Разработка стратегии развертывания приложений

Continuous deployment

Непрерывное развертывание (CD , Continuous deployment) – практика разработки программного обеспечения, направленная на полную автоматизацию поставки от среды разработки в промышленную среду.

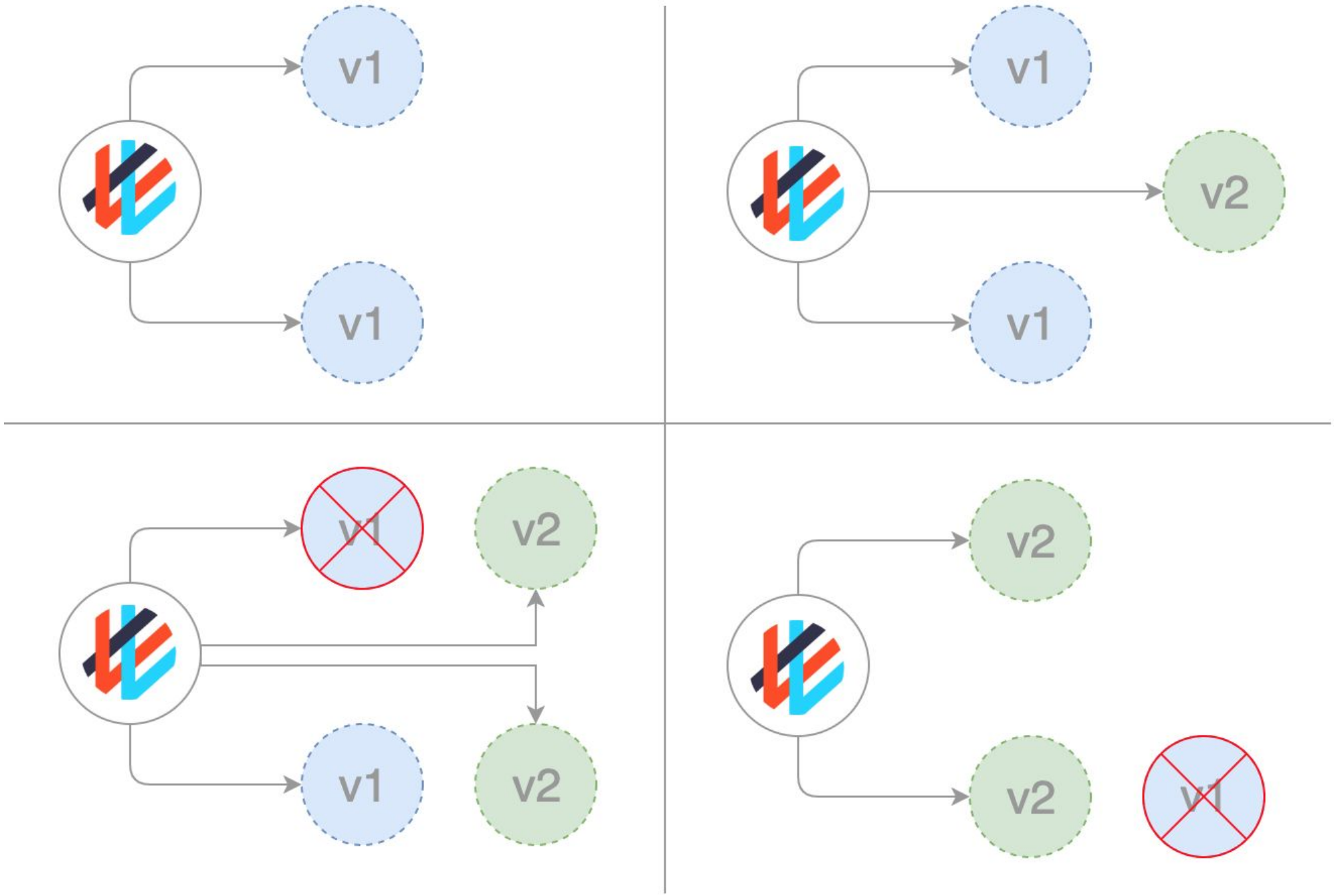
Стратегии развертывания

- Rolling (постепенный, «накатываемый» деплой)
- Recreate (повторное создание)
- Blue/Green (сине-зеленые развертывания)
- Canary (канареечные развертывания)
- Dark (скрытые) или A/B-развертывания

Rolling

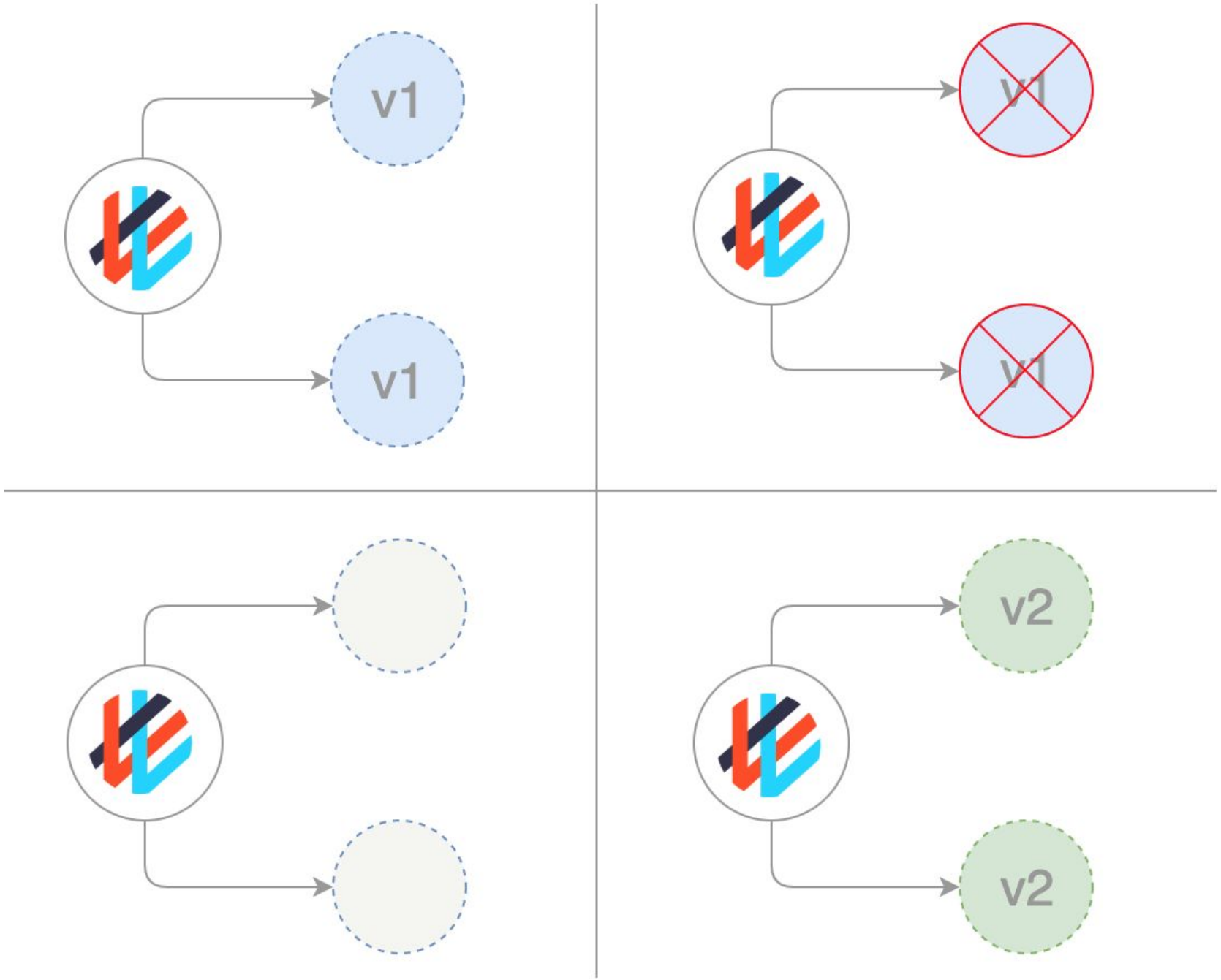
Это стандартная стратегия развертывания в Kubernetes. Она постепенно, один за другим, заменяет pod'ы со старой версией приложения на pod'ы с новой версией — без простоя кластера.

