ArgoCD

Argo CD is a declarative, GitOps continuous delivery (CD) tool for Kubernetes. It’s designed to automate the deployment of applications to Kubernetes clusters based on the state defined in a Git repository. Here’s a detailed explanation

**Key Concepts**

1. **GitOps**:
   * **GitOps** is a practice where Git repositories are the single source of truth for the system's desired state.
   * With GitOps, changes to your application or infrastructure are made through Git commits, and those changes are automatically applied to your Kubernetes cluster by a tool like Argo CD.
2. **Declarative Configuration**:
   * Argo CD uses declarative configuration. You define the desired state of your applications and infrastructure as YAML or JSON files and Argo CD ensures that your Kubernetes cluster reflects that state.
   * It continuously monitors your Kubernetes cluster and compares the actual state with the desired state defined in the Git repository.

**How Argo CD Works**

1. **Application Definition**:
   * In Argo CD, an **Application** is a custom Kubernetes resource representing a deployed application.
   * An Application resource defines where the source code is (e.g., a Git repository), the path to the manifests, and where it should be deployed (e.g., a specific Kubernetes namespace).
2. **Syncing**:
   * **Syncing** is the process of making the actual state of the application in the Kubernetes cluster match the desired state in the Git repository.
   * Argo CD can sync applications automatically or manually. During an automatic sync, Argo CD will detect changes in the Git repository and apply them to the cluster.
3. **Self-Healing**:
   * Argo CD can automatically fix any drift between the actual state in the cluster and the desired state in the Git repository.
   * If someone manually changes a resource in the cluster that differs from what’s defined in Git, Argo CD will revert it back to match the Git state.
4. **Multi-Cluster Support**:
   * Argo CD can deploy applications to multiple clusters from a single Argo CD instance.
   * This is useful for managing applications across different environments (e.g., development, staging, production) or across different Kubernetes clusters.

### Argo CD in CI/CD Pipelines

Argo CD is typically integrated into CI/CD pipelines as the CD component. The CI process (e.g., using Jenkins, GitHub Actions, or GitLab CI) builds the application, runs tests, and updates the Git repository with new manifests. Argo CD then takes over, deploying the new manifests to the Kubernetes cluster.

### Installation Of ArgoCD

Installing Argo CD involves several steps, including setting up a Kubernetes cluster (if you don’t already have one), deploying Argo CD to the cluster, and then accessing the Argo CD UI. Here’s a step-by-step guide:

### 1. Prerequisites

* **Kubernetes Cluster**: You need a running Kubernetes cluster (either local like Minikube, Kind, or a cloud-managed service like GKE, EKS, or AKS).
* **kubectl**: Ensure kubectl is installed and configured to access your Kubernetes cluster.

### 2. Install Argo CD

#### Step 1: Create the argocd Namespace

First, create a namespace where Argo CD will be installed:

kubectl create namespace argocd

#### Step 2: Install Argo CD

You can install Argo CD using the official YAML manifest provided by the Argo CD project:

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

This command will deploy all the necessary components for Argo CD, including the API server, controller, repository server, and UI.

### 3. Accessing the Argo CD API Server

Argo CD exposes a web UI that you can use to manage your applications. To access this UI, you can expose the Argo CD API server using a LoadBalancer service, NodePort, or port-forwarding.

#### Option 1: Port-Forwarding (for Local Development)

Use port-forwarding to access the Argo CD server locally:

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

Now, you can access the Argo CD UI by navigating to https://localhost:8080 in your browser.

#### Option 2: Expose the Service (for Production or Remote Access)

Alternatively, you can expose the Argo CD server via a LoadBalancer or NodePort service:

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

This will expose the service publicly (you’ll get an external IP if using a cloud provider) or via a specific port on your nodes.

