

Применение Go в SRE

Slava Bakhtmutov
Site reliability engineer at Dropbox



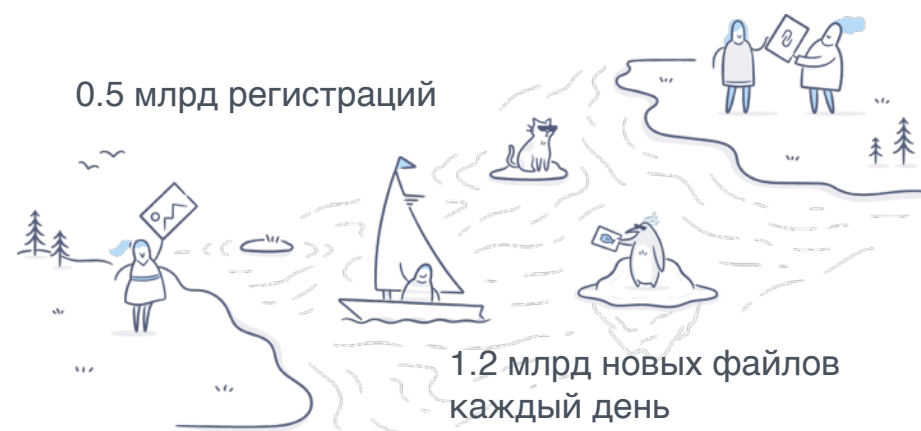
Go в SRE в Dropbox

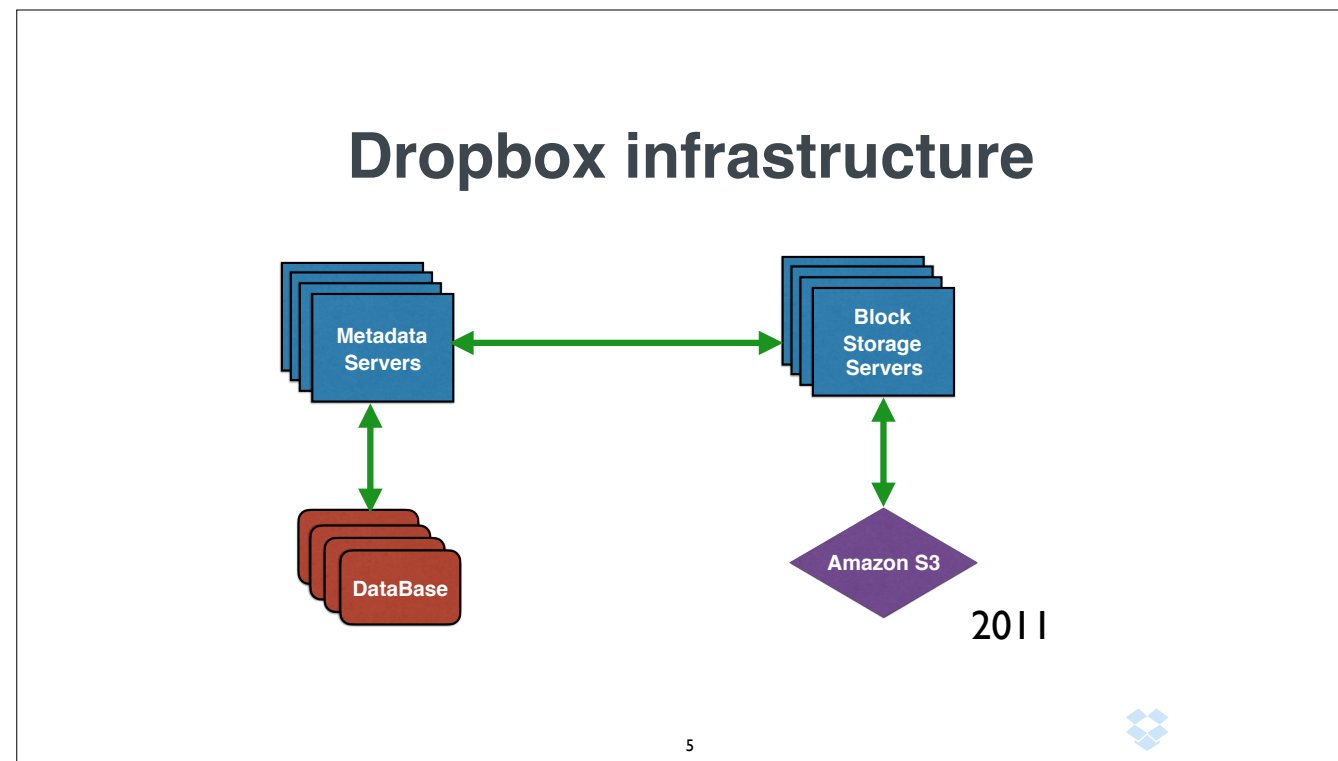


