Follow for
Daily Tech tips

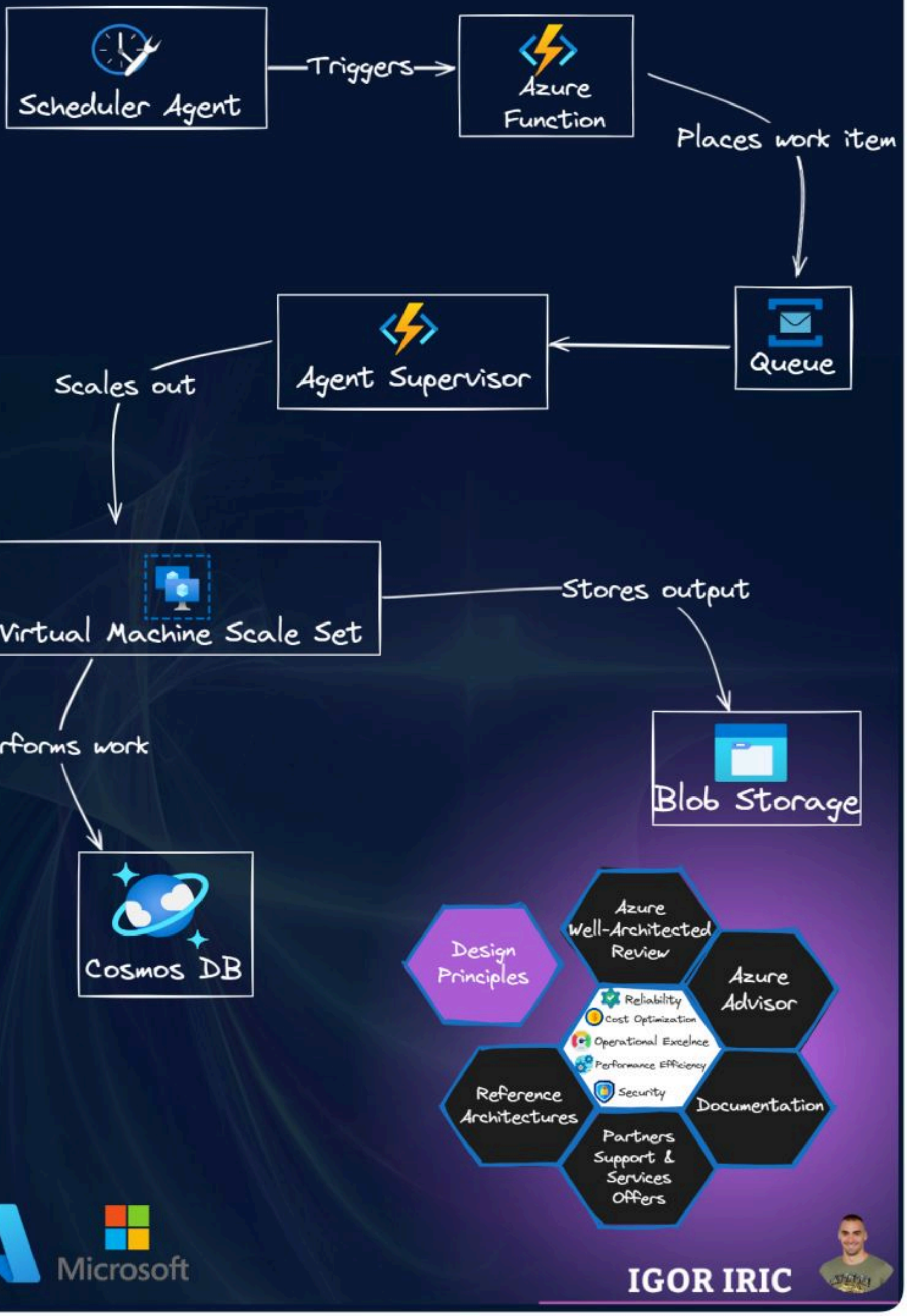
Repost

29 CLOUD DESIGN PATTERNS

Microsoft

IGOR IRIC

Scheduler Agent Supervisor pattern



The Scheduler Agent Supervisor Pattern in Azure is a powerful architectural pattern for orchestrating and automating workflows, particularly useful when you have tasks that need to scale out across multiple workers.

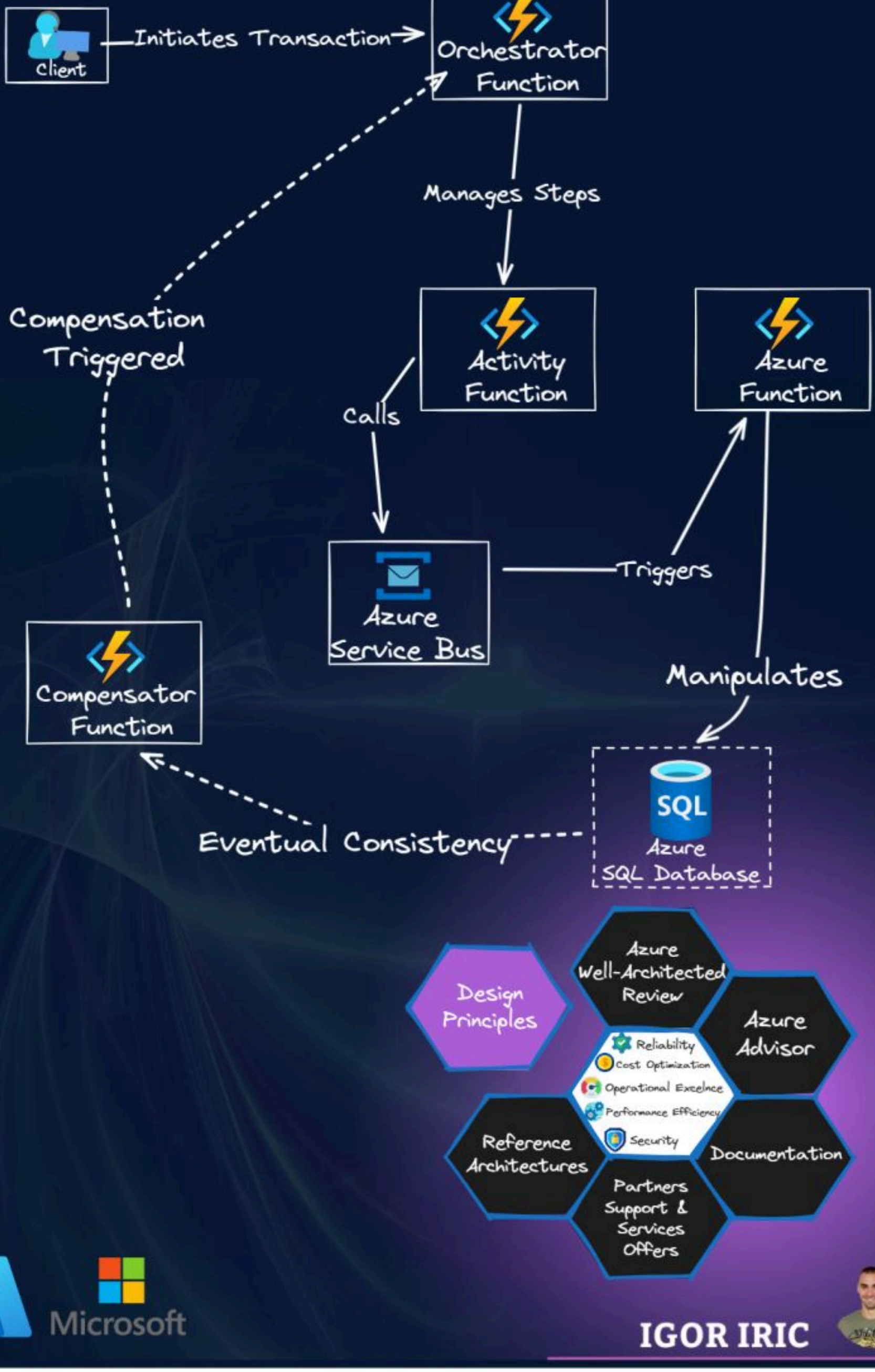
Here's why the Scheduler Agent Supervisor Pattern is great for cloud-based applications:

✅ It scales to meet demand without manual intervention.
✅ By automating task distribution, it optimizes resource usage.
✅ With supervision and scaling mechanisms, it ensures that the system can handle failures gracefully.

However, keep in mind:

— Setting up this pattern can be complex and requires a good understanding of Azure services.
— It's crucial to monitor the performance and health of the system to prevent bottlenecks.

Saga distributed transactions pattern



Microsoft

IGOR IRIC

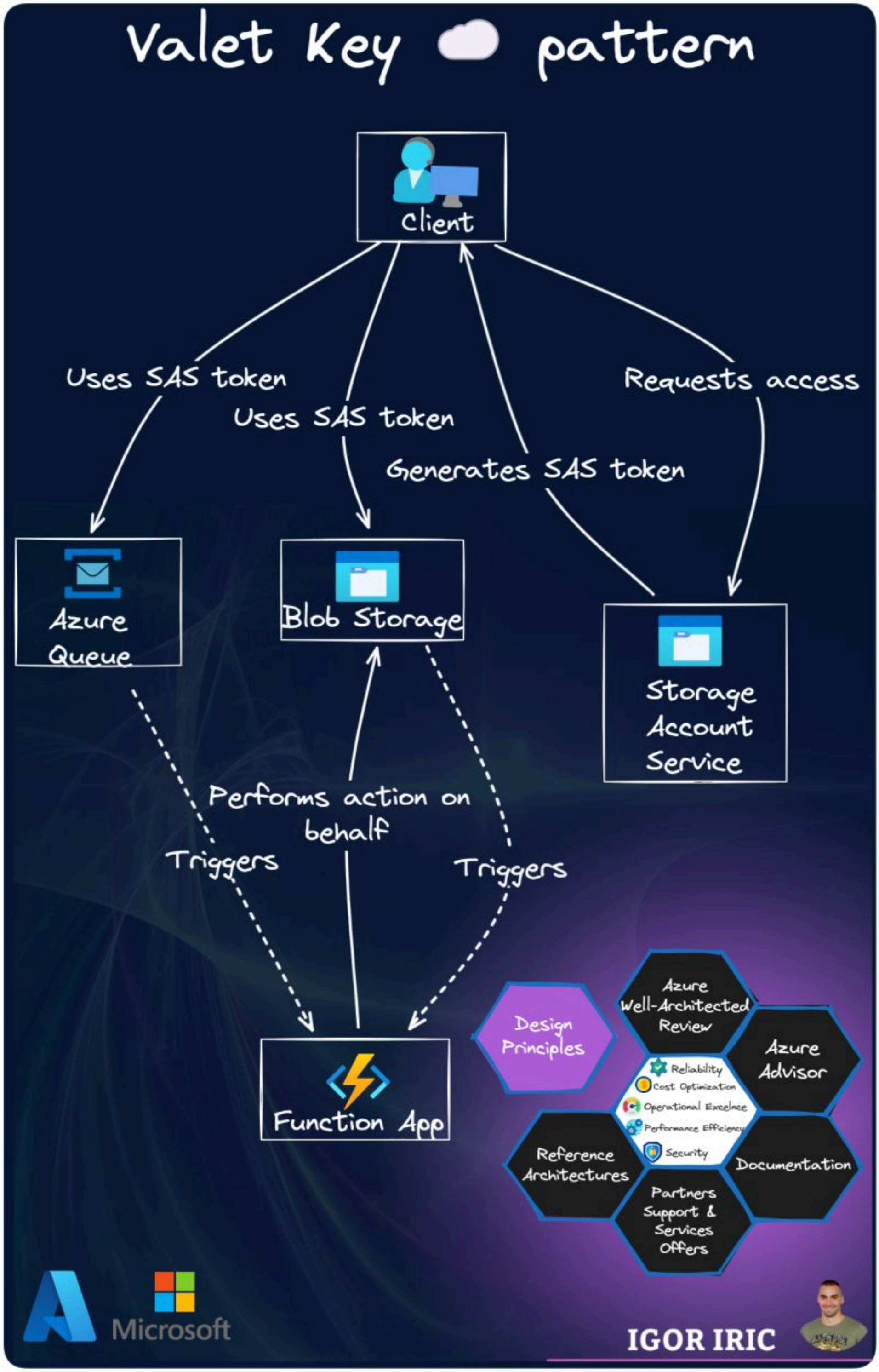
This Pattern in distributed systems is a strategy for managing transactions that span across multiple services, ensuring data consistency and integrity without locking resources. Unlike traditional transactions, Saga implements a sequence of local transactions, each with its own compensating action that can undo its changes in case of a failure.

