ГЛАВА 1

Введение в Argo CD

Платформа Kubernetes произвела настоящий переворот в мире технологий. Ее значение как краеугольного камня всей облачной экосистемы трудно переоценить. На базе Kubernetes был создан фонд Cloud Native Computing Foundation (CNCF), и вокруг него появилось множество инструментов с открытым исходным кодом, основанных на принципах неизменяемости (обеспечивает предсказуемость инфраструктуры) и декларативности (позволяет описывать желаемое состояние системы), что значительно упрощает управление облачными сервисами. По мере роста популярности и внедрения Kubernetes развивалась и вся облачная экосистема. Разнообразие задач привело к созданию Kubernetes-ориентированных проектов и инструментов (а также множества стартапов), которые ускорили распространение Kubernetes и развитие облачной архитектуры.

Одной из многочисленных проблем, возникших с внедрением Kubernetes, стало разрастание кластеров (cluster sprawl). Это явление во многом напоминало происходившее ранее с виртуальными машинами, когда бурно развивалась виртуализация. Концепция одноразовых, легко заменяемых кластеров стала популярной, заменив старую идею «единого центрального кластера для всего», которую популяризировали платформы виртуализации. Необходимость управлять жизненным циклом множества кластеров в масштабе оказалась критически важной для успеха Kubernetes и облачных развертываний. С этим столкнулись ранние пользователи Kubernetes, когда переводили свою облачную архитектуру в промышленную эксплуатацию.

В первой главе мы подробно рассмотрим эти темы, а также разберем, что такое Argo CD и какую роль он играет в экосистеме Kubernetes и облачной архитектуре.

Что такое Argo CD

Платформа Kubernetes появилась в 2014 году и быстро стала основным средством управления контейнеризированными нагрузками в масштабе. Ее декларативная природа позволила пользователям и компаниям описывать желаемое

состояние развертываний приложений — а выполнением занималась Kubernetes. Такой подход значительно отличался от традиционных императивных методов, которые использовались много лет. Тем не менее многие по-прежнему работали с Kubernetes в императивном стиле. Пользователи и компании управляли конфигурациями Kubernetes (например, YAML-файлами) вручную или с помощью событийных триггеров и сценариев. Так, SSH-команды заменялись вызовами `kubectl`, конфигурации применялись вручную, а изменения, как правило, не отслеживались.

Инструмент Argo CD появился в ответ на потребность управлять развертываниями приложений в нескольких кластерах Kubernetes, охватывающих разные окружения. Более того, стало важно не только управлять этими развертываниями, но и отслеживать их и хранить версии. Компания Intuit одной из первых внедрила Kubernetes и хорошо понимала трудности, связанные с освоением столь динамично развивающейся технологии. Это побудило Intuit сделать стратегический шаг в сторону превращения в технологическую компанию. В 2018 году она закрепилась на этом направлении, приобретя стартап Applatix (https://oreil.ly/hPi57). Это заложило основу для создания Argo Project — набора DevOps-инструментов, которые помогли ускорить адаптацию разработчиков к Kubernetes, микросервисам и облачной архитектуре в целом.

Одним из главных препятствий на пути внедрения Kubernetes был пользовательский опыт и опыт разработчиков. Kubernetes — мощная платформа, но изначально она создавалась с расчетом на то, что пользователи уже имеют серьезные навыки в администрировании систем. Основная же идея проекта Argo заключалась в создании инструментов с нуля с учетом не только GitOps, но и удобства разработчиков, — и именно здесь важная роль отводилась Argo CD.

