[Date]

Architecture Research

Individual Project: FestivalConnect

Name: Lucas Jacobs
Class: S-A-RB06
PCN: 490692
Student number: 4607368
Technical teachers: Felipe Ebert, Bartosz Paszkowski
Semester coach: Gerard Elbers

Table of Contents

Glossary

Term	Definition
Kubernetes	Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more—making it easier to manage applications.
Batch Job	A scheduled program is assigned to run on a computer without further user interaction.
Platform as a service (PaaS)	Is a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications.
Cold starts	The latency that occurs when a function is triggered for the first time or after a period of inactivity.
Infrastructure	All the operational and computational assets,—such as servers, storage arrays, and operating systems—required to successfully design, build, manage, and deliver an application and its services to end users are part of the application infrastructure.
horizontal scaling	Horizontal scaling means that you scale by adding more machines into your pool of resources to an existing machine. This allows you to distribute your workload across a larger pool of resources.

Introduction

There are a lot of different architecture styles to choose for an enterprise application. That is why FestivalConnect needs to look at these options and try to fit the best one that meets the requirements specified in the SRS (Jacobs, 2024, Software Requirements Specification).

Context

The architecture purpose needs to be in alignment with the (Jacobs, 2024, Software Requirements Specification). This outlines the most important qualities that FestivalConnect needs to keep in mind with what the users can expect from the application (behavior and function). With this, we need to look into multiple architectures and which one fits best to FestivalConnect. Furthermore, the stakeholders do not have a specific requirement, it needs to be at least well-argued.

Chosen architectures

To research several architectures, we will go over the following styles that can benefit FestivalConnect.

- **N-Tier Architecture**: FestivalConnect can benefit from an N-tier architecture, the application logic is divided into separate layers such as the presentation, business logic, and data access. This architecture can promote scalability because each tier can handle varying loads during for example peak festival periods. (N-tier architecture style, n.d.)
- **Web-Queue-Worker Architecture**: FestivalConnect can use this by handling incoming user requests, a message queue to manage asynchronous tasks, and worker processes to perform background tasks such as event notifications or data processing. (Web-Queue-Worker architecture style, n.d.)
- **Microservice Architecture**: This project can benefit from this due to its modular and scalable abilities. This architecture allows for great scaling for making services for each business case, such as event scheduling and community interactions. (Microservice architecture style, n.d.)
- **Event-Driven Architecture**: Looking at the dynamics of a festival, an event-driven architecture can be considered by FestivalConnnect, by facilitating real-time communication and responsiveness to various community features or event updates. Giving a smooth experience that is loose can benefit the user experience. (Event-driven architecture style, n.d.)
- **Service-Oriented Architecture (SOA):** The use of different services for each business capability, can promote flexibility to add different functional modules such as event creation and user management.

Architecture Options

This section will go over the architectures and will dive further into the benefits, disadvantages, and use cases that they can be used for.

N-Tier architecture style

An N-Tier architecture divides the functions of presentation, processing, and data in separate layers, both logically and physically. Each layer operates on itself, often having different locations, which can be good for scalability, finding errors, and performance optimization. This method is made that it is linear, meaning you need to pass through a layer in order to go to the next. To illustrate, in a N-tier, the most common way is the following.

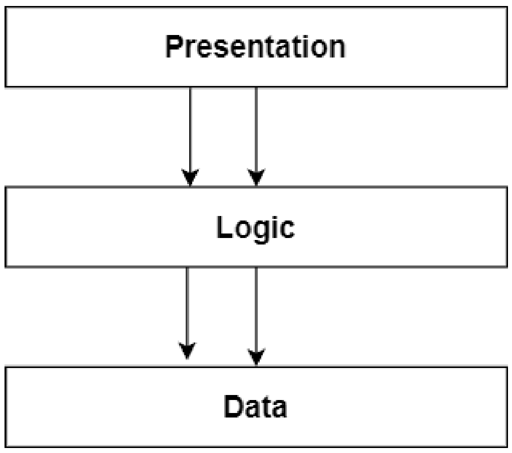

Figure 1:N-Tier

Presentation: Showing the user the information in an easily understood way.
Logic: Handling the business processes, including handling errors, calculations, and logical decisions.
Data: Handles the data stored, in a database.
(baeldung, n.d.)

Example

See the following example of how a system setup is, using a N-tier.

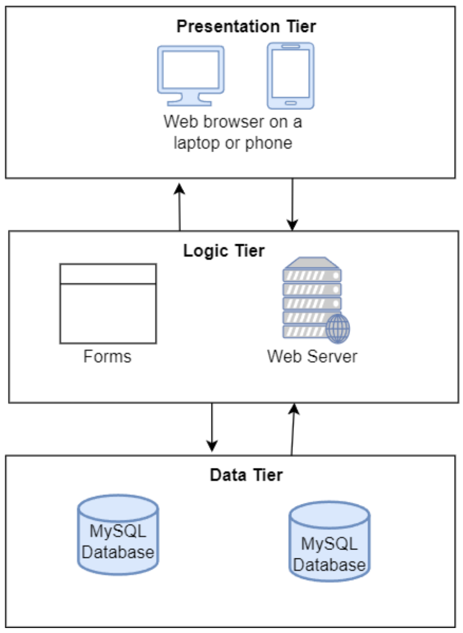

Figure 2: Example N-tier (baeldung, n.d.)

