От монолитно приложение към микросървисна архитектура

...

The journey

https://sli.do -> **#the-trip**

taulia®

- FinTech startup, базиран в San Francisco
- Основана 2009, в София от 2014
- Платформа за supply chain financing
- SaaS

- 15 години опит като програмист и мениджър
- Java, Groovy, Web, Microservices
- Director of Engineering @ Taulia

Мартин Атанасов

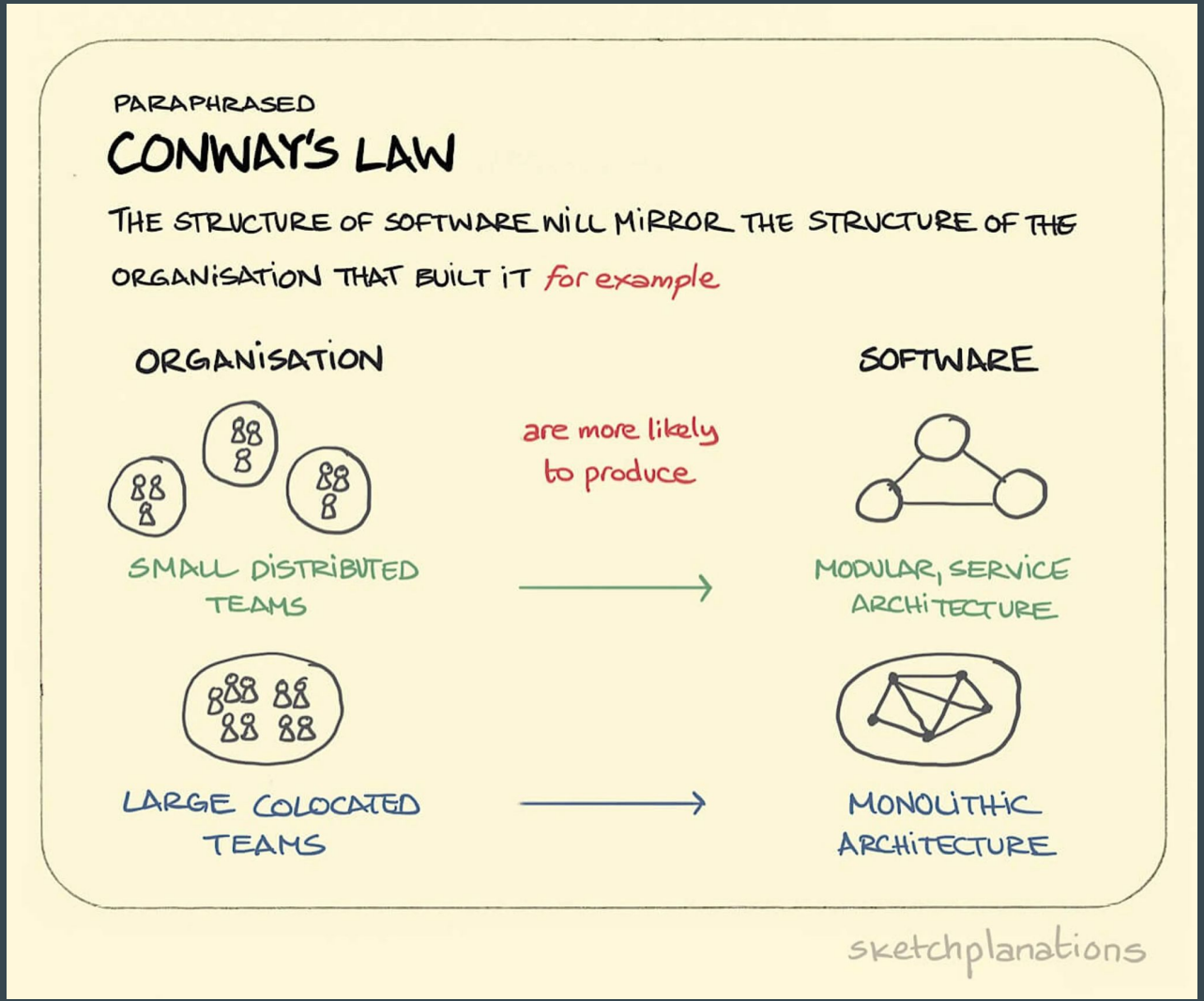
martin.atanasov@taulia.com

План на презентацията

Какво:	Как:	Защо:
Развитието на един **startup**: - **технологии и архитектура** - организация на компанията - проблеми за решаване	Проследяване на различните **етапи от развитието** на компанията. Разглеждане на **промените** и на **причините** зад тях.	Пътят на развитие е относително стандартен. “**Архитектурата** в една компания е **отражение** на организационната ѝ **структура**” - Conway’s law

Conway's law

Ще проследим развитието на технологичните и човешките процеси паралелно



Основни фази

Начален монолит	• избор на технологии • продукт на пазара
Растящ монолит	• performance проблеми • скорост на разработка
Микросървиси	• миграция на монолита • екипи и ownership
Стандартни микросървиси	• миграция на старите микросървиси • оркестрация, deployment, тестови среди

фаза 0:
Начален монолит

Сформиране на екип

- основатели + няколко наети
- един “офис”

Избор на технологии

- предпочитания на основателите
- популярни
- пърза разработка

Продукт на пазара


Groovy


GRAILS


Gradle


MySQL®


amazon
web services

Базиран на Java

компилира се до java byte code

работи върху java VM

съвместим с всички java frameworks (Spring)

Apache Software Foundation

По-гъвкав от Java

олекотен синтаксис

closures (преди java)

скриптова парадигма

(псевдо)слабо типизиран


```groovy
def monthMap = [ 'January' : 31, 'February' : 28, 'March' : 31 ]  // Declares a map
assert monthMap['March'] == 31  // Accesses an entry

monthMap['April'] = 30  // Adds an entry to the map
assert monthMap.size() == 4

