Clustering Infrastructure

Raúl Estrada

Octubre 2020

So far, we have seen how powerful Docker is but we have not unleashed the full potential of containers. You have learned how to run containers on a single host with the local resources without the possibility of clustering our hardware resources in a way that allows us to uniformly use them as one big host. This has a lot of benefits, but the most obvious one is that we provide a middleware between developers and ops engineers that acts as a common language so that we don't need to go to the ops team and ask them for a machine of a given size. we just provide the definition of our service and the Docker clustering technology will take care of it.

In this chapter, we are going to dive deep into deploying and managing applications on Kubernetes, but we will also take a look at how Docker Swarm works.

People usually tend to see Kubernetes and Docker Swarm as competitors, but in my experience, they solve different problems:

- **Kubernetes** is focused on advanced microservices topologies that offer all the potential of years of experience running containers in Google
- **Docker Swarm** offers the most straightforward clustering capabilities for running applications in a very simple way

In short, Kubernetes is more suited for advanced applications, whereas Docker Swarm is a version of Docker on steroids.

This comes at a cost: managing a Kubernetes cluster can be very hard, whereas managing a Docker Swarm cluster is fairly straightforward.

There are other clustering technologies that are used in the current DevOps ecosystem, such as DC/OS or Nomad, but unfortunately, we need to focus on the ones that are, in my opinion, the most suited for DevOps and focus specifically on Kubernetes that, in my opinion, is eating the DevOps market.

Why clustering ?

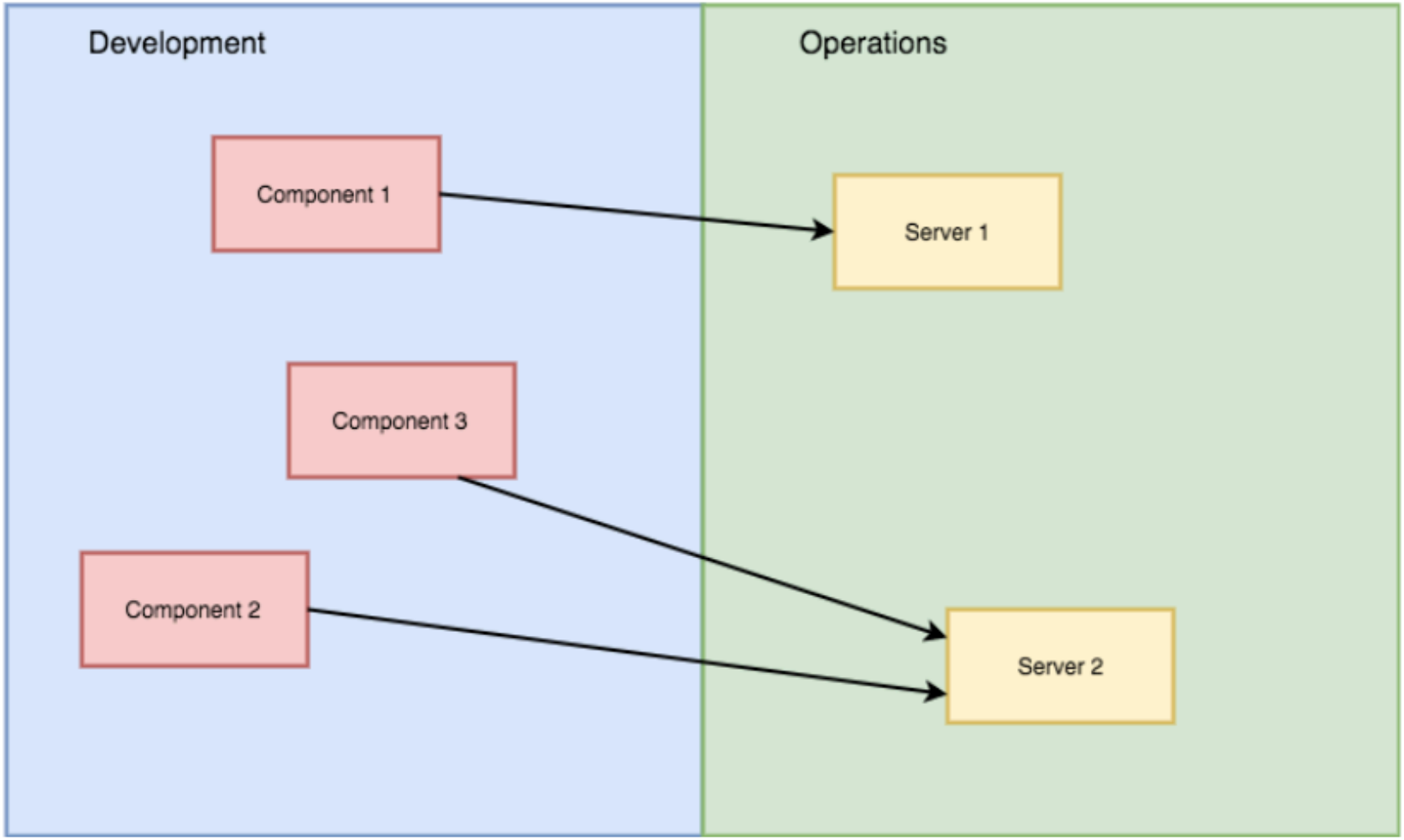
now you need to be a full stack DevOps engineer in order to get success in any project. Full stack DevOps means that you need to understand the business and the technology used in the organisation. Think about it; if you became a civil engineer instead of an IT engineer, it is mandatory to know the local rules (the business) plus the commercial names of the tools used to build roads and bridges (the technology) but also be able to coordinate their building (ops). Maybe not every engineer needs to know everything but they need to be aware of in the full picture in order to ensure the success of the project.