Benefits and Downsides

Benefits	**Challenges**
Scalability: Easy implementation of individual components.	**Latency**: Pysical separtation can add increased latency.
Maintenance: Maintain each components independently.	**Middle tier CRUD**: Middle tier can lead to only doing CRUD, added latency.
Resulabiliy: Logical separation allows for reuse in different projects.	**Monolithic design**: Prevents independent deployment of features.
Portability: easily used between clo9ud and on-premises environments.	**Network security**: Difficulty in managing network security in large systems.
Easy to understand.	
Open for environments such as Windows/Linux.	

(baeldung, n.d.), (N-tier architecture style, n.d.)

This architecture is mainly implemented or to be considered when you want to build a simple web application, move an existing software application from your local servers to the Microsoft cloud platform Azure, or for a software application using a single set of tools and processes, regardless if it is going to be deployed on your local servers or in the cloud.

FestivalConnect fit

The N-Tier architecture offers benefits such as scalability, ease of maintenance, etc. It also comes with challenges such as hard-to-manage network security for bigger applications, With the N-Tier architecture, it comes with great benefits but FestivalConnect needs to be a dynamic platform for managing festival-related activities, which needs a more flexible and agile architecture to meet diverse needs.

Web-Queue-Worker architecture style

This architecture has three main components:

- **Web front end**: Handles the client's request.
- **Worker**: does the intensive tasks, long-running workflows, or batch jobs.
- **Message queue**: The way the front end communicates with the worker.

(Web-Queue-Worker architecture style, n.d.)

What is a worker?

Also known as a concurrent worker, is a component responsible for processing jobs that are pushed into a queue. It operates as a literal queue, meaning it handles one job at a time from the queue. The worker picks up the next job in line, processes it, and upon completion, moves on to the next job. See the following figure for how a worker is doing their job.

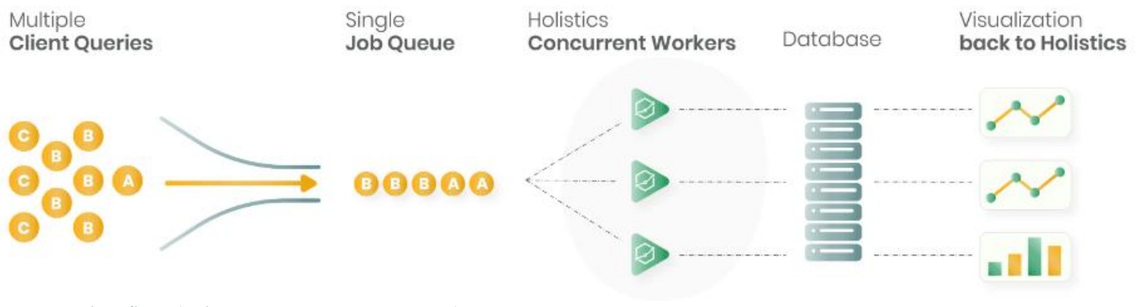

Figure 3: Worker flow (Job queue is Message queue)

Workers play a crucial role in the efficient handling of asynchronous tasks. As users interact with an application, and they want to view reports or schedules, jobs are created and pushed into the job queue. These jobs represent tasks that require processing. (Job Queue System and Workers, n.d.)

Architecture

This is how the architecture will look like.

