KARPOV.COURSES >>>
КОНСПЕКТ

# > Конспект >8 урок > Сценарий развёртывания, часть 2

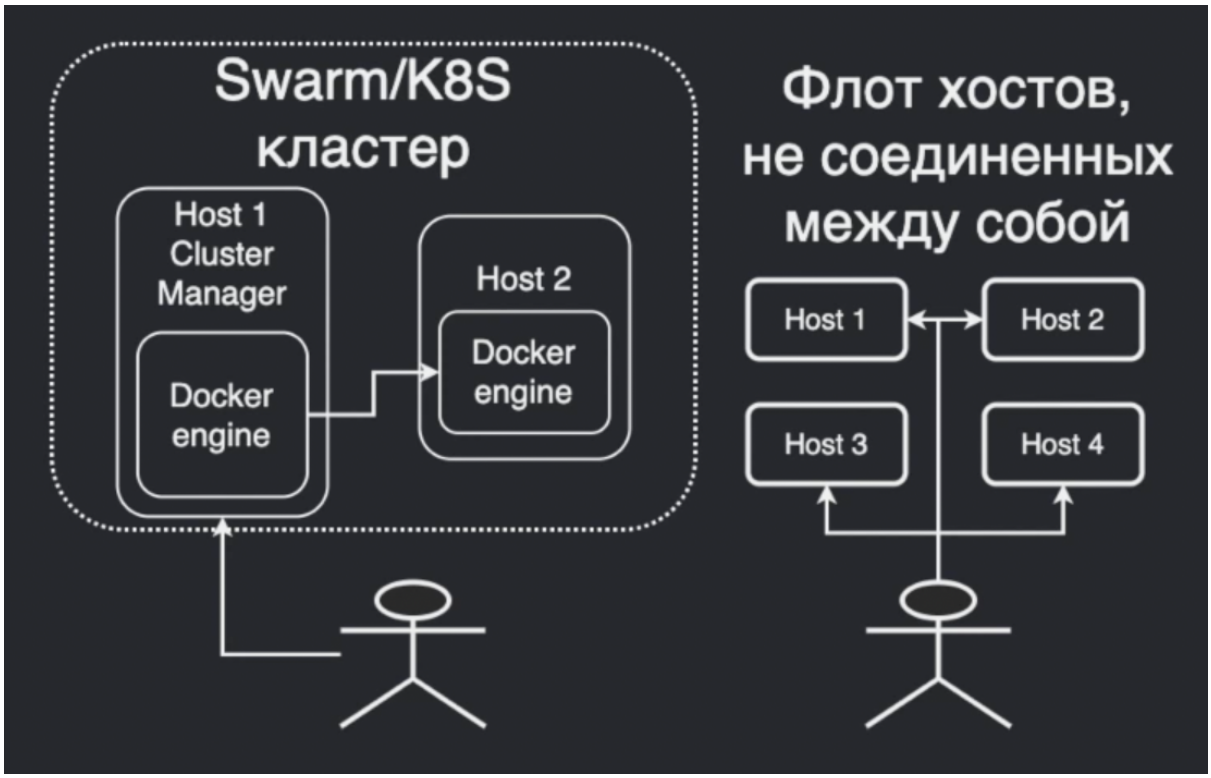
## > Оглавление


> Оглавление
> Bash
> Ansible
    > Сценарии
        > Переменные
        > Шаблоны
    > Резюме
> SaltStack
    > Характеристики
    > Конфигурации
    > Формулы
> Резюме


Материалы по Ansible из лекции: скачать.

## > Bash

Ранее в лекциях все действия выполнялись вручную и никакой автоматизации не предполагалось. На прошлом занятии мы говорили не о кластере, а о наборе машин, который часто называют флотом.




---

Рассмотрим кастомные сценарии, когда деплоить в кластер не подходит, даже когда он есть:

- Нет Docker и другого необходимого ПО;
- Коробочных сценариев развёртки и роллбэков недостаточно для логики;
- Влезает один инстанс аппликейшена;
- Есть корпоративные правила или другие ограничения.

