



DINO ESPOSITO

CTO, Clonnet  
Italy

SASHA GOLDSHTEIN

CTO, Sela Group  
Israel

DYLAN BEATTIE

CTO, Skills Matter  
UKА ТАКОЖ  
**WORKSHOP**  
by Dino Esposito

ДЕТАЛІ НА САЙТІ &gt;&gt;

## КОНФЕРЕНЦІЯ З .NET РОЗРОБКИ #1 У УКРАЇНІ

26-27 ЖОВТНЯ  
КИЇВ ■ 2018

СТАТТІ · 9 ноября 2016, 12:46 · 9367



Орхан Гасымов, Software Engineer в AppsFlyer

## Реактивные приложения на Java с Akka

Я работаю программистом уже более 13 лет: занимаюсь высоконагруженными и распределенными системами, рассматриваю и оцениваю разные подходы и решения.

В данный момент я выделяю два типа систем, которые определяют итоговый стек технологий, с которым следует иметь дело:

- системы, которые поставляют данные конечным пользователям;
- системы, предназначенные для вычислений.

Третьим типом может быть система, которая выполняет обе функции. Однако такие системы обычно строятся из подсистем, которые относятся к первым двум типам.

Я хочу поделиться с вами информацией об Akka и модели актеров. Если вы уже используете Akka в своих проектах, я верю, вы уже знаете все то, о чем я буду говорить здесь. Если же вы слышали

### Не пропустите



13 июня, Харьков  
**Tech Meetup:**  
**OpenStack, Java,**  
**Scheduler**



14 июня, Харьков  
**Съесть собаку #13:**  
**высоконагруженные**  
**системы**



15 июня, Киев  
**Интенсивная**  
**подготовка**  
**и проведение**  
**сертификации**

об Акка и не уверены, подходит ли вам этот инструмент, я поделюсь с вами достаточной информацией для того, чтобы вы сумели определиться.

## Многопоточность и масштабирование

С развитием технологий развивается и модель программирования. Ее эволюция началась с больших монолитных задач, которые на самом деле являлись набором инструкций для последовательного выполнения. Эволюционируя, эти задачи становились все больше и больше. Модульный подход позволил разделить большие задачи на мелкие и выполнять их последовательно, переключая контекст с одной задачи на другую. По дальнейшей эволюции стало ясно, что некоторые задачи могут выполняться независимо от других. Это стало основой для многопоточного программирования:

### тестировщиков ISTQB



19 – 20 июня, Киев  
**Курс Certified Scrum Product Owner**  
от ScrumAlliance  
на русском языке



19 – 20 июня, Киев  
**Тренинг**  
„Professional Scrum Product Owner со Славой Москаленко”



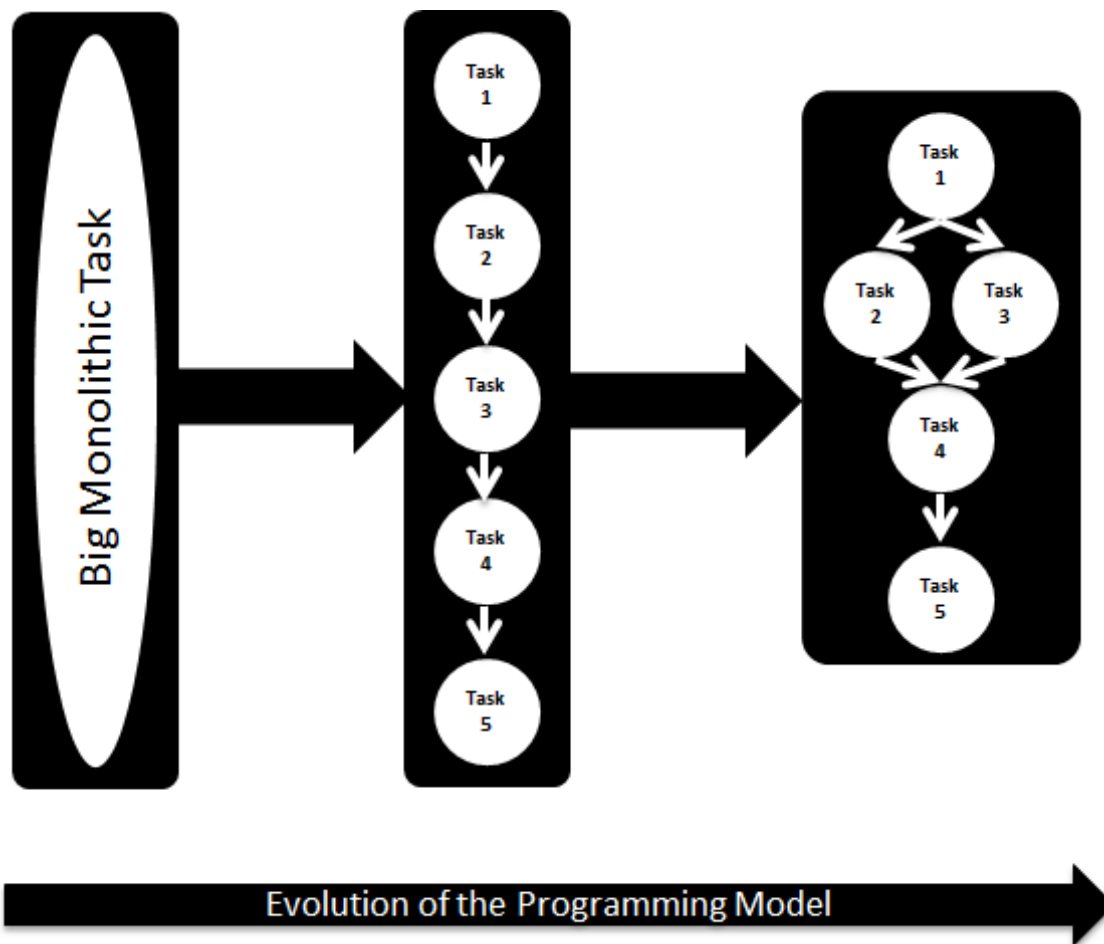
21 – 22 июня, Киев  
**Kanban Management Professional (KMP II)**  
by LeanKanban University



21 июня, Киев  
**GlobalLogic Embedded TechTalk #5: Building a Smart City**



21 – 22 июня, Киев  
**Курс Certified ScrumMaster**  
от ScrumAlliance



Однако даже с многопоточными программами мы на самом деле имеем на руках большое монолитное приложение, которому нужно все больше и больше ресурсов для дальнейшей эволюции. Для обеспечения достаточного количества доступных ресурсов мы дошли до больших серверных станций. При условии доступности ресурсов многопоточное программирование разрешило выполнять задачи параллельно.

