

- 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
- Argo CD CLI
- kubectl

## Argo CD Command line tool

Follow the instructions on [this](#) page to install the ArgoCD tool on your local computer.

## kubectl

Follow the instructions on [this](#) page to install the kubectl on your local computer.

## Labs

[Argo CD](#) is a part of the [Argo Project](#) and affiliated under the [Cloud Native Computing Foundation \(CNCF\)](#) . The project is just under three years old, completely open source, and primarily implemented in Go.

As the name suggests, Argo CD takes care of the continuous delivery aspect of CI/CD. Continuous integration is handled by a CI tool such as GitLab CI/CD, Jenkins, Tekton or GitHub Actions. The core of Argo CD consists of a Kubernetes controller, which continuously compares the live-state with the desired-state. The live-state is tapped from the Kubernetes API, and the desired-state is persisted in the form of manifests in YAML or JSON in a Git repository. Argo CD helps to point out deviations of the states, to display the deviations or to autonomously restore the desired state.

Argo CD is deployed and operated on a Kubernetes-based container platform. It is possible to connect multiple Kubernetes and OpenShift clusters to one Argo CD instance.

[Argo CD](#) is a declarative, GitOps continuous delivery tool for Kubernetes.

[GitOps](#) is a way to do Kubernetes cluster management and application delivery. It works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what's running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case. With Git at the center of your delivery pipelines, developers use familiar tools to make pull requests to accelerate and simplify both application deployments and operations tasks to Kubernetes.

Managing Kubernetes resources using a GitOps approach brings the following benefits:

- The definition of the manifests is done in a declarative way and a tool ensures the comparison between the desired and live manifests. The differences between the desired configurations and the actually

- acend gmbh

- applied manifests can be easily seen at any time.
- Rollbacks to older versions are easily possible with git revert or via the used GitOps tool (provided that the application also supports this).
- Manual adjustments directly on the container platform are immediately visible and can be automatically overwritten (self-healing).
- The git commit history is also a detailed audit log
- The developers describe the infrastructure in already known formats and tools like yaml and git.

Argo CD follows the GitOps pattern of using Git repositories as the source of truth for defining the desired application state.

### 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!

Kubernetes manifests can be specified in several ways:

- [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

Argo CD automates the deployment of the desired application states in the specified target environments. Application deployments can track updates to branches, tags, or pinned to a specific version of manifests at a Git commit. See [tracking strategies](#) for additional details about the different tracking strategies available.

For a quick 10 minute overview of Argo CD, check out the demo presented to the Sig Apps community meeting:

# Argo CD Architecture

Argo CD's core components are the API Server, the Repository Server and the Application Controller

