? Lambda will run those too.

Once that first Lambda is in place, the enterprise will have developed the skills and confidence to explore and expand the solution to other serverless components.

The key is to start small, learn, adapt, and integrate.

3. Is your enterprise ready for serverless?

That's a broad question with a great deal of nuance. Only the enterprise can truly make that determination. However, how would it do that?

3.1 Does the culture embrace agility?

Almost every enterprise will answer yes to this. There aren't many organizations out there that will admit to not being agile. However, serverless allows an enterprise to take agility to a higher level. Some of the key tenets of agile delivery are working software, allowing for change versus following a plan, and empowering teams to act autonomously and own their outcomes. Many enterprises struggle to realize these attributes due to the rigidity of their traditional infrastructure. Serverless architectures remove these legacy barriers and allow teams to get on with the business of shipping value and delighting its customers.

Is the organization ready to ship value quicker, fail faster, and adapt to key findings? It can be tough to allow a team to fail in front of a customer with a new feature. But with serverless, those failures can turn into successes as the team learns what directions they *shouldn't* take and can quickly pivot to what directions they *should* take. That's agility.

3.2 Is there a passion for observability?

Serverless systems often have characteristics that without observability are difficult to manage: they are sometimes event-driven, infrastructure is almost always ephemeral, and managed components generate their instrumentation.

Without a passion for observability, user requests and system events will fall into a giant abyss containing all these lost signals. When designing around observability, the three pillars of logs, traces, and metrics are where the journey begins.

Of course an enterprise doesn't need to have all of the instrumentation or plans for instrumentation in place before trying serverless. However, before any serious serverless feature is to be shipped to production, the practice of observability needs to be addressed, which includes tooling and instrumentation.

So is the enterprise comfortable learning and applying the techniques it requires to monitor serverless applications? And a follow-up question: are the teams that currently run operations and engineering capable of leaning in closer together to solve new problems and challenges?

3.3 What skills are in house currently?

Touching again on an earlier point, people matter when an enterprise decides to embrace serverless. Whether that enterprise is ready depends largely on the people and team that are blazing the path.

This isn't to say that the enterprise requires at least three serverless developers who are fluent in infrastructure as code and have shipped to production before. But it does require a general spark of curiosity and a willingness to lean into new techniques and paradigms.

If the enterprise is serious about empowering teams to ship early, adjust often, and embrace change in both requirements and scale, then its people must be willing to understand how. As mentioned, serverless will compress job responsibilities and create tighter teams. These people units will learn and grow faster together when corporate direction lines up with technical curiosity.

4. Tips from advanced practitioners

No one has all the knowledge required to embark on a journey like this from day one. However, being as prepared as possible reduces the chance of encountering a wide array of outcomes and scenarios. This section includes insights from enterprise serverless veterans about what they wished they knew when they started.

4.1 Establish solid architectural standards early

Serverless comes with new tools and new paradigms for shipping value to customers. It creates higher-level abstractions for developers to build upon. Learning and understanding these are required to take advantage of them at runtime.

Important standards to nail down early in adoption include which components the enterprise will allow and which problems they will solve. For example, event processing is usually a key piece of serverless architecture. In AWS, there are several components for handling events. SQS, EventBridge, and Kinesis are three that spring to mind. If the system needs to handle data, DynamoDB, S3, and other datastores can be leveraged.

A well-intentioned enterprise can quickly find itself with multiple ways to handle any number of scenarios, which creates complexity in design and management—even at low scale. This is because the agility discussed earlier can lead to confusion.

In addition to defining standards around which components, it's important to set good defaults for chosen services. In the SQS example, understanding visibility timeouts, message delays, and dead letter queues will set an enterprise up for success as serverless spreads throughout the organization.

Adopting serverless will come with a host of small examples like the above. Advanced practitioners know that understanding the non-functional requirements in a project will drive the answers to so many of these questions. Remember that serverless components are usually assembled like pieces in a puzzle, and there are more pieces than in a traditional monolithic build. Setting solid standards around choice and usage will enable the enterprise to understand runtime profiles and thus pivot when new outcomes are observed.