monthMap.each { // iterate map and print content
  println "The month of ${it.key} has ${it.value} days"
}
```

THE web framework for Groovy

Convention over configuration

Собствен template engine (gsp)

Собствен ORM framework (GORM)

Scaffolding

Rapid prototyping

Фаза 0

монолитен web server, базиран на grails

една mysql база

ръчен build & deployment на AWS

един екип на една локация

едно code repository

липса на **процес**

продукт и roadmap, управлявани от **основателите**

фаза 1:
Растящ монолит

Растеж на екипите

Скалиране на платформата

Развитие на продукта

Основни въпроси

Развитие на продукта

Как да **продължим** да печелим **клиенти**?

Как да **доставяме** повече **стойност** на текущите?

Растеж на екипите

Как да **увеличим броя на инженерите**, за да **доставяме повече** и по-бързо?

Как да структурираме **development процеса**?

Скалиране на платформата

Как да **подсигурим капацитет** на платформата за повече данни?

Как да я направим по-удобна за **разработка** и **deployment**?

Развитие на продукта

Основателите са инженери

- опит в SAP development света
- идея, започнала като SAP module
- развита до продукт
- product-market fit, финансиране

Наемане на опитни лидери

Product Management специалисти, които да дадат насока на развитие

Engineering Managers, които да развият development **процеси**

Развитие на екипите

Растеж и нови локации

Увеличен брой инженери в San Francisco и отваряне на 3 нови локации (едната в **София**)

Досегашният “процес” не работи (опитахме)

Team ownership

Сформиране на по-малки **Agile екипи**

Ownership област за всеки екип

Continuous delivery of value

Развитие на платформата

Растящ монолит

Модулиране и структуриране на кода (но все пак монолит)

Performance оптимизации (но базата данни е една)

Фокус на unit/integration тестове

Скалиране на ресурсите в AWS (но базата данни продължава да е една)

Растящи екипи

Ownership. Всеки екип работи по собствените си модули (но все пак в един компонент и в едно repository)

