Jurnal Internasional Penelitian Ilmiah di bidang Ilmu Komputer, Teknik, dan Teknologi Informasi

ISSN: 2456-3307 OPEN ACCESS

Tersedia daring di: www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT251112144

Memahami Arsitektur Microservices: Sebuah Tinjauan Komprehensif

Memandu

Surya Prabha Busi

Ford Motor Credit Company, AS

INFORMASI ARTIKEL

Riwayat Artikel:

Diterima: 28 Januari 2025

Diterbitkan: 31 Januari 2025

Edisi Publikasi

Volume 11, Edisi 1

Januari-Februari-2025

Nomor Halaman

Tahun 1440-1447

ABSTRAK

Arsitektur microservices telah muncul sebagai pendekatan transformatif untuk membangun sistem perangkat lunak yang kompleks, menawarkan organisasi cara untuk mengembangkan dan memelihara aplikasi skala besar secara lebih efektif. Artikel komprehensif ini mengeksplorasi konsep-konsep fundamental, manfaat, strategi implementasi, dan tantangan arsitektur microservices. Artikel ini mengkaji bagaimana microservices memungkinkan

Artikel ini membahas bagaimana organisasi dapat memanfaatkan microservices untuk memecah aplikasi kompleks menjadi layanan independen yang mudah dikelola dan berkomunikasi melalui API yang terdefinisi dengan baik. Artikel ini mengupas prinsip-prinsip arsitektur yang memandu implementasi microservices yang sukses, termasuk desain layanan, persyaratan infrastruktur, dan pola komunikasi. Artikel ini mencakup aspek-aspek penting seperti ketangkasan teknis, kemampuan penyebaran independen, dan manfaat skalabilitas, sekaligus mengatasi tantangan kritis seperti kompleksitas sistem terdistribusi, konsistensi data, dan definisi batasan layanan. Melalui analisis mendetail tentang praktik industri dan pola arsitektur, artikel ini memberikan wawasan tentang bagaimana organisasi dapat memanfaatkan microservices untuk membangun sistem yang lebih tangguh, mudah dipelihara, dan skalabel.

sekaligus mendorong otonomi tim dan fleksibilitas teknologi.

Kata kunci: Arsitektur Sistem Terdistribusi, Desain Berorientasi Layanan, Pengembangan Cloud-Native, Skalabilitas Sistem, Pola Layanan Mikro

Perkenalan

Dalam lanskap arsitektur perangkat lunak yang terus berkembang, microservices telah muncul sebagai pola yang ampuh untuk mengatasi tantangan membangun aplikasi yang kompleks dan terukur. Gaya arsitektur microservices mewakili cara khusus dalam mendesain aplikasi perangkat lunak sebagai rangkaian layanan yang dapat diimplementasikan secara independen [1]. Setiap layanan dalam arsitektur ini menjalankan prosesnya sendiri dan berkomunikasi melalui mekanisme yang terdefinisi dengan baik dan ringan, biasanya antarmuka pemrograman aplikasi (API) berbasis HTTP.

Transisi menuju arsitektur microservices adalah

Pada dasarnya berakar pada kebutuhan untuk memecah aplikasi kompleks menjadi komponen yang lebih mudah dikelola dan independen. Penelitian dan pengalaman industri menunjukkan bahwa pendekatan arsitektur ini muncul dari pengalaman nyata organisasi yang berurusan dengan basis kode monolitik besar yang semakin sulit untuk dipelihara dan dikembangkan [2]. Pola ini mendapatkan perhatian yang signifikan melalui implementasinya yang sukses di perusahaan seperti Amazon, Netflix, dan The Guardian, yang telah berbagi pengalaman mereka dalam memecah aplikasi monolitik menjadi layanan yang lebih kecil dan lebih mudah dikelola.

