



**UNIVERSITAS INDONESIA**

**ANALISIS KINERJA CONTAINER UNTUK VIRTUAL SCREENING  
TANAMAN OBAT ASLI INDONESIA DENGAN AUTODOCK DAN  
AUTODOCK VINA**

**SKRIPSI**

**AGUNG PUTRA PASARIBU  
1106016720**

**FAKULTAS FAKULTAS ILMU KOMPUTER  
PROGRAM STUDI ILMU KOMPUTER  
DEPOK  
JUNI 2015**



**UNIVERSITAS INDONESIA**

**ANALISIS KINERJA CONTAINER UNTUK VIRTUAL SCREENING  
TANAMAN OBAT ASLI INDONESIA DENGAN AUTODOCK DAN  
AUTODOCK VINA**

**SKRIPSI**

**Diajukan sebagai salah satu syarat untuk memperoleh gelar  
Sarjana Ilmu Komputer**

**AGUNG PUTRA PASARIBU**

**1106016720**

**FAKULTAS FAKULTAS ILMU KOMPUTER**

**PROGRAM STUDI ILMU KOMPUTER**

**DEPOK**

**JUNI 2015**

## HALAMAN PERSETUJUAN

**Judul** : Analisis Kinerja Container untuk Virtual Screening Tanaman  
Obat Asli Indonesia dengan Autodock dan Autodock Vina  
**Nama** : Agung Putra Pasaribu  
**NPM** : 1106016720

Laporan Skripsi ini telah diperiksa dan disetujui.

14 Juni 2015

Muhammad Hafizhuddin Hilman, S.Kom., M.Kom.

Pembimbing Skripsi

## **HALAMAN PERNYATAAN ORISINALITAS**

**Skripsi ini adalah hasil karya saya sendiri,  
dan semua sumber baik yang dikutip maupun dirujuk  
telah saya nyatakan dengan benar.**

**Nama : Agung Putra Pasaribu**  
**NPM : 1106016720**  
**Tanda Tangan :**

**Tanggal : 14 Juni 2015**



## **HALAMAN PENGESAHAN**

Skripsi ini diajukan oleh :

Nama : Agung Putra Pasaribu

NPM : 1106016720

Program Studi : Ilmu Komputer

Judul Skripsi : Analisis Kinerja Container untuk Virtual Screening  
Tanaman Obat Asli Indonesia dengan Autodock dan  
Autodock Vina

**Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.**

## **DEWAN PENGUJI**

Pembimbing : Muhammad Hafizhuddin Hilman, S.Kom., M.Kom. ( )

Penguji : Prof. Drs. Heru Suhartanto M.Sc., Ph.D ( )

Penguji : Amril Syalim S.Kom., M.Eng. ( )

Ditetapkan di : Depok

Tanggal : 30 Juni 2015

## KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yesus Kristus karena oleh penyertaan dan berkat-Nya, penulis dapat menyelesaikan tugas akhir yang berjudul "Analisis Kinerja Container untuk Virtual Screening Tanaman Obat Asli Indonesia dengan Autodock dan Autodock Vina" dengan waktu yang telah ditetapkan. Laporan ini disusun berdasarkan penelitian dan hasil yang diperoleh selama 5 bulan terakhir.

Laporan tugas akhir ini disusun dalam rangka memenuhi persyaratan untuk memperoleh gelar Sarjana Ilmu Komputer di Universitas Indonesia.

Penulis menyadari bahwa laporan ini tidak akan tersusun dengan baik tanpa adanya bantuan dari pihak - pihak yang terkait. Dalam kesempatan ini, penulis mengucapkan terima kasih kepada semua pihak yang telah membantu penulis dalam kegiatan penelitian "Analisis Kinerja Container untuk Virtual Screening Tanaman Obat Asli Indonesia dengan Autodock dan Autodock Vina" maupun dalam penyusunan laporan ini.

Secara khusus penulis ingin menyampaikan ucapan terima kasih kepada

1. Bapak Muhammad Hafizhuddin Hilman, S.Kom, M.Kom , selaku pembimbing tugas akhir yang telah memberikan kesempatan kepada penulis untuk berkenalan dengan topik terkait, membimbing dan memotivasi penulis setiap saat tanpa jemu.
2. Bapak Ir. Ito Wasito M.Sc., Ph.D. selaku pembimbing akademik yang telah memberikan bimbingan selama 4 tahun terakhir dalam perkuliahan.
3. Bapak dan Mama, yang selalu mengingatkan dan memaksa penulis untuk menyelesaikan tugas akhir, dengan sepenuh hati dan tanpa imbalan selalu mendukung penulis dalam setiap langkah yang penulis lalui
4. Teman - teman Fasilkom angkatan 2011, khususnya "anak - anak lantai 5" yang selalu menjadi inspirasi dan motivasi dalam menyelesaikan laporan tugas akhir. Dan juga David Gamaliel Sihombing, Hillary Goretha Pasaribu, Mathias Kevin Tobing, Erwinto Kasah Banurea yang menjadi tempat berbagi suka dan duka.
5. Agnes Cnythia, Gisheila Ruth, dan Glori Teofilus yang memberikan motivasi kepada penulis untuk dapat menyelesaikan tugas dan tidak lari dari tanggung jawab.

Penulis menyadari bahwa laporan ini tidak luput dari ketidaksempurnaan. Oleh karena itu, penulis memohon saran dan kritik yang konstruktif untuk kedepannya.

Depok, 14 Juni 2015

Agung Putra Pasaribu



## HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

**Nama** : Agung Putra Pasaribu  
**NPM** : 1106016720  
**Program Studi** : Ilmu Komputer  
**Fakultas** : Fakultas Ilmu Komputer  
**Jenis Karya** : Skripsi

demikian pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Analisis Kinerja Container untuk Virtual Screening Tanaman Obat Asli Indonesia  
dengan Autodock dan Autodock Vina

berserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok  
Pada tanggal : 14 Juni 2015  
Yang menyatakan

(Agung Putra Pasaribu)

## ABSTRAK

Nama : Agung Putra Pasaribu  
Program Studi : Ilmu Komputer  
Judul : Analisis Kinerja Container untuk Virtual Screening Tana-  
man Obat Asli Indonesia dengan Autodock dan Autodock  
Vina

*Drug discovery* merupakan runtutan aktivitas yang dilakukan oleh perusahaan farmasi. Dibutuhkan waktu yang cukup lama hingga puluhan tahun dalam menemukan kandidat obat yang tepat untuk menyembuhkan penyakit. Peran serta komputer memberikan kemudahan bagi penelitian khususnya dalam *drug discovery*. Proses *molecular docking* antara *receptor* dengan *ligand* dan *virtual screening* dari kumpulan data *receptor* membutuhkan sumber daya kinerja komputasi yang tinggi. Namun, kemudahan tersebut datang dengan segi negatif. Kendala fasilitas dan dana menyebabkan perlunya solusi alternatif dalam pengadaan sumber daya komputasi tinggi. Dengan berkembangnya *cloud computing*, penggunaan *resource* komputasi tidak menjadi kendala. Optimisasi virtualisasi dalam *cloud computing* dapat memberikan solusi yang menjanjikan. Docker sebagai *platform* virtualisasi memberikan konsep baru dan kemudahan dalam melakukan virtualisasi. Dalam penelitian ini dibahas mengenai pemanfaatan Docker untuk menguji kinerja aplikasi Autodock versi 4.2 dan Autodock Vina versi 1.1 dalam menjalankan proses *virtual screening*. Hasil dari penelitian ini adalah analisis *running time* kedua aplikasi tersebut dalam container

**Kata Kunci:** *cloud computing*, *drug discovery*, *virtual screening*, virtualisasi, Docker

## ABSTRACT

Name : Agung Putra Pasaribu  
Program : Ilmu Komputer  
Title : Analyzing Container's Performance for Virtual Screening of Indonesian Medicinal Plants with Autodock and Autodock Vina

Drug discovery is a sequence activities undertaken by pharmaceutical companies. It takes long time in finding the right candidate drug to cure disease. The role of the computer makes it easy for research, especially in drug discovery. The process of molecular docking with matching receptor with ligand and virtual screening of the data collection of receptor requires high computational resources. However, such convenience comes with negative terms. Facilities and funding constraints led to the need for alternative solutions in the procurement of high performance computing. With the development of cloud computing, the computing resource use is not an obstacle. Optimization of virtualization in cloud computing may provide a promising solution. Docker as virtualization platform provides a new concept and ease of doing virtualization. In this study addressed the use of Docker to test the performance of applications Autodock version 4.2 and Autodock Vina version 1.1 in carrying out virtual screening process. Results of this research is the analysis of the running time of the application in container.

**Keywords:** cloud computing, drug discovery, virtual screening, virtualization, Docker

## DAFTAR ISI

<b>HALAMAN JUDUL</b>	<b>i</b>
<b>LEMBAR PERSETUJUAN</b>	<b>ii</b>
<b>LEMBAR PERNYATAAN ORISINALITAS</b>	<b>iii</b>
<b>LEMBAR PENGESAHAN</b>	<b>iv</b>
<b>KATA PENGANTAR</b>	<b>v</b>
<b>LEMBAR PERSETUJUAN PUBLIKASI ILMIAH</b>	<b>vii</b>
<b>ABSTRAK</b>	<b>viii</b>
<b>Daftar Isi</b>	<b>x</b>
<b>Daftar Gambar</b>	<b>xii</b>
<b>Daftar Tabel</b>	<b>xiii</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Permasalahan . . . . .	4
1.2.1 Definisi Permasalahan . . . . .	4
1.2.2 Batasan Permasalahan . . . . .	4
1.3 Tujuan . . . . .	4
1.4 Posisi Penelitian . . . . .	4
1.5 Sistematika Penulisan . . . . .	5
<b>2 LANDASAN TEORI</b>	<b>6</b>
2.1 <i>Drug dicoverry</i> . . . . .	6
2.2 <i>Computational Drug discovery</i> . . . . .	7
2.3 <i>Molecular Docking</i> . . . . .	9
2.4 <i>Autodock dan Autodock Vina</i> . . . . .	11
2.5 <i>Kemampuan Komputasi</i> . . . . .	12
2.6 <i>Cloud Computing</i> . . . . .	12

	xii
2.7 Virtualisasi . . . . .	14
2.8 Virtual Machine . . . . .	15
2.9 Docker . . . . .	16
<b>3 METODOLOGI PENELITIAN</b>	<b>19</b>
3.1 Tahapan Penelitian . . . . .	19
3.1.1 Workflow Penelitian . . . . .	20
3.2 Sistem Pendukung Penelitian . . . . .	21
3.3 Data Penelitian . . . . .	22
3.3.1 Pemilihan Data . . . . .	22
3.3.2 Pemrosesan Data . . . . .	22
3.4 Skenario Eksperimen Penelitian . . . . .	23
3.5 Pengukuran Kinerja . . . . .	24
<b>4 HASIL PENELITIAN DAN ANALISIS</b>	<b>25</b>
4.1 Hasil Penelitian . . . . .	25
4.2 Analisis . . . . .	29
<b>5 PENUTUP</b>	<b>36</b>
5.1 Kesimpulan . . . . .	36
5.2 Kendala . . . . .	37
5.3 Saran . . . . .	37
<b>Daftar Referensi</b>	<b>38</b>
<b>LAMPIRAN</b>	<b>i</b>
<b>Lampiran A</b>	<b>xv</b>
<b>Lampiran B</b>	<b>xvii</b>
<b>Lampiran C</b>	<b>xix</b>
<b>Lampiran D</b>	<b>xxiii</b>

## DAFTAR GAMBAR

2.1	Diagram proses <i>drug discovery</i> tradisional dan modern [7] . . . . .	8
2.2	Ilustrasi <i>ligand</i> yang akan terikat dengan <i>receptor</i> [9] . . . . .	9
2.3	Contoh dari <i>pharmacophore</i> [12] . . . . .	9
2.4	<i>Proses molecular docking</i> [11] . . . . .	10
2.5	Logo Autodock [20] . . . . .	11
2.6	Layanan dalam <i>cloud computing</i> [25] . . . . .	14
2.7	Perbandingan virtualisasi pada VM dan <i>container</i> pada Docker[33] .	17
2.8	Logo Docker[34] . . . . .	17
3.1	Tahapan Penelitian . . . . .	19
3.2	<i>Workflow</i> penelitian . . . . .	21
4.1	Waktu eksekusi tercepat untuk masing - masing skenario Autodock Vina . . . . .	26
4.2	Waktu eksekusi terlama untuk masing - masing skenario Autodock Vina . . . . .	26
4.3	Grafik dari rata - rata waktu eksekusi masing - masing skenario Autodock Vina . . . . .	27
4.4	Waktu eksekusi tercepat untuk masing - masing skenario Autodock .	28
4.5	Waktu eksekusi terlama untuk masing - masing skenario Autodock .	28
4.6	Grafik rata - rata waktu eksekusi masing - masing skenario Autodock	29
4.7	Perbandingan waktu <i>running</i> pada situs resmi Autodock [19] . . . .	29

## DAFTAR TABEL

3.1	Spesifikasi PC desktop . . . . .	22
3.2	Preparasi data pada Autodock dan Autodock Vina . . . . .	23
3.3	Pembagian <i>container</i> dan data setiap <i>container</i> . . . . .	24
4.1	Hasil skenario eksperimen pada aplikasi Autodock Vina . . . . .	25
4.2	Hasil skenario pada eksperimen Autodock . . . . .	27
4.3	Waktu eksekusi Autodock dan Autodock Vina saat 1 <i>container</i> . . . . .	30
4.4	Persentasi <i>relative weight</i> pemanfaatan CPU . . . . .	31
4.5	Spesifikasi harga dan <i>resource cloud computing</i> yang disediakan oleh Amazon EC2 [2] . . . . .	32
4.6	Hasil pengukuran <i>virtual screening</i> [2] . . . . .	32
4.7	Waktu yang dibutuhkan dalam memproses 1 buah Molekul berdasarkan tabel 4.6 diatas . . . . .	32
4.8	Waktu yang dibutuhkan dalam memproses 1 buah Molekul . . . . .	32
4.9	Hasil <i>virtual screening</i> yang berjalan pada komputer <i>native</i> . . . . .	33
4.10	Biaya yang dikeluarkan oleh Bapak Muhammad H. Hilman dalam penelitiannya [2] . . . . .	33

## DAFTAR KODE

1	<i>Script</i> untuk menghitung atomic affinity maps . . . . .	xviii
2	<i>script</i> untuk mempersiapkan docking parameter file . . . . .	xix
3	<i>Script</i> yang digunakan pada <i>desktop</i> PC host dimana Docker terpasang	xx
4	<i>Script</i> yang digunakan untuk menjalankan aplikasi Autodock Vina dalam container . . . . .	xxii
5	<i>Script</i> yang digunakan untuk mengambil hasil dari menjalankan aplikasi Autodock Vina pada container . . . . .	xxii
6	<i>Script</i> yang digunakan untuk mengambil log dari masing - masing container . . . . .	xxiii
7	<i>Script</i> yang digunakan pada <i>desktop</i> PC untuk membuat container dan menjalankan container . . . . .	xxiv
8	<i>Script</i> yang dijalankan pada masing - masing container . . . . .	xxvi
9	<i>Script</i> yang digunakan untuk mengambil hasil dari menjalankan aplikasi Autodock dari masing - masing container . . . . .	xxvii
10	<i>Script</i> yang digunakan untuk mengambil log dari masing - masing container . . . . .	xxvii



# **BAB 1**

## **PENDAHULUAN**

Bab ini menjelaskan latar belakang, permasalahan, tujuan dan ruang lingkup penelitian, serta sistematika penulisan tugas akhir penelitian

### **1.1 Latar Belakang**

Perjalanan kehidupan manusia dipenuhi dengan berbagai kejadian. Manusia memerlukan usaha untuk dapat memenuhi kebutuhan dan menjaga eksistensi dari ancaman, baik itu dari dalam dan dari luar. Salah satu ancaman tersebut adalah penyakit. Perkembangan ilmu dan daya pikir manusia hanya bisa menyembuhkan tetapi tidak dapat menghapus penyakit tersebut. Muncul jenis penyakit baru yang juga memerlukan pengetahuan baru dalam mencegah penyebaran penyakit tersebut. Berbagai pengobatan dan kepercayaan dalam menyembuhkan penyakit bermunculan. Negara - negara Asia timur muncul dengan pengobatan herbal dan teknik akupuntur. Arkeolog menemukan bukti bahwa pemanfaatan tanaman obat herbal telah ada sejak 60.000 tahun silam (masa *Paleolithic*). Tulisan berumur 5.000 tahun tentang daftar tanaman obat juga ditemukan. Tulisan ini merupakan daftar herbal yang dibuat oleh penduduk Sumeria. Tidak hanya Sumeria, dalam sejarah Mesir, Yunani maupun Cina terdaftar sebagai negara yang mengembangkan pengobatan herbal. Cina tidak hanya mengembangkan herbal, mereka juga mengembangkan teknik akupuntur dalam menyembuhkan penyakit.

