LS Lab 2 - Container Orchestration & Load-balancing

> New team of two (not the same as in previous lab)

In this lab, you will learn how to work with the containerization of applications/microservices and their load-balancing, and also find out how this differs from real host virtualization and when it is more efficient.

Task 1 - Choose Container Engine & Orchestration

1. Choose a container engine. Some suggestions:
   - Docker **(default choice)**
   - Singularity
   - `chroot`
   - OpenVZ / Virtuozzo
   - *other exotic choices (be sure that you will be available to deal with it and firstly discuss with TA)*

   Deploy a true container cluster farm, across several team machines. It is however recommended to proceed in a virtual machine environments so the worker nodes can have the exact same system patch level (which makes it easier).

   Bonus: if you choose Docker, play with alternate storage drivers e.g. BTRFS or ZFS instead of OverlayFS.
2. Choose an orchestration engine then:
   - Docker Swarm
   - Kubernetes or local k8s cluster:
     - minikube: works on all major operating systems and uses providers like Docker, Virtualbox and KVM to power the node
     - k3s: lightweight Kubernetes can work on Linux (WSL2 should also work steps here)
     - MicroK8s: lightweight Kubernetes compatible with Ubuntu (Linux) distribution (*can work with MacOS and Windows when using mutlipass?*).
   - Rancher (advanced)
   - *other exotic choices (be sure that you will be available to deal with it and firstly discuss with TA)*
3. Analyze and describe the best practices of usage the chosen container engine.

Task 2 - Distributing an Application/Microservices

Base level (it means that this task will be evaluated very meticulously): deploy at least a simple application (e.g. a simple web page showing the hostname of the host node it is running upon) and validate that its instances are spreading across the farm. It is literally not necessary to create your own service/application: you can use something from a public repository, for example, from *DockerHub*. However, no one forbids you to work with self-written projects.

Hint: creating such application is particularly easy to achieve with Swarm when the nodes does not share storage. Same goes for K8s, but you need use a workaround e.g. volumes with `hostPath`.

Semi-Bonus 1: deploy a microservices instead of standalone application, e.g. full stack web application with web server, database...

Bonus 2: use Veewee/Vargant to build and distribute your development environment.

Bonus 3: if you use k8s, prepare a helm chart for your application.

Task 3 - Validation

Validate that when a node goes down a new instance is launched. Show how the redistribution of the instances can happen when the broken node comes back alive. Describe all steps of tests in your report in details.

Task 4 - Load-balancing

1. Choose a load-balancing facility, which will be installed and tested against your orchestrator. You can choose either l3 or l7 solutions. The disadvantage of layer-3 is that web sessions may break against statefull applications. There is a trick to deal with that, though, for example with OpenBSD Packet Filter.

*Note: for K8S, you can try to do **Ingress**.*

Choose a load-balancer to distribute the load to the worker nodes, for example either layer 3:

- layer-3 BSD `pf` / `npf` / `ipfilter/ipnat`
- layer-3 Linux Netfilter (`iptables`)
- layer 3 Linux nftables (`nft`)
- layer-3+7 HAProxy

or HTTP reverse-proxy:

- layer 7 NGINX / NGINX Plus (dynamic objects?)
- layer 7 Apache Traffic Server? (static objects?)
- layer 7 OpenBSD Relayd
- K8S Ingress / Ingress-NGINX

2. Swarm or K8S's network overlay is already spreading the requests among the farm, right? So why would a load-balancer still be needed? Explain and show briefly how a real-life network architecture look like with a small diagram.

Bonus - Autoscaling

1. For *those who chose Kubernetes*, setup a Horizontal Pod Autoscaler. Perform a stress test and validate that the number of pods is changing depending on the load.

   Hint: to perform a stress test you can use the `stress` utility.

2. Bonus for the others: how to scale instances in the Docker Swarm or another facility? Could it be done automatically?

Bonus - Updates & Monitoring (theoretical)

1. Suppose your application must be updated. Describe the process of applying image or instance updates with your orchestrators of choice, and do it with a minor change in your sample application. In other words, how to ship the containers depending on your engine and orchestrator?
2. It is always a good practice to do monitoring and logging. How can this be done with your orchestrators of choice? Are there built-in tools or must you use third-party ones?