Salah satu prinsip utama yang disoroti dalam penelitian arsitektur kontemporer adalah aspek organisasi dari microservices, di mana layanan dibangun berdasarkan kemampuan bisnis [1]. Pendekatan ini secara alami mengarah pada tim lintas fungsi yang menangani siklus hidup pengembangan penuh setiap layanan, dari pengembangan hingga penerapan produksi. Keselarasan organisasi ini, yang dikenal sebagai Hukum Conway, telah terbukti menjadi faktor penting dalam

Keberhasilan implementasi microservices, karena memungkinkan tim untuk mengembangkan, menyebarkan, dan menskalakan layanan mereka secara independen.

Studi industri telah menunjukkan bagaimana arsitektur microservices memungkinkan tim untuk memilih

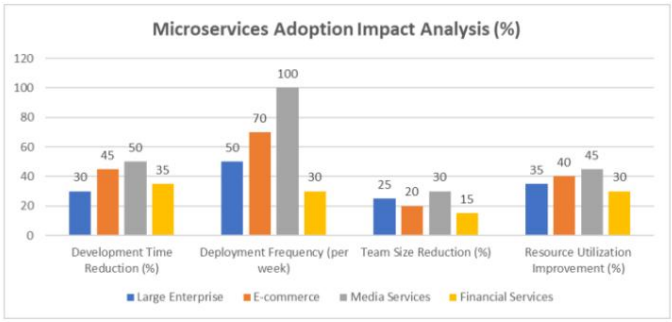
alat yang tepat untuk setiap pekerjaan [2]. Fleksibilitas teknologi ini memungkinkan organisasi untuk mengadopsi bahasa pemrograman, basis data, dan kerangka kerja yang berbeda untuk layanan yang berbeda berdasarkan kebutuhan spesifik mereka. Misalnya, layanan yang menangani analitik waktu nyata mungkin menggunakan tumpukan teknologi yang berbeda dibandingkan dengan layanan yang mengelola otentikasi pengguna, sehingga memungkinkan kinerja optimal untuk setiap penggunaan spesifik.

kasus.

Perjalanan menuju adopsi microservices mencerminkan pergeseran industri yang lebih luas menuju sistem terdistribusi dan arsitektur cloud-native. Arsitektur ini

Pendekatan ini telah terbukti sangat berharga bagi organisasi yang menangani operasi skala besar, sebagaimana dibuktikan oleh dokumentasi dan studi kasus yang ekstensif [1, 2]. Kemampuan penyediaan, penyebaran, dan penskalaan yang cepat yang melekat pada arsitektur microservices telah menjadi penting bagi organisasi modern.

pengembangan perangkat lunak, khususnya di lingkungan cloud di mana sumber daya dapat dialokasikan dan dikelola secara dinamis.



Gambar 1: Metrik Transisi Industri dari Monolitik ke Arsitektur Layanan Mikro [1, 2]

Apa Itu Microservices?

Arsitektur microservices mewakili gaya arsitektur yang menyusun aplikasi sebagai kumpulan layanan yang terhubung secara longgar, masing-masing mengimplementasikan kemampuan bisnis tertentu. Menurut yang telah ditetapkan layanan mikro pola dokumentasi, ini Arsitektur ini muncul sebagai respons terhadap keterbatasan aplikasi monolitik tradisional, khususnya dalam konteks aplikasi perusahaan berskala besar dan kompleks [3]. Pola ini mendefinisikan microservices sebagai layanan kecil dan otonom yang bekerja bersama, memodelkan cara mendesain aplikasi perangkat lunak sebagai rangkaian komponen yang dapat diimplementasikan secara independen.

Arsitektur referensi mendefinisikan karakteristik struktural utama yang membedakan microservices dari pendekatan arsitektur lainnya [4]. Setiap layanan memelihara penyimpanan datanya sendiri, memastikan independensi data dan memungkinkan layanan untuk memilih jenis basis data yang paling tepat untuk kebutuhan spesifik mereka. Pola basis data per layanan ini, seperti yang dijelaskan dalam pedoman arsitektur, memungkinkan layanan untuk mempertahankan kopling longgar sambil memastikan konsistensi data dalam batas layanan.