Seiring dengan berkembang ilmu pengetahuan dan teknologi manusia khususnya *proteomics* dan *genomics*, manusia dapat mempelajari penyakit pada tingkat molekul. Hal ini memberikan pengetahuan dan kepastian dalam pengembangan obat secara modern. *Drug discovery* merupakan runtutan skenario panjang dalam menemukan calon obat yang berpotensi. Sebelum masuk dalam *drug discovery*, dilakukan *pre-drug discovery*. Dalam tahap ini peneliti mempelajari :

- Karaktersitik penyakit tersebut
- Bagaimana penyakit tersebut mempengaruhi gen penderita
- Bagaimana gen yang terkena tersebut memproduksi protein
- Bagaimana protein tersebut berinteraksi dengan lingkungan sekitar

- Bagaimana protein tersebut mempengaruhi lapisan dimana protein tersebut berperan
- Bagaimana penyakit tersebut mempengaruhi *host* (tubuh yang terkena penyakit)

Dengan berbekal pengetahuan dari proses *pre-drug discovery* tersebut, peneliti perlu memastikan bagian yang berupa molekul tunggal (gen/protein) yang berpengaruh dalam penyakit tersebut. Setelah hal tersebut, perlu dilakukan validasi dengan cara menyuntikan "target" tersebut ke dalam sel hidup atau *speciment* percobaan. Hal ini untuk memastikan bahwa molekul target yang diambil adalah valid. Setelah molekul target valid, peneliti perlu mencari molekul yang dapat membalikan kerja molekul target. Diperlukan waktu yang cukup lama untuk mengidentifikasi molekul pembalik tersebut dan memastikan bahwa molekul tersebut dapat digunakan sebagai obat. Berdasarkan survey, waktu yang diperlukan dalam suatu *drug discovery* sekitar 10-15 tahun. Selain itu biaya yang dikeluarkan juga sangat besar (800 juta - 1 milyar dollar) dan kemungkinan gagal dalam menemukan obat baru yang berpotensi pun tetap ada.

Waktu yang dibutuhkan, biaya yang dikeluarkan, serta kemungkinan gagal dalam menemukan obat baru yang berpotensi menjadi perhatian utama dalam *drug discovery* mula - mula. Komputer turut mengambil andil dalam *drug discovery* modern. Pemanfaatan komputer tersebut menggantikan proses uji coba pencarian molekul yang dapat membalikan efek dari molekul target yang sebelumnya dilakukan dengan eksperimen terhadap makhluk hidup digantikan dengan simulasi. *Virtual screening*, yaitu mencari molekul yang dapat mengikat (*binding*) dengan molekul target dari *database* molekul yang tersimpan. Untuk memastikan molekul terikat tersebut, *virtual screening* memanfaatkan perhitungan fungsi/*scoring* untuk menentukan mana molekul yang "best-fit" dengan molekul target. Pada umumnya *virtual screening* tersebut berjalan secara otomatis dengan menggunakan program komputer. Dengan cara tersebut, para peneliti dapat memangkas waktu dan biaya yang dibutuhkan dalam *drug discovery*. Beberapa aplikasi *virtual screening* yang dapat digunakan antara lain : Auto Dock, DOCK, Gold, V Life MDS, Flex X.

*Virtual screening* yang mencakup evaluasi seluruh koleksi *database* molekul terhadap molekul target membutuhkan tenaga komputasi yang tidak sedikit. Tidak hanya itu, kapasitas memori penyimpanan untuk koleksi data molekul maupun hasil *virtual screening* yang cukup besar, sumber daya listrik yang digunakan untuk menjaga komputer tetap bekerja juga diperlukan. Kemampuan komputasi pada komputer terletak pada *processor*. Komputer akan mengenali *virtual screening* yang

dijalankan sebagai kumpulan instruksi yang harus dikerjakan oleh *processor*. Semakin cepat *processor*, semakin banyak instruksi yang dapat dikerjakan setiap detik. *Virtual screening* dapat dilakukan pada sebuah komputer yang memiliki koleksi data molekul yang besar, namun masih terdapat kendala, seberapa lama waktu yang dibutuhkan untuk memproses semua koleksi data molekul tersebut. Untuk memenuhi kebutuhan tenaga komputasi tersebut, dibutuhkan infrastruktur komputer yang dapat saling bekerja sama untuk melakukan suatu komputasi. Infrastruktur ini dikenal sebagai *supercomputer*. *Supercomputer* terdiri dari beberapa *processor* yang memiliki kemampuan komputasi yang tinggi. Komputer tersebut dapat tersebar diberbagai tempat dan dihubungkan dengan jaringan atau kumpulan komputer yang diletakan saling berdekatan pada suatu tempat. Proses kerja *supercomputer* adalah *centralization* dimana setiap *processor* mengerjakan *task* yang sama dan hasilnya akan kembali diolah oleh suatu *processor* untuk menghasilkan output akhir. Konsumsi energi yang dibutuhkan oleh *supercomputer* tergolong sangat besar dan sebagian besar hasil dari eksekusi *task* adalah panas. Dibutuhkan biaya tambahan untuk pemeliharaan *supercomputer* agar dapat terus bekerja pada suhu yang terjaga.

Besarnya biaya yang diperlukan oleh *supercomputer* merupakan kendala utama. *Grid computing* memberikan alternatif lain untuk permasalahan tersebut. Teknologi ini memanfaatkan sekumpulan komputer biasa secara fisik yang saling terhubung melalui suatu jaringan sebagai suatu kesatuan komputer (*super virtual computer*). Tidak seperti *supercomputer*, setiap komputer bekerja sama untuk menyelesaikan sebuah *task*, yang kemudian *task* tersebut akan hilang jika selesai dan tergantikan dengan *task* yang baru. Masing masing komputer penyusun tersebut akan bekerja secara parallel. Permasalahan kembali muncul ketika pengadaan komputer secara fisik yang banyak dan pengaturan jaringan yang perlu dipahami sebelum digunakan untuk *virtual screening*.

Terlepas dari pengadaan komputer secara fisik, teknologi *cloud computing* hadir sebagai solusi. Pengguna *cloud computing* tidak perlu memikirkan bagaimana biaya dan perawatan dari teknologi tersebut. Pada saat ini, pemanfaatan aplikasi bersifat *cloud*, dimana pengguna hanya disuguhkan dengan tampilan saja, sedangkan proses komputasi berada di "awan". Sifat virtualisasi yang menjadi andalan *cloud computing* membuat teknologi ini sangat murah untuk menambah komputer secara virtual. *Cloud computing* seolah - olah mampu membuat suatu kumpulan komputer yang saling bekerja sama menyelesaikan *task* seperti *grid computing*. Memaksimalkan virtualisasi merupakan kunci dalam teknologi *cloud computing*.

Teknologi virtualisasi yang ada saat ini terbentur dengan abstraksi komputer

pada level *hardware*. Pembuatan komputer virtual tersebut membutuhkan *resource* memori penyimpanan dan kemampuan komputasi yang telah ditentukan sebelumnya. Dengan begitu jumlah komputer virtual akan terbatas dengan *resource* yang dimiliki oleh *host*. Disatu sisi, kinerja virtual komputer virtual tidak akan optimal ketika terdapat komputer virtual yang tidak melakukan apa apa, sehingga alokasi *resource* komputer virtual tersebut dapat dibagi secara merata kepada komputer virtual lainnya. Docker, sebagai *platform* virtualisasi memberikan pandangan baru dalam mengoptimalkan proses virtualisasi tersebut.

## 1.2 Permasalahan

### 1.2.1 Definisi Permasalahan

Berdasarkan latar belakang yang telah dijelaskan, penulis berusaha menganalisis apakah pemanfaatan *platform* Docker pada teknologi *cloud computing* dapat diajukan sebagai solusi alternatif dari *supercomputer* dan mengukur *scalability platform* Docker. Untuk itu, penulis akan mencoba *virtual screening* dengan menggunakan aplikasi Autodock dan Autodock Vina yang telah terinstall pada *platform* Docker.

### 1.2.2 Batasan Permasalahan

Pada penelitian ini, penulis lebih berfokus kepada performa Docker dalam menjalankan aplikasi Autodock dan Autodock Vina untuk *virtual screening*. Penulis tidak akan membahas segi *virtual screening* dikarenakan keterbatasan pengetahuan yang dimiliki dalam *drug discovery*

## 1.3 Tujuan

Penelitian ini secara umum bertujuan untuk mengenalkan pemanfaatan *cloud computing* dengan *platform* Docker dalam virtualisasi komputer. Diharapkan performa yang diperoleh dapat menyaingi *supercomputer* maupun *grid computing* dalam *virtual screening*.

## 1.4 Posisi Penelitian

Penelitian ini mencari alternatif lain dari *supercomputer* dengan memanfaatkan teknologi *cloud computing*. Hasil dari penelitian ini akan dibandingkan dengan

hasil penelitian serupa yang telah dikerjakan oleh Bapak Muhammad Hafizhuddin Hilman, S.Kom., M.Kom. [1] [2]

## 1.5 Sistematika Penulisan

Penulisan ini terbagi dalam 5 bab :

- **BAB 1 PENDAHULUAN**  
Bagian ini berisikan latar belakang, permasalahan, tujuan dan ruang lingkup penelitian, serta sistematika penulisan tugas akhir penelitian.
- **BAB 2 LANDASAN TEORI**  
Bagian ini berisikan penjelasan *drug discovery* dengan cara *virtual screening* memanfaatkan teknologi informasi secara umum. Selain itu, *cloud computing* dan Docker akan dijelaskan secara detail.
- **BAB 3 METODOLOGI PENELITIAN**  
Bagian ini berisikan detail tahap-tahap penelitian dan data yang digunakan.
- **BAB 4 HASIL PENELITIAN DAN ANALISIS**  
Bagian ini berisikan hasil dari penelitian yang telah dilaksanakan dan analisis dari hasil yang diperoleh.
- **BAB 5 PENUTUP**  
Bagian ini berisikan kesimpulan penulis terkait dengan penelitian dan saran penulis dalam penelitian ke depannya.

## BAB 2

### LANDASAN TEORI

Bab ini akan menjelaskan sekilas tentang *computational drug discovery*, apa itu *cloud computing*, dan Docker sebagai platform virtualisasi komputer.

#### 2.1 *Drug discovery*

*Drug discovery* adalah suatu proses menemukan kandidat obat yang berpotensi. Aktivitas tersebut merupakan misi utama yang diemban oleh perusahaan farmasi bidang obat - obatan untuk mengerti cara kerja penyakit sehingga dapat ditemukan obat / penyembuh yang aman untuk dikonsumsi oleh pasien. Para peneliti berusaha memahami cara kerja penyakit pada level gen dan protein. Dalam hal ini, terdapat istilah "target", dimana calon potensial obat dapat bereaksi. Berikut merupakan tahapan - tahapan yang dilewati dalam *Drug discovery* [3] :

- Validasi target
- Menemukan molekul yang dapat berinteraksi dengan target
- Melakukan uji coba terhadap senyawa hasil interaksi tersebut untuk memastikan keamanan dan keberhasilan dari senyawa tersebut
- Mendapatkan izin untuk membuat dan mendistribusikan obat tersebut

Estimasi waktu yang diperlukan dalam proses ini adalah 10 - 14 tahun [4]. Secara lebih jelas tahapan yang dilakukan dalam *drug discovery* :

- ***Pre-discovery***

Pada tahapan ini peneliti bekerja untuk memahami penyakit. Dimulai dari bagaimana gen berubah sehingga mempengaruhi protein yang *diencode*, bagaimana protein tersebut berinteraksi dengan sel - sel hidup sekitar, bagaimana lapisan sel - sel hidup yang terpengaruh berubah dan bagaimana penyakit tersebut mempengaruhi pasien secara keseluruhan. Hasil yang didapat dalam tahapan ini merupakan kunci utama dalam menyelesaikan permasalahan (penyakit).

- ***Target Identification***

Setelah peneliti memahami penyakit dan elemen yang mempengaruhi

penyakit tersebut, peneliti akan menentukan "target" untuk calon obat baru. Target merupakan *single* molekul, seperti gen atau protein yang terkait dengan penyakit tersebut. Target tersebut harus dipastikan bersifat *drugable*, dimana dapat berinteraksi dengan dan dipengaruhi oleh molekul obat.

- **Target Validation**

Setelah target tersebut diidentifikasi, peneliti perlu memastikan apakah target tersebut benar - benar terlibat dalam penyakit dan bersifat *drugable*. Peneliti mengujicobakan target tersebut ke dalam sel hidup dan hewan uji coba lab.

- **Drug Discovery**

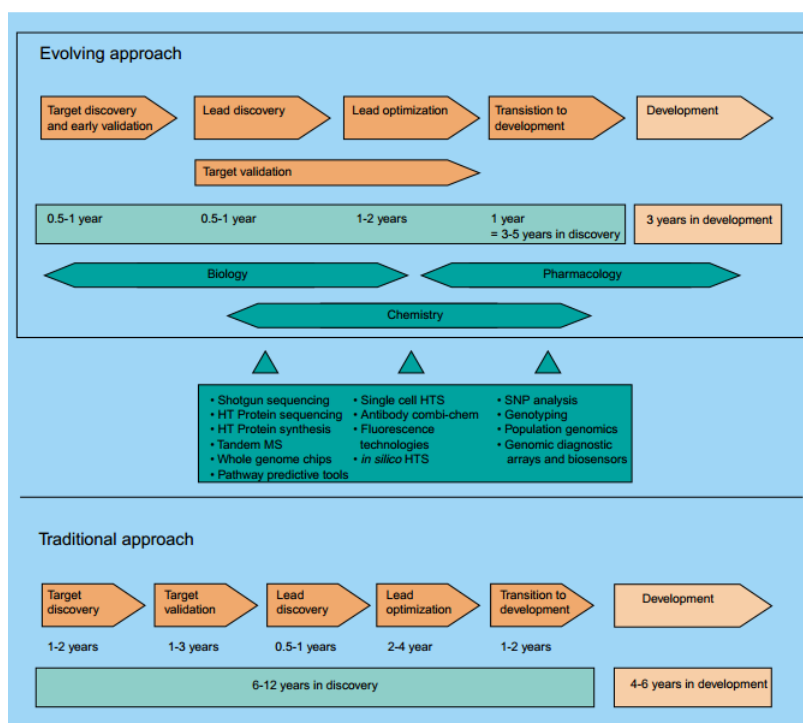
Dengan bekal pengetahuan yang didapat pada tahap sebelumnya, para peneliti akan mencari molekul yang akan bereaksi dengan target yang telah ditemukan sebelumnya dan mengembalikan efek yang disebabkan oleh target tersebut. Tahapan ini memakan separuh hingga sebagian besar waktu dari *drug discovery* tersebut. Molekul yang ditemukan dalam tahapan ini bisa menjadi suatu obat baru. Beberapa cara untuk mendapatkan molekul tersebut :

- **nature**, dengan memanfaatkan alam dan ikatan yang terkandung didalamnya. Sebagai contoh pemanfaatan bakteri dalam pengobatan.
- **De Novo**, dengan ilmu kimia dan pemanfaatan teknologi komputer, pembentukan molekul dari awal dengan memanfaatkan komputer untuk memodelkan molekul tersebut dan memprediksi sifat dari molekul tersebut.
- **High-throughput Screening**, dengan memanfaatkan tenaga komputasi komputer, memungkinkan untuk membandingkan ratusan ikatan molekul dengan target. Ikatan yang diperoleh dari perbandingan tersebut akan dipelajari kembali dikemudian hari untuk dipastikan kebenarannya.
- **Biotechnology**, dengan teknologi ini para peneliti dapat membuat mensimulasikan suatu organisasi kehidupan untuk menghasilkan molekul yang dapat melawan target.

## 2.2 Computational Drug discovery

Waktu yang dibutuhkan dalam proses *drug discovery* cukup lama. Tidak hanya itu, proses ini juga sangat beresiko dan membutuhkan dana yang besar [4].

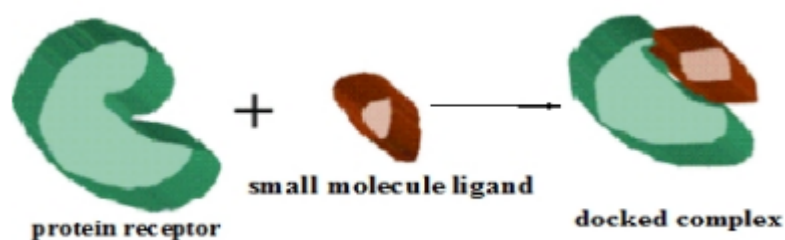
Walaupun demikian, investasi dana dalam *drug discovery* juga berkembang pesat. Namun, investasi tersebut tidak diiringi dengan hasil yang proporsional dengan investasi tersebut. Hal ini disebabkan oleh rendahnya efektivitas dan tingginya kemungkinan kegagalan dalam *drug discovery*. Beberapa pendekatan telah dilakukan dalam meningkatkan efektivitas, salah satu cara tersebut adalah CADD (*Computer Aided Drug Design*). Dengan begitu, penekanan biaya dan siklus waktu penemuan obat juga semakin cepat. Dalam CADD, komputer digunakan sebagai sarana komputasi dan penyimpanan data dalam *drug discovery* [5]. Tidak ketinggalan, CADD menggunakan aplikasi dalam mendesain ikatan kimiawi, memodelkan struktur kimia yang mengandung kandidat calon obat yang berpotensi dan pembuatan *library* yang dapat digunakan untuk riset selanjutnya [6].



