Detailed Handout: Understanding Argo CD – Kubernetes GitOps Made Simple

# 1. Introduction

Modern software delivery requires fast, repeatable, and reliable deployments. Kubernetes helps manage containers at scale, but deploying applications consistently across environments remains a challenge.  
  
This is where GitOps and Argo CD come into play.  
  
- GitOps → Uses Git repositories as the single source of truth for both infrastructure and applications.  
- Argo CD → A Kubernetes-native continuous deployment tool that automates the process of pulling changes from Git and applying them to Kubernetes clusters.

# 2. What is Argo CD?

Argo CD is:  
- A Kubernetes controller that continuously monitors apps.  
- A pull-based CD tool (unlike Jenkins/Spinnaker which push).  
- Designed to synchronize live cluster state with the desired state in Git.  
  
Why Use It?  
- Application + Infrastructure Management → Handles both workloads and cluster configs.  
- Drift Detection → Identifies differences between Git and live cluster.  
- Auditability → Every change is recorded in Git.

# 3. Key Features

Deployment Options:  
- Manual sync  
- Automatic sync (with auto-heal capability)  
  
Interfaces:  
- Web UI → Graphical view of applications & sync status  
- CLI → For scripting & automation  
  
Security & Governance:  
- RBAC → Role-based access  
- SSO → GitHub, LDAP, OIDC  
- Multi-cluster support  
  
Integration:  
- Webhooks from GitHub/GitLab/BitBucket  
- Works with Helm, Kustomize, YAML, Jsonnet

# 4. GitOps with Argo CD

GitOps workflow with Argo CD:  
1. Developer updates code → Commit & push.  
2. CI pipeline builds image → Pushes to container registry.  
3. Manifests updated in Git → PR merged.  
4. Argo CD detects changes → Syncs with Kubernetes.  
5. Cluster state updated → Verified against Git.  
  
Diagram (described): Git repo → CI pipeline → Container registry → Argo CD → Kubernetes cluster.

# 5. Argo CD Workflow (Step by Step Example)

Scenario: Deploying a new microservice to Kubernetes using Argo CD.  
  
1. Create Git repo with Kubernetes manifests (deployment.yaml, service.yaml).  
2. Configure Argo CD with repository URL.  
3. Argo CD clones repo → Renders manifests.  
4. Sync: Argo CD applies resources to Kubernetes.  
5. Monitor: UI shows status (Synced or OutOfSync).  
6. Drift Handling: If someone manually runs kubectl edit deployment, Argo CD will:  
 - Mark app as OutOfSync.  
 - Optionally auto-revert to match Git.

# 6. Argo CD Architecture

Core Components:  
  
- API Server  
 - Exposes gRPC/REST APIs for CLI & UI  
 - Handles RBAC, authentication, and webhooks  
  
- Repository Server  
 - Clones and caches Git repos  
 - Renders manifests (Helm, Kustomize, YAML)  
  
- Application Controller  
 - Continuously compares desired vs live state  
 - Applies sync or marks drift  
 - Supports hooks for pre/post deployment actions  
  
Diagram (described): Git → Repository Server → Application Controller → Kubernetes API Server → Cluster Resources.

# 7. Advanced Features

- Sync Policies: Manual, Auto, or Auto with Prune & Self-heal.  
- Health Checks: Built-in app health monitoring (e.g., Pods ready, Services running).  
- Rollbacks: Rollback to a previous Git commit.  
- Multi-tenancy: Different teams manage different applications safely.

# 8. Expert Best Practices

- Secrets Management:  
 - Avoid hardcoding secrets.  
 - Use Vault, SOPS, or External Secrets Operator.  
  
- CI/CD Integration:  
 - Use Jenkins/GitHub Actions for CI (build, test, push image).  
 - Use Argo CD for CD (sync manifests).  
  
- Policy Enforcement:  
 - Integrate OPA/Gatekeeper for compliance.  
  
- Testing Manifests Before Commit:  
 - Run kubectl apply --dry-run locally.  
 - Use Helm lint or Kustomize build validation.  
  
- Avoid Drift:  
 - Never apply changes directly via kubectl.  
 - Enable auto-sync + self-heal.

# 9. Installation Methods

Core Installation (minimal)  
kubectl create namespace argocd  
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/core-install.yaml  
  
Full Installation (multi-tenant)  
- Includes API server, UI, RBAC, SSO.  
- Supports High Availability.  
  
Helm Installation  
helm repo add argo https://argoproj.github.io/argo-helm  
helm install argocd argo/argo-cd -n argocd  
  
Kustomize Installation  
Maintain remote manifests and patch for customizations.

# 10. Argo CD in GitOps Ecosystem

- Git: Stores desired state.  
- CI tools: Build & push images.  
- Argo CD: Syncs Git → Kubernetes.  
- Kubernetes: Executes workloads.  
- Monitoring tools (Prometheus/Grafana): Observability.

# 11. Related Argo Projects

- Argo Rollouts → Canary, Blue-Green, Progressive Delivery.  
- Argo Workflows → Orchestration engine (useful for ML/data pipelines).  
- Argo Events → Event-driven automation for triggering workflows.

# 12. Hands-on Lab Exercises

Lab 1: Install Argo CD (Core)  
1. Create namespace:  
 kubectl create namespace argocd  
2. Apply manifests:  
 kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml  
3. Access UI:  
 - Port-forward service → kubectl port-forward svc/argocd-server -n argocd 8080:443  
 - Open https://localhost:8080  
  
Lab 2: Deploy an Application  
1. Create application.yaml with Argo CD Application CRD.  
2. Apply it: kubectl apply -f application.yaml  
3. Verify in Argo CD UI → App status shows Synced & Healthy.  
  
Lab 3: Drift Detection  
1. Manually edit a deployment: kubectl scale deployment guestbook-ui --replicas=5  
2. Argo CD marks app OutOfSync.  
3. Click Sync or rely on auto-heal to revert back.

# 13. Summary – Why Argo CD?

- Kubernetes-native → Works with Kubernetes API directly.  
- GitOps-first → Git as single source of truth.  
- Prevents Drift → Detects and remediates configuration mismatches.  
- Enterprise-ready → RBAC, SSO, multi-tenancy, HA.  
- Extensible → Works with Helm, Kustomize, OPA, Vault, and other tools.  
  
Argo CD enables predictable, secure, and automated Kubernetes deployments, making it one of the most important tools for modern DevOps teams.