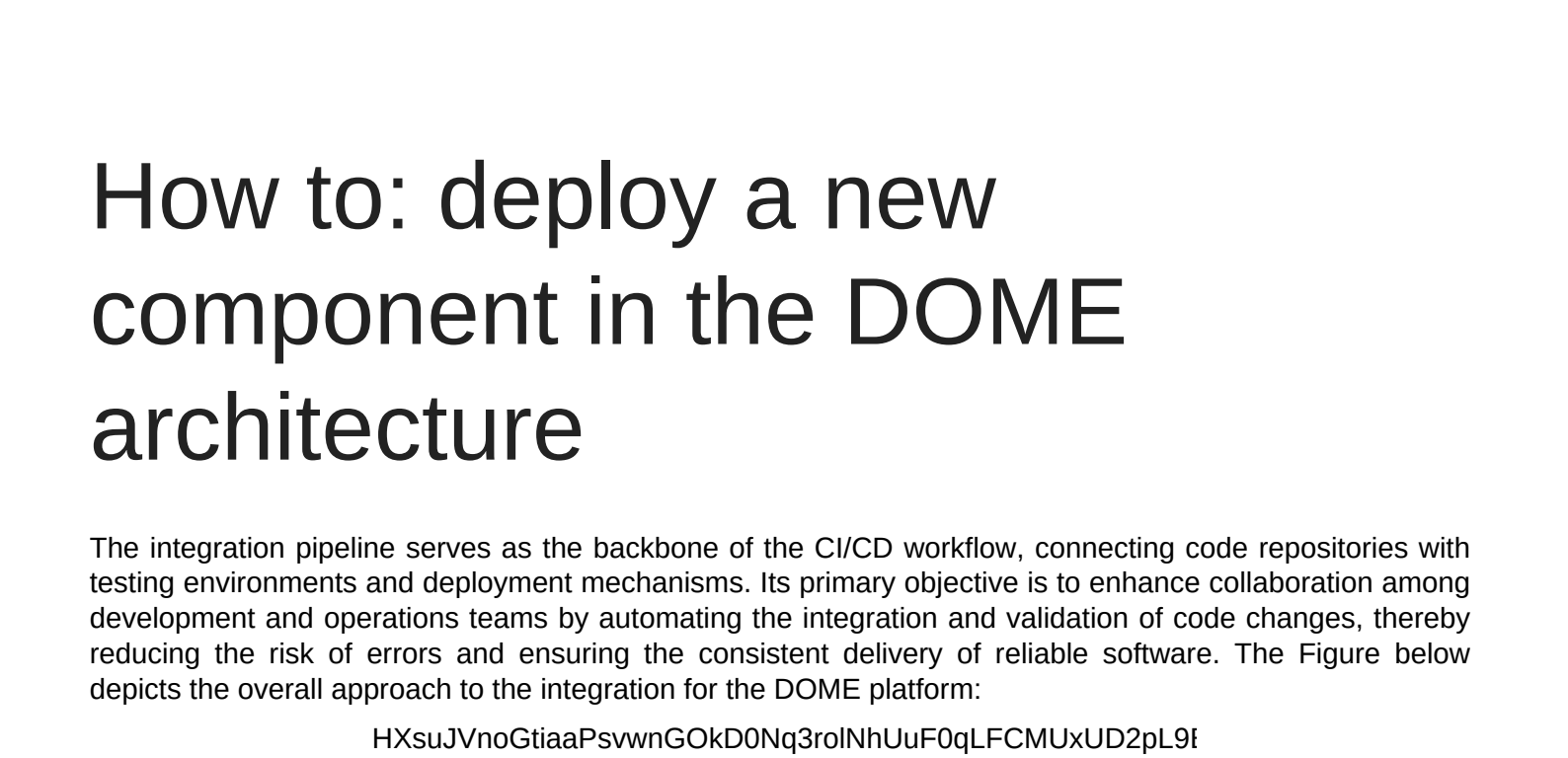
How to: deploy a new component in the DOME architecture

How to: deploy a new component in the DOME architecture

The integration pipeline serves as the backbone of the CI/CD workflow, connecting code repositories with testing environments and deployment mechanisms. Its primary objective is to enhance collaboration among development and operations teams by automating the integration and validation of code changes, thereby reducing the risk of errors and ensuring the consistent delivery of reliable software. The Figure below depicts the overall approach to the integration for the DOME platform:



In the integration approach designed for DOME, the team intending to integrate its component needs to adhere to a structured set of steps:

1. Fork the repository and create a new branch
2. Add component manifest files
3. Add ArgoCD application
4. Create a Pull Request and wait for merge

The following paragraphs detail the steps to integrate a new application into DOME.

Configuration of kubectl to access the remote cluster

Before starting the integration process, you need to configure **kubectl** in order to access the remote cluster locally; to do this, follow the steps below:

1. Request the cluster administrator to create a service account, providing the name of your organisation and the namespace within which your application will be deployed. You will be provided with a configuration file to access the cluster

2. Modify the **KUBECONFIG** environment variable by adding the path to the configuration file

3. Verify that the new context has been added by executing the following command:

    kubectl config get-contexts

4. Switch the context by executing the following command:

    kubectl config use-context **<name of the context>**

At this point, you should be able to run kubectl commands on the remote cluster.