Go в SRE в **Dropbox**



Dropbox



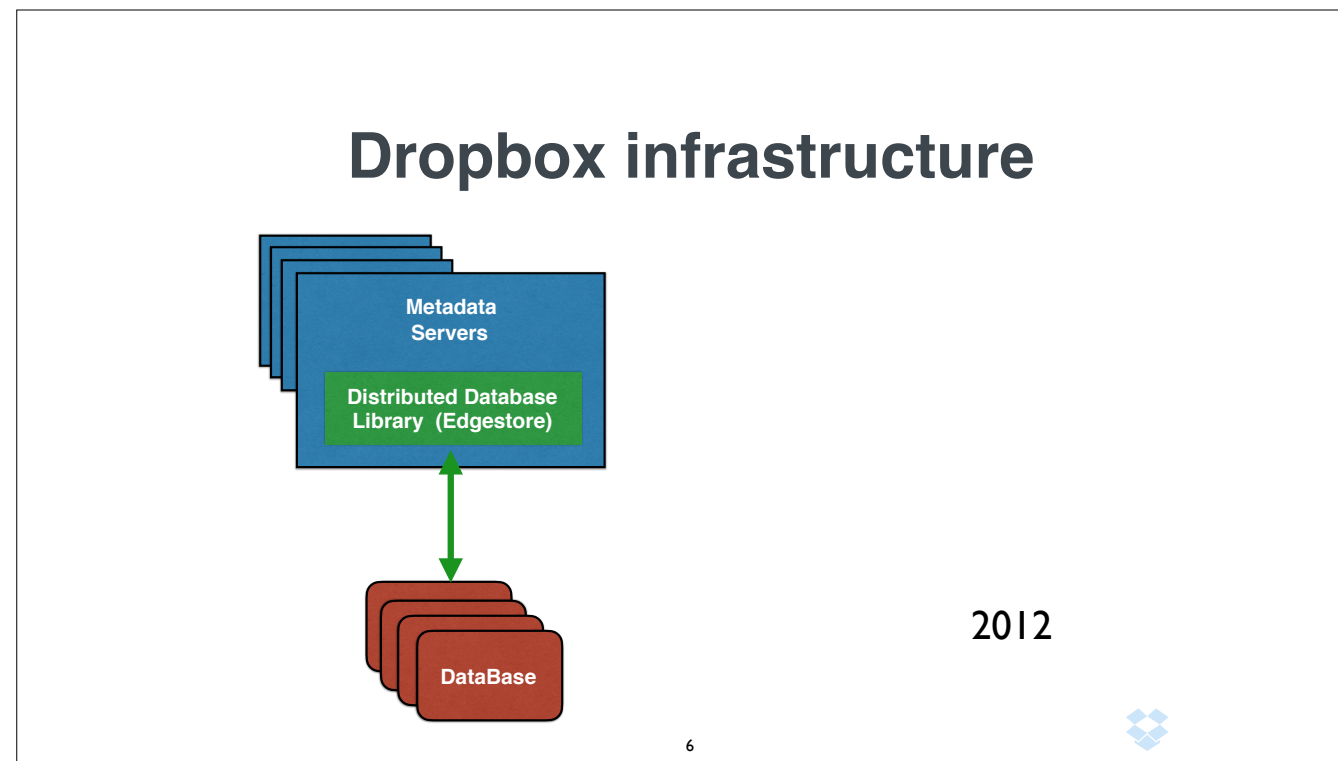


First we should go over brief history of Dropbox infrastructure.