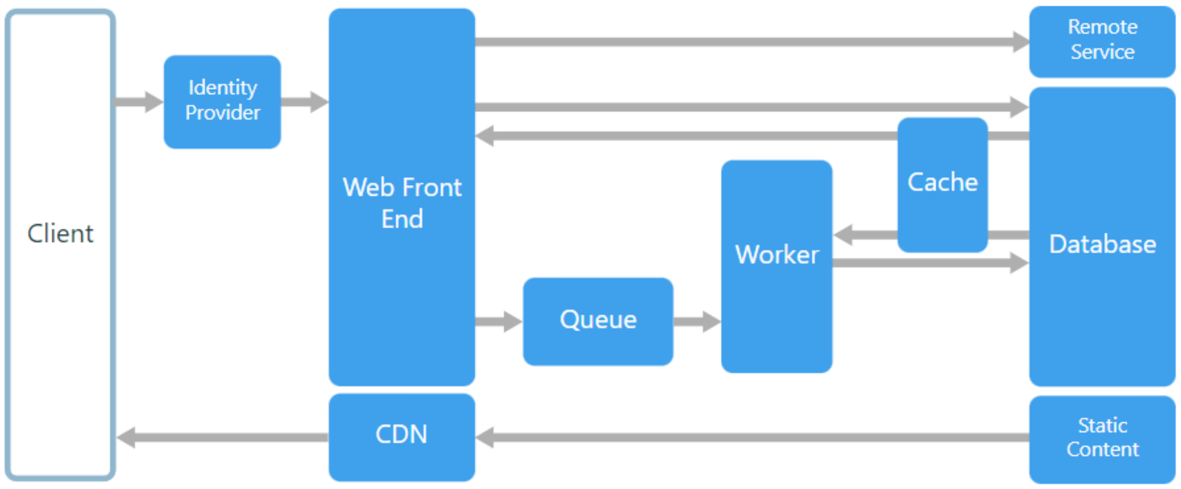

Figure 4Web-Queue-Worker architecture

It is used for a PaaS solution. In this style, the application will have a web front end that handles HTTP requests and a back-end worker that needs to perform heavy or long-running tasks. The communication is that the front end will communicate with the worker through an asynchronous message queue. It can be suitable for simple domains with some resource-intensive tasks. (Introduction to Microsoft Azure Architecture Styles, 2018)

Benefits and Challenges

Benefits	Challenges
Relatively Simple Architecture: Straightforward and give room for easier maintenance.	**Potential Monolithic Components**: without careful design, the front end and worker can become overly large, making it hard to maintain.
Easy Deployment and Management: The deployment process is easier, and managing is more straightforward.	**Consistency Issues**: Malfunctions in the web front end after successfully persisting to the database but before sending out messages to the queue can lead to consistency issues. Techniques like the transactional outbox pattern can mitigate this problem but require additional complexity.
Clear Separation of Concerns: The front end and worker are separated, same goes for the database.	**Development and Test Challenges**: Testing service dependencies and refactoring across service boundaries can be difficult, and existing tools may not be optimized for this architecture
Decoupled Front End and Worker: Asynchronous messaging, allowing the front end and worker to work independently.	**Hidden Dependencies**: There's a risk of hidden dependencies if the front end and worker share data schemas or code modules, which can complicate maintenance and updates.

(Web-Queue-Worker architecture style, n.d.), (Introduction to Microsoft Azure Architecture Styles, 2018), (Santos, 2018)

FestivalConnect fit

The Web-Queue-Worker architecture style can offer as a good framework for FestivalConnect, it has efficient handling of asynchronous tasks and user interactions. With taking care of separating the components, and looking at upcoming challenges, FestivalConnect can scale independently and maintain a responsive user experience. On the other side, FestivalConnect doesn’t have intensive tasks that require a worker component to do in a traditional sense. It can maybe focus on more community management and event notifications. But this is not a typical use case for a worker. Therefore, while it can fit in FestivalConnect, there can be a better suit to the specific requirements.

Microservices architecture style

A microservice style is a way of building an application with the use of multiple services. Each service consists of a business case, such as user management and community management. Microservices started as a solution to the limitations of monolithic architectures, during the phase from monolithic to service-oriented software development. While development speeds continued to increase and the applications were difficult to modify and update, with a required deployment of the entire system for minor changes. Therefore microservices came as a solution so that you could build independent services that could be updated and scaled individually. (Microservices, n.d.)
See the example for a comparison of monolithic and microservice.

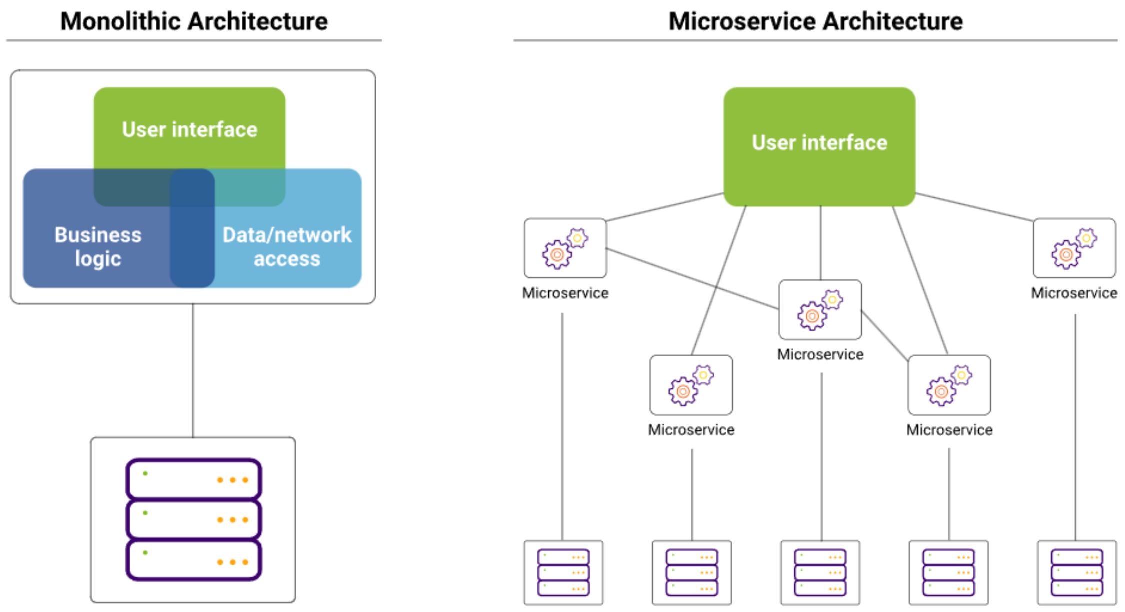

Figure 5: Difference Monolithic versus Microservice

How does it work and why is it important?

