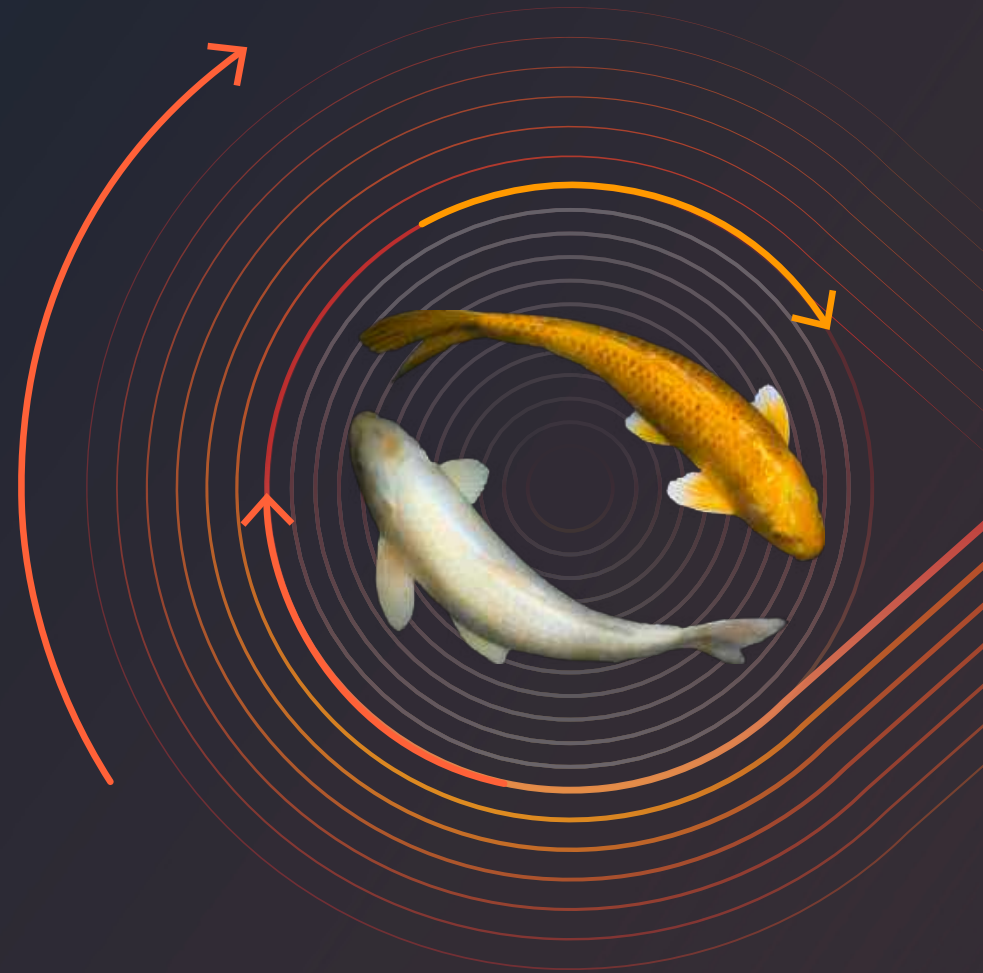


Modern Applications

Reinventing how your business delivers value



Modern applications are changing how you build customer value

Digital transformation has dramatically impacted the way companies deliver value and the rate at which they make changes to their products and services. Every company is becoming a technology company—companies are increasingly building products that are the technology itself or heavily influenced by technology. In order to compete in this new world, businesses must create better digital products, and they must do it at an increasingly rapid pace.

Many companies are innovating faster by changing the way they design, build, and manage applications, through what we call modern applications. Modern applications increase the agility of your teams and the reliability, security, and scalability of your applications. Modern applications are built faster than ever, with the ability to scale quickly to potentially millions of users, have global availability, manage petabytes if not exabytes of data, and respond in milliseconds.

“Invention requires two things: the ability to try a lot of experiments, and not having to live with the collateral damage of failed experiments.”

– Andy Jassy, CEO, Amazon Web Services



50%

of information communication technology is predicted to be directly allocated for digital transformation by 2023

67%

of executives believe they must pick up the pace to remain competitive

90%

of new applications are predicted to be cloud native by 2025

To build modern applications, you may need to reconsider your application's architectural patterns, operational model, and software delivery process. While these shifts are dramatic at an organizational level, the process doesn't need to be brutal. Many organizations take an inspired leap to build new modern apps in the cloud, but plenty of others take a step-wise approach, one app at a time.

We observed common characteristics of modern applications through our experience building applications for Amazon.com as well as from serving millions of AWS customers. Applications with these characteristics enabled our customers to increase agility, lower costs, and build better apps that support the success of their businesses. While you can modernize applications from any starting point, the outcome is the same: applications that are more secure, reliable, scalable, and quickly available for your customers and partners.

