Click here for DevSecOps & Cloud DevOps Course

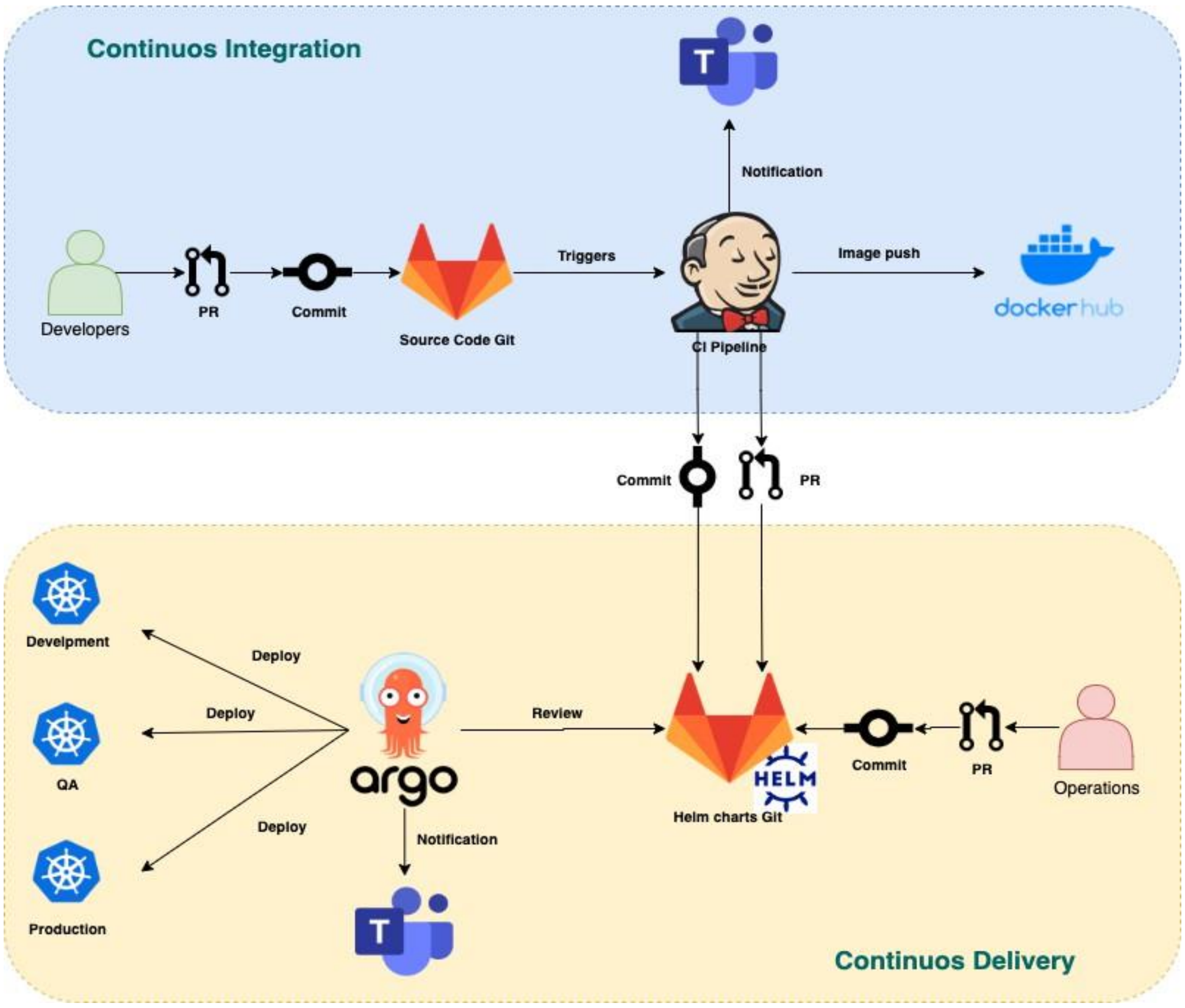
DevOps Shack

GitOps - The Argo CD Handbook

with Hands-on

Introduction

ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It automates the deployment of the desired application states defined in Git repositories, ensuring applications are always in sync with the declared configuration. ArgoCD is designed to provide application deployment and lifecycle management in a way that is both secure and scalable.