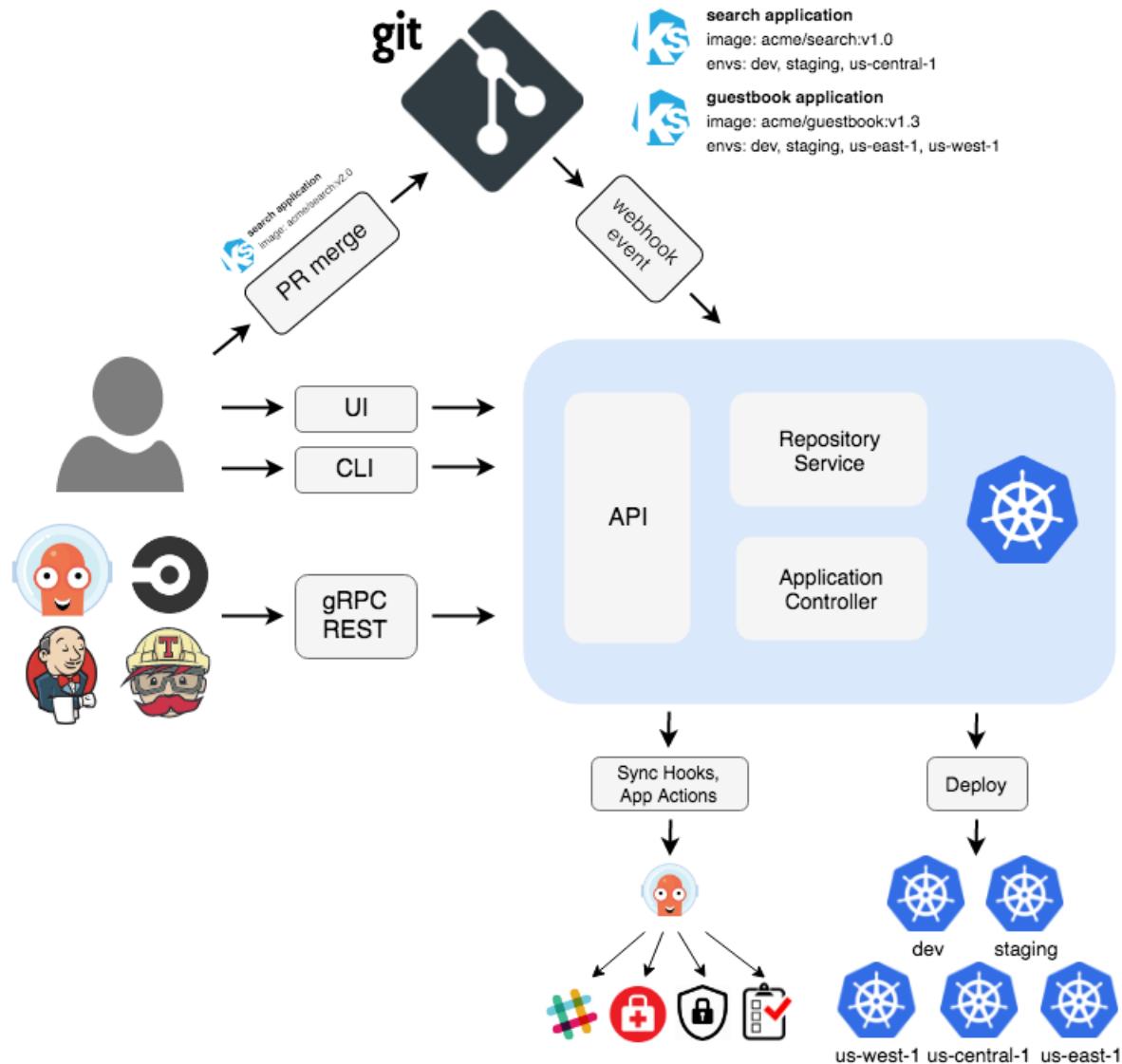


Image and component description source: <https://argoproj.github.io/argo-cd/>

## 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)

- acend gmbh
  - 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:

- 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 tool` or 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.

- acend gmbh

## Note

The URL and Credentials to the Web IDE will be 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 `kubectl` version by executing the following command:

```
...
```

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.

```
...
```

## Task 1.1.2: Login to ArgoCD

You can access Argo CD via Web UI (Credentials are provided by your teacher) or using the CLI. The Argo CD CLI Tool is already installed on the web IDE.

```
...
```

## Task 1.2: Argo CD CLI

The [Argo CD CLI](#) is a powerful tool to manage Argo CD and different applications. It's a self contained binary written in Go and available for Linux, Mac OS and Windows. Thanks to the fact that the CLI is implemented in Go, it can be easily integrated into scripts and build servers for automation purposes.



- acend gmbh

## 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>
- Argo CD Server: <https://argocd.training.cluster.acend.ch>
- Kubernetes Cluster

