BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Penelitian terkait telah dilakukan oleh Pribadi (2014) yang menganggap ketersediaan data atau layanan sangat dibutuhkan. Oleh karena itu, *server* membutuhkan cadangan yang dapat menangani ketika terjadi kerusakan data ataupun layanan. Pada penelitian ini, penulis mengimplementasikan teknologi *High-Availability* dengan menciptakan redundasi dalam setiap sistem dan subsistem untuk memastikan data atau layanan tetap tersedia. Penulis menemukan bahwa teknologi *High-Availability* dapat menjaga ketersedian data atau layanan karena server kedua menangani kerusakan pada *server* utama.

Penelitian selanjutnya dilakukan oleh Juliharta, Supedana dan Hostiadi (2015) yang membahas ketersediaan layanan suatu *website* untuk memberikan informasi sangat dibutuhkan. Penulis membangun sebuah *failover clustering* yang terdiri dari dua *web server* yaitu *server* aktif dan *server* pasif. Kedua *server* mereplikasi data agar tidak mengalami kehilangan data ketika terjadi kegagalan pada perangkat penyimpanan dari *web server*. Berdasarkan hasil penelitian, sistem yang dibangun dapat memastikan ketersediaan data dari *website* tetap terjaga dengan adanya *cluster* yang menggunakan redundansi data. *Client* dari *website* pun tidak mendapatkan gangguan untuk mengakses *website* ketika terjadi permasalahan pada salah satu *web server*.

Penelitian selanjutnya dilakukan oleh Muchtar, Sadjad dan Niswar (2014) dengan judul "Implementasi *Failover Clustering* pada Dua Platform yang Berbeda Untuk Mengatasi Kegagalan Fungsi *Server*". Tujuan penelitian adalah untuk menguji tingkat *availability server* dengan cara menghitung persentase dari *availability*. Penulis juga menguji performa ketika mengimplementasikan *failover* dengan parameter *packet loss* dan *latency*. Peneliti melakukan pengujian *packet loss* dan *delay* dengan cara melakukan *ping* kepada *server*.

2.2 Dasar Teori

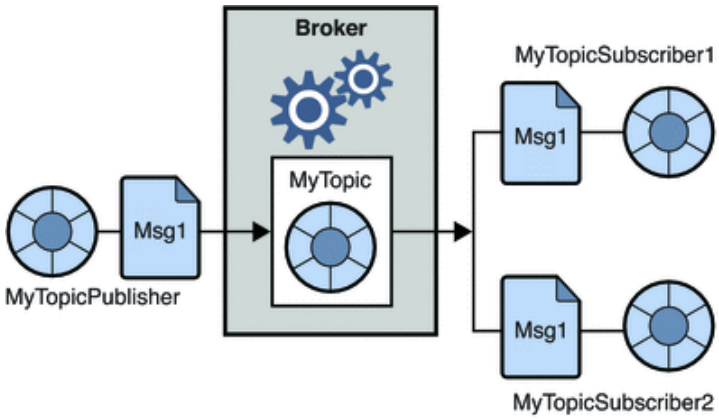
2.2.1 *Publish/Subscribe*

Publish/subscribe adalah interaksi yang digunakan untuk mendistribusikan informasi atau data. Prinsip dari *publish*/*subscribe* adalah adanya komponen yang mendaftarkan *interest* untuk mendapatkan suatu informasi. Proses mendaftarkan *interest* dinamakan dengan *subscription*, dan komponen yang berperan disebut *subscriber*. Sementara proses mempublikasikan informasi dinamakan *publish*, dan komponen yang berperan disebut *publisher*. Komunikasi data antara *publisher* dan *subscriber* dikoordinasikan oleh entitas yang disebut *broker*. *Broker* memastikan data dapat tersampaikan dari *publisher* kepada *subscriber* (Hunkeler, Truong & Stanford-Clark, 2008).

Dalam proses komunikasi antara *publisher* dan *subscriber*, pesan yang disampaikan memiliki sifat anonim. Maksud dari anonim itu adalah *publisher*

sebagai pengirim pesan tidak perlu mengetahui identitas dan keberadaan dari *subscriber*-nya. Komunikasi antara *publisher* dan *subscriber* juga terjadi secara asinkron. Makna dari asinkron adalah komunikasi tidak terjadi secara langsung tetapi melalui perantara *broker* (Hayun & Wibisono, 2017). Dalam sistem ini, seorang *client* dapat berperan menjadi *publisher* ataupun *subscriber* pada waktu yang sama. Ketika seorang *client* menyebarkan informasi, maka *client* tersebut dikatakan sebagai *publisher* dan ketika seorang *client* mendaftarkan *interest* pada suatu informasi atau data, maka *client* tersebut disebut sebagai *subscriber*.