**Gambar 2.1:** Diagram proses *drug discovery* tradisional dan modern [7]

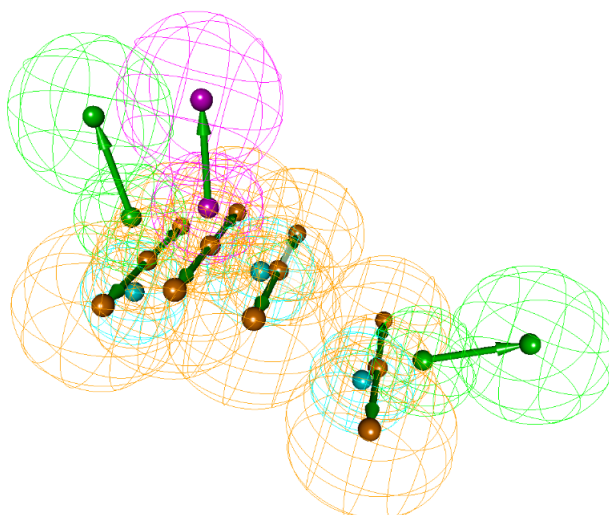
Dalam *computational drug discovery* (penemuan obat dengan memanfaatkan komputer), pendekatan yang dapat dilakukan dibagi menjadi *structure-based drug design* (SBDD), *ligand-based drug design* (LBDD) dan pendekatan berdasarkan sekuens [6]. *Structure-based drug design* (SBDD) terdiri dari proses *docking* kandidat *ligand* ke target *receptor* kemudian dilakukan penilaian dengan menggunakan *scoring function* untuk dapat menghitung kemungkinan *ligand* tersebut terikat pada target protein dengan daya tarik yang tinggi [8].





**Gambar 2.2:** Ilustrasi *ligand* yang akan terikat dengan *receptor* [9]

Dalam LBDD, diberikan sekumpulan *ligand* dan *receptor*, dimana model *receptor* dapat dibuat dengan mengumpulkan informasi dari *ligand* [10]. Model ini dikenal dengan *pharmacophore*. Kemudian, kandidat *ligand* akan dibandingkan dengan *pharmacophore*, apakah kompatibel dengan *pharmacophore* dan dapat terikat [11].



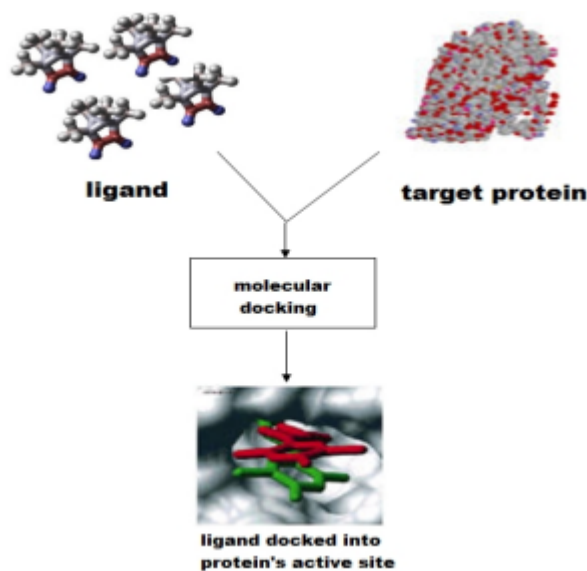
**Gambar 2.3:** Contoh dari *pharmacophore* [12]

## 2.3 Molecular Docking

*Virtual screening* berdasarkan *molecular docking* merupakan metode yang sering digunakan dalam SBDD. Penulis juga akan menggunakan metode tersebut yang terdapat dalam aplikasi Autodock dan Autodock Vina. Pemodelan struktur kimia menggunakan *molecular modeling* untuk mempelajari fenomena dari struktur kimia tersebut. Kemudahan dalam *molecular modeling* sekarang ini dibantu dengan aplikasi komputer yang tersedia, namun masih terdapat kesulitan yaitu bagaimana cara mendapatkan model struktur ikatan kimia yang benar dan interpretasi model yang tepat. Secara umum, *molecular modeling* dapat dikatakan sebagai pemanfaatan

komputer dalam mengkonstruksi molekul dan melakukan berbagai macam perhitungan untuk mempelajari karakteristik dan sifat struktur ikatan kimia. Istilah *molecular modeling* terkadang disamakan dengan istilah *computational chemistry*.

Proses *drug discovery* yang semula berupa *trial and error* berubah menjadi proses yang dibantu dengan perhitungan komputer. Perhitungan tersebut digunakan dalam menentukan struktur ikatan kimia baru berdasarkan struktur protein yang sudah diketahui [5]. Pendekatan ini terbagi menjadi dua : *de novo design* [13] dan *docking* [14]. *Docking* merupakan suatu proses untuk menebak struktur *complex* dari struktur *ligand* dan protein. Dalam *molecular modeling*, *docking* dapat dipandang sebagai suatu metode untuk memprediksi orientasi suatu molekul ketika diikat dengan molekul lainnya untuk membentuk *complex* yang stabil.



**Gambar 2.4:** Proses *molecular docking* [11]

*Molecular docking* adalah suatu proses komputasi dalam pencarian *ligand* yang paling cocok baik secara geometri maupun energi ketika diikat dengan suatu *receptor* (protein) yang telah diketahui [11]. Aspek utama dalam *molecular docking* adalah perhitungan energi interaksi dan konformasi dengan menggunakan metode dari kuantum mekanik hingga fungsi empiris energi. Permasalahan *molecular docking* sekilas dapat dilihat sebagai suatu permasalahan *lock and key*, dimana *receptor* protein dapat dipandang sebagai *lock* dan *ligand* sebagai *key*. Namun dalam praktiknya, *molecular docking* akan mencari *ligand* yang dapat menyesuaikan dengan *receptor*. Karena adaptasi tersebut, *molecular docking* dapat dipandang sebagai permasalahan "*best-fit*" *ligand* [15]. Diperlukan *scoring function* untuk dapat membatasi *best-fit* dari proses *molecular docking*. Fungsi tersebut biasanya berdasarkan *force field* yang digunakan dalam mensimulasikan protein. Beberapa *scoring func-*

tion juga menambahkan aspek perhitungan lainnya seperti entropi [11].

Dalam percobaan ini penulis memanfaatkan aplikasi *molecular docking* Autodock dan Autodock Vina dikarenakan kedua aplikasi tersebut sering dirujuk dalam karya ilmiah yang berkaitan dengan *docking* [16].

## 2.4 Autodock dan Autodock Vina

Aplikasi ini merupakan aplikasi yang dikembangkan oleh *The Scripps Research Institute*, lembaga riset nonprofit yang terletak di California, Amerika Serikat [17]. Terdapat 2 jenis aplikasi, Autodock dan Autodock Vina. Masing-masing merupakan aplikasi yang saling berbeda dalam segi *scoring function* yang digunakan dan optimisasi komputasi secara paralel oleh Autodock Vina. Autodock memanfaatkan *empirical free energy force field* dengan *Lamarckian Genetic Algorithm* dalam memprediksi koordinat ikatan antara *ligand* dan *receptor* [18]. Sedangkan dalam Autodock Vina, dilakukan optimisasi dengan memanfaatkan *particle swarm optimization* secara global dan *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* secara lokal [19]. Pada percobaan ini penulis menggunakan Autodock versi 4.2 dan Autodock Vina versi 1.1.2.

Autodock versi 4.2 terdiri dari 2 program, Autodock4 dan Autogrid4. AutoGrid4 akan menghasilkan *pre-calculated map* dari suatu *ligand*. Selain itu juga akan menghasilkan *map* tambahan, "d" untuk *desolvation* dan "e" untuk *electrostatic*. Pengoperasian Autodock membutuhkan beberapa file, yaitu : \*.dpf untuk pengaturan *docking* parameter , hasil *map* berupa \*.gpf dari AutoGrid , dan \*.pdbqt untuk *receptor* dan *ligand* yang akan diujicoba. Keluaran yang dihasilkan oleh Autodock berupa \*.dlg yang berisi hasil perhitungan konformasi posisi dan energi pada setiap *ligand* yang diujicoba. Paket aplikasi Autodock Vina hanya datang dengan sebuah program saja dikarenakan bagian *pre-processing docking* dibuat transparan sehingga pengguna tidak perlu tahu dan memahami bagaimana aplikasi tersebut bekerja. Autodock Vina membutuhkan 2 file, \*.pdbqt dari *ligand* dan \*.pdbqt dari tiap *receptor* yang akan diujicobakan. Keluaran yang dihasilkan juga berupa \*.pdbqt yang beisikan konformitas posisi dan energi.



Gambar 2.5: Logo Autodock [20]

## 2.5 Kemampuan Komputasi

*Molecular docking* yang dilakukan antara suatu *ligand* dan *receptor* akan menentukan apakah *ligand* tersebut merupakan calon dari obat yang sedang dicari. Dengan bermunculan berbagai aplikasi yang membantu dalam proses *molecular docking*, kesulitan ketika melakukan secara manual dengan menggunakan alat peraga akan berkurang serta perhitungan dan tingkat presisi akan lebih baik dibandingkan dengan perhitungan manusia. Kompleksitas dari *ligand* dan *receptor* juga akan mempengaruhi evaluasi fungsi yang dikerjakan dalam aplikasi *molecular docking*[11].

*Virtual screening* merupakan suatu teknik komputasi yang dilakukan dalam *drug discovery* dimana secara otomatis akan mengevaluasi kumpulan bank data *receptor* dengan suatu *ligand* untuk menentukan calon obat [21]. Dibutuhkan kemampuan komputasi yang dapat menjalankan eksekusi perintah dengan cepat. CPU, sebagai pusat pengolah data dan instruksi pada komputer membutuhkan *clock rate* yang cepat dan kemampuan menjalankan instruksi per *clock* yang besar. Terlepas dari kompleksitas molekul *ligand* maupun *receptor*, faktor tersebut mengambil peranan penting dalam riset yang memanfaatkan kemampuan komputasi.

Muncul infrastruktur yang dapat dikatakan memiliki kesamaan dengan *supercomputer*, *Computer Cluster*. *Computer Cluster* dapat dikatakan sebagai komputer yang saling terhubung dan dipandang sebagai satu kesatuan. Setiap anggota (*cluster*) akan mengerjakan perintah yang sama diatur dalam *software* penjadwalan. *Cluster* ini dapat membantu peneliti dalam mengerjakan risetnya dengan performa yang tidak berbeda dengan *supercomputer* [1]. Berbeda dengan *cluster*, muncul istilah *grid computing*, dimana secara struktur mirip dengan *cluster*, namun untuk setiap bagian komputer mengerjakan tugas yang berbeda beda (namun dalam satu tujuan) [22].

## 2.6 Cloud Computing

Teknologi *cloud computing* sudah berkembang sejak tahun 1990-an. Pada awalnya teknologi ini dikembangkan sebagai solusi dari permasalahan yang dihadapi oleh perusahaan dengan bertambahnya kapasitas data yang perlu disimpan dan pengeluaran yang cukup besar untuk pengadaan perangkat keras beserta konsumsi daya listrik yang dibutuhkan [23]. Definisi dari *cloud computing* dapat dikatakan sebagai virtualisasi dari perangkat komputasi, dimana pengguna mengakses perangkat tersebut dengan menggunakan jaringan internet [24]. Selain itu, teknologi ini memberikan kemudahan dalam penggunaan secara *multiuser*, dimana dapat diakses se-

cara bersamaan.

Terdapat 3 karakteristik dari teknologi *cloud computing*, yaitu virtualisasi, terdistribusi, dan kemudahan dalam pengembangan selanjutnya (*dynamically extendibility*). Virtualisasi merupakan faktor yang utama. Virtualisasi dapat membagi suatu perangkat komputasi secara fisik menjadi beberapa "virtual" perangkat komputasi. Dengan begitu, kinerja dari perangkat akan dioptimisasi. Alokasi sumber daya perangkat keras tidak ada yang diam secara percuma. Disamping itu, dengan adanya virtualisasi dapat menekan biaya pengeluaran pengadaan perangkat komputasi secara fisik. Yang dimaksud dengan terdistribusi adalah perangkat keras (*physical node*) yang digunakan dalam *cloud computing* digunakan secara terdistribusi. Dalam tahap pengembangan teknologi ini memberikan kemudahan. Kemudahan tersebut merupakan efek dari virtualisasi. Ketika pengguna ingin mengikuti perkembangan yang ada, tidak perlu merubah / membuat kembali dari awal. Pengguna dapat merubah dari level virtual (virtualisasi).

Terdapat 3 jasa yang disediakan dalam *cloud computing* [24] :

- **Software as a Service (SaaS)**

Layanan jasa ini menyediakan aplikasi dan penyimpanan data (*database*) yang langsung dapat digunakan. Pengguna dapat mengakses aplikasi secara bersamaan dengan pengguna lainnya. SaaS dikenal juga sebagai *on-demand software* dan biaya yang dikenakan oleh penyedia jasa ini merupakan *pay-per-use*. Umumnya, pengguna akan dikenakan biaya bulanan atau tahunan. Pengguna tidak perlu susah - susah untuk menginstall aplikasi pada *cloud* dikarenakan penyedia jasa tersebut akan melakukan hal tersebut. SaaS memberikan keuntungan bagi perusahaan dengan cara mengurangi biaya operasional IT (Information Technology). Hal ini dikarenakan biaya untuk pemeliharaan dan pengadaan perangkat komputasi ditanggungkan kepada pihak penyedia jasa (*outsourcing*). Contoh dari SaaS adalah layanan *e-mail*, media sosial.

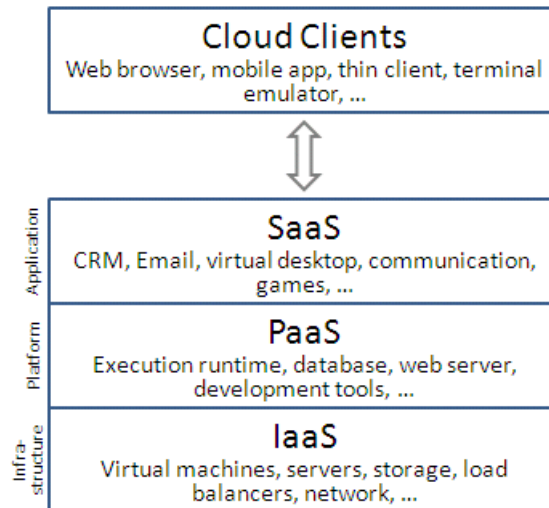
- **Platform as a Service (PaaS)**

Berbeda dengan SaaS, layanan ini menyediakan perangkat komputasi yang dapat digunakan untuk menjalankan aplikasi dari pengguna. Layanan ini dapat menyediakan sistem operasi atau *framework* yang dibutuhkan dalam menjalankan aplikasi tersebut. Contoh dari layanan jasa ini adalah : Windows Azure, Amazon Web Service, dan Google App Engine.

- **Infrastructure as a Service (IaaS)**

layanan ini menyediakan infrastruktur IT kepada pengguna sesuai dengan

permintaan yang dapat diakses dengan jaringan internet. Infrastruktur tersebut meliputi perangkat keras seperti *harddisk*, *memory*, *firewall*, tipe server, alamat IP, *virtual local area networks*(VLANs), maupun aplikasi yang harus terinstall dalam server



**Gambar 2.6:** Layanan dalam *cloud computing* [25]

## 2.7 Virtualisasi

Virtualisasi merupakan suatu konsep dimana suatu sistem komputer secara nyata dapat dijalankan lebih dari satu / *multiple* pada suatu *host*. Virtualisasi bisa berjalan dengan adanya VMM (*Virtual Machine Monitor*) yang mengatur komunikasi antara lingkungan virtualisasi dengan perangkat keras. Virtualisasi juga dapat dipandang sebagai teknik optimisasi perangkat keras sebagai sumber komputasi [26]. Terdapat 3 arsitektur dari VMM :

- **Full Virtualization**

Pada tipe ini, *hypervisor* dapat menjalankan beberapa sistem komputer virtual secara bersamaan. Setiap komputer virtual tersebut tidak perlu tahu bahwa mereka berjalan dalam lingkungan virtual

- **Para Virtualization**

Dalam tipe ini, sedikit modifikasi dilakukan dari Full Virtualization, dimana setiap sistem komputer virtual sadar bahwa mereka berjalan dalam lingkungan virtual dan sadar dengan keberadaan sistem komputer virtual lainnya. Selain itu, untuk setiap aplikasi yang dieksekusi pada masing - masing sistem komputer virtual akan langsung diteruskan ke perangkat keras tanpa harus melalui Virtual Machine Monitor

- **OS Partitioning**

Dalam tipe ini, virtualisasi dilakukan pada level OS, dimana *hypervisor* terinstall pada masing-masing OS pada sistem komputer virtual yang diinstall di atas OS host. Batasan dalam tipe ini adalah, OS pada sistem komputer virtual harus mengikuti OS pada host.

Virtualisasi digunakan pada teknologi *cloud computing* dalam mengoptimisasi kinerja server sehingga tidak ada sumber daya yang tidak terpakai. Virtualisasi yang dimaksud adalah virtualisasi perangkat keras (*hardware*) [27]. Virtualisasi tersebut memungkinkan "virtual" server yang dapat dibuat sesuai dengan kebutuhan yang diperlukan dan berjalan pada server fisik. Spesifikasi untuk masing-masing virtual server dapat disesuaikan dengan kebutuhan aplikasi yang akan dijalankan di atasnya dan tidak melebihi spesifikasi server fisik. Masing-masing virtual server akan saling terisolasi satu dengan yang lainnya.

Permasalahan virtualisasi tersebut muncul ketika hendak mengoptimisasikan sumber daya yang ada. Ketika suatu "virtual" server dibuat untuk menjalankan suatu aplikasi yang spesifik, maka keseluruhan sistem perlu dibuat (Sistem operasi, alokasi sumber daya memori, *processor*) termasuk *library* yang dibutuhkan dalam menjalankan sistem / aplikasi tersebut. "virtual" server tersebut akan diatur oleh suatu program VMM yang menjembatani komunikasi dengan server fisik. Dengan begitu, kapasitas memori dan alokasi sumber daya pada server tidak optimal dalam menjalankan "virtual" server secara bersamaan.