### Task 2.1: Login to the Gitea and Clone the Repo

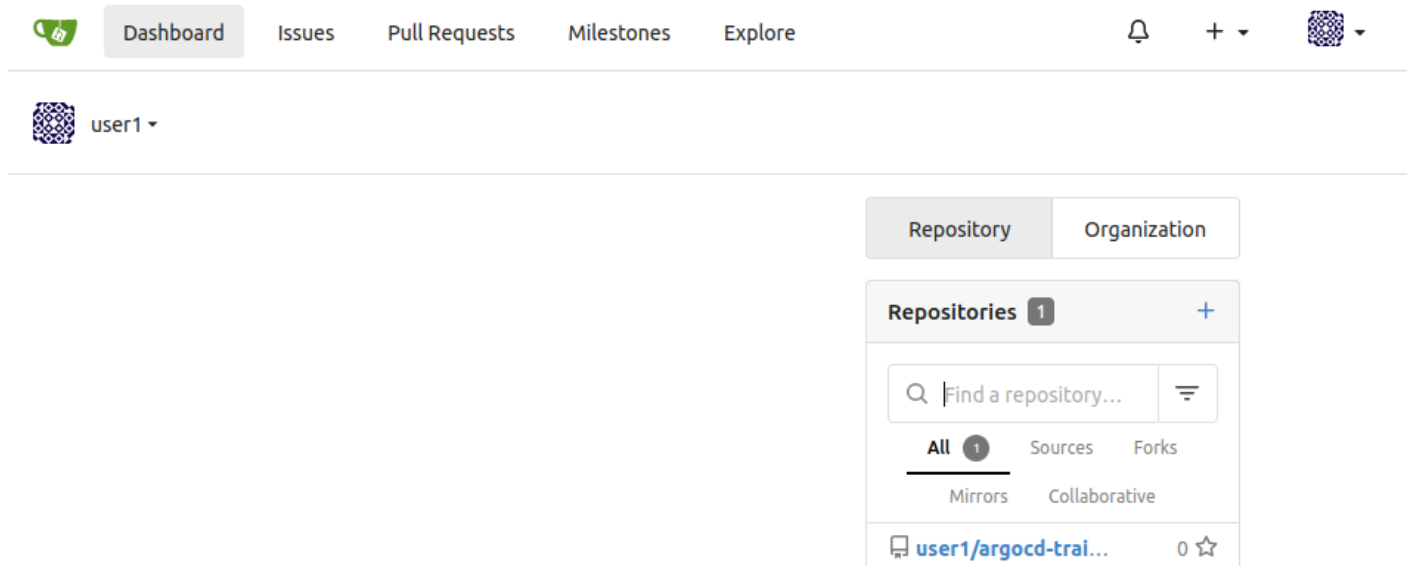
For this training we're using a Git Server deployed under <https://gitea.training.cluster.acend.ch> . We also forked the Argo CD Example Repo for your `<username>` .

Open your webbrowser and navigate to <https://gitea.training.cluster.acend.ch> . Login with the training credentials provided by the trainer (Login Button is in the upper right corner).

#### Note

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*.

The Git Repository is available under your Repositories



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

- acend gmbh

The screenshot shows a Git repository interface. At the top, there are navigation links: Dashboard, Issues, Pull Requests, Milestones, and Explore. On the right, there are icons for notifications, a plus sign, and a profile picture. Below the navigation, the repository name 'user1 / argocd-training-examples' is displayed, along with statistics: 1 Watch, 0 Stars, and 0 Forks. A menu bar includes Code, Issues, Pull Requests, Projects, Releases, Wiki, Activity, and Settings. The repository has 'No Description' and 'Manage Topics'. Below this, there are statistics: 27 Commits, 1 Branch, 0 Tags, and 58 KIB. A 'Branch: master' dropdown is visible, along with buttons for 'New Pull Request', 'New File', and 'Upload File'. The repository URL is 'http://gitea.labapp.acend.ch/use'. A list of recent commits is shown, including one by Benjamin Affolter with commit hash 89fd53a599, titled 'Revert "Set name env var"', and two folders: 'app-of-apps' (Update app of apps to specify namespace and name) and 'complex-application' (Add complex application).

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>` .

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 chosen to use your own Git Server.

