- acend gmbh

Setup

When not using the preinstalled Web IDE provided by your Trainer, it’s also possible to use your local computer.

Local Environment Requirements

In this Training its required to have the following tools locally installed on your computer:

- git
- git bash on Windows
- oc Tool (OpenShift Client) *Only when on OpenShift*
- kubectl

oc tool

Follow the instructions on this page to install the oc tool on your local computer.

kubectl

Follow the instructions on this page to install the kubectl on your local computer.

Labs

Argo CD Architecture

Argo CDs core components are the API Server, the Repository Server and the Application Controller

- acend gmbh

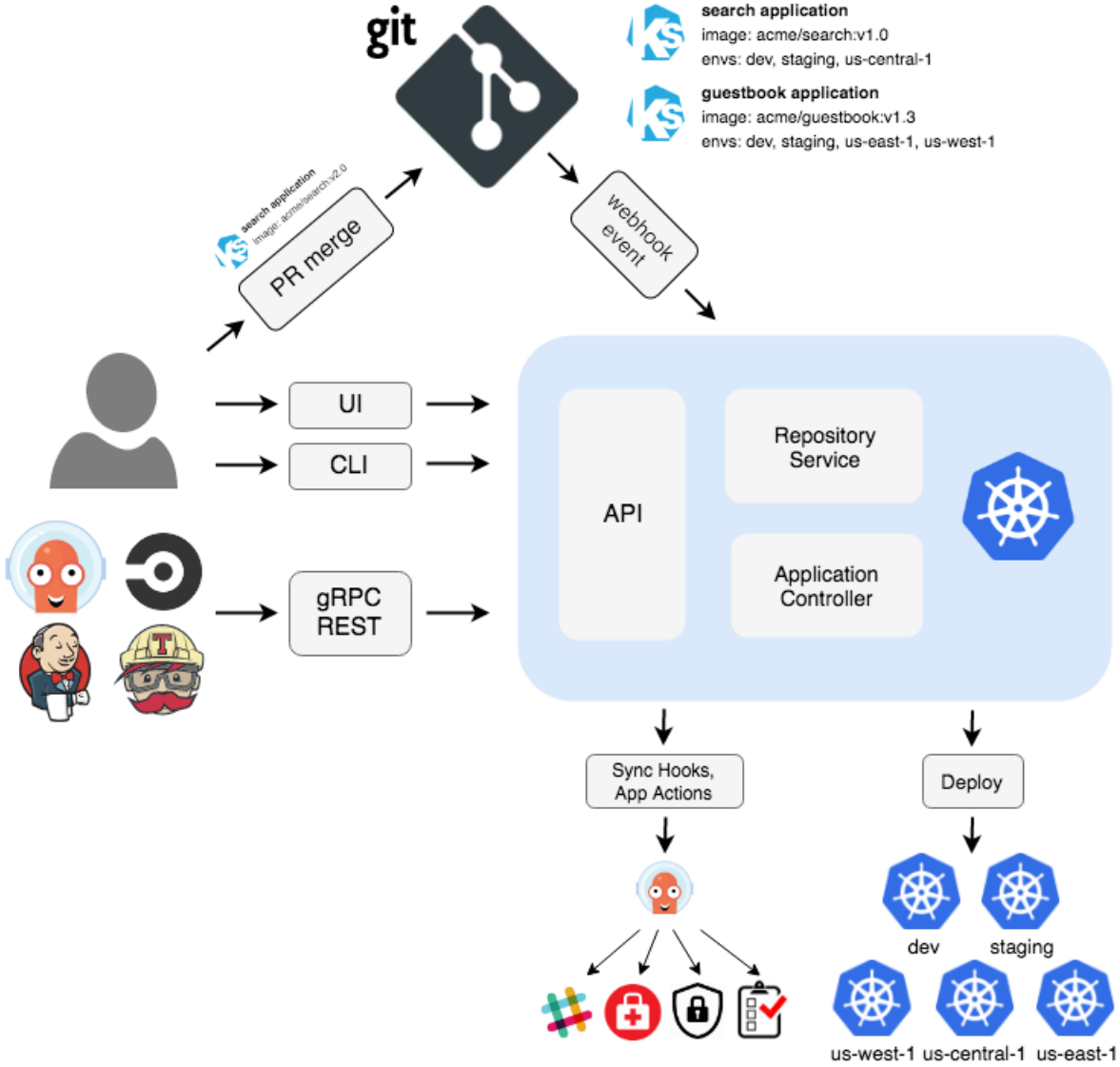


Image and component description source: https://argoproj.github.io/argo-cd/

API Server

The API server is a gRPC/REST server which exposes the API consumed by the Web UI, CLI, and CI/CD systems. It has the following responsibilities:

- application management and status reporting
- invoking of application operations (e.g. sync, rollback, user-defined actions)
- repository and cluster credential management (stored as K8s secrets)
- authentication and auth delegation to external identity providers
- RBAC enforcement
- listener/forwarder for Git webhook events

Repository Server

The repository server is an internal service which maintains a local cache of the Git repository holding the application manifests. It is responsible for generating and returning the Kubernetes manifests when provided the following inputs:

- acend gmbh
  - repository URL
  - revision (commit, tag, branch)
  - application path
  - template specific settings: parameters, ksonnet environments, helm values.yaml

Application Controller

The application controller is a Kubernetes controller which continuously monitors running applications and compares the current, live state against the desired target state (as specified in the repo). It detects OutOfSync application state and optionally takes corrective action. It is responsible for invoking any user-defined hooks for lifecycle events (PreSync, Sync, PostSync)

Argo CD Core Concepts

Those core Concepts exist in Argo CD:

- Clusters: pre configured Kubernetes Clusters (including OpenShift)
- Repositories : pre configured git repositories, including repository credentials (ssh, username-password).
- Applications : A group of Kubernetes resources, represented in a git repository. Usually the Kubernetes resources which will be applied in a Kubernetes namespace. Represented as CRD .
- Projects : A logical grouping of Argo CD applications. Various restrictions can be defined on project level. Useful when multiple Teams work with the same Argo CD instance

1. Getting started

Task 1.1: Web IDE

The first thing we're going to do is to explore our lab environment and get in touch with the different components.

The namespace with the name corresponding to your username is going to be used for all the hands-on labs. And you will be using the ArgoCD webconsole, to verify what resources and objects Argo CD created for you.

> **Note**
> You can also use your local installation of the cli tools. Make sure you completed the setup before you continue with this lab.

> **Note**
> The URL and Credentials to the Web IDE will provided by the teacher. Use Chrome for the best experience.

Once you're successfully logged into the web IDE open a new Terminal by hitting `CTRL + SHIFT + ¨` or clicking the Menu button –> Terminal –> new Terminal and check the installed ocversion by executing the following command:

```
```

- acend gmbh
The Web IDE Pod consists of the following tools:

- oc
- kubectl
- kustomize
- helm
- kubectx
- kubens
- tekton cli
- odo
- argocd

The files in the home directory under `/home/project` are stored in a persistence volume, so please make sure to store all your persistence data in this directory.

Task 1.1.1: Local Workspace Directory

During the lab, you'll be using local files (eg. YAML resources) which will be applied in your lab project.

Create a new folder for your `<workspace>` in your Web IDE (for example `argocd-training` under `/home/project/argocd-training`). Either you can create it with `right-mouse-click -> New Folder` or in the Web IDE terminal.

```
mkdir argocd-training && cd argocd-training
```

Task 1.1.2: Login to ArgoCD

You can access Argo CD via the Web UI. Open your browser and navigate to https://argocd.training.cluster.acend.ch and login with the credentials provided by your trainer.

Task 1.1.3: Lab Setup

Most of the labs will be done inside the OpenShift project with your username. Verify that your oc tool is configured to point to the right project:

```
oc project
```

```
Using project "<username>" on server "https://<redacted>:6443".
```

The returned project name should correspond to your username.

2. Simple Example

In this lab you will learn how to deploy a simple application using Argo CD.

