

Модели управления инфраструктурой

Не забудь включить запись!



План

- Ручная конфигурация
- Типы серверов
- Инфраструктура как код (IaC)
- Immutable infrastructure
- Terraform - IaC

Ручная конфигурация

Самый простой подход к управлению конфигурацией -- это ручная конфигурация:

Для каждого изменения мы заходим на сервер и выполняем команды в терминале и правим конфиги прямо на сервере.

Ручная конфигурация

Обычно это выглядит как-то так:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv  
9DA31620334BD75D9DCB49F368818C72E52529D4  
echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.0  
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.0.list  
sudo apt-get update  
sudo apt-get install -y mongodb-org  
sudo apt-get install -y vim  
vim /etc/mongod.conf  
sudo systemctl restart mongod
```

Проблемы? Конечно

- Трудно избежать ошибок
 - Ой, забыл перезапустить...
 - А я забыл добавить репозиторий с mongodb и установил старую версию из репозитория ОС...
 - А я сделал как обычно, а оно не заработало...

А что если

```
$ cd /tmp && wget https://https://github.com/express42/reddit/archive/monolith.zip \  
&& tar -C /home/appuser/monolith -zxvf monolith.zip && rm -f monolith.zip
```

```
Resolving https (https)... failed: nodename nor servname provided, or not known.  
wget: unable to resolve host address 'https'
```

Система в *непонятном* состоянии



У нас не сработала одна из команд в длинной цепочке и система осталась в непонятном состоянии.

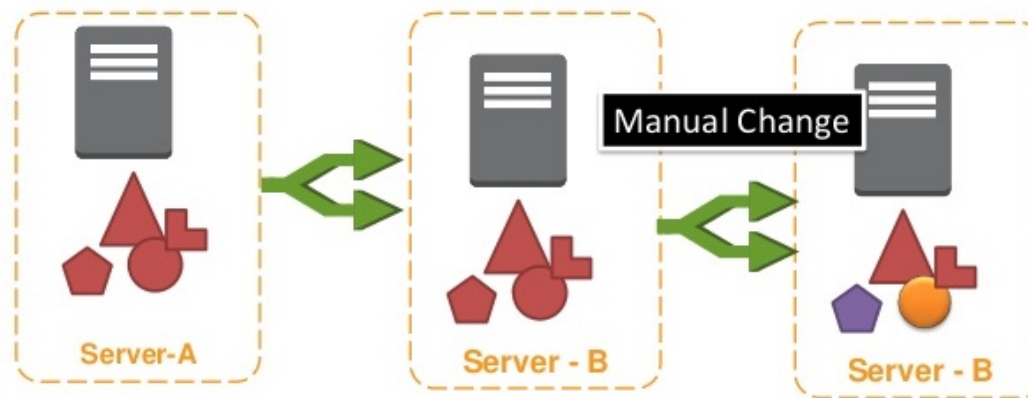
Приходится внимательно проследить за тем, какие изменения применились. И либо откатить их или понять как выполнить оставшиеся команды.

Отсутствует знание о состоянии

Допустим мы внимательно по большому чеклисту применяем команды и всё вроде сработало.

А потом понадобился срочный фикс и не было времени отметить его в этом чеклисте и конфигурация разошлась с задуманной

Настройки серверов различаются



В итоге, из-за того, что случаются ошибки и отсутствует знание о состоянии возникает проблема с тем, что настройки серверов различаются, появляется так называемый Configuration Drift.

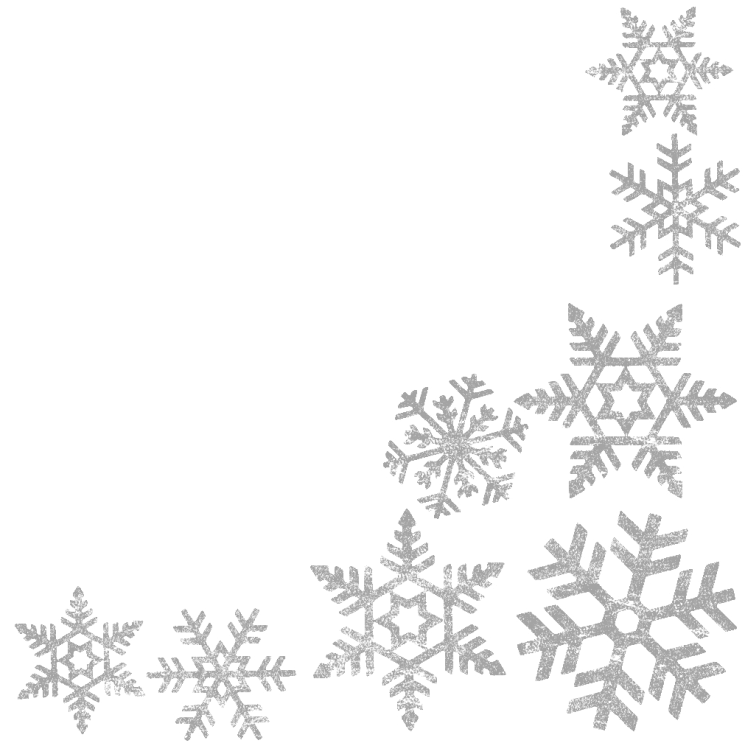
Проблемы ручного конфигурирования



- Отсутствуют знание о сделанных изменениях
- Отсутствует воспроизводимость
- Скучно! (пока не сломалось)
- И просто долго...

Серверы-снежинки vs

- Уникальные
- Сложные для понимания
- Хрупкие
- Неповторимые
- Мы любим их, знаем их характер и зовем по именам



... vs Феникс-серверы

- Предсказуемое состояние после перезагрузки
- Предсказуемое состояние после конфигурирования
- Да они все одинаковые! Давайте просто их пронумеруем



Методы управления конфигурацией

- Manual
- Scripts
- Infrastructure as Code
- Immutable Infrastructure

IaC

- Система контроля версий
- Единственный источник изменений
- Практики как для кода приложения
- Повторяемость

Какие преимущества это нам дает

- Качество документации
- Простота модификации
- Переиспользование и использование готовых решений
- Контроль над инфраструктурой
- Удобная работа в команде
- Идемпотентность

Immutable Infrastructure

В основе лежит понятие *базовых образов*, которые устанавливаются на сервер при его создании.

Для Baremetal:

- MaaS (Metal as a Service от Ubuntu)
- Razor от Puppet
- Еще собрать образ Packer и загрузить через PXE

Базовый образ

Что содержит базовый образ:

- Настройки ОС
- Пакеты, с низкой частотой изменения
- Логи, мониторинг, агенты систем управления конфигурацией

Immutable Infrastructure

- Образ VM - артефакт для деплоя
- Не изменяем запущенные инстансы
- Любые изменения: собираем новый образ
- Старый инстанс заменяем на новый







Fry

- Минимальный образ VM
- Настройка проходит уже после запуска
- Низкая скорость деплоя
- Проблемы с внешними репозиториями

Bake

- Включаем всё необходимое
- Настройки тоже включены
- Деплой ускоряется
- Деплой становится более надёжным
- Управление версиями

Hashicorp

DEVELOP	PROVISION		SECURE		RUN
					
Vagrant Create and configure portable development environments	Packer Create platform specific machine images from a single source	Terraform Create, combine and manage infrastructure across multiple providers	Vault Centrally store, secure and control access to distributed secrets	Nomad Cluster manager and scheduler to deploy applications across any infrastructure	Consul Distributed highly available tool for service discovery, configuration and orchestration

Vault



Позволяет безопасно хранить и управлять токенами, паролями и сертификатами и другими "чувствительными" данными используя UI, CLI, HTTP API

Vagrant



Позволяет создавать виртуальные машины и управлять ими

- Virtualbox
- VmWare
- Hyper-V
- Docker

Vagrant Vagrantfile