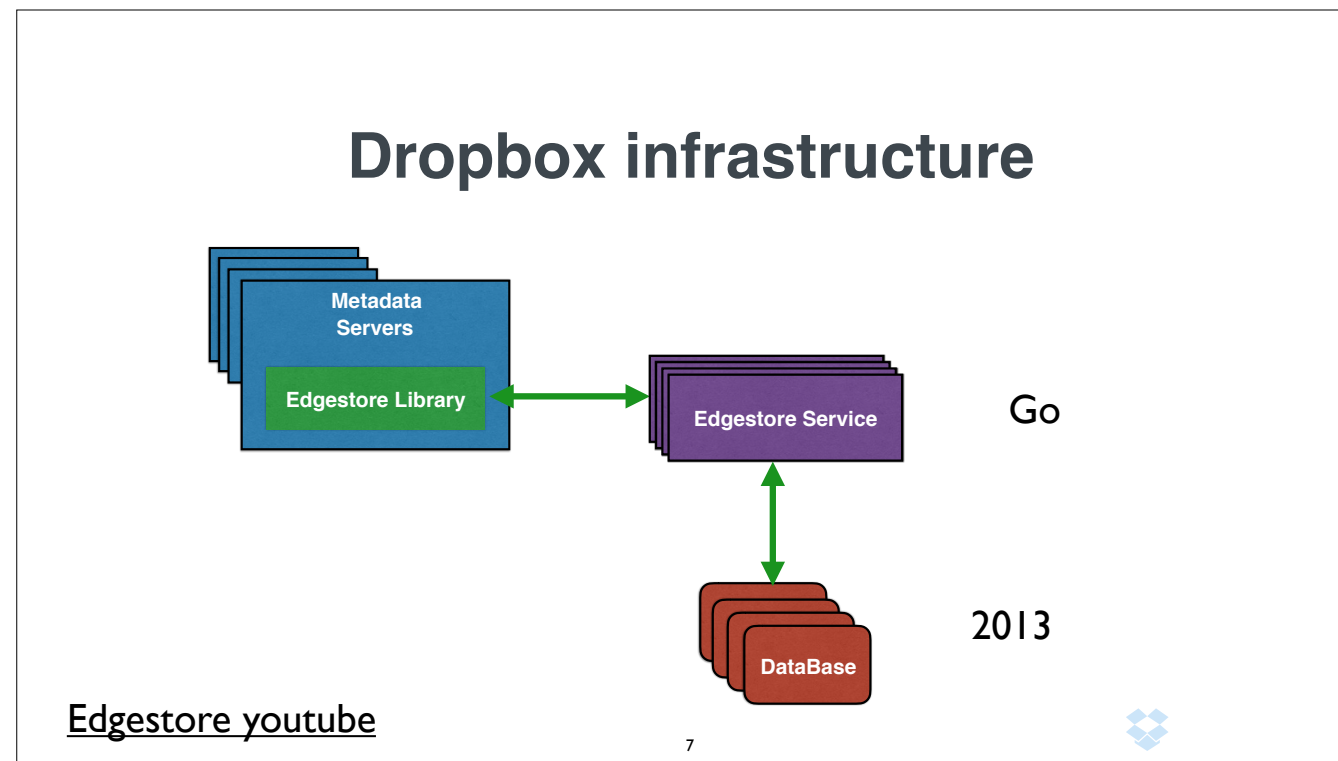
Simplified view of state of things in 2011. Two primary parts of the system Metadata servers, and Block Storage servers.

Metadata servers store metadata as you would expect. Block Storage servers are simpler mainly storing large blobs indexed by unique keys.

This is of course very simplified version of the world. There are different metadata server clusters for web and api traffic for example and there are bunch of other systems at play rather than just databases, but fundamentally Metadata server and Block Storage server are two monolithic binaries that can do everything.

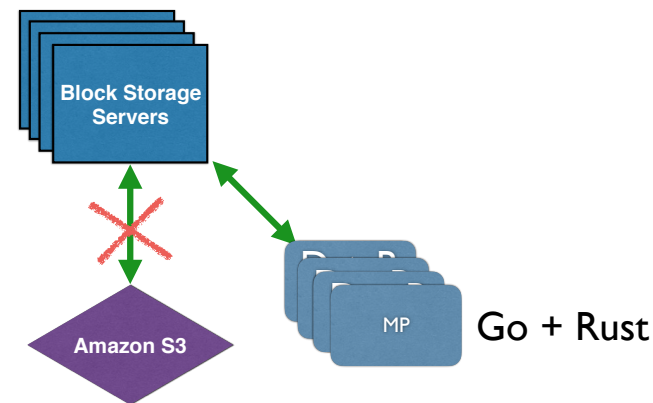


Monolithic system doesn't necessarily mean lack of abstraction. We had built our own Distributed Database system to abstract away data partitioning. However the system was built as a library. Pros of library: faster initial iteration. No need to worry about: deployment, discovery or communication aspects.



For stability, correctness and faster deployment need to separate as separate service with very light weight library.
More stable and mature infrastructure to take time to build this properly.
Time to think what language to build the service in. We knew we wanted to move away from python.

Dropbox infrastructure



2015

Эпичная история, как Dropbox ушёл от Amazon в своё облако - Wired



8

Magic Pocket was built using a new programming language from Google called Go. Here too, Dropbox is riding a much larger trend, languages designed specifically for the new world of massively distributed online systems. Apple has one called Swift, Mozilla makes one called Rust, and there's an independent one called D. All these languages let coders build software quickly that runs quickly—even executed across hundreds or thousands of machines.

Rust? Hell yeah! <https://news.ycombinator.com/item?id=11283538>

Dropbox infrastructure

Логи/исключения	Batch Jobs
Система push нотификаций	DRTs (как Chaos Monkey)
Метрики	wheelhouse (автоматизация долгих задач)
Маршрутизация/балансировка	Naoru - ремедиэйшны
Alerts/remediation	
Системы деплоя	
Hadoop/аналитика	и множество других

9



Помимо основных сервисов, инфраструктура так же состоит из множества поддерживающих компонентов, работа каждого из которых так же важна.

