"ArgoCD: A Deep Dive into GitOps “ – Authored & Compiled by Srujana 😊 ✍✍

In this article, Let’s Dive deep into Argo‘s features and use it to run a simple demo application.

Argo CD is “a declarative, Git Ops continuous delivery tool for Kubernetes.” It can monitor your source repositories and automatically deploy changes to your cluster.

Kubernetes orchestrates container deployment and management tasks. It starts your containers, replaces them when they fail, and scales your service across the compute nodes in your cluster.

Kubernetes is best used as part of a continuous delivery workflow. Running automated deployments when new code is merged ensures changes reach your cluster quickly after passing through a consistent pipeline.

What Is Argo CD?

Argo CD is a popular tool for setting up continuous delivery with Kubernetes. It automates application deployment into Kubernetes clusters by continually reconciling your repository’s state against your live workloads.

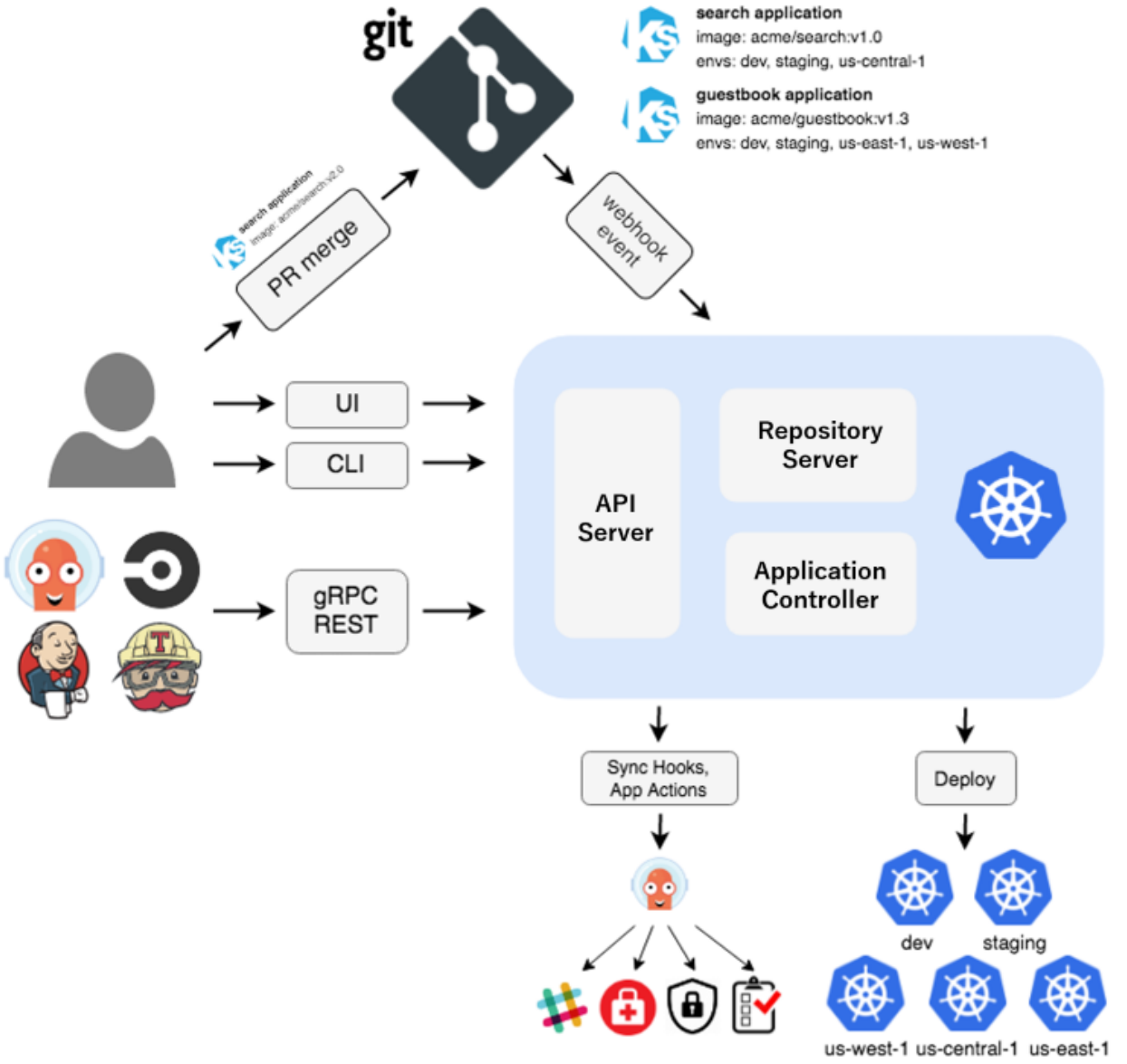
The Git Ops model is integral to Argo’s design. It makes the repository the single source of truth for your application’s desired state. All the Kubernetes manifests, Customize templates, Helm charts, and config files your app needs should be committed to your repository. These resources “declare” what a successful deployment of your app looks like.

Argo compares the declared state to what’s actually running in your cluster, then applies the correct changes to resolve any discrepancies. This process can be configured to run

automatically, preventing your cluster from drifting away from your repository. Argo resynchronizes the state whenever differences occur, such as after you manually run Kubectl commands.