This guide covers the following topics:

- Digital innovators
- Characteristics of modern applications
 - 1 Culture of ownership
 - 2 Architectural patterns: microservices
 - 3 Computing in modern applications: containers and AWS Lambda
 - 4 Data management: purpose built & decoupled
 - 5 Developer agility: abstraction, automation, and standardization
 - 6 Operational model: as serverless as possible
 - 7 Management & governance: leveraging programmatic guardrails
- Start building modern applications today

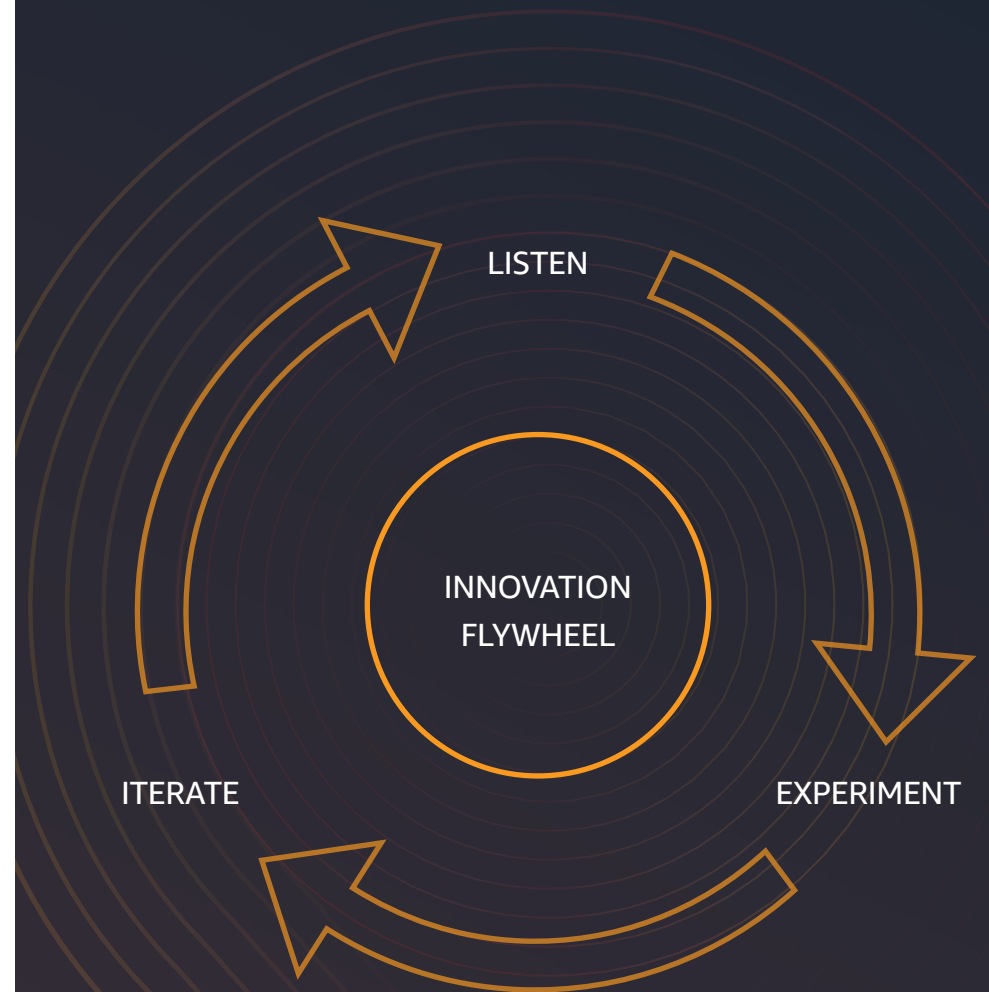
Innovation means listening to your customers

In a recent *Vision Report, Digital Rewrites the Rules of Business*, Forrester Research defines the customer-centric mindset of a *digital innovator*. The core mission of these modern disruptors is to:

*"...Harness digital assets and ecosystems to continually improve customer outcomes and, simultaneously, improve operational excellence...by applying digital thinking to customer experiences, operations, ecosystems, and innovation."*¹

Focusing on your customer means making business decisions by working backward from your customer's point of view. It means constantly evolving products and services to better deliver the outcomes that delight customers. And, finally, it means listening to what your customers truly care about so that you can continue inventing and iterating on their behalf. This is called the "innovation flywheel."

The basic idea is that the driver for any innovation begins with customer demand, improves with customer feedback, and repeats constantly (and profitably) until the demand changes and the whole cycle begins again. The faster your teams can get your own innovation flywheel spinning, the stronger your differentiation will be in the market and the more you will stand apart from competitors.



¹ <https://www.forrester.com/report/Digital+Rewrites+The+Rules+Of+Business/-/E-RES137090>

Modern application development is a powerful approach to designing, building, and managing software in the cloud. This proven approach increases the agility of your development teams and the reliability and security of your applications, allowing you to build and release better products faster. From our experience helping organizations of every kind build applications, we've identified common characteristics of modern applications that digital innovators rely on for success.

Characteristics of modern applications

- 1 Culture of ownership
- 2 Architectural patterns: microservices
- 3 Computing in modern applications: Containers and AWS Lambda
- 4 Data management: purpose built & decoupled
- 5 Developer agility: abstraction, automation, and standardization
- 6 Operational model: as serverless as possible
- 7 Management & governance: leveraging programmatic guardrails

Creating a culture of ownership

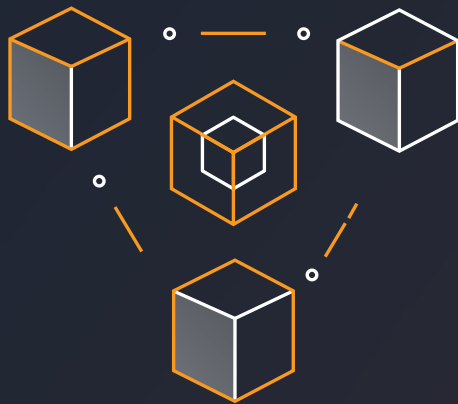
Innovation ultimately comes from people, and so enabling your people to deliver better customer outcomes is where modern application development starts. We use the concept of “products, not projects” to describe how this impacts team structure. Simply stated, it means that the teams that build products are responsible for running and maintaining them. It makes product teams accountable for the

development of the whole product, not just a piece of it.