Change the working directory to the cloned git repository:

- acend gmbh

```
git config --global user.name "acend gmbh"
git config --global user.email "acend@gmbh.com"
```

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

```
git config --global user.name "acend gmbh"
git config --global user.email "acend@gmbh.com"
```

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.

```
git config --global credential.helper store
```

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

```
git config --list
```

## 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.

To deploy the resources using the Argo CD CLI use the following command:

```
argocd app create --name argo-username --namespace argo --repo https://github.com:acend:acend-gmbh --path . --sync-policy auto
```

Expected output: application 'argo-username' created

### Note

We don't need to provide Git credentials because the repository is readable for non-authenticated users as well

### Note

If you want to deploy it in a different namespace, make sure the namespaces exists before syncing the app

Once the application is created, you can view its status:

```
argocd app get argo-username
```

```
NAME: argo-username
NAMESPACE: argo
STATUS: Synced
LAST SYNC: 2023-01-10 10:10:10
REPO: https://github.com:acend:acend-gmbh
PATH: .
SYNC POLICY: auto
```



- acend gmbh

The screenshot shows the Argo CD interface for an application named 'argo-hannelore42'. The top navigation bar includes 'APPLICATION DETAILS' and a 'Log out' button. Below the navigation are several action buttons: 'APP DETAILS', 'APP DIFF', 'SYNC', 'SYNC STATUS', 'HISTORY AND ROLLBACK', 'DELETE', and 'REFRESH'. The main content area shows the application's status as 'Missing' and 'OutOfSync'. A diagram illustrates the application's structure: 'argo-hannelore42' (Application) is connected to 'example-php-docker-helloworld' (Service) and 'example-php-docker-helloworld' (Deployment). The Service and Deployment are both marked as 'OutOfSync'.

The screenshot shows the Argo CD interface for the same application 'argo-hannelore42', now in a 'Synced' state. The status indicators have changed to 'Healthy', 'Synced', and 'Sync OK'. The diagram shows the application 'argo-hannelore42' (Application) is connected to 'example-php-docker-helloworld' (Service), 'example-php-docker-helloworld' (Deployment), and 'example-php-docker-hellowor...' (Pod). The Service and Deployment are marked as 'Synced', and the Pod is marked as 'Sync OK'. A button 'clear filters to show 3 additional resources' is visible.

## 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.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-php-docker-helloworld
spec:
  replicas: 2
  selector:
    matchLabels:
      app: example-php-docker-helloworld
  template:
    metadata:
      labels:
        app: example-php-docker-helloworld
    spec:
      containers:
        - name: example-php-docker-helloworld
          image: example-php-docker-helloworld
```

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.

```
git commit -m "Increase replicas to 2"
git push
```

After a successful push you should see the following output

- acend gmbh

```
argo cd --refresh
```

Check the state of the resources by cli:

```
argo cd --refresh
```

The parameter `--refresh` triggers an update against the Git repository. 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.

You will see that the Deployment is now `OutOfSync`:

```
argo cd --refresh
```

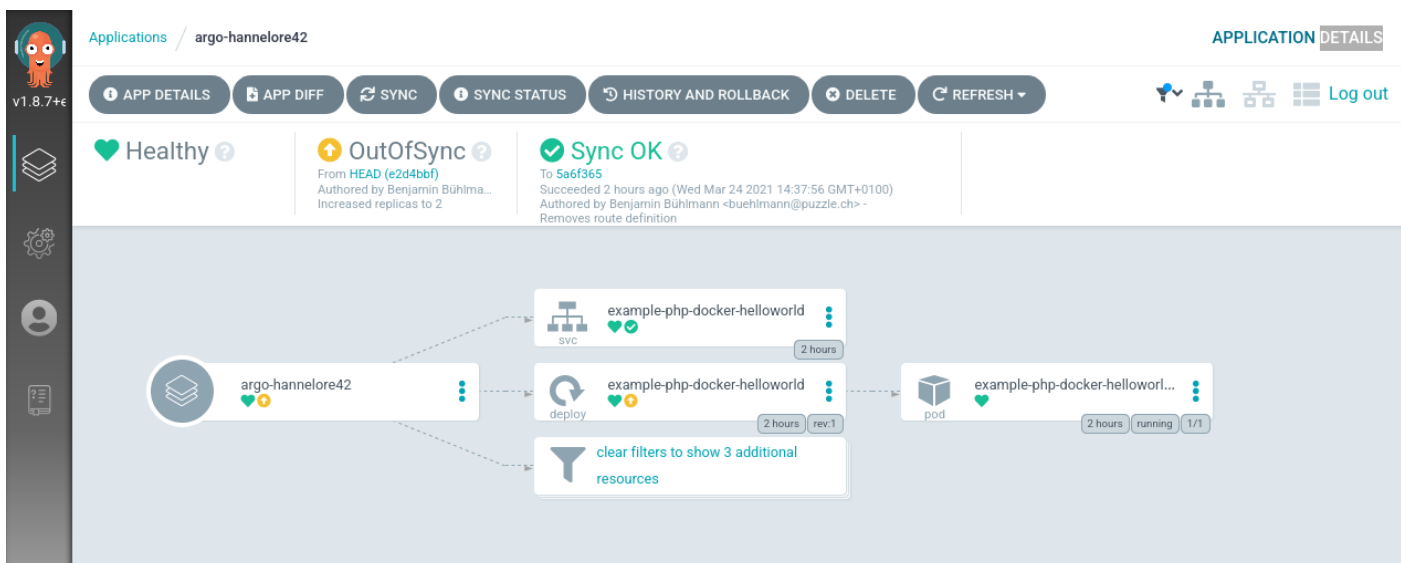
When an application is `OutOfSync` then your deployed 'live state' is no longer the same as the 'target state' which is represented by the resource manifests in the Git repository. You can inspect the differences between live and target state by cli:

```
argo cd --refresh
```

which should give you an output similar to:

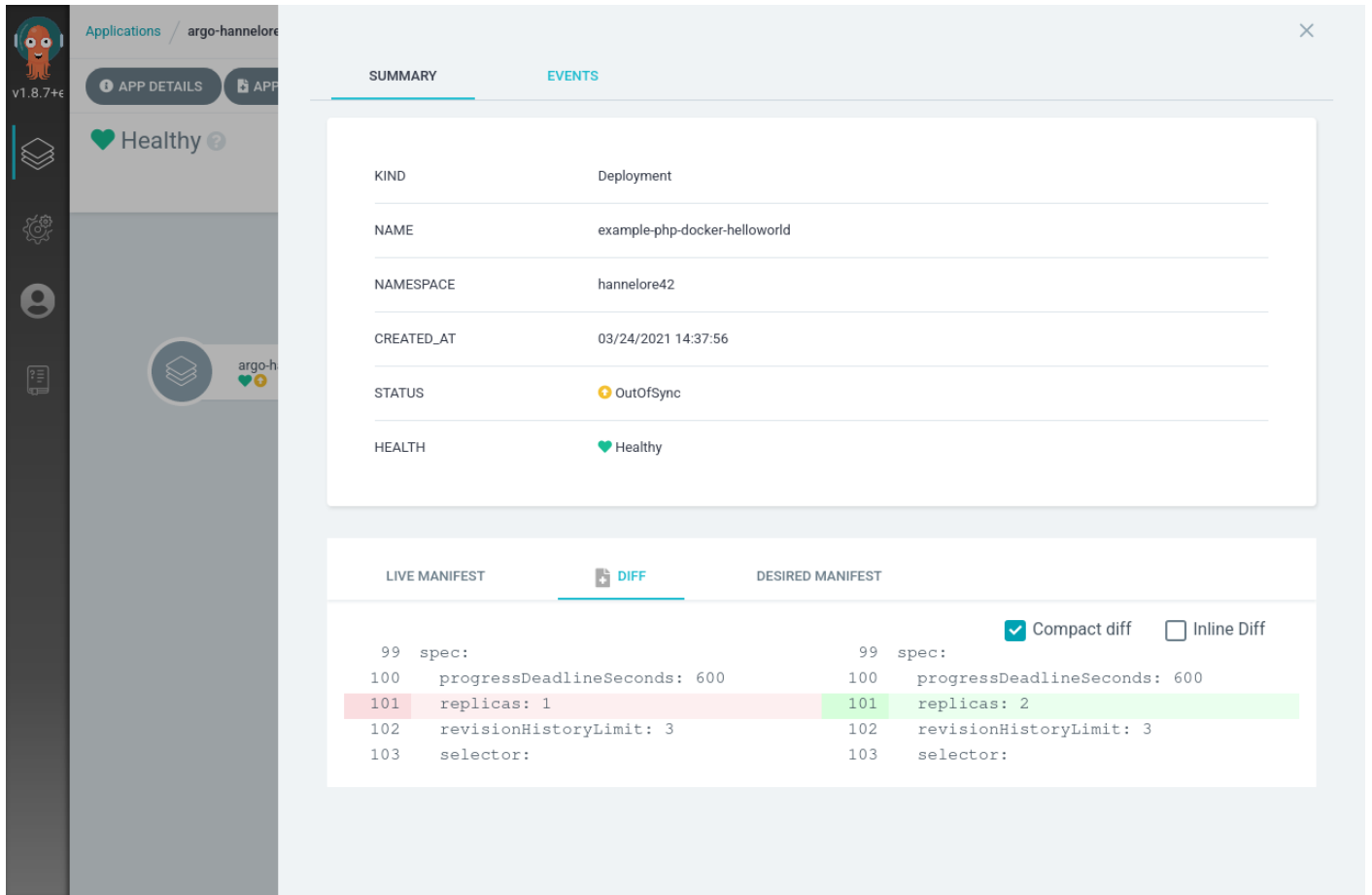
```
argo cd --refresh
```

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



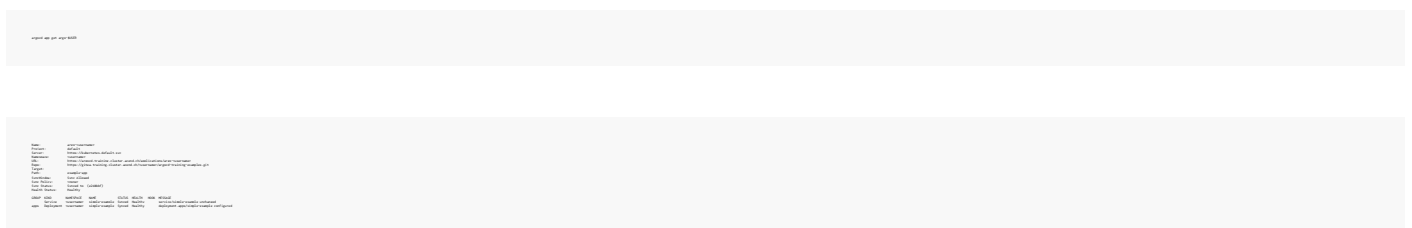
- acend gmbh

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:



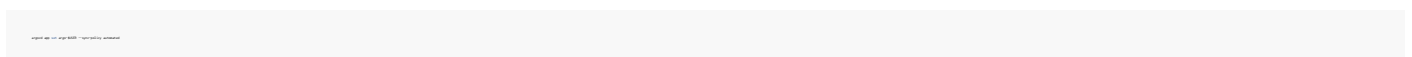
Now click `Sync` on the top left and let the magic happen ;) The application will be scaled up to 2 replicas and the resources are in Sync again.

Double-check the status by cli



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.

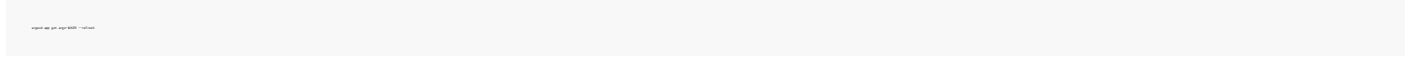
To configure automatic sync run (or use the UI):



- acend gmbh

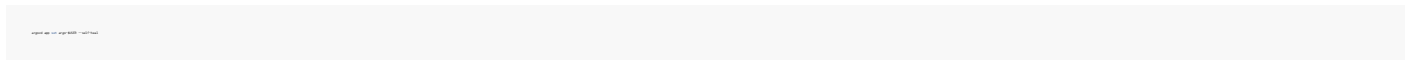
From now on Argo CD will automatically apply all resources to Kubernetes every time you commit to the Git repository.

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 or by cli:

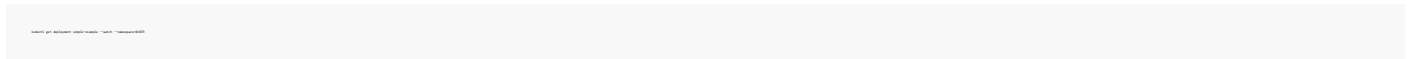


## Task 2.4: Automatic Self-Healing

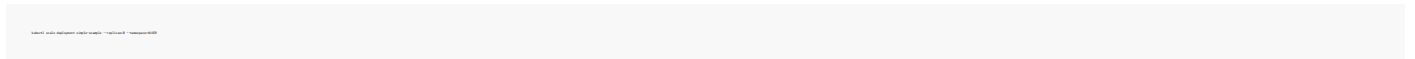
By default, changes made to the live cluster will not trigger automatic sync. To enable automatic sync when the live cluster's state deviates from the state defined in Git, run:



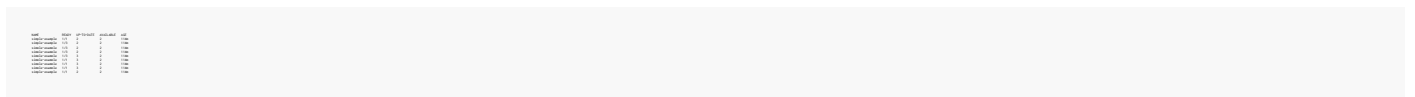
Watch the deployment `simple-example` in a separate terminal:



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



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

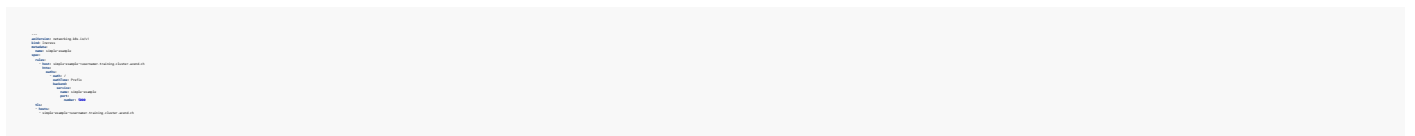


This is a great way to enforce a strict GitOps principle. Changes which are manually made on deployed resource manifests are reverted immediately back to the desired state by the ArgoCD controller.

## Task 2.5: Expose Application

This is an optional task.

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



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

