Разработка на DOCKER,

SPRING-CLOUD, Spring-BOOT

**Вы это создали и вам же с этим работать**

# Приложения, Doker

1. **Правила YML файлов**
   1. Все пробелы должны быть на месте, согласно образцам!  
      иначе не заработает
   2. Табуляций быть не должно!  
      иначе не заработает
2. **DOCKER**
   1. **Установка**
      1. … Взять из видео про докер
         1. Обязательно включить опции виртуализации
         2. Скачать Docker ToolBox и установить только его,   
            оставив все по умолчанию,  
            GIT должен быть установлен – есть галочка которую можно включить.  
            Для починки я еще ставил галочку драйвера NDIS 5
      2. Или предварительно установить Git, для хорошего терминала Git Bash, в котором можно работать как Linux, для стандартизации подходов и команд
      3. **Команды**  
         [ссылка](https://proglib.io/p/docker/)
      4. SSH [ссылка](https://habr.com/company/infopulse/blog/237737/)
   2. **Диагностика**
      1. docker ps   
         - посмотреть все процессы докера
      2. docker attach piggymetrics\_gateway\_   
         - Войти в докер, по имени машины
      3. **Взять настоящий адрес Docker на котором к нему можно присоединиться ИЗВНЕ!!**  
         docker-machine ip default   
         - взять IP войдя в терминал
      4. Обратиться к REST сервису на машине   
         curl http://$(docker-machine ip default):4000  
         curl http://$(localhost):4000
   3. Настройка приложения для старта как локально, так и докерах через compose
3. **ПРИЛОЖЕНИЕ gitblit-docker-example**
   1. ОПИСАНИЕ:  
      Приложение собираемое через maven, с плагином в нем, который делает образ
   2. Сборка  
      *mvn package docker:build*
   3. Запуск приложения   
      dockerexample/gitblit-docker-example
   4. Зайти в папку target  
      winpty docker run -p 8080:8080 -it dockerexample/gitblit-docker-example
   5. winpty – дает вывод на терминал
   6. посмотреть через браузер что все работает, по адресу   
      <http://192.168.99.100:8080/>   
      это адрес моего локального докера (проверить)
4. **ПРИЛОЖЕНИЕ Piggy-Metrics**
   1. ОПИСАНИЕ:  
      Docker + SpringBoot + Spring Cloud + Hystix + Api gateway + Monitoring   
      приложения запускаемые в Spring Cloud
   2. Запуск:  
      docker-compose up –d
   3. Настройка на локальный запуск:
      1. Для каждого приложения :
         1. создать рядом с файлом bootstrap.yml его копию,  
            файл bootstrap-development.yml, и в нем заменить облачный адрес  
            config:8888 на localhost:8888  
            [Настройка свойств spring cloud](https://cloud.spring.io/spring-cloud-commons/multi/multi__spring_cloud_context_application_context_services.html)
         2. Чтобы этот конфиг подхватился, надо из студии прописать настройку JVM аргументов:  
            -Dspring.profiles.active=development
      2. Минимальный локальный запуск:
         * 1. Config сервис  
              так как на него ссылаются все остальные
           2. Monitoring сервис  
              <http://localhost:8080/hystrix>
      3. **Остановка всех контейнеров**
         1. docker stop $(docker ps -a -q) [ссылка](https://coderwall.com/p/ewk0mq/stop-remove-all-docker-containers)
   4. Запуск
      1. Config – от него зависят все
      2. Мониторинг - Hystrix   
         <http://localhost:8080/hystrix>
      3. Приложение – gateway  
         <http://localhost:4000/>
      4. Registry Eureka -   
         <http://localhost:8761/>
   5. **Конфигурация**    
      Тут расположены все файлы с конфигурациями для всех сервисов – они их получают в RunTime  
      PROJECTS\PiggyMetrics\config\src\main\resources\shared