### 4. Log in to the Argo CD UI

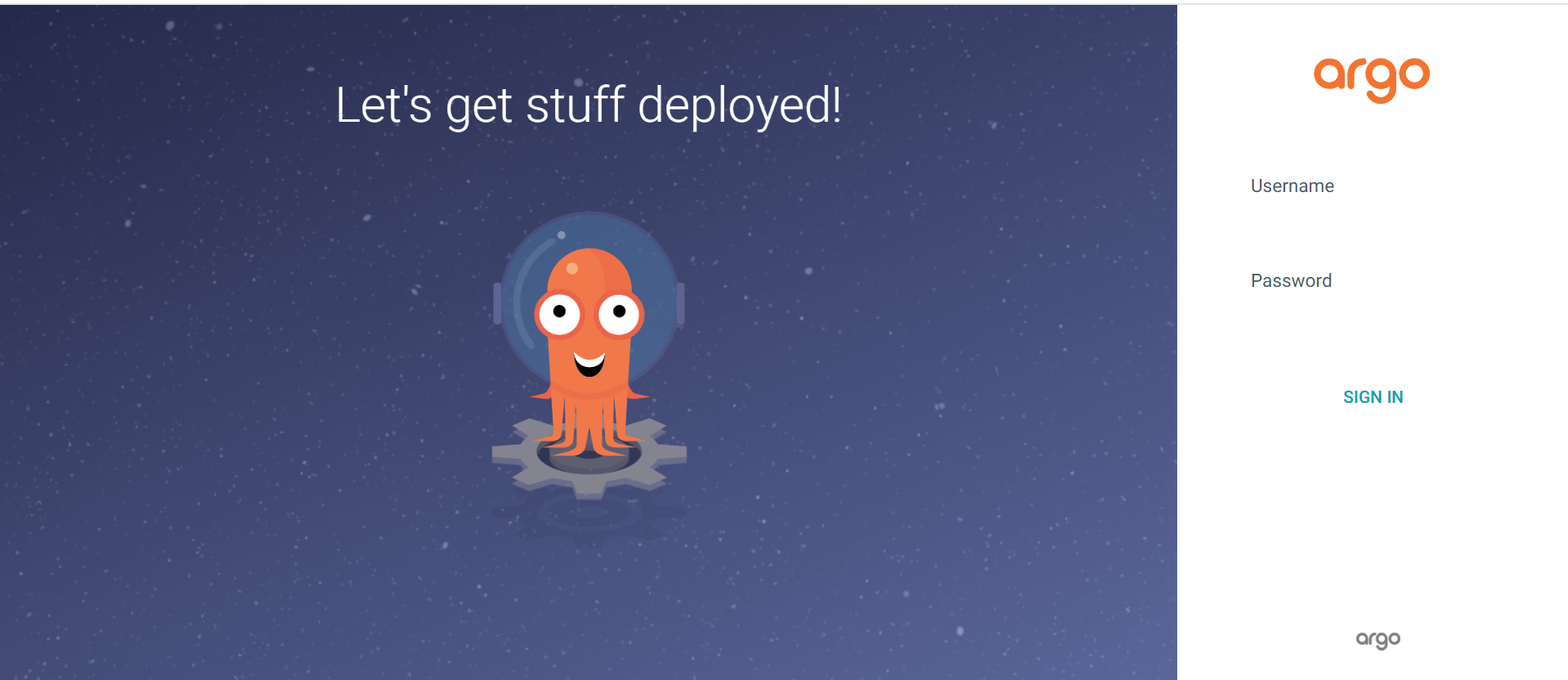
By default, the initial admin password is the name of the Argo CD server pod. You can retrieve it using the following command:

kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d

This command will output the password for the admin user.

* **Username**: admin
* **Password**: (output of the above command)

You can now log in to the Argo CD UI using these credentials.



### 5. Change the Admin Password

For security reasons, it’s recommended to change the initial admin password:

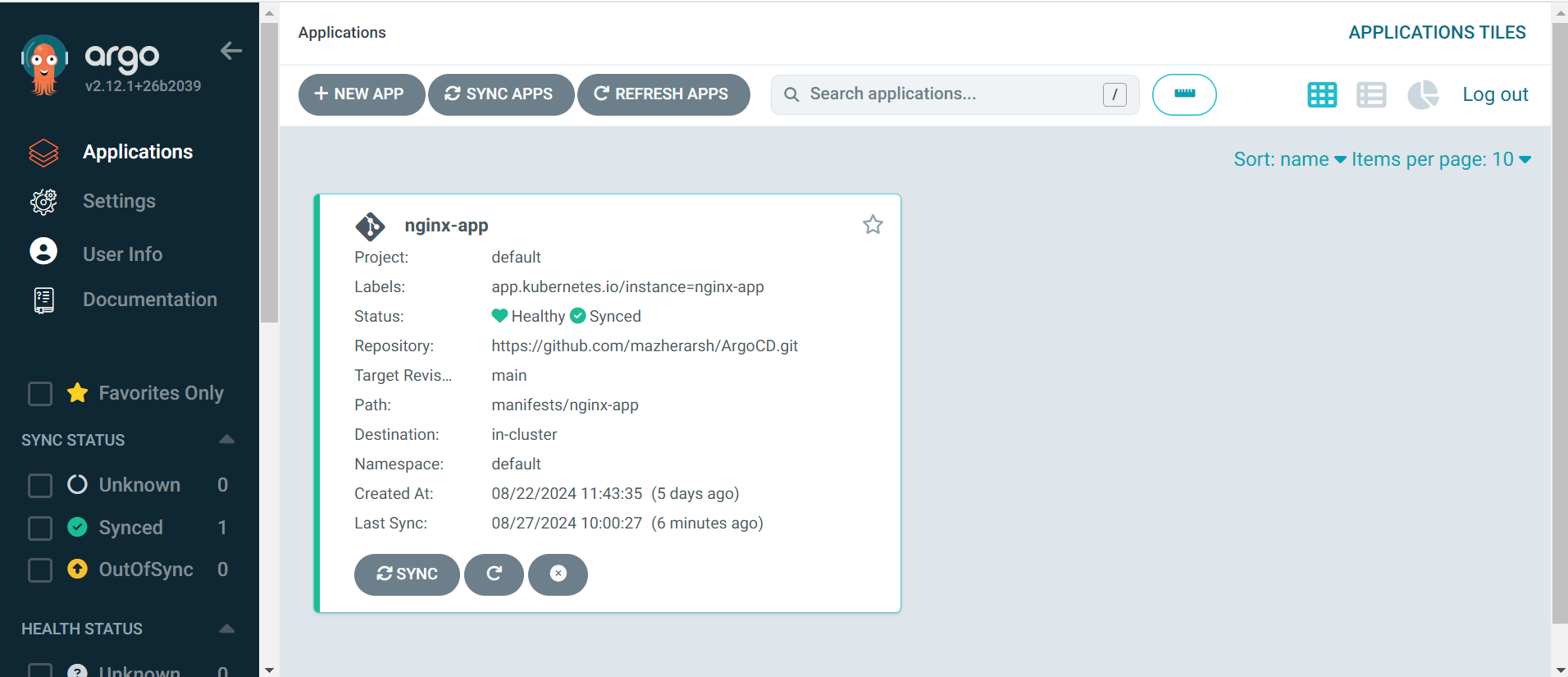
argocd login <ARGOCD\_SERVER>

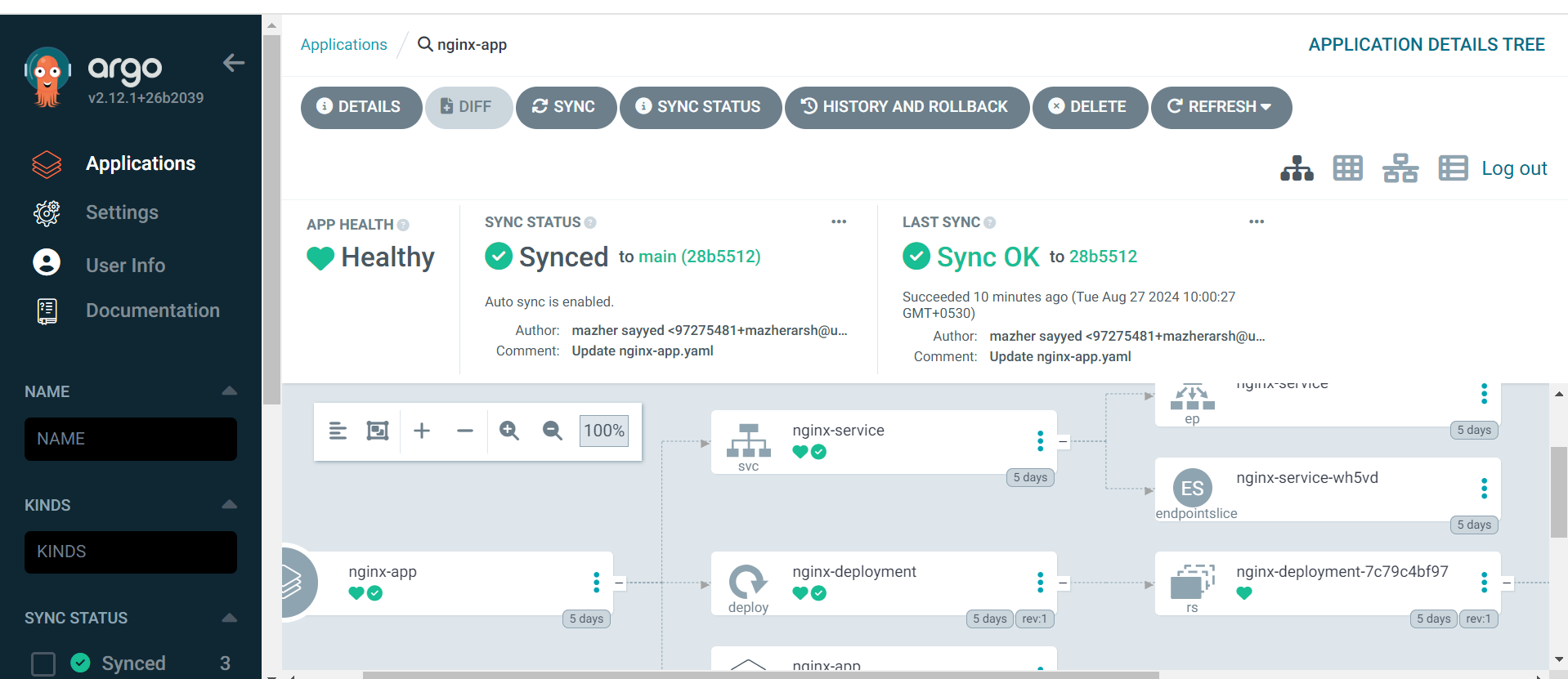
argocd account update-password

Replace <ARGOCD\_SERVER> with the appropriate address (e.g., localhost:8080 if using port-forwarding).

### 6. Create Your First Application

Once logged in, you can start creating your first application. This involves pointing Argo CD to a Git repository containing your Kubernetes manifests, defining the destination cluster and namespace, and setting up synchronization policies.

Sample Example ScreenShot NGINX application  
 



### Argo CD templates directly with YAML Overview of Using Plain YAML with Argo CD

1. **Simplicity**:
   * **Plain YAML** is more straightforward to manage for those who prefer a simpler approach without the complexity of Helm charts.
   * You avoid the additional layer of abstraction that Helm introduces, which can sometimes make debugging more challenging.
2. **Version Control**:
   * With plain YAML, every aspect of your Kubernetes resources is explicitly defined. This makes it easier to track changes over time and understand exactly what’s being deployed.
   * Argo CD can directly monitor your Git repository, where all your YAML files are stored, and automatically synchronize the state defined in those files with your Kubernetes cluster.
3. **GitOps and Declarative Configuration**:
   * Argo CD excels in a GitOps workflow where the entire state of your Kubernetes resources is defined declaratively in a Git repository.
   * You can organize your YAML files in a repository, and Argo CD will ensure that the live state in your cluster matches the desired state in Git.
4. **Flexibility**:
   * While Helm is powerful for parameterizing deployments across environments (e.g., dev, staging, production), plain YAML can still be flexible when combined with other tools like Kustomize, which Argo CD also supports.