## 2.8 Virtual Machine

Definisi dari *virtual machine* sendiri adalah suatu aplikasi atau *operating system* yang berjalan pada suatu aplikasi yang mengimitasi *hardware* asli yang sesuai dengan lingkungan aplikasi atau *operating system* tersebut. Pada umumnya virtualisasi yang sering digunakan adalah *full virtualization*, dimana terdapat suatu program tersendiri (*hypervisor*) yang mengatur *virtual machine* dengan pembagian *resource* hardware asli (*virtual machine* dapat dikatakan sebagai *guest* dan tempat dimana *virtual machine* diinstall dapat dikatakan sebagai *host*). Masing-masing *virtual machine* yang berjalan pada host akan terisolasi satu dengan lainnya, namun *resource* hardware host akan terbagi satu sama lain. terdapat 2 tipe *hypervisor* :

- **Type 1**

Pada tipe ini, *hypervisor* langsung terinstall pada *hardware* dan berfungsi untuk mengatur *hardware* dan *operating system* yang terinstall di atasnya (*virtual machine*)

- **Type 2**

Berbeda dengan tipe sebelumnya, *hypervisor* terinstall pada *operating system guest* dimana *virtual machine* terinstall layaknya suatu program.

Umumnya *hypervisor* tipe 2 sering digunakan pada komputer - komputer konvensional. Contoh aplikasi ini adalah VMWare[28] maupun VirtualBox [29].

Kebutuhan virtualisasi pada awalnya merupakan suatu cara untuk memberikan kemudahan dalam menginstall *operating system* pada mesin - mesin yang sudah lama. Selain itu, dengan virtualisasi kita dapat menginstall berbagai *operating system* dalam sebuah mesin tanpa harus takut pengaturan tiap - tiap *operating system* akan saling mempengaruhi *operating system* lainnya (prinsip virtualisasi dimana setiap *virtual machine* akan saling tertutup dengan *virtual machine* lainnya). Selain itu, seperti yang digunakan dalam *cloud computing*, virtualisasi mengoptimalkan pemanfaatan *resource* hardware untuk meningkatkan layanan. Sebagai contoh, para developer akan menguji sistem atau aplikasi yang telah dibuat pada *environment* sebenarnya, sehingga diperlukan suatu wadah uji coba yang sesuai dengan sistem atau aplikasi tersebut. Dengan adanya teknik virtualisasi tersebut, baik dengan *cloud computing* maupun pada komputer sendiri, developer tidak perlu takut sistem atau aplikasi yang dijalankan akan menyebabkan kerusakan pada *environment* dan berdampak juga pada pekerjaan sebelumnya.

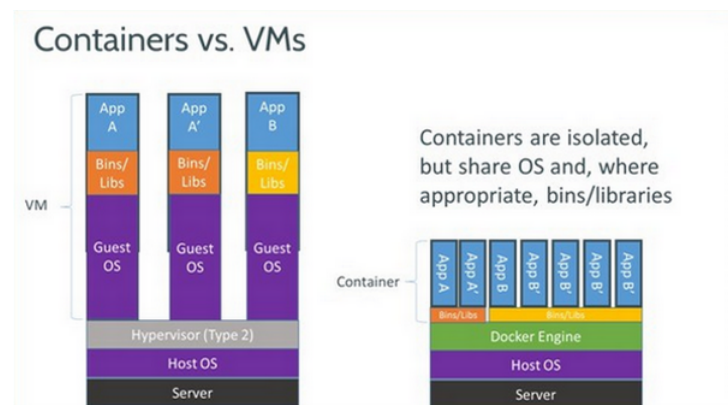
Teknik *full virtualization* juga memiliki kekurangan. Untuk mengoptimalkan kinerja *host* dalam meningkatkan kualitas aplikasi yang berjalan pada *virtual machine (guest)*. Untuk membentuk sebuah *guest*, *hypervisor* akan membutuhkan data memori, prosesor, jaringan yang *fixed*. Dengan begitu, jumlah *virtual machine* yang akan terbentuk akan terikat dengan besarnya *resource* hardware yang dimiliki oleh *guest*. Disatu sisi, ketika suatu *virtual machine* sedang menjalankan proses yang membutuhkan kemampuan komputasi tambahan, *hypervisor* tidak dapat memberikan *resource* tambahan, walaupun *virtual machine* lainnya dalam keadaan *idle* (tidak bekerja).

## 2.9 Docker

Docker, *platform* virtualisasi yang dikembangkan oleh Perusahaan Docker menggunakan teknik berbeda dengan *Full Virtualization*. Docker menggunakan *Docker engine* sebagai ganti dari *hypervisor*. *Docker engine* dikembangkan berdasarkan *Linux containers (LXC)*, dimana level virtualisasi terletak pada sistem operasi [30] (bentuk virtualisasi *OS Partitioning*). Dengan begitu suatu server linux dapat menjalankan beberapa sistem linux yang saling terisolasi satu sama lain.



Kernel pada "virtual" server, dalam hal ini disebut container yang dibentuk dengan libcontainer [31] akan menggunakan kernel yang sama dengan server fisik. Oleh karena itu, container yang dibentuk akan lebih kecil dan padat tanpa perlu menginstall sistem secara keseluruhan dari awal. container tidak perlu mengalokasikan sumber daya secara tetap dikarenakan *Docker engine* akan mengalokasikan sumber daya berdasarkan aplikasi yang berjalan pada container [32].



**Gambar 2.7:** Perbandingan virtualisasi pada VM dan *container* pada Docker[33]



**Gambar 2.8:** Logo Docker[34]

Walaupun begitu, untuk masing masing virtualisasi, baik itu yang menggunakan *hypervisor* maupun *Docker engine* pada Docker memiliki kelebihan dan kekurangan masing - masing. *Container* yang dibentuk pada Docker harus menggunakan sistem operasi yang sama dengan server fisik. Untuk saat ini, Docker baru mendukung sistem operasi Linux sebagai *container* dan berjalan pada sistem operasi Linux. Walaupun dapat berjalan pada sistem operasi lainnya, namun penggunaan Docker masih dijalankan dalam "virtual" komputer yang menggunakan sistem operasi Linux pada *Virtual Machine*. Dengan alokasi sumber daya yang tidak tetap (sesuai dengan proses yang berjalan), maka *container* yang terbentuk akan lebih

banyak dari "virtual" server yang dihasilkan dengan menggunakan "hypervisor" pada umumnya terikat dengan batasan sumber daya. Namun "virtual" server tersebut lebih beragam dan tidak terikat dengan sistem operasi yang berjalan pada server fisik.

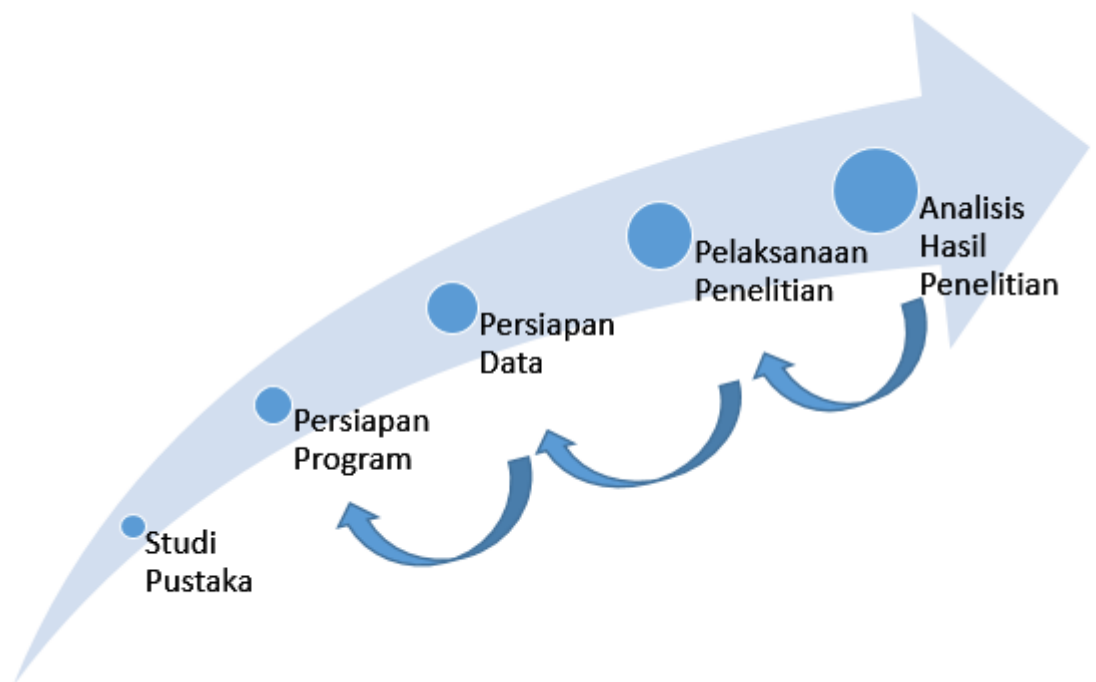
## BAB 3

### METODOLOGI PENELITIAN

Bab ini akan menjelaskan metodologi penelitian yang dilaksanakan. Bagian ini juga menjelaskan tahapan penelitian yang dilakukan, sistem pendukung penelitian, dan data penelitian

#### 3.1 Tahapan Penelitian

Tahapan dari penelitian yang dilakukan terdiri dari studi pustaka, percobaan aplikasi, persiapan program, persiapan data, pelaksanaan penelitian dan analisis hasil penelitian. Tahapan tersebut dapat dilihat pada gambar berikut ini



**Gambar 3.1:** Tahapan Penelitian

Pada tahap studi pustaka dilakukan eksplorasi mengenai konsep penemuan obat baik secara tradisional maupun memanfaatkan IT, *cloud computing* dan virtualisasi komputer berdasarkan literatur - literatur tulis yang ada.

Dalam tahap persiapan sistem, penulis mencoba membiasakan diri dengan mengoperasikan aplikasi yang digunakan dalam penelitian, Docker dalam hal virtual-

isasi maupun Autodock dan Autodock Vina dalam melakukan *virtual screening*. Kemudian dilakukan konfigurasi dan instalasi aplikasi tersebut. Aplikasi yang terlibat dalam penelitian ini diantaranya Autodock versi 4.2, Autodock Vina versi 1.1, dan Autodock Tools versi 1.5.2 yang akan digunakan untuk preparasi data. Selain itu Docker versi terakhir yang akan digunakan 1.5.0 sebagai platform virtualisasi komputer. Pada tahapan ini, penulis juga membuat beberapa *script* sederhana yang akan digunakan untuk menjalankan aplikasi Autodock dan Autodock Vina secara otomatis dari awal pengerjaan hingga pengumpulan hasil akhir. *Script* dapat dilihat pada lampiran C untuk Autodock Vina dan lampiran D untuk Autodock. Instalasi aplikasi yang digunakan dalam penelitian ini dapat dilihat pada lampiran A.

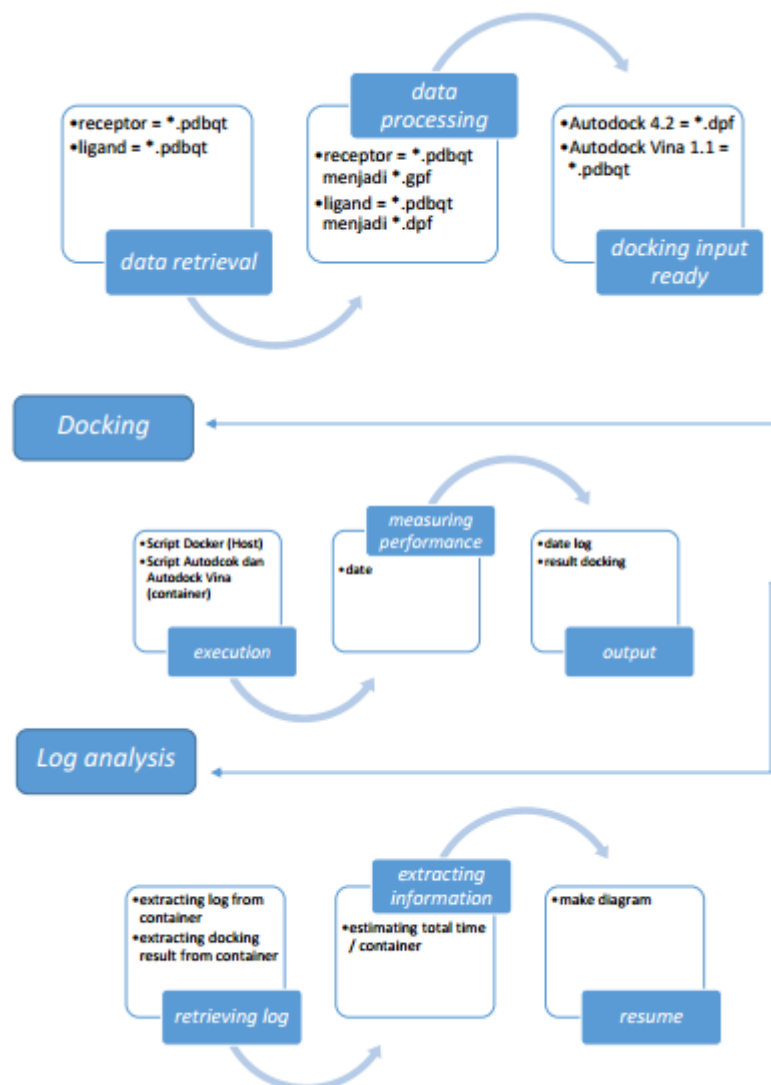
Pada tahap pemrosesan data dilakukan *pre-docking* atas data mentah penelitian. Proses *pre-docking* dilakukan sebagai syarat untuk menjalankan aplikasi Autodock 4.2. Untuk Autodock Vina 1.1, aplikasi dapat langsung dijalankan tanpa perlu data dipreparasi. Proses *pre-docking* pada Autodock dapat dilihat pada lampiran B.

Pelaksanaan penelitian dilaksanakan pada sebuah PC desktop pada Laboratorium Jaringan Komputer lantai 5 gedung B Fasilkom UI. Kedua aplikasi tersebut dijalankan secara bergantian pada *platform* Docker yang telah terpasang. Tahapan ini dapat dilakukan berulang kali hingga didapatkan data yang dapat dianalisis.

Tahap akhir dari penelitian ini adalah analisis hasil penelitian. Analisis hasil penelitian dilakukan atas data keluaran proses *virtual screening*. Data yang dianalisis adalah data keluaran berupa *running time* yang diperoleh dengan program *date* yang tersedia pada sistem operasi berbasis GNU/Linux.

### 3.1.1 Workflow Penelitian

Alur penelitian ini diawali dengan instalasi *platform* Docker pada *PC desktop* yang akan digunakan. Kemudian dalam tahapan *molecular docking* yang akan dilakukan dengan aplikasi Autodock dan Autodock Vina terbagi menjadi 3, yaitu *pre-docking*, *docking*, dan analisis hasil log *docking*. Proses ini dimulai dari mengunduh *ligand* dan *receptor* hingga mendapatkan hasil akhir berupa log hasil *molecular docking* dari aplikasi Autodock dan Autodock Vina. Berikut merupakan gambar *workflow* :



Gambar 3.2: Workflow penelitian

### 3.2 Sistem Pendukung Penelitian

Penelitian ini menggunakan *platform* Docker yang telah *terinstall* pada PC desktop yang terdapat pada Laboratorium Jaringan Komputer Gedung B Lantai 5 Fasilkom UI. Berikut ini merupakan spesifikasi PC desktop yang digunakan :

Spesifikasi Komputer	
Prosesor	Intel i7-2600 CPU @ 3.40GHz X 4
RAM	5.8 GiB
Hardisk	418 GB
Sistem Operasi	debian 8.0
Instruction Set	64-bit

**Tabel 3.1:** Spesifikasi PC desktop

### 3.3 Data Penelitian

Data yang digunakan dalam penelitian ini seluruhnya merupakan data yang diperoleh dari seorang rekan yang mengelola situs data tanaman obat Indonesia. Data yang digunakan valid dan merupakan salah satu persyaratan dalam penelitian ini untuk membandingkan hasil performa penelitian ini dengan penelitian terkait yang sebelumnya telah dilakukan.

#### 3.3.1 Pemilihan Data

Data yang digunakan dalam penelitian ini diambil dari Herbaldb [35]. Herbaldb merupakan situs yang menyediakan data tanaman obat Indonesia yang dimiliki oleh Fakultas Farmasi Universitas Indonesia. Data yang diperoleh telah diolah oleh Herbaldb dan jumlah data yang digunakan sebanyak 1406 data *receptor* dan 1 buah data *ligand*

#### 3.3.2 Pemrosesan Data

Data yang telah diperoleh harus menjalani proses *pre-docking* sebelum dapat dilakukan *virtual screening*. Pada kesempatan ini penulis telah memperoleh data *ligand* dan *receptor* dalam ekstensi yang sama, sehingga langkah *pre-processing* hanya dilakukan pada file yang akan digunakan untuk menjalankan aplikasi Autodock versi 4.2 [36]. Langkah *pre-processing* dilakukan dengan aplikasi AutodockGrid4 yang telah ada dari instalasi Autodock versi 4.2 :

- **Mempersiapkan AutoGrid Parameter Files untuk receptor**

AutoGrid Parameter Files pada receptor disiapkan dengan aplikasi Autodock Tools 1.5.2. AutoGrid Parameter Files berisi berkas yang siap dipetakan atas ligand menggunakan autogrid4. Hasil dari tahapan ini adalah berkas receptor dalam format \*.gpf

- **Menghitung atomic affinity maps**

Pemetaan yang sudah dilakukan pada receptor dan menghasilkan berkas receptor dalam format \*.gpf harus dipetakan untuk setiap berkas ligand. Proses pemetaan ini dilakukan dengan cara menghitung atomic affinity maps pada berkas receptor dengan menggunakan aplikasi autogrid4. Tahapan ini akan menghasilkan berkas pemetaan dalam format \*.glg, \*.map, \*.maps.xyz dan \*.maps.fld.