Our lab setup consists of the following components:

- Git Server (Gitea): https://gitea.training.cluster.acend.ch

- acend gmbh
  - Argo CD Server: https://argocd.training.cluster.acend.ch
  - OpenShift Cluster

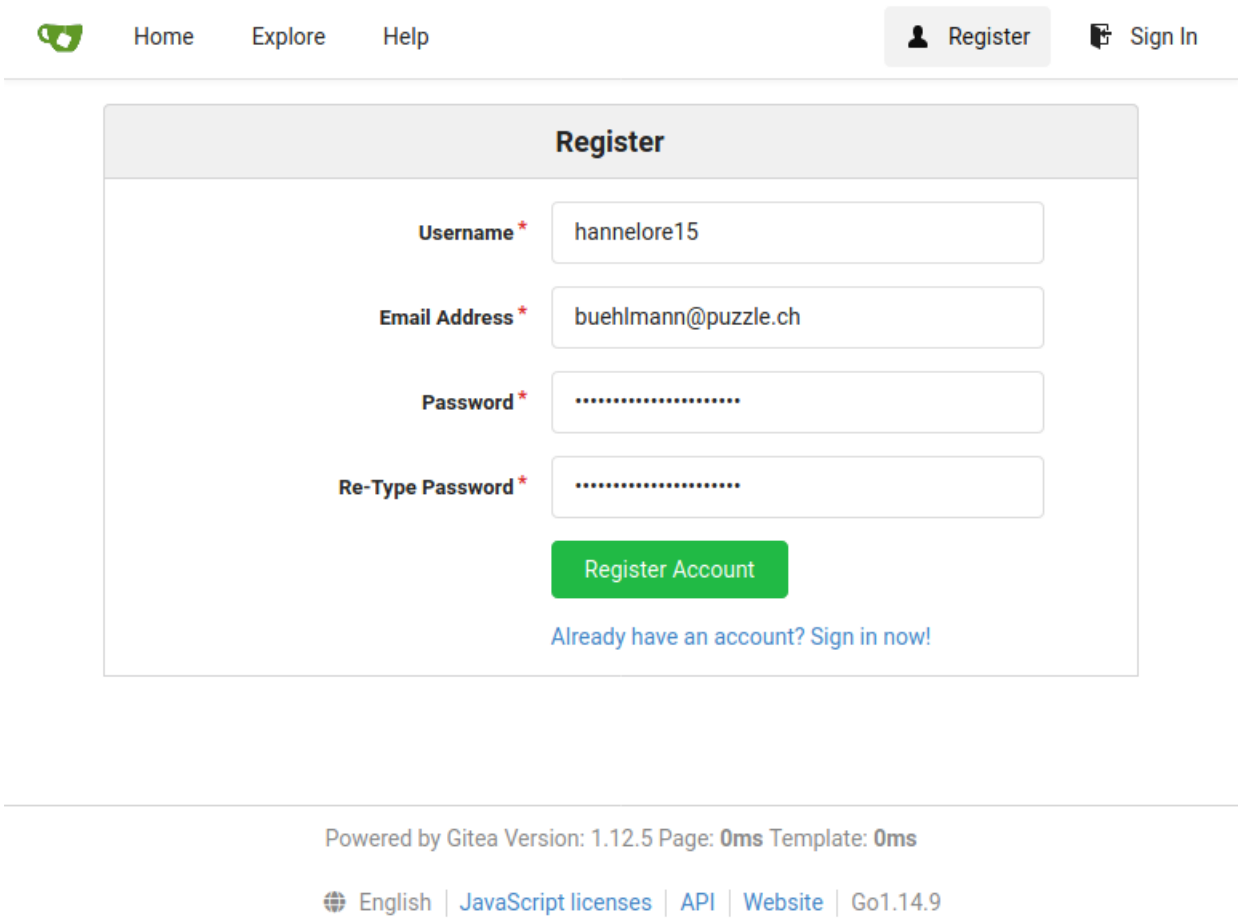
Task 2.1: Fork the Git repository

As we are proceeding according to the GitOps principle we need some example resource manifests in a Git repository which we can edit.

Users which have a personal Github account can just fork the Repository argocd-training-examples to their personal account. To fork the repository click on the top right of the Github on *Fork*.

All other users can use the provided Gitea installation of the personal lab environment. Visit https://gitea.training.cluster.acend.ch with your browser and register a new account with your personal username and a password that you can remember ;)

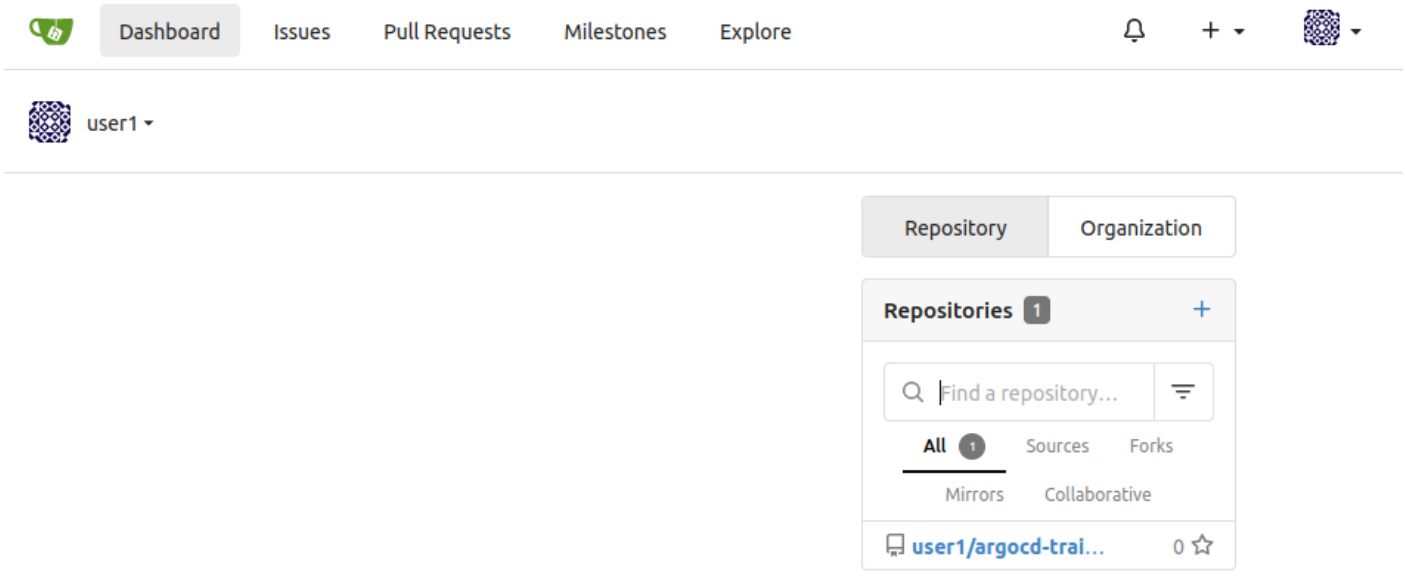
> **Note**
> All the cli commands in this chapter must be executed in the terminal of the provided Web IDE.



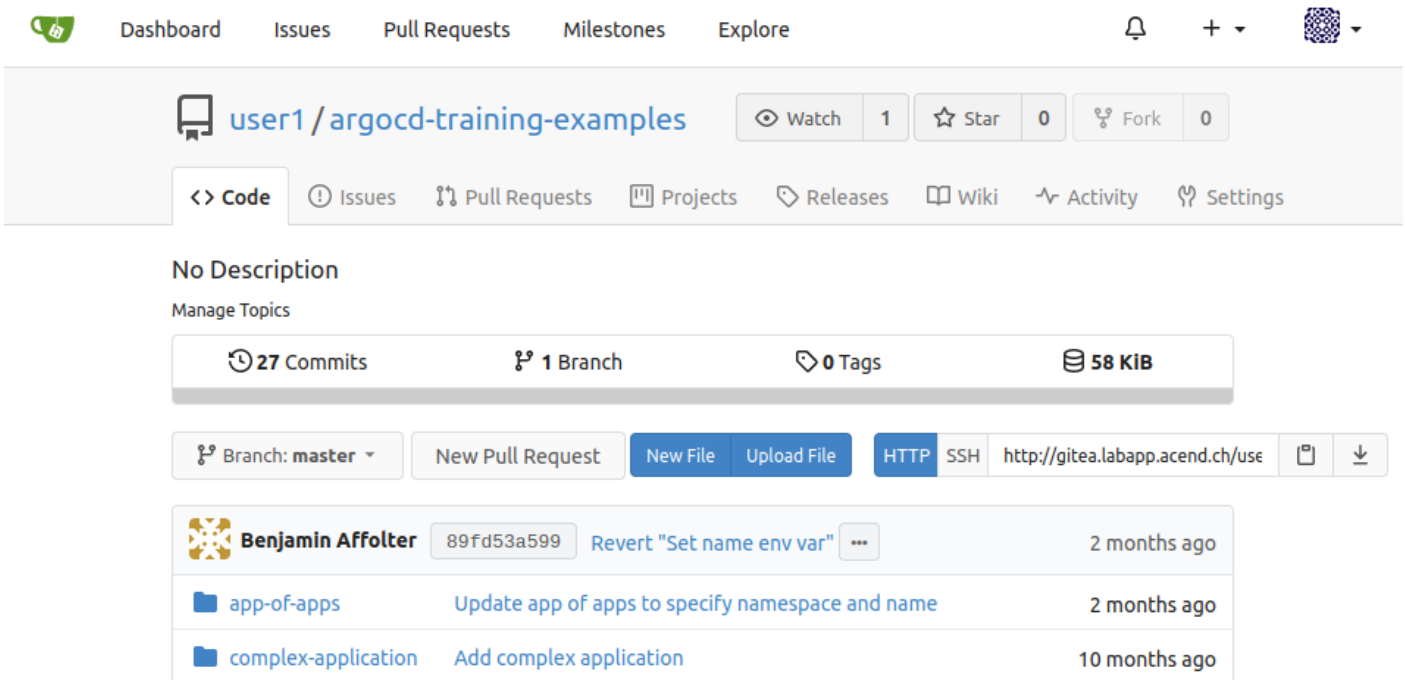
Login with the new user and fork the existing Git repository from Github:

1. Select *Create* on the top right -> *New Migration* -> Select *GitHub*
2. Migrate / Clone From URL: https://github.com/acend/argocd-training-examples.git
3. Click *Migrate Repository*

- acend gmbh
The Git Repository is available under your Repositories



By clicking on the repository link in the repository list you get to the detail page.



The **URL** of the Git repository, we'll be working with, will look like
`https://gitea.training.cluster.acend.ch/<username>/argocd-training-examples.git` .

Within the Web IDE we set the `USER` environment variable to your personal `<username>` .

- acend gmbh
Verify that with the following command:

```
...
```

The `USER` variable will be used as part of the commands to make the lab experience more comfortable for you.

> **Note**
> If you're **not** using our lab webshell to execute the labs, make sure to set the `USER` environment variable accordingly with the following command `export USER=<username>`

Clone the forked repository to your local workspace:

```
...
```

... or the corresponding URL if you have choosen to use your own Git Server.

Change the working directory to the cloned git repository:

```
...
```

When using the Web IDE: Configure the Git Client and verify the output

```
...
```

And we also want git to store our Password for the whole day so that we don't need to login every single time we push something.

```
...
```

Then use the following command to verify whether the git config for username and email were correctly added:

```
...
```

Task 2.2: Deploying the resources with Argo CD

Now we want to deploy the resource manifests contained in the cloned repository with Argo CD to demonstrate the basic features of Argo CD.

Create a file `example-application.yaml` with the following content:

- acend gmbh

```
[illegible]
```

Apply it to the cluster:

```
[illegible]
```

Expected output: `application 'example-application-<username>' created`

Argo CD will now detect the application. Once the application is created, you can view its status:

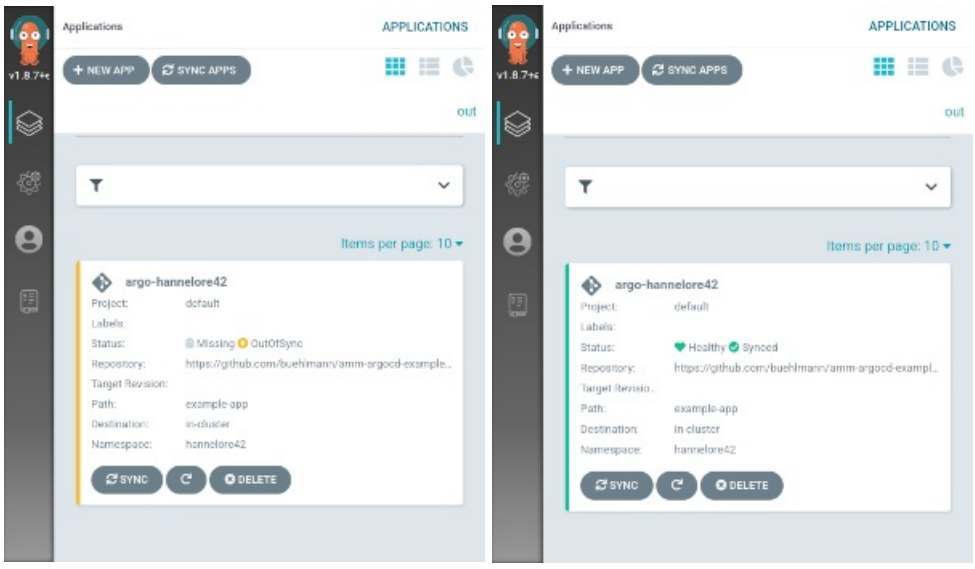
```
[illegible]
```

Open the Argo CD UI and click **Sync** to deploy the resources. This command retrieves the manifests from the git repository and performs a oc apply on them. From now on, all resources are managed by Argo CD. Congrats, the first step in direction GitOps! :)

Once synced the application status will show as **Healthy**.

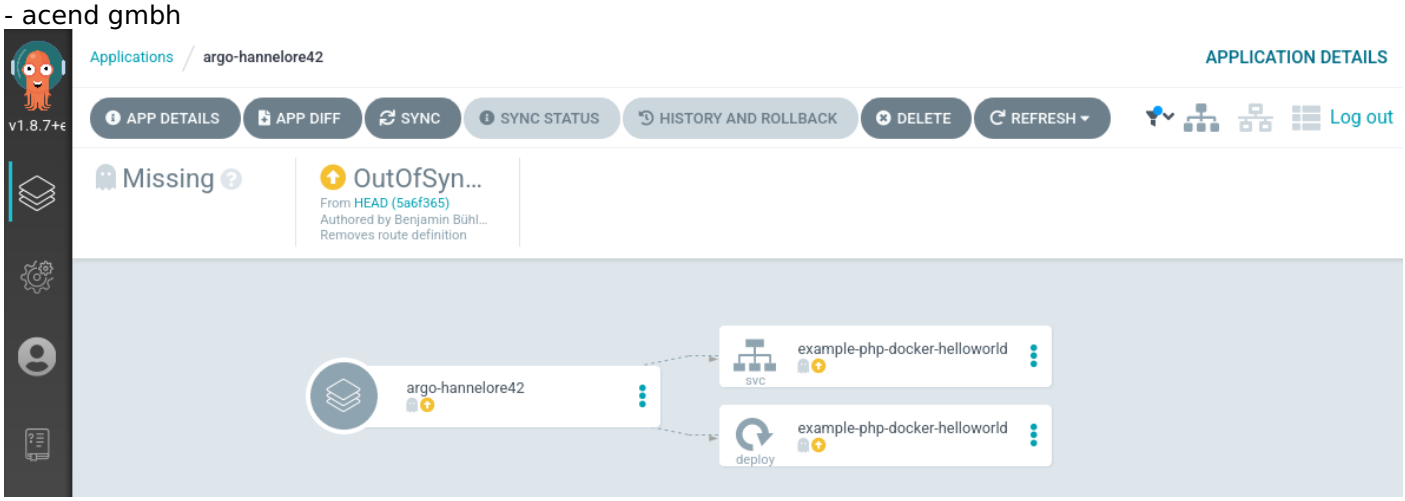
```
[illegible]
```

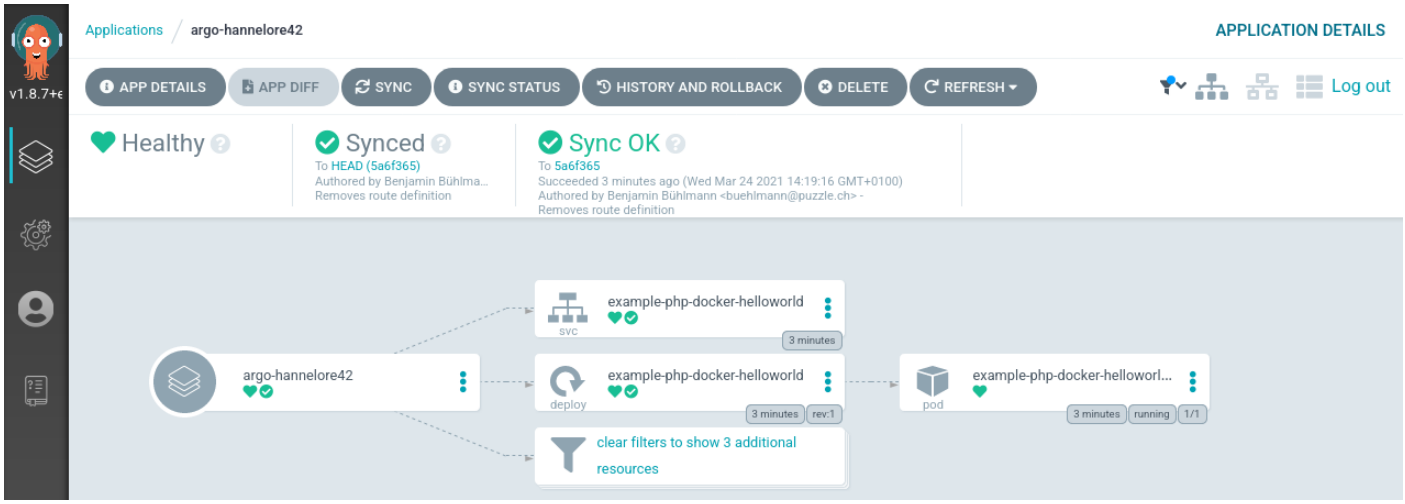
Application overview in unsynced and synced state



Detailed view of a application in unsynced and synced state

- acend gmbh





Task 2.3: Automated Sync Policy and Diff

When there is a new commit in your Git repository, the Argo CD application becomes OutOfSync. Let's assume we want to scale up our `Deployment` of the example application from 1 to 2 replicas. We will change this in the Deployment manifest.

