**GitOps Fundamentals**

**• What is GitOps?**

◦ GitOps uses Git as a single source of truth to deliver applications and infrastructure.

◦ It standardizes the process of deploying changes to Kubernetes resources, whether it's an application deployment or infrastructure updates (e.g., node configuration, namespaces, pods, admission controllers).

◦ It addresses the problem of lack of tracking and versioning for deployments, unlike source code which typically has proper version control.

◦ How it works: A DevOps engineer or actor submits a pull request to a Git repository containing Kubernetes YAML manifests. This pull request is reviewed and merged. Then, a GitOps controller (like Argo CD) reads this change and deploys it onto the Kubernetes cluster.

**• Why GitOps? (Drawbacks of Traditional CD)**

◦ Traditional CD pipelines (using shell scripts, Python scripts, Ansible, or CI orchestrator plugins) lack a tracking mechanism for deployments.

◦ They cannot easily identify or revert manual changes made directly on the Kubernetes cluster by developers or operations teams.

◦ Without GitOps, there's no versioning or auditing for Kubernetes cluster changes, making it hard to know who made what change or when.

◦ For CI, Git is the source of truth, but for CD, there's often no single source of truth in traditional setups.

**• Principles of GitOps**

**◦ Declarative:** The desired state of a system managed by GitOps must be expressed declaratively (e.g., in YAML manifests). What you see in Git is what is deployed on your cluster.

**◦ Versioned & Immutable:** Resources tracked in Git are versioned and immutable, providing a clear history of changes. GitOps is tightly coupled with the versioning concept and can even work with other versioned storage like S3 buckets, not just Git.

**◦ Pulled Automatically:** Changes made to the Git repository are automatically pulled and updated by the GitOps controller onto the Kubernetes cluster. This can be a pull mechanism or triggered by webhooks (push mechanism).

**◦ Continuously Reconciled:** The GitOps controller continuously watches for changes in both the Git repository and the Kubernetes cluster. If any direct manual change is made to the cluster, the GitOps controller will override that change to match the desired state in Git. This provides auto-healing behavior.

**• Advantages of GitOps**

**◦ Security:** GitOps overrides unwanted manual changes on the cluster, enhancing security by ensuring only changes from Git are applied.

**◦ Versioning:** All changes are version-controlled in Git, providing a clear audit trail.

**◦ Auto-upgrades:** Automatic deployment of changes from Git ensures applications are always updated to the desired version.

**◦ Auto-healing Behavior:** GitOps automatically reverts any drifts or manual modifications on the cluster back to the state defined in Git.

**• Is GitOps Kubernetes-only?**

◦ By principle, no, GitOps is not exclusively for Kubernetes.

◦ However, most popular GitOps tools like Argo CD and Flux CD currently target Kubernetes clusters.

Argo CD Basics

**• What is Argo CD?**

◦ Argo CD is a popular GitOps-based tool in the continuous delivery space.

◦ It is a Kubernetes controller designed to keep the state between Git (or any version control system) and Kubernetes in sync.

◦ Argo CD listens to a custom resource called "Applications" to perform actions.

**• Argo CD Architecture Components**

**◦ Server Component (API Server):** This is the primary interaction point for users via the UI or CLI. It takes all requests from users.

**◦ Repo Server:** Responsible for interacting with the Git-based Version Control System (e.g., GitHub, Bitbucket) to get the state of the Git repository. Argo CD only supports Git-based VCS.

**◦ Application Controller:** The core of Argo CD, responsible for the reconciliation logic. It compares the state from Git (via Repo Server) with the state of the Kubernetes cluster and reports if there's any out-of-sync situation. It then syncs the cluster to match the Git state.

**◦ Redis:** Used for caching state and information, ensuring persistence even if the application controller goes down.

**◦ Dex Server:** Handles SSO (Single Sign-On) capabilities, allowing integration with OIDC or OAuth identity providers (e.g., Google, Facebook).

**◦ Notifications Controller:** Introduced in Argo CD version 2.5/2.6, it handles notifications.

**◦ Application Sets Controller:** A way of generating multiple Argo CD Applications automatically, useful for managing deployments across many Kubernetes clusters. It uses different generators (e.g., cluster-based, matrix-based).

• Argo CD Installation Methods

**◦ Plain Manifests:** One of the easiest ways to install. It involves applying a single YAML manifest from the Argo CD documentation, which creates necessary components like custom resource definitions, service accounts, roles, deployments, services, and network policies in the argo-cd namespace.