Problem in Normal Continuous Delivery

Traditional continuous delivery methods often face several challenges:

1. **Complexity in Configuration Management**: Managing configurations across multiple environments can become cumbersome and error-prone.
2. **Lack of Consistency**: Ensuring consistency between environments (e.g., development, staging, production) is difficult.
3. **Manual Interventions**: Frequent manual interventions are required to handle deployments, leading to inefficiencies and potential errors.
4. **Poor Visibility**: Limited visibility into the deployment state and application status makes it difficult to troubleshoot issues quickly.

GitOps

GitOps is a paradigm for continuous delivery and operational tasks using **Git as the single source of truth**. It brings the following benefits:

1. **Declarative Descriptions**: The desired state of the system is described declaratively.

2. **Version Control**: Git's version control capabilities ensure every change is tracked and auditable.

3. **Automated Synchronization**: Tools like ArgoCD automatically synchronize the desired state in Git with the actual state in the cluster.
4. **Enhanced Collaboration**: GitOps encourages collaboration and review through Git's pull request workflow.

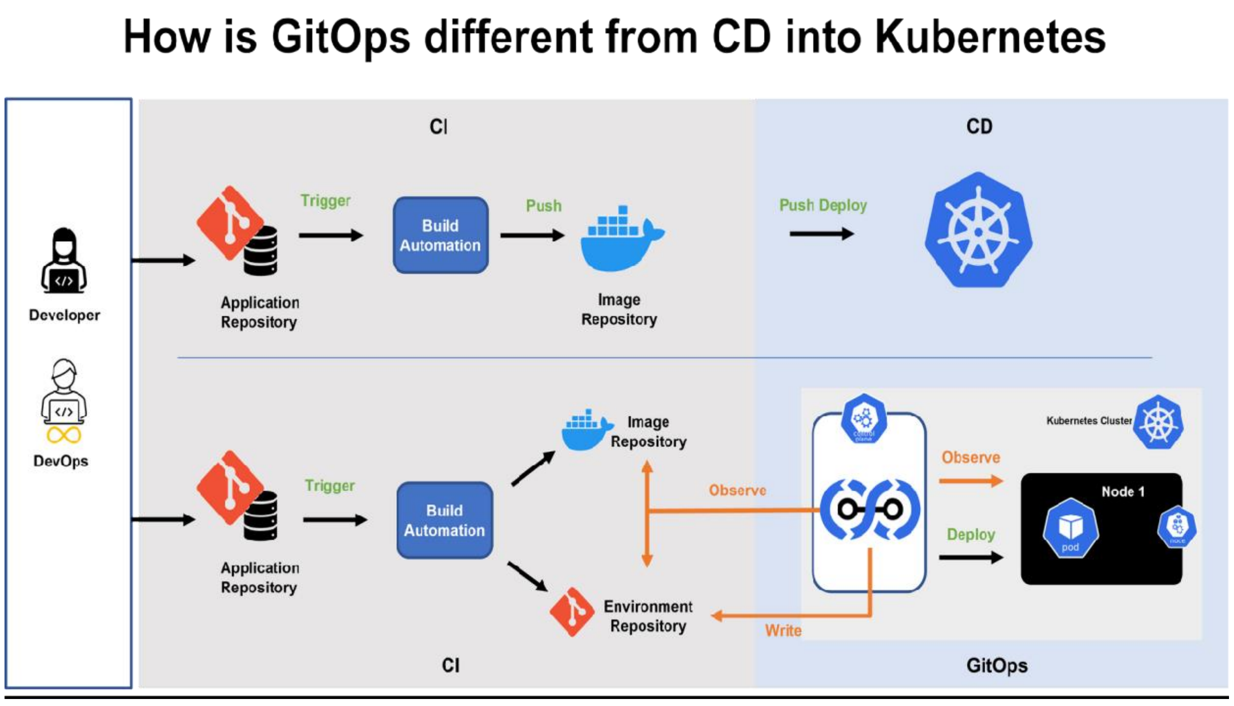
Difference between GitOps and Traditional CD

Traditional CD and GitOps differ on the core principles of push and pull-based deployments.

Most CI/CD processes work on a **push-based** mechanism, which means things move to their respective destination at the trigger of an event. For example, when a developer has finished writing his code, he must execute a set of commands to move his code into the server for deployment. In a Kubernetes environment, the developer has to configure the clusters using tools like **Kubectl** and **Helm** in the pipeline to apply changes.