Komunikasi Arsitektur di dalam layanan mikro pola mengikuti protokol dan standar tertentu.
Katalog pola microservices menekankan bahwa layanan biasanya berkomunikasi menggunakan protokol sinkron seperti HTTP/REST atau pesan asinkron [3]. Layanan-layanan ini diorganisasikan di sekitar kemampuan bisnis, dengan batasan layanan yang eksplisit dan seringkali sesuai dengan sumber daya REST. Dokumentasi pola secara khusus menguraikan bagaimana layanan harus bersifat stateless, sehingga memungkinkan layanan tersebut untuk diskalakan secara horizontal seiring meningkatnya permintaan.
Arsitektur referensi menekankan pentingnya penemuan layanan dan pola gerbang API dalam implementasi layanan mikro [4]. Gerbang API berfungsi sebagai titik masuk tunggal untuk semua klien, dengan kemampuan untuk menangani masalah lintas fungsional seperti otentikasi, penghentian SSL, dan

Pembatasan laju (rate limiting). Komponen arsitektur ini mengarahkan permintaan ke layanan mikro yang sesuai, menggabungkan hasilnya sesuai kebutuhan. Setiap layanan mendaftarkan dirinya ke registri layanan, yang memelihara basis data
Instansi layanan yang tersedia.

Aspek fundamental yang diuraikan dalam dokumentasi pola adalah strategi penyebaran [3].
Arsitektur microservices memanfaatkan alat deployment otomatis, dengan setiap layanan memiliki pipeline continuous delivery-nya sendiri. Pendekatan ini memungkinkan layanan untuk di-deploy secara independen, karakteristik utama yang membedakannya dari aplikasi monolitik. Arsitektur ini secara spesifik menjelaskan bagaimana layanan harus dapat di-deploy dan diskalakan secara independen, dengan perubahan pada satu layanan tidak memerlukan perubahan pada layanan lain.

Arsitektur referensi memberikan detail lebih lanjut mengenai hal-hal berikut:

implikasi organisasi dari layanan mikro [4].
Tim disusun berdasarkan layanan, bukan

Lapisan teknis tradisional, dengan setiap tim memikul tanggung jawab penuh atas layanan mereka. Ini mencakup tidak hanya pengembangan, tetapi juga penerapan, pemantauan, dan pemeliharaan. Arsitektur ini secara khusus menekankan bagaimana struktur organisasi ini memungkinkan tim untuk bergerak cepat dan mandiri, membuat keputusan tentang layanan mereka tanpa memerlukan koordinasi dengan tim lain.

Arsitektur Komponen	Fungsi Utama	Pelaksanaan Pola	Komunikasi Metode	Skalabilitas Mendekati
Layanan Inti	Bisnis Logika Pelaksanaan	Otonom Melayani	REST/HTTP	Penskalaan Horizontal
Penyimpanan Data	Manajemen Data	Basis data per- Melayani	Akses Langsung	Mandiri Penskalaan
Gerbang API	Klien Meminta Penanganan	Distribusi Beban SSL/Autentikasi Titik Masuk Tunggal		
Registri Layanan	Penemuan Layanan	Dinamis Pendaftaran	Pencarian Layanan	Pelacakan Instans
Penyebaran Saluran pipa	Pengiriman Berkelanjutan	Otomatis Penyebaran	Alat-alat Saluran Pipa	Mandiri Penyebaran
Struktur Tim	Kepemilikan Layanan	Lintas fungsi Tim	Otonomi Tim Tanggung Jawab Langsung	

Tabel 1: Karakteristik Arsitektur Layanan Mikro dan Pola Implementasinya [3, 4]

Mengapa Memilih Microservices?

