Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_11

DeepScience
Open Access Books

Chapter 11: Leveraging cloud-native microservices, containers, and serverless architectures for artificial intelligence pipelines

11.1. Introduction to AI Pipelines

AI pipeline automation can be described as orchestrating the machine learning or deep learning life cycle with a software suite. These software suites could be open-source popular tools such as Kube Flow or MLflow, commercial services by platform providers such as AWS SageMaker and Azure ML, or something between which is a solution built on top of orchestration tools such as Apache Airflow or Google Cloud Workflows. These AI pipelines can provide benefits such as organized code, flexible workflows, and easy reproducibility. In machine learning, being able to reproduce your work (and someone doing the same) is crucial in both debugging and research.

AI pipelines could also refer to the API for specific cloud-based AI services such as Google Cloud Vision or Azure's various Cognitive Services. The more granular the services being offered, the easier the life for most data scientists and engineers. Tasks such as image classification, object detection, image segmentation, optical character recognition, natural language processing, and so on, can be achieved with one API call and done in seconds. Often, the actual solution is calling the specific service and using a small amount of data to build specialized models. These specialized models can then be leveraged in conjunction with the data strategies used in traditional machine learning to achieve better quality predictions (Lakshman & Malik, 2010; Bernstein, 2014; Breck et al., 2017).

All models referred to in this talk are stored in git repositories. Machine learning projects require team members to collaborate and be able to refer back to models used to generate high-quality predictions. You need the project type controls of using code without the

bottlenecks of coding. These clear project goals and organization are model management systems. MLOps and other associated processes is the answer to reducing the timeline between idea and implementation, creating annotated models, and having the ability to reproduce.In recent years there have been revolutionary advances in cloud technology, allowing parties to create portable, dynamic, lightweight, and optimized applications without the headaches of typical deployment and operations. APaaS leverages cloud-native microservices, containers, and serverless architectures to enable agile and industrial strength enterprise-grade AI/ML pipelines for intelligent applications. Business units can supplement the core AI/ML competencies of the central teams by enabling these self-service enterprise pipelines, eliminating most of the enterprise pain points while allowing central teams to continue to dedicate their focus on innovation and expert models (Zaharia et al., 2010; Sato & Takahashi, 2020).

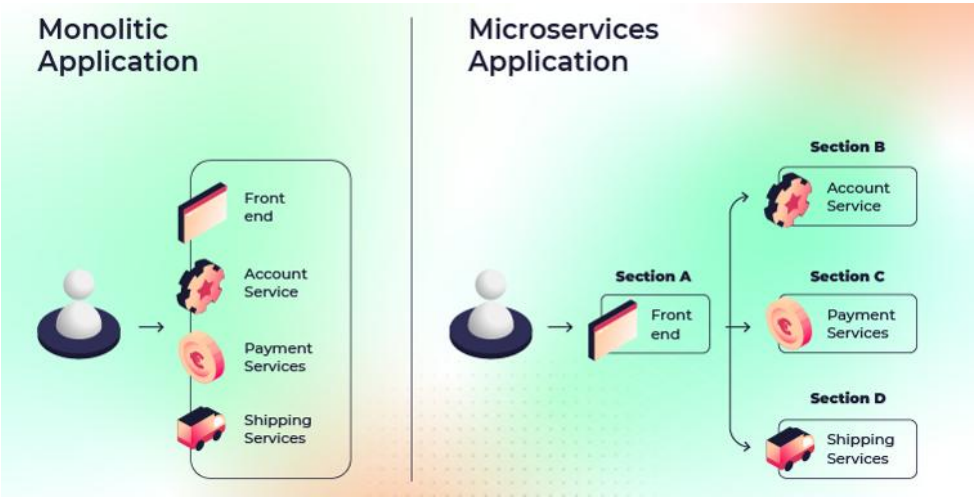

Fig 11.1: Cloud-Native Microservices

11.1.1. Background and significance

AI and ML are becoming integrated into most aspects of modern applications. Businesses seek to enhance various processes with AI and ML. However recently enterprise investments in AI/ML have slowed, as poor experiences with deserted prototypes abound. In many cases this is due to a lack of underlying platforms that are designed to support scalable AI/ML workflows based upon production-quality data and models.

