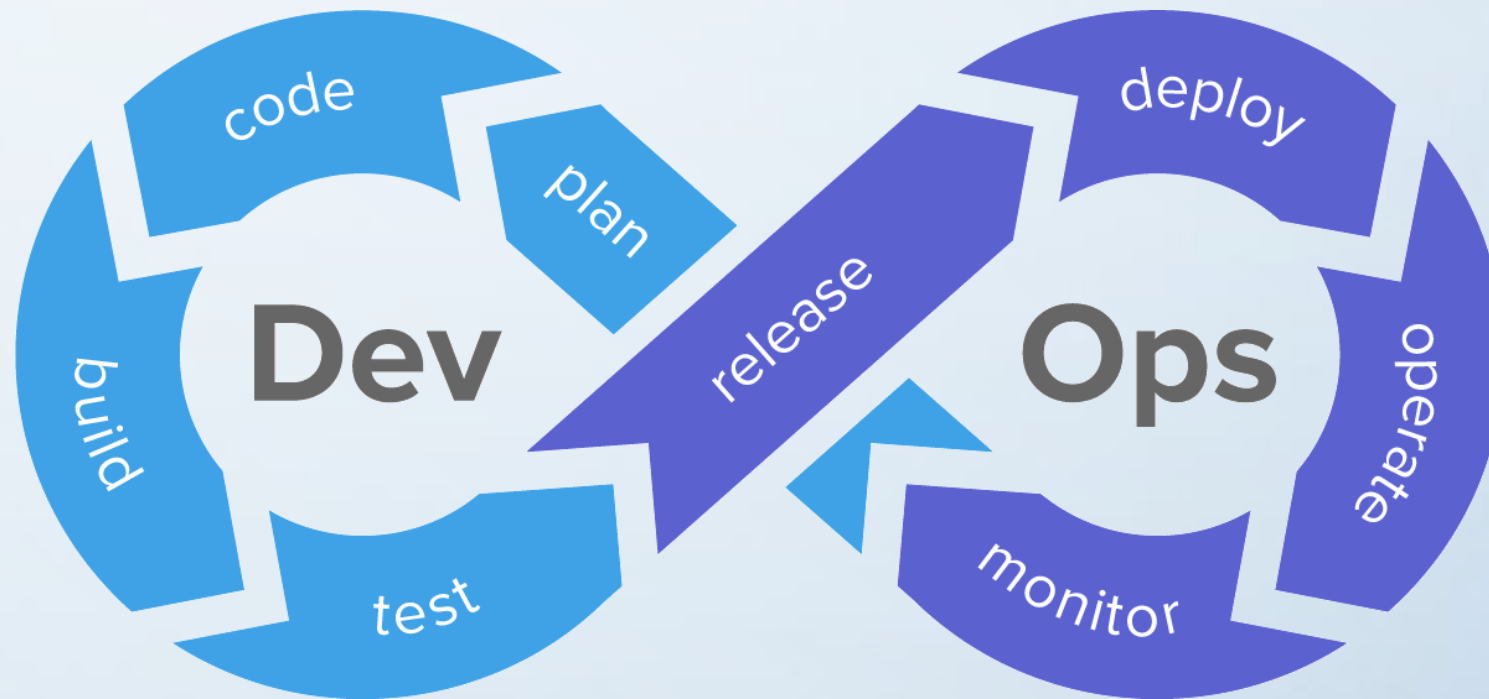




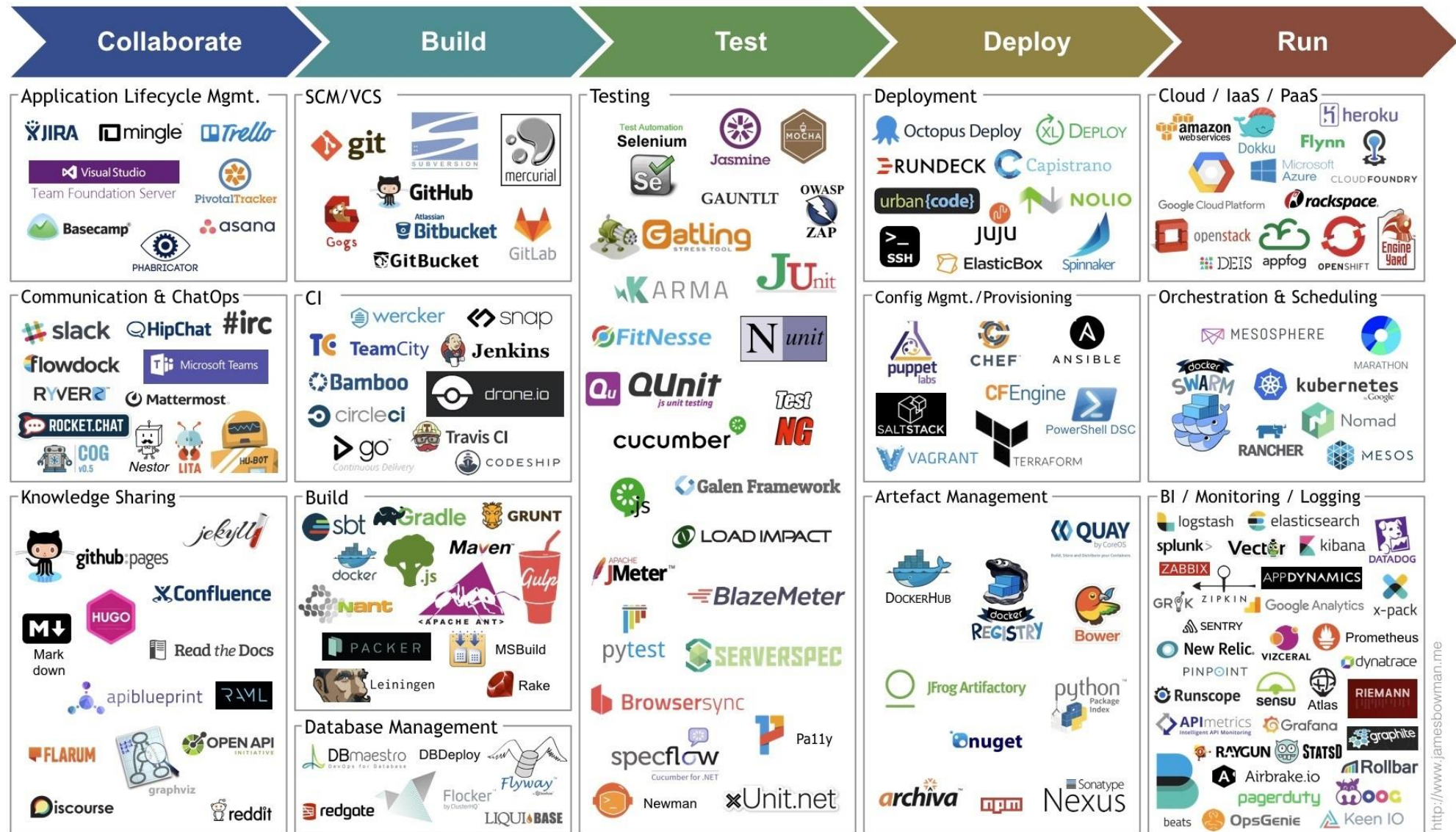
Введение в Методологию DevOps

О чем курс?

Автоматизация разработки и эксплуатации программного обеспечения



Зачем этот курс?

