Liferay
Boot Camp
Summer Edition

Liferay as Digital Experience Platform in the context of Microservices

Antonio Musarra

Senior Software Architect

Presentation topics

Presentation topics

1. Overview on Microservices
2. Liferay as Microservices Platform
3. Digital Experience Platform
4. Headless CMS open to Microservices

Overview on Microservices

Overview on Microservices

1. The architectural approach
2. An already known approach
3. From SOA architecture to Microservices
4. The advantages of a Microservices architecture
5. Disadvantages of the Microservices architecture

The architectural approach

The architectural approach

1. Microservices are an architectural approach to building applications

The architectural approach

1. Microservices are an architectural approach to building applications
2. What distinguishes architecture based on microservices from traditional monolithic approaches is the subdivision of the app into its basic functions

The architectural approach

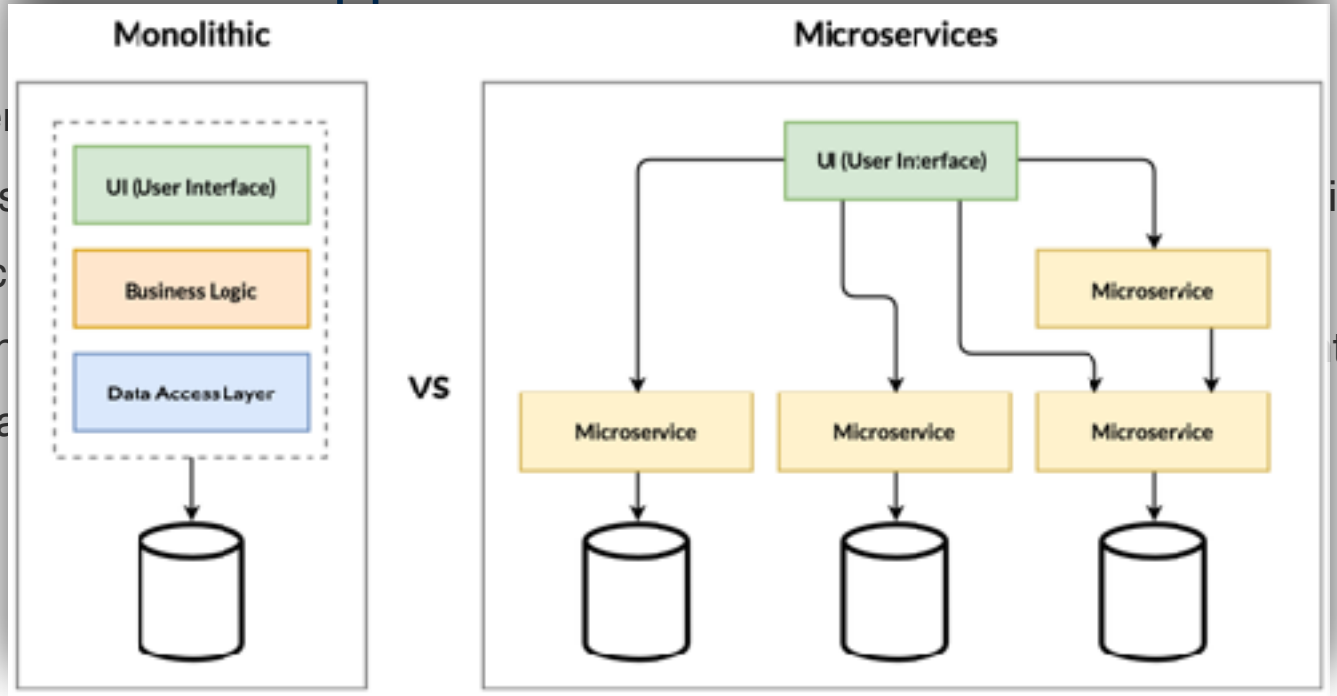
1. Microservices are an architectural approach to building applications
2. What distinguishes architecture based on microservices from traditional monolithic approaches is the subdivision of the app into its basic functions
3. Each function, called a service, can be compiled and implemented independently

The architectural approach

1. Microservices are an architectural approach to building applications
2. What distinguishes architecture based on microservices from traditional monolithic approaches is the subdivision of the app into its basic functions
3. Each function, called a service, can be compiled and implemented independently
4. Individual services may or may not work without compromising others

The architectural approach

1. Microse
2. What dis ithic
   approac
3. Each fur tly
4. Individua



The architectural approach

The architectural approach

1. Therefore, a microservice is a basic function of an application, which runs independently of the other services

The architectural approach

1. Therefore, a microservice is a basic function of an application, which runs independently of the other services

2. However, the architecture based on microservices not only involves the low coupling between the basic functions of an app, but proposes a restructuring of the development teams and of the communication framework between the services

The architectural approach

1. Therefore, a microservice is a basic function of an application, which runs independently of the other services

2. However, the architecture based on microservices not only involves the low coupling between the basic functions of an app, but proposes a restructuring of the development teams and of the communication framework between the services

3. This approach offers the possibility to manage unavoidable critical issues, supports dynamic scalability and facilitates the integration of new features

The architectural approach

1. Therefore, a microservice is a basic function of an application, which runs independently of the other services

2. However, the architecture based on microservices not only involves the low coupling between the basic functions of an app, but proposes a restructuring of the development teams and of the communication framework between the services

3. This approach offers the possibility to manage unavoidable critical issues, supports dynamic scalability and facilitates the integration of new features

4. To deploy microservices and take advantage of this approach, you need to adapt the basic elements of a Service-Oriented Architecture (SOA)

An already known approach

An already known approach

1. Dividing an app into its basic functions and avoiding the pitfalls of the monolithic approach may seem familiar concepts, because the architectural style of the microservices is similar to that of the SOA (Service-Oriented Architecture) architecture, a well-established software design style

An already known approach

1. Dividing an app into its basic functions and avoiding the pitfalls of the monolithic approach may seem familiar concepts, because the architectural style of the microservices is similar to that of the SOA (Service-Oriented Architecture) architecture, a well-established software design style

2. In the early days of application development, even a minimal change to an existing app required a complete update and a quality assurance cycle (QA) of its own, which risked slowing down the work of various secondary teams. This approach is often referred to as "monolithic" because the whole app's source code was compiled into a single deployment unit (for example, with the extension .war or .ear). If updates to part of the app caused errors, it was necessary to disconnect everything, step back and correct. This approach is still applicable to small applications, but growing companies cannot afford downtime

An already known approach

1. Dividing an app into its basic functions and avoiding the pitfalls of the monolithic approach may seem familiar concepts, because the architectural style of the microservices is similar to that of the SOA (Service-Oriented Architecture) architecture, a well-established software design style

2. In the early days of application development, even a minimal change to an existing app required a complete update and a quality assurance cycle (QA) of its own, which risked slowing down the work of various secondary teams. This approach is often referred to as "monolithic" because the whole app's source code was compiled into a single deployment unit (for example, with the extension .war or .ear). If updates to part of the app caused errors, it was necessary to disconnect everything, step back and correct. This approach is still applicable to small applications, but growing companies cannot afford downtime