Kubernetes дожидается готовности новых pod'ов к работе (проверяя их с помощью readiness-тестов), прежде чем приступить к сворачиванию старых. Если возникает проблема, подобное накатываемое обновление можно прервать, не останавливая всего кластера.


v1
v1
v1
v2
v1
v1
v2
v1
v2
v2
v2
v1

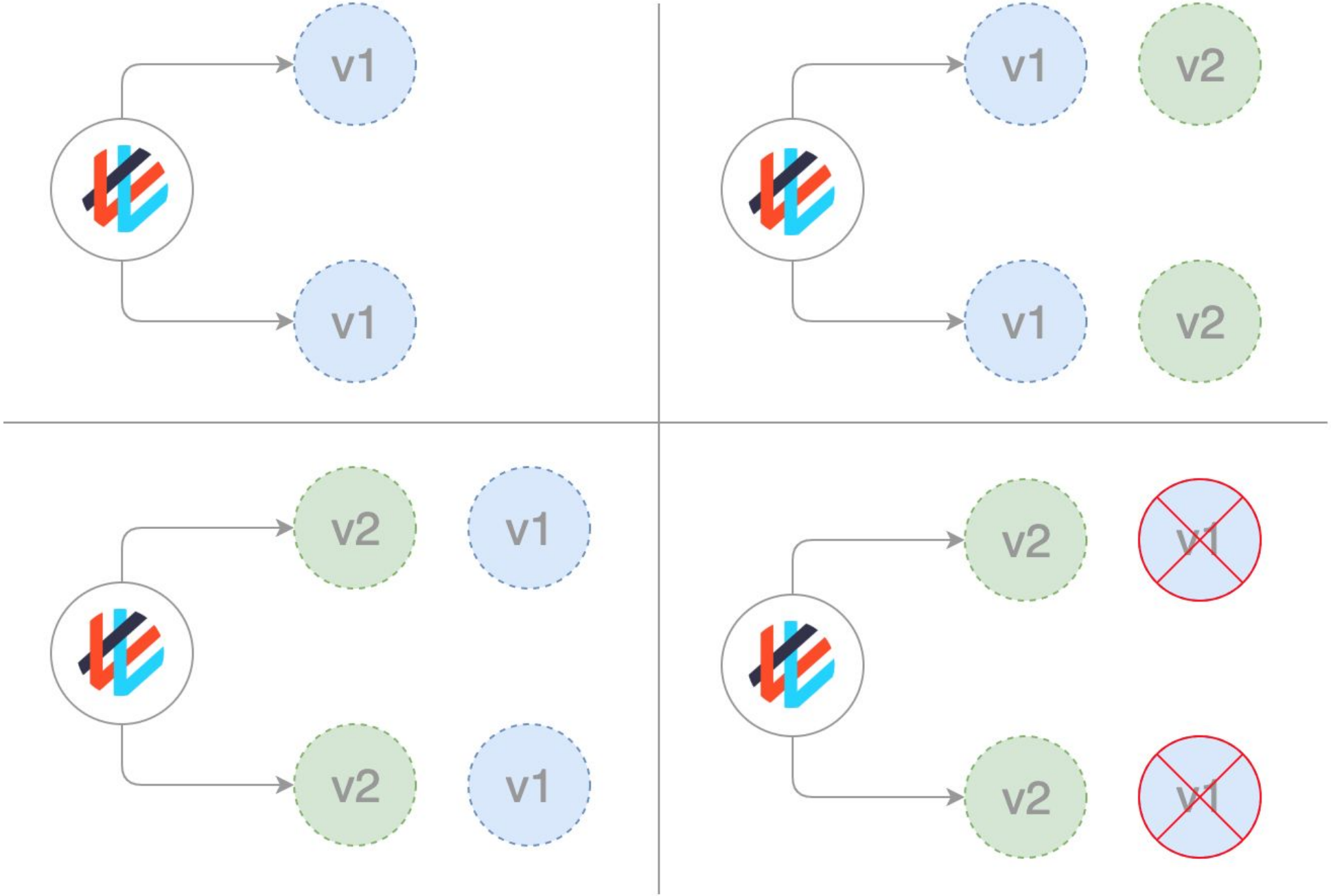
Recreate

В этом простейшем типе развертывания старые pod'ы убиваются все разом и заменяются новыми.