✅ The pattern prevents system failures by not locking resources and using compensating transactions for rollbacks.
✅ Because the transactions are distributed, the system can handle more load by scaling out.
✅ Different services can be managed, updated, and scaled independently.

But keep in mind:

— Implementing and maintaining a saga can be complex due to the distributed nature of transactions.
— Tracking down issues across multiple services and steps can be challenging.
— The overhead of coordinating multiple services may introduce delays.

Valet Key pattern



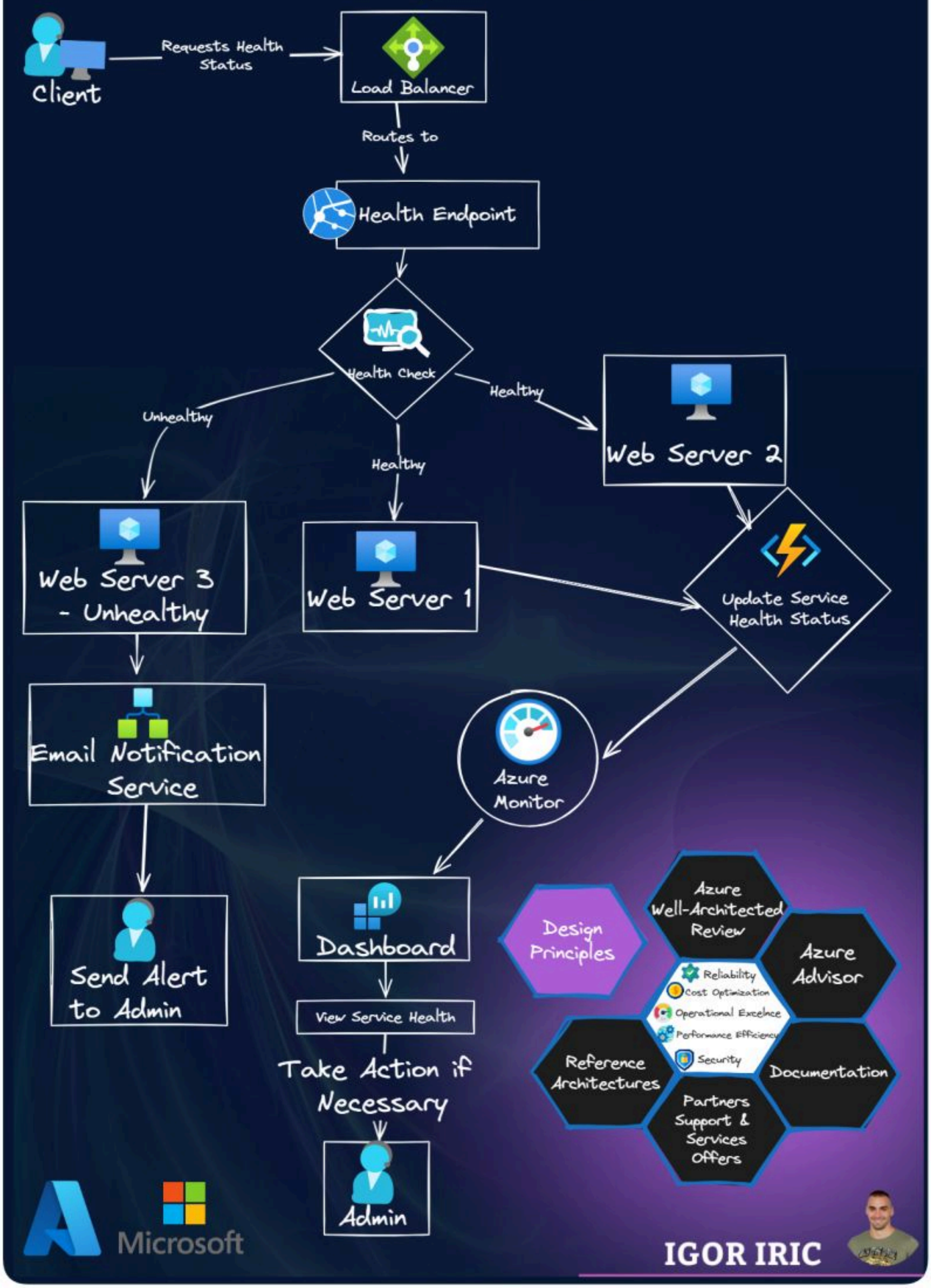
The Valet Key Pattern on Azure is all about providing direct access to a secure resource in a controlled manner, similar to how a valet key works with a car. It's a way to give clients temporary, restricted access to your resources without exposing more than necessary.

✅ Client requests access to a resource.
✅ Generate a Secure Access Signature (SAS) token which gives limited access.
✅ Client uses SAS token to interact directly with the Azure services like Blob Storage, Queue, or Table Storage.
✅ Triggers other actions, like logging with Azure Functions.

The benefits of using the Valet Key Pattern are:

✅ Direct access without going through your main service, reducing load and latency.
✅ Controlled permissions as SAS token can be set to limit actions to read, write, or delete, and it works for a set amount of time.
✅ Reduced server load.

Health Endpoint Monitoring pattern



It's essentially the health tracking system for your Azure services, always on the lookout to ensure they're running as expected.

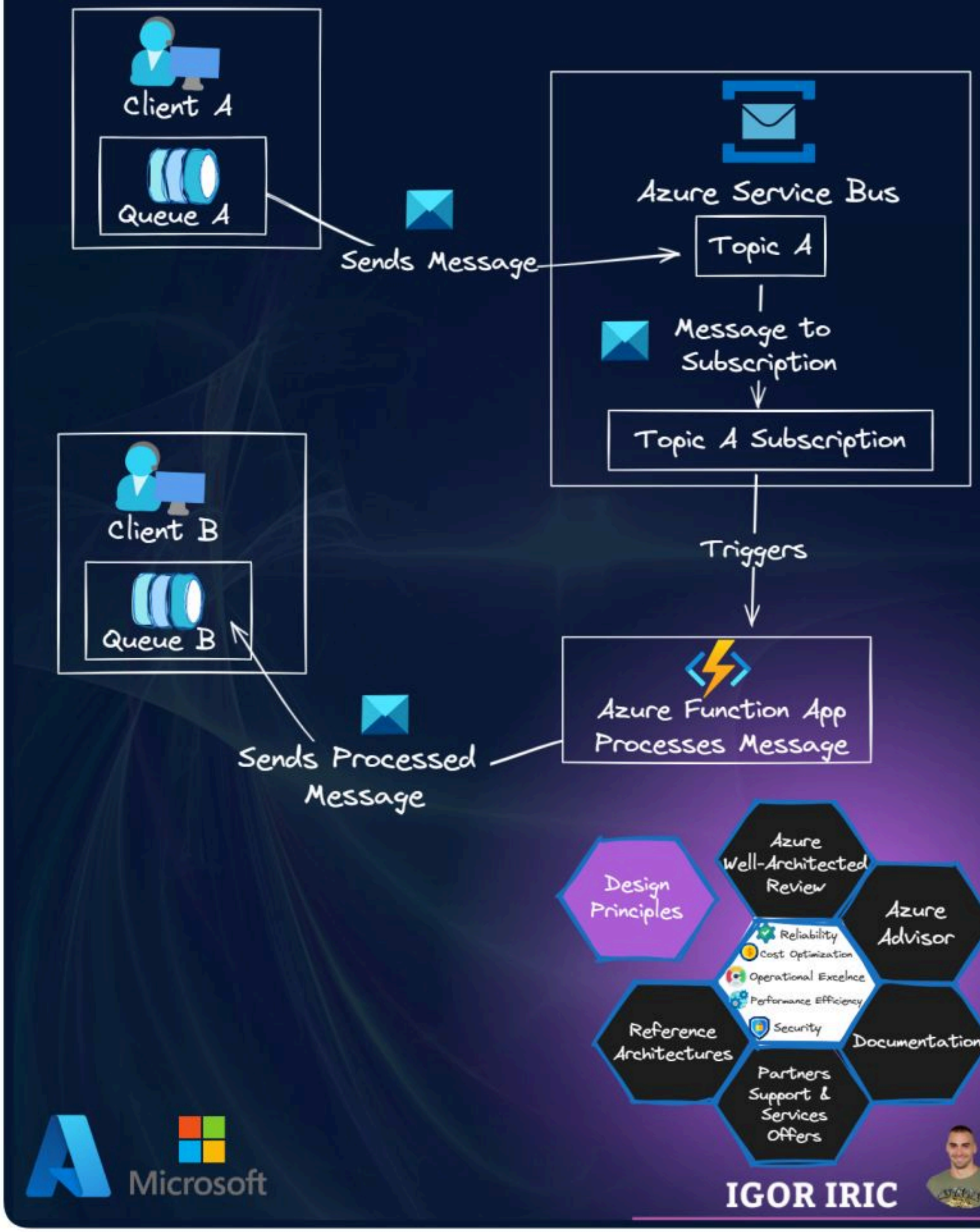
Here's how the pattern works, as shown in your diagram:

✅ Clients initiate a health check, asking "Are you OK?"
✅ The Load Balancer directs the request to the right spot.
✅ Web Servers are scrutinized to see if they're healthy.
✅ Those in good shape keep on working; those that aren't set off alerts.
✅ Admins get notified to take action, keeping things on track.

Why make this part of your Azure approach?

✅ Instant insights into your service health.
✅ Automated warnings mean quick fixes.
✅ Stay proactive, not reactive, to service downtime.

Messaging Bridge pattern



Microsoft

IGOR IRIC

The Messaging Bridge Pattern in Azure is designed to seamlessly transfer messages between clients and services, maintaining integrity and order.

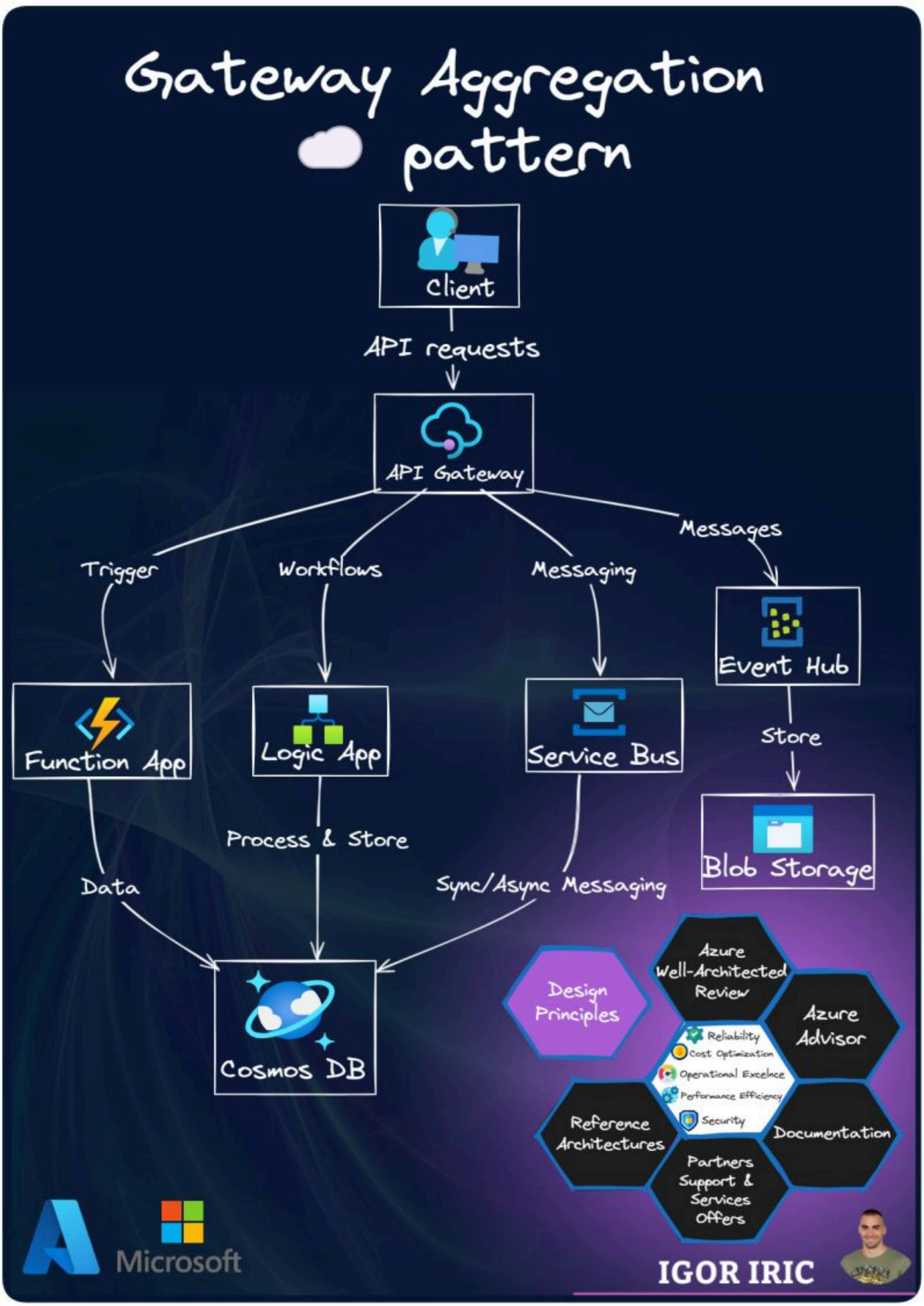
Here's the process:

✅ Client A drops a message off in their queue.
✅ Azure Service Bus then takes over, ensuring the message gets where it needs to go.
✅ Topic A Subscription holds the message briefly, ready for action.
✅ Azure Function App jumps into gear, processing the message.
✅ Client B picks up the final message from their own queue, ready to go.

What's so good about it?

✅ Makes sure every message finds its way correctly.
✅ Unifies communication across different services.
✅ Keeps message handling orderly and reliable.

Gateway Aggregation pattern



Gateway Aggregation Pattern focuses on reducing the number of client requests by combining them into a single aggregated request. This pattern simplifies client-side communication and consolidates data retrieval or processing.

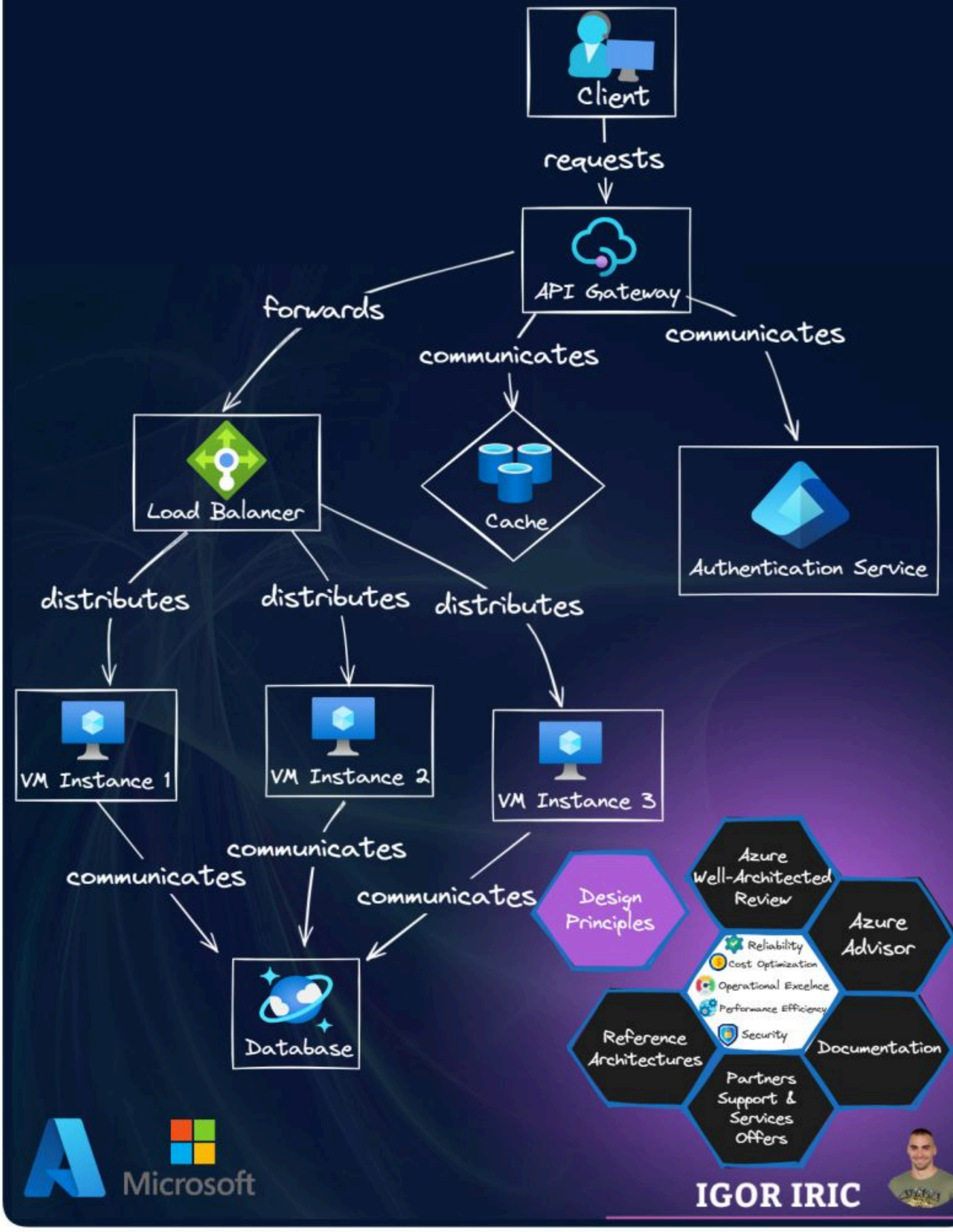
The benefits of using the Gateway Aggregation Pattern are:

✅ Reduces client complexity making a single call instead of multiple calls to various services.
✅ Improves response time.
✅ Services can be updated or maintained without affecting the client.

But here are some considerations:

— The API gateway can become complex as it handles more logic.
— If the gateway fails, the entire process is disrupted.
— The gateway must be designed to handle high loads without becoming a bottleneck.

Gateway Routing pattern



Microsoft

IGOR IRIC

The Gateway Routing Pattern on Azure organizes how requests are directed to your applications.

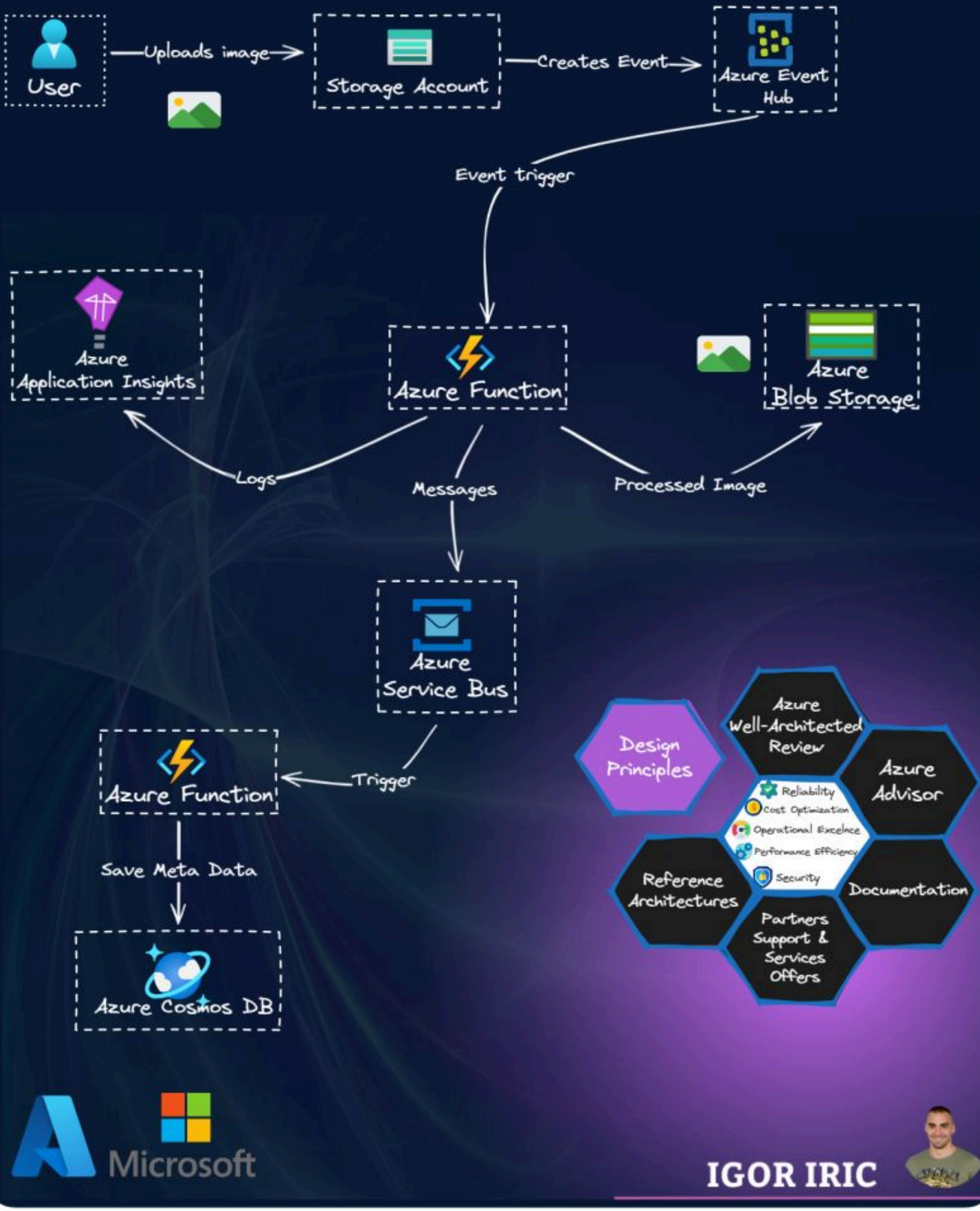
The benefits? Here they are:

✅ Centralized management of requests means easier monitoring and control.
✅ Load balancing ensures even distribution of traffic for better performance.
✅ Caching reduces load on backend services, giving users faster access to data.
✅ Authentication at the gateway enhances security across all services.

But keep in mind:

— Single point of setup complexity as the gateway handles multiple functionalities.
— Potential single point of failure if not designed with high availability.
— May introduce latency if the gateway becomes a bottleneck.

Pipes and Filters pattern



Microsoft

IGOR IRIC

The Pipes and Filters Pattern in Azure offers an efficient way to handle data by breaking down the process into manageable steps. It's like a production line, each part doing its bit to transform raw data into useful information.

