Prodapt powering global telecom

Give your apps a new shine with microservices

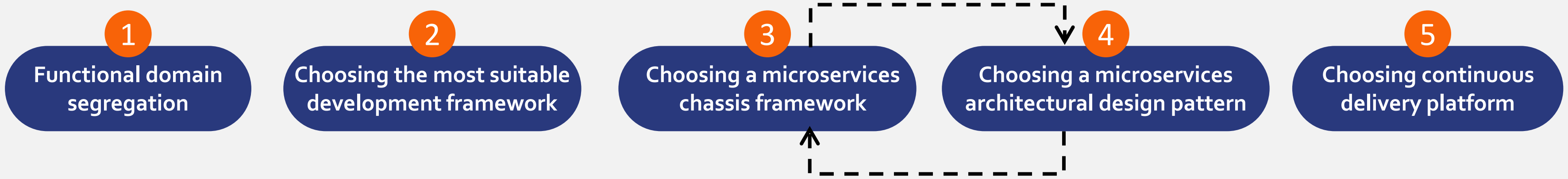
A simplified approach to breaking up your monolithic applications into microservices

Credits | Kalpana Angamuthu | Mogan A.B. | Madalai Muthu Jacob

Strategies for refactoring a legacy monolithic application to a microservices architecture

For digital native players, the strategy is quite straight forward since all the functional modules are developed from scratch as a microservices component. However, in case of larger DSPs, the siloed, bulky, monolithic systems are not flexible enough to scale with changing customer needs. Telcos are now looking to embrace the concept of microservices as a new architectural framework to achieve business goals such as agility and flexibility, to operationalize manifold digital business models.

The below infographic highlights the five mandatory selections for DSPs to consider before embarking on the microservices journey. Each stage has a proposed qualification checklist with key recommendations to navigate through the migration journey.


1
Functional domain segregation
2
Choosing the most suitable development framework
3
Choosing a microservices chassis framework
4
Choosing a microservices architectural design pattern
5
Choosing continuous delivery platform

The migration journey approach discussed throughout this insight is applicable to any DSPs who have huge, legacy monolithic applications in place and prepare to switch to microservices.

Stage 1 Prioritizing the functional modules for migrating into microservices

Part 1/2

Majority of the DSPs operate with multi-vendor based legacy OSS and BSS applications. However, one of the major challenge for DSPs is to understand and strategize the right place to start in their microservices journey.

For any DSP, there is no standard rule for selection of modules for migration and it is mainly driven by technical, business and operational needs.

Segregating the functional modules for migrating into microservices

Below checklist assists in segregating modules with respect to domain and sub-domain functions to help DSPs identify the top best-suited modules for microservices within a legacy application

