The path to cloud-native applications

8 steps to guide your journey

Red Hat
Application Services

Table of contents

Introduction

Applications have become the way a majority of organizations interact with their customers, partners, and employees. This rapid rise of new digitally native capabilities has disrupted traditional business models and required established companies and industry sectors to adapt and modernize their operations.

For the majority of organizations, creating innovative digital experiences means pivoting to a culture of organizational agility, where the rapid pace of demand can only be satisfied by faster and more flexible development and delivery models. But most organizations do not have the luxury of completely rebuilding their technology foundation or immediately adopting new practices and mindsets. Instead, they are embracing gradual yet fundamental shifts in culture, processes, and technology to support greater velocity and agility. This e-book outlines eight steps that any organization should consider when looking to adopt a cloud-native approach to applications.

"At its core, cloud-native development is as much about teams, people, and collaboration as it is about technology… Collaboration is key to building apps in an iterative, flexible way — stakeholders and makers all need to contribute to how the product is created, coded, tested, and deployed."

Red Hat research report: Cloud-native development outlook, June 2021

Step 1: Evolve culture and practices for the cloud

The path to cloud-native applications requires development, line-of-business, and IT operations teams to evolve in many different ways to build and deploy apps faster and more efficiently. Regardless of industry or size, every business needs to consider the wide range of activities, technologies, teams, and processes that require collaboration and coordination for the successful deployment of cloud applications. While traditional approaches to using public or hybrid cloud resources let teams make independent decisions and move quickly, those strategies have also created isolated data and environment processes that make it difficult to innovate.

In an era of rapid innovation, the complexity of managing multiple distributed environments, highly customized legacy applications, and new application workloads can create challenges for those organizations developing a unifying cloud strategy for their applications. Often, without an enterprise-wide cloud strategy, organizations are left with untapped potential in their application portfolio.

The adoption of a collaborative cloud culture relies not just on using new tools and technologies, but also on encouraging people's willingness and trust to embrace a more integrated and collaborative approach to developing and delivering applications. The culture of open source software projects can be a guide to building a cohesive connected cloud strategy for applications.

Step 2: Speed existing applications using microservices

When embarking on a cloud-native application journey, organizations should not only focus on new development. Many legacy applications are critical to business operations and revenue generation and cannot simply be replaced. Rather, they need to be integrated with and work alongside new cloud-native applications. But how do you speed up an existing monolith? The answer is to move your existing monolithic architecture to a more modular, microservices-based architecture and application programming interface (API)-based communication.

Before beginning the onerous task of refactoring monolithic applications into microservices, organizations should first create a solid foundation for their microservices architecture.

Moving to a microservices approach does not mean rushing to move everything at once. Break down your monolith into smaller components at your own pace, using a phased approach. Using a phased approach ensures that applications are built following solid design principles and properly defined domain boundaries. This approach supports a more gradual and less risky transition to a microservices architecture, if needed, and sets the foundation for a successful microservices architecture.

Applications that are highly dependent on legacy platforms can still be updated and deployed faster by moving the existing monolith to a container-based platform. This shift speeds deployment and delivers a higher return on investment (ROI). Subsequent integrations or features for the monolith can be built using cloud-native techniques and approaches.

Step 3: Use application services to speed development

Reusability has always been key to speeding software development, and cloud-native applications are no exception. However, reusable components for cloud-native applications must be optimized and integrated into the underlying infrastructure to fully provide the speed and scalability that cloud offers.

Why re-create a caching service, rules or workflow engine, integration connectors, mobile and API management capabilities, data virtualization service, messaging broker, or serverless framework when you don't have to? You can use existing components that have been optimized and integrated into the underlying container-based infrastructure. These application services—whether they are Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), or integration PaaS (iPaaS) offerings—are effectively ready-to-use developer tools.

While DevOps process and containerization accelerate application delivery and deployment, cloud-native applications may need one or more of these types of services to help accelerate development and get new applications to market faster. For example, developers can take advantage of application services specifically built to perform well in a container-based infrastructure. These services are designed to take advantage of platform capabilities, such as continuous integration/continuous delivery (CI/CD) pipelines, rolling and blue-green deployment, automated scalability, fault tolerance, and more.

