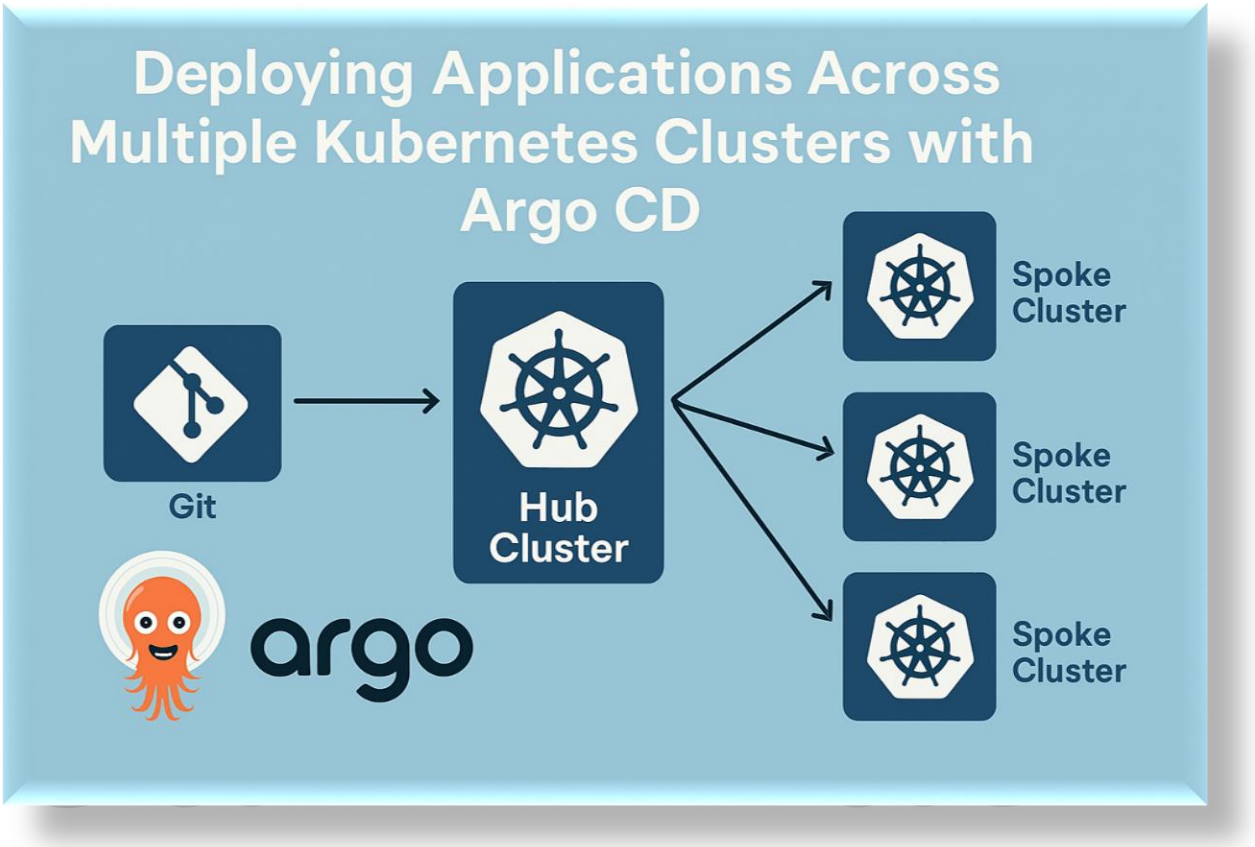
PROJECT : Multi-Cluster Kubernetes Deployment Using GitOps

Introduction



In today's rapidly evolving cloud-native ecosystem, Kubernetes has emerged as the de facto platform for container orchestration. Organizations leverage Kubernetes to deploy and manage applications with agility and scalability. However, as applications grow in complexity and scale, managing Kubernetes deployments across multiple clusters becomes a significant operational challenge.

Traditional Continuous Integration/Continuous Delivery (CI/CD) pipelines—while powerful for building and deploying software—often fall short when it comes to ensuring consistency and stability in production environments. They typically execute deployment scripts or workflows but lack continuous state reconciliation, which leads to configuration drift, unexpected outages, and security risks.

GitOps presents a paradigm shift by using Git repositories as the single source of truth for declarative infrastructure and application configuration. This approach not only provides full traceability and version control but also enables automated and continuous reconciliation between the desired state stored in Git and the actual cluster state.

