APPENDIX A

PostNL's Serverless Journey

With contributions from Luc van Donkersgoed, AWS Serverless Hero and Principal Engineer at PostNL

PostNL is the largest logistics supplier in the Benelux. In 2022 we delivered an average of 7.4 million letters and 1.1 million parcels per day. To achieve this logistical feat, PostNL employs more than 35,000 people, making it one of the largest employers in the Netherlands.

PostNL is also one of the oldest companies in the Netherlands. Our roots trace back to 1799, giving us well over 220 years of experience in logistics. Throughout its rich history, PostNL has been very comfortable in adapting to change. We've led transitions from horses to cars, from physical post offices to the internet, from letters to parcels, and from state-owned to a publicly listed enterprise.

In 2012, PostNL was one of the first major European enterprises to commit to a 100% cloud strategy, knowing that managing physical servers was not our core competence and should not be our focus. In 2017 we closed our last data center and began running all our solutions in the public cloud.

In 2019, PostNL took this evolution to its next logical step: we decided that to stay competitive, we had to design and build our own logistical software. We researched which technologies best fit a modern logistics technology company, and our strong partnership and history with AWS naturally led us to AWS cloud-native services—with a pronounced emphasis on serverless.

PostNL belongs to a very select group of large enterprises with a significant serverless footprint. The case study presented here explores why we chose serverless and how we effected the transition. As you will see, our serverless experience has been mostly positive and has allowed us to realize the benefits we aimed for. It has also introduced its own set of challenges, described toward the end of this case study.

Before Serverless

Before 2019, PostNL did not develop a significant amount of software. Instead, we relied on commercial off-the-shelf products and contracted partners and vendors to build custom solutions where necessary.

In fact, the entire organization was structured around the coordination of external parties. In-house business and market analysts designed products to improve our competitive position, architects fit these plans into the enterprise architecture, and solution consultants translated their needs into requirements. These requirements were communicated to a supplier, who built the solution. Staffing, delivery, operations, maintenance, and many other responsibilities were delegated to the supplier as well.

The Reasons for the Change

Inherent to industry-standard off-the-shelf products is that everybody uses them. This is fine for generic business needs like email, spreadsheets, or calendars—they are non-differentiating technologies. However, if multiple companies in the same market segment use commercial off-the-shelf products for their core businesses, they will end up with non-differentiating solutions. In other words, using commercial off-the-shelf products forces your company to follow the market.

Many great books have been written on enterprise software strategy. The current consensus is that to gain a competitive edge, you must build the software that is differentiating for your products and services, and buy only the software that isn't. The engineering strategy PostNL set out in 2019 was centered around this core concept. Here's an excerpt from our strategy deck:

> As we move to a future of intelligent logistics driven by AI, IoT, and Data, we will need to develop differentiating capabilities faster, which we should understand ourselves.
>
> Therefore, we propose a viable capability to design, build, and operate digital platforms and systems in integrated teams for PostNL.

The strategy is clear: PostNL wants to lead—even define—the market, and the only way to do that is by building our own software. By specifying that "we should understand ourselves" our differentiating solutions, it clearly departs from the vendor/supplier-based approach.

The Serverless Journey

With the direction and goals clearly defined, PostNL needed to decide how to achieve them. Designing an enterprise software engineering approach from scratch is a complex problem. On the one hand, we wanted to set clear guidelines and guardrails

to standardize a way of work and enforce best practices. On the other hand, we wanted to allow teams to design and implement applications the way they saw fit. Additionally, we wanted to give engineers the opportunity to influence standards and guidelines. After all, engineers know what works and what they need—a top-down approach is likely to encounter resistance and will lead to slower adoption.

As is often the case, the solution lies somewhere in the middle. In PostNL, technologies, products, and services are categorized according to a "fixed, flexible, free" model. In this model, the "fixed" category contains topics that are standardized across the entire organization. The "flexible" category contains ranges of products, services, and standards to choose from. Teams have the liberty to adopt any solution within these given ranges. The "free" category contains all other topics. Within this category, teams are free to decide what to use within the boundaries of their own budget, architecture, and experience.