   6. **Ошибка с размножением нескольких registry**<https://stackoverflow.com/questions/31230255/eureka-how-do-i-disable-configure-peer-replication>  
      Устранение:  
      eureka:  
       client:  
       serviceUrl:  
       defaultZone: <http://localhost:8761/eureka/>
5. **Порт в WINDOWS – НЕ ПРОБРАСЫВАЕТСЯ** 
   1. Можно посмотреть внутренний порт докер-контейнера  
      docker inspect --format '{{ .NetworkSettings.Networks.bridge.IPAddress }}' webserver
   2. Реально работает IP из команды  
      **docker-machine env**  
      docker-machine env | grep DOCKER\_HOST  
      <http://192.168.99.100/>  
      Лежит тут:  
      "c:\Program Files\Docker Toolbox\docker-machine.exe"  
      ДОКА:  
      <https://stefanscherer.github.io/yes-you-can-docker-on-windows-7/>
6. **Работа с mongoDb**
   1. **Установка:** 
      1. из приложения Piggy-Metrics, запустить docker-compose c   
         docker-compose-mongo.dev.yml где проброшены порты
      2. Обратить внимание:  
         эта конфигурация наследуется от docker-compose-mongo.dev
   2. **Вызов терминала Mongo** 
      1. **Внутри докера**:  
         Выполнить:  
         winpty docker exec -it piggymetrics\_notification-mongodb\_1 mongo
      2. **Снаружи докера**:  
         Перейти в папку bin где находится клиент mongo  
         Выполнить:  
         ./mongo --port 34000 --host 192.168.99.100 -u user -p passwd  
         ./mongo --port 34000 --host 192.168.99.100
7. Запуск BASH файла под windows
   1. start "%ProgramFiles%\Git\bash.exe" command
8. **Запуск терминала в Docker**
   1. **winpty docker exec -i -t vgrigor-mongodb-small bin/bash**  
      можно по имени, а можно по идентификатору контейнера
9. Запуск монго терминала из хоста
   1. **winpty docker exec -i -t vgrigor-mongodb-small mongo**
10. Установка программы mc в докер на основе Ubuntu
    1. apt-get install - Обязательно  
       apt-get update
11. **Работа с Docker напрямую**
    1. Собрать image  
       **docker build -f Dockerfile . -t mongodb\_3**
    2. Выполнить image   
       с именем контейнера test  
       **winpty docker run -p 25000:27017 --name test -it mongodb\_3**
12. **Зачистка образов и контейнеров**
    1. Остановить все контенеры  
       docker stop $(docker ps -a -q)
    2. Удалить контейнеры  
       docker rm $(docker ps -a -q)  
       docker rm $(docker ps -a -f status=exited -q)
    3. Удалить образы  
       docker rmi $(docker images -a -q)
    4. RESTARTIG  
       Как убрать постоянно рестартующие контейнеры   
       - при наличии в них ошибок, если указана опция «restart: always» то при остаанове контейнера он будет рестартовать и никогда не остановится.  
       >>> Собрать и запустить те же образы без ошибки или без опции рестарт  
       restart: "no"
    5. **Работающая команда запуска MONGO при условии что вызов файла init.sh закомментирован**  
       docker-compose --verbose -f docker-compose-mongo-3.yml -f docker-compose-mongo-3.dev.yml up **–build  
       --build** делает команду работающей
    6. **Работает в таком виде – 4 версия**docker-compose --verbose -f docker-compose-mongo-4.yml -f docker-compose-mongo-4.dev.yml up --build
13. Работа с МОНГО
    1. Create user from docker shell  
       mongo --eval "db.createUser({ user: 'user', pwd: 'passwd', roles: [ {role:'readWrite', db:'piggymetrics'} ]})"
    2. Узнать порт монго изнутри  
       mongo  
       db.serverCmdLineOpts()
14. Работа с занятыми портами:
    1. Если какой-либо порт занят, определить можно так:  
       Выполнить:  
       cmd с админскими привилегиями  
       Выполнить:  
       netstat –b > file  
       и искать в файле