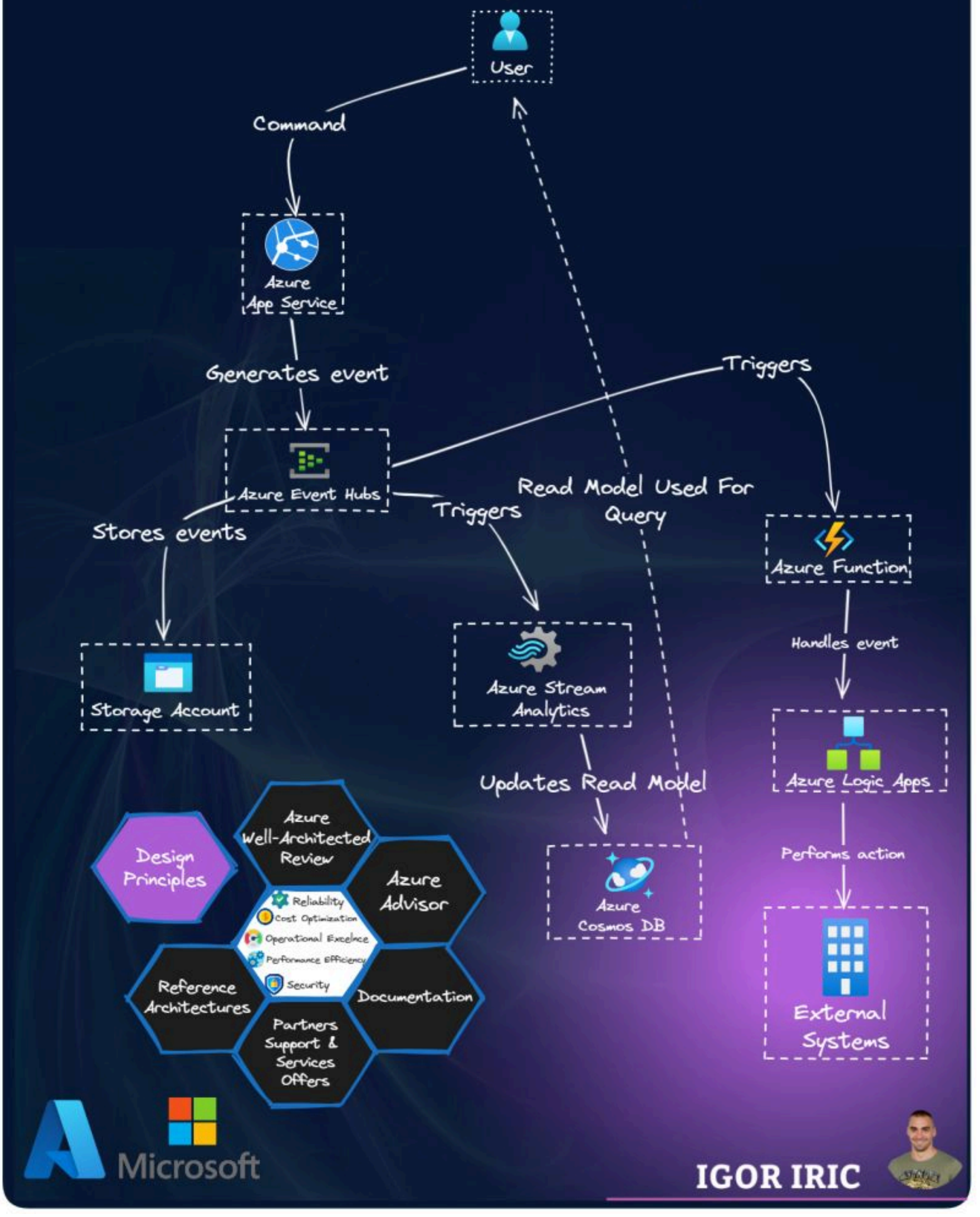
Why would you build your system with Pipes and Filters?

✅ Easier to maintain due to separation of concerns.
✅ Makes each part easy to replace or upgrade.
✅ Scales with you as your data grows.
✅ Lets you reuse parts for other jobs.

Some things you should keep in mind while implementing this pattern

— Fitting all the pieces together can take some work.
— May introduce extra latency as data passes through multiple services.
— Tracing and debugging through multiple components can be challenging.

Event Sourcing pattern



The Event Sourcing Pattern on Azure is a reliable method for capturing and storing each event in your applications, which can then reconstruct and analyze system state.

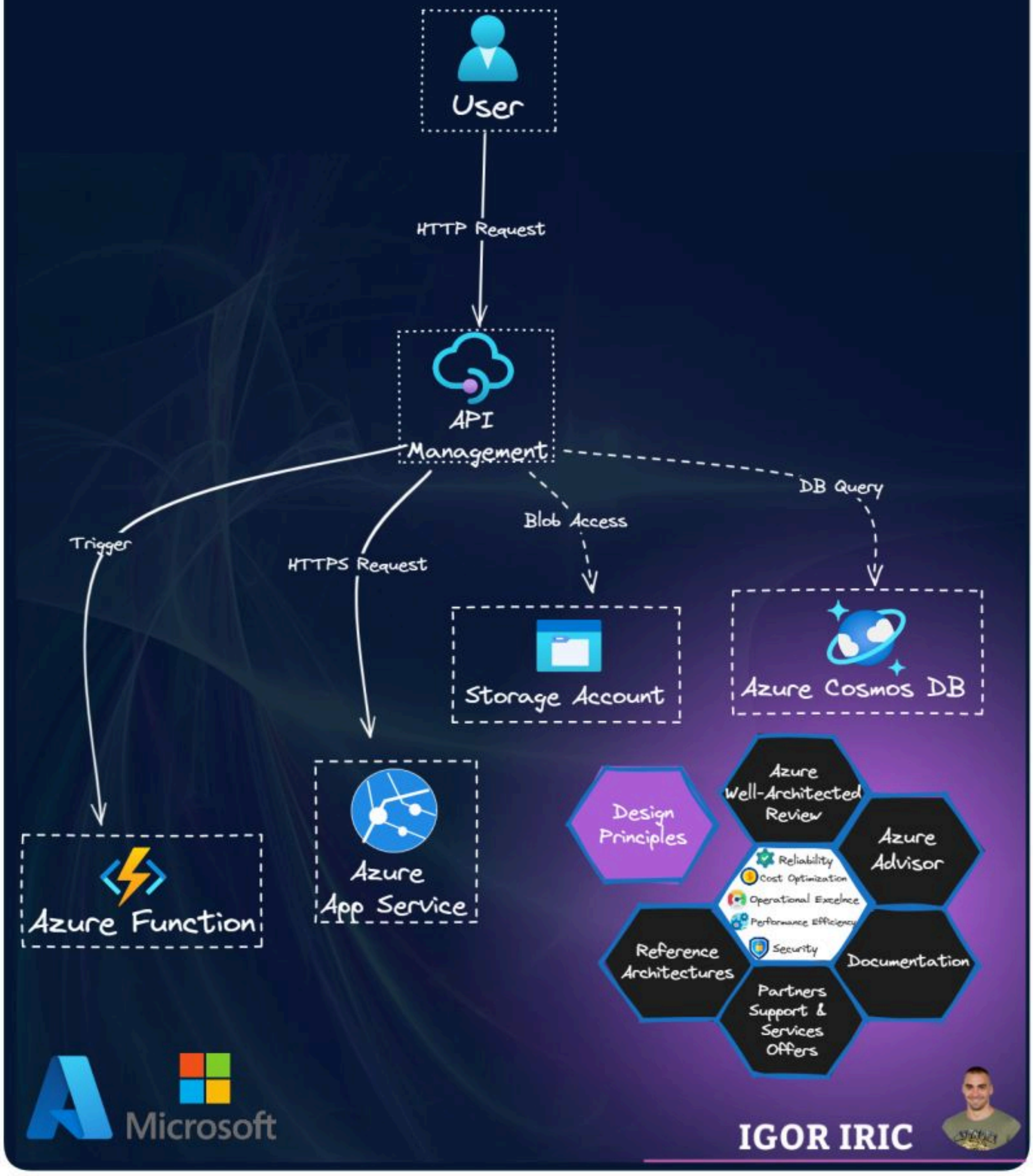
Why should you embrace Event Sourcing in your systems?

✅ Tracks every single change for full visibility.
✅ Helps with debugging by replaying events.
✅ Enables complex event processing and analytics.
✅ Facilitates integration with other systems and processes.

What are some points to consider?

— Can be complex to implement and maintain.
— May increase storage requirements over time.
— Requires careful design to handle large volumes of events.

Gateway Offloading pattern



The Gateway Offloading Pattern helps keep your services lightweight and lets an API gateway handle the common heavy lifting.

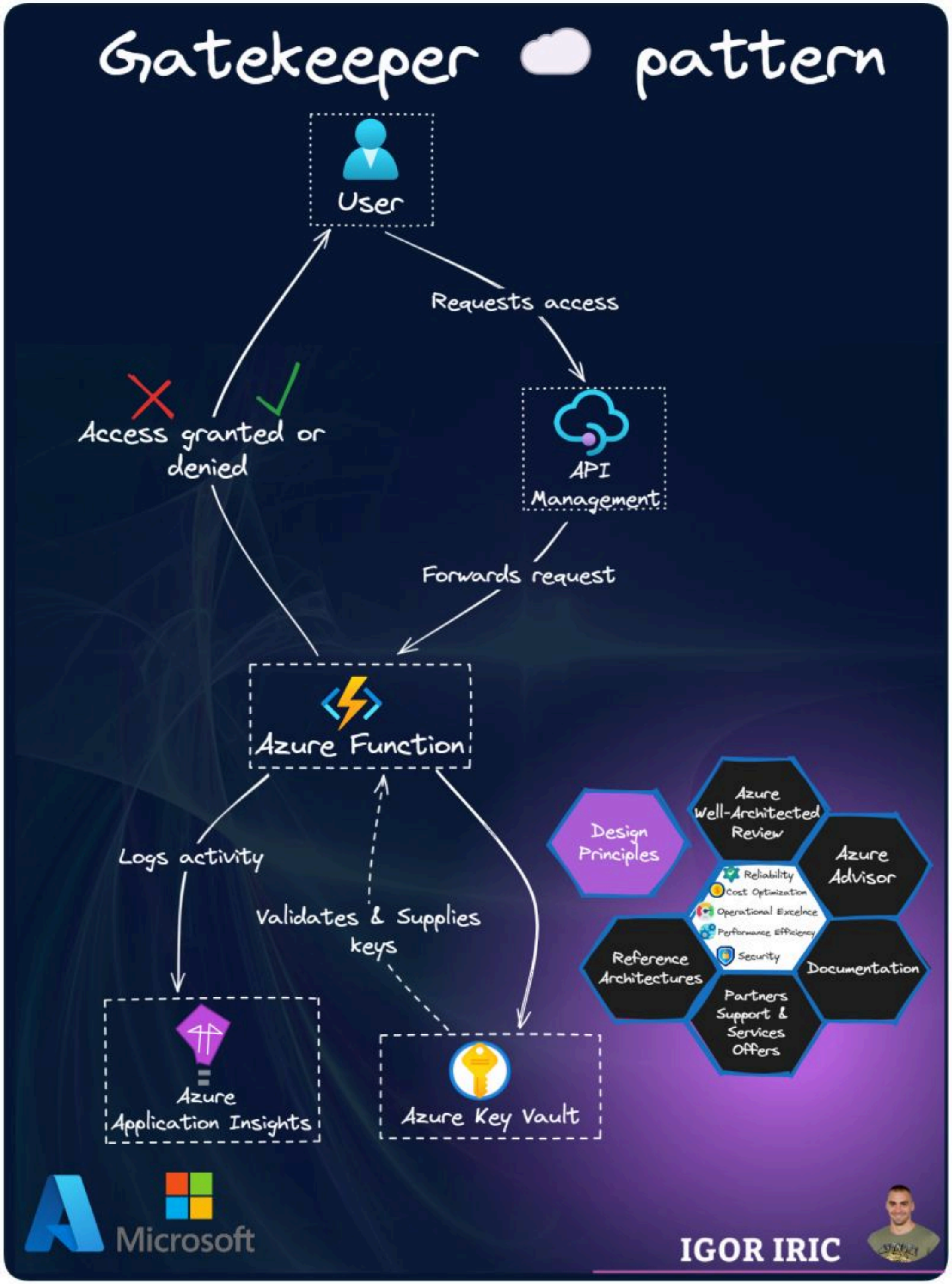
How does it works?

✅ The user starts the process by requesting access.
✅ API Management directs traffic, ensuring only valid requests pass.
✅ Azure Function does the ID check, making sure everyone's who they say they are.
✅ Azure App Service gets down to business, processing the logic needed.
✅ Storage Account and Azure Cosmos DB keep the data safe.

Why bring it into your Azure solution?

✅ Streamlines your microservices for easier maintenance.
✅ Puts repetitive tasks like authentication in one place.
✅ Boosts security with centralized control.

Gatekeeper pattern



Microsoft

IGOR IRIC

The Gatekeeper Pattern in Azure acts as a checkpoint for managing access requests, ensuring only the right people get through.

