확장본#3 - 깃옵스(GitOps)를 여행하려는 입문자를 위한 안내서

조 훈(Hoon Jo)

심근우(Gnu Shim)

문성주(Martin Moon)

깃옵스(GitOps)를 여행하려는 입문자를 위한 안내서

쿠버네티스를 활용하여 시스템을 운영하는 과정에서 팀과 시스템의 요구 사항을 충족시키기 위한 다양한 전략을 만나게 됩니다. 이러한 전략들은 Operation 을 의미하는 -Ops 를 붙이는 것이 일반적입니다.

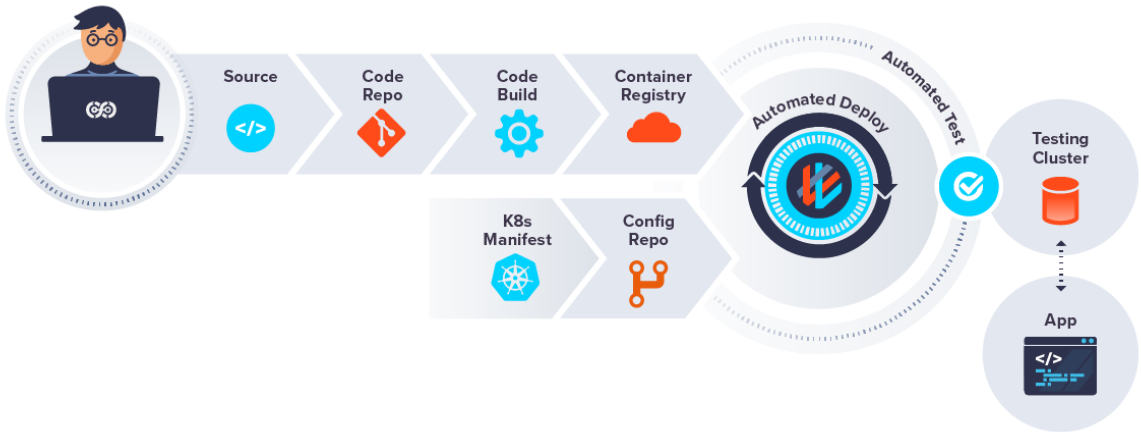
대표적인 전략으로는 **개발**(Dev)과 **운영**(Ops)을 유기적으로 결합하여 효과적인 협업을 통한 빠른 배포를 가능하게 하는 **데브옵스**(DevOps). **개발**(Dev)과 **보안**(Sec) 그리고 **운영**(Ops)을 결합하여 개발부터 운영까지 모든 단계에서 보안을 강화하는 **데브섹옵스**(DevSecOps). **머신러닝**(ML)과 **운영**(Ops)을 결합하여 머신러닝과 관련된 환경의 구축과 운영에 대한 효율성을 높여주는 **MLOps** 등이 있습니다. 이러한 전략들은 달성하고자 하는 목표에 적합한 자동화 도구를 적극적으로 활용한다는 공통점이 있습니다.

이 안내서에서는 자동화 도구와 소스 코드 관리 도구인 **깃**(Git)을 결합하여 쿠버네티스의 리소스를 배포하는 전략인 **깃옵스**(GitOps)의 개념을 알아보고 깃옵스에 활용할 수 있는 자동화 도구에 대해서도 살펴보겠습니다.

1. 깃옵스란 무엇일까요?