# Spring-BOOT

## SpringBootApplication

Аннотация собравшая все best-practices из Spring:  
Является:

Конфигурацией – можно порождать свои бины  
ComponentScan – все ваше подхватится

@EnableAutoConfiguration -

### Дока

[Spring Boot по русски](http://spring-projects.ru/guides/consuming-rest/)   
[По русски 2](http://spring-projects.ru/guides/spring-boot/)

## Spring Starter

### Зачем нужен

Spring Boot Starters make the bootstrapping process much easier and faster. The starter brings you required Maven dependencies as well as some predefined configuration bits. [Ссылка](https://dzone.com/articles/spring-boot-starters)

Всё удобство **Spring Boot** основано на использовании так называемых *Starter*, которые позволяют получить набор сконфигурированных бинов, готовых к использованию и доступных для конфигурации через properties-файлы. [Ref](https://habr.com/post/275337/)

Преждевременные выводы:

Выводы в первую очередь связаны с мнением о том, что Spring Boot — это большая магия, что там делается много всего, с чем потом невозможно разобраться.

Во-первых, разобраться можно. Это сложнее, чем разобраться со Spring, потому что Spring — это маленькая часть жизненного цикла Spring Boot. Это связано с тем, что он на себя взял ответственность за те проблемы, которые не любит брать на себя разработчик — конфигурации, зависимости, версии и запуск, сняв головную боль с нас. Кстати, часть проблем, связанных с неправильным использованием Spring, Spring Boot тоже забрал.

### Использование стартера

#### Сценарий- Быстрый старт

Включаем зависимость на стартер.

Получаем готовые бины, которыми можно рулить нужной системой.

Бин можно получить с помощью @Autowired

[КОД с пояснениями по конфигурации](https://github.com/infinispan/infinispan-spring-boot)

### Сценарий - Точно настроенное использование - настройка бинов - Модификация стартера

Бины настроены каким-то определенным образом, и если нам надо уточнить эту настройку, мы должны создать конфигурационный бин Spring, который возвращая ***разные*** бины используемые при конфигурации данной подсистемы, заменяют предопределенные бины.

Обратить внимание:

Стартеры включаются с помощью аннотации @EnableИмяСтартера

У каждого стартера есть несколько конфигураций.

Именно так надо проектировать свою подсистему, предназначенную для повторного использования: есть все готовое, но можно все перенастроить при желании.

#### Выгоды:

Мы можем сделать быстрый старт, опробировать готовое приложение без длительных проблем с настройками, а потом настроить так подсистему так точно как мы хотим, насколько это необходимо для вашей конкретной реальной ситуации использования.

Spring Boot упрощает инициализацию бинов, о которых вы не знаете, что они вам нужны. И он показал, как подключить удобные для управления сервисы.

### Не выгоды

Одна из целей: дать понимание того как все устроено, чтобы увидеть границу говна, ниже которого опускаться не рекомендуется чтобы не нанести вред проекту, излишним увлечением Spring начинкой которая имеет проблемы, ограничившись только лучшей и правильной частью.

Используйте Spring только для поднятия системной логики, а бизнес-логику организуйте отдельно.

### Конфигурирование

Вы можете создать свой класс конфигурации Spring, который будет возвращать бин настроенный на другой режим работы [Ссылка](https://dzone.com/articles/spring-boot-starters)

### Работа

СБ создает Uber-jar который можно запускать.

СБ стартер содержит 90 конфигураций с большим числом дочерних бинов, в том числе конфигураций, поэтому при старте поднимается большое число бинов. Этим отчаести можно управлять Conditional, но здесь начинаются сложности контроля: трудно достичь консистентности сочетаний условий. Все внутренние бины СБ прошиты Conditional.

**Физическая картина запуска**

Внутри СБ запускается Launcher который находит ваш класс SpringBootApplication и оттуда запускает main().  
Поэтому при окончании main() приложение не заканчивается.

### Зачем столько ? – «Узнайте, что вы можете делать с Spring Boot» [ref\_rus](http://spring-projects.ru/guides/spring-boot/)

Spring Boot предлагает быстрый путь для создания приложений. Он просматривает ваш classpath и бины, которые вы сконфигурировали, делает предположения о том, что нехватает и добавляет их. С Spring Boot вы можете сконцентрироваться больше на решении бизнес задач и меньше на инфраструктуру.

К примеру:

Требуется Spring MVC? Некоторые определенные бины вам почти всегда необходимы и Spring Boot добавит для них автоматически. Spring MVC приложению также необходим сервлет контейнер, поэтому Spring Boot автоматически настроит встроенный Tomcat

Требуется Jetty? Если так, то скорее всего вам не нужен Tomcat, а вместо него нужен встроенный Jetty. Spring Boot обработает эту задачу для вас

Требуется Thymeleaf? Для этого несколько бинов должны всегда быть добавлены в контекст вашего приложения; Spring Boot добавит их для вас

Это лишь несколько примеров, которые предоставляет автоманическая настройка Spring Boot. В то же время, Spring Boot не заработает с вашими настройками. Например, если Thymeleaf в вашем classpath, то Spring Boot добавит SpringTemplateEngine в ваш контекст приложения автоматически. Но если вы определите свой собственный SpringTemplateEngine с вашими настройками, то Spring Boot не добавит его. Это останется под вашим контролем и потребует от вас немного усилий.