Возможность выполнять задачи параллельно бросила вызов, связанный с масштабированием приложений. Масштабирование бывает вертикальное или горизонтальное.



29 июня, Online  
Курс ISTQB  
Advanced level Test  
analyst



3 июля, Киев  
Курс „Software  
Testing Foundations”

### 🔥 Горячие вакансии

[Android developer](#) в MEGOGO,  
Киев

[Java developer](#) в Infrascale, Киев

[Senior Python/C++ Developer \(World  
of Tanks Kyiv\)](#) в Wargaming, Киев

[Senior DBA \(MS SQL\)](#) в Itera, Киев

[Senior Full Stack PHP Developer](#)  
в SomProduct, \$1500–3000, Львов

[Senior Python Engineer](#) в  
 Ring Ukraine, Киев

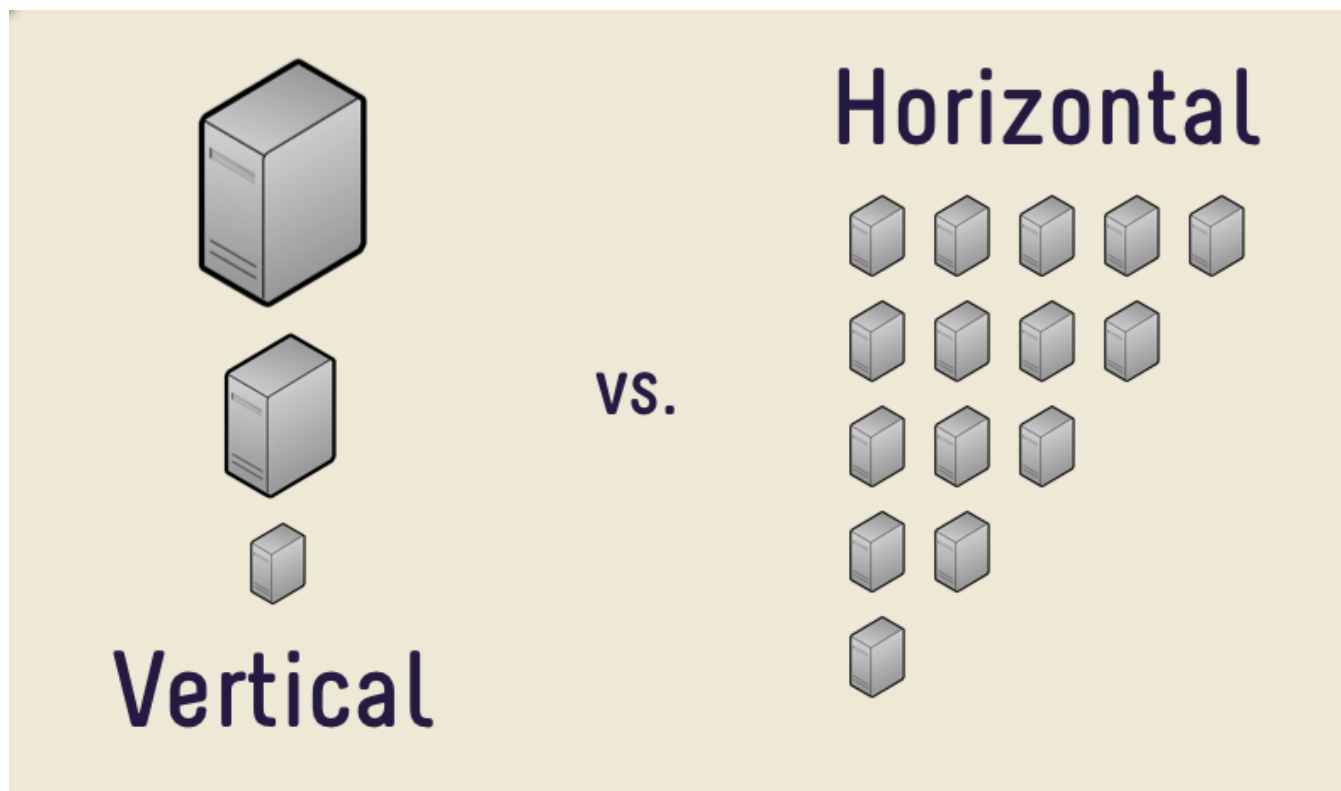
[QA Engineer](#) в Softwarium, \$800–  
1600, Киев

[Automation Lead](#) в Playtika, Киев

[Middle Developer C++](#) в  
 G5 Entertainment, Харьков,  
Львов

*При вертикальном масштабировании* вы развиваете сервер, добавляя больше вычислительной мощности. С каждым обновлением следующая опция обходится все дороже и дороже. Также есть большой риск отказа системы, так как всю работу делает один большой сервер. Вы можете иметь запасной сервер, зеркальный и т.п. Однако это не меняет модель развития; вы все еще в плоскости вертикального масштабирования. Этот вызов перенес нас в плоскость горизонтального масштабирования.

*При горизонтальном масштабировании* вы добавляете еще один сервер вместо обновления уже существующего. Это позволяет покупать недорогое оборудование в большем количестве — в сумме получается дешевле, чем покупка и обновление больших серверов. Вдобавок вы получаете отказоустойчивый кластер. Даже если один сервер из десяти откажет, будет доступно достаточно ресурсов для полноценной работы системы. Обновление такого кластера означает добавление еще одного недорогого сервера (узла).



Резюмировать обе модели масштабирования можно так:

Вертикальное масштабирование	Горизонтальное масштабирование
Опции дороже с каждым обновлением	По сравнению с вертикальным масштабированием, опции обходятся дешевле
Большой риск отказа системы при отказе оборудования	Отказ нескольких узлов не приводит к отказу всей системы (если система спроектирована правильно)
Ограниченные опции обновлений в будущем	Обновлять систему легко
Один узел для всего	Параллельные узлы разрешают параллельные

Давайте честно, даже масштабируя ресурсы горизонтально, будете ли вы все еще выполнять то же самое приложение на более слабых серверах за лoad-балансером? Думаю, нет. Мы все еще можем иметь два или три больших сервера за лoad-балансером. Зачем нам зоопарк маленьких серверов? Эти вопросы правильно поставлены, если вы все еще разрабатываете приложения объектно-ориентированно.