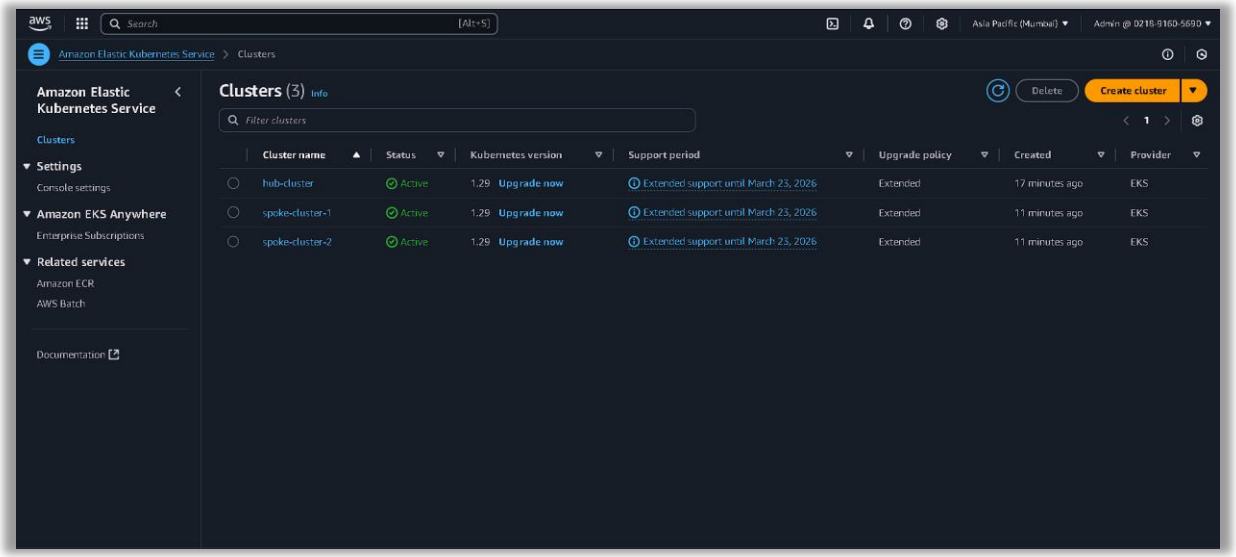
Argo CD, an open-source GitOps tool built specifically for Kubernetes, has become a cornerstone for organizations looking to adopt this model. By continuously monitoring Git repositories and Kubernetes clusters, Argo CD ensures that clusters remain in sync with the declared configurations, improving operational stability and governance.

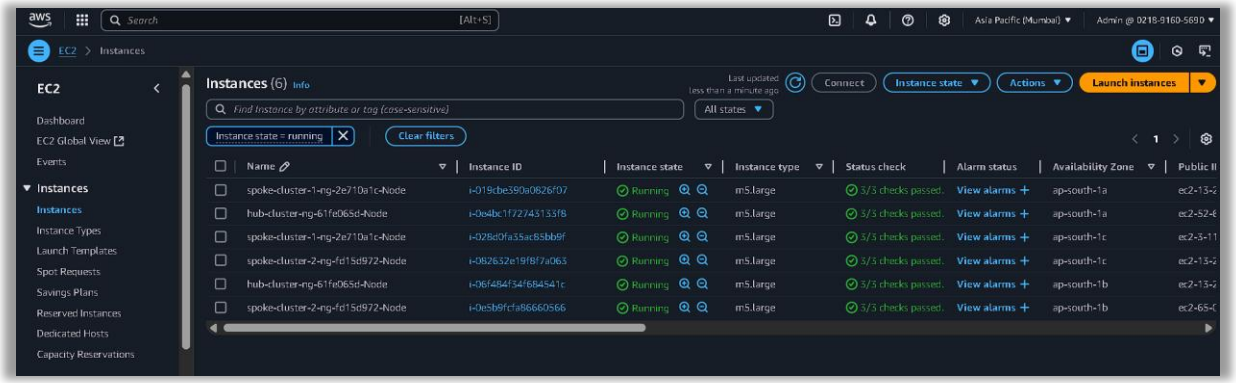
Core GitOps Principles and Benefits

GitOps is founded on several core principles that redefine how infrastructure and application deployments are managed:

- **Declarative State Management:** Kubernetes manifests and Helm charts, describing the desired state of clusters and applications, are stored and versioned in Git repositories. This allows teams to declaratively specify exactly what the infrastructure and application configurations should be.
- **Automated Reconciliation and Drift Detection:** Argo CD continuously compares the live state of clusters against the Git-stored desired state. If any manual or unexpected change occurs directly on the cluster (known as configuration drift), Argo CD detects this and either alerts operators or automatically corrects the state.
- **Improved Auditability and Security:** Every change must go through Git commits and pull requests, enabling full traceability of who changed what and when. This audit trail is crucial for compliance, security reviews, and rollback capabilities.
- **Simplified Recovery and Rollbacks:** Since Git history captures all state changes, rolling back to a previous stable configuration becomes as simple as reverting commits and letting Argo CD synchronize the cluster back.
- **Self-Healing Clusters:** Argo CD's reconciliation loop acts as a self-healing mechanism that prevents configuration drift from persisting, reducing the risk of unexpected outages and manual troubleshooting.