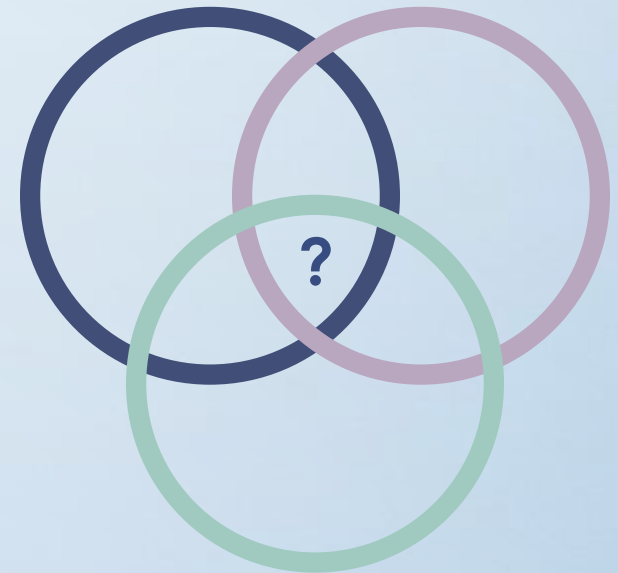
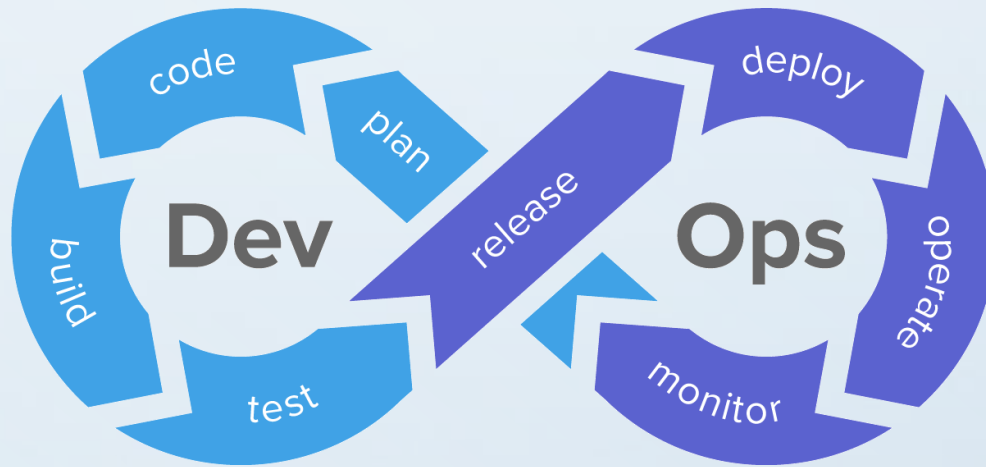


Часть 1 – DevOps

Что такое DevOps?

DevOps:

- **development** – разработка/развитие;
- **operations** – эксплуатация/использование.

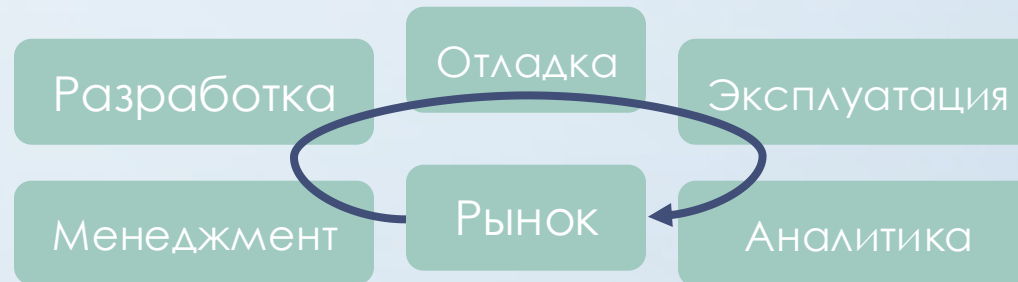


Почему DevOps?

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

-- *Melvin E. Conway*

[!] Время выхода на рынок - time-to-market

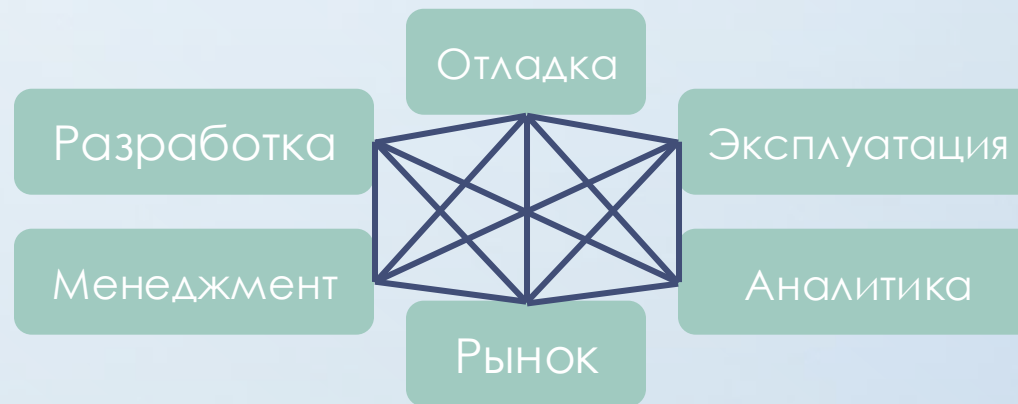


Почему DevOps?