After more than a decade of building and running the highly scalable web application, Amazon.com, we’ve learned firsthand the importance of giving autonomy to our teams. When we gave our teams ownership of the complete application lifecycle, including taking customer input, planning the road map, and developing and operating the application, they became owners and felt empowered to develop and deliver

new customer outcomes. Autonomy creates motivation, opens the door for creativity, and develops a risk-taking culture in an environment of trust.

While embracing a culture of ownership is not inherently technical, it remains one of the most challenging aspects of modern application development. Empowering teams to become product owners involves changing the mindset of your organization, the structure of your teams, and the work for which they are responsible.



Building a culture of innovation

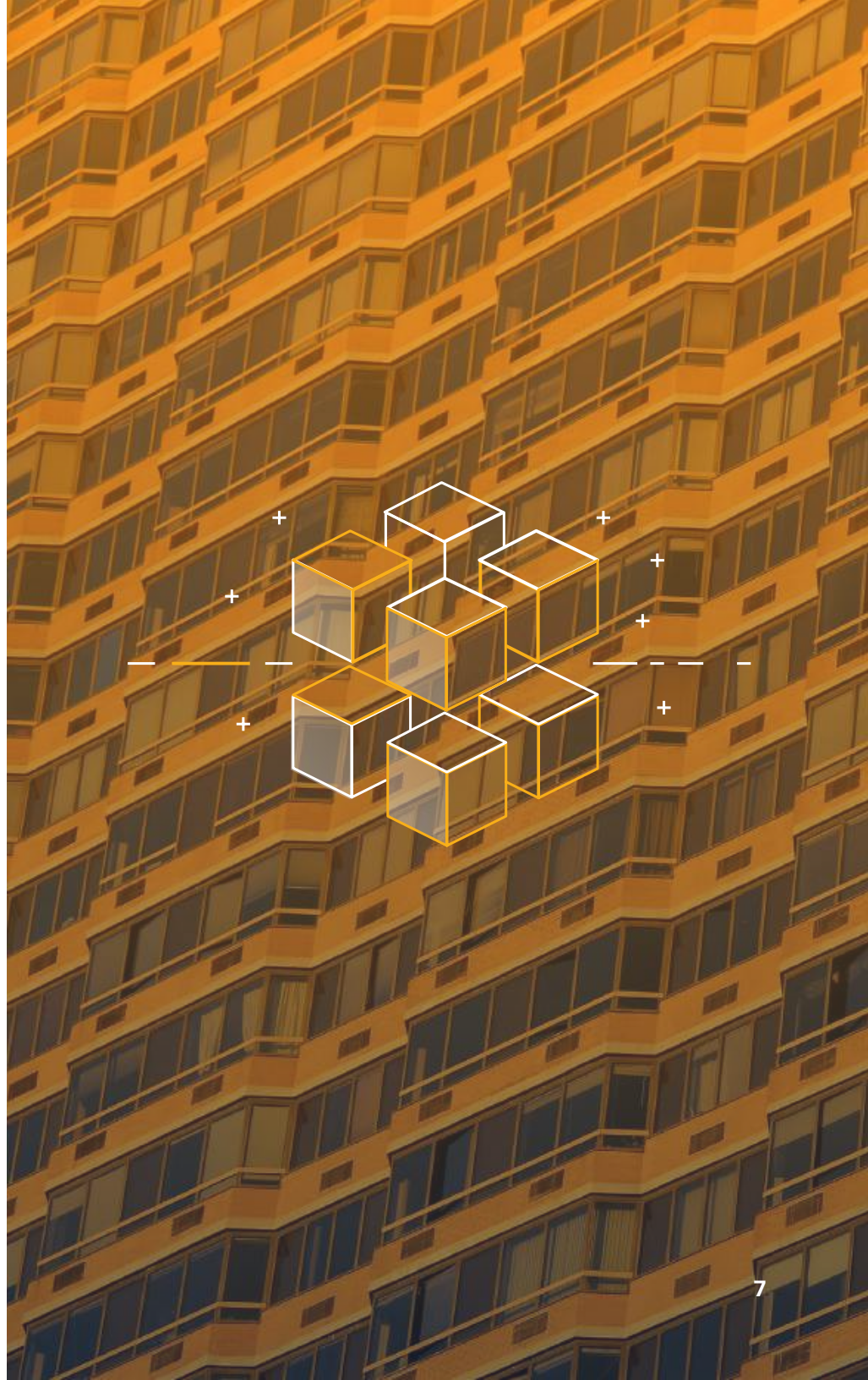
- 1 Start with the customers** – Every innovation should start with a customer need and ultimately delight your customers. Prioritize relentlessly to focus on customer demand.
- 2 Hire builders and let them build** – Remove any obstacles that slow the process of building and releasing products and features for customers. The faster you iterate, the faster your flywheel spins.
- 3 Support builders with a belief system** – Don’t pay lip service to innovation—live and breathe innovation in all areas of the business, from leadership to sales to support.