3. And here comes the SOA (Service-Oriented Architecture) architecture, in which the apps are structured in reusable services that communicate with each other via an Enterprise Service Bus (ESB)

An already known approach

An already known approach

1. In the SOA architecture, the individual services focus on a specific business process and follow a communication protocol, including SOAP, ActiveMQ or Apache Thrift, to be shared through the ESB

An already known approach

1. In the SOA architecture, the individual services focus on a specific business process and follow a communication protocol, including SOAP, ActiveMQ or Apache Thrift, to be shared through the ESB

2. Overall, this suite of services integrated through an ESB constitutes an application

An already known approach

1. In the SOA architecture, the individual services focus on a specific business process and follow a communication protocol, including SOAP, ActiveMQ or Apache Thrift, to be shared through the ESB

2. Overall, this suite of services integrated through an ESB constitutes an application

3. In addition, this method allows you to compile, test and modify multiple services simultaneously, freeing IT teams from monolithic development cycles. However, since ESB represents a single point of failure for the whole system, it could pose an obstacle for the whole organization

From SOA architecture to Microservices

From SOA architecture to Microservices

1. Microservices can communicate with each other, generally in stateless mode, allowing you to build apps with greater fault tolerance and less dependent on a single ESB

From SOA architecture to Microservices

1. Microservices can communicate with each other, generally in stateless mode, allowing you to build apps with greater fault tolerance and less dependent on a single ESB

2. They communicate through language-independent application programming interfaces (APIs) and this allows development teams to choose their own tools.

From SOA architecture to Microservices

1. Microservices can communicate with each other, generally in stateless mode, allowing you to build apps with greater fault tolerance and less dependent on a single ESB
2. They communicate through language-independent application programming interfaces (APIs) and this allows development teams to choose their own tools.
3. Considering the evolution of SOA, microservices are not an absolute novelty, but lately they have become more attractive thanks to the advances in containerization technologies.

From SOA architecture to Microservices

1. Microservices can communicate with each other, generally in stateless mode, allowing you to build apps with greater fault tolerance and less dependent on a single ESB
2. They communicate through language-independent application programming interfaces (APIs) and this allows development teams to choose their own tools.
3. Considering the evolution of SOA, microservices are not an absolute novelty, but lately they have become more attractive thanks to the advances in containerization technologies.
4. Today Linux containers allow you to run multiple parts of an app independently, on the same hardware, with far greater control over individual components and life cycles.

From SOA architecture to Microservices

1. Microservices can communicate with each other, generally in stateless mode, allowing you to build apps with greater fault tolerance and less dependent on a single ESB
2. They communicate through language-independent application programming interfaces (APIs) and this allows development teams to choose their own tools.
3. Considering the evolution of SOA, microservices are not an absolute novelty, but lately they have become more attractive thanks to the advances in containerization technologies.
4. Today Linux containers allow you to run multiple parts of an app independently, on the same hardware, with far greater control over individual components and life cycles.
5. In combination with the API and DevOps teams, containerized microservices form the basis of cloud-native applications.

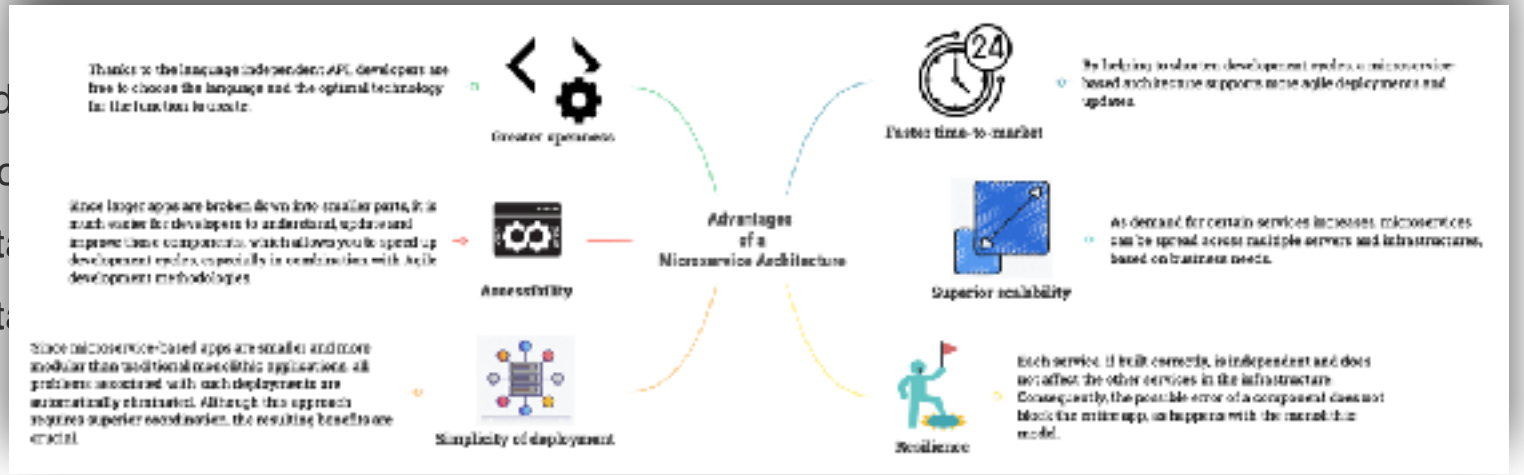
The advantages of a Microservices architecture

The advantages of a Microservices architecture

Based on a distributed architecture, microservices allow for more efficient development and routines. The ability to develop multiple microservices simultaneously allows multiple developers to work on the same app simultaneously, reducing development time.

The advantages of a Microservices architecture

Based
develo
simult
simult



Disadvantages of the Microservices architecture

Disadvantages of the Microservices architecture

1. If you have decided to switch to an architecture based on microservices, it is essential to be able to change the structure of communication and collaboration between the teams, not just that of the apps.

