Конфигурационное управление ИТ-инфраструктурой

Балашов Антон

План лекции

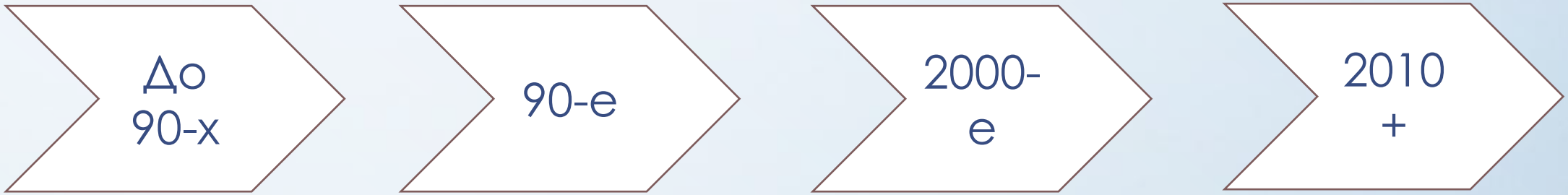
- Развитие ИТ-инфраструктуры
- Стандарты описания ИТ-инфраструктуры
- Подходы к управлению ИТ-инфраструктурой
- Инфраструктура как код
  - Ansible
  - Terraform

Особые обозначения

Наступать на грабли
совершать одинаковые ошибки снова и снова

Строить велосипед
Разрабатывать самому решение, уже реализованное другими, вместо того, чтобы им воспользоваться

Этапы развития



Мейнфреймы

- Клиентские ОС
- Локальные сети
- Интеграторы

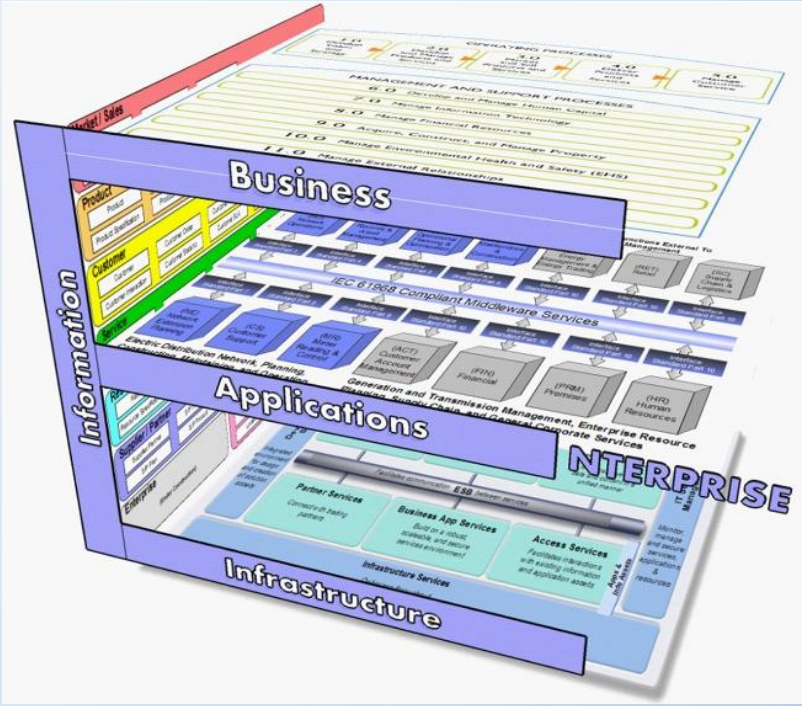
- Развитие интернета
- Дата-центры
- co-location
- Аутсорсинг

- облака
- сервис-провайдеры

Архитектура предприятия

(EA - Enterprise Architecture) определяет общую структуру и функции систем (бизнес и ИТ) в рамках всей организации в целом, включает в себя:

- общая модель (framework),
- стандарты и руководства для архитектуры уровня отдельных проектов,
- единое проектирование систем для обеспечения потребностей организации,
- взаимодействие и интеграция.



Архитектура предприятия

Архитектура предприятия описывает деятельность компании с двух основных позиций:

- **Бизнес-архитектура** описывает предприятие с позиции взаимодействия бизнес-процессов, правил и потоков информации.
- **Архитектура информационных технологий** описывает аппаратные и компьютерные средства, программное обеспечение, защиту и безопасность.

ИТ-архитектура

Включает в себя как логические, так и технические компоненты. Логическая архитектура предоставляет высокоуровневое описание миссии предприятия, его функциональных и информационных требований, системных компонентов и информационных потоков между этими компонентами.

- Enterprise Information Architecture (EIA) – информационная архитектура.
- Enterprise Solution Architecture (ESA) – архитектура прикладных решений.
- Enterprise Technical Architecture (ETA) – техническая архитектура.

Информационная архитектура (EIA - Enterprise Information Architecture)

В ходе разработки информационной архитектуры решаются следующие задачи:

- Идентификация существующих данных, определение их источников и процедур использования.
- Оптимизация данных за счет сокращения дублирования информации. Исключение неоднозначности и противоречивости информации.
- Минимизация перемещения данных за счет их оптимального расположения.
- Интеграция метаданных для обеспечения их целостного представления.
- Сокращение числа используемых технологий, обеспечивающих хранение и доступность информации.

Архитектура прикладных решений (ESA - Enterprise Solution Architecture)

Описание конкретных приложений:

- Компоненты и структура системы – внутренняя структура системы, включающая в себя информацию о программных модулях и базах данных.
- Взаимодействие с другими системами (интерфейсы) – описывает взаимодействие приложения с внешними объектами (программными продуктами, пользователями).

Архитектура прикладных решений (ESA - Enterprise Solution Architecture)

Классификация информационных систем в соответствии с их архитектурными стилями выделяет пять основных групп информационных систем:

- Приложения обслуживающие большое количество транзакций (**Transaction Processing**) - биллинговые, банковские системы.
- Операции в реальном времени (**Real-Time operations**) –системы, обеспечивающие бизнес процессы, требующие непрерывный мониторинг и информационное обеспечение, например, обеспечение транспортных операций в аэропорту.
- Аналитические приложения, бизнес-аналитика, поддержка принятия решений (**Analytical and Business Intelligence**) - управление знаниями, сбор и анализ больших массивов данных.
- Приложения поддержки совместной работы (**Collaborative**) - средства взаимодействия пользователей внутри компаниями.
- Корпоративные и обслуживающие приложения (**Utility**) –стандартные приложения, обеспечивающие функционирование основных бизнес-процессов компании: управление взаимоотношения с клиентами (CRM), управление ресурсами предприятия (ERP).

Архитектура прикладных решений (ESA - Enterprise Solution Architecture)

В соответствии с представленными выше критериями все ИС на предприятии можно разделить на следующие уровни критичности:

Level 1. Mission-Critical. Системы непрерывного действия для решения особо важных (критичных) задач. Сбой систем подобного уровня выводит из строя, парализует работу всего комплекса информационных систем или оказывает существенное влияние на функционирование компании.

Level 2. Business-Critical. Системы, критичные для бизнеса. Системы, обеспечивающие эффективное выполнение бизнес-процессов компании, но при этом не оказывающие прямого воздействия на них. Предприятие может функционировать без информационных систем этого уровня (т.к. подобные операции могут быть выполнены вручную), но, в случае их остановки, будет нести существенные финансовые потери.

Level 3. Business Operational. Системы, обеспечивающие функционирование бизнеса. Информационные системы данного уровня используются бизнесом для увеличения его эффективности, но при этом, их отключение на непродолжительное время не приведет к существенным финансовым потерям. Долгосрочное отключение этих систем будет влиять на эффективность бизнеса.

Level 4. Office Productivity. Системы внутреннего использования. К данному уровню относятся информационные системы, обеспечивающие эффективность выполнения офисных операций. Эти системы не являются важными для функционирования предприятия в целом, но необходимы для увеличения эффективности работы персонала.

Техническая архитектура предприятия (ETA - Enterprise Technical Architecture)

совокупность программно-аппаратных средств, методов и стандартов, обеспечивающих эффективное функционирование приложений:

- Информацию об инфраструктуре предприятия.
- Системное программное обеспечение (СУБД, системы интеграции).
- Стандарты на программно-аппаратные средства.
- Средства обеспечения безопасности (программно-аппаратные).
- Системы управления инфраструктурой.

