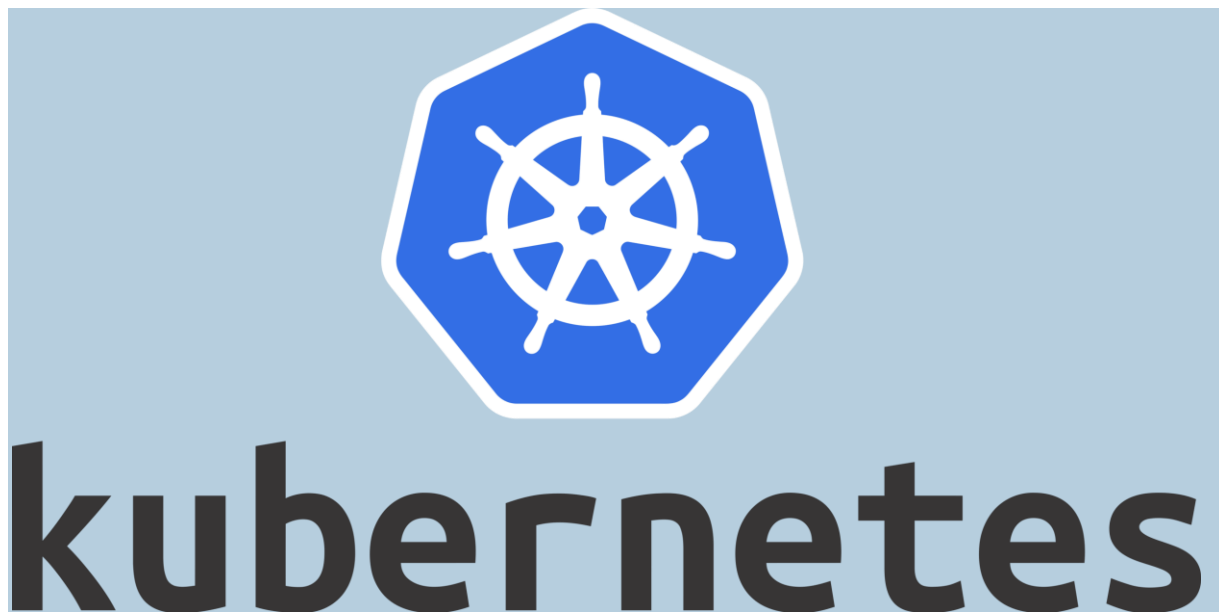


Kubernetes Real life use Cases

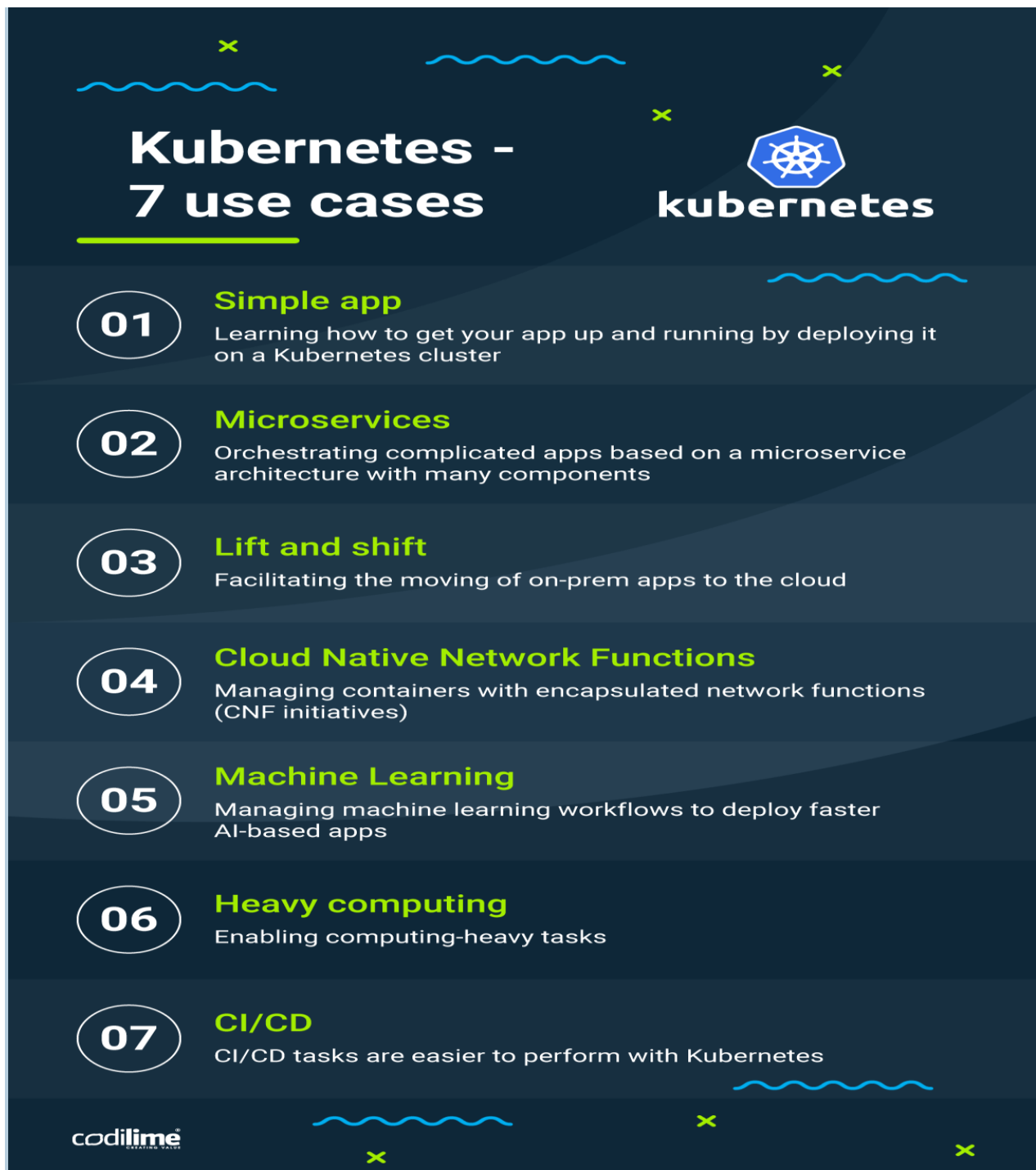


rahul banik

Mar 7 · 7 min read




Real Life Industry Use-cases of Kubernetes:



The infographic features a dark blue background with a subtle pattern of yellow 'x' marks and light blue wavy lines. The title 'Kubernetes - 7 use cases' is prominently displayed in white, with a yellow underline. To the right, the Kubernetes logo (a blue hexagon with a white ship's wheel) is shown above the word 'kubernetes' in white. The seven use cases are listed in a vertical column, each with a white circular number, a yellow title, and a white description. The bottom left corner features the 'codilime' logo in white.

Kubernetes - 7 use cases



kubernetes

- 01 Simple app**
Learning how to get your app up and running by deploying it on a Kubernetes cluster
- 02 Microservices**
Orchestrating complicated apps based on a microservice architecture with many components
- 03 Lift and shift**
Facilitating the moving of on-prem apps to the cloud
- 04 Cloud Native Network Functions**
Managing containers with encapsulated network functions (CNF initiatives)
- 05 Machine Learning**
Managing machine learning workflows to deploy faster AI-based apps
- 06 Heavy computing**
Enabling computing-heavy tasks
- 07 CI/CD**
CI/CD tasks are easier to perform with Kubernetes

codilime

- Learning **Kubernetes** by deploying a simple app. ...
- Microservices architecture. ...
- Lift and shift — from servers to cloud. ...
- Cloud-native Network Functions (CNF) ...

- Machine learning and **Kubernetes**. ...
- Computing power for resource-hungry tasks. ...
- CI/CD — software development lifecycle.

The Docker adoption is still growing exponentially as more and more companies have started using it in production. It is important to use an orchestration platform to scale and manage your containers. Imagine a situation where you have been using Docker for a little while, and have deployed on a few different servers. Your application starts getting massive traffic, and you need to scale up fast; how will you go from 3 servers to 40 servers that you may require? And how will you decide which container should go where? How would you monitor all these containers and make sure they are restarted if they die? This is where [Kubernetes](#) comes in.