Disadvantages of the Microservices architecture

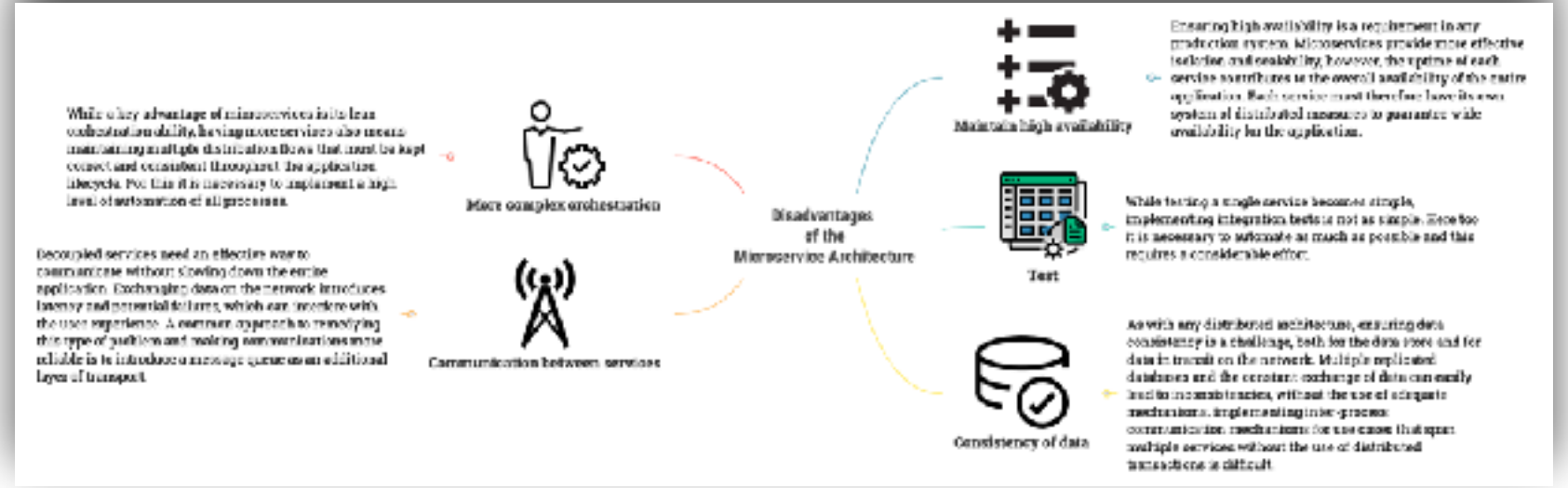
1. If you have decided to switch to an architecture based on microservices, it is essential to be able to change the structure of communication and collaboration between the teams, not just that of the apps.
2. Changing corporate culture can be difficult, in part because each team follows its own deployment cadence and is responsible for a single service for a specific group of customers. While not a problem strictly related to developers, overcoming it becomes essential for the success of the architecture based on microservices.

Disadvantages of the Microservices architecture

1. If you have decided to switch to an architecture based on microservices, it is essential to be able to change the structure of communication and collaboration between the teams, not just that of the apps.
2. Changing corporate culture can be difficult, in part because each team follows its own deployment cadence and is responsible for a single service for a specific group of customers. While not a problem strictly related to developers, overcoming it becomes essential for the success of the architecture based on microservices.
3. Partitioning an application into independent services also means that there are multiple moving parts to maintain. This is clearly quite evident in this type of system but there are consequently new factors to consider.

Disadvantages of the Microservices architecture

1. If you have decided to switch to an architecture based on microservices, it is essential to



system but there are consequently new factors to consider.

Liferay as Microservices Platform

Liferay as Microservices Platform

1. The Platform
2. OSGi *μ*Services
3. Deploy to Container Platform

The Platform

The Platform

1. OSGi as a core technology of Liferay

The Platform

1. OSGi as a core technology of Liferay
2. Defined best practices can be helpful

The Platform

1. OSGi as a core technology of Liferay
2. Defined best practices can be helpful
3. Transparent platform valuable for developers

The Platform

1. OSGi as a core technology of Liferay
2. Defined best practices can be helpful
3. Transparent platform valuable for developers
4. Opportunity for iterative modularization

The Platform

“

[…] you shouldn’t start with a microservices architectures. Instead begin with a monolith, keep it modular, and split it into microservices once the monolith become a problem.

”

Martin Fowler (2014)

https://martinfowler.com/articles/microservices.html

The Platform

The Platform

What are the key aspects of a platform like Liferay that embraces modularity?

The Platform

What are the key aspects of a platform like Liferay that embraces modularity?

1

Allow **independence** of development, deploy and evolution

The Platform

What are the key aspects of a platform like Liferay that embraces modularity?

1

Allow **independence** of development, deploy and evolution

2

Combine together the components that already exist

The Platform

What are the key aspects of a platform like Liferay that embraces modularity?

1. Allow **independence** of development, deploy and evolution
2. **Combine** together the components that already exist
3. **Reuse** and facilitate the **customization**

OSGi *μ*Services

“

What I am promoting is the idea of *μ*Services, the concepts of an OSGi service as a design primitive.

”

Peter Kriens (2010)

https://blog.osgi.org/2010/03/services.html

OSGi *μ*Services

OSGi *μ*Services

1. In OSGi Services have existed for even longer
    - A way to maintain loose coupling between modules
    - A dynamic representation of a changing physical environment

OSGi *μ*Services

1. In OSGi Services have existed for even longer
   - A way to maintain loose coupling between modules
   - A dynamic representation of a changing physical environment

2. OSGi services are ultra-lightweight microservices
   - Just a call from one object to another
   - OSGi services are usually in-process

OSGi μServices

1. In OSGi Services have existed for even longer
   - A way to maintain loose coupling between modules
   - A dynamic representation of a changing physical environment

2. OSGi services are ultra-lightweight microservices
   - Just a call from one object to another
   - OSGi services are usually in-process

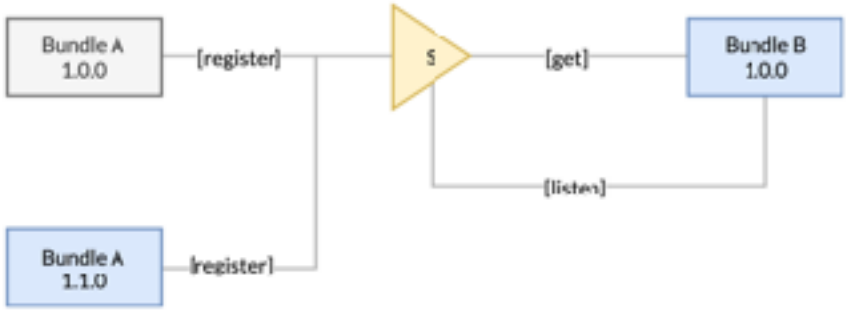
3. An OSGi service is a Java Object
   - It's registered in the OSGi Service Registry
   - The registration includes the Service's API

OSGi μServices

1. In OSGi Services have existed for even longer
   - A way to maintain loose coupling between modules
   - A dynamic representation of a changing physical environment

2. OSGi services are ultra-lightweight microservices
   - Just a call from one object to another
   - OSGi services are usually in-process

3. An OSGi service is a Java Object
   - It's registered in the OSGi Service Registry
   - The registration includes the Service's API



OSGi *μ*Services

OSGi μServices

1. OSGi services also have properties
   - These provide additional information
   - They can help clients when multiple services exist
   - They can also communicate other information...