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

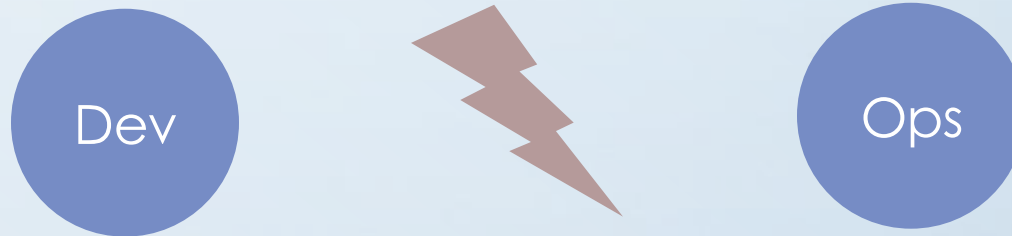
-- *Melvin E. Conway*

[!] Время выхода на рынок - time-to-market



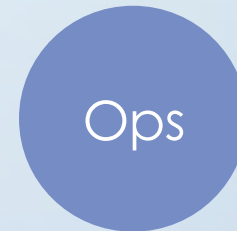
Зачем DevOps?

- **Dev** – Чаще отправлять изменения на продуктивную среду, не понимают как оно крутится на серверах.
- **Ops** – Реже отправлять изменения на продуктивную среду, меньше отказов, не понимают, как работает продукт.



Цели DevOps

- сокращение времени для выхода на рынок (time-to-market)
- снижение частоты отказов новых релизов
- сокращение времени выполнения исправлений
- уменьшение количества времени на восстановления



DevOps или SRE?

DevOps-инженер – квалифицированный специалист, отвечающий за автоматизацию всех этапов создания приложений и обеспечивает взаимодействие программистов и системных администраторов.

SRE - Site Reliability Engineering

Инженерные задачи VS Операционные задачи

Литература

O'REILLY®



Effective DevOps

BUILDING A CULTURE OF COLLABORATION,
AFFINITY, AND TOOLING AT SCALE

Jennifer Davis & Ryn Daniels

O'REILLY®



Site Reliability Engineering

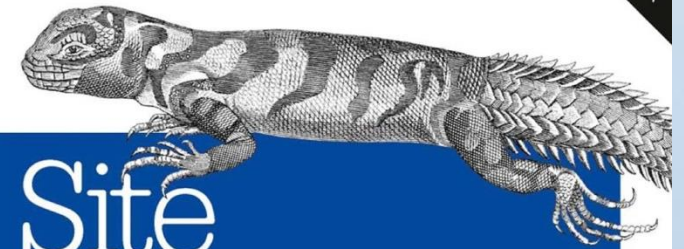
НАДЕЖНОСТЬ И БЕЗОТКАЗНОСТЬ КАК В GOOGLE

Бетси Бейер, Крис Джоунс,
Дженнифер Петофф,
Нейл Ричард Мёрфи

ПИТЕР®

O'REILLY®

Продолжение
бестселлера SRE Book



Site Reliability Workbook

ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

Бетси Бейер, Нейл Ричард Мёрфи,
Дэвид Рензин, Кент Кавахара, Стивен Торн

ПИТЕР®

Что нужно DevOps/SR-инженеру?