v1
v1
v1
v1
v2
v2

Blue/Green

Стратегия сине-зеленого развертывания (иногда ее ещё называют red/black, т.е. красно-чёрной) предусматривает одновременное развертывание старой (зеленой) и новой (синей) версий приложения. После размещения обеих версий обычные пользователи получают доступ к зеленой, в то время как синяя доступна для QA-команды для автоматизации тестов через отдельный сервис или прямой проброс портов:


v1
v1
v1
v2
v1
v2
v2
v1
v2
v1
v2
v1
v2
v1

Canary

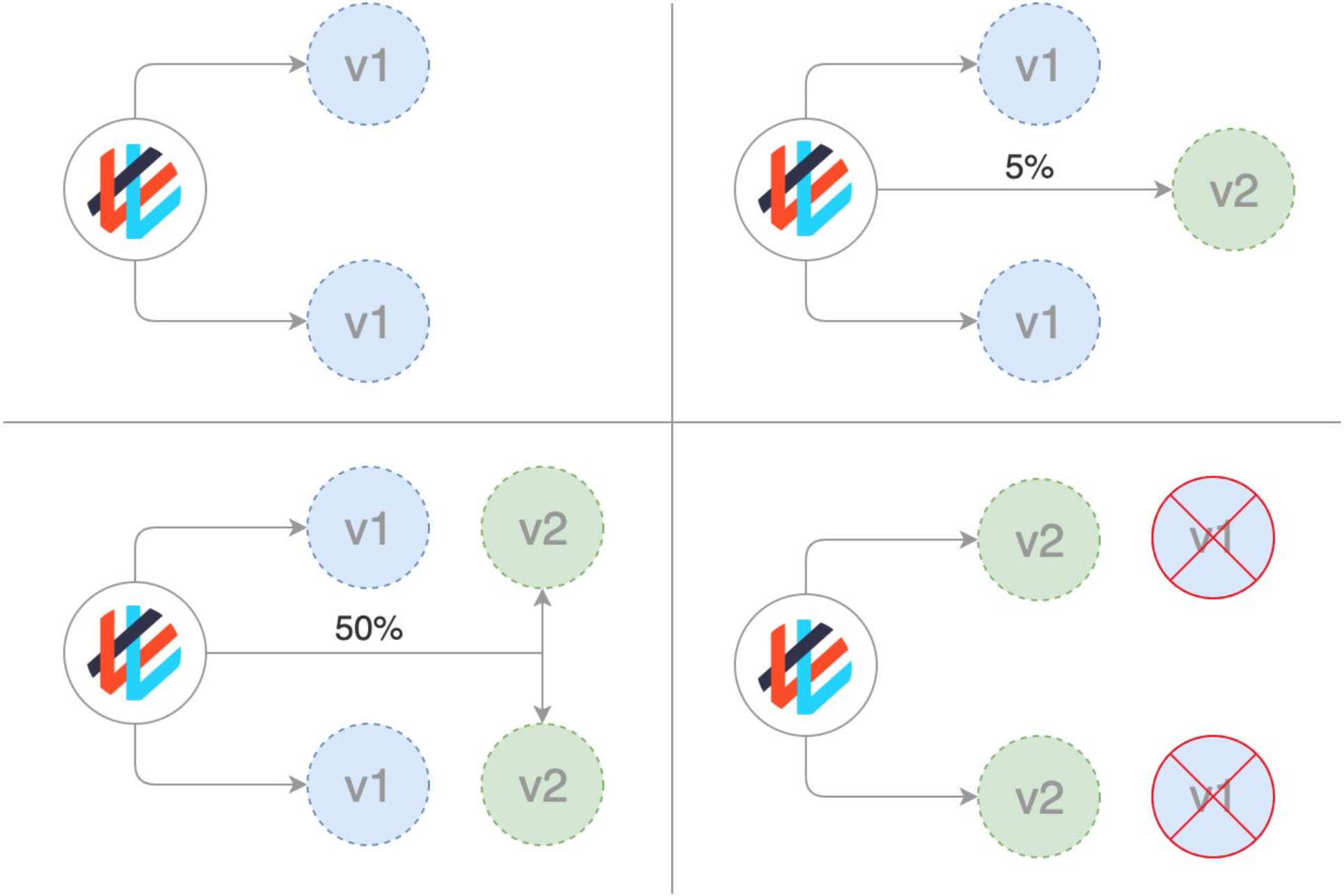
Канареечные выкаты похожи на сине-зеленые, но лучше управляются и используют прогрессивный поэтапный подход. К этому типу относятся несколько различных стратегий, включая «скрытые» запуски и A/B-тестирование.

Эта стратегия применяется, когда необходимо испытать некую новую функциональность, как правило, в бэкенде приложения. Суть подхода в том, чтобы создать два практически одинаковых сервера: один обслуживает почти всех пользователей, а другой, с новыми функциями, обслуживает лишь небольшую подгруппу пользователей, после чего результаты их работы сравниваются. Если все проходит без ошибок, новая версия постепенно выкатывается на всю инфраструктуру.

Canary

Хотя данную стратегию можно реализовать исключительно средствами Kubernetes, заменяя старые pod'ы на новые, гораздо удобнее и проще использовать service mesh вроде Istio.

Например, у вас может быть два различных манифеста в Git: обычный с тегом 0.1.0 и «канареечный» с тегом 0.2.0. Изменяя веса в манифесте виртуального шлюза Istio, можно управлять распределением трафика между этими двумя deployment'ами:

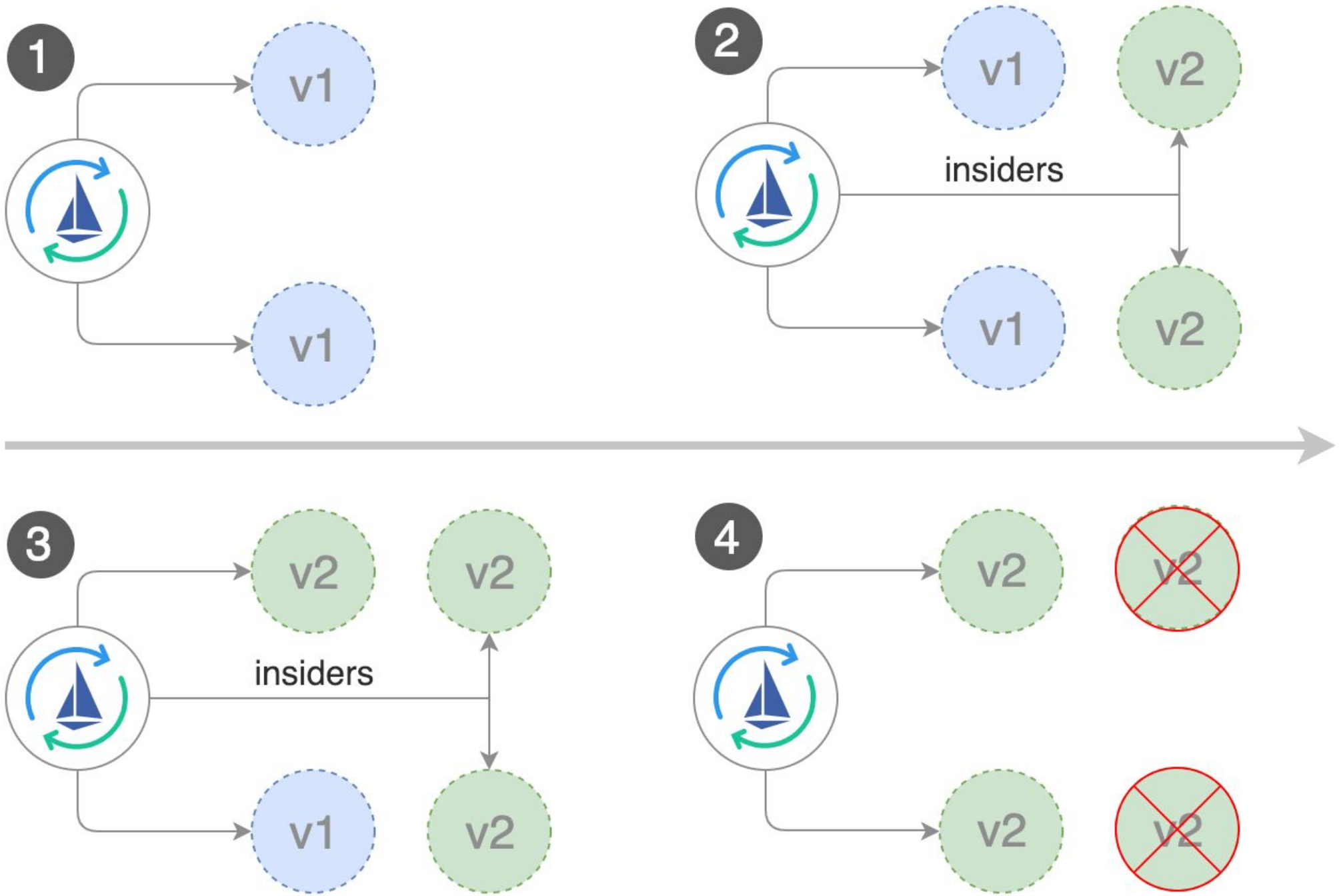

v1
v1
v1
5%
v2
v1
v1
v2
50%
v1
v2
v2
v1
v2
v1

Dark или A/B-развертывания

Скрытое развертывание — еще одна вариация канареечной стратегии. Разница между скрытым и канареечным развертыванием состоит в том, что скрытые развертывания имеют дело с фронтендом, а не с бэкендом, как канареечные.

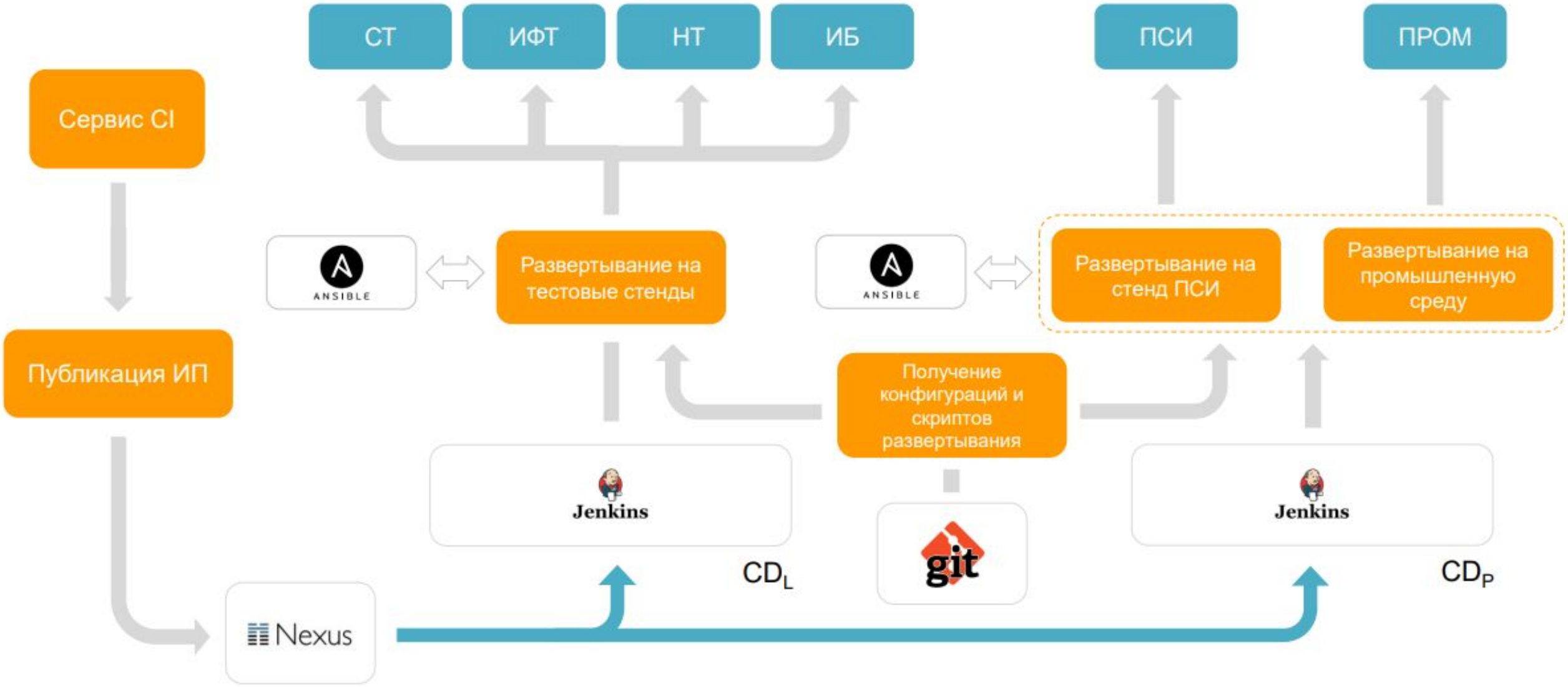
Другое название этих развертываний — A/B-тестирование. Вместо того, чтобы открыть доступ к новой функции всем пользователям, ее предлагают лишь ограниченной их части. Обычно эти пользователи не знают, что выступают тестерами-первопроходцами (отсюда и термин «скрытое развертывание»).