Argo comes with both a CLI and web UI. It supports multi-tenant and multi-cluster environments, integrates with SSO providers, produces an audit trail, and can implement complex rollout strategies such as canary deployments and blue/green upgrades. It also offers integrated rollbacks so you can quickly recover from deployment failures.

Architecture



Argo CD is implemented as a Kubernetes controller which continuously monitors running applications and compares the current, live state against the desired target state (as specified in the Git repo). A deployed application whose live state deviates from the target state is considered Out Of Sync. Argo CD reports & visualizes the differences, while providing facilities to automatically or manually sync the live state back to the desired target state. Any modifications made to the desired target state in the Git repo can be automatically applied and reflected in the specified target environments.

For additional details, see architecture overview.

Features

- Automated deployment of applications to specified target environments
- Support for multiple config management/templating tools (Customize, Helm, Jsonnet, plain-YAML)
- Ability to manage and deploy to multiple clusters
- SSO Integration (OIDC, OAuth2, LDAP, SAML 2.0, GitHub, GitLab, Microsoft, LinkedIn)
- Multi-tenancy and RBAC policies for authorization
- Rollback/Roll-anywhere to any application configuration committed in Git repository
- Health status analysis of application resources
- Automated configuration drift detection and visualization
- Automated or manual syncing of applications to its desired state
- Web UI which provides real-time view of application activity
- CLI for automation and CI integration
- Webhook integration (GitHub, Bit Bucket, GitLab)
- Access tokens for automation
- Pre Sync, Sync, Post Sync hooks to support complex application rollouts (e.g.blue/green & canary upgrades)
- Audit trails for application events and API calls
- Prometheus metrics
- Parameter overrides for overriding helm parameters in Git

Adoption

Currently, the following organizations are **officially** using Argo CD:

1. 127Labs
2. 3Rein
3. 4data
4. 7shifts
5. Adevinta
6. Adfinis

7. Adobe
8. Adventure
9. Adyen
10. AirQo
11. Akuity
12. Alarm.com
13. Alauda
14. Albert Heijn
15. Alibaba Group
16. Allianz Direct
17. AlphaSense
18. Amadeus IT Group
19. Ambassador Labs
20. Ancestry
21. Andgo Systems
22. ANSTO - Australian Synchrotron
23. Ant Group
24. AppDirect
25. Arctiq Inc.
26. Arturia
27. ARZ Allgemeines Rechenzentrum GmbH
28. Augury
29. Autodesk
30. Axians ACSP
31. Axual B.V.
32. Back Market
33. Bajaj Finserv Health Ltd.
34. Baloise
35. BCDevExchange DevOps Platform
36. Beat
37. Beez Innovation Labs
38. Bedag Informatik AG
39. Beleza Na Web
40. Believable Bots

41. BigPanda
42. BioBox Analytics
43. BMW Group
44. Boozt
45. Bosch
46. Boticario
47. Broker Consulting, a.s.
48. Bulder Bank
49. Cabify
50. CAM
51. Camptocamp
52. Candis
53. Capital One
54. CARFAX Europe
55. CARFAX
56. Carrefour Group
57. Casavo
58. Celonis
59. CERN
60. Chainnodes
61. Chargetrip
62. Chime
63. Cisco ET&I
64. Cloud Posse
65. Cloud Scale
66. CloudScript
67. CloudGeometry
68. Cloudmate
69. Cloudogu
70. Cobalt
71. Codefresh
72. Codility
73. Cognizant
74. Commonbond

75. Compatio.AI
76. Contlo
77. Coralogix
78. Crédit Agricole CIB
79. CROZ d.o.o.
80. CyberAgent
81. Cybozu
82. D2iQ
83. DaoCloud
84. Datarisk
85. Daydream
86. Deloitte
87. Deutsche Telekom AG
88. Devopsi - Poland Software/DevOps Consulting
89. Devtron Labs
90. DigitalOcean
91. Divar
92. Divistant
93. Dott
94. Doximity
95. EDF Renewables
96. edX
97. Elastic
98. Electronic Arts Inc.
99. Elementor
100. Elium
101. END.
102. Energisme
103. enigmo
104. Envoy
105. Factorial
106. Farfetch
107. Faro
108. Fave

109. Flexport
110. Flip
111. Fly Security
112. Fonoa
113. Fortra
114. freee
115. Freshop, Inc
116. Future PLC
117. Flagler Health
118. G DATA CyberDefense AG
119. G-Research
120. Garner
121. Generali Deutschland AG
122. Gepardec
123. Getir
124. GetYourGuide
125. Gitpod
126. Gllue
127. gloat
128. GLOBIS
129. Glovo
130. GlueOps
131. GMETRI
132. Gojek
133. GoTo Financial
134. GoTo
135. Greenpass
136. Gridfuse
137. Groww
138. Grupo MasMovil
139. Handelsbanken
140. Hazelcast
141. Healy
142. Helio

143. Hetki
144. hipages
145. Hiya
146. Honestbank
147. Hostinger
148. IABAI
149. IBM
150. Ibotta
151. IFS
152. IITS-Consulting
153. IllumiDesk
154. imaware
155. Indeed
156. Index Exchange
157. Info Support
158. InsideBoard
159. Instruqt
160. Intuit
161. Jellysmack
162. Joblift
163. JovianX
164. Kaltura
165. Kandji
166. Karrot
167. KarrotPay
168. Kasa
169. Kave Home
170. Keeeb
171. KelkooGroup
172. Keptn
173. Kinguin
174. KintoHub
175. KompiTech GmbH
176. Kong Inc.