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

K8s can be deployed in very different scenarios depending on the size of the company and its objectives:

- **In-house:** Organizations can transform their own data center into a K8s cluster. In this case, companies can take full advantage of their own resources.

- **Cloud:** The setup process is similar to an in-house deployment, but includes virtual machines on the cloud. This allows for the creation of a virtually infinite number of machines, depending on demand.
- **Hybrid:** An organization's data center might perform well for most of the day, but sometimes a peak occurs that local computing resources cannot handle. In this case, a hybrid solution works well. When necessary, K8s will create virtual machines on the cloud to better distribute computing resources when on-premise servers are full.
- **On-premise:** Some cloud providers have their own K8s implementation embedded. In this case, there is no need to deploy and configure Kubernetes itself; an organization just needs to manage the service. Since deploying Kubernetes can be tricky, this is a good solution for companies that do not have a big IT team capable of handling cluster configuration and maintenance.
- **Multi-cloud:** This is the next level of a hybrid cloud solution. Computing resources are deployed among two or more cloud vendors. In this case, companies need to avoid vendor lock-in and minimize risk if something goes wrong.

Self-Healing and Scaling Services

For simplicity, K8s process units can be detailed as [pods](#) and [services](#). A pod is the smaller deployment unit available on Kubernetes. A pod can contain several containers that will have some related communication — such as network and storage. Services are the interface that provides accessibility to a set of containers. These services can be for internal or public access and can load balance several container instances

Serverless, with Server

[Serverless architecture](#) has taken the world by a storm since AWS launched [Lambda](#). The principle is simple: just develop the code, and don't worry about anything else. Server and scalability are handled by the cloud provider and code just has to be developed as functions that handle specific events: from HTTP requests to queue messages.

Optimized Resource Usage with Namespaces

A [K8s namespace](#) is also known as a virtual cluster. Namespaces create a virtually separated cluster inside the real cluster. Clusters without namespaces probably have test, staging and production clusters. Virtual clusters usually waste some resources because they do not undergo continuous testing, and because staging is used from time to time to validate the work of a new feature. By using a virtual cluster, or a namespace, an operations team can use the same set of physical machines for different sets depending on a given workload.

Hybrid and Multiclouds

A hybrid cloud utilizes computing resources from a local, conventional data center, and from a cloud provider. A hybrid cloud is normally used when a company has some servers in an on-premise data center and wants to use the cloud's unlimited computing resources to expand or substitute company resources. A multicloud, on the other hand, refers to a cloud that uses multiple cloud providers to handle computing resources. Multiclouds are generally used to avoid vendor lock-in, and to reduce the risk from a cloud provider going down while performing mission-critical operations.