Architectural patterns: microservices

Although your monolithic app might be easy to manage today, challenges often arise as you grow, including how to distribute ownership of the app across your teams. You can build a strong culture of ownership but still struggle to scale up if your application architecture includes hard dependencies that prevent teams from taking ownership of the final product. This is why we recommend building microservices architectures for apps that grow and change rapidly.

Microservices are the architectural expression of a culture of ownership—they neatly divide complex applications into components that a single team can own and run independently. With a monolith, you have many developers all pushing changes through a shared release pipeline, which causes friction at many points of the lifecycle. Upfront, during development, engineers need to coordinate their changes to make sure they're not breaking someone else's code. To upgrade a shared library to take advantage of a new feature, you need to convince everyone else to upgrade at the same time—a tough ask! And if you want to quickly push an important fix for your feature, you still need to merge it with changes in progress.

After development, you also face overhead when you're pushing the changes through the delivery pipeline. Even when making a one-line change in a tiny piece of code, engineers need to coordinate their changes ahead of time, merge their code, resolve conflicts within releases, rebuild the entire app, run all of the test suites, and redeploy once again.

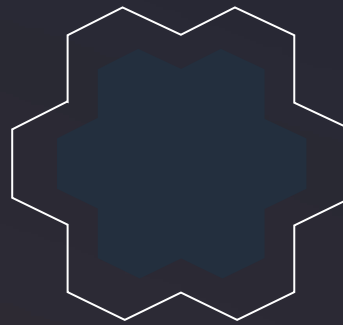


With a microservices architecture, an application is made up of independent components that run each application process as a service. Services are built for business capabilities, and each service performs a single function. Because it runs independently and is managed by a single development team, each service can be updated, deployed, and scaled to meet the demand for specific functions of an application. For example, an online shopping cart can be used by many more users during a sale. Microservices communicate data with each other via well-defined interfaces, using lightweight APIs, events, or streams. Our customers are increasingly relying on event-driven architectures—those in which actions are triggered in response to changes in data—to improve application scalability and resiliency while also reducing costs.

EVERYTHING VS. ONE THING: TWO TYPES OF APPLICATIONS

Monolith apps

Do everything



Single app

Must deploy entire app

One database

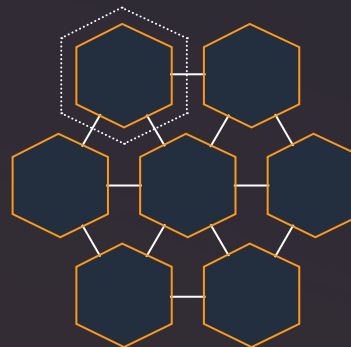
Organized around technology layers

State in each runtime instance

One technology stack for entire app

Microservices

Do one thing



Minimal function services

Deployed separately, interact together

Each has its own datastore

Organized around business capabilities

State is externalized

Choice of technology for each microservice

**centrica**

Centrica is a British multinational energy and services company that was looking to decrease their costs and increase their agility by changing the way their application was architected. They opted to refactor to a microservices architecture and adopt a serverless strategy to achieve those goals.

In order to change their organization they set up a serverless working group with representative teams and started building a pilot together. With that success behind them, other teams in the organization have now also adopted the approach—serverless is now common across the organization.

Now with serverless, they're able to see and respond to customer issues in real time, something they had not been able to do before.

[Watch the video >](#)

Data management: purpose built and decoupled

The shift to microservices architectures has a big impact on how organizations store and manage data. If each microservice is interacting with a monolithic database, the database is still a single point of failure. That is why modern applications are built with decoupled datastores, each of which is part of a single microservice. This may seem like a radical departure from a traditional application architecture, but when we acknowledge that a monolithic application poses scaling and fault tolerance challenges as it grows, we need to apply those same principles to a database.

In the past decade, databases have evolved significantly. Traditional applications such as ERP, CRM, and e-commerce mostly used relational databases because these applications mainly required recording transactions and storing structured data in GBs and occasionally TBs in size. But, approaches to building applications and application requirements themselves have changed. Applications need to work with TBs and PBs of data, support millions of customers distributed globally, and process millions of requests per second with very low latency. To support these needs, developers are increasingly building applications using microservice architectures and choosing relational and non-relational databases that are purpose-built to meet their application's specific needs, like storing key-value pairs and documents.





Launched in Japan in 1996, Pokémon has become one of the most popular entertainment brands worldwide. After the launch of Pokémon GO in 2016, the number of users requiring access to this system increased to more than 300 million in two years. The company decided to migrate to a fully managed data solution, Amazon Aurora, to free up time and resources to focus on strategic initiatives.

As a result of the move to Aurora The Pokémon Company has reduced their costs by tens of thousands each month. They have also dramatically improved reliability - after experiencing 168 hours of downtime or performance degradation in the six months prior to migration, they have now had zero since the migration.

[See the full story >](#)

"Using Amazon Aurora, we went from 300 nodes to 30, and we are no longer paying for database licenses. Our monthly database cost has dropped by tens of thousands of dollars each month."