- знать жизненный цикл ПО и методологию
- освоить разные архитектуры ПО, в т.ч. микросервисную
- знать основы программирования, выучить несколько ЯП
- уметь отлаживать программы и устранять уязвимости и ошибки
- понимать принципы работы операционных систем
- понимать принципы работы компьютерных сетей
- понимать виртуализацию, облачные и гибридные решения
- разбираться в системах оркестрации
- разбираться в системах управления конфигурацией (СУК)
- уметь налаживать мониторинг продукта
- общаться с другими людьми и командами

Основные термины

- Система управления исходным кодом (SCCS)
- Инструменты сборки
- Системы управления конфигурацией (СУК)
- Непрерывная интеграция
- Непрерывная доставка
- Тестирование
- Мониторинг
- Оркестрация
- Облака
- IT-инфраструктура

Основные процессы

- Поэтапное обновление
- Выявление проблем
- Безопасно откатывать
- Резервирование
- Масштабируемость
- "Graceful degradation"
- Оценка SLA, SLO, SLI
- Бюджет ошибок
- Мониторинг

Зависимости

План эвакуации:

А-А-А-А!



Архитектура

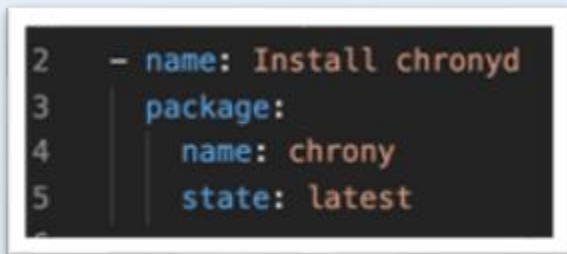
TFTDS - Theory of Fault-Tolerant Distributed Systems

- Горизонтальное масштабирование
- Резервирование
- Балансировка нагрузки

Инфраструктура как код (IaC)

Подход для управления и описания IT-инфраструктуры через конфигурационные файлы, а не через ручное редактирование конфигураций на серверах или интерактивное взаимодействие. Этот подход может включать в себя как декларативный способ описания инфраструктуры, так и императивный.

Примеры: **Ansible**, Chef, Saltstack, Puppet, Terraform, ...



```
2  - name: Install chronyd
3    package:
4      name: chrony
5      state: latest
```

Где размещать проект?

- На собственных серверах
- На виртуальных машинах в облаке
- В PaaS / Serverless

Часть 2 – Виртуализация и облачные решения

Ретроспектива

- Сервер с одним сервисом
- Сервер с набором сервисов
- Сервер с набором виртуальных машин
- Виртуальные машины в облаке
- Контейнерная виртуализация
- Оркестрация
- PaaS / Serverless

Виртуализация

Виртуализация — предоставление набора вычислительных ресурсов или их логического объединения, абстрагированное от аппаратной реализации, и обеспечивающее при этом логическую изоляцию друг от друга вычислительных процессов, выполняемых на одном физическом ресурсе.

Гипервизор - программа или аппаратная схема, обеспечивающая или позволяющая одновременное, параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере.

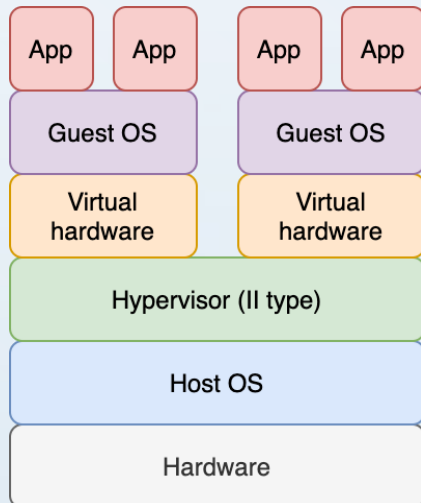
Классификация виртуализации

- Эмуляция
- Программная виртуализация
 - Трансляция команд
 - Паравиртуализация
- **Аппаратная виртуализация**
- **Контейнеризация**

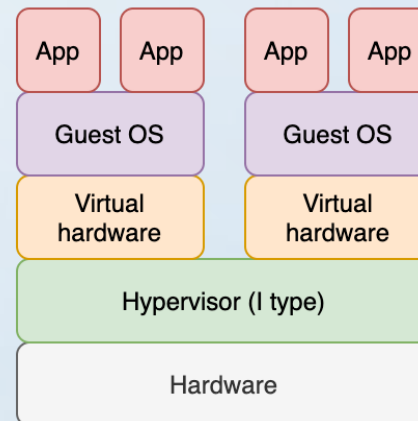
Гипервизоры

Задачи гипервизора:

- эмуляция виртуальных аппаратных ресурсов
- полная изоляция среды
- распределение физических аппаратных ресурсов



Гипервизор 2 типа
VirtualBox, VMware Workstation,
QEMU, Parallels, ...