Автоматизиран deployment с Jenkins

Release to production след всеки спринт

Фокус на качеството. Pre-production среда. **Automation testing екип.**

Фаза 1

(още по-дебел) монолитен web server, базиран на grails

една mysql база

автоматичен build & deployment на AWS

няколко dev екипа на няколко локации

отделен **automation** екип

едно code repository

Agile **процес** с **frequent delivery**

продукт и roadmap, управлявани от **експерти**

Платформата не може да скалира достатъчно

Базата данни е **performance bottleneck**
Multi tenant платформа с растящ обем от данни. При всеки login:

- ➔ users (10s of millions)
- ➔ companies (millions)
- ➔ invoices (100s of millions)
- ➔ lnvoice line items (billion)

Едно code repo е **efficiency bottleneck**

- merge конфликти
- дълга deployment опашка
- голям и тежък компонент за локална разработка
- сложен upgrade на технологии

фаза 2: Микросървиси

Избор на технологичен stack

Миграция от монолита

Оркестрация и deployment

Основни въпроси

Избор на tech stack

Какви **технологии**?

Еднакви за всички или по **преценка на екипа**?

Да продължим ли с **Groovy**?

Миграция от монолита

Как ще местим функционалност в микросървиси?

Инвестиране в миграция (**tech debt**) срещу инвестиране в **нови features**?

Оркестрация и deployment

Нови **инструменти**?

Нови **процеси** на deployment?

Избор на tech stack

Съображения

Единен tech stack (template) за всички екипи

По-лесна поддръжка на **production**

Няма причина да сменяме **Groovy**

Има причина да сменим **Grails**

Изисквания към микросървисите

REST + Message Queue интерфейси

Собствена база данни

Data ownership

Популярни технологии

Избор на tech stack

Jersey

RESTful Web Services in Java.

APACHE ActiveMQ

spring

cassandra

MySQL

Миграция от монолит

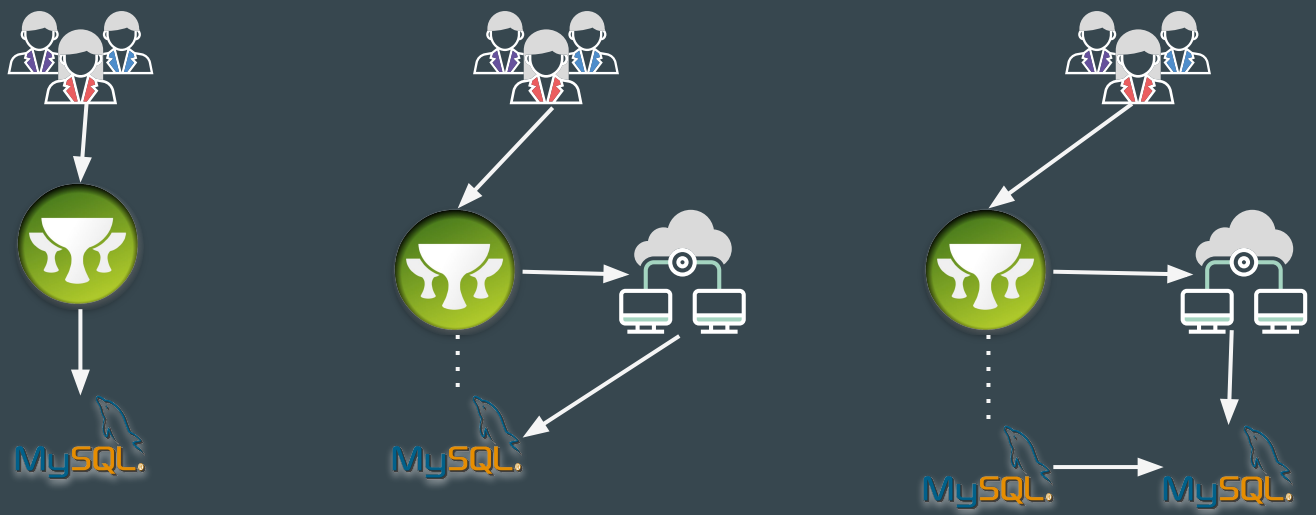
Съображения

Баланс между суха миграция и разработка на нови features

Policy: всеки нов фийчър е в микросървиси

Policy: без нови таблици в базата на монолита

Миграция от монолит - механизъм



Оркестрация и deployment

Нови технологии

Kubernetes container-orchestration

Google Cloud Platform

DevOps екип

Развитие на deployment **автоматизация** (модуляризиране с Jenkins pipelines)