Argo CD — один из инструментов, входящих в проект Argo. Он изначально создавался с учетом принципов GitOps и потребностей разработчиков. Его задача — доставлять изменения и обновления в один или сразу несколько кластеров Kubernetes, независимо от их масштаба. Argo CD отслеживает и предотвращает рассинхронизацию (дрифт) между состоянием кластеров и YAML-манифестами, хранящимися в Git-репозитории, используя при этом собственные возможности Kubernetes. Хотя Argo CD известен прежде всего как средство для реализации GitOps, его применяют и как универсальный DevOps-инструмент для развертывания и управления рабочими процессами в Kubernetes в средах, не основанных на GitOps. Такая гибкость сделала Argo CD одним из самых популярных инструментов в экосистеме CNCF.

Argo CD интегрируется с Helm и Kustomize, что обеспечивает еще большую гибкость и позволяет генерировать финальные YAML-манифесты, создаваемые этими инструментами, перед применением их к кластеру Kubernetes. Helm и Kustomize мы подробнее рассмотрим в главах 3 и 4.

Почему именно Argo CD

Существует множество причин, по которым специалисты DevOps выбирают Argo CD. В процессе распространения Kubernetes стало ясно, что традиционные императивные и событийно-ориентированные методы развертывания не раскрывают потенциал декларативного подхода, на котором основана эта платформа.

Унификация определений приложений

Argo CD объединил отдельные компоненты, из которых состоит приложение в Kubernetes, в сущность, управляемую как одно целое. Обычно развертывание приложения включает в себя несколько объектов Kubernetes — например, Deployment, Service и Namespace. Ранее каждый из них управлялся отдельно и между ними существовала лишь слабая связь. Argo CD собрал эти связанные объекты в единое атомарное целое, называемое *приложением* Argo CD (Argo CD Application). Благодаря этому пользователи могут управлять определениями приложений, конфигурациями и окружениями декларативно и под контролем системы управления версиями. Развертывание приложений и управление их жизненным циклом теперь можно автоматизировать, отслеживать и легко контролировать. Подробнее о структуре приложений Argo CD рассказывается в главах 4 и 5.

Дрифт конфигурации

Дрифт конфигурации (configuration drift) — проблема, сопровождающая развертывание приложений с момента их появления. Она уже довольно давно осложняет работу специалистов, и со временем было создано множество инструментов, предназначенных для борьбы с ней. Инструменты «инфраструктуры как кода» (Infrastructure as Code, IaC) помогли решить часть задач, связанных с дрифтом конфигурации, но справиться с ним по-настоящему удалось лишь с появлением неизменяемой инфраструктуры — когда Kubernetes и контейнеры начали использоваться совместно. В Argo CD применяется цикл сверки (reconciliation loop) Kubernetes, предотвращающий расхождения между реальным состоянием развертывания и его источником истины (source of truth). В отличие от событийно-ориентированных процессов, которые ждут срабатывания события для запуска согласования, Argo CD выполняет его непрерывно. Многие DevOps-специалисты используют этот инструмент, чтобы предотвращать дрифт конфигураций в масштабах всего предприятия. Передача кластеров под управление Argo CD гарантирует командам DevOps, что их окружение находится именно в том состоянии, в каком и должно.

Откат и восстановление после сбоев

Argo CD можно использовать для ускорения процессов отката изменений и восстановления после сбоев. Он поддерживает синхронизацию кластера с его источником истины, поэтому для возврата к рабочему состоянию достаточно откатить источник истины (обычно репозиторий Git) к стабильной версии. После этого Argo CD автоматически приведет кластер в нужное состояние. Аналогично осуществляется и восстановление после сбоев. DevOps-специалисты используют Argo CD для восстановления так: устанавливают его и указывают целевое состояние в Git (нужную версию). Все остальное система выполняет автоматически.

Методология GitOps

Похоже, GitOps стремительно превращается в новый модный термин в технологической индустрии и любимое слово маркетологов. Но если разобраться, что это такое, можно столкнуться с множеством объяснений, которые слабо связаны между собой. При этом среди них наверняка найдутся и знакомые вам идеи. Так что же такое *GitOps*? Инструмент ли это для разработчиков приложений? Или он предназначен скорее для инженеров инфраструктуры и системных администраторов, управляющих средами? А может быть, GitOps — просто новая оболочка для уже привычных понятий DevOps и непрерывной интеграции и доставки (CI/CD)?