Здесь и приходят на арену SRE.

Go в **SRE** в Dropbox

10



SRE это достаточно размытое понятие, и не так часто встречается за пределами крупных компаний.

История

1. Большие компьютеры. Ручное управление.
2. Дешёвые компьютеры. Автоматизация.
3. Облака. DevOps.

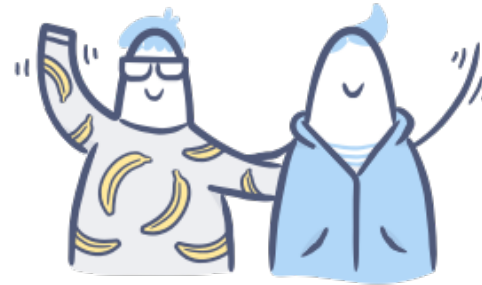
II



History

1. Old days. Big massive computers. Mainframes. Buy hardware. Buy Software.
 1. Hire operations staff
 2. Specific knowleges about hardware and software. Но инженеры, примерно только понимают на каком железе они будут запускаться.
 3. Hard to automate. Ходили с дисками и меняли их
2. Следующий шаг. Hardware prices down. Internet speeds up. 15 computers, don't care if they died.
 1. Nagios.
 2. Automatization.
 3. NoCs – network operations center. Есть алерт с диском, надо среагировать и поднять новую машину.
3. Cloud. Железо – абстрактная штука. SaaS, IaaS.
 1. Есть алёрт с диском – фигня, аллоцируем новую машину из облака.
 2. Разработчики теперь становятся ответственным за автоматизацию.
 3. DevOps! Бекапы, сами делаете, деплоите.
 4. Софт для автоматизации. Puppet, Chef.

SRE



Сисадмины + разработчики

Что такое Site Reliability Engineering (Ben Treynor)

12



SA – system administrators
SWE – software engineers
SRE – site reliability engineers

SRE – are software engineers who focus on how to deliver reliable service in a sustainable way.

Но гугл уже был здесь, и то что мы называем DevOps. В гугле это SRE. Но немного более расширенная версия. SRE это всего лишь программисты, которые специализируются на reliability. Как программисты игр, например. Главное – это чтобы продукт, работал. Writing Vs Running.

Developers работают в oncall ротации.

SRE работают в какой-то из команд, обычно там где больше всего нужно, есть команды где всегда есть SRE. SRE вникают в работу команды и привносят культуру. Потом они могут мигрировать в другую команду.

SRE в Dropbox

1. Распределённые San Francisco/Dublin
2. Fullstack (от железа до приложений)
3. Все системы (database, search, traffic, phit)



13



Распределённые, чтобы не дежурить ночью.

Умеют и программировать и администрировать

Знают все системы на хорошем уровне. from hardware model, network stack, hdfs, kafka, and application as well

SRE не магия

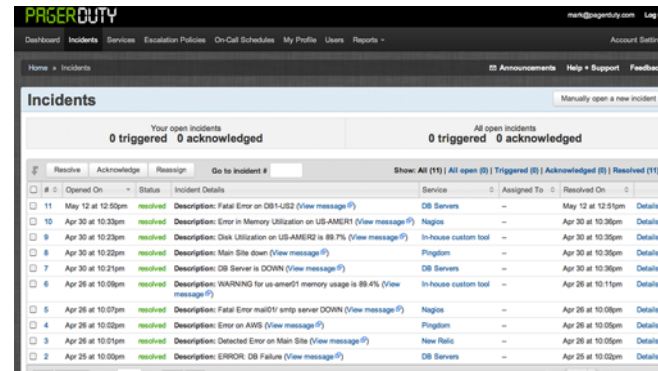
Определение где проблема и исправление её. Стараемся чтобы в будущем её не было или она была пофикшена быстро.

Durability, SLA, – на чём фокусируемся.

SRE операционная

Мониторинг!