## Spring Boot Actuator – сервис уровня бизнес-логики

Добавление зависимости spring-boot-starter-actuator

дает следующие возможности:

[ошибки](http://localhost:8080/error), [окружение](http://localhost:8080/env), [состояние](http://localhost:8080/health), [бины](http://localhost:8080/beans), [информацию](http://localhost:8080/info), [метрики](http://localhost:8080/metrics),[следы](http://localhost:8080/trace), [configprops](http://localhost:8080/configprops) и [дамп](http://localhost:8080/dump).  
  
[Статья](https://o7planning.org/ru/11757/monitoring-application-with-spring-boot-actuator)

# Полезные сведения для Использования Spring Boot

## Новый внутренний «InversionOfControl для Стартеров»

1. Используется в SpringBoot [Ref1](https://blog.frankel.ch/java-service-loader-vs-spring-factories/) [Ref\_video\_1](https://www.youtube.com/watch?v=yy43NOreJG4&t=917s)  
   @EnableAutoConfiguration для:
   1. гибкого управления зависимостями
   2. чтобы не было необходимости прописывать все зависимости
   3. Хорошие стартеры – несут в себе **внутри** свои SpringFactories и конфигурации, которые не надо прописывать нам.
2. [@EnableAutoConfiguration](http://docs.spring.io/spring-boot/docs/1.1.6.RELEASE/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) аннотация переключает на доступные по умолчанию настройки, основанные на содержимом вашего classpath. К примеру, он ищет любые классы, которые реализуют интерфейс CommandLineRunner и вызывает его метод run().  
     
   Посмотреть:  
   Class SpringFactoriesLoader
3. Посмотреть очень ценно:
   1. **Boot yourself, Spring is coming** [Статья](https://habr.com/company/jugru/blog/424503/) [Видео](https://www.youtube.com/watch?v=yy43NOreJG4&t=917s)

# Общие концепции

Разработка в SpringCloud позволяет лучшим образом организовать софт во многих отношениях, которые применимы к разработке правильной конструкции обычного монолитного софта с постепенным добавлением новых важных требований и возможностей:

1. Общие концепции в разработке правильного монолита и микросервисов
   1. Правильное разбиение программы на модули = хорошая конструкция системы
   2. Обеспечение отказоустойчивости модулей и всей программы при сбоях
      1. Самолечение
      2. Обеспечение работоспособности при падении отдельных модулей – вариант реализации StandIn
   3. Высокая доступность, соответственно отказоустойчивости
   4. Масштабирование – это типично первые шаги при ручном делении на монолита на большие функциональные части, когда ресурсов уже не хватает.
   5. Плагинность модулей
      1. Регистрация модулей
   6. Унифицированная конфигурация (в Cloud - config)
2. Удобная командная работа
   1. Автоматическое сопряжение разработанными разными командами модулей (сервисов), если об этом позаботились в виде соответствующего протокола
3. Удобная архитектурная работа
   1. Вынужденное применение хорошей архитектуры в виде деления на модули, обеспечение независимости их исполнения, унифицированности их взаимодействия, вместо системы беспорядочных вызовов.
   2. Boot гарантирует совместимость версий для большого числа компонент с помощью стартеров с одной версией

# Особенности

## Плохое и хорошее

Надо знать когда Spring работает понятно и надежно, а когда могут ожидать сложности, сбои в работе Spring, в казалось бы прямолинейных местах, и использовать лучшее, обходя сомнительное, для чего их надо знать.

## Когнитивная нагрузка

Если пойти неверным путем, то можно написать кучу инициализаций и конфигурацией, которые можно обойти правильным применением компонет Spring (Java в Облаке стр 81)

Для решения данной проблемы был придуман SpringBoot и ее автоматическое конфигурирование

## В чем же принципиальная выгода ?

SpringBoot на каждом этапе облегчает работу – с количеством конфигураций, версиями, автоматически лучшими практиками, и в результате получаем приложение которое легко писать, легко поддерживать, [при условии что вы не лезете в область ненадежного кода]

## Автоматический выбор компонент

SpringBoot анализирует ваши классы, и при наличии нужной аннотации, с помощью механизма condition включает нужные конфигурации.

