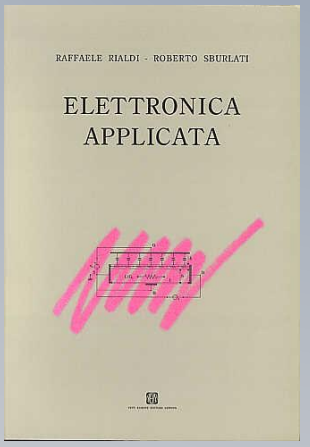
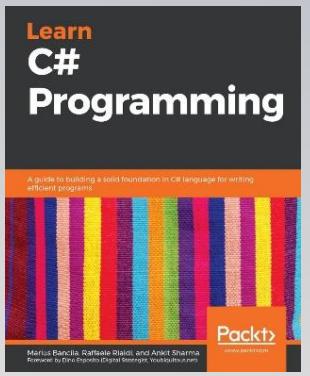
Orchestrare applicazioni distribuite con .NET Aspire 9

Ing. Raffaele Rialdi
Senior Software Architect - Consultant
@raffaeler - raffaeler@vevy.com

Chi sono

- Laurea Master in Ingegneria Elettronica (Unige)
- Lavoro professionalmente nel software dal 1987
- Insegno saltuariamente a Ingegneria Informatica (Unige)
- Membro della commissione ICT dell'Ordine degli Ingegneri
- Microsoft Most Valuable Professional per 21 anni consecutivi
- Libero professionista, Software Architect in diversi ambiti:
  - Financial, Manufacturing, Healthcare, F1 racing, ...
- Speaker in conferenze nel mondo (più di 200 interventi in 20 anni)
  - Europa, Asia, USA
- Co-Autore del libro "Elettronica Applicata"
- Co-Autore del libro "C# Programming"
- Presidente di DotNetLiguria a Genova





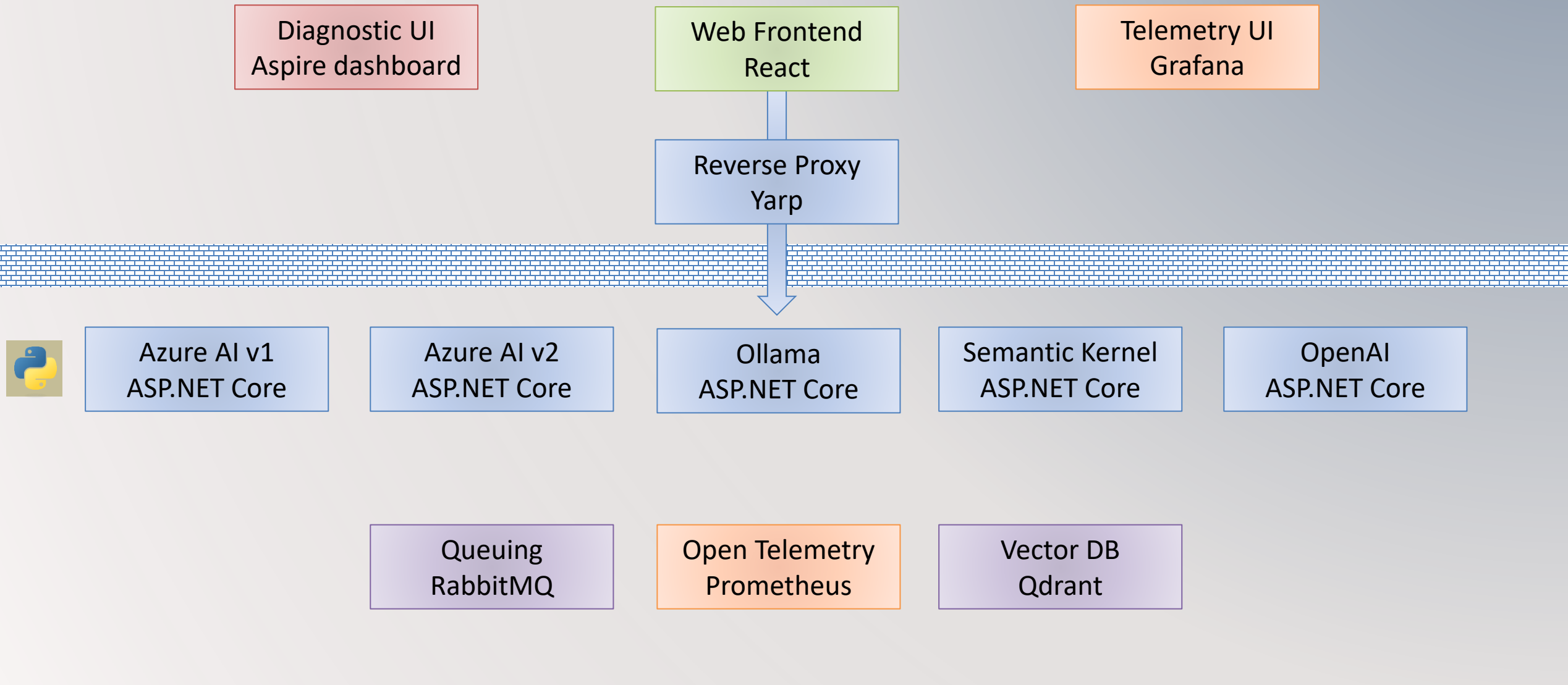
Perché Aspire?