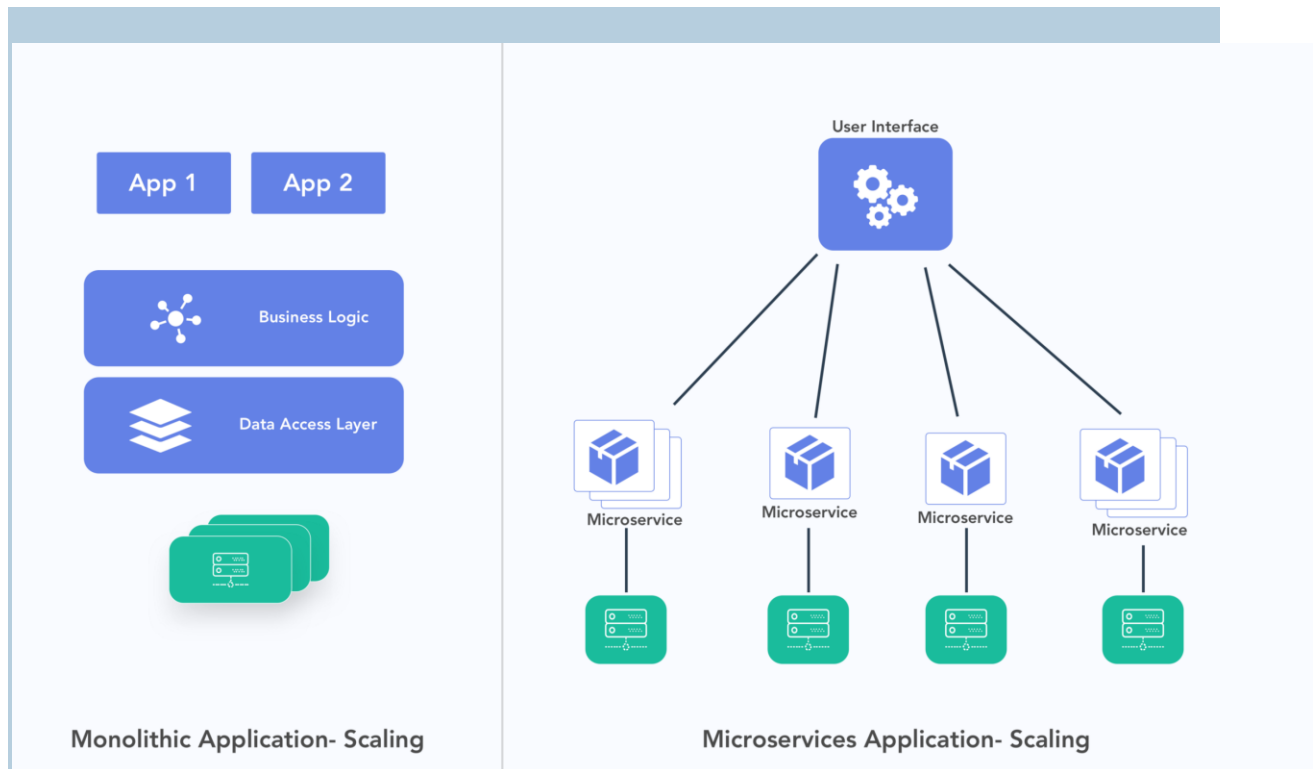
Learning Kubernetes by deploying a simple app



The first case where you can make use of Kubernetes may seem controversial, but still is very useful. Let's assume that we have a simple three-tier application with backend written in Python/PHP, a database and front-end created in React or Angular. To deploy it, you can use Kubernetes. Yes, from a purely practical point of view this would be not very reasonable: Kubernetes is complex and creating a Kubernetes cluster to run one simple app would mean doing unnecessary work. Further,

