Introduction to Distributed Services, REST, and Web Services

Introduktion till Distribuerade Tjänster, REST och Web Services

Hadi Saghir

Computer and Information Science

Web services and Distributed Systems

2022

Language: Swedish (Introduction in English)

Introduction

Overview

This article ventures into a meticulous examination and juxtaposition of diverse methodologies and paradigms within the spheres of web services, distributed architectures, and digital interfaces, colloquially known as web APIs. Its essence is to cultivate a profound comprehension by delicately unraveling these multifaceted concepts, particularly their synergy with the initiative of democratizing access to governmental data repositories.

The foundation of distributed systems

Grasping the quintessence of distributed frameworks, accentuated by the subtleties inherent in SOA, microservices, Web Services, RESTful constructs, and Web APIs, is paramount for an array of reasons:

- **Interoperability:** Adeptness in these realms fosters the creation of systems endowed with the ability to seamlessly synergize, transcending the confines of disparate programming dialects and infrastructural paradigms. This is indispensable in the governmental context, where a mosaic of systems must coalesce to dispense unified and accessible public utilities.

- **Scalability and Malleability:** Insight into distributed architectures paves the way for the engineering of scalable and malleable frameworks, poised to evolve in tandem with the dynamic landscape of technological advancements and policy shifts. This plasticity is crucial for governmental apparatuses tasked with accommodating the exponential proliferation of data and mutable policy requisites.

- **Operational Efficiency and Efficacy:** Profound understanding in these domains can catalyze the optimization of resource utilization and the enhancement of service and application efficacy. Systems engineered for efficiency can adeptly manage substantial loads and deliver services with alacrity, aligning with the anticipations of the citizenry and the commercial sector.

- **Innovative Integration:** Acquaintance with the principles of distributed systems kindles innovation and eases the integration of novel services and technologies, nurturing the proliferation of digital ecosystems. For governmental entities, this translates to the capability to proffer more sophisticated services and better fulfill the populace's requisites.

The paradigm shift towards distributed architectures marks a departure from the erstwhile, monolithic structures, where system components were inextricably intertwined. Distributed systems, by contrast, are an ensemble of autonomous units that communicate through a network. This architectural evolution is pivotal for several reasons:

- **Resilience:** Distributed frameworks are synonymous with enhanced reliability and availability. The failure of an individual component doesn't precipitate systemic collapse, ensuring uninterrupted service availability, a non-negotiable for governmental services.

- **Modularity:** The inherent modularity of distributed systems means that components can be independently developed, deployed, and scaled. This attribute facilitates seamless updates and maintenance, empowering governmental agencies to swiftly navigate new directives or technological paradigms.

- **Collaborative Data Stewardship:** The ethos of distributed systems advocates for a more open and collaborative stewardship of data and services. By dismantling data silos and enhancing data accessibility, governments can catalyze innovation and elevate public service quality.

- **Cost Efficacy:** Distributed systems can enhance resource efficiency and obviate the need for redundant infrastructural investments, potentially accruing cost savings for governmental bodies and enabling more judicious resource allocation.

SOA, webbtjänster och Web Services

Det finns några begrepp som är viktiga att kunna för att läsa vidare: tjänsteorienterad arkitektur (SOA), webbtjänster och Web Services.

Webbtjänst

En tjänst är självständigt utförande av ett väldefinierat arbete av en till en annan. En webbtjänst är följaktligen en tjänst som erbjuds på webben, dvs ett datorprogram som identifieras med en URI. Till exempel är Spotify en tjänst för digital musik och poddar. En webbtjänst stödjer direkt interaktion med andra agenter på webben med hjälp av XML-baserade meddelanden som utbyts via Internetbaserade protokoll.

SOA

SOA är ett paradigm som främjar att systemarkitektur byggs av distribuerade tjänster, istället för en monolitisk design där alla tjänster är samlade på ett ställe [1]. SOA -strukturen är baserad på begreppet "loose koppling" där de olika applikations komponenter delas upp i distinkta moduler vilket erbjuder mer flexibla , enklare applikationer. Till exempel, ett modul kan uppgraderas och det inte skulle påverka andra tjänster så länge de inte ändrar gränssnittet och kontraktet (beskrivs i nästa stycke)

SOA-tjänster kan därmed bygga komplexa tjänster och applikationer med hjälp av ett robust kommunikations- och kontroll lager som kallas en Enterprise Service Bus (ESB) [1]. SOA använder ett gränssnittet för att beskriver hur tjänsteleverantören kommer att hantera förfrågningar från konsumenten och ett kontraktet som definierar villkoren för interaktion mellan leverantören och service konsumenten.