Addressing Multicluster Complexity





Large enterprises rarely operate with a single Kubernetes cluster. Instead, multiple clusters are deployed to:

- Isolate workloads for different environments (Development, QA, Staging, Production).
- Support geographical distribution for latency, compliance, or disaster recovery.
- Manage clusters dedicated to specific teams, business units, or features.
- Increase security boundaries between critical and non-critical workloads.

This multicluster landscape introduces deployment complexities:

- **Consistency Challenges:** Deploying the same application version and configuration across clusters requires precision to avoid divergence.
- **Operational Overhead:** Managing many clusters independently demands significant effort for upgrades, security patches, and monitoring.
- **Access and Security Management:** Different clusters may have distinct access policies, requiring granular control over who can deploy and what can be changed.
- **Disaster Recovery and Scalability:** Centralized control simplifies backup, recovery, and scaling strategies.

GitOps, combined with Argo CD's multicluster capabilities, offers an elegant solution to these challenges by enabling centralized deployment management while maintaining cluster-level independence.

Argo CD Deployment Architectures

Two primary architectures exist for deploying Argo CD in multicluster environments:

Hub-Spoke (Centralized) Model

In this architecture:

- A single "Hub" Kubernetes cluster hosts one Argo CD instance.
- The Hub Argo CD instance manages deployment and synchronization across multiple "Spoke" clusters.
- Spoke clusters are registered as remote targets in the Hub Argo CD.
- This model centralizes visibility, control, and operational management.

- Simplifies cluster lifecycle management and reduces redundant Argo CD instances.
- Ideal for organizations with a central platform or DevOps team controlling deployments.

Advantages:

- Unified dashboard to view application health across all clusters.
- Simplified upgrades and backup procedures as only one Argo CD instance needs management.
- Easier enforcement of global policies and governance.

Challenges:

- Potential performance bottlenecks when managing a very large number of clusters or applications.
- Network dependencies between Hub and Spoke clusters.
- Increased blast radius if Hub cluster faces issues.

Standalone (Decentralized) Model

In this setup:

- Each Kubernetes cluster runs its own independent Argo CD instance.
- Clusters manage their own deployments autonomously.
- Better isolation and fault tolerance.
- Enables distributed operations and autonomy for teams managing separate clusters.

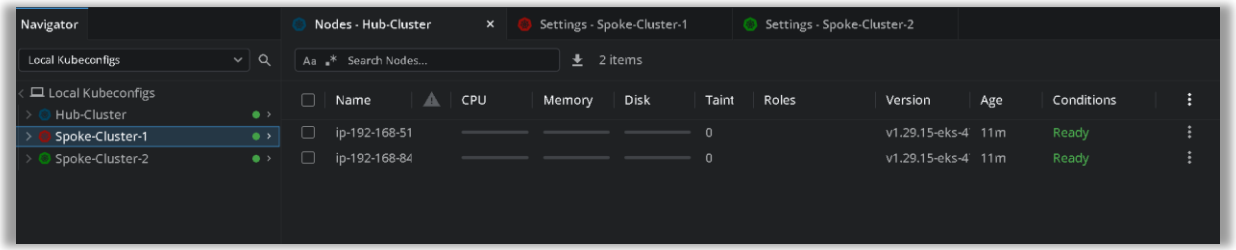
Advantages:

- Fault isolation prevents issues in one cluster's Argo CD from impacting others.
- Clusters can operate independently, even if connectivity is limited.
- Fits organizational models where teams control their own environments.

Challenges:

- Increased operational overhead to maintain multiple Argo CD instances.
- Duplication of monitoring and management tasks.
- Harder to enforce consistent global policies.