С помощью переключателей функциональности (feature toggles) и других инструментов можно следить за тем, как пользователи взаимодействуют с новой функцией, увлекает ли она их или они считают новый пользовательский интерфейс запутанным, и другими типами метрик.


1
v1
v1
2
v1
v2
insiders
v1
v2
3
v2
v2
insiders
v1
v2
4
v2
v2
v2
v2

Целевые инструменты CD

Sonatype	**Nexus OSS** – как хранилище всех ИП - артефакт, полученный по окончании процесса CI.
Jenkins	**Jenkins + Pipeline plugin** – как целевой инструмент оркестрации процессом развертывания на конкретную среду. Сам процесс развертывания создается с помощью языка Groovy с заранее доступными модулями расширениями в Pipeline plugin .
ANSIBLE	**Ansible** – целевой инструмент развертывания и настройки тестовых стендов. Все элементы развертывания должны быть реализованы в виде ролей Ansible, наиболее типовые роли (например: развертывание WAS, Nginx, и тд) должны быть получены из централизованного репозитория ролей СБТ.
git	**GIT** – версионное хранилище. В рамках целевого процесса CD тут должны храниться конфигурации тестовых сред, Groovy код процесса развертывания и сценарии развертывания.
Elasticsearch logstash kibana	**ELK** – стек технологий Elasticsearch, Logstash и Kibana – сбор, агрегация и визуализация журналов развертывания для отслеживания ошибок развертывания.


СТ
ИФТ
НТ
ИБ
ПСИ
ПРОМ
Сервис CI
ANSIBLE
Развертывание на тестовые стенды
ANSIBLE
Развертывание на стенд ПСИ
Развертывание на промышленную среду
Публикация ИП
Получение конфигураций и скриптов развертывания
Jenkins
CDL
git
Jenkins
CDP
Nexus

Nexus

Nexus — это централизованное хранилище. Это хранилище делится на два логических элемента.

Репозиторий разработки Nexus — хранилище содержит системное ПО; различные библиотеки, использующиеся как зависимости; результаты ночных сборок; сборок из feature веток. Также этот репозиторий зеркалит различные общедоступные в Интернет репозитории, которые необходимы для процесса разработки. В общих словах этот репозиторий содержит все, что нужно для процесса Continuous Integration. К этому хранилищу применяются обычные требования по отказоустойчивости и по сохранности данных. Сборки в этом репозитории хранятся не более одного месяца.

Nexus

Nexus — это хранилище инсталляционных пакетов (дистрибутивов) передаваемых в Банк. Этот Nexus служит гарантией того, что тот же самый дистрибутив, собранный один раз под одной версией, разворачивался и тестировался на всех этапах тестирования. То есть это хранилище служит для обеспечения непрерывной связи двух процессов — Continuous Delivery и Continuous Deployment. К этому хранилищу применяются повышенные требования по отказоустойчивости и по сохранности данных. То, как долго хранятся дистрибутивы в этом, репозитории определяется требованиями.

Jenkins

Jenkins — это проект с открытым исходным кодом, написанный на Java, созданный решать огромное число задач по оркестрации различных процессов. В нашем случае Jenkins выбран как целевое решение для оркестрации процессов развертывания как на средах тестирования в СБТ, так и на средах приемо-сдаточых испытаний и промышленных средах.

Не предполагается, что Jenkins будет оркестрировать весь процесс от среды разработки до промышленной среды. С помощью Pipeline plugin будут оркестрированы отдельные части процесса, например, развертывание ОРД в среды СТ и в среды ИФТ – это два разных Pipeline процесса в рамках одной проектной области в сервисе CD Jenkins.

Pipeline plugin

Pipeline plugin — решение, реализованное в виде плагина к Jenkins, позволяющее воплотить принцип «процесс развертывания как код», то есть с помощью этого плагина сам процесс развертывания представлен в виде Groovy кода. Pipeline plugin также предоставляет огромное число уже готовых модулей (функций Groovy) для реализации различных задач, а также имеет подробную документацию по каждому модулю — это обеспечит низкий порог вхождения, то есть не надо обучаться языку Groovy для того, чтобы начать создавать несложные процессы развертывания.

Используя данный плагин для всех своих развертываний мы решаем несколько задач:

- Версионирование сценария развертывания — версии Groovy кода процесса развертывания будут храниться не в самом Jenkins, а в GIT репозитории, что обеспечит надежное хранение и контроль изменений.
- Задача логично вытекающая из предыдущей — передача процесса развертывания между разными командами и частями процесса
- Визуализация процесса - Pipeline plugin имеет очень удобный и понятный интерфейс отображения ранее запущенных сценариев, с возможностью быстрого доступа к логам каждого этапа.

Ansible

Ansible — программное решение с открытым кодом для удаленного управления конфигурациями, реализованное на языке Python. Ansible берет на себя всю работу по приведению удаленных серверов в необходимое состояние посредством ssh протокола, необходимо лишь описать, как достичь этого состояния с помощью сценариев playbooks, выполненных в формате yaml.

С помощью Ansible можно решать различные задачи, например:

- Установка/удаление СПО.
- Конфигурирование СПО.
- Создание/удаление пользователей.
- Хранение пользовательских паролей/ключей в защищенном виде (Ansible Vault).
- Развертывание кода вашего ППО.
- Запуск скриптов авто-тестирования в сторонних системах.
- Управление системой создание/удаление контейнеров/виртуальных машин.

Ansible. Преимущества

Преимущества Ansible:

- Низкий порог вхождения. Обучиться работе с Ansible можно за очень короткие сроки.
- На управляемые узлы не нужно устанавливать никакого дополнительного ПО. Все работает через SSH.
- Код программы, написанный на Python, очень прост. Разработка дополнительных модулей не составляет особого труда.
- Декларативный язык yaml, на котором пишутся сценарии, также предельно прост.
- Встроенная возможность безопасного хранения чувствительной информации (пароли, ключи и тд) — Ansible Vault.
- Подробная и весьма просто написанная официальная документация. Она регулярно обновляется. Ее можно найти на официальном сайте.
- Ansible работает не только в режиме push, но и pull, как это делают большинство систем управления (Puppet, Chef).
- Имеется возможность последовательного обновления состояния узлов (rolling update).
- Сама структура инструмента позволяет создавать отдельные модули - роли, которые могут быть использованы разными командами в процессепоставки своего ПО. У нас имеется возможность централизованно хранить эти модули и обеспечить их использование разными командами наразных проектах.