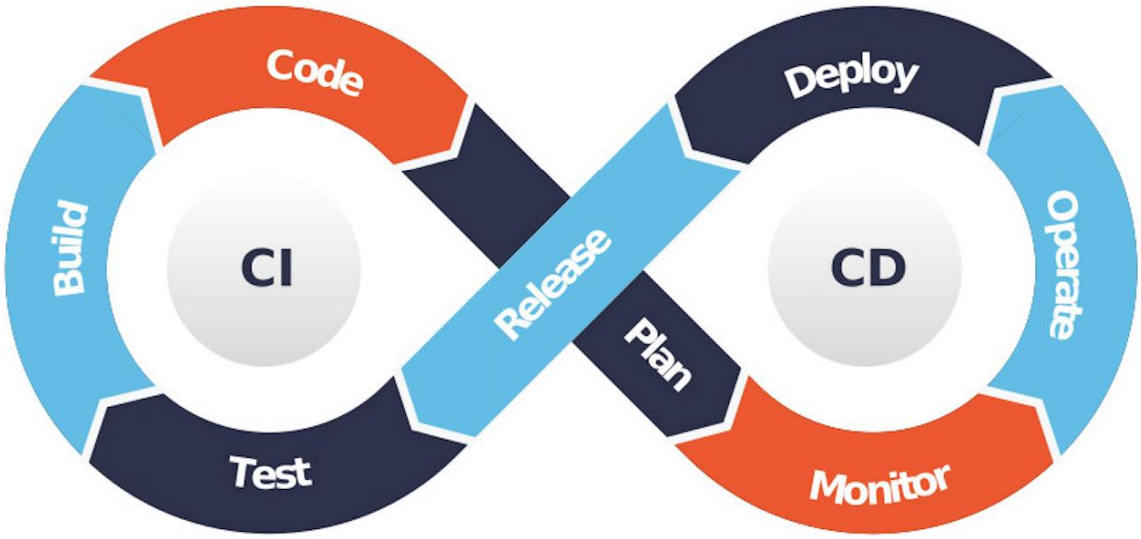
깃옵스란 **위브웍스**(Weaveworks)에서 2017 년 처음 사용한 단어로, 깃 저장소에 저장된 코드로 시스템의 구성 요소들을 정의하고, 이것을 **신뢰할 수 있는 단일 원천**(A single source of truth)으로 활용하여 현재 배포된 상태와 일치시키는 것을 말합니다. 조금 더 직관적으로 설명을 드리자면, 깃 저장소에 저장된 쿠버네티스 YAML 파일의 상태는 현재 쿠버네티스 클러스터에 배포된 리소스를 그대로 반영하고 있다는 의미입니다.

[그림 1] 깃옵스 파이프라인 (https://www.weave.works/technologies/gitops/)



깃옵스의 동작 방식을 좀 더 이해하기 위해 기존에 널리 사용하던 CI/CD 방식을 떠올려 봅시다. 기존 방식은 애플리케이션의 변경을 반영하기 위한 빌드 작업부터 프로세스가 시작됩니다. 변경된 애플리케이션을 컨테이너로 빌드하여 이미지 레지스트리에 업로드한 뒤 쿠버네티스 API 나 kubectl 등의 명령어를 통해 이미지 태그가 변경된 파드를 배포하면 CI/CD 의 사이클이 끝나는 것이 일반적이었습니다. 또한 서비스, 볼륨 혹은 컨피그맵 등의 변경은 애플리케이션의 변경과 별도의 작업으로 처리되곤 했습니다.

[그림 2] CI/CD 파이프라인 (https://www.weave.works/blog/gitops-fully-automated-ci-cd-pipelines)