```
Vagrant.configure(2) do |config|

  config.vm.define "otus-devops" do |subconfig|
    subconfig.vm.box = "ubuntu/xenial64"
    subconfig.vm.hostname="otus-devops"
    subconfig.vm.network :private_network, ip: "10.1.2.10"
    subconfig.vm.provider "virtualbox" do |vb|
      vb.memory = "2048"
      vb.cpus = "2"
    end
  end
end

end
```

Consul



- Secure Service Communication
- Zero Downtime Deployments
- Dynamic Load Balancing

Terraform

Terraform

- Инструмент для декларативного описания инфраструктуры
- Описание инфраструктуры хранится в конфигурационных .tf-файлах
- State

HCL

- JSON-based синтаксис
- Блоки и аргументы
- Переменные
- Функции

Configuration language

```
provider "Имя_провайдера" {  
  Ключ = Значение  
}
```

```
resource "Тип_ресурса" "Имя ресурса" {  
  Ключ = Значение  
}
```

```
variable "Имя_переменной" {  
  type      = "Тип_переменной"  
  default   = "Значение"  
}
```

Configuration language

- variables
- - input
 - output
 - local
- types
- - simple
 - construction
- functions

Структура проекта

- .terraform/
- main.tf
- variables.tf
- outputs.tf
- terraform.tfstate
- terraform.tfstate.backup
- terraform.tfvars