Можно переопределить любые компоненты, тогда будут браться ваши – тоже с помощью механизма условий.

#### Отладка выбора компонент

В парfметрах запуска укажите –Ddebug=true, и SpringBoot покажет все срабатывания условий выбора компонент

# SPRING-CLOUD

## Интегрированное решение

[Integrating Hystrix Support and Dashboard](https://dzone.com/articles/spring-cloud-with-turbine)

Ваше приложение развивается, и чтобы оно хорошо жило на новом уровне - в новом мире распределенных технологий его надо:

1. Делить
2. Делать «промышленного качества» - делать хорошие мониторинг и диагностику
3. Делать фреймворки чтобы приложение было написано в однородно одном стиле вместо 25 стилей.

Это нужно всем и обуславливает необходимость выделения лучших практик

### Что дает Spring-Cloud + Boot

1. Цели:
   1. Сделать более быстро
   2. Сделать более правильно
   3. Лучше воспользоваться уже сделанным
   4. Надеяться получить «правильную архитектуру»
      1. Управляемую известными системами управления
   5. Единая экосистема для многих подходов и технологий
2. Все сложные вещи сделает за вас, и все что можно тоже
   1. Включение зависимостей
   2. Интеграция и первоначальная настройка вполне сложных технологий:
      1. RabbitMQ
      2. Kafka
      3. NetFlix OSS
   3. Best Practices в использовании всех подходов Spring
      1. Есть возможность изменить предопределенное поведение (Как?)
   4. Простой запуск Boot приложений, из командной строки
      1. включая веб-сервер
   5. Взаимодействие (Rest, Spring Streams)
   6. Надежность вызовов (hystrix)
   7. Мониторинг
   8. Масштабируемость
   9. Тестируемость – классика Spring, Spring Test

## Проекты

В отдельном файле – большой список проектов входящих в SpringCloud, они достойны рассмотрения, как состав, так и архитектура (TODO).

## Аспекты

### ConfigServer - Конфигурирование и его возможности

1. В набор проектов всегда входит сервис конфигурации
   1. В нем запускается готовый компонент «ConfigServer», который является источником свойств PropertySource для всех остальных проектов.  
      Т.е. обладает готовой инфраструктурой для конфигурирования всех остальных проектов.
   2. Запускается конфигурацией spring-cloud-config-server в зависимостях сервера  
      spring-cloud-starter-config – для клиента.
   3. Сервер представляет REST API, по которому клиенты обращаются к нему.  
      Сервер является посредником между клиентами и источниками конфигурации, которые могут быть разными – локальными фалами или GIT хранилищем.
   4. Привязка клиентских конфигураций осуществляется по имени клиентского приложения, которое задается в настройках приложения Spring Cloud, и имени файла конфигурации на сервере.
   5. Конфигурирование может быть динамичным: аннотация @RefreshScope позволяет автоматически обновляться всем бинам помеченным ей.
   6. Активация происходит по событию обьекта ApplicationContext типа RefreshScopeRefreshedEvent.
   7. Снаружи обновление можно вызвать вызовом АКТУАТОРА  
      <http://127.0.0.1:8080/refresh> или в JMX актуатора  
      **! Необходимо отдельно вызвать для всех сервисов**
   8. SpringCloudBus поддерживает обновление сразу нескольких экземпляров сервисов  
      Он получает ссылки на все подключенные сервисы через SpringCloudStream,   
      вот как выглядит вызов:  
      <http://127.0.0.1:8080/bus/refresh>  
      Реализует это подключение «**автоматической конфигурации**» **spring-cloud-starter-bus-ampq**

### Discovery – Netflix Eureka

1. Проблемы:
   1. Нам нужна динамическое обнаружение сервисов, в условиях когда они падают
2. Возможности
   1. Это registry для всех приложений
3. Плюсы:
   1. Это организация работы типичного распределенного приложения, которое пишется самостоятельно очень сложно.
4. Запуск: spring-cloud-starter-eureka-server

**Проблема: Кластер – не понятно как делать**

### Gateway - Пограничные сервисы, или конечные точки

1. Это сервис которые принимает запросы «извне»
2. API – адаптеры (?)