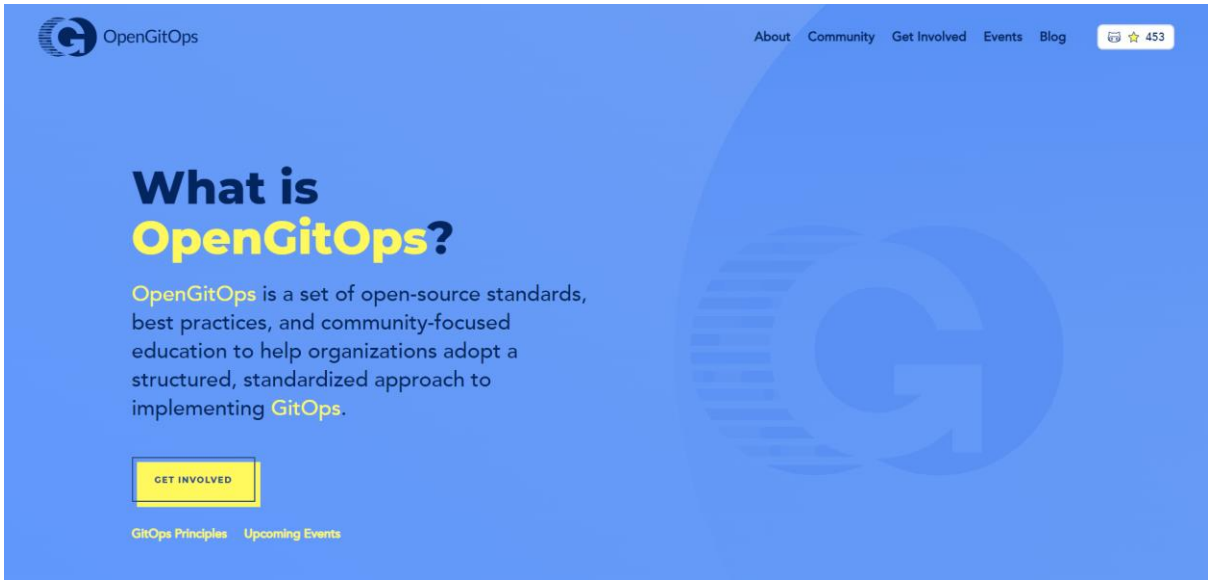


깃옵스를 채택할 경우 애플리케이션의 변경뿐 아니라 쿠버네티스 클러스터에 배포된 어떠한 리소스에 대해서든 해당 변경을 적용한 YAML 파일이 깃 저장소에 반영되면서 배포 프로세스가 시작됩니다. 깃옵스를 위한 자동화도구는 깃 저장소의 변경을 감지하여 현재 배포된 리소스와 차이를 쿠버네티스 클러스터에 반영하게 됩니다. 이러한 방식을 통해 애플리케이션의 변경이 반영된 파드 뿐 아니라 네트워크와 관련된 서비스, 스토리지와 관련된 PV, PVC 및 애플리케이션이 사용할 정보가 담긴 시크릿과 컨피그맵 등 배포된 모든 리소스에 대해 종합적인 관리가 가능합니다.

2. 깃옵스의 원칙과 깃옵스 성숙도 모델

이러한 깃옵스를 구현하기 위해 관련된 시스템과 도구들이 어떤 원칙 아래 동작하여야 하는지에 대해서 CNCF OpenGitOps Project (https://opengitops.dev/)는 아래와 같은 4 가지의 원칙을 제시하고 있습니다.

[그림 3] CNCF OpenGitOps Project 홈페이지



첫째, 깃옵스로 관리되는 시스템의 **희망하는 상태**(Desired state)는 **선언적**(Declarative)으로 표현되어야 함.

선언적인 표현이란 원하는 상태 그 자체를 표현하는 방식입니다. 이는 원하는 상태를 만들기 위해 필요한 동작을 지시하는 명령형과 대비되는 방식으로, 원하는 상태 그 자체를 전체적으로 기술한 뒤 반영하는 방식입니다. 하고자 하는 동작을 일일이 지시하지 않고, 최종적으로 원하는 상태를 기술하였기 때문에 선언적인 표현이 최종적으로 반영될 상태라는 것을 확신할 수 있는 장점이 있습니다.

쿠버네티스는 YAML 파일을 통해 선언적인 상태를 기술하고 있습니다.

둘째, 희망하는 상태는 버전 별로 기록되며 이미 기록된 상태는 불변성을 가짐.

선언적인 상태가 변경 시마다 버전 별로 기록되며, 이미 기록된 상태는 변하지 않는 것을 확신할 수 있다면, 변경 내용의 추적이 가능하며, 이전 버전을 사용하여 상태를 되돌릴 경우 원하는 상태로 온전히 돌아간다는 것을 확신할 수 있는 장점이 있습니다. 깃옵스에서는 깃의 특성을 활용하여 이 원칙을 적용하고 있습니다. 깃은 매 변경에

대해서 변경 내역을 기록하기 때문에 깃 저장소에 보관된 YAML 파일이 변경될 경우 자동적으로 내역에 대한 관리가 가능하며, 이미 기록된 내용에 대해서는 임의의 변경이 불가능합니다.

셋째, 깃옵스와 관련된 도구는 소스 코드 저장소에서 희망하는 상태를 자동으로 가져옴.

