THE ROLE OF MICROSERVICES IN DEVOPS

Technology is evolving faster than ever. People depend heavily on the internet for all kinds of regular tasks, from shopping to banking and healthcare. That's made it critical for service providers to fulfill this ever-increasing consumer demand—which itself is coupled with:

- Evolving user requirements
- The increasingly challenging security landscape

This forces service providers to abandon monolith software development methods and adopt Agile and DevOps approaches that help them quickly adapt to changing requirements.

Another trend is microservices-based architectures, where applications are built as multiple loosely coupled services. In this article, we will discuss the role of microservices in the DevOps process.

(This article is part of our DevOps Guide. Use the right-hand menu to navigate.)

DevOps basics

DevOps is a paradigm shift in the way organizations approach software development, deployment, and maintenance. DevOps shifts the whole software development lifecycle (SDLC) to a more collaborative process. Unlike in traditional SDLC, where Dev, QA, and Ops teams are considered separate individual entities, a DevOps team consists of cross-disciplined team members who bring their unique talents to the DevOps process.

This makes the DevOps team much more flexible and helps bridge the communication gap between larger teams such as Dev and QA. This ultimately leads to a faster, agile, and more efficient development environment with predictable deployments and update cycles.

Additionally, DevOps enables creating SaaS business models through which organizations can gain continuous revenue while offering an ever-improving service with stable constant updates.

DevOps offers a multitude of benefits for an organization. However, to gain those benefits you must properly implement DevOps with standardized concepts and proper tools such as:

- Continuous integration
- Continuous delivery platforms
- Automated testing

(Learn how to set up a CI/CD pipeline.)

Integrating DevOps to software development

The core components related to development and management will remain the same. But the technology stack you'll use in an application will change depending on your:

- Audience
- Platform
- Internal and external requirements

In this section, we will cover the core areas that need to be considered when creating a DevOps pipeline.

Version control

Version controlling is the cornerstone of any DevOps process. Technologies like Git and code repositories like Bitbucket, Azure Repos, and GitLab make it easier than ever to manage the source code. These tools allow developers to work without conflicting with each other's work while providing version-controlled changes to the application.

Even database code is now managed with version-controlled repositories as a part of the DevOps process. We can sum up version controlling as something that:

- Reduces code conflicts
- Increases the visibility and efficiency of development
- Enables easy rollbacks to previous versions in case of any issues

Automation

The second major consideration is automation, as DevOps rely on automated tasks for faster development. The DevOps team must try to automate any repetitive manual tasks and be constantly on the lookout for new tasks that can be automated.

For example, automated tools like Selenium, Appium, and Cucumber can significantly benefit the application testing stage as they allow users to automate most test cases. Furthermore, automation:

- Reduces human errors
- Saves time spent on mundane tasks
- Frees up team members to focus on more important matters

Deployment strategy

The next consideration is the deployment strategy. Most DevOps deployment stages are targeted at highly available and scalable cloud infrastructure with the popularity of cloud-based applications and all the advantages of cloud-based deployments.

Containerized applications are one of the factors that power this shift to cloud-based deployments. Containers allow users to create isolated and portable application environments that can be deployed anywhere with all the required dependencies. Platforms like Kubernetes and Rancher provide robust orchestration capabilities for containerized deployments. Here, automation also plays a major role by automating software packaging and deployments.

(Read our containerization introduction.)

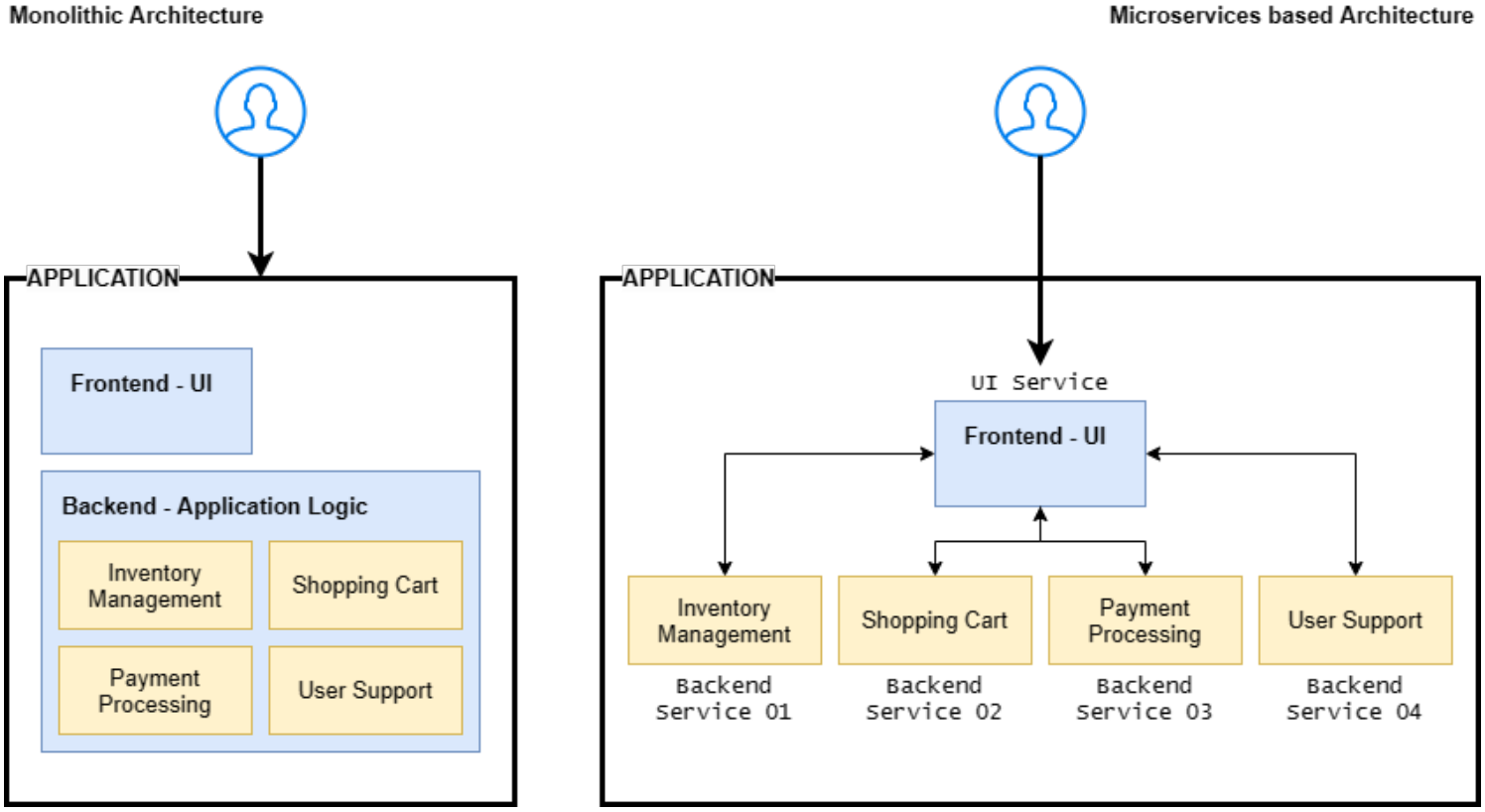
A version-controlled and automated DevOps pipeline with a proper deployment strategy allows organizations to create a pipeline that encompasses all the stages of SDLC. Now that we understand the primary considerations of a DevOps pipeline, let's see how microservices affect all these factors.