---

Как правило, там, где есть кастомные сценарии, много `Bash`


У оболочек есть неоспоримые преимущества. Одно из них заключается в том, что они находятся близко к операционной системе.

```
DEFAULT_REMOTES=('kc1' 'kc2')

# Делаем список
REMOTES=${4:-${DEFAULT_REMOTES[*]}}


for REMOTE in ${REMOTES[*]}; do
    echo --------- Deploying to $REMOTE
    ssh $REMOTE "echo kek"
    # rsync ....
done
```

Типовой скрипт для управления деплойментами, команды выполняются синхронно

Добавим ожидание. Если добавить `&` в конце команды, то выполнение будет осуществляться в бэкграунд режиме, не блокируя цикл.

```
DEFAULT_REMOTES=('kc1' 'kc2', 'kc1' 'kc2',)

# Делаем список
REMOTES=${2:-${DEFAULT_REMOTES[*]}}


for REMOTE in ${REMOTES[*]}; do
    echo -------- Deploying to $REMOTE
    ssh $REMOTE "echo kek; sleep 5;"&
    # rsync ....
done
```

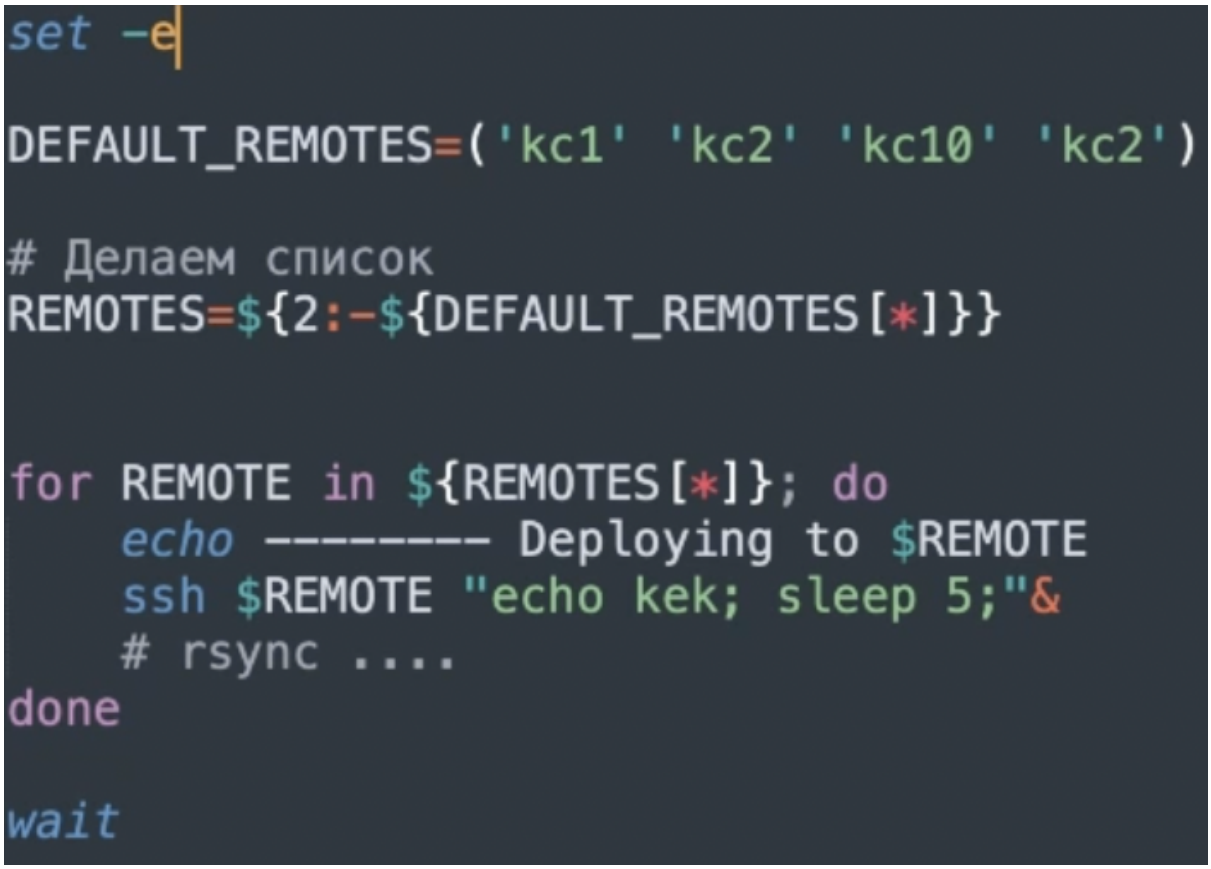
---

В данном случае управление сразу возвращается и неясно, всё ли выполнилось. Нужна минимальная синхронизация. Используем ключевое слово `wait`, которое будет ждать все бэкграунд таски.

```
DEFAULT_REMOTES=('kc1' 'kc2' 'kc1' 'kc2')

# Делаем список
REMOTES=${2:-${DEFAULT_REMOTES[*]}}


for REMOTE in ${REMOTES[*]}; do
    echo -------- Deploying to $REMOTE
    ssh $REMOTE "echo kek; sleep 5;"&
    # rsync ....
done

wait
```