Metode *publish/subscribe* memiliki 3 mekanisme yaitu *space decoupling*, *time decoupling*, dan *Synchronization decoupling*. *Space decoupling* memiliki makna yaitu interaksi antara *publisher* dan *subscriber* terjadi secara anonim atau tidak saling mengetahui satu sama lain. *Time decoupling* memiliki makna yaitu antara *publisher* dan *subscriber* tidak harus aktif pada waktu yang sama. Sehingga *publisher* dapat melakukan publikasi meskipun *subscriber* sedang tidak aktif dan *subscriber* dapat menerima informasi dari *publisher* ketika aktif. *Synchronization decoupling* memiliki makna yaitu komunikasi yang terjadi antara *publisher* dan *subscriber* terjadi secara asinkronus (Hayun & Wibisono, 2017).



Gambar 2.1 Model metode *publish/subscribe*

Sumber: docs.oracle.com

2.2.2 *Message Queueing Telemetry Transport* (MQTT)

MQTT adalah protokol untuk berkirim pesan yang muncul pada tahun 1999 oleh Andy Stanford-Clark dari IBM yang berkolaborasi dengan Alen Nipper dari Eurotech. Protokol MQTT menghubungkan *machine-to-machine* atau *Internet of Things (IoT)*. MQTT memiliki sifat yaitu sangat ringan dan mudah diimplementasikan. Dari karakteristiknya, MQTT sangat ideal untuk digunakan pada kondisi jaringan yang terbatas dalam hal *bandwidth*, prosesor, dan memori (Tarigan, Sitepu & Hutagalung, 2014). Protokol ini juga menawarkan kepastian dan keamanan dalam hal pengiriman pesan.

Protokol MQTT adalah protokol pada *layer* aplikasi yang didesain untuk perangkat dengan *resource* terbatas. MQTT menggunakan *Transmission Control Protocol (*TCP*)* dan IP pada *layer* dibawahnya. MQTT memiliki desain dengan *overhead* protokol yang lebih rendah apabila dibandingkan dengan *Hypertext*

Transfer Protocol (HTTP) (Thangavel, et al., 2014). MQTT memastikan pesan yang dikirimkan dapat diterima oleh *receiver* karena menggunakan TCP pada *network layer*. Protokol MQTT menggunakan 14 jenis tipe pesan yang digunakan dalam berkomunikasi yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 Jenis tipe pesan MQTT

Mnemonic	No.	Deskripsi
CONNECT	1	*Client request to connect to server*
CONNACK	2	*Connect Acknowledgment*
PUBLISH	3	*Publish Message*
PUBACK	4	*Publish Acknowledgment*
PUBREC	5	*Publish Received-assured delivery part 1*
PUBREL	6	*Publish Release-assured delivery part 2*
PUBCOMP	7	*Publish Completed-assured delivery part 1*
SUBSCRIBE	8	*Client Subscribe request*
SUBACK	9	*Subscribe Acknowledgment*
UNSUBSCRIBE	10	*Client Unsubscribe request*
UNSUBACK	11	*Unsubscribe Acknowledgment*
PINGREQ	12	*PING Request*
PINGRESP	13	*PING Response*
DISCONNECT	14	*Client is Disconnecting*