На деле GitOps объединяет разные подходы к автоматизации, доставке приложений, управлению инфраструктурой и обеспечению безопасности в рамках единой системы.

Разговор о GitOps почти всегда естественно начинается с методологии DevOps, ведь именно на ней он и основан. Подход DevOps возник из потребности автоматизировать доставку приложений и дал возможность командам, которые создают, разворачивают и сопровождают программное обеспечение, работать вместе ради общей цели. DevOps — это не отдел, а культура взаимодействия в организации. Как же соотносятся GitOps и DevOps? Все просто: GitOps — это DevOps. GitOps — естественное развитие идей DevOps, воплощающее все лучшее, что уже применяли инженеры, просто не называя это GitOps.

Происхождение GitOps

Создание термина GitOps приписывают компании Weaveworks (https://github.com/weaveworks). История началась в 2017 году, когда Weaveworks предоставляла свои решения по модели SaaS и размещала приложения своих клиентов на собственной Kubernetes-платформе. Однажды из-за неправильной конфигурации — того

самого «промаха по клавише» — вышла из строя вся платформа. Тем не менее DevOps-инженеры смогли восстановить систему за относительно короткое время. На вопрос, как им удалось сделать это так быстро, они описали свои процедуры и подходы. Все это сооснователь и генеральный директор Weaveworks Алексис Ричардсон и назвал *GitOps*.

Принципы OpenGitOps

В октябре 2021 года рабочая группа GitOps (GitOps Working Group) опубликовала принципы OpenGitOps (https://oreil.ly/UTE3V) — свод основных положений для управления программными системами. Цель публикации заключалась в том, чтобы четко определить, что такое GitOps, и не дать этому термину превратиться в очередное модное слово. Текущая версия — 1.0 — подразумевает четыре принципа.

Принцип 1. Декларативность

Первый принцип OpenGitOps гласит:

> система, управляемая GitOps, должна иметь желаемое состояние, выраженное декларативно.

Под *желаемым состоянием* понимается способ описать, как система должна работать в конечном виде, — это «итоговое состояние», которое достигается в результате изменений, вносимых в среду GitOps. В этом и состоит различие между императивным (указываются конкретные команды и шаги для достижения результата) и декларативным (описывается желаемое конечное состояние, а система сама определяет, как его достичь) подходами: как уже упоминалось, Kubernetes работает именно декларативно.

Принцип 2. Версионирование и неизменяемость

Второй принцип OpenGitOps гласит:

> желаемое состояние хранится таким образом, что обеспечивается неизменяемость, версионирование и сохранение полной истории версий.

Классический пример принципа «версионирование и неизменяемость» — Git, именно поэтому он и носит название GitOps. Архитектура Git основана на принципах версионирования и неизменяемости: каждое изменение фиксируется в новой версии, не затрагивая предыдущие. Благодаря этому можно вернуться к любому предыдущему состоянию и при этом сохранить полную историю всех изменений — своего рода журнал действий и доказательство прозрачности.

Принцип 3. Автоматическое извлечение

Третий принцип OpenGitOps гласит:

> программные агенты автоматически извлекают декларации желаемого состояния из источника.

Именно этот принцип отличает GitOps от традиционных, событийно-ориентированных процессов CI/CD.

Хотя запуск изменений и обновлений с помощью веб-хуков и иных событий — допустимый способ автоматизации сборок, сам по себе он еще не является GitOps. В GitOps специальные программные агенты — *GitOps-контроллеры* — регулярно извлекают декларации желаемого состояния из репозитория Git и проверяют их. Это не одноразовая операция, а *постоянный* процесс опроса и синхронизации. В GitOps нет веб-хука, который нужно вызвать для обновления. Вместо этого используется цикл согласования, благодаря которому состояние системы автоматически приводится к описанному в Git. Это подводит нас к последнему, четвертому принципу.