Strategic decisions are always categorized as "fixed." For example, choosing AWS as our public cloud provider, including the standardization of cost controls, networking, and security guardrails, is a highly impactful decision. Teams do not have the liberty to choose another cloud provider for their applications. Programming languages fall under the "flexible" category: teams are free to choose the language that best fits their team and application, but it must be selected from a predefined set: Java, TypeScript, Python, C#, or Rust. Choosing which IDE or remote development environment to use is an example of a "free" option. We trust the teams to choose the best fit for their context.

Another strategic, and thus "fixed," decision is to build applications with AWS cloud native technology. In other words, using cloud native services is mandatory at PostNL. In fact, our AWS platform team (named the Cloud Center of Excellence, or CCoE) has service control policies in place that make it impossible to spin up EC2 instances in our AWS accounts. As far as we know, this is rather unique among enterprises of our scale. The serverless-only approach has yielded many benefits in operational cost, attracting talent, and dealing with scaling and seasonality. However, the strategy is not without its challenges, as you will see later.

The Serverless Fit for Logistic Businesses

PostNL considers the use of serverless a strategic, and therefore mandatory, component of building in-house software. That implies we have a strong opinion of what serverless has to offer, as well as what it protects us against. To understand our motivations for choosing serverless, some additional context about PostNL is needed.

PostNL is a logistics company. Our core business is to transport parcels and mail from their sources to their destinations, at very high volume, preferably within 24 hours. To achieve this, we maintain significant physical infrastructure, including 34 sorting centers, over 5,000 retail locations, over 500 automated parcel lockers, and

hundreds of vans and trucks. We are the largest logistics supplier in the Benelux, but we're competing against global players. As a result, we're operating in a low-margin, high-stakes competitive environment. The business with the best performance, lowest costs, and highest customer satisfaction wins.

Logistics is also a very seasonal business. The holiday season, which starts at Black Friday in November and ends around Valentine's Day in February, is our busiest and most important period of the year. In these months, our infrastructure and applications are processing significantly more volume than in the summer months. Aside from the yearly seasonality, we also have weekly and daily patterns: on Sundays and at night, our systems are all but inactive, while we see peak loads on Tuesdays, specifically around 2 p.m. (Figure A-1).

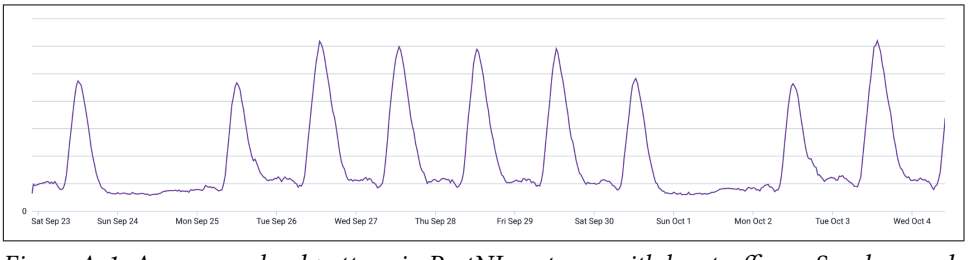


Figure A-1. A common load pattern in PostNL systems, with low traffic on Sundays and peaks on Tuesdays

This operational context forms the foundation of PostNL's decision to go all-in on AWS serverless. There are three core properties of serverless that make it a perfect fit for our business:

1. Elastic pricing
2. Effortless scaling
3. Delegating the hardest problems to AWS

Elastic pricing

Considering the daily, weekly, and yearly seasonality of the logistics business, PostNL needs its systems to scale elastically with demand. At a bare minimum, we required DynamoDB's autoscaling capacity, which increases the provisioned capacity units when predefined thresholds are crossed. Figure A-2 shows this scaling mechanism in practice.