Техническая архитектура предприятия (ETA - Enterprise Technical Architecture)

Gartner выделяет следующие основные сервисы, входящие в состав любой информационной архитектуры:

- **Сервисы данных**: системы управления базами данных, хранилища данных, системы поддержки принятия решений (Business Intelligence).
- **Прикладные сервисы**: языки программирования, средства разработки приложений, системы коллективной работы.
- **Программное обеспечение промежуточного слоя.**
- **Вычислительная инфраструктура**: операционные системы и аппаратное обеспечение.
- **Сетевые сервисы, локальные сети**: сетевое аппаратное обеспечение.
- **Сервисы безопасности, авторизация**: аутентификация, сетевая безопасность, физическая безопасность центров обработки данных

Техническая архитектура предприятия (ETA - Enterprise Technical Architecture)

Принципы построения ИТ инфраструктуры:

- Техническая инфраструктура масштабируется и расширяется
- Инфраструктура проста в эксплуатации и сопровождении
- Инфраструктура адекватна потребностям приложений и бизнеса
- Инфраструктура строится в строгом соответствии стандартам
- Стандартизация всех программно-аппаратных средств компании

Управление ИТ-подразделением как сервисом

Парадигма управления ИТ-инфраструктурой компании, основанная на:

SLA (соответствие обещаний поставщика услуги ожиданиям клиента) и

ITSM (IT Service Management, управление ИТ-услугами) - это концепция организации работы ИТ-подразделения и его взаимодействия с внешним или внутренним заказчиком, а также внешними контрагентами.

Управление ИТ-подразделением как сервисом

Концепция Управления ИТ-службами — **Information Technology Service Management (ITSM)**:

- формализация процессов функционирования информационных технологий;
- профессионализм и четкая ответственность сотрудников ИТ-отдела за определенный круг задач;
- **технологическая инфраструктура обеспечения качества услуг**:
  - собственно информационные технологии, служба поддержки пользователей;
  - служба управления конфигурациями и изменениями; - система контроля услуг;
  - служба тестирования и внедрения новых услуг и т.д.

ITIL **(IT Infrastructure Library)** — основа концепции управления ИТ-службами

Набор публикаций (библиотека), содержащий лучшие практики в области управления ИТ-услугами. ITIL содержит рекомендации по предоставлению качественных ИТ-услуг, процессов, функций, а также других средств, необходимых для их поддержки.

Структура ITIL основана на жизненном цикле услуги, который состоит из пяти стадий:

- стратегия,
- проектирование,
- преобразование,
- эксплуатация
- постоянное совершенствование

Также существуют дополнительные публикации, входящие в ITIL и содержащие специфичные рекомендации по индустриям, типам компаний, моделям работы и технологическим архитектурам

А в жизни…

Если принимать все положения как есть, без привязки к текущему положению, можно прийти к излишней формализации и значительному нарушению работы. Процесс внедрения принципов ITIL должен быть избирательным и адаптивным.

- Необходим анализ текущих процессов, а есть ли необходимость внедрения вообще, или же достаточно косметических изменений

- Если нет конечной четкой цели использования ITIL, то лучше не пытаться внедрять . Например- вопросы лицензирования ПО, а не просто «взять лучшие практики

Модели ИТ-инфраструктуры

- Локальная модель – все свое
- Гибридная модель:
  - **IaaS** (Infrastructure as a Service) – инфраструктура как услуга
  - **PaaS** (Platform as a Service) – платформа как услуга
  - **SaaS** (Software as a Service) – программное обеспечение как услуга

Модели ИТ-инфраструктуры

Локальная инфраструктура

Компания делает сама:

- Приложения
- Среда исполнения (ОС, фреймворк)
- Сервер и сеть, гипервизор
- Оборудование
- ЦОД

Модели ИТ-инфраструктуры

IaaS (Infrastructure as a Service) – инфраструктура как услуга.

Компания делает сама:

- Приложения
- Среда исполнения (ОС, фреймворк)

Поставщик услуги предоставляет:

- Сервер и сеть, гипервизор
- Оборудование
- ЦОД

«Каршеринг»

Модели ИТ-инфраструктуры

PaaS (Platform as a Service) – платформа как услуга

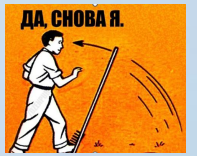
Компания делает сама:

- Приложения

Поставщик услуги предоставляет:

- Среда исполнения (ОС, фреймворк)
- Сервер и сеть, гипервизор
- Оборудование
- ЦОД

«Такси»



Модели ИТ-инфраструктуры

SaaS (Software as a Service) – программное обеспечение как услуга

Поставщик услуги предоставляет:

- Приложения
- Среда исполнения (ОС, фреймворк)
- Сервер и сеть, гипервизор
- Оборудование
- ЦОД

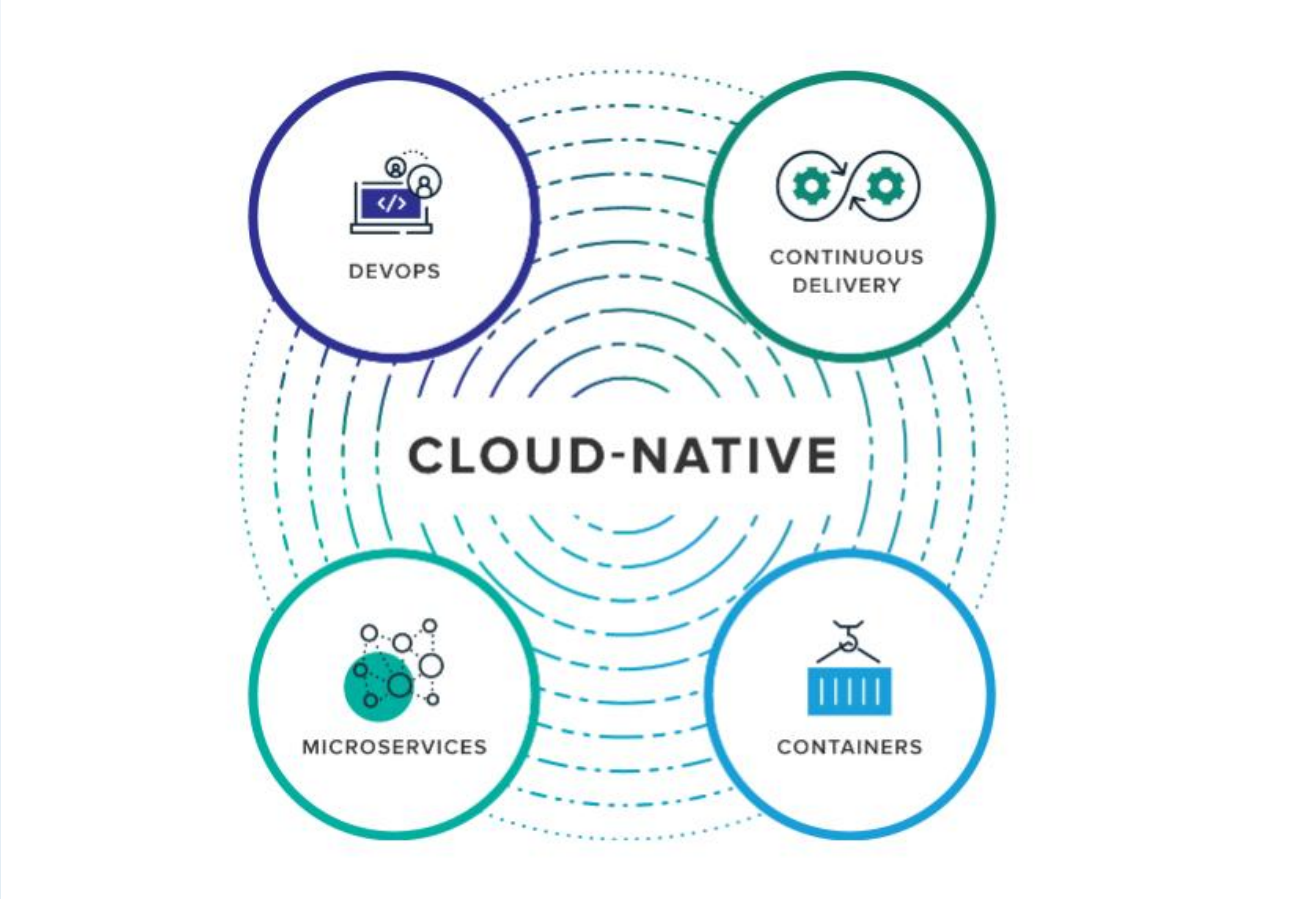
«Маршрутка»

Cloud native

Cloud native — подход к созданию и выполнению приложений, использующий преимущества облачной модели, подходит для частных и публичных облаков.

Обычно такие приложения строятся как набор микросервисов, слабо связанных между собой и упакованных в контейнеры, управляются они облачной платформой. Облачная платформа может предлагать вычислительные мощности по требованию.

Cloud native



Cloud native

DevOps
Реализация гибкой методологии разработки - постоянный и автоматизированный выпуск инкрементальных изменений программного обеспечения.

CI/CD
Релизы делаются чаще и с меньшими рисками за счет стандартизированных процедур. Быстрая обратная связь от пользователей.

