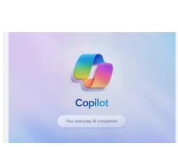
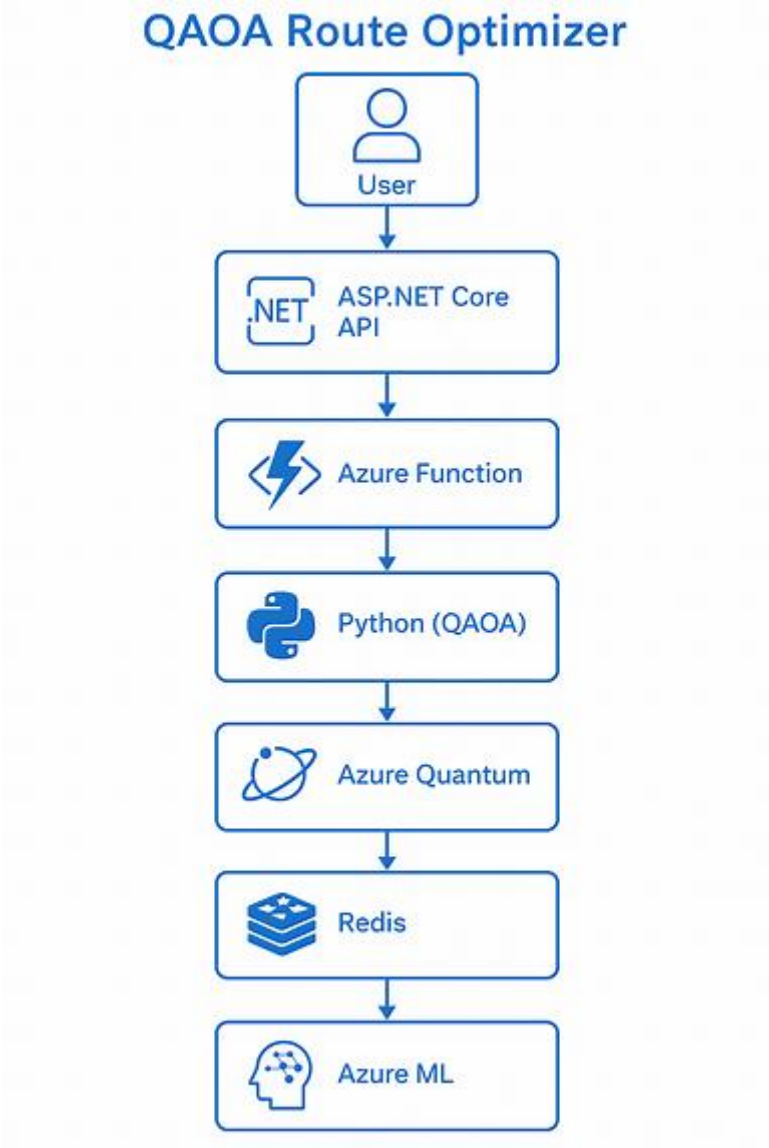


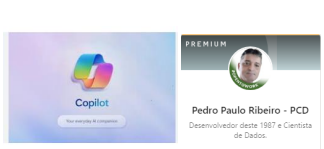
Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/

WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform





Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🚚 Complete Project: Quantum Route Optimizer (QAOA)

This chapter presents a real and complete project that uses quantum computing to solve a classic optimization problem: **finding the best route between multiple cities** , similar to the famous **Traveling Salesman Problem (TSP)** .

We will use the **QAOA (Quantum Approximate Optimization Algorithm)** , running on **Azure Quantum** , orchestrated by **Azure Functions** , with persistence in **Cosmos DB** , caching in **Redis** , AI for interpretation, and a corporate API in **.NET** .

🎯 9.1 Project Objective

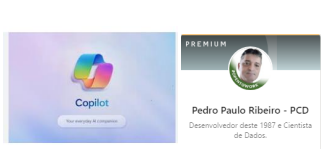
Create a system capable of:

- Receive a list of cities and distances.
- Building a route graph
- Generating a QAOA quantum circuit
- Running the circuit in Azure Quantum
- Storing results in Cosmos DB
- Using Redis for caching
- Using AI to interpret the best route
- Return the optimized route via the .NET API.

📦 9.2 Complete Project Architecture

Code

```
User → .NET API → Azure Function → Python (QAOA)
↓
Azure Quantum
↓
Cosmos DB
```




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
↓
Redis
↓
Azure ML (AI)
↓
.NET API
```

🔷 Components used

- **Python** → QAOA Algorithm
- **Azure Quantum** → Quantum Execution
- **Azure Functions** → Serverless Orchestration
- **Cosmos DB** → Persistence
- **Redis** → Cache
- **Azure ML** → Interpreting Results
- **.NET** → Enterprise API

💠 9.3 Problem Modeling (TSP)

The TSP consists of:

- A group of cities
- Distances between them
- Find the shortest route that visits all the cities at once.

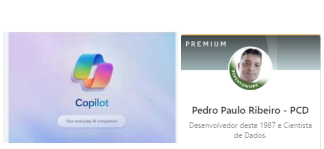
🔷 Graph representation

Each city is a node. Each route is a weighted edge (distance).

🔷 Quantum representation

QAOA transforms the graph into:

- Hamiltonian of cost
- Mixture Hamiltonian
- Parameterized circuit



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

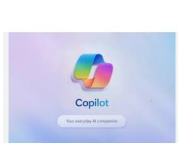
Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧪 9.4 QAOA Implementation in Python

Here is a simplified example of the QAOA circuit:

python

```
from qiskit import QuantumCircuit
from qiskit_optimization import QuadraticProgram
from qiskit_optimization.algorithms import MinimumEigenOptimizer
from qiskit.algorithms import QAOA
from azure.quantum.qiskit import AzureQuantumProvider

# Connecting to Azure Quantum
provider = AzureQuantumProvider(
resource_id="xxxx",
location="eastus"
)

# Create TSP template
qp = QuadraticProgram()
# ... add variables and constraints ...

# Configure QAOA
qaoa = QAOA(reps=2)
optimizer = MinimumEigenOptimizer(qaoa)

# Resolver
result = optimizer.solve(qp)

print(result)
```




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

☁ 9.5 Azure Function para Orquestrar o Job Quântico

📄 Exemplo em C#

csharp

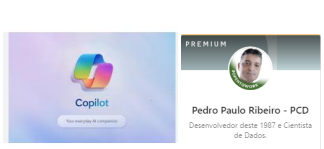
```
[FunctionName("RunQuantumTSP")]
public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Function,    "post")]    HttpRequest
req,
    ILogger log)
{
    var psi = new ProcessStartInfo("python", "qaoa_tsp.py");
    psi.RedirectStandardOutput = true;

    var process = Process.Start(psi);
string output = process.StandardOutput.ReadToEnd();

return new OkObjectResult(output);
}
```

🗄 9.6 Saving Results in Cosmos DB

📄 Example in C#

csharp

```
await container.CreateItemAsync(new {
id = Guid.NewGuid().ToString(),
type = "quantum-result",
algorithm = "QAOA",
result = output,
timestamp = DateTime.UtcNow
});
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

⚡ 9.7 Results Cache with Redis

📄 Python Example

python
```
import redis

r = redis.Redis(host="localhost", port=6379)
r.set("last_tsp_result", result)
```

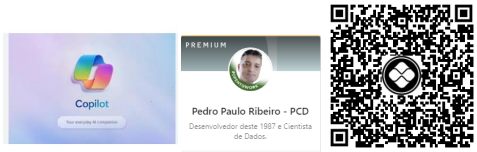
🧠 9.8 Interpretation with AI (Azure ML)

AI can:

- Interpreting quantum probabilities
- Convert bitstrings to routes
- Assess total cost
- Choosing the best route

📄 Example of interpretation

python
```
def interpret(bitstring):
# Converts bitstring to route
# Ex.: "011001" → [City 2, City 3, City 1]
Pass
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

9.9 .NET API for Exposing the Final Result

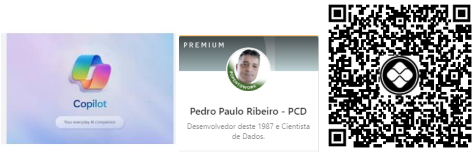
Example in C#

csharp