Microservices are serving for one business case, meaning that a small team of developers can work on it. Each service uses its own codebase, meaning that you can be flexible about which language you want to use since the only thing that needs to be in common is the HTTP/JSON communication (REST), which is supported by most languages. Each service holds there own data, which is different from a monolithic approach where they use separate data layers to handle data persistence.

To further clarify the structure, see the following figure.

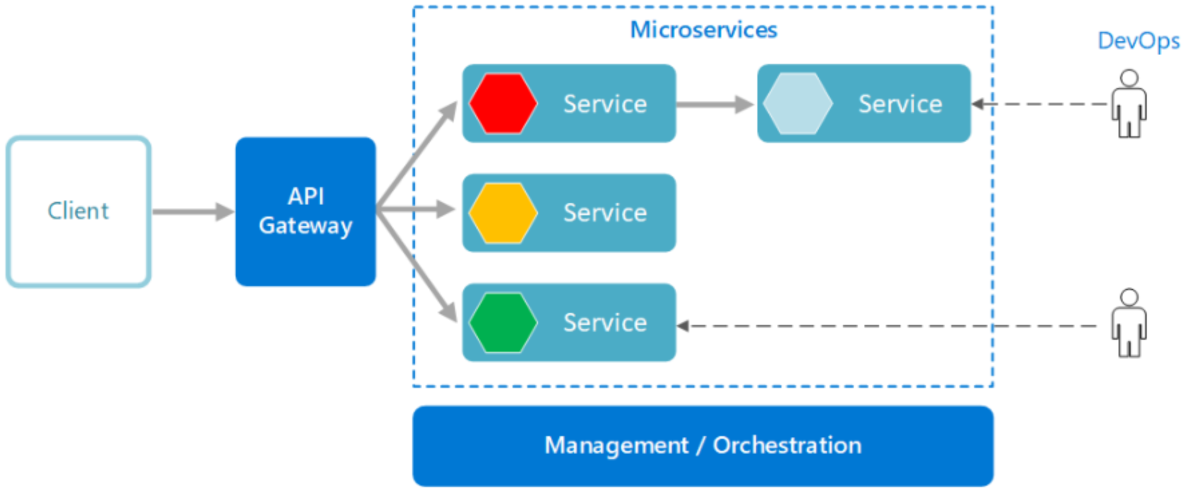

Figure 6: Microservice structure

Management/Orchestration: A component that coordinates tasks like deploying software, monitoring performance, and fixing issues. It acts as a conductor for system components, to make sure they will work efficiently together, an example to use is Kubernetes.

API Gateway: This is the entry point of clients when they make requests to get for example data. Instead of having a client directly calling the service, there is an API gateway, that will send the call to the correct service.

(Microservice architecture style, n.d.)

Benefits and Challenges

Benefits	**Challenges**
Agility: Enables independent deployment and updates	Complexity: due to moving parts
Small, focused teams: Promotes productivity	Development and testing: Requires a different approach and tools
Small code base: Minimizes the dependencies	Lack of governance: You need to set strict rules for general things.
A mix of technologies: Allows for innovation	Latency problems: When more services need to communicate with each other, can lead to performance loss.
Fault isolation: Not your whole application will crash, only a part will be unavailable	Data integrity: Microservice has its own data persistence, and consistency can become a challenge.
Scalability: Scales subsystems independently	Management: Need mature DevOps culture
Data isolation: Easier schema updates	Versioning: Making sure compatibility during updates.

(Microservice architecture style, n.d.)

With the use of microservices architecture, it can offer FestivalConnect the agility to adapt quickly to user needs and market change. This can be helpful for the festival-industry market since certain genres can become increasingly popular, therefore rapid updates are required. By carefully planning to tackle the challenges like complexity and performance, to get all the benefits like better productivity among smaller teams, easier maintenance, and minimized dependencies.