Web Services

Web Services är system som stödjer interoperabel interaktion genom att standardiserar kommunikation och format på gränssnitt som används för att specificera tjänster och tjänstekontrakt [1,2]. Med andra ord är Web Services ett sätt att integrera webbaserade applikationer med hjälp av SOAP, WSDL och UDDI öppna standarder över Internet Protocol.

Web Services tillåter arkitekturen av distribuerade systemet att inte förlita sig på ett programmeringsspråk och en plattform. Web Services för med sig användandet av interoperabilitet mellan heterogena tjänster –mjukvara skriven i olika programspråk på olika platform kan använda varandras tjänster och data.

Web Services använder den institutionaliserade branschstandarden för korrespondensen. Service transport, XML-meddelanden, Service Beskrivning och Service Discovery Layers använder väl-definierad protokoll [2]. Standardisering av protokollen ger verksamheten många fördelar såsom ett bredare utbud av tjänster, mindre kostnader för implementation, främja konkurrens och bättre kvalitet av tjänster [1].

Funktionaliteten hos en applikation kan ges åtkomst för fler klienter. Till exempel kan både tunna och rika klienter ta del av fördelarna med Web Services operationer. Tunna klienter(Thin-client) är en dator som körs från resurser lagrade på en central server.! Rika klienter(Rich-client) är en dator som kan tillföra funktionaliteter oberoende av en central server.

SOA contra mikrotjänst (microservices)

Mikrotjänster, även känd som Microservice Architecture, är en arkitektur där applikationer är uppbyggda av en samling mindre, autonoma tjänster där var och en har sin egen funktion [3]. Medan mikrotjänster och tjänsteorienterad arkitektur är lika på vissa sätt, kommer de viktigaste skillnaderna i deras funktionalitet. Typen av funktionalitet tjänster behöver bestämmer nämligen vilken arkitektur passar bättre.

SOA lämpar sig bättre i stora, komplexa distribuerade system. Däremot lämpar mikrotjänster sig där vi har en väldefinierade processflöden [3]. Detta är eftersom SOA bygger på tjänster eller moduler som återanvänds i ett ställe där resurser och data delas. Däremot bygger mikrotjänster på enskilda tjänster

som fungerar självständigt. Enligt [3] ger SOA följaktligen flexibiliteten för att kunna bygga de stora, komplexa distribuerade system medan mikrotjänster är en enkel arkitektur.

Feature	SOA	microservices
Kommunikation	ESB	API
Coupling	Loose Coupling	Bounded context
Data styrning	Gemensam data styrning i distribuerade system (kontrakt)	Ingen gemensam data styrning i distribuerade system
Interoperability	Stödjer flertal meddelande protokoll såsom SOAP	Använder enklare meddelande protokoll som HTTP och REST

Tab. 1

SOAP, WSDL & UDDI

SOAP definierar det enhetlig formatet för att skicka och ta emot XML-kodad data. SOAP tillåter kommunikation mellan applikationer på olika operativsystem och programspråk [1]. SOAP kan dock ha onödigt höga prestanda kostnader för exempelvis ett enkelt sträng meddelande då SOAP endast använder sig av XML-formatet.

(Web Services Definition Language) WSDL skapades för att beskriver och definierar slutpunkt, operationer, request- och svarsformat. WSDL tillåter tjänsteleverantörer att specificera vad en webbtjänst kan gör, var den finns och hur man anropar den och det tillåter klienten att kunna skicka specifika förfrågan. WSDL-element beskriver data och operationer som ska vara utförs på den [1].

Universal Description, Discovery and Integration (UDDI) tillhandahåller en katalog med gränssnitt som är definierat som SOAP för klienter att dynamiskt hitta andra webbtjänster. Den använder WSDL för att ta emot förfrågningar och ändringar. Det finns globala UDDI-förråd (www.uddi.org). UDDI är en standard för att hantera Web Services [1].

Den andra är en specifik uppsättning av globalt replikerade register [1]. UDDI använder sig vanliga taxonomier så att information kan upptäckas baserat på kategorisering och är inte knutet till en specifik teknik. UDDI kan innehålla vilken resurs som helst, vilket samtidigt gör UDDI sårbart för felaktiga och utdaterade register.

