**<https://devblogs.microsoft.com/dotnet/introducing-dotnet-aspire-simplifying-cloud-native-development-with-dotnet-8/>**

[**https://learn.microsoft.com/en-us/dotnet/aspire/deployment/manifest-format**](https://learn.microsoft.com/en-us/dotnet/aspire/deployment/manifest-format)

**Slide 1**

გამარჯობა მეგობრებო, დღეს მინდა წარმოგიდგინოთ .NET Aspire სიახლე cloud dev ში. იგი მიზნად ისახავს ქლაუდ დეველოპმენტის პროცესის გაუმჯობესებას/ასწრაფებას.

**Slide 2**

.NET Aspire ახალი თულია მაიკროსოფტისგან, რომელიც ქლაუდ აპლიკაციების კეთებაში დაგვეხმარება. შეგვიძლია მას ისე შევხედოდ როგორც თულებისა და მიდგომების კომბინაციას, რომელიც ქლაუდ აპლიკაციების მარტივად კეთება/შენახვაში გვეხმარება.

როგორც უკვე აღვნიშნეთ Aspire მიზნად ისახავს ქლაუდ დეველოპმენტის დროს ხშირად წარმოქმნილი პრობლემების გადაჭრას. მისი ძირითადი ფოკუსია აპლიკაციები გადააქციოს უფრო სანდოდ, მარტივად მონიტორირებად და ადვილად დასაკონფიგურირებლად. Aspire ის მიდგომა ეხება იმ საერთო გამოწვევებს, რომლებსაც ვეჩეხებით ქლაუდ დევში.

Aspire ში ჩადებული იდეები და ფუნქციონალები, როგორც მისი შემქმნელები ამბობენ, მოდის მაიკროსოფტის დიდი პროექტების გამოცდილებიდან, რომლებსაც მილიონობით ადამიანი იყენებს. ესენია სერვისები, რომლებიც უნდა იყვნენ სანდო და რომლებსაც უნდა სეეძლოთ დიდი ლოადის ატანა. გამომდინარე აქედან, როგორც თავად გვეუბნებიან, .NET Aspire შექმნილია პრაქტიკაზე დაფუძვნებით რეალური ჩელენჯების გასამკლავებლად მზად.

რომ შევაჯამოთ, Aspire გვაძლევს ხელსაწყოებისა და პრაქტიკების ნაზავს, რომელიც გამოცდილია დატვირთულ და მოთხოვნად გარემოებში.

**Slide 3**

გავეცნოთ ძირითად ტექნოლოგიებს რომლებიც .NET Aspire ში გამოიყენება.

დავიწყოთ Health Check ებით, YARP ით, HTTP Client Factory ითა და GRPC ით. ეს ტექნოლოგიები ძირითადია. Health Check ები ამონიტორინგებენ აპლიკაციების სიცოცხლისუნარიანობას. YARP (Yet Another Reverse Proxy) გვეხმარება ტრაფიკის განაწილებაში რომ სწორმა აპლიკაციამ მიიღოს რექვესთი. HTTP Client Factoy ამარტივებს http client ებთან მუშაობას და მათ მენეჯმენტს. საბოლოოდ gRPC რომელიც მნიშვნელოვანია სერვისებს შორის ინფორმაციის სწრაფი მიმოცვლისათვის.

სიაში შემდეგი გვყავს AOT (Ahead Of Time Compilation). თუ ამის შესახებ უკვე არ გსმენიათ, მარტივად რომ ვთქვათ AOT საშუალებას გვაძლევს ჩვენი აპლიკაციები გავხადოთ უფრო სწრაფები და შევამციროთ მათი ზომა. ეს შესაძლებელი ხდება აპლიკაციის პირდაპირ ბაინერიზე დამუშავებით, სანამ იგი გაეშვება, რათა რანთაიმში დრო არ დაიხარჯოს low level კოდიდან ბაინარიზე თარგმანში.

საბოლოოდ ასევე აღსანიშნავია SDK შემცველი ბილდები, რომლებიც გვიმარტივებენ deployment ის ამბავს.

**Slide 4**

ვისაუბროთ .NET Aspire ის შესახებ. იგი შეიცავს საკმაო რაოდენობის გამოსადეგ ხელსაწყოებს და კომპონენტებს. რეალურად Aspire ხომ ინტეგრაციების გასამარტივებლად შეიქმნა. იგი გვაწვდის გამზადებულ თულებს ქლაუდ დეველოპმენტისთვის.

.NET Aspire ში ჩაშენებული ფუნქციონალები შეიცავს Service Discovery ს, რომელიც ეხმარება ჩვენს აპლიკცაიებს მარტივ კომუნიკაციაში, და Telemetry ს, რომელიც გვეხმარება აპლიკაციების მონიტორინგში. Aspire ში ასევე ყურადღება გამახვილებულია ამტანობაზე ?? და აპლიკაციის მდგრადობაზე რათა 24/7 სერვისი მივიღოთ.

აქვე ყურადსაღები საკითხია რომ Aspire ცალკე გამოშვებული ფუნქციონალი არაა. იგი .NET 8 ის ნაწილია და მასთან ერთად განვითარებას განაგრძობს.

**Slide 5**

მოდი გავეცნოთ Aspire კომპონენტებს. კომპონენტები Lego სავითაა Aspire ში ჩამატებული და შეგვიძლია ნებისმიერი საჭირო კომპონენტი მარტივად დავამატოთ ჩვენს აპლიკაციაში.

მაგალითად Aspire ის ერთ-ერთი კომპონენტია Aspire.StackExchane.Redis რედისის ქეში. ჩვენ შეგვიძლია დავამატოთ საჭირო ბიბლიოთეკები Aspire პროექტში და თვალის დახამხამებაში გვექნება გამართული Redis. შეგვიძლია დავამატოთ როგორც უბრალოდ ბიბლიოთეკა არსებულ რედისის სერვერთან სამუშაოდ, ასევე კონტეინერი რომელიც სრულიად ახალ კონტეინერს გადმოწერს და გაუშვებს აპლიკაციასთან ერთად (საჭიროა გვეყენოს დოკერი). ეს და სხვა კომპონენტებიც, რომლებიც Aspire თვის შეიქმნა არიან წინასწარ დაკონფიგურირებულები ქლაუდისთვის. ანუ თავის ატკიება აღარ მოგვიწევს ცალკე რედისის სწორად კონფიგურაციზე, ცალკე SQL Server ის თუ სხვა კომპონენტებში მოხვედრილი ინფრასტრუქტურის ქლაუდზე გასაშვებად.