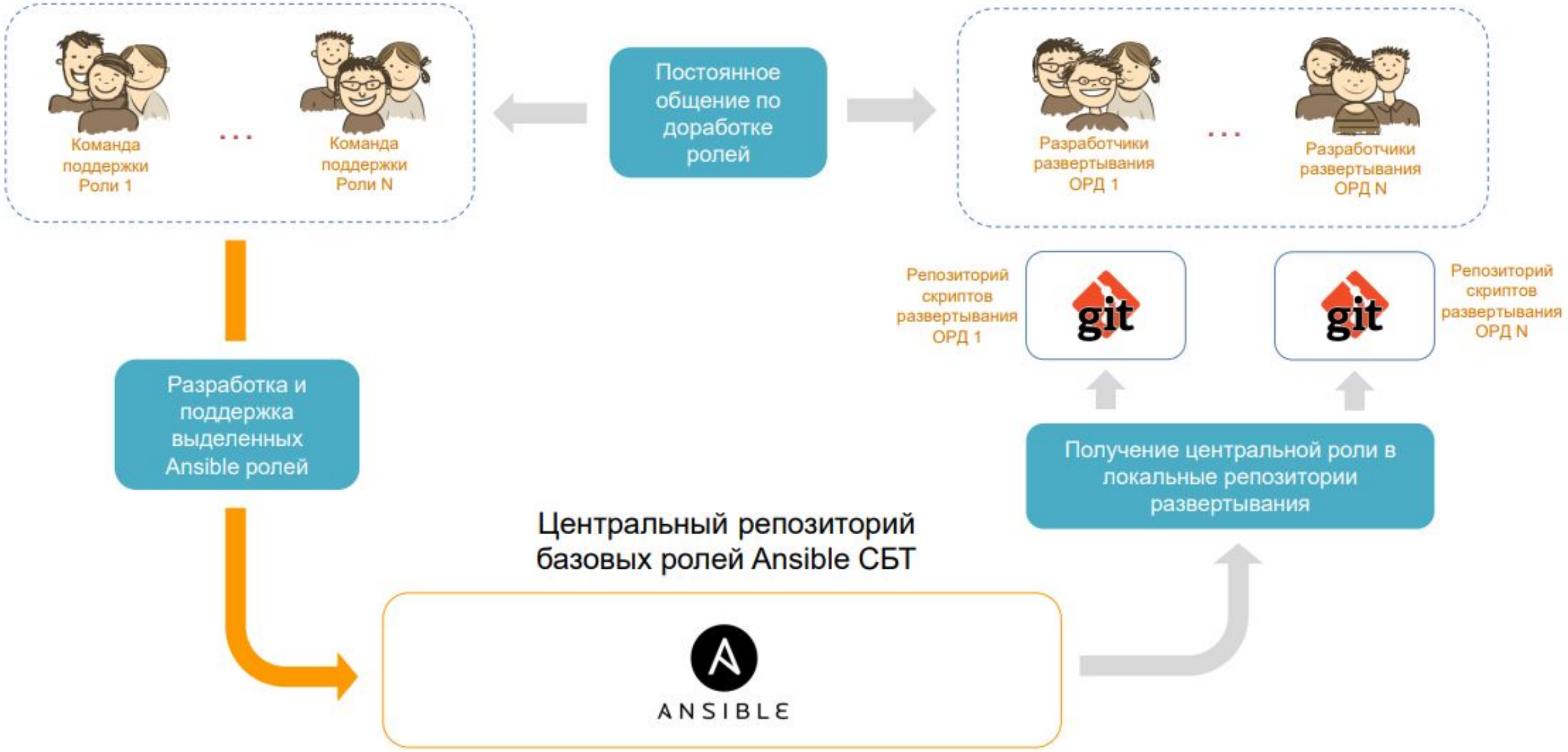
Центральный репозиторий базовых ролей Ansible

Центральный репозиторий базовых ролей Ansible в СБТ содержит роли Ansible, которые централизованно разрабатываются и поддерживаются выделенными командами и должны быть использованы разными командами в процессе развертывания каждого проекта в СБТ. По смыслу этот репозиторий – это аналог официального Ansible Galaxy, но отличается тем, что в нем делятся своими наработками исключительно сотрудники СБТ и Банка.

Роль Ansible – совокупность последовательности подзадач и всех связанных с этими подзадачами данных и вспомогательных элементов в одном месте для выполнения одной функциональной задачи. По сути это логические элементы, с помощью которых выстраивается весь процесс настройки или развертывания приложения. То есть будет отдельная роль «Установка WAS», «Установка Nginx», «Установка приложения в WAS» и тд.

Имя роли

- **files** — содержит файлы, которые будут скопированы на настраиваемые стенды. Также может содержать скрипты, которые позже будут запускаться на стендах.
- **handlers** — обработчики, которые будут использоваться при выполнении операционных задач (таких как перезагрузка сервисов и т. д.)
- **meta** — мета-данные об авторе роли, контактах для связи. Также описание зависимостей от других ролей.
- **tasks** — это по сути основная папка роли. Здесь содержится playbook со всеми задачами, которые исполняются в рамках этой роли.
- **templates** — содержит файлы-темплейты (.j2) с переменными. Эти темплейты в основном используются для конфигурационных файлов.
- **vars** — содержит локальные переменные роли.


Команда поддержки Роли 1
...
Команда поддержки Роли N
Постоянное общение по доработке ролей
Разработчики развертывания ОРД 1
...
Разработчики развертывания ОРД N
Репозиторий скриптов развертывания ОРД 1
git
git
Репозиторий скриптов развертывания ОРД N
Разработка и поддержка выделенных Ansible ролей
Получение центральной роли в локальные репозитории развертывания
Центральный репозиторий базовых ролей Ansible СБТ
ANSIBLE

Как происходит процесс развертывания?