Принцип 4. Непрерывное согласование

Последний принцип OpenGitOps — еще одно отличие GitOps от событийно-ориентированных рабочих процессов. Он гласит:

> программные агенты непрерывно отслеживают фактическое состояние системы и пытаются привести его к желаемому состоянию.

Этот принцип отражает работу контроллеров Kubernetes, но GitOps распространяет его на целое приложение или инфраструктурный стек, а не на отдельный объект. Как уже упоминалось, желаемое состояние извлекается из конфигурации, которая версионируется и хранится в неизменяемом хранилище. Если между желаемым и текущим состояниями возникает расхождение (дрифт), то они согласовываются путем изменения текущего состояния. Такой процесс происходит непрерывно, через заданные интервалы времени. В контексте индустрии слово «непрерывно» означает, что согласование выполняется регулярно, но не обязательно мгновенно.

Сравнение инструментов GitOps в экосистеме

Необходимость поддерживать синхронность систем существует уже довольно давно. Именно на этой идее возникла парадигма «инфраструктуры как кода (Infrastructure as Code, IaC)», а вместе с ней и целый ряд инструментов — таких как Terraform, Ansible, Puppet и Chef. Kubernetes не стал исключением. По мере

того как он набирал популярность, потребность в управлении крупномасштабными развертываниями и их поддержке в согласованном состоянии только усиливалась. Из нее родились два ключевых облачных GitOps-контроллера — Flux и Argo CD.

Flux

Инструмент Flux (https://fluxcd.io) был создан инженерами компании Weaveworks. Изначально он разрабатывался как внутреннее решение, чтобы поддерживать стабильную работу управляемых сервисов компании. Позже его доработали и выпустили в качестве проекта с открытым исходным кодом под эгидой CNCF. Текущая версия — Flux v2 — основана на концепции *наборов инструментов*. Она включает в себя отдельные библиотеки на языке Go (Golang), разработанные с учетом философии Unix: «Делай одну вещь, но делай ее хорошо». С технической точки зрения Flux v2 — это ПО, созданное на основе этих наборов инструментов. Благодаря модульной архитектуре разработчики при желании могут создавать собственные решения, используя компоненты набора инструментов Flux.

Argo CD

Как уже упоминалось, инструмент Argo CD был разработан компанией Intuit. Цель проекта — облегчить переход разработчиков Intuit на платформу Kubernetes. Этот подход можно рассматривать как раннюю форму *платформенной инженерии*, а инструмент Argo CD — как внутреннюю платформу разработчика (Internal Developer Platform, IDP) компании Intuit. Идея заключалась в том, чтобы скрыть все сложности и особенности развертывания приложений и управления ими в Kubernetes, предоставив разработчикам удобный уровень абстракции. Argo CD был не первым инструментом, созданным в Intuit: для полноценной экосистемы требовались и другие, ориентированные на DevOps решения. Позже эти инструменты были объединены под названием Argo Project (о котором подробнее рассказано далее в этой главе) и переданы в CNCF как проект с открытым исходным кодом.

Сравнение Flux и Argo CD

Подробное сравнение этих двух инструментов выходит за рамки книги. Главные различия между ними — в философии управления платформой Kubernetes. Особенно заметно, как Argo CD работает напрямую с исходными YAML-манифестами и стремится максимально повторять функциональность утилиты `kubectl`.

Рассмотрим пример — работу с Helm. Инструмент Flux развертывает Helm-чарты с помощью библиотеки Helm напрямую, а Argo CD сначала преобразует чарты в исходные YAML-файлы (с помощью команды `helm template`), а затем применяет их к кластеру Kubernetes. В среде Argo CD команда `helm ls` не выводит результаты,

тогда как в Flux список чартов отображается. Зато при использовании Argo CD доступно сравнение: можно увидеть различия между текущим и желаемым состоянием даже при развертывании через Helm-чарты. В Flux такой возможности нет: различия между состояниями скрыты.