Hur hänger SOAP, WSDL & UDDI ihop?

Dessa tre begrepp är som tidigare nämnt fundamentala byggstenar för Web Services. Det viktigaste skillnad är deras funktion: definiera (SOAP), beskriva (WSDL) och tillhandahålla (UDDI).

En service provider måste skicka ett WSDL-meddelande som beskriver webbtjänstens gränssnitt till server register/broker (UDDI) för att sparas där i registren. När en service requester skickar ett SOAP-meddelande på tjänsteförfrågan, innehåller det ett WSDL som skickas till UDDI. UDDI skickar tillbaks en WSDL som definierar hur service requester bör kommunicerar med service provider, dvs. webbtjänst gränssnittet [1, 2].

WSDL beskriver nämligen datatyperna och strukturerna för webbtjänster och berättar hur man mappar de i de meddelanden som utbytts. UDDI inkluderar ett XML-schema för SOAP-meddelanden som definierar en uppsättning av dokument som innehåller information om företag och tillhörande tjänster. Service requester och service provider kan, efter de har etablerat hur kommunikation bör ske, skicka fram och tillbaka SOAP-meddelandet mellan varandra.

Restful

REpresentational State Transfer (REST) är en arkitektur för standarder som underlättar och effektiviserar kommunikationen av olika system med varandra. REST är nämligen ett paradigm för att komma åt resurser där RESTful-system kännetecknas av hur de tillhandahåller sina resurser som tillståndslösa (stateless) och språk oberoende (language independent) vilket hjälper att skilja mellan klient och server. Resurser kan tillhandahållas i flera olika format, till exempel HTML, XML och JSON [4, 5]. En RESTful webbtjänst är därmed en lätt, underhållbar och skalbar tjänst som använder sig av en API för att på ett säkert, enhetligt, tillståndslöst sätt för att överföra data med andra (SOA använder ESB och mikrotjänster använder API) [6]. REST modellen liknar MVC-modellen där controller är RESTful API, M är servern och V är klienten. M och V inte känner till varandra men skicka över information i strängar (motsvarande av XML). Information sparas på var och en sida [5].

REST arkitektur bygger på alla eller några nedanstående begränsningar för kunna kännetecknas som RESTful eller delvis RESTful. Variation av de olika begränsningar ger webbtjänsten önskvärda icke-funktionella egenskaper: prestanda, skalbarhet, enkelhet, flexibilitet, interoperabilitet och tillförlitlighet. Följaktligen är alla varianter av RESTful modellen inte exakt likvärdiga [4]. De begränsningar är följande:

Uniform interface

Individuella resurser identifieras i förfrågningar och manipulation av resurser genom representation. Klienten har representation av resursen och den innehåller tillräckligt med information för att ändra eller ta bort resursen på servern. Servern och klienten skickar självbeskrivande meddelanden för att beskriva hur meddelandet ska behandlas så att servern enkelt kan analysera begäran. Meddelande inkludera länkar i varje svar så att klienten lätt kan upptäcka andra resurser, dvs. Hypermedia as the Engine of Application State (HATEOAS) [5]

klient-server arkitektur

Servern måste ha en RESTful web tjänst som tillhandahåller den nödvändiga funktionaliteten till klienten [4]. Klienten skickar en begäran till webbtjänsten på servern och servern skulle antingen avvisa begäran eller följa och ge ett adekvat svar till klienten. Inget beroende mellan servern och klienten möjliggör för applikationer att utvecklas oberoende, dvs. flexibilitet [4,5]. Till exempel kan man dela på backend och frontend eller mellan olika tjänster och ändringar av den ena inte påverkar implementeringen av den andra. Detta hjälper till exempel att implementera nya funktioner som en algoritm för att rekommendera filmer och utvecklarna behöver inte tänka på hur den visas.

Cache

Servern och klienten har därmed inte någon pågående session för att kommunicera utan sparas data i klient-sidan [4]. Detta innebär att det är upp till klienten att se till att all nödvändig information tillhandahålls till servern Detta krävs så att servern kan behandla svaret på rätt sätt. Servern ska inte upprätthålla någon form av information mellan förfrågningar från klienten. Det är en mycket enkel oberoende fråga-svar-sekvens [4].