   * You can create multiple YAML files for different environments or overlay specific configurations using Kustomize if needed.
5. **Reduced Complexity**:
   * By using plain YAML, you eliminate the need to manage Helm releases and charts. This reduces the operational overhead and potential points of failure.
   * YAML files are more transparent, as they don't require knowledge of Helm-specific syntax and templating logic, making them more accessible to a broader range of users.
6. **Direct Integration with Argo CD**:
   * Argo CD natively supports syncing plain YAML files, so there's no need to worry about Helm hooks, values files, or chart dependencies.
   * Argo CD can work directly with your YAML files to deploy, monitor, and maintain your Kubernetes applications.

**When to Consider Using Plain YAML Over Helm**

1. **Small to Medium Applications**:
   * For simpler applications or environments where there’s no need for extensive templating or environment-specific customization, plain YAML might be more efficient.
2. **Less Frequent Changes**:
   * If your application configurations are relatively static or you have a straightforward deployment pipeline, plain YAML is likely sufficient.
3. **Security Considerations**:
   * Reducing the number of tools in your pipeline (e.g., eliminating Helm) can also reduce the attack surface, which might be a consideration in highly secure environments.
4. **Ease of Onboarding**:
   * Teams with less experience in Kubernetes may find it easier to start with plain YAML rather than learning Helm's templating syntax and commands.

**Conclusion**

Using plain YAML with Argo CD is a fully supported and often advantageous approach, particularly for simpler use cases or teams that prefer a more transparent and straightforward configuration management process. By leveraging the declarative nature of Kubernetes and Argo CD's GitOps capabilities, you can achieve robust continuous delivery without the need for Helm.