Notes

- Ensure that the user name of the new context has not already been used in other contexts; if so, modify the user name in the configuration file.
- The **service account** provided to you will have **limited permissions**; you will be able to view all resources within your namespace but will only have write access to secrets.

Fork the repository and create a new branch

Team members begin by forking the main GitOps repository and creating a new branch for their specific integration work. This facilitates isolated development and changes.

After forking and subsequently cloning the project, navigate to the project directory to proceed with the rest of the guide.

Repository structure

The GitOps repository has the following structure:

```
applications/
  ????????? app_1.yaml
  ????????? app_2.yaml
  . . .
applications_dev/
  ????????? app_1.yaml
  ????????? app_2.yaml
ionos/
  ????????? app_1/
      ????????? Chart.yaml
      ????????? values.yaml
      . . .
```

```
ionos_dev/
  ????????? app_1/
      ????????? Chart.yaml
      ????????? values.yaml
      . . .
```

For each environment, two directories are defined:

- applications_<env>: it will host the environment-specific Argocd applications (see Add ArgoCD application)
- ionos_<env>: it will host application manifest files (see Add component manifest files)

N.B. The **applications** and **ionos** directories are currently **reserved** for the demo environment. For the continuation of the guide, we will use the ???dev??? environment, which serves as a playground where teams can test the integration of their components within the DOME ecosystem.

Add component manifest files

The team incorporates their integration by adding either a Helm chart or plain Kubernetes manifests to a properly named folder under the designated directory. This directory serves as a centralised location for all integrations.

First, create a directory for your application by executing the following commands:

cd ionos_dev/
mkdir <name of you application>

Then put your application manifest files or Helm charts inside this directory. Once done, the structure should look like the following:

```
ionos_dev/
  ????????? your-app/
      ????????? deployment.yaml
      ????????? service.yaml
      ????????? ingress.yaml
      ????????? secret.yaml
      ????????? configmap.yaml
```

or, if you are using Helm:

```
ionos_dev/
  ????????? your-app/
      ????????? templates/
          ????????? secret.yaml
          ????????? configmap.yaml
```

????????? Chart.yaml
????????? values.yaml

Add secrets

Using GitOps, means every deployed resource is represented in a git-repository. While this is not a problem for most resources, secrets need to be handled differently. We use the bitnami/sealed-secrets project for that.

To encrypt secrets from your local machine to the remote cluster, it is necessary to install kubeseal. Once installed, you can create an encrypted secret as follows:

1. Create a plain secret manifest file named <secret name>-plain-secret.yaml (**IMPORTANT**: Make sure the file name ends with *-plain-secret.yaml* so it will be ignored when pushing to repository)

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret name>
  namespace: <app namespace>
data:
  <key>: <base64 encoded value>
```

2. Seal the secret by running the following command:

kubeseal -f **<secret name>**-plain-secret.yaml -w **<secret name>**-sealed-secret.yaml --controller-namespace sealed-secrets --controller-name sealed-secrets

Or using the script SealSecret:

Windows PowerShell

.\scripts\SealSecret.ps1 -secretPath **<path to plain secret>**

Shell

```
# chmod +x ./scripts/SealSecret.sh
./scripts/SealSecret.sh <path to plain secret>
```

Note: If you are using Helm charts, make sure to place the sealed secret file under the directory *your-app/templates*.

Add ArgoCD application

Now that the manifest files for your component are ready, the next step is to create the application for ArgoCD.

First, navigate to the ???application??? directory:

cd applications_dev/

and create a file name *your-app.yaml* with the following content:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: <app name>
  namespace: argocd
  labels:
    purpose: <app purpose>
spec:
  destination:
    namespace: <app namespace>
    server: https://kubernetes.default.svc
  project: default
  source:
    path: ionos_dev/<app name>
    repoURL: https://github.com/DOME-Marketplace/dome-gitops
    targetRevision: HEAD
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

If your application is complex and consists of multiple components but you still want to manage it as a single ArgoCD application, you can use the following approach:

1. During the step ???Add component manifest files???, create a directory for each sub-component (i.e. *ionos_dev/your-app/sub-component-1*, *ionos_dev/your-app/sub-component-2* etc)
2. Create the directory *applications/your-app*
3. Inside *applications_dev/your-app*, create an ArgoCD application file for each sub-component, ensuring that it points to the respective subfolder under *ionos_dev/your-app*

```
    source:
  path: ionos_dev/<app name>/<sub-component name>
  repoURL: https://github.com/DOME-Marketplace/dome-gitops
```

targetRevision: HEAD

4. Create an ArgoCD application file for your entire application and make it point to applications/your-app

source:
path: applications_dev/**<app name>**
repoURL: https://github.com/DOME-Marketplace/dome-gitops
targetRevision: HEAD

You can use the marketplace application or dome-trust as a reference.

Create a Pull Request

Upon completing the changes, the team initiates a pull request from their branch to the main one. This PR serves as a formal request for the integration to be reviewed and merged. Team members must wait for the review process to be completed.

Once the pull request is approved and merged into the main branch, the GitOps pipeline automatically triggers the deployment process. This involves synchronising the desired state of the cluster with the changes introduced in the merged pull request. The deployment is executed based on the Helm chart or manifest configurations added to the repository.