Layered system

Ett system består av fler lager som fungera oberoende och endast interagera med de lager som är direkt kopplade till det. Denna strategi gör det möjligt att skicka förfrågan utan att kringgå andra lager. Om en proxy eller lastbalanserare placeras mellan klienten och servern, kommer det inte att påverka deras kommunikation, och det kommer inte att finnas något behov av att uppdatera klient- eller serverkoden. Mellanliggande servrar kan förbättra systemets skalbarhet genom att möjliggöra lastbalansering och genom att tillhandahålla delade cachar. På så sätt kan de inkommande förfrågningarna levereras till lämplig serverinstans. I detta fall behöver klientsidan inte förstå hur servern kommer att fungera; den gör bara förfrågningar till samma URI. Säkerhet ökar men underhållbarhet och flexibilitet minskar eftersom fler lager innebär säkerhetsåtgärd kan tillämpas och upprepas i flera ställe men underhållsarbete innebär att ändra fler lager eller byta ut hela proxyn.

Webb-API

Webb-API:er och Web Services låter som samma sak eftersom båda är ett sätt för två datorer att kommunicera med varandra över internet och därmed bygger på XML och HTTP som är grundstenar för webbs kommunikation protokoll [2, 4]. Den största skillnaden är att en Web Services är en nätverksbaserad resurs som tillhandahåller maskin-till-maskin-interaktion genom gränssnitt, medan ett API ger applikation-till-applikation-interaktion genom att tillhandahålla data och funktionalitet [1]. API definierar det protokollet för kommunikation mellan applikationerna medan Web Services omfattar ytterligare delar såsom ovannämnda delar SOAP, WSDL och UDDI för att definiera nätverksbaserad kommunikation mellan maskinerna som kör applikationerna och därför [1] skriver:

“All Web Services are APIs, but not all APIs are Web Services”

REST och Web Services

Web 2.0 API:er använder oftast paradigmen REST och SOAP. REST-API:er använder HTTP-metoder för att komma åt resurser via URL-kodade parametrar och använder JSON eller XML för att överföra data. Däremot standardiseras SOAP-protokoll av W3C och kräver användning av XML. REST definierar ett arkitektur ansats (statslös och språkoberoende) medan SOAP utgör en begränsning av formatet för XML-överföring av data mellan requester och provider. SOAP och REST är därmed inte direkt jämförbara men var har sina för- och nackdelar. REST-API:er (REST) kan används tillsammans med SOAP-API:er (Web Services) för att skapa RESTful Web Services

[6] listar skillnader webb-API kan till skillnad ha REST-API:er har olika protokoll och format. Protokoll av en webb-API är HTTP/s protokoll och URL protokoller. Den enda information som klientprogrammet bör känna till är URI: för att begära resurser. Ett API ger applikation-till-applikation-interaktion genom att tillhandahålla data och funktionalitet över localhost, nätverk eller webben. Webb-API:er kan stödja andra format på förfrågningar och svar än XML, REST och SOAP. Ett webb API kan nås med HTTP-protokoll [6]. I samband med webbutveckling definieras ett applikationsprogrammeringsgränssnitt vanligtvis som en uppsättning standarder, till exempel HTTP-begärande meddelanden och en definition av strukturen för svarsmeddelanden, vanligtvis i XML eller JSON [6].

REST API kan används för att skicka förfrågningar från en konsument som kommer antingen att accepteras eller avvisas av servern. På samma sätt bör ett serverprogram skicka klientprogram till begärda data via HTTP utan några ändringar. [7] skriver en utförlig beskrivning av hur Twitters REST API ger åtkomst med tillämpning av HTTP-protokoll för att läsa och skriva data med hjälp; av vilka man kan integrera till exempel i sin egen applikation.

Ett REST är en standardiserad arkitekturstil för att skapa API:er för webbaserade applikationer [4, 8]. I denna arkitekturstil använder klienter och servrar oberoende av varandra och överför information med hjälp av ett standardiserat gränssnitt [4, 8].

Web Services använder SOAP-API:er som bygger på XML-baserade system, vilket innebär att mängden data i sig är större, vilket i sin tur innebär högre belastning och kostnad att bygga anpassade servrar för att hantera belastningen. En Web Service som körs på en dator lyssnar efter förfrågningar från andra datorer. När en begäran, beskriven i ett SOAP-meddelande skrivet i XML, från en annan dator tas emot, via ett nätverk, returnerar webbtjänsten de begärda resurserna. Denna resurs kan vara JSON, XML eller filer som text, bild eller ljud.