Процесс развертывания в идеале должен содержать следующие логические шаги:

1. Получение всего необходимого для развертывания (ИП — приложение и скрипты развертывания, конфигурации)

2. Подготовка к восстановлению уже развернутого приложения (опциональный шаг, зависящий от архитектуры ФП)

3. Подготовка среды для развертывания (все ли папки созданы, права выданы, все ли скрипты на месте и готовы для развертывания)

4. Выполнение развертывания.

5. Проверки после развертывания (проверка успешности развертывания + проверка работоспособности (Smoke testing))

6. Если предыдущий шаг не успешен — восстановление приложения на предыдущую версию (опциональный шаг, зависящий от архитектуры ФП)

7. Информирование заинтересованных лиц.

Как реализован процесс развертывания?

В целом процесс разделен по следующим инструментам:

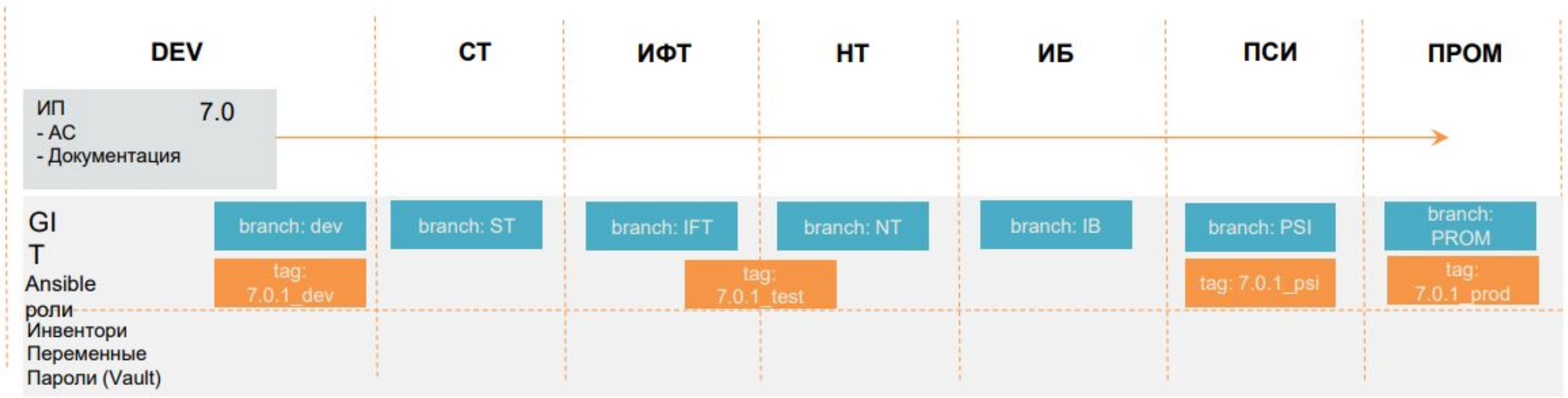
- Jenkins + Pipeline plugin в виде Groovy кода описывает общий процесс развертывания.
- Ansible роли – это логические элементы развертывания.
- GIT должен хранить все скрипты развертывания, конфигурации (сервера куда ставить – inventory, различные переменные - variables, включая чувствительные данные – пароли в зашифрованном виде)

Получение ИП из Nexus
Распаковка ИП
Получение скриптов, inventory, variables (в том числе чувствительных – пароли) из GIT
Запуск Ansible роли «Подготовка к восстановлению»
Запуск скриптов подготовки перед развертыванием
Запуск Ansible playbook, который содержит множество ролей развертывания
Запуск скриптов проверки после развертывания
Запуск автотестов в сторонних системах
Запуск Ansible роли «Отката» (если 5 шаг неуспешен)
Информирование заинтересованных лиц
Nexus
GIT
Стенд
Стенд
Стенд
Стенд
HP ALM
Стенд
Сценарий Pipeline (Groovy скрипт)
1 шаг
2 шаг
3 шаг
4 шаг
5 шаг
6 шаг
7 шаг
Jenkins Pipeline

Управление конфигурациями сред

Конфигурации сред — это конфигурации, сохраняющие целевое состояние сред как на системном уровне, так и на уровне приложения. Эти конфигурации необходимы для процесса развертывания приложения.

Конфигурации сред хранятся в централизованном GIT репозитории. Для каждой ФП создан отдельный репозиторий с разными ветками — каждая среда развертывания имеет свою отдельную ветку и своего ответственного. За ветку «dev» отвечает команда разработки, за ветки «СТ», «ИФТ», «НТ», «ИБ» - инженер развертывания тестовых сред, за ветки «ПСИ» и «ПРОМ» - представитель отдела промышленной эксплуатации. Кроме того, чтобы зафиксировать состояние конфигураций и связать их с конкретной версией ИП, должны создаваться тэги (имя тэга — версия ИП). Таким образом, если появляются изменения в конфигурациях представители разработки создают тэг, который сообщается всем остальным командам, для того чтобы они привели свои конфигурации в соответствующее состояние.


DEV
СТ
ИФТ
НТ
ИБ
ПСИ
ПРОМ
ИП
- АС
- Документация
7.0
GIT
Ansible роли
Инвентори
Переменные
Пароли (Vault)
branch: dev
branch: ST
branch: IFT
branch: NT
branch: IB
branch: PSI
branch: PROM
tag: 7.0.1_dev
tag: 7.0.1_test
tag: 7.0.1_psi
tag: 7.0.1_prod

Система управления потоками развертывания

- Управление сквозными потоками развертывания
- Визуализация сквозного маршрута инсталляционного пакета
- Реализация оперативного контроля производства технологических решений
- Сбор информации о процессах и предоставление операционной аналитики
- Сбор фактов по прохождению маршрута инсталляционным пакетом
- Синхронизация данных между сетями