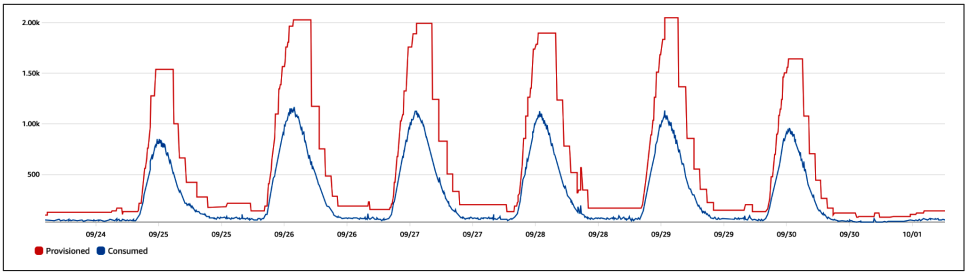


Figure A-2. An example of DynamoDB provisioned scaling, where the provisioned capacity autoscales in response to the current load

With our usage patterns, autoscaling solutions like these are significantly more cost-effective than statically provisioned capacity. Additionally, this feature allows the system to respond to unexpected behavior without human intervention. But there is also an obvious inefficiency in this graph: at all times, the provisioned capacity is configured at a much higher value than actual usage. This margin allows the system to respond to unexpected surges, but most of the time the extra capacity simply means wasted resources.

Following this line of thinking, the ideal pricing model for the services we use is "pay per use." Common examples of AWS services applying this model are Lambda (cost per GB-sec spent on invocations), EventBridge (cost per 64 KB of custom events published), SQS (cost per API call), and Step Functions (cost per state transition). All of these services are billed based on exact usage, which means our systems hardly incur any cost during off-peak hours.

Effortless scaling

Most seasonality is predictable. For example, we know when Black Friday will take place. Other events are harder to predict, such as the COVID-19 pandemic. Thanks to our serverless strategy, we do not need to worry about any of these kinds of shifts at all. Because PostNL uses serverless services across a large part of the application landscape, there are hardly any mission-critical systems that do not automatically scale. This means our development teams do not need to think about capacity planning: their applications automatically scale with demand, whatever the cause of that demand might be. Figure A-3 shows the Lambda invocations for one application over a six-month period. Usage steadily grew, then peaked, then gradually decreased again—and nobody even noticed. During this time, no engineer had to spend time managing, monitoring, or massaging the application under increased load.

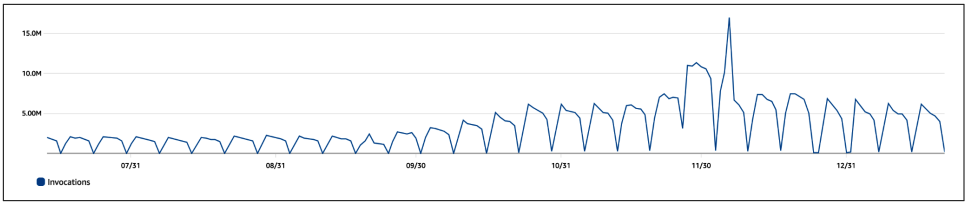


Figure A-3. Number of invocations of a Lambda function over a period of six months

Delegating the hardest problems to AWS

PostNL runs large-scale systems; we generally think of volumes and operations on the scale of millions and billions. For example, our delivery drivers generate over three million interaction events on any given weekday. On the same day, our central event broker validates and routes well over six million events, and our IoT systems process almost two billion raw device signals. Designing and building mission-critical systems that operate at these volumes is not easy. Common problems involve capacity management, redundancy, networking, elastic scaling, throttling, and software maintenance. These challenges are real, and they need to be handled. Embracing serverless allows PostNL to delegate the most difficult problems to AWS. Here are some examples:

- Our systems need physical CPUs and memory to run on. But we don't think about this. We use Lambda and Step Functions, and the capacity is managed invisibly.
- Our systems communicate over a secure and scalable network. But we don't need to know what that network looks like or how it scales. We call AWS APIs, and it works.
- We need to be able to handle dynamic workloads. But we (almost) don't need to think about how to scale our systems. We invoke our Lambda functions and state machines, and the capacity is managed transparently.
- Our customers rely on our services, so we need our systems to be highly available. But with serverless, this requires no additional effort. AWS has designed its infrastructure such that an incident affecting a single server, rack, server room, or even an entire data center does not propagate to our applications.
- We use operating systems like Linux and runtimes such as Node.js and Python. These constantly receive security updates and patches, but again, we (mostly) don't need to think about them. The software is tested, patched, upgraded, and rolled out without us ever noticing. This is especially valuable when critical security issues are discovered, such as Log4Shell.

Allowing these challenges to be handled by AWS has been very beneficial to PostNL. However, delegating responsibility also requires a significant commitment from our

organization and engineers. For example, by choosing serverless, we're implicitly choosing to build applications "the AWS way." If AWS decides an event cannot be larger than 256 KB or the API Gateway request timeout is limited to 29 seconds, we have to accept that.

We consider serverless's inherent limits and opinionated design patterns to be trade-offs. We get to use the most scalable, robust, and continually evolving services, and in return, we design our systems to AWS's requirements. We also know that AWS has not designed and defined these limits on a whim. Serverless services are opinionated, but the opinions come from experience. We believe there is value in following the prescriptive guidance of our very experienced partners.

Strong Foundations

PostNL went through three phases in its journey toward in-house software development. First, we worked with partners and vendors in the logistics domain. But we realized that to be truly competitive, we needed to build differentiating software in-house. So, we moved on to the second phase: we started building small-scale innovative solutions ourselves, supported by AWS and front-running AWS consulting partners. In this phase, we experienced firsthand which choices worked, which didn't, and what the implications of our technology choices were. The solutions we built were mostly container-based, and we realized that this added its own flavor of orchestration and maintenance overhead. This led us to the third phase, in which serverless was chosen as the preferred solution for our in-house application development.

Building serverless applications requires a strong foundation of AWS accounts and shared cloud infrastructure. When PostNL started building its first applications, the existing cloud infrastructure was maintained by our AWS partners. As it became clear PostNL was pivoting toward becoming an independent software development business, we decided we needed to own our cloud environments. This gave us the unique opportunity to design an entirely new AWS organization, including its guardrails and governance, its tools and services, and its networking and security architecture. This was to be a greenfield AWS organization with funding and support from our established enterprise. We got one chance to do it right.

We took this opportunity very seriously. We organized multiday workshops with AWS and partners. We talked to other enterprises with an AWS footprint. We documented our learnings from previous phases. We took six months to prepare as well as we could. Then we collected all our insights and distilled them into four principles:

1. All provisioning is automated.
2. Every aspect of our applications is defined as code (as opposed to manual changes).

3. Operational burden is reduced to an absolute minimum.
4. Production data is isolated.

The principles were in part formed from a need to change outdated practices, in part a solidification of known best practices, and in part a vision for the future. For example, the "all provisioning is automated" principle was added because our IT landscape at the time consisted of many individual desks. If a team wanted to build a new application, they had to visit one desk for AWS accounts, another desk for DNS names, another for IP ranges, another for TLS certificates, and so on. After weeks of the enterprise equivalent of haggling, the team finally got all the resources they needed. Then they still had to configure and combine them to create the environment their application could run in. Only when the environment was up and running could they start building for actual business value. Needless to say, this was not the foundation we could build our serverless future on. The "all provisioning is automated" principle was a clear guideline to solve this problem.

On the other hand, the "production data is isolated" principle was a continuation of known best practices. Our physical infrastructure was already segmented into production and non-production networks, and the new cloud infrastructure had to be designed with these security layers in mind.