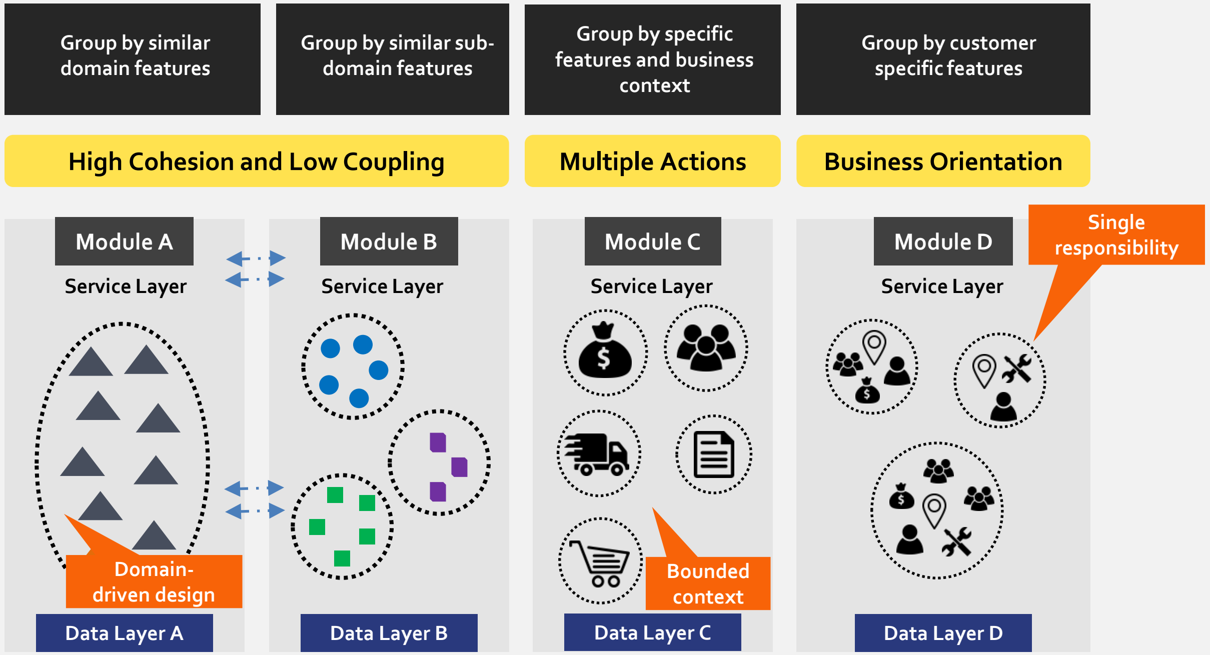
- ✓ Does the module ***handle large volume of events/traffic***?
- ✓ Does the module require ***dynamic scaling during run time***?
- ✓ Does the module get ***frequently modified or enhanced***?
- ✓ Does the module get ***customized for different customers***?
- ✓ Is the module subject to ***frequent maintenance*** releases?
- ✓ Is the module being ***developed by geographically distributed teams***?
- ✓ Does the module ***perform similar function*** ?
- ✓ Does the module have ***limited interfaces*** with other modules?
- ✓ Does the module perform a ***highly computationally intensive task***?

If most of the answers to the checklist are YES, then it's the most probable candidate for migrating to microservices.

The next step shows an example of a huge monolithic application comprising of various modules and the principles to identify direct candidates for microservices migration.

Stage 1 How domain segregation helps DSPs to identify direct microservices candidates

Part 2/2



Domain-driven design or development

This technique primarily focuses on business domain rather than technology. It is used to group the existing legacy modules with bounded context of domain or sub-domain level.

Single responsibility principle

- The functional relatedness of the elements of a module
- Measure of the strength of relationship between the methods and data of a class

After identifying the top candidates for microservices focus shifts to developing microservice modules.

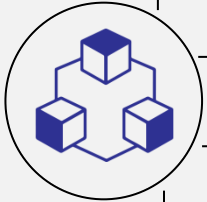
Stage 2 Choosing the most suitable development framework for microservices

There are multiple frameworks, which help in developing microservices. The most suitable framework is chosen based on business requirement and features supported. It is also possible to use more than one framework seamlessly integrated for building microservices. Below is a list of the widely used java-based microservices development frameworks.

Checklist for choosing development Framework

- ✓ Is there a need for **ease of use**?
- ✓ Is there a need for **quick project building**?
- ✓ Does the application require quick introduction of advanced/new features?
- ✓ Does the application require **most advanced RESTful support**?
- ✓ Does the team expect **support from open community**?
- ✓ Does the **framework has minimal size** and **occupy less memory**?
- ✓ Does the framework enable **production-ready** and **fast deployment**?
- ✓ Is **performance** a key factor for the application?

Recommendations on each development framework



Drop Wizard
Quick project building, bootstrapping and fast phased development

Spark
It takes time to introduce advanced features, simple to use and runs fast

Restlet
Powerful and supports most advanced RESTful support, steep learning curve and closed community

Play
Easy to develop, quick project building and bootstrapping, huge base and not suitable for large projects

Spring Boot
Production ready and fast development, feature complete framework (*Spring portfolio*)

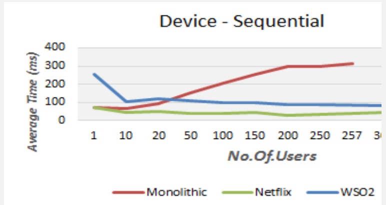
MS4J
A very fast start-up time, low memory footprint, high throughput and low latency

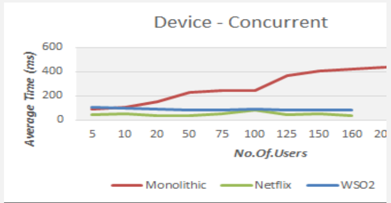
After developing microservices modules, the next step is to make them operational ready with help of chassis framework.