Step 4: Choose the right tool for the right task

With cloud-native applications, the chosen development language or framework is increasingly tailored to the specific business application need. To manage the resulting increase in complexity and application diversity, organizations need a container-based application platform that supports the right mix of frameworks, languages, and architectures to support cloud-native development.

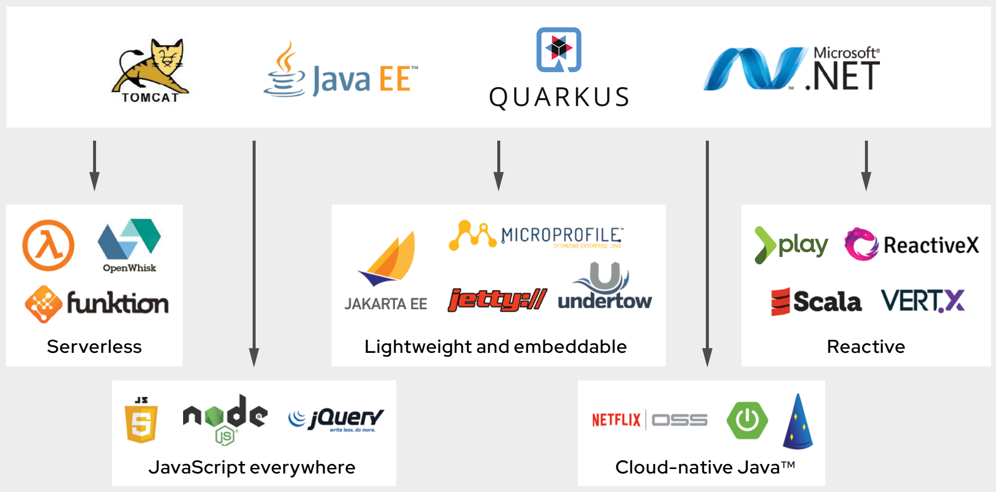

Figure 1. Cloud-native application development becomes even more diverse

Cloud-native development also requires choosing the right tool for the right task. Cloud-native applications can be implemented using a 12-factor approach, domain-based design, test-based design and development, a monolith-first or fast monolith strategy, miniservices, or microservices. Whatever approach you take, your cloud-native platform must offer the right mix of frameworks, languages, and architectures to support the appropriate development requirements. In addition, the underlying container-based platform should support a set of curated runtimes and frameworks that is continuously updated in line with technological changes.

Step 5: Provide self-service, on-demand infrastructure

Agile methods have helped developers create and update software quickly but often lack an efficient mechanism for timely infrastructure access when and where it is required. When releasing applications to production, overall speed to market is affected. Filing a ticket and waiting weeks for IT operations to release resources is no longer a sustainable model in an era when infrastructure is inexpensive and engineering talent is costly.

Self-service and on-demand infrastructure provisioning provides a compelling alternative to unauthorized shadow IT—allowing developers to access the infrastructure when they need it. But this model can only be effective if IT operations teams have control and visibility across what is often a dynamic and complex environment.

Containers and container orchestration technology abstract and simplify access to the underlying infrastructure and provide robust application life cycle management across various infrastructure environments, such as datacenters, private clouds, and public clouds. A container platform offers additional self-service, automation, and application life cycle management capabilities. This model lets developers and operations teams spin up consistent environments quickly, allowing developers to focus on building applications without the obstacles and delays associated with provisioning infrastructure.

Red Hat
Application Services

Containers also support application portability, including the creation of a cloud-native application that can be deployed and run on any cloud provider. Portability offers the freedom to select any cloud provider at any point in time, easily migrate from one cloud provider to another, optimize related costs, and develop a multicloud application without coding to a specific cloud provider API.

Step 6: Automate IT to accelerate application delivery

IT or infrastructure automation is essential to accelerating the delivery of cloud-native applications by eliminating manual IT tasks. Automation can integrate with and apply to any task or component, from network and infrastructure provisioning to application deployment and configuration management.

IT management and automation tools create repeatable processes, rules, and frameworks that can replace or reduce labor-intensive human interaction that delays time to market. They can extend further into specific technologies, like containers, or methods, like DevOps, into broader areas, such as cloud computing, security, testing, monitoring, and alerting. As a result, automation is key to IT optimization and digital transformation, speeding overall time to value.