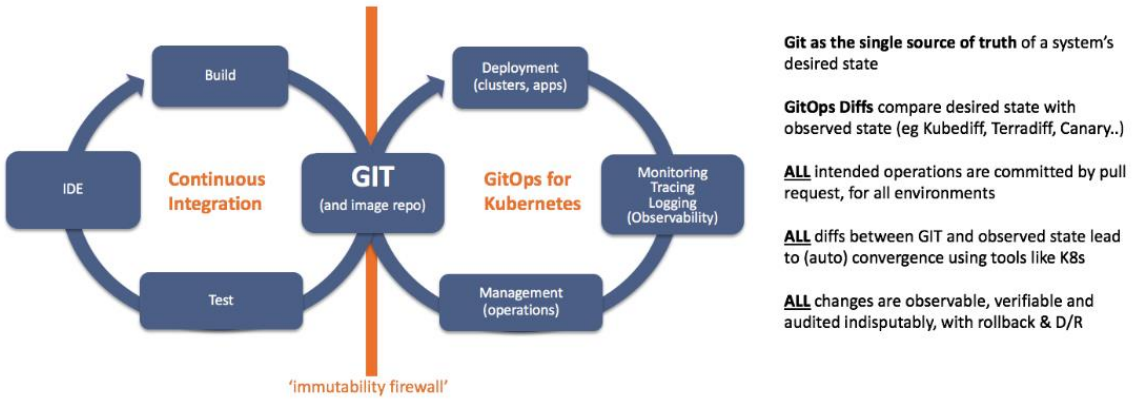
배포 과정에서는 사람의 개입을 최소화하고 자동화된 프로세스에 많은 것을 맡겨야 휴먼 에러의 가능성이 줄어들고, 반복 작업 시에도 빠르고 일관적인 동작을 할 수 있습니다. 아래에서 설명할 깃옵스와 관련된 자동화 도구들은 선언적인 상태와 관련된 파일이 관련된 파일이 깃 저장소로 부터 자동으로 쿠버네티스 클러스터에 반영될 수 있도록 하는 기능을 가지고 있습니다.

넷째, 깃옵스와 관련된 도구는 시스템의 상태를 지속적으로 관찰하며, 희망하는 상태로 반영하도록 시도함

시스템은 문제가 생기거나 임의의 조작으로 인해 원하지 않은 상태로 변경될 수 있습니다. 이 때 깃옵스와 관련된 도구는 깃 저장소에 담긴 YAML 파일을 앞서 설명한 신뢰할 수 있는 단일 원천으로 사용하고 있으므로, 끊임 없이 시스템을 관찰하며 깃 저장소에 담겨진 YAML 과 일치하는 상태로 쿠버네티스 클러스터의 리소스를 반영하도록 시도하는 기능을 가지고 있습니다. 이를 통해 클러스터의 상태를 유지하고, 부가적으로 차이가 나는 부분에 대해 사용자에게 알림을 제공할 수 있게 됩니다.

위브웍스는 위의 원칙에 대해서 아래와 같은 모델을 제시하였습니다.

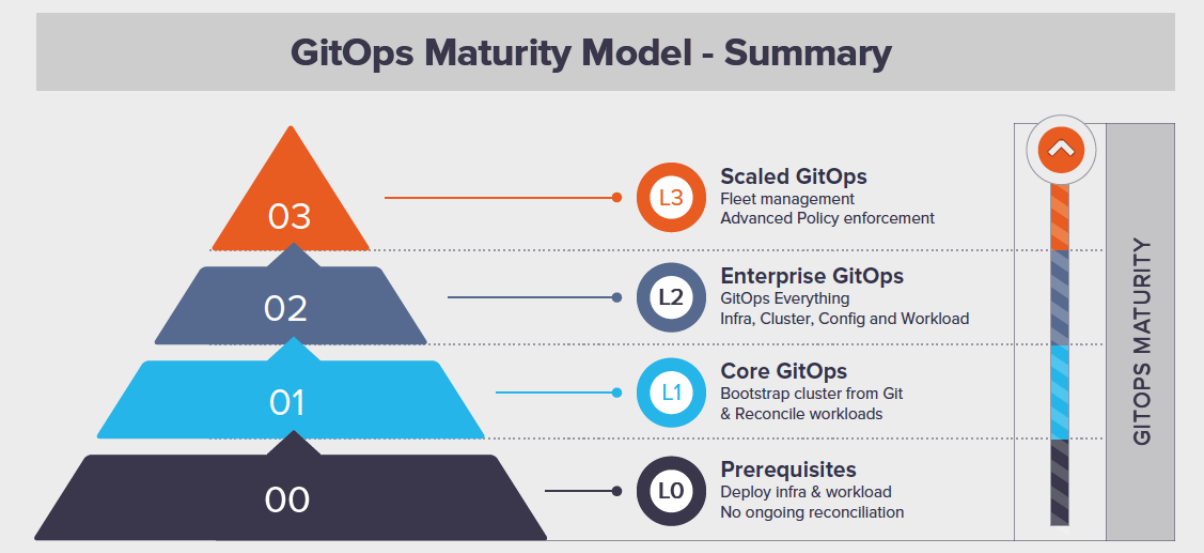
[그림 4] 깃옵스를 통한 쿠버네티스 운영 모델 (https://www.weave.works/blog/what-is-gitops-really)



