Application Development On Cloud

How cloud is transforming the way we envision & develop applications

Abstract:

- Cloud technologies like platform-as-a-service, evolution in software defined environments, and tooling frameworks have given rise to true realization of the microservices architectural style. Microservices architecture is about developing a single application as a suite of small services, each running in its own process on the network, and communicating with each other via lightweight mechanisms, often an API utilizing the features of the cloud.

- The architectural style promotes agility, improves productivity, resilience and scalability of the application. Though Microservices have become an essential architectural style for white-space development of systems of engagement applications, they are part of the overall enterprise strategy for transforming the technical debt when the application pattern fits the architectural style. These services can be independently deployed and managed; implemented using diverse, polyglot programming.

- This is not suitable for applications that have centralized data models, repositories, and governance – and hence most legacy & systems of record applications will struggle to adopt microservices architectural style.

- Application Programming Interfaces (API) are a set of subroutine definitions, protocols, and tools and are a vital enabler of microservices architecture. They provide the most logical model for building interfaces between the various components of a microservice architecture. Each individual microservice, enabled by APIs, is able to communicate with every other microservice in the architecture as well as with the applications and Web sites they power and the databases from which they draw real-time information, essential to their functioning offer these capabilities inherently.

Building applications through microservices on cloud

The architecture pattern in developing a single application as a suite of small services, each running in its own process and communicating with each other via lightweight mechanisms, often an HTTP resource API is called microservices and is the predominant architecture for applications on cloud. These services are logically independent, built by multiple teams, focusing on User Experience & business capabilities & typically experience automated, multiple deployments/day.

Traditional application architecture comprises of n-tiers / layers typically distributed amongst Presentation, Business Logic and Data Persistence. These layers are coded using the same programming language, at times compromising the User Experience. This architecture is monolithic, since code changes require entire codebase to be re-built; also, selective scaling of application by business functionality / user experience is not possible. Over a period, the application tends to lose modularity. Over a period, these frustrations resulted in realization of an architectural style that is about building application as a suite of services. These services can be independently deployed and managed; implemented using diverse, polyglot programming. This also promotes de-centralized governance and team ownership. This nature of micro-services make them a natural fit for cloud based development & deployment.

There is no formal definition of the microservices architectural style – but it could be described through certain common characteristics.

Microservices: Key characteristics

- User Experience Driven, Fine Grained Services that do “one thing well”
- Polygot by definition
- Cross-functional Teams
- Runs on the network
- Communicate using lightweight mechanisms (messaging with services like Kafka, direct calls with JSON)
- Decentralized governance
- Data coupled w/Microservice
- Rapid Provisioning, Infrastructure automation & Integrated DevOps
- Design for failure – Cloud as a key enabler
- Scalability and elasticity
- Security

A microservices component is always defined as a unit of software that is independently replaceable & upgradeable. Microservices have their own User Interface, business logic and persistence implementations – essentially a self-contained business function, and has clear non-functional specifications. Examples are make a payment, complete booking of a car etc. This is not suitable for applications that have centralized data models, repositories, and governance – and hence most legacy & systems of record applications will struggle to adopt microservices architectural style. This helps create independent & stateless services interacting with each other OR with other Systems of Record via lightweight API calls. E.g microservices interact with Mainframe based applications via an adaptor exposing web service. To support better user experience; architecture layers in a micro-service can be implemented using different programming languages – mostly leveraging just in time scripting languages such as JavaScript, python etc. as opposed to compiled languages that hinder agility.

User Experience Driven; business capability based functional componentization and polyglot nature helps in creating self- contained, independent executables built, deployed and instrumented using easily integrate-able DevOps tools. These are developed by multiple independent but collaborative set of squads. These squads follow their own release cycles, often releasing multiple times in a day. For example, an online shopping website has close to 450 micro-services, roughly 3 per developer.

Microservices based approach enables a continuum wherein same team builds as well as runs & supports it on production. This brings developers closer to business and enables them to think how can the software enhance the business capabilities – which brings the “product” thinking.

Traditional integration approaches infuse significant smartness in communication mechanism itself (e.g. Enterprise Services Bus etc.). Microservices based applications tend to be decoupled and as cohesive as possible.