## Технологии

### Spring Cloud Stream

Spring Cloud Stream Application Starters are standalone executable applications that communicate over messaging middleware such as Apache Kafka and RabbitMQ. These applications can run independently on variety of runtime platforms including: **Cloud Foundry, Apache Yarn, Apache Mesos, Kubernetes, Docker, or even on your laptop**.

Spring Cloud Stream is a framework for building highly scalable event-driven microservices connected with shared messaging systems.  
The framework provides flexible programming model built on already established and familiar Spring idioms and best practices, including support for persistent pub/sub semantics, consumer groups, and stateful partitions

Ценная статья: [Удивительно простой обмен сообщениями с Spring Cloud Stream](https://habr.com/post/350454/)

[Быстрый старт](https://cloud.spring.io/spring-cloud-stream/#quick-start)

Sample: [Binding to Aeron](https://github.com/viniciusccarvalho/spring-cloud-stream-binder-aeron)

### Spring Cloud Data Flow - Samples

[Ref](https://docs.spring.io/spring-cloud-dataflow-samples/docs/current/reference/htmlsingle/#_scdf_metrics_with_prometheus_and_grafana)

### Comparison to other Platform architectures

[Ref](https://docs.spring.io/spring-cloud-dataflow-server-mesos/docs/1.0.0.RELEASE/reference/html/arch-microservice-style.html#arch-comparison) [Spring Cloud Data Flow Server for Apache Mesos](https://docs.spring.io/spring-cloud-dataflow-server-mesos/docs/1.0.0.RELEASE/reference/html/index.html)

### Оценка скорости SpringCloudDataFlow через RabbitMQ

[Код и описание](https://github.com/rabbitmq/scdf-performance/tree/master/scst/scst-rabbitmq-performance)

TODO: отличный пример насчет попробовать завернуть сложное приложение в Docker

### Spring Cloud Bus

Spring Cloud Bus - это абстракция, построенная на вершине Spring Cloud Stream (и, следовательно, kafka и rabbitmq). Это не общая цель, а построена для отправки административных команд сразу нескольким узлам службы. Например, отправка обновления (из весеннего облачного сообщества) во все узлы пользовательской службы. Есть только один канал, где в весеннем облачном потоке их много. Подумайте об этом как о распределенном пружинном приводе.

The only implementation currently is with an **AMQP** broker as the transport, but the same basic feature set (and some more depending on the transport) is on the roadmap for other transports.

Программа ценного курса:   
**«Coordinating Services: Java Microservices with Spring Cloud»** [**Ссылка**](https://www.pluralsight.com/courses/java-microservices-spring-cloud-coordinating-services)