Argo is a CD tool that uses a **pull-based** mechanism. A pull-based CD mechanism means the destination triggers an event to pull the data from the source(git) to deploy at the destination. Argo CD, which resided inside the cluster for reasons explained later on the blog, pulls the most recent verified version of code into the cluster for deployment. There are a lot of benefits to this model, like improved security and ease of use.

This pull-based mechanism is called GitOps, where the source code management systems like Git are treated as the only source of truth for application and configuration data.



Structure of Argo CD

ArgoCD consists of several key components:

1. **API Server**: Provides a REST API for managing applications and retrieving the current state.
2. **Controller**: Monitors the Git repository and the cluster state, making sure they are in sync.
3. **Repository Server**: Clones the Git repository and provides the manifests to the controller.
4. **Web UI**: A web-based interface to manage and visualize application states and deployments.
5. **CLI**: Command-line interface for managing ArgoCD applications and configurations.

Working

Argo CD works in a reversed flow mechanism as compared to push-style deployments. The new mechanism enables ArgoCD to run from inside a Kubernetes cluster. Kubernetes faces challenges with the traditional CD mechanism because CI/CD tools, like Jenkins, sit outside the cluster, whereas Argo CD sits inside the cluster. While inside the cluster, Argo CD pulls changes from git and applies them to the residing cluster. Instead of pushing changes like older generation tools by being inside the cluster, ArgoCD prevents sensitive information from being exposed to other tools outside the Kubernetes cluster and environment.

Argo CD can be set up in two simple steps.

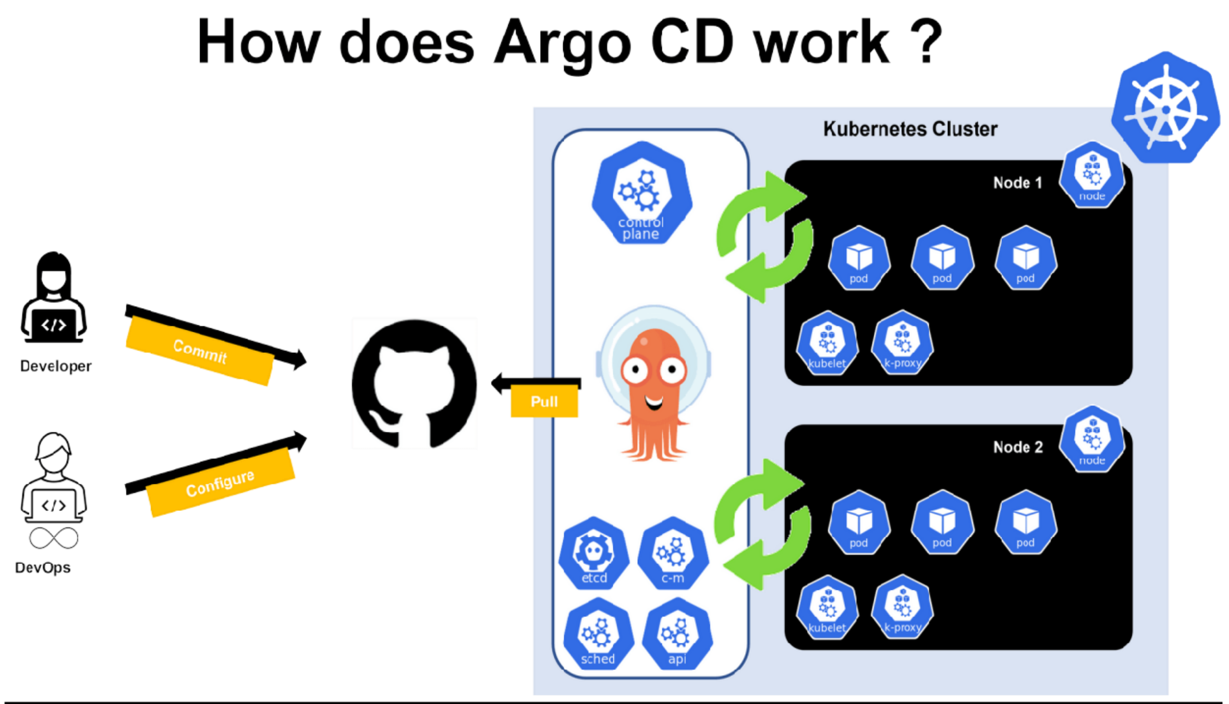
1. Deploy the ArgoCD agent to the cluster.
2. Configure Argo CD to track a Git repository for changes.