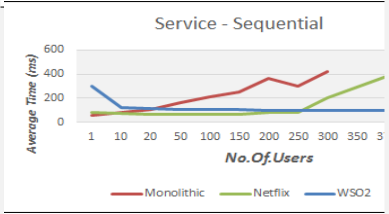
Stage 3 Piloting various chassis frameworks with test drive approach

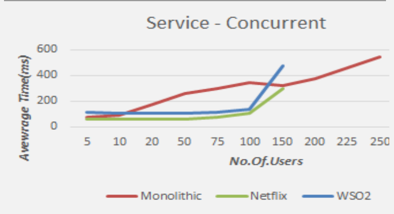
Choosing a microservices chassis framework requires small trials or POCs to evaluate the right one that best suits the application needs and it is recommended to go with the test-drive approach.

A sample test drive approach for choosing the chassis framework









In this context, device and service modules are selected from sample legacy monolithic application for refactoring.

- Spring boot is used to develop two small microservices and implement it in Netflix OSS chassis framework.
- MS4J is used to develop two small microservices and implement it in WSO2 chassis framework.

Then the results are compared to choose the best suited frameworks.

Sample test conducted on a O/BSS monolithic application showed that Netflix and WSO2 frameworks provided almost similar results during trails

- For sequential operations, both the frameworks were found competitive
- For concurrent operations, the performance of both the frameworks was impacted after reaching a threshold
- Nevertheless, both frameworks yielded better results as compared to the monolithic application

Selecting a chassis framework alone does not guarantee application's performance or stability. The success largely depends on the microservices design patterns that make the application fully functional with high performance, resilience and fault tolerance.

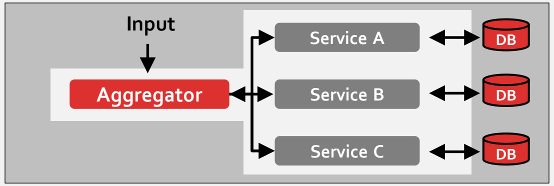
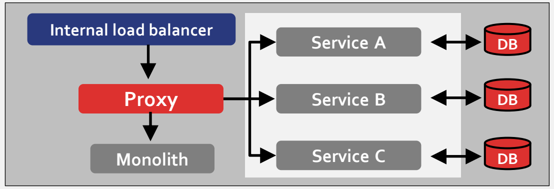
Stage 4 The right choice of microservices design patterns determines application performance

Part 1/2

Microservices design patterns are repeatable solution to a commonly occurring problem in software design. Design patterns act as a template for how to solve a problem that can be used in different situations. Based on individual application characteristics, scalability, migration strategy, performance and deployment needs, DSP should choose the most appropriate one.

The following list explains the most widely used design patterns for migrating telco related applications.

Top recommended microservices design patterns

	When to use	Where to use (examples)
Aggregator microservices pattern 	When aggregation or orchestration of data from multiple services is required.	**Self-care or Customer-Care portals, Billing applications** Various services like user, VAS, interconnect, account and transaction services from various components get aggregated before the actual bill generation process.
Proxy microservices pattern 	In phased migration approach - both monolithic and microservices are co-existing in production environment.	**Migration of selected functions from monolithic** When migrating legacy application to microservices in phased manner, proxy microservices pattern is used. The service requests from front end application will be proxied to monolithic and/or microservices as needed.