Increase the number of replicas in your file `<workspace>/example-app/deployment.yaml` to 2.

Commit the changes and push them to your personal remote Git repository. After the Git push command a **password** input field will appear at the top of the Web IDE.

After a successful push you should see the following output

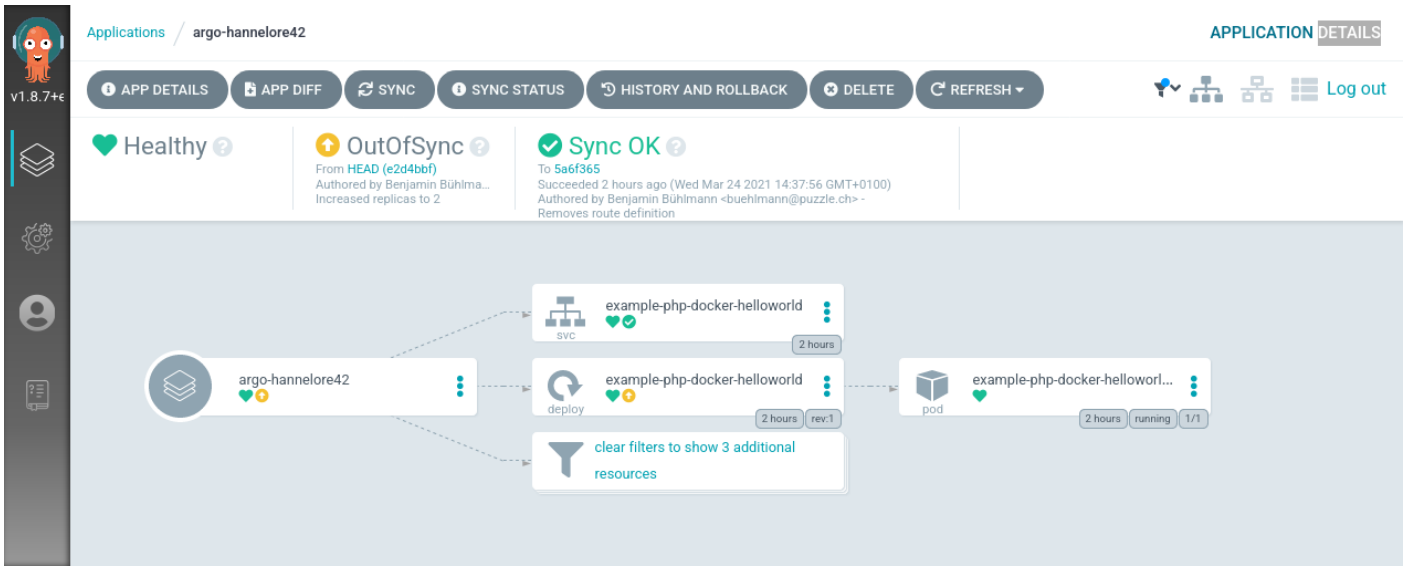
- acend gmbh

```
[redacted]
```

Out of the box Git will be polled by Argo CD in a predefined interval (defaults to 3 minutes). To use a synchronous workflow you can use webhooks in Git. These will trigger a synchronization in Argo CD on every push to the repository.

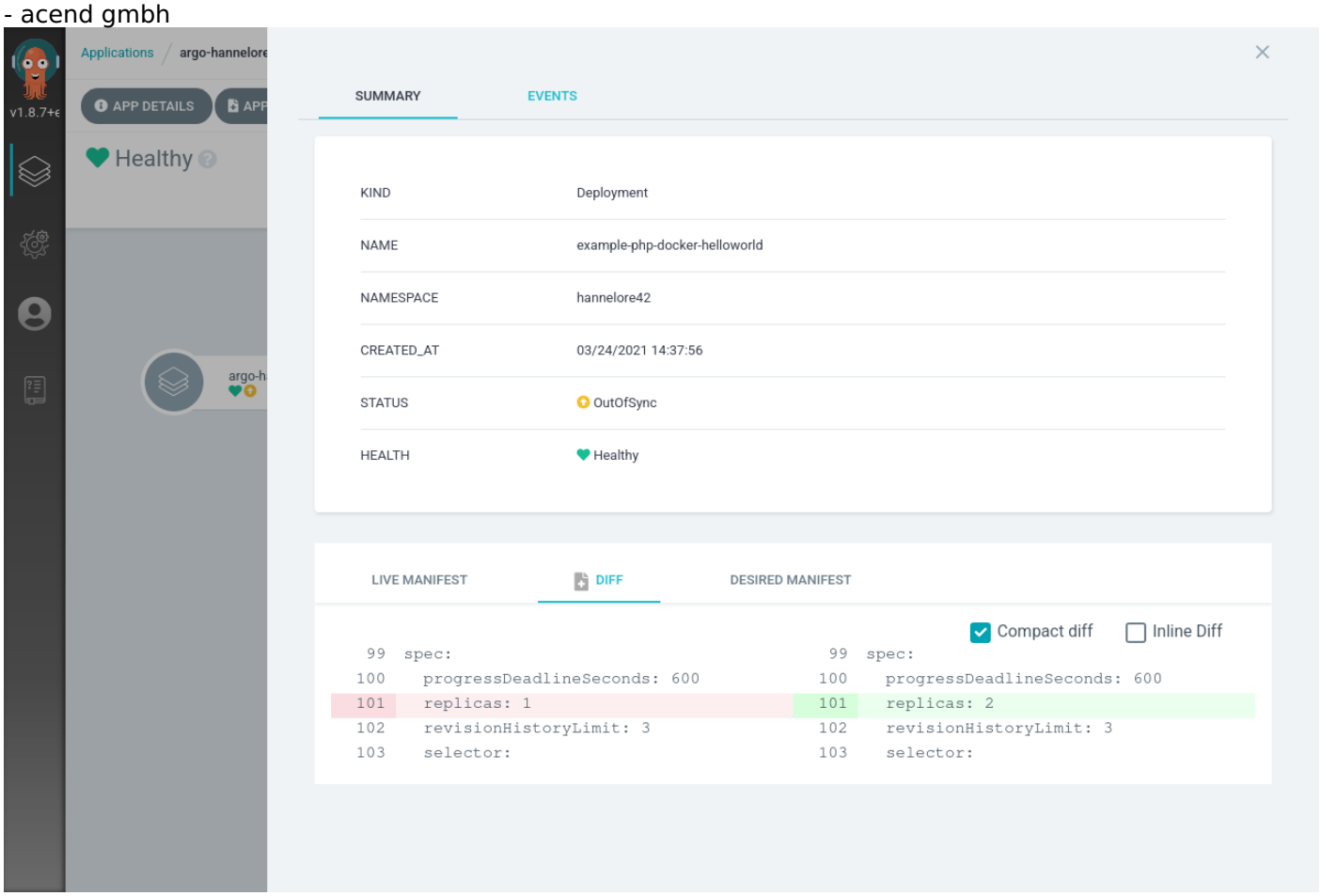
Open the Argo CD UI and click **Refresh** on the `argo-$USER` application to trigger an immediate update.

Now open the web console of Argo CD and go to your application. The deployment `simple-example` is marked as 'OutOfSync':



When an application is OutOfSync then your deployed 'live state' is no longer the same as the 'target state' which is represented by the resource manifests in the Git repository. You can inspect the differences between live and target state with a click on Deployment > Diff:

- acend gmbh



Now click `Sync` on the top left and let the magic happen ;) The application will be scaled up to 2 replicas and the resources are in Sync again.