4.2 Cost is measured against usage

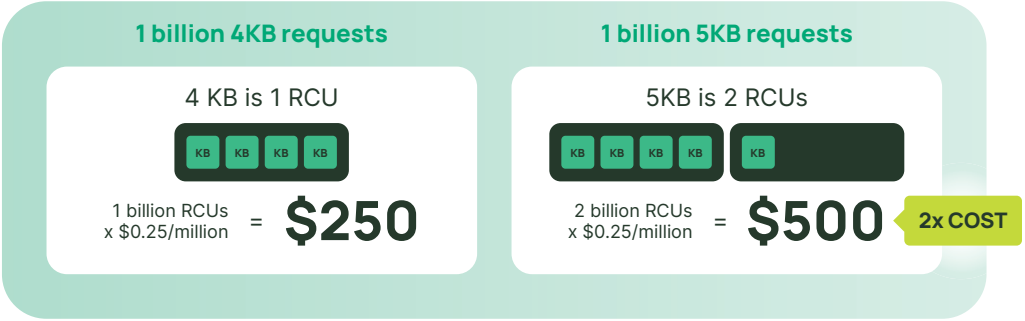
A serverless component's cost is measured in usage. That usage could be the number of events handled, duration of execution paired with memory allocated, or the amount of data saved or retrieved. The internet is littered with arguments that serverless is expensive. But to take an opposing and more encompassing view of cost, an enterprise needs to look at the design and the human cost to maintain—the total cost of ownership (TCO).

Let's look at an example with DynamoDB, which often gets labeled as "expensive". DynamoDB is a highly available key-value store database that when modeled correctly will perform consistently with single-digit millisecond response times.

DynamoDB charges a customer at the simplest level based on how many read and write units the customer requires. Read Capacity Units (RCUs) are clearly defined around 4KB boundaries. RCUs are then charged at $0.25 per 1 million (as of this writing).

At low volume, these types of metrics don't mean a great deal. So what if an item size is 5KB instead of 4KB and the application uses consistent reads everywhere, incurring 2 RCUs per request? They only cost $0.25 per million. No big deal, right? Wrong—in reality, the feature is eating through the 1 million requests twice as fast.

System Design Impact on DynamoDB Costs



Experts know that cost matters and that it follows usage. Modeling the impact of a system's design early in the process can yield huge savings down the line.

Additionally, savvy executives know that the TCO must include the software purchased and the people required to manage that software. With serverless, an enterprise takes advantage of all the right optimizations baked into the price. It also avoids licensing fees to a software vendor. And lastly, it doesn't have to patch or maintain any of this infrastructure.

Understanding Total Cost of Ownership

After considering all the benefits of upfront design planning, cost following usage, and the lack of infrastructure management, it's apparent the argument that serverless is expensive usually stems from just one angle: the cost for usage. An enterprise must look deeper at the TCO—including the cost of people, infrastructure, maintenance, and licensing.

4.3 Plan to observe from day one

As mentioned earlier, observability needs to be a passion in enterprises adopting serverless. Advanced practitioners know just how important this topic is when deploying serverless builds into production.

By planning for the three pillars of observability (logs, traces, and metrics, as a refresher), an enterprise can set itself up for success when rolling out its first serverless feature—and provide comfort when remediating issues in production. Take the time to decide on the following:

1. What level of dimensionality needs to exist?
2. What tooling is needed to support organizational visibility?
3. Plan for the cost of tooling, as observability costs can grow quickly with volume.
4. What techniques or skills are needed to make observability a key part of the architecture?

4.4 Embrace infrastructure as code (IaC)

This is one of those insider tips that has multiple benefits, but let's focus on two:

1. IaC allows for parity between environments thus reducing risk in infrastructure drift.
2. When infrastructure is coded as part of the serverless feature, there are fewer dependencies.

With the abstraction of physical server infrastructure, serverless components will be powered by configuring and using APIs to engage with resources. When a build is just in a sandbox, using a GUI tool can be just fine to get things going. However, as soon as that sandbox needs to migrate to a proper dev, test, and production setup, getting those configurations right can be a headache ripe for error. This is often known as "Click-Ops".