- Nella preistoria informatica, c'era l'applicazione monolitica
- Il passaggio distribuite è doloroso:
  - indirizzi e porte, configurazioni, secrets, variabili di environment, etc.
- Il container è oggi l'unità di deploy ed ha consolidato questioni dolorose
  - Virtualizzazione di network e storage
- Le architetture a microservizi amplifica questi problemi
  - la necessità di documentare la relazione tra microservizi
  - i manifesti di docker compose e kubernetes sono un esempio di come rappresentare i requisiti
- Aspire consente di esprimere questi requisiti per ottenere un ambiente di esecuzione consistente durante lo sviluppo.

Quando è utile Aspire?

- Aspire è utile nel cosiddetto "Inner Loop":
  - Scrivere codice, compilare, debuggare
- Nelle applicazioni distribuite il problema è la frammentazione di processi
  - Progetti parte della stessa Solution
  - Progetti realizzati esternamente o in linguaggi diversi
  - Container
  - Eseguibili esterni
- E di conseguenza la comunicazione tra processi
  - Indirizzi e porte
  - Variabili di Environment (secrets ed altro)
  - Altre configurazioni.

Esempio: un custom Copilot



Il glossario di Aspire

- Resource
  - L'elemento orchestrato (progetto, container, eseguibile, database, servizio, ...)
- Application Model
  - Tutte le risorse che realizzano l'applicazione distribuita
- Integration
  - Un pacchetto NuGet che contiene il codice che integra una Resource nel sistema
  - Esistono integrazioni client e server
- Reference
  - La connessione tra le Resource, espresse sotto forma di dipendenza
- Application Host (orchestratore)
  - L'applicazione che governa tutto il sistema
  - Qui sono definite le relazioni tra le Resource.

Impatto di Aspire nei progetti esistenti

- Non aggiunge dipendenze di Runtime
- Non produce alcun binario
  - Si avvale di un eseguibile pre-compilato per l'orchestrazione
- Abilita alcune funzionalità di runtime ai progetti configurati con Aspire
  - Es: Open Telemetry, Resiliency, HealthCheck, ...
- Configura la Discovery e la configurazione delle dipendenze tra risorse
  - Es: indirizzi di rete, porte, variabili di Environment, etc.
- Semplifica la creazione e la configurazione dei container
  - Uso di immagini o creazione tramite Dockerfile
- Fornisce una dashboard per la diagnostica durante lo sviluppo
- Può esportare un manifesto utile alla pubblicazione (deploy).

L'ecosistema di Aspire: le integrazioni

- Sia Microsoft che terze parti stanno creando integrazioni pronte all'uso
- Sono tutte distribuite come pacchetti NuGet (prefisso Aspire.Hosting.)
  - SQL server, Redis, RabbitMQ, Keycloak, Qdrant, Orleans, Azure, Azure.Storage, NodeJs, PostgreSQL, Dapr, MongoDB, Kafka, MySQL, AWS, Python, Azure.Search, Azure.EventHubs, Azure.Functions, ElasticSearch, Oracle, ...
- Esistono due tipi di integrazioni:
  - Estensioni che configurano la risorsa "server" nell'AppHost
    - Si trovano su NuGet con il filtro: «owner: aspire tags: aspire hosting integration»
  - Estensioni client usate per accedere alla controparte server
    - Si trovano su NuGet con il filtro: «owner: aspire tags: aspire client integration»
- Le integrazioni semplificano la preparazione
  - Telemetria (logging, tracing, metrics), resilienza, health-check, configurazioni.