Микросервисы.
Архитектурный подход к разработке приложения как набора небольших сервисов. Каждый сервис реализует определенную логику, его можно развернуть, обновить, масштабировать или перезапустить независимо от других служб приложения-нет необходимости выключать всю систему целиком при обновлении.

Контейнеры
Контейнеры эффективнее и быстрее стандартных виртуальных машин (ВМ). Используя виртуализацию на уровне операционной системы (ОС), один экземпляр ОС динамически распределяется между одним или несколькими изолированными контейнерами, у каждого из которых уникальная файловая система и свой объем выделенных ресурсов.

Цикл CI/CD

- **CI (Continuous Integration)** — непрерывная интеграция. Проверка основной ветки репозитория: каждый раз после мёржа в рамках CI-пайплайна выполняются автоматические тесты.
- **CD, (Continuous Delivery)** — непрерывная поставка - автоматическое развертывание на стенды и тестовые окружения.

Инструментальные средства автоматизации развертывания

- SSH (SCP) – свобода творчества
- Jenkins / Teamcity/Gitlab(Github) – CI/CD
- Urban code deploy - CD
- *Kubernetes – not today ☺*
- Infrastructure as code:
  - Ansible
  - Terraform

Инструментальные средства автоматизации развертывания

SSH (SCP)

scp опции пользователь1@хост1:файл пользователь2@хост2:файл

Опции:

-1 - использовать протокол SSH1;

-2 - использовать протокол SSH2;

-B - пакетный режим для передачи нескольких файлов;

-C - включить сжатие;

- l - установить ограничение скорости в кбит/сек;

-o - задать нужную опцию SSH;

-p - сохранять время модификации;

-r - рекурсивное копирование директорий;

-v - более подробный режим.

Инструментальные средства автоматизации развертывания

- Jenkins / Teamcity / Gitlab (Github)

Агенты внутри

Внешние Агенты

Инструментальные средства автоматизации развертывания

- Urban code deploy

urban{code}

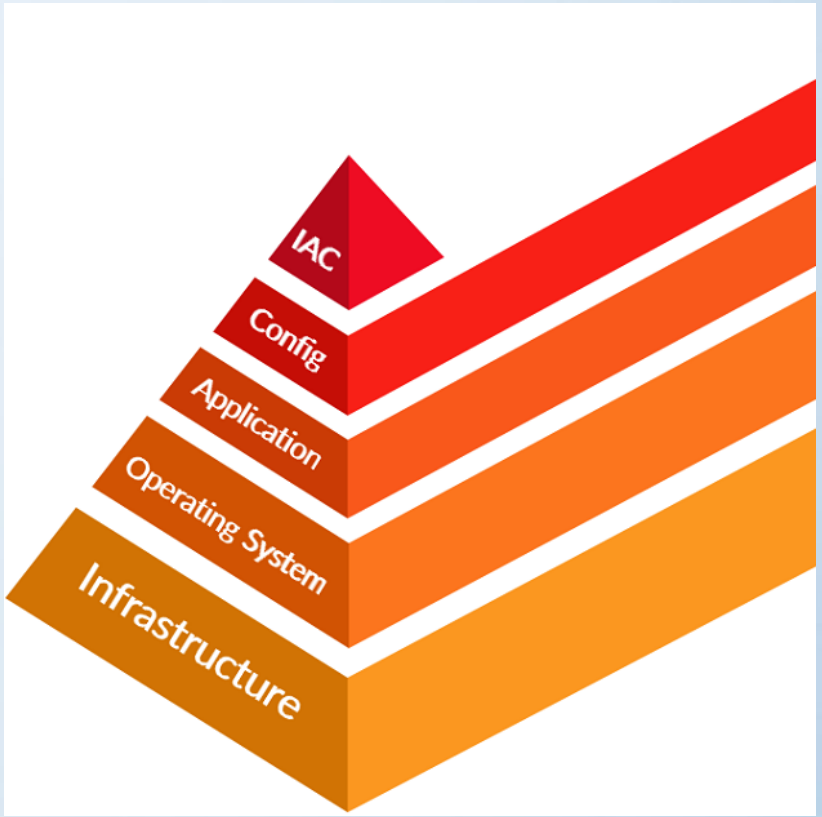
НЕ CI, только CD, требует установки агентов

UrbanCode Deploy - это решение для автоматизации выпуска приложений, совмещающее средства просмотра, отслеживания и контроля в одном оптимизированном пакете. Поддерживается масштабирование до развертывания уровня предприятия, включающего несколько тысяч серверов.

Infrastructure as Code

IAC, или **Infrastructure as Code**, представляет собой процесс подготовки (provisioning) и управления компьютерными центрами обработки данных с помощью машиночитаемых файлов определений, а не физической конфигурации оборудования или интерактивных инструментов конфигурации.

Хотя область IAC является относительно новой по сравнению с конвейером автоматизации DevOps, уже существует достаточно много IAC-инструментов, и новые технологии продолжают развиваться даже в этот самый момент.



Преимущества

- **Скорость и уменьшение затрат:** IaC позволяет быстрее конфигурировать инфраструктуру и направлен на обеспечение прозрачности, чтобы помочь другим командам со всего предприятия работать быстрее и эффективнее. Это освобождает дорогостоящие ресурсы для выполнения других важных задач.

- **Масштабируемость и стандартизация:** IaC предоставляет стабильные среды быстро и на должном уровне. Командам разработчиков не нужно прибегать к ручной настройке - они обеспечивают корректность, описывая с помощью кода требуемое состояние сред. Развертывания инфраструктуры с помощью IaC повторяемы и предотвращают проблемы во время выполнения, вызванных дрейфом конфигурации или отсутствием зависимостей. IaC полностью стандартизирует инфраструктуру, что снижает вероятность ошибок или отклонений.

- **Безопасность и документация:** Если за создание всех вычислительных, сетевых и служб хранения отвечает код, они каждый раз будут развертываться одинаково. Это означает, что стандарты безопасности можно легко и последовательно применять в разных компаниях. IaC также служит некой формой документации о правильном способе создания инфраструктуры и страховкой на случай, если сотрудники покинут компанию, обладая важной информацией. Поскольку код можно версионировать, IaC позволяет документировать, регистрировать и отслеживать каждое изменение конфигурации вашего сервера.

- **Восстановление в аварийных ситуациях:** IaC — чрезвычайно эффективный способ отслеживания инфраструктуры и повторного развертывания последнего работоспособного состояния после сбоя или катастрофы любого рода.

Недостатки

- **Логика и соглашения:** необходимо понимать IaC скрипты независимо от того, написаны ли они на языке конфигурации HashiCorp (HCL) или на обычном Python или Ruby

- **Обслуживаемость и возможность отслеживания:** хотя IaC предоставляет отличный способ отслеживания изменений в инфраструктуре и мониторинга, обслуживание сетапа IaC само по себе становится проблемой при достижении определенного масштаба. Когда IaC широко используется в организации с несколькими командами, отслеживание и управление версиями конфигураций не так просты, как может показаться на первый взгляд.

- **RBAC:** Основываясь на нем, управление доступом тоже становится сложной задачей. Установка ролей и разрешений в различных частях организации, которые внезапно получают доступ к скриптам для быстрого развертывания кластеров и сред, может оказаться довольно сложной задачей.

- **Запаздывание фич:** Инструменты IaC, не зависящие от поставщика (например, Terraform), часто запаздывают с фичами в сравнении с продуктами, привязанными к конкретному поставщику. Это связано с тем, что поставщикам инструментов необходимо обновлять провайдеров, чтобы полностью охватить новые облачные фичи, выпускаемые с постоянно растущими темпами.