Practical Steps for Multicluster Deployment with Argo CD




```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                                       AUTHINFO
*         Admin@hub-cluster.ap-south-1.eksctl.io        hub-cluster.ap-south-1.eksctl.io              Admin@hub-cluster.ap-south-1.eksctl.io
          Admin@spoke-cluster-1.ap-south-1.eksctl.io    spoke-cluster-1.ap-south-1.eksctl.io          Admin@spoke-cluster-1.ap-south-1.eksctl.io
          Admin@spoke-cluster-2.ap-south-1.eksctl.io    spoke-cluster-2.ap-south-1.eksctl.io          Admin@spoke-cluster-2.ap-south-1.eksctl.io
```

Cluster Provisioning

Tools such as eksctl enable rapid provisioning of Amazon EKS clusters with minimal configuration. However, practical challenges like AWS CloudFormation stack conflicts, permission issues, and resource quotas must be carefully managed. Proactive resource cleanup and manual console inspections help troubleshoot provisioning failures.

Argo CD Installation and Configuration

```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl get ns
NAME              STATUS   AGE
argocd            Active   55s
default           Active   23m
kube-node-lease   Active   23m
kube-public       Active   23m
kube-system       Active   23m

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get deployment -n argocd
NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
argocd-applicationset-controller   1/1     1            1           54s
argocd-dex-server                  1/1     1            1           54s
argocd-notifications-controller    1/1     1            1           54s
argocd-redis                       1/1     1            1           54s
argocd-repo-server                 1/1     1            1           54s
argocd-server                      1/1     1            1           54s

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get svc -n argocd
NAME                                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller           ClusterIP   10.100.36.210    <none>        7000/TCP,8080/TCP            76s
argocd-dex-server                          ClusterIP   10.100.76.2      <none>        5556/TCP,5557/TCP,5558/TCP   76s
argocd-metrics                             ClusterIP   10.100.193.143   <none>        8082/TCP                     76s
argocd-notifications-controller-metrics    ClusterIP   10.100.193.60    <none>        9001/TCP                     76s
argocd-redis                               ClusterIP   10.100.6.89      <none>        6379/TCP                     76s
argocd-repo-server                         ClusterIP   10.100.119.24    <none>        8081/TCP,8084/TCP            76s
argocd-server                              ClusterIP   10.100.255.232   <none>        80/TCP,443/TCP               76s
argocd-server-metrics                      ClusterIP   10.100.115.21    <none>        8083/TCP                     76s

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get pods -n argocd
NAME                                               READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                    1/1     Running   0          81s
argocd-applicationset-controller-88f476b85-fbx9k   1/1     Running   0          81s
argocd-dex-server-654fdb64b4-qkpkm                 1/1     Running   0          81s
argocd-notifications-controller-6689f6dcd6-5zh6f   1/1     Running   0          81s
argocd-redis-69967d6fc7-xt4hg                      1/1     Running   0          81s
argocd-repo-server-56bd65c7c-jvk2r                 1/1     Running   0          81s
argocd-server-6f6b89fb4f-67hxd                     1/1     Running   0          81s
```

```
sahil@SAHI-SNEH MINGW64 ~
$ argocd version
argocd: v3.0.0+e98f483
  BuildDate: 2025-05-06T11:50:03Z
  GitCommit: e98f483bfd5781df2592fef1aeed1148f150d9c9
  GitTreeState: clean
  GoVersion: go1.24.1
  Compiler: gc
  Platform: windows/amd64
{"level":"fatal","msg":"Argo CD server address unspecified","time":"2025-05-11T19:40:45+05:30"}
```

```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl get cm -n argocd
NAME                        DATA   AGE
argocd-cm                   9      6m32s
argocd-cmd-params-cm        0      6m32s
argocd-gpg-keys-cm          0      6m32s
argocd-notifications-cm     0      6m32s
argocd-rbac-cm              0      6m32s
argocd-ssh-known-hosts-cm   1      6m32s
argocd-tls-certs-cm         0      6m31s
kube-root-ca.crt            1      6m42s

sahil@SAHI-SNEH MINGW64 ~
$ kubectl edit configmap argocd-cmd-params-cm -n argocd
configmap/argocd-cmd-params-cm edited

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP
argocd-applicationset-controller          ClusterIP   10.100.36.210    <none>
argocd-dex-server                         ClusterIP   10.100.76.2      <none>
argocd-metrics                            ClusterIP   10.100.193.143   <none>
argocd-notifications-controller-metrics   ClusterIP   10.100.193.60    <none>
argocd-redis                              ClusterIP   10.100.6.89      <none>
argocd-repo-server                        ClusterIP   10.100.119.24    <none>
argocd-server                             ClusterIP   10.100.255.232   <none>
argocd-server-metrics                     ClusterIP   10.100.115.21    <none>

sahil@SAHI-SNEH MINGW64 ~
$ kubectl edit svc argocd-server -n argocd
```