Penerapan arsitektur microservices didorong oleh beberapa manfaat menarik yang mengatasi tantangan pengembangan perangkat lunak modern. Menurut panduan arsitektur Microsoft, microservices sangat berharga dalam aplikasi yang kompleks dan terukur di mana komponen aplikasi yang berbeda berkembang dengan kecepatan yang berbeda [5]. Gaya arsitektur ini telah terbukti sangat efektif untuk aplikasi besar yang membutuhkan kecepatan rilis yang tinggi atau aplikasi yang diterapkan pada lingkungan cloud hybrid.

Ketangkasan teknis merupakan keuntungan mendasar dari adopsi microservices. Panduan arsitektur Microsoft Azure menekankan bagaimana microservices memungkinkan tim untuk menggunakan teknologi yang tepat untuk setiap layanan [5]. Fleksibilitas ini memungkinkan organisasi untuk memanfaatkan kerangka kerja dan teknologi penyimpanan data yang berbeda berdasarkan persyaratan layanan tertentu. Sebagai contoh, dokumentasi arsitektur.

Artikel ini menjelaskan bagaimana aplikasi microservices dapat menggunakan basis data relasional untuk katalog produk, basis data graf untuk mesin rekomendasi, dan basis data dokumen untuk event sourcing, yang masing-masing dipilih berdasarkan karakteristik beban kerja tertentu.

Kemampuan penyebaran independen merupakan manfaat penting dari pendekatan microservices. Seperti yang didokumentasikan dalam riset industri O'Reilly, organisasi yang mengadopsi microservices melaporkan peningkatan signifikan dalam frekuensi dan keandalan penyebaran [6]. Riset tersebut secara khusus menyoroti bagaimana microservices memungkinkan tim untuk menyebarkan pembaruan ke layanan individual tanpa memerlukan koordinasi di seluruh aplikasi. Independensi ini mengurangi risiko penyebaran dan memungkinkan tim untuk mempertahankan rilis yang berbeda.

Siklus berdasarkan kebutuhan bisnis.

Skalabilitas dalam arsitektur microservices memberikan keuntungan yang berbeda untuk manajemen sumber daya. Panduan arsitektur Microsoft secara eksplisit merinci bagaimana microservices dapat diskalakan secara independen, memungkinkan organisasi untuk memperluas subsistem jika diperlukan tanpa perlu menskalakan seluruh aplikasi [5]. Pendekatan penskalaan granular ini terbukti sangat berharga di lingkungan cloud, di mana sumber daya dapat

dialokasikan secara lebih efisien berdasarkan kebutuhan spesifik dari setiap layanan.

Otonomi tim muncul sebagai manfaat organisasi utama, dengan penelitian O'Reilly menunjukkan bahwa organisasi microservices yang berhasil menerapkan biasanya mengatur tim mereka di sekitar

kemampuan bisnis daripada lapisan teknis [6].

Penelitian ini menekankan bagaimana keselarasan antara struktur tim dan arsitektur memungkinkan percepatan yang lebih tinggi.

Siklus pengembangan dan akuntabilitas yang lebih jelas. Tim dapat membuat keputusan tentang layanan mereka secara independen, memilih alat dan teknologi yang tepat sambil mempertahankan batasan layanan yang terdefinisi dengan baik melalui API.

Dokumentasi arsitektur Microsoft lebih lanjut

menekankan bagaimana layanan mikro mendukung teknologi modern praktik pengembangan seperti integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD) [5]. Setiap layanan dapat memiliki pipeline CI/CD sendiri, memungkinkan tim untuk menyebarkan pembaruan lebih sering dan dengan keyakinan yang lebih besar. Pendekatan ini selaras dengan temuan dari penelitian O'Reilly, yang mengidentifikasi kemampuan penyebaran otomatis sebagai faktor keberhasilan kritis dalam adopsi microservices [6].

Kategori Manfaat	Keunggulan Utama dan Dampak Bisnis		Pelaksanaan Daerah	Mendukung Teknologi
Ketangkasan Teknis	Tumpukan Teknologi Kebebasan	Solusi yang Dioptimalkan	Melayani Perkembangan	Beberapa Kerangka kerja
Penyebaran Kemerdekaan	Individu Melayani Pembaruan	Pengurangan Risiko	Melepaskan Pengelolaan	Pipeline CI/CD
Skalabilitas Sumber Daya, Penskalaan Independen, Efisiensi Biaya			Awan Infrastruktur	Alat Penskalaan Otomatis
Organisasi Tim	Selaras dengan Bisnis Tim	Struktur Tim Pengembangan yang Lebih Cepat		Manajemen API
Fleksibilitas Basis Data	Spesifik untuk Tujuan Tertentu Penyimpanan	Ditingkatkan Pertunjukan	Manajemen Data	Basis Data Campuran Jenis
Pengiriman Berkelanjutan	Otomatis Penempatan	Lebih tinggi Melepaskan Frekuensi	Praktik DevOps dan Alat Otomatisasi	

Tabel 2: Manfaat Utama dan Dampak Bisnis dari Arsitektur Layanan Mikro [5, 6]

Bagaimana Cara Mengimplementasikan Microservices?

Keberhasilan implementasi arsitektur layanan mikro memerlukan pertimbangan cermat terhadap beberapa aspek kunci yang selaras dengan prinsip dan pola cloud-native. Pendekatan implementasi harus menyeimbangkan otonomi layanan dengan kohesi sistem sambil mengikuti pedoman arsitektur yang telah ditetapkan [7].

4.1. Prinsip Desain Layanan:
Landasan dari microservices yang efektif

Implementasi terletak pada pola arsitektur cloud-native. Menurut pola yang telah ditetapkan, layanan harus dirancang mengikuti prinsip bounded context, di mana setiap layanan mewakili domain bisnis tertentu dengan batasan dan tanggung jawab yang jelas [7].

Pendekatan ini memastikan bahwa

Layanan tetap terfokus dan mudah dipertahankan sementara

Mendukung tujuan bisnis secara keseluruhan. Pola cloud-native secara khusus menekankan bagaimana layanan harus bersifat stateless sebisa mungkin, sehingga memungkinkan skalabilitas dan ketahanan yang lebih baik.

Manajemen data dalam microservices memerlukan pertimbangan cermat terhadap pola persistensi. Pedoman arsitektur cloud-native Red Hat menekankan pentingnya otonomi data dan batasan layanan yang tepat [8]. Setiap microservice harus memelihara penyimpanan datanya sendiri, menerapkan pola database-per-service untuk memastikan kopling longgar. Dokumentasi arsitektur secara khusus menguraikan bagaimana layanan harus berinteraksi melalui API yang terdefinisi dengan baik.

alih-alih berbagi basis data, mempromosikan kemandirian layanan dan enkapsulasi data.

Desain API merupakan aspek penting dalam implementasi microservices. Pola cloud-native menyoroti pentingnya desain API-first, di mana antarmuka diperlakukan sebagai warga negara kelas satu dalam

arsitektur [7]. Dokumentasi pola merekomendasikan penerapan versi API yang konsisten, pemeliharaan dokumentasi yang komprehensif, dan perancangan antarmuka yang dapat berevolusi tanpa merusak klien yang ada. Praktik-praktik ini memastikan bahwa layanan dapat dimodifikasi dan ditingkatkan secara independen sambil menjaga stabilitas sistem.

4.2. Persyaratan Infrastruktur Pola

arsitektur cloud-native menekankan komponen infrastruktur penting untuk keberhasilan penerapan microservices [7]. Mekanisme penemuan layanan memungkinkan lokasi dan komunikasi layanan dinamis di lingkungan kontainer.

Pola-pola tersebut secara khusus merekomendasikan penerapan arsitektur service mesh untuk menangani komunikasi antar layanan. komunikasi, menyediakan fitur-fitur seperti penyeimbangan beban, pemutusan sirkuit, dan penemuan layanan secara langsung.

Pedoman arsitektur microservices Red Hat merinci pentingnya pola gateway API dalam mengelola akses eksternal ke layanan [8]. Gateway berfungsi sebagai titik masuk terpadu untuk klien eksternal, menangani masalah lintas sektor seperti otentikasi, perutean permintaan, dan pembatasan laju. Pola ini sangat penting di lingkungan cloud-native di mana layanan dapat didistribusikan di beberapa klaster atau wilayah.

Observabilitas dalam arsitektur microservices memerlukan solusi pemantauan yang komprehensif. Pola cloud-native menekankan implementasi pemeriksaan kesehatan, pengumpulan metrik, dan pelacakan terdistribusi [7]. Kemampuan ini memungkinkan tim untuk memahami perilaku sistem, mendiagnosis masalah, dan keandalan di lingkungan terdistribusi. Pola-pola tersebut secara khusus mempertahankan layanan merekomendasikan

Menerapkan tiga pilar observabilitas: log, metrik, dan jejak.

4.3. Pola Komunikasi

Komunikasi antar layanan mikro dapat mengikuti

Berbagai pola berdasarkan persyaratan spesifik. Dokumentasi arsitektur cloud-native menguraikan hal tersebut.

Pendekatan komunikasi sinkron dan asinkron [8]. API RESTful tetap menjadi pilihan umum untuk komunikasi sinkron, sementara gRPC menawarkan manfaat kinerja untuk komunikasi layanan internal. Pedoman arsitektur menekankan bagaimana memilih pola komunikasi yang tepat berdampak pada keandalan dan kinerja sistem.

Pola berbasis peristiwa memainkan peran penting dalam arsitektur layanan mikro. Pola berbasis cloud-native mendokumentasikan bagaimana arsitektur berbasis peristiwa memungkinkan kopling longgar antar layanan [7]. Pendekatan ini mendukung alur kerja kompleks melalui antrian pesan dan bus peristiwa, memungkinkan layanan untuk merespons perubahan status sistem secara asinkron. Pola-pola tersebut secara khusus merinci bagaimana teknologi seperti Apache Kafka atau RabbitMQ dapat digunakan untuk mengimplementasikan streaming peristiwa dan pengiriman pesan yang andal.

Tantangan dan Solusi Umum

Meskipun arsitektur microservices menawarkan banyak hal

Selain manfaatnya, arsitektur microservices juga menghadirkan kompleksitas signifikan yang harus ditangani secara efektif oleh organisasi. Sebagaimana diuraikan dalam panduan arsitektur microservices, tantangan ini terutama muncul dari sifat terdistribusi sistem dan kebutuhan untuk menjaga konsistensi antar layanan sambil memastikan batasan layanan yang tepat [9].

5.1. Kompleksitas Sistem Terdistribusi

Kompleksitas sistem terdistribusi menghadirkan tantangan mendasar dalam arsitektur layanan mikro.

Menurut pola desain microservices, perhatian utama berpusat pada ketahanan layanan dan komunikasi antar layanan [10]. Pola Circuit Breaker muncul sebagai solusi penting, mencegah kegagalan berantai dengan memantau kegagalan dan merangkum logika pencegahan kegagalan.

Kegagalan dapat terjadi secara terus-menerus. Jika diimplementasikan dengan benar, pola ini membantu menjaga stabilitas sistem dengan menyediakan mekanisme cadangan ketika layanan gagal. Implementasi service mesh, seperti yang dijelaskan dalam dokumentasi pola microservices Capital One, menyediakan lapisan infrastruktur khusus untuk menangani komunikasi antar layanan [10]. Pola ini menangani persyaratan operasional yang kompleks seperti penemuan layanan, penyeimbangan beban, pemulihan kegagalan, pengumpulan metrik, dan pemantauan. Dokumentasi tersebut secara khusus menyoroti bagaimana solusi service mesh dapat diimplementasikan menggunakan platform seperti Istio, yang menyediakan fitur-fitur seperti manajemen lalu lintas dan keamanan tanpa memerlukan perubahan pada kode aplikasi.

5.2. Konsistensi Data

Konsistensi data dalam lingkungan terdistribusi memerlukan pendekatan arsitektur tertentu. Panduan microservices langkah demi langkah menekankan pentingnya memilih pola konsistensi yang tepat berdasarkan persyaratan bisnis [9]. Konsistensi bertahap muncul sebagai solusi praktis ketika konsistensi waktu nyata tidak penting, memungkinkan sistem untuk mencapai keadaan konsisten dari waktu ke waktu daripada mempertahankan konsistensi langsung di semua layanan.

Pola Saga, seperti yang dijelaskan dalam pola desain Capital One, menawarkan solusi yang kuat untuk mengelola transaksi terdistribusi [10]. Pola ini menangani transaksi yang berjalan lama dengan memecahnya menjadi serangkaian transaksi lokal yang lebih kecil. Setiap transaksi memperbarui data dalam satu layanan dan menerbitkan peristiwa untuk memicu transaksi berikutnya dalam saga. Dokumentasi pola secara khusus menguraikan bagaimana transaksi kompensasi dapat diimplementasikan untuk menjaga konsistensi data ketika terjadi kegagalan.

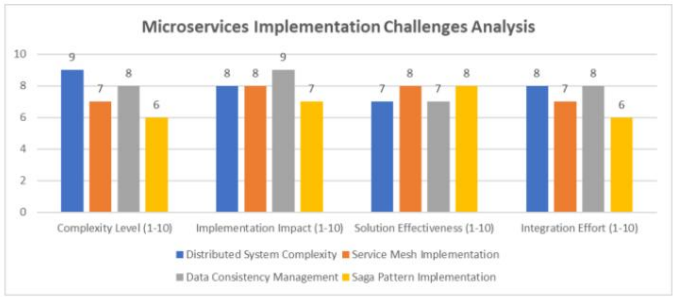
Command Query Responsibility Segregation (CQRS), yang disorot dalam panduan arsitektur microservices, menyediakan pendekatan terstruktur untuk menangani operasi data yang kompleks [9]. Pola ini memisahkan operasi baca dan tulis, memungkinkan masing-masing dioptimalkan secara independen. Panduan ini secara khusus merinci bagaimana CQRS dapat diimplementasikan bersamaan dengan event sourcing. untuk mempertahankan jejak audit lengkap dari semua perubahan sekaligus menyediakan model baca yang dioptimalkan untuk berbagai penggunaan. kasus.

5.3. Batasan Layanan

Menentukan batasan layanan yang tepat merupakan tantangan berkelanjutan dalam implementasi microservices. Panduan arsitektur microservices menekankan pentingnya prinsip Domain-Driven Design (DDD) dalam menetapkan batasan layanan yang efektif [9]. Pendekatan ini memastikan bahwa layanan selaras dengan domain bisnis, sehingga membuatnya lebih mudah dipelihara dan beradaptasi dengan perubahan kebutuhan bisnis.

Event storming, seperti yang didokumentasikan dalam panduan pola desain, menyediakan pendekatan kolaboratif untuk mengidentifikasi batas layanan [10]. Teknik ini menyatukan pakar domain dan tim teknis untuk memetakan proses bisnis dan mengidentifikasi batas layanan alami. Dokumentasi tersebut secara khusus menguraikan bagaimana sesi event storming dapat membantu tim menemukan kompleksitas dan ketergantungan tersembunyi antara berbagai bagian sistem.

Layanan mikro Dokumentasi pola menekankan pentingnya penyempurnaan terus-menerus dari batasan layanan [10]. Hal ini melibatkan pemantauan rutin interaksi dan ketergantungan layanan, mengidentifikasi area di mana layanan mungkin terlalu terikat erat, dan melakukan refactoring batasan bila diperlukan.
Dokumentasi tersebut secara khusus menekankan bagaimana definisi batasan harus berkembang seiring dengan kebutuhan bisnis, memastikan bahwa arsitektur tetap selaras dengan kebutuhan organisasi.



Gambar 2: Penilaian Komparatif Mikroservis Tantangan Arsitektur [9, 10]

Kesimpulan

Arsitektur microservices merupakan pendekatan yang ampuh bagi organisasi yang ingin membangun sistem perangkat lunak yang skalabel, mudah dipelihara, dan tangguh. Meskipun menawarkan manfaat signifikan dalam hal ketangkasan teknis, fleksibilitas penerapan, dan otonomi tim, implementasinya memerlukan pertimbangan berbagai faktor hati-hati berhasil termasuk prinsip desain layanan, persyaratan infrastruktur, dan potensi tantangan. Organisasi harus mengevaluasi kebutuhan dan kemampuan spesifik mereka sebelum mengadopsi microservices, memastikan mereka memiliki keahlian teknis dan dukungan infrastruktur yang diperlukan. Perjalanan menuju adopsi microservices membutuhkan pendekatan yang seimbang yang mempertimbangkan baik keuntungan maupun kompleksitasnya, dengan menyadari bahwa microservices bukanlah solusi universal tetapi lebih merupakan alat yang, ketika

Jika diterapkan dengan tepat, hal ini dapat secara signifikan meningkatkan kemampuan suatu organisasi untuk mengembangkan dan memelihara aplikasi kompleks di lingkungan cloud modern.

Referensi

[1]. Martin Fowler, "Panduan Layanan Mikro," martinfowler.com, 2019. [Online]. Tersedia: https://martinfowler.com/microservices/

[2]. Sam Newman, "Membangun Layanan Mikro: Merancang Sistem yang Lebih Detail," O'Reilly Media, 2015. [On line]. Tersedia: https://github.com/rootusercop/Buku-Buku-DevOps-Gratis-1/blob/master/book/Building%20Microservices%20-%20Designing%20Fine-Grained%20Systems.pdf

[3]. Chris Richardson, "Pola: Layanan Mikro [Online]. Arsitektur," Microservices.io. Tersedia: https://microservices.io/patterns/microservices.html

[4]. Layanan mikro, "Referensi Arsitektur," Microservices.com. [On line]. Tersedia: https://www.microservices.com/reference-architecture/

[5]. Microsoft, "Desain arsitektur microservices," Panduan Arsitektur Microsoft Azure. [Online]. Tersedia di: https://learn.microsoft.com/en-us/azure/architecture/microservices/

[6]. Mike Loukides dan Steve Swoyer, "Adopsi Layanan Mikro pada tahun 2020," O'Reilly Media, 2020. [On line]. Tersedia: https://www.oreilly.com/radar/adopsi-layanan-mikro di-2020/

[7]. Bijit Ghosh, "Pola dan Prinsip Arsitektur Cloud Native: Jalur Emas," Medium, 2023. [On line]. Tersedia di: https://medium.com/@bijit211987/cloud-native-architecture-patterns-and-principles-golden-path-250fa75ba178

[8]. Red Hat, "Mengembangkan Aplikasi Berbasis Awan dengan Arsitektur Layanan Mikro," Pelatihan Red Hat. [Online]. Tersedia di: https://www.redhat.com/en/services/training/do092-mengembangkan-aplikasi-cloud-native-arsitektur-mikroservis

[9]. Bayram Zengin, "Menguasai Arsitektur Layanan Mikro: Panduan Langkah demi Langkah untuk Sistem Terdistribusi," LinkedIn, 2024. [Online]. Tersedia: https://www.linkedin.com/pulse/mastering-microservices-architecture-step-by-step-guide-bayram-zengin-ihqxe

[10]. Capital One Tech, "10 pola desain microservices untuk arsitektur yang lebih baik," Capital One Tech, Medium, 2023. [Online]. Tersedia: https://medium.com/capital-one-tech/10-Pola desain layanan mikro untuk arsitektur yang lebih baik - befa810ca44e