## Реактивные приложения

Если честно, услышав впервые термин «реактивные приложения», я был удивлен. Я мог представить реактивные двигатели, самолеты и даже машины, но не программы. Почитав немного, я понял, что имеется в виду: речь идет о распределенных системах, которые являются большими по своей природе. Большие системы нужно поддерживать некоторыми договоренностями и правилами, чтобы они оставались актуальными и расширяемыми спустя годы.

Реактивные приложения поддерживаются Реактивным манифестом. Реактивный манифест — это свод правил и договоренностей, которые могут удержать систему на плаву, если имплементированы правильно. На самом деле в нем нет ничего нового, многие из нас уже знакомы с этими подходами. Увы, иногда правила нарушаются желанием быстро имплементировать какую-то «фичу» или какими-то другими требованиями. Я верю в то, что Реактивный манифест был собран не для того, чтобы нас научить, а для того, чтобы напомнить «почему?», «как?» и «что?» мы делаем. Прочитать полный текст Реактивного манифеста можно [на официальном сайте](#). В момент написания статьи актуальной была версия 2.0. Также доступен [перевод на русский](#) (правда, старой версии).

Основная идея реактивности заключается в том, что системы старого типа в основном развертывались на компьютерах, обрабатывали гигабайты информации, и был допустим отклик системы в несколько секунд. Современные системы развертываются на чем угодно — от мобильных устройств до облачных кластеров — и обрабатывают терабайты информации, а допустимое время отклика опустилось до десятков миллисекунд.

Чтобы называться реактивными, приложения должны быть отзывчивыми, упругими, эластичными и общаться асинхронным обменом сообщений:

— *Отзывчивость* — это быстрая реакция на заданный запрос. То есть приложение должно всегда и всегда обрабатывать и отвечать на запросы быстро, даже при больших нагрузках. Это также значит, что если где-то произошел сбой, система должна быстро обработать ситуацию и дать соответствующий ответ.

— *Упругость* — это означает, что приложение должно оставаться отзывчивым даже при возникновении отказа в некоторой части системы. Для этого отказ одного компонента обрабатывается другим, который в свою очередь тоже может делегировать обработку своего отказа другому компоненту. Таким образом, отказ одного компонента в системе не приводит к отказу всей системы.

— *Эластичность* — говорит о том, что такие приложения могут масштабироваться по надобности, что приводит к архитектуре, в которой не остается узких мест и повышается общая отзывчивость системы целиком.

— *Асинхронный обмен сообщениями для коммуникации между компонентами системы* — это уменьшает зависимость компонентов системы друг от друга, позволяет масштабировать их независимо и разворачивать в разных местах.

В итоге большие системы строятся из меньших систем, поддерживая их реактивные свойства в себе. Такие приложения и называются реактивными.

## Модель актеров

Если присмотреться, все вышеперечисленные свойства уже соблюдаются в Java EE. Хотя, иногда мы тратим очень много времени для достижения этих свойств и еще больше для поддержки итогового приложения. Конечно, если приложение уровня Enterprise будет развернуто на паре Enterprise серверов приложений (EAS) в кластере за лод-балансером, я однозначно выберу Java

ЕЕ и воспользуюсь поставляемым в коробке функционалом. Если же у меня будет десяток серверов, я скорее подумаю, сколько ресурсов освободится, если избавиться от Enterprise серверов приложений. Только подумайте, сколько памяти, процессорного времени и денег можно сэкономить.

Есть модели программирования, которые извлекают пользу из кластера серверов, превращая их в один суперкомпьютер. Одна из таких моделей программирования — модель актеров, о которой мы и будем говорить.

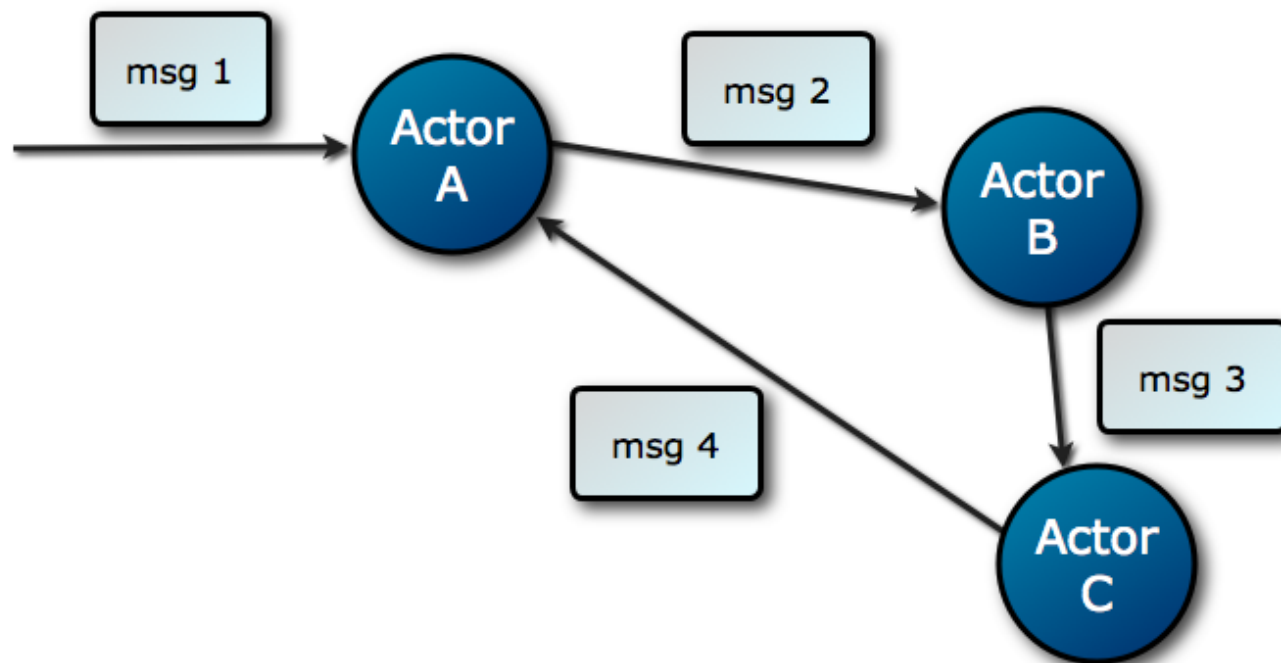
Актер — это вычислительная единица. В ответ на сообщение, которое он получил, актер может:

- отправить определенное количество сообщений другим актерам;
- создать определенное количество новых актеров;
- определить поведение для обработки следующего сообщения, которое он получит.

