KTH Royal Institute of Technology
DD2482

Containerized vs serverless architecture - When to switch to serverless and when not to

Samuel Söderberg - ssoderbe@kth.se

May 3, 2022

Contents

1 Introduction

Software and the distribution of software have evolved a lot the last decades. With that evolution, the demand for quick and easy updates, patches and fixes, and new and improved features have increased a lot. Today we expect bugs to be fixed and solved within the hour of someone reporting the bug, and as such the need for continuously developing and deploying software in a safe and controllable way has increased.

Choosing how to structure or architecture your code is an important part of how your software will be developed and delivered, and new ways of doing that arise frequently. The containerized and the serverless architecture are two somewhat newer approaches which both are microservice architecture. Both approaches tackle how microservices are deployed and distributed, but how do they differ and is one or the other better than the other?

After reading up and studying the two architectures, it is clear that the two approaches offer different services and levels of control for different needs. Some products might use the serverless architecture in a most efficient way while others might see it too constraining and the containerized approach much more suitable to their needs.

2 Microservices - State of the art

Having a CI/CD-pipeline when delivering software is in todays standard almost mandatory. Updates, patches, and features should be continuously added to whatever software is out there, and it should happen seemingly seamless. A microservice architecture enables this by letting each microservice have its own pipeline, one that could be determined by the

product owner, other services tech stacks, or the developers behind it. This makes sure that each microservice can have a CI/CD-pipeline which suites that service the best, enabling scalable and flexible State of the art deliveries and deployment of code.

Having a containersized architecture or a serverless architecture are two ways to establish a microservice architecture. Containers are frequently used today by many services, many using the open-source state of the art container orchestrator Kubernetes. Serverless is another approach which is being used today by many to host microservices, for example with AWS Lambda.

3 Defining a containerized architecture

Deploying software with various dependencies flawlessly to different machines running different operating systems and/or versions of underlying software makes for a tricky task. Updating and making sure that every requirement is met on the system that your software installed on gives us developers a lot more hurdles when developing and deploying. Packaging software with *all* its dependencies in a single unit is one way of tackling this problem, putting the software with all needed dependencies in a *container*.

Containers are not all different from virtual machines in that they contain whatever dependencies are needed in a single unit, but containers are a lot more light weight than VMs. All VMs are isolated units that contain and replicate full operating systems, a guest OS with their own binaries, drivers, and kernel that they run on, all the way down to the hardware. As such, replicating, booting up, installing, and maintaining several VMs may eat up a lot of memory and not be as time efficient as one might need software to be.

Containers on the other hand can share an OS along with needed binaries and drivers since they only contain and replicate software that is above the operating system. Containers have images that hold what is needed for an application to be executed but uses the host OS and kernel, and boot up time and memory is therefore handled much more efficiently when running several containers. Container images consists of several image layers of dependencies, where layers can be reused in subsequent builds by other containers further minimizing needed memory [2].

3.1 Why containerization?

Having a containerized architecture enables light weight deployment that is quick and memory efficient. Housing and managing servers or ordering cloud services for hosting is a mayor expense for growing software companies, and thus minimizing and optimizing needed hardware will reduce expenses a lot compared to using VMs. A containerized architecture also allows for complete control over the software, over policies, load, resource management, and more [1].

Most container technology is open-source which further lowers the cost of using containers since no advanced needed software is locked behind expensive vendors. Open-source also enables developers to more easily contribute with pinpointing problems or providing solutions or features.

Firing up new containers when load is increasing takes a fraction of what it would take to fire up a VM, and scalability is therefore greatly enabled when using a containerized architecture.

4 Defining a serverless architecture

Serverless evolved due to the large amount of shifts towards microservices and containers, and it presents a new way of deploying software in the cloud. The name suggests that no servers are being used, but that is not the case. A serverless approach lets the cloud provider manage and handle the servers that are being used instead of the developers and thus speeding up development of applications and software [4].

Building an application or a software that will need some form of cloud computing requires you signing a plan with a cloud provider, often charging for constantly keeping servers up no matter the load or number of requests. A serverless architecture is an event-driven approach where the providers dynamically allocate needed resources to processes, and only charging whenever resources have been allocated. This is called FaaS, Function as a Service, as the code is in the form of functions that are sent to the cloud provider to be run when an event is triggered. The execution runs on the cloud providers servers, providing the service you set it out to provide.

4.1 Why serverless?

As mentioned, this greatly reduces the time needed for managing and scaling servers for the developers, increasing the time they can spend on actually developing the software. It lowers costs for both the developers having the serverless architecture and the cloud providers by more effectively allocating resources only when needed.

5 Comparison

So how do the two architectures differ? Why would one want to choose one over the other? This section will be divided into 6 parts comparing different aspects of development and deployment and some reasoning on when one would outshine the other.

5.1 Scalability

Scalability is an important aspect of software. If a product gains popularity substantially over a night then we would want that product to able to handle that new increased load. Providing new users with a slow or laggy initial experience could easily scare them away from your product.

A containerized architecture would demand the developers to know how many containers the consumers might be interested in. Adding more containers is an easy task, but there will always be a roof.