- **Mempersiapkan docking parameter files untuk setiap ligand**

Tahapan terakhir adalah menyiapkan docking parameter files untuk setiap ligand dibantu modul python dari Autodock Tools 1.5.2 yang bernama . Docking parameter files ini berisi informasi terkait ligand seperti jumlah atom, koordinat atom dan jumlah torsi aktif yang dimiliki yang sudah dipetakan ke dalam informasi pada receptor. Hasil dari tahapan ini adalah berkas docking parameter files dalam format \*.dpf.

Sedangkan tahapan *pre-docking* pada aplikasi Autodock Vina versi 1.1 tidak perlu dilakukan lagi dikarenakan ekstensi file telah memenuhi persyaratan dalam menjalankan aplikasi tersebut.

Berikut merupakan perbandingan tahapan *pre-docking* antara Autodock versi 4.2 dan Autodock versi 1.1

Autodock 4.2	Autodock Vina 1.1
1. Mempersiapkan AutoGrid parameterfiles untuk receptor	1.Tahap pre-docking selesai
2. Menghitung atomic affinity maps	Transparansi proses pre-docking Autodock Vina 1.1
3. Mempersiapkan docking parameter files untuk setiap ligand	
4. Tahap pre-docking selesai	

**Tabel 3.2:** Preparasi data pada Autodock dan Autodock Vina

### 3.4 Skenario Eksperimen Penelitian

Dalam penelitian ini, penulis akan melaksanakan 10 skenario eksperimen dengan 5 skenario melibatkan Autodock dan 5 skenario melibatkan Autodock Vina. Tiap - tiap skenario pada Autodock maupun Autodock vina adalah sama, yaitu menjalankan aplikasi tersebut pada *platform* Docker dengan data yang telah dipreparasi. Perbedaan mendasar untuk setiap skenario adalah jumlah *container* yang digunakan dan persebaran data yang ditampung oleh tiap - tiap *container*. Persebaran data dapat diestimasi dengan total data *receptor* yang digunakan dibagi dengan jumlah

*container* untuk setiap skenario. Maka untuk setiap *container* akan memperoleh jumlah data yang sama (hasil bagi) dan jika tidak dapat dibagi sama rata, ada beberapa *container* yang berisikan data sebanyak sisa jumlah data tersebut (modulo dari jumlah data dengan jumlah *container*).

Eksperimen	Jumlah <i>container</i>	Jumlah data receptor per container
1	50	28
2	100	14
3	150	9
4	200	7
5	250	5

**Tabel 3.3:** Pembagian *container* dan data setiap *container*

Angka tersebut didapatkan dari percobaan yang penulis lakukan untuk mengetahui maksimum jumlah *container* yang dapat dijalankan pada *desktop PC*. Didapatkan angka 289 *container* yang dapat dijalankan dalam 1 waktu, namun menyebabkan *desktop PC* hang. Maka penulis mencoba untuk menetapkan *range container* dengan kelipatan 50 dalam skenario penelitian.

### 3.5 Pengukuran Kinerja

Dalam mengukur kinerja penelitian ini, penulis menggunakan 1 parameter yaitu *running time*. *Running time* digunakan program *date* yang tersedia dalam sistem operasi debian. Penulis akan mengumpulkan setiap *running time* untuk masing masing *container* yang dijalankan pada setiap eksperimen dan akan diolah (proses terlama, tercepat dan rata rata waktu proses). Pada bagian akhir, penulis mencoba untuk menghitung waktu proses 1 molekul untuk masing - masing skenario.



## BAB 4

### HASIL PENELITIAN DAN ANALISIS

Bab ini akan membahas hasil penelitian yang telah dilakukan dan analisis terhadap hasil penelitian tersebut.

#### 4.1 Hasil Penelitian

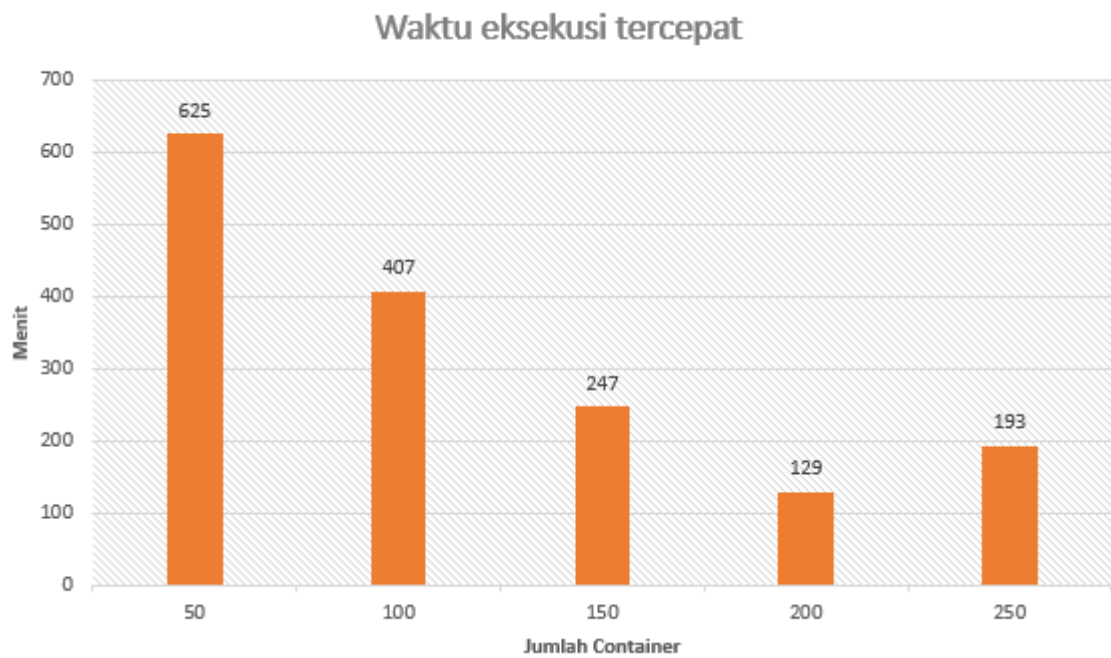
Berdasarkan kerangka penelitian yang telah dijelaskan pada bab sebelumnya, terdapat 10 skenario percobaan dimana masing - masing 5 skenario pada aplikasi Autodock dan 5 skenario pada aplikasi Autodock Vina. Secara garis besar, skenario tersebut adalah menjalankan aplikasi yang telah terpasang pada *container* Docker yang telah dibuat. Untuk setiap skenario jumlah data pada masing - masing *container* akan dibagi berdasarkan total data (1406 data *receptor*) dibagi dengan jumlah *container* yang terbentuk. Setelah data terbagi, masing - masing *container* akan menjalankan aplikasi (Autodock ataupun Autodock Vina) dan waktu akan dicatat dari awal eksekusi data pertama hingga waktu akhir ketika aplikasi selesai mengeksekusi data terakhir.

Berikut merupakan data yang diperoleh dari 5 skenario aplikasi Autodock Vina

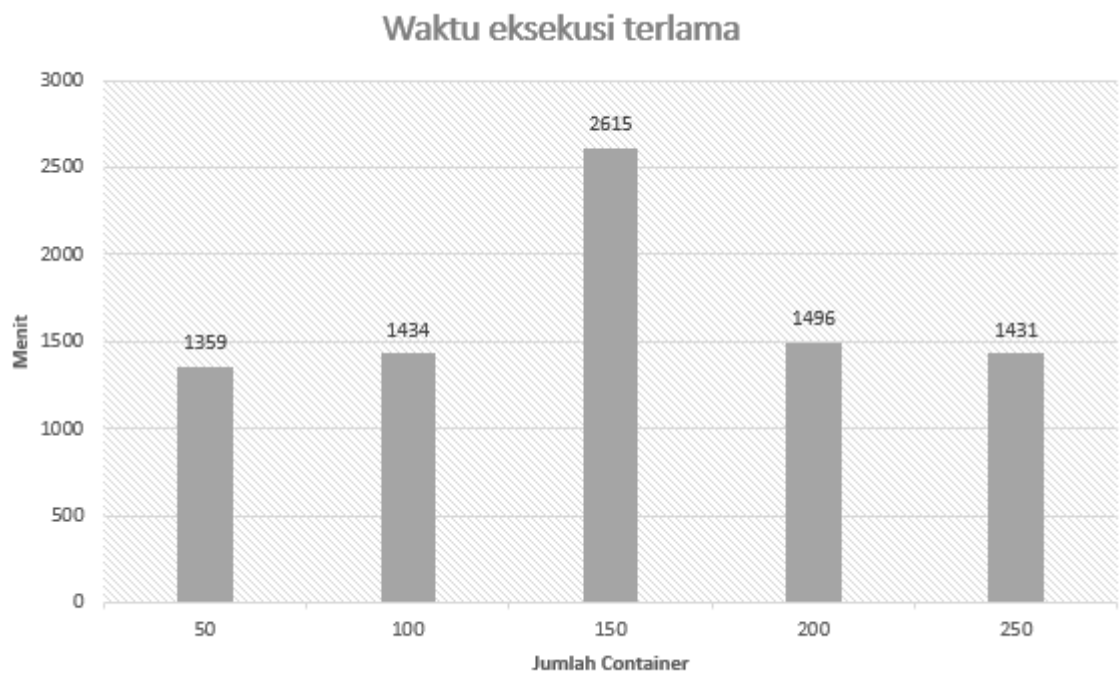
Jumlah <i>container</i>	Waktu eksekusi (menit)		
	tercepat	terlama	Rata-rata
50	625	1359	968
100	407	1434	891
150	247	2615	745
200	129	1496	705
250	193	1431	720

**Tabel 4.1:** Hasil skenario eksperimen pada aplikasi Autodock Vina

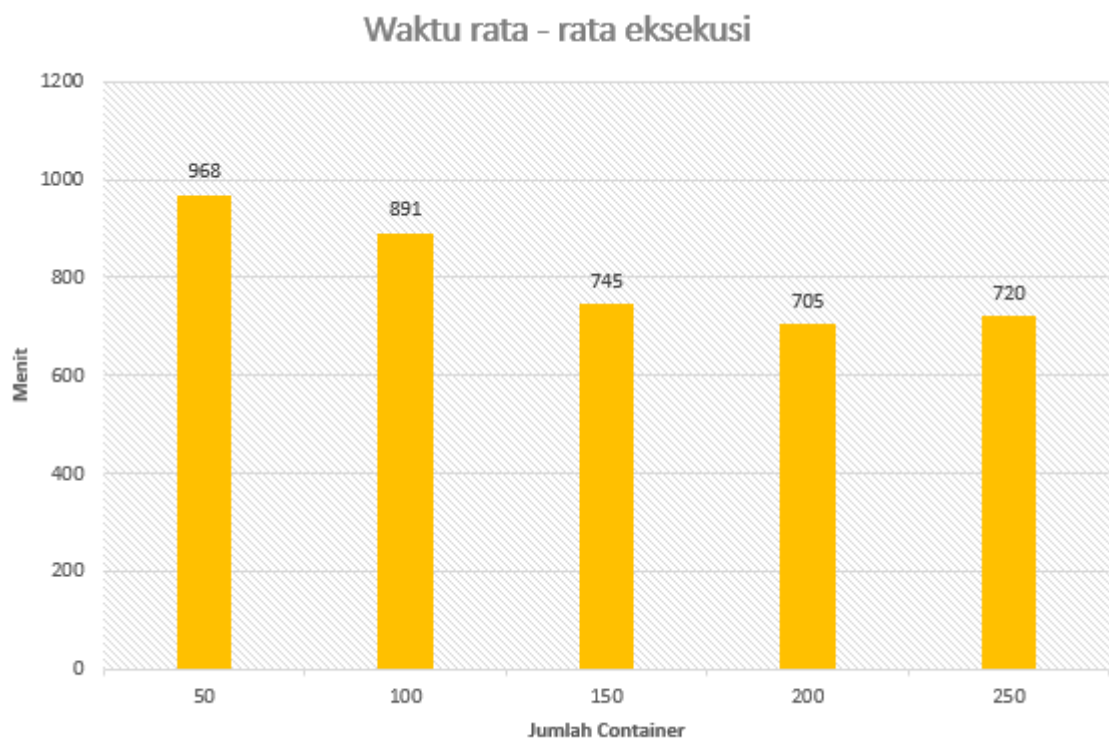
Untuk masing - masing data waktu eksekusi tersebut dapat disajikan menjadi tabel grafik. Tabel yang disajikan berupa waktu eksekusi tercepat, terlama dan rata - rata eksekusi setiap *container* untuk setiap skenario.



**Gambar 4.1:** Waktu eksekusi tercepat untuk masing - masing skenario Autodock Vina



**Gambar 4.2:** Waktu eksekusi terlama untuk masing - masing skenario Autodock Vina



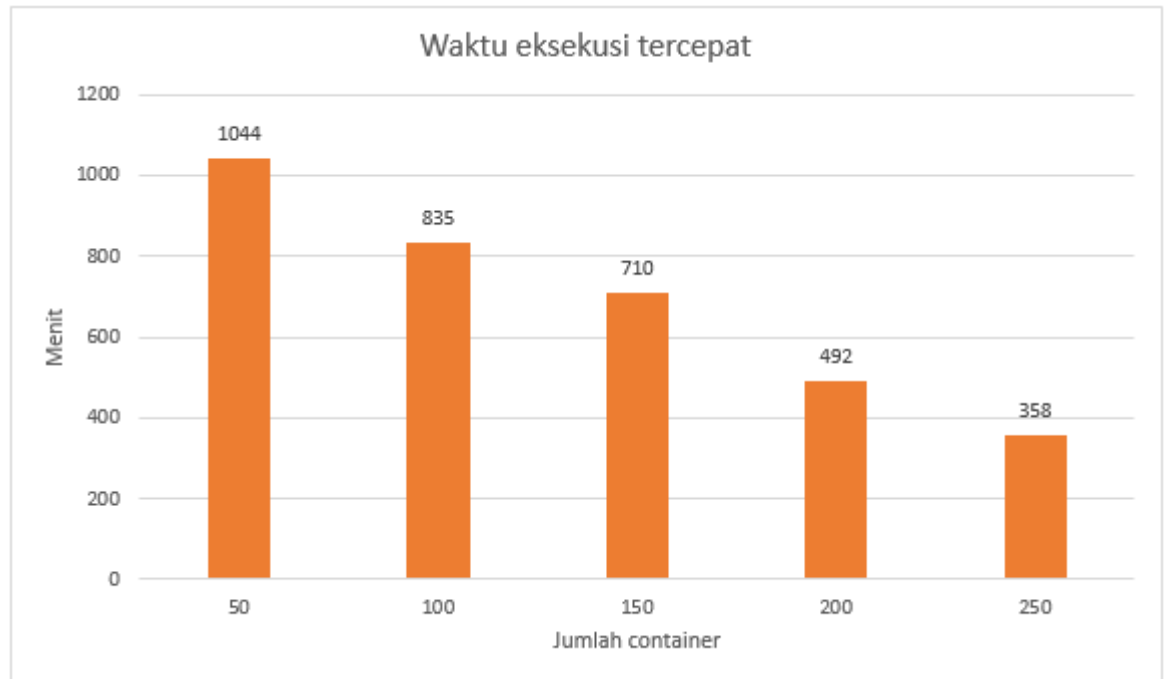
**Gambar 4.3:** Grafik dari rata - rata waktu eksekusi masing - masing skenario Autodock Vina

Berikut merupakan data yang diperoleh dari 5 skenario aplikasi Autodock

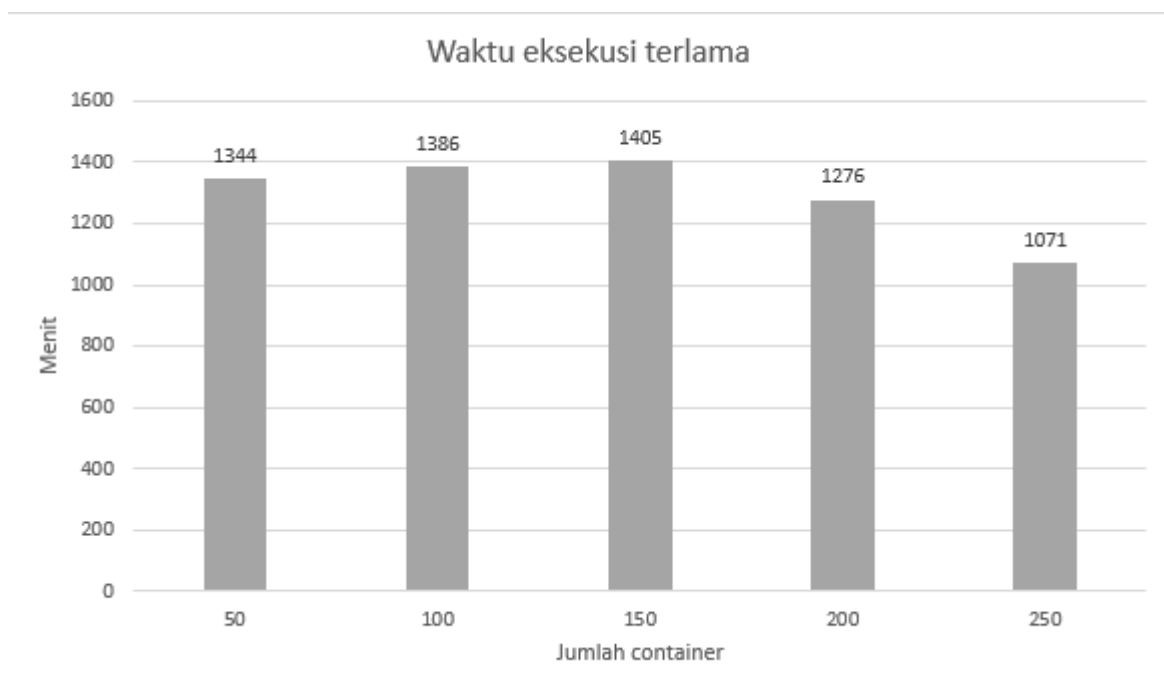
Jumlah <i>container</i>	Waktu eksekusi (menit)		
	tercepat	terlama	Rata-rata
50	1044	1344	1255
100	835	1386	1259
150	710	1405	1216
200	482	1276	1007
250	358	1071	811

