**Assignment solution**

In the assignment description it is emphasized that there are many PHP applications which are quite similar, so this solution refers to **one** PHP application and can repetitively be applied to every other application in the set.

When we are talking about web servers that run PHP applications there are two competitors in the market nowadays, *Apache* and *Nginx*. This particular application is chosen to be run on *Nginx*. The selection is based on the fact that the application produces only one tenth of dynamic PHP content and the rest of it is its static asset. Based on benchmark tests *Nginx* proved to be much more faster than the *Apache* web server. More precisely, two and a half times faster in serving traffic for applications that provide mainly static rather than dynamic content ran for 1,000 concurrent connections. Also, *Nginx* solves the web server load balancing request. *Nginx* can be used as a software load balancer, which will be presented.

Solution assumptions:

* All PHP applications are web applications.
* All server machines have recommended hardware resources (specifications) for average web application requirements (2 x 1,6 GHz CPU; 3,5 GB RAM; 40 GB HDD).
* All server machines are running some Linux distribution OS.
* The solution does not consider hardware failure.

Solution:

The server machine is going to run at least two *Nginx* web server instances of the same PHP application. First, the *Nginx* server is going to have two functions, as a load balancer and as a static content deliver. However, due to the fact that static content does not require much CPU power, when the GET request for static content arrives it will respond by serving it. If a request that requires dynamic data generation arrives, first the *Nginx* server will forward it to another *Nginx* web server intended for dynamic content serving. Web servers are divided based on the type of requests that they serve, because dynamic content generation is CPU intensive. Thus, it must be considered how many instances of the same web application should be run only for delivering dynamic content. It depends on how much of dynamic PHP traffic there is, number of requests and the hardware specifications of the machine that is running particular web servers. In this case, there is only ten percent of PHP dynamic traffic, so the assumption is made that there should be at least one and maximum two web servers only for dynamic content serving. The decision is based on the fact that although many requests for dynamic content arrive, which require a lot of processing power, processing will not take as long as there is only ten percent of dynamic traffic.

So, the conclusion would be that one web server will be run with the role as a load balancer and as a static content deliverer, and one or two web servers with the role of dynamic static deliverers. All servers would be run on the same hardware machine. In order to make this work properly, the load balancer will be run on port 80 and would receive all requests for the PHP application, some of them would serve, others would pass on. All other web servers would be run on non well known (registered or dynamic) port numbers, for example 8000, 8001…

Also, *Nginx* can host more than one web application on one web server instance, by making virtual web servers. This is how this solution would be implemented if containers are not used. With containers, all *Nginx* web server instances could be isolated and run separately, otherwise virtual web servers must be used. Hence, depending on server machine specification and CPU utilization, it can run multiple different PHP applications on one *Nginx* web server instance.

Ansible configuration:

In order to set the configuration on the server machine with Ansible two files are attached, *inventory* and *playbook.yml*. The *inventory* file has specified the name and IP address of the server machine and necessary variables. For testing purposes the IP of the machine is set to localhost and can be easily changed for any server on the Internet, [*simpphpweb.com*](http://simpphpweb.com/) is used as a fake domain also. *Playbook.yml* file consists of three plays. First one is updating current Linux packages, second one is installing PHP fast process manager and third one is installing *Nginx*web server and configuring three virtual servers running the same *simple-php-website* application as mentioned before.

Ansible playbook is run by executing the command ***ansible-paybook -i inventory playbook.yml*** .

If hardware failure was to be considered, then we would use at least two physical server machines and the *Nginx* server would be running on both executing the same PHP application. Load balancer could be implemented on one of those two servers as a *Nginx* virtual web server, as described earlier in the text, or on a third physical server machine. Also, a hardware load balancer can be used between those two physical servers, but they are quite outdated.

CI/CD pipeline:

CI (Continuous Integration) refers to continuous developing and testing of the code. During the writing of the code, every version of the code would be saved by using some version control system, for example GitHub or Bitbucket. The code would be saved on a remote server repository which provides simple and fast access to every code version (commit) in the past. After the code is written it must be built (compiled with all necessary libraries) in order to be tested. Most web application frameworks use the Maven tool in order to automatically download all dependent libraries that the application needs. Continuous testing could be implemented in many ways. The most simple is writing unit tests which will separately test all application functionalities. Other more sophisticated ways would be using static or dynamic automated tools for testing. Static testing tools, such as SonarQube, run through written code and look for any potential vulnerabilities and mistakes. On the other hand, dynamic testing tools such as GitLab, generate millions of requests while the application is running in order to find potential vulnerabilities that could harm the application asset.

CD (Continuous Deployment) refers to deploying code to test servers and after deploying (delivering) to regular servers as releasing the code. When deployed on test servers, the application is showed as a simulator and open to use. If it turns out that is not capable to support necessary functionalities its turned back to some of the previous steps to be fixed. After the application showed that it is ready for production it is deployed to servers which are open to clients. The last phase of CD is monitoring the performance of the application that is released. That can assure system engineers if the application is properly working or not, or if it is targeted by hackers. For this PHP application, the best monitoring solution would be the Stackify tool. Monitoring of web applications never stops, which lead us to the fact that CI/CD pipeline is infinite.

AWS auto scaling:

Considering emphasized requirements, the Amazon EC2 auto scaling service would be purchased which will automatically scale EC2 (very similar to VM) instances in the cloud, based on policies that would be configured. When the service is purchased, first the launch configuration is defined, and it describes the parameters (OS, application server and security group) of each EC2 instances that would be used in the auto scaling group. Afterwards, the auto scaling group configuration is defined. Most relevant configs would be the maximum number of EC2 instances which would be used and what would be the load metric used for auto scaling (scaling policy). For the scaling policy, the option from another AWS service would be chosen, which is the request count per target from the Elastic Load Balancer service. This would include load balancer in our auto scaling group and the horizontal auto scaling would be applied based on the number of requests arriving.

Docker:

Containers were mentioned earlier in text. The point is, that containers would allow for running many separate (isolated) *Nginx* web server instances on the same physical server machine and it would not be necessary to use *Nginx* virtual web servers. What is better? The answer should be based on performance tests of the *Nginx* web server and physical server machine, type of PHP web application and number of coming request also. At this point an answer can not be constructed.

Docker containers can be implemented in several ways:

* Manually make *Nginx* image file with all necessary configurations and PHP application by writing Dockerfile.
* Run docker official *Nginx* image and mount local OS directories to key container configuration folders over Docker host by writing bash scripts for example.
* Copy local OS directories to Docker host while creating container, which will be used by container instance when run by writing Dockerfile.

Comprehensively speaking, running containers has a vital advantage in fault tolerance. If one container instance fails, another can be very fast and easily run, comparing to the failing of the *Nginx* web server which would lead to falling all *Nginx* virtual web servers on one physical server.