When Argo CD monitors change, it automatically applies them to the Kubernetes cluster. When developers commit the new code updates to the Git repository, automated CI pipelines will auto-start the build process and build the container image. Then as per the configurations, the CI pipeline will push and update the Kubernetes manifest files. The pipelines will update the new image version name and details on the `deployment.yaml` file. Argo CD can track this new update, pull the image, and deploy it onto the target cluster.

When the Kubernetes cluster is ready, Argo CD sends a report about the status of the application and that the synchronization is complete and correct. ArgoCD also works in the other direction, monitoring changes in the Kubernetes cluster and discarding them if they don't match the current configuration in Git.

ArgoCD operates in the following manner:

1. **Repository Monitoring**: ArgoCD continuously monitors specified Git repositories for changes.
2. **Application Synchronization**: When changes are detected, ArgoCD synchronizes the cluster state with the desired state defined in the repository.
3. **Health Assessment**: It assesses the health of applications to ensure they are deployed correctly.
4. **Rollback**: If issues are detected, ArgoCD can roll back to the last known good state.



Constraints

While ArgoCD provides significant benefits, there are some constraints to consider:

1. **Initial Setup Complexity**: Setting up ArgoCD and configuring GitOps workflows can be complex.
2. **Learning Curve**: Teams need to understand both Kubernetes and GitOps principles.
3. **Scalability**: Managing large numbers of applications or very complex environments can be challenging.
4. **Access Control**: Ensuring proper access control and security policies can be difficult without proper configuration.

Setting up ArgoCD

Create EKS Cluster From UI

1. **Create Role for EKS Cluster**:
    - Go to AWS Management Console.
    - Navigate to IAM (Identity and Access Management).
    - Click on "Roles" and then click on "Create role".
    - Choose "AWS Service" as the trusted entity.
    - Choose "EKS-cluster" as the use case.
    - Click "Next" and provide a name for the role.
2. **Create Role for EC2 Instances**:
    - Go to AWS Management Console.
    - Navigate to IAM (Identity and Access Management).
    - Click on "Roles" and then click on "Create role".
    - Choose "AWS Service" as the trusted entity.
    - Choose "EC2" as the use case.
    - Click "Next".
    - Add policies: [AmazonEC2ContainerRegistryReadOnly, AmazonEKS_CNI_Policy, AmazonEBSCSIDriverPolicy, AmazonEKSWorkerNodePolicy].
    - Provide a name for the role, e.g., "myNodeGroupPolicy".
3. **Create EKS Cluster**:
    - Go to AWS Management Console.
    - Navigate to Amazon EKS service.
    - Click on "Create cluster".
    - Enter the desired name, select version, and specify the role created in step 1.
    - Configure Security Group, Cluster Endpoint, etc.

   - Click "Next" and proceed to create the cluster.

4. **Create Compute Resources**:
   - Go to AWS Management Console.
   - Navigate to Amazon EKS service.
   - Click on "Compute" or "Node groups".
   - Provide a name for the compute resource.
   - Select the role created in step 2.
   - Select Node Type & Size.
   - Click "Next" and proceed to create the compute resource.

5. **Configure Cloud Shell**:
   - Open AWS Cloud Shell or AWS CLI.
   - Execute the command:
   - aws eks update-kubeconfig --name shack-eks --region ap-south-1
   - Replace "shack-eks" with the name of your EKS cluster and "ap-south-1" with the appropriate region if different.

These steps should help you in setting up your EKS cluster along with necessary roles and compute resources.