---

Если в скрипте будет ошибка, то Bash её проигнорирует и продолжит выполнение. Для того чтобы сразу падать с ошибкой, необходимо указать `set -e`. К сожалению, это не работает для команд в бэкграунд режиме.



```
set -e

DEFAULT_REMOTES=('kc1' 'kc2' 'kc10' 'kc2')

# Делаем список
REMOTES=${2:-${DEFAULT_REMOTES[*]}}


for REMOTE in ${REMOTES[*]}; do
    echo --------- Deploying to $REMOTE
    ssh $REMOTE "echo kek; sleep 5;"&
    # rsync ....
done

wait
```

---

Bash имеет все языковые конструкции (кроме классов), которые позволяют реализовать необходимую логику. Например, можно написать функцию для проверки наличия нужного шелла на машине.

## > Ansible

Ansible — это agentless configuration management система, т.е. для управления системой и её конфигурирования не нужно использовать посредников и иметь установленных агентов.

Для того чтобы накатить конфигурацию, нужно минимум времени — малое время бутстрапа.

Пользователь декларативно описывает стейты, к которым нужно привести систему. Ansible составляет из этого команды, при запуске с клиента подключается по `ssh` к каждому из описанных таргетов и выполняет их.

Для конфигурирования таргетов необходимо использовать файл `/etc/ansible/hosts`. Таргеты можно разделить на группы, например `prod` и `test`.

Таски, которые должны выполняться последовательно на всех таргетах, в Ansible записываются в playbook — YAML файл.

После выполнения плейбука, Ansible проверяет, поменялся ли стейт.

Для Ansible каждый хост — отдельная машина, он не оперирует несколькими хостами связно.

---

## > Сценарии

Несколько плейбуков при помощи ролей могут быть объединены в сценарии. Для этого необходимо создать папку `roles`. Внутри находятся логические блоки. Плейбук должен быть в `main.yml` внутри подпапки `tasks`.
Файл `deploy.yml` объединит все блоки при помощи секции `roles`.

---

### > Переменные

Чтобы каждый раз явно не пробрасывать переменные в скрипты, можно в папке роли создать подпапку `vars`, где в `main.yml` будут перечислены определения.