177. KPMG
178. KubeSphere
179. Kurly
180. Kvist
181. Kyriba
182. LeFigaro
183. Lely
184. LexisNexis
185. Lian Chu Securities
186. Liatrio
187. Lightricks
188. Loom
189. Lucid Motors
190. Lytt
191. LY Corporation
192. Magic Leap
193. Majid Al Futtaim
194. Major League Baseball
195. Mambu
196. MariaDB
197. Mattermost
198. Max Kelsen
199. MeDirect
200. Meican
201. Meilleurs Agents
202. Mercedes-Benz Tech Innovation
203. Mercedes-Benz.io
204. Metacore Games
205. Metanet
206. MindSpore
207. Mirantis
208. Mission Lane
209. mixi Group
210. Moengage

211. Money Forward
212. MOO Print
213. Mozilla
214. MTN Group
215. Municipality of The Hague
216. My Job Glasses
217. Natura &Co
218. Nethopper
219. New Relic
220. Nextbasket
221. Nextdoor
222. Next Fit Sistemas
223. Nikkei
224. Nitro
225. NYCU, CS IT Center
226. Objective
227. OCCMundial
228. Octadesk
229. Octopus Deploy
230. Olfeo
231. omegaUp
232. Omni
233. Oncourse Home Solutions
234. Open Analytics
235. openEuler
236. openGauss
237. OpenGov
238. openLooKeng
239. OpenSaaS Studio
240. Opensurvey
241. OpsMx
242. OpsVerse
243. Optoro
244. Orbital Insight

245. Oscar Health Insurance
246. Outpost24
247. p3r
248. Packlink
249. PagerDuty
250. Pandosearch
251. Patreon
252. PayIt
253. PayPay
254. Peloton Interactive
255. Percona
256. PGS
257. Pigment
258. Pipedrive
259. Pipefy
260. Pipekit
261. Pismo
262. PITS Globale Datenrettungsdienste
263. Platform9 Systems
264. Polarpoint.io
265. Pollinate
266. PostFinance
267. Preferred Networks
268. Previder BV
269. Priceline
270. Procore
271. Productboard
272. Prudential
273. PT Boer Technology (Btech)
274. PUBG
275. Puzzle ITC
276. Pvotal Technologies
277. Qonto
278. QuintoAndar

279. Quipper
280. RapidAPI
281. rebuy
282. Red Hat
283. Redpill Linpro
284. Reenigne Cloud
285. reev.com
286. Relex Solutions
287. RightRev
288. Rijkswaterstaat
289. Rise
290. Riskified
291. Robotinfra
292. Rocket.Chat
293. Rogo
294. Rubin Observatory
295. Saildrone
296. Salad Technologies
297. Saloodo! GmbH
298. Sap Labs
299. Sauce Labs
300. Schwarz IT
301. SCRM Lidl International Hub
302. SEEK
303. SEKAI
304. Semgrep
305. Shield
306. SI Analytics
307. Sidewalk Entertainment
308. Skit
309. Skribble
310. Skyscanner
311. Smart Pension
312. Smilee.io

313. Smilegate Stove
314. Smood.ch
315. Snapp
316. Snyk
317. Softway Medical
318. South China Morning Post (SCMP)
319. Speee
320. Spendesk
321. Splunk
322. Spores Labs
323. Statsig
324. SternumIOT
325. StreamNative
326. Stuart
327. Sumo Logic
328. Sutpc
329. Swiss Post
330. Swisscom
331. Swissquote
332. Syncier
333. Synergy
334. Syself
335. TableCheck
336. Tailor Brands
337. Tamkeen Technologies
338. TBC Bank
339. Techcombank
340. Technacy
341. Telavita
342. Tesla
343. TextNow
344. The Scale Factory
345. ThousandEyes
346. Ticketmaster

347. Tiger Analytics
348. Tigera
349. Toss
350. Trendyol
351. tru.ID
352. Trusting Social
353. Twilio Segment
354. Twilio SendGrid
355. tZERO
356. U.S. Veterans Affairs Department
357. UBIO
358. UFirstGroup
359. ungleich.ch
360. Unifonic Inc
361. Universidad Mesoamericana
362. Upsider Inc.
363. Urbantz
364. Vectra
365. Veepee
366. Verkada
367. Viaduct
368. VietMoney
369. Vinted
370. Virtuo
371. VISITS Technologies
372. Volvo Cars
373. Voyager Digital
374. VSHN - The DevOps Company
375. Walkbase
376. Webstores
377. Wehkamp
378. WeMaintain
379. WeMo Scooter
380. Whitehat Berlin by Guido Maria Serra +Fenaroli

381. Witick
382. Wolffun Game
383. WooliesX
384. Woolworths Group
385. WSpot
386. Yieldlab
387. Youverify
388. Yubo
389. ZDF
390. Zimpler
391. ZipRecuiter
392. ZOZO

Push vs. Pull-based CI/CD

Historically, most CI/CD implementations have relied on push-driven behavior. This requires you to connect your cluster to your CI/CD platform, then use tools like Kubectl and Helm within your pipeline to apply Kubernetes changes.

Argo is a pull-based CI/CD system. It runs *inside* your Kubernetes cluster and pulls source *from* your repositories. Argo then applies the changes for you without a manually configured pipeline.