Есть еще одно важное различие: Flux можно использовать с графическим интерфейсом, подключив Weave GitOps, но собственного интерфейса у него нет — Flux работает исключительно через CLI и API. Инструмент Argo CD, напротив, изначально задумывался как завершенный продукт, в котором есть полнофункциональный графический интерфейс, система разграничения доступа (RBAC) и инструменты для реализации многоарендной среды.

Выбор между Flux и Argo CD зависит от множества факторов, и рассмотреть все нюансы в рамках одной книги невозможно. Далее мы будем исходить из того, что в качестве основного инструмента GitOps вы выбрали Argo CD, ведь именно поэтому вы читаете эту книгу!

Экосистема Argo

Обычно, когда в контексте облачных технологий говорят об *Argo*, первым на ум приходит инструмент *Argo CD*. Однако это лишь один из подпроектов, входящих в более широкий проект Argo. Argo Project — это набор облачных DevOps-инструментов, которые упрощают работу SR-инженеров и разработчиков, а также помогают новичкам быстрее освоиться в мире Kubernetes. Многие удивляются, узнав, что самый популярный инструмент в составе Argo Project — Argo Workflows: по количеству звезд на GitHub он опережает остальные проекты. В состав Argo Project входят следующие инструменты.

Argo Workflows

Облачная система управления рабочими процессами, особенно популярная и активно применяемая в сообществах искусственного интеллекта и машинного обучения. В последнее время растет интерес к использованию Workflows в контексте решения задач непрерывной интеграции (CI).

Argo CD

Один из самых обсуждаемых инструментов в облачном мире; реализует подход GitOps в рамках управления и развертывания приложений в Kubernetes на уровне масштабных систем.

Argo Rollouts

Продвинутый контроллер прогрессивной доставки, который работает в связке с Argo CD (но при желании может использоваться и отдельно). Он помогает выполнять предварительные и сине-зеленые развертывания, используя собственные контроллеры Ingress или Service Mesh.

Argo Events

Открытый фреймворк для автоматизации на основе событий.

Кроме того, существует Argo Labs (https://github.com/argoproj-labs) — подразделение, выполняющее роль «инкубатора» инструментов, связанных с экосистемой Argo Project. Например, компонент Argo CD ApplicationSet (теперь входящий в состав Argo CD) изначально развивался в Argo Labs, а затем был добавлен в основной релиз Argo CD в виде общедоступной функции (general availability, GA).

Резюме

В этой главе мы рассмотрели ключевые проблемы и решения экосистемы Kubernetes и облачных технологий, отметив стремительное распространение этой платформы и связанную с этим необходимость эффективного управления кластерами. Одна из основных трудностей — «расползание» кластеров, что потребовало появления более надежных инструментов для управления множеством кластеров в масштабах организации. Argo CD, инструмент, созданный в рамках проекта Argo, помогает решать эти задачи, предоставляя подход GitOps для управления и развертывания приложений в кластерах Kubernetes. Инструмент опирается на декларативную природу Kubernetes, отслеживает развертывания приложений и управляет их версиями, предотвращая дрифт конфигураций и поддерживая процессы отката и восстановления после сбоев.

Кроме того, в этой главе мы рассмотрели принципы методологии GitOps, ее эволюцию из DevOps и преимущества использования Argo CD по сравнению с другими инструментами, такими как Flux. Наконец, вы познакомились с экосистемой Argo, в которую входят инструменты Argo Workflows, Argo Rollouts и Argo Events, образующие полноценный набор решений для DevOps и управления Kubernetes. Полученные знания помогут вам лучше понять контекст и причины выбора архитектурных решений, и вы сможете перейти к практической части книги — конфигурации и настройке Argo CD.