그렇다면 사용자가 어느 수준으로 깃옵스를 적용하고 있는지에 대해서 어떻게 확인할 수 있을까요?

위브웍스는 이에 대해 4 단계로 구성된 **깃옵스 성숙도 모델**(GitOps Maturity Model)을 제시하였습니다.

[그림 5] 깃옵스 성숙도 모델 요약 (https://www.weave.works/blog/the-gitops-maturity-model)



L0 – 사전 단계

깃을 통해서 애플리케이션의 소스 코드 및 일부 YAML 파일이 관리되고 있으며 배포와 관련된 자동화 도구도 사용하고 있으신가요? 이 단계에 해당하고 있으며, 깃옵스로 전환할 준비가 되어 있습니다. 다만 깃에 저장된 코드가 쿠버네티스 클러스터에 반영된 리소스와 일치하지 않기 때문에 깃옵스라고 보기에는 미흡한 단계입니다.

L1 – 핵심적인 부분에 대한 깃옵스

쿠버네티스 클러스터에 반영할 리소스 파일들이 깃 저장소에 반영되어 있으며 깃옵스 도구를 통해 깃 저장소와 쿠버네티스 클러스터의 리소스 상태를 일치시키고 있습니다. 파드, 서비스, 컨피그맵 등 중요 리소스에 대해서 일관된 관리가 이루어지고 있으며, 차이가 발생할 경우 깃옵스 도구를 활용하여 빠르게 이를 알아챌 수 있으며 반영이 이루어집니다. 대부분의 경우 이 단계의 깃옵스를 잘 활용하는 것으로 충분합니다.

L2 – 엔터프라이즈 수준의 깃옵스

쿠버네티스 클러스터 전체 수준에서 깃옵스가 적용되고 있습니다. Cluster API 등에 대해서도 선언적인 관리가 이루어지고 있어 새로운 클러스터를 구성하거나 개발 환경 전체를 운영 환경으로 이행되는 경우, 혹은 여러 리전에 걸친 배포가 이루어질 때도 일관적인 관리를 할 수 있습니다.

L3 – 대규모로 확장된 깃옵스

멀티 클라우드 혹은 하이브리드 클라우드에 걸쳐 수백개 이상의 대규모 클러스터에도 대응할 수 있는 깃옵스가 적용되어 있습니다. 다수의 클라우드 공급자 및 환경에 대응할 수 있는 정책까지 깃옵스를 통해 관리합니다.

위의 성숙도 모델은 반드시 더 높은 단계를 달성해야 한다는 것은 아니며, 조직과 시스템이 성장함에 따라 목표로 할 수 있는 하나의 지표 역할을 하는 것입니다.

3. 어떤 깃옵스 도구를 사용할 것인가? – Flux vs Argo CD

앞서 깃옵스의 개념과 동작 방식 등에 대해서 살펴보았습니다. 깃옵스를 실제 배포에 적용하기 위하여 택할 수 있는 도구로 대표적인 것은 **Flux**와 **Argo CD**가 있습니다. 두 도구는 모두 위에서 설명한 깃옵스의 원칙을 따르도록 구현되었으며, 핵심적인 동작에는 큰 차이가 없습니다. 그러나 약간의 차이가 있으므로, 두 도구의 특징을 중심으로 어떤 도구를 선정할지 알아보도록 하겠습니다.