Нет определенной последовательности вышеперечисленных действий, они могут быть выполнены параллельно. Философия модели актеров гласит, что «все вокруг актеры». Это похоже на философию «все вокруг объекты». При этом объектно-ориентированные программы обычно выполняются секвентально, когда актеры работают параллельно. Конечно, вы можете использовать потоки, чтобы добавить конкурентность в объектно-ориентированные программы. Обратите внимание, что в 64-битной системе поток Java занимает 1 мегабайт памяти, когда актер занимает до 300 байтов. Потоки будут ограничены в количестве, и придется пользоваться спецификой объектно-ориентированного подхода. Используя модель актеров, вы думаете об актерам вместо объектов и потоков. Можно иметь намного больше актеров, чем потоков.

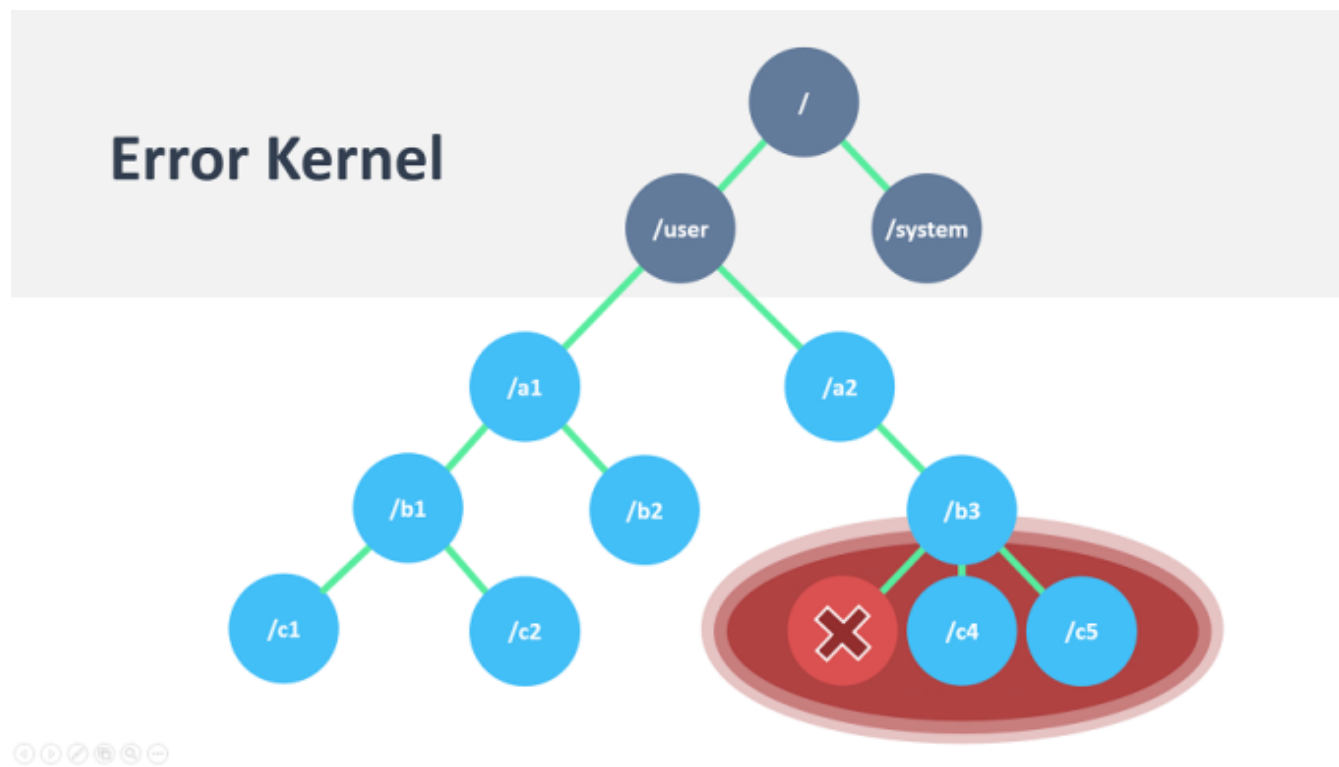
Актеры инкапсулируют состояние и поведение и общаются исключительно посредством обмена сообщениями. Сообщения, полученные актером, ложатся в его почтовый ящик. Актеров лучше рассматривать как людей. Моделируя решение с моделью актеров, представьте группу людей, которым вы назначаете подзадачи. Подзадачи — это результат деления одной большой задачи на мелкие куски, которые делегируются актерам для выполнения. Вы можете назначить эти задачи актерам, организовав их в структуру, словно членов команды в экономической организации. Также подумайте о том, как эскалировать отказы на каждом уровне вашей организационной структуры.





Каждый актер может создавать дочерних актеров и быть их супервизором, словно менеджер команды. В практике лучше будет разделить задачу актера на мелкие управляемые подзадачи и разрешить подчиненным выполнить их в параллели. Если актер имеет важное состояние, которое нельзя потерять (например, если получим Exception), тогда всю задачу целиком лучше отдать на выполнение подчиненным и просто следить за процессом выполнения. Поэтому актеры никогда не приходят одни. Они приходят в системах и образуют иерархии (как файлы в UNIX). Системы актеров также могут формировать кластеры. Если представить актера как звезду, тогда система актеров – это кластер звезд. Кластеры систем актеров – это кластеры кластеров звезд. Таким образом, актеры создают свою вселенную. Это вселенная после будет развернута на предоставленном оборудовании и будет выполнять запрограммированную логику.

В мире актеров следуют принципу «Let It Crash» («позволь ему отказать») и так называемому шаблону Error Kernel Pattern (шаблон ядра ошибок или, точнее, обработки ошибок). Этот шаблон ведет нас к системе актеров, где супервизор каждого уровня иерархии в ответе за локальные ошибки. В данном контексте, ошибки — это exceptions, возникшие у подчиненных актеров, то есть сами актеры не должны их обрабатывать. В свою очередь, супервизор при возникновении ошибки может отловить ее и, как правило, продолжить, перезагрузить, остановить работу актера или же эскалировать exception своему супервизору, сообщая о падении всей своей иерархии подчинения. Ядро ошибок самой системы актеров формируется из системных актеров, которые отвечают за жизненный цикл всей системы. При этом каждый уровень иерархии в организационной структуре является локальным ядром ошибок, отвечающим за стратегию обработки ошибок, которая определит, как нужно реагировать на определенные ошибки дочерних актеров.