Event-driven architecture style

This architecture is a way of a design pattern, which is where different parts can communicate with each other by sending messages about what is going on. This message, called events, could be anything from registering an account to joining a community. Instead of directly talking to each other, the parts use a broker, to send and receive these messages. This can make things flexible since the sender only needs to know about the events it cares about, rather than everything that is going on. See the following for a more clear view. (Seetharamugn, 2023)

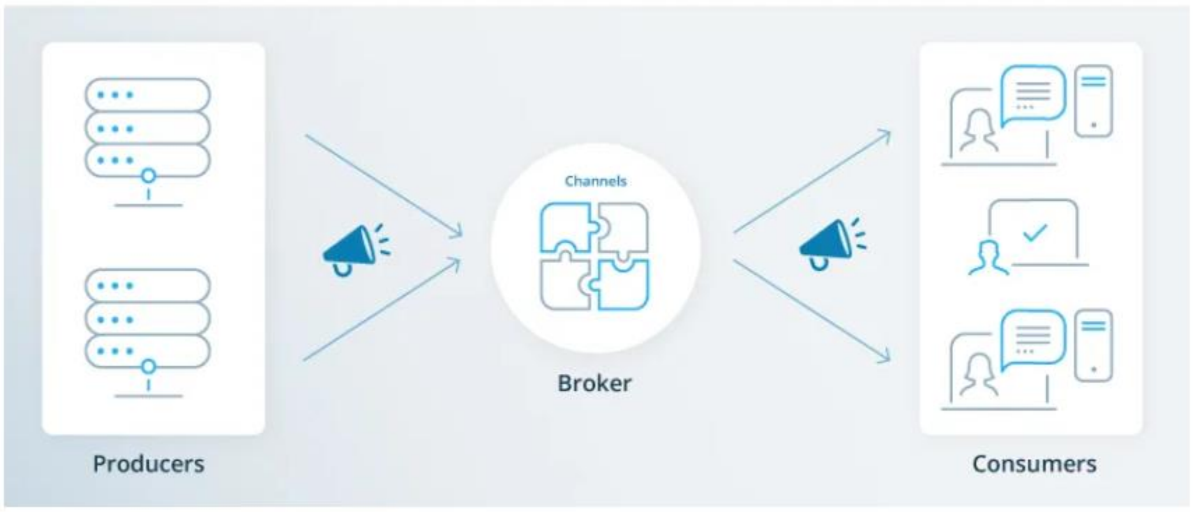

Figure 7: Event-Driven Architecture

There are two different models to use for this architecture.

- **Pub/Sub (publish/subscribe):** a messaging model where a central infrastructure manages subscriptions. When an event is published, it is then sent to all the subscribers. Once received, they can not be replayed, and new subscribers miss the past event. This makes sure that it is a real-time delivery of events.
- **Event streaming:** This is done by doing it by writing events to a log in a strictly ordered manner. You do not subscribe; they read from any part of the stream, going further with their position. This allows clients to join at any time, replay events, and manage their progress within the stream independently.

(Event-driven architecture style, n.d.)

Event

In an Event-driven architecture, it is everything that happens within and to a business case, so texting and joining a community, etc.

Broker

Event brokers are middleware that will make the user the communication between event producers and consumers in event-driven architecture. A broker can be a hardware appliance, software, or software as a service, see the following example.

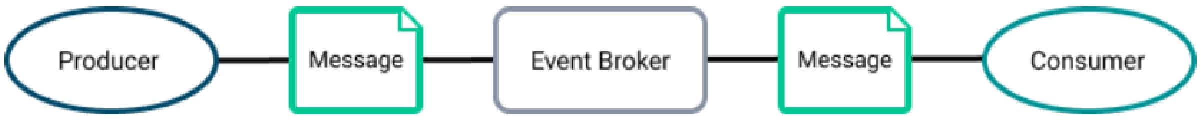

Figure 8: Broker example

(What are Event Brokers?, n.d.)

Benefits and Challenges

The architecture can be used when sub-systems must process the same events, real-time processing with minimum delay, complex event processing, or high volume and high velocity in data.
This may fit FestivalConnect due to the need for real-time interaction, scalability during peak festival periods, complex events processing, and maintaining loose coupling between components.
To further inspect, here are the benefits and challenges that this architecture has.

Benefits	**Challenges**
Decoupled Producers and Consumers	To make guaranteed delivery possible.
Everything is processed as fast as possible, not waiting on each other.	Processing events in order, or exactly once.
No point-to-point integrations.	Consistency across the system can be complex.
Highly scalable and distributed.	Debugging and monitoring are harder since events can trigger step-by-step effects across multiple components.
Subsystems have independent views of the event stream.	
Secure, uses event authentication, encryption, and access control.	

(Doshi, 2024), (Seetharamugn, 2023), (Event-driven architecture style, n.d.)

Event-driven architecture can suit FestivalConnect's real-time needs regarding the functions and give the users a loose experience that can feel like a smooth experience, that festival goers appreciate. Also, it can come with challenges such as event delivery and consistency. Maybe other architecture can provide a better suit, otherwise, this can be a strong contender, comparing the benefits and challenges.

Service-Oriented Architecture (SOA)

With this technology, several services will communicate with each other. It is focused on creating reusable services that communicate across platforms to build new applications. A service is a self-contained unit designed for a specific task (business case), accessed via a simple interface. It can make complex systems into loose and reusable services accessed by various applications.
In SOA, there are four components.