This model is more secure than push-based workflows. You don't have to expose your cluster's API server or store Kubernetes credentials in your CI/CD platform. Compromising a source repository only gives an attacker access to your code instead of the code and a route to your live deployments.

Argo CD Concepts

Argo is easy to learn once you understand its basic concepts. Here are some terms to get familiar with.

- **Argo controller** – Argo's Application Controller is the component you install in your cluster. It implements the Kubernetes controller pattern to monitor your applications and compare their state against their repositories.
- **Application** – An Argo application is a group of Kubernetes resources which collectively deploy your workload. Argo stores the details of applications in your cluster as instances of an included Custom Resource Definition (CRD).
- **Live state** – The live state is the current state of your application inside the cluster, such as the number of Pods created and the image they're running.
- **Target state** – The target state is the version of the state that's declared by your Git repository. When the repository changes, Argo will apply actions that evolve the live state into the target state.
- **Refresh** – A refresh occurs when Argo fetches the target state from your repository. It will compare the changes against the live state but doesn't necessarily apply them at this stage.
- **Sync** – A Sync is the process of applying the changes discovered by a Refresh. Each Sync moves the cluster back towards the target state.

Now we've covered the core concepts, we can use Argo to deploy an example app to Kubernetes. You can read more about Argo's terminology and the tool's architecture in the official docs.

Practical Example: Using Argo CD to Deploy to Kubernetes

Let's use Argo to run a basic NGINX web server instance in Kubernetes. We'll assume you've already got access to a Kubernetes cluster, and you've got the Kubectl and Helm CLIs available on your machine.

Create Your App's GitHub Repository

First, head to GitHub and create a new repository for your app. Afterward, clone your repo to your machine, ready to commit your Kubernetes manifests:

```
$ git clone https://github.com/<username>/<repo>.git
```

Copy the following YAML and save it as deployment.yaml inside your repository:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: argo-demo
  labels:
    app.kubernetes.io/name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app.kubernetes.io/name: nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: nginx
    spec:
```

```
    containers:
    - name: nginx
      image: nginx:latest
      ports:
        - name: http
          containerPort: 80
```

It defines a basic Kubernetes Deployment object that runs three NGINX replicas.

Next, copy this second YAML file and save it to service.yaml. It sets up a LoadBalancer service to expose your Deployment outside your cluster:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: argo-demo
spec:
  type: LoadBalancer
  selector:
    app.kubernetes.io/name: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: http
```

Finally, add a manifest that will create your application's namespace:

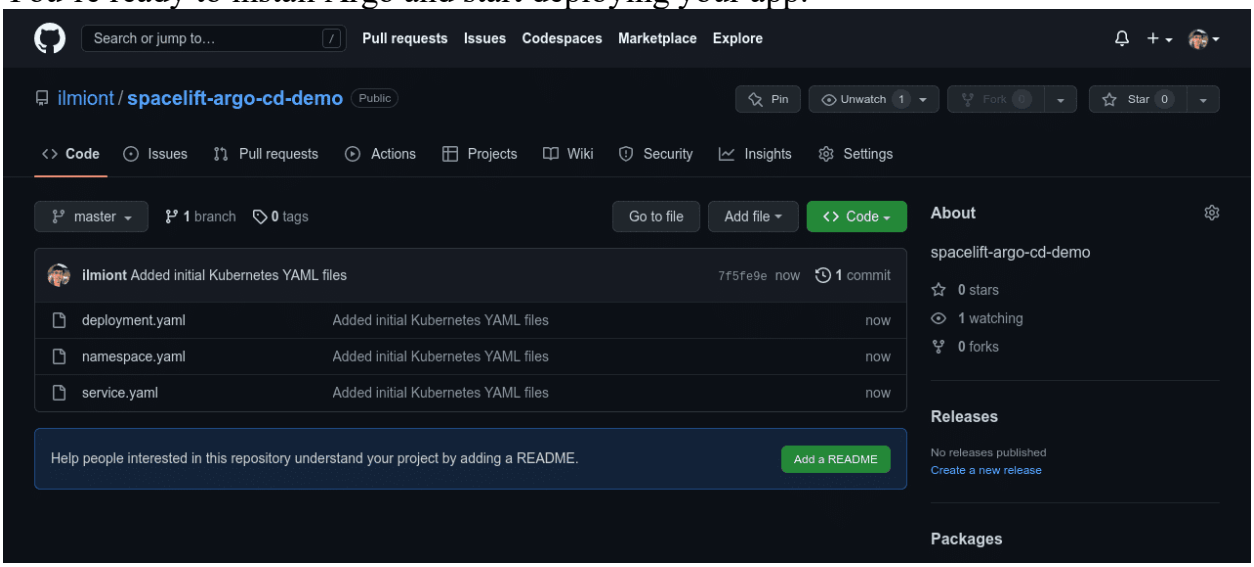
```
apiVersion: v1
kind: Namespace
metadata:
  name: argo-demo
```

Commit your changes to your repository, then push them up to GitHub:

```
$ git add .
```

```
$ git commit -m "Added initial Kubernetes YAML files"
$ git push
```

You're ready to install Argo and start deploying your app.



Get the Argo CLI

Argo's CLI lets you interact with your applications from your terminal. You'll need it later to register your app with your Argo instance.