OSGi μServices

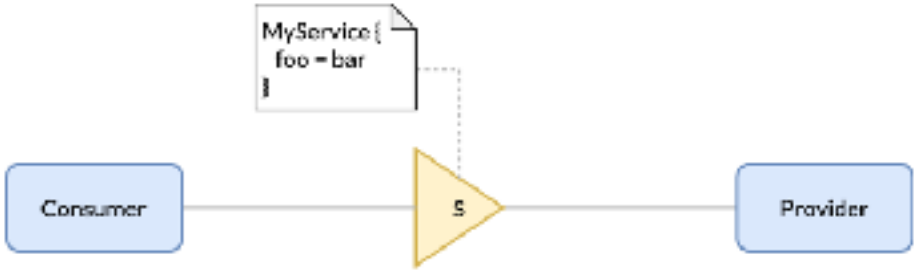
1. OSGi services also have properties
   - These provide additional information
   - They can help clients when multiple services exist
   - They can also communicate other information...

2. OSGi Services are an excellent way to build microservices
   - Easy ways to Publish an Service
   - Easy ways to Consume an Service

OSGi μServices

1. OSGi services also have properties
   - These provide additional information
   - They can help clients when multiple services exist
   - They can also communicate other information...

2. OSGi Services are an excellent way to build microservices
   - Easy ways to Publish an Service
   - Easy ways to Consume an Service



OSGi μServices

Microservice according to Fowler	OSGi μServices
single application as a suite of small services	✔
running in its own process	✘
communicating with lightweight mechanisms	✔
built around business capabilities	✔
independently deployable	✔
minimum of centralized management	✔
may be written in different programming languages	✔
use different data storage technologies	✔

OSGi *μ*Services

OSGi *μ*Services

1. Separate processes do come with a benefit
   - Full isolation
   - Easy Scalability

OSGi μServices

1. Separate processes do come with a benefit
   - Full isolation
   - Easy Scalability

2. Two OSGi specifications
   - Remote Services: Transport
   - Remote Service Admin: Topology, Service Discovery

OSGi *μ*Services

1. Separate processes do come with a benefit
   - Full isolation
   - Easy Scalability

OSGi Remote Services!

2. Two OSGi specifications
   - Remote Services: Transport
   - Remote Service Admin: Topology, Service Discovery

OSGi *μ*Services

OSGi μServices

The best of both worlds

- Independent deployment, iterative development
- Fault tolerant – but no full isolation
- No network latencies, reduced call stack
- More freedom to choose technology stack
- Less demanding for operations
- Established monitoring and deployment strategies

Deploy to Container Platform

Deploy to Container Platform

- If you are building a microservice architecture, containers are the ideal deployment unit for any microservice.

Deploy to Container Platform

- If you are building a microservice architecture, containers are the ideal deployment unit for any microservice.
- Liferay has long embraced and works constantly to support the technology of containers.

Deploy to Container Platform

- If you are building a microservice architecture, containers are the ideal deployment unit for any microservice.
- Liferay has long embraced and works constantly to support the technology of containers.
- Liferay has integrated the development environment with tools that facilitate the IT team for creating docker images and this is enabling for a context of DevOps.

Deploy to Container Platform

- If you are building a microservice architecture, containers are the ideal deployment unit for any microservice.
- Liferay has long embraced and works constantly to support the technology of containers.
- Liferay has integrated the development environment with tools that facilitate the IT team for creating docker images and this is enabling for a context of DevOps.
- Directly from the Liferay Workspace we are able to create our docker image that contains the Liferay platform and the additional modules that we have developed and that we want to release.

Deploy to Container Platform

Deploy to Container Platform

Liferay provides Docker images for:

- Liferay Portal
- Liferay DXP
- Liferay Commerce
- Liferay Portal Snapshots

You can pull Liferay's Docker images from Docker Hub and manage them yourself. Liferay Workspace, however, provides an easy way to integrate Docker development into your existing development workflow with preconfigured Gradle tasks.

Deploy to Container Platform

Liferay provides

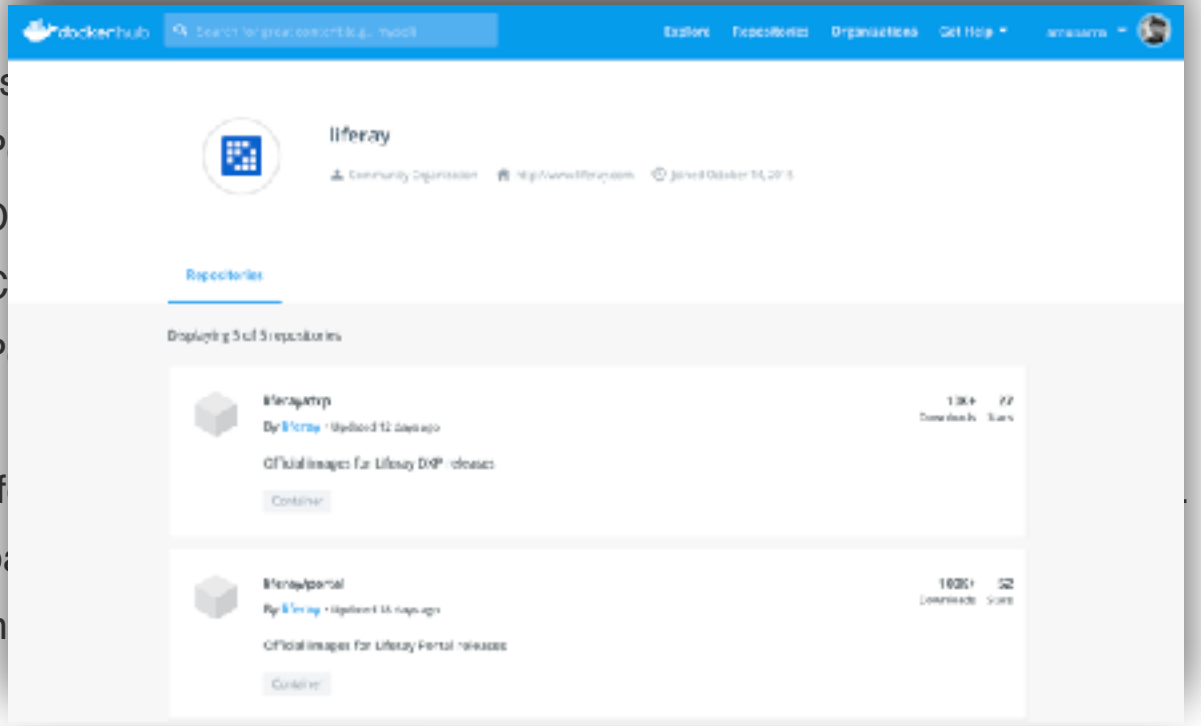
- Liferay P
- Liferay D
- Liferay C
- Liferay P

You can pull Lif

Liferay Workspa

development in

tasks.