AI/ML applications process vast amounts of production data at speed and scale, extracting insights from user-generated documents, emails, text, images, voice, and video. Scalable platform pipelines are required to prepare the data for suitable model training, evaluate and improve the model performance, and deploy suitable models to the application so that AI/ML can continuously innovate. An intelligent business should

have streamlined, automated pipelines for monitoring these processes to enable continuous tuning of AI for changing enterprise conditions.

11.2. Understanding Cloud-Native Architectures

We do not claim that cloud-native architectures are a novel idea. The convincing advantages of microservices and containers argue for such software development principles ever since. However, the deployment of cloud-native architectures in practice shows to be problematic in the context of AI pipelines. AI-related software is often not deployed in compliance with the recommendations for microservices and containers, and lead to poor performance and inefficient resource utilization. The term cloud-native is to express the idea that AI pipelines may benefit tremendously from existing software development concepts.

The foundation of cloud-native solutions can be traced back to microservices, containers, and serverless architectures. Microservices are a way of structuring an application. Back in the days, applications used to be monolithic: a single big codebase with a lot of dependencies. It was difficult to maintain that codebase, and it involved considerable risk to deploy it. Modern applications are built in microservices, individual executables that communicate via a light-weight communication protocol. They are loosely coupled and can be maintained individually and deployed independently. Cloud-native architectures have become possible by the advance of container technology. Containers are the lightweight, standardized way of packaging executables, the code and the dependencies. This enables seamless deployment, scaling, and management of microservices since it removes all the friction when deploying code. Serverless architectures use the abstraction of containers and microservices to let cloud vendors provide automatic scaling and pay-per-request business models. In a serverless architecture the vendor takes care of additional scaling and management burdens when running microservices, relieving developers of the necessity to do this.

11.2.1. Research design

This exploratory research utilizes a single case study methodology to investigate architectures for naturalistic AI pipelines in higher education, the automation of student success prediction. The project was selected as a well-known case representative of the emerging use of AI for higher education; the production technology was the first full-time, fully-automated proposal of national intent. Data were collected from several sources: site visits; real-time observations of pipeline construction, tests, and results; semi-structured interviews with faculty, post-doctoral researchers, and system builders; and urgent responses to conceptual and implementation questions submitted by email.

A participative observation strategy was adopted given the increasing demands for rapid-scale, production-ready implementations of data science solutions across sectors and domains. Therefore, lengths of data collection were chosen to enable deep engagement with production team members but not restructuring of work utilization. The primary period of data collection occurred over three months. Both email and in-person interviews were conducted throughout that primary data collection period. Additional follow-up data collection occurred at various intervals over a year, as faculty built and deployed two more AI solutions for higher education. Existing models provided a sound foundation for analysis. The exploration heuristic provided the analysis of foundational, application, and automation layers of the AI pipeline.

11.3. Microservices: The Building Blocks of AI Pipelines

Microservices have become an important architectural pattern for building complex, distributed software applications. Pipeline-based architectures are very well understood within the AI community. Every AI pipeline typically employs various building block components for processing, transforming, filtering, and storing data along the way to the final results. These building block components become natural candidates for being designed and built as independent microservices. Not only does that enable us to execute the pipeline in parallel across the various independent microservices, but also increases the reuse of the components in various other AI pipelines and workloads. Coupled with the natural capabilities of container platforms to deploy and manage containers, and offered as a service by platforms for workloads with bursty execution characteristics, microservices are an ideal choice of architecture for hosting the components of AI pipelines.