**Tabel 4.2:** Hasil skenario pada eksperimen Autodock

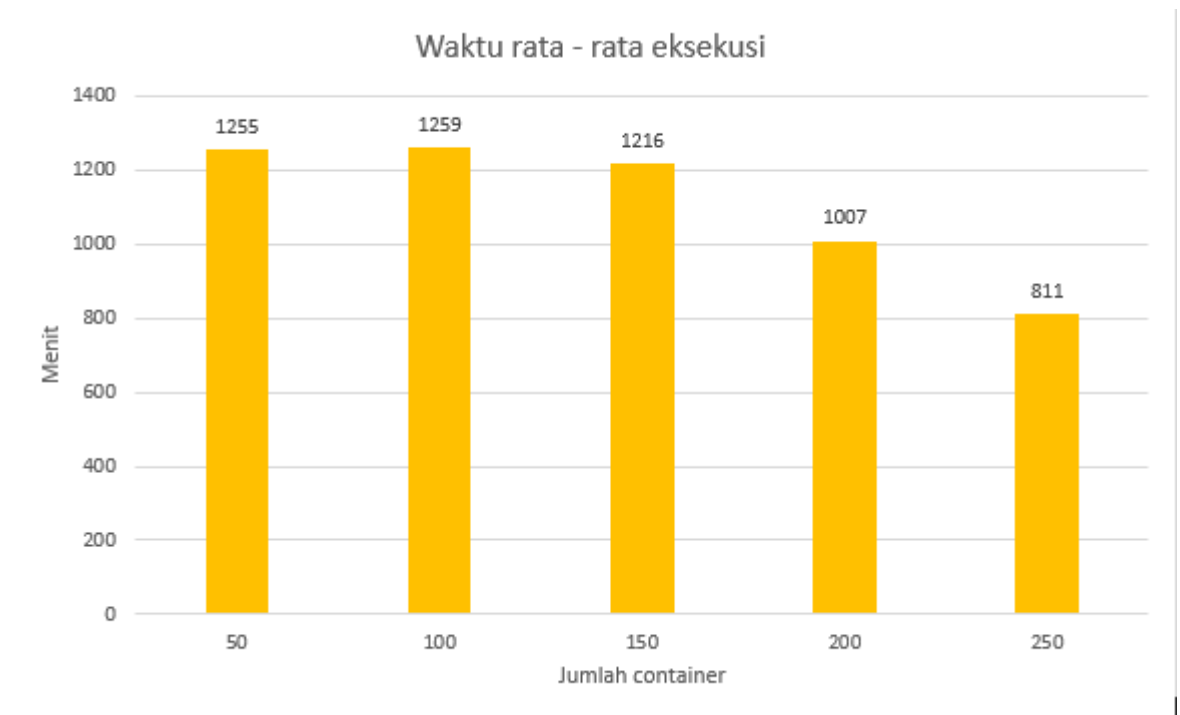
Untuk masing - masing data waktu eksekusi tersebut dapat disajikan menjadi tabel grafik :



**Gambar 4.4:** Waktu eksekusi tercepat untuk masing - masing skenario Autodock



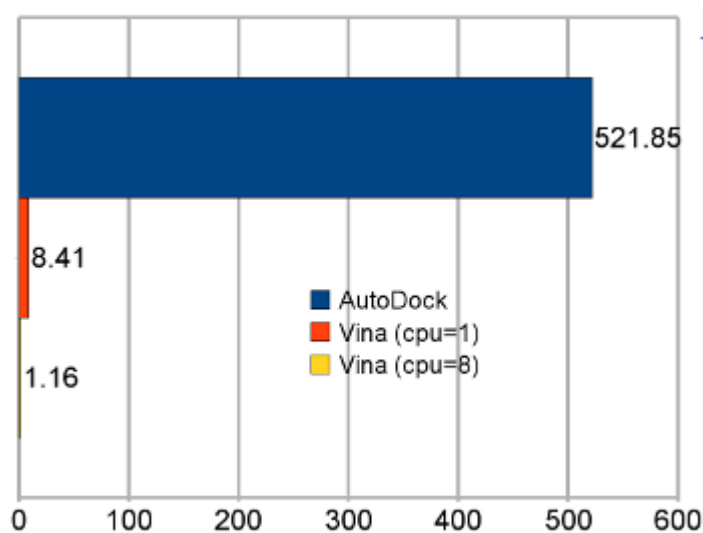
**Gambar 4.5:** Waktu eksekusi terlama untuk masing - masing skenario Autodock



**Gambar 4.6:** Grafik rata - rata waktu eksekusi masing - masing skenario Autodock

## 4.2 Analisis

Pada situs resmi Autodock telah ditampilkan perbandingan performa antara Autodock dan Autodock Vina [41] seperti berikut :



**Gambar 4.7:** Perbandingan waktu *running* pada situs resmi Autodock [19]

Eksekusi pada Autodock Vina jauh lebih cepat dibandingkan dengan Autodock, hal ini dikarenakan pada aplikasi Autodock Vina memanfaatkan *multithreading*

sehingga dapat mengoptimalkan kinerja dari banyak CPU [19]. Gambar 4.7 di atas memberikan gambaran awal bahwa dalam skenario penelitian yang dilakukan, Autodock Vina akan memberikan waktu eksekusi yang lebih cepat dibandingkan dengan Autodock. Untuk setiap rata - rata waktu eksekusi pada skenario Autodock Vina dibandingkan dengan rata - rata waktu eksekusi pada skenario Autodock memberikan hasil yang sama dengan grafik yang diberikan pada situs resmi Autodock.

Pada tabel waktu rata - rata eksekusi untuk aplikasi Autodock dan Autodock Vina dapat dilihat bahwa grafik dari setiap skenario memberikan *trend* menurun. Dengan begitu dapat dilihat bahwa jumlah *container* dan jumlah data menjadi faktor yang mempengaruhi waktu proses tersebut. Dapat disimpulkan bahwa, semakin banyak *container* yang dibentuk maka semakin sedikit data yang digunakan pada aplikasi Autodock maupun Autodock Vina pada masing - masing *container*. Semakin banyak *container* juga memperbanyak eksekusi aplikasi yang dilakukan secara paralel. Pada gambar grafik 4.3 dan 4.6 di atas terlihat *trend* semakin banyak *container* maka waktu eksekusi akan semakin sedikit. Namun dapat dilihat juga pada gambar tabel 4.3, *container* 200 menghasilkan waktu eksekusi yang lebih kecil dibandingkan dengan *container* 250. Penulis memperkirakan bahwa kompleksitas molekul juga mengambil andil dalam lamanya waktu yang dibutuhkan oleh aplikasi Autodock maupun Autodock Vina. Untuk dapat memastikan bahwa jumlah *container* mempengaruhi, penulis mencoba untuk menghitung waktu eksekusi aplikasi Autodock dan Autodock Vina dalam memproses 1406 data receptor.

Waktu eksekusi (menit)		
Jumlah Container	Autodock	Autodock Vina
1	4797	1459

**Tabel 4.3:** Waktu eksekusi Autodock dan Autodock Vina saat 1 *container*

Dapat dilihat pada tabel 4.3 bahwa waktu yang dibutuhkan dalam menjalankan skenario percobaan dengan hanya menggunakan 1 buah *container* memakan waktu yang lebih lama dibandingkan dengan banyak *container* pada skenario penelitian.

Dari skenario yang telah dilakukan, penulis juga mengamati kinerja CPU dan memori yang digunakan oleh Docker. Berdasarkan penjelasan Marek Goldmann [37], setiap *container* yang dibentuk akan membagi sama rata *resource* CPU. Pembagian tersebut hanyalah *relative weight*, dimana setiap *container* dapat menggunakan *shared* CPU, bukan kecepatan kasar CPU yang digunakan oleh *container*. Container yang terbentuk awalnya akan memiliki 1024 *shared* CPU dari CPU fisik, dan akan terbagi sama rata selama *container* baru akan terbentuk. Hasil dari penelitian ini baik menjalankan aplikasi Autodock maupun Autodock Vina dapat dilihat

sebagai berikut :

<i>Relative weight CPU (%)</i>		
Jumlah <i>container</i>	Autodock Vina	Autodock
50	15,9 - 16,9	15,6 - 19,6
100	7,9 - 12,3	8 - 8,6
150	5 - 5,6	6-7,6
200	3,3 - 3,9	4,6 - 5
250	0,7 - 3	3,6 - 4

**Tabel 4.4:** Persentasi *relative weight* pemanfaatan CPU

Walaupun dari tabel diatas tidak seperti perhitungan yang dikatakan oleh Marek Goldmann, angka - angka tersebut tetap mendekati pembagian secara merata. Penelitian yang dilakukan menggunakan *desktop* CPU dengan prosesor Intel i-7 , dimana prosesor secara fisik memiliki 4 core namun dengan adanya *Hyper Threading*, terdapat logical core untuk masing - masing core sehingga terbaca menjadi 8 core (800 %). Dalam kenyataannya, nilai *relative weight* dapat berubah mengikuti besarnya *task* yang dikerjakan oleh *container* tersebut dan nilai *relative weight container* lain. Nilai *relative wieght container* dalam keadaan *idle* dapat terbagi sama rata untuk *container* lain yang masih mengerjakan *task* (menyesuaikan dengan *task* dalam *container*). Dalam hal pembagian memori, setiap *container* dapat menggunakan memori pada *host* hingga kapasitas maksimum memori pada *host*. Pada eksperimen yang dijalankan, terlihat bahwa untuk aplikasi Autodock Vina maupun Autodock yang dijalankan hanya menggunakan 0,1 % - 0,9 % dari memori pada komputer *host*

Dalam penelitian ini penulis juga membandingkan performa yang diberikan oleh Docker dengan penelitian yang telah dilakukan oleh Muhammad H. Hilman [2]. Beliau menggunakan aplikasi Autodock Vina dalam *virtual screening*. Perbandingan dilakukan dengan mencari waktu untuk memproses 1 buah molekul. Waktu ini didapat dengan membagi waktu yang dihasilkan dari awal eksekusi aplikasi Autodock maupun Autodock Vina pada *container* hingga aplikasi selesai bekerja dengan jumlah data yang digunakan dalam eksperimen penelitian, yaitu 1406 buah data. Berikut merupakan gambar tabel waktu yang dibutuhkan dalam memproses 1 molekul yang dapat dikerjakan dalam *virtual cluster* dalam Amazon EC2 :

Parameter	SCluster	LCluster	XLCluster
Memory	1,7 GB	7,5GB	15 GB
Compute Unit / Processors	1,2 GHz X 1	1,2 GHz X 4	1,2 GHz X 8
Platform	32-bit	64-bit	64-bit
Harga/jam	\$0,085	\$0,34	\$0,68

**Tabel 4.5:** Spesifikasi harga dan *resource cloud computing* yang disediakan oleh Amazon EC2 [2]

Berdasarkan penelitian yang dilakukan sebelumnya, masing - masing *cluster* terdiri dari 4 node. *Hardware* komputasi tidak sama untuk masing - masing *cluster*, Amazon memberikan gambaran besarnya *clock speed* dan berapa jumlah *processor*.

Parameter	SCluster	LCluster	XLCluster
Total ExecTime(s)	375.624	108.179	48.421

**Tabel 4.6:** Hasil pengukuran *virtual screening* [2]

Tabel diatas merupakan hasil waktu yang diperoleh dalam menjalankan Autodock Vina pada jasa *cloud computing* Amazon EC2. Waktu yang dihasilkan dihitung dalam ribuan detik. Untuk menghitung waktu eksekusi 1 molekul dapat dibagi dengan banyaknya molekul yang digunakan, 1406 molekul. Dengan begitu didapatkan tabel berikut :

Pemrosesan 1 Molekul dalam detik (s)			
Parameter	SCluster	LCluster	XLCluster
AvgExecTime(s)	267	77	34

**Tabel 4.7:** Waktu yang dibutuhkan dalam memproses 1 buah Molekul berdasarkan tabel 4.6 diatas

Hasil yang diperoleh dari eksperimen penelitian ini :

Pemrosesan 1 Molekul dalam detik (s)	
Jumlah <i>container</i>	Autodock Vina
1	62,26
50	41,30
100	38,02
150	32,19
200	30,08
250	31,02

**Tabel 4.8:** Waktu yang dibutuhkan dalam memproses 1 buah Molekul



Dapat dilihat bahwa hasil yang diberikan oleh Docker pada sebuah *desktop* PC tidak jauh berbeda dengan yang diberikan oleh Amazon EC2. Dari tabel tersebut dapat dilihat dalam eksekusi pada Docker dengan *container* terkecil dapat memberikan hasil yang lebih cepat dibandingkan dengan LCluster. Waktu tercepat masih dipegang oleh XLCluster, namun hasil yang diberikan oleh Docker dapat mendekati *virtual cluster* yang digunakan oleh Amazon EC2.

Untuk memastikan bahwa penggunaan *platform* Docker dapat mengoptimisasi kinerja CPU, penulis melakukan skenario tambahan dengan melakukan *virtual screening* di atas *desktop* PC yang digunakan pada percobaan.

Waktu eksekusi pada komputer <i>native</i>		
Parameter	Autodock	Autodock Vina
Waktu total (menit)	4942,8	1460
Waktu per 1 molekul (detik)	211,33	62,3

**Tabel 4.9:** Hasil *virtual screening* yang berjalan pada komputer *native*

Dapat dilihat bahwa hasil yang diberikan mendekati hasil skenario ketika menjalankan *virtual screening* pada 1 buah *container* Docker. Semakin banyak *container* yang digunakan maka persebaran data akan semakin sedikit. Menjalankan program secara paralel dengan data sedikit akan lebih cepat dibandingkan dengan menjalankan sebuah program dengan data yang banyak. Selain itu, dapat ditarik kesimpulan bahwa semakin banyak jumlah *container*, *overhead* yang akan terjadi semakin tinggi. Oleh karena itu, kemungkinan *processor* dalam keadaan *idle* akan semakin sedikit dibandingkan dengan menjalankan *virtual screening* di atas komputer *native* maupun dengan 1 buah *container*.

Dari segi pengeluaran biaya, pemanfaatan teknologi *cloud computing* akan lebih murah dibandingkan dengan pengadaan komputer secara fisik. Hal ini disebabkan pihak penyedia jasa *cloud computing* dalam hal ini Amazon EC2, memberikan daftar harga sesuai dengan *resource* yang digunakan dibandingkan dengan membayar secara keseluruhan perangkat keras yang disediakan untuk pengguna. Penulis membandingkan kembali dari biaya *cluster* yang dilakukan oleh Bapak Muhammad H. Hilman pada penelitian beliau [2]

Parameter	SCluster	LCluster	XLCluster
Harga per jam (\$)	0,085	0,0292	0,0252
Total Cost(\$)	35,36	40,8	35,36
Average Cost(\$)	0,0252	0,0292	0,0252

**Tabel 4.10:** Biaya yang dikeluarkan oleh Bapak Muhammad H. Hilman dalam penelitiannya [2]

Pada tabel tersebut, average cost merupakan biaya yang dibutuhkan dalam memproses 1 buah molekul (Total Cost dibagi dengan 1406 molekul). Jika dilakukan kurs mata uang akan didapatkan Rp. 224,00 — Rp. 260,00 untuk 1 buah molekul. Dibandingkan dengan biaya penggunaan *desktop* PC yang digunakan dalam penelitian ini. Karena tidak relevan menghitung biaya dengan sekali penggunaan *virtual screening*, maka penulis menggunakan beberapa asumsi:

- Umur ekonomis perangkat komputer berdasarkan Surat Edaran Direktur Jenderal Pajak Nomor SE-07/PJ.42/2002 tanggal 8 Mei 2002 adalah 4 tahun
- Penggunaan komputer untuk sebuah proyek *virtual screening* 1 bulan (30 hari kerja) sehingga diperoleh 720 jam kerja. Besarnya listrik yang digunakan oleh komputer 500 watt

Dari asumsi tersebut akan didapatkan total biaya :

- Biaya pengadaan komputer : Rp 14,250,000 (Berdasarkan spesifikasi komputer pada situs Bhinneka.com)
- Biaya maintenance sebesar 10 % dari investasi : Rp 1,425,000
- Konsumsi listrik :  $4 \times 720 \text{ jam} \times 500 \text{ watt} = 1440 \text{ kwh}$
- Biaya konsumsi listrik (tarif gol. pemerintahan): Rp. 23,800 per kwh x 2880 kwh = Rp 34,272,000
- Asumsi jika terjadi kerusakan, biaya teknisi 4 tahun x 1 bulan x Rp. 2,000,000 = Rp 8,000,000

Maka total biaya yang dikeluarkan Rp 57,947,000. Untuk dapat mengetahui biaya per 1 molekul maka perlu dibagi dengan lamanya eksekusi *virtual screening* 1 molekul. Dalam hal ini penulis menggunakan waktu autodock vina saat *container* berjumlah 150. Didapatkan 1 buah molekul memakan waktu 32,19 detik.