You can download the latest CLI release from GitHub. Select the right binary for your platform, then make it executable and move it to a location in your path. The following steps work for most Linux systems – substitute the latest version number instead of 2.6.1 below, first:

```
$ wget https://github.com/argoproj/argo-cd/releases/download/v2.6.1/argocd-linux-amd64
$ chmod +x argocd-linux-amd64
$ mv argocd-linux-amd64 /usr/bin/argocd
```

Check you can now run argocd commands:

```
$ argocd version
argocd: v2.6.1+3f143c9
  BuildDate: 2023-02-08T19:18:18Z
```

```
...
```

The CLI's also distributed in Homebrew's package list. Use the `brew install` command to add `argocd` to your system using this method:

```
$ brew install argocd
```

Install Argo In Your Cluster

Next, install Argo in your Kubernetes cluster. This will add the Argo CD API, controller, and Custom Resource Definitions (CRDs).

Begin by creating a namespace for Argo:

```
$ kubectl create namespace argocd
namespace/argocd created
```

Next, use Kubectl to apply Argo CD's YAML manifest to your cluster. You can inspect the manifest before you use it to see the resources that'll be created.

```
$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

It can take several seconds for all of Argo's components to be running in your cluster. Monitor progress by using Kubectl to list the deployments in the `argocd` namespace.

You can continue once the deployments are ready:

```
$ kubectl get deployments -n argocd
NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
argocd-applicationset-controller   1/1     1            1           67s
argocd-dex-server                  1/1     1            1           67s
argocd-notifications-controller    1/1     1            1           67s
argocd-redis                       1/1     1            1           67s
argocd-repo-server                 1/1     1            1           67s
argocd-server                      1/1     1            1           67s
```

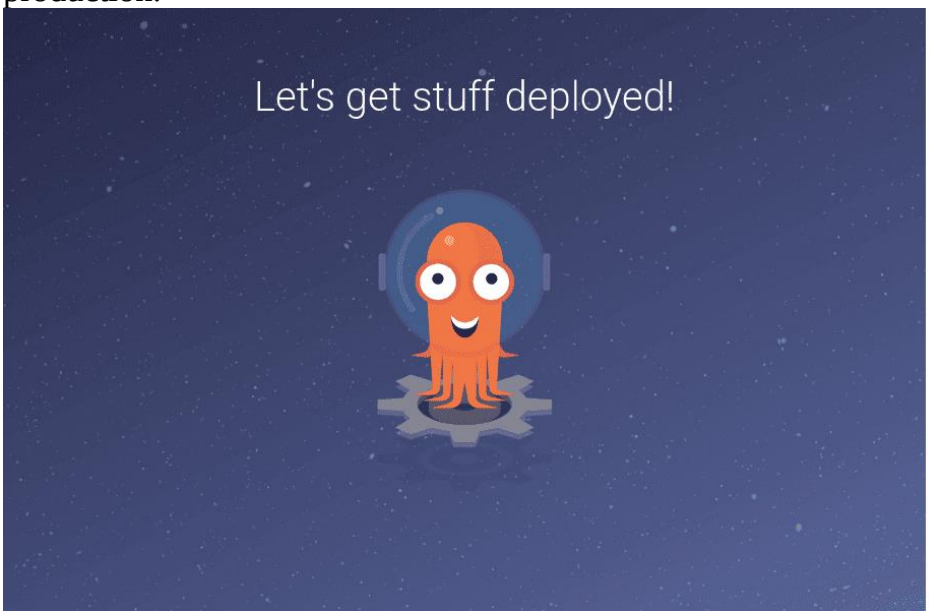
Connecting to Argo

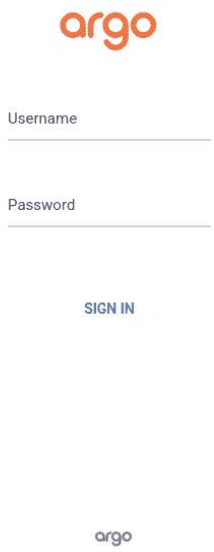
Argo CD doesn't automatically expose its API server on an external IP. You can connect to it by starting a new Kubectl port-forwarding session instead. Open another terminal window and run the following command:

```
$ kubectl port-forward svc/argocd-server -n argocd 8080:443
```

This redirects your local port 8080 to port 443 of Argo's service. Visit localhost:8080 in your browser to access the Argo UI. You'll be warned that the page is insecure because it uses a self-signed certificate.

You can continue with this setup while you're experimenting, but you should follow the detailed steps in the Argo docs to configure TLS with an Ingress route before you move to production.



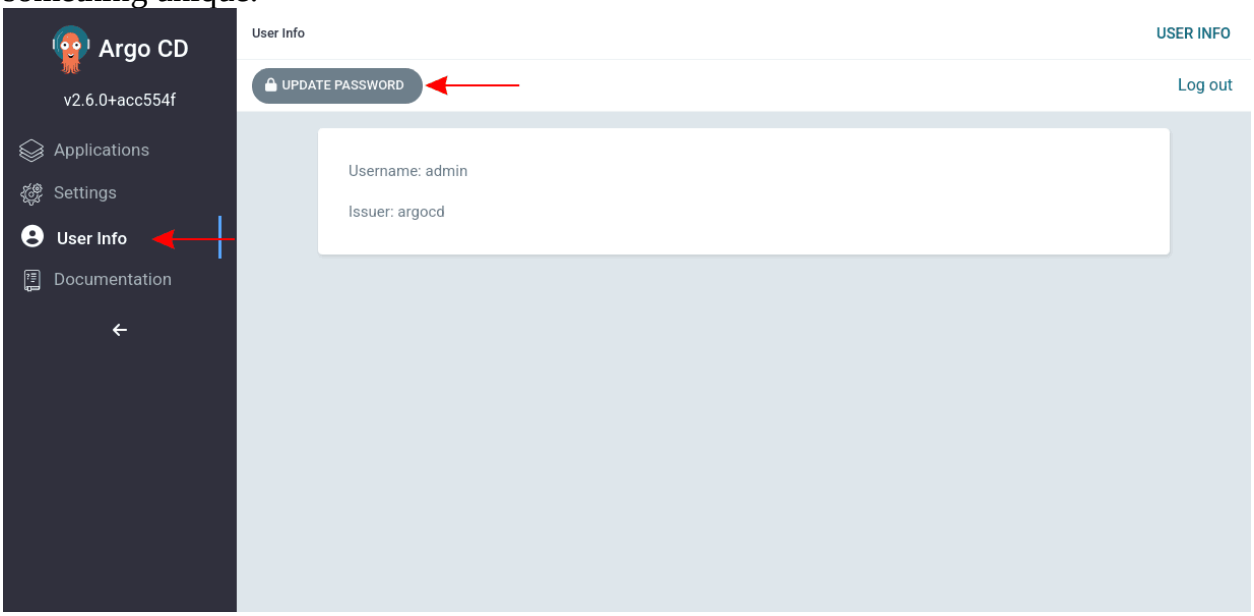


Before you can login, you need to retrieve the password for the default admin user. This is generated automatically during Argo's installation process. You can access it by running the following argocd command:

```
$ argocd admin initial-password -n argocd
zHKv74zvDNtVMaOB
```

Use these credentials to login to Argo.

Once you're in, head straight to the User Info item in the left sidebar, then click the "Update Password" button at the top of the screen. Follow the prompts to change your password to something unique.



Now you can delete the Kubernetes secret that contains the original password for the admin account:

```
$ kubectl delete secret argocd-initial-admin-secret -n argocd
secret "argocd-initial-admin-secret" deleted
```

Login to the CLI

To login to the Argo CLI, run argocd login and supply the API server's URL as an argument:

```
$ argocd login localhost:8080
```

Similarly to the browser warning encountered above, you'll be prompted to accept Argo's built-in self-signed certificate if you haven't configured your own TLS:

```
WARNING: server certificate had error: x509: certificate signed by unknown authority. Proceed insecurely (y/n)?
```

Accept the prompt by typing y and pressing return. You'll then be asked to enter your username and password. The CLI should successfully authenticate to your Argo instance:

```
'admin:login' logged in successfully
Context 'localhost:8080' updated
```

Deploying Your App With Argo

Everything's ready to start deploying apps to Argo! First, run the following CLI command to register your app:

```
$ argocd app create argo-demo \
  --repo https://github.com/<username>/<repo>.git \
  --path . \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace argo-demo
application 'argo-demo' created
```

Let's unpack what's happening here:

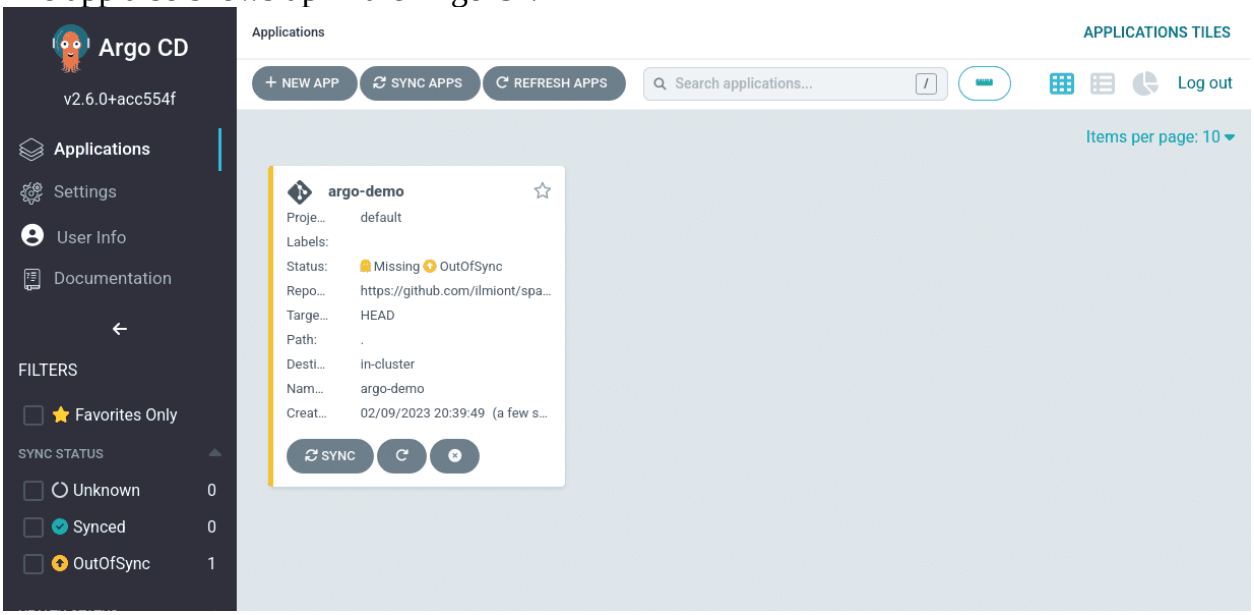
- The `--repo` flag specifies the URL of your Git repository.
- The `--path` flag instructs Argo to search for Kubernetes manifests, Helm charts, and other deployable assets inside this path within your repo. `.` is used here because the example manifests are stored in the repo's root.
- The `--dest-server` flag specifies the URL of the Kubernetes cluster to deploy to. You can use `kubernetes.default.svc` when you're deploying to the same cluster that Argo's running in.
- `--dest-namespace` sets the Kubernetes namespace that your app will be deployed into. This should match the `metadata.namespace` fields set on your resources.

Your app will now be registered with Argo. You can retrieve its details with the `argocd app list` command:

```
NAME  CLUSTER  NAMESPACE  PROJECT  STATUS  HEALTH
SYNCPOLICY  CONDITIONS  REPO  PATH  TARGET
```

```
argocd/argo-demo https://kubernetes.default.svc argo-demo default OutOfSync Missing
<none> <none> https://github.com/ilmiont/spacelift-argo-cd-demo.git
```

The app also shows up in the Argo UI:



Your First Sync

The app shows as “missing” and “out of sync.” Creating the app doesn’t automatically sync it into your cluster. Perform a sync now to have Argo apply the target state currently defined by your repo’s content:

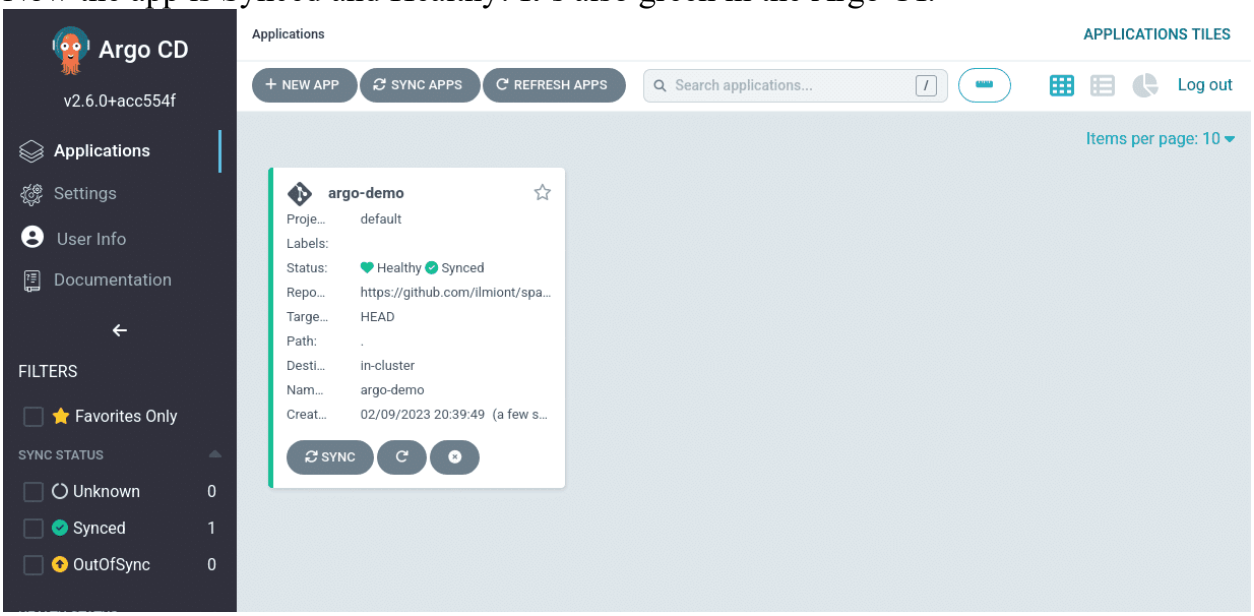
```
$ argocd app sync argo-demo
...
GROUP KIND NAMESPACE NAME STATUS HEALTH HOOK MESSAGE
Namespace argo-demo argo-demo Running Synced namespace/argo-demo created
Service argo-demo nginx Synced Progressing service/nginx created
apps Deployment argo-demo nginx Synced Progressing deployment.apps/nginx created
Namespace argo-demo Synced
```

The sync results display in your terminal. You should see the Namespace, Service, and Deployment objects all get synced into your cluster, as in the command output above. The messages for all three objects confirm they were created successfully.

Repeat the apps list command to check the app's new status:

```
$ argocd app list
NAME                CLUSTER                         NAMESPACE  PROJECT  STATUS  HEALTH
SYNCPOLICY  CONDITIONS  REPO                                                PATH  TARGET
argocd/argo-demo    https://kubernetes.default.svc  argo-demo  default  Synced  Healthy
<none>      <none>      https://github.com/ilmiont/spacelift-argo-cd-demo.git  .
```

Now the app is Synced and Healthy! It's also green in the Argo UI:



As a final proof, use Kubectl to inspect the deployments in the app's namespace. This should confirm that nginx is up and running three replicas:

```
$ kubectl get deployment -n argo-demo
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   3/3     3            3           7m56s
```

Syncing App Updates

Now let's make a change to our app. Modify the spec.replicas field in your deployment.yaml so there's now five Pods in the Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx
  namespace: argo-demo
  labels:
    app.kubernetes.io/name: nginx
spec:
  replicas: 5
  ...
```

Commit and push your changes:

```
$ git add .
$ git commit -m "Run 5 replicas"
$ git push
```

Next, repeat the `argocd app sync` command to apply your changes to your cluster.