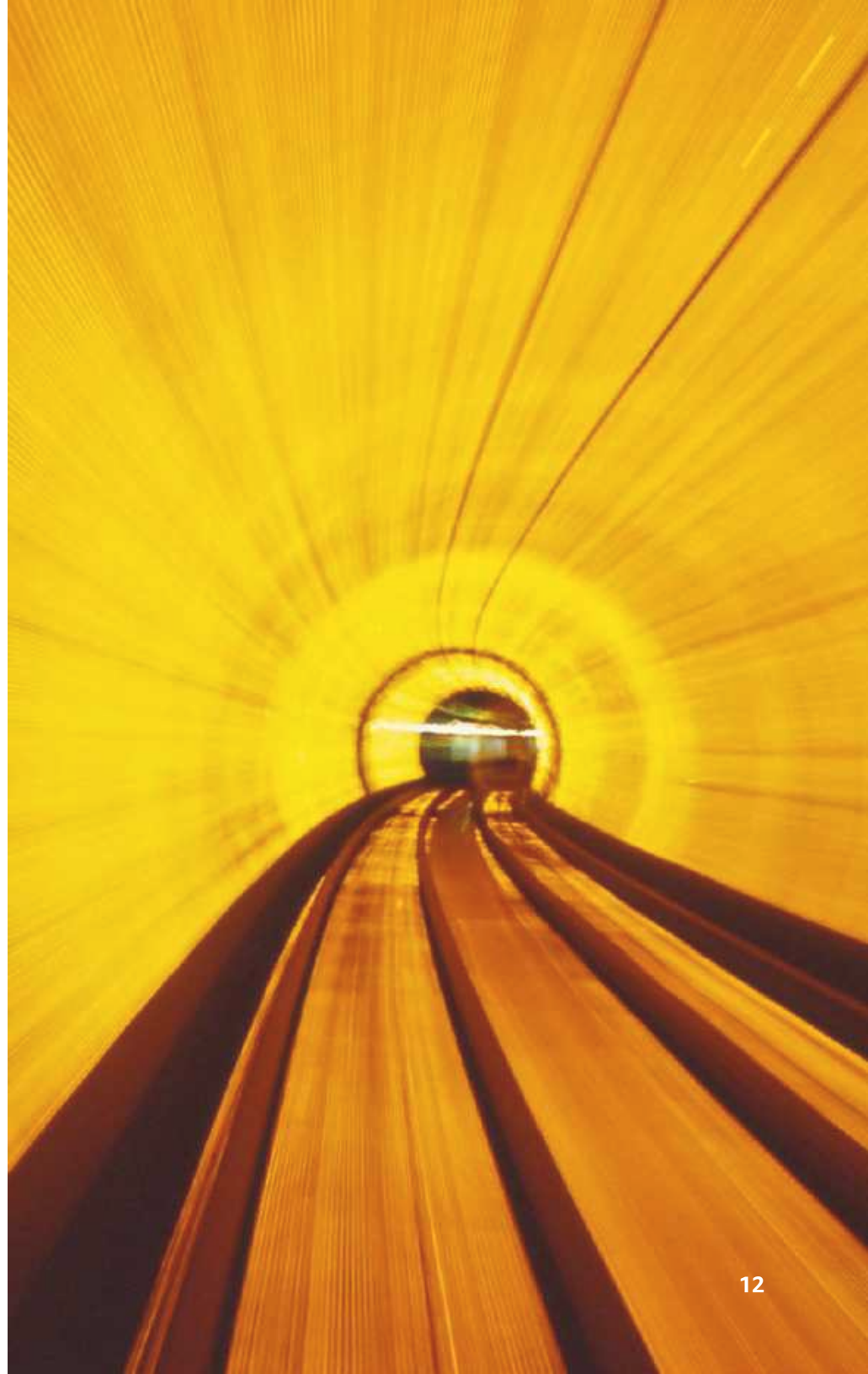
– Jeff Webb, Development Manager, The Pokémon Company International.

Leveraging containers and AWS Lambda

Moving from monolith to microservices affects the way you package and run code—changing the way we compute in modern applications. Instances are no longer the only option; modern applications now use either containers or serverless event-driven compute service AWS Lambda. Much of this change arose from a desire on the part of developers to manage dependencies within each service through a pipeline, which in turn resulted in new ways of packaging applications.

Choosing between instances, containers, and Lambda boils down to how much your enterprise requires flexibility versus simplicity—which we define as seamless integration with other services and features. There are natural tradeoffs the more you move toward simplicity. And one of the more common problems that we continue to try to resolve is how to move toward simplicity without giving up all the flexibility.

Increasingly, we're seeing our customers choose containers and Lambda for compute. Containers have emerged as the most popular way to package code, which makes this a great way to modernize legacy applications. Containers offer excellent portability and flexibility over applications' settings. Lambda offers the most simplicity so that the only code you write is business logic, and it enables you to benefit from the most agility that the cloud has to offer.





AbiBird is a wholly owned division of ATF Services, an Australian group of companies with 350 employees and contractors, and 60 branches in Australia and New Zealand. AbiBird offers a service consisting of infrared sensors for homes, which help monitor the activities of elderly residents through a smartphone-based app.

Abibird was running on Microsoft Azure and found that their they were raising about 62 support tickets with their cloud provider to keep their service running. The needed more stability and scalability.

Abibird chose to move to AWS and uses a combination of compute services based on their needs. They use AWS Lambda on the back end due to its ease of use and its scalability as well as for the managed nature of the service.

They also use AWS container services, hosting a public API on Amazon ECS and leveraging AWS Fargate to run their containers without the hassle of administering their own fleet of virtual machines.

