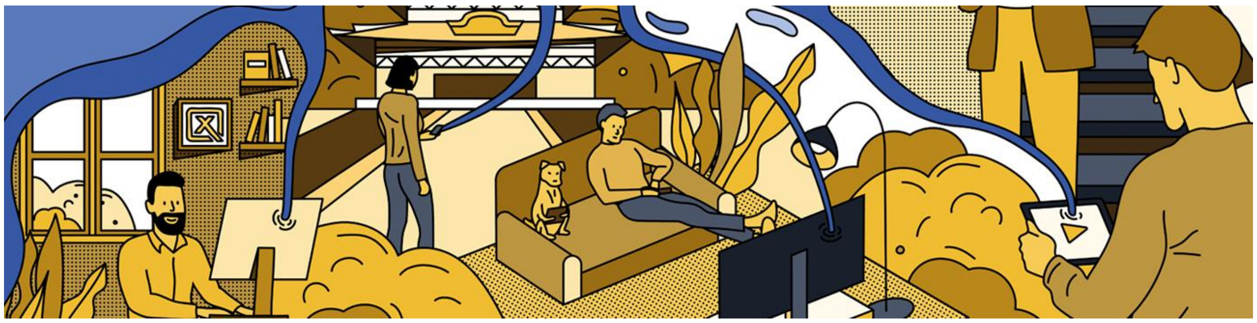


Why is Kubernetes a great fit for OTT

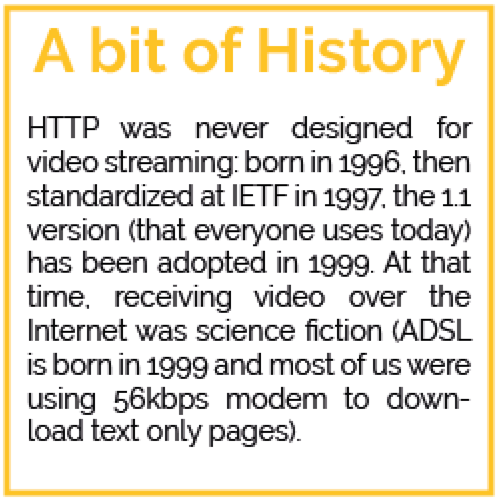
Marc Baillavoine, September 24th, 2019

Since its creation in 2014, Kubernetes has been widely adopted by many forward-thinking companies for several reasons: its open, rapid and safe development, and its ability to schedule, automate and manage distributed applications on dynamic container infrastructure. This new development paradigm (initiated by Google) is driving an inexorable transformation of how modern applications are built, delivered and deployed in the enterprise.

The video delivery ecosystem has many specificities that have slowed down Kubernetes adoption, but the number of features and the level of maturity of the latest versions pave the way for a new era in deploying and operating streaming services

Reason 1: HTTP rocks!

OTT is all about HTTP. It's even part of the name of the (by far) most widely used OTT format: **H**TTP **L**ive **S** treaming (HLS). When you receive an OTT video stream, the player makes a succession of HTTP requests to download chunks of video that are decoded and assembled in your device, producing a continuous stream. HTTP has some very nice built-in properties:



A bit of History

HTTP was never designed for video streaming: born in 1996, then standardized at IETF in 1997, the 1.1 version (that everyone uses today) has been adopted in 1999. At that time, receiving video over the Internet was science fiction (ADSL is born in 1999 and most of us were using 56kbps modem to download text only pages).

- Being based on TCP, it comes with built-in error correction in case of bad network connection,
- Because HTTP easily crosses firewall and routers, allowing HTTP streaming on any network is straightforward,
- Being client-initiated (pull model), it allows advanced client-side behaviors so as to re-emit a request or to select another rendition ladder.

Although any type of network traffic can be used in Kubernetes, it comes with several built-in objects for handling HTTP traffic; communicating between the several microservices of a Kubernetes cluster is easily achieved in HTTP, and lots of HTTP development frameworks are a perfect fit with a dockerized approach.

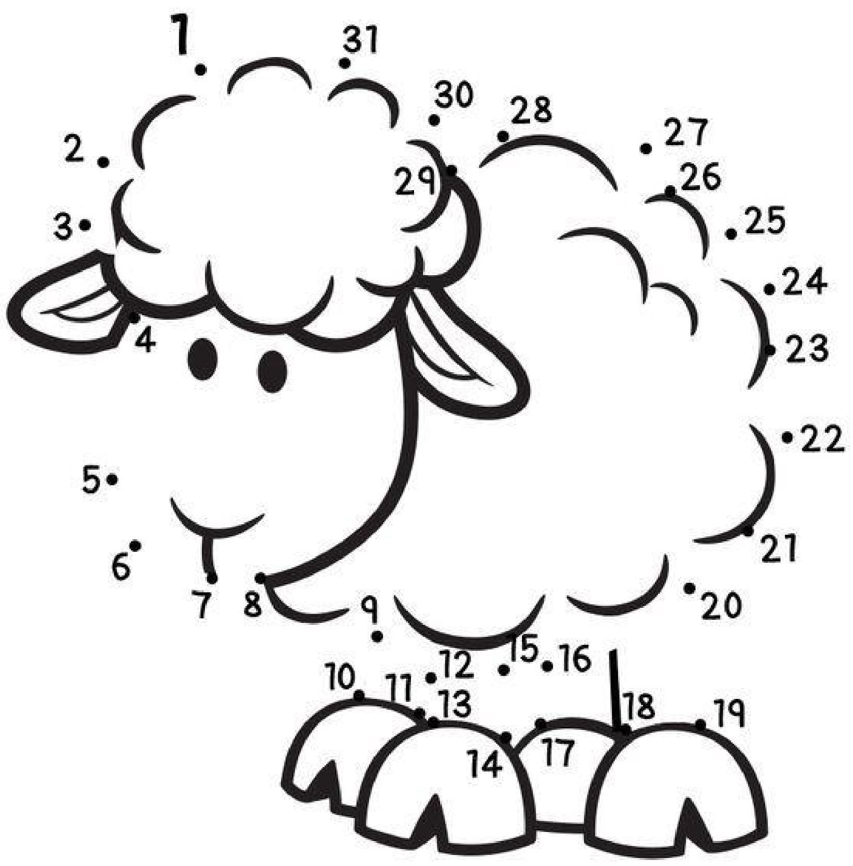
Reason 2: Focus on “what”, not on “how”