Deploy to Container Platform

Liferay provides Docker images for:

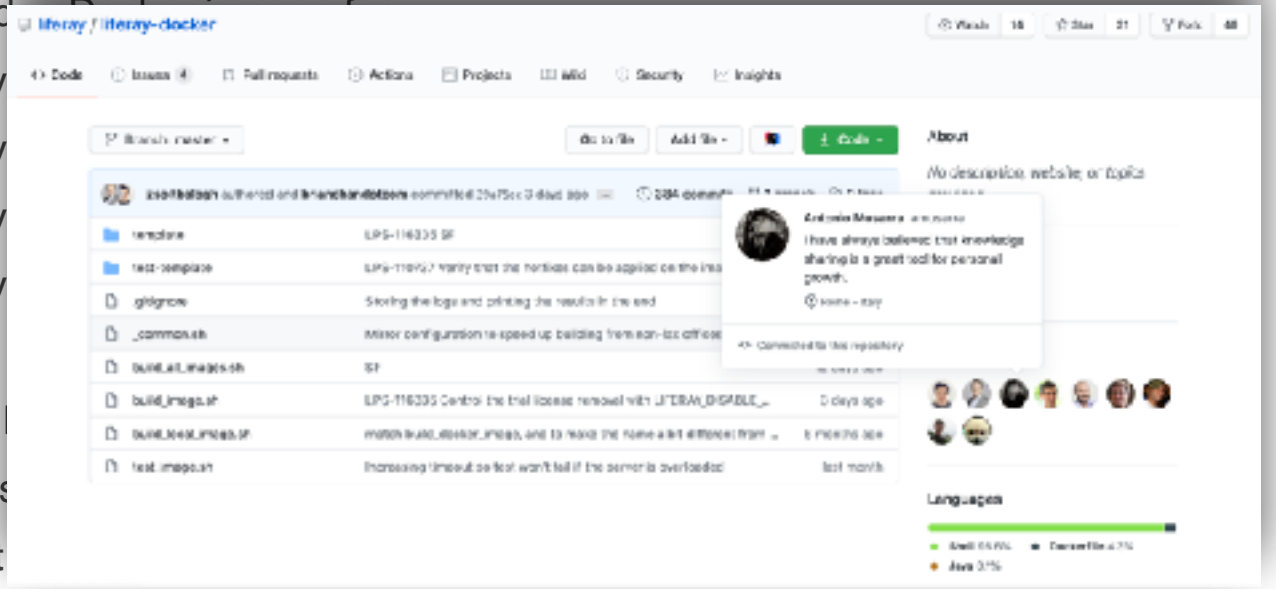
- Liferay Portal
- Liferay DXP
- Liferay Commerce
- Liferay Portal Snapshots

You can pull Liferay's Docker images from Docker Hub and manage them yourself. Liferay Workspace, however, provides an easy way to integrate Docker development into your existing development workflow with preconfigured Gradle tasks.

Deploy to Container Platform

Liferay provid

- Liferay
- Liferay
- Liferay
- Liferay



You can pull

Liferay Works

development

tasks.

Deploy to Container Platform

Liferay provides Docker images for:

- Liferay Portal
- Liferay DXP
- Liferay Commerce
- Liferay Portal Snapshots

You can pull Liferay's Docker images from Docker Hub and manage them yourself. Liferay Workspace, however, provides an easy way to integrate Docker development into your existing development workflow with preconfigured Gradle tasks.

Deploy to Container Platform

Liferay provides Do

- Liferay Porta
- Liferay DXP
- Liferay Comm
- Liferay Porta

You can pull Liferay self.
Liferay Workspace
development into y e
tasks.

```
50      #
51      # This property is available for backwards compatibility. Please set the
52      # property "liferay.home" instead.
53      #
54      # Env: LIFERAY_RESOURCE_PERIOD_REPOSITORIES_PERIOD_ROOT
55      #
56      resource.repositories.root=${default.liferay.home}
57
58  ##
59  ## Portal Context
60  ##
61
62      #
63      # Set the application server's protocol. Lucene will use it to load the
64      # index from the cluster when the hostname and port are not detected on the
65      # first request. Note that this property refers to the application server's
66      # protocol, and not the web server's as specified in the property
67      # "web.server.protocol".
68      #
69      # Env: LIFERAY_PORTAL_PERIOD_INSTANCE_PERIOD_PROTOCOL
70      #
71      portal.instance.protocol=
72      #portal.instance.protocol=http
73      #portal.instance.protocol=https
```

Deploy to Container Platform

Liferay provides Docker images for:

- Liferay Portal
- Liferay DXP
- Liferay Commerce
- Liferay Portal Snapshots

You can pull Liferay's Docker images from Docker Hub and manage them yourself. Liferay Workspace, however, provides an easy way to integrate Docker development into your existing development workflow with preconfigured Gradle tasks.

Deploy to Container Platform

Deploy to Container Platform

Thanks to the support for containers, it is possible to deploy our Liferay applications on Container Platform, such as **Google Kubernetes Engine** or **Read Hat OpenShit Container Platform**.

Deploy to Container Platform

Thanks to the support for containers, it is possible to deploy our Liferay applications on Container Platform, such as **Google Kubernetes Engine** or **Read Hat OpenShit Container Platform**.

Marcial Calvo Valenzuela (consultant at Liferay) recently published on the Liferay blog, the article Deploying Liferay 7.3 CE in Kubernetes. This article describe how to deploy Liferay 7.3 CE in conjunction with a service stack on Kubernetes, Nginx as an ingress, MySQL 5.7 as a database and ElasticSearch as engine search indexer, as well as, how to manage K8s from *GUI* dashboard.