## Работа с Akka

Есть много имплементаций модели актеров, и одна из них нацелена исключительно на извлечение пользы из кластера серверов. Это Akka — она имплементирует модель актеров и поддерживает Реактивный манифест, разрешая развертку приложений на кластере с минимальными изменениями в коде или даже вообще без них.

Akka — это самостоятельный инструмент. Ему не нужен сервер приложений, достаточно JVM и Java SE. С помощью Akka можно объединить несколько JVM в кластер. Akka предлагает модель актеров вместо объектно-ориентированной, которая считается распараллеленной по умолчанию.

С Akka мы получаем:

- параллелизм вместо конкурентности;
- асинхронное поведение по умолчанию;
- неблокируемые вызовы;
- выполнение без Deadlock, Starvation, Live-lock и Race Condition;
- разработка в однопоточной среде, когда задачи выполняются в параллели.

Разработка с моделью актеров в Akka не сложная. На момент написания статьи актуальной версией Akka была 2.4. Для начала подключите все необходимые зависимости в проект. Всю необходимую информацию можно получить на сайте [akka.io](http://akka.io). Также есть исчерпывающая документация для Java разработчиков на [akka.io/docs](http://akka.io/docs).

Для быстрого старта нужно знать о нескольких классах из пакета akka.actor. Это:

- ActorSystem — класс имплементирующий систему актеров;
- UntypedActor — класс, который нужно унаследовать для создания класса актера и переопределить метод onReceive для обработки входящих сообщений данным актером;
- ActorRef — класс, инкапсулирующий ссылку на актер. Он же используется для отправки сообщений актеру.

Используя эти классы:

- вызовите `ActorSystem.create()` для создания системы актеров и получения объекта `ActorSystem`;
- вызовите `ActorSystem.actorOf()` для создания экземпляра актера и получения его `ActorRef`;
- используйте `ActorRef.tell()` для отправки сообщений актеру.

Это все! На самом деле, есть достаточно много нюансов, но, для первых шагов этого достаточно.

Остановимся на минуточку. Что, если мы разрабатываем локальное приложение? Помните `ExecutorService` или даже `Multithreading API`? Представьте систему, использующую все это. Насколько оно эффективно? Действительно, оно может эффективно использовать доступные ресурсы. Скажем, мы выучили Akka API и начали программировать актеров. Хорошо, что дальше? Мы получили те же результаты, как если бы использовать стандартные механизмы многопоточности в Java. Так ли оно? Да и нет. Да, потому что результаты могут быть похожи, и нет, потому что с Akka мы получаем похожие результаты, программируя в однопоточной среде, обходя все сложности разработки многопоточных приложений. Вы только описываете актеров и логику обработки сообщений, которые они получают. Кстати, сообщением может быть любой объект. Единственное правило, которое нужно помнить: сообщения должны быть `immutable` объектами. Потому что с Akka мы пишем код в однопоточной среде, когда система актеров выполняется в многопоточной среде.

Самая интересная часть работы с Akka наступит после того, как вы разработаете первое приложение. Система актеров может быть развернута как самостоятельное приложение. Актеры также могут обмениваться сообщениями по сети. Это позволяет организовать совместную работу отдельных систем актеров. И наконец, вы можете развернуть кластер из нескольких машин. Можно развернуть единую систему актеров на кластере машин, связанных в одну сеть. Таким образом, вы получаете настоящее параллельное выполнение актеров.

Есть много опций для лoad-балансинга и масштабирования вашего приложения. Вы можете заменить имплементации почтовых ящиков и определить свои правила приоритезации сообщений в очереди. Можно использовать маршрутизаторы, которые организуют пулы и группы актеров вместо одного актера, который обрабатывает сообщения. Можно заменить диспетчеры, которые определяют, как доступные потоки используются актерами и даже больше.

Дополнительно ко всему перечисленному, Akka предоставляет Functional Futures, имплементируя Java обертку над фьючерами из Scala. С помощью этого API вы даже можете реализовать симуляцию map-reduce. Доступно еще больше с таким функционалом и модулями, как Agents, Streaming API, HTTP стэк, Camel и другие.

## Резюме

С Akka вы можете получить полноценный инструментарий, который позволяет разрабатывать высокопроизводительные отказоустойчивые распределенные приложения, которые разрешают рассматривать кластеры машин как единую систему.

Конечно, доступно много других фреймворков и инструментов, позволяющих разработчикам достичь похожих результатов. На мой взгляд, итоговое решение зависит от ответа на один основной вопрос: что будет делать ваше приложение? Я выбрал бы Akka как минимум для параллельных вычислений с небольшими затратами ресурсов. Akka умеет намного больше, и для меня это тот минимум, который определяет, где Akka может пригодиться.

Буду рад, если эта статья поможет определить, интересна ли вам Akka для дальнейшего изучения. Желаю удачи!

Темы: [Java](#)



### Свежее

9 июня, 10:00 · [Ссылки](#)

[.NET дайджест #23: улучшения производительности в .NET Core 2.1, принципы проектирования агрегатов](#)

### Популярное за месяц ▾

29 мая · 👁 38975

[Junior дайджест: курси, стажування, вакансії. Червень'18](#)

29 мая · 👁 29740

8 июня, 13:17 · [Статьи](#)

[Реактивный подход к валидации полей ввода на Android](#)

8 июня, 10:00 · [Статьи](#)

[Разработка для отдела Business Intelligence: автоматизируем ad hoc задачи по выгрузкам из БД](#)

7 июня, 13:43 · [Статьи](#)

[DOU Проектор: CleverStaff — сервис для автоматизации рекрутинга](#)

7 июня, 10:00 · [Статьи](#)

[Жизнь и приключения программиста в Болгарии](#)

[Все материалы](#)

[Прислать статью](#)

[Рейтинг вишів DOU 2018: Могилянка знову в лідерах. Львівська політехніка наприкінці списку](#)

10 мая ·  28655

[Как найти первую работу в IT: план действий для начинающих](#)

22 мая ·  24073

[Fail review: неудачные собеседования vol.2](#)

11 мая ·  21813

[Країна велосипедів і щасливих людей.](#)

[Українська програмістка про життя і роботу в Нідерландах](#)



**Aliaksandr Valialkin** програмист в  [Adtelligent](#)

13.11.2016 00:18

Актёры отлично реализуются в Go с помощью [goroutines & channels](#). Только большинство практических задач проще и эффективнее решаются без актёров.