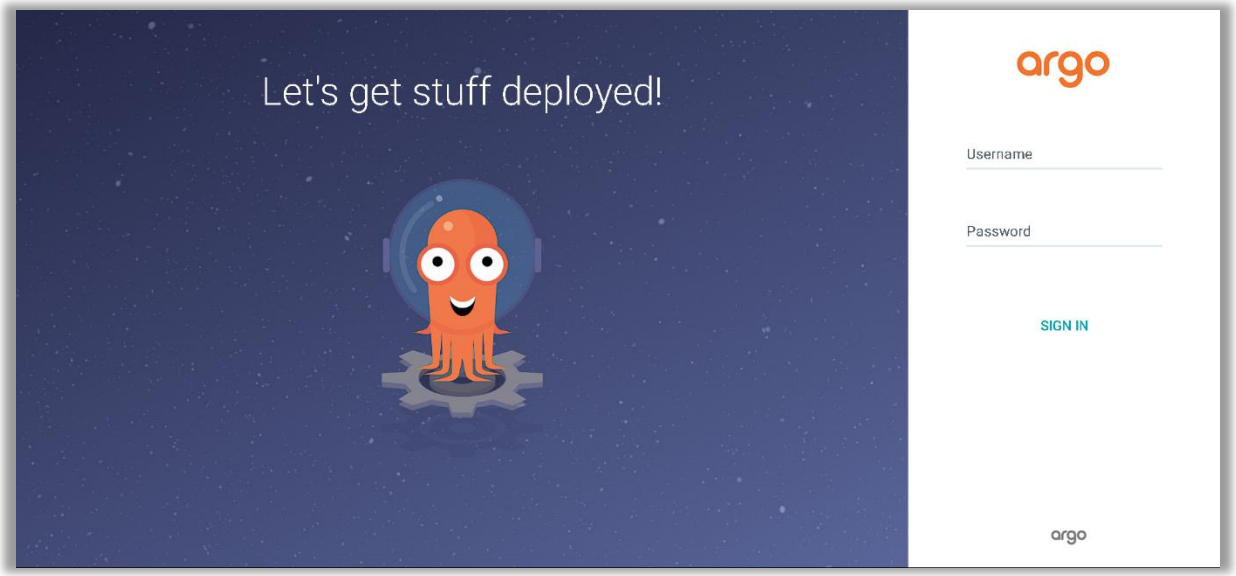
kubec

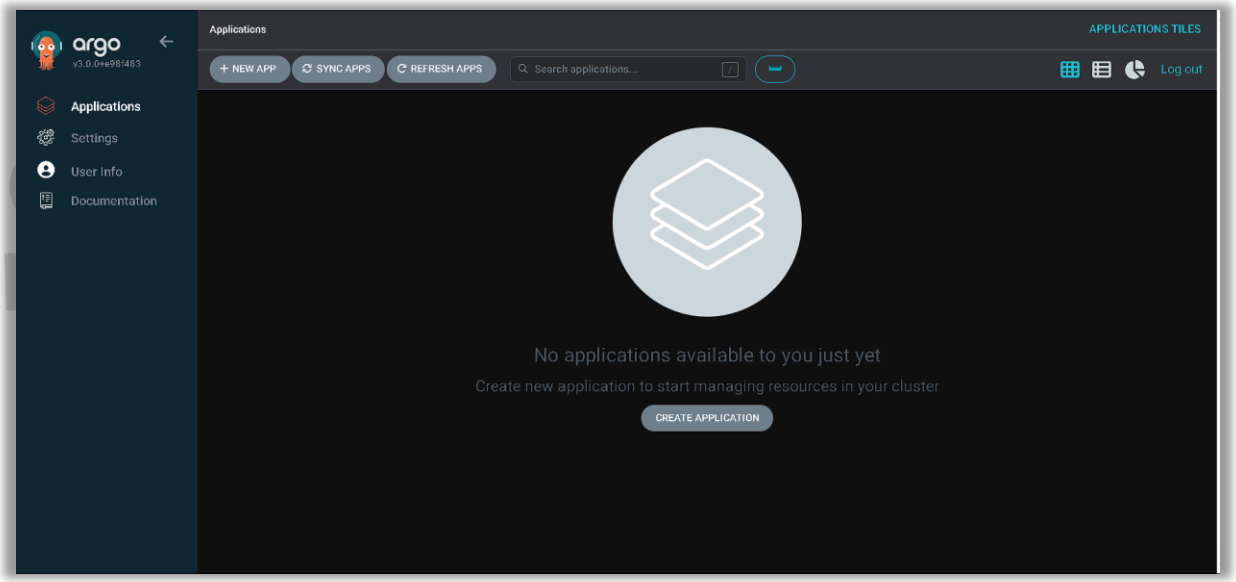
File Edit View

```
spec:
  clusterIP: 10.100.255.232
  clusterIPs:
  - 10.100.255.232
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  - name: https
    port: 443
    protocol: TCP
    targetPort: 8080
  selector:
    app.kubernetes.io/name: argocd-server
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

Ln 40, Col 17 | 14 of 1,466 chara | 100% | Windows ((| UTF-8