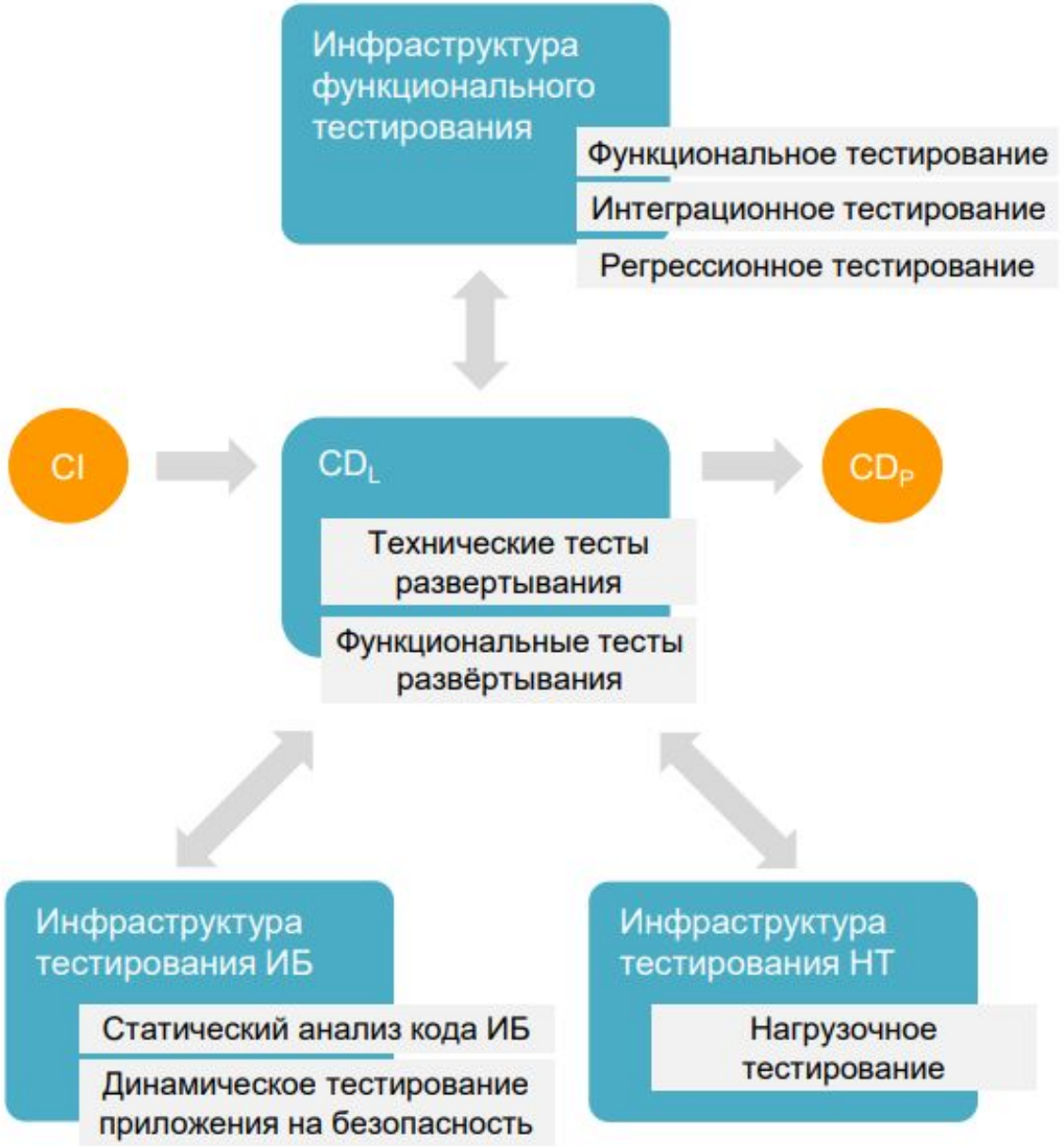
Как это работает

- В конце цикла CI сформированный ИП публикуется в **Nexus** и вместе с этим в DPM отправляется сигнал о появлении новой версии ОРД.
- DPM после получения сигнала запускает поток развертывания, соответствующего ОРД.
- Поток развертывания представляет из себя набор последовательных шагов, в рамках которых DPM запускает задания в системе CD$_L$, затем в CD$_P$, получает от них статус исполнения задания и сохраняет соответствующие статусы ИП.
- DPM на специальном дашборде визуализирует процесс выполнения потока развертывания.

Интеграции процессов развертывания и тестирования

Процессы CDL и CDP – процессы, отвечающие за все , что касается поставки уже готового собранного приложения по всем необходимым этапам тестирования и в качестве конечной цели - в промышленную эксплуатацию. В рамках этих процессов обеспечивается гарантия целостности, неизменяемости и уникальности дистрибутива, проходящего от разработки до промышленных сред.

Однако CDL и CDP не дают гарантию качества и соответствия политикам безопасности Банка самого дистрибутива. Для реализации этой задачи есть отдельные процессы – Continuous Testing и тестирование информационной безопасности (ТИБ) . Эти процессы глубоко интегрированы между собой, что обеспечивает минимизацию рисков ошибок, сбоев и взломов промышленной среды и как следствие возникновения возможных нежелательных последствий для Банка.


Инфраструктура функционального тестирования
Функциональное тестирование
Интеграционное тестирование
Регрессионное тестирование
CI
CDL
CDP
Технические тесты развертывания
Функциональные тесты развёртывания
Инфраструктура тестирования ИБ
Статический анализ кода ИБ
Динамическое тестирование приложения на безопасность
Инфраструктура тестирования НТ
Нагрузочное тестирование

Continuous Delivery

Непрерывная поставка (Continuous delivery) – практика разработки программного обеспечения, гарантирующая то, что программное обеспечение постоянно готово к развертыванию в промышленную эксплуатацию. Практика включает в себя автоматизацию развертывания на тестовые среды, автоматизацию тестирования успешности развертывания на уровне администраторов (технические и функциональные тесты развертывания), а также интеграцию с процессами автоматизированного функционального, нагрузочного тестирования и динамического тестирования на безопасность.

Continuous Delivery

Практика включает в себя:

- Хранение инсталляционного пакета (дистрибутива) в централизованном хранилище.
- Гарантирование того, что единый дистрибутив пройдет все циклы тестирования.
- Хранение конфигураций тестовых сред в централизованном версионном хранилище.
- Автоматизацию развертывания приложения на среды тестирования.
- Автоматические проверки успешности процесса развертывания.
- Интеграцию с инструментами тестирования – функционального, нагрузочного и динамического тестирования на безопасность.
- Тесное сотрудничество команды разработки, команды тестирования (ДК) и команды эксплуатации (ЦИ, ЦСПС)

Динамическая инфраструктура
Развертывание на среде тестирования информационной безопас
Тестирование ИБ (сканирование при ения)
Развертывание на среде нагрузочного тестирования (НТ)
Тестировани е НТ
Развертывание на среде системного тестирования (СТ)
Тестирован на СТ
Развертывание н среде интеграционно- функционального тестирования (ИФТ)
Тестирование на ИФТ