

# ভার্চুয়ালাইজেশন ও ভার্চুয়াল মেশিন

ভার্চুয়াল ইনভায়রমেন্ট এর সাথে টেক লাভাররা কম বেশি সবাই পরিচিত। মোদ্যাকথায়, একটা ওএস এর মধ্যে ইন্সটল করা ছাড়া আরেকটা ওএস চালানো, সেইম হার্ডওয়্যার এ, এটাই হলো ভার্চুয়াল ইনভায়রনমেন্ট।

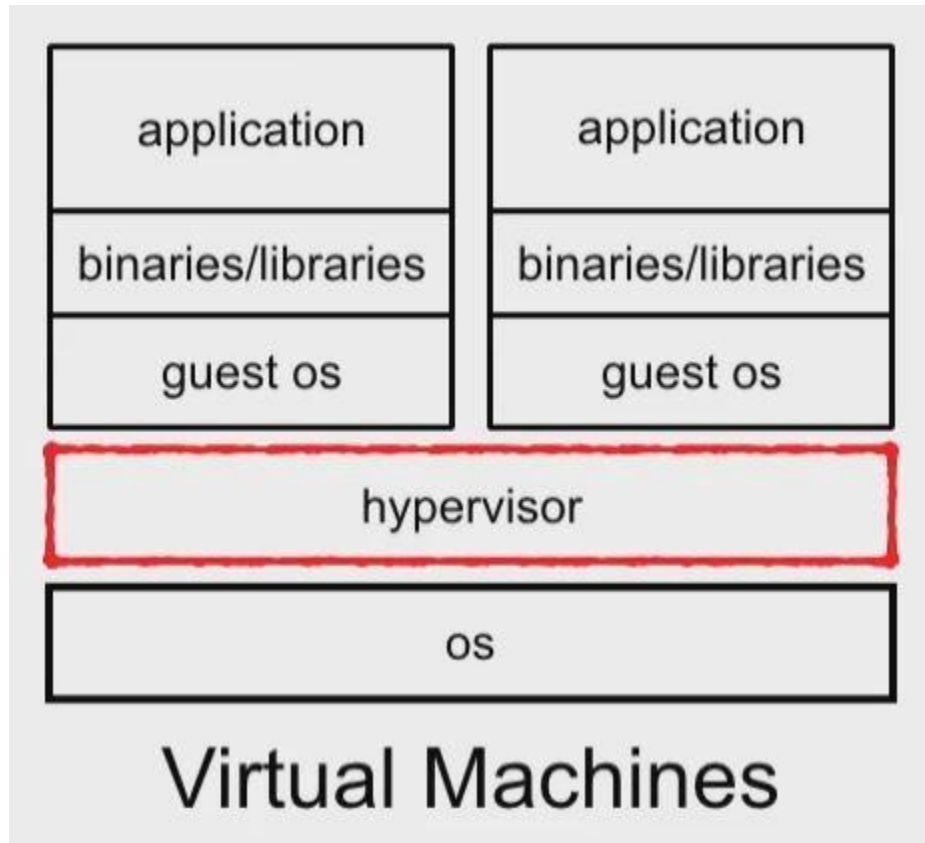
ভার্চুয়ালাইজেশনের সুবিধা কি?

ধরেন আপনি একটা সুন্দর বাড়ি তৈরি করলেন। এবার ভাড়াটিয়ারা যখন আপনার বাসায় থাকবে আপনি চাইলেন না আপনার সুন্দর বাড়িটি নষ্ট করুক। তাই তাদের মত করে একটা পরিবেশ বানিয়ে দিলেন, আপনারটাও আপনার মতই থেকে গেলো।

ক্লাউড হোস্টিং এ ঠিক এভাবে করেই আপনাকে হোস্টিং স্পেস দেয়া হয়। আপনি যখন কোন ভিপিএস হোস্ট সার্ভিস নিচ্ছেন আপনাকে একটি নতুন আইসোলেটেড সিস্টেম দেয়া হচ্ছে। আপনার কি মনে হয় আপনাকে তারা একটি নতুন পিসি দিয়েছে? না, মোটেই না। এতে করে তাদের ৫০ হাজার ইউজারদের জন্য ৫০ হাজার পিসি লাগতো। আবার একটা পিসিতে একটা ওএস এর সাথে আরেকটা ওএস বুট করাও যেতো না। এ সমস্যা দূর করার জন্যই আপনাকে তারা ভার্চুয়াল পিসি দেয়, যেখানে তারা তাদের মূল পিসির অরিজিনাল ওএস এর উপরেই আপনার পছন্দের ওএস দিয়ে দেয় যেটা দিয়ে আপনি ভিপিএস রান করবেন। অর্থাৎ একটা পিসির হার্ডওয়্যার বা ইনফ্রাস্ট্রাকচারের মধ্যেই অনেকজন হোস্টিং স্পেস পায়, যাদের আলাদা আলাদা ওএস, আলাদা সফটওয়্যার, আলাদা ভার্চুয়াল মেশিন থাকে। তাদের মাঝে কোন কনফ্লিক্ট ও হয় না।

হোস্টিং এর মূল পিসির হার্ডওয়্যারে যে ওএসটা থাকে তাকে বলা হয় “হোস্ট ওএস” আর তার উপরে ভার্চুয়ালাইজেশনে যে নতুন ওএস দেয়া হয় তাকে বলে “গেস্ট ওএস”। ভার্চুয়াল পিসিকে

“ভার্চুয়াল মেশিন” বলে। কখন কোন ভার্চুয়াল মেশিন তৈরি হবে, ডিলেট হবে ইত্যাদি কাজ করে “হাইপারভাইজর” নামক একটা সফটওয়্যার সিস্টেম।



আগেই বলেছি, গেস্ট ওএস এ আপনি নিজের ইচ্ছামত সফটওয়্যার ইন্সটল করে আপনার নিজের একটা আলাদা ইনভায়রনমেন্ট বানিয়ে নিতে পারেন। এতে করে হোস্ট ওএস এর সবকিছু অক্ষত থাকলো।

কিন্তু সমস্যা হলো, প্রতিটা গেস্ট ওএস রান করতে প্রতিবার বুট করতে হয়। তারমানে হোস্ট ওএস এর বুট টাইম তো আছেই, হোস্ট রানিং থাকলেও গেস্ট ওএস চালু করতে সময় লেগে যায়। এটা একটা মেজর সমস্যা।

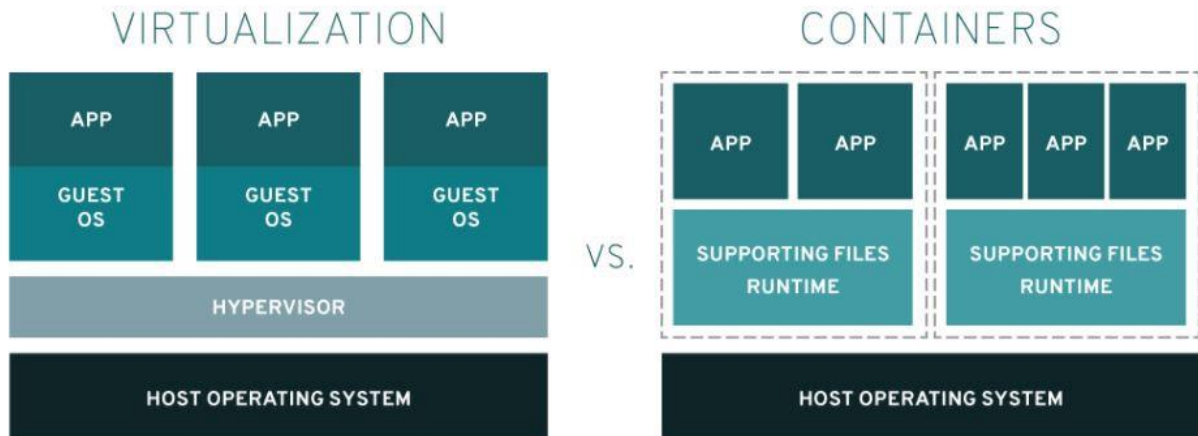
আরেকটা সমস্যা হলো, আপনি যদি আপনার নিজের পিসিতে ভার্চুয়লাইজেশন করে ডেভলপমেন্ট এর কাজ করতে চান, যখন আপনি প্রোডাকশন সার্ভারে শিপ করতে যাবেন তখন সেই সার্ভারে আপনাকে সবকিছু আবার নতুন করে সেটাপ করতে হবে, সকল সফটওয়্যার এর ভার্সন মিলানো সহ। একটা অমিল থাকলেই বাগ ধরা দিবে। তাই ভার্চুয়াল মেশিনের লোকাল সার্ভার থেকে প্রোডাকশন সার্ভারে যেতে গেলে এটা একটা মেজর ড্রব্যাক।

## ডকার ও লিনাক্স কন্টেইনার – ভার্চুয়লাইজেশনের নতুন রূপ

লিনাক্স কন্টেইনার বা LXC কী?

ভার্চুয়াল মেশিনে আমরা একটা হোস্ট ওএস এর উপরে আরেকটা গেস্ট ওএস ইন্সটল দিয়ে আলাদা আলাদা এনভায়রনমেন্ট বানিয়ে দেই। যারা শুধুমাত্র হার্ডওয়্যার রিসোর্স শেয়ার করে। কিন্তু এতে করে প্রতিটা ভার্চুয়াল মেশিনের জন্য আলাদা আলাদা রিসোর্স শেয়ার করতে হয়, যা অনেক এক্সপেনসিভ। যা অনেক সিপিইউ পাওয়ার, ডিস্ক স্পেস ও র‍্যাম ব্যবহার করে থাকে। কিন্তু কন্টেইনারে শুধুমাত্র একটি ওএস, দরকারি এপ্লিকেশন/লাইব্রেরি আর সিস্টেম রিসোর্স লাগে। যার ফলে একটা কন্টেইনারে ভার্চুয়াল মেশিনের তুলনায় অনেক বেশি পরিমাণে এপ্লিকেশন সার্ভ করতে পারবেন। অনেক মজার ব্যাপার তাই না?

কন্টেইনার কিভাবে কাজ করে এবং ভার্চুয়াল মেশিনের সাথে এর পার্থক্যগুলো কী কী?



কন্টেইনারে আলাদা কোন নতুন করে গেস্ট অপারেটিং সিস্টেম (ওএস) ইন্সটল করার কোন প্রয়োজন নেই। যার ফলে ভার্চুয়াল মেশিনের মত “ডাবল বুট টাইম” সমস্যাও আর হয় না। ভার্চুয়াল মেশিন যদি কেউ সেটাপ করে থাকেন “ভার্চুয়ালবক্স সফটওয়্যার” দিয়ে তাহলে দেখে থাকবেন প্রতিটা ভার্চুয়াল মেশিন বা **ভিএম** এর জন্য আলাদা আলাদা ডিস্ক স্পেস, সিপিইউ পাওয়ার ও র‍্যাম নির্দিষ্ট করে দিতে হয়। কিন্তু কন্টেইনারে যেহেতু কোন নতুন গেস্ট ওএস লাগে না তাই বাড়তি এসব কিছু নির্দিষ্ট করে লাগে না। কন্টেইনার কেবল হোস্ট ওএসই ব্যবহার করে, যাকে বলে “**ওএস লেভেল ভার্চুয়লাইজেশন**”। যার ফলে এসব কন্টেইনারের ওএস সুপারভাইজ করার জন্য আলাদা কোন হাইপারভাইজরও লাগে না। একটা সিস্টেমে সেটাপ করা সকল কন্টেইনার তার হোস্ট ওএস এর কার্নেল ব্যবহার করে থাকে আলাদা একটা এনভায়রনমেন্ট বানানোর জন্য।

### কন্টেইনারের কিছু সাধারণ বিষয়

**কন্টেইনার ইমেজ** হলো একটা কম্পোজড সিস্টেম যেখানে একটা ভার্চুয়াল এনভায়রনমেন্টের সকল এপ্লিকেশন সফটওয়্যার ও তার সকল ডিপেন্ডেন্সি দেয়া থাকে। যার ফলে এই ইমেজকে আপনি খুব সহজেই এক মেশিন থেকে অন্য মেশিনে ব্যবহার করতে পারবেন, সফটওয়্যার ডিপেন্ডেন্সির কোন ঝামেলা নেই, যা পূর্বেকার এই সমস্যার সমাধান করে “**But this works on my machine**”। ইমেজ বানিয়ে সেটাকে রান করলে ডকার প্রতিটা ইমেজের জন্য আলাদা আলাদা কন্টেইনার তৈরি করে। প্রতিটা কন্টেইনার এর স্বতন্ত্র ভার্চুয়লাইজড সিস্টেম থাকে।

এখন শুধু এটা আর বলা যাবে না যে আপনার এপ্লিকেশন সিস্টেম শুধু আপনার মেশিনে চলেছে অন্যটায় চলতেছে না। কারণ যে কন্টেইনার ইমেজ আপনি ব্যবহার করে ডেভলপমেন্ট ও টেস্ট করেছেন সেটাকে যেকোন মেশিনে পোর্ট করে আপনার ডেভলপ করা এপ্লিকেশন তাতে চালাতে পারবেন, রেজাল্ট একদম একই হবে। বুঝতে পেরেছি ব্যাপারটা?

কন্টেইনার এর আরেকটা বড় সুবিধা হচ্ছে **ডিপ্লয়মেন্ট**। আগে যখন আপনার লোকাল সার্ভারে ডেভলপমেন্ট করে ক্লাউডে সেটাকে হোস্ট করতে যেতেন তখন আপনাকে সকল এপ্লিকেশন (Apache, Nginx, MySQL, Python/django, PHP ইত্যাদি) আবার নতুন করে ভার্সন টু ভার্সন মিলিয়ে ক্লাউড হোস্ট সেটাপ করতে হতো। যেটা মোটেও সুবিধাজনক না। কিন্তু এখন আপনি আপনার ডেভলপমেন্টে ব্যবহার করা কন্টেইনার ইমেজকে খুব সহজেই ক্লাউড হোস্ট পোর্ট করে ফেলতে পারেন, এতে করে বাড়তি কোন ঝামেলা থাকবে না। কারণ আপনার কন্টেইনার ইমেজে তো আগে থেকেই সকল দরকারী এপ্লিকেশন ও ডিপেন্ডেন্সি দেয়াই আছে। দারুন সুবিধা তাই না?

তাহলে কন্টেইনার কি ভার্চুয়ালাইজেশনই?

হ্যাঁ এবং না দুটোই! আমরা জেনেছি ভার্চুয়ালাইজেশনে একটা সিঙ্গেল সিস্টেমে অনেকগুলি OS একসাথে চলে। কিন্তু কন্টেইনারে একটা সিঙ্গেল OS এর কার্নেলকে সবগুলি কন্টেইনারের সাথে শেয়ার করে প্রতিটা কন্টেইনারের জন্য আলাদা আলাদা পরিবেশ বানিয়ে দেয়। খুব সহজেই আমরা বুঝে গেছি কন্টেইনার হচ্ছে ভার্চুয়াল মেশিনের চাইতে অনেক অনেক লাইটওয়েট।

আজ তাহলে এ পর্যন্তই...

ওয়েট! ওয়েট! তাহলে **ডকার** কি?

ডকার হচ্ছে কন্টেইনারাইজ করার একটা এপ্লিকেশন সিস্টেম যেটা কন্টেইনার বানানোর প্রসেস থেকে শুরু করে সকল কিছু অনেক সহজ করে দিয়েছে। আপনি যদি উপরের তথ্যগুলি বুঝে

শুনে গিলতে পারেন তাহলে ডকার শেখা ব্যাপার হবে না। মূল কনসেপ্টটা ক্লিয়ার হয়েছে আশা করি। অনলাইনে একটু ডকার নিয়ে ঘাটুন, তবে আগে অবশ্যই ভার্চুয়ালাইজেশন ও কন্টেইনার সম্পর্কে সলিড ধারণা নিবেন।

কন্টেইনারের আরো সুবিধা হচ্ছে মাইক্রোসার্ভিস আর্কিটেকচার ও এপ্লিকেশন স্কেলিং। এগুলি নিয়ে আরেকটা লেখা হয়ে যাবে।

**তাহলে সামারাইজ করলে কী দাঁড়াচ্ছে?**

১. কন্টেইনার খুবই লাইটওয়েট, খুব কম হার্ডওয়্যার রিসোর্স ব্যবহার করে
২. কন্টেইনারের সাহায্যে “But this works on my machine” এর সমস্যা দূর করা গেছে
৩. কন্টেইনারের সাহায্যে ক্লাউড ডিপ্লয়মেন্ট খুব সহজ ও পেইনলেস হয়েছে

## অল্প কথায় স্বল্প Docker

এতদিনে অনেকেই **ডকার** নামটা শুনেছি। কিন্তু এটা কি জিনিস সেটা দশ জনকে জিজ্ঞেস করলে হয়তো বারো রকমের উত্তর পাওয়া যাবে! অনেকেই “ডকার কি জিনিস বা এটা কেন দরকার” যদিও কিছুটা বুঝি, অন্য কাউকে বোঝাতে গেলে আটকে যাই। তাই কথা খুব বেশি না বাড়িয়ে, অন্তত চায়ের টং-এ বা কফির দোকানে গল্প-আড্ডায় ডকার শব্দটা চলে আসলে কি বলতে হবে সেটা জেনে নেই চলুন!

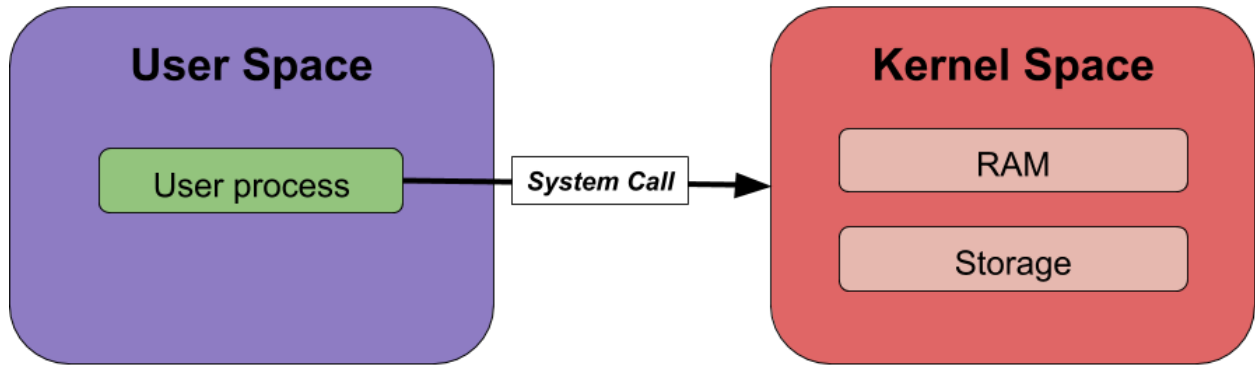
**ডকার কি ও কেন?**

মাইক্রোসার্বিস ডেভেলপার বা ডেভ-অপস যারা আছেন তাদের কাছে ডকার নামটা বেশ পরিচিত। একটু গভীরে গিয়ে ডকারের ব্যাপারে জানতে হলে **অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন** ([Operating System Level Virtualization](#)) কি জিনিস সেটা বুঝতে হবে। আপাতত নিচের সংজ্ঞাটা দেখে নেই, বিস্তারিত পরে আসছে।

*একটি কার্নেলের ওপর একাধিক isolated ইউসার স্পেসের অস্তিত্বকে অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন বলে [সোর্স: [Wikipedia](#)]*

একটু খুলে বললে আধুনিক অপারেটিং সিস্টেমগুলোতে আমাদের অ্যাপ্লিকেশনগুলো **ভার্চুয়াল মেমোরি** ([Virtual Memory](#)) নামক একটা মেমোরি এড্রেস স্পেসে চলে। এই ব্যাপারটা আমরা অনেকেই জানি। এভাবে ভার্চুয়াল মেমোরি ব্যবহার করার পিছনে অনেকগুলো কারনের মধ্যে অন্যতম একটি হচ্ছে **মেমোরি প্রটেকশন** ([Memory Protection](#)), মানে রান টাইমে একটি প্রসেস যাতে অন্য আরেকটি প্রসেসের মেমোরি স্পেসে (Memory Space) প্রবেশ করতে না পারে সেটা নিশ্চিত করা।

অপারেটিং সিস্টেমে কিছু প্রসেস আছে যাদের সিস্টেম রিসোর্সগুলো ([System Resource](#)) সরাসরি অ্যাক্সেস করার প্রিভিলেজ ([Privilege](#)) থেকে থাকে। এই প্রসেসগুলো সাধারণত কার্নেল ([Kernel](#)) এবং ডিভাইস ড্রাইভার ([Device Drivers](#)) হয়ে থাকে। এসমস্ত প্রিভিলেজসম্পন্ন প্রসেসসমূহের মেমোরি প্রটেকশনের জন্যে ভার্চুয়াল মেমোরি স্পেসের একাংশকে dedicate করে দেয়া হয়। এর কারনে ভার্চুয়াল মেমোরি দ্বিখণ্ডিত হয়। একটি খণ্ডে চলে কার্নেল আর ডিভাইস ড্রাইভারদের মত প্রিভিলেজসম্পন্ন প্রসেসসমূহ আর এই খণ্ডটা **কার্নেল স্পেস** ([Kernel Space](#)) নামে পরিচিত। আরেকটি খণ্ডে চলে বাকি সব সাধারণ প্রসেস যেমন ব্রাউসার, টেক্সট এডিটর, গেমস, ইত্যাদি আর এই খণ্ডটা **ইউসার স্পেস** ([User Space](#)) নামে পরিচিত।



প্রভিলেজের এরূপ বৈষম্যের কারনে ইউসার স্পেসের প্রসেসগুলো (বা ইউসার প্রসেস) সিস্টেম রিসোর্সগুলোকে সরাসরি এক্সেস করতে পারে না, বরং তারা কার্নেল ও ডিভাইস ড্রাইভারদের মাধ্যমে সিস্টেম রিসোর্সগুলো এক্সেস করে থাকে। আর এটা করতে ইউসার প্রসেস কার্নেলের সাথে **সিস্টেম কল** ([System Call](#)) এর মাধ্যমে communicate করে।

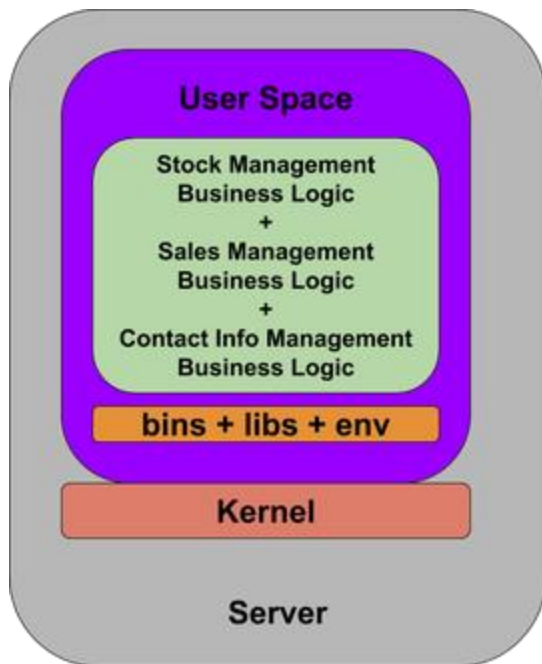
*Patience (sometimes) is a virtue!*

অনেক এদিক সেদিক করলাম, এখন তাহলে মূল কথায় আসা যাক।

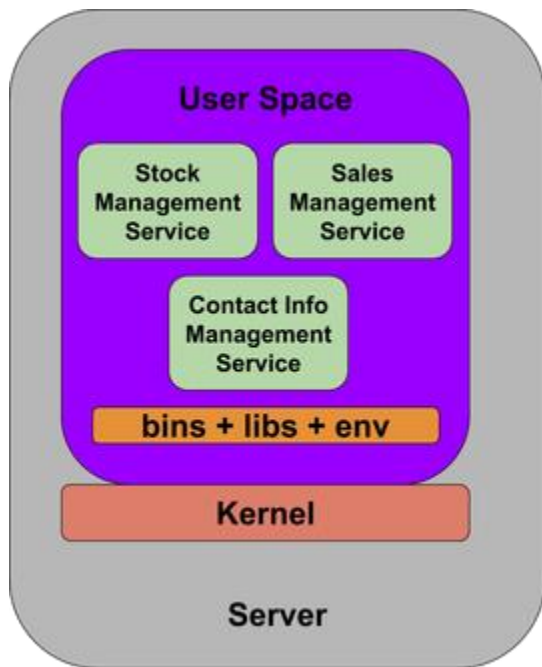
মূল কথাঃ এক

একটা সময় ছিল যখন একটা সার্ভার আর একটা সার্ভিস-অ্যাপ্লিকেশন হাজারটা কাজ করত। মানে মনে করেন আপনার একটা ইনভেন্টরি ম্যানেজমেন্ট ওয়েব অ্যাপ্লিকেশন আছে। আর সেই ওয়েব অ্যাপ্লিকেশনের জন্য একটা ফ্রন্টএন্ড, একটা ব্যাকএন্ড আর একটা ডেটাবেস আছে। এই ধরনের অ্যাপ্লিকেশন আর্কিটেকচারকে বলা হয় থ্রি-টিয়ার-আর্কিটেকচার ([Three-Tier-Architecture](#))। আর আপনার সেই একটা ব্যাকএন্ডেই সব বিজনেস লজিক ([business logic](#)) handle করে। মানে অথেনটিকেশন থেকে শুরু করে ইনভেন্টরির যত কাজ কর্ম, যেমন স্টক ম্যানেজমেন্ট, সেলস ম্যানেজমেন্ট, কন্টাক্ট ইনফরমেশন ম্যানেজমেন্ট, ডেটাবেস এক্সেস, লগিং, টেস্টিং সব! এই ধরনের ব্যাকএন্ড আর্কিটেকচারকে বলা হয় **মনোলিথিক-আর্কিটেকচার** ([Monolithic-Architecture](#))।





মনোলিথিক-আর্কিটেকচার



মাইক্রোসার্ভিস-আর্কিটেকচার

মনোলিথিক আর্কিটেকচারের সমস্যা কোথায় সেই কথা বলতে গেলে সময় ফুরিয়ে যাবে। যাই হোক, মোদাকথা একটা ব্যাকএন্ড সার্ভিস দ্বারা সব কাজ করার বিপরীতে কাজগুলোকে ছোট ছোট সার্ভিসে ভেঙ্গে দেয়াটাই অনেক সময়ে শ্রেয়। মানে ইনভেন্টরির স্টক ম্যানেজমেন্টের জন্য একটা সার্ভিস, সেলস ম্যানেজমেন্টের জন্য একটা সার্ভিস, কন্টাক্ট ইনফরমেশন ম্যানেজমেন্টের জন্য একটা সার্ভিস, অথেনটিকেশনের জন্য একটা সার্ভিস, লগিং-এর জন্য একটা সার্ভিস, ইত্যাদি। এই ধরনের অ্যাপ্লিকেশন আর্কিটেকচারকে বলা হয় **মাইক্রোসার্ভিস-**

**আর্কিটেকচার** ([Microservice-Architecture](#))।

গিটু লেগে যাচ্ছে এতসব জটিল জটিল কথাবার্তায়? ভয় পাবার কিছু নেই, কিছুক্ষনের মধ্যেই সব জলবৎ তরলং হয়ে যাবে।

ইউসার স্পেইস কি জিনিস সেটা আমরা দেখলাম। আর মাইক্রোসার্ভিস কি জিনিস সেটাও একটু আকটু বুঝলাম। এখানে লক্ষণীয় ব্যাপার এই যে আমাদের মাইক্রোসার্ভিসগুলো কিন্তু সাধারণত ইউসার অ্যাপ্লিকেশন হয়ে থাকে। মানে তারা ইউসার স্পেইসে চলে। আর তারা যেই সার্ভারে থাকে সেই সার্ভারের সিস্টেম রিসোর্সগুলো এক্সেস করতে তাদের সেই সার্ভারের কার্নেলের সাহায্য নিতে হয়।

আর তাই আমাদের ইনভেন্টরি ম্যানেজমেন্টের **আলাদা আলাদা সার্ভিসগুলো** (বা মাইক্রোসার্ভিসগুলো), যেমন স্টক ম্যানেজমেন্ট সার্ভিস, সেলস ম্যানেজমেন্ট সার্ভিস, কন্টাক্ট ইনফরমেশন ম্যানেজমেন্ট সার্ভিস, ইত্যাদি **সবগুলো যদি একটা সার্ভারে থাকে** তাহলে সবগুলো সার্ভিসই **একই ইউসার স্পেইস** শেয়ার করবে আর **একই কার্নেলের** সাথে কথা বলবে।

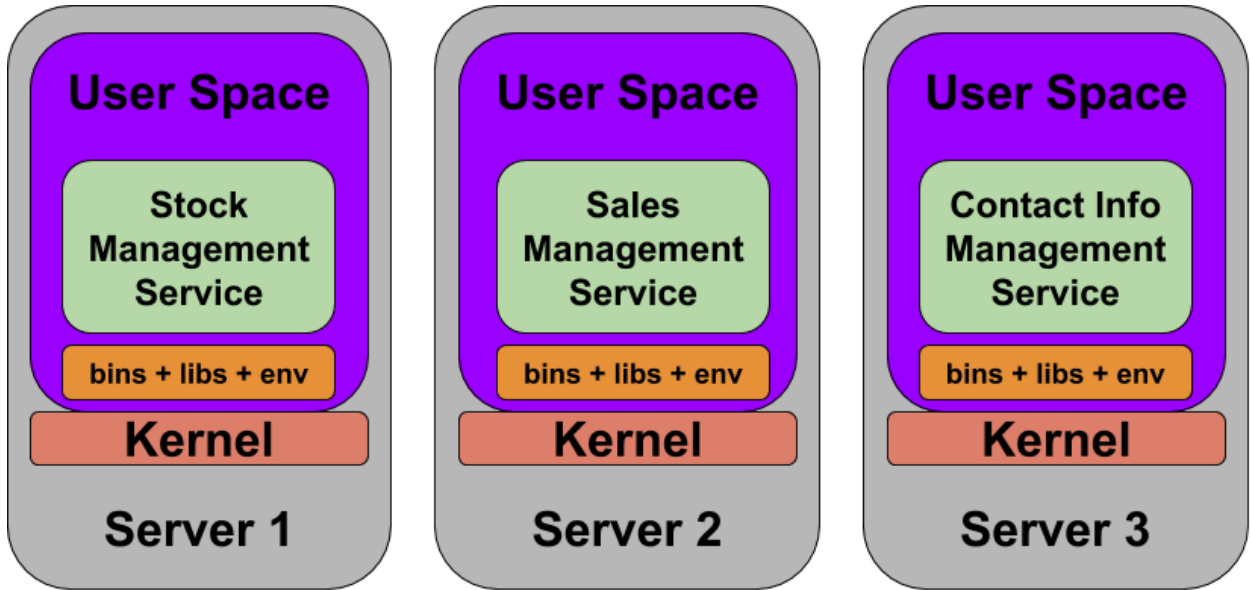
But life is not that simple. আমাদের প্রত্যেকটা সার্ভিস যে একই রানটাইম এনভায়রনমেন্ট ([Runtime Environment](#)) বা একই ইউসার সেটিংসের ওপর নির্ভরশীল হবে সেটার কি গ্যারান্টি? বরং অধিকাংশ ক্ষেত্রেই **সার্ভিসগুলো ভিন্ন ভিন্ন রানটাইম এনভায়রনমেন্ট আর ভিন্ন ভিন্ন ইউসার সেটিংসের ওপর চলে।** সেটাই বেশি কমন।

মানে মনে করেন আমাদের স্টক ম্যানেজমেন্ট সার্ভিসটি চলে পাইথনের জ্যাংগো ফ্রেইমওয়ারকে, আমাদের সেলস ম্যানেজমেন্ট সার্ভিসটি চলে জাভার স্প্রিং ফ্রেইমওয়ারকে, ইত্যাদি। শুধু তাই না: আবার এমনও হতে পারে যে আমাদের দুই বা তার অধিক সার্ভিস একই পোর্টে চলে। বা একই এনভায়রনমেন্ট ভ্যারিয়েবল ([Environment Variable](#)) ব্যবহার করে, কিন্তু তার ভ্যালু ভিন্ন।

এখানে একটু বলে নেয়া ভালো যে, আমরা যারা ছোটোখাটো ডেভেলপার তাঁদের হয়তো মনে হতে পারে যে আমার একটা প্রোজেক্টকে এরকম ছোটো ছোটো সার্ভিসে ভাগ কেন করবো? আবার সেই সব সার্ভিস একেকটা আলাদা আলাদা ল্যাস্‌সুয়েজ, ফ্রেমওয়ার্কে কেন চলবে? মূলত ছোটোখাটো এপ্লিকেশনের জন্য চিন্তা করলে এটা অদ্ভুত মনে হতেই পারে। কিন্তু আপনি যখন অনেক বড় মাপের কোনো এপ্লিকেশন নিয়ে কাজ করবেন যার লাখ লাখ ইউজার তখন আপনার সার্ভিসকে আরো রিলায়েবল, এফিসিয়েন্ট করার জন্য *অনেক ক্ষেত্রেই* মাইক্রোসার্ভিসই ভালো আর্কিটেকচার বলে মনে হতে পারে। মাইক্রোসার্ভিস কেন, কখন, কোথায় দরকার সেটা যেহেতু এই আর্টিকেলের আলোচ্য বিষয় না, তাই আপাতত ওদিকে যাচ্ছি না।

যাই হোক মূল কথায় ফিরে আসি। এধরনের অনেক কারনে সার্ভিসগুলো একই ইউসার স্পেসেই চালানো মোটেই ভাল আইডিয়া না। তাহলে কি করবো? **একেকটা সার্ভিসের জন্য একেকটা physical server চালাবো?**

কিন্তু বাস্তবতা হল ডেভেলপাররা অলস, আর কোম্পানিরা গরিব! তাই চাইলেও অধিকাংশ সময়ে প্রতিটা সার্ভিসের জন্য আলাদা আলাদা ফিজিক্যাল সার্ভার রাখাটা feasible না।



একেকটা সার্ভিসের জন্য একেকটা ফিজিক্যাল সার্ভার। এটা অধিকাংশ সময়েই feasible না।

তাহলে উপায়? আমরা দেখেছি যে একটা সার্ভারে একটা কার্নেল থাকে আর একটা ইউসার স্পেস থাকে। আর আমাদের সমস্যাটাতো ইউসার স্পেস নিয়ে। কার্নেল নিয়ে তো আমাদের কোনও ইস্যু নেই। আমরা যদি কোন কায়দায় **একটা সার্ভারের একটা কার্নেলের ওপরেই অনেকগুলো ইউসার স্পেস চালাতে পারতাম? প্রত্যেকটা ইউসার স্পেসে একটা করে সার্ভিস চলবে। তাহলে ব্যাপারটা কেমন হয়?**

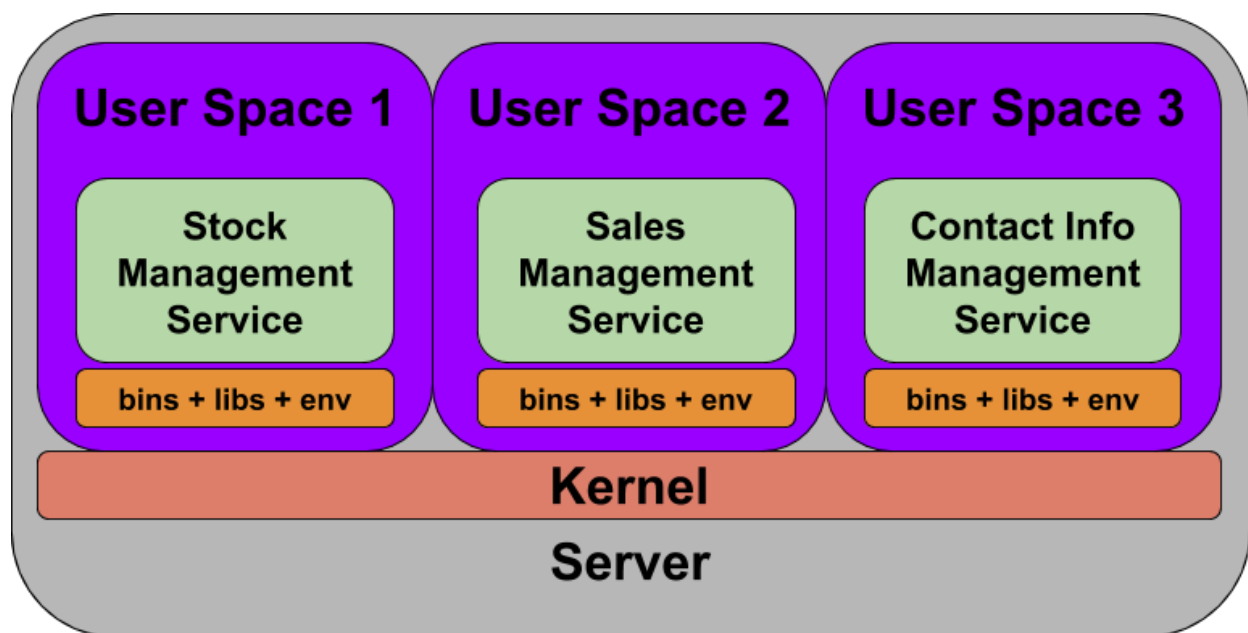
**অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশনের** সংজ্ঞাটা মনে আছে তো? আচ্ছা আবার একবার দেখে নেই নিচে:

*একটি কার্নেলের ওপর একাধিক isolated ইউসার স্পেসের অস্তিত্বকে অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন বলে [সোর্স: [wikipedia](https://en.wikipedia.org/wiki/Operating_system_virtualization)]*

ওকে, তো যেভাবে ভাবছিলাম সেটার সাথে **অপারেটিং সিস্টেম লেভেল**

**ভার্চুয়ালাইজেশনের** সংজ্ঞাটার মিল পাওয়া যাচ্ছে। লক্ষ্য করুন সংজ্ঞাটাতে **isolated** শব্দটা ব্যবহার করা হয়েছে। **Isolated** বলতে এখানে বোঝানো হচ্ছে **প্রতিটা ইউসার স্পেসের রানটাইম এনভায়রনমেন্ট, ইউসার সেটিংস, ইত্যাদি ভিন্ন ভিন্ন**। আর একটা ইউসার স্পেসের প্রসেস আরেকটা ইউসার স্পেসে প্রবেশ করতে পারে না। মানে মেমোরি প্রটেকশন, যেটার ব্যাপারে আগে বলেছি।

গ্রেট, তাহলে তাই করি! মানে একটা সার্ভারে একটা কানেলের ওপর অনেকগুলো isolated ইউসার স্পেস, আর একেকটা ইউসার স্পেসে একেকটা সার্ভিস চলবে।

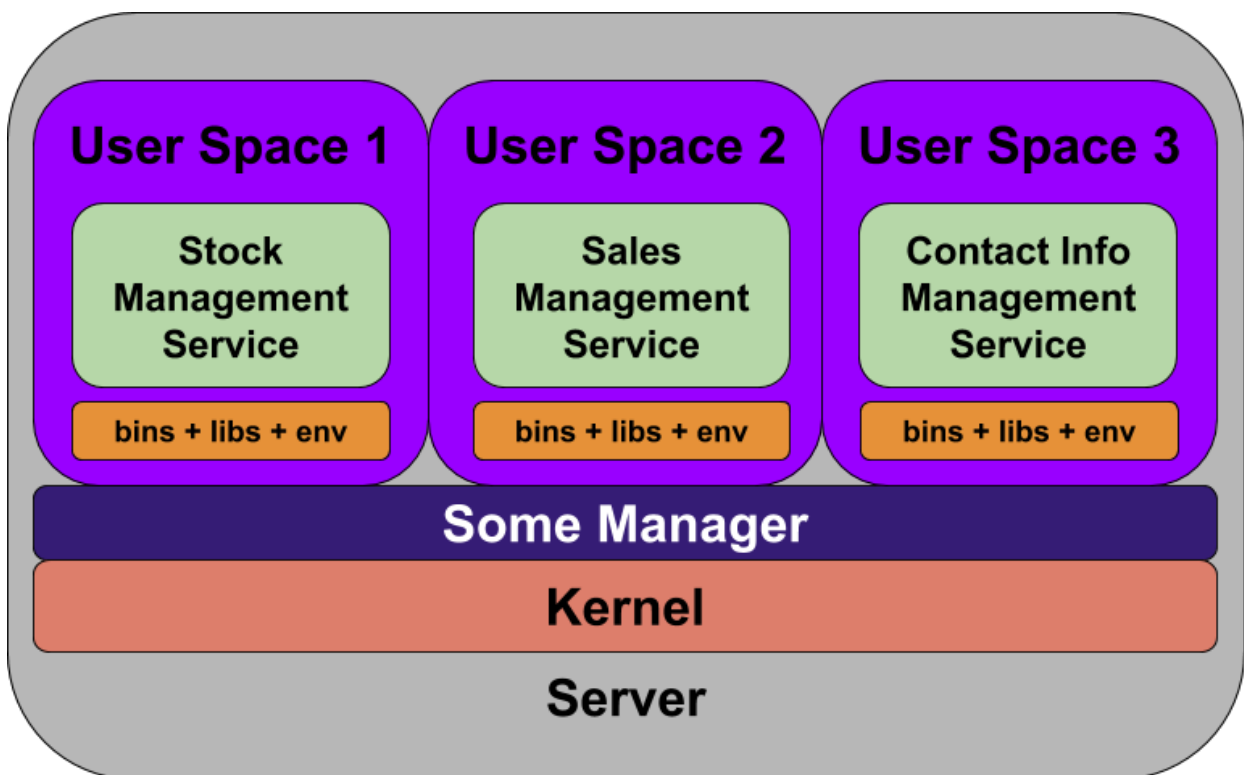


অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন

মানে পাইথনের জ্যাংগো ফ্রেইমওয়ারকের ওপর স্টক ম্যানেজমেন্ট সার্ভিসটি একটা ইউসার স্পেসে চলবে, জাভার স্প্রিং ফ্রেইমওয়ারকের ওপর সেলস ম্যানেজমেন্ট সার্ভিসটি আরেকটা ইউসার স্পেসে চলবে, ইত্যাদি।

অসাধারণ! সব চিন্তা শেষ! অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন for the win!

কিন্তু চিন্তা শেষ হয়েও হয়না শেষ! এই isolation মেইন্টেইন করার দায়িত্ব কে নেবে?! আরও লক্ষ্য করুন, একটা কার্নেলের ওপর অনেকগুলো ইউসার স্পেস। মানে একই সিস্টেম রিসোর্স বিভিন্ন ইউসার স্পেসের প্রসেস একই সময়ে এক্সেস করতে চাইতে পারে। আবার একটা ইউসার স্পেসের প্রসেস আরেকটা ইউসার স্পেসের প্রসেসের সাথে কথা বলতে চাইতে পারে। এক কথায় অনেকগুলো জটিলতা দেখা দিবে আর এসব ম্যানেজ করার জন্য কিছু একটা লাগবে। মানে ইউসার স্পেসগুলোর সার্বিক ব্যবস্থাপনা, তাদের কার্নেলের সাথে যোগাযোগ এবং একে ওপরের সাথে যোগাযোগের জন্যে ম্যানেজারের মত কিছু একটা দরকার।

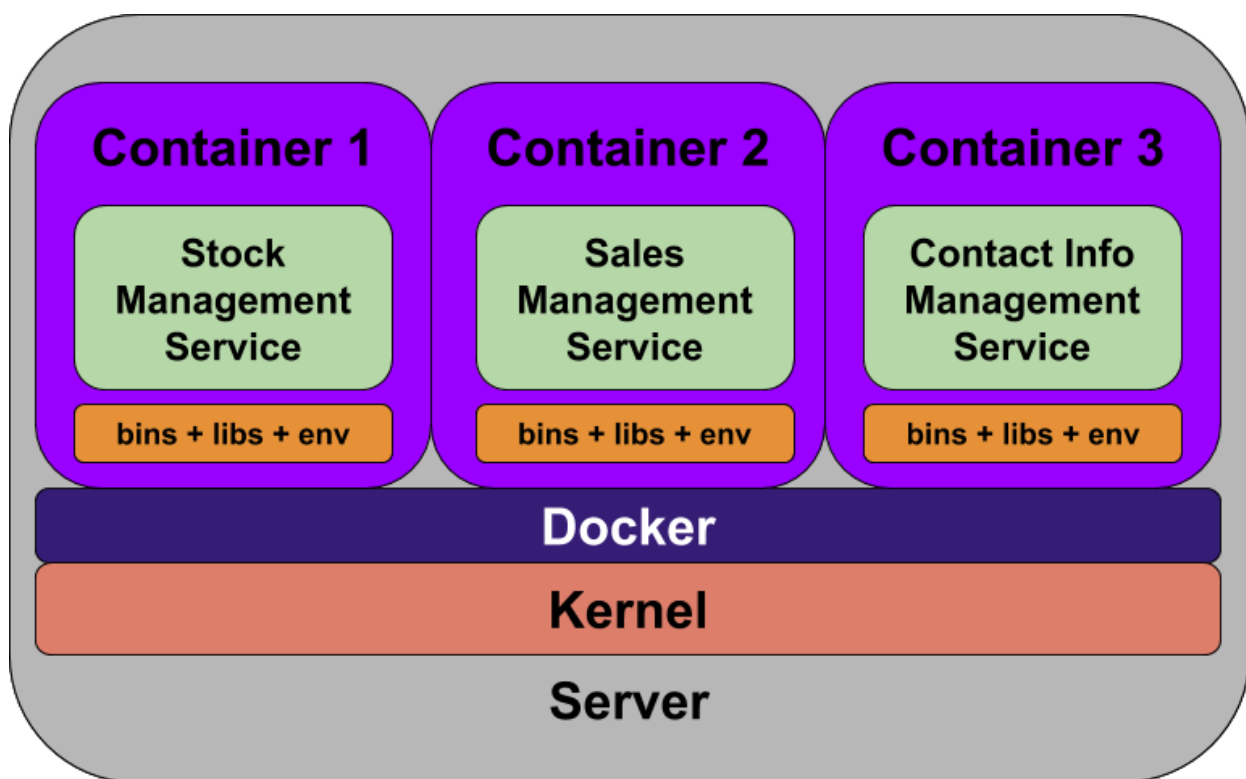


অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন

আশা করি এতক্ষণে অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন কি জিনিস আর কেন লাগতে পারে সেটার একটা ধারণা পেয়েছি। কিন্তু এটাতো একটা কনসেপ্ট মাত্র। এই কনসেপ্টের

অনেকগুলো ইমপ্লিমেন্টেশন আছে। অপারেটিং সিস্টেম লেভেল ভার্চুয়ালাইজেশন কনসেপ্টটার অনেকগুলো ইমপ্লিমেন্টেশনের মধ্যে অন্যতম একটা ইমপ্লিমেন্টেশন হচ্ছে ডকার (**Docker**) !

আর একটা কার্নেলের ওপর আলাদা আলাদা isolated ইউসার স্পেসগুলোকে **কন্টেইনার** (**Container**) বলা হয়। [ বাস্তবে সবগুলো কন্টেইনারই একই ইউসার স্পেসের ভেতরে থাকে, কিন্তু ভার্চুয়ালি আলাদা আলাদা ইউসার স্পেস মনে হয়]



ডকার

মূল কথাঃ দুই

একটা সময় ছিল যখন একটা সার্ভারেই কোড ডেভেলপ করা হতো, সেই সার্ভারেই টেস্ট হতো, আর সেই সার্ভারেই ডিপ্লয় হতো। কিন্তু এখন অধিকাংশ সময়েই একটা প্রজেক্টের পিছে অনেকজন পারসোনেল থাকেন আর একটা অ্যাপ্লিকেশন অনেকগুলো সার্ভারে ডিপ্লয় করা হয়।

মানে মনে করেন আমাদের ইনভেন্টরি ম্যানেজমেন্ট প্রজেক্টের কোডটা হয়তো লেখা হয় কয়েকজন ডেভেলপারের ল্যাপটপে, সেটা টেস্ট করা হয় একটা লোকাল সার্ভারে আর সেটা ডিপ্লয় করা হয় [AWS-এর EC2-তে](#)।

এর কারনে প্রায় সময়েই দেখা যায় দুজন ভিন্ন ডেভেলপারের ল্যাপটপের রানটাইম এনভায়রনমেন্ট ভিন্ন হওয়ায় অ্যাপ্লিকেশনটাও ভিন্ন ভাবে behave করে। আবার এমনটাও হয় যে ল্যাপটপে সব ঠিকঠাক কিন্তু লোকাল সার্ভারে টেস্ট করার সময়ে অ্যাপ্লিকেশনটা ঠিক মত চলছে না কারন সার্ভারের এনভায়রনমেন্ট ল্যাপটপের এনভায়রনমেন্ট থেকে ভিন্ন। আর এইসব এনভায়রনমেন্ট ঠিকঠাক করতেই সবার দিন রাত শেষ!

ফাঁকিবাজ ডেভেলপারদের খুব কমন একটা এক্সকিউজ :3

আমরা যদি কোন কায়দায় প্রতিটা ডেভেলপমেন্ট ল্যাপটপে আর লোকাল সার্ভারে আর EC2-তেও, মানে সবগুলো জায়গায়, একই ডকার কন্টেইনারে আমাদের অ্যাপ্লিকেশনটা চালাতে পারতাম? সবগুলো জায়গায় একই রানটাইম এনভায়রনমেন্ট আর সেটিংস নিশ্চিত হতো। তাহলে ব্যাপারটা কেমন হয়?

ঠিক এই কাজটা করাই ডকার কন্টেইনার ইমেইজের কাজ। আরো একটা সংজ্ঞা দেখে নেই।

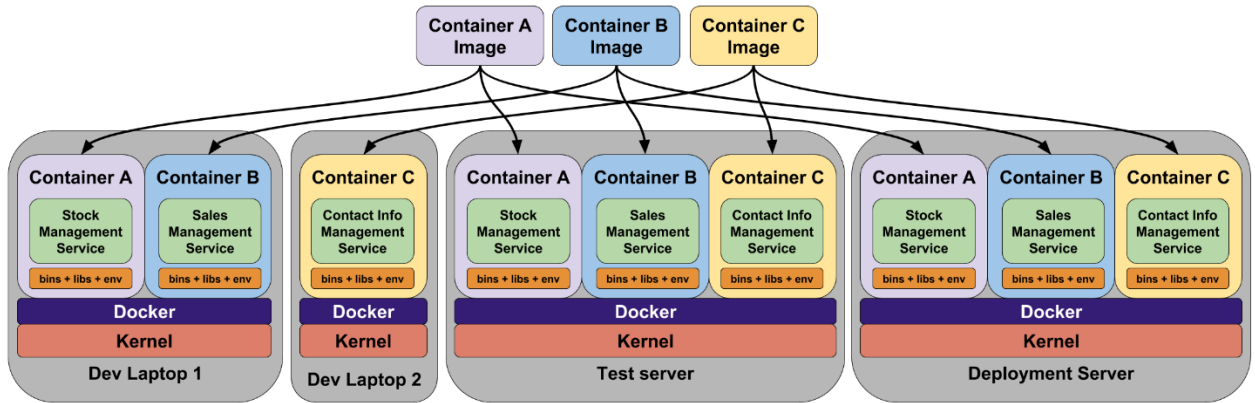
*A **docker container image** is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries and settings. [সোর্স: [docker.com](https://docs.docker.com/)]*

মোদাকথা আমাদের অ্যাপ্লিকেশন কোড, আর সেই কোড রান করার জন্য যাবতীয় রানটাইম এনভায়রনমেন্ট আর সেটিংস সব কিছু মিলেই একটা ডকার কন্টেইনার হয়।



সেটা আমরা দেখলাম। আর এই কন্টেইনারটার executable package কেই বলে কন্টেইনার ইমেইজ।

মূলত কন্টেইনার ইমেইজ হচ্ছে একটা ডকার কন্টেইনার বানানোর description বা বিবরণ। Executable package বলতে এখানে এটা বোঝানো হচ্ছে যে একটা কন্টেইনার ইমেইজের মাধ্যমে যেকোনো মেশিনেই একই ডকার কন্টেইনার রান করা যায়। [অবশ্যই মেশিনগুলোতে ডকার ইন্সটল করা থাকতে হবে]



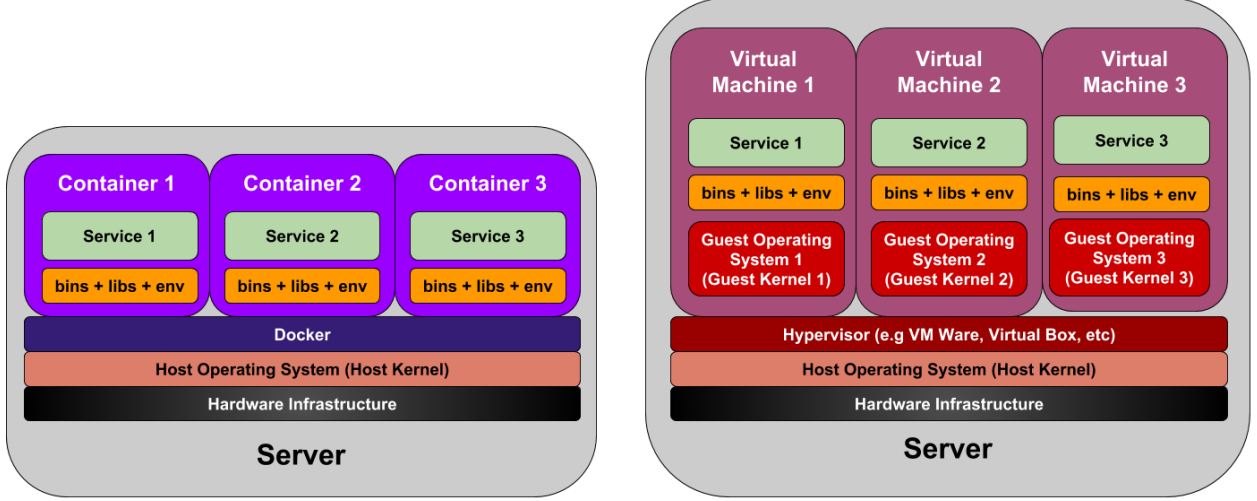
কন্টেইনার ইমেইজ

কন্টেইনার ইমেইজের দ্বারা একটা কন্টেইনার বিভিন্ন মেশিনে শেয়ার করা যায়। আরো সহজে বললে কন্টেইনার ইমেইজ হচ্ছে একটা কন্টেইনারের shareable স্ল্যাপশট।

## ডকার ও ভার্চুয়াল মেশিন

ডকারের প্রথম রিলিজ হয় ২০১৩ সালে। এর আগ পর্যন্ত এনভায়রনমেন্ট ভার্চুয়ালাইজেশনের কাজটা মূলত ভার্চুয়াল বক্স আর ভি.এম.ওয়ারের মত বিভিন্ন ভার্চুয়াল মেশিন ([Virtual Machine](#)) দিয়ে করা হতো। আমরা অনেকেই ভার্চুয়াল মেশিন কম-বেশি ব্যবহার করেছি। কন্টেইনার আর ভার্চুয়াল মেশিনের মধ্যে মৌলিক পার্থক্যটা এই যে কন্টেইনারদের নিজস্ব কার্নেল থাকে না, তারা সার্ভারের হোস্ট অপারেটিং সিস্টেমের

কার্নেলটাই শেয়ার করে। আর ভার্চুয়াল মেশিনেরা সবাই নিজস্ব আলাদা আলাদা অপারেটিং সিস্টেম আর কার্নেল ব্যবহার করে।



ডকার আর ভার্চুয়াল মেশিনের আর্কিটেকচারের পার্থক্য

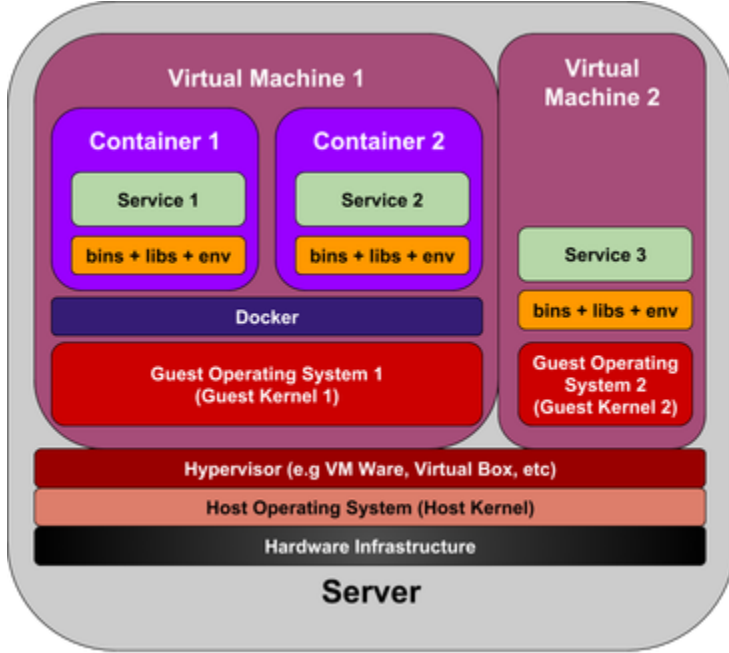
প্রত্যেকটা ভার্চুয়াল মেশিনের নিজ নিজ অপারেটিং সিস্টেম আর কার্নেল থাকায় স্বাভাবিক ভাবেই ভার্চুয়াল মেশিনের ইমেইজগুলো (বা iso ফাইলগুলো) ডকারের কন্টেইনার ইমেইজগুলোর থেকে অনেক বড় হয়। আর একই কারনে ভার্চুয়াল মেশিন চালু হতেও অনেক বেশি সময় নেয়। এসব কারণ ছাড়াও আরও বেশ কিছু কারণে অনেক ক্ষেত্রেই ভার্চুয়াল মেশিনের চেয়ে ডকারের ব্যবহারটাই আজকাল বেশি জনপ্রিয়তা পাচ্ছে। নিচে আরও কিছু পার্থক্য মেনশান করলাম [সাধারণ ক্ষেত্রে এই পার্থক্যগুলো দেখা যায়]।

ডকার	ভার্চুয়াল মেশিন	বিজয়ী
একটা কার্নেল শেয়ার করে	নিজ নিজ কার্নেল থাকে	স্পিডের দিক দিয়ে ডকার আর সিকিউরিটির দিক দিয়ে ভার্চুয়াল মেশিন
কন্টেইনার ইমেইজ ফাইলের সাইজ সাধারণত কয়েকশত মেগাবাইট হয়	iso ইমেইজ ফাইলের সাইজ সাধারণত কয়েক গিগাবাইট হয়	ডকার
চালু হতে কয়েক সেকেন্ড নেয়	চালু হতে কয়েক মিনিট নেয়	ডকার
রানটাইম স্ল্যাপশট নিতে সাধারণত কয়েক সেকেন্ড সময় লাগে	রানটাইম স্ল্যাপশট নিতে সাধারণত কয়েক মিনিট সময় লাগে	ডকার
গিট-এর মত ছোট ছোট কমিট বা লেয়ারে লেয়ারে ডেভেলপ করা যায়	এমন কিছু করা যায় না	ডকার
কন্টেইনারগুলো লেয়ারে লেয়ারে থাকে এবং তারা লেয়ারগুলো শেয়ার করে	এমন কিছু করে না	ডকার
Stateless	Statefull	ডকার
সাধারণ ল্যাপটপে একসাথে অনেকগুলো কন্টেইনার চালাতে যায়	সাধারণ ল্যাপটপে একসাথে দুই তিনটার বেশি ভার্চুয়াল মেশিন চালাতে যায় না	ডকার
গুণু লিনাক্স কার্নেলের কন্টেইনার ইমেজ হয়	লিনাক্স বা উইন্ডোজ থেকে কোনো অপারেটিং সিস্টেমেরই iso ফাইল হয়	ভার্চুয়াল মেশিন

ডকার আর ভার্চুয়াল মেশিনের পার্থক্য

ডকারের সাথে ভার্চুয়াল মেশিন ?!

আমরা চাইলে কিন্তু [ভার্চুয়াল মেশিন আর ডকার একসাথেও ব্যবহার করতে পারি](#)। এটা নিয়ে বেশি কিছু বলবো না। নিচের ছবিটাতে ব্যাপারটা বোঝানো হয়েছে।



ভার্চুয়াল মেশিনের সাথে ডকার

ডকার এ পাইথন ওয়েব এপ্লিকেশন চালানো এবং  
ডকার এর মূল ধারণা

প্রফেশনাল কাজের জন্য আমি একই সাথে ব্যাক এন্ড এবং ডিপ্লয়মেন্ট, মনিটরিং, স্কেলিং অর্থাৎ ডেভ অপ্স এর কাজ গুলিও করে থাকি। ডেভ অপ্স এর একটা গুরুত্বপূর্ণ জিনিস হলো ডকার।

তো এই ডকার নিয়ে আজকের এই পোস্ট। এটি একই সাথে তথ্যীয় গাইড এবং উদাহরণ হিসেবে পাইথন ওয়েব এপ্লিকেশন রান করা দেখাবো। তো চলুন দেরি না করে শুরু করি।

## পূর্ব শর্ত

ভার্চুয়লাইজেশন ও কন্টেইনারাইজেশন সম্পর্কে ধারণা, পাইথন, অল্প স্বল্প লিনাক্স, বেসিক ওয়েব ডেভলপমেন্ট ইত্যাদি।

## ডকার এর কনসেপ্ট

ডকার নিয়ে জানার আগে আপনাকে জানতে হবে ভার্চুয়লাইজেশন ও কন্টেইনারাইজেশন সম্পর্কে। আমি এগুলি নিয়ে আগে ২ টা আর্টিকেল লিখেছিঃ

০১. ভার্চুয়লাইজেশন ও ভার্চুয়াল মেশিন – <https://goo.gl/i35t6j>

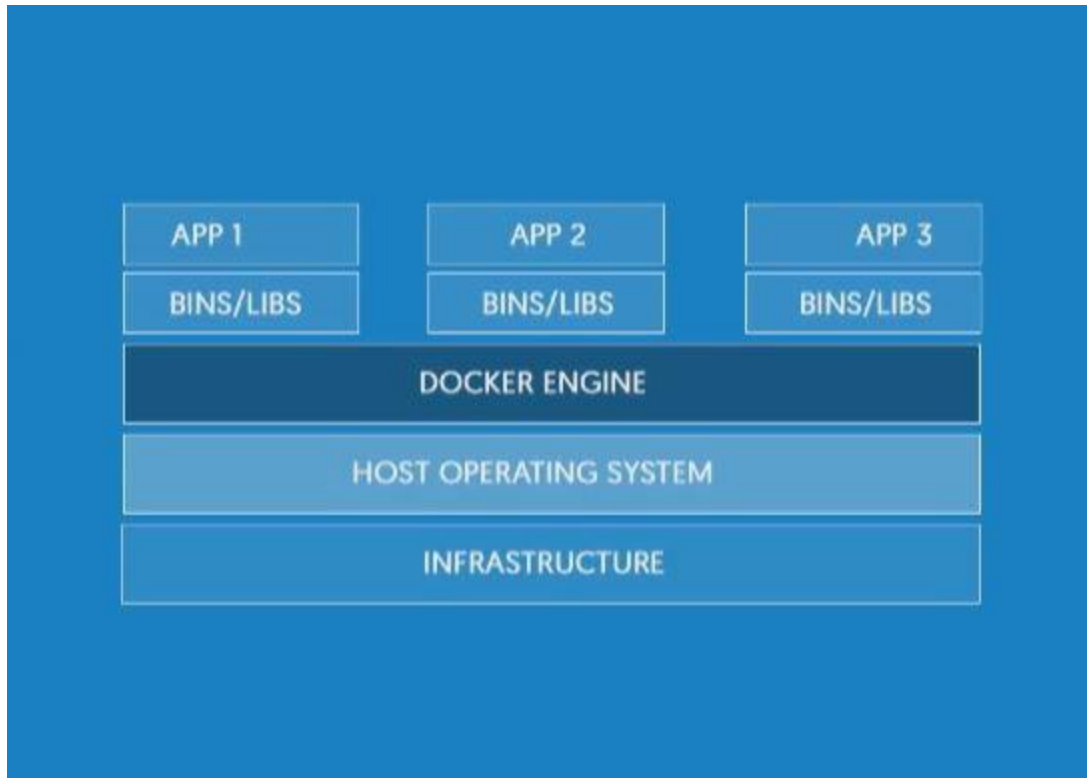
০২. ডকার ও লিনাক্স কন্টেইনার – ভার্চুয়লাইজেশনের নতুন রূপ – <https://goo.gl/2SzhAF>

ঝটপট পড়ে ফেলুন। এরপর আমরা সামনে আগাবো।

## ডকার কি?

ডকার হচ্ছে একটা এপ্লিকেশন সিস্টেম যেটা আপনাকে কন্টেইনার নির্ভর ভার্চুয়লাইজেশন এর সুবিধা দিয়ে থাকে। ডকার কন্টেইনারে আপনি সবকিছু পাবেনঃ রানটাইম, সিস্টেম টুলস, সিস্টেম লাইব্রেরি – মোট কথা প্রতিটা ডকার কন্টেইনার এক একটা ছোটখাটো লিনাক্স সার্ভার।

কন্টেইনারে কি কি থাকে তা আগের পোস্টে জানিয়েছিলাম তাই আবার রিপিট করলাম না।



ডকার এর আর্কিটেকচার

ডকার ইন্সটল করা

ডকার পুরোটাই লিনাক্স কার্নেল এর উপর ভিত্তি করে তৈরি। তাই লিনাক্সে ডকার চালানো খুব সহজ উইন্ডোজ এর চাইতে।

**Ubuntu:** `sudo apt-get install docker docker-engine docker.io`

**Windows:** যদি আপনি উইন্ডোজ ১০ এর চেয়ে কম ভার্সন ব্যবহার করে থাকেন তাহলে [এই লিঙ্ক](#) এ গিয়ে ডকার টুলবক্স ডাউনলোড করে ইন্সটল করুন।

আর যদি উইন্ডোজ ১০ এর ব্যবহারকারী হন তাহলে [এই লিঙ্ক](#) এ গিয়ে Docker for Windows ডাউনলোড করে ইন্সটল করুন।

**Mac:** যদি আপনি Mac OS X El Capitan 10.11 অথবা আরো নতুন ভার্সন ব্যবহার করে থাকেন তাহলে এই লিঙ্ক এ গিয়ে Docker for Mac ডাউনলোড করে ইন্সটল করুন।

আগের ভার্সনের জন্য [এই লিঙ্ক](#) এ গিয়ে ডকার টুলবক্স ডাউনলোড করে ইন্সটল করুন।

ডকার Hello World রান করা

ইন্সটল করা ডকারটি রান করুন এবং `docker --version` কমান্ড দিয়ে ডকার এর ভার্সন চেক করুন:

```
Cyan Tarek TF@Engineer MINGW64 ~  
$ docker --version  
Docker version 18.02.0-ce, build fc4de447b5  
  
Cyan Tarek TF@Engineer MINGW64 ~  
$
```

এবার ডকারে **docker run hello-world** কমান্ড রান করুন। প্রথমবার ডকার তার রিপো থেকে **hello-world** ইমেজ ডাউনলোড করে রান করবে। সাকসেস হলে নিচের মতো মেসেজ দেখতে পাবেন:



Docker is configured to use the default machine with IP 192.168.99.100  
For help getting started, check out the docs at <https://docs.docker.com>

```
start interactive shell
```

Cyan Tarek TF@Engineer MINGW64 ~

```
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://cloud.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/engine/userguide/>

Cyan Tarek TF@Engineer MINGW64 ~

14

এবার আসুন ব্যাখ্যা করি কমান্ড এরঃ **docker run hello-world**

**docker** অংশের মাধ্যমে আমরা ডকার এর যাবতীয় কমান্ড রান করতে পারি

**run** দিয়ে আমরা কোন ডকার ইমেজ কে কন্টেইনার রূপে রান করতে পারি

**hello-world** হচ্ছে ইমেজ এর নাম যেটাকে আমরা ডকারে রান করবো।

ডকার ইমেজ কি তা আমরা একটু পরই জানবো। এখন এইটুকু জেনে রাখুন **docker run hello-world** কমান্ড এর সাহায্যে আমরা ডকার এর অফিসিয়াল **hello-world** ইমেজকে অনলাইন [ডকার হাব](#) ইমেজ রিপো থেকে ডাউনলোড করে, বিল্ড করে রান করেছি।

## ডকার এর মূল কনসেপ্ট

**ডকার ইমেজ:** এটিকে অফিসিয়ালি ডকার কন্টেইনার এর ম্যাপশট বলা হয়। ইমেজ কে বিল্ড করে রান করলে সেটি একটি ডকার কন্টেইনার এর ভেতরে রান হয়।

**ডকার ফাইল: Dockerfile** হলো ডকার ইমেজ তৈরি করার মূল নীল নকশা লেখা। এই ফাইলে লিখতে হয় একটা ডকার ইমেজে কি কি থাকবে।

**ডকার কন্টেইনার** কি তা তো আমরা আগেই জেনেছি। এবার চলুন আমাদের মূল কাজে যাই, ডকার এর ভেতরে পাইথন ওয়েব এপ্লিকেশন চালানো। আমি এর জন্য পাইথনের মাইক্রো ফ্রেমওয়ার্ক ফ্লাস্ক ব্যবহার করেছি।

## স্টেপ ১

প্রথমে একটা ফোল্ডার তৈরি করুন ডকার এ গিয়ে: **mkdir python\_flask** এরপর সেই ফোল্ডারে যান: **cd python\_flask**

## স্টেপ ২

এবার আমরা সিম্পল একটি হ্যালো ওয়ার্ল্ড ফ্লাস্ক এপ বানাবো। এর জন্য নতুন একটি ফোল্ডার বানান **mkdir app** নামে এবং সেখানে যান **cd app**। সেখানে **main.py** নামে একটি ফাইল বানান এবং নিচের কোডটি লিখুন:

```
#main.py
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
```



```
    return "Hey I'm running python flask app inside Docker!"if
__name__ == "__main__":
    app.run(host="0.0.0.0", debug=True, port=5000)
```

## স্টেপ ৩

এবার সেখানে **requirements.txt** নামে একটি ফাইল বানান এবং নিচের লেখাটি লিখুনঃ

```
Flask
```

## স্টেপ ৪

এরপর আমাদের কাজ হচ্ছে ডকার ইমেজ বানানো। আমরা চাইলে অনলাইনে থাকা আরেকজনের বানানো ইমেজ ব্যবহার করতে পারি অথবা আমরা নিজেসই একদম শুরু থেকে বানাতে পারি। আমি একদম শুরু থেকে স্টেপ বাই স্টেপ বানানো দেখাবো এতে করে আপনাদের শেখাও হয়ে যাবে।

ডকার ইমেজ বানানোর জন্য আমাদের কে **Dockerfile** বানাতে হবে, কারণ প্রতিটা ইমেজ তৈরি হয় ডকার ফাইল থেকে। তাই **python\_flask** এ ফোল্ডারে গিয়ে (যদি app ফোল্ডারে থেকে থাকেন তাহলে কমান্ডঃ **cd ..**) একটা ডকার ফাইল বানানঃ **ubuntu/mac** হলে **nano Dockerfile**, আর উইন্ডোজ হলে একটু ট্রিকি। কারণ Dockerfile এর কোন এক্সটেনশন নেই। উইন্ডোজ ইউজাররা মাই কম্পিউটার থেকে আগের তৈরি করা **python\_flask** ফোল্ডারে যান, এটা সাধারণত ডিফল্টভাবে **C:/Users/<UserName>/** এ পাবেন।

অর্থাৎ **C:/Users/<UserName>/python\_flask/** এবার সেখানে নোটপ্যাড দিয়ে এক্সটেনশন বিহীন **Dockerfile** নামে একটা ফাইল বানান। যদি না জেনে থাকেন কিভাবে উইন্ডোজে এক্সটেনশন ছাড়া ফাইল বানায় তাহলে [এই লিঙ্কে](#) গিয়ে দেখে আসতে পারেন।

এবার নিচের কোডগুলি **Dockerfile** এ লিখে সেভ করুনঃ

```
FROM python:3.6.4-slim
COPY ./app /app
RUN pip install -r /app/requirements.txt
```

```
CMD [ "python", "/app/main.py" ]  
EXPOSE 5000
```

নোটঃ যদি আপনি ম্যাক বা লিনাক্স ব্যবহারকারী হন তাহলে উপরের **CMD** [ "python", "/app/main.py" ] এর python এর বদলে python3 লিখবেন।

## ব্যখ্যা

অনেকটা এসেসবলি ল্যাগুয়েজ প্রোগ্রামিং এর মত স্টেপ বাই স্টেপ প্রসিডিউর।

প্রথম লাইনের সাহায্যে আমরা আমাদের কন্টেইনার এর একটা মূল ইমেজ নিয়েছি, যেটি একটি পাইথন ইমেজ (সাথে লাইটওয়েট লিনাক্স ডিস্ট্রো আছে)। ডকার এটি তৈরি করার পরে আমাদের বানানো app ফোল্ডারটিকে কন্টেইনারে কপি করবে। এরপর আমাদের বানানো requirements.txt ফাইলকে পাইথন pip দিয়ে রান করার মাধ্যমে আমরা কন্টেইনারে ফ্লাস্ক ইন্সটল করলাম। ব্যাস এবার আমরা পাইথন কমান্ড লাইনের মাধ্যমে main.py ফাইলটিকে রান করলাম। আমাদের ফ্লাস্ক এপ্লিকেশন কন্টেইনারে রান হয়ে গেছে। কিন্তু এটাকে বাহিরে থেকে কিভাবে দেখবো?

এর জন্য প্রথমেই আমাদেরকে ফ্লাস্ক যে পোর্টে রান হয়েছে সেটিকে কন্টেইনারে এক্সপোজ করতে হবে। তাই আমরা পোর্ট ৫০০০ কে এক্সপোজ করেছি।

## স্টেপ ৫

আমরা **Dockerfile** বানিয়ে ফেললাম। এবার আমাদেরকে এই ফাইল থেকে ইমেজ বানাতে হবে। এর জন্য ডকার এ গিয়ে এই কমান্ডটি রান করি (অবশ্যই ডকার কমান্ড লাইনে যেনো **python\_flask** ফোল্ডারে থাকে কারণ এখানেই আমরা Dockerfile বানিয়েছিলাম)

## docker build -t myapp .

শেষের . এর মাধ্যমে আমরা বুঝিয়েছি বর্তমান ফোল্ডারে থাকা ডকারফাইলকে যেনো বিল্ড করে। **myapp** দিয়ে ইমেজ এর একটি নাম দিয়েছি যা পরে কাজে লাগবে। এরপর ডকার ফাইলের প্রতিটা স্টেপ রান হতে থাকবে। বিভিন্ন ফাইল ডাউনলোড হবে। নিচে একটি স্ক্রিনশটঃ

```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker build -t myapp .
Sending build context to Docker daemon 4.608kB
Step 1/5 : FROM python:3.6.4-slim
--> 43431c5410f3
Step 2/5 : COPY ./app /app
--> Using cache
--> a28480813766
Step 3/5 : RUN pip install -r /app/requirements.txt
--> Running in 3d0a495bf7b3
Collecting Flask (from -r /app/requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting Werkzeug>=0.7 (from Flask->-r /app/requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting Jinja2>=2.4 (from Flask->-r /app/requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting itsdangerous>=0.21 (from Flask->-r /app/requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting click>=2.0 (from Flask->-r /app/requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask->-r /app/requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
    Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
    Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c75e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
    Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
    Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: Werkzeug, MarkupSafe, Jinja2, itsdangerous, click, Flask
Successfully installed Flask-0.12.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
Removing intermediate container 3d0a495bf7b3
--> 424a2f3eae0e
Step 4/5 : CMD [ "python", "/app/main.py" ]
--> Running in 51efee4df131
Removing intermediate container 51efee4df131
--> 91f72aaaa8d5
Step 5/5 : EXPOSE 5000
--> Running in b99c78bc6290
Removing intermediate container b99c78bc6290
--> a3ffe5866425
Successfully built a3ffe5866425
Successfully tagged myapp:latest
SECURITY WARNING: You are building a Docker image from windows against a non-windows Docker host. All files and directories added to build context will
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ notepad Dockerfile
```

## স্টেপ ৬

আমরা ইমেজ বিল্ড করেছি। এবার আমরা সেই ইমেজটিকে কন্টেইনারে রান করবো ডকারে এই কমান্ড দিয়েঃ

**docker run -d -p 80:5000 --name hello-flask myapp**

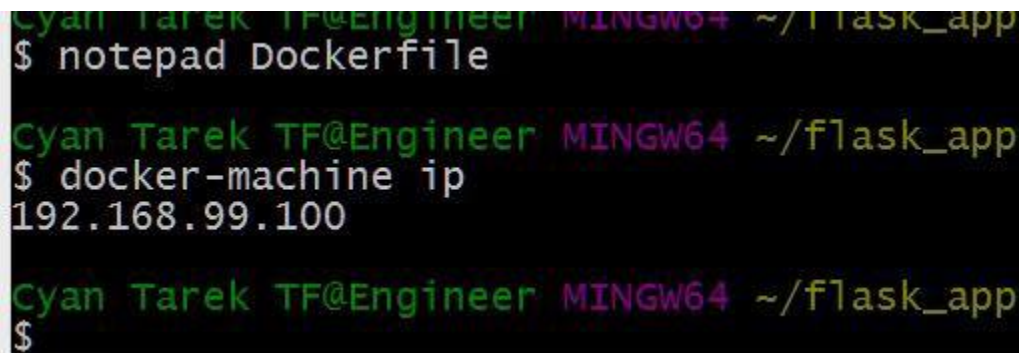
এর মাধ্যমে ডকার **myapp** ইমেজ কে রান করবে **hello-flask** কন্টেইনারে। এবং আমরা যেহেতু আমাদের কন্টেইনারের ভেতরে পোর্ট ৫০০০ এ ফ্লাস্ক রান করেছি এবং ৫০০০ পোর্টকে এক্সপোজ করেছি তাই এবার আমরা এই কন্টেইনারের ৫০০০ পোর্টকে আমাদের মূল হোস্ট ওএস এর পোর্ট ৮০ এর সাথে ম্যাপ বা ফরওয়ার্ড করে দিলাম। এর ফলে এবার আমরা আমাদের ব্রাউজারে নিচের লিঙ্কে গিয়ে ডকারে রান হওয়া আমাদের ফ্লাস্ক এপ কে এক্সেস করতে পারবো:

যদি **Docker for Windows/Mac** চালান তাহলে ব্রাউজারে গিয়ে ইউনিভার্সাল আইপি: <http://0.0.0.0:80> অথবা <http://0.0.0.0> রান করুন

যদি **Docker Toolbox Windows/Mac** চালান তাহলে ব্রাউজারে গিয়ে ডকার মেশিন আইপি দিয়ে দেখতে পারেন। ডকার মেশিন আইপি দেখার জন্য ডকারে গিয়ে এই কমান্ড রান করুন:

**docker-machine ip**

তাহলে আপনার ডকার মেশিন এর আইপিটা দেখাবে:



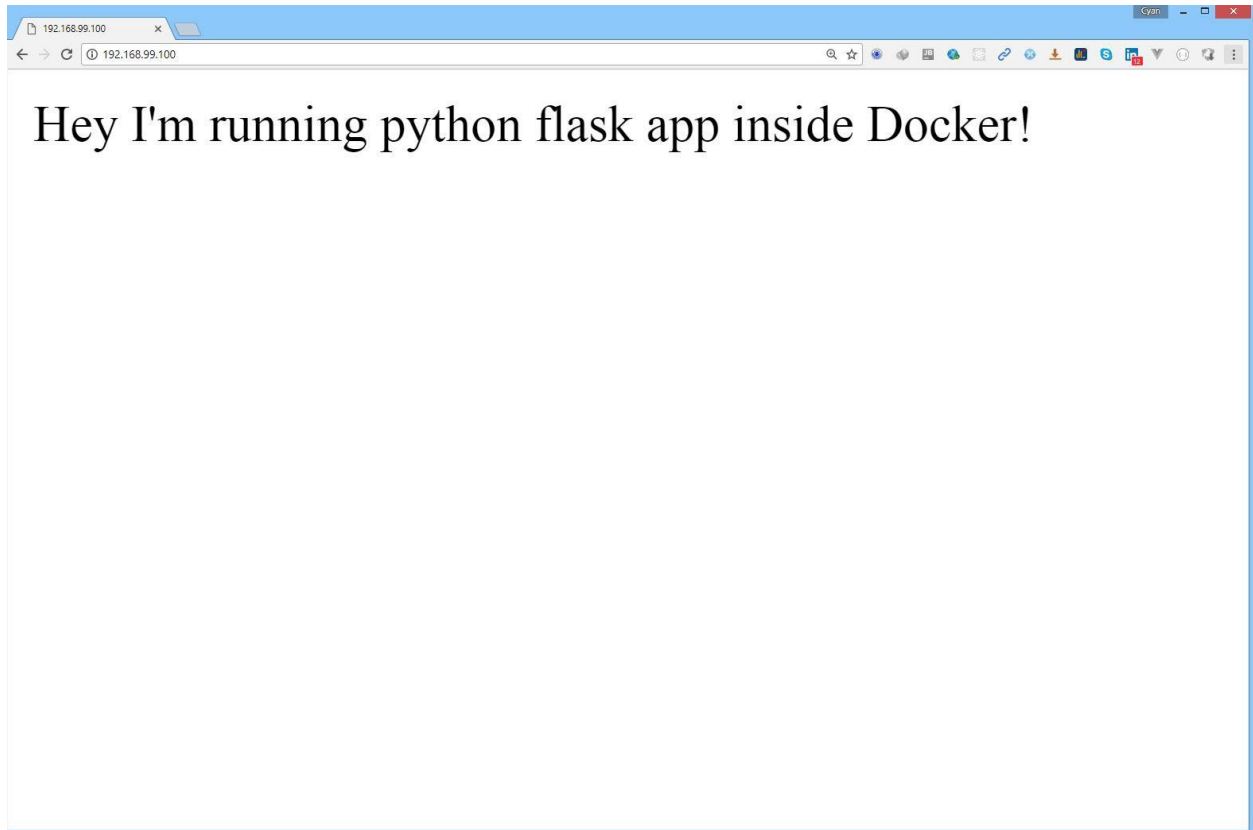
```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ notepad Dockerfile

Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker-machine ip
192.168.99.100

Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$
```

এবার ব্রাউজারে

গিয়ে [http://your\\_docker\\_machine\\_ip:80](http://your_docker_machine_ip:80) অথবা [http://your\\_docker\\_machine\\_ip](http://your_docker_machine_ip) দিলে নিচের মতো দেখতে পাবেনঃ



কনগ্রেটস! আপনি সফলভাবে ডকার এর ভিতরে পাইথন ফ্লাস্ক ওয়েব এপ্লিকেশন চালিয়েছেন। এবার আপনি যেকোন এপ্লিকেশনকে ডকারে চালাতে পারেন নিজে ডকার ইমেজ বানিয়ে বা অন্যের বানানোটা ব্যবহার করে। পরবর্তীতে আমি ডকার জ্যাঙ্গে দিয়ে একটি টিউটোরিয়াল দেখাবো এবং তারপর ডকার এর সাহায্যে ইঞ্জিনেক্স, পোস্টগ্রেস ডাটাবেস, মাইক্রোসার্ভিস, ক্লাউড হোস্ট (যেমনঃ AWS, Linode, DigitalOcean ইত্যাদি) এ ডিপ্লয় এবং রান করা এসব কিছুই দেখাবো এক এক করে ইন শা আল্লাহ।

এর মাঝে ডকার নিয়ে পড়াশোনা করতে থাকুন। নিচে কিছু রিসোর্স এর নাম দিলাম আমার পার্সোনাল ফেভারিটঃ

**বইঃ**

*01. Oreilly Docker Up and Running*

*02. Manning Docker in Action*

**ভিডিও কোর্সঃ**

*01. Pluralsight — Docker for Web Developers*

*02. Udemy The Complete Docker Course for DevOps and Developers*

*03. O'Reilly — Learning Docker*

[Ishtiaque](#) ভাই এর ডকার নিয়ে লেখা দারুন এই আর্টিকেলটিও পড়ে ফেলুন

বাটপটঃ <https://goo.gl/bM8XyM>

আল্লাহ হাফেজ

# ডকার এ ডাটাবেজ সার্ভার সেটাপ করা এবং ওয়েব এপ্লিকেশনের সাথে চালানো

সফটওয়্যার এর অন্যতম মূল জিনিস হলো ডাটা। আর এই ডাটাকে আমরা জমা রাখি ডাটাবেজ এ।

ডকার সিরিজের আগের পোস্টে আমি দেখিয়েছি কিভাবে করে আপনি ডকারে পাইথন ওয়েব এপ্লিকেশন চালাতে পারেন। এবার দেখাবো কিভাবে করে ডকার কন্টেন্ট ইনারে ডাটাবেজ সার্ভার সেটাপ করতে হয় ও তা ব্যবহার করতে হয়।

আমি এখানে মূলত PostgreSQL ডাটাবেজ সিস্টেম দেখিয়েছি। একই রকমভাবে আপনি যেকোন ডাটাবেজ সিস্টেম সেটাপ করতে পারবেন যদি সম্পূর্ণতা বুঝে থাকেন।

তো চলুন শুরু করা যাক

## পোস্টগ্রেস সেটাপ

আমার আগের পোস্ট দেখে থাকলে এবং প্র্যাকটিস করে থাকলে আশা করি মনে থাকার কথা ডকার এর সেটপগুলি।

প্রথম সেটপ ছিল ডকার ফাইল রচনা করা। তবে আমি এবার নতুন করে নিজের মতো ডকার ফাইল বানানো দেখাবো না। ওটা দেখিয়েছিলাম আপনাদের বোঝার সুবিধার জন্য। এবার দেখাবো কিভাবে অনলাইন ডকার রিপোজিটোরি তে থাকা আরেকজনের বানানো ডকারফাইল ব্যবহার করতে হয়। এতে করে আমাদের সময়ও বাঁচলো।

প্রথমে ডকার এ যান এবং নিম্নোক্ত কমান্ডটি রান করুনঃ

```
docker run --name postgres_container -d -p 5432:5432 postgres
```

এর ফলে ডকার তার রিপোজিটোরি থেকে অফিসিয়াল **postgres** ডাটাবেজ ইমেজটি ডাউনলোড করে একটি কন্টেইনারে রান করবে। ডিফল্টভাবে অফিসিয়াল পোস্টগ্রেস

ইমেজটি **postgres\_container** নামক কন্টেইনারে 5432 পোর্টে রান হবে, যেটার নাম আমরা দিয়েছি। আর 5432 হলো পোস্টগ্রেস এর ডিফল্ট পোর্ট এবং ডকার কন্টেইনারেও এই পোর্টটি এক্সপোজ করা আছে। নিচে আমি উদাহরণ স্বরূপ ডকার এর অফিসিয়াল পোস্টগ্রেস ইমেজ এর ডেফিনিশন দেখালামঃ

```
108     rm -rf /var/lib/apt/lists/*; \
109     \
110     if [ -n "$tempDir" ]; then \
111     # if we have leftovers from building, let's purge them (including extra, unnecessary build deps)
112         apt-get purge -y --auto-remove; \
113         rm -rf "$tempDir" /etc/apt/sources.list.d/temp.list; \
114     fi
115
116 # make the sample config easier to munge (and "correct by default")
117 RUN mv -v "/usr/share/postgresql/$PG_MAJOR/postgresql.conf.sample" /usr/share/postgresql/ \
118     && ln -sv ../postgresql.conf.sample "/usr/share/postgresql/$PG_MAJOR/" \
119     && sed -ri "s!^#?(listen_addresses)\s*=\s*\S+.*!1 = '*'!" /usr/share/postgresql/postgresql.conf.sample
120
121 RUN mkdir -p /var/run/postgresql && chown -R postgres:postgres /var/run/postgresql && chmod 2777 /var/run/postgresql
122
123 ENV PATH $PATH:/usr/lib/postgresql/$PG_MAJOR/bin
124 ENV PGDATA /var/lib/postgresql/data
125 RUN mkdir -p "$PGDATA" && chown -R postgres:postgres "$PGDATA" && chmod 777 "$PGDATA" # this 777 will be replaced by 7
126 VOLUME /var/lib/postgresql/data
127
128 COPY docker-entrypoint.sh /usr/local/bin/
129 RUN ln -s usr/local/bin/docker-entrypoint.sh / # backwards compat
130 ENTRYPOINT ["docker-entrypoint.sh"]
131
132 EXPOSE 5432
133 CMD ["postgres"]
```

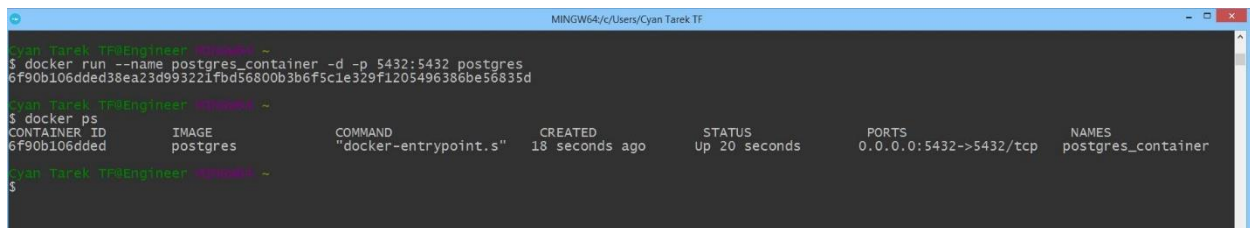
আমরা যেহেতু পোস্টগ্রেসকে আমাদের হোস্ট কম্পিউটার থেকে ব্যবহার করবো তাই কন্টেইনার এর পোর্ট 5432 এর সাথে হোস্ট এর পোর্ট 5432 ম্যাপ/ফরওয়ার্ড করে দিলাম।



উপরের কমান্ডটি ডকার এ প্রথমবার রান করলে ডকার দরকারী সবকিছু স্টেপ বাই স্টেপ ডাউনলোড করে ইমেজটিকে বিন্ড করবে এরপর একটি কন্টেইনারে রান করবে।

সবকিছু সাকসেসফুল হলে আমরা হ্যাশ কোড পাবো, যা এই কন্টেইনার এর লং আইডি।

সত্যিই কি রান হয়েছে কিনা চেক করার জন্য **docker ps** কমান্ডটি দেই। এই কমান্ড এর মাধ্যমে আমরা দেখতে পারি ডকারে কোন কোন কন্টেইনার রান করা আছেঃ



```
Cyan Tarek TPEngineer@localhost ~$ docker run --name postgres_container -d -p 5432:5432 postgres
6f90b106dded38ea23d993221fbd56800b3b6f5c1e329f1205496386be56835d

Cyan Tarek TPEngineer@localhost ~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f90b106dded        postgres           "docker-entrypoint.s" 18 seconds ago      Up 20 seconds      0.0.0.0:5432->5432/tcp   postgres_container

Cyan Tarek TPEngineer@localhost ~$
```

তাহলে আমরা দেখতে পাচ্ছি যে ডকারে পোস্টগ্রেস রান হয়েছে। এবার আমরা একে এক্সেস করবো ২ ভাবে

ডকার থেকে পোস্টগ্রেস এক্সেস করাঃ

কমান্ডঃ

```
docker exec -it postgres_container bash
```

এরফলে আমরা ডকারে রান হয়ে থাকা **postgres\_container** এর টার্মিনাল এ প্রবেশ করলাম।

এবার আমরা পোস্টগ্রেস সার্ভারে লগইন করবো। ডিফল্টভাবে পোস্টগ্রেস এর ইউজারনেম

postgres এবং পাসওয়ার্ড খালি থাকে। তাই এই ক্রিডেনশিয়াল দিয়ে লগইন করি আমরাঃ

```
psql -U postgres
```

```
Cyan Tarek TP@Engineer ~$ docker run --name postgres_container -d -p 5432:5432 postgres
6f90b106dded38ea23d993221fbd56800b3b6f5c1e329f1205496386be56835d

Cyan Tarek TP@Engineer ~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f90b106dded        postgres           "docker-entrypoint.s"  18 seconds ago     Up 20 seconds      0.0.0.0:5432->5432/tcp   postgres_container

Cyan Tarek TP@Engineer ~$ docker exec -it postgres_container bash
root@6f90b106dded:/# psql -U postgres
psql (10.3 (Debian 10.3-1.pgdg90+1))
Type "help" for help.

postgres=#
```

ওয়াও! আমরা ডকারে থাকা পোস্টগ্রেসকে এক্সেস করলাম। এবার আমরা চাইলে ডাটাবেজ, টেবিল বানাতে, লিস্ট করতে সবকিছুই করতে পারি।

```
Cyan Tarek TP@Engineer ~$ docker run --name postgres_container -d -p 5432:5432 postgres
6f90b106dded38ea23d993221fbd56800b3b6f5c1e329f1205496386be56835d

Cyan Tarek TP@Engineer ~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
6f90b106dded        postgres           "docker-entrypoint.s"  18 seconds ago     Up 20 seconds      0.0.0.0:5432->5432/tcp   postgres_container

Cyan Tarek TP@Engineer ~$ docker exec -it postgres_container bash
root@6f90b106dded:/# psql -U postgres
psql (10.3 (Debian 10.3-1.pgdg90+1))
Type "help" for help.

postgres=# \l
      name | owner  | encoding | list of databases | collate | ctype | access privileges
-----+-----+-----+-----+-----+-----+-----
 postgres | postgres | UTF8     | en_US.utf8         | en_US.utf8 | en_US.utf8 | =c/postgres
 template0 | postgres | UTF8     | en_US.utf8         | en_US.utf8 | en_US.utf8 | postgres=CTC/postgres
 template1 | postgres | UTF8     | en_US.utf8         | en_US.utf8 | en_US.utf8 | =c/postgres
 postgres=CTC/postgres
(3 rows)

postgres=#
```

পোস্টগ্রেস থেকে পুনরায় কন্টেইনার টার্মিনালে যেতে \q কমান্ড দিন এবং কন্টেইনার টার্মিনাল থেকে ডকার টার্মিনালে যেতে exit লিখুন।

আপনার কম্পিউটার হোস্ট থেকে ডকারের পোস্টগ্রেস এক্সেস করা

এবার আমরা কন্টেইনার এর বাহিরে থেকে ডকারের পোস্টগ্রেস এক্সেস করবো। এরজন্য উইন্ডোজ, ম্যাক বা লিনাক্স এর জন্য পোস্টগ্রেস ডাউনলোড করে ইন্সটল করে নিন। উইন্ডোজ ইউজাররা অবশ্যই postgres এর বাইনারি ফোল্ডারটি ইনভায়রনমেন্ট ভেরিয়েবল হিসেবে সেট করে নিবেন, নয়তো CMD থেকে এক্সেস করতে পারবেন না।

**ডাউনলোড এর জন্য লিঙ্কঃ**

উইন্ডোজ – <https://www.postgresql.org/download/windows/>

ম্যাক – <https://www.postgresql.org/download/macosx/>

লিনাক্স – `sudo apt-get install postgresql-9.6`

উইন্ডোজে কিভাবে করে ইনভায়রনমেন্ট ভেরিয়েবল সেট করতে

হয়ঃ <https://www.youtube.com/watch?v=bEroNNzqlF4>

এবার টার্মিনাল এ গিয়ে নিচের কমান্ড লিখুনঃ

Docker Toolboxব্যবহার করলে:

```
psql -h docker_machine_ip -p 5432 -U postgres
```

docker\_machine\_ip এর জায়গায় আপনার ডকার মেশিন এর আইপি বসান। কিভাবে ডকার মেশিন এর আইপি বের করতে হয় জানতে ডকার নিয়ে আমার **আগের পোস্টটি** দেখুন।

Docker for Mac/windows হলেঃ

```
psql -h 0.0.0.0 -p 5432 -U postgres
```

ওয়াও! আমরা সফলভাবে ডকারে চলতে থাকা পোস্টগ্রেসকে রিমোটভাবে আমাদের হোস্ট কম্পিউটার থেকে রান করতে পেরেছি।

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Cyan Tarek TF>psql -h 192.168.99.100 -p 5432 -U postgres
psql (9.6.4, server 10.3 (Debian 10.3-1.pgdg90+1))
WARNING: psql major version 9.6, server major version 10.
Some psql features might not work.
WARNING: Console code page (437) differs from windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

আসুন কিছু ডাটা নিয়ে কাজ করি।

পোস্টগ্রেসে থাকা অবস্থায় নিচের কমান্ডগুলি রান করুনঃ

01. CREATE DATABASE test\_db;
02. \c test\_db
03. CREATE TABLE company(id SERIAL, name char(50));
04. INSERT INTO company(name) VALUES('Big Idea Software');
05. \q

এখানে আমরা প্রথমে test\_db নামে একটি ডাটাবেজ বানিয়ে সেখানে company নামে একটি টেবিল বানিয়ে একটি রেকর্ড ইনসার্ট করেছি।

**বোনাসঃ ডকারে থাকা পাইথন ওয়েব এপ আর পোস্টগ্রেস ডাটাবেজ একসাথে কাজ করা**

এবার আমি দেখাবো কিভাবে করে ডকারে রান হয়ে থাকা আমাদের আগের পাইথন ফ্লাস্ক এপ্লিকেশন থেকে আমরা এই পোস্টগ্রেসকে ব্যবহার করতে পারি। খুবই মজার টপিক।

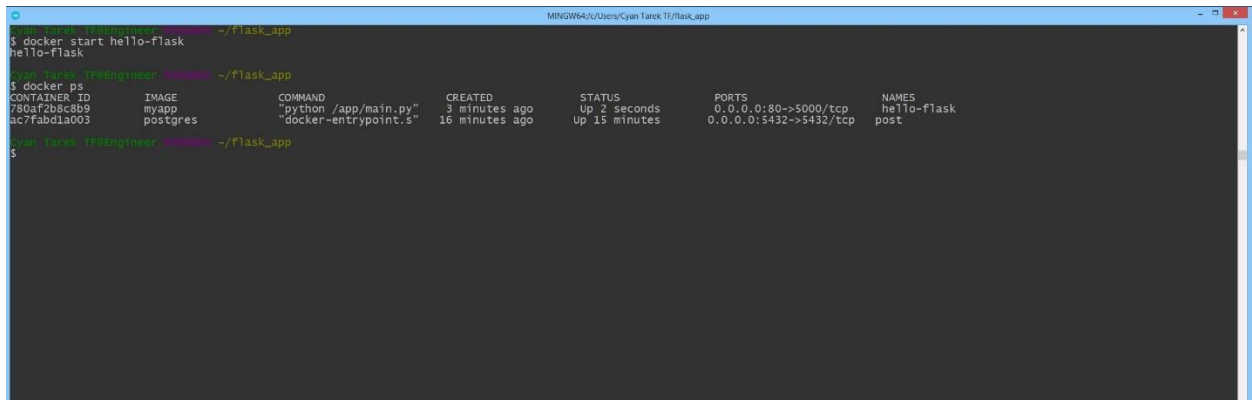
প্রথমেই আমাদের আগের বানানো ফ্লাস্ক এপটা রান করি কন্টেইনারে। এরজন্য আমরা নিচের কমান্ড ব্যবহার করবোঃ

```
docker start hello-flask
```

এখানে hello-flask হচ্ছে আমাদের আগের বানানো ফ্লাস্ক এপ এর কন্টেইনার এর নাম।

**docker ps** কমান্ড দিয়ে চেক করে দেখুন রান হয়েছে কিনা।

আশা করি আপনি একই সাথে দুটি কন্টেইনার রানিং পাবেন, একটি ফ্লাস্ক এপ এর, আর আরেকটি পোস্টগ্রেস ডাটাবেজ এর। দুইটাই কন্টেইনারে রান করা আছে। এবার আমরা দুইটাকে একত্রে ব্যবহার করবো।



```
$ docker start hello-flask
hello-flask
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
780af2b8c8b9   myapp    "python /app/main.py"    3 minutes ago Up 2 seconds  0.0.0.0:80->5000/tcp             hello-flask
ac7fabd1a003   postgres "docker-entrypoint.sh"   16 minutes ago Up 15 minutes  0.0.0.0:5432->5432/tcp           postgres
```

প্রথমেই আপনার নেটওয়ার্কিং সম্পর্কে মোটামুটি ধারণা থাকতে হবে। উপরের চিত্রে দেখবেন যে hello-flask এর PORTS: 0.0.0.0:80->5000/tcp দেয়া আর postgres এর PORTS: 0.0.0.0:5432->5432/tcp দেয়া। এর মানে হলো এগুলি প্রতিটাই তাদের কন্টেইনারে ইউনিভার্সাল আইপি 0.0.0.0 তে রান হয়ে আছে।

তাই আমরা এক কন্টেইনার থেকে আরেক কন্টেইনারকে container\_ip:port দিয়ে এক্সেস করতে পারি।

ডকার এর প্রতিটা কন্টেইনার এর একটা নিজস্ব প্রাইভেট আইপি এড্রেস আছে যার মাধ্যমে ডকার কন্টেইনারগুলি একটা আরেকটার সাথে যোগাযোগ করে। কোন ডকার কন্টেইনার এর আইপি বের করতে পারি আমরা এভাবেঃ

`docker inspect --format '{{ .NetworkSettings.IPAddress }}' container_hash`

এখানে **container\_hash** হলো ডকার কন্টেইনার এর ইউনিক আইডি যেটা আমরা **docker ps** বা **docker ps -a** দিয়ে বের করতে পারি। এখানে দেখুন **postgres** এর কন্টেইনার এর হ্যাশ কোড দেয়া আছে প্রথমেই।

```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker start hello-flask
hello-flask
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                    NAMES
780af2b8c8b9   myapp     "python /app/main.py"   3 minutes ago Up 2 seconds  0.0.0.0:80->5000/tcp     hello-flask
ac7fabd1a003   postgres  "docker-entrypoint.s"   16 minutes ago Up 15 minutes  0.0.0.0:5432->5432/tcp   post
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$
```

আমাদের যেহেতু **postgres\_container** এর আইপি দরকার তাই আমরা এভাবে রান করিঃ

`docker inspect --format '{{ .NetworkSettings.IPAddress }}' ac7fabd1a003`

তাহলে আমরা নিচের মত আমাদের কন্টেইনার এর আইপি পেয়ে যাবো।

```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' ac7fabd1a003
172.17.0.2
```

আমরা এখন ফ্লাস্ক এপ থেকে পোস্টগ্রেস ব্যবহার করবো। এর জন্য প্রথমে আপনার কম্পিউটারে থাকা `main.py` ফাইলটি মডিফাই করতে হবে। প্রথমে রান হয়ে থাকা `hello-flask` কন্টেইনারটি বন্ধ করি, নাহলে পোর্ট কনফ্লিক্ট হবেঃ

```
docker stop hello-flask
```

এরপর আমাদের কম্পিউটারে আমরা আগের টিউটোরিয়ালে যেখানে **main.py** ফাইলটি বানিয়েছিলাম সেই ফোল্ডারে গিয়ে **main.py** ফাইলটি ওপেন করুন (প্রয়োজনে আগের পোস্টটি আবার দেখে নিন)

```
#main.py
from flask import Flask
import psycopg2

connection = psycopg2.connect(dbname='test_db', user="postgres",
host="172.17.0.2", port=5432)
cur = connection.cursor()

cur.execute("SELECT * FROM company")
data = cur.fetchone()

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hey I'm %s and I'm running python flask app inside
    Docker!" % data[1]

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True, port=5000)
```

এখানে আমরা আমাদের একটু আগে পোস্টগ্রেসে তৈরিকৃত ডাটাবেজ কে পাইথন থেকে কানেক্ট করলাম এবং ক্যুয়েরি করে আমার রেকর্ডকে ডাটাবেজ থেকে রিড করে ফ্লাস্ক এপ এ শো করলাম।

এরপর একই ফোল্ডারে থাকা **requirements.txt** ফাইলকে এডিট করে নিচের মত করে দিনঃ

```
Flask
psycopg2
```

এবার ডকার এ গিয়ে এই কমান্ডটি রান করি (অবশ্যই ডকার কমান্ড লাইনে

যেনো **python\_flask** ফোল্ডারে থাকে কারণ এখানেই আমরা **Dockerfile** বানিয়েছিলাম)  
`docker build -t myapp .`

এরপর ডকার ফাইলের প্রতিটা স্টেপ রান হতে থাকবে। বিভিন্ন ফাইল ডাউনলোড হবে। নিচে একটি স্ক্রিনশটঃ

```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker build -t myapp .
Sending build context to Docker daemon 4.608kB
Step 1/5 : FROM python:3.6.4-slim
--> 43431c5410f3
Step 2/5 : COPY ./app /app
--> Using cache
--> a28480813766
Step 3/5 : RUN pip install -r /app/requirements.txt
--> Running in 3d0a495bf7b3
Collecting Flask (from -r /app/requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting Werkzeug>=0.7 (from Flask->-r /app/requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting Jinja2>=2.4 (from Flask->-r /app/requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting itsdangerous>=0.21 (from Flask->-r /app/requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting click>=2.0 (from Flask->-r /app/requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask->-r /app/requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c75e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcb25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: Werkzeug, MarkupSafe, Jinja2, itsdangerous, click, Flask
Successfully installed Flask-0.12.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
Removing intermediate container 3d0a495bf7b3
--> 424a2f3eae0e
Step 4/5 : CMD [ "python", "/app/main.py" ]
--> Running in 51efee4df131
Removing intermediate container 51efee4df131
--> 91f72aaaa8d5
Step 5/5 : EXPOSE 5000
--> Running in b99c78bc6290
Removing intermediate container b99c78bc6290
--> a3ffe5866425
Successfully built a3ffe5866425
Successfully tagged myapp:latest
SECURITY WARNING: You are building a Docker image from windows against a non-windows Docker host. All files and directories added to build context will
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ notepad Dockerfile
```

আমরা ইমেজ বিন্ড করেছি। এবার আমরা সেই ইমেজটিকে কন্টেইনারে রান করবো ডকারে এই কমান্ড দিয়েঃ

**docker run -d -p 80:5000 — name flask-postgres myapp**

এর মাধ্যমে ডকার **myapp** ইমেজ কে রান করবে **flask-postgres** কন্টেইনারে।

যদি **Docker for Windows/Mac** চালান তাহলে ব্রাউজারে গিয়ে ইউনিভার্সাল

আইপিঃ **http://0.0.0.0:80** অথবা **http://0.0.0.0** রান করুন

যদি **Docker Toolbox Windows/Mac** চালান তাহলে ব্রাউজারে গিয়ে ডকার মেশিন আইপি

দিয়ে দেখতে পারেন। ডকার মেশিন আইপি দেখার জন্য ডকারে গিয়ে এই কমান্ড রান করুনঃ



```
docker-machine ip
```

তাহলে আপনার ডকার মেশিন এর আইপিটা দেখাবেঃ

```
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ notepad Dockerfile

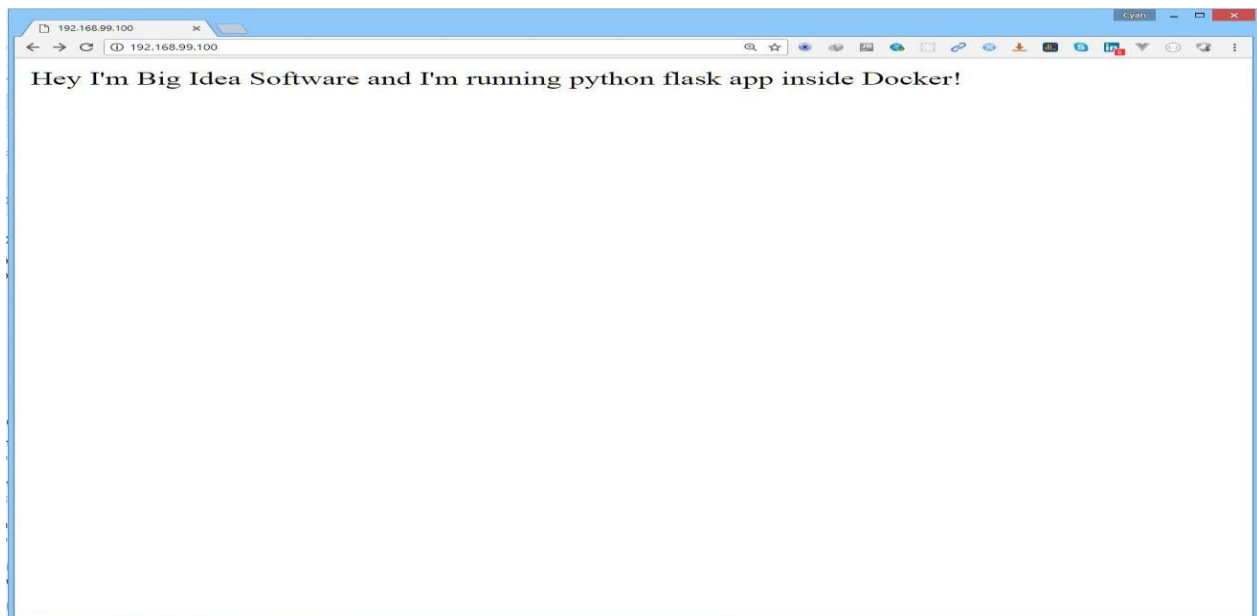
Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$ docker-machine ip
192.168.99.100

Cyan Tarek TF@Engineer MINGW64 ~/flask_app
$
```

এবার ব্রাউজারে

গিয়ে **http://your\_docker\_machine\_ip:80** অথবা **http://your\_docker\_machine\_ip**

আশা করি এরপর নিচের মত এই রেজাল্ট পাবেনঃ



কংগ্রেচুলেশন! আপনি যদি এই পর্যন্ত এসে থাকেন তাহলে আপনি অনেক কিছু শিখে গেছেন।

দেখুন এখানে ডাটাবেজ থেকে ডাটা শো করতেছে।

তার মানে আমরা শিখলামঃ

০১. কিভাবে করে ডকারে ডাটাবেজ সার্ভার সেটাপ করতে হয়

০২. কিভাবে করে ডকারে রান হওয়া ডাটাবেজ সার্ভারকে আমরা ডকার থেকে এবং রিমোট হোস্ট থেকে এক্সেস করতে পারি

০৩. এরপর দেখলাম কিভাবে করে ডকার এর ডাটাবেজ সার্ভারে ডাটাবেজ তৈরি, ডাটা ইনসার্ট ইত্যাদি করতে পারি

০৪. এরপর দেখলাম কিভাবে করে ডকারে একই সাথে রান হওয়া একটি ওয়েব এপ্লিকেশন ও একটি ডাটাবেজ সার্ভারের মাঝে কানেকশন তৈরি করে ডাটা রিড করতে পারি

তাহলে, আজকেই পোস্টে মোটামুটি অনেক কিছু শেয়ার করলাম। আশা করি আপনাদের কাজে লাগবে।

## ডকার নেটওয়ার্কিং

আস সালামু আলাইকুম। কেমন আছেন সবাই? আশা করি ভালো আছেন।

ডকার সিরিজ এর আজকের এই পর্বে আমি আলোচনা করবো ডকার এর অত্যন্ত গুরুত্বপূর্ণ একটি কনসেপ্টঃ ডকার নেটওয়ার্কিং নিয়ে।

পূর্ব শর্তঃ ভার্চুয়ালাইজেশন, কন্টেইনারাইজেশন, ডকার বেসিক এবং নেটওয়ার্কিং বেসিক

আগে চলুন নেটওয়ার্কিং নিয়ে জেনারেল কিছু কনসেপ্ট আলোচনা করিঃ

নেটওয়ার্কিং বেসিকঃ আইপি এড্রেসিং

ইন্টারনেট প্রোটোকল(IP) এড্রেস হলো একটা নেটওয়ার্কিং নোড এর ইউনিক আইডেন্টিফায়ার। নেটওয়ার্কিং নোড হতে পারে একটা ডেস্কটপ পিসি, ল্যাপটপ, মোবাইল, অথবা (আমাদের ক্ষেত্রে) একটা ডকার কন্টেইনার।

একটা নেটওয়ার্ক এড্রেস সাধারণত ৩২ বিট এর হয়, যেমন?

[www.cmyip.com](http://www.cmyip.com) এ গিয়ে আপনার আইপি টি দেখুন।

ধরুন আপনার আইপিঃ 141.112.125.118

এখানে ৩টি করে সংখ্যার মোট ৪ টা ঘর আছে। প্রতি ঘরে ০ থেকে ২৫৫ পর্যন্ত মোট ২৫৬ ধরনের সংখ্যা থাকতে পারে। এই ৪ টা ঘরকে ৪ টা ক্লাস বলেঃ A, B, C, D

কম্পিউটার দশমিক বোঝে না, বাইনারি বোঝে। তাই প্রতিটা ঘরকে বাইনারি তে কনভার্ট করলে সেগুলি,  $2^3 = ৮$  বিট এর হয়। কিভাবে?

৩ টা সংখ্যা, মোট ১ বাইট এর। ১ বাইট = ৮ বিট। আর ১ বিট এ ০ বা ১ এই দুইটা থাকতে পারে তাই মোট কম্বিনেশন  $2^3 = ৮$  টি। তাহলে ৮ বিট x ৪ ঘর = মোট ৩২ বিট।

কঠিন লাগতেছে?

৩২ বিট এ মোট কয়টি এড্রেস পাচ্ছি তাহলে?

এর জন্য ৩২ বিটকে কম্বিনেশন করি।

এক বিট এ যেহেতু ০ আর ১ এই ২টি সংখ্যা থাকে, তাই ৩২ বিটে থাকে  $2^{32} = 4294967296$  টি

দশমিক দিয়েও বের করি: প্রতি ঘরে ২৫৬ সংখ্যা, তাহলে ৪ ঘরের মোট এড্রেসঃ  $256^4 = 4294967296$  টি

তাহলে একই সংখ্যক এড্রেসই পাচ্ছি বাইনারি বা ডেসিমাল যেভাবেই ক্যালকুলেট করি না কেন।

মোট প্রায় 4.3 বিলিয়ন আইপি এড্রেস পাচ্ছি আমরা। এটা তো গেলো আইপি ভার্সন ৪ এর, এছাড়াও ভার্সন ৬ এর ও আলাদা হিসাব। যাই হোক আমরা ভার্সন ৪ নিয়েই আলোচনা করি, কারণ এটাই সবচেয়ে প্রচলিত এবং ডকারেও এটা ব্যবহার করে।

কিন্তু পৃথিবীতে ৭.৬ বিলিয়ন মানুষ আছে, আবার এদের অনেকগুলি ডিভাইস আছে। তাই এতগুলি এড্রেসকে ইন্টারনেট এ জায়গা দেওয়ার জন্য সম্পূর্ণ নেটওয়ার্ককে দুই ভাগে ভাগ করা হয়েছেঃ

০১. পাবলিক নেটওয়ার্ক

০২. প্রাইভেট নেটওয়ার্ক

পাবলিক নেট এর নোডগুলি গ্লোবালি ইউনিক, আর প্রাইভেট নেট এর নোডগুলি শুধুমাত্র তার নিজস্ব নেটওয়ার্ক (রাউটার এর আন্ডারে) এ ইউনিক।

পাবলিক নেটওয়ার্ক এর নোডগুলি সরাসরি ইন্টারনেট এর সাথে যুক্ত একটি করে ইউনিক আইপি দিয়ে। কিন্তু প্রাইভেট গুলি সরাসরি যুক্ত নয়, বরং এরা হয়তো একটি পাবলিক আইপির আন্ডারে অনেকগুলি প্রাইভেট ডিভাইস হিসেবে থাকে, যেগুলি NAT এর মাধ্যমে ইন্টারনেট এ যুক্ত হয়। এটা আরেক টপিক। কখনো যদি নেটওয়ার্কিং নিয়ে লিখি তাহলে বুঝাবো।

এই দুইটার জন্য আবার ২ ধরনের আইপি এড্রেস রেঞ্জ আছে। একটু আগে ৪ টা ক্লাস বলেছিলাম মনে আছে? এদের মাঝে ক্লাস A, B, C এর ব্যবহার সর্বোচ্চ।

নিচে ক্লাস অনুযায়ী প্রাইভেট আইপি রেঞ্জ দেখালামঃ

Class	Private Address Range
A	10.0.0.0 to 10.255.255.255
B	172.16.0.0 to 172.31.255.255
C	192.168.0.0 to 192.168.255

এগুলি সব প্রাইভেট আইপি।

হাহ! আইপি এড্রেসিং বুঝলাম। এবার আসুন ডকার এর দিকে যাই।

## ডকার নেটওয়ার্কিং এবং এড্রেসিং

ডকার ডিফল্টভাবে ক্লাস B এর প্রাইভেট আইপি ব্যবহার করে। যেমনঃ একটা ডকার কন্টেইনার এর আইপিঃ 172.17.42.1। একটা হোস্ট মেশিনের প্রতিটা ডকার এর আইপি আলাদা আলাদা, এর ফলে ডকার ইঞ্জিন তার নেটওয়ার্ক এর মাধ্যমে প্রতিটা কন্টেইনারকে আলাদা করে বের করে।

ডকার এ ৪ ধরনের নেটওয়ার্কিং আছে সাধারণতঃ

০১. ব্রিজ

০২. হোস্ট

০৩. কোনটাই না

দেখুন এই কমান্ড দিয়েঃ `docker network ls`

```
Cyan Tarek TP@Engineer MINGW64 ~
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1f99f44431f7        bridge             bridge              local
ad48b626477c        host               host                local
e3719dbde021        none               null                local
```

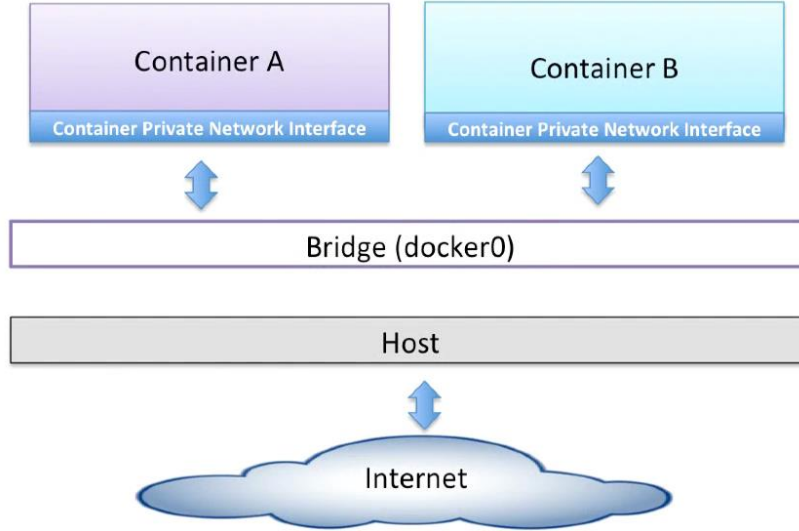
এছাড়াও আছে ওভারলে, কন্টেইনার, Macvlan ইত্যাদি।

আমি এই আর্টিকেলে ডকার ব্রিজ নেটওয়ার্ক নিয়ে বিস্তারিত আলোচনা করবো।

## ডকার ব্রিজ নেটওয়ার্কিং

এটা ডকার এর ডিফল্ট নেটওয়ার্কিং মেথড। খুবই সহজ এবং ডেভলপমেন্ট এর কাজে খুব উপকারী।

## Bridge Network



প্রতিটা কন্টেইনার একেকটা লাইটওয়েট লিনাক্স মেশিন, তাই এগুলি প্রতিটারই নিজস্ব ইথারনেট আছে, যেটাকে ভার্চুয়াল ইথারনেট বলে। এই ইথারনেট গুলি সব ডকার ব্রিজ এর সাথে যুক্ত থাকে। এই ব্রিজ আবার হোস্ট মেশিনের নেটওয়ার্কে যুক্ত থাকে। আর হোস্ট মেশিন যুক্ত থাকে ইন্টারনেট এ।

তাই, ব্রিজ এর আন্ডারে প্রতিটা কন্টেইনার ইন্টারনেট এক্সেস করতে পারে ব্রিজ থেকে হোস্ট, হোস্ট থেকে ইন্টারনেট এ।

ডিফল্টভাবে একটা ব্রিজ এর আন্ডারে প্রতিটা কন্টেইনার একে অপরের সাথে কথা বলতে পারে, কিন্তু ভিন্ন ভিন্ন ব্রিজ এর গুলি একে অপরের সাথে কথা বলতে পারে না।

ব্রিজ নেটওয়ার্ক তৈরি করা

আসুন প্রথমে একটা ব্রিজ নেটওয়ার্ক বানাইঃ

কমান্ডঃ docker network create nw\_1

nw\_1 নামে একটা ব্রিজ নেটওয়ার্ক তৈরি হলো

```
Cyan Tarek TF@Engineer MINGW64 ~  
$ docker network ls  
NETWORK ID          NAME                DRIVER              SCOPE  
1f99f44431f7        bridge             bridge              local  
ad48b626477c        host               host                local  
e3719dbde021        none               null                local  
2b82723b0485        nw_1               bridge              local
```

এবার আমরা দেখি এই ব্রিজ নেটওয়ার্ক এর আন্ডারে কি কি কন্টেইনার আছেঃ

কমান্ডঃ docker network inspect nw\_1

```
Cyan Tarek TF@Engineer MINGW64 ~  
$ docker network inspect nw_1  
[  
  {  
    "Name": "nw_1",  
    "Id": "2b82723b04854f22d8f2d672de862f68b1cac0bc6f72e8994a60a7a6c7e4bb5d",  
    "Created": "2018-03-27T09:01:58.578320882Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.18.0.0/16",  
          "Gateway": "172.18.0.1"  
        }  
      ]  
    },  
    "Internal": false,  
    "Attachable": false,  
    "Ingress": false,  
    "ConfigFrom": {  
      "Network": ""  
    },  
    "ConfigOnly": false,  
    "Containers": {},  
    "Options": {},  
    "Labels": {}  
  }  
]
```



কোন কন্টেইনার নেই, কারণ এটা আমরা মাত্র বানালাম। আরো খেয়াল করুন এর গেটওয়েঃ 172.18.0.1, অর্থাৎ এই ব্রিজ এর নেটওয়ার্ক হলো 172.18.\*.\* রেঞ্জ এর। তাই এর আন্ডারে আমরা 172.18.0.2–172.18.0.254 পর্যন্ত কন্টেইনার রাখতে পারবো।

*নোটঃ 172.18.0.0, 172.18.0.1, 172.18.0.255 এগুলি বিশেষ এড্রেস তাই এগুলি সাধারণত কোনো সাধারণ নোড এ থাকে না।*

এবার আসুন কিছু কন্টেইনার যুক্ত করি এই নেটওয়ার্কে।

আমরা ২ ভাবে কন্টেইনারকে নেটওয়ার্কে যুক্ত করতে পারিঃ

০১. কন্টেইনার বানানোর সময়

০২. বানানোর পরে

বানানোর সময় চাইলে আমরা --net=network\_name ফ্লাগ দিয়ে করতে পারি। যেমনঃ

```
docker run --net=nw_1 --name cont_01 -d busybox sleep 100000
```

```
Cyan Tarek TF@Engineer MINGW64 ~  
$ docker run --net=nw_1 --name cont_01 -d busybox sleep 100000  
cfb3e3c2e008107c6787ef1c4eeffa858f51fb620b142739a7bc9547f
```

এবার আমরা আমাদের nw\_1 টি আবার চেক করে দেখিঃ

```

Cyan Tarek TPEngineer MINGW64 ~
$ docker network inspect nw_1
[
  {
    "Name": "nw_1",
    "Id": "2b82723b04854f22d8f2d672de862f68b1cac0bc6f72e8994a60a7a6c7e4bb5d",
    "Created": "2018-03-27T09:01:58.578320882Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "cfb3e3c2e008107c6787ef1c4eefa858f51fb620b142739a7bcaf29732c9547f": {
        "Name": "cont_01",
        "EndpointID": "525af32c5eef9669f48dc93bee21961cebc119ec14b762c4dc56cf2cf21f73ae",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]

```

ইয়েস, আমাদের কন্টেইনারটি এই ব্রিজ এ যুক্ত হয়েছে। দেখুন এই কন্টেইনার এর আইপি এড্রেস 172.18.0.2 অর্থাৎ এটি nw\_1 ব্রিজ এর আন্ডারে।

এবার ধরি, আগে cont\_02 নামক কন্টেইনারটি বানিয়েছি এবং এটি nw\_1 নেটওয়ার্কে যুক্ত নেই। আমরা একে যুক্ত করতে চাইলে:

docker network connect nw\_1 cont\_02 কমান্ড দেই।

এবার চেক করে দেখি:

```

$ docker network inspect nw_1
[
  {
    "Name": "nw_1",
    "Id": "2b82723b04854f22d8f2d672de862f68b1cac0bc6f72e8994a60a7a6c7e4bb5d",
    "Created": "2018-03-27T09:01:58.578320882Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "9609e79ee7553048e6eaa709c5d2b68a0417fc58e63e857d5736e73cf8358665": {
        "Name": "cont_02",
        "EndpointID": "a0b05b96df00af87592535ac61a12fd3eda88d49ce44040b52697344b414f2e",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
      },
      "cfb3e3c2e008107c6787ef1c4eefa858f51fb620b142739a7bcdf29732c9547f": {
        "Name": "cont_01",
        "EndpointID": "525af32c5eef9669f48dc93bee21961cebc119ec14b762c4dc56cf2cf21f73ae",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]

```

তাহলে আমরা শিখলাম কিভাবে করে নতুন ব্রিজ নেটওয়ার্ক বানানো যায় এবং কোনো কন্টেইনারকে একটি নেটওয়ার্ক এ যুক্ত করা যায়।

এবার দেখি এরা নিজেদের মাঝে কথা বলতে পারে কিনা।

বাই দে ওয়ে, কথা বলা মানে কানেকশন।

**কন্টেইনার কানেকশন**

প্রথমে দেখি cont\_01 আর cont\_02 কথা বলতে পারে কিনা একে অপরের সাথে:

আইপি এড্রেস:

cont\_01: 172.18.0.2

cont\_02: 172.18.0.3

কমান্ডঃ docker exec -it cont\_01 ping 172.18.0.3

```
Cyan Tarek TFEngineer MINGW64 ~  
$ docker exec -it cont_01 ping 172.18.0.3  
PING 172.18.0.3 (172.18.0.3): 56 data bytes  
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.150 ms  
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.169 ms  
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.140 ms  
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.123 ms  
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.124 ms  
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.122 ms  
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.128 ms  
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.124 ms  
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.155 ms  
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.123 ms  
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.099 ms  
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.127 ms  
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.123 ms  
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.100 ms  
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.123 ms  
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.096 ms  
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.185 ms  
^C  
--- 172.18.0.3 ping statistics ---  
17 packets transmitted, 17 packets received, 0% packet loss  
round-trip min/avg/max = 0.096/0.130/0.185 ms
```

ইয়েস, পেয়েছে। এবার দেখি এরা ইন্টারনেট এ যুক্ত হতে পারে কিনা। এর জন্য আমি গুগলের পাবলিক DNS 8.8.8.8 এ পিং করে দেখিঃ

```
Cyan Tarek TF@Engineer MINGW64 ~  
$ docker exec -it cont_02 ping 8.8.8.8  
PING 8.8.8.8 (8.8.8.8): 56 data bytes  
64 bytes from 8.8.8.8: seq=0 ttl=54 time=63.596 ms  
64 bytes from 8.8.8.8: seq=1 ttl=54 time=63.407 ms  
64 bytes from 8.8.8.8: seq=2 ttl=54 time=62.862 ms  
64 bytes from 8.8.8.8: seq=3 ttl=54 time=255.155 ms  
64 bytes from 8.8.8.8: seq=4 ttl=54 time=417.844 ms  
64 bytes from 8.8.8.8: seq=5 ttl=54 time=91.412 ms  
64 bytes from 8.8.8.8: seq=6 ttl=54 time=258.568 ms  
64 bytes from 8.8.8.8: seq=7 ttl=54 time=151.322 ms  
^C  
--- 8.8.8.8 ping statistics ---  
8 packets transmitted, 8 packets received, 0% packet loss  
round-trip min/avg/max = 62.862/170.520/417.844 ms
```

এটাও পেরেছে।

এবার দেখি দুইটা ব্রিজ এর নেটওয়ার্ক কিভাবে কানেক্ট করতে পারে।

নতুন আরেকটা নেটওয়ার্ক বানান আগের নিয়মানুসারে, নাম দিন nw\_2

এবার নতুন একটা কন্টেইনার cont\_03 বানাইএবং nw\_2 নেটওয়ার্ক এ যুক্ত করি

```

Ryan Tarek TFAEngineer MINGW64 ~
$ docker network create nw_2
645b311ed43c193c616e905e5b0a2a512938782f56565da66d48d66610327155

Ryan Tarek TFAEngineer MINGW64 ~
$ docker run -d --name cont_03 --net nw_2 busybox sleep 10000
ff606728fa8f042d220cdcd6cf1a95f3fc33acd884f32714801264e61c0fe5b5

Ryan Tarek TFAEngineer MINGW64 ~
$ docker network inspect nw_2
[
  {
    "Name": "nw_2",
    "Id": "645b311ed43c193c616e905e5b0a2a512938782f56565da66d48d66610327155",
    "Created": "2018-03-27T09:25:50.123150637Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "ff606728fa8f042d220cdcd6cf1a95f3fc33acd884f32714801264e61c0fe5b5": {
        "Name": "cont_03",
        "EndpointID": "fb898eb11f1143d205ec41abf37a449a36d7d66c3f047df73e271c5a03f0e3f7",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]

```

এই নেটওয়ার্ক এড্রেস 172.19.\*.\* এর, যা আগের নেটওয়ার্ক থেকে ভিন্ন। অর্থাৎ এটি আলাদা নেটওয়ার্ক।

এবার দেখি nw\_1 এর কন্টেইনার nw\_2 এর কন্টেইনার এর সাথে কথা বলতে পারে কিনাঃ

nw\_1 — cont\_02,

nw\_2 — cont\_03 — IP: 172.19.0.2

```
Cyan Tarek TF@Engineer MINGW64 ~/docker
$ docker exec -it cont_02 ping 172.19.0.2
PING 172.19.0.2 (172.19.0.2): 56 data bytes
^C
--- 172.19.0.2 ping statistics ---
16 packets transmitted, 0 packets received, 100% packet loss
```

পারলো না। তার মানে একটা ব্রিজ এর আন্ডারে প্রতিটা কন্টেইনার একে অপরের সাথে কথা বলতে পারে, কিন্তু ভিন্ন ভিন্ন ব্রিজ এর গুলি একে অপরের সাথে কথা বলতে পারে না।

কানেক্ট করতে চাইলে কন্টেইনারগুলিকে একই ব্রিজ এ কানেক্ট করতে হবে। cont\_03 কে nw\_1 নেটওয়ার্ক এ যুক্ত করি আগের নিয়মে।

একটি কন্টেইনার একই সাথে একাধিক নেটওয়ার্ক ব্রিজ এ কানেক্ট থাকতে পারে। cont\_03 এর নেটওয়ার্ক দেখিঃ docker inspect cont\_03

```
{
  "Networks": {
    "nw_1": {
      "IPAMConfig": {},
      "Links": null,
      "Aliases": [
        "ff606728fa8f"
      ],
      "NetworkID": "2b82723b04854f22d8f2d672de862f68b1cac0bc6f72e8994a60a7a6c7e4bb5d",
      "EndpointID": "b18459a650a6976db395befcd1f04c9544177f632c74596f4d1321b09c9353dc",
      "Gateway": "172.18.0.1",
      "IPAddress": "172.18.0.4",
      "IPPrefixLen": 16,
      "IPv6Gateway": "",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "MacAddress": "02:42:ac:12:00:04",
      "DriverOpts": null
    },
    "nw_2": {
      "IPAMConfig": null,
      "Links": null,
      "Aliases": [
        "ff606728fa8f"
      ],
      "NetworkID": "645b311ed43c193c616e905e5b0a2a512938782f56565da66d48d66610327155",
      "EndpointID": "fb898eb11f1143d205ec41abf37a449a36d7d66c3f047df73e271c5a03f0e3f7",
      "Gateway": "172.19.0.1",
      "IPAddress": "172.19.0.2",
      "IPPrefixLen": 16,
      "IPv6Gateway": "",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "MacAddress": "02:42:ac:13:00:02",
      "DriverOpts": null
    }
  }
}
```

এবার cont\_03 থেকে cont\_01 এর আইপি তে পিং করিঃ

```
Cyan Tarek TF@Engineer MINGW64 ~/docker
$ docker exec -it cont_03 ping 172.18.0.3
PING 172.18.0.3 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.128 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.141 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.131 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.131 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.127 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.131 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.127 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.132 ms
^C
--- 172.18.0.3 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.127/0.131/0.141 ms
```

ইয়েস এবার কাজ করতেছে। আশা করি বুঝতে পেরেছেন।

### নেটওয়ার্ক লিঙ্কিং

ব্রিজ নেটওয়ার্ক এর আইপি এড্রেসিং স্ট্যাটিক না, বরং DHCP নির্ভর, মানে ডায়নামিক। তাই একটা কন্টেইনার রিস্টার্ট করলেই আইপি চেঞ্জ হয়ে যায়। এটা অনেক বড় ধরনের সমস্যা।

যেমন ধরুন, আপনি আপনার ডাটাবেজকে db কন্টেইনারে রান করেছেন যার আইপি 172.18.0.4, আর আপনার ওয়েব এপ্লিকেশনকে web কন্টেইনারে রান করেছেন যার আইপি 172.18.0.6

web কন্টেইনার অবশ্যই ডাটাবেজ এর db কন্টেইনারকে এক্সেস করবে। এর জন্য ডাটাবেজ কানেকশন এর HOST অংশে db এর আইপি দিবেন। কিন্তু db একবার রিস্টার্ট করলেই আইপি চেঞ্জ হবে, তাই তখন আর web থেকে এক্সেস করতে পারবেন না।



এ থেকে মুক্তির জন্য ডকার এ নেটওয়ার্ক লিঙ্কিং এর সুবিধা দেয়। এটার মাধ্যমে হার্ডকোড করে আইপি লেখার বদলে আপনি কন্টেইনার এর নাম দিবেন HOST অংশে। ডকার প্রতিটা কন্টেইনার এর আইপি জমা করে রাখে, চেষ্টা হলে আপডেট করে। তাই এক্ষেত্রে আর সমস্যা হবে না।

যেমনঃ আমার একটা মাইক্রোসার্ভিস প্রজেক্টে আমি MongoDB ডাটাবেজ এবং Django ওয়েব এপ্লিকেশন ফ্রেমওয়ার্ক ব্যবহার করেছি। MongoDB এর কন্টেইনারের নাম দিয়েছিলাম mongod

নিচে দেখুন এখানে Django এর ডাটাবেজ কনফিগারেশনে আমি HOST এ mongod কন্টেইনারের আইপির বদলে কন্টেইনারের নাম দিয়েছি ডকার নেটওয়ার্ক লিঙ্ক এর সুবিধা নিয়ে।

```
DATABASES = {  
    'default': {  
        'NAME': 'emails',  
        'HOST': 'mongod',  
        'PORT': 27017  
    }  
}
```

### সার সংক্ষেপ

ডকার ব্রিজ নেটওয়ার্কিং খুব ছোট একটা নেটওয়ার্কিং এবং একই মেশিন নেটওয়ার্কিং এর জন্য ভালো। কিন্তু প্রোডাকশন লেভেলে মাল্টি মেশিন এবং অনেকগুলি কন্টেইনারের মাঝে নেটওয়ার্কিং এর জন্য “ওভারলে” নেটওয়ার্কিং খুবই কার্যকর।

একাধিক মেশিনে একাধিক কন্টেইনার ম্যানেজ করার জন্য ডকার সোয়ার্ম বা কুবারনেটিস খুবই কার্যকর। আমরা যখন ডকার সোয়ার্ম বা কুবারনেটিস দিয়ে একাধিক মেশিনে একাধিক কন্টেইনার ডিপ্লয় করবো তখন দেখবো কিভাবে করে ডকার ওভারলে নেটওয়ার্কিং করতে হয়।

সামনে পর্বে ডকার সোয়ার্ম দিয়ে অনেকগুলি ডকার কন্টেইনার অর্কেস্ট্রেশন বা ম্যানেজমেন্ট দেখবো, তারপর দেখবো সোয়ার্ম এর বদলে কুবারনেটিস দিয়ে।

আজকে এই পর্যন্তই। কোথাও ভুল থাকলে অথবা বুঝতে সমস্যা হলে অবশ্যই জানাবেন।

## ডকার সোয়ার্ম — অনেক অনেক ডকার কন্টেইনার

ধরুন আপনি একজন কন্সট্রাকশন কাজের ম্যানেজার। আপনি ক্লায়েন্ট এর কাছ থেকে কাজ নেন এবং প্রয়োজন মতো লোকজন নিয়োগ করে কাজ দেখাশোনা করেন।

আপনার হাতে একটা নতুন কাজ এসেছে। আপনি তেমন একটা আন্দাজ করতে পারতেছেন না যে কাজটা কতটুকু বড় হবে। ক্লায়েন্ট শুধু বলেছে সামনে কাজ বাড়লে বাড়তেও পারে তবে শিওর না। তাই আপনি মাত্র একজন মিস্ট্রিকে দায়িত্ব দিলেন কন্সট্রাকশন এর কাজটি করার জন্য। (কেউ হাসবেন না)

কিছুদিন পরেই উপলব্ধি করতে পারলেন, একজন দিয়ে জীবনেও বাস্তব জগতের কন্সট্রাকশনের কাজ করা যাবে না। তাই আপনি আরো ৫ জন মিস্ট্রি যোগ করলেন। এরপর আরো ১০ জন। এভাবে করে ঐ কাজে মিস্ট্রির সংখ্যা দাড়ালো ৫০ জন।

এখন আপনার কাঁধে দায়িত্ব আসলো সব মিস্ত্রিদের ম্যানেজ করা। তারা ঠিকমত কাজ করছে কিনা, সবাই সুস্থ অবস্থায় আছে কিনা, কেউ অসুস্থ হলে তার বদলে আরেকজন মিস্ত্রিকে যোগ করা এসব কাজ আপনি দেখাশোনা বা মনিটরিং করছেন।

কি ভাবছেন? লেখার টপিক ডকার নিয়ে কিন্তু আলোচনা করতেছি বাড়ি ঘরের কাজ নিয়ে? একটু উদাহরণ দিলাম। আপনাদের বুঝতে সুবিধা হবে।

ডকার নিয়ে আমার নতুন লেখায় স্বাগতম। আমি আব্দুল্লাহ আল তারেক। কম্পিউটার সায়েন্স এন্ড ইঞ্জিনিয়ারিং পড়ুয়া একজন সফটওয়্যার ইঞ্জিনিয়ার, ব্যাকএন্ড নিপুণ, ডেভঅপ্স প্লেয়ার এবং মাইক্রোসার্বিস এ অভিজ্ঞ। পাইথন/জ্যাঙ্গো খুব ভালোবাসি ❤️

আগের লেখাগুলি পড়ে থাকলে আপনি নিশ্চয় এখন ডকার সম্পর্কে ভালো প্র্যাকটিকাল ধারণা পেয়ে গেছেন।

## প্রোডাকশন লেভেল

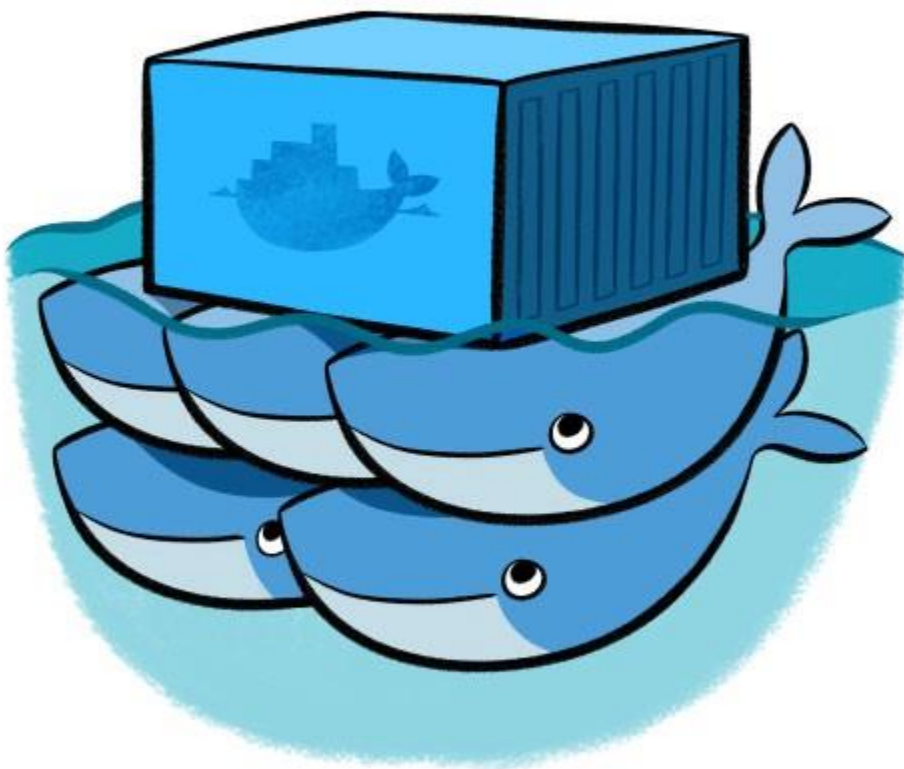
এতদিন যে লেখাগুলি লিখেছিলাম সেগুলি ছিলো সিঙ্গেল কন্টেইনারের মত প্রজেক্ট। উপরের উদাহরণের মতো প্রোডাকশন লেভেলে মাত্র একটা কন্টেইনার দিয়ে কোন কাজই হবে না। তাই আপনাকে অনেকগুলি কন্টেইনার চালাতে হবে।

অনেক গুলি কন্টেইনার চালাতে গেলেই আপনি উপরের উদাহরণের মত একটা কর্তব্যে পড়বেনঃ আপনাকে প্রতিটা কন্টেইনার ম্যানেজ করতে হবে সব সময়। কোনো কন্টেইনার ডাউন হয়ে গেলে আপনার প্রজেক্ট স্টপ হয়ে থাকবে। আর কন্টেইনার খুব সহজেই একটা এরোর পেলেই

স্টপ হয়ে যায় (যদিও Restart Always ফ্ল্যাগ দিয়ে অটো রিস্টার্ট করা যায়)। এগুলিকে বলে কন্টেইনার লাইফ সাইকেল।

আপনি একা ৫০০ কন্টেইনারের লাইফ সাইকেল ম্যানেজ করতে পারবেন? ৫০০০? ১ লাখ?

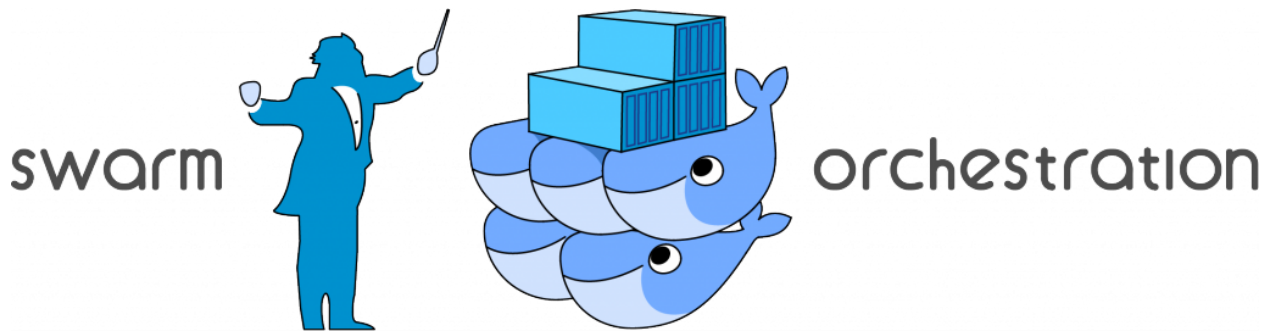
পারবেন না। এটা অনেক সময় স্বাপেক্ষ আর বিরক্তিকর কাজ। কখন কোন পার্ট এর লোড বেশি হলে সেই পার্টে আরো কিছু কন্টেইনার যোগ করার জন্য আপনি বসে থাকবেন না নিশ্চয়। কোনো কন্টেইনার ডাউন হলে আবার আরেকটা বানানোর জন্যও আপনি বসে থাকবেন না নিশ্চয়।



এসব কাজকে ম্যানেজ করার জন্যই **ডকার সোয়ার্ম**।

ডকার সোয়ার্ম আসলে কিছুই না, অনেক অনেক কন্টেইনার ম্যানেজ করার একটা সিস্টেম যেখানে আপনাকে ম্যানেজ করার ঝামেলা অনেক কম পোহাতে হবে।

অনেক কিছু এক সাথে থাকলে সেটাকে **ক্লাস্টার** বলে। সুতরাং ডকার সোয়ার্ম এর কাজ হলো ডকার কন্টেইনার ক্লাস্টার ম্যানেজ করা। এই ম্যানেজ করাকে ডকার বা কন্টেইনারের ভাষায় বলা হয় “**কন্টেইনার অর্কেস্ট্রেশন**”



ডকারে আমরা যখন docker run কমান্ড ব্যবহার করি তখন একটি কন্টেইনার বানাই। কিন্তু ডকার সোয়ার্মে আমরা সরাসরি কন্টেইনার বানানোর কমান্ড দেই না। বুঝতে সমস্যা হচ্ছে? আসুন প্র্যাকটিক্যালি যাই।

ডকার সোয়ার্ম এর কিছু কনসেপ্ট

**নোড:** এক একটি হোস্ট কম্পিউটার যেখানে ডকার চালানো থাকে এবং যারা সোয়ার্ম এ অংশ নেয় তারাই নোড

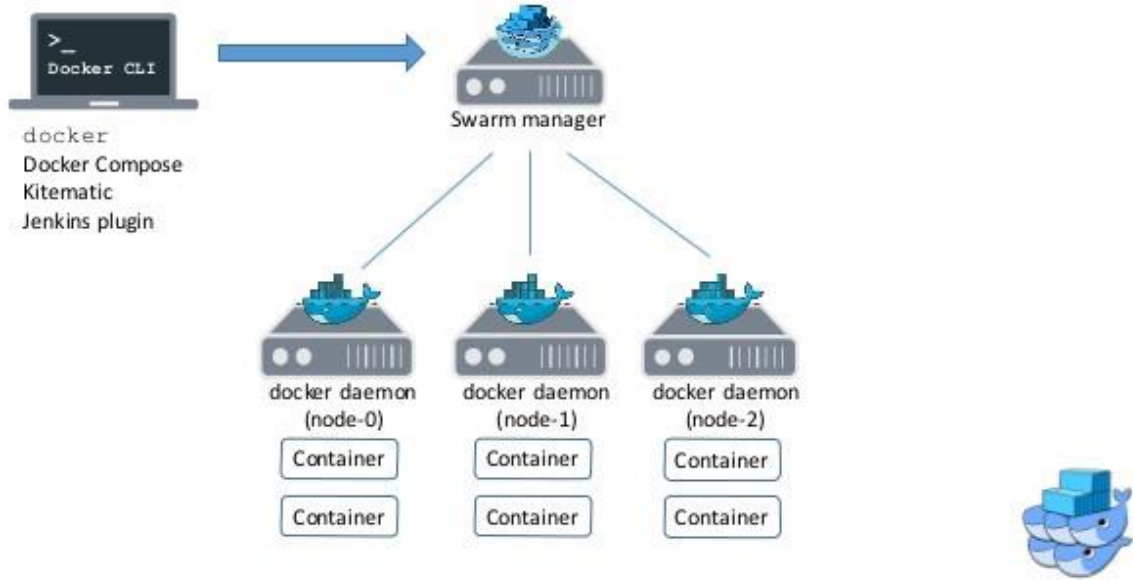
**ম্যানেজার নোড:** সোয়ার্ম এর স্পেশাল নোড যারা কন্টেইনার ক্লাস্টার ও অন্যান্য নোড সমূহকে ম্যানেজ করে

**ওয়ার্কার নোড:** সোয়ার্ম এর সাধারণ নোড যারা সোয়ার্মে থেকে সকল কাজগুলি করে থাকে।  
ম্যানেজার নোড নিজেও ওয়ার্কার নোড হতে পারে

**লিডার:** একটা সোয়ার্মে অনেকগুলি ম্যানেজার থাকতে পারে কিন্তু তাদের মাঝে লিডার থাকবে  
মাত্র একজন যে সকল ম্যানেজমেন্ট এর কাজ করে থাকে

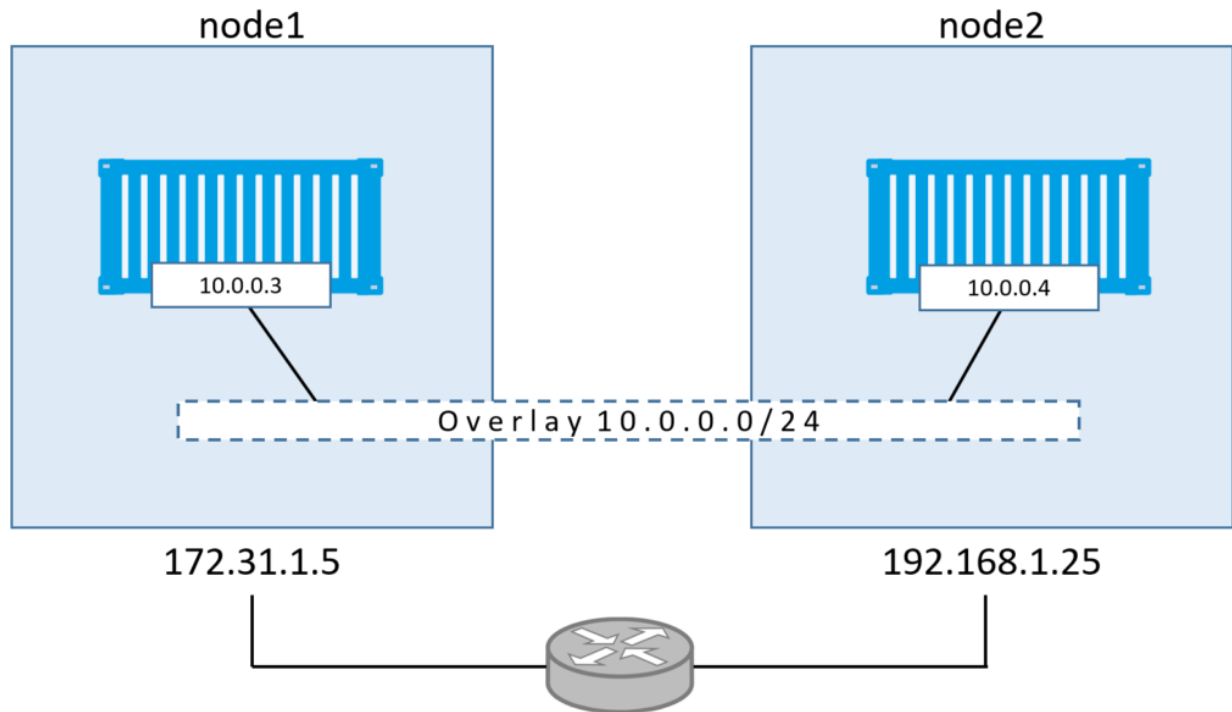
**সার্ভিস:** ডকার ইমেজকে কন্টেইনারে রান করার সিস্টেম, কিন্তু সোয়ার্মে। আগে কন্টেইনারে  
run কমান্ড দিলে একই ধরনের ইমেজ এর মাত্র একটা কন্টেইনার রান হতো। কিন্তু সার্ভিস এ  
একই কাজ করা হয়, কিন্তু একই ধরনের ইমেজ এর যতগুলি ইচ্ছা কন্টেইনার রান করা যাবে।  
আরো সুবিধা হলো, সাধারণ ডকার নোডে রান কমান্ড দেয়া হলে কন্টেইনার শুধু সেই নোডেই  
তৈরি হতো। কিন্তু সার্ভিসে ম্যানেজার নোডে কন্টেইনার সার্ভিস বানাতে সেগুলি সোয়ার্মের সাথে  
যুক্ত সকল নোডে তৈরি হবে।

**রেপ্লিকেশন:** একই ধরনের কন্টেইনারকে যতগুলি ইচ্ছা ততগুলি বানিয়ে বিভিন্ন ওয়ার্কার নোডে  
রান করাই হলো রেপ্লিকেশন। নোড যদি ৩ টা থাকে আর একটা কন্টেইনার সার্ভিসকে যদি ১০ টা  
রেপ্লিকেশন করা হয় তাহলে ৩ টা নোডে মাল্টিপল টাইম একই কন্টেইনার তৈরি হবে। এতে করে  
একটা নোডে যদি একই কন্টেইনার এর ৩ টা ইনস্টেন্স থাকে আর তার মধ্যে একটা ডাউন হয়ে  
যায় তাহলে বাকিগুলি ব্যাকআপ দিতে পারবে। আর এর মধ্যে ম্যানেজার নতুন আরেকটি  
কন্টেইনার রান করে রেপ্লিকেশন এর মান সমান রাখবে এবং ক্ষতির হাত থেকে বাঁচাবে। খুবই  
দারুণ একটা জিনিস। এই সিস্টেমকে বলে “হাই এভেইলাবিলিটি”



**মাল্টি নোড/হোস্ট নেটওয়ার্কিং:** আমার আগের ডকার নেটওয়ার্কিং লেখায়

আমি “ওভারলে নেটওয়ার্কিং” এর কথা বলেছিলাম যে এটা সোয়ার্মে কাজ লাগে, এটাও বলেছিলাম যে সোয়ার্ম বানানোর সময় এটা দেখাবো। সোয়ার্মে যেহেতু একাধিক নোড কানেক্টেড থাকে যারা পৃথিবীর বিভিন্ন প্রান্তে থাকতে পারে, তাই তাদের মাঝে মাল্টি নোড বা মাল্টি হোস্ট একটা নেটওয়ার্কিং দরকার। ডকার এটাকে **ওভারলে নেটওয়ার্কিং** বলে। আর সোয়ার্মে এই ওভারলে নেটওয়ার্কিং কে “ইনগ্রেস” নামে ডাকা হয়।



প্রতিটা নোডের নিজস্ব পাবলিক আইপি আছে। তারা যখন ডকার সোয়ার্মে অংশ নেয় তখন তাদের মাঝে একটা মাল্টি নোড ভার্চুয়াল ওভারলে নেটওয়ার্ক বানানো হয় যেটা প্রাইভেট নেটওয়ার্ক। এখানে ডকার প্রতিটা নোডের পাবলিক আইপিকে ওভারলে নেটওয়ার্কের প্রাইভেট আইপির সাথে ম্যাপিং করে দেয়। এতে করে সোয়ার্মের সব নোড সহজেই কানেক্টেড থাকতে পারে

এবার আসুন আমরা সোয়ার্ম নিয়ে কাজ করি

প্রথমেই দেখে নিন আপনার কম্পিউটারটি কি কোনো সোয়ার্মে যুক্ত কিনাঃ  
`docker info`



```
Server Version: 18.02.0-ce
Storage Driver: aufs
Root Dir: /mnt/sda1/var/lib/docker/aufs
Backing Filesystem: extfs
Dirs: 220
Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald js
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 9b55aab90508bd389d7654c4ba
runc version: 9f9c96235cc97674e935002fc3d78361
init version: 949e6fa
Security Options:
seccomp
```

তার মানে আমাদের কম্পিউটার কোনো সোয়ার্মে যুক্ত নেই। আসুন আমরা সোয়ার্ম বানাই। প্রথমে আমি দেখাবো একটা সিঙ্গেল নোডে সোয়ার্ম করে। এরপরের লেখায় দেখাবো মাল্টি নোডে সোয়ার্ম করা।

আপনার কম্পিউটারকে ম্যানেজার হিসেবে রেখে ডকার সোয়ার্ম বানাতে চাইলে নিচের কমান্ড দিন

```
docker swarm init --advertise-addr your_machine_ip
```

এখানে your\_machine\_ip এর জায়গায় আপনার কম্পিউটারের আইপি এড্রেস দিবেন। আমি যেহেতু ডকার মেশিনে ডকার চালাচ্ছি তাই আমি ডকার মেশিনের আইপি দিবো, অন্যথায় আমার

পিসির লোকাল আইপি দিতাম। আপনি যদি এটা কোনো ডেডিকেটেড সার্ভারে বানাতে যান তাহলে অবশ্যই ঐ সার্ভারের ডেডিকেটেড পাবলিক আইপি দিবেন।

এরপর নিচের মত একটা আউটপুট পাবেনঃ

```
PS C:\Users\Tushar\Documents> docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (6d2j2e0hyn735sxj1c6c01gtv) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join --token SWMTKN-1-0bktk9q0w95qn1hb8wk9di0serprjiw2n5a20x1qz9we1l99cw-b19tj14temib9um4mds1dxhc0 192.168.99.100:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

তৈরি হয়ে গেলো। সাথে আমরা একটি টোকেন পেয়েছি। এই টোকেন এর সাহায্যে আমরা চাইলে যেকোন কম্পিউটার থেকে এই সোয়ার্মে অংশ নিতে পারবো। কিন্তু আমরা যেহেতু প্রাইভেট নেটওয়ার্কে বা লোকালি সোয়ার্ম বানিয়েছি, তাই শুধুমাত্র এই নেটওয়ার্ক এর অন্তর্গত পিসি গুলি সোয়ার্মে যুক্ত হতে পারবে। পাবলিক নেটওয়ার্কের কম্পিউটারে সোয়ার্ম বানাতে যে কোনো পিসিই যুক্ত হতে পারবে (কিন্তু শেয়ার্ড আইপি থেকে মাত্র একটা পিসি যুক্ত হতে পারবে, কারণ আইপি একটা শেয়ার্ড এ)

এবার ডকার ইনফো তে গিয়ে দেখুন আমাদের কম্পিউটার সোয়ার্মে যুক্ত হয়েছেঃ

```
Log: awslogs fluentd gcplogs gelf journald
Swarm: active
NodeID: 6d2j2e0hyn735sxj1c6c01gtv
Is Manager: true
ClusterID: 9i4cwg3hd58lwscwagt8dkjhr
Managers: 1
Nodes: 1
Orchestration:
  Task History Retention Limit: 5
```

এই সোয়ার্মে কতগুলি নোড যুক্ত আছে তা দেখতেঃ

```
docker node ls
```

```
Pyon Tarek Tarek@MINGW64 ~  
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
6d2j2e0hyn735sxjl6c0lgtv *	default	Ready	Active	Leader

মাত্র একটি নোড যুক্ত আছে এবং এটা আমাদের কম্পিউটার।

এবার আসুন আমরা এই সোয়ার্মে অনেকগুলি কন্টেইনারের ক্লাস্টার বানাই। এর জন্য আমাদের কে ঐ কন্টেইনার ইমেজ এর একটা সার্ভিস বানাতে হবে

```
docker service create -p 82:80 --name hello-swarm tutum/hello-world
```

এটা ঠিক আমাদের docker run কমান্ডের মতই।

এখানে আমরা tutum/hello-world নামের একটা ইমেজকে কন্টেইনারে রান করেছি। কিন্তু এটা সার্ভিস হিসেবে। সার্ভিসে কন্টেইনারগুলি সোয়ার্মে কানেক্ট থাকা সকল নোডে তৈরি হবে রিপ্লিকেশন এর পরিমাণের উপর নির্ভর করে। আসুন দেখি সার্ভিসের সুবিধা কি

প্রথমেই দেখি আমাদের এই সোয়ার্মের সার্ভিস লিস্ট

```
docker service ls
```

```
Pyon Tarek Tarek@MINGW64 ~  
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
h75jgmrfxufy	hello-swarm	replicated	1/1	tutum/hello-world:latest	*:82->80/tcp

আমরা মাত্র এই সার্ভিসটি বানিয়েছি। দেখুন সার্ভিসটি রিপ্লিকেটেড, অর্থাৎ একই কন্টেইনার এর কয়েকটা ইনস্টেন্স আমরা বানাতে পারি।

আসুন দেখি এই সার্ভিসের কন্টেইনারগুলি কোন কোন নোডে তৈরি হয়েছে (যদিও কন্টেইনার রেন্সিকেট হয়েছে মাত্র ১ টি আর আমাদের নোডও একটি)  
`docker service ps hello-swarm`

```
$ docker service ps hello-swarm
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
5i4uedu1qkp6	hello-swarm.1	tutum/hello-world:latest	default	Running	Running 5 minutes ago	

কন্টেইনারটি আমাদের নোডে তৈরি হয়েছে, আমাদের নোডের নাম default

তাহলে আসুন আমরা দেখি আসলেই কি আমাদের নোডে কন্টেইনারটি তৈরি হয়েছে কিনা:

```
ivan@tutum: ~$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1442a8acfa7d	tutum/hello-world:latest	"/bin/sh -c 'php-fpm'"	6 minutes ago	Up 6 minutes	80/tcp	hello-swarm.1.5i4ue

ইয়েস! তৈরি হয়েছে। আমরা কিন্তু docker run এর বদলে docker service দিয়ে বানিয়েছি সোয়ার্মে। এতে ইন-ডিবেক্টলি আমাদের নোডে কন্টেইনার বানিয়েছি সোয়ার্ম দিয়ে।

এবার আমাদের ব্রাউজারের [http://machine\\_ip:82](http://machine_ip:82) (Docker machine হলে)

বা <http://localhost:82> তে গিয়ে দেখি

← → ↻ | 192.168.99.102:82



**Hello world!**

My hostname is 1442a8acfa7d

ইয়েস, রান হয়েছে।

ধরি, আমরা এই সার্ভিসটিকে আপ স্কেল করে ৫ টা বানাতে চাই। তাহলে আমরা যা করবোঃ

```
docker service scale hello-swarm=5
```

অথবা

```
docker service update hello-swarm --replicas 5
```

```
MINGW64 ~
$ docker service update hello-swarm --replicas 5
hello-swarm
overall progress: 5 out of 5 tasks
1/5: running [=====]
2/5: running [=====]
3/5: running [=====]
4/5: running [=====]
5/5: running [=====]
verify: Service converged

MINGW64 ~
$ docker service ps hello-swarm
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR
534uedu1qkp6 hello-swarm.1 tutum/hello-world:latest default Running Running 13 minutes ago
a7mj1voewwyg hello-swarm.2 tutum/hello-world:latest default Running Running 19 seconds ago
m90znn638ean hello-swarm.3 tutum/hello-world:latest default Running Running 19 seconds ago
s23f0qa30mw1 hello-swarm.4 tutum/hello-world:latest default Running Running 19 seconds ago
kmywpc0hrbo3 hello-swarm.5 tutum/hello-world:latest default Running Running 19 seconds ago

MINGW64 ~
$ docker service ls
ID NAME MODE REPLICAS IMAGE PORTS
h75jgmrfxufy hello-swarm replicated 5/5 tutum/hello-world:latest *:82->80/tcp
```

বুম! মাত্র এক কমান্ডে আমরা ৫ টা কন্টেইনার বানিয়ে ফেললাম। যেহেতু নোড একটা তাই এই কন্টেইনারগুলি সব এক নোডেই তৈরি হয়েছে।

আমরা আমাদের ব্রাউজারে আগের এড্রেসে গেলে ডকার সোয়ার্ম লোড ব্যালেন্স করে একেকবার একেক কন্টেইনার এ দেখাবে। দারুন!

এবার আসুন দেখি একটা কন্টেইনার ডাউন হয়ে গেলে কি হয়। এর জন্য আমরা একটা কন্টেইনারকে স্টপ করে ডাউন করে দেই আমাদের নোড থেকেঃ

```

Cyan Tarek TPSEngineer@MINGW64 ~
$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS              NAMES
4138be0a84fd        tutum/hello-world:latest    "/bin/sh -c 'php-fpm'"   3 minutes ago       Up 3 minutes       80/tcp             hello-swarm.3.m90zn
e7c9d5bb7cf5        tutum/hello-world:latest    "/bin/sh -c 'php-fpm'"   3 minutes ago       Up 3 minutes       80/tcp             hello-swarm.4.s23fc
a27c1c3b3e3e        tutum/hello-world:latest    "/bin/sh -c 'php-fpm'"   3 minutes ago       Up 3 minutes       80/tcp             hello-swarm.2.a7mj1
44eac59f17b0        tutum/hello-world:latest    "/bin/sh -c 'php-fpm'"   3 minutes ago       Up 3 minutes       80/tcp             hello-swarm.5.kmwyp
1442a8acfa7d        tutum/hello-world:latest    "/bin/sh -c 'php-fpm'"   16 minutes ago      Up 16 minutes      80/tcp             hello-swarm.1.514ue

Cyan Tarek TPSEngineer@MINGW64 ~
$ docker container stop 4138be0a84fd
4138be0a84fd

Cyan Tarek TPSEngineer@MINGW64 ~
$ docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
h75jgmrfxufy     hello-swarm         replicated          4/5                 tutum/hello-world:latest    *:82->80/tcp

Cyan Tarek TPSEngineer@MINGW64 ~
$ docker service ps hello-swarm
ID                NAME                IMAGE                NODE                DESIRED STATE       CURRENT STATE       ERROR
514uedu1qkp6     hello-swarm.1       tutum/hello-world:latest    default            Running              Running 17 minutes ago
a7mj1voewwyq     hello-swarm.2       tutum/hello-world:latest    default            Running              Running 4 minutes ago
49mto2h13f7q     hello-swarm.3       tutum/hello-world:latest    default            Running              Running 9 seconds ago
m90znn638ean     hello-swarm.3       tutum/hello-world:latest    default            Shutdown             Complete 15 seconds ago
s23f0qa30mw1     hello-swarm.4       tutum/hello-world:latest    default            Running              Running 4 minutes ago
kmwypc0hrbo3     hello-swarm.5       tutum/hello-world:latest    default            Running              Running 4 minutes ago

Cyan Tarek TPSEngineer@MINGW64 ~
$

```

দেখুন আমাদের একটা কন্টেইনার শাট ডাউন দেখাচ্ছে। একটু অপেক্ষা করে আবার চেক করে ম্যাজিক দেখুনঃ

```

Cyan Tarek TPSEngineer@MINGW64 ~
$ docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
h75jgmrfxufy     hello-swarm         replicated          5/5                 tutum/hello-world:latest    *:82->80/tcp

```

অসাধারণ! মাত্র কয়েক সেকেন্ডের ব্যবধানে ডকার সোয়ার্ম ম্যানেজার আরেকটা নতুন কন্টেইনার তৈরি করে ক্ষতিপূরণ করেছে। এটাই হচ্ছে ডকার সোয়ার্ম এর অসাধারণ সুবিধা।

এবার ধরি আমরা চাচ্ছি আমাদের সার্ভিসকে ডাউন স্কেল করবো, অর্থাৎ কন্টেইনার সংখ্যা কমিয়ে ৩ টা করবো। এর জন্যঃ

```
docker service scale hello-swarm=3
```

অথবা

```
docker service update hello-swarm --replicas 3
```

```

$ docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
h75jgmrfxufy     hello-swarm         replicated           5/5                 tutum/hello-world:latest *:82->80/tcp

$ docker service scale hello-swarm=3
hello-swarm scaled to 3
overall progress: 3 out of 3 tasks
1/3: running [=====]
2/3: running [=====]
3/3: running [=====]
verify: Service converged

$ docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
h75jgmrfxufy     hello-swarm         replicated           3/3                 tutum/hello-world:latest *:82->80/tcp

```

বুম! অসাধারণ তাই না? আমাদের যখন সার্ভারে লোড বেশি হবে তখন আমরা আপ স্কেল করবো, আবার যখন লোড কম হবে তখন ডাউন স্কেল করবো। এতে করে আমরা যদি প্রতি ঘন্টা খরচের ভিপিএস ব্যবহার করি (যেমনঃ Amazon AWS) তাহলে খরচ অনেক বাচাতে পারবো।

কোনো সার্ভিসকে রিমুভ করতে চাইলে

```
docker service rm hello-swarm
```

এর ফলে সোয়ার্ম ম্যানেজার অটোম্যাটিকালি ঐ সার্ভিসের সব কন্টেইনারকে রিমুভ করে দিবে।

```

$ docker service rm hello-swarm
hello-swarm

$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES

$ docker container ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES

$

```

ফাইনালি সোয়ার্ম থেকে বের হতে চাইলে এই কমান্ড দিন (যদি ম্যানেজার না হন)  
 docker swarm leave

আর ম্যানেজার হলেঃ

```
docker swarm leave --force
```

আজকে এই পর্যন্তই। আগামী পর্বে দেখাবো কি করে একাধিক হোস্টের মাঝে সোয়ামিং করবেন।  
এরপরের এবং সর্বশেষ পর্বে দেখাবো সোয়াম এর অল্টারনেটিভ গুগলের কুবারনেটিস নিয়ে।