Cloud Native Development part 2

Stateless, autoscaling, fault tolerance.

A lot depends on the correct architecture of the application. There is a vast difference between a random tutorial on the Internet and a real industrial application. The same can be said about the source code generated by AI. Currently, AI is unable to design large, complex systems with the correct architecture without active human assistance.

Thus, the specialist is responsible for the fate of the project. Let's look at this from the scaling side.

Imagine that you have created a food delivery application. It works perfectly in a test environment and processes one order in five seconds. That is how long it takes for a request from the mobile application to the server, calculation of the delivery route, writing to the database, and response back to the mobile app. This application functions identically on another server, which we have designated as a production server. What could go wrong?

Here's what. Our partner was a chain of snack bars. Our city alone has a hundred restaurants, with ten couriers for each. During rush hours, when all the couriers are busy, we receive thousands of tasks every hour. The partner aims to connect neighboring cities, and we must prepare for expansion. As the number of tasks increases significantly, the application's performance and reliability will inevitably suffer.

And here application architecture plays a key role. It determines our scaling capabilities. Their diversity comes down to two basic strategies: vertical and horizontal.

Vertical scaling involves adding more resources to our server. Previously, it was necessary to manually adjust the processor, insert new RAM boards into the slots, and change the hard drive. Now, in cloud technologies, we buy the necessary resources from the vendor. This approach is more convenient because we rent these resources, rather than accepting them as fixed assets on the enterprise's balance sheet. We do not need to worry about their depreciation, failure, or obsolescence. We can refuse them if they are no longer required.

Horizontal scaling involves creating clones of our server. Each instance is no different from the others. An additional layer appears above the servers - a load balancer. This mechanism ensures that incoming requests are transferred to the least loaded server. As a rule, the cloud platform itself provides such mechanisms. In some cases, the platform can launch new instances itself and turn them off if the load has decreased. This is called autoscaling.

In an ideal situation, we should use both strategies. Whether this is possible depends on the specific architecture of the application. If we design the application to store all sessions in one place, it will be challenging to scale it horizontally. On the contrary, if an instance processes a task by accessing a third intermediary service for the necessary data, then it does not matter how many instances we have. This approach is called stateless. Once again, we emphasize that stateless does not store user data. For most tasks, this is necessary. How else can a customer's order be processed if the application doesn't remember their preferred dish? However, this data should not be stored in the memory of a specific instance, but placed, say, in a database. Then the instance can get the order details at any time, based on the transmitted identifier (it comes in the request from the client application). Let's say a hungry customer checks his smartphone every five minutes to find out the state of his dinner. Each request has the same identifier, and the requests are processed by different instances, and it does not matter which of them received the request this time.

Such flexible architecture is not always found, especially when it comes to old applications. They are characterized by vertical scaling, and horizontal scaling requires a lot of work; the solution will be a homemade solution.

Therefore, applications have a scaling limit. With vertical scaling, this is quite obvious because no vendor can place infinite resources in a specific virtual machine. With horizontal scaling, things are better. Theoretically, the number of instances can be any. In practice, an excessive number increases the load on the load balancer. In this case, you have to create several load balancers and create a hierarchy of them. In this case, a stateless instance can operate without knowledge of its clone count or the control superstructure above it.

In conclusion, let's talk about fault tolerance. This is a measure of how the system resists failure. If we are talking about a system managed by a load balancer, a catastrophe will not happen. The failure of one instance will entail an increase in the load on the remaining ones. And if the autoscaling mode is enabled, the system itself will create another instance to replace the faulty one. The situation is much worse in the case of old systems, where there is only one server or a small number of them. Often, maintenance (restart) of such systems is in manual or semi-automatic mode. The failure of the server will entail the failure of the system. Which often happens.