Install ArgoCD

Here are the steps to install ArgoCD and retrieve the admin password:

1. **Create Namespace for ArgoCD**:

```
kubectl create namespace argocd
```

2. **Apply ArgoCD Manifests**:

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
```

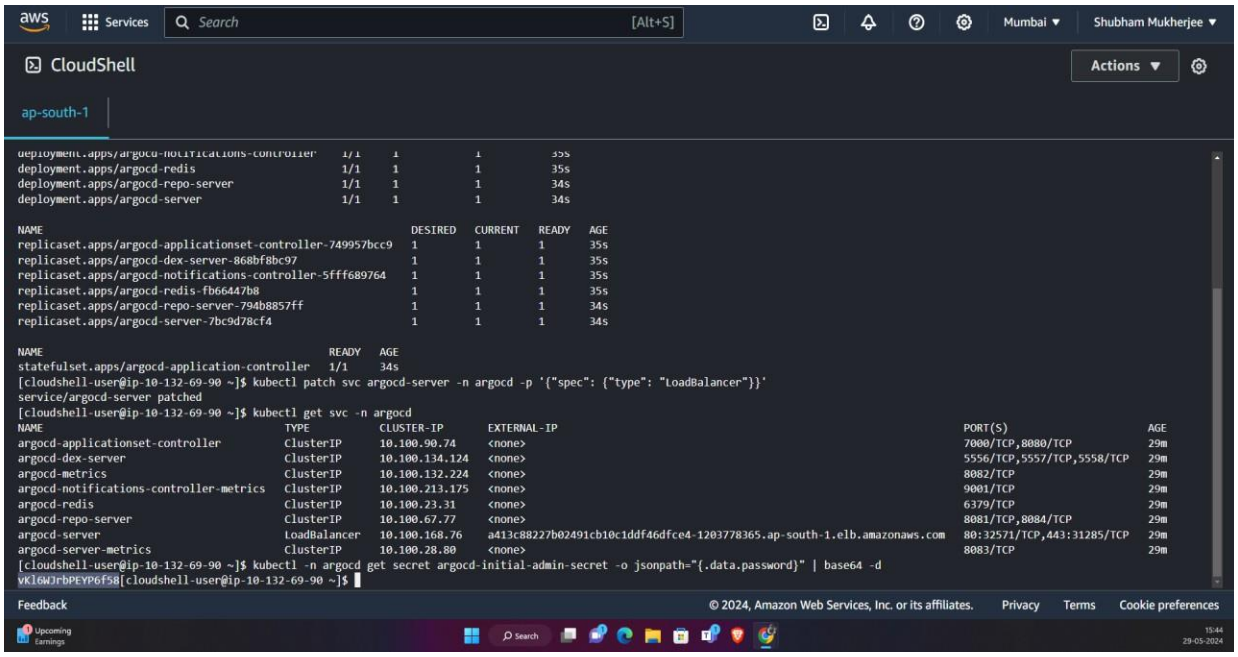
3. **Patch Service Type to LoadBalancer**:

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

4. **Retrieve Admin Password**:

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

These commands will install ArgoCD into the specified namespace, set up the service as a LoadBalancer, and retrieve the admin password for you to access the ArgoCD UI.



```
deployment.apps/argocd-notifications-controller   1/1    1           1           35s
deployment.apps/argocd-redis                      1/1    1           1           35s
deployment.apps/argocd-repo-server                1/1    1           1           34s
deployment.apps/argocd-server                     1/1    1           1           34s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/argocd-applicationset-controller-749957bcc9   1         1         1       35s
replicaset.apps/argocd-dex-server-868bf8bc97        1         1         1       35s
replicaset.apps/argocd-notifications-controller-5fff689764   1         1         1       35s
replicaset.apps/argocd-redis-fb66447b8              1         1         1       35s
replicaset.apps/argocd-repo-server-794b8857ff       1         1         1       34s
replicaset.apps/argocd-server-7bc9d78cf4            1         1         1       34s