```
[HttpGet("tsp/optimized")]
public async Task<IActionResult> GetOptimizedRoute()
{
    var cache = redis.GetDatabase();
    var cached = cache.StringGet("last_tsp_result");

    if (!cached.IsNullOrEmpty)
        return Ok(cached);

    var query = container.GetItemQueryIterator<dynamic>(
        new QueryDefinition("SELECT TOP 1 * FROM c ORDER BY
c.timestamp DESC")
    );

    var result = (await query.ReadNextAsync()).First();
    return Ok(result);
}
```

9.10 Fluxo Completo do Projeto

1. Usuário envia cidades



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🟪 **2. API .NET valida**

🟩 **3. Azure Function chama Python**

🟧 **4. Python generates QAOA circuit**

🟥 **5. Azure Quantum runs**

🟨 **6. Cosmos DB stores**

🟫 **7. Redis guarda cache**

🟦 **8. Azure ML interprets**

🟪 **9. .NET API returns optimized route**

🧠 9.11 Best Practices for Quantum Projects

✔ **Use simulators before using real hardware.**

✔ **Store everything in Cosmos DB**

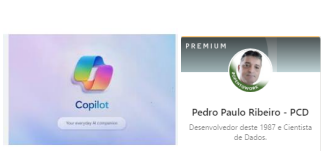
✔ **Use Redis to speed up queries.**

✔ **Separate responsibilities by language.**

✔ **Use Azure Functions for orchestration**

✔ **Use AI to interpret quantum results**

✔ **Monitor everything with Application Insights**



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Summary

In this chapter, you learned:

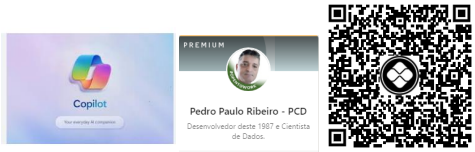
- How to build a complete project using quantum computing.
- How to apply QAOA to the routing problem
- How to integrate Python, .NET, Azure Functions, and Azure Quantum
- How to store results in Cosmos DB
- How to use Redis for caching
- How to use AI to interpret results
- How to expose everything via a corporate API.

This chapter concludes the book, showing how all the components come together to form a real and functional solution.

Real-time deployment and implementation of the complete project: Quantum Route Optimizer (QAOA) in AZURE QUANTUM IA

In the most advanced and strategic part **of the entire book** : **how to deploy and put into production, in real time, the complete Quantum Route Optimizer (QAOA) project using Azure Quantum + AI + Python + .NET + Functions + Cosmos DB + Redis** .

I prepared this chapter as a **professional deployment guide** , in the style of Microsoft cloud architects, covering:



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Infrastructure
- CI/CD Pipelines
- Automated deployment
- Monitoring
- Security
- Real-time execution
- High availability

We're going to transform the project into a production-ready, **enterprise quantum service .**

🚀 Real-time Deployment of the Complete Project: Quantum Route Optimizer (QAOA) on Azure Quantum AI

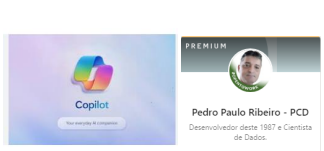
Deploying a hybrid quantum system requires a robust architecture, automated pipelines, and integration between multiple services. Here you will learn how to put the complete project into production, with **continuous deployment** , **real-time execution** , and **enterprise monitoring** .

🛞 1. Final Production Architecture

The production architecture is as follows:

Code

```
User → .NET API (App Service)
↓
Azure API Management
↓
Azure Functions (Orchestration)
↓
Python + QAOA (Container)
↓
Azure Quantum (Execution)
↓
Cosmos DB (Persistence)
↓
Redis (Cache)
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
↓
Azure ML (Interpretation)
↓
.NET API (Final Response)
```

◆ Deployed components

Component	Azure Service	Function
.NET API	App Service	Corporate entrance
Python QAOA	Container Apps	Quantum execution
Azure Functions	Premium Functions	Orchestration
Azure Quantum	Workspace	Real quantum execution
Cosmos DB	Cosmos DB SQL	Persistence
Redis	Azure Cache for Redis	Cache
AI	Azure ML	Interpretation
Security	API Management + Key Vault	Protection

📦 2. Preparing the Infrastructure in Azure

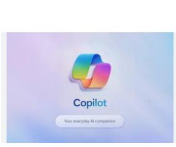
🟦 2.1 Create the Azure Quantum Workspace

In the Azure portal:

1. Create resource → **Azure Quantum**
2. Select providers (IonQ, Quantinuum, Rigetti)
3. Create credentials
4. Enable integration with Python and Qiskit

🟪 2.2 Creating Cosmos DB

- API: **Core SQL**
- Database: `quantumdb`
- Container: `results`
- Partition key: `/provider`



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🟥 2.3 Creating Redis

- Service: **Azure Cache for Redis**
- Tier: Standard or Premium
- Enable TLS
- Create access key

🟩 2.4 Create Azure ML Workspace

- Create workspace
- Create a compute cluster
- Create an environment using Python + Qiskit + Azure Quantum SDK

🟦 2.5 Create Azure Functions

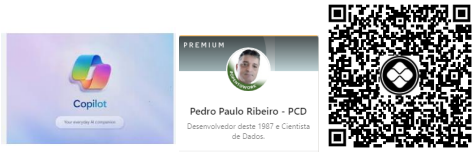
- Plan: **Premium** (for long-term execution)
- Runtime: .NET 8
- Enable VNET Integration
- Configure Key Vault

🟨 2.6 Creating a .NET API in App Service

- Runtime: .NET 8
- Deploy via GitHub Actions
- Integrate with API Management

🐳 3. Deploying Python QAOA in Containers

The Python module that runs QAOA must run in isolation within a container.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

3.1 Dockerfile

dockerfile

```
FROM python:3.11

RUN pip install qiskit azure-quantum redis numpy

COPY . /app
WORKDIR /app

CMD ["python", "qaoa_tsp.py"]
```

3.2 Publish to Azure Container Registry

Code

```
az acr build --registry myACR --image qaoa:latest.
```

3.3 Deploy to Azure Container Apps

Code

```
az containerapp create \
--name qaoa-runner \
--image meuACR.azurecr.io/qaoa:latest \
--my-environment \
--cpu 2 --memory 4Gi
```

4. Azure Function Deployment (Orchestration)

The Function calls the Python container and sends the job to Azure Quantum.

4.1 Setting environment variables

- `QUANTUM_WORKSPACE_ID`

- `REDIS_CONNECTION`
- `COSMOS_CONNECTION`
- `CONTAINERAPP_URL`

4.2 Deploy via GitHub Actions

Arquivo `.github/workflows/function-deploy.yml`:

yaml

```yaml
name: Deploy Functions

on:
  push:
    branches: [ "main" ]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: Azure/functions-action@v1
        with:
          app-name: "QuantumFunctions"
          package: "."
```

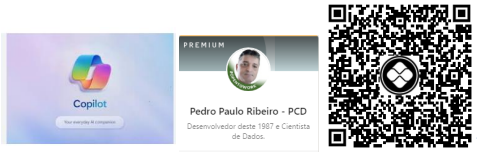
5. Deploying the .NET API

5.1 Publish to App Service

Via GitHub Shares:

yaml

```yaml
- uses: azure/webapps-deploy@v2
with:
app-name: "QuantumAPI"
package: "./publish"
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

6. Deploying AI on Azure ML

6.1 Create an interpretation pipeline

- Python script that converts bitstrings into routes.
- ML model that evaluates cost
- REST endpoint exposed by Azure ML

7. Security and Governance

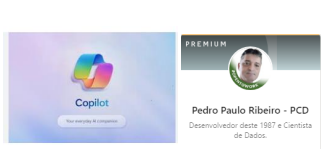
7.1 Key Vault

Store:

- Redis Keys
- Cosmos DB Connection
- Azure Quantum credentials
- API Secrets

7.2 API Management

- Rate limiting
- Authentication
- Logging
- Corporate security



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📡 8. Real-Time Execution

The workflow in production works like this:

🟦 **1. User submits cities → .NET API**

🟪 **2. API calls Azure Function**

🟩 **3. Function calls Python container**

🟧 **4. Python generates QAOA circuit**

🟥 **5. Azure Quantum runs**

🟨 **6. Cosmos DB saves**

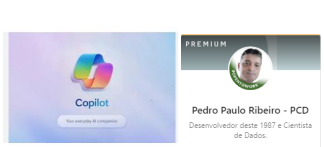
🟫 **7. Redis guarda cache**

🟦 **8. Azure ML interprets**

🟪 **9. .NET API returns optimized route**

Average time:

- QAOA simulator: 1–3 seconds
- QAOA real hardware: 5–30 seconds
- Complete pipeline: 8–40 seconds



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📊 9. Monitoring and Observability

Use:

- **Application Insights**
- **Log Analytics**
- **Azure Monitor**
- **Real-time dashboard**

Monitor:

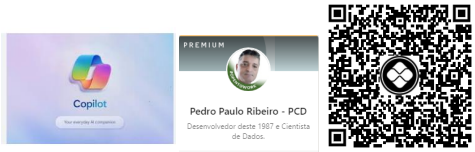
- Latency of quantum jobs
- Execution errors
- RU Consumption in Cosmos DB
- Cache hits in Redis
- API Performance

🎯 10. Final Result

After deployment, you will have:

- A **corporate quantum service**
- Running QAOA in real time
- With AI interpreting results
- Serving users with a .NET API
- With Functions orchestrating
- With Cosmos DB persisting
- With Redis accelerating
- Optimizing with Azure ML
- With Azure Quantum running on real hardware

It is literally a **complete quantum platform in production** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Here is a theoretical guide:

📓 Quantum Computing and Hybrid Architectures in the Cloud with Azure Quantum, .NET, Python, AI, Cosmos DB, Redis, and Azure Functions

🎡 Preface

Modern computing is entering a new era. The combination of **quantum computing** , **artificial intelligence** , **serverless computing** , **distributed databases** , and **hybrid languages like Python and .NET** creates a powerful ecosystem for solving previously impossible problems.

This book was written for professionals who want to master this technological convergence and build real solutions using the Microsoft Azure ecosystem.

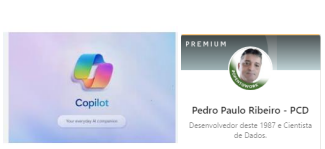
💠 Chapter 1 — Fundamentals of Quantum Computing

⚛️ 1.1 What is Quantum Computing?

- Difference Between Bits and Qubits
- Overlap
- Entanglement
- Interference
- Measurement

🌀 1.2 Types of Quantum Computers

- Gate-based
- Quantum Annealing
- Topological



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧪 1.3 Problems that Quantum Computing Solves

- Optimization
- Molecular simulation
- Post-quantum cryptography
- Quantum Machine Learning

☁ Chapter 2 — Azure Quantum

◈ 2.1 What is Azure Quantum?