One of the keys of a successful OTT deployment is that the delivery infrastructure must be continuously changed to accommodate for the variations in terms of audience and network conditions. This is a radical change compared to traditional delivery schemes that do not depend on these conditions. And this requires new ways of defining your application and your infrastructure.

One of the most undervalued properties of Kubernetes is that it allows a declarative deployment of your application. A declarative description allows to focus on the final shape it must embrace, without focusing on how this is achieved under the hood, whereas an imperative approach describes all the steps to build that shape.

If this is a little bit obscure, let’s take a first example. “Draw me a sheep” is declarative. No matter how the sheep is drawn, that’s not your problem. The imperative form of “draw me a sheep” would be to give step by step drawing directions to end up with a sheep.

Still unclear ? Let’s take a broadcast example. Legacy systems usually adhere to imperative principles: you take a serie of actions to apply configuration delta to an existing system state, leading to a new system state. If something goes wrong in the middle of an operation, it is completely up to you to list the required imperative steps to go back to a normal situation. It is also pretty difficult to take a snapshot of the system configuration at one given time. This usually leads to “if it works, don’t touch it” situations that greatly impedes changes and innovation.



Now, think of the benefits of a declarative approach for redundancy. No matter what happened on the physical layer (hardware failure, for instance), Kubernetes will continuously make sure that the current state matches the desired state by re-scheduling jobs (that are called “pods”) or instantiating new nodes (with the help of the cloud provider) in the background. If your application is correctly built, you will not even notice that something went wrong.

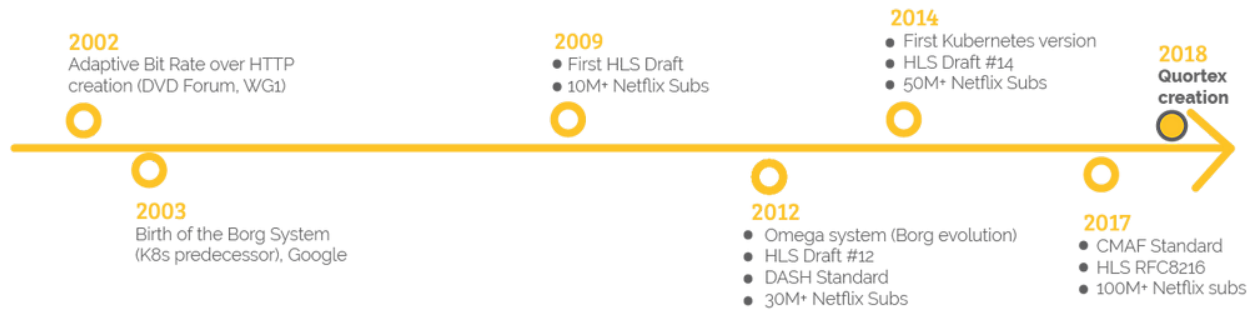
Reason 3: Born to scale

We all know that scaling is a key for successful OTT deployments. Obviously, it doesn’t take the same amount of resources for streaming the first round of the Rapid City Pinball tournament than for streaming the Superbowl final. You also need to scale the compute resources accordingly if you want to use start of the art encoding techniques such as audience aware encoding.

Guess what? Kubernetes was born from an internal Google project (called Borg) in the goal of being able to massively scale an application through HTTP load balancing and replication of processing jobs. Kubernetes was literally born to solve the exact problem that everyone faces with sudden peaks of audience for live streaming.

Reason 4: Share the same development principles

It's striking to notice how OTT and Kubernetes development phases are intimately interleaved. They were roughly born in the same years and they evolved together over the past two decades.



One of the reasons of the massive HLS adoption has been its fast-paced evolution. Unlike traditional DVB standards (that evolved marginally over the years because they are built to last for decades and offer a very high level of interoperability), HLS has gone through 23 differents drafts between 2009 and 2017 before it was finally published as a RFC. If you want to always offer a cutting edge streaming service, it's obvious that you have to use continuous operation schemes. Kubernetes is a great tool for embracing such methods (with concepts of rolling update, for instance) and is widely used by DevOps teams all around the world to provide such schemes.

The future is now

The video industry suffered from many defective cloud/docker implementations, where a software that was never designed for the cloud was brutally ported into a VM or into a docker, ending up with no benefits and more constraints for the customer.

The Quortex solution was built with and for Kubernetes. It was never designed to run outside a dockerized/kubernetes infrastructure. It heavily uses HTTP internally, comes with native scaling and offers a declarative deployment and configuration schemes. Kubernetes has been widely adopted in many different industries and is a rock-solid platform, supported by a very large community. Live streaming and Kubernetes have evolved alongside, it’s now time to use them together to build the future of OTT!