Инструменты IAC

Имя	Описание	Синтаксис	Лицензия	Сайт	GitHub репозиторий
Terraform	Terraform — это программный IAC-инструмент, созданный HashiCorp. Он известен как декларативный provisioning-инструмент без агентов и без мастера.	.tf файл (похож на JSON)	MPL 2.0	terraform.io	github.com/hashicorp/terraform
Ansible	Поддерживаемый Red Hat, Ansible — это программный IAC-инструмент, вмещающий в себе provisioning, управление конфигурацией и развертывания приложений.	YAML	GPL 3.0	ansible.com	github.com/ansible/ansible
Chef	Chef автоматизирует процесс управления конфигурациями, гарантируя, что каждая система правильно настроена и согласована.	Ruby	Apache 2.0	chef.io/products/chef-infra	github.com/chef/chef
Puppet	Puppet — это инструмент управления конфигурацией программного обеспечения, который имеет собственный декларативный язык для описания конфигурации системы.	Язык Puppet (похож на JSON) или Ruby	Apache 2.0	puppet.com	github.com/puppetlabs/puppet
SaltStack	Поддерживаемый VMWare, SaltStack — это программное обеспечение с открытым исходным кодом на основе Python для управляемой событиями (event-driven) IT-автоматизации, удаленного выполнения задач и управления конфигурацией.	Python	Apache 2.0	repo.saltproject.io	github.com/saltstack/salt
Pulumi	IAC SDK с открытым исходным кодом Pulumi позволяет создавать, развертывать и управлять инфраструктурой в любом облаке, используя ваши любимые языки.	Различные языки программирования	Apache 2.0	pulumi.com	github.com/pulumi/pulumi

Agent vs Agentless

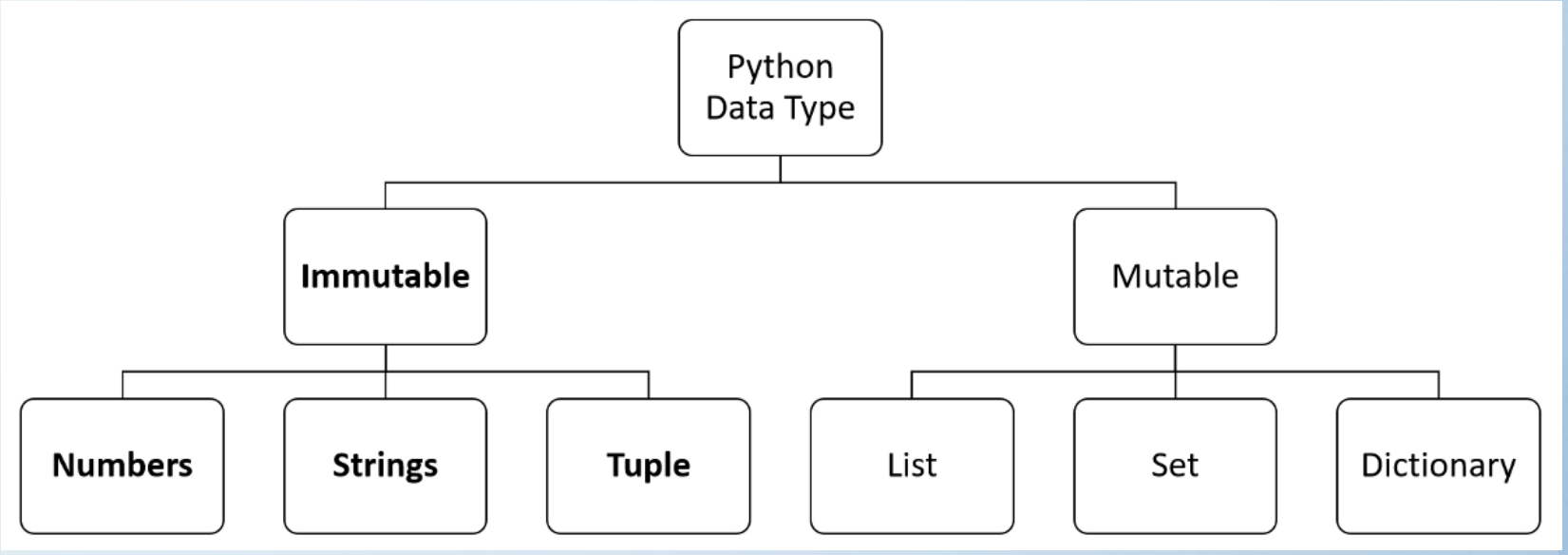
IAC-инструмент может требовать запуска агента на каждом сервере, который вы хотите настроить.

Если нет, то такой инструмент называется «agentless».

Mutable vs Immutable

Некоторые инструменты, такие как Terraform, занимаются не изменением уже подготовленной (provisioned) инфраструктуры, а развертывают новый сервер, что означает, что они следуют парадигме неизменяемой (immutable) инфраструктуры.

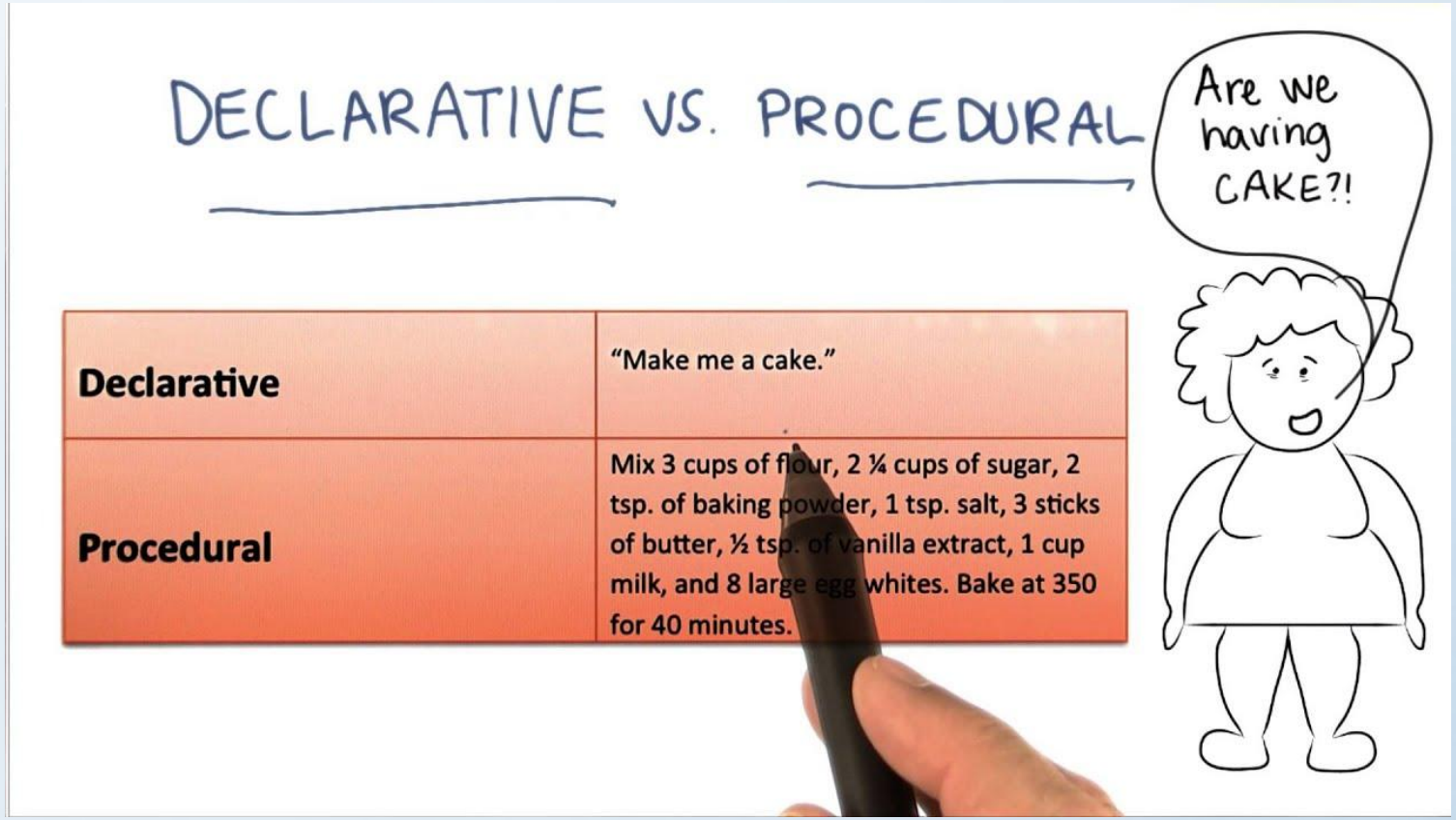
Другие инструменты, такие как Ansible, Chef, SaltStack и Puppet, могут изменять существующие ресурсы, а это означает, что эти инструменты следуют парадигме изменяемой (mutable) инфраструктуры.



Procedural vs Declarative

Процедурные языки, такие как Ansible и Chef, позволяют описывать с помощью кода поэтапное выполнение.

Декларативные языки, такие как Terraform, SaltStack и Puppet, позволяют просто указать желаемое состояние.