[그림 6] Flux 와 Argo CD 로고

두 도구를 선택하는데 있어서 몇 가지 요소를 살펴보겠습니다. 살펴볼 요소는 성숙도, 유저 인터페이스, 권한 및 멀티 클러스터 지원, 기타 고급 기능 지원입니다.

1) 성숙도

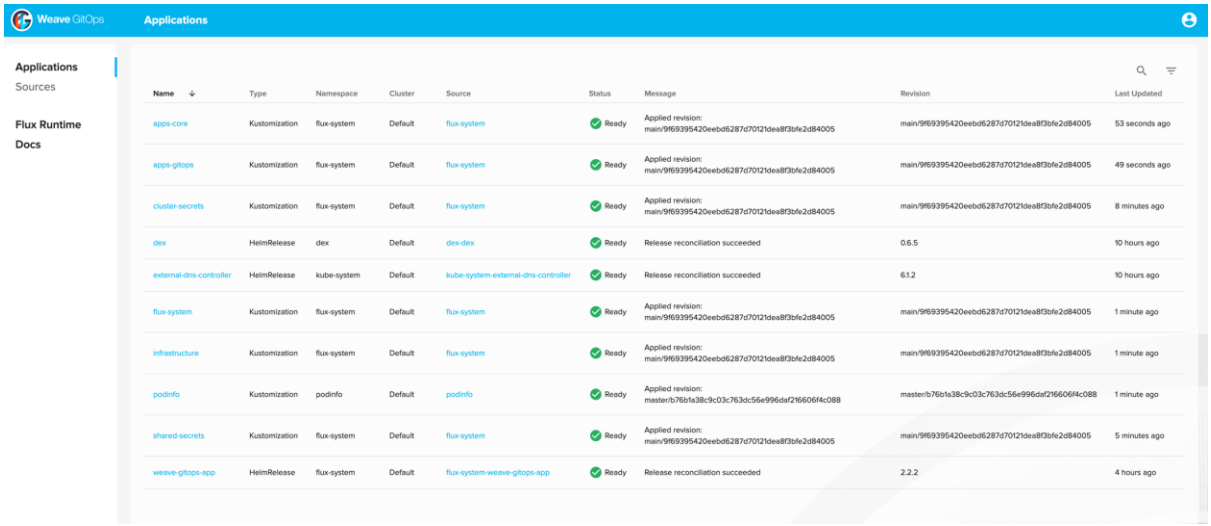
두 프로젝트는 모두 이 글의 발표 시점에서 모두 졸업(Graduated) 상태이므로 충분한 성숙도를 가지고 있습니다. 기능적으로 충분히 검증되었으며 많은 사용 사례가 있으므로 성숙도 측면에서는 두 도구 모두 믿을 수 있습니다.