- Waktu kerja total = 4 tahun X 720 jam = 2880 jam
- Besarnya waktu kerja dalam satuan detik 2880 jam X 60 menit X 60 detik = 10.368.000 detik
- Banyaknya molekul yang diproses =  $10.368.000 / 32,19 = 322.087$  molekul

Didapatkan  $\text{Rp } 57,947,000 / 322.087 \text{ molekul} = \text{Rp } 179,911$ . Walaupun cukup murah, penggunaan komputer tersebut belum tentu penuh selama 4 tahun. Terdapat

faktor lain yang dapat mempengaruhi seperti adanya proyek *virtual screening*, putusnya sambungan listrik yang menyebabkan proyek *virtual screening* perlu diulang kembali dari awal, dan lain sebagainya yang menghambat. Sedangkan dengan jasa *cloud computing*, walaupun dari perhitungan lebih mahal dibandingkan dengan penggunaan komputer biasa, biaya yang dikeluarkan hanya saat menggunakan *resource* komputasi saja dan kemungkinan terjadinya kegagalan pada perangkat keras tidak akan terjadi.

## BAB 5

### PENUTUP

Bab ini merupakan bagian penutup dari laporan tugas akhir. Bagian ini akan menjelaskan kesimpulan dari penelitian yang telah dilakukan beserta kendala yang dihadapi. Selain itu juga akan berisikan saran untuk penelitian terkait yang akan datang.

#### 5.1 Kesimpulan

Pada penelitian yang telah dilakukan, terdapat beberapa kesimpulan yang merujuk kepada permasalahan mengapa penelitian ini dilaksanakan :

1. Dari hasil analisis penelitian didapatkan bahwa waktu yang dibutuhkan dalam menjalankan proses *virtual screening* dengan memanfaatkan *platform* Docker tidak berbeda jauh dengan hasil yang diperoleh dengan yang dijalankan pada jasa *cloud computing* Amazon EC2 dimana Amazon menggunakan *platform* virtualisasi ciptaannya sendiri. Hal ini dapat dilihat pada tabel 4.6 dan tabel 4.7.
2. Semakin banyak jumlah container yang digunakan maka waktu yang dihasilkan dalam eksperimen akan semakin cepat. Jumlah data yang terdistribusi untuk masing - masing container akan semakin sedikit pula. Jumlah container yang dapat dibuat tanpa harus dijalankan akan terbatas dengan besarnya kapasitas memori penyimpanan (*hardisk*). Sedangkan jumlah container yang dapat menjalankan *task* tertentu akan terbatas dengan besarnya RAM.
3. Semakin banyak jumlah container, maka semakin kecil nilai *relative weight* untuk setiap container dalam berbagi *resource* CPU. Nilai *relative weight* akan bertambah untuk setiap container jika terdapat container yang *idle* (tidak melakukan apa - apa). Ketika container yang *idle* diberikan suatu *task*, maka nilai *relative weight* akan terbagi secara merata kembali dengan container lainnya.
4. Kompleksitas molekul mempengaruhi waktu eksperimen. Semakin kompleks molekul tersebut, semakin lama waktu komputasi yang dilakukan oleh aplikasi Autodock dan Autodock Vina.

5. Dengan memanfaatkan *job scheduling* pada komputer, akan memberikan hasil optimal jika dalam suatu mesin dijalankan banyak. program/task. Dengan begitu, waktu *idle* yang mungkin terjadi dalam prosesor serta *resource* lainnya dapat diminimalkan. *Overhead* yang dihasilkan akan semakin besar yang berakibat waktu *idle processor* akan semakin kecil.

## 5.2 Kendala

Kendala yang penulis hadapi dalam melaksanakan penelitian ini adalah :

1. Pendalaman teori *drug discovery* merupakan hal baru bagi penulis dan memakan waktu yang cukup lama.
2. Dalam menjalankan eksperimen, beberapa kali Lab yang digunakan dalam penelitian mengalami mati listrik sehingga penulis harus mengulang eksperimen yang gagal.

## 5.3 Saran

Pada penelitian ini, penulis menyarankan untuk penelitian terkait kedepannya untuk menggunakan Autodock Vina. Aplikasi tersebut memberikan kemudahan bagi pengguna pemula dalam menjalankan aplikasi tersebut. Selain itu kemudahan *pre-processing* data yang transparan tidak akan membuat bingung pengguna pemula dibandingkan dengan aplikasi Autodock. Penulis juga berharap adanya penelitian lebih lanjut terkait dengan penelitian ini. Tidak menutup kemungkinan penelitian ini akan menjadi suatu inovasi baru yang dapat diimplementasikan dalam kehidupan sehari - hari. Penulis juga terbuka untuk saran yang membangun untuk dapat diterapkan dalam penelitian selanjutnya.

## DAFTAR REFERENSI

- [1] Muhammad H. Hilman. Juni 2010. "*Evaluasi Kinerja Autodock 4.2 dan Autodock Vina 1.1 dalam Proses Molecular Docking dan Virtual Screening di Lingkungan Cluster Hastinapura*". Fakultas Ilmu Komputer, Universitas Indonesia.
- [2] Muhammad H. Hilman. Januari 2012. "*Analisis Teknik Data Mining dan Kinerja Infrastruktur Komputasi Cloud Sebagai Bagian dari Sistem Perancangan Obat Terintegrasi*". Fakultas Ilmu Komputer, Universitas Indonesia.
- [3] Pharma *Drug Discovery and Development : UNDERSTANDING THE R and D PROCESS*. February 2007. [http://www.phrma.org/sites/default/files/pdf/rd\\_brochure\\_022307.pdf](http://www.phrma.org/sites/default/files/pdf/rd_brochure_022307.pdf).
- [4] Myers S, Baker A. *Drug discovery - an operating model for a new era*. Nat Biotechnol 2001; 19: 72730
- [5] Matthew Segall. *Can we really do computer-aided drug design?*. J Comput Aided Mol Des (2012) 26:121124
- [6] Si-sheng OU-YANG, Jun-yan LU, Xiang-qian KONG, Zhong-jie LIANG, Cheng LUO, Hualiang JIANG. *Review Computational drug discovery*. Acta Pharmacologica Sinica (2012) 33: 11311140
- [7] Gambar diagram drug discovery. Diambil dari artikel : Myers S, Baker A. *Drug discovery - an operating model for a new era*. Nat Biotechnol 2001; 19: 72730
- [8] Rajamani R, Good AC. 2007. "*Ranking poses in structure-based lead discovery and optimization: current trends in scoring function development*". Current Opinion in Drug Discovery and Development 10 (3): 30815.
- [9] Gambar ilustrasi molecular docking. Diambil dari artikel : Bachwani Mukesh, Kumar Rakesh. *Molecular Docking : A Review*. IJRAP (2011) 2 (6) 1746 - 1751.
- [10] *Drug design*. [http://strbio.biochem.nchu.edu.tw/classes/special\\_topics\\_biochem/course\\_ppts/rational\\_drug\\_design-2014.pdf](http://strbio.biochem.nchu.edu.tw/classes/special_topics_biochem/course_ppts/rational_drug_design-2014.pdf). diakses pada 3 Juni 2015 pukul 04:49

- [11] Bachwani Mukesh, Kumar Rakesh. *Molecular Docking : A Review*. IJRAP (2011) 2 (6) 1746 - 1751.
- [12] Contoh model *pharmacophore*. [http://en.wikipedia.org/wiki/Pharmacophore#/media/File:PharmacophoreModel\\_example.svg](http://en.wikipedia.org/wiki/Pharmacophore#/media/File:PharmacophoreModel_example.svg) diakses 3 Juni 2015 pukul 18:09
- [13] Schneider G, Fechner U. August 2005. "Computer-based de novo design of drug-like molecules". *Nat Rev Drug Discov* 4 (8): 64963.
- [14] Kitchen DB, Decornez H, Furr JR, Bajorath J. 2004. "Docking and scoring in virtual screening for drug discovery: methods and applications". *Nature reviews. Drug discovery* 3 (11): 93549.
- [15] Wei BQ, Weaver LH, Ferrari AM, Matthews BW, Shoichet BK. 2004. "Testing a flexible-receptor docking algorithm in a model binding site". *J. Mol. Biol.* 337 (5): 116182.
- [16] Morris. 2007, May 04. "AutoDock is the most cited docking software" <http://autodock.scripps.edu/news/autodock-is-the-most-cited-docking-software> diakses 3 Juni 2015 pukul 06:59
- [17] Website resmi The Scripps Research Institute, <http://www.scripps.edu/about/index.html> diakses 6 Juni 2015 pukul 14.39
- [18] Garrett M. Morris et al. 2009. "AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility". *J Comput Chem.* 2009 December ; 30(16): 27852791. doi:10.1002/jcc.21256
- [19] Oleg Trott and Arthur J. Olson. 2010. "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization and multithreading". *J Comput Chem.* 2010 January 30; 31(2): 455461. doi:10.1002/jcc.21334.
- [20] Logo Autodock. [http://www.kdm.wcss.wroc.pl/w/images/thumb/Autodock\\_logo.gif/200px-Autodock\\_logo.gif](http://www.kdm.wcss.wroc.pl/w/images/thumb/Autodock_logo.gif/200px-Autodock_logo.gif) diakses 3 Juni 2015 pukul 07:49.
- [21] Rester, U. July 2008. "From virtuality to reality - Virtual screening in lead discovery and lead optimization: A medicinal chemistry perspective". *Curr Opin Drug Discov Devel* 11 (4): 55968.
- [22] Ian Foster. Juli 2002. "Lecture Note : What is the Grid? A Three Point Checklist". <http://dlib.cs.odu.edu/WhatIsTheGrid.pdf> . diakses 3 Juni 2015 19.00

- [23] Fatima A. Alali, Chia-Lun Yeh. 2012. "Cloud Computing: Overview and Risk Analysis". JOURNAL OF INFORMATION SYSTEMS Vol. 26, No. 2 Fall 2012 pp. 1333
- [24] Shuang Zhang, Shufen Zhang, Xuebin Chen, Xiuzhen Huo. 2010. "Cloud Computing Research and Development Trend". 2010 Second International Conference on Future Networks.
- [25] Layan cloud. [http://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Cloud\\_computing.svg/2000px-Cloud\\_computing.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/Cloud_computing.svg/2000px-Cloud_computing.svg.png) diakses 3 Juni 2015 pukul 20.18
- [26] Amir Ali Semnanian. May 2013. "A STUDY ON VIRTUALIZATION TECHNOLOGY AND ITS IMPACT ON COMPUTER HARDWARE" Department of Computer Engineering and Computer Science California State University, Long Beach.
- [27] IBM Global Education White Paper. October 2007. "Virtualization in Education". [http://www-07.ibm.com/solutions/in/education/download/Virtualization\\_in\\_Education.pdf](http://www-07.ibm.com/solutions/in/education/download/Virtualization_in_Education.pdf)
- [28] Situs resmi VMWare. <http://www.vmware.com/ap>. diakses 12 Juni 2015 pukul 22.49
- [29] Situs resmi VirtualBox. <https://www.virtualbox.org/>. diakses 12 Juni 2015 pukul 22.55
- [30] Bobby Banerjee. November 2014. "LXC vs LXD vs Docker - Making sense of the rapidly evolving container ecosystem". <https://www.flockport.com/lxc-vs-lxd-vs-docker-making-sense-of-the-rapidly-evolving-container-ecosystem/>. diakses 3 Juni 2015 pukul 20.45
- [31] Steven J. Vaughan-Nichols. 11 Juni 2014. "Docker libcontainer unifies Linux container powers". <http://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/>. diakses 12 Juni 2015 pukul 23.00
- [32] Chris Wolf. August 2014. "VMware and Docker - Better Together". <http://blogs.vmware.com/cto/vmware-docker-better-together/> diakses 3 Juni 2015 pukul 20.35
- [33] Virtual machine vs docker. <http://www.rightscale.com/blog/sites/default/files/docker-containers-vms.png>



- [34] Logo Docker. <http://getcloudify.org/img/docker.png>
- [35] Arry Yanuar, Abdul Munim , Akma Bertha Aprima Lagho, Rezi Riadhi Syahdi, Marjuqi Rahmat, and Heru Suhartanto. 2011. *Medicinal Plants Database and Three Dimensional Structure of the Chemical Compounds from Medicinal Plants in Indonesia*. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 1, September 2011
- [36] The Scripps Research Institute, Tutorial Autodock 4.2, <http://autodock.scripps.edu/faqs-help/tutorial> diakses 6 Juni 2015 pukul 10.29
- [37] Marek Goldmann. 11 Sept 2014. "*Resource management in Docker*". <https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/>. diakses pada 12 Juni 2015 pukul 16.08
- [38] Top 500 List. <http://www.top500.org/lists/2014/11/> diakses 3 Juni 2015 pukul 18:11
- [39] Dinesh. "*Disdvantages of Virtualization, What's Your Opinion*". <http://www.sysprobs.com/disadvantages-virtualization-opinion>
- [40] The Scripps Research Institute, Tutorial Autodock Vina 1.1 , <http://vina.scripps.edu/tutorial.html> diakses 6 Juni 2015 pukul 10.29
- [41] Gambar tabel perbandingan Autodock dan Autodock Vina. <http://vina.scripps.edu/>.

# LAMPIRAN

## LAMPIRAN A

### Tutorial Instalasi Docker

Docker mendukung sistem operasi Windows, Mac OS, maupun Linux. Dalam sistem operasi Windows maupun Mac OS, Docker terinstall dalam virtual box yang termasuk dalam paket unduhan. Berikut merupakan langkah - langkah instalasi pada kedua sistem operasi tersebut :

1. Download file instalasi, untuk windows :  
<https://github.com/boot2docker/windows-installer/releases/tag/v1.7.0>  
sedangkan untuk Mac OS : <https://github.com/boot2docker/osx-installer/releases/tag/v1.7.0>
2. setelah file selesai didownload, *double click* file tersebut dan silahkan melanjutkan instalasi dengan pilihan yang disesuaikan kebutuhan yang diinginkan

Untuk instalasi yang dilakukan dalam penelitian ini, penulis menggunakan sistem operasi Debian 8.0 . Debian versi tersebut telah menggunakan kernel 3.16 dimana syarat kernel yang harus digunakan oleh Docker adalah 3.8 dan repositori Docker telah tersedia, sehingga cara instalasi cukup mudah.

1. pastikan bahwa repositori docker telah diaktifkan
2. ketikkan perintah ‘sudo apt-get update’ pada terminal
3. lalu ketikkan perintah ‘sudo apt-get install docker.io’

Sedangkan untuk sistem operasi Debian versi dibawah 8.0 belum memenuhi persyaratan kernel Docker, sehingga perlu dilakukan instalasi kernel.

1. tambahkan tulisan ‘deb <http://deb.debian.net/debian> wheezy-backports main’ pada file yang terletak di ‘/etc/apt/sources.list’ dengan menggunakan text editor yang diinginkan
2. ketikkan perintah ‘sudo apt-get update’ pada terminal anda
3. install linux-image-amd64 dengan menuliskan perintah ‘sudo apt-get install -t wheezy-backports linux-image-amd64’ pada terminal
4. restart komputer anda sehingga kernel yang dipasang dapat digunakan

5. setelah komputer kembali menyala, masukan perintah ‘curl -sSL <https://get.docker.com/> | sh’ pada terminal untuk melakukan instalasi Docker secara otomatis

### **Tutorial Instalasi Autodock versi 4.2**

Ada beberapa tahapan yang harus dilalui dalam melakukan instalasi dan konfigurasi Autodock versi 4.2 pada *Desktop* PC. Pada tutorial ini akan dijelaskan bagaimana cara menyiapkan mesin dengan aplikasi Autodock versi 4.2 sampai dapat digunakan untuk melakukan molecular docking.

#### **1. Pra-syarat instalasi**

Sebelum melakukan instalasi harus disiapkan dulu mesin yang akan digunakan. Untuk instalasi pada *Desktop* PC, sediakan *Desktop* PC dengan sistem operasi GNU/Linux dan pasang terlebih dahulu aplikasi Autodock Tools 1.5.2 yang termasuk dalam paket aplikasi MGL Tools 1.5.2. Perlu diperhatikan bahwa aplikasi Autodock Tools 1.5.2 membutuhkan kompilator python versi 2.5 yang sudah terpasang sebelumnya.

#### **2. Mengunduh Autodock versi 4.2**

Aplikasi Autodock versi 4.2 dapat diperoleh pada situs resmi The Scripps Research Institute untuk aplikasi Autodock versi 4.2, <http://autodock.scripps.edu/>. Unduh Autodock versi 4.2 versi source code untuk GNU/Linux. Mengkompilasi sendiri aplikasi Autodock 4.2 pada mesin perlu dilakukan karena versi executable dari Autodock 4.2 sering tidak cocok dengan mesin yang digunakan.

#### **3. Memasang Autodock 4.2 pada mesin**

Ekstrak berkas Autodock 4.2 yang telah diunduh pada mesin yang akan dipasang. Kemudian ikuti petunjuk pemasangan yang ada dalam berkas README dan INSTALL. Setelah kompilasi, jangan lupa untuk memasukkan lokasi dimana Autodock 4.2 dipasang pada PATH sistem operasi. Autodock 4.2 siap untuk digunakan.

### **Tutorial Instalasi Autodock Vina 1.1**

Ada beberapa tahapan yang harus dilalui dalam melakukan instalasi dan konfigurasi Autodock Vina 1.1 pada *Desktop* PC. Pada tutorial ini akan dijelaskan bagaimana cara menyiapkan mesin dengan aplikasi Autodock Vina 1.1 sampai dapat digunakan untuk melakukan molecular docking.