[Ответить](#)



[Поддержать](#)



**Олександр Шпак** IT-nigga

13.11.2016 10:31

Простіше то воно може й простіше. Тільки от є проблема, коли проста система виростає з дитячих штанців, починається ад.

[Ответить](#)



[Поддержать](#)



**Aliaksandr Valialkin** программист в  [Adtelligent](#)

13.11.2016 18:14

У нас на go-проектах без акторов пока все просто и понятно. Доросли до 1.5 миллиона запросов в секунду от 7 миллионов одновременно подключенных http-клиентов. Что ж, будем готовиться к аду :)

[Ответить](#)



[Поддержать](#)



**Дмитрий Думанский** Co-Founder в  [Blynk](#)

13.11.2016 20:56

1.5 миллиона запросов в секунду

Это впечатляет. У унылого фейсбук месенджера лишь 600к рек-сек.

[Ответить](#)



[Поддержать](#)



**Sergii Marushchenko** Senior Software Developer в  [InterLogic](#)

14.11.2016 00:08

Трекинг какой-то, наверное.

[Ответить](#)



[Поддержать](#)



**Aliaksandr Valialkin** программист в  [Adtelligent](#)

14.11.2016 12:53

Да, сбор статистики

[Ответить](#)



[Поддержать](#)



**Олександр Шпак** IT-nigga

13.11.2016 22:00

А по кількості коду до чого доросли?

[Ответить](#)



[Поддержать](#)



**Aliaksandr Valialkin** программист в [Adtelligent](#)

14.11.2016 12:59

1Мб нашего кода на go, состоящего из 45K строчек:

```
$ find src/ -name *.go | xargs cat | wc
45188 126307 1128623
```

Сторонний код в расчет не берется — он лежит в отдельной папке vendor:

```
find vendor/ -name *.go | xargs cat | wc
981888 3706581 28710948
```

[Ответить](#)



[Поддержать](#)



**Олександр Шпак** IT-nigga

14.11.2016 13:45

Вважаєте це великим проектом? :) Разів в 100 більше хоча б було...

[Ответить](#)



[Поддержать](#)



**Mykola Gurov** Жирный тролль

14.11.2016 18:24

на вашей галере по LOC count, небось, премию считают?

[Ответить](#)



[Поддержать](#)



**Олександр Шпак** IT-nigga

14.11.2016 18:28

Досвід Майкрософта показує, що це абсолютно неефективно...

[Ответить](#)



[Поддержать](#)



**Aliaksandr Valialkin** программист в [Adtelligent](#)

15.11.2016 00:29



К сожалению, наш проект написан не на java или scala. Иначе в нем бы было раз в 100 больше кода при аналогичной функциональности :)

[Ответить](#)



[Поддержать](#)



**Dmitriy Onykyenko** dev

15.11.2016 08:06

Сомневаюсь.

[Ответить](#)



[Поддержать](#)



**Oleksandr Olgashko**

15.11.2016 22:54


До тех пор, пока это дело влазит на одну машину. Кроме того, как только появляется необходимость в супервайзинге, или в сложной логике по типу FSM, или хотя бы в безболезненной обработке мутабельности — akka становится более предпочтительным вариантом, а голанг обрывается велосипедами.

[Ответить](#)



[Поддержать](#)



**Igor Azarny** Team Leader в  [Luxoft](#)

11.11.2016 11:40

Для Java лучше использовать [vertx.io](#) по многим причинам:

1. построенное на базе netty & hazelcast дает отличную производительность
2. отличное масштабируется
3. ha из коробки
4. встроенные метрики
5. поддержка http/tcp/websockets/mqtt из коробки
6. не надо притаскивать скалу (проблемы со скалой не будем перепечислять)
7. предлагает не только асинхронную парадигму обработки, но и синхронную, что в некоторых случаях облегчает жизнь.

8. перевірено в проде

9. 10. 11. 12 ....

[Отповісти](#)



[Підтримати](#)



**Олександр Шпак** IT-nigga

11.11.2016 12:07

Мінуси вертекса теж не забувайте перераховувати...

[Отповісти](#)



[Підтримати](#)



**Дмитрий Думанский** Co-Founder в [B Blynk](#)

12.11.2016 01:19

Про які мінуси мова?

[Отповісти](#)



[Підтримати](#)



**Олександр Шпак** IT-nigga

12.11.2016 16:20

1. Через те, що використовується патерн обсерверів, код швидко починає бути схожим на спагетті. В складних випадках взагалі читати неможливо.
2. Асинхронність важко дається людям, які звикли думати синхронно. Поріг входження підвищується.
3. Можлива швидка деградація через неправильне використання.

[Отповісти](#)



[Підтримати](#)



**Дмитрий Думанский** Co-Founder в [B Blynk](#)

12.11.2016 16:42

1. Лише частково так. З досвідом ця проблеми зникає
2. Трю.
3. Як і з любим іншим інструментом.

Ну тобто то вже проблема застосування, людського фактору і точно не привід відкидати технологію через це.

[Ответить](#)



[Поддержать](#)



**Олександр Шпак** IT-nigga

12.11.2016 20:23

Я не казав, що треба відкидати :)

[Ответить](#)



[Поддержать](#)



**Igor Azarny** Team Leader в [Luxoft](#)

13.11.2016 13:29

1. Так це можливо
2. Call back hell лікується [vertx.io/docs/vertx-sync/java](https://vertx.io/docs/vertx-sync/java) , тому vertx підтримує асинхронні та синхронні парадігми
3. Це можливо з будь якою технологією

[Ответить](#)



[Поддержать](#)



**Andrey Dobrov** Senior Software Programmer

11.11.2016 15:38

1. 2. Ой далеко не всегда плюс.

Почему? Читай 8.

[Ответить](#)



[Поддержать](#)



**Дмитрий Думанский** Co-Founder в [Blynk](#)

12.11.2016 01:27

Можете рассказать подробнее?

[Ответить](#)



[Поддержать](#)



**Andrey Dobrov** Senior Software Programmer

12.11.2016 07:02

Используем веб-балансер, три мощные железяки и грид на 1000 жвм. Хезлкаст для кэширования и бегают он тоже на гриде.

Все это получает миллионы запросов-тасок на пересчет рисков.