Since launching this system on AWS in 2019, Abibird has not had to raise a single support ticket, which enables them to run efficiently with a minimal amount of support overhead.

[See the full story >](#)

"We believe our technology running on AWS will allow the growing number of elderly people to live independently at home—where they are happier."

– Robin Mysell, CEO, ATF Services and AbiBird

Developer agility: abstraction, automation, and standardization

Microservices architectures make teams agile and enable them to move faster, which means you're building more things that need to get released—great! However, you won't get new features to your customers faster if your build and release process does not keep up with your team's pace. Traditional development processes and release pipelines are slowed mainly by manual processes and custom code. Custom code is ultimately a long-term liability because it introduces the possibility for errors and long-term maintenance. Manual steps—from code changes and build requests to testing and deploying—are the greatest drag on release velocity. The solution involves abstraction, automation, and standardization.

In order to speed the development process, abstract away as much code as possible, particularly the lines of non-business logic code, required to develop and deliver production-ready apps. One way to do this is to employ frameworks and tooling that reduce the complexity of provisioning and configuring resources. This gives developers an ability to move quickly, while also enforcing best practices for security, privacy, reliability, performance, observability, and extensibility throughout the development process. Development frameworks give you confidence that your architecture will support your business growth long term.

A bit more detail:

Continuous integration - Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g., a CI or build service) and a cultural component (e.g., learning to integrate frequently).

Continuous delivery - Continuous delivery is a software development practice where code changes are automatically prepared for a release to production. Continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.

Learn how Amazon automates safe, hands-off deployments >

By defining your software delivery process through the use of best-practice templates, you can provide a standard for modeling and provisioning all infrastructure resources in a cloud environment. These “infrastructure as code” templates help teams get started on the right foot because the template provisions the entire technology stack for an application through code rather than using a manual process.

Through automation, you can create a repeatable motion that speeds up your software delivery life cycle. Automating the release pipeline through continuous integration and continuous delivery (CI/CD) helps teams release high-quality code faster and more often. Teams that practice CI/CD ship more code, do it faster, and respond to issues quicker. In fact, according to the *Puppet state of DevOps* report, teams that employ these practices have a failure rate that is five times lower, a commit-to-deploy rate that is 440 times faster, and a rate of deployment that is 46 times more frequent.² Most notably, teams that practice CI/CD spend 44 percent more of their time creating new features and code instead of managing processes and tools.

CI/CD pipelines have become the new factory floor for building modern applications. At Amazon, we started using CI/CD to increase release velocity, and the results were dramatic—we have achieved millions of deployments a year and grow faster every year. To help companies benefit from our experience, we built a suite of developer tools based on the tools we use internally, so our customers can deliver code faster.

² <https://puppet.com/resources/report/2017-state-devops-report/>



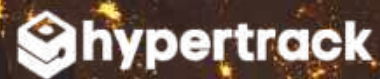
With AWS, lululemon Athletica can stand up development environments in minutes instead of days, automate its environments, and enable continuous integration and deployment. The Canadian company sells yoga-inspired apparel and other clothing at more than 350 locations throughout the world. lululemon runs its dev and test environments—as well as an upcoming mobile app—on the AWS Cloud.

lululemon decreased its time to build new production accounts from two days to a few minutes, using AWS CloudFormation templates and AWS CodePipeline. With that increased agility, lululemon dev teams can now experiment and get to the best solutions rather than having to settle for what they have resources committed to.

[See the full story >](#)

“Any continuous integration and deployment pipeline should be automated, easy to manage, and discoverable, and that’s exactly what we get using AWS. We get a level of simplicity and transparency we simply couldn’t have in our previous on-premises environment.”

– Sam Keen, Director of Product Architecture, lululemon



HyperTrack is a self-serve cloud platform for live location tracking through apps. When it launched in late 2015, HyperTrack needed to build a platform that could scale automatically to meet their anticipated growth, without reducing the time their developers spent on building new features.

HyperTrack opted to use AWS Amplify for a mobile development framework and a serverless architecture, in order to scale up and down automatically without engineering intervention.

As a result, the company has realized a 30 percent cost savings compared to the architecture they were using before we switched to serverless. A big part of that savings comes from not needing operational resources to focus on server management - HyperTrack saves 40 hours of work, every single week, while managing millions of events.

[See the full story >](#)

Operational model: as serverless as possible

As your architectural patterns and software delivery processes change, you will probably want to adopt an operational model that enables you to offload any activity that isn't a core competency of your business. To gain agility that can enable rapid innovation, we recommend building microservices architecture, operating and deploying software using automation for things like monitoring, provisioning, cost management, deployment, and security and governance of applications. Choosing a serverless-first strategy—opting for serverless technologies wherever possible—enables you to maximize the operational benefits of AWS.

A serverless operational model allows you to build and run applications and services without provisioning and managing servers. This eliminates server management, provides flexible scaling, enables you to pay only for value, and automates high availability. This model lets you build and manage the aspects of your application that deliver customer value without having to worry about the underlying detail.

Whether you are building net-new applications or migrating legacy, building with serverless primitives for compute, data, and integration will enable you to benefit from the most agility the cloud has to offer.

How do we define serverless at AWS?