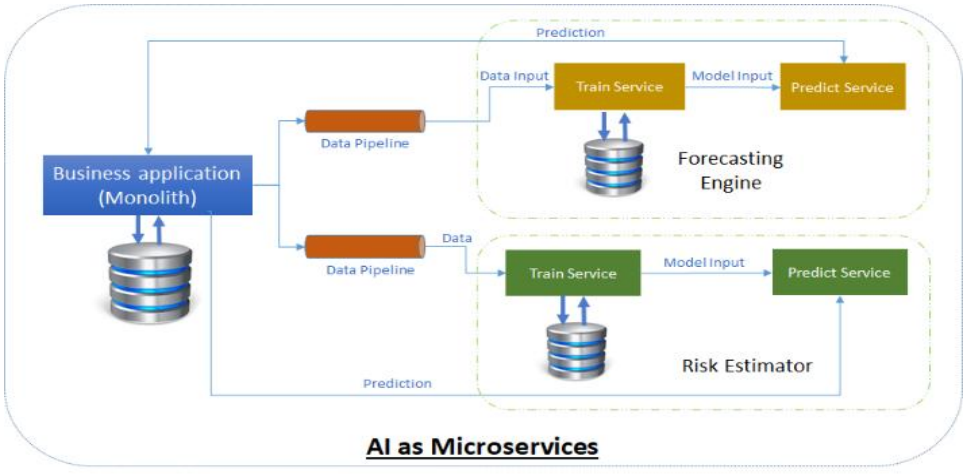

Fig 11.2: AI models as Microservices

A microservice architecture is a grouped collection of loosely coupled independent software components built around business capabilities. Compared to a monolithic application, microservices are often easier to build, easier to manage, easier to understand, easier to scale out, make it easier to deploy new services, easier to add new features, and easier to adopt new technologies. Microservice components communicate with each other using a lightweight mechanism. Services are developed independently in parallel with the autonomy to select the programming language or storage technologies of choice. Services are scalable and enable scaling the components that are in demand, without needing to scale the entire application. A microservices architecture is conducive to DevOps, Agile development methodology, enables Continuous Deployment, Continuous Testing, and Continuous Integration, helps build applications at a faster speed, and reduces the time taken for experiments.

11.3.1. Defining Microservices

Even though microservices are commonly lumped together with other cloud-based technologies such as containers and serverless computing, microservices are a separate concept that is worth discussing in more detail, because they provide deep, lasting benefits for creation of AI pipelines that are not necessarily available with containers or serverless architectures, alone or in combination. Microservices are also the building blocks of containers and serverless architectures.

So, what are microservices, and how do these differ from containers, serverless architectures, and similar technologies? Microservices are a specific software architecting approach that prioritize distributed, decomposed design over monolithic design. Each microservice is a standalone process that offers well-defined functionality as a service to other processes, via network-based API calls. Microservices freely communicate over a shared network, but they maintain their own process state, utilize their own dependencies. Third parties can implement microservices at different time intervals and organizational units. Microservices enable teams to leverage heterogeneous tech stacks: one group can build a consumer microservice in Python, another can build a service in Ruby/Rails, yet another can consume these through a React outage, and so forth. The inherent interface contracts keep the entire system working seamlessly, whether it is simply evolving through ad hoc changes or becoming a major product that benefits one or more organizations. Each microservice can use a different database, from NoSQL to graph-based, as required, and virtually any data movement required can be implemented. Most importantly, microservices make DevOps easy.

11.3.2. Benefits of Microservices in AI

While large corporations on the cutting edge of innovation are primarily responsible for these and many other stunning recent breakthroughs in AI, such as self-driving cars, commendable strides in applied AI are also being made by academic institutions and in the for-profit and non-profit sectors. What do these corporations have in common that makes them able to afford the sizable overhead associated with such research? Simple! They make an astounding amount of money, while the budgets of those who wish to keep playing catch-up, in academia and elsewhere, are limited. Microservices can help developers in the for-profit and nonprofit sectors be more productive, so they can develop new AI products and related AI capabilities faster and at lower cost.

Owing to the tight timelines, high velocity of change, and demand for high availability normally associated with contemporary systems, the microservice architecture has evolved in the domains of web scale and enterprise management systems toward the highly granular, autonomous services so hot in these domains today. These systems tend to have many microservices, comprising common capabilities like provisioning, management, metering, monitoring, and orchestration, that compose AI services. The existing microservices are often simple wrappers around libraries or binaries from the existing rich ecosystem of flowing AI tools. Incorporating other aspects of AI product development is greatly facilitated by the microservice architecture, as most AI pipelines today have multiple stages that may be deployed to different cloud environments or edge devices. The pipeline steps depend on capabilities like storage, processing, modeling, scheduling, and orchestration that are common across pipelines, and that the microservice architecture provides.