Here's how it keeps your system safe:

✅ API Management stands as the front door, handling access requests.

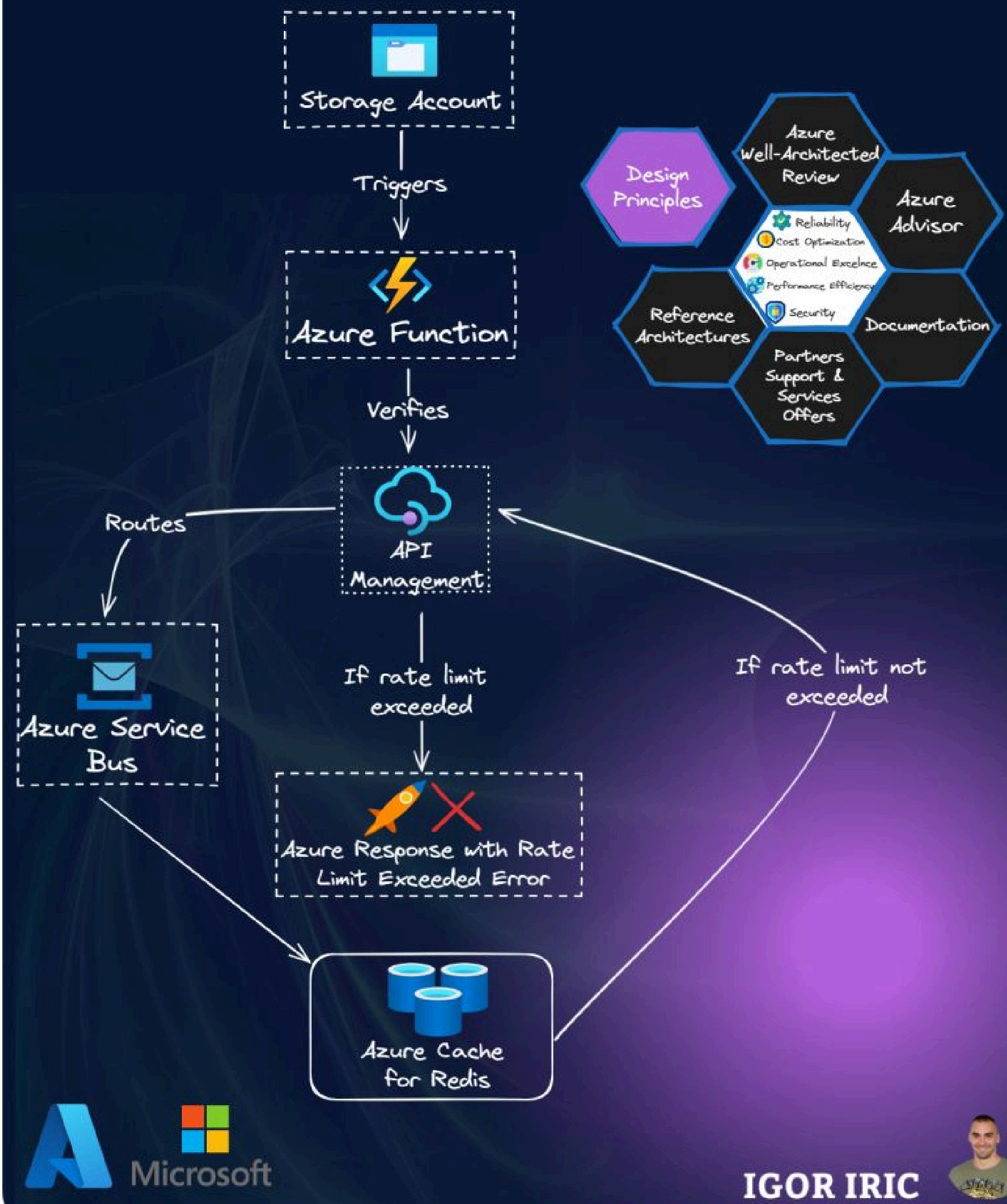
✅ Azure Function is the security guard, checking IDs and making sure everything's in order.

✅ Azure Key Vault is the key box, holding all the secrets the guard needs to verify identities.

✅ Access Granted or Denied is where the guard opens the door or keeps it shut, based on what they find.

✅ Azure Application Insights is the logbook, keeping a record of who's requesting.

Throttling pattern



Microsoft

IGOR IRIC

The Throttling Pattern in Azure is a smart way to handle traffic in your applications by controlling the number of processed requests. It prevents your system from being overloaded, making sure your system doesn't get overwhelmed.

Here's how it works:

✅ Storage Account holds your data securely.

✅ Triggers alert your system to new requests.

✅ Azure Function checks out the request.

✅ API Management is the traffic light, saying 'yes' or 'no' to requests.

✅ Routes are the paths for requests to go when the light is green.

✅ Rate Limit Exceeded means the system says 'stop' to prevent a traffic jam.

Publisher-Subscriber pattern

Publisher

Publish Message

Topic on Azure Service Bus

Subscription 1

Subscription 2

Subscriber 1

Subscriber 2

Process Message

Process Message

Azure Cosmos DB Data Store

Storage Account

Design Principles

Azure Well-Architected Review

Azure Advisor

Reliability

Cost Optimization

Operational Excelnce

Performance Efficiency

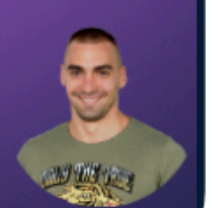
Security

Reference Architectures

Documentation

Partners Support & Services Offers

Microsoft



IGOR IRIC

Pros

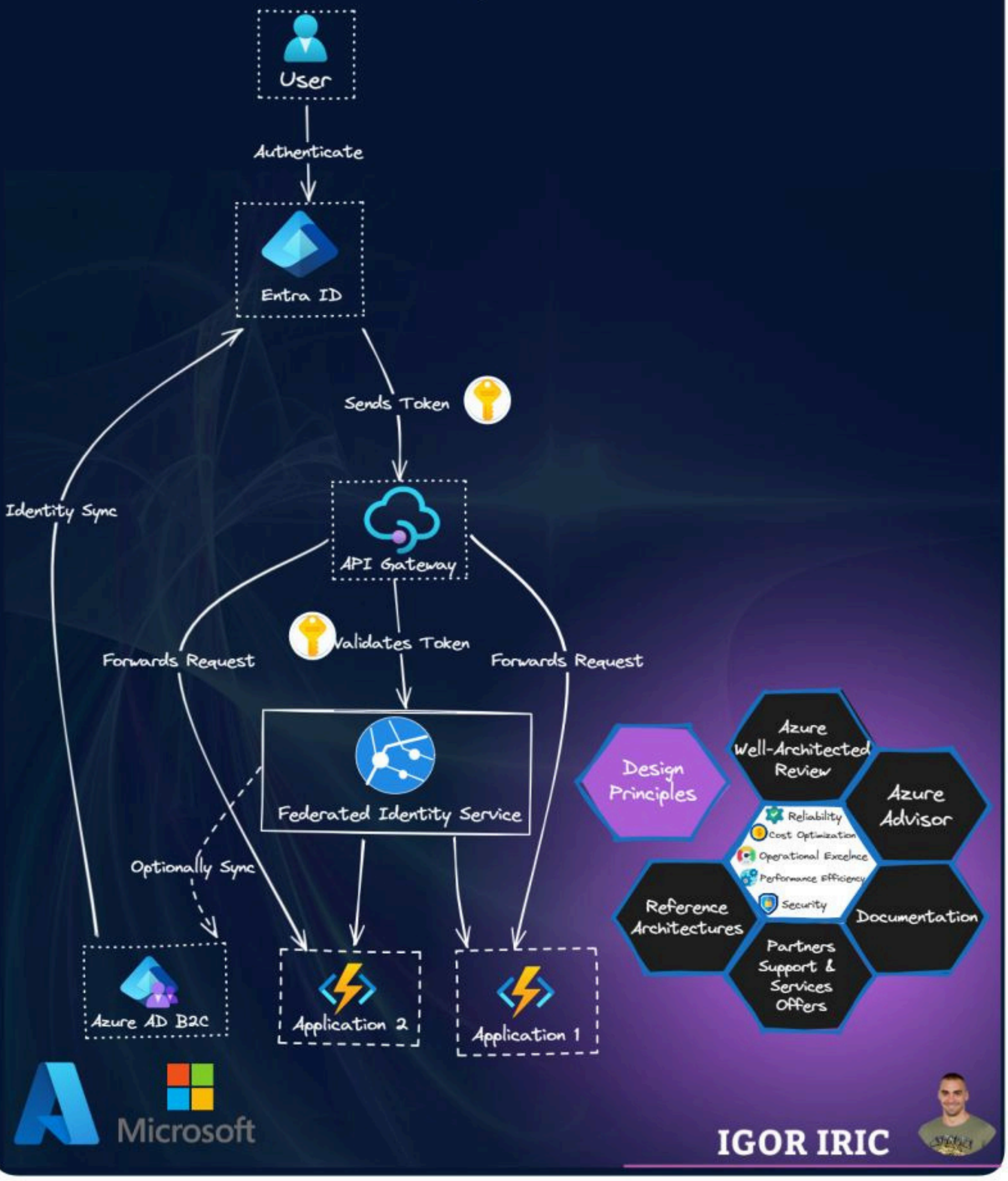
✅ Decoupling – Publishers and subscribers operate independently, knowing nothing about each other, which simplifies app maintenance and scalability.

✅ Scalability – You can easily scale the number of publishers or subscribers without affecting the rest of the system.

✅ Flexibility – Subscribers can choose what information they need, and publishers can send messages without worrying about who receives them.

✅ Reliability – Azure Service Bus guarantees message delivery, even if subscribers are temporarily unavailable.

Federated Identity pattern



✅ The user logs in just once using an identity provider.

✅ Azure's identity management service steps in, handling user identities and permissions.

✅ API Gateway checks that the user's token is valid, so no one gets in who shouldn't.

✅ Federated Identity Service is a central hub that makes sure the user's identity can be used across different apps.

✅Whether it's App 1, 2, or 3, they all know the user is who they say they are, thanks to the identity service.

Anti-corruption Layer pattern

Microsoft

IGOR IRIC

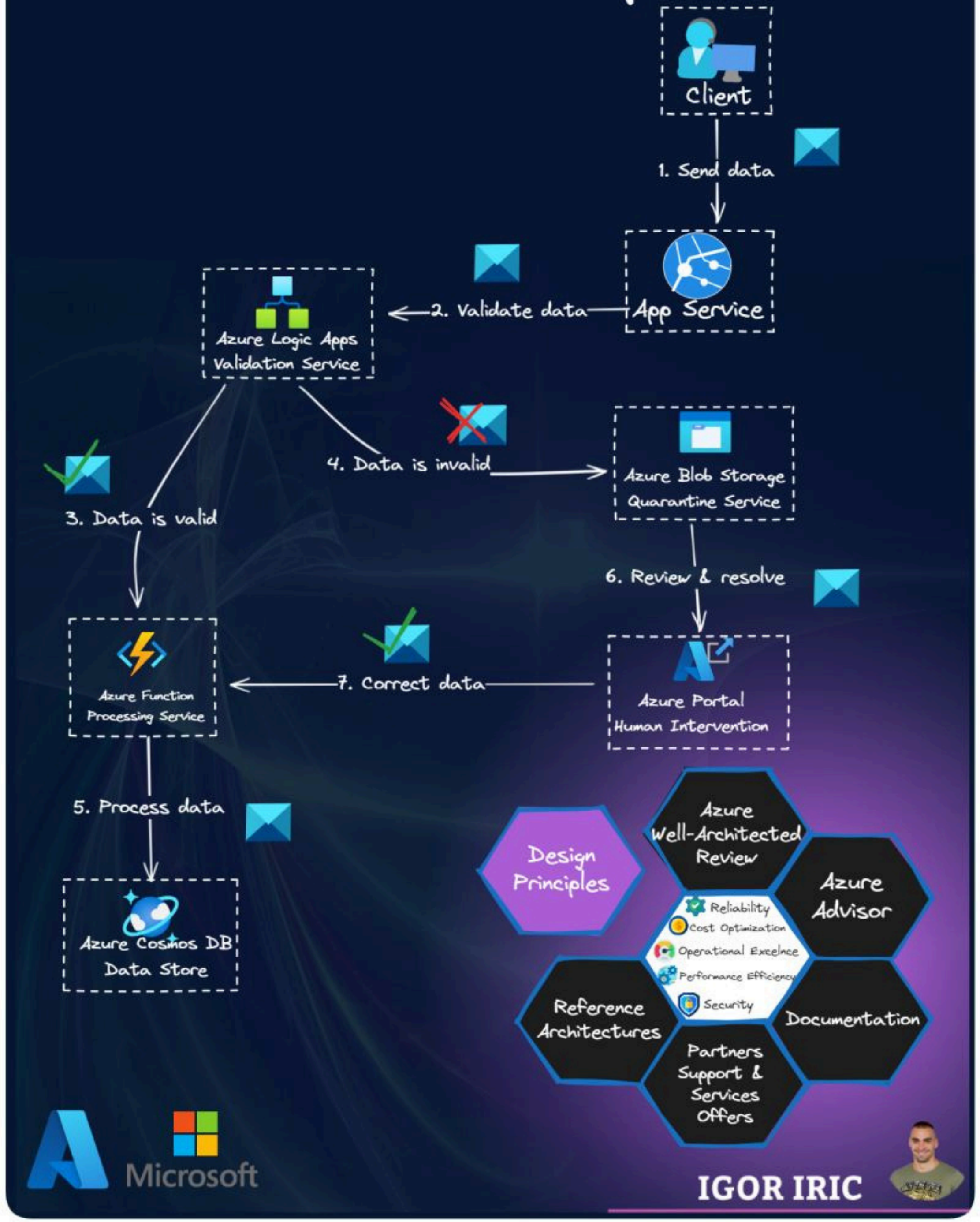
✅ Azure API Management takes requests from clients and directs them where they need to go.

