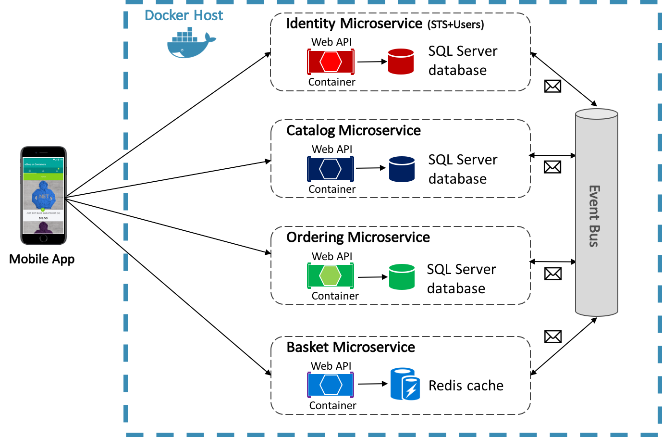
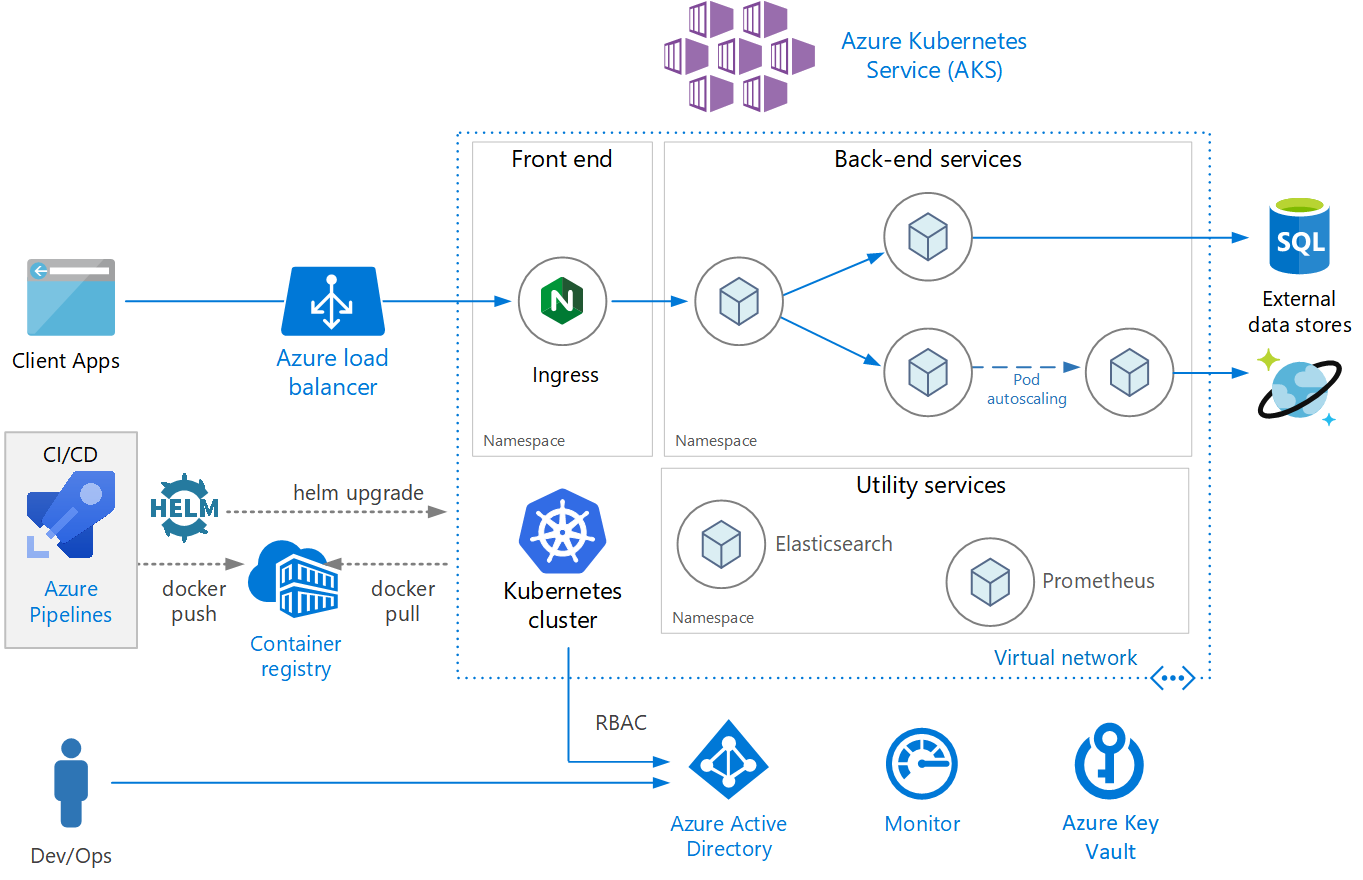
**Microservices Azure Automation guide**