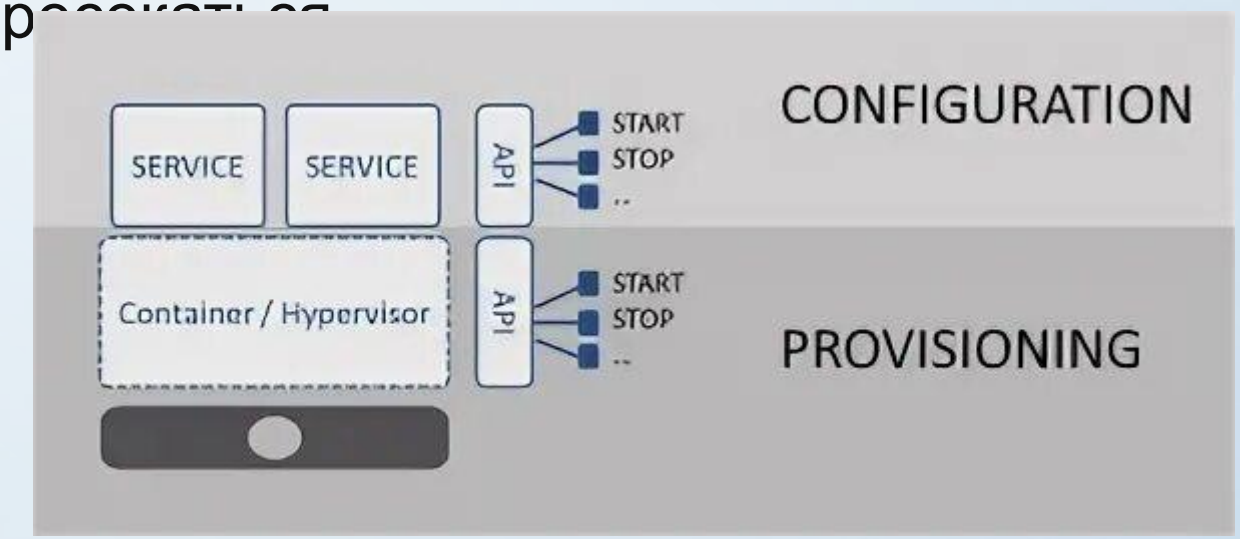
Master vs Masterless

Языки, такие как Chef, требуют, чтобы вы запускали отдельный главный сервер (master), чтобы обеспечить дополнительное управление и постоянные состояния.

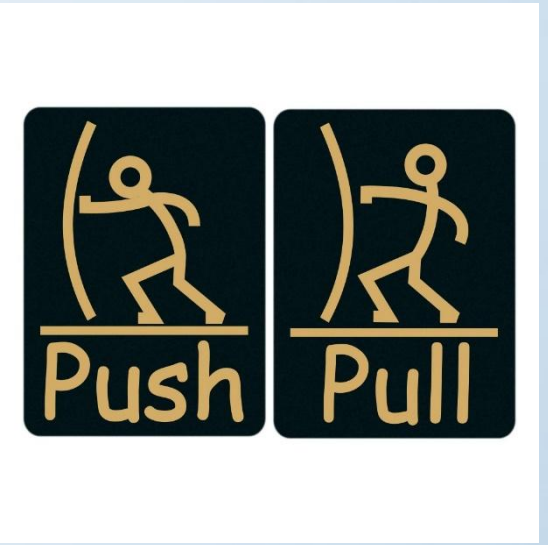
Другие языки, такие как Ansible и Terraform, не нуждаются в определении мастера.

Configuration vs Provisioning

- Ansible, Chef, SaltStack и Puppet известны как инструменты управления конфигурацией, что означает, что их основная цель — настроить ресурсы. Другие инструменты, такие как Terraform и Pulumi, являются инструментами обеспечения (provisioning), а это означает, что их основная цель — предоставлять ресурсы. Однако по мере того, как инструменты развиваются, их функционал может начать пере[illegible]



Pull & Push



В основе подхода pull лежит тот факт, что все изменения применяются изнутри кластера. Внутри кластера есть оператор, который регулярно проверяет связанные репозитории Git и Docker Registry. Если в них происходят какие-либо изменения, состояние кластера обновляется изнутри. Обычно считается, что подобный процесс весьма безопасен, поскольку ни у одного внешнего клиента нет доступа к правам администратора кластера.

В push-подходе внешняя система (преимущественно CD-пайплайны) запускает развертывания в кластер после коммита в Git-репозиторий или в случае успешного выполнения предыдущего CI-пайплайна. В этом подходе система обладает доступом в кластер.

Chef

В отличие от Terraform или Ansible, Chef требует установки мастер-сервера и запустили агентов на серверах, с которыми вы хотите работать. Наличие мастера и агентов может иметь несколько преимуществ.

Мастер-сервер может быть центральным местом, из которого можно управлять и контролировать инфраструктуру.

Наличие агентов может гарантировать правильную работу обновлений. Однако, агенты и мастер-сервер также предполагают дополнительное обслуживание большую уязвимость. Когда вы выбираете IAC для начала, вы должны убедиться, какой инструмент лучше всего подходит для вашей ситуации.

Файл конфигурации Chef основан на языке программирования Ruby, который может дать вам некоторую гибкость, если вы хотите попробовать дополнительную логику управления.

Puppet

- **Puppet** — это инструмент управления конфигурацией программного обеспечения с декларативным синтаксисом, требующий наличия мастер-сервера и агентов. Puppet — это инструмент с открытым исходным кодом, имеющий лицензию **Apache 2.0**. Язык программирования Puppet — это декларативный язык, который описывает состояние компьютерной системы в терминах «ресурсов», которые представляют собой базовые конструкции сети и операционной системы. Пользователь собирает ресурсы в манифесты, которые описывают желаемое состояние системы.

SaltStack

SaltStack, также известный как **Salt,** это event-driven ПО с открытым исходным кодом на основе Python для автоматизации IT, удаленного выполнения задач и управления конфигурацией. Основой Salt является механизм удаленного выполнения, который создает высокоскоростную, безопасную и двунаправленную сеть связи для групп систем. Помимо этой системы связи, Salt предоставляет чрезвычайно быструю, гибкую и простую в использовании систему управления конфигурациями, называемую Salt States. Вы также должны познакомиться с такими терминами Salt,
как **grains**, **pillars** и **mine**.

Pulumi

Pulumi — это IAC-инструмент с открытым исходным кодом для создания, развертывания и управления облачной инфраструктурой. Хотя Pulumi — самый молодой инструмент из этого списка, в последнее время он набирает огромную популярность. Самым уникальным аспектом этого инструмента является то, что вы можете написать код конфигурации на любом из следующих языков программирования: Python, Typescript, Javascript, C# или Go.

Ansible

Программное решение для удаленного управления конфигурациями. Оно позволяет настраивать удаленные машины. Главное его отличие от других подобных систем в том, что Ansible использует существующую инфраструктуру SSH.

Особенности:

- **Безагентное**. В клиенте не установлено программное обеспечение или агент, который общается с сервером.
- **Идемпотентное**. Независимо от того, сколько раз вы вызываете операцию, результат будет одинаковым.
- **Простое и расширяемое**. Программа Ansible написана на Python и использует YAML для написания команд. Оба языка считаются относительно простыми в изучении.

Ansible

Состав:

Control machine — управляющий хост. Сервер Ansible, с которого происходит управление другими хостами

Manage node — управляемые хосты

Inventory — инвентарный файл. В этом файле описываются хосты, группы хостов, а также могут быть созданы переменные

Playbook — файл сценариев

Play — сценарий (набор задач). Связывает задачи с хостами, для которых эти задачи надо выполнить

Task — задача. Вызывает модуль с указанными параметрами и переменными

Module — модуль Ansible. Реализует определенные функции

Ansible

Установка:

Требования к управляющему хосту:

- поддержка Python 3)
- Windows не может быть управляющим хостом

pip install ansible

Ansible

Группы серверов

Список групп серверов для управления, два способа получения:

- Локальный файл: **/etc/ansible/hosts**
- Внешний скрипт, в официальном github-репозитории есть готовые скрипты для получения списка из:
  - Digital Ocean,
  - OpenStack Nova,
  - Openshift,
  - Vagrant,
  - Zabbix и др

Ansible

Hosts-файл

/etc/ansible/hosts – по-умолчанию, но может быть задано переменной окружения $ANSIBLE_HOSTS или параметром -i при запуске ansible и ansible-playbook.

Пример:

[group1]

one.example.com

two.example.com

[group2]

three.example.com

Помимо списка управляемых узлов, в файле hosts могут быть указаны и другие сведения, необходимые для работы: номера портов для подключения по SSH, способ подключения, пароль для подключения по SSH, имя пользователя, объединения групп

Ansible

Hosts-файл

Можно добавить диапазон хостов:

[routers]

192.168.255.[1:5]

Или по имени

[switches]

switch[A:D].example.com

Ansible

Hosts-файл, дочерние группы

[routers]

192.168.255.[1:5]

[switches]

switch[A:D].example.com

[devices:children]

routers

switches

Ad-hoc команды

ansible 192.168.0.1 -i hosts.ini -c network_cli –k -u user -m ios_command -a "commands='sh clock'"

- 192.168.0.1 – хост, к которому нужно применить действия, должен существовать в инвентарном файле
- -i myhosts.ini - параметр -i позволяет указать инвентарный файл
- -c network_cli - тип подключения. В данном случае через SSH
- -u user – выполнить от имени пользователя
- -k - аутентификация по паролю
- -m ios_command – используемый модуль
- -a "commands='sh ip int br'" – команда для выполнения

Ansible

Модули

В состав Ansible входит большое количество модулей для развёртывания, контроля и управления различными компонентами:

- облачные ресурсы и виртуализация (Openstack, libvirt);
- базы данных (MySQL, Postgresql, Redis, Riak);
- файлы (шаблонизация, регулярные выражения, права доступа);
- мониторинг (Nagios, monit);
- оповещения о ходе выполнения сценария (Jabber, Irc, почта, MQTT, Hipchat);
- сеть и сетевая инфраструктура (Openstack, Arista);
- управление пакетами (apt, yum, rhn-channel, npm, pacman, pip, gem);
- система (LVM, Selinux, ZFS, cron, файловые системы, сервисы, модули ядра)

Ansible Playbooks

Playbook (файл сценариев) – файл с описанием действий для выполнения на хосте и группе хостов.