1. Причины и Симптомы
2. День и ночь
3. Влияние в playbooks



#	Opened On	Status	Incident Details	Service	Assigned To	Resolved On
11	May 12 at 12:52pm	resolved	Description: Fatal Error on DB1-US2 (View message)	DB Servers	--	May 12 at 12:51pm
10	Apr 30 at 10:33pm	resolved	Description: Error in Memory Utilization on US-AMER1 (View message)	Nagios	--	Apr 30 at 10:35pm
9	Apr 30 at 10:23pm	resolved	Description: Disk Utilization on US-AMER2 is 89.7% (View message)	In-house custom tool	--	Apr 30 at 10:35pm
8	Apr 30 at 10:22pm	resolved	Description: Main Site down (View message)	Pingdom	--	Apr 30 at 10:35pm
7	Apr 30 at 10:21pm	resolved	Description: DB Server is DOWN (View message)	DB Servers	--	Apr 30 at 10:35pm
6	Apr 26 at 10:08pm	resolved	Description: WARNING for us-amer01 memory usage is 89.4% (View message)	In-house custom tool	--	Apr 26 at 10:11pm
5	Apr 26 at 10:07pm	resolved	Description: Fatal Error mail01 smtp server DOWN (View message)	Nagios	--	Apr 26 at 10:08pm
4	Apr 26 at 10:02pm	resolved	Description: Error on AWS (View message)	Pingdom	--	Apr 26 at 10:05pm
3	Apr 26 at 10:01pm	resolved	Description: Detected Error on Main Site (View message)	New Relic	--	Apr 26 at 10:05pm
2	Apr 25 at 10:02pm	resolved	Description: ERROR: DB Failure (View message)	DB Servers	--	Apr 25 at 10:02pm

Operations should be less then 50%

Monitoring/troubleshooting

Alerting philosophy

1. Cause vs Symptom
2. Day vs Night
3. Impact in playbooks

Причина – например умерла машина

Симптом – понизился сла.

Причина не всегда должна порождать Симптом, Причина – это что-то уже известное, Симптом часто это что-то не ясное.

Есть причина – машина умерла, но ваш софт умеет с этим справляться, SLA не упал, симптома нет. Но не обязательно ждать до утра, чтобы починить машину, это можно автоматизировать.

Вокруг автоматизации должен быть тоже мониторинг. Потому что мы потеряли вчера 2 машины и мы потеряли 45 машин – это разные вещи.

SRE

1. Наблюдение
2. Разбирательство
3. Решение
4. Действие



15



Наблюдаем, реагируем на алерты и пейджеры, смотрим на графики.

Разбирательство – не бежим чинить, пытаемся разобраться, найти корень и импакт

Решение – точно знаем, что делать, чтобы предотвратить

Действие – чиним

DRTs

1. Зачем

1. Ловим проблемы
 2. Доказываем суждения
 3. Спокойствие для дежурных
- ## 2. Регулярно и в продакшене

Hope is not a strategy
- Ben Treynor (Google)

Как параноя помогает нам в dropbox ¹⁶



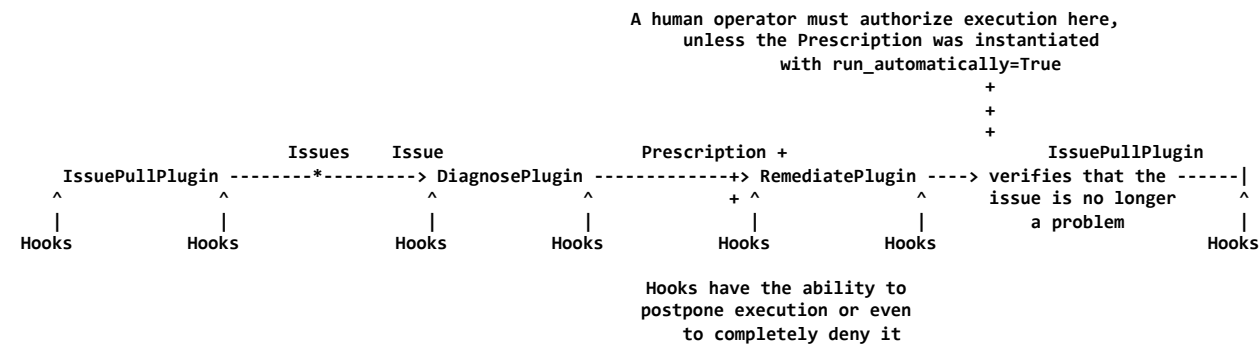
Ловим проблемы – проверяем что все системы работают, алерты и ремедейшены.

Доказываем assertions – например если система на трёх нодах (например кафка) говорит, что при выходе из строя одной ноды, всё продолжит работать – это надо проверить.

Успокаиваем дежурных – находим проблемы в дневное время и когда мы этого сами ожидаем.

мы не продаём надежду что всё будет работать, мы даём реальные обещания, знаем как мерить их, знаем как тестировать это и проверять.

SRE автоматизация



Naoru - auto-remediation framework

17



Naoru – Параноидальная автоматизация

Scary Tasks:

Rebooting servers – too many, single point of failure

Hardware replacements – return whole rack, or reallocate. Do we have enough capacity?