Гипервизор 1 типа
VMware ESXi, Citrix XenServer

Облачные решения

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- IBM cloud computing
- VK CS (MCS)
- Yandex.cloud
- BASIS

 DASHBOARD

 SPACES

 CLOUD

Virtual machines

Computes


Disks

Disk Types

Images

 NETWORK






 MANAGED SERVICES

 SYSTEM

 API

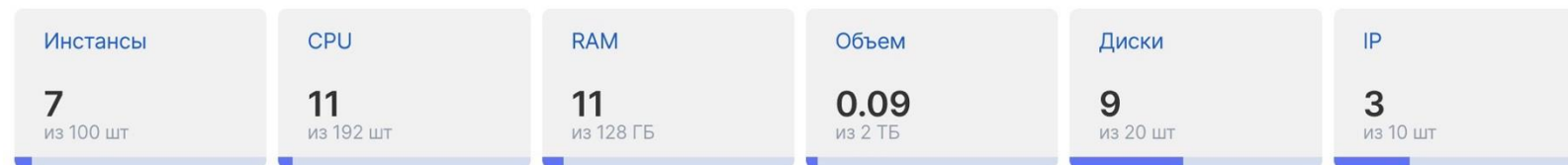
Dashboard Virtual machines x Computes x



<input type="checkbox"/>	ID 	Name 	Architectur	Status 	Tech Statu	Account ID	Account N	RG ID 	RG Name 	CPU	MEM	Disk Size
<input type="checkbox"/>	8297	laptev-vm	X86_64	ENABLED	STARTED	120	Vuz	1660	laptev-mi rea	4	4 096 MB	60 GB
<input type="checkbox"/>	8288	Ippo_vm_ 01	X86_64	ENABLED	STARTED	120	Vuz	1699	Ippo	2	1 024 MB	24 GB
<input type="checkbox"/>	8259	Lilo	X86_64	ENABLED	STARTED	120	Vuz	1675	km1	4	8 704 MB	63 GB
<input type="checkbox"/>	8220	terminus	X86_64	ENABLED	STARTED	120	Vuz	1711	Valery_H ozik_PUB	2	4 096 MB	20 GB
<input type="checkbox"/>	8186	st-is-1	X86_64	ENABLED	STARTED	120	Vuz	1703	VOR	4	4 096 MB	50 GB
<input type="checkbox"/>	8184	ghost	X86_64	ENABLED	STARTED	120	Vuz	1711	Valery_H ozik_PUB	1	1 024 MB	10 GB
<input type="checkbox"/>	8173	VM_ppa_u buntu	X86_64	ENABLED	STARTED	120	Vuz	1690	ppas	4	4 096 MB	50 GB
<input type="checkbox"/>	8172	ubnt	X86_64	ENABLED	STARTED	120	Vuz	1646	ST-001	4	4 096 MB	50 GB