```
git commit -m "update"
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-  
<username>.training.cluster.acend.ch
```

The result should look similar to this:

```
simple-example-  
<username>.training.cluster.acend.ch
```

## 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 `ingress.yaml` from Git repository and push the changes:

```
git rm service.yaml ingress.yaml
git commit -m "delete"
git push
```

Check the status of the application with

```
argocd app get simple-example-  
<username>.training.cluster.acend.ch
```

You will see that even with auto-sync and self-healing enabled the status is still `OutOfSync`

```
NAME: simple-example-  
<username>.training.cluster.acend.ch  
STATUS: OutOfSync  
LAST_SYNCED: 2023-01-10 10:10:10  
LAST_OPERATION: Sync  
HEALTH: Syncing  
RECONCILING: false  
SELF_HEALING: true  
AUTO_SYNC: true  
PRUNING_DISABLED: false
```

Now enable the auto pruning explicitly:

```
argocd app edit simple-example-  
<username>.training.cluster.acend.ch
```

Recheck the status again

```
argocd app get simple-example-  
<username>.training.cluster.acend.ch
```



- acend gmbh  
You will see the following error

```
-----
```

Argo CD can't any longer access the protected repository without providing credentials for authentication. Next assign credentials to used Git repository. You have to provide the Gitea password interactively.