The "operational burden is reduced to an absolute minimum" principle is aimed at using managed services to their fullest extent. Embracing serverless is one of the ways we put this principle in practice, as we'll cover in the next section.

From Theory to Practice

With the four principles defined, PostNL set out to build the systems and platforms required to achieve them. We founded the CCoE, whose mandate covers the following topics:

- Building and maintaining the shared AWS infrastructure
- Monitoring and optimizing cost across all accounts, including providing tools, insights, and advice to both management and development teams
- Monitoring and improving security across all accounts, providing guardrails, tools, insights, and advice to both management and development teams
- Supporting and providing training and education to improve the cloud skills of PostNL's engineering community based on business-wide and individual team requirements

The CCoE designed and built the platform that codifies, enables, and enforces PostNL's guiding principles. We call this platform the Landing Zone PostNL Engineering (LPE). One of its core responsibilities is bootstrapping new AWS accounts. When a

team starts building a new application, they request a new account, using an online form to supply the name of the application, details about its users, and its cost code. They also specify whether the application needs a full development, test, acceptance, and production (DTAP) set of accounts or a subset of those. The account creation request is automatically forwarded for approval. When the approval is given, other automated systems provision the AWS accounts, including integrations with our single sign-on platform, CI/CD pipelines, security platforms like GuardDuty and SecurityHub, and cost monitoring in QuickSight. The account provisioning system also deploys predefined hosted zones and domain names, and sets up the integration with PostNL's private network through AWS Transit Gateway. The entire process does not take more than 30 minutes after approval, allowing development teams to hit the ground running.

The LPE also maintains the guardrails, or limits, for PostNL's AWS accounts. For example, service control policies (SCPs) prohibit teams from creating IAM users, which would pose an unacceptable security risk. They also prohibit the use of Regions where we have no business activities, and they prohibit manual changes to acceptance and production systems by enforcing read-only access to the AWS console, which guarantees that any changes to these environments are deployed through infrastructure as code, as set out by our guiding principles.

The most distinctive restriction in the SCPs denies all actions related to EC2 instances. In effect, this rule enforces the use of serverless and fully managed services. In the early days, restrictions were also included on services like RDS and Fargate, thus limiting applications to the use of core serverless services like DynamoDB and Lambda. However, we found these rules to be too strict, to the point where they hampered adoption and speed of development. PostNL's current strategy is to allow any new service by default. Only when services significantly increase the risk of operational burden, as is the case with EC2 and Amazon Elastic Kubernetes Service (EKS), are they actively blocked by default.

It is no surprise that simplifying and automating common processes has empowered our teams to achieve their goals faster. Strong guardrails and limits enforce best practices and strategic decisions, while teams retain the freedom to architect their solutions within those boundaries. We will cover these liberties in further detail in the next section.

Freedom Within Guardrails

Imagine a network of highways. They connect many cities and enable people to reach their destinations quickly, reliably, and predictably. People who want to use the highways have to abide by a number of rules and laws. As long as they do, everybody is allowed to travel as they see fit. Whether it's by car, motorcycle, or van, whether they like to drive at the speed limit or a bit below it, whether the windows are opened

or closed—it's up to the drivers. If the rules of the road were too restrictive—for example by mandating the exact driving speed, the makes of vehicles allowed, or the color of the cars—people would not use the highway network to the same extent.

At PostNL, we think of our cloud environments the same way. There are some basic rules set across the entire organization, like the prohibition on the use of EC2 instances. Within those boundaries, however, our teams have the liberty to architect and build solutions the way they see fit. We trust our teams to make the right architectural choices, driven by performance needs, pace of development, and cost-effectiveness. We also have proactive and reactive systems in place to help teams make the right decisions. These include architectural reviews by peers, cost and security monitoring, and support by domain architects.