- Unified platform
- Support for multiple providers (IonQ, Quantinuum, Rigetti)
- Local and cloud-based simulators

🧰 2.2 Languages and SDKs

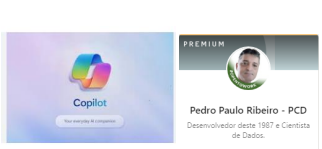
- Q#
- Python
- .NET

🧪 2.3 Creating your first quantum program

Simple example in Q#:

qsharp

```
namespace Quantum.Example {
open Microsoft.Quantum.Intrinsic;
open Microsoft.Quantum.Canon;

operation HelloQubit() : Result {
using (q = Qubit()) {
H(q);
let result = M(q);
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
Reset(q);
return result;
}
}
}
```

🚀 2.4 Running on Azure Quantum

- Creating a Workspace
- Submitting Jobs
- Monitoring Executions

🧠 Chapter 3 — Artificial Intelligence Integrated with Quantum Computing

🤖 3.1 Quantum Machine Learning

- QML
- VQE
- QAOA

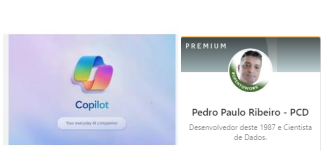
🔗 3.2 Hybrid: Classical AI + Quantum AI

- Classical preprocessing
- Quantum optimization
- Post-processing with neural networks

🪐 3.3 Example in Python with Qiskit + Azure Quantum

python

```
from qiskit import QuantumCircuit
from azure.quantum import Workspace

ws = Workspace(
subscription_id="xxxx",
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
resource_group="xxxx",
name="quantum-ws",
location="eastus"
)

qc = QuantumCircuit(1,1)
qc.h(0)
qc.measure(0,0)

job = ws.submit(qc)
print(job.result())
```

📦 Chapter 4 — Azure Functions: Serverless Computing

⚙️ 4.1 What is Serverless?

- Automatic scalability
- Payment for execution
- Ideal for quantum pipelines.

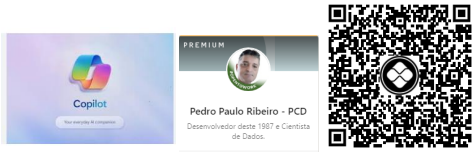
🧩 4.2 Creating a Function in .NET

csharp

```
public static class QuantumTrigger {
    [FunctionName("QuantumTrigger")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function,  "get")]  HttpRequest
req,
        ILogger log)
    {
        log.LogInformation("Executando job quântico...");
        // Chamada ao Azure Quantum aqui
        return new OkObjectResult("Job enviado!");
    }
}
```

🔗 4.3 Orquestrando Jobs Quânticos com Functions

- Triggers HTTP
- Timers



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Event Grid

🪐 Chapter 5 — Cosmos DB: Globally Distributed Database

🌍 5.1 Why Cosmos DB

- Low global latency
- Multi-model
- Elastic scalability

🗃️ 5.2 Storing quantum results

- JSON
- Collections
- Partitioning

📃 5.3 Example in .NET

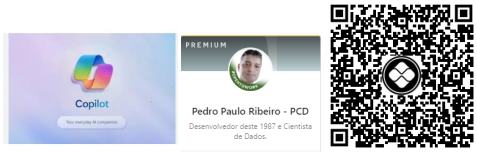
csharp

```
var client = new CosmosClient(connectionString);
var container = client.GetContainer("quantumdb", "results");

await container.CreateItemAsync(new {
id = Guid.NewGuid().ToString(),
jobId = "123",
result = "Zero"
});
```

⚡ Chapter 6 — Redis: Cache and Messaging for Quantum Systems

🚀 6.1 Why Redis

- Low latency
- Pub/Sub
- Results cache



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🔧 6.2 Example in Python

python

```
import redis

r = redis.Redis(host='localhost', port=6379)
r.set("last_quantum_result", "1")
print(r.get("last_quantum_result"))
```

🧬 Chapter 7 — Python and .NET Working Together

🔄 7.1 Interoperability

- Python for quantum algorithms
- .NET for APIs, Functions, and Integrations

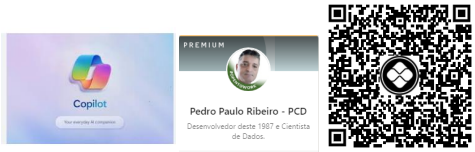
📦 7.2 Using Python within .NET

- Python.NET
- External processes
- Containers

🏗️ Chapter 8 — Complete Architecture of a Quantum System in Azure

☸️ 8.1 Components

- Azure Quantum
- Azure Functions



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Cosmos DB
- Redis
- Python
- .NET
- AI

🗺️ 8.2 Full Flow

1. User sends request
2. Azure Function triggers quantum job.
3. Python executes circuit
4. Result stored in Cosmos DB
5. Redis guards cache
6. AI interprets results.
7. The .NET API returns a response.

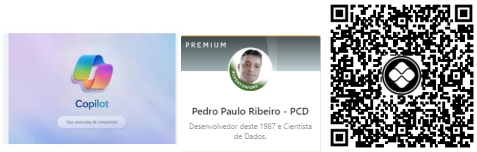
🏁 Conclusion

The integration of quantum computing, AI, Python, .NET, and Azure services creates a powerful ecosystem for solving complex problems. This book provides the foundation for building real-world, scalable solutions.

⚛️ Chapter 1 — Fundamentals of Quantum Computing

Quantum computing represents a radical shift in how we process information. While classical computers have evolved over decades, increasing speed and parallelism, they remain limited by traditional physical principles. Quantum computers, on the other hand, exploit phenomena of quantum mechanics to perform calculations that are impossible—or impractical—for classical machines.

This chapter lays the necessary foundation for understanding the rest of the book.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/

WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ 1.1 What is Quantum Computing?

Quantum computing is an information processing paradigm based on the laws of quantum mechanics. Instead of traditional bits, it uses **qubits** , which can exist in multiple states simultaneously.

📌 Difference between Bit and Qubit

Bit (Classic)	**Qubit (Quantum)**
It can be 0 **or** 1.	It can be 0, 1 **, or both at the same time.**
Deterministic	Probabilistic
Linear processing	Exponential processing
Based on classic electronics	Based on quantum phenomena

🌀 1.2 Fundamental Principles of Quantum Mechanics

Quantum computing is based on four main pillars:

✨ Overlap

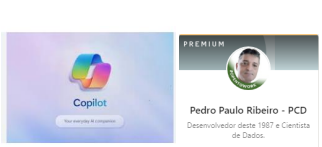
A qubit can be in a combination of 0 and 1 simultaneously. This allows exploring multiple computational paths at the same time.

🧬 Entanglement

Two qubits can become correlated in such a way that the state of one depends on the other, even when physically separated. This is the basis for powerful quantum algorithms.

🌊 Interference

Quantum states can reinforce or cancel each other out, allowing correct responses to be amplified and incorrect ones eliminated.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📏 Measurement

When measuring a qubit, it "collapses" to 0 or 1. The art of quantum computing is to manipulate states before measurement to maximize the probability of obtaining the desired answer.

💠 1.3 Types of Quantum Computers

There are different technological approaches to building quantum computers:

📦 Gate-Based (Quantum Logic Gates)

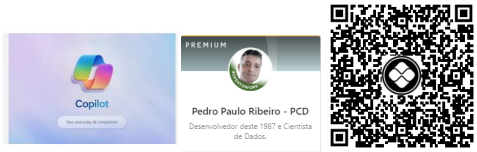
- A model closest to classical computing.
- It uses ports such as H, X, CNOT.
- Ideal for algorithms like Shor and Grover.
- Used by Azure Quantum, IBM, Google, IonQ, Quantinuum

🔄 Quantum Annealing

- Focused on optimization problems
- Based on minimum energy
- Used by D-Wave

🌀 Topological

- Extremely noise resistant
- Still under development.
- Promise of more stable qubits



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧪 1.4 Important Quantum Algorithms

🔐 Shor's algorithm

It solves factorization of large numbers exponentially faster than classical computers. Direct impact on cryptography.

🔍 Grover's Algorithm

Accelerates unstructured searches. Offers quadratic gain.

🧬 VQE (Variational Quantum Eigensolver)

Hybrid: part classical + part quantum. Used for chemistry, energy, and materials.

🧮 QAOA (Quantum Approximate Optimization Algorithm)

It solves combinatorial optimization problems. Widely used in logistics and finance.

🧠 1.5 Why Quantum Computing is Important

🚀 Exponential acceleration

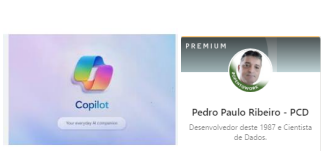
Some problems simply cannot be solved by classical computers in a timely manner.

🧬 Simulation of natural systems

Molecules, proteins, materials — all of these are inherently quantum.

🔐 Post-quantum cryptography

Quantum computing challenges current security models.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Quantum AI

Hybrid models can accelerate training and optimization.

1.6 Current Limitations

Despite its potential, quantum computing still faces challenges:

Noise and incoherence

Qubits are extremely sensitive to their environment.

Extreme temperatures

Many quantum computers operate close to absolute zero.

Limited number of useful qubits

We are still in the NISQ (Noisy Intermediate-Scale Quantum) era.

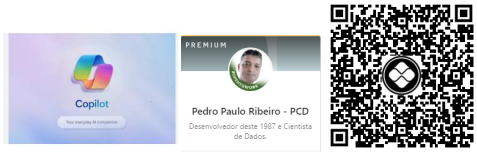
The need for hybrid algorithms

Current quantum computing works best when combined with classical computing.