Процеси на работа

Улеснена локална разработка

Продължаващ фокус на автоматизирано тестване

Разпускане на automation екипа

Фаза 2

Съвкупност от микросървиси (монолит все още е жив)

База данни за всеки микросървис (mysql and/or nosql)

Google cloud, kubernetes оркестрация

няколко dev екипа на няколко локации

automation е отговорност на всеки екип

много code repositories

Agile **процес** с **frequent delivery**

продукт и roadmap, управлявани от **експерти**

Труден upgrade на микросървисите

Технологиите **остаряват бързо**
Custom tech stack, custom upgrade

- ➔ бъг в cassandra изисква upgrade до следваща major version
- ➔ cassandra driver в микросървиса е несъвместим със spring версията
- ➔ следващата spring версия работи с по-нова java/groovy

Натрупване и честа работа с **technical debt**

- постоянен цикъл на upgrade (no product value)
- фрагментирани dev/prod среди, трудни за troubleshoot
- склонност към частични ъпгрейди/кръпки -> допълнителна фрагментация

фаза 3: Стандартни микросървиси

Избор на стандартен stack

Миграция от стария stack

Предизвикателства

Избор на стандартен stack	Миграция от стария stack	Оркестрация и deployment
Кой **stack**? Да продължим ли с **Groovy**?	Инвестиране в миграция (**tech debt**) срещу инвестиране в **нови features**?	(Unrelated) Продължаване на **подобрения и оптимизации** на deployment процеса

Избор на tech stack

Съображения

Industry standard tech stack

Няма причина да сменяме **Groovy**

Модерен stack за микросървиси

Стандартен и модулярен

Нови версии и миграция

Миграция от стария stack

Технологично

Няма панацея. Пренаписване.

Не е от 0. Пренасяне и адаптиране на кода към новата структура.

Миграция vs нови features

Няма панацея. Баланс.

Policy: всеки нов микросървис е spring boot.

Пилотен екип мигрира и документира.

Промъкване на migration effort към нови features.

Фаза 3

Съвкупност от **няколко вида** микросървиси (монолит все още е жив)

База данни за всеки микросървис (mysql and/or nosql)

Google cloud, kubernetes оркестрация

итеративни подобрения

няколко dev екипа на няколко локации

Agile **процес** с **frequent delivery**

automation е отговорност на всеки екип

продукт и roadmap, управлявани от **експерти**

много code repositories

Много компоненти - труден development

Локална dev среда **не е практична**
Твърде много компоненти за подкарване на цялостен flow

➔ много overhead да се ъпдейтват локално до последната версия
➔ прекален товар за dev машина

Нужда от цялостен **CI/CD**

- много компоненти, конфигурции и среди - всяка ръчна стъпка е рискова
- нужда от различен **deployment cadence** за различни проекти / екипи

Какво следва?

~~Революционни~~ Еволюционни промени

CI/CD вече е добър

Напълно автоматизиран **regression**

Ежечасови **stability tests** на всички среди, включително **production**

Автоматичен **deployment**

Следва: до production

Dev процес вече е добър

Екипите са **автономни** и **самодостатъчни**

- всички необходими роли
- автономност на процеси*

Следва: deployment автономност

Среда за разработка

Локална разработка чрез **автоматизирани тестове**

E2E разработка чрез **personal dev environments**

Следва: подобрения по PDEs

В обобщение

Следете за процесите

Начинът на **разработка** и deployment говори за **организационната структура**

Пътят на развитие е сравнително **стандартен.**

А защо не прескочим?

There is a difference between knowing the path and walking the path

- **Time to value**
- Agile: **итеративен** подход, нисък риск
- Крайната цел е **променлива**

Благодаря!