When we say serverless, we mean it's the removal of the undifferentiated heavy lifting that is server operations. This is an important distinction because it allows you to focus on the building of the application rather than the management and scaling of the infrastructure to support the application. The four tenets of a serverless operational model are:

1. **No server management** – There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.
2. **Flexible scaling** – Your application can be scaled automatically or by adjusting its capacity through toggling the units of consumption (e.g., throughput, memory) rather than units of individual servers.
3. **Pay for value** – Instead of paying for server units, pay for what you value—consistent throughput or execution duration.
4. **Automated high availability** – Serverless provides built-in availability and fault tolerance. You don't need to architect for these capabilities since the services running the application provide them by default.



A serverless operational model is ideal for high-growth companies that want to innovate quickly. Serverless enables teams to move even faster and keep a laser focus on the activities that differentiate your business, so you can speed up your innovation flywheel.



iRobot

iRobot uses technology to help their customer offload a lot of tasks people don't want to do like lawn and floor care. They apply the same concept in their own development practices, offloading their own operational management to AWS to enable their developers to focus on building technology for their business.

In 2015 they launched their first connected Roomba and realized it wasn't going to scale to meet the customer demand. By adopting a serverless AWS architecture, iRobot now offers its internet-connected robotic vacuums and mops on a global scale. When there are peaks in demand, like the 20 times traffic spike

the company experiences during Christmas time, it's now a non-event. Their infrastructure automatically scales to meet the need.

[Watch the video >](#)

Management & governance: leveraging programmatic guardrails

Managing your organization securely, legally, and safely is priority one for nearly every company. But often, strong governance becomes checkpoints that slow down innovation. Philosophically when it comes to operations, organizations have two options. They can move slowly and safely, or they can impose no limitations and move at lightening speed, but introduce serious risks to the application and the business.

Ideally, you can have it both ways, and that's why companies are increasingly adopting the concept of guardrails. Guardrails enable our teams to operate quickly without becoming a risk to the business.

Guardrails are mechanisms, such as processes or practices, that reduce both the occurrence and blast radius of undesirable application behavior. Usually expressed as code, guardrails are established and standardized through a central team and delivered and updated to teams of builders in automated and programmatic ways. This is where operators, or those individuals in roles tapped to establish, manage, and maintain the multiple, flexible pieces of distributed architectures, are key.

There are a few different areas we tend to see operators working. In a modern application world, operators set guardrails for the monitoring, provisioning, deployment, cost management, and security and governance of applications. Operators have a role in not only establishing the boundaries, rules, and best practices for each of these areas, but also in building automated solutions, and packaging and deploying these guardrails throughout the organization.

Governance Philosophies

Free for all	Guardrails	Central control
Fast dev time, but risk to legal & app reliability <i>Chaos</i>	Fast time & low risk to the business <i>Win win</i>	Low risk but very slow to release <i>Dependencies & time lags</i>