Reformatting hard drives – we have copy of data from it somewhere

Naoru

Alerting



Алерт.

```
if __name__ == "__main__":  
    try:  
        subprocess.check_call(["ssh", hostname, "uptime"])  
        print "OK"  
        sys.exit(0)  
    except Exception:  
        print "Critical: host is not reachable"  
        sys.exit(2)
```



Can we just have a cronjob that run all this scripts mapped on alerts?
Horrrifying – all servers rebooted at the same time! Oh

Naoru



Diagnose plugin – взять алёрт, определить основную причину и решить какое действие предпринять

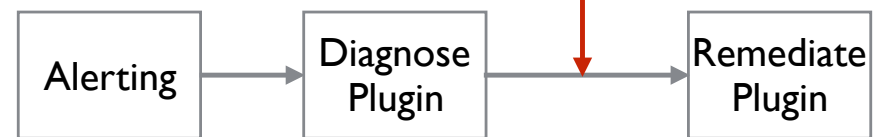
Диагностика

1. Проверить что ssh работает
2. Подключиться по IPMI
 - A. Нет ответа - Перезагрузка
 - B. Зависло в initramfs - Деаллокация машины
 - C. CPU soft lockup - Перезагрузка



Naoru

Авторизация
оператором



Naoru

After a new disk is added
a reboot is necessary for the block device to appear.

This remediation will reboot the host

The following will be run:

```
dropbox/naoru/naoru_dropbox/bin/naoru reboot_host ash-ra9-6a
```

May this run? [y/n]

23



Human automized tool.

Screen with human automatized execution

What happened and what we are going to do about it. May this run?

Первые четыре месяца после запуска надо было написать:

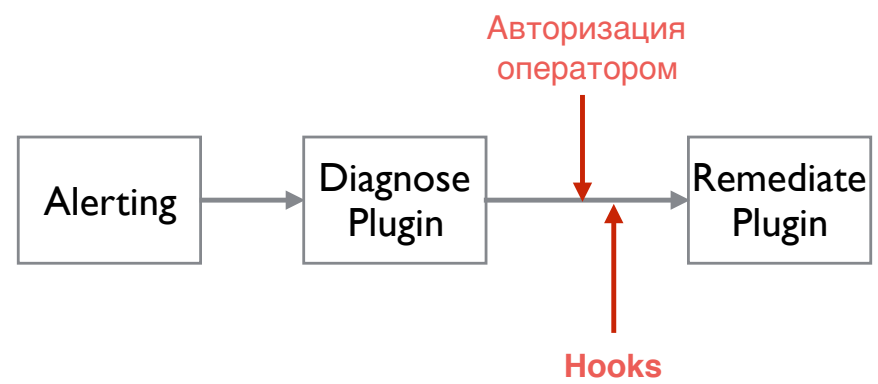
Эта выглядит для меня правильным, запустить. – Нужно было написать в правильном регистре и с запятыми. С проверкой что это tty и рандомным регистром.

Почему я не должен этого запускать? Сейчас SEV. Или причина неверная итд.

Automate verification of safety. Rate limits, check replication status etc.

С каждой ручной проверкой – улучшается скрипт верификации

Naoru



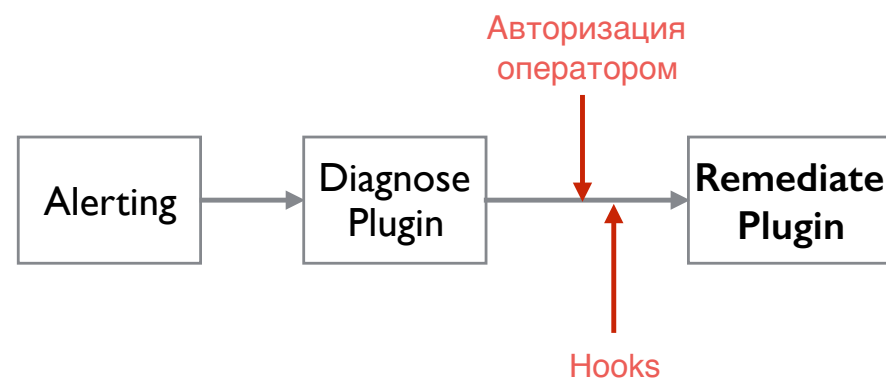
Хуки

```
class ZookeeperEnsembleAllUp(Hook):  
    """ Если предписание выписано для хоста Зукипера, тогда удостовериться  
    что все члены его группы запущены и работают.  
    """  
    def pre_remediate(self, remediate_plugin, prescription):  
        hostname = prescription.issue.attributes["hostname"]  
        if zookeeper_ensemble_all_up(hostname):  
            return Hook.EXECUTE_PROCEED  
        else:  
            return Hook.EXECUTE_POSTPONE
```