**◦ Helm Charts:** Argo CD can be installed using community-maintained Helm charts from the Argo Helm repository.

**◦ Operator:** Argo CD can also be installed using a Kubernetes operator.

**• Argo CD Basic Usage (Hands-On)**

**◦ Accessing the UI:**

▪ By default, the Argo CD server service is in ClusterIP mode.

▪ It can be changed to NodePort mode to access from a browser.

▪ For local minikube setups, minikube service argo-cd-server -n argo-cd can create a tunnel and provide an IP address.

▪ On cloud providers, it can be exposed as a LoadBalancer type service.

▪ The server can be configured to run in insecure HTTP mode for demos, but HTTPS is recommended for organizations.

**▪ Initial Login:** Username is admin. The password is in a Kubernetes secret named argo-cd-initial-admin-secret within the argo-cd namespace. This base64-encoded secret needs to be decoded to get the password.

**◦ Using the CLI:**

▪ The Argo CD CLI tool is similar to kubectl for interacting with Argo CD.

▪ It can be installed via brew install argocd (for Mac) or other OS-specific commands.

**▪ Logging in via CLI:** Requires argocd login <Argo CD API server URL:Port> and providing the admin credentials.

**▪ Creating an Application via CLI**: Use argocd app create <app-name> --repo <git-repo-url> --path <folder-path> --dest-namespace <namespace> --dest-server <cluster-url>.

▪ Syncing via CLI: argocd app sync <app-name>. Automatic sync can also be configured.

▪ Getting Application Status: argocd app get <app-name> or kubectl get application -n argo-cd (if application CRD is enabled in the cluster where Argo CD is installed).

**• Argo CD Application Concepts**

◦ An "Application" in Argo CD refers to a custom resource that Argo CD listens to. It's not the microservice you are deploying, but Argo CD's way of defining what to manage.

◦ When creating an application, you define: Application Name, Project (for RBAC), Sync Policy (e.g., Automatic for continuous reconciliation), Repository URL, Path (folder within the repository), Destination Cluster URL, and Destination Namespace.

**◦ Automatic Sync Policy:** Ensures that whenever changes are made in the Git repository, Argo CD automatically deploys them to the Kubernetes cluster. It also enables auto-healing.

◦ Prune Resources and Self Heal are other configurations that can be set for applications.

◦ Argo CD is not opinionated about how you store your Kubernetes YAML manifests in Git. It **supports:**

▪ Plain Kubernetes Manifests (e.g., deployment.yaml, service.yaml).

▪ Helm Charts (applications structured in Helm format with Chart.yaml, values.yaml, templates).

▪ Ksonnet format.

▪ Kustomize (using base and overlays concepts).

▪ Custom Plugins.

◦ This flexibility means DevOps engineers can structure manifests in the format their organization prefers, and Argo CD will seamlessly work with them.

**• Argo CD Deployment Models**

**◦ Standalone Model (Single Instance Model):** Each Argo CD instance is installed on a specific Kubernetes cluster and is responsible only for managing that particular cluster.

**▪ Pros**: Localized management, potentially less complex resource management for the Argo CD instance itself.

▪ **Cons**: Requires managing multiple Argo CD instances (upgrades, configuration, DR) across different clusters.

**◦ Hub-Spoke Model:** A centralized Kubernetes cluster (Hub cluster) has a single Argo CD instance installed. This instance then manages deployments to multiple other Kubernetes clusters (Spoke clusters).

**▪ Pros:** Easier centralized management of Argo CD (upgrades, configuration, backups, DR), especially for organizations with a centralized DevOps team.

**▪ Cons:** The single Argo CD instance can become resource-intensive if managing a very large number of clusters and Kubernetes objects (e.g., 50,000-60,000 objects), requiring high availability and dynamic sharding configurations.

**▪ Adding Clusters to Hub:** Spoke clusters are added to the Hub Argo CD instance using the Argo CD CLI command argocd cluster add <context-name> --server <argo-cd-server-address>.

In essence, GitOps, powered by tools like Argo CD, turns your Git repository into the ultimate blueprint for your infrastructure and applications. Just as a conductor keeps an orchestra in perfect harmony with the score, Argo CD continuously ensures your live Kubernetes clusters are in perfect sync with the declarative state defined in your Git repository. If any musician plays out of tune, the conductor brings them back into line, making sure the performance (your deployed applications) remains exactly as intended.