NAME                                             READY   AGE
statefulset.apps/argocd-application-controller   1/1     34s
[cloudshell-user@ip-10-132-69-90 ~]$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
[cloudshell-user@ip-10-132-69-90 ~]$ kubectl get svc -n argocd
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP      10.100.90.74     <none>                                                                     7000/TCP,8080/TCP            29m
argocd-dex-server                         ClusterIP      10.100.134.124   <none>                                                                     5556/TCP,5557/TCP,5558/TCP   29m
argocd-metrics                            ClusterIP      10.100.132.224   <none>                                                                     8082/TCP                     29m
argocd-notifications-controller-metrics   ClusterIP      10.100.213.175   <none>                                                                     9001/TCP                     29m
argocd-redis                              ClusterIP      10.100.23.31     <none>                                                                     6379/TCP                     29m
argocd-repo-server                        ClusterIP      10.100.67.77     <none>                                                                     8081/TCP,8084/TCP            29m
argocd-server                             LoadBalancer   10.100.168.76    a413c88227b02491cb10c1ddf46dfce4-1203778365.ap-south-1.elb.amazonaws.com   80:32571/TCP,443:31285/TCP   29m
argocd-server-metrics                     ClusterIP      10.100.28.80     <none>                                                                     8083/TCP                     29m
[cloudshell-user@ip-10-132-69-90 ~]$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d
vKl6WJrbPEYP6f58[cloudshell-user@ip-10-132-69-90 ~]$
```

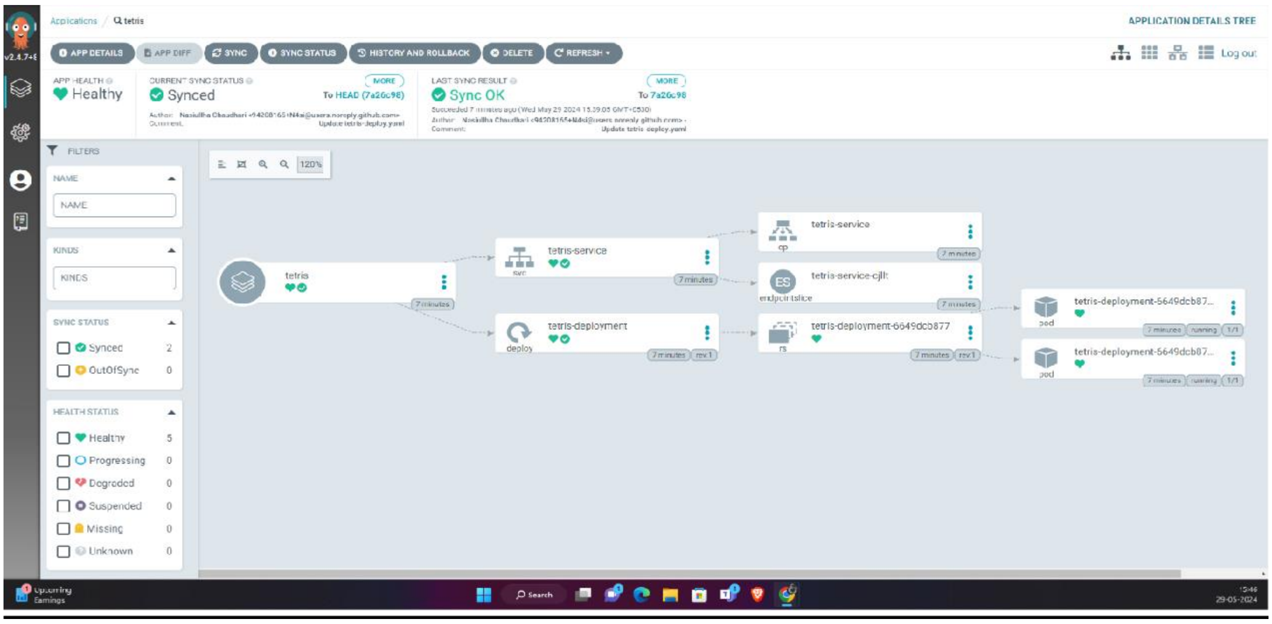
Configuring ArgoCD and Creating an Application

1. **Select Repository**:
    - Navigate to ArgoCD UI.
    - Go to **Settings** > **Repositories**.
    - Click on **Connect Repo** using **HTTPS**.
