turbaszek.medium.com

Application Scalability, Part 1: What the Heck? - Tomasz Urbaszek - Medium

Tomasz Urbaszek

5-6 minutes

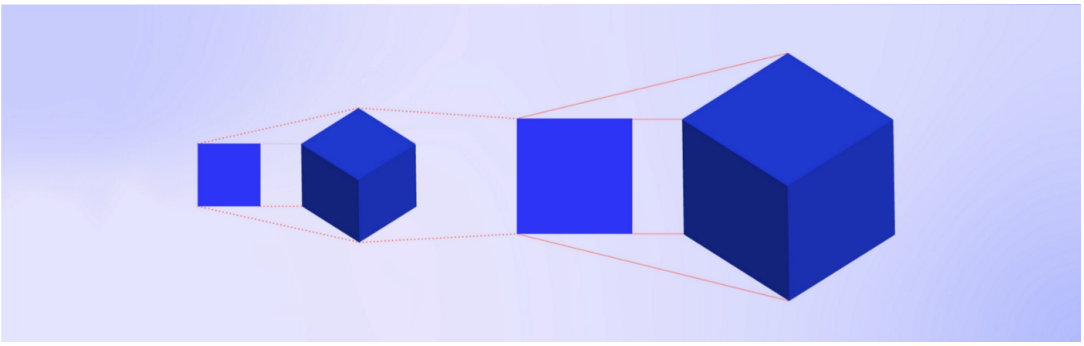

Illustration by Alejandra Ramos

Today, most tech companies process more data per second than ever. Such amount of data is not only a result of the number of served users, but it's also related to the tendency to "measure" everything so we can make better decisions in the future (for example, recommendations). **To make such data-intensive systems successful and reliable, we have to make them scalable.** In this short series of articles, I want to discuss what application scalability means and how it can be achieved on Kubernetes clusters.

Scalability

Probably everyone grasps the idea of scalability. It simply means that we want to make something bigger or smaller, depending on current conditions. In the world of applications, those conditions can

be dictated by data volume or I/O operations number that has to be processed at a single moment. **We want to scale the infrastructure on which we are running our application that is supposed to handle all this load.** Now, when we know what scalability means in general, we can dive into its types.

Vertical scaling

This is a straightforward approach with which all computer game players are familiar (or at least it was common some years ago). If your computer is too lousy to run the new super-demanding game you want to play, all you need to do is add more CPU, RAM, or probably GPU. And that is what we call **vertical scaling.**

Nowadays, when most of our applications run on cloud infrastructure, we can join multiple CPUs and memory volumes under one operating system to simulate a single powerful machine. And this may make the vertical scaling look like the best answer to scalability. Just add more computing power, and everything will be good. **The problem is that more computing power (i.e., more CPU) means higher cost, and the relationship between those two is nearly never linear.**

Another interesting problem with vertical scaling is that to make this “single machine” really powerful, its resources have to be located as close to each other as possible. **It means that we may be limiting ourselves to a single geographical location (single datacenter).**

Horizontal scaling

Another approach to scalability is **horizontal scaling**, also known as shared-nothing architecture. In this approach, we are replacing the single powerful machine with many smaller ones, usually called nodes. Each node is independent of others and performs its task on its own. **The problem of location-bounded machines is gone as we can add new nodes in locations closer to new requests.**

Horizontal scaling is gaining a lot of popularity (i.e., it is supported natively by Kubernetes). **However, its elasticity has a cost: your application must work in a distributed, multi-node fashion.** So, for example, if you need to keep a global state of the application, it must be persisted in a place that can be accessed by any node (i.e., database or object storage) and which will not be destroyed when nodes will be deleted (during scaling down).

Another advantage of horizontal scaling is that the operation of adding or removing nodes is quite smooth. It should not impact the performance of the whole infrastructure as it may happen during vertical scaling, where we are extending components of still running machines. What is more interesting is that with a horizontal approach, we can even scale to zero, meaning that sometimes there will be no machines running our app. **This is a huge cost cutter because we are going to pay only when there's a need for our application.** That's why it's often recommended to keep nodes resources as small as possible.

Autoscaling

Regardless of the scaling type we choose for our application, one thing is certain: we do not want to do this manually. Luckily, the times where DevOps had to monitor app metrics and add more disks or buy more VMs are gone. **Most cloud solutions provide features of autoscaling, which means that the scaling is done automagically.** To get rid of manual intervention in the scaling process, we need two tightly coupled things:

- **Metric** — that provides information about the current load of our application (i.e., number of requests per second, CPU usage);
- **Scaler** — a mechanism that digests the metric, make a decision, and scale infrastructure running our application up or down (either vertically or horizontally).

Usually, as users, we have to decide what metric we want to use and what part of our app should be scaled. The second, scaler part, is quite often provided by the infrastructure we are using to serve our app (i.e., Kubernetes or data clusters, cloud lambdas).

Summary

Now, we know what scalability means and how it can be achieved in the cloud world. We are also aware that we don't have to monitor our application by ourselves, but we can leverage the autoscaling mechanisms. **In the next part, we will take a closer look at Kubernetes autoscaling capabilities, including the native mechanisms as well as pluggable ones.** So stay tuned for more information!

Next in the series Part 2: Kubernetes.

This blog post was originally published at https://www.polidea.com/blog/