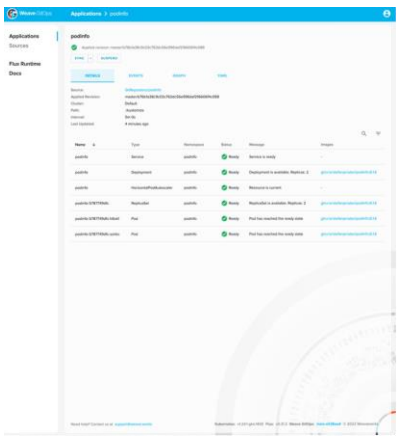
[그림 7] Flux와 Argo 졸업

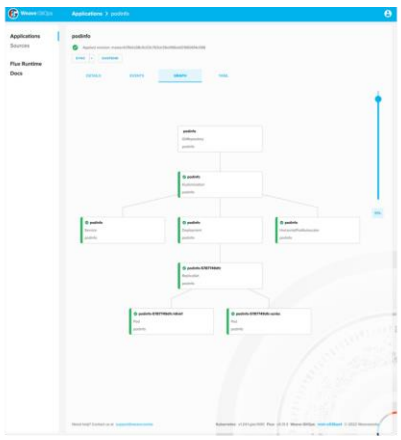
2) 유저 인터페이스

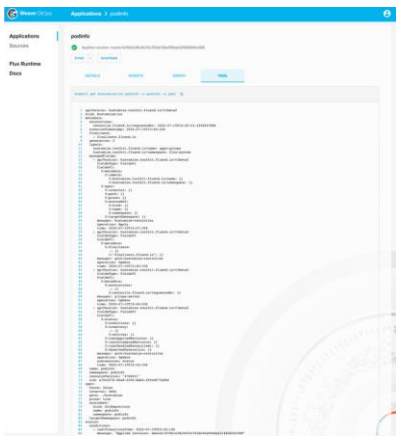
Flux 의 가장 큰 특징은 대부분의 동작이 전용 명령어 도구를 이용한 CLI(Command Line Interface)를 통하여 이루어진다는 점입니다. 또한 대부분의 설정을 CRD(Custom Resource Definition)를 따르는 YAML 파일을 작성하여 적용해야 하기 때문에 설정을 적응하는데 직관성이 떨어집니다. 이외에도 Flux 에 GUI 대시보드 기능을 사용하기 위해서는 위브웍스에서 추가로 제공하는 Weave GitOps 를 설치해야 하는 번거로움이 있습니다.

[그림 8] Weave GitOps 대시보드 (https://github.com/weaveworks/weave-gitops)



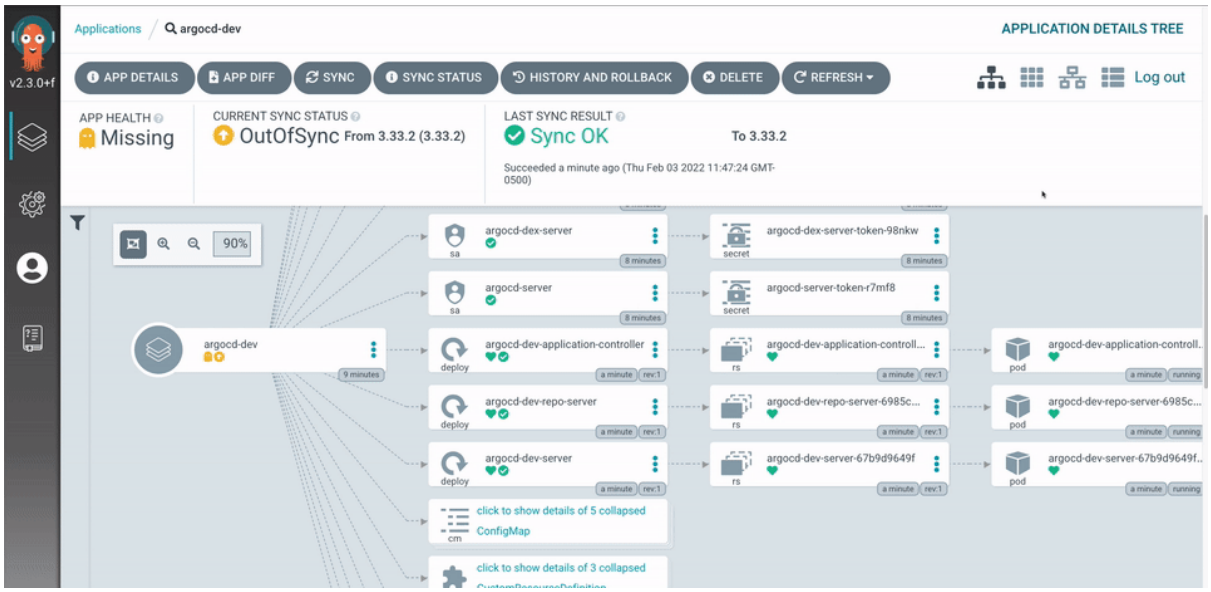






그에 반해 Argo CD는 CLI 이외에도 GUI 또한 지원하므로 사용자가 GUI를 통해 친숙하게 작업할 수 있습니다. 또한 대시보드에서 클러스터에 배포된 리소스를 시각적으로 잘 표현해주고 있어 클러스터의 리소스 관리 목적으로도 활용도가 높습니다. 또한 이후에 설명할 권한 관리 또한 Argo CD의 GUI 와 함께 통합되어 있어서 편의성의 측면에서 높은 점수를 줄 수 있습니다.