Структура:

- play – сценарий, состоит из описания, конфигурации и списка задач
- task - конкретная задача реализуемая в рамках сценария. В задаче должно быть:
  - описание (название задачи можно не писать, но очень рекомендуется)
  - модуль и команда (действие в модуле)

Ansible Playbooks

```
- name: Install aptitude and required packs
  hosts: workers
  tasks:
    - name: Install aptitude
      apt:
        name: aptitude
        state: latest
        update_cache: true

    - name: Install required system packages
      apt:
        pkg:
          - apt-transport-https
          - ca-certificates
          - curl
          - software-properties-common
        state: latest
        update_cache: true
    - name: Add Docker GPG apt Key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present
```

Ansible Playbooks

Include

позволяют подключать в текущий playbook файлы с задачами.

```
---

- name: Install aptitude and required packs
  hosts: workers
  tasks:
    - name: Install aptitude
      apt:
        name: aptitude
        state: latest
        update_cache: true
    - include: tasks/install_apt.yml
```

Ansible

Переменные

Переменные можно создавать:

- в inventory
- в playbook
- в файлах для групп/хостов
- в отдельных файлах, которые добавляются в playbook через include
- в ролях
- передавать при вызове playbook

Ansible

Переменные в inventory

[routers]

192.168.255.[1:5]

[routers:vars]

ansible_connection=network_cli

ansible_user=user

ansible_password=password

Ansible

Переменные в playbook

```
- name: Run command on host
  hosts: workers
  gather_facts: false
  vars:
    cmd: cat /file
  tasks:
    - name: run command
      command: "{{cmd}}"
```

Ansible

Переменные в файлах групп, хостов

Во время развертывания, необходимо не только установить приложение, но и настроить его в соответствии с определенными параметрами на основании принадлежности к группе серверов или индивидуально :

- Файлы переменных групп в директории “group_vars/имя_группы”;
- Файлы переменных хостов в директории “hosts_vars/имя_хоста”;
- Файлы с переменными в директории “имя_роли/vars/имя_задачи.yml”;

```
├── group_vars _
│   ├── all.yml
│   ├── routers.yml
│   └── switches.yml
├── host_vars
│   ├── 192.168.0.1
│   ├── 192.168.0.2
└── myhosts.ini
```

Ansible

Роли

Роли это способ логического разбития файлов Ansible, это просто автоматизация выражений include- не нужно явно указывать полные пути к файлам с задачами или сценариями, а достаточно лишь соблюдать определенную структуру файлов

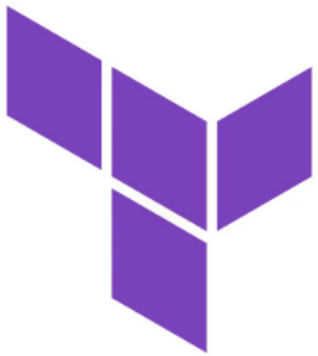
```
├── all_roles.yml
├── role1.yml
├── role2.yml
└── roles
    ├── role1
    │   ├── files
    │   ├── templates
    │   ├── tasks
    │   ├── handlers
    │   ├── vars
    │   ├── defaults
    │   └── meta
```

Ansible

Роли

- Все роли определены в каталоге roles
- Дочерние каталоги называются именем ролей
- Внутри каталога роли могут быть предопределенные каталоги - как минимум, tasks.
- Внутри каталогов tasks, handlers, vars, defaults, meta автоматически считывается всё, что находится в файле main.yml
- Файлы из каталогов добавляются через include, на файлы s можно ссылаться не указывая путь к ним, достаточно имени файла

Terraform



HashiCorp
Terraform

Infrastructure automation tool

Open-source and vendor agnostic

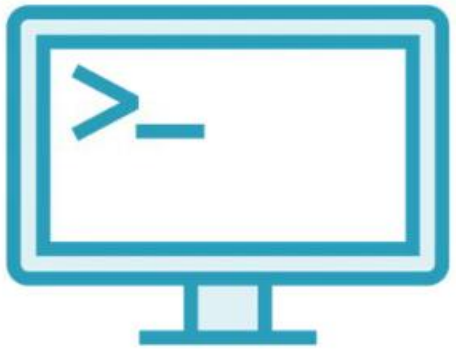
Single binary compiled from Go

Declarative syntax

HashiCorp Configuration Language or JSON

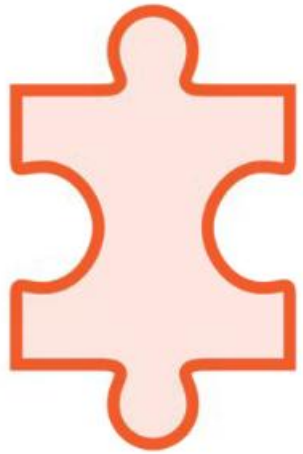
Push based deployment

Core Components

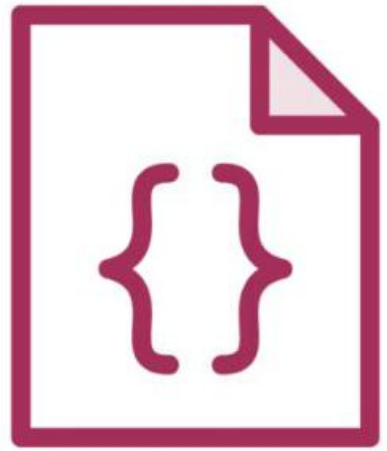


Executable

Configuration files



Provider plugins



State data

Terraform executable

SANCTIONS

Скачать с зеркала:

https://hashicorp-releases.yandexcloud.net/terraform/

export PATH=$PATH:/path/to/terraform

VK Cloud

Yandex Cloud

Hashicorp Configuration Language

- Для автоматизации работы с инфраструктурой Terraform использует собственный язык написания конфигурационных файлов Hashicorp Configuration Language (HCL). По сути, этот язык описывает желаемое состояние инфраструктуры в конфигурационном файле.
- Для описания того или иного создаваемого элемента необходимо подготовить блок, содержащий заключенные в фигурных скобках названия переменных, и их значения, передаваемые функциям.
- Как и в большинстве языков программирования, в HCL используются аргументы для присвоения значений переменным. В Terraform эти переменные являются атрибутами, связанными с определенным типом блока. Таким образом, весь код HCL состоит из подобных блоков.

Hashicorp Configuration Language

Хранение файлов

Все конфигурационные файлы Terraform должны размещаться в одном каталоге. Так, для того примера. По умолчанию Terraform предполагает, что все файлы с *.tf расширениями в данном каталоге являются частью конфигурации, независимо от имен файлов.

Для грамотной организации больших конфигурации потребуются три файла:

- variables.tf – для всех объявленных входных переменных.
- provider.tf – для объявленных поставщиков, которых вы используете.
- main.tf – для объявления фактических ресурсов, которые будут созданы.