This approach has proven very effective. Many teams have chosen Lambda, DynamoDB, SQS, and API Gateway as the foundation of their applications. TypeScript is the most commonly used language, though some teams have opted to use .NET, Rust, or Python, driven by specific integration or performance needs, or because of existing expertise. We also see teams choosing RDS, Neptune, and Timestream for their databases. These decisions are generally based on specific workload requirements. And finally, we have teams that have determined that Lambda was not the best compute platform within their context and chose Fargate or Step Functions instead.

We believe that setting clear boundaries and trusting teams to make the right decisions within those guardrails is the right way to scale software development in an enterprise. The platform team maintains the highways and the signage, and the application teams decide if they need a hatchback, a van, or a semi truck for their business.

The Outcome of Serverless Adoption

PostNL journeyed from contracting third-party vendors for all its software needs to building its own in-house software where it matters. The primary reasons for the change were increasing innovation, improving time to market, and reducing overhead. It is fair to say we have achieved all these goals, but we're far from done.

In practice, we've seen that our in-house serverless development strategy has allowed us to scale our core business processes. When we started in 2019 we had no in-house development teams. Currently, we have over 30 teams who are responsible for the full software lifecycle of their applications. We have built the foundations to easily and quickly onboard new teams and applications, which allows us to scale, respond to market trends, and innovate faster than ever before.

The number of operational incidents has also significantly decreased. This is partly due to a very stable and scalable landing zone platform, partly due to the excellent guardrails and limitations that guide our teams to use established best practices, and

in large part due to the mandatory use of cloud native services, which delegates much of the operational burden to AWS.

Shifting from software development by external partners to internal development teams has drastically reduced overhead—both in a financial as well as in a productivity sense. Our teams are better aligned with our strategic goals, allowing them to build exactly what PostNL needs. If there are questions or uncertainties about goals, integrations, or standards, the right people are readily available to answer them. From an architectural viewpoint, we have a much firmer grasp on the past, current, and future enterprise application landscape, allowing us to reduce duplicate efforts and increase the reuse of existing services.

The results described here are strong incentives to continue on our strategic trajectory. To do so, we need to continually attract new talent. We have found that our serverless-only approach has had an unexpected positive effect in this area. Many enterprises—and maybe old logistics enterprises even more than others—suffer from a stuffy, boring image. But our proven track record of building high-volume, large-scale serverless applications with real-world impact does miracles to dispel that image. We have found that both young, eager engineers and experienced software veterans are very interested in working for a company that dares to change, embraces the future, and focuses on reducing operational burden. After all, nobody likes getting called out of bed at 3 a.m. because of a crashed hard drive.

What Were the Pain Points and Learnings?

Like any significant shift, PostNL's pivot to serverless self-engineering was not achieved in a day or without any hurdles. The most significant challenge has centered around the skill set required by our engineers. We have found that software engineers are often focused on a single language, stack, or framework. They may have valid and valuable skills within their areas of expertise, but they don't always translate well to a serverless-oriented enterprise (think monolithic applications in Java, or Kubernetes-based applications). Some topics can be taught on the job or through training, like distributed systems or NoSQL databases. In other cases, it requires "unlearning" patterns already ingrained into an engineer's approaches. As a result, achieving a good fit often requires an investment from both the engineer and the organization, while finding engineers who are a natural match to the job description is a rare occurrence. PostNL and AWS recognize this problem, and together we have created a continual learning and development program to close the gap between present and required experience. Throughout the year, this program organizes classroom sessions, workshop days, knowledge shares, and meetups. We consider this an essential component in supporting our engineers in a fast-moving, quickly evolving IT landscape.

Another challenge we've found in our adoption of serverless is the difficulty of supporting junior developers. Because serverless engineering requires knowledge of software development and cloud infrastructure, distributed systems and integration patterns, AWS quirks and limits, decoupling and failure handling, and many, many more advanced topics, it has been difficult to onboard junior developers and get them productive within an acceptable amount of time. In our case, the path of least resistance often leads to simply not hiring juniors, which is a lost opportunity for both parties involved.