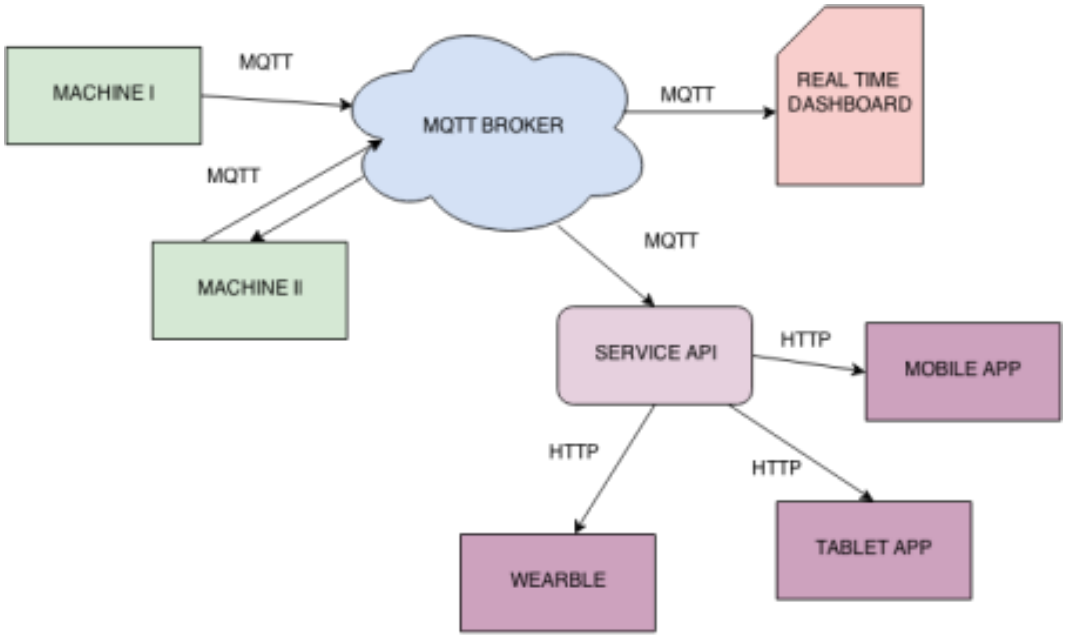
Sumber: Tarigan, Sitepu & Hutagalung. (2014)

Protokol MQTT menggunakan metode *publish/subscribe* untuk berkomunikasi. MQTT menggunakan arsitektur *topic-based publish/subscribe* dan karakter *string* untuk memberikan dukungan dari hierarki suatu topik. MQTT didesain dengan cara agar implementasi pada sisi *client* menjadi sangat mudah. Kompleksitas pada sistem ini ada pada sisi broker. MQTT tidak menspesifikasikan suatu *routing* atau teknik jaringan. Maknanya adalah jaringan dibawahnya mendasari *point-to-point*, *session-oriented*, *auto-segmenting* layanan transportasi data yang sesuai seperti TCP/IP dan menggunakan layanan ini untuk pertukaran pesan (Hunkeler, Truong & Stanford-Clark, 2008).

Protokol MQTT mendukung *basic end-to-end Quality of Service (QoS)* dalam proses pengiriman pesannya. MQTT menggunakan 3 level *QoS* yang berbeda. *QoS* level 0 adalah yang paling sederhana. *QoS* level 0 melakukan pengiriman sekali atau tidak sama sekali kepada tujuan. *QoS* level 0 Tidak melakukan pengiriman ulang atau pendefinisian *acknowledgment*. *QoS* level 1 menyediakan pengiriman yang lebih terpecaya. Pada *QoS* level 1, pesan dikirimkan berulang-ulang sampai diterima oleh penerima. Oleh karena itu, pesan mungkin sampai

beberapa kali pada penerima karena adanya *retransmission*. Level tertinggi dari *QoS* adalah *QoS* level 2 yang memastikan pesan dapat diterima dan juga pesan dikirimkan hanya sekali kepada penerima (Hunkeler, Truong & Stanford-Clark, 2008).

Karena karakteristiknya, protokol MQTT banyak digunakan dalam perangkat *mobile*. Selain itu, protokol ini dapat digunakan pada perangkat sensor, sistem *embedded* ataupun komputer dalam skala besar. Dan saat ini, protokol MQTT banyak digunakan pada perangkat *IoT* karena ringan dan mengadopsi pola pengiriman pesan *publish/subscribe*.



Gambar 2.2 Arsitektur MQTT

Sumber: thejackalof*java*script.com

2.2.3 *Broker Cluster*

Broker Cluster adalah sekumpulan *broker* yang bekerja sama untuk menyediakan pengiriman pesan kepada *client*. *Cluster* menggunakan layanan *Message Queue* untuk mengukur proses berkirim pesan dengan mendistribusikan konektivitas *client* kepada beberapa *broker*. Karena adanya beberapa *broker* dalam *cluster*, *cluster* melindungi kegagalan dari salah satu broker.

Message Queue mendukung dua model *clustering*. Masing-masing menyediakan perbedaan level dari ketersediaan layanan pesan:

- ***Conventional Broker Clusters***

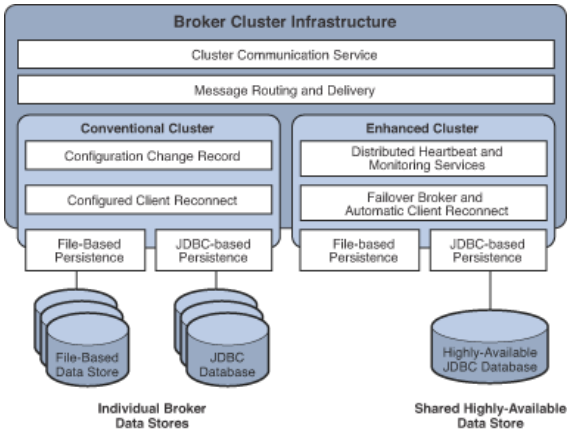
Model *conventional broker cluster* menyediakan *service availability*. Ketika *broker* atau koneksi gagal, *client* yang terhubung akan melakukan *reconnect* pada *broker* lain dalam *cluster*. Namun, pesan dan informasi yang tersimpan pada

broker yang gagal tidak dapat di *recovery* hingga *broker* tersebut aktif kembali. *Broker* atau koneksi yang gagal dapat mengakibatkan *delay* yang signifikan.

- ***Enhanced Broker Clusters***

Model *Enhanced Broker Clusters* menyediakan *data* dan *service availability*. Ketika *broker* atau koneksi gagal, *broker* lain mengambil seluruh proses yang tertunda pada *broker* yang gagal. *Failover broker* memiliki akses pada pesan dan informasi dari *broker* yang gagal. *Client* akan melakukan *reconnect* kepada *failover broker*. Dibandingkan dengan model *cluster* konvensional, model ini akan mengirimkan pesan secepat mungkin dari *broker* yang gagal melalui *failover broker*.

Kedua model dari *broker cluster* dibangun dengan infrastruktur dan mekanisme pengiriman pesan yang sama. Perbedaan nya hanyalah bagaimana *broker* dalam *cluster* melakukan sinkronisasi serta bagaimana *cluster* mendeteksi dan merespons *failures*. Perbedaan dari model *broker cluster* ditunjukkan pada Gambar 2.3. Perbedaan keduanya dijelaskan dalam Tabel 2.2 yang meringkaskan perbedaan dari kedua model *cluster* secara fungsional.



Gambar 2.3 Perbedaan infrastruktur model *broker cluster*

Sumber: docs.oracle.com

Tabel 2.2 Perbedaan model *broker cluster*

Fungsionalitas	*Conventional*	*Enhanced*
Performance	Lebih cepat dibandingkan model *cluster Enhanced*.	Lebih lambat dibandingkan model *cluster Conventional*.
Service Availability	Iya, tapi beberapa operasi tidak dapat dilakukan apabila *master broker down*.	Iya.
Data Availability	Tidak. Informasi pada	Iya setiap waktu.

	broker yang gagal tidak tersedia sampai *broker* aktif kembali.	
Recovery at Failover	Tidak. Pesan dan kondisi yang ada tidak tersedia sampai *broker* yang gagal aktif kembali. Ketika *client reconnect, client* mungkin tidak bisa apabila kesalahan terjadi saat transaksi dilakukan.	Iya. Pesan dan kondisi dari *broker* yang gagal diambil alih, transaksi yang belum siap akan dikembalikan dan pesan akan diproses untuk pengiriman. Apabila kegagalan terjadi saat transaksi dilakukan, maka pemberitahuan akan diberikan untuk menunjukkan bahwa transaksi tidak dapat dilakukan.
Configuration	Mengatur konfigurasi yang tepat untuk masing-masing *broker*.	Mengatur konfigurasi yang tepat untuk masing-masing *broker*.
Additional Requirements	Tidak ada.	*Database* yang selalu tersedia.
Restricted to Subnet	Tidak.	Iya.

Sumber: docs.oracle.com

2.2.4 *Failover*

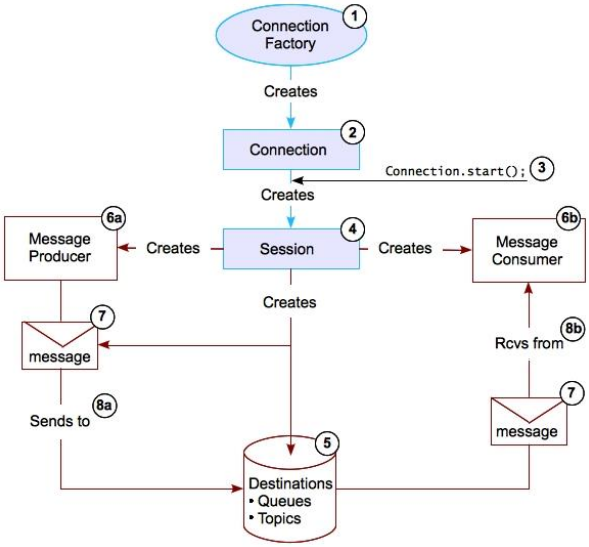
Failover atau juga disebut *High-Availability clusters* adalah grup dari beberapa komputer yang mendukung aplikasi *server* dengan *downtime* yang minimal karena adanya grup yang terdiri dari beberapa *server*. *Failover* bekerja dengan memanfaatkan beberapa komputer dalam suatu grup atau *cluster* yang akan memberikan layanan secara kontinyu ketika komponen dari sistem gagal. Tanpa *clustering*, server yang menjalankan suatu aplikasi dan *crash*, maka aplikasi tidak bisa digunakan hingga server yang *crash* diperbaiki. *Cluster* dengan jenis seperti itu dapat mendeteksi kesalahan pada *hardware*/*software* dan segera *restart* kembali aplikasi pada sistem lain tanpa membutuhkan intervensi administratif disebut dengan *failover* (Kahanwal & Singh, 2012).

