Make it a virtual problem

Put it into a container, and a pod, on virtual cluster

Onkobu Tanaake

2019-02-19T21:00:00

It started two years ago when nearly everybody in my near surrounding working as an IT expert was burning for Docker, containers, virtualization, clustering and all that fancy stuff with Netflix-, Amazon or Google-cloud foundation. It was back in 2015 when I had first contact with IBM Bluemix Garage and the new craze Ionic and/ or AngularJS. Until then I was a heavy weight champion of JEE development with an application server running 50-100 servlets at once. Now I sat with two guys in my office and tried to port the important parts of a few servlets to the new platform – check if it works out and estimate transition cost.

I'm always courious to watch a fancy gadget lift off and… ditch into ground. JavaScript – or formally correct an ECMA Script realization – on client- and server-side was the cure to all problems, especially time to market. Forget about bundling classes into JARs, wrapping JARs into WARs and WARs and additional classes into EARs. Take piece of dynamic typed language, put it into a provider's runtime and see the magic happen right at your hands. This is true for the pure technical part, getting from no output to the famous line »Hello World«. It never teaches me a class of design (software- not UI) nor architecture. So I was not convinced, that a different VM with a different syntax solved user self service, service configuration or input validation any better than what he already had. (Correct me if I'm wrong but user input into a client side JavaScript putting something into the DOM while also issueing a REST-request to a backend that will check again is not a real reduction of attack surface/ bug probability.)

So here we are in 2019. Angular introduced TypeScript and moves towards Java-syntax with big steps. This feels more like GWT or Vaadin. JSPs were cleverer and more mature. I can still deliver the same magic from a servlet, even a PHP-backend can do the REST-trick with a fancy Bootstrap- something, including Fontawesome-glyphs. But none of the technologies keeps me from building a so called *release monolith*. Many single page apps suffer the same problem from back in the days of storing libraries next to UI modules: It needs to fit together. And while the user self service uses version A of a fancy JavaScript library another component requires version B – jQuery, Angular, Bootstrap, an important polyfill for a time or date picker to also support the 1% probable browsers still running on Windows XP-machines. How does Amazon AWS solve this? Angular doesn't solve it neither. And also Node does not provide a hint on that. Spring Boot isn't any better. Architecture is what counts. And if you do canary deployments or A/B or green/blue-testing, how do you configure the routes to get the right resources together and not spilling garbage into a No-SQL-database? It isn't the famous switch from the magazine's opener. You need an experienced Principal Kubernetes Cluster Manager.

So I step way back in time, let's say to the golden ages of package managers for *nix-systems. Why didn't any of my employers go with a RPM to deliver software? I suggested this at Deutsche Bahn/ DB Systems in 2014. Put everything into a readily available mature package manager compatible with the target environment – something from RedHat. They looked at me as if I suggested to use cereals (yes, corn, flakes, muesli…overnight oat) with a CD-drive. All they needed was a container with stuff inside that didn't change and had not only a label on it but also it's expectation about the eco system it will spill its beauty onto – if it fits.

What happened instead was a transition from either Administrator or Developer to so called DevOps. An easy piece of code needs to be tied to gazillions of other similar easy pieces with the help of: a management cluster, a cluster of virtual machines, each virtual node/ pod/ virtualization with an iptable (really!) to lock all network traffic in only poking holes into the safety bag with the help of space-levelled YML-file…holding 100.000 lines. This is what Netflix, Spotify and such are all about: a horde of intelligent and lazy comrads of IT business covering the dirty regions with abstraction. And the rest of us are copy cats – the best strategy to survive in a world full of concurrency. Escapism at its best.

Notes from the textbook/ History class: I was part of a high availability high throughput coding team at the beginning of the millenium. A single HP-UX Super Dome machine with 64 processors ran (and still runs) an ORACLE-database that rendered its own HTML-UI, from PL/SQL. You can't go any faster than that. A standard communication tunnel from Frankfurt Main Airport got a response from Wiesbaden (roughly 1hr by train) within guaranteed limits of under 1s. And it had to search through 10-20 million rows, joined accross 2-6 tables. It was always below the 500ms-boundary. I learned a lot these days about mission critical architecture and software design from freelancers who could call any day rate and got payed. But they didn't study blogs or IT magazines to find the perfect language or platform. They matched the problem space to the tools at hand – experience and methodology.

One simple task is left for you: reflect upon the past 2-3 years, how fast a library, tool or paradigm got outdated, replaced, irrelevant or let's call it *legacy*. Will the technology you managed today still be of use tomorrow? Isn't Amazon Lambda vendor lock-in? All the Google-components you bound into your client calls home through URLs, doesn't this slow down your loading time? A content delivery network requires sophisticated traffic logging to get the hot spots/ click rates that once could be extracted from your local access.log. And an outage of a DNS-cluster leaves Azure-nodes alone in the woods. Do you really run thousands of services that need transaction-tracing through gigabytes of logstash?