**Reference Architecture:**





**Steps**

|  |  |  |
| --- | --- | --- |
| **1** | Containerized(Install Docker on Local machine) | [**https://learn.microsoft.com/fr-fr/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices**](https://learn.microsoft.com/fr-fr/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices) |
|  | Common MicroserviceRule | **Microservice Design Principal**   1. high cohesion and low coupling. 2. object-oriented design we follow the SOLID principles 3. microservice design we propose developers follow the “IDEALS”: interface segregation, deployability (is on you), event-driven, availability over consistency, loose-coupling, and single responsibility. 4. **12 Factor** 5. **Key components of Microservice architecture**   Microservices,Containers,Service mesh,Service discovery,API gateway. |
| 2 | Create Microservices | Microservices conteneurisés - Xamarin | Microsoft Learn  **Basket Microservice:**  [**https://medium.com/@FurryMogwai/building-a-basket-micro-service-using-asp-net-core-and-akka-net-ea2a32ca59d5**](https://medium.com/@FurryMogwai/building-a-basket-micro-service-using-asp-net-core-and-akka-net-ea2a32ca59d5) |
| **3** | Authentication & Authorization | High-level overview of the sign-in process  Combination of OpenID Connecter and OAuth 2.0 combines the two fundamental security concerns of authentication and API access, and IdentityServer 4  [**https://learn.microsoft.com/fr-fr/xamarin/xamarin-forms/enterprise-application-patterns/authentication-and-authorization**](https://learn.microsoft.com/fr-fr/xamarin/xamarin-forms/enterprise-application-patterns/authentication-and-authorization) |
|  | One-to-many communication |  |
|  |  |  |
|  |  |  |
|  | How do you handle microservices failure? | 1. Use asynchronous communication (for example, message-based communication) across internal microservices. ... 2. Use retries with exponential backoff. ... 3. Work around network timeouts. ... 4. Use the Circuit Breaker pattern. ... 5. Provide fallbacks. ... 6. Limit the number of queued requests. |
|  | Event-based Microservices: Error Handling | connectivity issues, serialization/deserialization issues, downstream system outages, peer system outages, bugs,   |  |  | | --- | --- | | Transient errors: | Non-transient errors: | | Connectivity issuesSystem outages | Serialization/deserialization issuesBugs |   <https://medium.com/usertesting-engineering/event-based-microservices-error-handling-7c84e3cb1332#:~:text=When%20a%20system%20is%20distributed,and%20the%20list%20goes%20on>. |

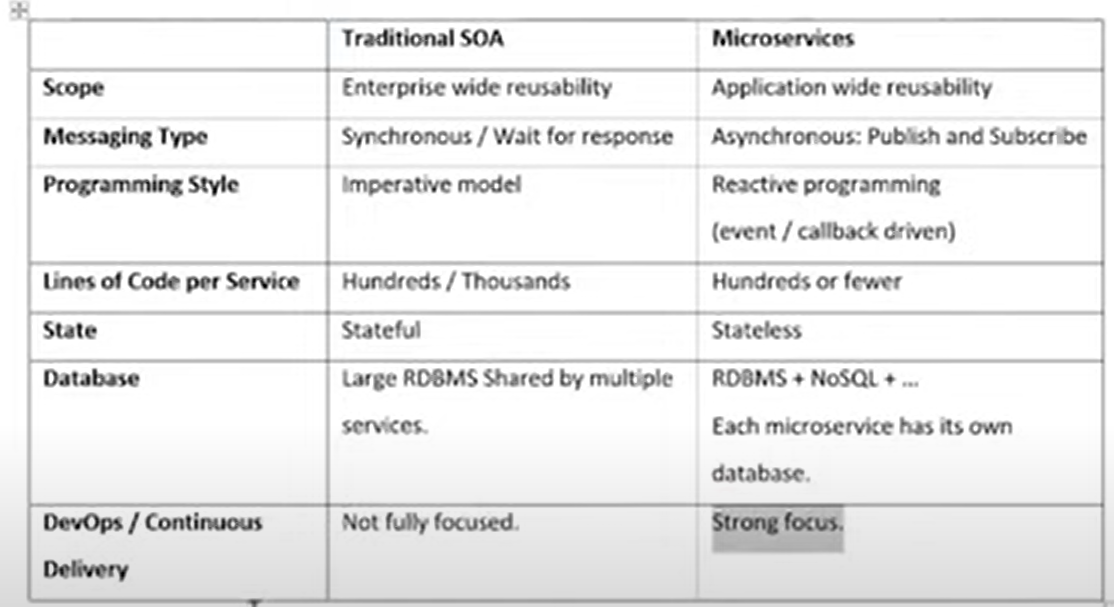
|  |  |
| --- | --- |
| **Bas**  **Ket** | [**https://medium.com/@FurryMogwai/building-a-basket-micro-service-using-asp-net-core-and-akka-net-ea2a32ca59d5**](https://medium.com/@FurryMogwai/building-a-basket-micro-service-using-asp-net-core-and-akka-net-ea2a32ca59d5)  [**https://gitlab.com/pnieuwenhuis/newhouse-basket-service**](https://gitlab.com/pnieuwenhuis/newhouse-basket-service)          m |
| **Health**  **Check**  **Controller**  **.cs** | using System;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Mvc;  namespace NewhouseIT.BasketService.Baskets.Routes  {  **[Route("/api/ping")]**  public class HealthCheckController  {  [HttpGet]  public **IActionResult Get()** => new OkObjectResult("Service is healthy");  }  } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