- **Service**: This is the main element of SOA, it has a specific functionality and is characterized by its implementation, contract, and interface. This can be private or public, containing their code to execute.
- **Service Provider**: This is used to create or offer the service, the service provider can be an organization or third party. It makes sure that users get availability to services.
- **Service Consumer**: It makes requests to the service provider.
- **Service Registry**: Maintains a directory of available services.

(Barney, n.d.)

See the following to get a better understanding of SOA.

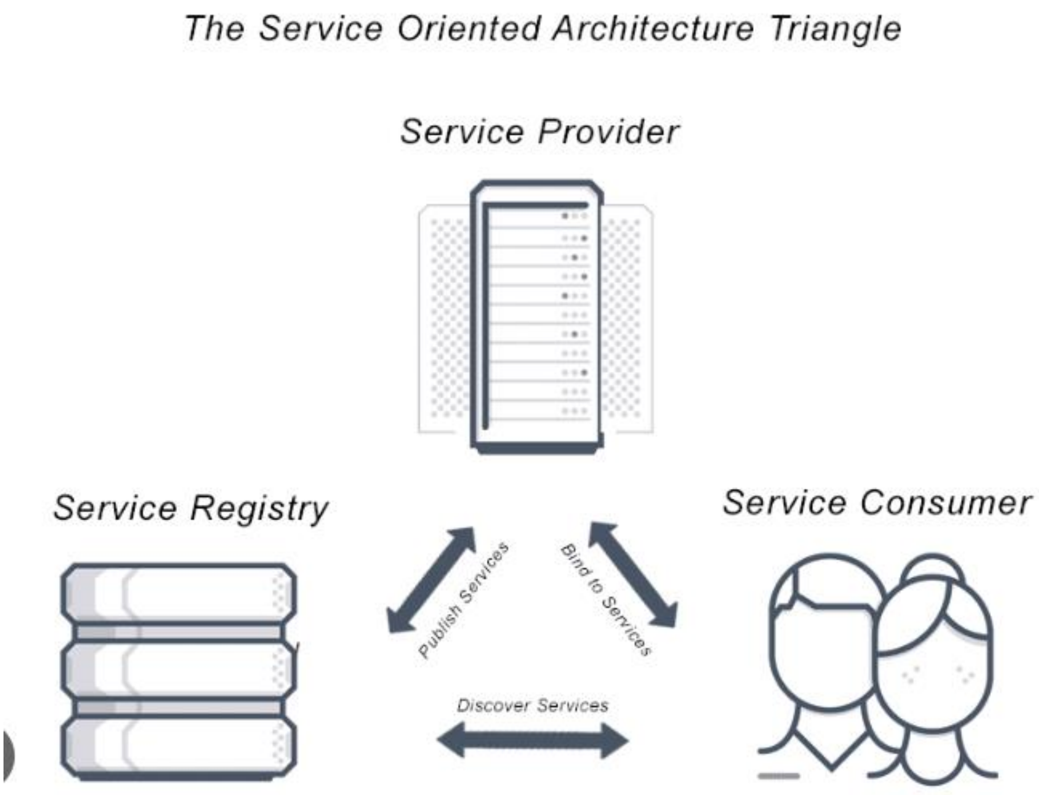

Figure 9 SOA Triangle (Service-Oriented Architecture Definition, n.d.)

SOA vs. Microservices

These two can be confused with each other. The main difference between them is their scope and implementation strategy. SOA is an enterprise-wide architectural approach that focuses on creating reusable services that communicate across various systems and platforms. Microservices, on the other hand, are an implementation strategy within application development by teams. It is more thinking of breaking down an application into smaller, independent services that communicate with each other.

A SOA is in summary an enterprise-wide architectural approach, and microservices are a specific implementation strategy within application development teams.
(What is service-oriented architecture (SOA)?, 2020)

Benefits and Challenges

Benefits	**Challenges**
Reusable: Services can be reused, allow for small independent functional components, and reduce time and cost	**Management**: Requires correct management, on quality, security, and performance.
Easy maintenance: Services can be updated or modified without affecting the consumers.	**Reliability and scalability**: Tradeooff regarding network latency and dependencies that can down-grade performance
Integration: Easy to integrate on platforms.	

(What are the advantages and disadvantages of service-oriented architecture?, n.d.)

SOA can fit FestivalConnect, with loose and easy-to-maintain functions. However, Microservice can be more advanced due to the agility, fault, isolation, and small focus teams, which can be beneficial for rapid updates and adaptability in the festival industry. Therefore, making a microservices architecture likely a better choice for FestivalConnect's dynamic requirements and growth potential.

Serverless architecture

You still build an application based on services but without the need to manage infrastructure. All the server management is done by cloud computing, for example, AWS. This way the developers can focus on the core product, instead of thinking about managing and operating servers or runtimes. This reduces time and energy and can lead to faster development. (Building Applications with Serverless Architectures, n.d.)