[그림 9] Argo CD 대시보드(https://argo-cd.readthedocs.io/en/stable/)



3) 권한 및 멀티 클러스터 지원

권한 및 멀티 클러스터의 지원에 대해서 Flux와 Argo CD는 확연하게 다른 철학을 가지고 있습니다. Flux는 쿠버네티스의 역할 기반 접근 제어(RBAC)과 밀접하게 연관되어 있습니다. 그렇기 때문에 깃옵스 시스템에 대한 자체적인 사용자 권한 관리나 인증을 지원하지 않습니다. 또한 Flux는 멀티 클러스터를 지원하기 위해 KubeConfig 설정을 통해 특정 클러스터에서 접근할 수 있는 다른 클러스터들을 구성하고 해당 클러스터 내부에서 Flux를 동작 시켜서 멀티 클러스터를 관리하는 접근 방식을 취합니다. Flux는 쿠버네티스의 기본 메커니즘과 통합된 동작 방식을 지원하지만, 이 부분은 쿠버네티스 운영에 관여하지 않고, 단지 애플리케이션의 배포에 집중하고자 하는 사용자에게는 추가적인 부담으로 작용할 수 있습니다.

반면 Argo CD는 독립적인 사용자 관리 메커니즘이 있습니다. 사용자별 접근할 수 있는 기능을 제한할 수 있으며, Okta, Keycloak, Azure AD 등과 같은 외부 도구와 통합하여 사용자를 관리할 수 있기 때문에 사용자별, 팀별 권한 분리가 필요한 경우 Argo CD는 조

직 전체의 깃옵스 도구로서 훌륭한 역할을 수행할 수 있습니다. 또한 멀티 클러스터의 지원에 있어서도 Argo CD의 내부에서 관리할 클러스터를 구성할 수 있으며, 이 동작 또한 앞에서 설명한 GUI를 통해서 직관적으로 접근할 수 있기 때문에, 배포에 집중하기 위한 사용자에게는 Argo CD가 좀 더 접근에 대한 부담을 낮춰줄 수 있습니다.

[표 1] Flux와 Argo CD 비교

	Flux	Argo CD
성숙도	Graduated	Graduated
유저 인터페이스	CLI	CLI / GUI 지원으로 활용도 높음
권한 관리	쿠버네티스 RBAC과 통합	자체적으로 사용자, 팀별 관리가 가능
기능 제어	쿠버네티스와 밀접하게 통합	Argo CD 만으로 사용 가능
멀티 클러스터	KubeConfig 관리가 필요	Argo CD에서 제어 가능

지금까지 대표적인 두 깃옵스 도구의 특징에 대해서 알아보았습니다. 두 도구의 핵심 기능은 깃 저장소를 중심으로 쿠버네티스 클러스터의 리소스를 반영한다는 결과는 동일합니다. 그러나 그 사용성의 측면에서는 약간의 차이를 보입니다. Argo CD는 GUI를 통해서 좀 더 편리한 관리가 가능하며, 배포 도구는 쿠버네티스 운영자뿐 아니라 배포 단계에 주로 관여하는 개발자에게도 영향을 크게 미치는데, **Flux**는 깃옵스 목적으로 잘 사용하기 위해서 쿠버네티스 자체에 대한 이해와 추가적인 구성 및 설치가 더 필요한 반면 **Argo CD**는 각종 기능에 대해서 자체적인 관리가 가능하며 깃옵스를 주된 목적으로 하는 도구들이 이미 통합되어 있기 때문에 적은 노력으로 효과적인 깃옵스를 구현하기에는 Argo CD에 더 손을 들어주고 싶습니다.

다만 Argo CD에 이미 통합되어 있는 내용이 불편하고 CLI로 깃옵스를 구현하고자 하는 문화를 가진 조직이라면 Flux가 더 적합할 수 있습니다. 모두가 좋다고 하는 도구라고 할지라도 각 조직에 맞는 도구가 꼭 정해져 있지 않다는 점을 이해하고 사용하시기를 권장드립니다. 2023년에는 독자 여러분의 쿠버네티스 세계가 더 편하게 접근하고 사용되길 그리고 함께 더 발전하길 희망합니다.