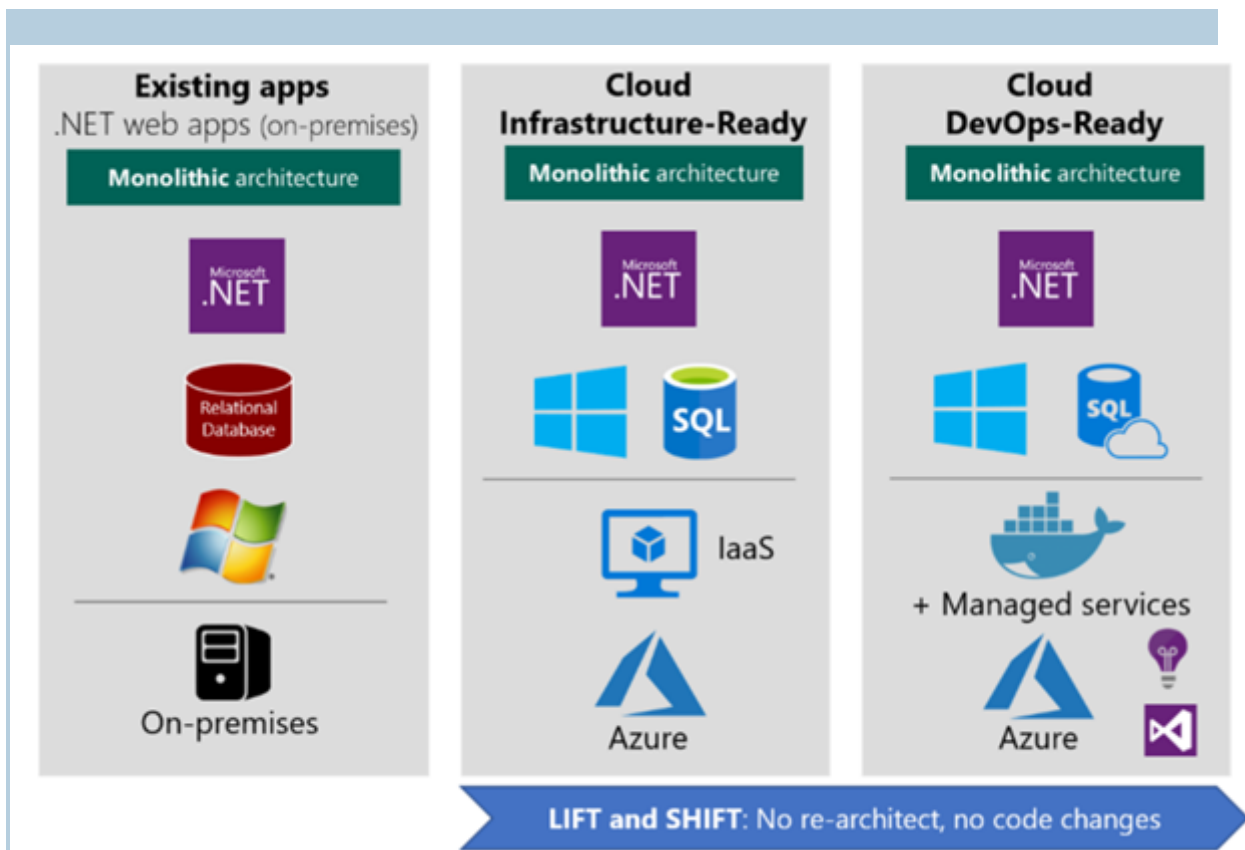
you can deploy such an app using other, less expensive solutions. But there is an educational purpose that shouldn't be overlooked. In undertaking such a deployment, you will learn how to run a Kubernetes cluster and deploy applications on it.

Microservices architecture



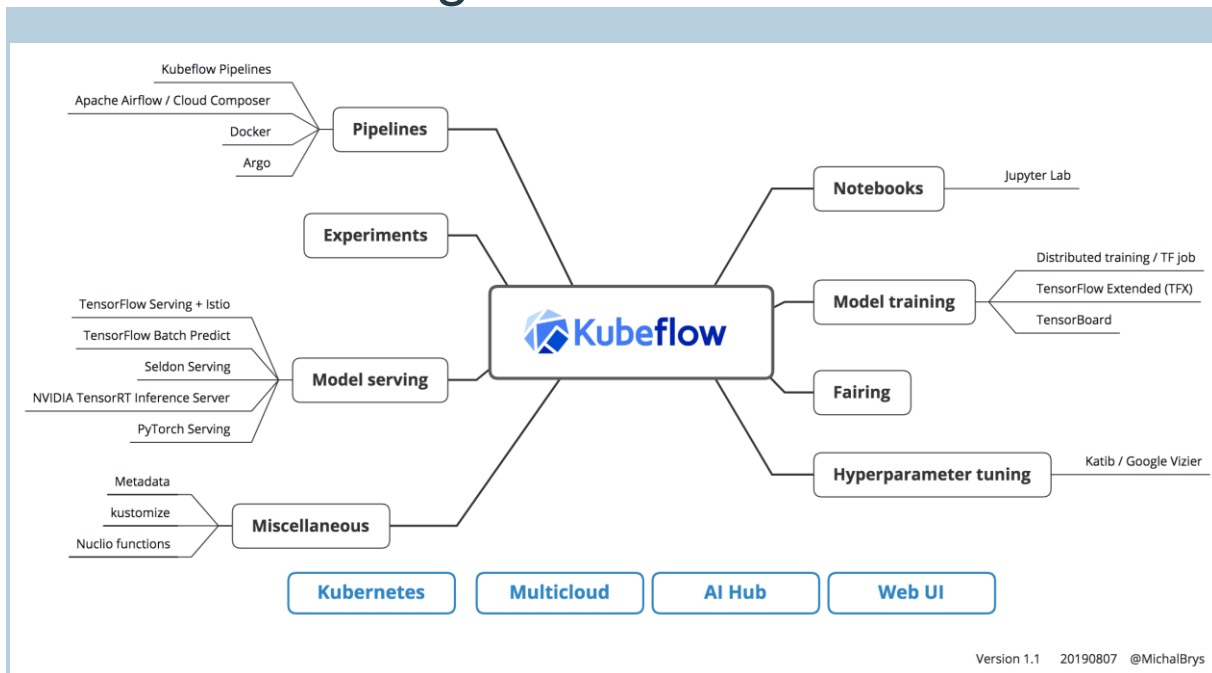
A use case where you want to deploy a more complicated app with many components that will communicate with one another is a classic scenario for Kubernetes. In fact, its origins go back to Google deploying, managing and scaling apps in a more efficient way by using containers. That's how the container orchestration platform Kubernetes was born. So, we now have a K8s cluster with one complicated app deployed. This app has numerous components that communicate with one another. Kubernetes helps you manage this communication.