1.7 The Future of Quantum Computing

Evolution points to:

- More stable qubits
- Hybrid architectures (classical + quantum)
- Quantum computing as a service (QaaS)
- Integration with AI and Big Data
- Real-world commercial applications in logistics, chemicals, finance, and energy.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧪 1.8 Simple Example of a Quantum Circuit

Here is a basic circuit that puts a qubit into superposition:

python

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(1,1)
qc.h(0) # Hadamard Gate → creates overlap
qc.measure(0,0)

print(qc)
```

This is the first step in understanding how quantum algorithms are built.

🎯 Chapter Summary

In this chapter, you learned:

- What is quantum computing?
- Difference Between Bits and Qubits
- Fundamental principles of quantum mechanics
- Types of quantum computers
- Main algorithms
- Current limitations
- The future of the area

This knowledge will be essential for understanding how **Azure Quantum** , **Python** , **.NET** , **AI** , **Cosmos DB** , **Redis,** and **Azure Functions** connect in the following chapters.

☁ Chapter 2 — Azure Quantum: Microsoft's Quantum Platform

Azure Quantum is the gateway to modern quantum computing within the Microsoft ecosystem. It offers a unified platform that integrates **quantum hardware providers** ,

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

simulators , **programming languages** , **development tools** , and enables the creation of **hybrid applications** that combine classical and quantum computing.

This chapter presents everything you need to know to start developing real quantum solutions using Azure.

◈ 2.1 What is Azure Quantum?

Azure Quantum is a cloud service that enables:

- Running quantum algorithms on real hardware
- Using high-performance quantum simulators
- Creating hybrid applications (classical + quantum)
- Integrate Python, Q#, .NET and AI
- Working with multiple hardware providers

It functions as a **quantum hub** , connecting developers to different quantum technologies without needing to learn each platform individually.

🌐 Supported Providers

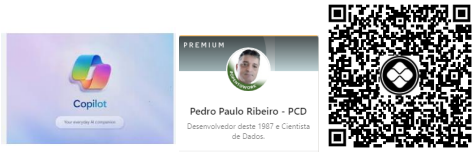
Currently, Azure Quantum offers access to:

- **IonQ** — trapped ion qubits
- **Quantinuum** — high-fidelity ion trap qubits
- **Rigetti** — superconducting qubits
- **Microsoft** — Q# simulators and resources

This diversity allows testing the same algorithm on different architectures.

🧰 2.2 Supported Languages and SDKs

Azure Quantum is flexible and supports multiple languages:



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🔵 Q# (Q-Sharp)

A programming language created by Microsoft for quantum computing.

- Clear syntax
- Native quantum types
- Integration with .NET
- Ideal for complex algorithms.

🐍 Python

The most widely used language in data science and AI.

- Support via Azure Quantum SDK
- Compatible with Qiskit and Cirq.
- Perfect for hybrid pipelines.

🔵 .NET / C#

Used for:

- APIs
- Azure Functions
- Job orchestration
- Corporate integration

🧪 2.3 Creating Your First Quantum Program in Azure Quantum

Let's start with a simple example in **Q#** , which creates a qubit in superposition and measures it.

📄 Q# Code

qsharp

```
namespace Quantum.Example {
open Microsoft.Quantum.Intrinsic;
open Microsoft.Quantum.Canon;
```

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
operation HelloQubit() : Result {
using (q = Qubit()) {
H(q); // Hadamard gate → superposition
let result = M(q); // Measurement
Reset(q);
return result;
}
}
}
```

▶ Running locally

You can run this code using:

Code

```
dotnet run
```

or by using the local quantum simulator.

🚀 2.4 Running a Job in Azure Quantum

To submit a job to Azure Quantum, you need to:

1. Create a **Workspace**
2. Choosing a **provider**
3. Submit the **circuit**
4. Wait for the result.

🛞 Creating a Workspace

In the Azure portal:

1. Create resource → Azure Quantum
2. Choose region
3. Select providers

4. Create credentials

Python Example

python

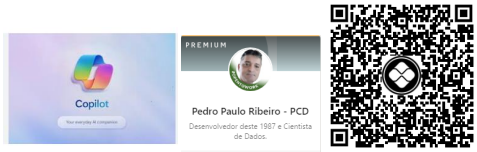
```
from azure.quantum import Workspace
from azure.quantum.qiskit import AzureQuantumProvider
from qiskit import QuantumCircuit

# Connecting to the workspace
ws = Workspace(
subscription_id="xxxx",
resource_group="xxxx",
name="quantum-ws",
location="eastus"
)

provider = AzureQuantumProvider(ws)

Simple circuit
qc = QuantumCircuit(1,1)
qc.h(0)
qc.measure(0,0)

backend = provider.get_backend("ionq.simulator")
job = backend.run(qc)
print(job.result())
```

2.5 Quantum Simulators in Azure

Azure offers simulators for testing algorithms at no high cost.

◈ Types of simulators

- **Full-state simulator**
  - Simulates all possible states.
  - Ideal for up to ~30 qubits
- **Resource estimator**
  - Estimate the resources needed for the actual hardware.
- **Toffoli simulator**
  - Focused on reversible circuits



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Simulators are essential for development before using real hardware.

🔗 2.6 Integration with AI and Classical Computing

Azure Quantum is designed to work together with:

- Azure Machine Learning
- Azure Functions
- Cosmos DB
- Redis
- .NET APIs
- Python

This allows you to create **hybrid pipelines** , such as:

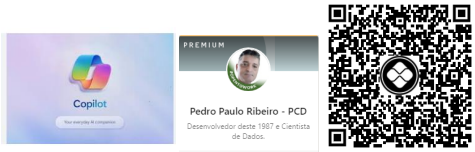
1. AI generates parameters
2. Quantum algorithm performs optimization.
3. AI interprets results.
4. The .NET API delivers a response to the user.

📦 2.7 Execution Architecture in Azure Quantum

🗺️ Execution flow

Code

```
Code (Q#, Python, .NET)
↓
Azure Quantum Workspace
↓
Provider (IonQ, Quantinuum, Rigetti)
↓
Result
↓
Application (AI, API, Functions, DB)
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Safety and isolation

- Jobs are isolated.
- Data is transmitted encrypted.
- Logs and metrics are stored in the workspace.

2.8 Costs and Practical Considerations

Costs vary by:

- Provider
- Hardware type
- Number of qubits
- Execution time
- Number of shots (repeated executions)

Tip

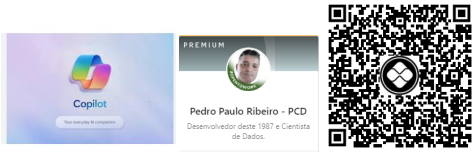
Use simulators for development and real hardware only for final validation.

Chapter Summary

In this chapter, you learned:

- What is Azure Quantum?
- How does the workspace work?
- Supported languages (Q#, Python, .NET)
- How to create and run a circuit
- How to submit jobs to real providers
- How to use simulators
- How to integrate with AI and Azure services

You are now ready to move on to **Chapter 3 — Artificial Intelligence Integrated with Quantum Computing** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧠 Chapter 3 — Artificial Intelligence Integrated with Quantum Computing

The convergence between **Artificial Intelligence (AI)** and **Quantum Computing** represents one of the most promising areas of modern technology. While classical AI dominates tasks of perception, classification, and prediction, quantum computing offers new ways of optimization, sampling, and simulation. Together, they form a hybrid ecosystem capable of solving previously impossible problems.

This chapter explores how AI and quantum computing complement each other, how hybrid algorithms work, and how to integrate these technologies using **Azure Quantum** , **Python** , **.NET** , and Azure services.

🤖 3.1 What is Quantum Machine Learning (QML)?

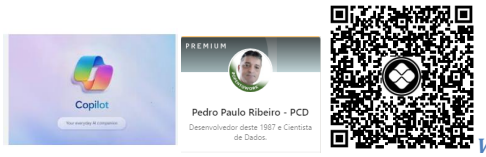
Quantum Machine Learning (QML) is the field that combines:

- **Quantum algorithms**
- **Machine learning models**
- **Hybrid optimization**

The central idea is to use qubits and quantum operations to accelerate specific parts of the learning process.

📌 Areas where QML excels

- Quantum classification
- Hyperparameter optimization
- Sampling of complex distributions
- Simulation of physical systems
- Dimensionality reduction



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧬 3.2 Hybrid Models: Classical + Quantum

Current quantum computing is in the **NISQ** (Noisy Intermediate-Scale Quantum) era, meaning that purely quantum algorithms are still limited. Therefore, the most efficient models are **hybrid** , combining:

▨ Classic Part

- Data preprocessing
- Parameter adjustment
- Optimization via gradients
- Traditional AI (neural networks, regression, clustering)

▨ Quantum part

- Evaluation of complex functions
- Combinatorial optimization
- Quantum sampling
- Calculation of energetic states

🧬 Typical hybrid flow

Code

```
Data → Classical AI → Quantum Circuit → Result → Classical AI → Decision
```

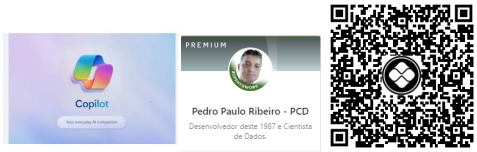
🔄 3.3 Quantum Algorithms for AI

◈ VQE (Variational Quantum Eigensolver)

Used to find minimum energy states. Applications: chemistry, materials, optimization.

◈ QAOA (Quantum Approximate Optimization Algorithm)

Solves combinatorial optimization problems. Applications: logistics, finance, routing.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ QSVM (Quantum Support Vector Machine)

Quantum version of the classical SVM. Applications: classification and pattern analysis.

◈ Quantum Neural Networks (QNNs)

Neural networks implemented with quantum gates. Applications: pattern recognition and prediction.

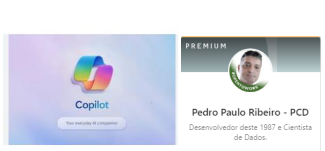
🧪 3.4 Practical Example: Quantum Classifier in Python

Here is a simple example using **Qiskit + Azure Quantum** to create a circuit that can be used as a building block for a quantum classifier.

python