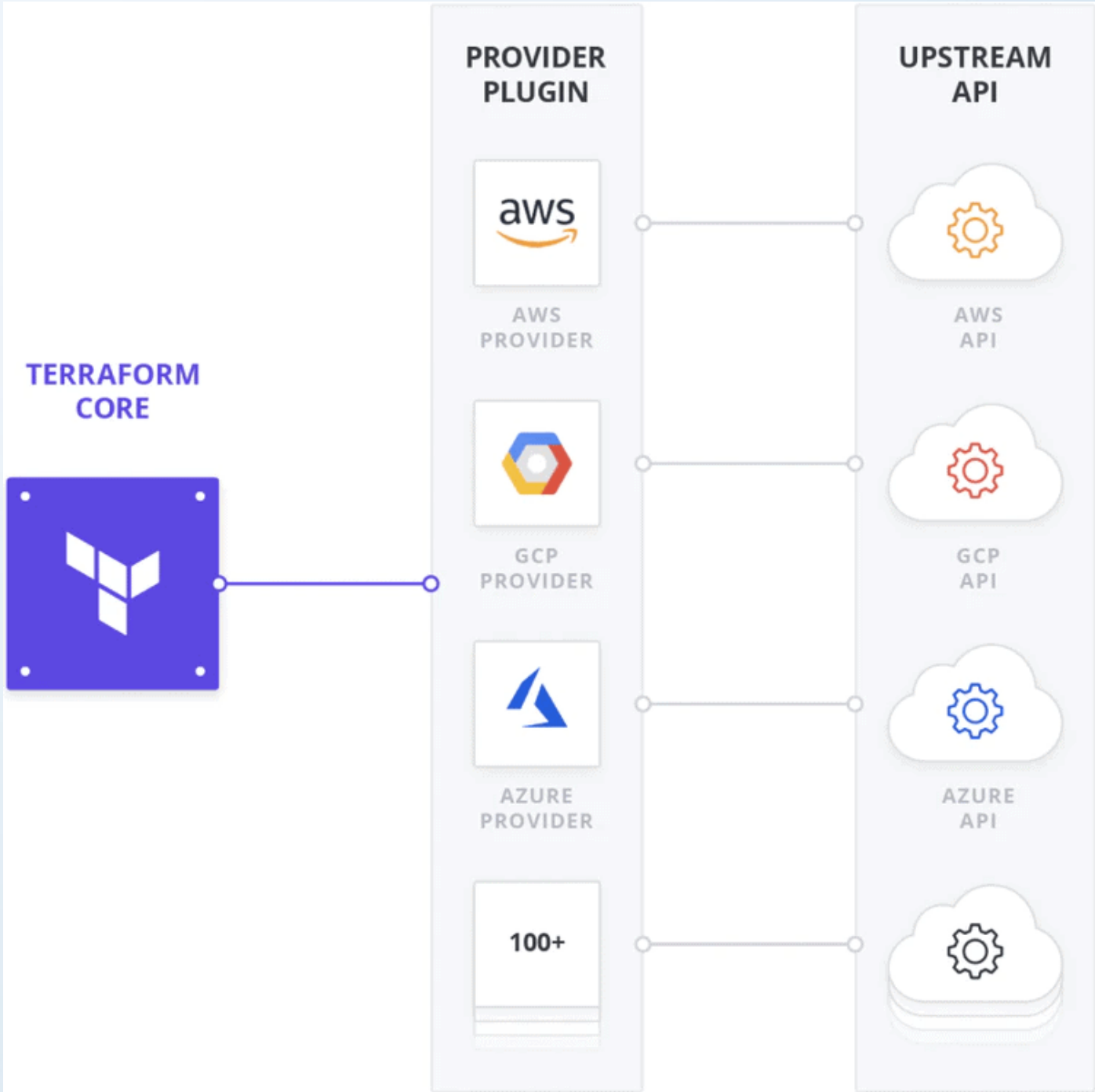
Однако, такая структура не является обязательной. Можно поместить весь код в один файл и все будет корректно работать.

Пример

```
terraform {
 required_providers {
   aws = {
     source  = "hashicorp/aws"
     version = "~> 3.0"
   }
 }
}
```

- В первой строке, открываем блок terraform, который хотя и не является обязательным с точки зрения синтаксиса системы, но рекомендуется его указывать.
- Вложенный блок required providers, определяет требуемых провайдеров. Нам потребуется поставщик aws с указанными параметрами source и version.

Providers


PROVIDER PLUGIN
UPSTREAM API
TERRAFORM CORE
aws
AWS PROVIDER
AWS API
GCP PROVIDER
GCP API
AZURE PROVIDER
AZURE API
100+

State

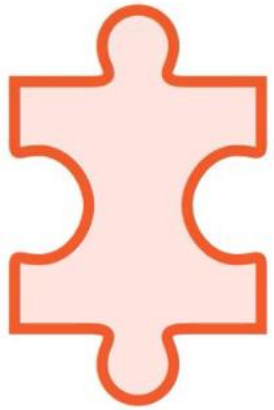
Состояние Terraform - это файл, в котором хранятся все сведения о ресурсах, которые были созданы в контексте данного проекта.

Например, если объявить ресурс azure_resourcegroup и запустить Terraform, в файле состояния будет сохранен его идентификатор.

Основная цель файла состояния - предоставить информацию об уже существующих ресурсах, поэтому, когда изменяются файлы конфигурации ресурсов, Terraform может определить, что ему нужно делать.

Важным моментом в отношении файлов состояния является то, что они могут содержать конфиденциальную информацию. Примеры включают начальные пароли, используемые для создания базы данных, закрытые ключи и так далее.Terraform использует концепцию серверной части для хранения и извлечения файлов состояния. Серверной частью по умолчанию является локальная серверная часть, которая использует файл в корневой папке проекта в качестве места хранения

Terraform Object Types



Providers

Resources

Data sources

Providers

Provider работает как драйвер устройства операционной системы. Он предоставляет набор типов ресурсов, используя общую абстракцию, таким образом маскируя детали того, как создавать, изменять и уничтожать ресурс.
Terraform автоматически загружает поставщиков из своего публичного реестра по мере необходимости, исходя из ресурсов данного проекта. Он также может использовать пользовательские плагины, которые должны быть установлены пользователем вручную.
За некоторыми исключениями, использование провайдера требует настройки его с некоторыми параметрами
Хотя это и не является строго необходимым, считается хорошей практикой явно указывать, какого поставщика мы будем использовать проекте Terraform, и указывать его версию:

```
provider "kubernetes" {
   version = "~> 1.10"
}
```

Resources

В Terraform ресурс – это все, что может быть целью для операций CRUD в контексте данного поставщика. Некоторыми примерами являются экземпляр EC2, Azure MariaDB или запись DNS.

```
resource "aws_instance" "web" {

        ami = "some-ami-id" instance_type = "t2.micro"

}
```

Сначала всегда есть ключевое слово resource, с которого начинается определение. Далее тип ресурса, который обычно соответствует типу провайдера. В приведенном выше примере aws_instance – это тип ресурса, определенный поставщиком AWS. После этого появляется определяемое пользователем имя ресурса, которое должно быть уникальным для этого типа ресурса в том же модуле – подробнее о модулях позже.

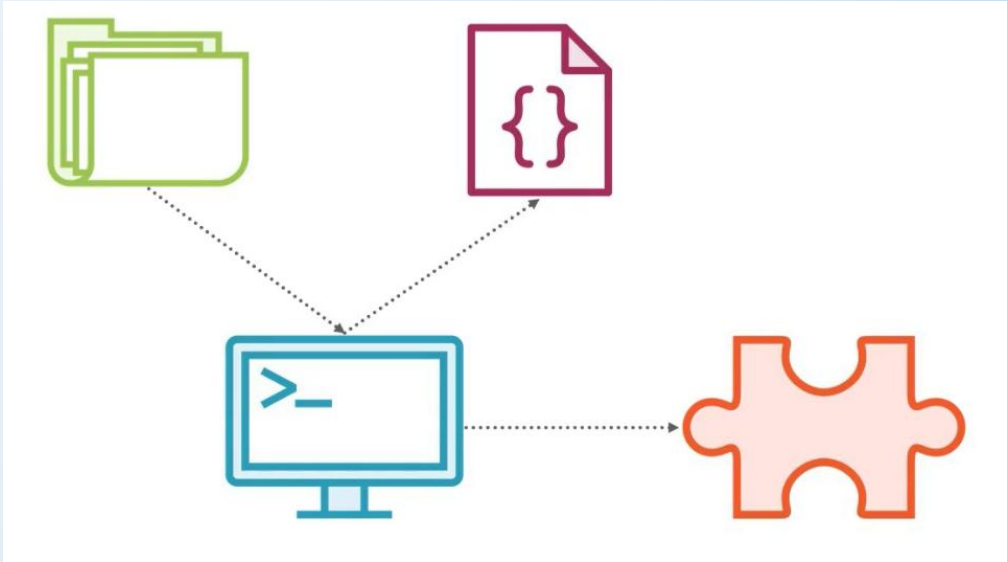
Data sources

```
data "aws_ami" "ubuntu" {

   most_recent = true

   filter {

      name = "name"

      values = ["ubuntu/images/hvm-ssd/ubuntu-
trusty-14.04-amd64-server-*"]

   }

   filter {

      name = "virtualization-type"

      values = ["hvm"]

   }

   owners = ["099720109477"] # Canonical

}
```

Источники данных работают в значительной степени как ресурсы, доступные только для чтения, в том смысле, что мы можем получать информацию о существующих, но не можем создавать или изменять их. Обычно они используются для извлечения параметров, необходимых для создания других ресурсов.

