What are microservices?

The central idea behind microservices is that some types of applications become easier to build and maintain when they are broken down into smaller, composable pieces which work together. Each component is continuously developed and separately maintained, and the application is then simply the sum of its constituent components. This is in contrast to a traditional, "monolithic" application which is all developed all in one piece.

Applications built as a set of modular components are easier to understand, easier to test, and most importantly easier to maintain over the life of the application. It enables organizations to achieve much higher agility and be able to vastly improve the time it takes to get working improvements to production. This approach has proven to be superior, especially for large enterprise applications which are developed by teams of geographically and culturally diverse developers.

There are other benefits:

- **Developer independence**: Small teams work in parallel and can iterate faster than large teams.
- **Isolation and resilience**: If a component dies, you spin up another while and the rest of the application continues to function.
- **Scalability**: Smaller components take up fewer resources and can be scaled to meet increasing demand of that component only.
- **Lifecycle automation**: Individual components are easier to fit into continuous delivery pipelines and complex deployment scenarios not possible with monoliths.
- **Relationship to the business**: Microservice architectures are split along business domain boundaries, increasing independence and understanding across the organization.

The common definition of microservices generally relies upon each microservice providing an API endpoint, often but not always a stateless REST API which can be accessed over HTTP(S) just like a standard web page. This method for accessing microservices make them easy for developers to consume as they only require tools and methods many developers are already familiar with.

Is this a new concept?

The idea of separating applications into smaller parts is nothing new; there are other programming paradigms which address this same concept, such as Service Oriented Architecture (SOA). However, recent technology advances coupled with an increasing expectation of integrated "digital experiences" have given rise to a new breed of development tools and techniques used to meet the needs of modern business applications.

More on microservices

- How to explain microservices to your CEO (https://blog.openshift.com/microservices-how-to-explain-them-to-your-ceo/?intcmp=7016000000127cYAAQ)
- Free eBook: Microservices vs. service-oriented architecture (https://www.openshift.com/promotions/microservices.html?intcmp=7016000000127cYAAQ)
- Secured DevOps for microservices (https://opensource.com/business/16/11/secured-devops-microservices?intcmp=7016000000127cYAAQ)

Microservices depend not just on the technology being set up to support this concept, but on an organization having the culture, know-how, and structures in place for development teams to be able to adopt this model. Microservices are a part of a larger shift in IT departments towards a DevOps culture, in which development and operations teams work closely together to support an application over its lifecycle, and go through a rapid or even continuous release cycle rather than a more traditional long cycle.

Why is open source important for microservices?

When you design your applications from the ground up to be modular and composable, it allows you to use drop-in components in many places where in the past you may have required proprietary solutions, either because the licensing of the components, or specialized requirements. Many application components can be off-the-shelf open source tools, and there are myriad open source projects that implement cross-cutting requirements of microservice architectures such as authentication, service discovery, logging and monitoring, load balancing, scaling, and several more.

A focus on microservices may also make it easier for application developers to offer alternative interfaces to your applications. When everything is an API, communications between application components become standardized. All a component has to do to make use of your application and data is to be able to authenticate and communicate across those standard APIs. This allows both those inside and, when appropriate, outside your organization to easily develop new ways to utilize your application's data and services.

Where do container technologies come in?

The modern concept of lightweight OS containers was introduced in the early 2000s as part of the FreeBSD (https://www.freebsd.org/) project. Docker (https://opensource.com/resources/what-docker) provided an improved user experience for creating and sharing container images and as a result saw great adoption starting in 2013. Containers are a natural fit for microservices, matching the desire for lightweight and nimble components that can be easily managed and dynamically replaced. Unlike virtual machines, containers are designed to be pared down to the minimal viable pieces needed to run whatever the one thing the container is designed to do, rather than packing multiple functions into the same virtual or physical machine. The ease of development that Docker and similar tools provide help make possible rapid development and testing of services.

Of course, containers are just a tool, and microservice architecture is just a concept. It is entirely possible to build an application which could be described as following a microservices approach without using containers, just as it would be possible to build a much more traditional application inside of a container, which may make sense when you want to take advantage of container orchestration capabilities without re-writing a large, monolithic app.

How do you orchestrate microservices?

In order to actually run an application based on microservices, you need to be able monitor, manage, and scale the different constituent parts. There are a number of different tools that might allow you to accomplish this. For containers, open source tools like Kubernetes (http://kubernetes.io/) will probably be a part of your solution. Alternatively, for non-container pieces of an application, other tools may be used for orchestrating components: for example, in an OpenStack (https://opensource.com/resources/what-is-openstack) cloud you might use Heat for managing application components.

Another option is to use a Platform as a Service (PaaS) tool, which lets developers focus on writing code by abstracting some of the underlying orchestration technology and allowing them to easily select off-the-shelf open source components for certain parts of an application, like a database storage engine, a logging service, a continuous integration server, web server, or other pieces of the puzzle. Some PaaS systems like [OpenShift (https://www.openshift.org/)](https://www.openshift.org/) directly use upstream projects like Docker and Kubernetes for managing application components, while others try to re-implement management tools themselves.

What about existing applications?

While utilizing microservices may be an important component of an organization's IT strategy going forward, there are certainly many applications which don't meet this model, nor is it likely that those applications will be re-architected overnight to meet this new paradigm. There is a cultural and technical cost to moving to a microservices architecture, but fortunately microservices and traditional applications can work together in the same environments, provided the organization has a solid bi-modal IT strategy.

Bi-modal IT, according to Gartner, is the ability to deliver on both traditional IT applications with a focus on stability and uptime, and newer, more agile but possibly less tested applications through newer methods involving things like the ability of developers to self-provision machines and short development cycles.

Many if not most organizations will need to be adapted to work with both approaches for many years to come.

Where can I learn more?

Opensource.com has a number of resources for those who are interested in learning more about different tools and design patterns that might be a part of a microservices-oriented application. Here are a few we recommend you check out:

- [Containers, microservices, and orchestrating the whole symphony (https://opensource.com/business/14/12/containers-microservices-and-orchestrating-whole-symphony)](https://opensource.com/business/14/12/containers-microservices-and-orchestrating-whole-symphony) by Uri Cohen
- [Google shares gRPC as alternative to REST for microservices (https://opensource.com/bus/15/3/google-grpc-open-source-remote-procedure-calls)](https://opensource.com/bus/15/3/google-grpc-open-source-remote-procedure-calls) by Luis Ibáñez
- [NGINX: The secret heart of the modern web (https://opensource.com/business/15/2/interview-sarah-novotny-nginx)](https://opensource.com/business/15/2/interview-sarah-novotny-nginx) by

Jason Hibbets

- Smart API integrations with Python and Zato (https://opensource.com/business/15/5/api-integrations-with-python-and-zato) by Dariusz Suchojad
- Senior software engineer Petazzoni on the breathtaking growth of Docker (https://opensource.com/business/14/7/interview-jerome-petazzoni-docker) by Richard Morrell

Want to master microservices? Learn how (https://www.redhat.com/en/engage/openshift-storage-testdrive-20170718?intcmp=7016000000127cYAAQ) to run OpenShift Container Platform in a self-paced, hands-on lab environment.