IaC is source code just like application code that sits right next to the actual business value. The benefit of IaC as it relates to consistency across environments is also in the fact that many of the tools come with best practices and defaults out of the box. Every enterprise will need to evaluate that they are correct, but they provide great starting points. There are many flavors of IaC tooling ranging from third-party solutions to native cloud vendor-supplied options that an organization will need to evaluate. The important thing is to choose the one that fits the skills of where the team is today and where they want to be in the future. Once the selection is made, then standardize around it to gain efficiencies as the serverless journey scales in the enterprise.

Deployment repeatability is a key advantage to serverless and IaC. As an enterprise, don't miss out by not using it. Many enterprises believe that they don't have enough environments or feature scale to warrant the investment in IaC. The truth is, every organization—regardless of scale and size—benefits from the consistency that comes from deploying infrastructure as code.

In addition to repeatability, IaC increases speed and agility. By pairing it with serverless designs, which also enable agility, that benefit is doubled. It also has the nice side effect of reducing dependencies between teams. If the engineering team can build and design its infrastructure with proper architecture, then the burden on the DevOps team is reduced. This can have a tremendous impact on staffing and morale.

5. Conclusion

Serverless isn't new. Going back to 2006 when AWS released S3 and SQS, serverless could be said to be almost 20 years old. It boils down to some distinct advantages for enterprises:

- Ship earlier, faster, and more often.
- Enable teams to take ownership of their delivery pipeline through IaC, rapid delivery, and compression of roles.
- Manage costs at the usage level while reducing the total cost of ownership with no infrastructure to maintain, patch, or provision.

As an executive or a practitioner, it might be difficult to see a path forward to achieve some of these benefits. Perhaps there's legacy code or a skeptical production organization that might be unwilling to lean into something new.

Remember, start small. Monoliths might exist for a reason. Everything doesn't have to be serverless. Pick a small feature, rewrite an existing one, or extend something that already exists. Conditions don't have to be perfect to get started. Just start.

The biggest point is to remember that nothing is ever absolute. At the end of the day, only the enterprise can decide how it wants to ship and manage value. Serverless can be *the* strategy, or it can be a part of the strategy. Many enterprises adopt what is called a serverless-first approach. They acknowledge that not all problems fit the serverless pattern, but they start there and then pivot when needed. Their executives and builders come together, identify places that serverless could help their organization, and start realizing the benefits that serverless can bring.

For example, fintech giant CapitalOne deals with strict industry regulations and has been operating at internet scale for years with Lambda, Fargate, and many other serverless components. Taco Bell, a popular American quick service food chain, leverages serverless to power high-volume and spiky workloads in their mobile app. And Wyze, one of the leading smart home device providers, uses Momento Cache to manage billions of thumbnail images across millions of IoT cameras every single day.

They're experiencing the power of building with serverless.

Is your enterprise ready for it?

Learn more about Wyze's success with Momento

Read Case Study

Momento is taking serverless to the next level—making infrastructure easy at enterprise scale. With automatic resource management in Cache, Topics, and Storage, your developers can build what's next with simpler architecture and no distractions.

Cache

Quickly improve your performance, reduce costs, and handle load at any scale.

Topics

A low-latency pub-sub service built for massive scale with unbeatable connectivity.

Storage

An intelligent datastore for high-volume workloads that automatically optimizes costs according to access patterns.

We'd love to talk about where serverless can start transforming your business. Contact us today.

About the Author

Benjamen Pyle is a technology executive and software developer with over 20 years of experience across startups and large enterprises. He is Co-Founder and CEO of Pyle Cloud Technologies, an AWS-focused cloud consultancy specializing in cloud strategy, architecture, training, and cost optimization.

He's also an AWS Community Builder recognized for his deep expertise in serverless computing, event-driven architecture, and cloud-native and containerized solutions.

When Benjamen doesn't have his head in the clouds, he's either playing golf with his wife and two boys or outside with their 12 paws.

Sources

- Momento's Serverless Litmus Test
- Amazon DynamoDB Pricing
- Blogs from Benjamen Pyle
  - Intersection of Technology and People
  - Serverless and Agile
  - Serverless: A CTO's Perspective

More from Momento

- Discover Momento case studies
- Read the Momento blog
- Explore Momento use cases
- Learn more about Momento Cache
- Learn more about Momento Topics