Виртуальные машины

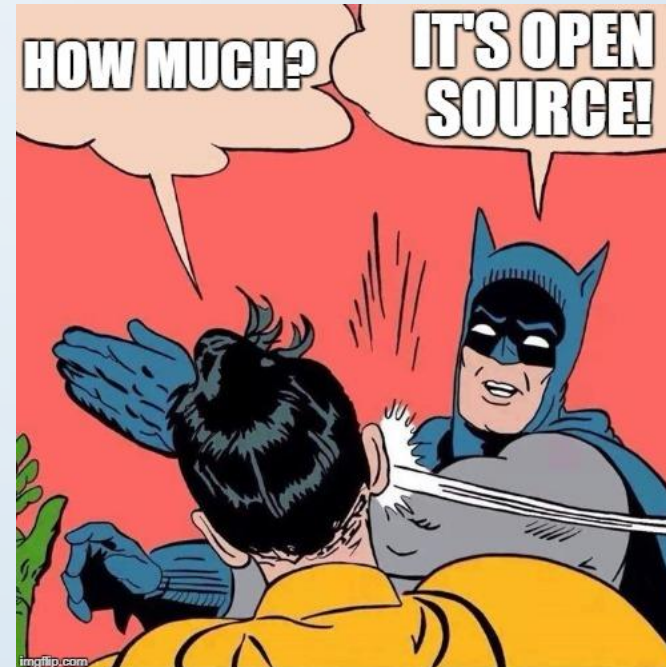
Скрыть квоты

☐ + Добавить Поиск по инстансам

<input type="checkbox"/>	Виртуальные машины ↕	IP-адреса ↕	Тип ↕	
<input type="checkbox"/>	centos79_Standard-2-2_10GB	10.10.56.208	Standard-2-2	...
<input type="checkbox"/>	euclide	10.0.0.21 10.10.83.12	Basic-1-1	...
<input type="checkbox"/>	gauss	10.10.83.10	Standard-2-2	...
<input type="checkbox"/>	overmind	10.10.83.1	Standard-2-2	...

Open source

- OpenStack
 - OpenNebula
 - CloudStack
 - Eucalyptus
- } IaaS



Project

API Access

Compute

Overview

Instances

Images

Key Pairs

Server Groups

Volumes

Container Infra

Network

Orchestration

Admin

Identity

Project / Compute / Overview

Overview

Limit Summary

Compute



Instances
Used 79 of 512



VCPUs
Used 427 of 612



RAM
Used 954GB of 1.3TB

Volume



Volumes
Used 59 of 200



Volume Snapshots
Used 1 of 200



Volume Storage
Used 3.8TB of 4.5TB

Network



Floating IPs
Allocated 107 of 384



Security Groups
Used 18 of 64



Security Group Rules
Used 74 of 256



Networks
Used 4 of 10



Ports
Used 111 of 512



Routers
Used 2 of 10

Usage Summary

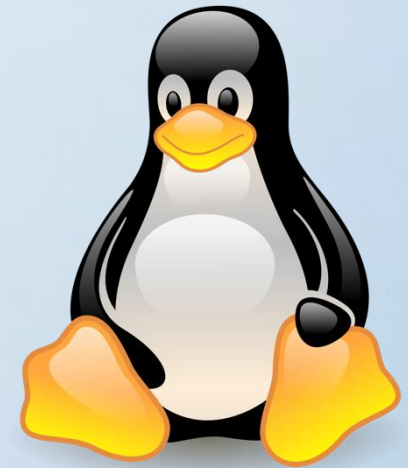
Часть 3 – GNU/Linux

GNU/Linux

GNU/Linux - семейство операционных систем на основе ядра Linux и программ проекта GNU. Не являются системами семейства Unix, однако работают по схожим принципам, частично соответствуют стандартам POSIX и признаются Unix-подобными.

Рекомендуем повторить:

syscall, процесс, поток, файловая система



Командная оболочка

Bash (от англ. Bourne again shell, каламбур «Born again» shell — «возрождённый» shell) — усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX.

\$# - общее количество параметров переданных скрипту
\$* - все аргументы переданные скрипту (выводятся в строку)
\$@ - аргументы выводятся в столбик
\$! - PID последнего запущенного в фоне процесса
\$\$ - PID самого скрипта

Советуем повторить:

- top, atop, htop
- ps, kill
- lsof, df, du, iostat
- tcpdump, netstat
- fg, bg, jobs

```
1  #!/bin/bash
2
3  var1=$1 - первый параметр скрипта
4  var2=$2 - второй параметр скрипта
5
6  if [[ "$var" -eq "substring" ]]
7  then
8      echo "Равно"
9  else
10     echo "Не равно"
11 fi
12
13 for i in {1..10}
14 do
15     echo "Номер $i"
16     echo 'Номер $i' # Просто текст
17 done
```