Elemen dari *cluster* bekerja dengan adanya beberapa *node* redundan yang menyediakan layanan ketika salah satu bagian dari *cluster* mengalami kegagalan. *Node* yang digunakan minimal sebanyak dua *node* untuk melakukan redundansi. Implementasinya yaitu dengan menggunakan redundansi pada salah satu komponen *cluster* untuk menutupi kegagalan yang dialami pada satu titik (*Single Point of Failure*) (Pribadi, 2013).

2.2.5 *ActiveMQ*

ActiveMQ adalah *open source message broker* yang dimiliki oleh *Apache Software Foundation*. *ActiveMQ* ditulis dengan Bahasa *java* dan mengimplementasikan *JMS standard* versi 1.1. *Broker* dari *ActiveMQ* dapat dikolaborasikan dengan tujuan untuk meningkatkan performa dalam hal skalabilitas. Tipe kolaborasi ini dinamakan jaringan *broker* dan dapat mendukung bermacam-macam topologi. (Ionescu, 2015).

Implementasi dari aplikasi *ActiveMQ* memiliki tahapan seperti pada Gambar 2.4.



Gambar 2.4 Tahapan dalam *ActiveMQ*

Sumber: access.redhat.com

ActiveMQ mendukung performa tinggi dalam melakukan *load balancing* pada *message queue* dan dapat diandalkan. *ActiveMQ* juga memiliki tingkat *availability* yang lebih baik dibandingkan dengan menggunakan *load balancer*. Komunikasi dalam *ActiveMQ* diatur dengan fitur seperti *computer clustering* dan memiliki beberapa *mode* untuk *High-Availability*.

2.2.6 *Availability*

Availability adalah kemampuan secara fungsional dari sebuah unit dalam keadaan siap memberikan layanan atau melakukan fungsi yang dibutuhkan sesuai dengan kondisi selama interval waktu tertentu. Secara sederhana, *availability* didefinisikan dengan nilai persentase dari jumlah waktu suatu jaringan dapat memberikan pelayanan dibagi dengan jumlah waktu ekspektasi pada jaringan tersebut untuk memberikan pelayanan (Thulin, 2004). Ketika jaringan tidak dapat memberikan pelayanan, waktu tersebut dinamakan *downtime*. *Availability* dapat digambarkan sebagai keberhasilan pengiriman data dari titik A ke titik B (Thulin, 2004).

2.2.7 *Packet Loss*

Packet loss adalah persentase dari paket yang hilang dalam pengiriman. *Packet loss* mengindikasikan adanya kegagalan pada jaringan (Thulin, 2004). *Packet loss* terjadi ketika ada satu atau lebih paket data yang dikirimkan gagal mencapai tujuan nya. *Packet loss* dapat disebabkan oleh beberapa faktor yaitu, sinyal yang terlalu lemah dan adanya kesalahan pada *hardware* atau *driver* dari jaringan. *Packet loss* dapat menyebabkan terjadinya peningkatan nilai *latency*.

2.2.8 *Latency*

Latency atau disebut juga dengan *delay* adalah waktu yang dibutuhkan untuk mengirimkan suatu paket dari sumber kepada penerima (Thulin, 2004). Secara umum, *latency* dikatakan sebagai periode waktu dari satu komponen dalam sistem yang menunggu komponen lain. Oleh karena itu, *latency* dikatakan sebuah waktu yang terbuang. *Latency* menggambarkan kecepatan dan kemampuan suatu jaringan dalam mengirimkan paket data. Semakin tinggi nilai *latency*, maka semakin lambat suatu paket data dikirimkan. Sementara semakin rendah nilai *latency*, maka semakin cepat suatu paket dapat tersampaikan.