```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl edit svc argocd-server -n argocd
service/argocd-server edited

sahil@SAHI-SNEH MINGW64 ~
$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.100.36.210    <none>        7000/TCP,8080/TCP            12m
argocd-dex-server                         ClusterIP   10.100.76.2      <none>        5556/TCP,5557/TCP,5558/TCP   12m
argocd-metrics                            ClusterIP   10.100.193.143   <none>        8082/TCP                     12m
argocd-notifications-controller-metrics   ClusterIP   10.100.193.60    <none>        9001/TCP                     12m
argocd-redis                              ClusterIP   10.100.6.89      <none>        6379/TCP                     12m
argocd-repo-server                        ClusterIP   10.100.119.24    <none>        8081/TCP,8084/TCP            12m
argocd-server                             NodePort    10.100.255.232   <none>        80:30521/TCP,443:31884/TCP   12m
argocd-server-metrics                     ClusterIP   10.100.115.21    <none>        8083/TCP                     12m
```

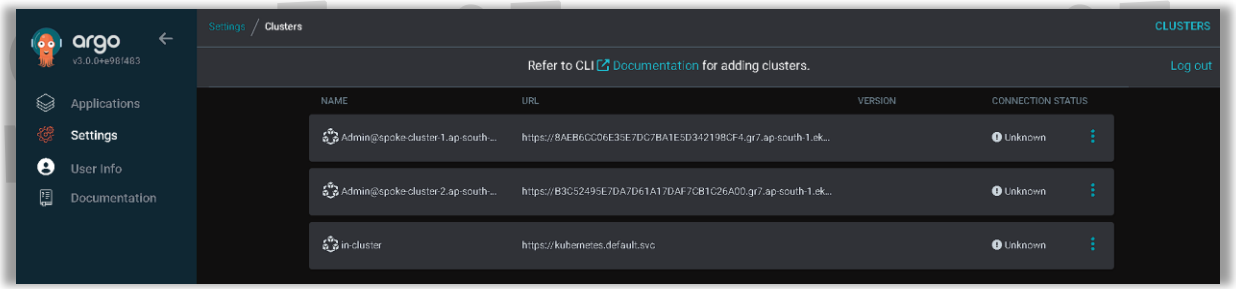


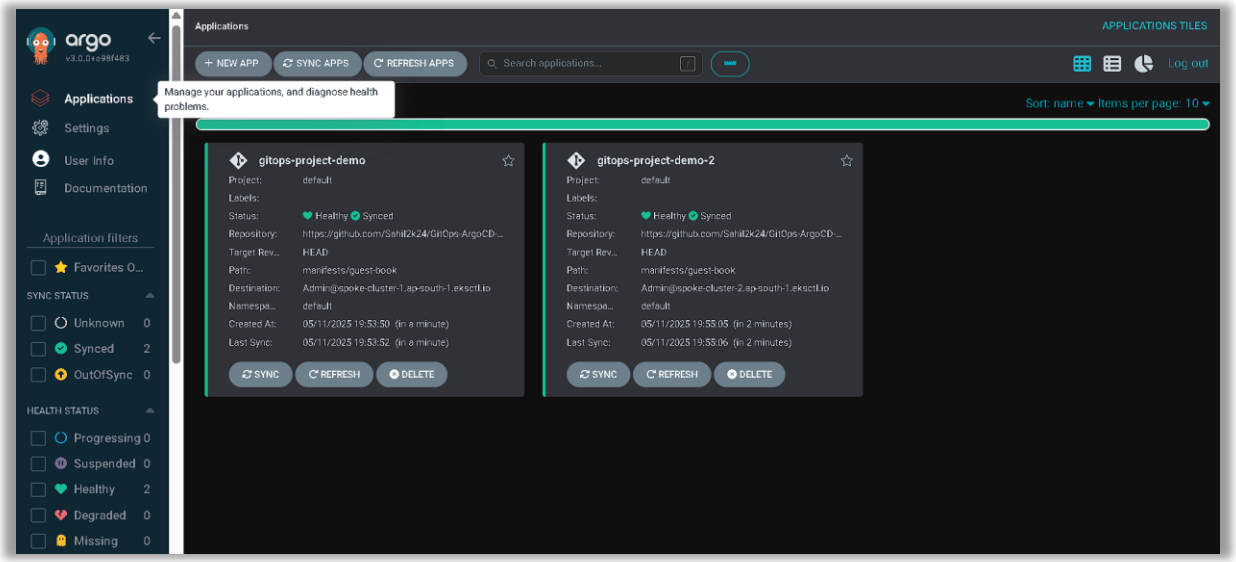


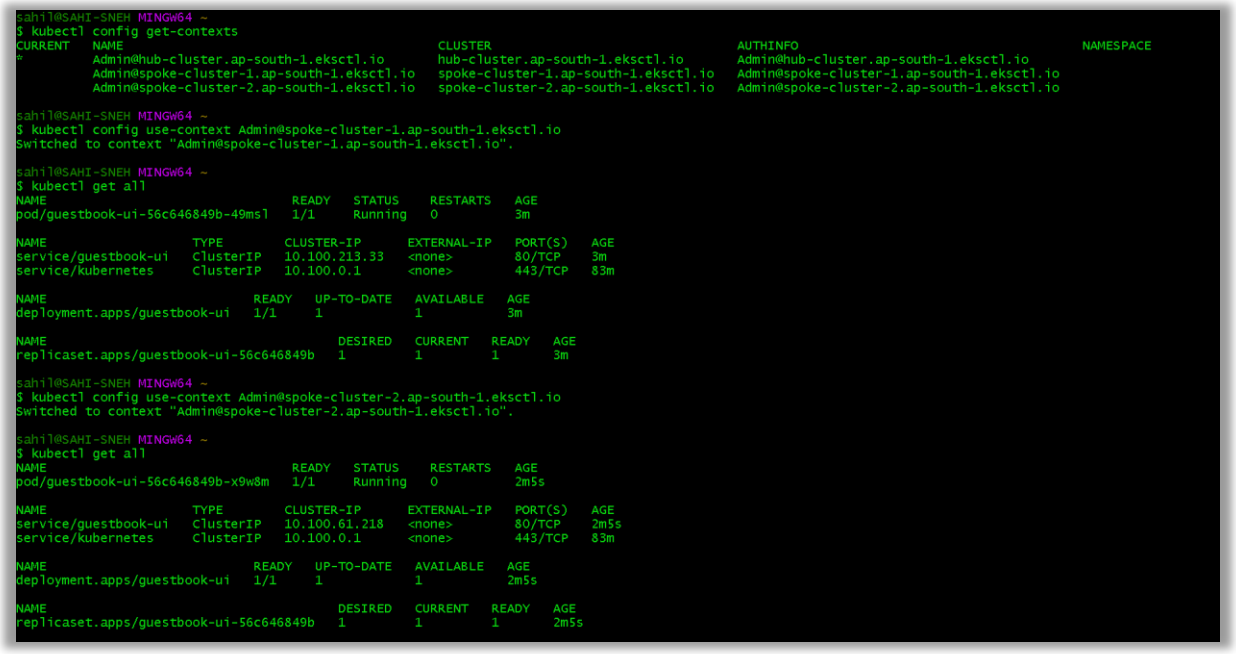
- Deploy Argo CD components into a dedicated namespace (commonly argocd) on the Hub cluster.
- For demonstration or testing, insecure HTTP mode may be enabled, but production setups must use HTTPS and enforce strong authentication.
- Expose the Argo CD UI using NodePort or Ingress, balancing accessibility with network security best practices.
- Use the Argo CD CLI (argocd) to manage cluster registrations, as the web UI currently lacks the ability to add external clusters.

Cluster Registration and Application Deployment