✅ Azure Functions takes the new requests and converts them into a language the old system understands and also processes messages from and to Azure Service Bus.

✅ Azure Service Bus carries the translated requests to the new system and makes sure the responses get back.

✅ Cosmos DB is for new data, and Azure SQL holds the old data.

Quarantine pattern



Microsoft

IGOR IRIC

✅ Clients send data to your App Service, acting as your front end.

✅ Azure Logic Apps, your Validation Service, checks if the data is good or bad.

✅ Good data passes forward to Azure Function, your Processing Service, doing the transformation and saving of the data.

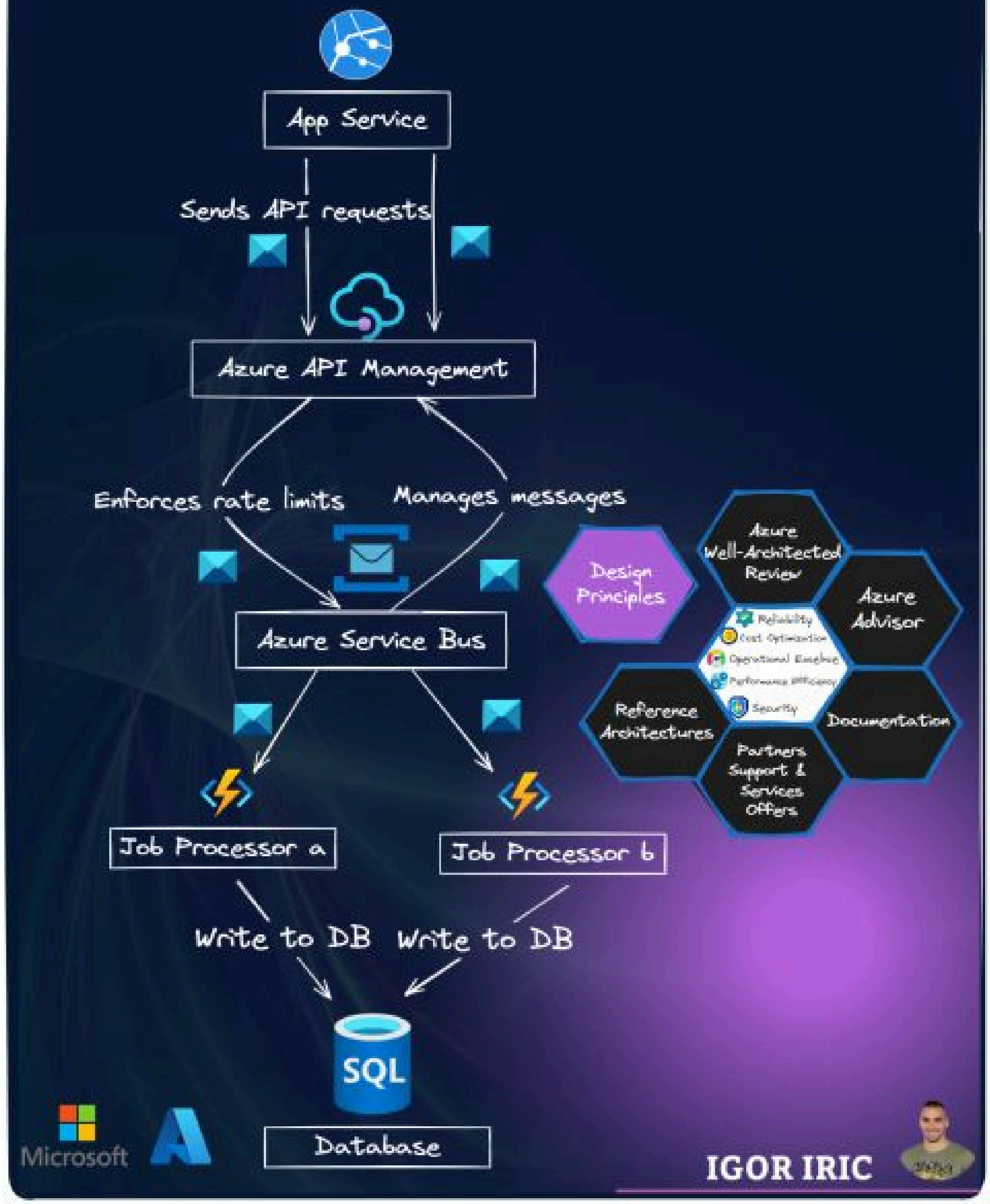
✅ Once the data is processed, it rests in Azure Cosmos DB, your Data Store.

✅ The bad data? It's sent to Azure Blob Storage, the Quarantine Service, keeping your environment stable and healthy.

✅ The quarantined data isn't discarded or deleted it’s reviewed and resolved through the Azure Portal, giving it a second chance.

✅ Corrected data re-enters the flow, back to the Processing Service, to fulfill its purpose.

Rate Limiting pattern



✅ App Service is the starting point, handling your business operations.

✅ Azure API Management steps in, putting caps on API requests to prevent traffic jams.

✅ Azure Service Bus orchestrates the message traffic, keeping everything in check.

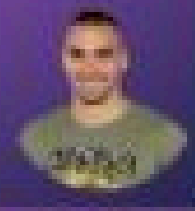
✅ Job Processors, a & b, stay busy processing tasks, making sure work gets done.

✅ SQL Database stands strong, storing all the outcomes of your app's hard work.

Sidecar pattern

Write to DB

SQL Elastic Pool



IGOR IRIC

✅ Main Application remains the core, focused on its business logic.

✅ Sidecar App attaches to the main app, to include logging, monitoring, and more.

✅ Azure Blob Storage integrates with the sidecar for data management.

✅ Azure Monitor integrates with the sidecar to log activities without breaking everything.

✅ Azure Metrics syncs for real-time performance and health checks.

✅ The main app continues uninterrupted, efficiently writing to an SQL Elastic Pool.

Strangler Fig pattern

IGOR IRIC

✅Identify service to replace in the legacy system.

✅Create replacement service that replicates the functionality of the legacy feature.

✅Route traffic to new service instead of the legacy one using API Gateway or Router.

✅Maintain routes to legacy features that have not yet been replaced.

✅Use tools like Azure Advisor to ensure the new service is reliable, performant, and cost-effective.

✅Once all features have been replaced, the new system should be complete and fully functional.

✅After a period of parallel running and once confident in the new system's capabilities, decommission the legacy system.

CQRS cloud pattern

IGOR IRIC

✅ Clients issue commands or queries, initiating interactions with your system.

✅ API Management acts as a gateway, intelligently routing commands and queries.

✅ Command Handlers are tasked with processing requests to change data, ensuring each action is captured with precision.

✅ Azure Event Hubs plays a crucial role, distributing information about state changes across the system.

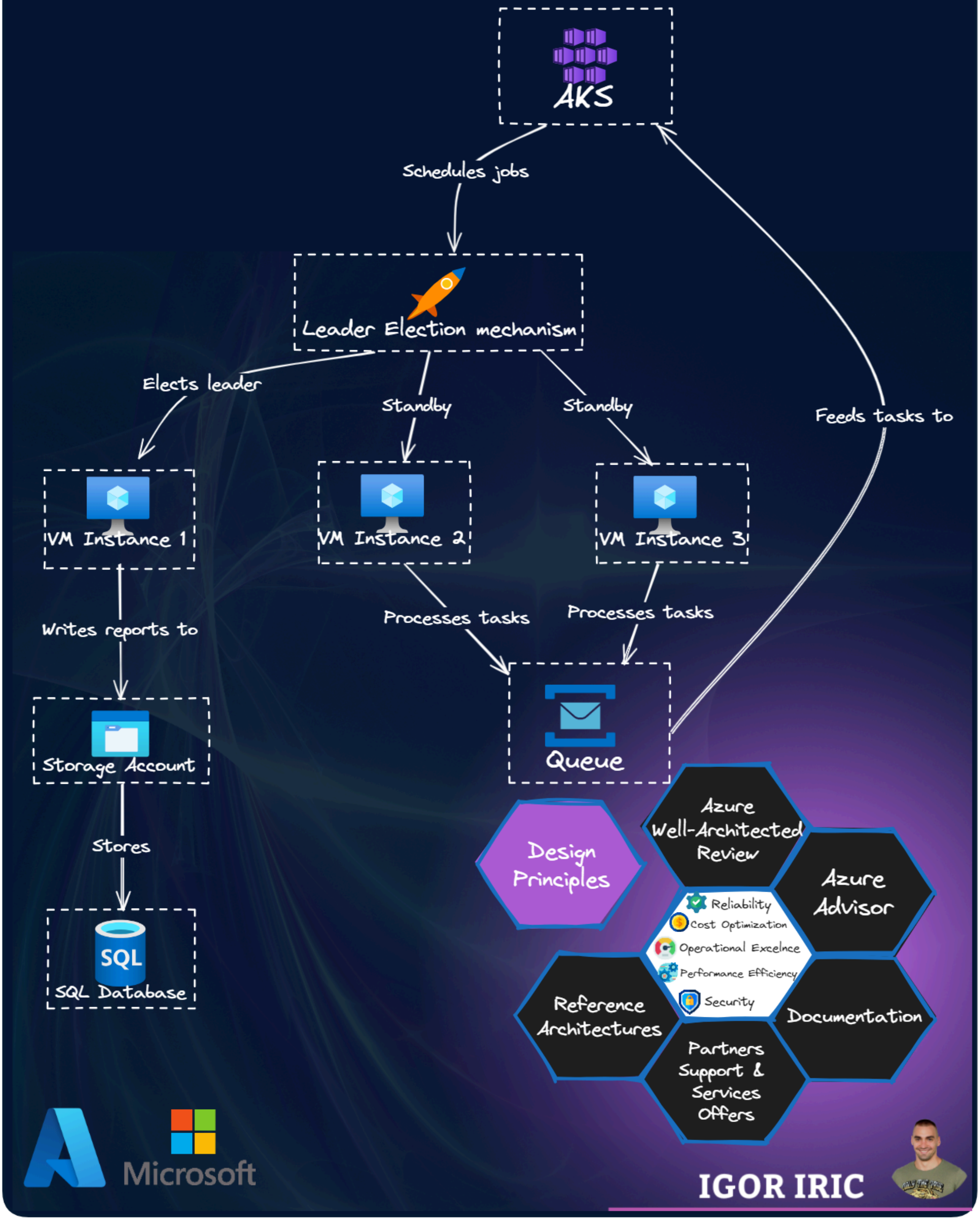
✅ Command DB is the source of truth for writes, thoroughly tracking state mutations.