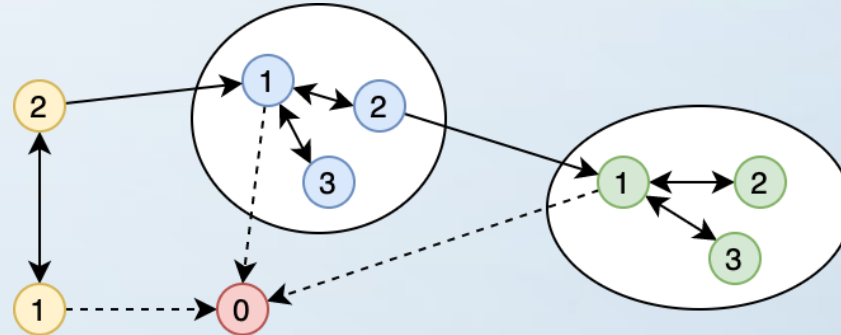
chroot

- Операция (системный вызов и утилита) изменения корневого каталога в Unix-подобных операционных системах.
- Простейший способ изоляции на уровне ФС

Namespaces

Namespace - механизм изоляции и группировки структур данных ядра.

Внутри **pid-ns** процессы нумеруются с 1, как на пустом сервере. Другие процессы вне пространства имен обнаружить нельзя.



Посмотреть можно в `/proc/$PID/ns`.

Namespaces - список

- `mount` - пространство ФС - копия дерева файловой системы, ассоциированная с процессом
- `uts` - пространство имени хоста и доменного имени
- `ipc` - пространство ресурсов межпроцессного взаимодействия
- `pid` - пространство номеров процессов, потомок внутри с PID 1 имеет родителя 0, т.е. невозможно понять иерархию снизу вверх, но можно сверху вниз.
- `network` - пространство имен сетевых настроек (интерфейсов, маршрутизации) - управляется через **`ip netns`** ...
- `user` - пространство номеров пользователей - можно заставить думать процесс, что он запущен от `root`.

CGroups

Control groups - механизм изоляции ресурсов ядра. Работает поверх **sysfs**. Описывают иерархию ресурсов.

sysfs – особая файловая система, позволяет создать иерархию объектной модели. Каталог - объект, файл внутри - атрибут. Можно читать/писать атрибуты, атрибуты могут быть составными.

Посмотрим список подсистем, которые могут ограничивать что-то: **ls /sys/fs/cgroup/**

CGroups - подсистемы

- **blkio** — устанавливает лимиты на чтение и запись с блочных устройств;
- **cpuacct** — генерирует отчёты об использовании ресурсов процессора;
- **cpu** — обеспечивает доступ процессов в рамках контрольной группы к CPU;
- **cpuset** — распределяет задачи в рамках контрольной группы между процессорными ядрами;
- **devices** — разрешает или блокирует доступ к устройствам;
- **freezer** — приостанавливает и возобновляет выполнение задач в рамках контрольной группы
- **hugetlb** — активирует поддержку больших страниц памяти для контрольных групп;
- **memory** — управляет выделением памяти для групп процессов;
- **net_cls** — помечает сетевые пакеты специальным тэгом, что позволяет идентифицировать пакеты, порождаемые определённой задачей в рамках контрольной группы;
- **netprio** — используется для динамической установки приоритетов по трафику;
- **pids** — используется для ограничения количества процессов в рамках контрольной группы.

CGroups - cpuset

Создать новую группу в подсистеме cpuset:

```
mkdir /sys/fs/cgroup/cpuset/group0
```

Добавим процесс нашего текущей командной оболочки в группу, смотрим доступные процессу логические ядра:

```
echo $$ > /sys/fs/cgroup/cpuset/group0/tasks  
cat /proc/$$/status | grep '_allowed'
```

Привяжем процессы группы к 0-му ядру, смотрим доступные процессу ядра:

```
echo 0 > /sys/fs/cgroup/cpuset/group0/cpuset.cpus  
cat /proc/$$/status | grep '_allowed'
```

CGroups - memory

Теперь создадим новую группу в подсистеме memory, добавим туда наш шелл и ограничим 40 MiB:

```
mkdir /sys/fs/cgroup/memory/group0  
echo $$ > /sys/fs/cgroup/memory/group0/tasks  
echo 40M > /sys/fs/cgroup/memory/group0/memory.limit_in_bytes
```

Спасибо за внимание!