Slide 1: Introduction to Docker

დღეს ვისაუბრებთ ერთ ერთ მნიშვნელოვან და ძლიერ ხელსაწყოზე, რომელსაც Docker ეწოდება. დოკერი პლატფორმაა რომელიც საშუალებას გვაძლევს შევკრათ ერთ პაკეტში ჩვენი აპლიკაცია და ის ყველაფერი რაზეც ჩვენი აპლიკაცია დამოკიდებულია. ასეთი კონტეინერი გვაძლევს საშუალებას დარწმუნებულები ვიყოთ, რომ ჩვენი აპლიკაცია ყველანაირ გარემოში ერთნაირად იმუშავებს.

Docker იყენებს ოპერაციული სისტენების დონის ვლირტუალიზაციას კონტეინერების იზოლაციისთვის, რის შედეგადაც თითოეულ კონტეინერში გაშვებულ აპლიკაციას უნიკალური, თავისი, გარემო აქვს.

ოპერაციული სისტემის დონეზე ვირტუალიზაცია aka OS-Level virtualization ტექნოლოგიაა რომელიც საშუალებას იძლევა მრავალი იზოლირებული ინსტანსის გაშვებისა ერთ ფიზიკურ მანქანაზე. ეს ინსტანსები, რომლებსაც კონტეინერებს ვუწოდებთ, იზიარებენ ერთ ოპერაციულ სისტემას კერნელს, თუმცა ერთმანეთისგან დამოუკიდებლად მოქმედებენ. ეს მიდგომა განხვავდება ტრადიციული ვირტუალური მანქანებისაგან, რომლებიც მოითხოვდნენ, რომ მთლიანი ვირტუალკა ამოწეულიყო თავის OS იანად.

Docker კიდევ ერთ მნიშვნელოვან პრობლემას აგვარებს, რომელიცაა „ჩემს მანქანაზე მუშაობს“. გამომდინარე იქიდან, რომ დოკერის კონტეინერი შეიცავს არა მარტო აპლიკაციას არამედ ყველა ამ აპლიკაციის დამოკიდებულებას, კონფიგურაციასა და ბიბლიოთეკას, თუ ჩვენს აპლიკაციას შევამოწმებთ დოკერში კონტეინერში გაშვების შემდეგ, შეგვეძლება ვთქვათ რომ იგი იმუშავებს სატესტო სერვერზეც და პროდზეც. ეს მიდგომა გვიხსნის გარემოს გამოწვეული პრობლემების მეტ წილს.

Docker ის ერთ ერთი მნიშვნელოვანი დადებითი თვისებაა ეფექტურობა. კონტეინერები სისტემის რესურსს უფრო ეფექტურად ყენებენ ვიდრე ტრადიციული ვირტუალკები და მათი რესტარტი/ჩართვა/გამორთვა წამებში შეიძლება. ეს საშუალებას გვაძლევს რომ დანერგვები შევასრულობთ უფრო სწრაფად და მოქნილად.

Docker ის მთავარი კომპონენტი არის Docker Engine, რომელიც კონტეინერებს ამუშავებს. აქვე გვაქვს Docker Hub რომელიც წარმოადგენს დიდ პაკეტების ბიბლიოთეკას. იგი დეველოპერებმა შეკრიბეს და უზრუნველჰყოფს რომ ყველა მსგავსი კონფიგურაცეიბით მუშაობს.

რომ შევაჯამოთ, Docker არამხოლოდ აპლიკაციის დანერგვას გვიმარტივებს, არამედ ასევე გარანტიას გვაძლევს რომ ჩვენი აპლიკაცია ნებისმიერ გარემოში ერთნაირად იმუშავებს.

ვინაიდან დოკერს უკვე გავეცანით და ვიცით ვინაა და რას წარმოადგენს, ახლა შეგვიძლია მის რამდენიმე კომპონენტს დეტალურად გავეცნოთ.

Docker Images and Containers:

https://www.techtarget.com/searchitoperations/definition/Docker-image

მოდი გავაგრძელოთ საუბარი დოკერის ორ მნიშვნელოვან კონცეპტზე: იმიჯები და კონტეინერები. მათი გაცნობა დაგვეხმარება დოკერის ზედა დონეზე გაგებაში.

დავიწყოთ დოკერ იმიჯებით. წარმოვიდგინოთ რომ იმიჯი არის ნახაზი, რომელზე წარმოდგენილი ინსტრუქციების მიხედვითაც იქმნება კონტეინერი. Docker image ები ინსტრუქციების კრებულივითაა რომელთა თანმიმდევრობით განხორციელების შედეგად კონტეინერს მივიღებთ. იგი შეიცავს როგორც აპლიკაციის კოდს ასევე სხვა აპლიკაციის დამოკიდებულებებს რომლებიც მის გასაშვებად საჭიროა, მაგალითად: ბიბლიოთეკებს, თულებს ფაილებს და ა.შ. ერთი იმიჯი შეგვიძლია რამდენჯერმე გავუშვათ და მივიღოთ რამდენიმე მუშა კონტეინერი. Image ები გათვლილად კეთდება ისე რომ იყოს ეფექტური და მრავალჯერადი მრავალი შრის გამოყენებით (multiple layer).

1. **შრეებიანი სტრუქტურა(Layered Structure):** თითოეული იმიჯი იქმნება რამდენიმე შრის ერთმანეთზე დალაგების შედეგად. თითოეული შრე ამატებს რამე ახალ ფაილებს ან ცვლის რაიმეს წინა შრიდან მოწოდებული ფაილებიდან. თითოეული ეს შრე იქმნება DockerFile ის ინსტრუქციის მიხედვით, რომელშიც შეიძლება გაწერილი იყოს რაიმე სოფტის ინსტალაცია, ფაილების კოპირება, გენერაცია და ა.შ.
2. **უცვლელი შრეები (Immutable Layers):** მას შემდეგ რაც შრის შექმნა დასრულდება მისი ცვლილება შეუძლებელია რადგან იგი read only ხდება.
3. **სისწრაფე და ეფექტურობა (Speed and Efficiency):** როცა დოკერი იმიჯს ბილდავს ქმნის მას მხოლოდ იმ შრეების განახლება სჭირდება რომლებიც წინა ბილდის შემდეგ შეიცვალა, დანარჩენები უცვლელი რჩება და შესაბამისად ზედმეტი დრო არ იხარჯება.
4. **ჩაწერადი შრე (Writable Container Layer):** როცა უკვე ჩვენს კონტეინერს გავუშვებთ დოკერი კიდევ ერთ შრეს ამატებს ზემოდან უკვე არსებულებს, რომლის მოდიფიკაციაც შესაძლებელია. ეს გულისხმობს იმას რომ ამ შრეში შეგვიძლია ჩავწეროთ სოფტი, შევცვალოთ ფაილები და ა.შ. ეს შრე კონტეინერის შრეა. იგი მხოლოდ ერთ კონტეინერს ეხება და სხვა კონტეინერებზე და თავად იმიჯზე გავლენა არ აქვს.