Lift and shift — from servers to cloud



This scenario occurs frequently today, as software is migrated from on-prem infrastructure to cloud solutions. Let's imagine the following situation. We have an application deployed on physical servers in a classical data center. For practical or economic reasons, it has been decided to move it to the cloud: either to a Virtual Machine or to big pods in Kubernetes. Of course, moving it to big pods in K8s isn't a cloud native approach, but it can be treated as an intermediary phase. First, such a big app working outside the cloud is moved to the same big app in Kubernetes. It is then split into smaller components to become a regular cloud native-app. Such methodology is called "lift and shift" and is a good use case where Kubernetes can be used effectively.

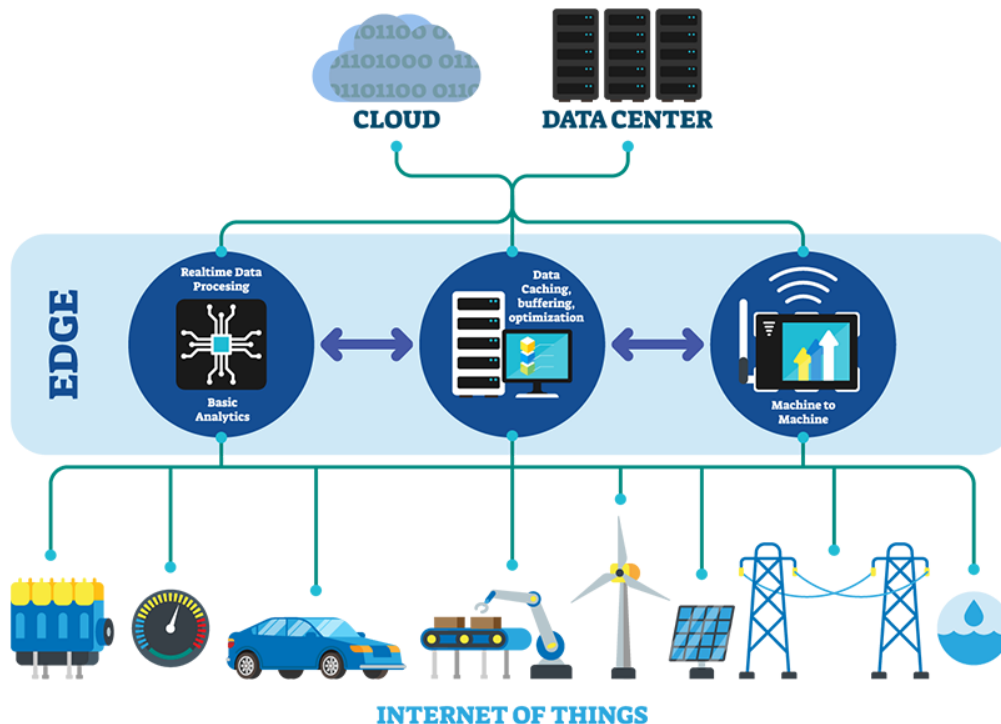
Machine learning and Kubernetes



Machine learning techniques are now widely used to solve real-life problems. Successes have come in multiple fields — self-driving cars, image recognition, machine translation, speech recognition, game playing ([Go](#) or [poker](#)). Machine learning models have beaten even humans in games like Go, which was once thought to be too difficult a game for machines to crack. Moreover, AI could lead to real breakthroughs in detecting cancer and drug discovery. The business world has not failed to get in on the technology, either. Google, Microsoft and Amazon, to name three behemoths, have all put machines to good use, while other companies are investing heavily to boost their AI capabilities