Argo CD can automatically sync an application when it detects differences between the desired manifests in Git, and the live state in the cluster. A benefit of automatic sync is that CI/CD pipelines no longer need direct access to the Argo CD API server to perform the deployment. Instead, the pipeline makes a commit and push to the Git repository with the changes to the manifests in the tracking Git repo.

To configure automatic sync, edit the example-application.yaml (or use the UI):

```
```

and re-apply the manifest:

```
```

From now on Argo CD will automatically apply all resources to Kubernetes every time you commit to the Git repository.

Decrease the replicas count to 1 and push the updated manifest to remote. Wait for a few moments and see check that ArgoCD will scale the deployment of the example app down to 1 replica. The default polling interval is 3 minutes. If you don’t want to wait you can force a refresh by clicking `Refresh` in the UI .

- acend gmbh

Task 2.4: Automatic Self-Healing

By default, changes made to the live cluster will not trigger automatic sync. To enable automatic sync when the live cluster's state deviates from the state defined in Git, edit `example-application.yaml` to set `selfHeal: true` and re-apply:

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

```
oc apply -f example-application.yaml
```

Watch the deployment `simple-example` in a separate terminal:

```
oc get deployment simple-example -watch -n<namespace>%USER
```

Let's scale our `simple-example` Deployment and observe whats happening:

```
oc scale deployment simple-example --replicas=5 -n<namespace>%USER
```

Argo CD will immediately scale back the `simple-example` Deployment to `1` replicas. You will see the desired replicas count in the watched Deployment.

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
simple-example   1/5     5            1           119m
simple-example   2/5     5            2           119m
simple-example   3/5     5            3           119m
simple-example   4/5     5            4           119m
simple-example   5/5     5            5           119m
simple-example   4/5     1            4           119m
simple-example   3/5     1            3           119m
simple-example   2/5     1            2           119m
simple-example   1/5     1            1           119m
```

This is a great way to enforce a strict GitOps principle. Changes which are manually made on deployed resource manifests are reverted immediately back to the desired state by the ArgoCD controller.

Task 2.5: Expose Application

This is an optional task.

To expose an application we need to specify a so called `route` resource. Create a `route.yaml` file next to the `deployment.yaml` in the example-app directory.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: simple-example
spec:
  port:
    targetPort: 8080
  to:
    kind: Service
    name: simple-example
    weight: 100
  wildcardPolicy: None
```

Commit and Push the changes again, like you did before:

```
git add .
git commit -m "Finished application"
git push
```

After ArgoCD syncs the changes, you can access the example applications url: `https://simple-example-<username>.training.cluster.acend.ch`

Verify using the following command:

```
curl https://simple-example-$USER.training.cluster.acend.ch
```

The result should look similar to this:

```
<h1 style=color:#e81198>Hello golang</h1><h2>ID: e81198</h2>
```

Task 2.6: Pruning

You probably asked yourself: how can I delete deployed resources on the container platform? Argo CD can be configured to delete resources that no longer exist in the Git repository.

First delete the files `service.yaml` and `route.yaml` from Git repository and push the changes:

```
git add .
git add --all && git commit -m 'Removes service and ingress' && git push
```

Open the [Argo CD UI]() and click **Refresh** on the application. You will see that even with auto-sync enabled the resources are still OutOfSync.

To enable pruning, edit `example-application.yaml` and re-apply:

```
syncPolicy:
  automated:
    selfHeal: true
    prune: true
```

```
oc apply -f example-application.yaml
```

Click **Refresh** again in the UI. The Service and Ingress/Route will now be pruned (deleted) by Argo CD.

The Service was successfully deleted by Argo CD because the manifest was removed from git. See the HEALTH and MESSAGE of the previous console output.

Task 2.7: State of ArgoCD

Argo CD is largely built stateless. The configuration is persisted as native Kubernetes objects. And those are stored in Kubernetes *etcd*. There is no additional storage layer needed to run ArgoCD. The Redis storage under the hood acts just as a throw-away cache and can be evicted anytime without any data loss.

The configuration changes made on ArgoCD objects through the UI or by CLI are reflected in updates of the ArgoCD Kubernetes objects `Application` and `AppProject` in the `openshift-gitops` namespace.

Let’s list all Kubernetes objects of type `Application` (short form: `app`)

- acend gmbh

```
```

```
```

You will see the application which we created. To see the complete configuration of the `Application` as *yaml* use:

```
```

You even can edit the `Application` resource by using:

```
```

This allows us to manage the ArgoCD application definitions in a declarative way as well. It is a common pattern to have one ArgoCD application which references n child Applications which allows us a fast bootstrapping of a whole environment or a new cluster. This pattern is well known as the *App of apps* pattern.

Task 2.8: Accessing a private Git repository

You can define credential templates when using the same credential for multiple Git repositories. The configured credentials are used for each Git repository beginning with the configured URL.

A credential template is a `Secret` with the label `argocd.argoproj.io/secret-type: repo-creds` and a URL prefix instead of a full repository URL. Argo CD will use its credentials for every repository whose URL starts with that prefix.

```
```

```
```

For example, a template for `https://gitea.training.cluster.acend.ch/my-group` would cover all repositories within that group without needing a separate secret per repository.

Finally make your personal Git repository public again for the following labs. Uncheck the option `Visibility: Make Repository Private` under `Settings -> Repository` in the Gitea UI.

> **Note**
> TLS certificates and SSH private keys are supported alternative authentication methods by Argo CD. Proxy support can be configured as well in the repository settings.

Have a look in the documentation for detailed information about accessing private repositories.

Since the forked repository is public, no additional credential configuration is needed. Private repository

access is managed via the Argo CD UI under **Settings → Repositories** if required.

Task 2.9: Delete the Application

You can cascading delete the ArgoCD Application with the following command:

```
oc delete application argo-$USER -n openshift-gitops
```

This will delete the `Application` resource. Since automated pruning is enabled, Argo CD will also delete the managed `Deployment` and `Service` from the namespace.

3. Resource Hooks

In this Lab you are going to learn about [Resource Hooks](#) .

Resource Hooks

Hooks allow to run scripts before, during and after the Argo CD **sync** operation is running. They give you more control over the sync process. They can also run when the sync operation fails for example. The concept is very similar to the concept of [Helm Hooks](#) . Argo CD supports many Helm hooks by mapping the Helm annotations onto Argo CD’s own hook annotations. You can see the full mapping of the Helm hooks [in the ArgoCD documentation](#)

Some examples when hooks can be useful:

- `PreSync` hook. Upgrading a Database, Performing a migration before deploying a new version of the application.
- `PostSync` hook. Run integration, smoke and other tests after the deployment to verify its status.
- `Sync` hook. Allows to run more complex deployment strategies. e.g.: Blue-Green or Canary Deployments
- `SyncFail` hook. Clean up a failed deployment.

Hooks are annotated `argocd.argoproj.io/hook: <hook>` Kubernetes resources in the source repository, which Argo CD will apply during the sync operation.

A `PreSync` Hook to run a database migration might therefore look like this:

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  generateName: schema-migrate-
  annotations:
    argocd.argoproj.io/hook: PreSync
...
```

It’s basically a [Kubernetes Job](#) which starts a Pod that executes some sort of code.

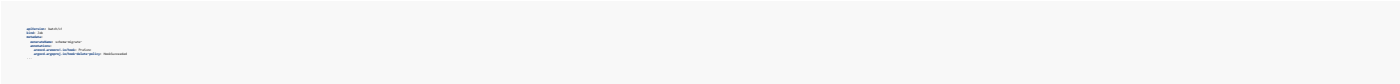
> **Note**
> Named hooks (i.e. ones with `/metadata/name`) will only be created once. If you want a hook to be re-created each time either use BeforeHookCreation policy or `/metadata/generateName`.

> **Note**
> Hooks are not run during a [selective sync](#)

- acend gmbh

Hook Deletion Policies

The hook deletion policy defines when a hook should be deleted. It's also configured with an annotation `argocd.argoproj.io/hook-delete-policy` on the hook resource.



- `HookSucceeded` : will be deleted after the hook succeeded
- `HookFailed` : will be deleted after a hook failed
- `BeforeHookCreation` : Any hook resource will be deleted before the new one is created.

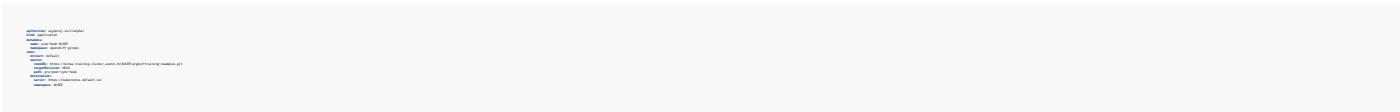
Task 3.1: Hook Example

In this task we're going to deploy an example which has `pre` and `post` hooks.

Create the new application `argo-hook-$USER` with the following command. It will create a service, a deployment and two hooks as soon as the application is synced.

- PreSync: before Job
- Sync: Deployment with name `pre-post-sync-hook`
- PostSync: after Job

Create a file `argocd-hook-application.yaml` with the following content and apply it:



Sync the application

▼ Hint
Open the Argo CD UI and click **Sync** on the `argo-hook-$USER` application.

And verify the deployment:

Or in the web UI.

Task 3.2: Post-hook Curl (Optional)

Alter the post sync hook command from `sleep` to `curl https://acend.ch` (Could be used to send a notification to a Chat channel) The curl command is not available in the minimal `quay.io/acend/example-web-go` image. You can use `quay.io/acend/example-web-python` or different image.

- acend gmbh
Edit the hook under `pre-post-sync-hook/post-sync-job.yaml` accordingly, commit and push the changes and trigger the sync operation.

Task 3.3: Delete the Application

Delete the application after you've explored the Argo CD Resources and the managed Kubernetes resources.

▼ Hint

4. Sync Phases and Waves

In this Lab you are going to learn about Sync Phases and Waves .

Sync Phases and Waves

At a high-level, Argo CD executes the sync operation in the three phases pre-sync, sync and post-sync.

Within each phase you can have one or more waves, that allows you to ensure certain resources are healthy before subsequent resources are synced.

When Argo CD starts a sync, it orders the resources in the following precedence:

- The phase
- The wave they are in (lower values first)
- By kind (e.g. namespaces first)
- By name

It then determines the number of the next wave to apply. This is the first number where any resource is out-of-sync or unhealthy.

It applies resources in that wave.

It repeats this process until all phases and waves are in-sync and healthy.

How to specify waves and phases

Pre-sync and post-sync can only contain hooks defined on annotations `argocd.argoproj.io/hook: PreSync` .

You can specify the wave in the sync phase by setting an annotation `argocd.argoproj.io/sync-wave` . Hooks and resources are assigned to wave zero by default. The wave can be negative, so you can create a wave that runs before all other resources.

Task 4.1: Sync Wave Example

Let’s now get our hands on a sync wave example.

Create the new application `argo-wave-$USER` with the following command. The Application consist of the following resources, phases and waves:

- PreSync
  - Job: upgrade-sql-schema
- Sync Wave 0
  - Deployment: backend
  - Service: backend
- Sync Wave 1
  - Job: maintenance-page-up
- Sync Wave 2
  - Deployment: frontend
  - Service: frontend
- Sync Wave 3
  - Job: maintenance-page-down

Create a file `argocd-wave-application.yaml` with the following content and apply it:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-wave-$USER
  namespace: openshift-gitops
spec:
  project: default
  source:
    repoURL: https://gitea.training.cluster.acend.ch/$USER/argocd-training-examples.git
    targetRevision: HEAD
    path: sync-wave
  destination:
    server: https://kubernetes.default.svc
    namespace: $USER
```

```
oc apply -f argocd-wave-application.yaml
```

Sync the application:

▼ Hint

Open the Argo CD UI and click **Sync** on the `argo-wave-$USER` application.

And verify the deployment:

```
oc get pod --namespace $USER --watch
```

Task 4.2: Delete the Application

Delete the application after you’ve explored the Argo CD Resources and the managed Kubernetes resources.

▼ Hint

```
oc delete application argo-wave-$USER -n openshift-gitops
```

5. Tools

- acend gmbh
In this Lab you are going to learn about different application source tools .

Tools

As mentioned in the introduction Argo CD supports many different formats in which the Kubernetes manifests can be defined:

- kustomize applications
- helm charts
- ksonnet applications (deprecated)
- jsonnet files
- Plain directory of YAML/json manifests
- Any custom config management tool configured as a config management plugin

So far you have been using **plain YAML** manifest in the previous labs.

> **Warning**
> Argo CD provides a mechanism to override the parameters of Argo CD applications. The Argo CD parameter overrides feature is provided mainly as a convenience to developers and is intended to be used in dev/test environments, vs. production environments.
>
> Many consider this feature as anti-pattern to GitOps. So only use this feature when no other option is available!

Tool Detection

When the build tool is not specified explicitly in the Argo CD Application CRD it will be detected:

- `Helm` if there's a file matching `Chart.yaml` .
- `Kustomize` if there's a `kustomization.yaml` , `kustomization.yml` , or `Kustomization`
- `jsonnet` if there's a `*.jsonnet` file.

You are now going to deploy an application in the different formats.

You can also find additional examples here .

5.1. Helm

This lab explains how to use Helm as manifest format together with Argo CD.

Helm Introduction

Helm is a Cloud Native Foundation project to define, install and manage applications in Kubernetes.

It can be used to package multiple Kubernetes resources into a single logical deployment unit.

Helm Charts are configured using `values.yaml` files. (e.g. images, image tags, hostnames, ...).

When using `helm` charts together with Argo CD we can specify the `values.yaml` like this:

- acend gmbh

```
# In the application spec.source.helm section:
valueFiles:
  - values-production.yaml
```

The `--values` flag can be repeated to support multiple values files.

> **Info**
> Values files must be in the same git repository as the Helm chart. The files can be in a different location in which case it can be accessed using a relative path relative to the root directory of the Helm chart.

Helm Parameters

Similar to when using `helm` directly (`helm install <release> --set replicaCount=2 ./mychart --namespace <namespace>`), you are able to overwrite values from the values.yaml, by setting parameters.

```
# In the application spec.source.helm section:
parameters:
  - name: replicaCount
    value: '2'
```

> **Warning**
> Argo CD provides a mechanism to override the parameters of Argo CD applications. The Argo CD parameter overrides feature is provided mainly as a convenience to developers and is intended to be used in dev/test environments, vs. production environments.
>
> Many consider this feature as anti-pattern to GitOps. So only use this feature when no other option is available!

Helm Release Name

By default, the Helm release name is equal to the Application name to which it belongs. Sometimes, especially on a centralised ArgoCD, you may want to override that name, and it is possible with the `release-name` flag on the cli:

```
# In the application spec.source.helm section:
releaseName: my-release
```

> **Warning**
> Please note that overriding the Helm release name might cause problems when the chart you are deploying is using the app.kubernetes.io/instance label. ArgoCD injects this label with the value of the Application name for tracking purposes.

Helm Hooks

Helm hooks are similar to the Argo CD Hooks from lab 4 .

Further Docs

Read more about the helm integration in the official documentation

Task 5.1.1: Deploy the simple-example as Helm Chart

Let’s deploy the simple-example from lab 1 using a helm chart .

First you’ll have to create a new Argo CD application.

Create a file `argocd-helm-application.yaml` with the following content and apply it:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-helm-$USER
  namespace: openshift-gitops
spec:
  project: default
  source:
    repoURL: https://gitea.training.cluster.acend.ch/$USER/argocd-training-examples.git
    targetRevision: HEAD
    path: helm/simple-example
    helm:
      valueFiles:
        - values.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: $USER
```

```
oc apply -f argocd-helm-application.yaml
```

Sync the application

▼ Hint
To sync (deploy) the resources you can simply click sync in the web UI.

And verify the deployment:

```
oc get pod --namespace $USER --watch
```

Tell the application to sync automatically, to enable self-healing and auto-prune

▼ Hint
Edit `argocd-helm-application.yaml` to add automated sync policy, then re-apply:

```yaml
  syncPolicy:
    automated:
      selfHeal: true
      prune: true
```

```
oc apply -f argocd-helm-application.yaml
```

Task 5.1.2: Scale the deployment to 2 replicas

We can set the `helm` parameter with the following command:

Edit `argocd-helm-application.yaml` to add the parameter override in `spec.source.helm` , then re-apply:

```yaml
    helm:
      valueFiles:
        - values.yaml
      parameters:
        - name: replicaCount
          value: "2"
```

```
oc apply -f argocd-helm-application.yaml
```

Warning

- acend gmbh

> Only use this way of setting params in dev and test stages. Not for Production!

Since the `sync-policy` is set to `automated` the second pod will be deployed immediately.

Task 5.1.3: Ingress

The proper and production ready way of overwriting values is by doing it in git.

Change the `helm/simple-example/values.yaml` file in your git repository