Ett RESTful Web Service behöver därmed förlita sig på SOAP men det innebär relativt tightare-coupling mellan klient och server, användning av standard SOAP-bibliotek, XML och uppmärksamhet på SOAP-standarden, medan REST fokuserar på HTTP-transport, loose-coupling och representation av statslösa resurser.

När Web Service inte behöver hantera informationens status för en begäran till en annan, kan man använda REST. Anledningen är att SOAP är mer lämpad för informationsflöde som till exempel behöver e-handel hålla reda på sina kunders kundvagn för att överföra varorna till betalningssidan. Utvecklarna måste använda REST om det behövs för att cachelagra många förfrågningar. Det låter användare använda samma resurser flera gånger. REST-tjänster förenklar kodningen. Så det gör implementeringen av data snabbare än SOAP.

Referenslista

[1] N. M. Josuttis, SOA in practice. O'Reilly, 2007.

[2] “What are web services?,” Tutorials Point. [Online]. Tillgänglig: https://www.tutorialspoint.com/webservices/what_are_web_services.htm. [Hämtad: 15-Dec-2022].

[3] “SOA vs microservices: What's the difference?: CrowdStrike,” crowdstrike.com, 27-Sep-2022. [Online]. Tillgänglig: https://www.crowdstrike.com/cybersecurity-101/cloud-security/soa-vs-microservices/#:~:text=Microservices%20vs%20SOA%3A%20Identifying%20the%20Differences&text=In%20an%20SOA%20model%2C%20services,microservices%20has%20an%20application%20scope. [Hämtad: 15-Dec-2022].

[4] A. Walker, “RESTful web services tutorial: What is REST API with example,” Guru99, 22-Oct-2022. [Online]. Tillgänglig: https://www.guru99.com/restful-web-services.html. [Hämtad: 15-Dec-2022].

[5] J. Webber, S. Parastatidis, and I. Robinson, Rest in practice. Sebastopol, CA: O'Reilly, 2010.

[6] D. A. N. I. E. L. JACOBSON, Apis: A strategy guide. SEBASTOPOL, CA: EVOLVED MEDIA, 2012.

[7] Kulkarni , A. et al. (2022) Web api vs REST API simplified: 4 critical differences, Hevo. Hevo. Tillgänglig: https://hevodata.com/learn/web-api-vs-rest-api/ (Hämtad: December 17, 2022).

[8] P. Yadav, “Understand restful API design with Twitter API as an example,” OpenGenus IQ: Computing Expertise & Legacy, 01-Sep-2019. [Online]. Tillgänglig: https://iq.opengenus.org/restful-api-design/. [Hämtad: 15-Dec-202

Bilaga 1

Du arbetar på en myndighet som ska öppna upp sina data och får i uppdrag att beskriva webbtjänster, distribuerade system och webb-APIer för dina kolleger. Dina kollegor har grundläggande kunskaper om saker som webben, HTML och närliggande begrepp. Din artikel ska därför beskriva, utreda och jämföra olika tekniker inom området. Mer specifikt ska du besvara följande frågor:

Många talar om SOA, webbtjänster och Web Services som om de vore samma sak. Är det så?

– Definiera ovan nämnda begrepp på ett enkelt vis.

– Vad gör dem önskvärda? Vilka är de viktigaste principerna?

– Hur förhåller sig microservices till SOA?

Under ditt utredningsarbete stöter du på ett antal begrepp och förkortningar. Beskriv dem.

– Du stöter på begreppen SOAP, WSDL och UDDI. Vad betyder och innebär dessa begrepp?

Vilka problem löser de? Vilka problem har teknikerna? Varför nämns teknikerna tillsammans och hur samverkar det?

– På annat håll stöter du på begreppet REST som ofta beskrivs som ett bättre sätt att realisera en distribuerad arkitektur. Vad är REST? Varför tycker många att REST är enkelt? Är alla varianter av REST likvärdiga?

– Är de två alternativen (REST visavi Web Services) likvärdiga? Konkurrerar de? Löser de samma problem? Vad är några typiska användningsområden för de båda?

– Hur kommer begreppet webb-API in i bilden?