Related to the topic of skill sets and training, we also find the problem of retention. At PostNL, we hire good people and train them to be even better. These engineers work on very interesting challenges and apply serverless on a scale not seen in many other companies. This experience makes our people very valuable, which makes it more difficult to retain them.

Luckily, the challenges and tech stack we offer are also positive differentiators in a very competitive job market, which helps us to attract new talent. In fact, we have heard from some of our engineers that they joined PostNL explicitly because of our outspoken strategy around public cloud and serverless technologies.

The last significant learning from our serverless journey is centered around the balance between a dogmatic and viable approach. As described earlier, PostNL started out with a very principled view on serverless: it did not include relational databases or containers, because both increased the potential operational overhead. In practice, this dogmatic approach hampered our business goals. For example, we found that some applications are more easily modeled on RDS than on DynamoDB, and not every team is immediately capable of delivering a FaaS-based application within an acceptable amount of time. This made pure serverless a very difficult sell, even though it guaranteed low operational overhead. We overcame this challenge by settling on our own definition of "cloud native": it includes serverless and fully managed services like RDS and Fargate but excludes services that require teams to manage compute instances or clusters. Within this definition, we still expect teams to choose the best solution for their given context, but we're no longer dogmatic on a pure serverless play.

Five Pieces of Advice to New Serverless Adopters

PostNL has only been building serverless applications since 2019. In these few short years, we have grown from 0 teams and applications to over 30 teams building and maintaining over 75 applications. We have experienced the value but also the challenges and frictions of serverless in an enterprise. In this last section of this case study, we will share our most valuable insights. We hope they will help you embrace serverless, so you can lead your market like we are leading ours.

Start from Guiding Principles

We did not choose serverless because we liked the name. We did not choose it because our engineers liked the technology. We did not choose it because AWS told us it was the future. No, embracing serverless has never been the goal in itself.

Instead, we defined our business goals first. We want to deliver the best possible experience to our customers. We want to be the market leader. We want to innovate. We want to reduce operational costs. Based on these goals, we defined the guiding principles that would help us achieve them. These guiding principles (described in "Strong Foundations" on page 7) are the core tenets we can base every IT decision on.

With our guiding principles in hand, we investigated and finally chose our solutions—and we found that self-engineering, fully automated landing zones, and serverless were the best tools for the job.

If your enterprise is considering a cloud pivot or a self-engineering strategy, start by defining your goals and guiding principles. You will find that the resulting solutions—serverless or not—are easier to implement, easier to explain, and easier to reassess when they are built on strong foundations.

Automate Everything

Speaking of strong foundations, investing in a platform team and a landing zone platform is invaluable for the success of your serverless adoption. Make sure the platform team automates every repetitive manual action. Keep in mind that serverless allows development teams to drastically reduce the time from proof of concept to production—but this benefit can be completely undone by slow provisioning of requirements like AWS accounts, networking, certificates, and domains. By investing in the automated creation of fully functional landing zones, you will ensure that your teams will be able to deliver value at unprecedented speed.

The need for automation also applies to all other types of undifferentiated heavy lifting. This includes provisioning the tools that every team needs, such as version control, CI/CD pipelines, security monitoring and alerting, cost analysis, issue tracking, incident management, single sign-on, and so on. Engineers should not have to implement or configure any of these tools, and frankly, neither should anyone else. Automating the provisioning of these services for every team and application will deliver outsized results for your organization.

Embrace the Cloud for All It Has to Offer

There are various levels of cloud adoption. The first is lift-and-shift to EC2 instances. The second is the use of managed services like Amazon RDS and Application Load Balancer. The current pinnacle of cloud nativeness is serverless. As an organization,

you can choose to dip your toes in the cloud waters and tentatively experiment with a few core services like EC2 or VPC. In doing so, you might be underwhelmed with the results. After all, these services are not that different from what a colocation facility or private cloud might offer.

