This content is part of the Essential Guide:

Developers' guide to deploying microservices and containers

DEFINITION

microservices

Posted by: Margaret Rouse WhatIs.com

Contributor(s): Alexander Gillis and Brein Matturro

Microservices, or microservice architecture, is an approach to application development in which a large application is built as a suite of modular components or services.

Each module supports a specific task or business goal and uses a simple, well-defined interface, such as an application programming interface (API), to communicate with other sets of services. Software developer and author Martin Fowler is credited with promoting the idea of breaking down services in a service-oriented architecture (SOA) into microservices. In a 2014 article, he stated: "The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery."

How microservices work

In microservice architecture, an application is divided into services. Each service runs a unique process and usually manages its own database. A service can generate alerts, log data, support user interfaces (UIs), handle user identification or authentication, and perform various other tasks.

The microservice paradigm provides development teams with a more decentralized approach to building software. Microservices enable each service to be isolated, rebuilt, redeployed and managed independently. For example, if a program isn't properly generating reports, it can be easier to trace the problem to that specific service. That specific service could then be tested, restarted, patched and redeployed as needed, independent of other services.

Microservices vs. monolithic architecture

In a monolithic architecture, all of the code is in one principal executable file, which can be tougher to troubleshoot, test and update. If there is a problem in a code base, that problem could be located anywhere within the software. There would be more testing, and the tests would take longer due to the amount of monolithic code involved. In a monolithic application, any small change or update requires building and deploying an entirely new version of the application. This means any monolithic application development entails significant planning, preparation, time and expense.

Also, monolithic applications are more difficult to scale. When a monolithic application reaches a limitation of its capacity, such as data throughput, or some other bottleneck, the only practical alternative is to deploy another complete iteration of the entire monolithic application -- managing traffic between the instances using load balancers. By comparison, it is possible to scale only the services of a microservice application by adding container instances of only those services. This makes microservice scaling far more resource-efficient than scaling applications using a monolithic architecture.

Microservices make it easier to test and deploy changes. Because each service is separated from the others, fault isolation is improved. If there is a problem in the software, the problematic service can be isolated, remediated, tested and redeployed without the need to regression test the entire application as with traditional monolithic application architectures. Microservice architectures enhance business agility with faster software development and deployment compared to monolithic software architecture.

Microservices aren't management-free, however. With the same amount of services as a monolithic software product, the management required in a microservice architecture could be more complex since each service is separated from one another. This can lead to difficulties managing all the parts of a whole. For example, careful monitoring and management are needed to track the availability and performance of all the component services operating within a microservice application.

Microservice pros and cons

Microservices pose a series of tradeoffs for software developers. In terms of advantages, microservices:

- are easily deployed;
- require less development time;
- can scale quickly;
- can be reused in different projects;
- contain better fault isolation;
- can be deployed in relatively small teams; and
- work well with containers.

However, there are also drawbacks with microservices, such as:

- potentially too much granularity;
- extra effort designing for communication between services;
- latency during heavy use; and
- complex testing.

Microservices and DevOps

DevOps combines tasks between application and system operations teams. With increased communications between developers and operations staff, an IT team is able to better create and manage infrastructure. IT operations ensure budgeting for capacities, operational tasks, upgrades and more. Developers and application teams can manage databases, servers, software and hardware used in production.

Teamwork and collaboration between development and operations teams are needed in order to support the lifecycle of microservices, lending itself to DevOps teams. It's also why experienced DevOps teams are well-equipped for employing microservice-type architectures in software development projects.

Microservice architecture vs. SOA

SOA is a software architecture where each of its services utilizes protocols. This enables users to combine functionalities and form applications built from previous services. SOA has been the standard development practice for nearly two decades. However, the resourcefulness of SOA comes into question when working with cloud computing. With the cloud, SOA lacks scalability and slows down with work request changes, limiting application development.

Many developers find microservices to be a more granular approach to SOA. Proponents of the SOA model believe that the microservice architecture is the natural evolution of SOA needed in order to accommodate cloud computing and meet increasing demands for faster software development cycles.

Others believe microservices are a more platform-agnostic approach to application development and, therefore, should have a unique name. This group could argue SOA lives on in the layers of microservice management.

Microservices and containers

A container is an individual and executable package of software, including all of the dependencies that are needed to function independently. Containers are separated from the rest of the software surrounding them, and many containers can be employed in the same environment. In a microservice architecture, each service is individually containerized under the same environment, such as the same or related servers.

A virtual machine (VM) can be used as an alternative to containers to create microservices. A VM simulates computer systems to produce functionalities of a physical computer. Each service could potentially utilize a VM to host an intended feature. However, VMs are often avoided for microservices because of the individual operating system (OS) and other overhead needed for each VM. Containers are much more resource-efficient because only the underlying code and related dependencies are required to operate the service.

Microservice security

Microservice architecture can alleviate some security issues that arise with monolithic applications. Microservices simplify security monitoring because the various parts of an application are isolated. A security breach could happen in one section without affecting other areas of the project. Microservices provide resistance against distributed denial-of-service (DDoS) attacks when used with containers by minimizing an infrastructure takeover with too many server requests.

However, there are still challenges when securing microservices applications, including:

- More network areas are open to vulnerabilities.
- Less overall consistency between app updates allows for more security breaches.
- There's a greater area of attack, through multiple ports and APIs.
- There's a lack of control of third-party software.
- Security needs to be maintained for each service.

Microservice developers have come up with strategies to alleviate security issues. To be proactive, use a security scanner, utilize access control limitations, secure internal networks -- including Docker environments -- and operate outside of silos, thus communicating with all parts of the operation.

Deploying microservice applications

Three main developments have occurred to make microservices a viable software architecture.

The first breakthrough, containers, enables a consistent and resource-efficient means of packaging individual services. Docker is a popular tool for developers, which allows for containers to be used on premises or the public or private cloud. This offers a wide variety of deployment alternatives for microservice developers and businesses.

Margaret Rouse asks:

How has your enterprise successfully moved from mo to microservices?

Join the Discussion

The second important development is the emergence of orchestration tools, such as Kubernetes. These tools help with automating scaling, deployment and container management.

A third major breakthrough for microservices is the evolution of service mesh. A service mesh is a layer of infrastructure dedicated for communication between individual services. Service meshes make these communications faster, more secure, visible and reliable. When hundreds of services are communicating with each other, it becomes complicated to tell what services are interacting with each other. Linkerd is an orchestration tool that accomplishes secure, fast, visible communications between services by capturing behaviors, such as latency-aware load balancing or service discovery.

The levels of complexity in microservices are steadily dwindling due to these breakthroughs, such as the intricacies of monitoring and logging, as more organizations and development teams adopt microservices.

This was last updated in March 2018