11.4. Containerization for AI Workloads

In the picturesque dawn of CI/CD, microservices began illuminating the path of traditional software architecture, slowly eroding much of the monolithic implementations of yore. These diminutive, liberating constructs became the foundation of agility in software development; so too has containerization become the foundation for most AI workloads in the modern world. Whether it is APIs for inferencing, microservices for training, large-scale data engineering for pipelines or model serving for previously-tuned models, the ability to specify AI workloads with little friction, and deploy to any platform in the cloud for rapid development of beginning phases is made possible via containerization and orchestration services.

Through the process of containerization, developers can focus on the actual logic required in code to make the AI model relevant for end-users. When the model is trained, it can take on several environments or flavors for model-inference services - each

specific to the type of service the model is doing, the size or scale of the model associated with that flavor, and thus the development languages and libraries that can get utilized to create that optimized model-inference AI service. The decreased friction with respect to resource utilization and management allows for rapid prototyping of services and user acceptance tests for data-driven solutions. In conclusion, containerization has greatly changed the landscape of AI workloads. Projects relying on a single monolithic development environment, or languishing in QA stagnation, have jumped on the lightweight nature of containers to redefine and reshape. The flexibility and lightweight nature of containers has extended the power of microservices to AI workloads.

11.4.1. Introduction to Containers

Containers provide an efficient and well-defined environment for applications. Unlike virtual machines, which abstract away hardware and offer a guest operating system to run applications, containers share the host operating system kernel and simplify startup, shutdown, and end-user experience. In a cloud environment, containers, by virtue of sharing resources at the process level, provide a much lower overhead compared to virtual machines. But how do applications running in different containers communicate?

Containerization uses networking namespaces and mount namespaces to provide the illusion of separate network and filesystem. Unlike operating system processes that directly expose the network and file system of the host machine, the files on the filesystem and the network interfaces of a container are not visible to the host or the world outside. Container or service discovery – making applications aware of other applications communicating with one another – is also necessary. The simplicity of the container abstraction is that no libraries or SDKs are needed on the client side to use containerized services.

Containers contain everything that an application needs, and any language or framework that the application depends on, to run. Because container images are portable and self-describing, containerizing an application guarantees that it will run the same on any infrastructure that supports containers, from a laptop to cloud-based virtual machines and serverless platforms. Containers are ideal for unit testing computer vision algorithms using common dependencies that are packaged as images.

11.4.2. Docker and Kubernetes in AI

Docker and Kubernetes have incredible importance for the development and deployment of AI models. This section quickly explains those tools, where to find them, and how to configure TensorFlow to use them properly. The solution does not always work

smoothly, even for simple configurations, such as running K8s for Local. AI solutions that require a lot of external storage, be it large datasets for analysis or a large database to serve the outputs of a trained model or a tool that helps with weight tuning or both are often a headache to combine with K8s. Still, containerizing your AI solution is, for several reasons, an important activity, especially if the company already containerized other business intelligence modules, making a share of tools, for example.

Containers are an abstraction for Packaging solutions, developed by Docker technology, which have been around for some years. To be a bit more precise, containers are a lightweight version of full system virtualization. Because of this lightness, Containers could be integrated not only in the deployment and production processes but also into the development processes. Because Containers are some file systems shared with a lot of mini virtual machines, they are also very good in use for distributed computing. Google found that the infrastructure they virtualized was potentially variable for the entire life of the process, and tools that virtualized the entire process were very resource-hungry. Some years later, the Kubernetes orchestration system appeared. All systems that depended on Containers to accelerate their use of resources started using K8s to orchestrate their deployments.