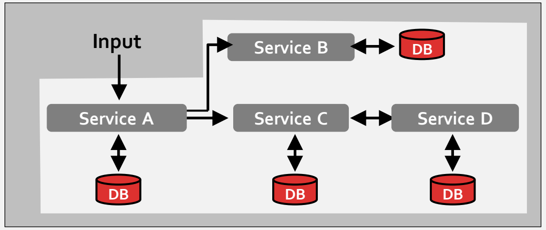
It is recommended for DSPs to perform iterative testing with the choice of chassis framework and design pattern in order to achieve better application performance and scalability.

Stage 4 The right choice of microservices design patterns determines application performance

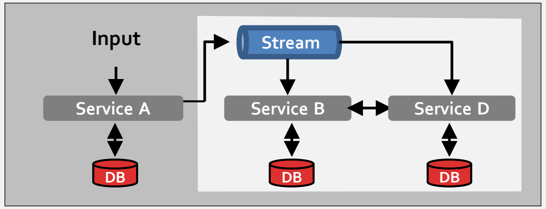
Part 2/2

	When to use	Where to use (examples)
Branch microservices pattern	Scenarios where a request is processed by a single and chained microservices to provide a consolidated response	**Customer usage reports in billing** To view a usage report, Service-A fetches the user details from user management microservice (Service B) and usage details from chained microservices - service & usage microservices (Service C & D).
Event-driven microservices pattern	When asynchronous request handling is required	**Order management** Change of plan request: Service A places the request onto Stream and triggers relevant microservices (e.g. Service B & D) to complete the tasks in the backend in an asynchronous manner. Upon completion, user gets notified.
Chained microservices pattern	Whenever single consolidated response is required after sequential processing	**VAS systems** When the user requests for value added service, Service A invokes policy enforcement microservices (i.e. Service B) to validate the eligibility. After validation, it invokes the credit check service (Service D) to ensure sufficient balance before activation.