Learn more about the important role of IT automation in "The automated enterprise."

Download the e-book

Guide for IT automation

1. Adopt an enterprise-wide, programmatic automation approach to IT operations. Embrace collaborative dialog across the organization to design service requirements.
2. Consider automation sandboxes as the foundation for learning the automation language and processes.
3. Think hard about automation. Make sure every unnecessary manual step is eliminated, even if it is tempting to retain manual controls for peace of mind.
4. Consider adopting automation incrementally in small, achievable steps using systematic methods. Each step builds on the previous one to create a widespread automation practice.
5. Start by automating one task or service—whether compute, network, storage, or provisioning. Share that automation with others and build upon it systematically.
6. Implement self-service catalogs that equip users with what they need and speed delivery.
7. Implement metering, monitoring, and chargeback policies and processes.

Over time, integrated, full-scale automation will not only become a reality but will also yield higher efficiency, faster DevOps pipelines, and rapid innovation.

Step 7: Implement continuous delivery and advanced deployment techniques

Continuous delivery (CD) is a software development practice that uses automation to speed the release of new code. It establishes a process through which a developer's changes to an application can be pushed to a code repository or container registry through automation.

Long release cycles mean longer delays between the discovery and resolution of software bugs, as well as an inherent barrier to timely responses to changes in customer and market demand. For high-traffic applications—such as mobile, web, Internet of Things (IoT), or edge computing applications—an unresolved bug can affect many users, resulting in poor customer experiences, security or safety issues, and reduced productivity or revenue. Even for other internal business applications, outages or delays in addressing software bugs can have high business costs.

Agile development methods evolved to create a model of release early, release often. DevOps and continuous delivery approaches extend these methods by closely uniting developers, operations, quality assurance, and security teams to improve software delivery processes. As a result, code changes can be pushed to production quickly and reliably to provide fast feedback to developers. This iterative, fast feedback loop is enabled through CI/CD, extending infrastructure automation to an end-to-end, automated delivery system that covers all aspects of application delivery, including automated testing, vulnerability scanning, security compliance, and regulation checks. The goal of automated delivery pipelines is to provide updates without affecting operational capacity, reducing delivery risks.

"Advanced deployment techniques bring structure and clarity to innovation. Mature deployment methodologies create an environment that allows true experimentation, feedback, and analysis. Better experimentation leads to better innovation."

Burr Sutter
Director of Developer Experience, Red Hat

The first step in achieving continuous delivery (CD) is to enable continuous integration (CI). CI systems are build systems that watch various source control repositories for changes, run any applicable tests, and automatically build the latest version of the application from each source control change.

Advanced deployment patterns aim to reduce the risk of software releases and build an environment for experimentation with controlled outcomes without unintended negative consequences for customers. This goal is essential for increasing innovation across an organization.

Advanced deployment techniques change the nature of delivery from an off-hour weekend activity, with service windows and downtimes, to a routine workday activity with zero downtime in production, while the application is still available to the customers.

By removing the inconvenience of new deployment for the customers, these techniques let organizations deliver updates and releases at the frequency that business demands. The following are some of the common deployment techniques that can be used to achieve zero-downtime deployment, depending on the application use cases:

Rolling deployment is a pattern in which—instead of updating all instances of an application at once—each instance is updated individually by excluding it from the load balancer so that it does not receive traffic. It is updated and then included again in the load balancer. This process continues until all instances are updated.

Blue-green deployment describes the practice of running two identical environments, one active and the other idle. Changes are rolled out to the idle environment, then, once the change is verified in production, the live traffic is switched to the updated environment. Rolling back to the previous version is as simple as switching the traffic back, provided that the data transition is also taken into consideration.

Canary deployment is similar to blue-green deployment in that it uses two identical environments. However, it differs in the way rollout is controlled. After deploying a new release, a small subset of customers is sent to the new release to test it in production. If the new release verification succeeds, traffic is incrementally shifted to the new version while the outcomes are monitored and verified until all users are sent to the new release.