```
from qiskit import QuantumCircuit
from azure.quantum import Workspace
from azure.quantum.qiskit import AzureQuantumProvider

# Conexão com o workspace
ws = Workspace(
    subscription_id="xxxx",
    resource_group="xxxx",
    name="quantum-ws",
    location="eastus"
)

provider = AzureQuantumProvider(ws)

# Circuito quântico simples para classificação
qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0,1], [0,1])

backend = provider.get_backend("ionq.simulator")
job = backend.run(qc)
print(job.result())
```

This circuit can be integrated into an AI pipeline for binary classification.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

3.5 Classical AI Supporting Quantum Computing

AI can also improve quantum algorithms:

Parameter optimization

Neural networks can adjust the angles of quantum gates.

Noise reduction

AI models can correct quantum errors.

Results Analysis

AI can interpret probability distributions generated by circuits.

Circuit generation

Generative AI can create new optimized quantum circuits.

3.6 Quantum Computing Supporting AI

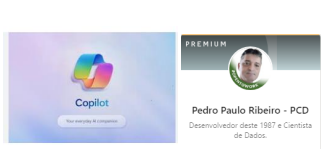
Quantum computing can accelerate:

Model Optimization

QAOA can find better weights for neural networks.

Sampling

Quantum sampling can improve probabilistic models.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Simulation

Quantum simulations can generate realistic synthetic data.

Quantum neural networks

QNNs can learn patterns that are impossible for classical networks.

3.7 AI + Quantum Computing on Azure

Azure enables the integration of AI and quantum computing using:

◈ Azure Quantum

Execution of quantum circuits.

◈ Azure Machine Learning

Training on classic and hybrid models.

◈ Azure Functions

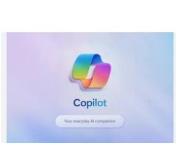
Serverless orchestration of hybrid pipelines.

◈ Cosmos DB

Quantum result storage.

◈ Redis

State and parameter cache.




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ .NET + Python

Corporate integration and hybrid pipelines.

🏗️ 3.8 Architecture of a Hybrid AI + Quantum Pipeline

🗺️ Full flow

Code

```
Data → Azure ML → Preprocessing
↓
Azure Function
↓
Azure Quantum
↓
Cosmos DB / Redis
↓
Azure ML (post-processing)
↓
.NET API
↓
End User
```

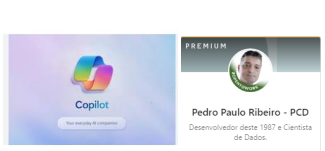
📦 Benefits

- Scalability
- Low cost
- Modularity
- Integration with AI
- True hybrid execution

🧪 3.9 Complete Example: Hybrid Pipeline

▨ Step 1 — AI generates parameters

Python + Azure ML



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🟦 Step 2 — Azure Function sends quantum job

C# + Azure Functions

🟦 Step 3 — Azure Quantum executes the circuit

IonQ / Quantinuum / Rigetti

🟦 Step 4 — Cosmos DB stores results

Structured JSON

🟦 Step 5 — AI interprets results

Neural network adjusts weights

🟦 Step 6 — .NET API returns a response

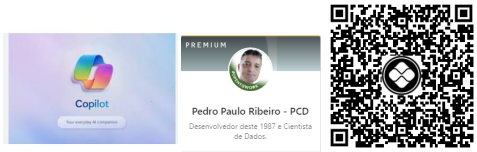
Corporate application

🎯 Chapter Summary

In this chapter, you learned:

- What is Quantum Machine Learning?
- How hybrid models work
- Key quantum algorithms for AI
- How AI and quantum computing complement each other.
- How to integrate AI + Quantum in Azure
- How to build real hybrid pipelines

You are now ready to move on to **Chapter 4 — Azure Functions: Serverless Computing for Quantum Pipelines** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

⚙ Chapter 4 — Azure Functions: Serverless Computing for Quantum Pipelines

Serverless computing has changed the way modern applications are built. Instead of managing servers, scalability, and infrastructure, the developer focuses solely on the code. Azure Functions takes this concept to the extreme, allowing the creation of small, independent, and highly scalable functions that can be triggered by events.

In this chapter, you will learn how to use Azure Functions to orchestrate **quantum jobs** , integrating **Python** , **.NET** , **Cosmos DB** , **Redis** , and **AI** , creating powerful hybrid pipelines.

☁ 4.1 What is Azure Functions?

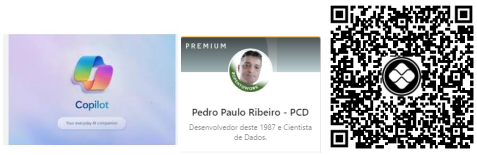
Azure Functions is a serverless service that allows you to run code on demand, without needing to provision or manage servers.

◈ Main features

- Automatic scalability
- Payment for execution
- Support for multiple languages (C#, Python, JavaScript, PowerShell)
- Native integration with Azure services
- Ideal for quantum pipelines and AI.

🔗 When to use Azure Functions

- Orchestrating quantum jobs
- Create lightweight APIs
- Process events in real time.
- Automating AI workflows
- Integrate Cosmos DB, Redis, and external services.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

4.2 Types of Triggers and Bindings

Azure Functions works with **triggers** (events that start the function) and **bindings** (automatic connections to services).

Most commonly used triggers

- **HTTP Trigger** → Serverless APIs
- **Timer Trigger** → Scheduled Executions
- **Queue Trigger** → Asynchronous Processing
- **Cosmos DB Trigger** → React to changes in the database.
- **Event Grid Trigger** → Azure Service Events

Useful bindings for quantum computing

- Cosmos DB (input/output)
- Storage Queue
- Redis (via extensions)
- Service Bus
- HTTP

4.3 Creating your First Azure Function in .NET

Here's a simple example of a Function that triggers a quantum job in Azure Quantum.

Code in C# (.NET 8)

csharp

```
public static class QuantumTrigger
{
[FunctionName("QuantumTrigger")]
public static async Task<IActionResult> Run(
[HttpTrigger(AuthorizationLevel.Function, "post")] HttpRequest req,
Log (Logger)
{
log.LogInformation("Receiving request for quantum job...");
```

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
Here you would call Azure Quantum.
// Simplified example:
var jobId = Guid.NewGuid().ToString();

return new OkObjectResult(new {
message = "Quantum job sent successfully!",
jobId = jobId
});
}
}
```

This function can be called by:

- .NET Applications
- Python
- AI
- Front-ends
- Other Functions

🧪 4.4 Azure Functions + Python + Azure Quantum

You can also use Python within Functions to send quantum circuits.

📄 Python Example

python

```
import azure.functions as func
from azure.quantum import Workspace
from qiskit import QuantumCircuit

def main(req: func.HttpRequest) -> func.HttpResponse:
    ws = Workspace(
        subscription_id="xxxx",
        resource_group="xxxx",
        name="quantum-ws",
        location="eastus"
    )

    qc = QuantumCircuit(1,1)
    qc.h(0)
    qc.measure(0,0)
```

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
    backend = ws.get_backend("ionq.simulator")
    job = backend.run(qc)

    return func.HttpResponse(f"Resultado: {job.result()}")
```

🔗 4.5 Integrando Azure Functions com Cosmos DB

Azure Functions pode salvar automaticamente resultados quânticos no Cosmos DB.

📄 Exemplo de binding de saída

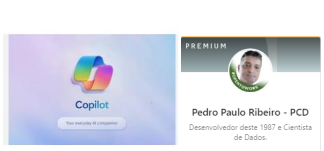
csharp

```
public class QuantumResult {
    public string id { get; set; }
    public string result { get; set; }
}

[FunctionName("SaveQuantumResult")]
public static async Task Run(
    [QueueTrigger("quantum-results")] string result,
    [CosmosDB(
        databaseName: "quantumdb",
        containerName: "results",
Connection   =   "CosmosDBConnection")]   IAsyncCollector<QuantumResult>
output,
Log (Logger)
{
await output.AddAsync(new QuantumResult {
id = Guid.NewGuid().ToString(),
result = result
});
}
```

⚡ 4.6 Integrating Azure Functions with Redis

Redis is perfect for:

- Quantum result cache
- Store AI parameters
- Synchronize pipelines



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📄 Example in C#

csharp

```
var redis = ConnectionMultiplexer. Connect("localhost");
var db = redis.GetDatabase();

db.StringSet("last_quantum_result", "1");
var value = db.StringGet("last_quantum_result");
```

🪐 4.7 Azure Functions as a Quantum Pipeline Orchestrator

Azure Functions can coordinate:

- AI (Azure ML)
- Quantum computing (Azure Quantum)
- Storage (Cosmos DB)
- Cache (Redis)
- APIs (.NET)

🗺️ Typical flow

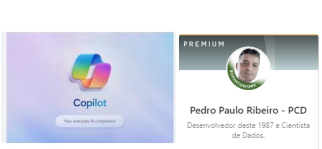
Code