~



Naoru

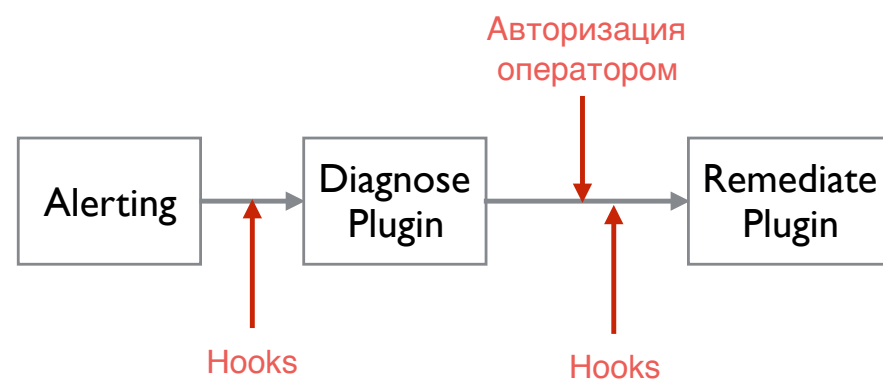


Naoru

```
class RebootToReconfigureKernel(RemediatePlugin):  
    def remediate(self, prescription):  
        hostname = prescription.issue.attributes["hostname"]  
        puppet.run_puppet_agent(hostname)  
        reboot_via_shutdown(hostname, self)
```



Naoru





Go vs Python

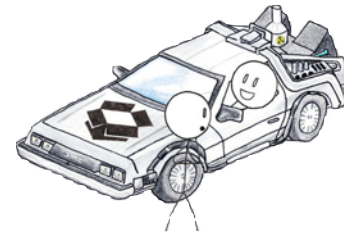
Производительность

Корректность

Параллелизм

Потребление ресурсов

Рефакторинг



30



Performance – всё понятно, есть потуги, даже у нас есть проект Pyston, который пытается ускорить питон тем или иным образом. PyPy итд
Но это не главная проблема

Correctness – и это даже не про статическую типизацию, хотя конечно это важно и даже для питона мы делаем шаги в этом направлении, например используем MyPy и периодически тестируем всю нашу кодовую базу на корректность (это занимает ну очень много времени)

Это больше про то, как в си плюс плюс, вы импортируете какую-нибудь библиотеку, о которой вы возможно даже не знаете, она импортируется через третьи руки. Какой-нибудь логгер. А в ней некорректная работа с рандомной памятью и вот уже в вашей основной библиотеке какая-то ошибка, которую вы даже не сможете отловить тестами. Всё тоже самое происходит и с питоном, когда вы импортируете сишные библиотеки, да и вообще питон просто подталкивает вас на изменение кода на лету.

Parallelism это тоже очень важно и стоит отдельно от производительности, потому что вы можете очень хорошо ускорить линейную производительность, но совсем не использовать параллелизм по тем или иным причинам. Например этим страдают инструменты от перконы, оптимизированные под одно ядро.

Code Refactoring – тут понятно. Типизация во все поля (MyPy и тесты несколько спасают)

Go vs Java(JVM) vs C++ vs Rust

31



Субъективное мнение

Go. Плюсы

- **Тесты**
- Горутины
- Простота языка (легко переходят с Python)
- Инструментарий. Исследование проблем, отлов утечек
- Малое потребление ресурсов, производительность
- Быстрая компиляция
- Нет зависимостей
- Богатая stdlib. HTTP2!