    - For **Type**, select **Git**.
    - Enter the **Repository URL**- **https://github.com/N4si/tetris-game.git**
    - Click **Connect**.
2. **Create Application**:
    - In the ArgoCD UI, go to **Applications**.
    - Click on **Create Application**.
    - Fill in the details:
        - **Application Name**: Provide a name for your application.
        - **Project Name**: Select **default**.
        - **Sync Policy**: Select **Automatic** and check **Prune resources** and **Self heal**.
        - **Auto Create Namespace**: Ensure this is checked.
    - **Source**:
        - For **Repo URL**, enter your repository URL.
        - Provide the **Manifest file name** (e.g., **path/to/manifest**).
    - **Destination**:
        - **Cluster URL**: Enter **https://kubernetes.default.svc**.
        - **Namespace**: Enter the namespace where you want to deploy (e.g., **default**).
3. **Save and Sync**:
    - Click **Create** to create the application.

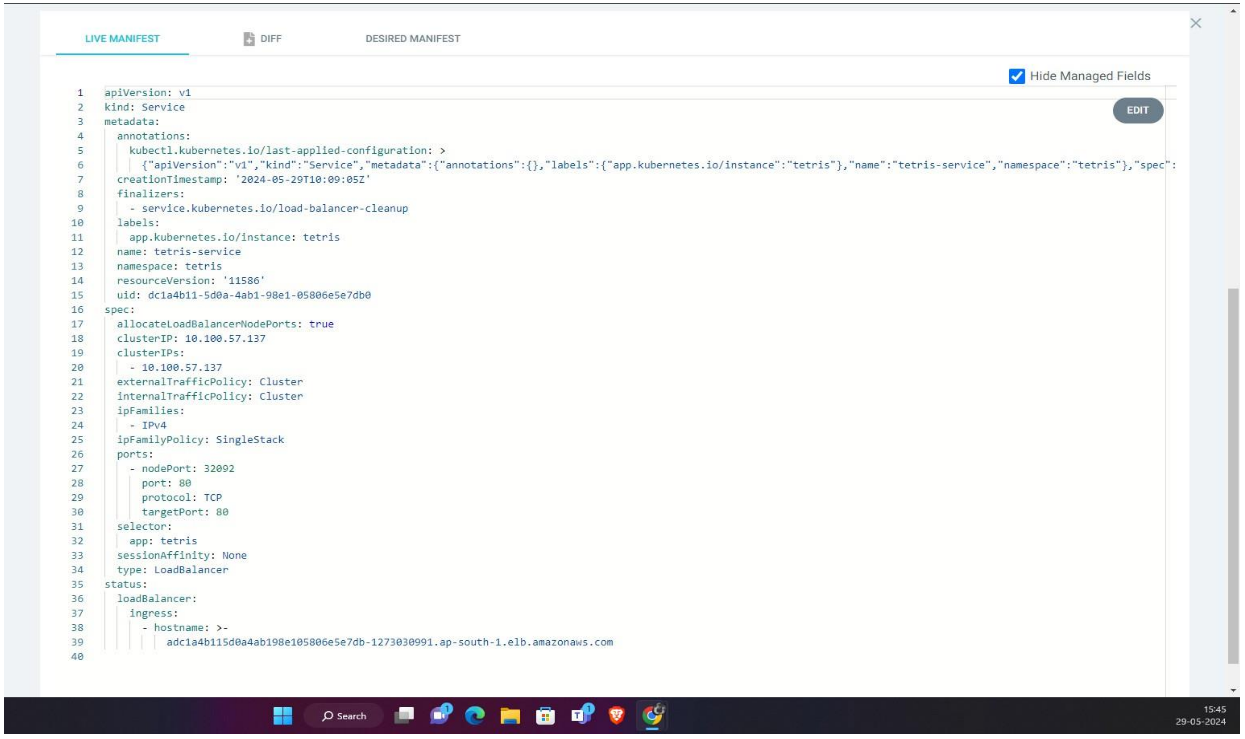
- ArgoCD will automatically synchronize the application state to match the repository state.

These steps should help you quickly configure ArgoCD, connect a repository, and create an application with automatic sync and self-healing capabilities.

Results



Live Manifests



```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: >
      {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app.kubernetes.io/instance":"tetris"},"name":"tetris-service","namespace":"tetris"},"spec":
  creationTimestamp: '2024-05-29T10:09:05Z'
  finalizers:
    - service.kubernetes.io/load-balancer-cleanup
  labels:
    app.kubernetes.io/instance: tetris
  name: tetris-service
  namespace: tetris
  resourceVersion: '11586'
  uid: dc1a4b11-5d0a-4ab1-98e1-05806e5e7db0
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.100.57.137
  clusterIPs:
    - 10.100.57.137
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - nodePort: 32092
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: tetris
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
      - hostname: >-
          adc1a4b115d0a4ab198e105806e5e7db-1273030991.ap-south-1.elb.amazonaws.com
```