Step 8: Evolve a more modular architecture

In a microservices-based architecture approach to writing software, applications are broken down into their smallest components, independent from each other. Instead of a traditional, monolithic approach where everything is built into a single piece, microservices are separated components that work together to accomplish the same tasks. This approach to software development values granularity, being lightweight, and sharing similar processes across multiple apps. Although a microservices architecture does not impose a specific underlying infrastructure, a container-based platform is an optimal foundation.

Evolving a microservices-based architecture might provide an extra benefit for very large teams or organizations that conduct production deployments multiple times a day. From an architectural standpoint, using microservices requires breaking out each service into its own deployment unit. Each microservice is then managed and deployed independently, potentially with different teams responsible for their life cycles.

Another alternative to microservices is miniservices. A miniservice is a collection of services that are split by domain and usually run on an application server. Miniservices improve agility and scale without the complexity of microservices-based design and infrastructure. Miniservices still require an investment in agile, DevOps, and CI/CD approaches, making a modern application server or an offering with multiple frameworks, architectures, and languages in combination with a container-based infrastructure ideal.

A platform that supports different frameworks, languages, and approaches to cloud-native application development (e.g., microservices, miniservices, or monolith-first) is key to success with cloud-native applications.

How Red Hat can help

Whatever stage you are at within your cloud-native journey and your priorities, Red Hat has the technologies and services to support you.

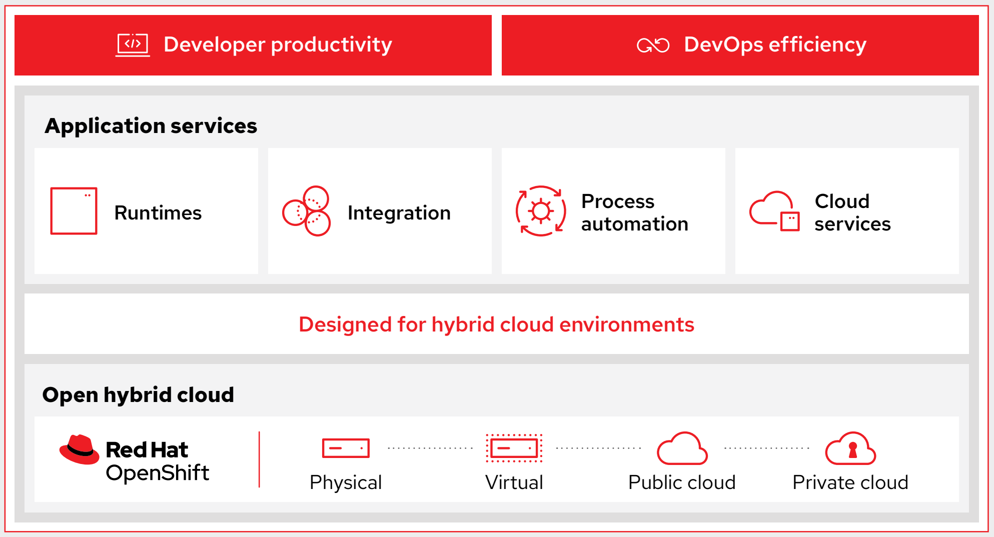

Figure 2. The cloud-native journey with Red Hat

Some organizations may be focused on only one cloud-native use case, while others may be prioritizing a few use cases simultaneously. Whether you take an evolutionary or a revolutionary approach, your path is highly individual and not necessarily linear. Whatever your path, getting applications to market faster requires the right technology, DevOps practices, and culture.

Red Hat helps support this journey with Red Hat® OpenShift®, a cloud-native container development and orchestration platform. Red Hat Application Services is a comprehensive portfolio for application development that runs on Red Hat OpenShift. The Red Hat Application Services portfolio includes frameworks, tools, and solutions for developing, deploying, and scaling cloud-native applications. If your organization needs to get to market faster, Red Hat OpenShift application services (part of Red Hat Cloud Services) are hosted and managed cloud services for Red Hat OpenShift that deliver a streamlined developer experience for developing, deploying, and scaling cloud-native applications.

To help navigate the complexity of cloud-native development, Red Hat Consulting offers strategic advice as well as in-depth technical expertise. From Red Hat Open Innovation Labs to discovery sessions and project implementation plans, our consultants can help you on every step of your cloud-native journey.