On the other side of the spectrum sits serverless. These services are very different from what you might be used to. Out of the box, serverless services offer scalability, reliability, and feature sets no organization could emulate. If your company is capable of embracing serverless, you will find that you can delegate entire classes of problems to AWS. This does require commitment, confidence, and a bit of guts. Instead of trying the waters, you're diving in at the deep end. PostNL has taken the plunge, and we can tell you that the benefits are real. In our experience, the payoff has a linear relationship with commitment—and thus, a fully committed enterprise will reap the best results.

Comprehensively Analyze the Total Cost of Ownership

One of the hurdles of embracing serverless in the enterprise is the perceived increased cost. Analysis might focus on prices per request, prices per gigabyte, or prices per minute. On any of these metrics, serverless services might appear more expensive than their serverful alternatives. But comparisons like these are incomplete and do not take the total cost of ownership into account.

One of the main objectives of the serverless adoption at PostNL has been to reduce operational costs. We have largely achieved this by reducing the amount of hours spent on operational tasks, such as patching, provisioning, and managing operating systems, instances, and clusters. In the end, the serverless cost per request, gigabyte, or user might be higher, but this is easily compensated for by the money saved on human labor. This is the big promise of serverless: it allows companies and teams to delegate the most difficult (and expensive!) parts of service operation to AWS. It should be no surprise that the responsibility AWS carries is reflected in the unit price of serverless services.

Another component of the TCO analysis should be the cost of incidents. Because serverless forces application developers to design distributed and scalable systems, the risk of incidents can be drastically reduced. At PostNL, we have seen that the adoption of cloud native application development has had a measurable and significant positive impact on the number of P1 incidents. Each incident avoided represents a net reduction of loss across all metrics, including engineering hours, customer satisfaction, profits, and reputation.

These benefits are not immediately obvious when you look at the AWS pricing pages, but they should always be taken into account in the total cost of ownership. In a comprehensive analysis, you will find that the price of serverless is hard to beat.

Never Stop Learning

The final piece of advice is to never stop learning. When starting out with serverless, you might be overwhelmed by the sheer volume of services, features, limitations, and peculiarities. This is okay. You don't need to know every detail to build your first application. You will find out what works and what alternatives are available as you go. The important thing is to have an open mind, stay up-to-date, investigate your options, and change your approach if needed.

Keep in mind that the feature velocity at AWS is staggering. Every year, thousands of changes are being released, from minor features to entirely new services and regions. There are few people in the world that are able to keep up with all of them, but luckily there is a very large and active community that filters and distills the important from the mundane. You can keep up through social media, newsletters, Slack groups, RSS feeds, meetups, and conferences.

Some of the AWS service updates might allow you to replace certain of your application's features, hundreds of lines of code, or even entire services with AWS-managed offerings. This can reduce cost, operational burden, and the risk of incidents. Staying involved and investing in the continuous improvement of your services is one of the best ways to stay ahead of the competition.

Organizations, too, should never stop learning. PostNL has always been open to change, which has allowed us to transition from steam trains in 1799 all the way to serverless in 2019, with many different stages in between. This has only been possible because we chose strategies that made sense at the time and revised them when new insights emerged. We can only advise your organization to take the same dynamic view on technology, or you will lose your edge to competitors who do.

In the end, the only constant is change. The people and organizations best equipped to react to change will be the leaders of tomorrow.

About the Contributor

Luc van Donkersgoed is a well-known speaker and author in the AWS serverless community. He has a long history of building high-volume, mission-critical web infrastructure, first on bare metal, then on self-hosted virtual machines, then on EC2, and finally on AWS serverless.

Luc has experienced firsthand how serverless allows developers to stop thinking about servers, networking, and (most of) the problems associated with scaling. These complex challenges are delegated to AWS, allowing developers to focus on the business logic of their applications. However, using serverless at scale requires a thorough understanding of common patterns, failure modes, and limits. Luc strives to help his team, colleagues, and local and global AWS communities embrace and grow these skill sets so they, too, can experience the focus and scale serverless brings to application development.