```
User → .NET API → Azure Role → Azure Quantum
↓
Cosmos DB
↓
Redis
↓
Azure ML (AI)
↓
.NET API
```

🧠 4.8 Durable Functions for Complex Flows

Durable Functions enable:

- Multi-step workflows
- Long runs (perfect for quantum work)
- Persistent states
- Fan-out/fan-in



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/

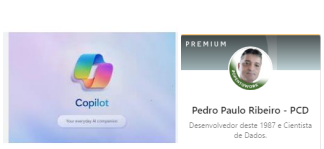
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

Example of Orchestrator

csharp

```
[FunctionName("QuantumOrchestrator")]
public static async Task<string> RunOrchestrator(
[OrchestrationTrigger] IDurableOrchestrationContext context)
{
var jobId = await
context.CallActivityAsync<string>("SubmitQuantumJob", null);
var result = await
context.CallActivityAsync<string>("GetQuantumResult", jobId);
return result;
}
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

4.9 Best Practices for Azure Functions in Quantum Systems

✔ **Use small, independent Functions**

✔ **Prefer bindings to manual code .**

✔ **Use Durable Functions for long-term jobs .**

✔ `Store` **results in Cosmos DB**

✔ **Use Redis for caching**

✔ **Separate orchestration (Functions) from execution (Quantum)**

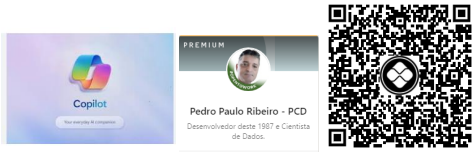
✔ **Monitor with Application Insights**

Chapter Summary

In this chapter, you learned:

- What is Azure Functions?
- How to create Functions in .NET and Python
- How to integrate with Azure Quantum
- How to save results in Cosmos DB
- How to use Redis for caching
- How to orchestrate hybrid pipelines
- How to use Durable Functions for complex flows

You are now ready to move on to **Chapter 5 — Cosmos DB: A Globally Distributed Database for Quantum Applications** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🌍 Chapter 5 — Cosmos DB: Globally Distributed Database for Quantum Applications

Azure **Cosmos DB** is a globally distributed, highly scalable, and low-latency database designed for modern applications that demand consistent performance on a planetary scale. In quantum systems, it plays a vital role by storing:

- Quantum job results
- AI parameters
- intermediate states of hybrid pipelines
- Logs and telemetry
- Input data for quantum algorithms

This chapter explores how to use Cosmos DB efficiently in quantum architectures.

🧩 5.1 What is Azure Cosmos DB?

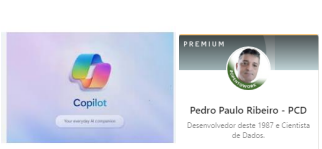
Cosmos DB is a multi-model NoSQL database with:

- Global replication
- Low latency (<10 ms)
- Elastic scalability
- 99.999% SLA
- Support for multiple data models.

🗃️ Supported models

- **Documents (JSON)** — Core API (SQL)
- **Key-value** — API Table
- **Graphs** — Gremlin API
- **Wide columns** — Cassandra API
- **MongoDB** — API compatible

For quantum applications, the most commonly used model is **JSON documents** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

⚡ 5.2 Why Cosmos DB is ideal for Quantum Computing

◈ Low global latency

Quantum jobs can be launched from any region.

◈ Automatic scalability

Quantum pipelines can generate thousands of results per minute.

◈ Flexible JSON

Quantum results vary in structure.

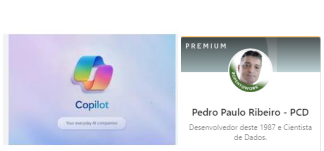
◈ Configurable consistency

Choose from:

- Strong
- Bounded Staleness
- Session
- Consistent Prefix
- Possible

◈ Native integration with Azure Functions

Bindings simplify recording and reading.



Voluntary Contribution
https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📦 5.3 Data Structures for Quantum Results

A typical quantum job result can be stored like this:

json

```
{
"id": "job-123",
"timestamp": "2026-01-07T14:00:00Z",
"provider": "IonQ",
"shots": 1000,
"circuit": "H -> CNOT -> Measure",
"result": {
"00": 512,
"01": 488
},
"metadata": {
"user": "pedro",
"pipeline": "QAOA-optimizer"
}
}
```

🧪 5.4 Creating a Container in Cosmos DB

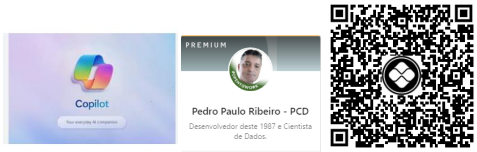
◈ Steps in the Azure portal

1. Create resource → Cosmos DB (Core SQL API)
2. Create a database (e.g., `quantumdb`)
3. Create a container (e.g., `results`)
4. Define partition key (e.g., `/provider`)

🛠 5.5 Accessing Cosmos DB with .NET

Here is a complete example in C#:

csharp



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/

WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
using Microsoft.Azure.Cosmos;

var client = new CosmosClient(connectionString);
var container = client.GetContainer("quantumdb", "results");

var item = new {
    id = Guid.NewGuid().ToString(),
    provider = "IonQ",
    result = "01",
    timestamp = DateTime.UtcNow
};

await container.CreateItemAsync(item, new PartitionKey("IonQ"));
```

◈ Leitura de dados

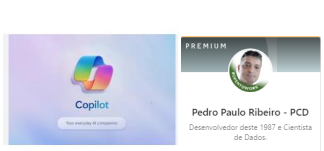
csharp

```
var query = container.GetItemQueryIterator<dynamic>(
    new QueryDefinition("SELECT * FROM c WHERE c.provider = @p")
        .WithParameter("@p", "IonQ")
);

while (query.HasMoreResults)
{
    foreach (var doc in await query.ReadNextAsync())
        Console.WriteLine(doc);
}
```

5.6 Acessando Cosmos DB com Python

python

```
from azure.cosmos import CosmosClient

client = CosmosClient(url, credential=key)
database = client.get_database_client("quantumdb")
container = database.get_container_client("results")

item = {
    "id": "job-001",
    "provider": "Quantinuum",
    "result": "00",
    "timestamp": "2026-01-07T14:00:00Z"
}

container.create_item(item)
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🔗 5.7 Integração com Azure Functions

Azure Functions pode gravar automaticamente no Cosmos DB usando bindings.

📄 Exemplo de binding de saída

csharp

```
[FunctionName("SaveQuantumResult")]
public static async Task Run(
    [QueueTrigger("quantum-results")] string result,
    [CosmosDB(
        databaseName: "quantumdb",
        containerName: "results",
        Connection  =  "CosmosDBConnection")]  IAsyncCollector<dynamic>
output)
{
await output.AddAsync(new {
id = Guid.NewGuid().ToString(),
provider = "IonQ",
result = result,
timestamp = DateTime.UtcNow
});
}
```

⚙ 5.8 Cosmos DB + Redis: Storage + Cache

◈ Cosmos DB

Stores permanent results.

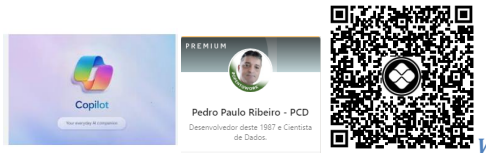
◈ Redis

Stores recent results for quick access.

🔗 Typical flow

Code

```
Azure Quantum → Azure Function → Redis (cache)
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
↓
Cosmos DB (persistence)
```

🧠 5.9 Advanced Queries for AI and Quantum

🔍 Search for results by probability

SQL

```
SELECT c.id, c.result
FROM c
WHERE c.result["00"] > 500
```

🔍 Search for jobs by pipeline

SQL

```
SELECT * FROM c WHERE c.metadata.pipeline = "QAOA-optimizer"
```

🔍 Search results by date

SQL

```
SELECT * FROM c
WHERE c.timestamp > "2026-01-01T00:00:00Z"
```

🪐 5.10 Best Practices for Cosmos DB in Quantum Systems

✔ Use well-defined partitions

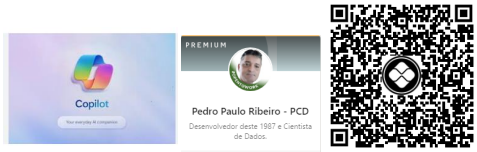
Ex: `/provider` , `/pipeline` , `/user`

✔ Avoid giant documents

Prefer job-specific granularity.

✔ Use TTL for temporary data

Interim results may expire automatically.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

✔ Combine Cosmos DB + Redis

Cache + persistence is the perfect combination.

✔ Use custom indexes

Improves AI queries.

Chapter Summary

In this chapter, you learned:

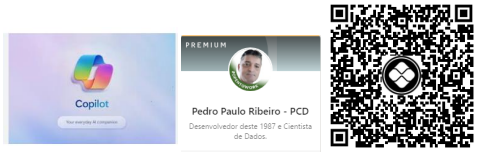
- What is Cosmos DB?
- Why is it ideal for quantum computing?
- How to model quantum data in JSON
- How to access Cosmos DB with .NET and Python
- How to integrate with Azure Functions
- How to combine Cosmos DB + Redis
- Best practices for hybrid pipelines

You are now ready to move on to **Chapter 6 — Redis: Caching and Messaging for Quantum Systems** .

Chapter 6 — Redis: Cache and Messaging for Quantum Systems

Redis is one of the world's fastest in-memory databases, designed for low-latency operations. In architectures involving **Azure Quantum** , **AI** , **Cosmos DB** , **Azure Functions** , **Python ,** and **.NET** , Redis plays a fundamental role as:

- Quantum result cache
- Messaging between services
- Temporary storage of AI parameters
- Synchronization of hybrid pipelines
- State control in distributed executions



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

This chapter explores how to use Redis efficiently in modern quantum systems.