```
sahil@SAHI-SNEH MINGW64 ~
$ kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                                       AUTHINFO                                      NAMESP
ACE
*         Admin@hub-cluster.ap-south-1.eksctl.io        hub-cluster.ap-south-1.eksctl.io              Admin@hub-cluster.ap-south-1.eksctl.io
          Admin@spoke-cluster-1.ap-south-1.eksctl.io    spoke-cluster-1.ap-south-1.eksctl.io          Admin@spoke-cluster-1.ap-south-1.eksctl.io
          Admin@spoke-cluster-2.ap-south-1.eksctl.io    spoke-cluster-2.ap-south-1.eksctl.io          Admin@spoke-cluster-2.ap-south-1.eksctl.io

sahil@SAHI-SNEH MINGW64 ~
$ argocd login 52.66.210.243:30521
WARNING: server certificate had error: tls: failed to verify certificate: x509: certificate signed by unknown authority. Proceed insecurely (y/n)
? y
Username: admin
Password:
'admin:login' logged in successfully
Context '52.66.210.243:30521' updated

sahil@SAHI-SNEH MINGW64 ~
$ argocd cluster add Admin@spoke-cluster-1.ap-south-1.eksctl.io --server 52.66.210.243:30521
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `Admin@spoke-cluster-1.ap-south-1.eksctl.io` wi
th full cluster level privileges. Do you want to continue [y/N]? y
{"level":"info","msg":"ServiceAccount \"argocd-manager\" created in namespace \"kube-system\"","time":"2025-05-11T19:44:52+05:30"}
{"level":"info","msg":"ClusterRole \"argocd-manager-role\" created","time":"2025-05-11T19:44:52+05:30"}
{"level":"info","msg":"ClusterRoleBinding \"argocd-manager-role-binding\" created","time":"2025-05-11T19:44:52+05:30"}
{"level":"info","msg":"Created bearer token secret for ServiceAccount \"argocd-manager\"","time":"2025-05-11T19:44:52+05:30"}
Cluster 'https://8AEB6CC06E35E7DC7BA1E5D342198CF4.gr7.ap-south-1.eks.amazonaws.com' added

sahil@SAHI-SNEH MINGW64 ~
$ argocd cluster add Admin@spoke-cluster-2.ap-south-1.eksctl.io --server 52.66.210.243:30521
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `Admin@spoke-cluster-2.ap-south-1.eksctl.io` wi
th full cluster level privileges. Do you want to continue [y/N]? y
{"level":"info","msg":"ServiceAccount \"argocd-manager\" created in namespace \"kube-system\"","time":"2025-05-11T19:45:27+05:30"}
{"level":"info","msg":"ClusterRole \"argocd-manager-role\" created","time":"2025-05-11T19:45:27+05:30"}
{"level":"info","msg":"ClusterRoleBinding \"argocd-manager-role-binding\" created","time":"2025-05-11T19:45:27+05:30"}
{"level":"info","msg":"Created bearer token secret for ServiceAccount \"argocd-manager\"","time":"2025-05-11T19:45:28+05:30"}
Cluster 'https://B3C52495E7DA7D61A17DAF7CB1C26A00.gr7.ap-south-1.eks.amazonaws.com' added
```







- Add each target cluster (Spoke) to Argo CD running on the Hub using CLI commands.
- Define Argo CD Application resources in Git that specify the Git repository, target cluster context, namespace, and sync policies.
- Deploy sample or real applications declaratively, ensuring all manifests reside in version-controlled Git repositories.
- Enable automated sync policies for self-healing behavior, or manual sync for controlled deployments.

Monitoring and Reconciliation

- Argo CD continuously monitors the live cluster state.
- Any manual change or drift triggers an "Out of Sync" alert.
- Operators can manually sync to restore the Git-defined state or configure automatic sync to auto-heal.
- Real-time visibility in Argo CD UI aids troubleshooting and operational awareness.

Scaling Deployment with ApplicationSets

For large-scale environments, managing individual Argo CD Applications per cluster is inefficient and error-prone. ApplicationSets offer a scalable solution by:

- Dynamically generating Argo CD Applications based on cluster or environment metadata.
- Supporting templates and parameterization to customize deployments per cluster.
- Enabling seamless onboarding of new clusters by automatically applying configurations.
- Greatly reducing manual effort and configuration duplication.

This capability is essential for organizations managing hundreds or thousands of Kubernetes clusters.

Security Best Practices

- Always secure Argo CD with HTTPS and restrict UI/API access using firewalls, VPNs, or security groups.
- Integrate enterprise authentication mechanisms like Single Sign-On (SSO) and OAuth.
- Employ Role-Based Access Control (RBAC) within Argo CD to enforce least privilege principles.
- Maintain strict Git repository security; ensure only authorized users can push changes.
- Regularly audit Git commit history and Argo CD logs to detect unauthorized activities.
- Avoid making manual cluster changes outside GitOps workflows to maintain auditability.

Conclusion

GitOps represents a transformative approach to managing Kubernetes deployments, especially in complex multicluster environments. By leveraging Git as the single source of truth and Argo CD as the continuous delivery engine, organizations can achieve:

- Reliable, repeatable, and consistent deployments
- Enhanced operational efficiency and reduced manual toil
- Strong compliance and auditability for regulatory requirements
- Self-healing infrastructure that minimizes downtime
- Scalable management of multi-environment and multi-cluster landscapes

For DevOps engineers, platform teams, and SREs, mastering GitOps with Argo CD unlocks the potential to scale Kubernetes deployments with confidence and control. This shift not only streamlines delivery workflows but also significantly improves the stability and security posture of cloud-native applications.