---

### > Шаблоны

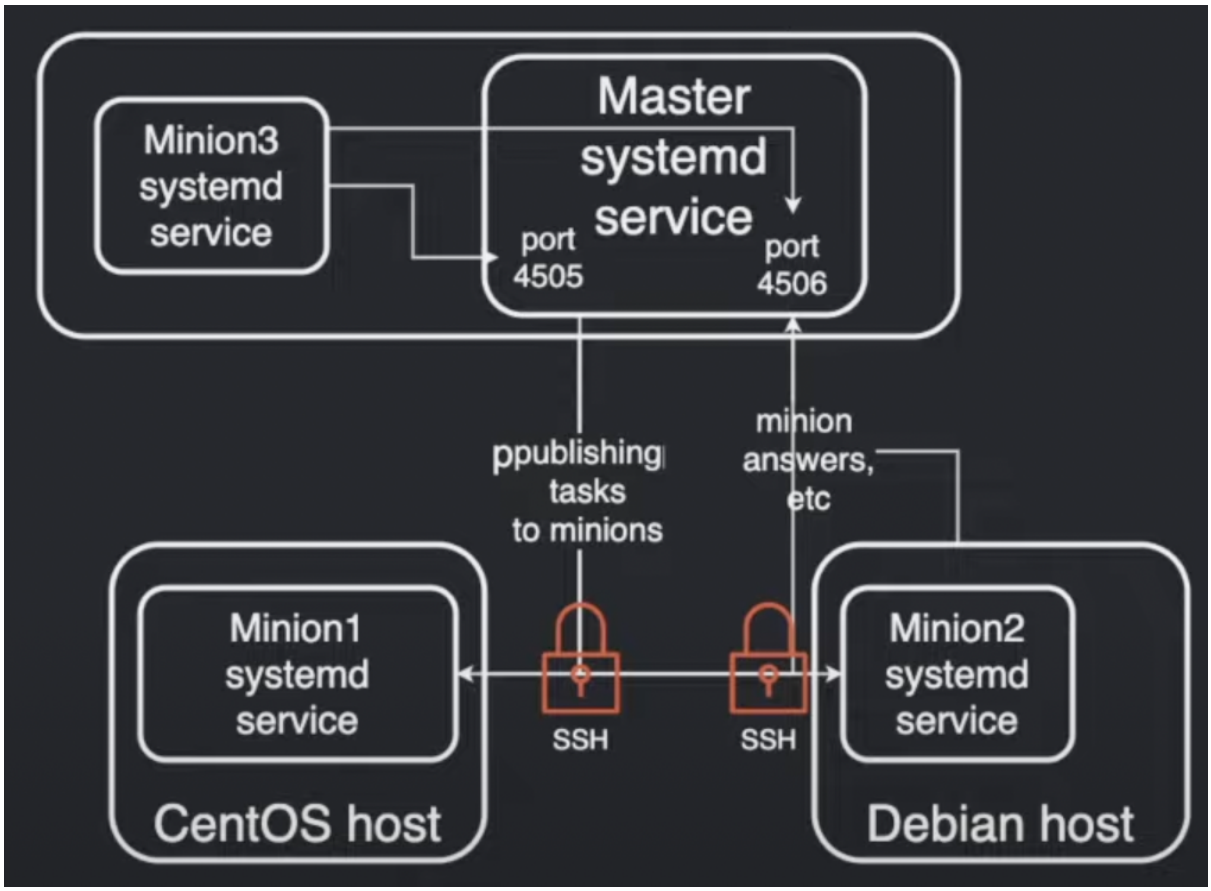
Внутри папки роли может быть подпапка `templates`, куда в формате `.j2` можно положить шаблоны, заполняющиеся переменными из `vars`. В тетрадке необходимо использовать языковую конструкцию Jinja2, которая во время рендеринга подтянет значения. Не забудьте про обрамление `%`, чтобы Jinja2 распознал подстановку.

---

## > Резюме

- Ansible в принципе позволяет делать всё, что можно написать на Bash;
- Благодаря требованиям к структуре поддерживается порядок;
- Недостатком является замедление работы при масштабировании по хостам.

# > SaltStack




Архитектура: будем работать с мастером, а миньоны будут подчиняться

На хосте мастера должны быть доступны следующие порты:

- `4505` — миньоны подключаются к мастеру и получают таски: мастер обходит флот по ssh и пытается заставить машины сделать то, что описано декларативно;
- `4506` — остальная коммуникация.

Команды, поступающие миньонам от мастера, раздаются всего лишь один раз, поэтому, хотя это и похоже на кластер, это трудно назвать таковым.

---

## > Характеристики

- По умолчанию не используются кэши и БД, но можно включить;
- Если нужно узнать состояние системы, необходимо выполнить команду, которая его соберёт;

- Вся работа на миньонах;
- Всё зашифровано;
- Порядок выполнения детерминирован.

---

Причины перехода с Ansible:

- Быстрота исполнения стендов;
- Считается, что проще писать конфиги.

---

## > Конфигурации

Конфигурации хранятся в `/etc/salt/`. Есть главные конфигурационные файлы, например `minion`. Из папки `minion.d` конфигурации будут подгружены `systemd` сервисом.

Важная папка — `pki`, в ней хранятся ключи для аутентификации мастера у миньонов, и наоборот. На мастере в `minions_pre` хранятся все ключи миньонов, которые запросили подключение к мастеру, но ещё не были авторизованы. Авторизация: перенос ключа миньона из `minions_pre` в `minions`. Вместо переноса вручную можно использовать `salt-key`.

В конфиге можно из миньонов сформировать группы (`nodegroups`), чтобы не перечислять каждый отдельно.

Передача переменных может быть организована при помощи настройки раздела `Pillar settings`.

---

## > Формулы

Формула — сторонний плагин (репозиторий). Формулы должны лежать в папке `srv/formulas`.

# > Резюме

Зачем нужны SaltStack и Ansible:

- Способ воспроизводимого конфигурирования машин;
- Автоматизация рутинных действий;
- Оркестрация флота машин;

- Большая гибкость;
- Устоявшиеся паттерны.