🚀 6.1 What is Redis?

Redis (Remote Dictionary Server) is an extremely fast, in-memory database used for:

- Cache
- Queues
- Pub/Sub
- Sessions
- Accountants
- Temporary storage

🔥 Key Features

- Latency in the microsecond range.
- Support for advanced data structures
- Optional persistence
- High availability
- Clustering
- Support for Lua scripts
- Pub/Sub native

💠 6.2 Why Redis is essential in Quantum Computing

Modern quantum systems require:

- Low latency
- High request rate
- Temporary storage
- Synchronization between services

- Cache of repeated results

Redis perfectly meets these requirements.

📌 Examples of use in quantum pipelines

- Cache of repeated circuit results
- QAOA parameter storage
- Synchronization between Azure Functions
- Quantum job queue
- Pub/Sub between AI and Quantum
- Workflow state control

📦 6.3 Redis Data Structures for Quantum

Redis offers ideal frameworks for quantum computing:

◈ Strings

For storing simple results.

◈ Hashes

To store job metadata.

◈ Lists

For quantum job queues.

◈ Sets / Sorted Sets

For ranking of results.

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ Pub/Sub

For communication between services.

◈ Streams

For real-time data pipelines.

6.4 Using Redis with Python

Basic example

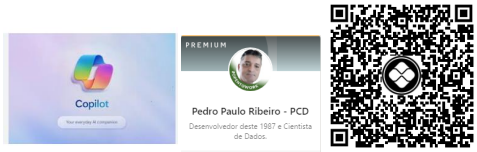
python

```
import redis

r = redis.Redis(host='localhost', port=6379)

# Armazenar resultado quântico
r.set("last_quantum_result", "01")

# Recuperar
print(r.get("last_quantum_result"))
```

Armazenando resultados complexos

python

```
import json

result = {
    "jobId": "123",
    "provider": "IonQ",
    "result": {"00": 512, "01": 488}
}

r.set("quantum:123", json.dumps(result))
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

6.5 Usando Redis com .NET

Exemplo em C#

csharp

```
using StackExchange.Redis;

var redis = ConnectionMultiplexer.Connect("localhost");
var db = redis.GetDatabase();

db.StringSet("last_quantum_result", "00");
var value = db.StringGet("last_quantum_result");

Console.WriteLine(value);
```

Hashes para metadados

csharp

```
db.HashSet("job:123", new HashEntry[] {
    new HashEntry("provider", "Quantinuum"),
    new HashEntry("shots", 1000),
    new HashEntry("status", "completed")
});
```

6.6 Redis + Azure Functions

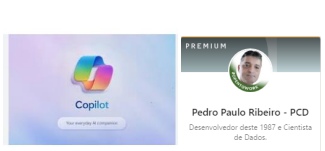
Redis is perfect for:

- Results cache
- Pipeline synchronization
- State control
- Job queues

Example of use in Azure Function

csharp

```
[FunctionName("QuantumCache")]
public static async Task<IActionResult> Run(
[HttpTrigger(AuthorizationLevel.Function, "get")] HttpRequest req)
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
{
var redis = ConnectionMultiplexer. Connect("localhost");
var db = redis.GetDatabase();

var result = db.StringGet("last_quantum_result");

return new OkObjectResult(result);
}
```

📡 6.7 Redis Pub/Sub for Quantum Messaging

Redis Pub/Sub enables instant communication between services.

📄 Posting messages

python

```
r.publish("quantum_channel", "job_completed:123")
```

📄 Listening to messages

python

```
pubsub = r.pubsub()
pubsub.subscribe("quantum_channel")

for message in pubsub.listen():
print(msg)
```

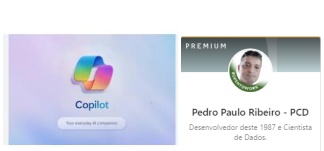
📌 Applications

- Notify AI when quantum job finishes.
- Synchronize multiple Azure functions
- Update dashboards in real time.

🔄 6.8 Redis Streams for Quantum Pipelines

Redis Streams is ideal for:

- Real-time data processing



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- AI Pipelines
- Quantum job logs

📄 Example

python

```
r.xadd("quantum_stream", {"job": "123", "result": "01"})
```

🧬 6.9 Redis + Cosmos DB: Cache + Persistence

Redis and Cosmos DB make a perfect pair:

◈ Redis

- Temporary data
- Cache
- High speed

◈ Cosmos DB

- Persistence
- History
- Complex consultations

🗺️ Typical flow

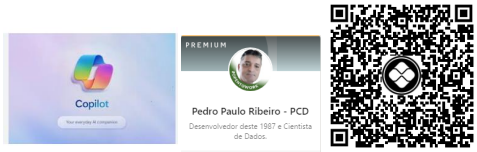
Code

```
Azure Quantum → Azure Function → Redis (cache)
↓
Cosmos DB (persistence)
```

🧠 6.10 Best Practices for Redis in Quantum Systems

✔ Use TTL for temporary data

Avoid unnecessary clutter.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

✔ Use standardized keys

Ex.: `quantum:job:123`

✔ Combine Redis + Cosmos DB

Cache + persistence is essential.

✔ Use Pub/Sub for synchronization

Ideal for distributed pipelines.

✔ Avoid storing huge documents .

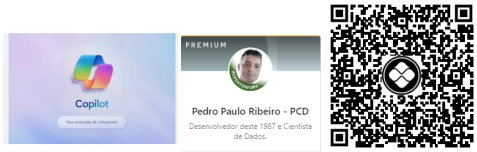
Redis is optimized for small data sets.

🎯 Chapter Summary

In this chapter, you learned:

- What is Redis?
- Why is it essential in quantum computing?
- How to use Redis with Python and .NET
- How to integrate Redis with Azure Functions
- How to use Pub/Sub and Streams
- How to combine Redis + Cosmos DB
- Best practices for hybrid pipelines

You are now ready to move on to **Chapter 7 — Python and .NET Working Together in Quantum Architectures** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🧬 Chapter 7 — Python and .NET Working Together in Quantum Architectures

Python and .NET are two of the most powerful and popular platforms in the world. Python dominates areas such as **AI, data science, quantum computing, and automation** , while .NET is widely used in **enterprise systems, APIs, high-performance applications, and business integrations** .

In modern quantum architectures, the combination of these two technologies creates an extremely efficient hybrid ecosystem, enabling:

- Python → quantum algorithms, AI, optimization
- .NET → APIs, orchestration, security, integrations
- Azure → infrastructure, scalability, managed services

This chapter explores how to bring these two worlds together in an elegant and productive way.

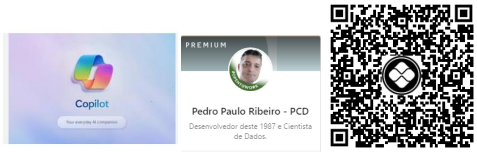
🧬 7.1 Why integrate Python and .NET?

▨ Python is ideal for:

- Quantum algorithms (Qiskit, Cirq, Azure Quantum SDK)
- Machine Learning (TensorFlow, PyTorch, Scikit-Learn)
- Data manipulation
- Rapid prototyping

▨ .NET is ideal for:

- Enterprise APIs



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Azure Functions
- Microservices
- Security and authentication
- Integration with Cosmos DB, Redis, and Azure.

🎯 Benefits of integration

Combining Python and .NET allows you to create **hybrid pipelines** where each language does what it does best.

🧩 7.2 Hybrid Architecture Python + .NET

A typical architecture in quantum systems:

Code

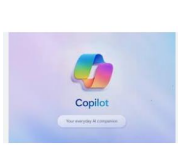
```
User → .NET API → Azure Function → Python (Quantum/IA)
↓
Cosmos DB / Redis
↓
.NET API
```

📦 Components

- **.NET** exposes APIs, controls flow, and authenticates users.
- **Python** → executes quantum circuits and AI
- **Azure Functions** → orchestrates everything
- **Cosmos DB** → stores results
- **Redis** → caching and synchronization

🐍 7.3 Calling Python from .NET

There are three main approaches:




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ 1. Python.NET (pythonnet)

Allows you to import Python modules directly into C#.

csharp

```
using Python.Runtime;

PythonEngine.Initialize();
using (Py.GIL())
{
dynamic np = Py.Import("numpy");
Console.WriteLine(np.cos(np.pi * 2));
}
```

◈ 2. External processes (most common in production)

Executes Python scripts via the command line.

csharp

```
var psi = new ProcessStartInfo();
psi.FileName = "python";
psi.Arguments = "quantum_job.py";
psi.RedirectStandardOutput = true;

var process = Process.Start(psi);
string output = process.StandardOutput.ReadToEnd();
```

◈ 3. Containers (Docker)

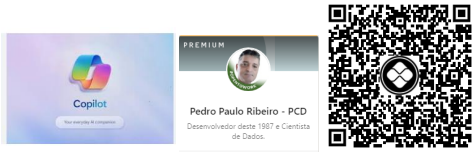
Python and .NET run in separate containers and communicate via HTTP.

🔵 7.4 Calling .NET from Python

Python can call .NET APIs using HTTP.

📄 Python Example

python



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
import requests

response = requests.get("https://meuapi.net/quantum/result/123")
print(response.json())
```

📌 Applications

- Python queries results in Cosmos DB via .NET API
- Python sends parameters to Azure Functions.
- Python integrates AI with enterprise systems.