Common Uses of Guardrails



Monitoring

- CPU utilization
- Database throughput
- Business processes



Provisioning

- Access permissions
- Resource availability
- Configuration



Security & compliance

- CPU utilization
- Database throughput
- Business processes



Cost management

- Resource costs
- Resource utilization
- Spend run rates



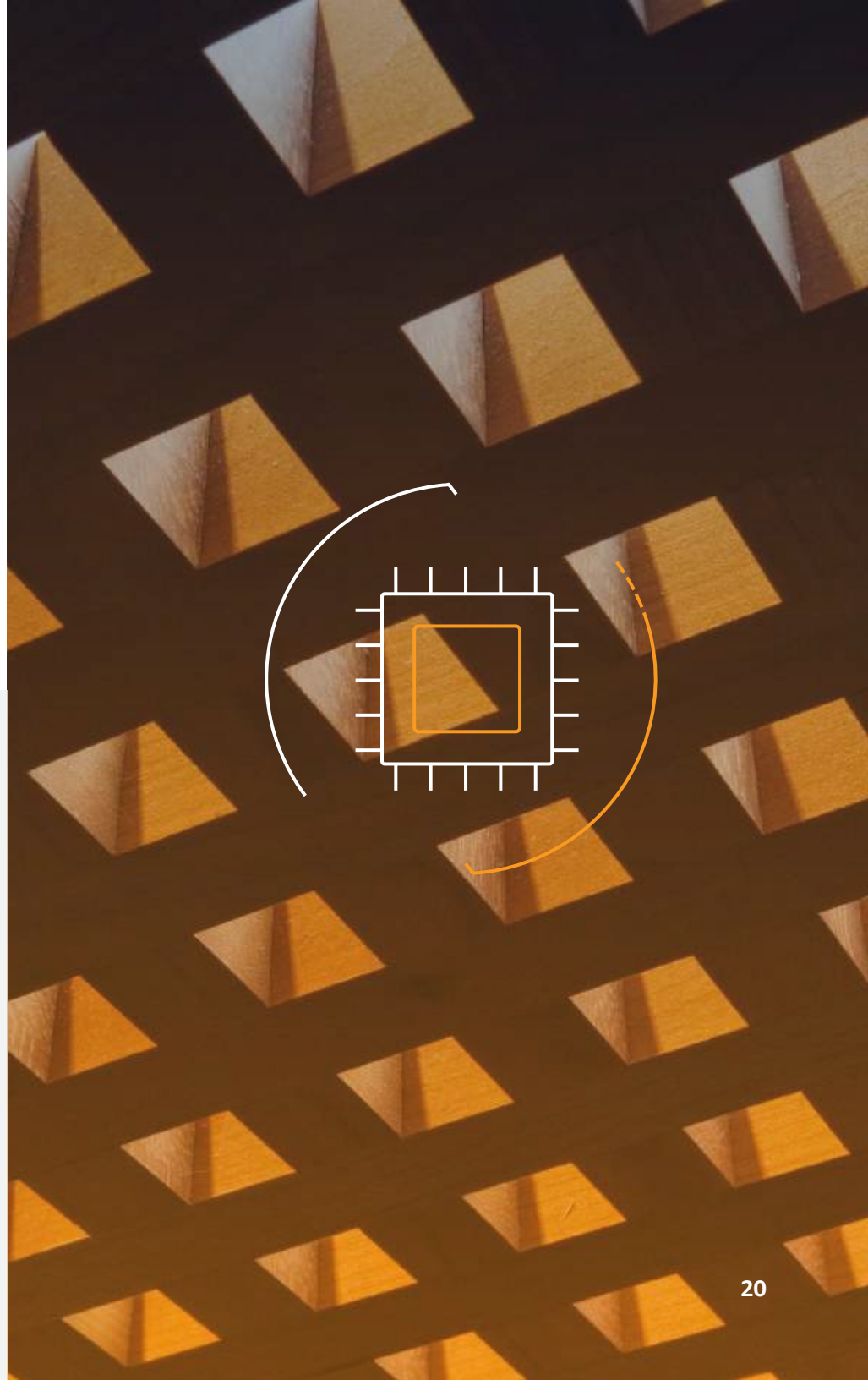
Deployment


- Time window
- Toolsets available
- Size or timing of test releases

The concept of guardrails is essential to application security as well. It isn't surprising that *2017 State of DevOps* report by Puppet states that guardrails, or integrating security deeply into the software delivery lifecycle, makes teams more than twice as confident of their security posture. The report also suggests that firms at the highest level of security integration are able to deploy to production on demand at a significantly higher rate than firms at all other levels of integration—61 percent are able to do.

A bit more detail:

- › **DevOps** is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.
- › **DevSecOps** is the philosophy of integrating security practices within the DevOps process. DevSecOps involves creating a "Security as Code" culture with ongoing, flexible collaboration between release engineers and security teams.





coinbase

Coinbase is a digital currency wallet and platform company based in San Francisco. To protect its customers from attacks, Coinbase engineers must be able to quickly, reliably, and securely deploy updates and new features for all company systems, with some software deployments needing 20 hours or longer to finish. Adopting a serverless architecture, anchored by AWS Lambda and Step Functions, enables the company to increase the rate of successful mission critical deployments from 90 percent to 97 percent.

The new approach substantially reduces the complexity of Coinbase's architecture, which in turn improves visibility for the team to monitor and troubleshoot, ultimately reducing the number of trouble tickets about failed deploys. It also simplifies the process of auditing, since they have a single audit trail for all deployments.

[See the full story >](#)

“With deployers built on AWS Step Functions and AWS Lambda, our engineers can move code into production safely. The upshot is that we can release new features more often, respond quicker to security threats, and more easily achieve our SLAs. This adds up to an even better, more secure, customer experience.”

—Graham Jenson, Senior Infrastructure Engineer, Coinbase

Build modern applications today

Modern applications create competitive differentiation by enabling rapid innovation. By changing your architectural pattern, operational model, and software delivery process, you can shift resources from standard operations to differentiating activities. You can experiment more, and turn ideas into releases faster. You can foster an environment where builders spend more time building. Modern applications are how organizations, including Amazon, innovate with speed and agility.

Modern applications on AWS help to:

- Speed up time to market
- Increase innovation
- Improve reliability
- Reduce costs

Any starting point. Any application.
Any innovation. AWS is how modern
application development happens.

Modern application development on AWS

AWS is trusted by millions of customers around the world—including the fastest-growing startups, largest enterprises, and leading government agencies—to power their infrastructure, increase their agility, and lower costs. AWS offers a comprehensive portfolio of services to support your business as you develop modern applications.

Learn more about implementing the best practices of modern application development →

AWS services for modern application development

DEVELOPER TOOLS



Mobile/Front-end Development
AWS Amplify



Source code management
AWS CodeCommit



Pipeline automation
AWS CodePipeline



Infrastructure as Code:
Typescript, Java, Python, C#
AWS Cloud Dev. Kit (CDK)



Build and test automation
AWS CodeBuild



Artifact Repository
AWS CodeArtifact



Infrastructure as Code:
YAML/JSON
AWS CloudFormation



Deployment
AWS CodeDeploy

COMPUTE



Serverless Event-Driven Compute
AWS Lambda



Serverless Containers
AWS Fargate



Kubernetes container orchestration
Amazon EKS



Container orchestration
Amazon ECS

INTEGRATION



GraphQL API
AWSAppSync



Message Queues
Amazon SQS



Pub/Sub & push notifications
Amazon SNS



Visual Workflows
AWS Step Functions



REST API
Amazon API Gateway



Service Mesh
AWS App Mesh



Data streaming
Amazon Kinesis



Event Bus
Amazon EventBridge

DATA STORES



Object storage
Amazon S3



Key-value database
Amazon DynamoDB



Relational database
Amazon Aurora



File system
Amazon EFS



Document database
Amazon DocumentDB



In-memory database
Amazon ElastiCache