1 и 2 на гриде часто падает (болезни гридов и разных клаудов) Не вина это хэзлкаста конечно, но клиент получает таймаут а мы фу-фу-фу :)

Да и вся эта асинхронность в магичесаих фреймворках ограничена возможностями железяки. Когда пиковые нагрузки — все эти волшебные экзекьюторы и тред пулы, не всегда оптимально шарят сри — а мы опять получаем таймауты и фу-фу-фу :)

Как проверялось на пункте 8 написал выше.

[Ответить](#)



[Поддержать](#)



**Дмитрий Думанский** Co-Founder в [Blynk](#)

12.11.2016 12:40

Спасибо.

1 и 2 на гриде часто падает (болезни гридов и разных клаудов)  
Не вина это хэзлкаста конечно, но клиент получает таймаут  
а мы фу-фу-фу :)

Падает железо или JVM?

[Ответить](#)



[Поддержать](#)



**Andrey Dobrov** Senior Software Programmer

12.11.2016 15:40

Железо или сеть.

[Ответить](#)



[Поддержать](#)



**Дмитрий Думанский** Co-Founder в [Blynk](#)

12.11.2016 16:02

Железо или сеть.

А где хоститесь? Ну и как отказ харда связан с netty и hazelcast?  
Это скорее уже проблема вашего лoad балансера.

Можно уточнить на всякий — что Вы подразумеваете под гридом?

[Ответить](#)



[Поддержать](#)



**Andrey Dobrov** Senior Software Programmer

12.11.2016 16:29

- 1) У нас свои датацентры.
- 2) В том то и дело, что у нас претензий к хазелкасту почти нет. Мой пост о том, что в продакшине как наш, использование распределенного кэша не является решением всех проблем (равно как и использование других фреймворков) просто случаются проблемы другого уровня, которые в комбинации с например хезлкастом дают интересные результаты.
- 3) В нашем случае это виртуальные машины под управлением IBM Grid.

[Ответить](#)



[Поддержать](#)



**Дмитрий Думанский** Co-Founder в

[Blynk](#)

12.11.2016 16:38

Мой пост о том, что в продакшине как наш, использование распределенного кэша не является решением всех проблем

Ну так... Софт для решения бизнес задачи это одно, а кластер из 1000 машин это уже другое :). Конечно что хацелькаст не способен решить эти проблемы. Это уже задача проектирования кластера. Вообще было бы интересно почитать про ваш проект. Статью не планируете?

Я лично юзаю netty, но от хацелькаста отказался, так как код внутри там совсем «не очень» да и хацель, который ничего не делает отжирает 30мб RAM (я разворачиваю сервера на low-end железе, для меня 30 МБ это возможность обслужить 1000 юзеров).

[Ответить](#)  [Поддержать](#)



**Mykola Gurov** Жирный тролль

14.11.2016 00:38

Тут кстати [youtu.be/7IbdWcdIYOI](https://youtu.be/7IbdWcdIYOI) жаловались на хазелькаст из vert.x и на 33 минуте советуют вместо его дефолтного Zookeeper юзать. Боян не мой, просто как раз услышал...

[Ответить](#)  [Поддержать](#)



**Дмитрий Думанский** Co-Founder в  [Blynk](#)


12.11.2016 01:26

Для Java лучше использовать [vertx.io](https://vertx.io) по многим причинам:

Используете в продакшене? Какой лoad? Какая инфраструктура?

[Ответить](#)  [Поддержать](#)



**Igor Azarny** Team Leader в  [Luxoft](#)

13.11.2016 13:23

Gambling, 10k users, docker + k8s on bare metal. PoC на z480 hp давал такую картинку [s21.postimg.org/n1xhfum3b/Untitled.png](https://s21.postimg.org/n1xhfum3b/Untitled.png), что коррелируется с [www.techempower.com/...=data-r12&hw=ph&test=json](http://www.techempower.com/...=data-r12&hw=ph&test=json)

Ответить



Поддержать



**Infa tum** Unity3d Dev

10.11.2016 14:44

Порт Akka под .NET: [getakka.net](http://getakka.net)

И bootcamp для тех, кто хочет попрактиковаться: [github.com/petabridge/akka-bootcamp](https://github.com/petabridge/akka-bootcamp)

Ответить



Поддержать



**Igor Prots** Senior Erlang developer в [S SoftServe](#)

10.11.2016 12:41

Посмотрите на Erlang, там все это имплементировано из коробки. Порог входа не большой, попробовать модель акторов можно прям в консолюке.

Ответить



Поддержать



**Дмитрий Думанский** Co-Founder в [B Blynk](#)

12.11.2016 01:28

А разрабов где искать, если проект выстрелит?

Ответить



Поддержать



**Igor Azarny** Team Leader в [Luxoft](#)

13.11.2016 13:33

Скоро на всех новостных сайтах — «Требуется PM со знанием некромантии и Erlang».

Ответить



Поддержать



**Igor Prots** Senior Erlang developer в [S SoftServe](#)

13.11.2016 20:52

Зачем ПМу знать эрланг? Прочитает лукоморье и хватит с него.

[Ответить](#)



[Поддержать](#)



**Igor Prots** Senior Erlang developer в [SoftServe](#)

13.11.2016 20:49

В ПриватБанке =)

[Ответить](#)



[Поддержать](#)



**Alexandr Sova** Developer

14.11.2016 23:47

Посмотрите на Erlang, там все это имплементировано из коробки  
оттуда всё позаимствовано. Что создатели Akka, собственно, и не скрывают.

[Ответить](#)



[Поддержать](#)



**Олег Кардаш** Java Developer

10.11.2016 11:06

Дякую! Хороша стаття! Зараз якраз активно погрузився в Akka, так як проект на якому працюю досить тісно побудований на Akka...

[Ответить](#)



[Поддержать](#)



**Yegor Chumakov** Team Lead

10.11.2016 03:25

Где код?

[Ответить](#)



[Поддержать](#)



**Włodzimierz Rożkow** Software Development Engineer

10.11.2016 01:41

Я три роки тому питаю в людей хто що [dou.ua/forums/topic/8237](http://dou.ua/forums/topic/8237) використовує і реальні кейси, відписалось півтори людини.



З того часу про акку я не чув ніде крім, здається, лінкедіну, та Gatling — перформенс тулзи (яка правда працює на локалхості :)) яку ми використовували для тестування свого сервісу.

А горизонтально масштабуватися чудово можна і з якимось Spring Boot, ну і асинхронні сервлети воно вже давним-давно вміє без зайвих клопот робити.