☁ 7.5 Python + .NET in Azure Functions

Azure Functions allows you to mix languages in the same project:

- Functions in C# → orchestration
- Functions in Python → quantum execution

▨ Example of a C# function calling Python

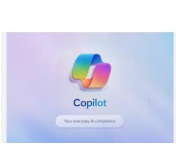
csharp

```
[FunctionName("RunQuantumPython")]
public static async Task<IActionResult> Run(
[HttpTrigger(AuthorizationLevel.Function, "get")] HttpRequest req)
{
var psi = new ProcessStartInfo("python", "quantum.py");
psi.RedirectStandardOutput = true;

var process = Process.Start(psi);
string result = process.StandardOutput.ReadToEnd();

return new OkObjectResult(result);
}
```

🧪 Python 7.6 for Quantum Computing

Python is the primary language for:

- Qiskit
- Circus




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Azure Quantum SDK
- PennyLane
- Braket

📄 Simple circuit example

python

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(1,1)
qc.h(0)
qc.measure(0,0)

print(qc)
```

📦 7.7 .NET for Orchestration and Integration

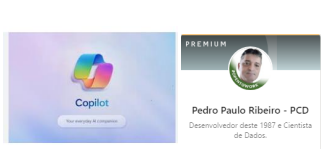
.NET is ideal for:

- REST APIs
- Microservices
- Azure Functions
- Authentication (Azure AD)
- Integration with Cosmos DB and Redis

📄 Example of a .NET API to return quantum results

csharp

```
[HttpGet("quantum/result/{id}")]
public async Task<IActionResult> GetResult(string id)
{
var result = await _cosmos.GetItemAsync(id);
return Ok(result);
}
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🔄 7.8 Communication between Python and .NET via Messaging

◈ Redis Pub/Sub

Python publishes → .NET consumes .NET publishes → Python consumes

◈ Azure Queue Storage

Python sends jobs → .NET processes them.

◈ Event Grid

Quantum job events → .NET APIs

🧠 7.9 Architectural Patterns Python + .NET

▨ Pattern 1 — .NET Orchestration, Python Execution

More common in companies.

Pattern 2 — Python as the AI engine, .NET as the API

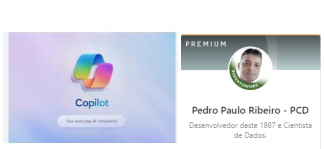
Ideal for AI + Quantum.

▨ Pattern 3 — Independent Microservices

Python and .NET communicate via HTTP.

▨ Standard 4 — Containers

Both run on Docker.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

7.10 Good Practices

✔ `Use Python` **for AI and Quantum**

✔ **Use .NET for APIs and integrations**

✔ **Use Redis for caching and synchronization .**

✔ **Use Cosmos DB for persistence**

✔ `Use Azure Functions` **for orchestration**

✔ **Separate responsibilities by language .**

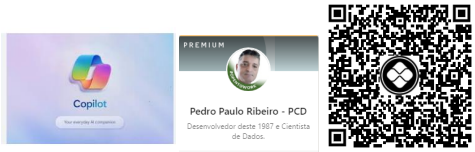
✔ `Use containers` **for isolation**

Chapter Summary

In this chapter, you learned:

- Why integrate Python and .NET?
- How to create hybrid architectures
- How to call Python from .NET
- How to call .NET from Python
- How to use Azure Functions with both languages
- How to integrate AI, Quantum, Cosmos DB, and Redis
- Recommended architectural patterns

You are now ready to move on to **Chapter 8 — Complete Architecture of a Quantum System in Azure** .



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

🏛️ Chapter 8 — Complete Architecture of a Quantum System in Azure

Quantum computing, by itself, does not solve problems in isolation. It needs to be integrated into an ecosystem of classical services, AI, databases, messaging, and APIs. Azure offers all the necessary components to build **hybrid quantum-classical systems** , and this chapter presents a complete architecture that combines:

- **Azure Quantum**
- **Azure Functions**
- **Cosmos DB**
- **Redis**
- **Python**
- **.NET**
- **AI (Machine Learning)**

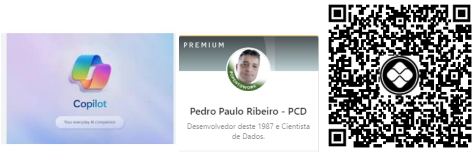
The goal is to show how these elements connect to form a robust, scalable, and production-ready solution.

🎡 8.1 Architectural Overview

The complete architecture can be represented as follows:

Code

```
User → .NET API → Azure Function → Azure Quantum
↓
Cosmos DB
↓
Redis
↓
Azure ML (AI)
↓
.NET API
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

◈ Summary of the flow

1. The user sends a request to the .NET API.
2. The API calls an Azure Function.
3. A Function sends a job to Azure Quantum.
4. The result is stored in Cosmos DB.
5. Redis stores the cache for quick access.
6. Azure ML interprets or optimizes the result.
7. The .NET API returns the final response to the user.

☁ 8.2 Architectural Components

.NET API

- System entry point
- Authentication and authorization
- Exposing REST endpoints
- Integration with Cosmos DB and Redis

Azure Functions

- Serverless orchestration
- Execution of hybrid pipelines
- Azure Quantum Calls
- Asynchronous processing

Azure Quantum

- Execution of quantum circuits
- Support IonQ, Quantinuum, Rigetti
- Resource simulators and estimators

Cosmos DB

- Persistent storage

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Job history
- Structured data in JSON

🟦 Redis

- Results cache
- Pub/Sub for messaging
- Temporary storage

🟦 Azure Machine Learning

- Interpretation of results
- Parameter optimization
- Hybrid AI (classical + quantum)

🟦 Python

- Quantum algorithms
- AI and preprocessing
- Circuit execution

🔗 8.3 Detailed Architecture Flow

🟦 Step 1 — User → .NET API

The user sends a request, for example:

Code

```
POST /api/quantum/optimize-route
```

The API validates:

- Authentication
- Parameters

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Permissions

And it sends the request to an Azure Function.

Step 2 — .NET API → Azure Function

The function receives the data and triggers the pipeline:

- Prepare parameters
- It generates quantum circuits.
- Call Python when needed.
- Send job to Azure Quantum

Step 3 — Azure Function → Azure Quantum

The Function sends the circuit:

python

```
job = backend.run(qc, shots=1000)
```

Azure Quantum runs the job on the chosen provider.

Step 4 — Azure Quantum → Cosmos DB

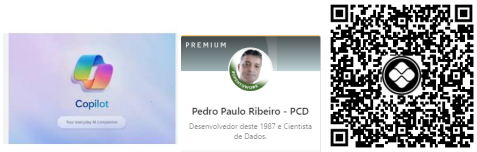
When the job ends:

- The function receives the result.
- Stores in Cosmos DB
- Update the cache in Redis.

Example document:

json

```
{
```



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

```
"id": "job-123",
"provider": "IonQ",
"result": {"00": 512, "01": 488},
"timestamp": "2026-01-07T14:00:00Z"
}
```

🟦 Step 5 — Cosmos DB → Redis

Redis guarda:

- Last result
- AI parameters
- Temporary states

This speeds up subsequent queries.

🟦 Step 6 — Redis → Azure ML

Azure ML uses:

- Quantum results
- Pipeline parameters
- Historical data

To:

- Adjust hyperparameters
- Training hybrid models
- Interpreting results

🟦 Step 7 — Azure ML → .NET API

The .NET API queries:

- Redis (cache)
- Cosmos DB (persistence)

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

- Azure ML (interpretation)

And it returns the final answer to the user.

🧪 8.4 Real-World Case Example: Quantum Route Optimizer

🎯 Objective

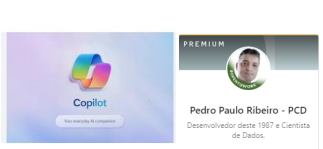
Find the best logistics route using QAOA.

📦 Components used

- Python → QAOA
- Azure Quantum → Execution
- Azure Functions → Orchestration
- Cosmos DB → Storage
- Redis → Cache
- Azure ML → Interpretation
- .NET → Enterprise API

🗺️ Flow

1. User submits list of cities
2. .NET API validation
3. Function generates graph
4. Python creates QAOA circuit
5. Azure Quantum runs
6. Cosmos DB stores
7. Redis guards cache
8. Azure ML interprets
9. The API returns an optimized route.



Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

📦 8.5 Recommended Architectural Patterns

✔ Pattern 1 — Serverless Orchestration

Azure Functions controls everything.

✔ Pattern 2 – Quantum Execution in Python

Python is the ideal language for circuits.

✔ Pattern 3 – Persistence in Cosmos DB

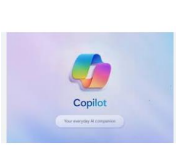
Flexible JSON for quantum results.

✔ Pattern 4 — Redis Cache

Avoids reprocessing.

✔ Pattern 5 — .NET API as an enterprise layer

Security, governance, and integration.




Voluntary Contribution

https://www.linkedin.com/in/pedro-paulo-ribeiro-pcd-1451498/
WhatsApp 55 21 999618643

Quantum Computing with ◈ Azure Quantum, Azure Functions, CosmoDB, Redis, and Artificial Intelligence (AI) and Python on the .NET Platform

8.6 Best Practices for Quantum Architectures

✔ **Use simulators before using real hardware .**

✔ `Store everything` **in Cosmos DB**

✔ **Use Redis to accelerate pipelines**

✔ **Separate responsibilities by language .**

✔ **Use Durable Functions for long-term jobs .**

✔ **Monitor everything with Application Insights**

✔ **Use AI to interpret quantum results**

Chapter Summary

In this chapter, you learned:

- How to assemble a complete architecture for quantum computing.
- How to integrate Azure Quantum, Functions, Cosmos DB, Redis, Python, .NET, and AI.
- How does end-to-end flow work?
- How to build real hybrid pipelines
- Good architectural practices and standards