Deploy to Container Platform

Thanks

applicat

Read H

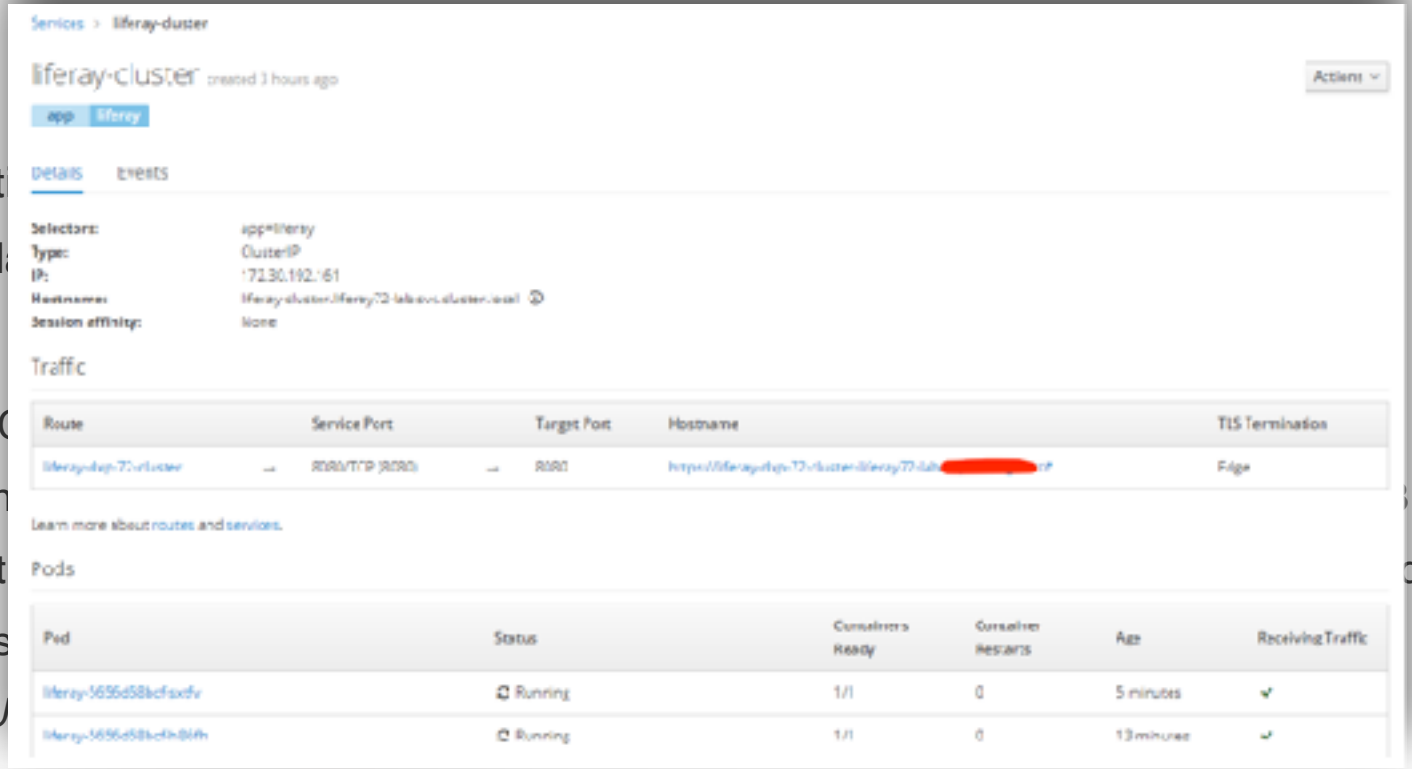
Marcial

Deployin

conjunct

and Elas

from *GU*



the article

CE in

database

Deploy to Container Platform

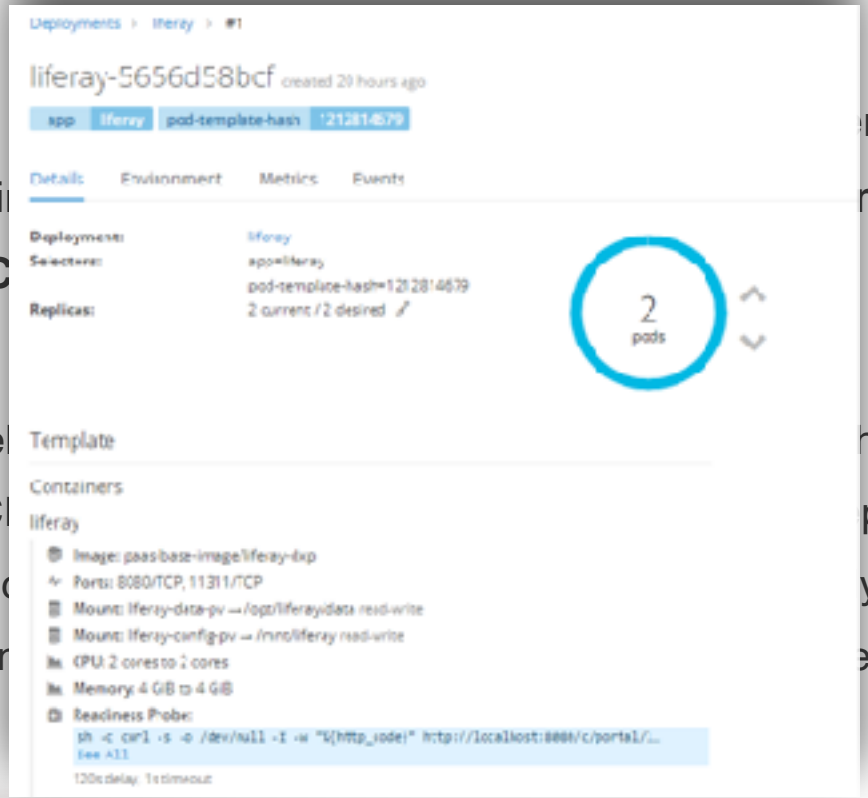
Thanks to the support for containers, it is possible to deploy our Liferay applications on Container Platform, such as **Google Kubernetes Engine** or **Read Hat OpenShit Container Platform**.

Marcial Calvo Valenzuela (consultant at Liferay) recently published on the Liferay blog, the article Deploying Liferay 7.3 CE in Kubernetes. This article describe how to deploy Liferay 7.3 CE in conjunction with a service stack on Kubernetes, Nginx as an ingress, MySQL 5.7 as a database and ElasticSearch as engine search indexer, as well as, how to manage K8s from *GUI* dashboard.

Deploy to Container Platform

Thanks to the support ...eray
applications on Contai...**ngine** or
Read Hat OpenShit C...

Marcial Calvo Valenzuel...he Liferay blog, the article
Deploying Liferay 7.3 C...ploy Liferay 7.3 CE in
conjunction with a servi...ySQL 5.7 as a database
and ElasticSearch as er...e K8s
from *GUI* dashboard.



Digital Experience Platform

Digital Experience Platform

Digital Experience Platform

Liferay is a DXP platform. What are the most common reasons for not using a DXP platform?

Digital Experience Platform

Liferay is a DXP platform. What are the most common reasons for not using a DXP platform?

1. You can build your digital presence from scratch without a platform approach. This strategy will give you the most freedom, since you can create the ultimate customized solution to meet your needs.

Digital Experience Platform

Liferay is a DXP platform. What are the most common reasons for not using a DXP platform?

1. You can build your digital presence from scratch without a platform approach. This strategy will give you the most freedom, since you can create the ultimate customized solution to meet your needs.
2. Digital Experience Platforms are monoliths and it is not “cool” to use a monolithic architecture in 2020. Besides, with a monolith, it is harder to take advantage of containerization (think Docker and Kubernetes)

Digital Experience Platform

Liferay is a DXP platform. What are the most common reasons for not using a DXP platform?

1. You can build your digital presence from scratch without a platform approach. This strategy will give you the most freedom, since you can create the ultimate customized solution to meet your needs.
2. Digital Experience Platforms are monoliths and it is not “cool” to use a monolithic architecture in 2020. Besides, with a monolith, it is harder to take advantage of containerization (think Docker and Kubernetes)
3. A Digital Experience Platform is a heavyweight compared to JavaScript front-ends with RESTful microservice APIs.

Digital Experience Platform

Liferay is a DXP platform. What are the most common reasons for not using a DXP platform?