```
ingress:
  enabled: true
  hosts:
    - host: helm-<username>-training.cluster.acend.ch
      paths:
        - path: /
  tls:
    - hosts:
        - helm-<username>-training.cluster.acend.ch
      secretName: acend-wildcard
```

Commit and push the changes to your repository.

▼ Hint

```
git add helm/simple-example/values.yaml
git commit -m "Values ingress"
git push
```

Open your Browser and verify whether you can access the application.

Task 5.1.4: Create a second application representing the production stage

Let’s now also deploy an application for the production stage.

Create a new values.yaml file for the production stage: `helm/simple-example/values-production.yaml` And copy the content from the default `helm/simple-example/values.yaml` file.

Change the host in the `helm/simple-example/values-production.yaml` to the production url

```
ingress:
  enabled: true
  hosts:
    - host: helm-<username>-prod-training.cluster.acend.ch
      paths:
        - path: /
  tls:
    - hosts:
        - helm-<username>-prod-training.cluster.acend.ch
      secretName: acend-wildcard
```

Commit and push the changes to your repository.

▼ Hint

```
git add helm/simple-example/values-production.yaml
git commit -m "Add prod image"
git push
```

Let’s create the production stage Argo CD application with the name `argo-helm-prod-$USER` and enable automated sync, self-healing and pruning.

▼ Hint

Create a file `argocd-helm-application-prod.yaml` with the following content and apply it:

- acend gmbh

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-hello-prod-6d89
  namespace: acend-ch-groups
spec:
  project: default
  source:
    repoURL: https://gitea.training.cluster.acend.ch/6d89/argocd-training-examples.git
    targetRevision: HEAD
    path: helm-simple-example
    helm:
      valueFiles:
        - values-production.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: 6d89
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```
oc apply -f argocd-hello-applicationprod.yaml
```

And verify the deployment:

```
oc get pod --namespace 6d89 --watch
```

Change for example the ingress hostname to something different in the `values-production.yaml` and verify whether you can access the new hostname.

Task 5.1.5: Delete the Applications

Delete the applications after you've explored the Argo CD Resources and the managed Kubernetes resources.

▼ Hint

```
oc delete application argo-hello-6d89 argo-hello-prod-6d89 -n acend-ch-groups
```

5.2. Kustomize

This lab explains how to use kustomize as manifest format together with Argo CD.

Kustomize Introduction

Kustomize introduces a template-free way to customize application configuration that simplifies the use of off-the-shelf applications. It is built into `kubectl` and `oc` with the command `kubectl apply -k` or `oc apply -k` .

It uses a concept called overlays, which allows to reduce redundant configuration for multiple stages (e.g. dev, prod, test) without a use of a template language.

Argo CD supports kustomize manifests out of the box.

Kustomize Overlays

When you want to use Kustomize with an overlay, you have to point the Argo Application to the Overlay

Kustomize Configuration

The following configuration options are available for Kustomize:

- `namePrefix` is a prefix appended to resources for Kustomize apps
- `nameSuffix` is a suffix appended to resources for Kustomize apps

- acend gmbh
  - `images` is a list of Kustomize image overrides
  - `commonLabels` is a string map of an additional labels
  - `commonAnnotations` is a string map of an additional annotations

Use the following command to set those parameters:

```
# In the application.spec.source.kustomize section:
namePrefix: newPrefix-
```

Further Docs

Read more about the kustomize integration in the official documentation

Task 5.2.1: Deploy the simple-example with kustomize

Let’s deploy the simple-example from lab 1 using kustomize .

First you’ll have to create a new Argo CD application.

Create a file `argocd-kustomize-application.yaml` with the following content and apply it:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-kustomize-<USER>
  namespace: acend-k8s-argo
spec:
  destination:
    server: default
  project:
    repoURL: https://github.com/acend/acend.ch/<USER>/argo-training/examples.git
    targetRevision: HEAD
    path: argo-training/kustomize/example
  destination:
    server: https://kubernetes.default.svc
    namespace: <USER>
```

```
oc apply -f argocd-kustomize-application.yaml
```

Sync the application

▼ Hint
To sync (deploy) the resources you can simply click sync in the web UI.

And verify the deployment:

```
oc get pod --namespace <USER> --watch
```

Tell the application to sync automatically, to enable self-healing and auto-prune

▼ Hint
Edit `argocd-kustomize-application.yaml` to add automated sync policy, then re-apply:

```
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```
oc apply -f argocd-kustomize-application.yaml
```

- acend gmbh

Task 5.2.2: Set a configuration parameter

We can set the `kustomize` configuration parameter with the following command:

Edit `argocd-kustomize-application.yaml` to add the nameprefix in `spec.source.kustomize`, then re-apply:

```
kustomize:
  namePrefix: acend
```

```
oc apply -f argocd-kustomize-application.yaml
```

And take a look at the application in the web UI.

> **Warning**
> Only use this way of setting params in dev and test stages. Not for Production!

Task 5.2.3: Create a second application representing the production stage

Let’s now also deploy an application for the production stage.

This does mean we deploy an overlay which specifically configures the production stage.

Let’s create the production stage Argo CD application (path: `kustomize/overlays-example/overlays/production`) with the name `argo-kustomize-prod-$USER` and enable automated sync, self-healing and pruning.

▼ Hint
Create a file `argocd-kustomize-application-prod.yaml` with the following content and apply it:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-kustomize-prod-$USER
  namespace: userXX-gitops
spec:
  project: default
  source:
    repoURL: https://gitea.training.cluster.acend.ch/$USER/argocd-training-examples.git
    targetRevision: HEAD
    path: kustomize/overlays-example/overlays/production
  destination:
    server: https://kubernetes.default.svc
    namespace: $USER
  syncPolicy:
    automated:
      selfHeal: true
      prune: true
```

```
oc apply -f argocd-kustomize-application-prod.yaml
```

And verify the deployment:

```
oc get pod --namespace $USER --watch
```

Task 5.2.4: Delete the Applications

Delete the applications after you’ve explored the Argo CD Resources and the managed Kubernetes resources.

▼ Hint

```
oc delete application argo-kustomize-$USER argo-kustomize-prod-$USER -n openshift-gitops
```

6. Multiple Applications

When it comes to managing a larger amount of application or bootstrapping whole environments, it’s not very practical to manage the ArgoCD application manually using the CLI Tool `argocd app create` .

In ArgoCD there are two approaches which allow us to manage a set of applications. In the following two sub chapters you’ll learn how the App of Apps and ApplicationSet pattern work.

6.1. Application Sets

With the ApplicationSet ArgoCD adds support for managing ArgoCD Application across a large number of clusters and environments. Plus it adds the capability of managing multitenant Kubernetes clusters.

ApplicationSets are defined through Custom Resource Definition and are processed by the ApplicationSet controller.

The ApplicationSet provides following features

- The ability to use a single Kubernetes manifest to target multiple Kubernetes clusters with Argo CD
- The ability to use a single Kubernetes manifest to deploy multiple applications from one or multiple Git repositories with Argo CD
- Improved support for monorepos: in the context of Argo CD, a monorepo is multiple Argo CD Application resources defined within a single Git repository
- Within multitenant clusters, improves the ability of individual cluster tenants to deploy applications using Argo CD (without needing to involve privileged cluster administrators in enabling the destination clusters/namespaces)

A list of parameters, which come from so called generators , render the ArgoCD Application Template to create a list of Applications.

The ApplicationSet resources work in a similar way as Helm templates do. You can define a set of placeholders `{{placeholder}}` which then are replaced with the actual value during the processing of the ApplicationSet.