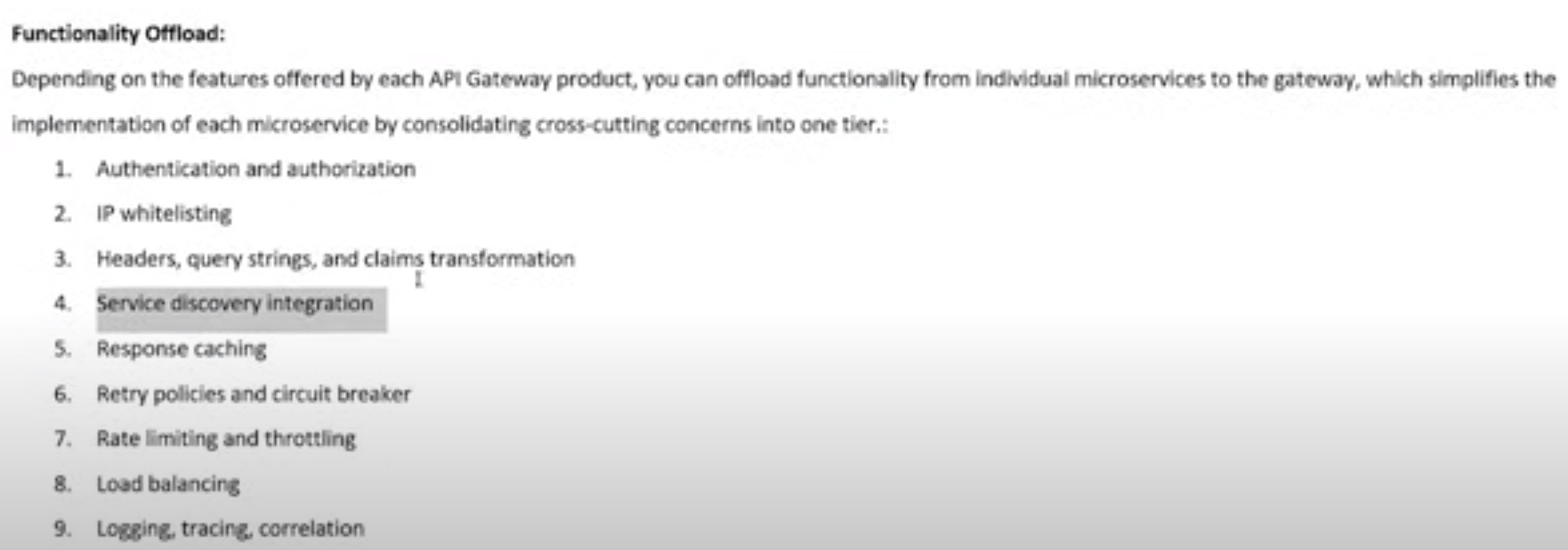
|  |  |
| --- | --- |
| API Communication | API Management, azure Load balancer, traffic manager, |
| Microservices  Transaction | SAGA , Azure Service Bus Topic, 2 way, 3 way communication, |
| Microservices  Authentication | Azure AD, Role |
| Microservices  Logging, Monitoring |  |
| Common Storage |  |
| Gateway |  |
| Search |  |
| Microservices  Deployment |  |
| Technology | Why .NET Core  .NET Core (very small imaging in memory, path is very small. compare framework, light weight) |
| Microservice  Disadvantage | Distributed service adds more network communication  Deployment is complex  Dashboard (Monitoring) – Which is failed which is up.  Messaging (Async) – Difficult testing.  Every microservice has log – we want all log in one place  Need service discovery solution – Where is service located.(if crased node in cluster then new location node ?  Require Failure/ Recovery code  Inter service communication using protocol like HTTP or AMPQ  Debugging and testing very difficult |

Microservices Common Architecture

|  |  |
| --- | --- |
|  | 12 factor Follow, Repository pattern, Dapper, DAO, |
|  | Cloud design pattern |
|  | DDD, CQRS, Retry, Circuit breaker pattern, .NET Core (singleton, Scope, T |
| .NET Core | Small |

|  |  |
| --- | --- |
| resilency | patterns like Bulkhead and Circuits Breaker which can help you achieve better resilency. |
| **Bulkhead** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |





Microservice Best Practices:

<https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>