🥑 How Argo CD Works & Key Features

A Deep Dive into Argo CD's Capabilities

🔄 The Core Engine: Reconciliation

Argo CD's main job is to make your cluster match your Git repo. It does this in a continuous loop called **reconciliation**.

```
      +-----------------+      +------------------+
----->| 1. Fetch & Diff |----->|  2. Is it Synced?  |----
|     +-----------------+      +--------+---------+     |
|                                       | Yes           |
|                                       | (Wait 3 mins) |
|      +-----------------+      +-------+----------+    |
------| 4. Apply to k8s |<-----|  3. OutOfSync?   |<---
      +-----------------+      +------------------+
```

1. **Fetch & Diff:** Argo CD fetches the latest code from Git and compares it to the live state in the Kubernetes cluster.
2. **Check Status:** It determines if the application is `Synced` or `OutOfSync`.
3. **Wait or Act:** If `Synced`, it waits for 3 minutes (by default) and starts over. If `OutOfSync`, it proceeds to the sync phase.
4. **Apply:** The manifests from Git are applied to the cluster to bring it to the desired state.

🚀 Sync Strategy: Manual vs. Automated

You control *when* Argo CD applies changes.

Manual Sync (Default)	**Automated Sync**
You are in control.	**Git is in control.**
Argo CD detects changes and marks the app as `OutOfSync`.	Argo CD detects changes and **immediately** starts syncing.
You must click "Sync" in the UI or run a CLI command.	No human intervention needed.
Good for: Production environments where you want a human to approve changes.	**Good for:** Development or staging environments where you want rapid iteration.

🔧 Sync Options: Fine-Tuning Deployments

Argo CD gives you powerful options to control *how* it syncs.

- **Prune Resources:** This is a critical feature. When enabled, Argo CD will **delete** resources from the cluster if they are removed from the Git repository. This prevents orphaned resources.
- **Dry Run:** You can perform a "dry run" sync to see what *would* happen without actually changing anything. This is great for validating changes.
- **Sync Phases & Waves:** You can use annotations to control the order in which resources are synced. For example, you can ensure a database schema migration Job runs *before* the application Deployment is updated.

❤️ Self-Healing vs. Auto-Sync

These two concepts are often confused but are fundamentally different.

- **Auto-Sync:** Syncs the cluster when there is a **new commit in Git**.
  - *Trigger:* A change in the Git repository.
  - *Purpose:* To apply **new, desired changes**.
- **Self-Healing:** Syncs the cluster when there is a **manual change in the cluster itself**.
  - *Trigger:* A difference between the live state and the last-synced Git commit.
  - *Purpose:* To **revert undesired changes** and enforce the Git state.

Example: If a developer uses `kubectl scale` to manually change the number of replicas, Self-Healing will automatically change it back.

🔄 Rollbacks: Safe and Easy Reversions

Because Git is your source of truth, rolling back is straightforward.

1. **The Git-Native Way (Recommended):**
   - Use `git revert <commit-hash>` to create a new commit that undoes the previous one.
   - Push the new commit.
   - Argo CD will see this as a new change and automatically sync your cluster back to the previous state. This is fully auditable.

Rollbacks: Safe and Easy Reversions

2. **The Argo CD UI Way:**
    - The UI keeps a history of every commit that has been synced.
    - You can click "Rollback" on a previous deployment.
    - Argo CD will apply the manifests from that older commit.
    - **Warning:** This puts your cluster in a state that no longer matches the `HEAD` of your Git branch. It should be used for temporary emergencies only.

🩺 Health Checks: Is My Application Okay?

Argo CD goes beyond `kubectl` to determine if an application is truly healthy.

- **Built-in Logic:** It has smart health checks for most standard Kubernetes resources.
    - For a `Deployment`, it's not "Healthy" until the new replica set is fully rolled out and all its pods are running and available.
    - For a `Service`, it checks if it has a `LoadBalancer` IP (if applicable).
- **Custom Health Checks:** If you have custom resources (CRDs), you can write your own health checks in Lua to tell Argo CD how to understand their health status.

Health Statuses: `Healthy`, `Progressing`, `Degraded`, `Suspended`, `Missing`, `Unknown`.

📊 The Argo CD UI: Your GitOps Dashboard

Argo CD provides a powerful web interface to visualize and manage your applications.

Argo CD UI

- **A) Application Tiles:** One for each application, showing its real-time sync and health status.
- **B) Resource Tree:** Visualizes all the Kubernetes resources that make up your application and how they relate to each other.
- **C) Sync Status:** Clearly indicates if the live state matches the desired state in Git.
- **D) Health Status:** Shows the overall health of the application based on its components.

✅ Summary

- Argo CD's **reconciliation loop** is the core engine that keeps your cluster synced with Git.
- You can choose between **manual** and **automated** sync strategies.
- **Self-healing** reverts manual changes, while **auto-sync** applies new commits.
- Rollbacks are safe and auditable, especially when done via `git revert`.
- Argo CD has **deep health checks** to truly understand application status.
- The **UI** provides a powerful dashboard for visualizing and managing your deployments.

🙏 Questions?