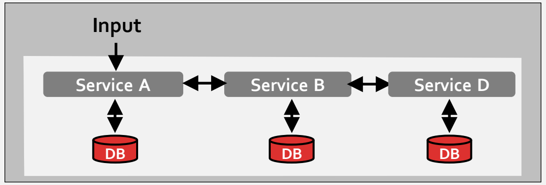
Branch microservices pattern



Event-driven microservices pattern



Chained microservices pattern

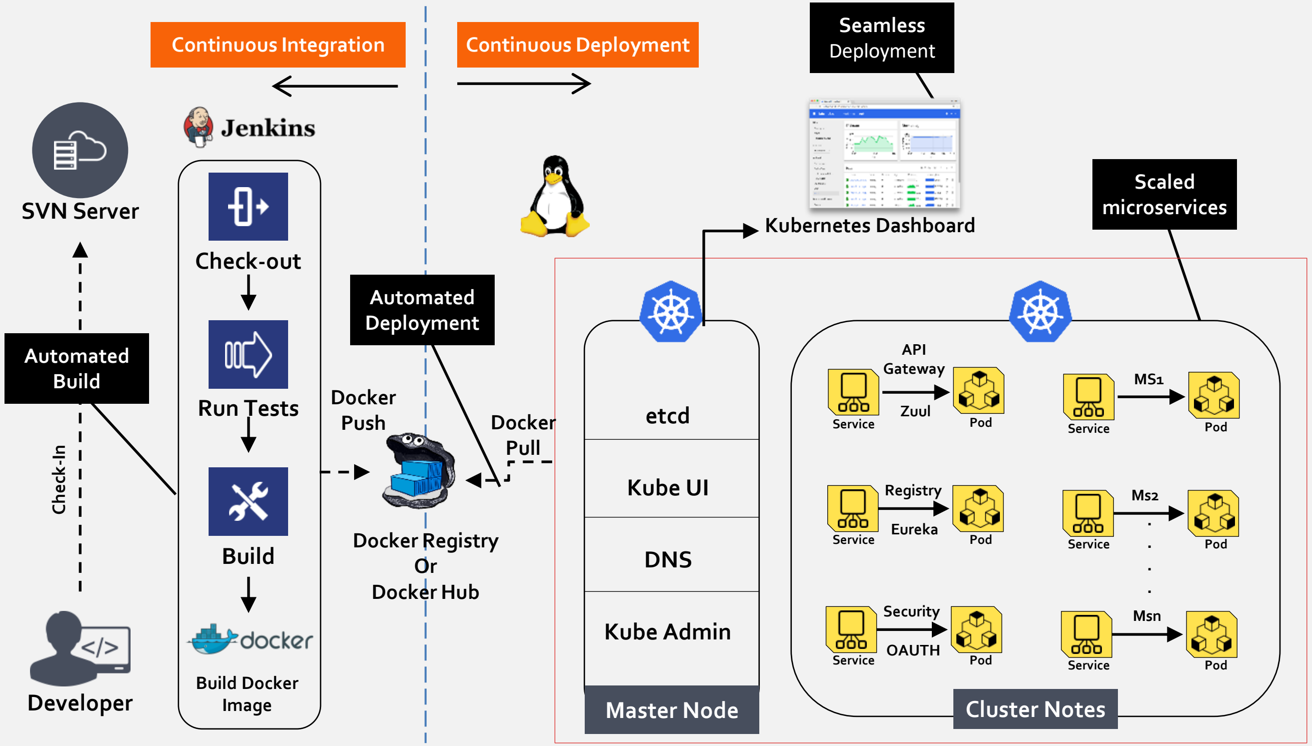


Choosing the right combination of design pattern, chassis and development framework is a critical success factor for the microservices transformation journey.

Stage 5 Continuous delivery platform for DSPs with CI-CD pipeline for microservices

Microservices based architecture brings in a lot of agility, which makes adoption of DevOps culture easy. The final step before the complete development of microservices is to setup a continuous integration and continuous deployment pipeline, which enables the below functionalities.

- Quick independent deployments
- Seamless upgrade of microservices versions
- Auto scaling of services
- Ease of management and reliability



Zero-touch automation capabilities of microservices modules enables DSPs to rollout frequent releases on weekly or daily basis, propelling digital transformation strategy.

Key takeaways

80% of enterprises are counting on microservices

33% intend to transform existing monoliths to microservices

5X faster deployment of new software releases for enterprises who use microservices

9% of enterprises have already adopted microservices

38% enterprises use microservices for new module development while retaining the legacy system

Source: According to recent surveys with multi-corporate enterprises, *leanIX*

Get in touch

USA

Prodapt North America
Tualatin: 7565 SW Mohawk St.,
Phone: +1 503 636 3737

Dallas: 1333, Corporate Dr., Suite 101, Irving
Phone: +1 972 201 9009

New York: 1 Bridge Street, Irvington
Phone: +1 646 403 8158

CANADA

Prodapt Canada Inc.
Vancouver: 777, Hornby Street,
Suite 600, BC V6Z 1S4

UK

Prodapt (UK) Limited
Reading: Davidson House,
The Forbury, RG1 3EU
Phone: +44 (0) 11 8900 1068

EUROPE

Prodapt Solutions Europe
Amsterdam: Zekeringstraat 17A, 1014 BM
Phone: +31 (0) 20 4895711

Prodapt Consulting BV
Rijswijk: De Bruyn Kopsstraat 14
Phone: +31 (0) 70 4140722

Prodapt Germany GmbH
Aschheim: Sonnenstraße 31, 85609
Germany

SOUTH AFRICA

Prodapt SA (Pty) Ltd.
Johannesburg: No. 3,
3rd Avenue, Rivonia
Phone: +27 (0) 11 259 4000

INDIA

Prodapt Solutions Pvt. Ltd.
Chennai: Prince Infocity II, OMR
Phone: +91 44 4903 3000

"Chennai One" SEZ, Thoraipakkam
Phone: +91 44 4230 2300

Bangalore: "CareerNet Campus"
2nd floor, No. 53, Devarabisana Halli,
Phone: +91 44 4903 3000

THANK YOU!

insights@prodapt.com | www.prodapt.com

Prodapt powering global telecom