Alternatively, you can click the “Sync” button within the user interface.

```
$ argocd app sync argo-demo
GROUP  KIND        NAMESPACE  NAME       STATUS   HEALTH       HOOK  MESSAGE
       Namespace   argo-demo  argo-demo  Running  Synced             namespace/argo-demo unchanged
       Service     argo-demo  nginx      Synced   Healthy            service/nginx unchanged
apps   Deployment  argo-demo  nginx      Synced   Progressing        deployment.apps/nginx configured
       Namespace             argo-demo  Synced
```

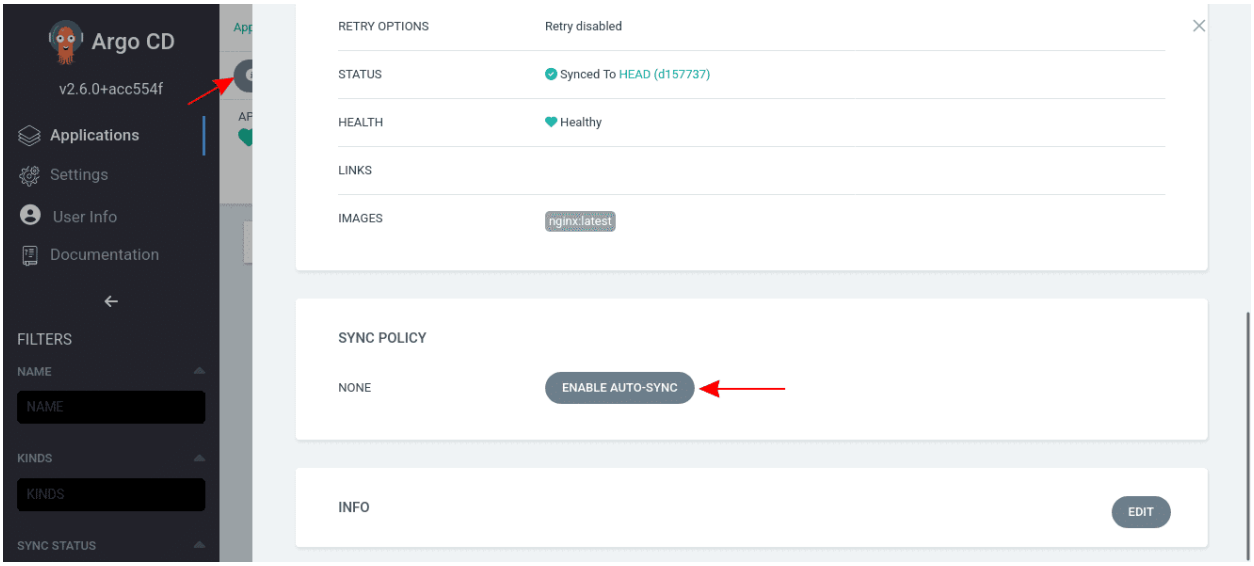
Argo refreshes your app’s target state from the repo, then takes action to transition the live state. The Deployment is reconfigured and now runs five Pods:

```
$ kubectl get deployment -n argo-demo
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   5/5     5            5           12m
```

Enabling Auto-Sync

The change to five replicas didn't apply until you repeated the sync command. Argo can automatically sync changes from your repo though, eliminating the need to issue the command each time. This fully automates your delivery workflow.

You can activate auto-sync for an app by clicking the "App Details" button within the user interface and scrolling down to the "Sync Policy" section. Click the "Enable Auto-Sync" button.



Auto-sync can also be enabled using the CLI by running the following command:

```
$ argocd app set argo-demo --sync-policy automated
```

To test auto-sync out, revert the spec.replicas field back to three replicas:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: argo-demo
  labels:
    app.kubernetes.io/name: nginx
```

```
spec:
  replicas: 3
  ...
```

Commit and push the change:

```
$ git add .
$ git commit -m "Back to 3 replicas"
$ git push
```

This time Argo will automatically detect the repository's state change. You should see your Deployment scale back to 3 replicas within a few minutes of pushing the commit:

```
$ kubectl get deployment -n argo-demo
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   3/3     3            3           23m
```

Auto-sync runs every three minutes by default. You can change this value by modifying Argo's config map, if you need more frequent deployments.

Managing Your App

Argo's CLI and web app offer extensive options for managing and monitoring your deployments. While these are outside the scope of this beginner's tutorial, you can start exploring the CLI commands and UI panels to take control of your app. Most features are implemented in both interfaces.

Clicking your app's tile from the home screen displays an overview that visualizes its components with their current sync and health states. You can view the app's details, edit its config, and view events that describe Argo's activity by clicking the "App Details" button in the top-left.

You can access previous deployments via the “History and Rollback” button. This lists all the syncs that Argo has performed, including an option to restore an older version. If a deployment introduces a bug, you can use this screen to rollback before you push a fix to your repo.

"Every word in this document has been carefully collected, refined, and written by Srujana 😊 with passion and precision.

A blend of research, creativity, and dedication—crafted uniquely with my touch."

"If my work resonates with you or if you find value in the content I share, your support means the world to me.

Feel free to explore my latest blog on Argo CD (or any specific project), and if it speaks to you, your thoughts, likes, or shares would be greatly appreciated.

Your support fuels my growth and helps me bring more valuable content to life."