✅ Query Handlers focus mostly on retrieval, delivering data rapidly and efficiently.

✅ Query DB is optimized for reads, providing quick access to the latest information.

✅ Update Processors keeps the data landscape consistent, aligning the Command and Query sides.

Leader Election pattern



Microsoft

IGOR IRIC

✅ Azure Kubernetes Service (AKS) is like manager, setting up jobs for the work.

✅ The Leader Election mechanism is the audition process, picking out the the VM to lead the process.

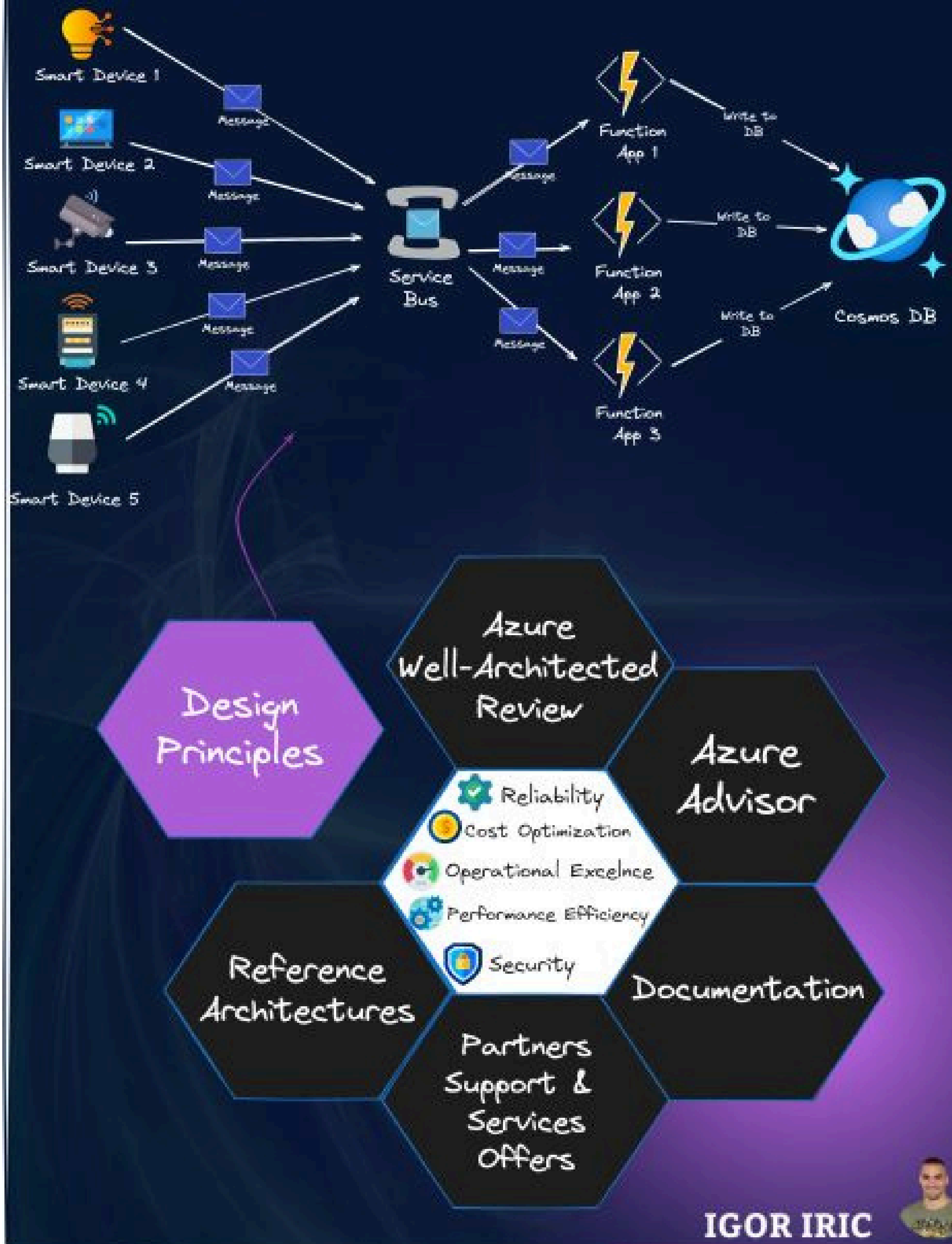
✅ One VM steps into the spotlight, taking the lead on doing all the important jobs it's elected to finis.

✅ The other VMs aren't out of the job – they're the supporting the work, tackling all the other tasks.

✅ The leading VM then puts the finishing touches by safely storing all the data in the storage room.

✅ While this is happening, the rest of the VM's keeps the work running smoothly, making sure everything's ready for the next job.

Competing Consumers cloud pattern



IGOR IRIC

5 smart IoT devices sending messages to Azure Service Bus.
– These messages are picked up by three Azure Functions.
– Each Function processes messages and sends metadata to Azure Cosmos DB.

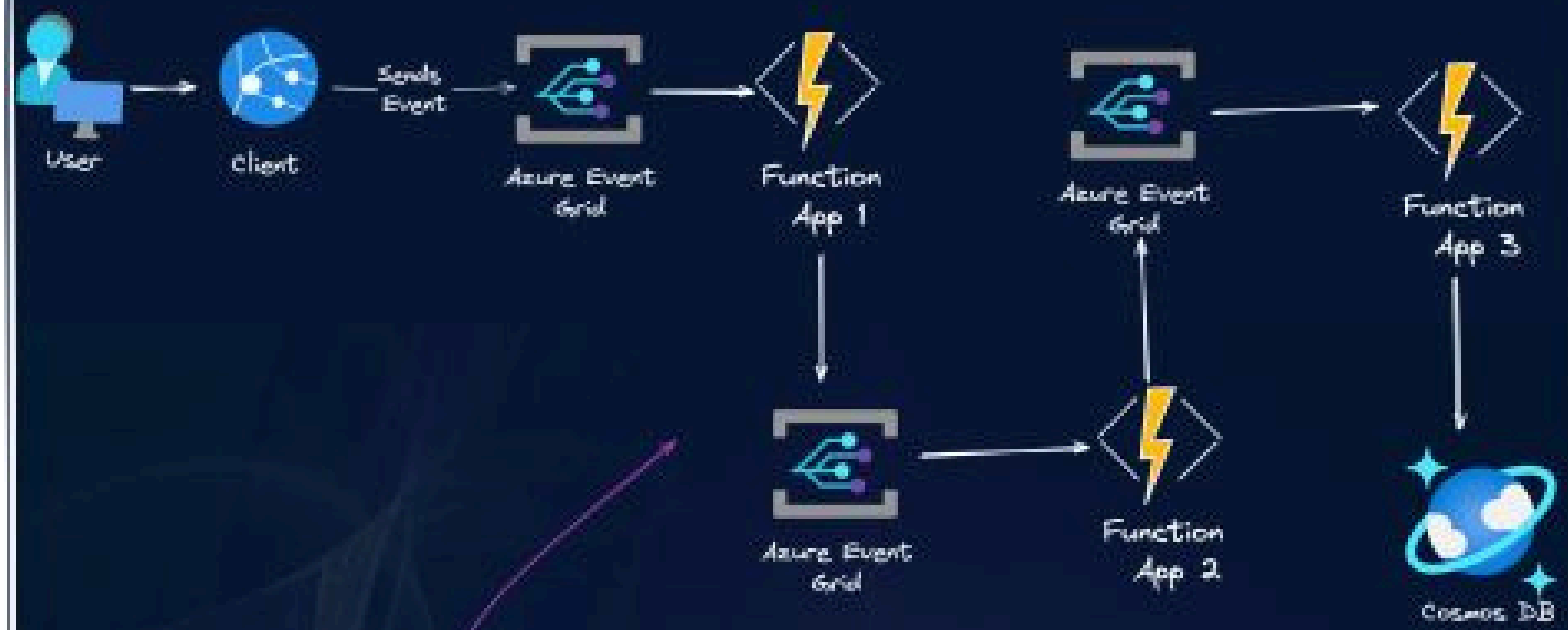
Now, why use the Competing Consumers pattern here?

✅ Efficient message processing: Multiple consumers handle messages concurrently, speeding up processing.

✅ Load balancing: The workload is distributed across consumers, preventing overload.

✅ Scalability: Easily scale the number of consumers based on the workload.

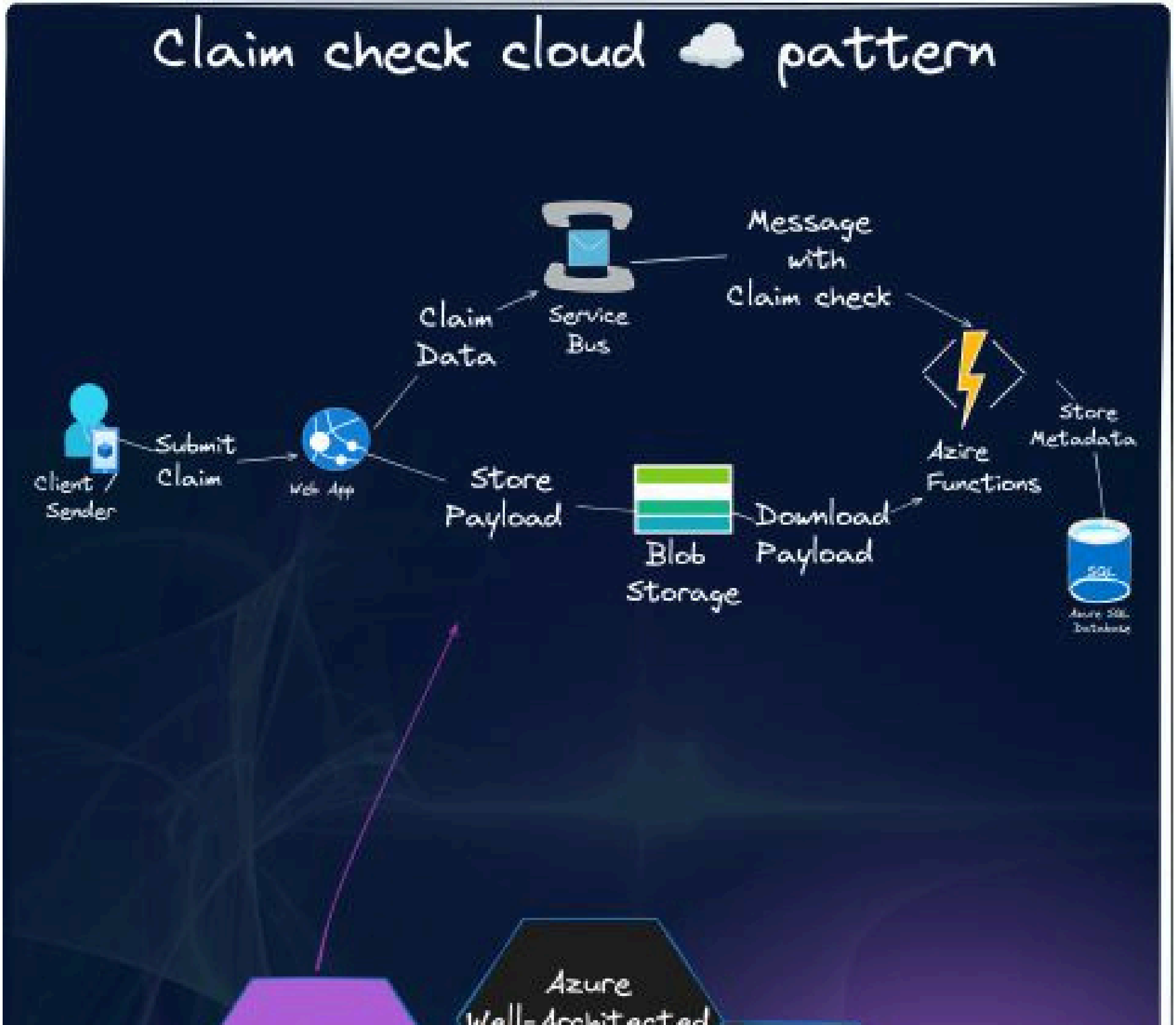
Choreography cloud pattern



IGOR IRIC

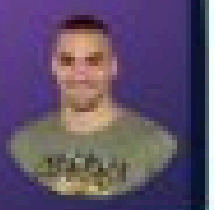
Search over my profile for more explanations and hit Like, Repost and comment.