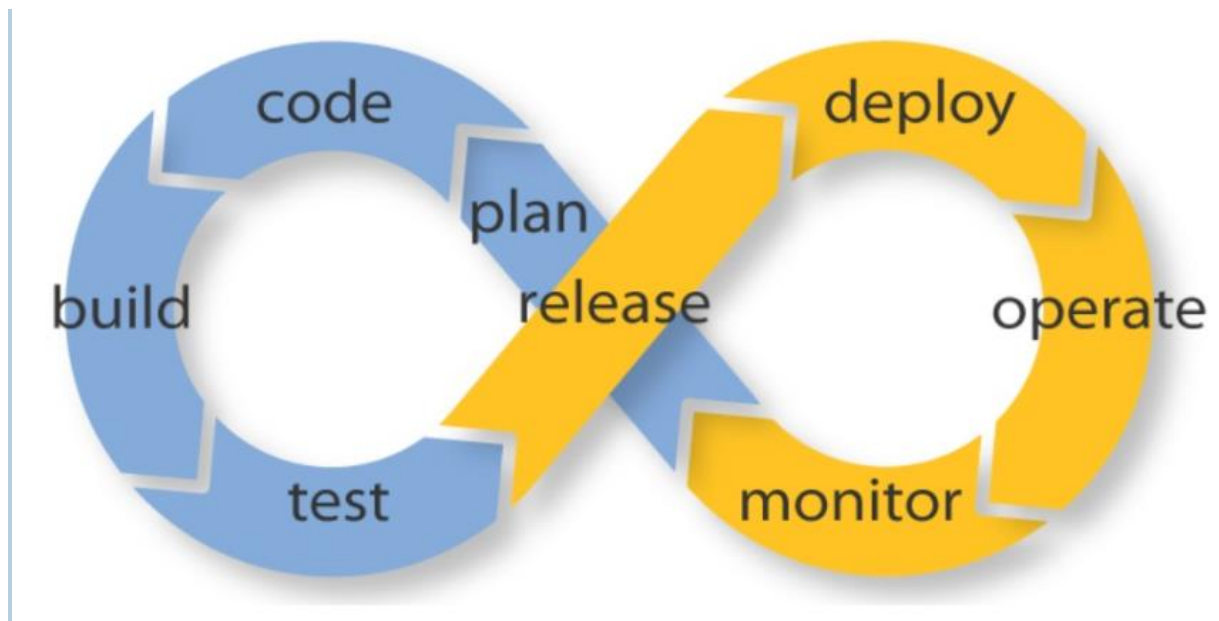
Computing power for resource-hungry tasks

Edge Computing



Recently, [Emma Haruka Iwao](#), a Google employee, broke the Guinness World Record in computing Pi by reaching 31.4 trillion digits. Such calculations require huge computing power, so a Kubernetes cluster would be a natural solution here to manage the distribution of the calculations across multiple computers. Were we to follow in Iwao's footsteps, we would only need to write a program to perform the calculations. Kubernetes would handle the rest. Another computation-heavy case that could make use of the power of K8s is drug discovery.

CI/CD — software development lifecycle



Kubernetes also brings considerable benefits to Continuous Integration/Continuous Deployment or Continuous Delivery methodology (you can read [more about CI/CD in our blog post](#)). This is a logical continuation of the use cases presented in points 1 and 2. Once an app is deployed into operations, how it works must be monitored constantly. That's in addition to gathering users' feedback and developing new features. Whether it's for testing, frequent releases or deploying newer versions of an app, Kubernetes makes everything simpler and more manageable.