Task 6.1.1: Create an ApplicationSet

First delete the Ingress resource under `~/argocd-training-examples/example-app/ingress.yaml`

For better understanding we create our first ApplicationSet. Create a yaml file with the following content under `~/argocd-training-examples/application-set/simple-example/application-set.yaml` and replace the `<username>` placeholder with your actual username.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: application-set-<username>
spec:
  generators:
  - list:
      elements:
        - cluster: dev-cluster
          url: https://kubernetes.default.svc
          env: dev
          traininguser: <username>
        - cluster: prod-cluster
          url: https://kubernetes.default.svc
          env: prod
          traininguser: <username>
  template:
    metadata:
      name: '{{cluster}}-as-example-{{traininguser}}'
    spec:
      project: default
      source:
        repoURL: 'https://gitea.training.cluster.acend.ch/{{traininguser}}/argocd-training-examples.git'
        targetRevision: HEAD
        path: example-app
      destination:
        server: '{{url}}'
        namespace: '{{traininguser}}-{{env}}'
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

Now let’s make sure to apply this to the cluster. But wait, we can either directly apply the yaml or we can

- acend gmbh
create an ArgoCD Application just containing the ApplicationSet. Let’s go the GitOps path:

```
git add .
git commit -m "Add ApplicationSet"
git push origin main
```

And now create the ArgoCD Application, which references the ApplicationSet definition:

Create the following `application.yaml` and apply it:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-appset-0420
  namespace: acend-ttt-gitops
spec:
  project: default
    source:
      repoURL: https://github.com/org-cluster-acend-ch/0420/argocd-training-examples.git
      targetRevision: HEAD
      path: 06-argocd/application-set/simple-example
    destination:
      server: https://kubernetes.default.svc
      namespace: acend-ttt-gitops
    syncPolicy:
      automated:
        prune: true
        selfHeal: true
```

```
oc apply -f application.yaml
```

> **Note**
> Please notice the `dest-namespace`, ApplicationSets needs to be deployed within the `argocd` namespace

You should now be able to see three ArgoCD Applications postfixed with your `<username>` :

- `argo-appset-<username>` The application containing your ApplicationSet.
- `dev-cluster-as-example-<username>` ArgoCD Application for the fist set of key value pairs: `dev`
- `prod-cluster-as-example-<username>` ArgoCD Application for the second set of key value pairs: `prod`

Generators

The generators (`generators` spec in the ApplicationSet yaml) are the building block on how to specify the list of parameters that will be used to generate the Applications. There are several built in generators. Check out the official documentation for more information.

You have even the possibility to combine multiple generators together using the Matrix generator.

Matrix generator

The Matrix generator combines the parameters generated by two child generators, iterating through every combination of each generator’s generated parameters.

- **SCM Provider Generator + Cluster Generator:** Scanning the repositories of a GitHub organization for application resources, and targeting those resources to all available clusters.
- **Git File Generator + List Generator:** Providing a list of applicatations to deploy via configuration files, with optional configuration options, and deploying them to a fixed list of clusters.
- **Git Directory Generator + Cluster Decision Resource Generator:** Locate application resources contained within folders of a Git repository, and deploy them to a list of clusters provided via an external custom resource.
- And so on...

Task 6.1.2: Matrix Example ApplicationSet

- acend gmbh
In this lab section we're going to create an ApplicationSet for an multi-environment.

- Multiple Clusters
- Multiple Applications out of a git directory

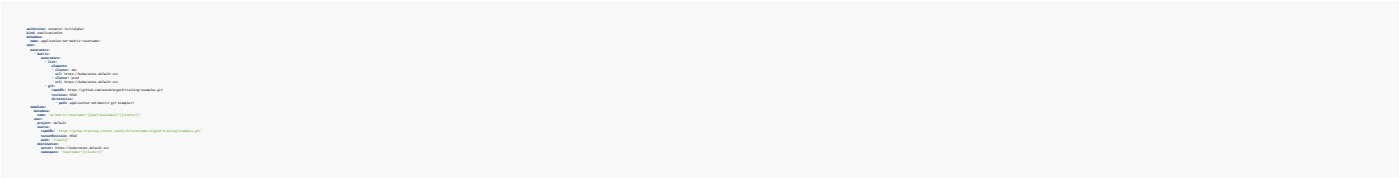
Since we don't have multiple clusters configured in our ArgoCD Cluster, we're going to use the list generator instead of the cluster generator, with two entries `dev` and `prod` both pointing to the local cluster at https://kubernetes.default.svc . The list generator generating values for two clusters `dev` and `prod` looks like this:

The git generator which for the Applications will therefore look like this:

Both generators generate two sets of parameters

cluster	url	path	path.basename	
dev	https://kubernetes.default.svc	application-set/matrix-git-example/application1	application1	
dev	https://kubernetes.default.svc	application-set/matrix-git-example/application2	application2	
prod	https://kubernetes.default.svc	application-set/matrix-git-example/application1	application1	
prod	https://kubernetes.default.svc	application-set/matrix-git-example/application2	application2	

Next let's put everything together, create the Application Set `~/argocd-training-examples/application-set/matrix-example/matrix-example-application-set.yaml` and add the following content:



Make sure to replace all `<username>` occurrences with your username.

Push the changes to your git repository.

And let's create an ArgoCD Application containing the Matrix ApplicationSet with the following command:

Create the following `application-matrix.yaml` and apply it:

- acend gmbh

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-appset-matrix-%USER
  namespace: openshift-gitops
spec:
  project: default
  source:
    repoURL: https://github.com/acend/kubernetes-training-examples.git
    targetRevision: HEAD
    path: applications/matrix-example
  destination:
    server: https://kubernetes.default.svc
    namespace: openshift-gitops
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```
oc apply -f applicationmatrix.yaml
```

Next check the ArgoCD web ui, you should see the 4 generated ArgoCD applications together with the ArgoCD Application, which contains the ApplicationSet itself.

Task 6.1.3: Delete the Application

Delete the two applications (`argo-appset-$USER` and `argo-appset-matrix-$USER`) after you've explored the Argo CD Resources and the managed Kubernetes resources.

▼ Hint

```
oc delete application argo-appset-%USER -n openshift-gitops
oc delete application argo-appset-matrix-%USER -n openshift-gitops
```

7. Projects

Argo CD applications can be linked to a project which provides a logical grouping of applications. The following configurations can be made on a project:

- Source repositories: Git repositories where application manifests are permitted to be retrieved from
- Destination Clusters: Permitted destination Kubernetes or OpenShift clusters
- Destination Namespaces: Destination namespaces where the manifests are permitted to be deployed to
- Permitted resource kinds to be synced (e.g. `ConfigMap`)
- Sync windows: Time windows when an application is permitted to be synced by Argo CD.
- Roles: Roles and policies assigned to the project. The roles are bound to OIDC groups and/or JWT tokens)
- GPG Signature Keys: GnuPG keys that commits must be signed with in order to be allowed to sync them
- Resource Monitoring: Visualization and monitoring of orphaned resources

In summary, a project defines who can deploy what to which destination. This is very useful to keep the isolation between different user groups working on the same Argo CD instance and enables the capability of multi tenancy.

Task 7.1: Create a new empty project

Now we want to create a new empty Argo CD project.

Create a file `appproject.yaml` with the following content and apply it:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: project-%USER
  namespace: openshift-gitops
spec:
  sourceRepos: []
  destinations: []
```

- acend gmbh

```
oc apply -f appproject.yaml
oc get appproject -n openshift-gitops
```

Task 7.2: Define permitted sources and destinations

The next step is to deploy a new application and assign it to the created project `project-<username>` by using the flag `--project`

Update `appproject.yaml` to add the permitted sources and destinations, then re-apply:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: project-%USER%
  namespace: openshift-gitops
spec:
  sourceRepos:
  - '*'
  destinations:
  - server: https://kubernetes.default.svc
    namespace: '*'
```

```
oc apply -f appproject.yaml
```

Now create a file `application.yaml` with the following content and apply it:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: project-app-%USER%
  namespace: openshift-gitops
spec:
  project: project-%USER%
  source:
    repoURL: https://github.com/acend/argocd-training/examples.git
    targetRevision: HEAD
    path: example-app
  destination:
    server: https://kubernetes.default.svc
    namespace: %USER%
```

```
oc apply -f application.yaml
```

Open the Argo CD UI and click **Sync** on the `project-app-$USER` application.

> **Note**
>
> The feature of limiting source repositories and destination clusters/namespaces is a powerful construct of Argo CD as roles and policies can be assigned to projects. With this tool you can enforce a fine grained permission model to control the access of the users to the different clusters and namespaces.

Task 7.3: Deny resources by kind

On a project there is the possibility to restrict the kind of resources that can be synchronized. The restrictions are defined by whitelisting for cluster scoped resources and blacklisted for namespace scoped resources.

Let’s extend our existing project and deny the synchronization of `Services` .

Update `appproject.yaml` to add a namespace resource blacklist, then re-apply:

```
spec:
  namespaceResourceBlacklist:
  - group: ''
    kind: Service
```

```
oc apply -f appproject.yaml
```

- acend gmbh
Open the Argo CD UI and click **Sync** on `project-app-$USER`. The sync will fail because `Service` is now blocked.

To allow `Service` again, remove the `namespaceResourceBlacklist` entry from `appproject.yaml` and re-apply:

```
oc apply -f appproject.yaml
```

Sync the application again in the UI.

Task 7.4: Cleanup

Delete the resources created in this chapter by running the following commands:

```
oc delete application argocd-$USER -n openshift-gitops
oc delete appproject project-$USER -n openshift-gitops
```