Claim check cloud pattern



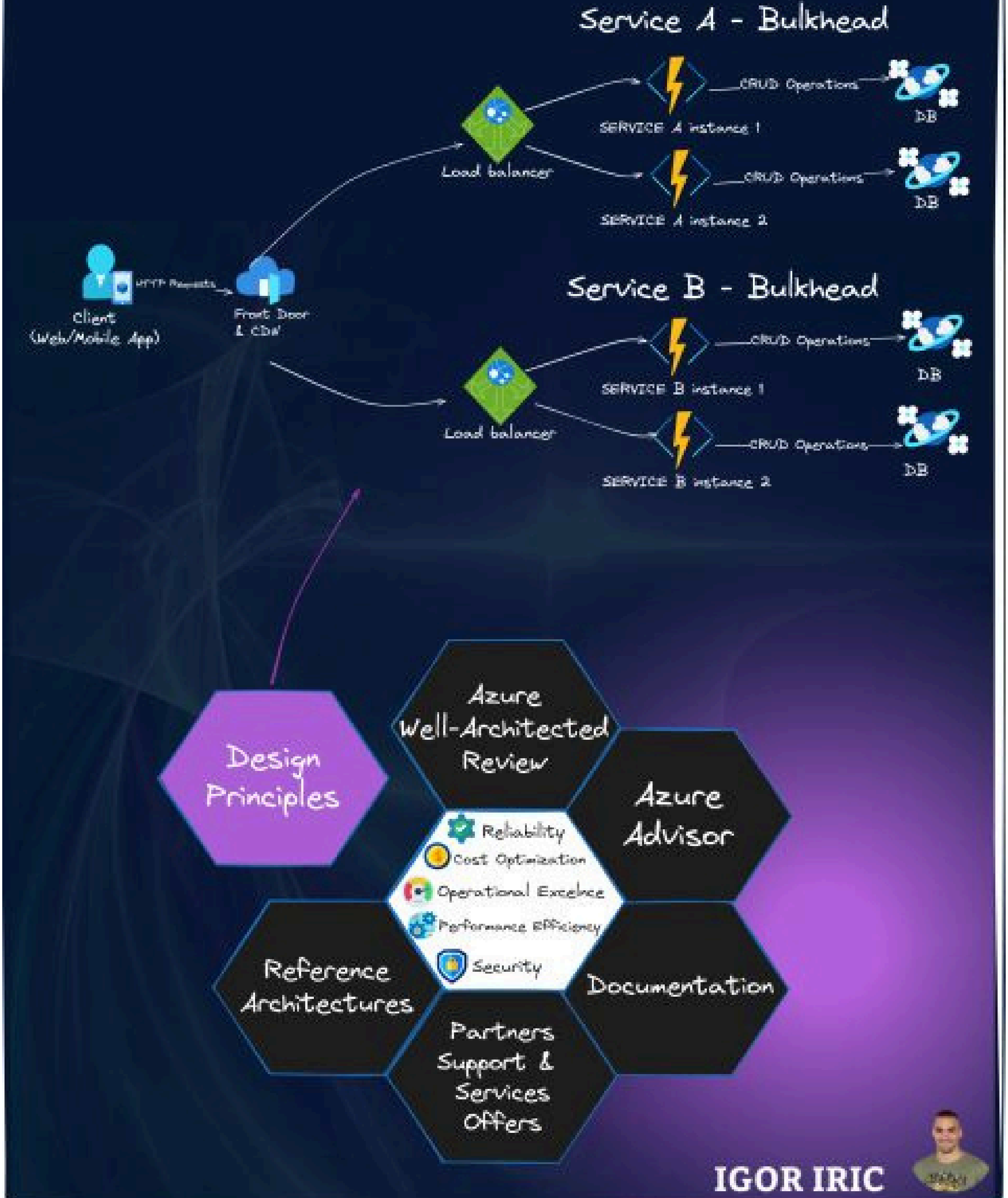
IGOR IRIC

Circuit breaker cloud pattern

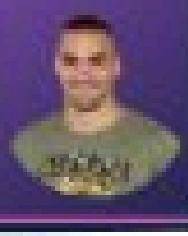


IGOR IRIC

Bulkhead cloud pattern

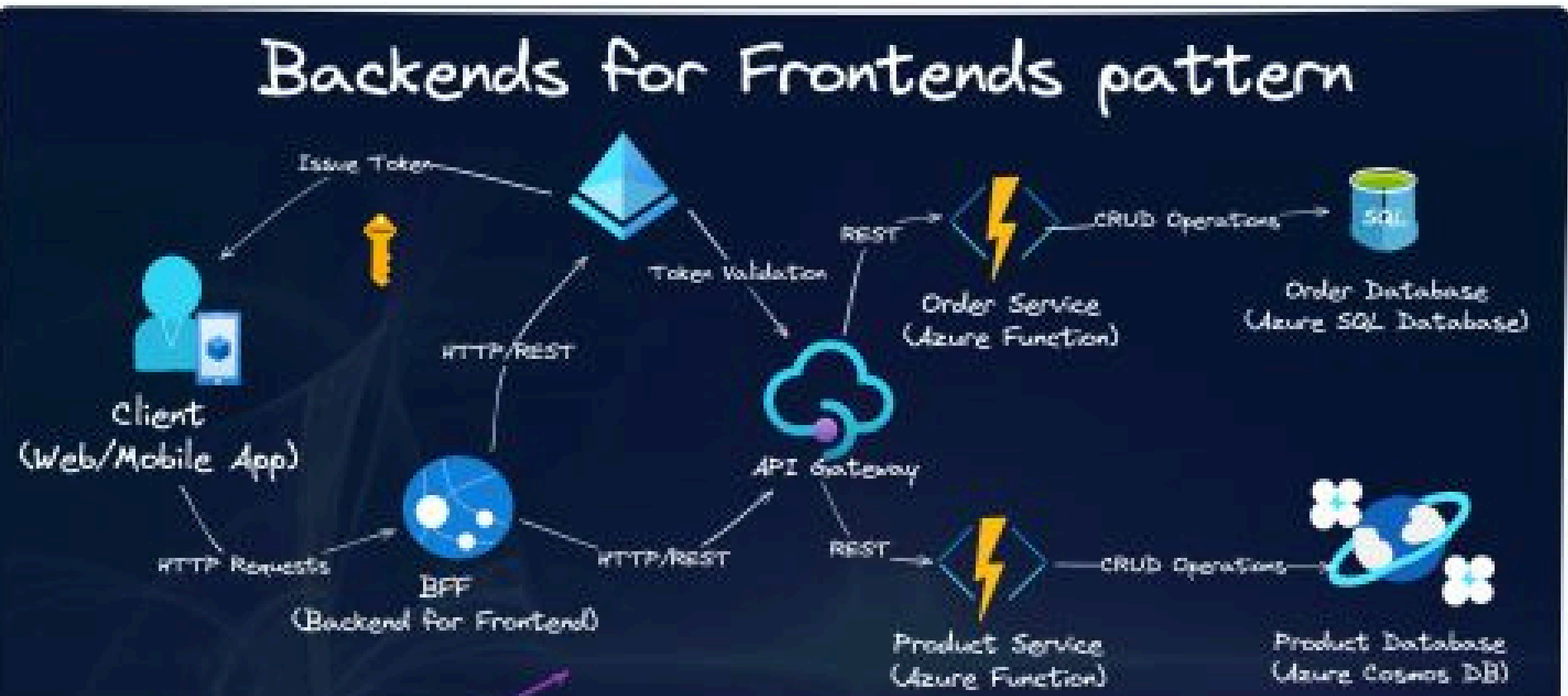


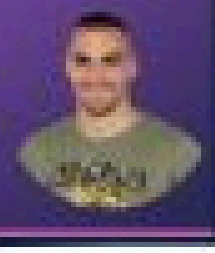
Cache-Aside cloud pattern



IGOR IRIC

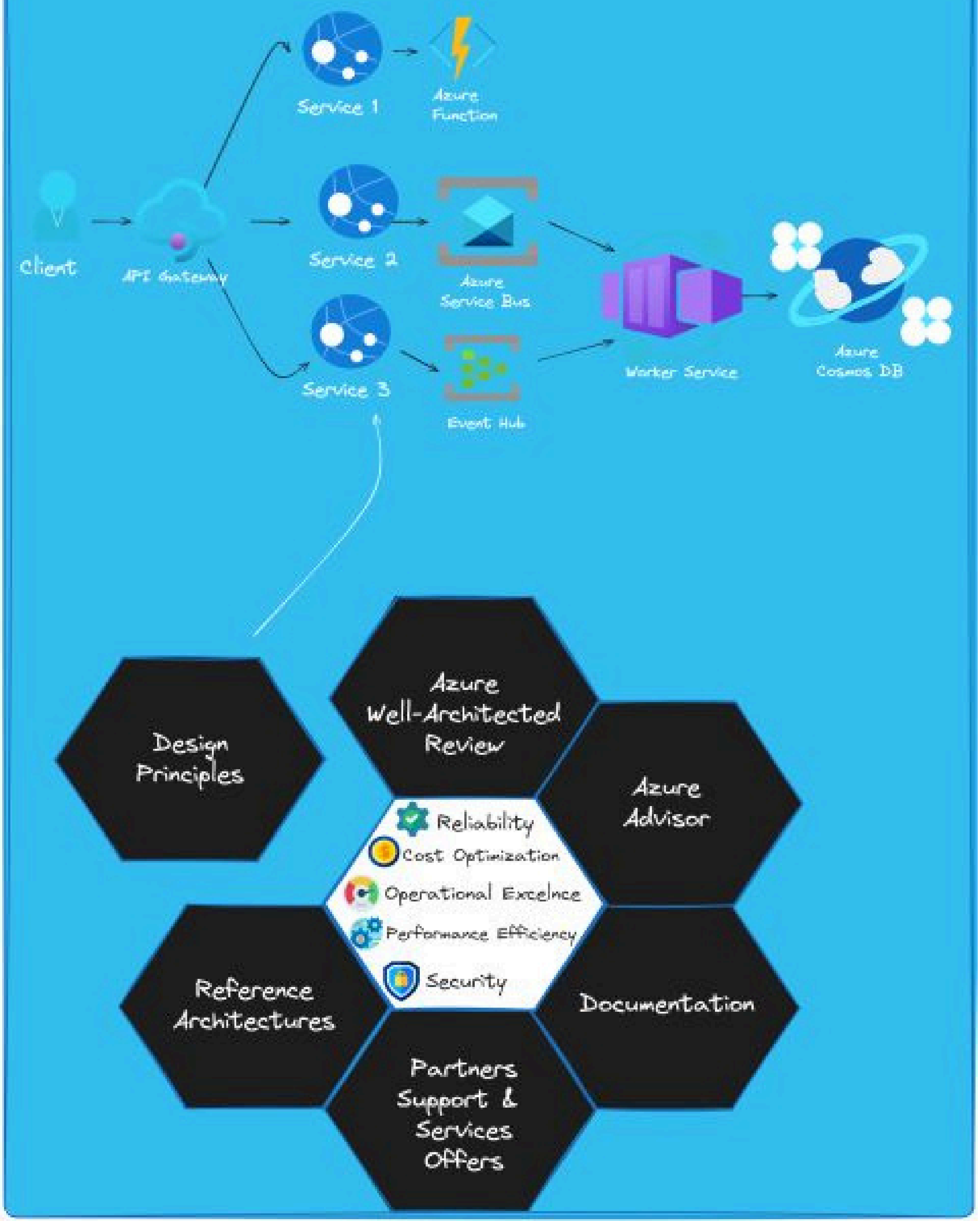
Backends for Frontends pattern





IGOR IRIC

Ambassador Design Pattern



I hope you find these tips helpful!

Let me know if you have any questions.

Follow

Like

Comment

Igor Iric