The most commonly used is a Function as a Service (FaaS), where developers create small pieces of code to do specific jobs. These pieces are triggered by events, like when someone visits a website or sends an email. When events happen, the cloud provider runs the code. Developers do not need to think about the servers, this is all taken care of. See the following.

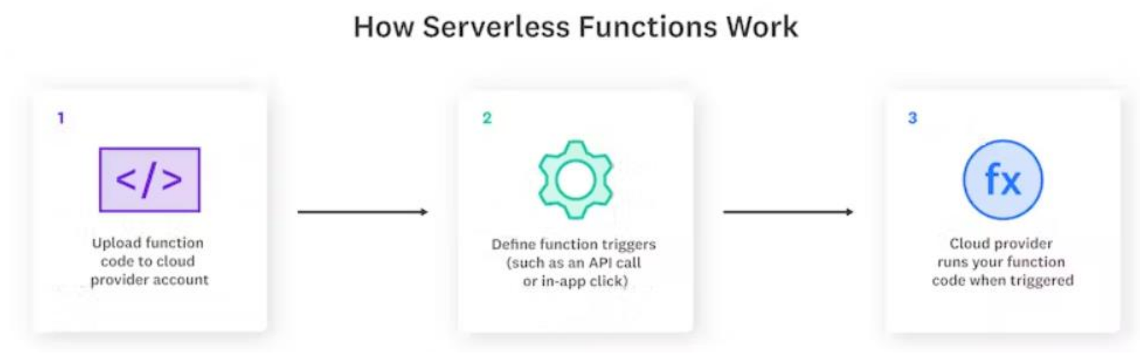
How Serverless Functions Work
1
Upload function code to cloud provider account
2
Define function triggers (such as an API call or in-app click)
3
Cloud provider runs your function code when triggered

Figure 10: FaaS works with serverless

(Serverless Architecture Overview, n.d.)

Serverless vs. Container Architecture

Between serverless and container-based architecture, are some differences that can be crucial in the decision of FestivalConnect.

- **Management of infrastructure:**
  - Serverless: The Cloud provider manages the infrastructure entirely.
  - Containers: Developers are responsible for managing containers.
- **Scalability**:
  - Serverless: Scales automatically based on demand, no developer interruption.
  - Containers: Manual configuration or use of tools.
- **Granularity**:
  - Serverless: Functions are small, units of code to perform a specific task.
  - Containers: Applications are packaged into larger, self-contained units that can include multiple services or components.
- **Cold Start Performance**:
  - Serverless: May experience when a function is a long time not called.
  - Containers: Faster startup times, containers are always running.
- **User Cases**:
  - Serverless: Event-driven, short-lived tasks with unpredictable traffic patterns.
  - Containers: Long-running applications, microservices architectures, and with consistent traffic.

(Serverless Architecture Overview, n.d.)

Benefits and Challenges

Benefits	Challenges
Cost-Effective: You only pay for what you use, also the hardware can save money instead of maintaining servers.	**Loss of Control**: You rely entirely on the cloud provider, as well as machines when it has faults or any issues.
Scalability: Automatically handle scaling by creating or removing instances based on incoming traffic.	**Security**: The code of customers may run on similar servers, without proper configuration, this can lead to security vulnerabilities and leaking data.
Increased Productivity: Only working on writing and deploying code.	**Performance Impact**: Cold starts, when the server needs to "wake up", causing a delay.
	Testing Complexity: Integration testing, can be challenging, complicating the test process
	Vendor Lock-In: Hard to switch providers, depending too much on one

(Serverless Architecture Overview, n.d.)

FestivalConnect Fit

Serverless architecture can fit FestivalConnect due to its event-driven nature and scalability requirements. With cloud functions for features like user registration and community management, can serverless benefit FestivalConnect with scalability and reduced operational management. However, the consideration of potential vendor lock-in and potential complexity in managing integrations can make another architecture more suitable. When looking at microservices, this can offer more control over infrastructure and easier flexibility between cloud providers, minimizing the vendor lock-in problem. So, serverless offers great features, but in the long-term solution, microservices can be a better solution for FestivalConnect.

Architecture Choice

For FestivalConnect, Microservices architecture is the one that stands out as the best choice due to the benefits of agility, scalability, and fault isolation. While other architectural styles like N-Tier, Web-Queue-Worker, Event-Driven, and Service-Oriented architectures offer certain advantages, they come with trade-offs.

- N-Tier offers scalability but lacks agility and fault isolation while also not being able to deploy small updates separately from microservices.
- Web-Que-Worker and Event-Driven architectures have great asynchronous tasks and real-time processing but may not offer the flexibility and independence of microservices.
- Service Oriented Architecture (SOA) also provides reusable services but lacks the agility and granularity of microservices.
- Serverless offers cost-efficiency and scalability but risks too much on vendor lock-in and has testing complexity.

Therefore, while each option has great advantages, microservices have a great balance for FestivalConnect's requirements and growth future.