[Ответить](#)  [Поддержать](#)



**Дмитрий Думанский** Co-Founder в [Blynk](#)

09.11.2016 20:00

Спасибо конечно, но ситуация с аккой мне напоминает ситуацию с big data —

Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it

Уже не первый раз встречаю статью поверхностного описания акки, но без реального применения. Если Вы в проектах используете акку, пишите высоконагруженные системы, строите сложную распределенную архитектуру — расскажите же о б этом!

Расскажите нам какая задача решалась, как Вы ее решили с помощью акки, какая нагрузка на систему и почему именно акка зарешала в вашем конкретном случае. Это же самый сок, черт побери. Надеюсь, что будет вторая часть.

Теперь по статье.

При горизонтальном масштабировании вы добавляете еще один сервер вместо обновления уже существующего. Это позволяет покупать недорогое оборудование в большем количестве — в сумме получается дешевле, чем покупка и обновление больших серверов.

Очень сильно зависит от ситуации. Вертикальное масштабирование обычно всегда дешевле горизонтального, так как нету дополнительных расходов на менеджмент кластера и подготовку ПО. Основная причина горизонтального масштабирования — физическая ограниченность вертикального роста.

Вдобавок вы получаете отказоустойчивый кластер.

Отказоустойчивый кластер нужно сначала построить. И это время и деньги. + дополнительные люди для мониторинга систем кластера.

Обратите внимание, что в 64-битной системе поток Java занимает 1 мегабайт памяти, когда актер занимает до 300 байтов.

Ну актер не поток. И у них разные задачи. Зачем сравнивать несравнимое?

Потоки будут ограничены в количестве,

И чья это проблема?

[Ответить](#)  [Поддержать](#)



**Alexandr Sova** Developer

09.11.2016 19:50

Вертикальное масштабирование

Забыли упомянуть что это прежде всего утилизация максимума ресурсов доступных системе. Например распределение нагрузки по всем ядрам процессора. Уж слишком часто системы используют только мелкую часть того что им дают админы.

нацелена исключительно на извлечение пользы из кластера серверов

Akka-cluster это модуль который длительное время был экспериментальным и Akka жила себе отлично и всё это время. Как по мне преимущества модели акторов (да, именно акторов — действующих единиц, а не актёров) начинаются когда нужно просто выполнять задачи используя весь доступный ресурс, например если скорость выполнения задачи пропорциональна доступному ресурсу. И наиболее ярко преимущества этой модели выражаются когда ты не знаешь — система будет работать на одном маленьком сервере или на кластере высокопроизводительных серверов. Так что очень далеко не **исключительно**.

Akka предлагает модель акторов вместо объектно-ориентированной

Вот нет же. Это как раз ведь та самая ОО-модель что задумывалась изначально. Та модель когда предполагалось что объекты будут коммуницировать между собой отсылая асинхронные сообщения. То что мы вместо этого получили синхронные сообщения (вызовы методов) — просто исторически сложившиеся обстоятельства.

[Ответить](#)  [Поддержать](#)



**Andrey Dobrov** Senior Software Programmer

09.11.2016 17:09

Оно конечно классно, что вы перевели общее резюме Акка фреймворка на русский ;)  
Хотелось бы деталей. Из личного опыта и все такое (уж если вы затронули такую тему)  
Предпоследний абзац — ну полный абзац ;)

[Ответить](#)  [Поддержать](#)



**Viktoria Muzychko** пишу кодик в  [mimacom Ukraine](#)

09.11.2016 13:01

Спасибо!

[Ответить](#)  [Поддержать](#)



**Орхан Гасымов** Software Engineer в  [AppsFlyer](#)

09.11.2016 14:00

Не за что :)

[Ответить](#)  [Поддержать](#)



**Viktoria Muzychko** пишу кодик в  [mimacom Ukraine](#)

09.11.2016 16:14

Есть ли у вас линки на посоветовать по quick start and best practices?

[Ответить](#)  [Поддержать](#)



**Max Voloshin** Team Lead в [@OWOX](#)

09.11.2016 16:25

[tudorzureanu.com/akka-starter-kit](http://tudorzureanu.com/akka-starter-kit)

[Ответить](#)



[Поддержать](#)



**Alexandr Sova** Developer

09.11.2016 19:33

[Lightbend](#) (бывш. Typesafe) предоставляет т.н. seeds и templates — примеры и стартовые шаблоны приложений. Там есть в т.ч. примеры как разворачивается akka в разных стэках или standalone, на кластере и прочее и не только о akka. Обычно примеры дублируются на Java и Scala, или используется минимум сахара Scala для того чтоб было сразу понятно где что и как даже если читатель Scala до этого не видел ни разу.

[Ответить](#)

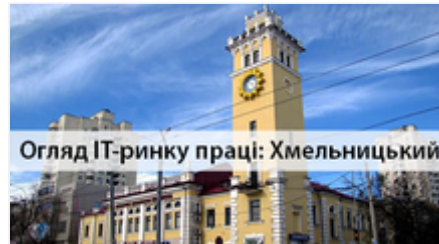


[Поддержать](#)

## Советуем почитать



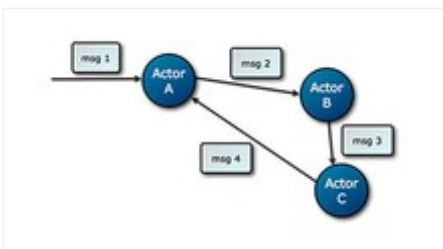
DOU Ревизор в Rentberry:  
«Квартира в доме, где жила  
Ахматова»



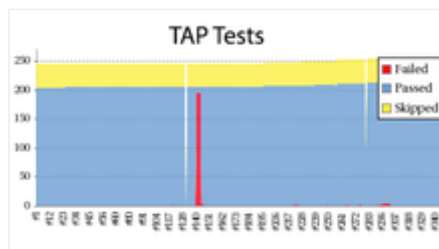
Огляд ІТ-ринку праці:  
Хмельницький



Почему ваш стартап зафейлится  
с вероятностью 99%



Реактивные приложения на Java с Akka



Контроль качества в Open Source: опыт проекта CRIU



Холакратия в действии: как каждый сотрудник может повлиять на стратегию компании

© 2005–2018 [DOU.ua](https://dou.ua)

[Українська](#) · [Русский](#) · [English](#)

Нас уже 241 020. Мы в соцсетях:



[Поиск программистов на Джинне](#)

[Контакты](#)

[Реклама](#)

[Legal](#)

Пишите нам на [support@dou.ua](mailto:support@dou.ua)