Esempio: l'applicazione Copilot

La Resource a partire dal Dockerfile

```
var param1 = builder.AddParameter("envraf");
builder.AddDockerfile("pythoncontainer1", "../PythonContainer1", "Dockerfile")
        .WithOtlpExporter()
        .WithEndpoint(port: 8111, targetPort: 8111, scheme: "http", env: "PORT")
        .WithEnvironment("PORT", "8111")
        //.WithEnvironment("OTEL_EXPORTER_OTLP_ENDPOINT", "http://localhost:19192")
        .WithEnvironment("OTEL_EXPORTER_OTLP_ENDPOINT", "http://host.docker.internal:19192")
        //.WithEnvironment("OTEL_EXPORTER_OTLP_ENDPOINT", "https://host.docker.internal:21103")
        .WithEnvironment("OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED", "true")
        .WithEnvironment("arg1", param1)
    ;
```

La Resource a partire da un servizio Python

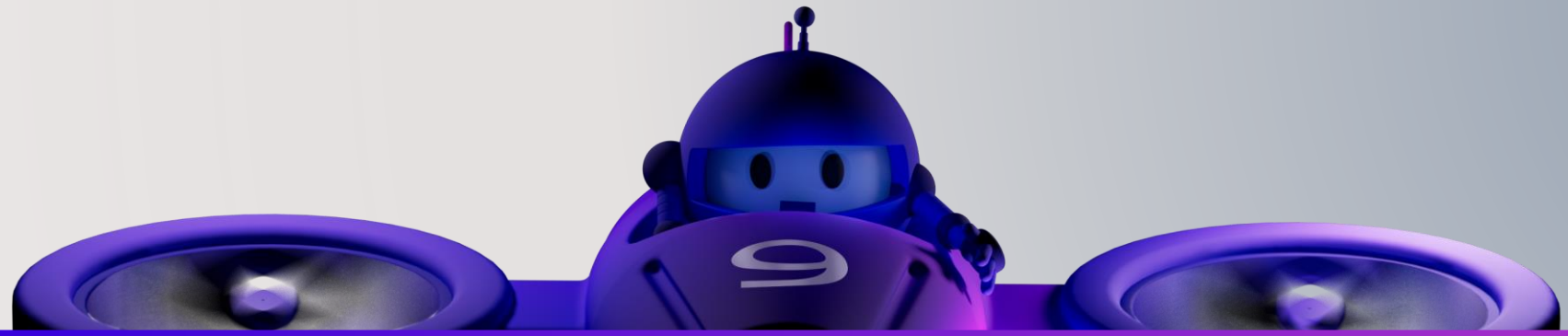
```
builder.AddPythonApp("Python", "../PythonService1", "hello_python.py")
       .WithEndpoint(targetPort: 8111, scheme: "http", env: "PORT", isProxied:false)
       .WithEnvironment("OTEL_EXPORTER_OTLP_ENDPOINT", "http://localhost:19192")
       .WithEnvironment("OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED", "true")
       .WithEnvironment("PORT", "8111");

builder.AddPythonApp("Python", "../PythonService1", "app2.py")
       .WithEndpoint(targetPort: 8111, scheme: "http", env: "PORT")
       .WithEnvironment("PORT", "8111")
       .WithOtlpExporter();

builder.AddExecutable("Python", "cmd", "../PythonContainer1", "/C", "run.cmd")
       .WithEndpoint(targetPort: 8111, scheme: "http", env: "PORT")
       .WithEnvironment("OTEL_EXPORTER_OTLP_ENDPOINT", "http://localhost:19192")
       .WithEnvironment("OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED", "true");
```

Ecosistema attuale

- Repository ufficiale:
  - https://github.com/dotnet/aspire
- Esempi ufficiali:
  - https://github.com/dotnet/aspire-samples
- .NET Aspire Community Toolkit
  - https://learn.microsoft.com/en-us/dotnet/aspire/community-toolkit/overview
  - https://github.com/CommunityToolkit/Aspire
- Aspir8: il tool per il deploy della soluzione
  - https://aspireify.net/a/240520/aspir8:-aspire-to-deployments
  - https://github.com/prom3theu5/aspirational-manifests
  - https://github.com/devkimchi/aspir8-from-scratch

Ambizioni di Aspire (potenziali, ancora da discutere)

- Dashboard sempre attiva con possibilità di hot reload, attach del debugger, etc.
- Forte supporto per mixare processi sviluppati con linguaggi diversi
- Supporto a TLS e dominio .local per lo sviluppo locale
- Lanciare i progetti in container per simulare l'ambiente di deploy
- Supporto semplificato per reverse proxy e configurazione di rete
- Semplificare la distribuzione di servizi tra Teams diversi
- Sviluppo remoto con DevBox e GitHub Codespaces
- CLI specifica di Aspire
- Supporto al Deploy di altri target oltre a Kubernetes e Docker Compose
- Condivisione di setting e secrets nei team di sviluppo
- Totale trasparenza tra sviluppo monolitico e distribuito (in termini di effort/strumenti).



Get .NET 9

Download .NET 9
aka.ms/get-dotnet-9

Compilate i Feedback
Grazie!