Coming back to containers and DevOps, making concepts simple for everyone to understand is something that's mandatory nowadays. You want to ensure that all the engineers in your project are able to trace the software from conception (requirements) to deployment (ops) but also have in mind predictability so that the business people that barely speak tech are able to plan strategies around the products that you build.

One of the keys to achieving the flow described here is predictability, and the way to achieve predictability is making uniform and repeatable use of your resources. As you learned earlier, cloud data centers such as Amazon Web Services or Google Cloud Platform provide us with a virtually unlimited pool of resources that can be used to build our systems in a traditional way:

- Define the size of the VMs
- Provision VMs
- Install the software
- Maintain it

Or, if we want to draw a diagram so we can understand it better, it would be similar to the next one:



Here are a few considerations:

- Clear separation between Development and Operations (this may vary depending on the size of your company
- Software components owned by Development and deployments and configuration owned by Operations
- Some servers might be relatively underutilized (**Server 1**) and on a very low load

This has been the picture for 40 odd years of software development, and it is still the picture if we are running Docker containers, but there are few problems in it:

- If a problem arises in **Component 3** in production, who is responsible for it?
- If there is a configuration mismatch, who will fix it if developers are not supposed to see what is going on in production?
- **Server 1** is running a software component that might be called only once or twice a day (imagine an authentication server for workstations); do we need a full VM just for it?
- How do we scale our services in a transparent manner?

These questions can be answered, but usually, they get an answer too late in the game plus "the hidden requirements" are only seen once the problems arise at the worst possible time:

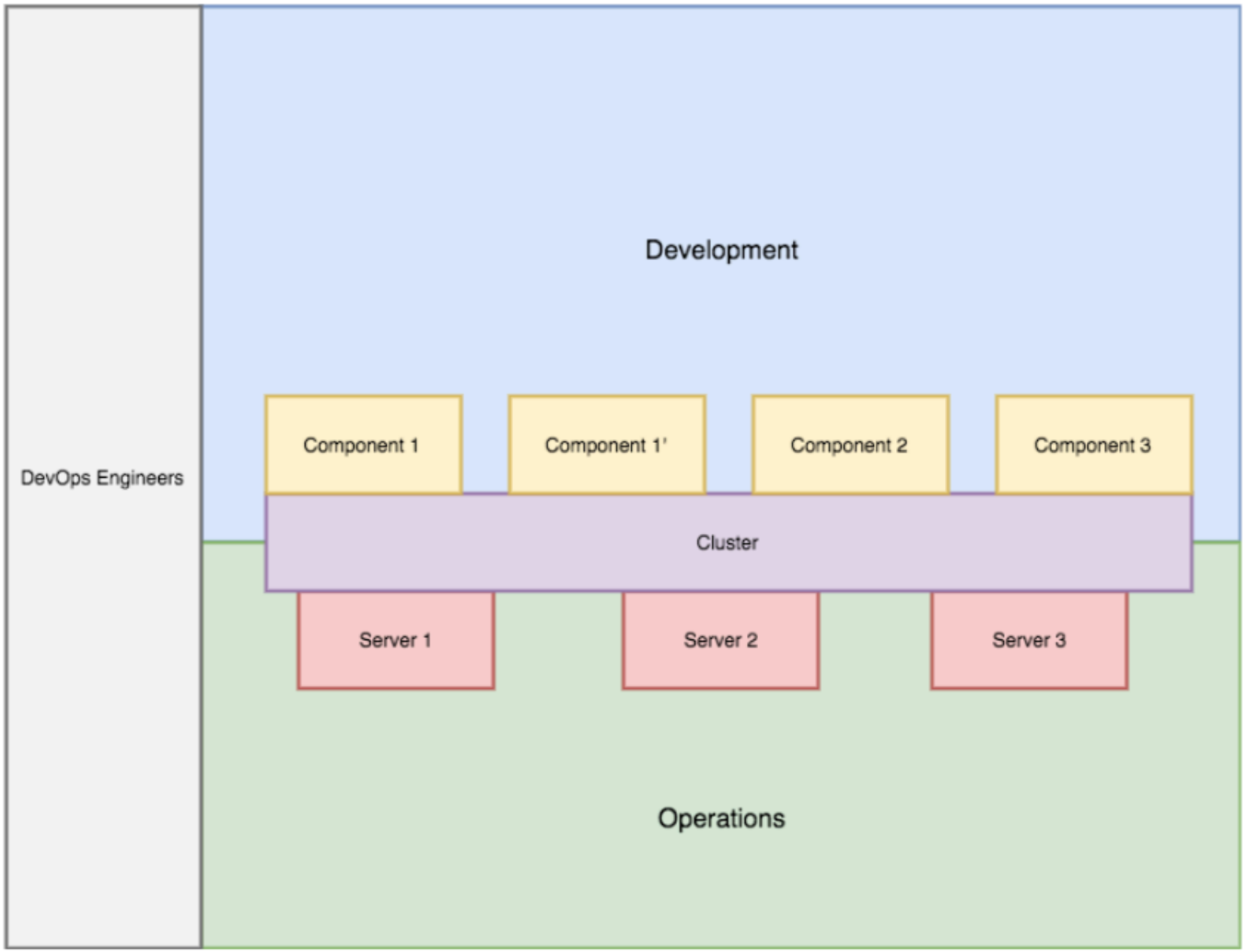
- Service discovery
- Load balancing
- Self-healing infrastructure
- Circuit breaking

During college years, one of the things in common across all the different subjects was reusability and extensibility. Your software should be extensible and reusable so that we can potentially build libraries of components creating the engineering sweet spot (not just software development): build once, use everywhere.

This has been completely overlooked in the operations part of the software development until recent years. If you get a job as a Java developer in a company, there is a set of accepted practices that every single Java developer in the world knows and makes use of so you can nearly hit the ground running without too many problems (in theory). Now let's raise a question: if all the Java apps follow the same practices and set of common patterns, why does every single company deploy them in a different way?

A continuous delivery pipeline has the same requirements in pretty much every company in the IT world, but I have seen at least three different ways of organizing it with a huge amount of custom magic happening that only one or two people within the company know of.

Clusters are here to save us. Let's reshuffle the image from before:



In this case, we have solved few of our problems:

- Now development and ops are connected via a middleware: the cluster.
- Components can be replicated (refer to component 1 and component 1') without provisioning extra hardware.
- DevOps engineers are the glue between the two teams (development and ops), making things happen at a fast pace.
- The stability of the full system does not depend on a single server (or component) as the cluster is built in a way that can accept some level of failure by just degrading performance or taking down the less critical services: it is okay to sacrifice e-mailing in order to keep the accounting processes of the company running.

And about the hidden requirements. Well, this is where we need to make a decision about which clustering technology we want to use as they approach the service discovery, load balancing, and auto-scaling from different angles.