Traditional monolithic applications need centralized governance on how components are built etc. Polygot approach helps decentralize development and teams could chose appropriate frameworks / tools to develop the services. Microservices frameworks & PaaS platform services including DevOps help decentralize the governance. This also helps decentralize data storage decisions and makes data model design simpler. This gives rise to transaction-less co-ordination across services and problems are dealt with compensating transactions. This is the way businesses manage their processes – ensure focus on quick responses with ability to fix mistakes with quick reversals.

PaaS platforms, Containerization, and Software defined environments are central to automation and right level of instrumentation. This reduces complexity of building, testing (automated), deploying and operating microservices through automating the entire process. These instrumentations also include failure detection, restoration of services through sophisticated monitoring, event management and cloud technologies offer these capabilities inherently.

Individual function scalability is at the core of micro-services architecture. This, in turn requires elastic infrastructure that can scale up/down, preferably in a horizontal manner to meet the application demands. Micro Services often intersect the boundary between hybrid IT Landscape covering on-premise, private, public cloud and Platform/Software As a service. This requires support of newer security mechanisms like OAuth, SAML and cloud based authentication/authorization.

APIs and Microservices

Microservices are business focused and APIs are a manifestation of how the functionality is exposed to the consumer. In the context of cloud & modern web architectures, APIs are interfaces implemented as REST (JSON/HTTP) typically. APIs are often times an implementation of Microservices consumed by a system of engagement application. APIs are less about reuse and more about consumption and monetization. APIs can also be implemented as web services or other mechanisms. APIs have both intra-enterprise (private) and inter-enterprise (public or partner) use.

Microservices-based applications access individual services through an API gateway/fabric that is the single entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

As an example, Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the AWS Management Console, API can be created that acts as a “front door” for applications to access data, business logic, or functionality from your back-end services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, or any Web application. Amazon API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

Microservices Development Skills

The culture and organization of the teams to operate in a cross-functional, independently executing, very agile and DevOps oriented, is critical for effective conceptualization, design and implementation and effective leverage of microservices. Core to microservices development is the DevOps & Infrastructure automation whereas the services (components) could be built in any of the popular technologies (java, .net, python, node.js etc.). So, every microservices developer should be familiar with the Microservices chassis (and associated technologies). Core development language skills alone does not serve much purpose as the build squads essentially not only writes code, but also builds the failure recovery, monitoring, logging and all that capabilities into the microservices they develop. Testing is automated and built into the development framework through right kind of automation toolsets (chai / mocha / selenium are all relevant toolset examples here).

Microservices development is closely linked with DevOps and Microservices chassis as well as the underlying infrastructure automation frameworks. It is critical for a team that is working on microservices to understand the Cloud concepts, PaaS frameworks, 12-factor design principles, Core microservices chassis related technologies & Services etc. A typical microservices squad comprises of Solution architects, cloud platform architects, microservices specialists, Hybrid cloud platform / PaaS platform specialists, DevOps consultants and not but the least – Agile consultants.

Key examples are

- A major bank is building microservices in a private cloud based on Kubernetes & Docker leveraging Spring Cloud based chassis. This integrates with system of record applications through Enterprise Services Bus based middleware.
- A major telco is establishing an on- premise cloudfoundry based fabric to standardize and streamline microservices based development.
- A major manufacturer of lifts is building microservices in Bluemix containers that interface with their ERP systems as well as IBM’s IoT solution that exposes APIs for large real-estate companies to build integrated control solutions for their townships / campuses.

Summary

Microservices are foundational to application development on cloud and are core to realization of full set of cloud capabilities.

Microservices architectural style helps develop and deploy services independently – but the underlying tooling framework (DevOps) enables the agility, componentization, de-centralization and promote the design for failure approaches.
Microservices development needs a suite of technology frameworks that enable realization of the architectural

style. The frameworks that help build microservices come with necessary libraries & tooling. Underlying aspects of these frameworks are cloud foundry based environments, Docker or container based environments or virtualized environments with significant level of software defined environment aspects.

Hence, leveraging cloud concepts & frameworks is not only essential to development of microservices, but is an imperative.

XXX00000-USEN-00