#### **1. Pra-syarat instalasi**

Sebelum melakukan instalasi harus disiapkan dulu mesin yang akan digunakan. Untuk instalasi pada PC Desktop, sediakan PC Desktop dengan sistem

operasi GNU/Linux dan pasang terlebih dahulu aplikasi Autodock Tools 1.5.2 yang termasuk dalam paket aplikasi MGL Tools 1.5.2. Perlu diperhatikan bahwa aplikasi Autodock Tools 1.5.2 membutuhkan kompilator python versi 2.5 yang sudah terpasang sebelumnya.

## 2. **Mengunduh Autodock Vina versi 1.1**

Aplikasi Autodock Vina versi 1.1 dapat diperoleh pada situs resmi The Scripps Research Institute untuk Autodock Vina versi 1.1 , <http://vina.scripps.edu/>. Unduh Autodock Vina 1.1 versi executable untuk GNU/Linux. Mengkompilasi sendiri aplikasi Autodock Vina 1.1 pada mesin tidak perlu dilakukan karena versi executable dari Autodock Vina 1.1 sudah cocok dengan mesin yang digunakan.

## 3. **Memasang Autodock Vina versi 1.1 pada mesin**

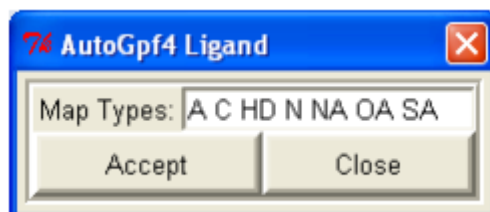
Ekstrak berkas Autodock Vina 1.1 yang telah diunduh pada mesin yang akan dipasang. Kemudian ikuti petunjuk pemasangan yang ada dalam berkas README. Setelah kompilasi, jangan lupa untuk memasukkan lokasi dimana Autodock Vina 1.1 dipasang pada PATH sistem operasi. Autodock Vina 1.1 siap untuk digunakan.

## LAMPIRAN B

### Mempersiapkan AutoGrid Parameter Files untuk receptor Pada Autodock versi 4.2

AutoGrid Parameter Files disiapkan dengan menggunakan Autodock Tools 1.5.2. Tahapan ini menghasilkan berkas yang siap dipetakan atas ligand menggunakan autogrid4. Keluaran dari tahapan ini adalah berkas \*.gpf. Berikut ini adalah langkah-langkah yang harus dilakukan :

- Buka file receptor dengan format \*.pdbqt, simpan ulang receptor dalam format \*.gpf. Pilih Grid -> Output -> Save GPF
- Buka berkas \*.gpf. Pilih Grid -> Open GPF
- Buka pemetaan atom dalam berkas receptor. Pilih Grid -> Set Map , Types -> Directly
- Tambahkan pemetaan atas atom CL, F, S, dan BR dalam Map Types



- Simpan berkas receptor dalam format \*.gpf

### Menghitung atomic affinity maps

Pemetaan yang sudah dilakukan pada receptor dan menghasilkan berkas receptor dalam format \*.gpf harus dipetakan untuk setiap berkas ligand. Proses pemetaan ini dilakukan dengan cara menghitung atomic affinity maps pada berkas receptor dengan menggunakan aplikasi autogrid4. Tahapan ini akan menghasilkan berkas pemetaan dalam format \*.glg, \*.map, \*.maps.xyz dan \*.maps.fld. Untuk mendapatkan hasil tersebut dapat dijalankan *script* berikut pada *desktop* PC yang telah terpasang Autodock Tools :

**Daftar Kode**

**1:** *Script* untuk menghitung atomic affinity maps

```
#!/bin/csh
/path/to/autogrid4 -p receptor.gpf -l receptor.glg
```

### Mempersiapkan Docking Parameter Files

Menyiapkan docking parameter files untuk setiap ligand dibantu modul python dari Autodock Tools 1.5.2 yang bernama `prepare_dpf4.py`. Docking parameter files ini berisi informasi terkait ligand seperti jumlah atom, koordinat atom dan jumlah torsi aktif yang dimiliki yang sudah dipetakan ke dalam informasi pada receptor. Hasil dari tahapan ini adalah berkas docking parameter files dalam format `*.dpf`. Jalankan modul `prepare_dpf4.py` dengan *script* berikut ini

#### Daftar Kode

2: *script* untuk mempersiapkan docking parameter file

```
for f in M*.pdbqt;do
name=`basename $f .pdbqt`
echo $name
/path/to/pythonsh /path/to /prepare_dpf.py -l $f -d
/path/to/ligand_dict.py -l `basename $f` -r
receptor.pdbqt \
-p ga_num_evals=1750000 \
-p ga_pop_size=150 \
-p ga_run=20 \
-p rmstol=2.0
End
```

## LAMPIRAN C

### Kode yang digunakan dalam menjalankan eksperimen Autodock Vina versi 1.1

**Daftar Kode**      **3:** Script yang digunakan pada *desktop* PC host dimana Docker terpasang

```
#!/bin/bash

FILES="/percobaan2/*.pdbqt"
COUNTER_FILE=1
COUNTER_CONTAINER=1
NAMA_CONTAINER="vina"
COUNT=1
JUMLAH_FILE=0
LIMIT_FILE=0
RESIDU_FILE=0
JUMLAH_CONTAINER=$1

#menghitung banyaknya file yang dicopy kedalam container
let JUMLAH_FILE=`ls -l /percobaan2/ | wc -l`
let LIMIT_FILE=$JUMLAH_FILE/$JUMLAH_CONTAINER
let RESIDU_FILE=$JUMLAH_FILE%$JUMLAH_CONTAINER
if [ $RESIDU_FILE -eq 0 ]
then
let LIMIT_FILE=LIMIT_FILE+0
else
let LIMIT_FILE=LIMIT_FILE+1
fi

#main method
for f in $FILES
do
if [ $COUNTER_FILE -eq 1 ]
then
echo "Membuat container" $NAMA_CONTAINER$COUNTER_CONTAINER
docker run -i -t --name $NAMA_CONTAINER$COUNTER_CONTAINER -d
    agungputrap/riset_docker:latest
CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER`
```



```

echo "ini nomor container = " $CONTAINER_ID
cp /3LP10.pdbqt /var/lib/docker/aufs/diff/$CONTAINER_ID/3LP10.
    pdbqt
cp /conf.txt /var/lib/docker/aufs/diff/$CONTAINER_ID/conf.txt
echo "mengcopy file "$COUNTER_FILE " ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy " $COUNT
let COUNT=COUNT+1
let COUNTER_FILE=COUNTER_FILE+1
elif [ $COUNT -eq $JUMLAH_FILE ]
then
echo "harusnya nilai terakhir "$COUNT
echo "mengcopy file terakhir ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
cp /running_vina /var/lib/docker/aufs/diff/$CONTAINER_ID/
    running_vina
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /
    running_vina
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER ./running_vina
elif [ $COUNTER_FILE -lt $LIMIT_FILE ]
then
echo "mengcopy file "$COUNTER_FILE" ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy " $COUNT
let COUNT=COUNT+1
let COUNTER_FILE=COUNTER_FILE+1
#case paling terakhir dijalanin
#     elif [ $COUNT -eq $JUMLAH_FILE ]
#     then
#         echo "harusnya nilai terakhir "$COUNT
#         echo "mengcopy file terakhir ke container"
#         $NAMA_CONTAINER$COUNTER_CONTAINER
#         cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`
#         basename $f`
#         cp /running_vina /var/lib/docker/aufs/
#         $CONTAINER_ID/running_vina
#         docker exec -d
#         $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /running_vina
#         docker exec -d
#         $NAMA_CONTAINER$COUNTER_CONTAINER ./running_vina
else

```

```
#mengcopy file terakhir dalam suatu container dan menjalankan
  script dalam container
echo "mengcopy file "$COUNTER_FILE" ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy "$COUNT
let COUNT=COUNT+1
cp /running_vina /var/lib/docker/aufs/diff/$CONTAINER_ID/
    running_vina
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /
    running_vina
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER ./running_vina
let COUNTER_FILE=1
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
fi
done
```

**Daftar Kode**      **4:** *Script yang digunakan untuk menjalankan aplikasi Autodock Vina dalam container*

```
#!/bin/bash
FILES="/*.pdbqt"
echo "Waktu mulai " >> logvina.txt
date >> logvina.txt
for f in $FILES
do
./autodockvina/bin/vina --config conf.txt --ligand `basename $f`
done
echo "Waktu selesai " >> logvina.txt
date >> logvina.txt
```

**Daftar Kode**      **5:** *Script yang digunakan untuk mengambil hasil dari menjalankan aplikasi Autodock Vina pada container*

```
#!/bin/bash
COUNTER_CONTAINER=1
COUNTER=0
NAMA_CONTAINER="vina"
FILES="/var/lib/docker/aufs/diff/"
JUMLAH_FILE=0
let JUMLAH_FILE=`ls -l /percobaan2/ | wc -l`
while [ $COUNTER -lt $JUMLAH_FILE ];
do
```

```

CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER `
FILEX="$FILES$CONTAINER_ID/*_out.pdbqt"
echo $FILEX
for f in $FILEX
do
docker cp $NAMA_CONTAINER$COUNTER_CONTAINER:`basename $f` /
    hasilvina/
let COUNTER=COUNTER+1
done
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
done

```

**Daftar Kode**     **6:** *Script* yang digunakan untuk mengambil log dari masing - masing container

```

#!/bin/bash
COUNTER_CONTAINER=1
COUNTER=0
NAMA_CONTAINER="vina"
FILES="/var/lib/docker/aufs/diff/"
JUMLAH_FILE=$1
while [ $COUNTER -lt $JUMLAH_FILE ];
do
CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER `
FILEX="$FILES$CONTAINER_ID/logvina*"
echo $FILEX
for f in $FILEX
do
docker cp $NAMA_CONTAINER$COUNTER_CONTAINER:`basename $f` /
    logvina/$NAMA_CONTAINER$COUNTER_CONTAINER/
let COUNTER=COUNTER+1
done
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
done

```

## LAMPIRAN D

### Kode yang digunakan dalam menjalankan eksperimen Autodock versi 4.2

**Daftar Kode**      7: *Script* yang digunakan pada *desktop* PC untuk membuat container dan menjalankan container

```
#!/bin/bash

FILES="/percobaan2/*.pdbqt"
COUNTER_FILE=1
COUNTER_CONTAINER=1
NAMA_CONTAINER="dock"
COUNT=1
JUMLAH_FILE=0
LIMIT_FILE=0
RESIDU_FILE=0
JUMLAH_CONTAINER=$1

#menghitung banyaknya file yang dicopy kedalam container
let JUMLAH_FILE=`ls -l /percobaan2/ | wc -l`
let LIMIT_FILE=$JUMLAH_FILE/$JUMLAH_CONTAINER
let RESIDU_FILE=$JUMLAH_FILE%$JUMLAH_CONTAINER
if [ $RESIDU_FILE -eq 0 ]
then
let LIMIT_FILE=LIMIT_FILE+0
else
let LIMIT_FILE=LIMIT_FILE+1
fi

#main method
for f in $FILES
do
if [ $COUNTER_FILE -eq 1 ]
then
echo "Membuat container" $NAMA_CONTAINER$COUNTER_CONTAINER
docker run -i -t --name $NAMA_CONTAINER$COUNTER_CONTAINER -d
    agungputrap/riset_docker:latest
CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER`
echo "ini nomor container = " $CONTAINER_ID
```

```

cp /3LP10* /var/lib/docker/aufs/diff/$CONTAINER_ID/
#           memproses ligand menjadi .dpf dan mengcopy ke
           dalam container beserta ligand
echo "memproses file "$COUNTER_FILE
./home/agungpp/MGLTools-1.5.6/MGLToolsPckgs/AutoDockTools/
    Utilities24/prepare_dp4.py -l $f -r /3LP10.pdbqt
NAMA=`basename $f .pdbqt`
OUTPUT_PREPARE=$NAMA"_3LP10.dpf"
cp /$OUTPUT_PREPARE /var/lib/docker/aufs/diff/$CONTAINER_ID/
    $OUTPUT_PREPARE
echo "mengcopy file "$COUNTER_FILE " ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy " $COUNT
let COUNT=COUNT+1
let COUNTER_FILE=COUNTER_FILE+1
elif [ $COUNT -eq $JUMLAH_FILE ]
then
echo "harusnya nilai terakhir "$COUNT
echo "memproses file terakhir "
./home/agungpp/MGLTools-1.5.6/MGLToolsPckgs/AutoDockTools/
    Utilities24/prepare_dp4.py -l $f -r /3LP10.pdbqt
NAMA=`basename $f .pdbqt`
OUTPUT_PREPARE=$NAMA"_3LP10.dpf"
cp /$OUTPUT_PREPARE /var/lib/docker/aufs/diff/$CONTAINER_ID/
    $OUTPUT_PREPARE
echo "mengcopy file " $COUNTER_FILE " ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
cp /running_dock /var/lib/docker/aufs/diff/$CONTAINER_ID/
    running_dock
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /
    running_dock
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER ./running_dock
elif [ $COUNTER_FILE -lt $LIMIT_FILE ]
then
echo "memproses file " $COUNTER_FILE
./home/agungpp/MGLTools-1.5.6/MGLToolsPckgs/AutoDockTools/
    Utilities24/prepare_dp4.py -l $f -r /3LP10.pdbqt
NAMA=`basename $f .pdbqt`
OUTPUT_PREPARE=$NAMA"_3LP10.dpf"
cp /$OUTPUT_PREPARE /var/lib/docker/aufs/diff/$CONTAINER_ID/
    $OUTPUT_PREPARE
echo "mengcopy file "$COUNTER_FILE" ke container "
    $NAMA_CONTAINER$COUNTER_CONTAINER

```

```

cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy " $COUNT
let COUNT=COUNT+1
let COUNTER_FILE=COUNTER_FILE+1
#case paling terakhir dijalanin
#     elif [ $COUNT -eq $JUMLAH_FILE ]
#         then
#             echo "harusnya nilai terakhir "$COUNT
#             echo "mengcopy file terakhir ke container"
#             $NAMA_CONTAINER$COUNTER_CONTAINER
#             cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`
#             basename $f`
#             cp /running_vina /var/lib/docker/aufs/
#             $CONTAINER_ID/running_vina
#             docker exec -d
#             $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /running_vina
#             docker exec -d
#             $NAMA_CONTAINER$COUNTER_CONTAINER ./running_vina
else
#mengcopy file terakhir dalam suatu container dan menjalankan
#script dalam container
echo "memproses file " $COUNTER_FILE
./home/agungpp/MGLTools-1.5.6/MGLToolsPckgs/AutoDockTools/
Utilities24/prepare_dp4.py -l $f -r /3LP10.pdbqt
NAMA=`basename $f .pdbqt`
OUTPUT_PREPARE=$NAMA"_3LP10.dpf"
cp /$OUTPUT_PREPARE /var/lib/docker/aufs/diff/$CONTAINER_ID/
$OUTPUT_PREPARE
echo "mengcopy file "$COUNTER_FILE" ke container "
#             $NAMA_CONTAINER$COUNTER_CONTAINER
cp $f /var/lib/docker/aufs/diff/$CONTAINER_ID/`basename $f`
echo "file dicopy "$COUNT
let COUNT=COUNT+1
cp /running_dock /var/lib/docker/aufs/diff/$CONTAINER_ID/
running_dock
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER chmod u+x /
running_dock
docker exec -d $NAMA_CONTAINER$COUNTER_CONTAINER ./running_dock
let COUNTER_FILE=1
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
fi
done

```

**Daftar Kode****8:** Script yang dijalankan pada masing - masing container

```
#!/bin/bash
FILES="/*_3LP10.dpf"
echo "Waktu mulai " >> logdock.txt
date >> logdock.txt
for f in $FILES
do
./autodock/autodock4 -p $f -l `basename $f .dpf`.dlg
done
echo "Waktu selesai " >> logdock.txt
date >> logdock.txt
```

**Daftar Kode 9:** *Script* yang digunakan untuk mengambil hasil dari menjalankan aplikasi Autodock dari masing - masing container

```
#!/bin/bash
COUNTER_CONTAINER=1
COUNTER=0
NAMA_CONTAINER="dock"
FILES="/var/lib/docker/aufs/diff/"
JUMLAH_FILE=0
let JUMLAH_FILE=`ls -l /percobaan2/ | wc -l`
while [ $COUNTER -lt $JUMLAH_FILE ];
do
CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER `
FILEX="$FILES$CONTAINER_ID/*.dlg"
echo $FILEX
for f in $FILEX
do
docker cp $NAMA_CONTAINER$COUNTER_CONTAINER:`basename $f` /
    hasildock/
let COUNTER=COUNTER+1
done
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
done
```

**Daftar Kode 10:** *Script* yang digunakan untuk mengambil log dari masing - masing container

```
#!/bin/bash
COUNTER_CONTAINER=1
COUNTER=0
NAMA_CONTAINER="dock"
FILES="/var/lib/docker/aufs/diff/"
JUMLAH_FILE=$1
while [ $COUNTER -lt $JUMLAH_FILE ];
```

```
do
CONTAINER_ID=`docker inspect -f '{{.Id}}'
    $NAMA_CONTAINER$COUNTER_CONTAINER `
FILEX="$FILES$CONTAINER_ID/logdock*"
echo $FILEX
for f in $FILEX
do
docker cp $NAMA_CONTAINER$COUNTER_CONTAINER:`basename $f` /
    logdock/$NAMA_CONTAINER$COUNTER_CONTAINER/
let COUNTER=COUNTER+1
done
let COUNTER_CONTAINER=COUNTER_CONTAINER+1
done
```