1. You can build your digital presence from scratch without a platform approach. This strategy will give you the most freedom, since you can create the ultimate customized solution to meet your needs.
2. Digital Experience Platforms are monoliths and it is not “cool” to use a monolithic architecture in 2020. Besides, with a monolith, it is harder to take advantage of containerization (think Docker and Kubernetes)
3. A Digital Experience Platform is a heavyweight compared to JavaScript front-ends with RESTful microservice APIs.
4. A Digital Experience Platform is harder to scale up for internet-scale applications, and it is harder to handle traffic bursts.

Digital Experience Platform

Digital Experience Platform

What are the reasons for using a DXP platform?

Digital Experience Platform

What are the reasons for using a DXP platform?

1. A single unified platform that can be used to handle multiple use cases around which an enterprise can train its team.

Digital Experience Platform

What are the reasons for using a DXP platform?

1. A single unified platform that can be used to handle multiple use cases around which an enterprise can train its team.
2. A whole slew of out-of-the-box functionality including content management, personalization and targeting, security, collaboration, forms, workflow, analytics and optimization, commerce and more.

Digital Experience Platform

What are the reasons for using a DXP platform?

1. A single unified platform that can be used to handle multiple use cases around which an enterprise can train its team.
2. A whole slew of out-of-the-box functionality including content management, personalization and targeting, security, collaboration, forms, workflow, analytics and optimization, commerce and more.
3. Multiple deployments. It is highly unlikely your organization will be deploying just one page, or even just one site. A Digital Experience Platform offers a way to manage all your pages and sites from one place, with a robust permissioning system to define roles and workflows.

Digital Experience Platform

What are the reasons for using a DXP platform?

1. A single unified platform that can be used to handle multiple use cases around which an enterprise can train its team.
2. A whole slew of out-of-the-box functionality including content management, personalization and targeting, security, collaboration, forms, workflow, analytics and optimization, commerce and more.
3. Multiple deployments. It is highly unlikely your organization will be deploying just one page, or even just one site. A Digital Experience Platform offers a way to manage all your pages and sites from one place, with a robust permissioning system to define roles and workflows.
4. Better metrics. A Digital Experience Platform often provides engagement metrics that help businesses understand which content is performing well and which isn't so they can make informed decisions about what pages and messaging are resonating with customers.

Digital Experience Platform

Digital Experience Platform

1. Collaboration is key to creating modern digital experiences. Business users in marketing and other departments need to be empowered to modify content and messaging on the fly with minimal intervention from IT. A Digital Experience Platform allows developers to create pre-set content fragments that business users can reuse at will across sites and pages.

Digital Experience Platform

1. Collaboration is key to creating modern digital experiences. Business users in marketing and other departments need to be empowered to modify content and messaging on the fly with minimal intervention from IT. A Digital Experience Platform allows developers to create pre-set content fragments that business users can reuse at will across sites and pages.

2. Headless and even decoupled capabilities for organizations with robust front-end development teams that wish to create their own presentation layers.

Digital Experience Platform

Digital Experience Platform

Let's take a few minutes to respond to opponents of the DXP platform.

Digital Experience Platform

Let's take a few minutes to respond to opponents of the DXP platform.

1. While the option to build your own digital platform and solution can look enticing to your engineering team, it is not to be taken lightly. Do you have the resources to build, maintain and improve your platform for years to come?

Digital Experience Platform

Let's take a few minutes to respond to opponents of the DXP platform.

1. While the option to build your own digital platform and solution can look enticing to your engineering team, it is not to be taken lightly. Do you have the resources to build, maintain and improve your platform for years to come?
2. It is true that a digital experience platform is a monolith from the distribution point of view but Liferay is internally modular. By the way, you can successfully containerize Liferay.

Digital Experience Platform

Let's take a few minutes to respond to opponents of the DXP platform.

1. While the option to build your own digital platform and solution can look enticing to your engineering team, it is not to be taken lightly. Do you have the resources to build, maintain and improve your platform for years to come?
2. It is true that a digital experience platform is a monolith from the distribution point of view but Liferay is internally modular. By the way, you can successfully containerize Liferay.
3. Yes, a Digital Experience Platform is heavier compared to JavaScript front-ends with RESTful microservice APIs, but it also comes with significant, out-of-the-box functionality that can be used to create personalized, collaborative experiences for your users.

Digital Experience Platform

Let's take a few minutes to respond to opponents of the DXP platform.

1. While the option to build your own digital platform and solution can look enticing to your engineering team, it is not to be taken lightly. Do you have the resources to build, maintain and improve your platform for years to come?
2. It is true that a digital experience platform is a monolith from the distribution point of view but Liferay is internally modular. By the way, you can successfully containerize Liferay.
3. Yes, a Digital Experience Platform is heavier compared to JavaScript front-ends with RESTful microservice APIs, but it also comes with significant, out-of-the-box functionality that can be used to create personalized, collaborative experiences for your users.
4. Yes, a Digital Experience Platform is harder to scale up for internet-scale applications and traffic bursts compared to JavaScript front-ends and microservices. However, Digital Experience Platforms scale just fine for 99% (or higher) of use cases, many with millions of users and over a hundred million page hits a month.

Headless CMS open to Microservices

What Is a Headless CMS?

What Is a Headless CMS?

A Headless CMS is a backend only content management system which works as a content repository and gives access to this content via REST (or GraphQL) services. **Liferay fully reflects this definition**.

What Is a Headless CMS?

A Headless CMS is a backend only content management system which works as a content repository and gives access to this content via REST (or GraphQL) services. **Liferay fully reflects this definition**.

In traditional CMS, you have the following core subsystems:

- Content creation and management
- Publication workflow
- Content delivery
- Analysis and monitoring

What Is a Headless CMS?

A Headless CMS is a backend only content management system which works as a content repository and gives access to this content via REST (or GraphQL) services. **Liferay fully reflects this definition**.

In traditional CMS, you have the following core subsystems:

- Content creation and management
- Publication workflow
- Content delivery
- Analysis and monitoring

Headless CMS focuses just on content creation and publication workflow. It is your application's responsibility to get the content and display it in appropriate way based on your users' needs, devices they use, and channels they operate on.

Liferay as a Headless platform

Liferay as a Headless platform

More flexibility and control

Liferay's foundations as a development platform continue to evolve with the addition of headless APIs for out-of-the-box services. Developers now have unparalleled flexibility to integrate Liferay into all systems, whether it's collecting data in Liferay or bringing Liferay into an existing ecosystem.

Liferay as a Headless platform

More flexibility and control

Liferay's foundations as a development platform continue to evolve with the addition of headless APIs for out-of-the-box services. Developers now have unparalleled flexibility to integrate Liferay into all systems, whether it's collecting data in Liferay or bringing Liferay into an existing ecosystem.

Compliant with OpenAPI standards

Liferay's API level supports the OpenAPI specification, the most popular open source framework for RESTful APIs. GraphQL is also supported for easier development.