The serverless approach automatically scales when there is a demand for it. The cloud provider takes full responsibility for providing your service to whichever customer demands it, and only charges you for the resources allocated.

Serverless offers a much more scalable solution and is therefore good when the workload fluctuates and the need for scaling might come ad hoc. There is no need to try to predict the load on your product and it is far more efficient to only pay for what resources you have used in those cases.

5.2 Maintenance

Maintaining a server and making sure they are up and running and scaling as needed could be quite the hassle. As mentioned in section 4.1, serverless offers a much more manage-free solution than containers, since there is no need for managing the servers as this is left to the cloud providers. In a containerized architecture the developers would still need to manage, update and determine scaling of containers.

5.3 Workloads

Setting up functions in a serverless architecture has some limitations set by the providers. For example, AWS Lambda has a limitation on the deployment package at 50 MB, a disk space limitation of 512 MB, and a maximum execution time of 15 minutes [3]. So a function which takes longer than 15 minutes would need to be split up into smaller functions.

This is not a problem when using containers which can be allocated however much resources are needed. Containers are therefore a better solution when handling heavier software.

5.4 Security and control

The serverless approach is one that is easy to learn and easy to use. It enables a lot more time to be spent on development of the actual software instead of on server maintenance or management. A containerized architecture on the other hand requires a lot more knowledge in the field and a lot more time spent on management, but it also gives the developers full control over the software, servers, and the system as a whole. It could take a while to learn how to fully extract the potential of a containerized architecture using for example Kubernetes, but it might lead to a system that the developers are in better control of. Managing resources, scaling, and security of your software lets the developers see both inside and outside the software, as opposed to a serverless approach where some parts of the pipeline might be considered a black box.

Determining how much control you would want over your software depends a lot on the software. A more data-sensitive or high-performance software might want to stay away from serverless, while a smaller product where the underlying disk handling or OS optimization might not matter too much could better make use of the serverless approach.

5.5 Testing

Testing is an important part of software engineering, it is highly important that whatever services you host does what you and the customer expect it to. As mentioned earlier with the serverless architecture, the back-end and server processes are somewhat in a black-box, and replicating what they do is not always possible and testing code in a serverless architecture is therefore difficult.

Containers in software with the containerized architecture on the other hand runs on the system that they where deployed upon, and therefore a lot easier to test than code with the serverless approach.

5.6 Cold start

A problem with the serverless architecture is the cold start, where the set-up after an initial invocation of a function takes a longer time. A cold start could be present if a service have been unused for a longer period. Services which needs to establish connections quickly with other services might time-out, and getting responses from the service could from time to time take longer than expect.

The containerized architecture does not have this problem since the server that is running the containers is constantly up, and firing a new container when necessary will not take any longer than usual.

6 Conclusion

Microservices are frequent in our current state of software development, and a serverless or a containerized architecture are great ways to enable a microservice architecture and a flexible pipeline for products. So when should you switch to serverless?

Containers provide a controllable and safe environment in which the developers have complete control over the software. It provides a solution which can handle any type of workload or heavier process.

Serverless provides a easy to use, setup, and quick to launch solution which minimizes the need for server and scalability maintenance. It provides a black box from the cloud providers which makes sure that your software is up and running and accesible to anyone anytime.

6.1 Key take-away

Deploying a younger or smaller product quickly to make it accessible to any and all might see the serverless approach as an efficient architecture, while systems which require more control over the underlying workings might see serverless too limited or constraining. In the end, choosing which architecture to follow depends a lot on the software, they are two different approaches which suits different needs.

7 Reflective part

As a student I still lack quite a lot of knowledge and work experience in the software technology area and i have never prior to this semester examined neither the serverless or the containerized architecture. I have heard a few times of Kubernetes and that it is most effective and advanced. With that said it was really interesting reading up on both approaches and seeing how they differ and outshine one another. The serverless approach seems like an innovative idea that is somewhat of a game changer for smaller software products, it seems like such an enabler when it comes to easily getting an initial prototype unto the market. Some cloud providers don't have support for all languages so it might not make the fit for all, but there seems to be enough so that most will have some languages they can work with.

The containerized architecture seems to fit a lot better with larger software products and larger companies which consists of several teams each handling their own part of a software. Maybe having their own CI/CD-pipeline for their service. Testing is something that I deem

very important, and the lack of it a serverless architecture in a larger context is quite the downside.

If I would be in a start-up or smaller product team trying to figure out how we should develop a new product, I would probably root for a serverless architecture to start with as this also enables more focus on developing the product. But, the containerized architecture is also very interesting and i would probably choose that in a larger setting, allowing for easier testing and a more manageable system. I had a first experience using Docker for a previous assignment and it seemed really efficient and not very troublesome. I look forward to working and getting experience in both architectures.

References

[1] *Containerized Architecture: Components and Design Principles*. 2022. URL: https://www.aquasec.com/cloud-native-academy/container-security/containerized-architecture/.

[2] *Containers vs. virtual machines*. 2022. URL: https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms.

[3] *Exploring AWS Lambda Deployment Limits*. 2020. URL: https://dashbird.io/blog/exploring-lambda-limitations/.

[4] *What is serverless?* 2017. URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless.