**BAB I**

**PENDAHULUAN**

**1.1 Latar Belakang**

Dewasa ini pengolahan data di sebuah instansi menjadi semakin penting dan krusial di era digital ini. Pengolahan data yang baik mencerminkan bagaimana instansi ataupun organisasi tersebut bekerja, secara garis besar integritas data juga mempengaruhi integritas sebuah organisasi. Seiring perkembangan zaman, pengolahan data juga semakin *modern* dengan pengolahan data dengan sebuah sistem yang telah dikomputerisasi sehingga dapat lebih efektif dan cepat.

Pengolahan data secara komputerisasi juga diterapkan di berbagai bidang seperti pabrik, perkantoran dan instansi pendidikan. Penggunaan sistem informasi juga diaplikasikan dalam pengolahan data akademik pada sebuah instansi pendidikan mulai dari pendidikan dasar hingga perguruan tinggi. Sistem Informasi Akademik atau juga disebut SIAKAD mengelola berbagai data yang berkaitan dengan perihal akademik dalam sebuah instansi pendidikan.

Penggunaan SIAKAD dalam sebuah instansi pendidikan terbukti mempermudah pekerjaan administrasi di sebuah instansi kampus. Sistem tersebut dapat mengelola berbagai kebutuhan mulai dari keuangan hingga pengelolaan nilai dan dokumen-dokumen yang berhubungan dengan aktifitas perkuliahan. SIAKAD dapat diakses dari manapun dan kapanpun, jadi dengan pengaplikasian sistem tersebut dapat meningkatkan ketersediaan dan produktifitas dari staff kampus tersebut.

Namun, pengguna aplikasi ini bukan saja staff dari instansi tersebut, mahasiswa juga menjadi pengguna dari aplikasi tersebut. Dengan SIAKAD mahasiswa dapat melakukan beberapa prosedur administrasi secara daring dan langsung dan mempermudah pekerjaan staff akademik. Sebagai contoh kegiatan seperti KRS, mahasiswa yang mengajukan KRS secara daring juga akan diberi persetujuan secara online pula melalui sistem.

Penerapan berbagai kegiatan perkuliahan melalui sebuah sistem memang sangat mempermudah pekerjaan, akan tetapi ada harga yang harus dibayar dengan harga yang pantas. Penerapan SIAKAD untuk proses administrasi membutuhkan sumber daya yang sangat besar dan kompleks. Hal ini menjadi sebuah kekurangan tersendiri untuk penerapan SIAKAD disebuah kampus, saat sistem diakses secara bersamaan beban akan semakin besar dan bisa menyebabkan galat pada sistem.

SIAKAD pada umumnya menggunakan arsitektur *monolith* yang memusatkan sistem pada satu bagian utuh yang besar dan kompleks. Hal ini menyebabkan jika satu layanan yang ada di dalam SIAKAD mengalami galat maka seluruh sistem dapat mengalami galat secara bersamaan dan membuat sistem tidak berjalan.

Namun galat tidak semata-mata dikarenakan arsitektur dari aplikasi tersebu, ada beberapa hal teknis lain yang juga berdampak terhadap sistem. Dalam penelitian ini pengarang mengusulkan sebuah konsep pengembangan yang menerapkan arsitektur microservices untuk SIAKAD di STIKI MALANG yang disebut dengan SAKTI. Diharapkan dengan penerapan arsitektur yang tepat dapat meningkatkan ketersediaan dari SIAKAD itu sendiri.

**1.2 Rumusan Masalah**

1. Bagaimana cara menerapkan arsitektur *microservice* pada SAKTI STIKI?
2. Bagaimana menerapkan arsitektur *microservice* pada sebuah sistem informasi ?
3. Bagaimana penerapan arsitektur *microservice* yang tepat?

**1.3 Tujuan Penelitian**

Tujuan penelitian ini adalah untuk menemukan metode penerapan arsitektur *microservice* yang tepat pada sebuah Sistem Informasi Akademik dengan mempertimbangkan faktor-faktor tertentu untuk menyesuaikan teknik yang akan digunakan

**1.4 Batasan Masalah**

Batasan masalah dalam penelitian ini sebagai berikut:

1. Penerapan arsitektur hanya diterpakan pada Sistem Informasi Akademik STIKI
2. Penelitian ini hanya berfokus pada arsitektur dan layanan yang ada pada Sistem Informasi akademik STIKI

**1.5 Manfaat penelitian**

Peneliti berharap hasil penelitian ini dapat dijadikan dasar untuk pengembangan SAKTI STIKI kedepannya agar dapat memberikan layanan yang lebih baik lagi dari sebelumnya

**BAB II**

**LANDASAN TEORI**

1. ***Microservice***

*Microservice* adalah sebuah model arsitektur pengembangan aplikasi yang membagi modul aplikasi berdasarkan layanan yang diberikan. Berkebalikan dengan arsitektur *monolith* yang menjadikan sistem sebagai satu bagian utuh dari sebuah aplikasi sehingga terkesan berat dan sangat kompleks. *Microservice* memiliki kelebihan dari sisi pembagian daya dan pengembangan yang lebih cepat dan pembagian yang jelas sesuai dengan layanan yang ada.(Richardson, 2018)

1. **API**

API kependekan dari (*Application Programming Interface*) adalah sebuah layanan yang memperbolehkan setiap aplikasi dengan arsitektur dan bahasa pemprograman yang berbeda-beda dapat berkomunikasi. Jadi API dapat dibayangkan sebagai sebuah bahasa universal yang dapat digunakan oleh semua aplikasi dengan platform dan bahasa pemprrograman yang berbeda. API sendiri digunakan untuk bertukar data dan mengirim data. (Richardson, 2018)

1. ***Load Balancing***

*Load balancing* adalah metode *management request* pada tingkat jaringan dan *server*. *Load balancing* digunakan untuk memeratakan akses request pada server sehingga dapat menghemat penggunaan sumber data dan mempercepat akses dari layanan yang diberikan. Dengan begitu layanan dapat digunakan lebih lama(Richardson, 2018)

1. **Virtualisasi Berbasis *Container***

Virtualisasi berbasis container adalah virtualisasi pada level Operasi Sistem. Jadi pada virtualisasi ini sistem akan membuat lingkungan yang serupa dengan Operasi Sistem yang kita tuju tanpa memperhatikan level infrastruktur. Dengan virtualisasi berbasis container selain mempermudah proses isolasi aplikasi juga mempermudah management infrastruktur (Richardson, 2018)

**2.5 Orkestrasi *Container***

Orkestrasi *container* adalah sebuah alat yang digunakan untuk mempermudah manajerial *instance* yang digunakan. Dengan orkestrasi juga bisa dilakukan automatisasi dan backup secara berkala dan otomatis. (Richardson, 2018)

**2.6 *Database***

*Database* adalah representasi digital dari pengolahan data yang dikomputerisasi. Jadi *database* bisa dibilang adalah kumpulan data digital yang dikumpulkan dalam bentuk tabel-tabel yang memiliki relasi antar satu sama lain.

**BAB III**

**METODE PERANCANGAN**

**3.1 Sistematika Perancangan**

**3.1.1 Kasus Galat Sistem Pada KRS daring SAKTI STIKI**

Kegagalan sistem pada saat proses pengambilan KRS adalah sebuah hal yang sering sekali ditemui oleh mahasiswa khususnya pada saat pergantian semester. Menurut pengamatan peneliti, kejadian tersebut selalu terjadi berulang-ulang dikarenakan saat diakses secara bersamaan *server* mengalami *load* yang ringgi sehingga membebani sistem dan menyebabkan kegagalan sistem. Keadaan tersebut diperparah dengan KRS daring yang sifatnya seperti *battle royale* yang berdampak pada adu cepat antar mahasiswa.

Jika digambarkan kejadian ini hamper sama dengan *flash sale* yang biasa dilakukan oleh market place pada saat promo. Jadi bisa dibayangkan bagaimana besarnya *request* yang diterima oleh server. Karena itu kejadian galat sistem selalu terjadi pada saar KRS Daring setiap tahunnya.

**3.1.2 Proses Pemetaan Pelayanan**

Secara garis besar ada beberapa layanan yang diberikan oleh SAKTI STIKI, yaitu ;

1. Keuangan
2. Mata Kuliah
3. Kehadiran
4. Akademik
5. Penilaian Mahasiswa

Layanan tersebut belum termasuk dengan beberapa sistem yang juga terintegrasi dengan beberapa layanan pihak ke-3 yang digunakan oleh kampus STIKI.

1. 1. **Proses Pembagian Layanan**

Setelah didapatkan layanan apa saja yang digunakan pada SAKTI, maka didapatkan penggunaan seperti di bawah ini;

|  |  |  |  |
| --- | --- | --- | --- |
| **Layanan** | **Jumlah Pengguna**  **Rata-rata/bulan** | **presentase error** | **Rata-rata waktu** |
| Keuangan | 20 | 0.0075% | 100ms |
| Mata Kuliah | 100 | 0.05% | 1230ms |
| Kehadiran | 30 | 0.01% | 200ms |
| Akademik | 20 | 0.0155% | 1350ms |
| Penilaian Mahasiswa | 10 | 0.0003% | 150ms |

Tabel 1.1 adalah table perbandingan layanan dan tingkat kegalatan

Dari tabel diatas dapat disimpulkan layanan Mata Kuliah dan Akademik adalah 2 layanan yang paling sering mengalami galat dan banyak pengakses. Maka dari sini kita dapatkan menyimpulkan layanan mana yang paling mengalami galat dan dapat diperbaiki.

Pada dasarnya *merefactor* sebuah aplikasi *monolith* ke arsitektur *microservice* dapat dilakukan secara parsial dan hanya sebagian. Jadi dari dasar data di atas dapat disimpulkan cukup layanan yang berhubungan dengan KRS yang perlu di-*refactor* menjadi *microservices.*

1. 1. **Implementasi**

Dalam Implementasi ada tahapan yang peneliti lalui untuk membuktikan teori yang digunakan, yaitu ;

1. Pemisahan KRS daring dari SAKTI STIKI

Langkah pertama yang dilakukan adalah memisahkan layanan KRS Daring dari SAKTI STIKI dan mengembangkannya dengan arsitektur *microservices.* Dengan begini layanan akan bersifat independen dan tidak bergantung pada sistem utama SAKTI STIKI.

Tahapan lebih lanjut pemisahan layanan akan dijelaskan pada tahapan selanjunya. Pada prinsipnya kunci dari tahapan ini adalah memisahkan layanan kecil dari layanan utama. Dengan begini pengembangan kedepannya dari layanan ini akan lebih mudah dan cepat.

1. Pemisahan Backend & Frontend dengan API

*Gambar 1.1 Ilustrasi REST API*

Langkah selanjutnya membangun sebuah REST API yang dapat diakses dari sisi backend dan frontend. Hal ini dapat mengurangi beban *server*, beban sistem dan juga meningkatkan kemudahan pengembangan layanan ini kedepannya. Jadi Frontend dan Backend tidak terhubung secara langsung melainkan menggunakan perantara API yang kemudian akan terhubung ke *database*

1. Penulisan Ulang Aplikasi

Selanjutnya dari sisi pemprograman, dilakukan pengecekan ulang terhadap baris kode yang sudah ditulis untuk memastikan tidak ada baris yang menyebabkan pemborosan sumber daya maupun galat pada sistem secara meyeluruh.

SAKTI menggunakan platform web yang terhitung cukup mudah untuk melakukan proses pengecekan ulang. Jadi pemilihan bahasa pemprograman juga menjadi faktor pendukung dalam tahapan ini.

1. Optimasi *Query*

Selanjutnya kita perlu memastikan bahwa *query* atau proses pemanggilan data dari database. Seringkali terjadi kesalahan pemilihan *query* yang menyebabkan proses yang tidak optmal dan membuang banyak sumber daya dan memakan banyak waktu eksekusi.

1. Containerization dengan docker

Setelah membagi layanan ke beberapa layanan berbeda selanjutnya membedakan *instance* atau server dari layanan. Hal ini perlu dilakukan untuk memastikan bahwa layanan KRS daring benar-benar independen dan terisolasi. Sehingga layanan dapat berjalan lebih optimal karena terisolasi dalam sebuah *instance* mandiri.

1. Load Balancing

*Gambar 1.2 Ilustrasi Load Balancing*

Setelah instance dipisahkan selanjutnya proses load balancing dan manajerial sumber-sumber yang digunakan oleh layanan KRS. Setelah dipisahkan maka proses pemanggilan sumber daya secara terpisah dan lebih cepat.

1. 1. **Sumber Data**

Data didapatkan dari analisis *log server* dan layanan, serta survei ke beberapa mahasiswa yang sudah pernah menggunakan layanan dan mengalami galat ataupun pelambatan sistem.

1. 1. **Teknik Pengambilan Data**

Teknik pengambilan data menggunakan *sampling log* dan survei ke beberapa mahasiswa.

1. 1. **Analisa Data**

Analisis dilakukan sebelum layanan dilakukan dan setelah layanan berjalan sesuai dengan pendekatan dan instrument yang digunakan dalam penelitian ini.

**BAB IV**

**PENUTUP**

1. **KESIMPULAN**

Penerapan arsitektur *microservice* yang kurang tepat dapat mempersulit pengelolaan dan pengembangan aplikasi kedepannya. Penggunakan arsitektur harus disesuaikan dengan kebutuhan bukan merubah secara keseluruhan. Perubahan secara keseluruhan akan mempersulit pengelolaan, pengerjaan dan beberapa hal yang bersifat teknis seperti *instance* dan *database*.

1. **SARAN**

Untuk penerapannya disarankan untuk mengimpelementasikan arsitektur secara parsial sesuai dengan bagian yang memang membutuhkan layanan secara mandiri.

**DAFTAR PUSTAKA**

Richardson, Chris. (2018) *Microservice Pattern With Java Example*, 34-71. Manning:San Franscisco

“Istiadi,Muhammad Abrar”, (2020) Penguatan KRS Online Demi Kelancaran KRS War<https://medium.com/ictipb-tech/penguatan-krs-online-demi-> kelancaran-krs- war-7a712aa79de5( diakses 14 januari 2020).

**BAB I**

**PENDAHULUAN**

**1.1 Latar Belakang**

Dewasa ini pengolahan data di sebuah instansi menjadi semakin penting dan krusial di era digital ini. Pengolahan data yang baik mencerminkan bagaimana instansi ataupun organisasi tersebut bekerja, secara garis besar integritas data juga mempengaruhi integritas sebuah organisasi. Seiring perkembangan zaman, pengolahan data juga semakin *modern* dengan pengolahan data dengan sebuah sistem yang telah dikomputerisasi sehingga dapat lebih efektif dan cepat.

Pengolahan data secara komputerisasi juga diterapkan di berbagai bidang seperti pabrik, perkantoran dan instansi pendidikan. Penggunaan sistem informasi juga diaplikasikan dalam pengolahan data akademik pada sebuah instansi pendidikan mulai dari pendidikan dasar hingga perguruan tinggi. Sistem Informasi Akademik atau juga disebut SIAKAD mengelola berbagai data yang berkaitan dengan perihal akademik dalam sebuah instansi pendidikan.

Penggunaan SIAKAD dalam sebuah instansi pendidikan terbukti mempermudah pekerjaan administrasi di sebuah instansi kampus. Sistem tersebut dapat mengelola berbagai kebutuhan mulai dari keuangan hingga pengelolaan nilai dan dokumen-dokumen yang berhubungan dengan aktifitas perkuliahan. SIAKAD dapat diakses dari manapun dan kapanpun, jadi dengan pengaplikasian sistem tersebut dapat meningkatkan ketersediaan dan produktifitas dari staff kampus tersebut.

Namun, pengguna aplikasi ini bukan saja staff dari instansi tersebut, mahasiswa juga menjadi pengguna dari aplikasi tersebut. Dengan SIAKAD mahasiswa dapat melakukan beberapa prosedur administrasi secara daring dan langsung dan mempermudah pekerjaan staff akademik. Sebagai contoh kegiatan seperti KRS, mahasiswa yang mengajukan KRS secara daring juga akan diberi persetujuan secara online pula melalui sistem.

Penerapan berbagai kegiatan perkuliahan melalui sebuah sistem memang sangat mempermudah pekerjaan, akan tetapi ada harga yang harus dibayar dengan harga yang pantas. Penerapan SIAKAD untuk proses administrasi membutuhkan sumber daya yang sangat besar dan kompleks. Hal ini menjadi sebuah kekurangan tersendiri untuk penerapan SIAKAD disebuah kampus, saat sistem diakses secara bersamaan beban akan semakin besar dan bisa menyebabkan galat pada sistem.

SIAKAD pada umumnya menggunakan arsitektur *monolith* yang memusatkan sistem pada satu bagian utuh yang besar dan kompleks. Hal ini menyebabkan jika satu layanan yang ada di dalam SIAKAD mengalami galat maka seluruh sistem dapat mengalami galat secara bersamaan dan membuat sistem tidak berjalan.

Namun galat tidak semata-mata dikarenakan arsitektur dari aplikasi tersebu, ada beberapa hal teknis lain yang juga berdampak terhadap sistem. Dalam penelitian ini pengarang mengusulkan sebuah konsep pengembangan yang menerapkan arsitektur microservices untuk SIAKAD di STIKI MALANG yang disebut dengan SAKTI. Diharapkan dengan penerapan arsitektur yang tepat dapat meningkatkan ketersediaan dari SIAKAD itu sendiri.

**1.2 Rumusan Masalah**

1. Bagaimana cara menerapkan arsitektur *microservice* pada SAKTI STIKI?
2. Bagaimana menerapkan arsitektur *microservice* pada sebuah sistem informasi ?
3. Bagaimana penerapan arsitektur *microservice* yang tepat?

**1.3 Tujuan Penelitian**

Tujuan penelitian ini adalah untuk menemukan metode penerapan arsitektur *microservice* yang tepat pada sebuah Sistem Informasi Akademik dengan mempertimbangkan faktor-faktor tertentu untuk menyesuaikan teknik yang akan digunakan

**1.4 Batasan Masalah**

Batasan masalah dalam penelitian ini sebagai berikut:

1. Penerapan arsitektur hanya diterpakan pada Sistem Informasi Akademik STIKI
2. Penelitian ini hanya berfokus pada arsitektur dan layanan yang ada pada Sistem Informasi akademik STIKI

**1.5 Manfaat penelitian**

Peneliti berharap hasil penelitian ini dapat dijadikan dasar untuk pengembangan SAKTI STIKI kedepannya agar dapat memberikan layanan yang lebih baik lagi dari sebelumnya

**BAB II**

**LANDASAN TEORI**

1. ***Microservice***

*Microservice* adalah sebuah model arsitektur pengembangan aplikasi yang membagi modul aplikasi berdasarkan layanan yang diberikan. Berkebalikan dengan arsitektur *monolith* yang menjadikan sistem sebagai satu bagian utuh dari sebuah aplikasi sehingga terkesan berat dan sangat kompleks. *Microservice* memiliki kelebihan dari sisi pembagian daya dan pengembangan yang lebih cepat dan pembagian yang jelas sesuai dengan layanan yang ada.(Richardson, 2018)

1. **API**

API kependekan dari (*Application Programming Interface*) adalah sebuah layanan yang memperbolehkan setiap aplikasi dengan arsitektur dan bahasa pemprograman yang berbeda-beda dapat berkomunikasi. Jadi API dapat dibayangkan sebagai sebuah bahasa universal yang dapat digunakan oleh semua aplikasi dengan platform dan bahasa pemprrograman yang berbeda. API sendiri digunakan untuk bertukar data dan mengirim data. (Richardson, 2018)

1. ***Load Balancing***

*Load balancing* adalah metode *management request* pada tingkat jaringan dan *server*. *Load balancing* digunakan untuk memeratakan akses request pada server sehingga dapat menghemat penggunaan sumber data dan mempercepat akses dari layanan yang diberikan. Dengan begitu layanan dapat digunakan lebih lama(Richardson, 2018)

1. **Virtualisasi Berbasis *Container***

Virtualisasi berbasis container adalah virtualisasi pada level Operasi Sistem. Jadi pada virtualisasi ini sistem akan membuat lingkungan yang serupa dengan Operasi Sistem yang kita tuju tanpa memperhatikan level infrastruktur. Dengan virtualisasi berbasis container selain mempermudah proses isolasi aplikasi juga mempermudah management infrastruktur (Richardson, 2018)

**2.5 Orkestrasi *Container***

Orkestrasi *container* adalah sebuah alat yang digunakan untuk mempermudah manajerial *instance* yang digunakan. Dengan orkestrasi juga bisa dilakukan automatisasi dan backup secara berkala dan otomatis. (Richardson, 2018)

**2.6 *Database***

*Database* adalah representasi digital dari pengolahan data yang dikomputerisasi. Jadi *database* bisa dibilang adalah kumpulan data digital yang dikumpulkan dalam bentuk tabel-tabel yang memiliki relasi antar satu sama lain.

**BAB III**

**METODE PERANCANGAN**

**3.1 Sistematika Perancangan**

**3.1.1 Kasus Galat Sistem Pada KRS daring SAKTI STIKI**

Kegagalan sistem pada saat proses pengambilan KRS adalah sebuah hal yang sering sekali ditemui oleh mahasiswa khususnya pada saat pergantian semester. Menurut pengamatan peneliti, kejadian tersebut selalu terjadi berulang-ulang dikarenakan saat diakses secara bersamaan *server* mengalami *load* yang ringgi sehingga membebani sistem dan menyebabkan kegagalan sistem. Keadaan tersebut diperparah dengan KRS daring yang sifatnya seperti *battle royale* yang berdampak pada adu cepat antar mahasiswa.

Jika digambarkan kejadian ini hamper sama dengan *flash sale* yang biasa dilakukan oleh market place pada saat promo. Jadi bisa dibayangkan bagaimana besarnya *request* yang diterima oleh server. Karena itu kejadian galat sistem selalu terjadi pada saar KRS Daring setiap tahunnya.

**3.1.2 Proses Pemetaan Pelayanan**

Secara garis besar ada beberapa layanan yang diberikan oleh SAKTI STIKI, yaitu ;

1. Keuangan
2. Mata Kuliah
3. Kehadiran
4. Akademik
5. Penilaian Mahasiswa

Layanan tersebut belum termasuk dengan beberapa sistem yang juga terintegrasi dengan beberapa layanan pihak ke-3 yang digunakan oleh kampus STIKI.

1. 1. **Proses Pembagian Layanan**

Setelah didapatkan layanan apa saja yang digunakan pada SAKTI, maka didapatkan penggunaan seperti di bawah ini;

|  |  |  |  |
| --- | --- | --- | --- |
| **Layanan** | **Jumlah Pengguna**  **Rata-rata/bulan** | **presentase error** | **Rata-rata waktu** |
| Keuangan | 20 | 0.0075% | 100ms |
| Mata Kuliah | 100 | 0.05% | 1230ms |
| Kehadiran | 30 | 0.01% | 200ms |
| Akademik | 20 | 0.0155% | 1350ms |
| Penilaian Mahasiswa | 10 | 0.0003% | 150ms |

Tabel 1.1 adalah table perbandingan layanan dan tingkat kegalatan

Dari tabel diatas dapat disimpulkan layanan Mata Kuliah dan Akademik adalah 2 layanan yang paling sering mengalami galat dan banyak pengakses. Maka dari sini kita dapatkan menyimpulkan layanan mana yang paling mengalami galat dan dapat diperbaiki.

Pada dasarnya *merefactor* sebuah aplikasi *monolith* ke arsitektur *microservice* dapat dilakukan secara parsial dan hanya sebagian. Jadi dari dasar data di atas dapat disimpulkan cukup layanan yang berhubungan dengan KRS yang perlu di-*refactor* menjadi *microservices.*

1. 1. **Implementasi**

Dalam Implementasi ada tahapan yang peneliti lalui untuk membuktikan teori yang digunakan, yaitu ;

1. Pemisahan KRS daring dari SAKTI STIKI

Langkah pertama yang dilakukan adalah memisahkan layanan KRS Daring dari SAKTI STIKI dan mengembangkannya dengan arsitektur *microservices.* Dengan begini layanan akan bersifat independen dan tidak bergantung pada sistem utama SAKTI STIKI.

Tahapan lebih lanjut pemisahan layanan akan dijelaskan pada tahapan selanjunya. Pada prinsipnya kunci dari tahapan ini adalah memisahkan layanan kecil dari layanan utama. Dengan begini pengembangan kedepannya dari layanan ini akan lebih mudah dan cepat.

1. Pemisahan Backend & Frontend dengan API

*Gambar 1.1 Ilustrasi REST API*

Langkah selanjutnya membangun sebuah REST API yang dapat diakses dari sisi backend dan frontend. Hal ini dapat mengurangi beban *server*, beban sistem dan juga meningkatkan kemudahan pengembangan layanan ini kedepannya. Jadi Frontend dan Backend tidak terhubung secara langsung melainkan menggunakan perantara API yang kemudian akan terhubung ke *database*

1. Penulisan Ulang Aplikasi

Selanjutnya dari sisi pemprograman, dilakukan pengecekan ulang terhadap baris kode yang sudah ditulis untuk memastikan tidak ada baris yang menyebabkan pemborosan sumber daya maupun galat pada sistem secara meyeluruh.

SAKTI menggunakan platform web yang terhitung cukup mudah untuk melakukan proses pengecekan ulang. Jadi pemilihan bahasa pemprograman juga menjadi faktor pendukung dalam tahapan ini.

1. Optimasi *Query*

Selanjutnya kita perlu memastikan bahwa *query* atau proses pemanggilan data dari database. Seringkali terjadi kesalahan pemilihan *query* yang menyebabkan proses yang tidak optmal dan membuang banyak sumber daya dan memakan banyak waktu eksekusi.

1. Containerization dengan docker

Setelah membagi layanan ke beberapa layanan berbeda selanjutnya membedakan *instance* atau server dari layanan. Hal ini perlu dilakukan untuk memastikan bahwa layanan KRS daring benar-benar independen dan terisolasi. Sehingga layanan dapat berjalan lebih optimal karena terisolasi dalam sebuah *instance* mandiri.

1. Load Balancing

*Gambar 1.2 Ilustrasi Load Balancing*

Setelah instance dipisahkan selanjutnya proses load balancing dan manajerial sumber-sumber yang digunakan oleh layanan KRS. Setelah dipisahkan maka proses pemanggilan sumber daya secara terpisah dan lebih cepat.

1. 1. **Sumber Data**

Data didapatkan dari analisis *log server* dan layanan, serta survei ke beberapa mahasiswa yang sudah pernah menggunakan layanan dan mengalami galat ataupun pelambatan sistem.

1. 1. **Teknik Pengambilan Data**

Teknik pengambilan data menggunakan *sampling log* dan survei ke beberapa mahasiswa.

1. 1. **Analisa Data**

Analisis dilakukan sebelum layanan dilakukan dan setelah layanan berjalan sesuai dengan pendekatan dan instrument yang digunakan dalam penelitian ini.

**BAB IV**

**PENUTUP**

1. **KESIMPULAN**

Penerapan arsitektur *microservice* yang kurang tepat dapat mempersulit pengelolaan dan pengembangan aplikasi kedepannya. Penggunakan arsitektur harus disesuaikan dengan kebutuhan bukan merubah secara keseluruhan. Perubahan secara keseluruhan akan mempersulit pengelolaan, pengerjaan dan beberapa hal yang bersifat teknis seperti *instance* dan *database*.

1. **SARAN**

Untuk penerapannya disarankan untuk mengimpelementasikan arsitektur secara parsial sesuai dengan bagian yang memang membutuhkan layanan secara mandiri.

**DAFTAR PUSTAKA**

Richardson, Chris. (2018) *Microservice Pattern With Java Example*, 34-71. Manning:San Franscisco

“Istiadi,Muhammad Abrar”, (2020) Penguatan KRS Online Demi Kelancaran KRS War<https://medium.com/ictipb-tech/penguatan-krs-online-demi-> kelancaran-krs- war-7a712aa79de5( diakses 14 januari 2020).

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dewasa ini pengolahan data di sebuah instansi menjadi semakin penting dan krusial di era digital ini. Pengolahan data yang baik mencerminkan bagaimana instansi ataupun organisasi tersebut bekerja, secara garis besar integritas data juga mempengaruhi integritas sebuah organisasi. Seiring perkembangan zaman, pengolahan data juga semakin modern dengan pengolahan data dengan sebuah sistem yang telah dikomputerisasi sehingga dapat lebih efektif dan cepat.

Pengolahan data secara komputerisasi juga diterapkan di berbagai bidang seperti pabrik, perkantoran dan instansi pendidikan. Penggunaan sistem informasi juga diaplikasikan dalam pengolahan data akademik pada sebuah instansi pendidikan mulai dari pendidikan dasar hingga perguruan tinggi. Sistem Informasi Akademik atau juga disebut SIAKAD mengelola berbagai data yang berkaitan dengan perihal akademik dalam sebuah instansi pendidikan.

Penggunaan SIAKAD dalam sebuah instansi pendidikan terbukti mempermudah pekerjaan administrasi di sebuah instansi kampus. Sistem tersebut dapat mengelola berbagai kebutuhan mulai dari keuangan hingga pengelolaan nilai dan dokumen-dokumen yang berhubungan dengan aktifitas perkuliahan. SIAKAD dapat diakses dari manapun dan kapanpun, jadi dengan pengaplikasian sistem tersebut dapat meningkatkan ketersediaan dan produktifitas dari staff kampus tersebut.

Namun, pengguna aplikasi ini bukan saja staff dari instansi tersebut, mahasiswa juga menjadi pengguna dari aplikasi tersebut. Dengan SIAKAD mahasiswa dapat melakukan beberapa prosedur administrasi secara daring dan langsung dan mempermudah pekerjaan staff akademik. Sebagai contoh kegiatan seperti KRS, mahasiswa yang mengajukan KRS secara daring juga akan diberi persetujuan secara online pula melalui sistem.

Penerapan berbagai kegiatan perkuliahan melalui sebuah sistem memang sangat mempermudah pekerjaan, akan tetapi ada harga yang harus dibayar dengan harga yang pantas. Penerapan SIAKAD untuk proses administrasi membutuhkan sumber daya yang sangat besar dan kompleks. Hal ini menjadi sebuah kekurangan tersendiri untuk penerapan SIAKAD disebuah kampus, saat sistem diakses secara bersamaan beban akan semakin besar dan bisa menyebabkan galat pada sistem.

SIAKAD pada umumnya menggunakan arsitektur monolith yang memusatkan sistem pada satu bagian utuh yang besar dan kompleks. Hal ini menyebabkan jika satu layanan yang ada di dalam SIAKAD mengalami galat maka seluruh sistem dapat mengalami galat secara bersamaan dan membuat sistem tidak berjalan.

Namun galat tidak semata-mata dikarenakan arsitektur dari aplikasi tersebu, ada beberapa hal teknis lain yang juga berdampak terhadap sistem. Dalam penelitian ini pengarang mengusulkan sebuah konsep pengembangan yang menerapkan arsitektur microservices untuk SIAKAD di STIKI MALANG yang disebut dengan SAKTI. Diharapkan dengan penerapan arsitektur yang tepat dapat meningkatkan ketersediaan dari SIAKAD itu sendiri.

1.2 Rumusan Masalah

1. Bagaimana cara menerapkan arsitektur microservice pada SAKTI STIKI?

2. Bagaimana menerapkan arsitektur microservice pada sebuah sistem informasi ?

3. Bagaimana penerapan arsitektur microservice yang tepat?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah untuk menemukan metode penerapan arsitektur microservice yang tepat pada sebuah Sistem Informasi Akademik dengan mempertimbangkan faktor-faktor tertentu untuk menyesuaikan teknik yang akan digunakan

1.4 Batasan Masalah

Batasan masalah dalam penelitian ini sebagai berikut:

1. Penerapan arsitektur hanya diterpakan pada Sistem Informasi Akademik STIKI

2. Penelitian ini hanya berfokus pada arsitektur dan layanan yang ada pada Sistem Informasi akademik STIKI

1.5 Manfaat penelitian

Peneliti berharap hasil penelitian ini dapat dijadikan dasar untuk pengembangan SAKTI STIKI kedepannya agar dapat memberikan layanan yang lebih baik lagi dari sebelumnya

BAB II

LANDASAN TEORI

2.1 Microservice

Microservice adalah sebuah model arsitektur pengembangan aplikasi yang membagi modul aplikasi berdasarkan layanan yang diberikan. Berkebalikan dengan arsitektur monolith yang menjadikan sistem sebagai satu bagian utuh dari sebuah aplikasi sehingga terkesan berat dan sangat kompleks. Microservice memiliki kelebihan dari sisi pembagian daya dan pengembangan yang lebih cepat dan pembagian yang jelas sesuai dengan layanan yang ada.(Richardson, 2018)

2.2 API

API kependekan dari (Application Programming Interface) adalah sebuah layanan yang memperbolehkan setiap aplikasi dengan arsitektur dan bahasa pemprograman yang berbeda-beda dapat berkomunikasi. Jadi API dapat dibayangkan sebagai sebuah bahasa universal yang dapat digunakan oleh semua aplikasi dengan platform dan bahasa pemprrograman yang berbeda. API sendiri digunakan untuk bertukar data dan mengirim data. (Richardson, 2018)

2.3 Load Balancing

Load balancing adalah metode management request pada tingkat jaringan dan server. Load balancing digunakan untuk memeratakan akses request pada server sehingga dapat menghemat penggunaan sumber data dan mempercepat akses dari layanan yang diberikan. Dengan begitu layanan dapat digunakan lebih lama(Richardson, 2018)

2.4 Virtualisasi Berbasis Container

Virtualisasi berbasis container adalah virtualisasi pada level Operasi Sistem. Jadi pada virtualisasi ini sistem akan membuat lingkungan yang serupa dengan Operasi Sistem yang kita tuju tanpa memperhatikan level infrastruktur. Dengan virtualisasi berbasis container selain mempermudah proses isolasi aplikasi juga mempermudah management infrastruktur (Richardson, 2018)

2.5 Orkestrasi Container

Orkestrasi container adalah sebuah alat yang digunakan untuk mempermudah manajerial instance yang digunakan. Dengan orkestrasi juga bisa dilakukan automatisasi dan backup secara berkala dan otomatis. (Richardson, 2018)

2.6 Database

Database adalah representasi digital dari pengolahan data yang dikomputerisasi. Jadi database bisa dibilang adalah kumpulan data digital yang dikumpulkan dalam bentuk tabel-tabel yang memiliki relasi antar satu sama lain.

BAB III

METODE PERANCANGAN

3.1 Sistematika Perancangan

3.1.1 Kasus Galat Sistem Pada KRS daring SAKTI STIKI

Kegagalan sistem pada saat proses pengambilan KRS adalah sebuah hal yang sering sekali ditemui oleh mahasiswa khususnya pada saat pergantian semester. Menurut pengamatan peneliti, kejadian tersebut selalu terjadi berulang-ulang dikarenakan saat diakses secara bersamaan server mengalami load yang ringgi sehingga membebani sistem dan menyebabkan kegagalan sistem. Keadaan tersebut diperparah dengan KRS daring yang sifatnya seperti battle royale yang berdampak pada adu cepat antar mahasiswa.

Jika digambarkan kejadian ini hamper sama dengan flash sale yang biasa dilakukan oleh market place pada saat promo. Jadi bisa dibayangkan bagaimana besarnya request yang diterima oleh server. Karena itu kejadian galat sistem selalu terjadi pada saar KRS Daring setiap tahunnya.

3.1.2 Proses Pemetaan Pelayanan

Secara garis besar ada beberapa layanan yang diberikan oleh SAKTI STIKI, yaitu ;

a. Keuangan

b. Mata Kuliah

c. Kehadiran

d. Akademik

e. Penilaian Mahasiswa

Layanan tersebut belum termasuk dengan beberapa sistem yang juga terintegrasi dengan beberapa layanan pihak ke-3 yang digunakan oleh kampus STIKI.

3.1.2 Proses Pembagian Layanan

Setelah didapatkan layanan apa saja yang digunakan pada SAKTI, maka didapatkan penggunaan seperti di bawah ini;

Layanan

Jumlah Pengguna

Rata-rata/bulan

presentase error

Rata-rata waktu

Keuangan

20

0.0075%

100ms

Mata Kuliah

100

0.05%

1230ms

Kehadiran

30

0.01%

200ms

Akademik

20

0.0155%

1350ms

Penilaian Mahasiswa

10

0.0003%

150ms

Tabel 1.1 adalah table perbandingan layanan dan tingkat kegalatan

Dari tabel diatas dapat disimpulkan layanan Mata Kuliah dan Akademik adalah 2 layanan yang paling sering mengalami galat dan banyak pengakses. Maka dari sini kita dapatkan menyimpulkan layanan mana yang paling mengalami galat dan dapat diperbaiki.

Pada dasarnya merefactor sebuah aplikasi monolith ke arsitektur microservice dapat dilakukan secara parsial dan hanya sebagian. Jadi dari dasar data di atas dapat disimpulkan cukup layanan yang berhubungan dengan KRS yang perlu di-refactor menjadi microservices.

3.1.3 Implementasi

Dalam Implementasi ada tahapan yang peneliti lalui untuk membuktikan teori yang digunakan, yaitu ;

a. Pemisahan KRS daring dari SAKTI STIKI

Langkah pertama yang dilakukan adalah memisahkan layanan KRS Daring dari SAKTI STIKI dan mengembangkannya dengan arsitektur microservices. Dengan begini layanan akan bersifat independen dan tidak bergantung pada sistem utama SAKTI STIKI.

Tahapan lebih lanjut pemisahan layanan akan dijelaskan pada tahapan selanjunya. Pada prinsipnya kunci dari tahapan ini adalah memisahkan layanan kecil dari layanan utama. Dengan begini pengembangan kedepannya dari layanan ini akan lebih mudah dan cepat.

b. Pemisahan Backend & Frontend dengan API

Langkah selanjutnya membangun sebuah REST API yang dapat diakses dari sisi backend dan frontend. Hal ini dapat mengurangi beban server, beban sistem dan juga meningkatkan kemudahan pengembangan layanan ini kedepannya. Jadi Frontend dan Backend tidak terhubung secara langsung melainkan menggunakan perantara API yang kemudian akan terhubung ke database

c. Penulisan Ulang Aplikasi

Selanjutnya dari sisi pemprograman, dilakukan pengecekan ulang terhadap baris kode yang sudah ditulis untuk memastikan tidak ada baris yang menyebabkan pemborosan sumber daya maupun galat pada sistem secara meyeluruh.

SAKTI menggunakan platform web yang terhitung cukup mudah untuk melakukan proses pengecekan ulang. Jadi pemilihan bahasa pemprograman juga menjadi faktor pendukung dalam tahapan ini.

d. Optimasi Query

Selanjutnya kita perlu memastikan bahwa query atau proses pemanggilan data dari database. Seringkali terjadi kesalahan pemilihan query yang menyebabkan proses yang tidak optmal dan membuang banyak sumber daya dan memakan banyak waktu eksekusi.

e. Containerization dengan docker

Setelah membagi layanan ke beberapa layanan berbeda selanjutnya membedakan instance atau server dari layanan. Hal ini perlu dilakukan untuk memastikan bahwa layanan KRS daring benar-benar independen dan terisolasi. Sehingga layanan dapat berjalan lebih optimal karena terisolasi dalam sebuah instance mandiri.

f. Load Balancing

Setelah instance dipisahkan selanjutnya proses load balancing dan manajerial sumber-sumber yang digunakan oleh layanan KRS. Setelah dipisahkan maka proses pemanggilan sumber daya secara terpisah dan lebih cepat.

3.2 Sumber Data

Data didapatkan dari analisis log server dan layanan, serta survei ke beberapa mahasiswa yang sudah pernah menggunakan layanan dan mengalami galat ataupun pelambatan sistem.

3.3 Teknik Pengambilan Data

Teknik pengambilan data menggunakan sampling log dan survei ke beberapa mahasiswa.

3.4 Analisa Data

Analisis dilakukan sebelum layanan dilakukan dan setelah layanan berjalan sesuai dengan pendekatan dan instrument yang digunakan dalam penelitian ini.

BAB IV

PENUTUP

4.1 KESIMPULAN

Penerapan arsitektur microservice yang kurang tepat dapat mempersulit pengelolaan dan pengembangan aplikasi kedepannya. Penggunakan arsitektur harus disesuaikan dengan kebutuhan bukan merubah secara keseluruhan. Perubahan secara keseluruhan akan mempersulit pengelolaan, pengerjaan dan beberapa hal yang bersifat teknis seperti instance dan database.

4.2 SARAN

Untuk penerapannya disarankan untuk mengimpelementasikan arsitektur secara parsial sesuai dengan bagian yang memang membutuhkan layanan secara mandiri.

DAFTAR PUSTAKA

Richardson, Chris. (2018) Microservice Pattern With Java Example, 34-71. Manning:San Franscisco

“Istiadi,Muhammad Abrar”, (2020) Penguatan KRS Online Demi Kelancaran KRS War https://medium.com/ictipb-tech/penguatan-krs-online-demi- kelancaran-krs- war-7a712aa79de5( diakses 14 januari 2020).

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dewasa ini pengolahan data di sebuah instansi menjadi semakin penting dan krusial di era digital ini. Pengolahan data yang baik mencerminkan bagaimana instansi ataupun organisasi tersebut bekerja, secara garis besar integritas data juga mempengaruhi integritas sebuah organisasi. Seiring perkembangan zaman, pengolahan data juga semakin modern dengan pengolahan data dengan sebuah sistem yang telah dikomputerisasi sehingga dapat lebih efektif dan cepat.

Pengolahan data secara komputerisasi juga diterapkan di berbagai bidang seperti pabrik, perkantoran dan instansi pendidikan. Penggunaan sistem informasi juga diaplikasikan dalam pengolahan data akademik pada sebuah instansi pendidikan mulai dari pendidikan dasar hingga perguruan tinggi. Sistem Informasi Akademik atau juga disebut SIAKAD mengelola berbagai data yang berkaitan dengan perihal akademik dalam sebuah instansi pendidikan.

Penggunaan SIAKAD dalam sebuah instansi pendidikan terbukti mempermudah pekerjaan administrasi di sebuah instansi kampus. Sistem tersebut dapat mengelola berbagai kebutuhan mulai dari keuangan hingga pengelolaan nilai dan dokumen-dokumen yang berhubungan dengan aktifitas perkuliahan. SIAKAD dapat diakses dari manapun dan kapanpun, jadi dengan pengaplikasian sistem tersebut dapat meningkatkan ketersediaan dan produktifitas dari staff kampus tersebut.

Namun, pengguna aplikasi ini bukan saja staff dari instansi tersebut, mahasiswa juga menjadi pengguna dari aplikasi tersebut. Dengan SIAKAD mahasiswa dapat melakukan beberapa prosedur administrasi secara daring dan langsung dan mempermudah pekerjaan staff akademik. Sebagai contoh kegiatan seperti KRS, mahasiswa yang mengajukan KRS secara daring juga akan diberi persetujuan secara online pula melalui sistem.

Penerapan berbagai kegiatan perkuliahan melalui sebuah sistem memang sangat mempermudah pekerjaan, akan tetapi ada harga yang harus dibayar dengan harga yang pantas. Penerapan SIAKAD untuk proses administrasi membutuhkan sumber daya yang sangat besar dan kompleks. Hal ini menjadi sebuah kekurangan tersendiri untuk penerapan SIAKAD disebuah kampus, saat sistem diakses secara bersamaan beban akan semakin besar dan bisa menyebabkan galat pada sistem.

SIAKAD pada umumnya menggunakan arsitektur monolith yang memusatkan sistem pada satu bagian utuh yang besar dan kompleks. Hal ini menyebabkan jika satu layanan yang ada di dalam SIAKAD mengalami galat maka seluruh sistem dapat mengalami galat secara bersamaan dan membuat sistem tidak berjalan.

Namun galat tidak semata-mata dikarenakan arsitektur dari aplikasi tersebu, ada beberapa hal teknis lain yang juga berdampak terhadap sistem. Dalam penelitian ini pengarang mengusulkan sebuah konsep pengembangan yang menerapkan arsitektur microservices untuk SIAKAD di STIKI MALANG yang disebut dengan SAKTI. Diharapkan dengan penerapan arsitektur yang tepat dapat meningkatkan ketersediaan dari SIAKAD itu sendiri.

1.2 Rumusan Masalah

1. Bagaimana cara menerapkan arsitektur microservice pada SAKTI STIKI?

2. Bagaimana menerapkan arsitektur microservice pada sebuah sistem informasi ?

3. Bagaimana penerapan arsitektur microservice yang tepat?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah untuk menemukan metode penerapan arsitektur microservice yang tepat pada sebuah Sistem Informasi Akademik dengan mempertimbangkan faktor-faktor tertentu untuk menyesuaikan teknik yang akan digunakan

1.4 Batasan Masalah

Batasan masalah dalam penelitian ini sebagai berikut:

1. Penerapan arsitektur hanya diterpakan pada Sistem Informasi Akademik STIKI

2. Penelitian ini hanya berfokus pada arsitektur dan layanan yang ada pada Sistem Informasi akademik STIKI

1.5 Manfaat penelitian

Peneliti berharap hasil penelitian ini dapat dijadikan dasar untuk pengembangan SAKTI STIKI kedepannya agar dapat memberikan layanan yang lebih baik lagi dari sebelumnya

BAB II

LANDASAN TEORI

2.1 Microservice

Microservice adalah sebuah model arsitektur pengembangan aplikasi yang membagi modul aplikasi berdasarkan layanan yang diberikan. Berkebalikan dengan arsitektur monolith yang menjadikan sistem sebagai satu bagian utuh dari sebuah aplikasi sehingga terkesan berat dan sangat kompleks. Microservice memiliki kelebihan dari sisi pembagian daya dan pengembangan yang lebih cepat dan pembagian yang jelas sesuai dengan layanan yang ada.(Richardson, 2018)

2.2 API

API kependekan dari (Application Programming Interface) adalah sebuah layanan yang memperbolehkan setiap aplikasi dengan arsitektur dan bahasa pemprograman yang berbeda-beda dapat berkomunikasi. Jadi API dapat dibayangkan sebagai sebuah bahasa universal yang dapat digunakan oleh semua aplikasi dengan platform dan bahasa pemprrograman yang berbeda. API sendiri digunakan untuk bertukar data dan mengirim data. (Richardson, 2018)

2.3 Load Balancing

Load balancing adalah metode management request pada tingkat jaringan dan server. Load balancing digunakan untuk memeratakan akses request pada server sehingga dapat menghemat penggunaan sumber data dan mempercepat akses dari layanan yang diberikan. Dengan begitu layanan dapat digunakan lebih lama(Richardson, 2018)

2.4 Virtualisasi Berbasis Container

Virtualisasi berbasis container adalah virtualisasi pada level Operasi Sistem. Jadi pada virtualisasi ini sistem akan membuat lingkungan yang serupa dengan Operasi Sistem yang kita tuju tanpa memperhatikan level infrastruktur. Dengan virtualisasi berbasis container selain mempermudah proses isolasi aplikasi juga mempermudah management infrastruktur (Richardson, 2018)

2.5 Orkestrasi Container

Orkestrasi container adalah sebuah alat yang digunakan untuk mempermudah manajerial instance yang digunakan. Dengan orkestrasi juga bisa dilakukan automatisasi dan backup secara berkala dan otomatis. (Richardson, 2018)

2.6 Database

Database adalah representasi digital dari pengolahan data yang dikomputerisasi. Jadi database bisa dibilang adalah kumpulan data digital yang dikumpulkan dalam bentuk tabel-tabel yang memiliki relasi antar satu sama lain.

BAB III

METODE PERANCANGAN

3.1 Sistematika Perancangan

3.1.1 Kasus Galat Sistem Pada KRS daring SAKTI STIKI

Kegagalan sistem pada saat proses pengambilan KRS adalah sebuah hal yang sering sekali ditemui oleh mahasiswa khususnya pada saat pergantian semester. Menurut pengamatan peneliti, kejadian tersebut selalu terjadi berulang-ulang dikarenakan saat diakses secara bersamaan server mengalami load yang ringgi sehingga membebani sistem dan menyebabkan kegagalan sistem. Keadaan tersebut diperparah dengan KRS daring yang sifatnya seperti battle royale yang berdampak pada adu cepat antar mahasiswa.

Jika digambarkan kejadian ini hamper sama dengan flash sale yang biasa dilakukan oleh market place pada saat promo. Jadi bisa dibayangkan bagaimana besarnya request yang diterima oleh server. Karena itu kejadian galat sistem selalu terjadi pada saar KRS Daring setiap tahunnya.

3.1.2 Proses Pemetaan Pelayanan

Secara garis besar ada beberapa layanan yang diberikan oleh SAKTI STIKI, yaitu ;

a. Keuangan

b. Mata Kuliah

c. Kehadiran

d. Akademik

e. Penilaian Mahasiswa

Layanan tersebut belum termasuk dengan beberapa sistem yang juga terintegrasi dengan beberapa layanan pihak ke-3 yang digunakan oleh kampus STIKI.

3.1.2 Proses Pembagian Layanan

Setelah didapatkan layanan apa saja yang digunakan pada SAKTI, maka didapatkan penggunaan seperti di bawah ini;

Layanan

Jumlah Pengguna

Rata-rata/bulan

presentase error

Rata-rata waktu

Keuangan

20

0.0075%

100ms

Mata Kuliah

100

0.05%

1230ms

Kehadiran

30

0.01%

200ms

Akademik

20

0.0155%

1350ms

Penilaian Mahasiswa

10

0.0003%

150ms

Tabel 1.1 adalah table perbandingan layanan dan tingkat kegalatan

Dari tabel diatas dapat disimpulkan layanan Mata Kuliah dan Akademik adalah 2 layanan yang paling sering mengalami galat dan banyak pengakses. Maka dari sini kita dapatkan menyimpulkan layanan mana yang paling mengalami galat dan dapat diperbaiki.

Pada dasarnya merefactor sebuah aplikasi monolith ke arsitektur microservice dapat dilakukan secara parsial dan hanya sebagian. Jadi dari dasar data di atas dapat disimpulkan cukup layanan yang berhubungan dengan KRS yang perlu di-refactor menjadi microservices.

3.1.3 Implementasi

Dalam Implementasi ada tahapan yang peneliti lalui untuk membuktikan teori yang digunakan, yaitu ;

a. Pemisahan KRS daring dari SAKTI STIKI

Langkah pertama yang dilakukan adalah memisahkan layanan KRS Daring dari SAKTI STIKI dan mengembangkannya dengan arsitektur microservices. Dengan begini layanan akan bersifat independen dan tidak bergantung pada sistem utama SAKTI STIKI.

Tahapan lebih lanjut pemisahan layanan akan dijelaskan pada tahapan selanjunya. Pada prinsipnya kunci dari tahapan ini adalah memisahkan layanan kecil dari layanan utama. Dengan begini pengembangan kedepannya dari layanan ini akan lebih mudah dan cepat.

b. Pemisahan Backend & Frontend dengan API

Langkah selanjutnya membangun sebuah REST API yang dapat diakses dari sisi backend dan frontend. Hal ini dapat mengurangi beban server, beban sistem dan juga meningkatkan kemudahan pengembangan layanan ini kedepannya. Jadi Frontend dan Backend tidak terhubung secara langsung melainkan menggunakan perantara API yang kemudian akan terhubung ke database

c. Penulisan Ulang Aplikasi

Selanjutnya dari sisi pemprograman, dilakukan pengecekan ulang terhadap baris kode yang sudah ditulis untuk memastikan tidak ada baris yang menyebabkan pemborosan sumber daya maupun galat pada sistem secara meyeluruh.

SAKTI menggunakan platform web yang terhitung cukup mudah untuk melakukan proses pengecekan ulang. Jadi pemilihan bahasa pemprograman juga menjadi faktor pendukung dalam tahapan ini.

d. Optimasi Query

Selanjutnya kita perlu memastikan bahwa query atau proses pemanggilan data dari database. Seringkali terjadi kesalahan pemilihan query yang menyebabkan proses yang tidak optmal dan membuang banyak sumber daya dan memakan banyak waktu eksekusi.

e. Containerization dengan docker

Setelah membagi layanan ke beberapa layanan berbeda selanjutnya membedakan instance atau server dari layanan. Hal ini perlu dilakukan untuk memastikan bahwa layanan KRS daring benar-benar independen dan terisolasi. Sehingga layanan dapat berjalan lebih optimal karena terisolasi dalam sebuah instance mandiri.

f. Load Balancing

Setelah instance dipisahkan selanjutnya proses load balancing dan manajerial sumber-sumber yang digunakan oleh layanan KRS. Setelah dipisahkan maka proses pemanggilan sumber daya secara terpisah dan lebih cepat.

3.2 Sumber Data

Data didapatkan dari analisis log server dan layanan, serta survei ke beberapa mahasiswa yang sudah pernah menggunakan layanan dan mengalami galat ataupun pelambatan sistem.

3.3 Teknik Pengambilan Data

Teknik pengambilan data menggunakan sampling log dan survei ke beberapa mahasiswa.

3.4 Analisa Data

Analisis dilakukan sebelum layanan dilakukan dan setelah layanan berjalan sesuai dengan pendekatan dan instrument yang digunakan dalam penelitian ini.

BAB IV

PENUTUP

4.1 KESIMPULAN

Penerapan arsitektur microservice yang kurang tepat dapat mempersulit pengelolaan dan pengembangan aplikasi kedepannya. Penggunakan arsitektur harus disesuaikan dengan kebutuhan bukan merubah secara keseluruhan. Perubahan secara keseluruhan akan mempersulit pengelolaan, pengerjaan dan beberapa hal yang bersifat teknis seperti instance dan database.

4.2 SARAN

Untuk penerapannya disarankan untuk mengimpelementasikan arsitektur secara parsial sesuai dengan bagian yang memang membutuhkan layanan secara mandiri.

DAFTAR PUSTAKA

Richardson, Chris. (2018) Microservice Pattern With Java Example, 34-71. Manning:San Franscisco

“Istiadi,Muhammad Abrar”, (2020) Penguatan KRS Online Demi Kelancaran KRS War https://medium.com/ictipb-tech/penguatan-krs-online-demi- kelancaran-krs- war-7a712aa79de5( diakses 14 januari 2020).