Back End Framework Fit

There are a lot of benefits when it comes to using microservices with .NET Core API. But not all products are perfect, therefore I listed some of the benefits and downsides of the system.

Benefits	**Downsides**
Cross-Flatform: .NET Core has become cross-platform, meaning that you are flexible in where you want to deploy your environment.	**Platform Dependency**: Depending on Windows, can limit the deployment options.
.NET Ecosystem: A lot of libraries, frameworks, and tools are available, meaning the implementation of microservices becomes easier.	**Vendor Lock-In**: Using Microsoft technologies can result in vendor lock-in, complicating migration.
Performance: C# offers strong performance, benefitting the architecture.	**Licensing Costs**: Choices of development and deployment, can come with costs to certain Microsoft products or tools.
Scalability: Support of horizontal scaling, using other technologies such as Docker and Kubernetes.	
Security: Libraries for authentication and authorization with other security features, help secure the microservices.	

(How to Write Microservices in C#, n.d.)

With the right approach, .NET Core API can be great fit when it comes to implementing microservices. FestivalConnect want agility and flexibility in order to have a well maintained application, which microservices provide by breaking the application down into smaller, independent services With the .NET Core API features, it can further improved the development efficiencies such as high-performance, scalable and secure built-in mechanics. Making microservices a great choice for FestivalConnect's backend framework.

References

baeldung. (n.d.). *N-Tier Architecture*. Retrieved from baeldung: https://www.baeldung.com/cs/n-tier-architecture

Barney, N. (n.d.). *service-oriented architecture (SOA)*. Retrieved from techtarget: https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA

Building Applications with Serverless Architectures. (n.d.). Retrieved from aws.amazon: https://aws.amazon.com/lambda/serverless-architectures-learn-more/#:~:text=What%20is%20a%20serverless%20architecture,management%20is%20done%20by%20AWS.

Doshi, R. (2024, 02 08). *Event-Driven Architecture : Benefits, Challenges, and Examples*. Retrieved from medium: https://medium.com/@rohitdoshi9/event-driven-architecture-benefits-challenges-and-examples-c957c269420a#:~:text=In%20summary%2C%20event%2Ddriven%20architecture,event%20ordering%2C%20and%20eventual%20consistency.

Event-driven architecture style. (n.d.). Retrieved from learn.microsoft: https://learn.microsoft.com/en-gb/azure/architecture/guide/architecture-styles/event-driven

Introduction to Microsoft Azure Architecture Styles. (2018, 12 18). Retrieved from anarsolutions: https://anarsolutions.com/architecture-styles/

Job Queue System and Workers. (n.d.). Retrieved from docs.holistics: https://docs.holistics.io/docs/job-queues-and-workers#:~:text=%E2%80%8B,proceeds%20to%20the%20next%20one.

Microservice architecture style. (n.d.). Retrieved from learn.microsoft: https://learn.microsoft.com/en-gb/azure/architecture/guide/architecture-styles/microservices

Microservices. (n.d.). Retrieved from synopsys: https://www.synopsys.com/glossary/what-are-microservices.html

N-tier architecture style. (n.d.). Retrieved from learn.microsoft: https://learn.microsoft.com/en-gb/azure/architecture/guide/architecture-styles/n-tier

Santos, M. (2018, 09 18). *Web-Queue-worker*. Retrieved from linkedin: https://www.linkedin.com/pulse/web-queue-worker-marco-santos/

Seetharamugn. (2023, 08 29). *The Complete Guide to Event-Driven Architecture*. Retrieved from medium: https://medium.com/@seetharamugn/the-complete-guide-to-event-driven-architecture-b25226594227

Serverless Architecture Overview. (n.d.). Retrieved from datadoghq: https://www.datadoghq.com/knowledge-center/serverless-architecture/

Service-Oriented Architecture Definition. (n.d.). Retrieved from avinetworks: https://avinetworks.com/glossary/service-oriented-architecture/

Web-Queue-Worker architecture style. (n.d.). Retrieved from learn.microsoft: https://learn.microsoft.com/en-gb/azure/architecture/guide/architecture-styles/web-queue-worker

What are Event Brokers? (n.d.). Retrieved from solace: https://docs.solace.com/Get-Started/what-are-event-brokers.htm#:~:text=An%20event%20broker%20is%20middleware,to%20provide%20the%20messaging%20infrastructure.

What are the advantages and disadvantages of service-oriented architecture? (n.d.). Retrieved from linkedin: https://www.linkedin.com/advice/3/what-advantages-disadvantages-service-oriented-architecture

What is service-oriented architecture (SOA)? (2020, 07 27). Retrieved from redhat: https://www.redhat.com/en/topics/cloud-native-apps/what-is-service-oriented-architecture#:~:text=In%20a%20service%2Doriented%20architectural,reducing%20the%20dependencies%20between%20them.

Jacobs, L. (2024). Software Requirements Specification (Unpublished manuscript), FontysICT.