Go. Тесты

`go test -bench`

`go test -bench -cpu=1,2,4`

`go test -benchmem`

`go test -cpuprofile`

`go test -memprofile`

`go test -blockprofile`



Go. Отладка

Профайлер горутин

Трејсер GC

Трејсер планировщика

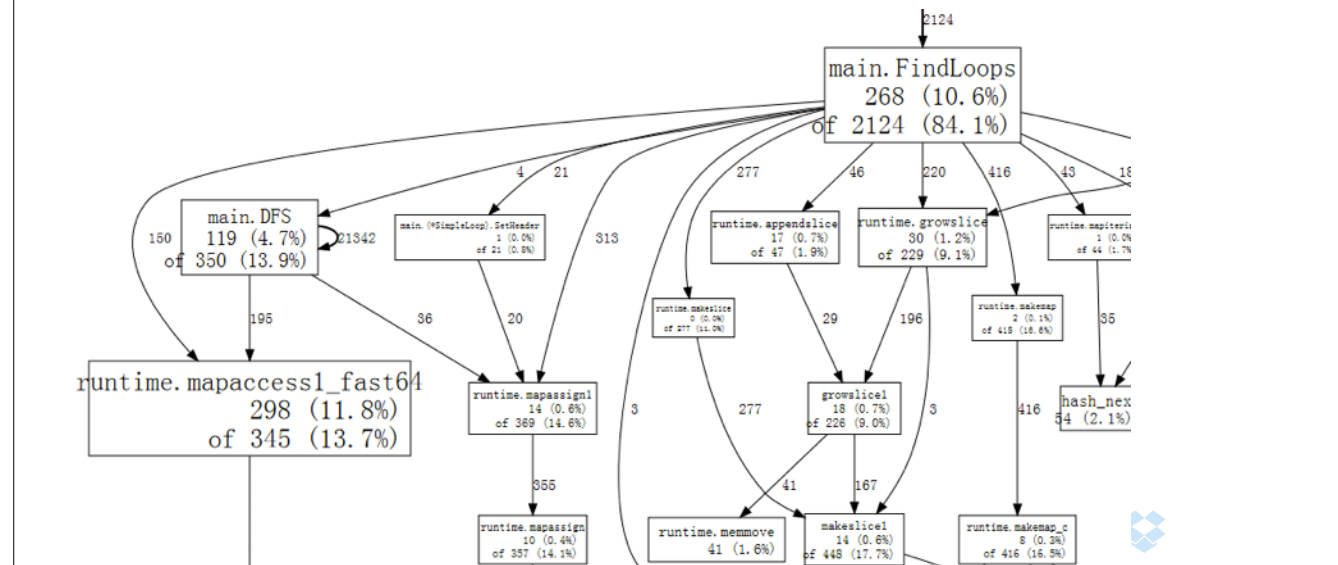
Трејсер аллокатора памяти

Статистика аллокатора памяти

Дамп кучи



Go. Отладка



Go. Гонки

```
=====
WARNING: DATA RACE
Read by goroutine 5:
  main.Routine()
    /Users/bill/Spaces/Test/src/test/main.go:29 +0x44
  gosched0()
    /usr/local/go/src/pkg/runtime/proc.c:1218 +0x9f

Previous write by goroutine 4:
  main.Routine()
    /Users/bill/Spaces/Test/src/test/main.go:33 +0x65
  gosched0()
    /usr/local/go/src/pkg/runtime/proc.c:1218 +0x9f
```



Go. Плюсы

- Тесты
- Инструментарий. Отладка
- Горутины
- Простота языка
- Малое потребление ресурсов
- Производительность
- Быстрая компиляция
- Нет зависимостей
- Богатая stdlib. HTTP2!



Go in Dropbox

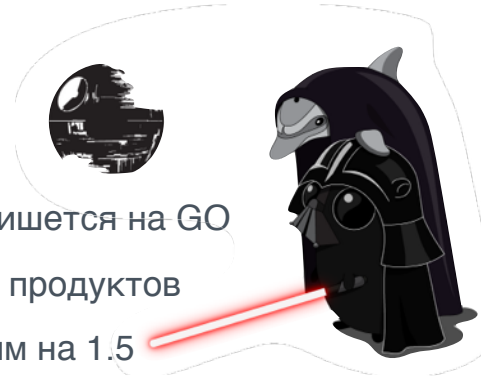
~400k строк кода

~130 разработчиков

Почти вся инфраструктура пишется на GO

Экспериментируем с GO для продуктов

Используем Go 1.3, переходим на 1.5



Go. Выводы

Начинать с переписывания одной из основных систем

Свои библиотеки легче поддерживать и менять

Осторожная работа со сборщиком мусора



Go. Экосистема



Картинка из статьи Go в 2015 году

40



Go. Экосистема



CoreOS



Go

С чего начать новичку (ссылки)

Митапы

Русскоязычный чат (1000+ человек)

Подкаст

Статьи и книги (Керниган!)

Редакторы (idea, vim, sublime)



Вопросы?

twitter.com/m0sth8

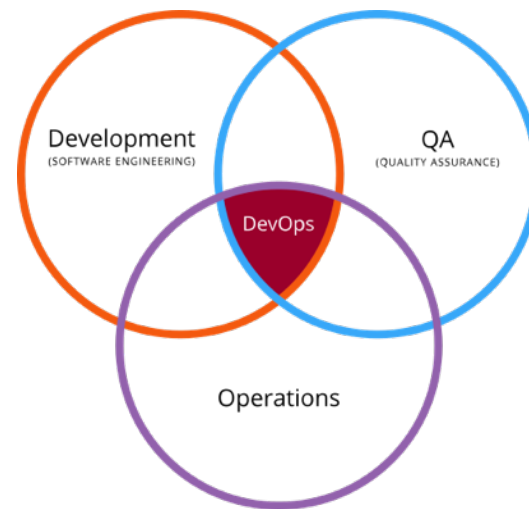
m0sth8@gmail.com

slava@dropbox.com

Slava Bakhmutov
Site reliability engineer at Dropbox



SRE



SA

SWE



Devops and SRE