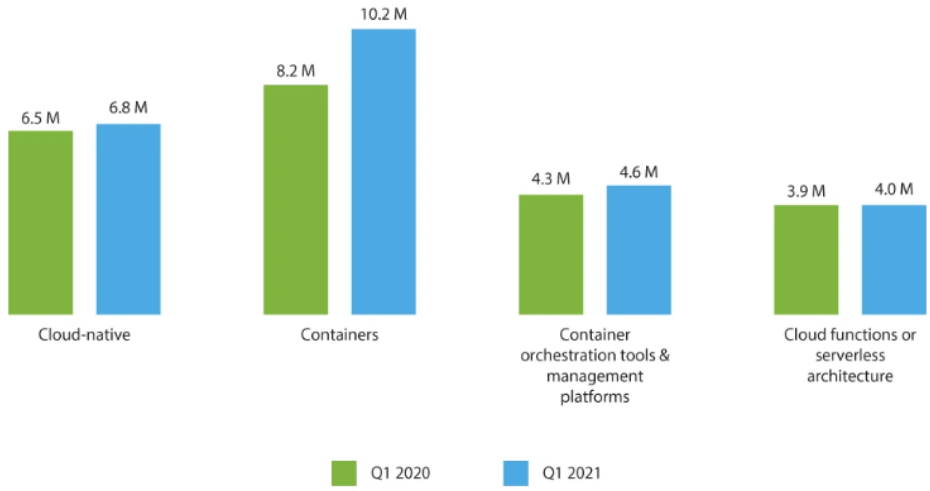

Fig : AI Pipelines of Leveraging Cloud-Native Microservices

11.5. Conclusion

After introducing the reasons why a cloud-native infrastructure is appropriate for AI, the usage of the cloud-native paradigm of microservices, containers, and serverless architectures can efficiently deliver flexibility, reusability, scalability, and maintainability for AI pipelines development and deployment. Following the best

practices of the microservices architecture allows for inserting specialized building blocks for each step inside the pipeline. The pipelines, in turn, can be assembled into a DAG workflow in order to define metadata, driving the end-to-end workflow management task, which relies on the service responsible for driver loading and automatic calls to each specific function of every microservice exposed inside the pipeline. Each building block can be triggered whenever required - scheduled, triggered by an event, or invoked on demand - and inside the same pipeline, there can be functions deployed as a notebook, a serverless function, a long-running serverless function, a container running as a job or as a service - either batch or streaming could be implemented through this flexibility. Inside the same container used for AI tasks, other steps could be there - for example, for logging and maintaining metrics, for standard input or output, and so on. Other implementing considerations may regard the GPU provisioning and scaling, performance monitoring, scheduling, and orchestration. The examples are just a few ideas on how to leverage these utilized paradigms to improve AI-related projects development and deployment. In fact, there is a lack of concrete examples in the cloud-native AI solutions currently available on the market.

As a last note, the emerging technologies can widen the user base of AI solutions because they allow for investing less energy and less technical expertise. It is, therefore, worth discussing how cloud-native AI paradigms and these emerging technologies can affect each other. There are many emerging technologies that may enhance cloud-native AI solutions - such as chatbot platforms, development platforms, App Services, Low Code solutions, DBaaS, Function as a Service development, orchestration services, event-driven Cloud architectures.

11.5.1. Emerging Trends

Over the years, cloud-native computing has transformed enterprise IT, and the adoption of open-source microservices-based architectures, combined with Kubernetes, has truly become mainstream. AI development, on the other hand, is still in its early days, with advances in scalable ML and data-centric AI using cloud-native methodologies just beginning to materialize. Here are the initial trends that I see emerging: 1. Cloud-native for MLops: While ML advances have been rapid, there are still observable pain points for organizations deploying and maintaining ML models at scale. Organizations are investing heavily to scale ML for a large number of use cases and are looking to cloud-native and composable architectures to mitigate operational risks. Building internal MLOps platforms and frameworks is on every large organization's strategic roadmap. 2. Data Centers for AI: Building large language models has been hugely resource-intensive, costing hundreds of millions to train a single model. As organizations scale up their investments in AI, there is a significant need for an infrastructure play to build data

centers to support training and inference workloads. Established players and newer companies are architecting and building solutions that will offer AI as a Service. 3. Ecosystem Expansion: Growth spurts often lead to the development of a sprawling ecosystem that presents challenges in finding a cohesive architecture to simplify building and managing AI applications. More and more companies are building companies around cloud-native composable services. The challenge for organizations will be to stitch together these capabilities into something that can be standardized. 4. Fusion AI: AI applications represent the intersection of multiple AI techniques — generative AI, multimodal learning, synthetic data generation, retrieval augmented generation, etc. Over the coming years, a number of these trends will come together as companies stitch together these capabilities into something useful.

References

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 1(3), 81–84.

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In NIPS Workshop on ML Systems.

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44(2), 35–40.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Sato, M., & Takahashi, K. (2020). MLOps: Continuous delivery and automation pipelines in machine learning. In IEEE/ACM International Workshop on Cloud Engineering.