1. Introducing Spring Cloud and Microservices Coordination Scenarios  
   how to use them to deliver more resilient, scalable, and maintainable microservices.
   1. Overview
   2. Popularity, Characteristics, and Challenges of Microservices
   3. Spring Cloud and How We'll Use It in This Course
   4. Course Goals and Prerequisites
2. Locating Services at Runtime Using Service Discovery  
   discuss some options for advanced configuration before wrapping up  
   client side load balancing
   1. Overview
   2. The Role of Service Discovery and the Problem with the Status Quo
   3. About Spring Cloud Eureka
   4. Creating a Eureka Server and Using the Eureka Dashboard
   5. Demo: Creating a Eureka Server
   6. Registering a Service with Eureka
   7. Demo: Registering Multiple Services with Eureka
   8. Discovering a Service with Eureka
   9. Demo: Discovering and Load Balancing Services with Eureka
   10. Configuring Service Health Information
   11. Adding a Custom Health Check to a Eureka-registered Service
   12. High Availability Architecture for Eureka
   13. Advanced Configuration Options
   14. Summary
3. Protecting Systems with Circuit Breakers  
   discuss the idea of Turbine for aggregating results from different Hystrix Protected Services and pulling that into a single Dashboard
   1. Overview
   2. The Role of Circuit Breakers and the Problem with the Status Quo
   3. About Spring Cloud Hystrix
   4. Creating a Hystrix-protected Service
   5. Demo: Adding a Circuit Breaker to Your Application
   6. All About the Hystrix Dashboard
   7. Demo: Setting up and Using the Hystrix Dashboard
   8. Advanced Hystrix Configuration
   9. What Turbine Adds to Hystrix?
   10. Demo: Using Turbine Stream to Aggregate Metrics
   11. Summary
4. Routing Your Microservices Traffic  
   configure Spring Cloud Ribbon to use it more effectively  
    how you can extend Zuul with different filters  
   with Zuul, we'll see some more advanced cloud load balancing as we go to the fast pass service, and get a full sense of how Ribbon and Zuul work with the rest of the family of Spring Cloud projects
   1. Overview1m
   2. The Role of Routing and the Problem with the Status Quo
   3. About Spring Cloud Ribbon
   4. Configuring Ribbon in Your Applications
   5. Demo: Using Ribbon Without Eureka Integration
   6. How Ribbon and Eureka Work Together
   7. Demo: Using Ribbon with Eureka Integration
   8. Customizing Ribbon Through Configuration
   9. Demo: Changing Ribbon Behavior Through Configuration
   10. About Spring Cloud Zuul
   11. Creating a Zuul Proxy with Configurable Routes
   12. Demo: Creating a Zuul Proxy and Experimenting with Routes
   13. About Zuul Filters and Stages
   14. Demo: Creating and Using Zuul Filters
   15. Summary
5. Connecting Microservices Through Messaging  
    exciting abstractions in Spring Cloud Stream, specifically things like consumer groups as well as doing partition based processing
   1. Overview1m
   2. The Role of Messaging and the Problem with the Status Quo
   3. About Spring Cloud Stream
   4. Understanding Binders and Configuring Stream Applications
   5. Demo: Creating Message Senders and Receivers
   6. More Options for Producing and Consuming Messages
   7. The Role of Processors in Spring Cloud Stream
   8. Demo: Creating Custom Interfaces and Using Processors
   9. RabbitMQ and Apache Kafka Binding Propertie
   10. Using Consumer Groups to Scale
   11. Demo: Pushing Messages and Configuring Consumer Groups
   12. Stateful Processing with Partitions
   13. Demo: Setting up and Using Partitions
   14. Working with Content Types
   15. Spring Cloud Stream Health and Metrics0m
   16. Summary
6. Building Data Processing Pipelines Out of Microservices  
   Spring Cloud Data Flow
   1. Overview
   2. The Role of Orchestration and the Problem with the Status Quo
   3. About Spring Cloud Data Flow
   4. About the Data Flow Server
   5. Consider Streams vs. Tasks
   6. Where 'Apps' Come From
   7. Installing Spring Cloud Data Flow
   8. Demo: Getting Spring Cloud Data Flow up and Running
   9. About Stream Starter Apps
   10. Creating Streams with Spring Cloud Data Flow
   11. Deploying Data Pipelines
   12. Demo: Creating and Deploying Streaming Pipelines
   13. Creating Tasks with Spring Cloud Data Flow
   14. Demo: Creating and Executing Tasks
   15. Using the Spring Cloud Data Flow Dashboard and Flo
   16. Demo: Building, Deploying, and Tapping Streams from the Dashboard
   17. Creating Custom Stream or Task Apps
   18. Creating, Registering, Using, and Partitioning Custom Apps
   19. Creating Composed Tasks
   20. Creating Custom Task Apps and Arranging in a Composed Task
   21. Monitoring Spring Cloud Data Flow Pipelines
   22. Updating Apps, Streams, and Tasks
   23. Summary2m

# YARN – новое поколение вычислительной платформы Hadoop

[Ref](https://www.ibm.com/developerworks/ru/library/bd-yarn-intro/index.html)

Участвует в SpringCloud, но интересен как отдельный компонент управления распределенными вычислениями

Общая дока [Spring Cloud Data Flow for Apache YARN](https://docs.spring.io/spring-cloud-dataflow-server-yarn/docs/1.2.2.RELEASE/reference/htmlsingle/)

#### Apache Twill for ****Hadoop and YARN****

[Ref](https://jaxenter.com/developing-distributed-applications-with-apache-twill-107728.html)

Почему :Twill  
YARN has been called the “Operation System of the Data Center”, and it honors this title with very generic and powerful APIs and protocols. But with that power comes complexity, and learning YARN is a steep uphill climb.

# Ссылки

Основы Docker за Х часов и Y дней

<https://habr.com/post/337306/>

# Лабораторная работа: введение в Docker с нуля. Ваш первый микросервис

<https://habr.com/post/346634/>