Liferay as a Headless platform

More flexibility and control

Liferay's foundations as a development platform continue to evolve with the addition of headless APIs for out-of-the-box services. Developers now have unparalleled flexibility to integrate Liferay into all systems, whether it's collecting data in Liferay or bringing Liferay into an existing ecosystem.

Compliant with OpenAPI standards

Liferay's API level supports the OpenAPI specification, the most popular open source framework for RESTful APIs. GraphQL is also supported for easier development.

Support for Headless CMS and Commerce

Liferay supports APIs in all of our content and commerce features, allowing you to create a fluid front end for the entire Customer Journey.

Liferay as a Headless platform

More flexibility and

Liferay's foundatio headless APIs
for out-of-the-box eray into all
systems, whether stem.

Compliant with Oper

Liferay's API level amework for RESTful APIs.
GraphQL is also s

Support for Headles

Liferay supports A e a fluid front end for the entire
Customer Journey

Liferay as a Headless platform

More flexibility and

Liferay's foundatio headless APIs
for out-of-the-box eray into all
systems, whether stem.

Compliant with Oper

Liferay's API level amework for RESTful APIs.
GraphQL is also s

Support for Headles

Liferay supports A e a fluid front end for the entire
Customer Journey

Liferay as a Headless platform

More flexibility and

Liferay's foundatio headless APIs

for out-of-the-box eray into all

systems, whether stem.

Compliant with Oper

Liferay's API level amework for RESTful APIs.

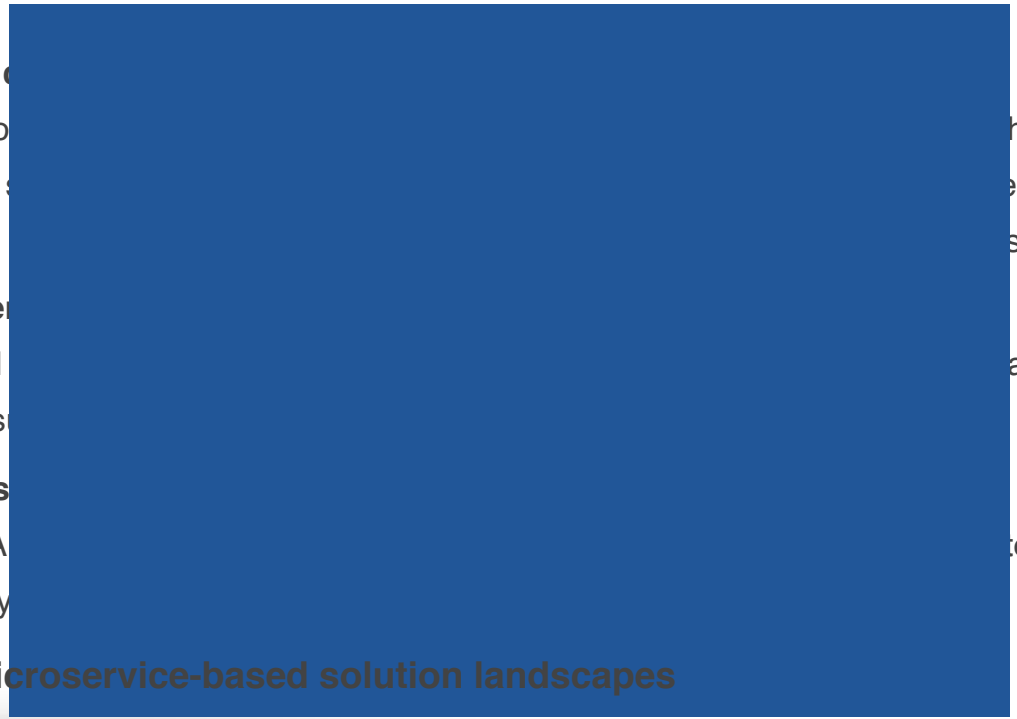
GraphQL is also s

Support for Headles

Liferay supports A e a fluid front end for the entire

Customer Journey

Nicely fits into microservice-based solution landscapes

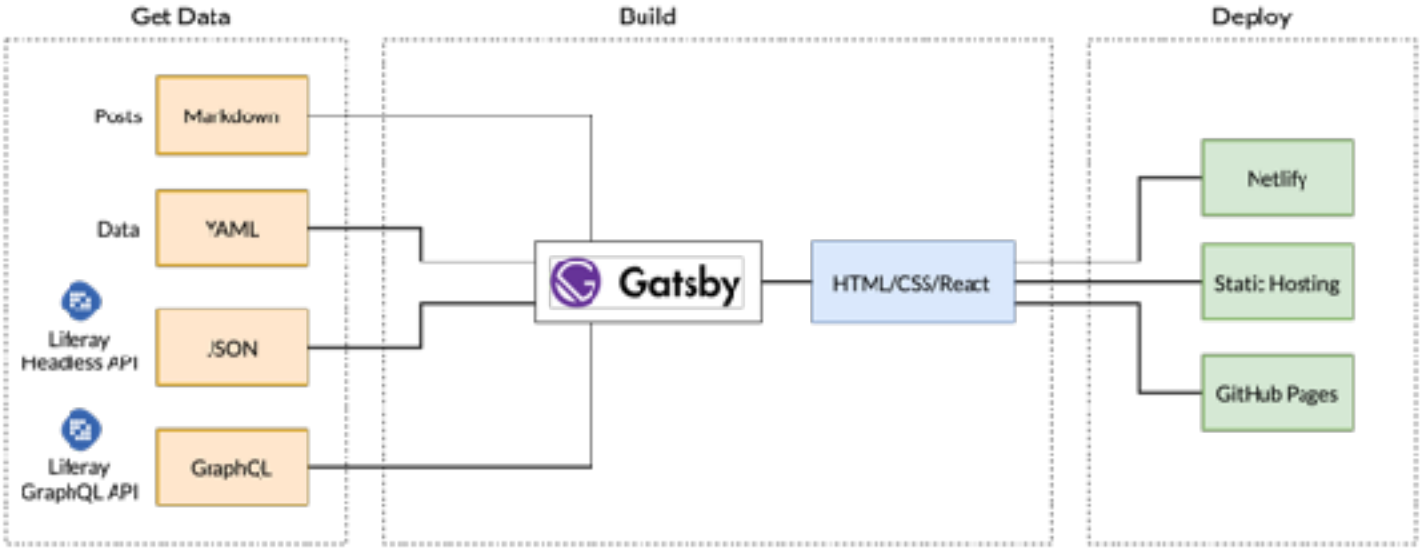


A Headless API example using Gatsby

A Headless API example using Gatsby

Chris Mount on Liferay's blog has published two articles dealing with this topic.

A Headless API example using Gatsby



Chris Mount on Liferay's blog has published two articles dealing with this topic.

Domande

CONTATTI

Antonio Musarra

Email antonio.musarra@smc.it

Skype amusarra

GitHub https://github.com/amusarra

Mobile +39 345 1113480

Phone

Thank you ☺