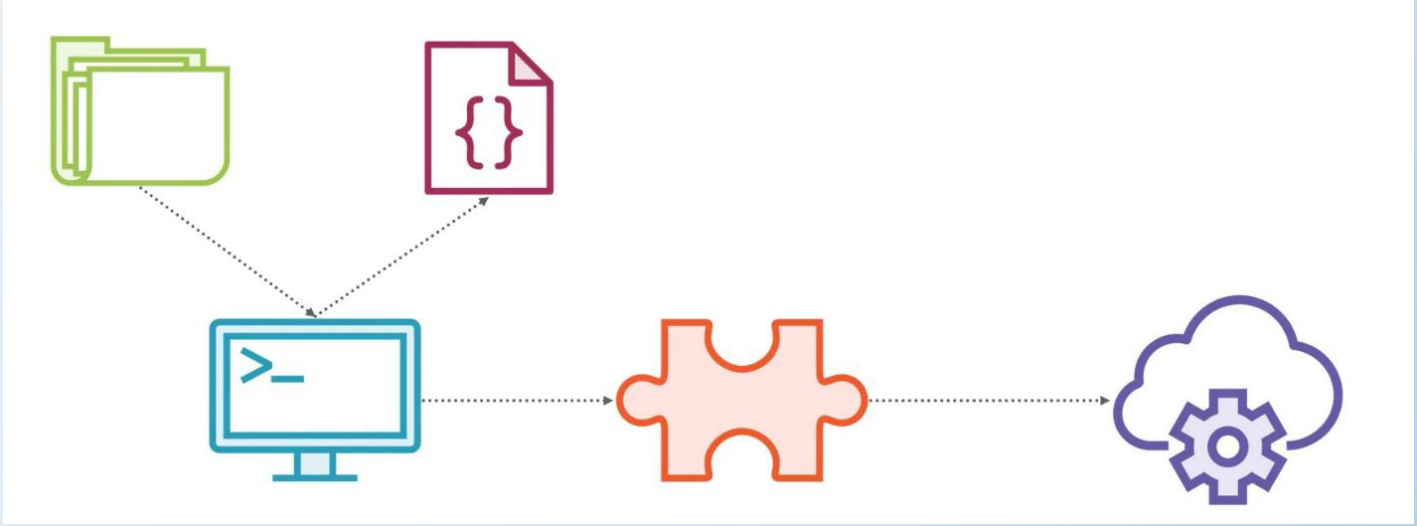
Workflow

- Terraform init
  - Ищет конфигурационные файлы
  - Смотрит, нужно ли скачивать плагины и скачивает если нужно
  - Сохраняет состояние



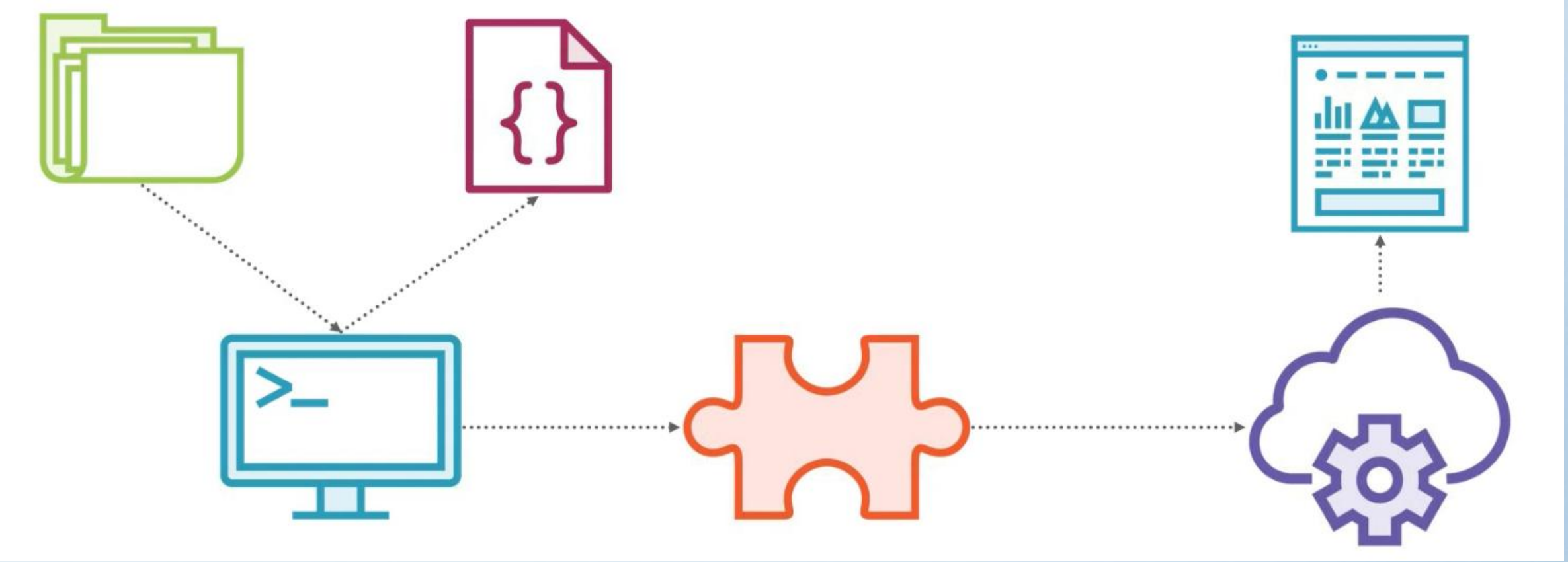
Workflow

- Terraform plan
  - Сравнивает конфигурцию и состояние и строит план преобразования



Workflow

- Применяет изменения к инфраструктуре



Provider: local

main.tf

```
provider "local" {
  version = "~> 2.2.3"
}
resource "local_file" "hello" {
  content = "Hello, Terraform"
  filename = "hello.txt"
}
```

Provider: local

- **terraform init**

Initializing the backend...

Initializing provider plugins...

- Finding hashicorp/local versions matching "~> 2.2.3"...

- Installing hashicorp/local v2.2.3...

- Installed hashicorp/local v2.2.3 (unauthenticated)

Provider: local

> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

```
  + create
```

Terraform will perform the following actions:

```
  # local_file.hello will be created
  + resource "local_file" "hello" {
      + content              = "Hello, Terraform"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "hello.txt"
      + id                   = (known after apply)
    }
```

Provider: local

> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

```
  + create
```

Terraform will perform the following actions:

```
  # local_file.hello will be created
  + resource "local_file" "hello" {
      + content              = "Hello, Terraform"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "hello.txt"
      + id                   = (known after apply)
    }
```

Provider: local

> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

```
  + create
```

Terraform will perform the following actions:

```
  # local_file.hello will be created
  + resource "local_file" "hello" {
      + content              = "Hello, Terraform"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "hello.txt"
      + id                   = (known after apply)
    }
```

Provider: local

> terraform plan

local_file.hello: Refreshing state... [id=392b5481eae4ab2178340f62b752297f72695d57]

No changes. Your infrastructure matches the configuration.

Provider: local

> cat .\hello.txt

Hello, Terraform

> echo foo > hello.txt

> terraform plan

local_file.hello: Refreshing state... [id=392b5481eae4ab2178340f62b752297f72695d57]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

Terraform will perform the following actions:

  # local_file.hello will be created

Terraform destroy

- Terraform destroy
  - Освобождение ресурсов – удаляет инфраструктуру
  - Пользоваться надо с осторожностью

Minikube + Terraform

- curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
- sudo dpkg -i minikube_latest_amd64.deb

Стартуем

- minikube start --vm-driver=docker --base-image='kicbase/stable:v0.0.32'

Проект

minikube/main.tf

Yandex cloud

- https://cloud.yandex.ru/docs/iam/concepts/authorization/oauth-token
- yc config set token <OAuth-токен>
- export YC_TOKEN=$(yc iam create-token)
- export YC_CLOUD_ID=$(yc config get cloud-id)
- export YC_FOLDER_ID=$(yc config get folder-id)

Yandex cloud

nano ~/.terraformrc

```
provider_installation {
    network_mirror {
        url = "https://terraform-mirror.yandexcloud.net/" include = ["registry.terraform.io/*/*"]
    }
    direct
    {
        exclude = ["registry.terraform.io/*/*"]
    }
}
```


ВОПРОС ?
ОТВЕТ !

Список литературы

- https://habr.com/ru/company/alloy_software/blog/274167/
- https://cdto.wiki/Развитие_ИТ-инфраструктуры
- https://mcs.mail.ru/blog/cloud-native-prilozheniya-bystro-zagruzhayutsya-snizhayut-riski-stimuliruyut-rost-biznesa
- https://losst.ru/kopirovanie-fajlov-scp
- https://habr.com/ru/company/southbridge/blog/691876/
- https://habr.com/ru/company/tinkoff/blog/532546/
- https://habr.com/ru/post/305400/
- https://habr.com/ru/company/otus/blog/574278/
- https://habr.com/ru/post/305400/
- https://ansible-for-network-engineers.readthedocs.io/_/downloads/ru/latest/pdf/
- https://docs.ansible.com/