Microservices overview

Microservices is an architectural approach to development that contrast with traditional, monolithic applications (where the entire application is considered and developed as a single entity). The microservice architecture breaks the application into different loosely coupled services.

For instance, let's consider an online shopping platform:

- In the traditional development process, the complete platform from inventory management, payments, and shopping cart is developed together as components of the application. For any change, development, and deployment happen as a single entity.
- On the other hand, microservices architecture will break the application into distinct services (inventory management service, payment service, shopping cart service), which can be developed and deployed independently. This approach isolates any issues of that specific service and thereby reduces their impact on the application.



The DevOps approach is ideal for Microservices-based applications as it allows for easy development, provides seamless updates, and manages each service without the risk of complete application failures.

Role of microservices in DevOps

In order to understand the role of Microservices in DevOps, let's consider a DevOps pipeline for developing a microservices-based application.

The first thing that microservices changes is how organizations approach development. As everything is broken down into separate services, development teams can also be divided to tackle each service. This will, in turn, reduce the scope of the development while making the development process more flexible.

Overall flexibility

This flexibility allows DevOps teams to address issues effectively. For instance, when a production bug is discovered, the development team can fix that bug in the affected service and deploy the service without affecting the SDLC of other services and causing minimal downtime. Moreover, if any service requires a new feature, developers can develop and deploy it in production without affecting the development process of other services. This decoupled approach simplifies the development and testing process while allowing services to be modified independently.

Containers

Containerization is another factor that extends and complements microservices-based architectures. Packaging each service as a container image further reduces the complexity while streamlining the continuous delivery pipeline.

Services can act as fully independent entities with all the dependencies and requirements bundled within the container. This makes the services system-agnostic and reusable while allowing them to interact with any other system.

(Learn more about containers & microservices.)

APIs

APIs are another factor that goes hand in hand with microservices. With decoupled services, users need a robust communication method to communicate between services.

APIs enable developers to expose only the relevant endpoints and information while hardening the service and providing a universally compatible interface. This also becomes a concern when creating reusable system-agnostic services.

(See how microservices rely on APIs.)

Automation

Next comes automation. In a microservices architecture, most testing, packaging, and deployment tasks can be automated for each service. As each service resides in an independent DevOps pipeline, any issues in a single automated task do not affect the other services. Plus, feedback loops become much shorter when addressing bugs with the simplified codebase.

The stages at which microservices shine most with automation are deployment and maintenance. An automated task will be triggered in the deployment process to deploy the service as a container once all the testing is done and the container image has been uploaded to a container registry. This simplifies the deployment by eliminating the need for specialized network configurations or dependency management prior to deployments.

Increased availability, scalability

Finally, the availability and scalability of the application automatically increase with service containers deployed in clusters and managed via an orchestration platform.

- If there is a failure in service, it can be quickly replaced with a brand new service container.

- If there is a high load for a service, a new container can be created to facilitate the surge in demand.

This allows scale-out strategies without relying on scale-up strategies based on resource increases which are cost-prohibitive.

Microservices are perfect for DevOps

Microservices architecture is tailor-made for DevOps with its services-based approach that allows organizations to break down the application into smaller services. This enables delivery teams to tackle individual services as separate entities—ultimately simplifying the development, testing, and deployment. (This doesn't mean microservices should be used for every application, however. They do come with certain challenges.)

The role microservices plays in DevOps includes streamlining the DevOps process and increasing productivity and quality of the application while moving developments to a flexible architecture. This leads to the development of cloud-native applications that are capable of fulfilling any user demand.

Related reading

- BMC DevOps Blog
- DevOps Team Structure
- Service-Oriented Architecture vs Microservices Architecture: Comparing SOA to MSA
- Software Project Management Phases & Best Practices
- Implementing GitOps To Deliver Cloud Native Applications
- Container Sprawl: What It Is & How To Avoid It