```
-----
```

### Note

You can provide the password through the cli by using the flag `-password`.

Now the sync should work. Argo CD use the configured credentials to authenticate against your repository in Gitea.

```
-----
```

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.

The following command will create a credential which matches all git repositories for your username (e.g. `https://<username>@gitea.training.cluster.acend.ch/<username>`)

```
-----
```

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.

## Task 2.9: Delete the Application

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

```
-----
```

Hit `y` to confirm the deletion and this will delete the `Application` manifests of ArgoCD and all created resources by this application. In our case the `Application`, `Deployment` and `Service` will be deleted. With the flag `--cascade=false` only the ArgoCD `Application` will be deleted and the created resources `Deployment` and `Service` remain untouched.

- acend gmbh

## 3. Resource Hooks

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

### Resource Hooks

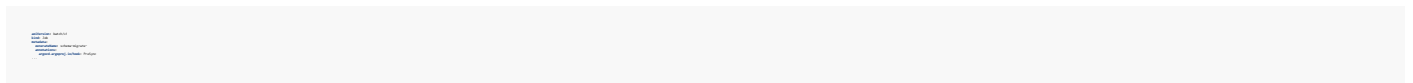
Hooks allow to run scripts before, during and after the Argo CD **sync** operation is running. They give you more control over the sync process. They can also run when the sync operation fails for example. The concept is very similar to the concept of [Helm Hooks](#) . Argo CD supports many Helm hooks 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:



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

#### Note

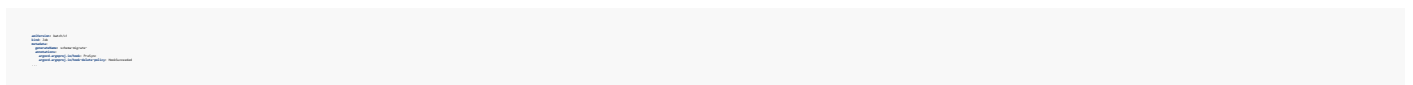
Named hooks (i.e. ones with `/metadata/name`) will only be created once. If you want a hook to be re-created each time either use `BeforeHookCreation` policy or `/metadata/generateName`.

#### Note

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

### 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.

- acend gmbh

## 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

```
argo create --name argo-hook-$USER --service --deployment --pre-sync-hook --post-sync-hook
```

Sync the application

▼ Hint

```
argo sync --name argo-hook-$USER
```

And verify the deployment:

```
kubectl get deployment --namespace argo-hook-$USER
```

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.

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

```
apiVersion: argoproj.io/v1alpha1
kind: Hook
name: post-sync-job
type: PostSync
command:
- curl
- https://acend.ch
```

## Task 3.3: Delete the Application

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

▼ Hint

```
argo delete --name argo-hook-$USER
```

- acend gmbh

## 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

- acend gmbh

Sync the application:

▼ Hint

And verify the deployment:

## Task 4.2: Delete the Application

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

▼ Hint

## 5. Tools

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!

- acend gmbh

## 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:

```
helm install myapp mychart --values values.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.

```
helm install myapp mychart --set replicaCount=2 --namespace myns
```

#### 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!

- acend gmbh

## 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:

```
helm install --release-name <release-name> <chart-name> <chart-version>
```

### 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.

```
argocd app create simple-example --helm-chart <chart-name> --helm-version <chart-version>
```

Sync the application

#### ▼ Hint

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

or execute the following command:

```
argocd app sync simple-example
```

And verify the deployment:

```
argocd app get simple-example
```

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

#### ▼ Hint

- acend gmbh

## Task 5.1.2: Scale the deployment to 2 replicas

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

```
helm upgrade --install simple-example simple-example --set helm.sync-policy=automated
```

### Warning

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

```
helm upgrade --install simple-example simple-example --set helm.sync-policy=automated --set ingress.enabled=true
```

Commit and push the changes to your repository.

▼ Hint

```
helm upgrade --install simple-example simple-example --set helm.sync-policy=automated --set ingress.enabled=true --set ingress.host=production.example.com
```

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

```
helm upgrade --install simple-example simple-example --set helm.sync-policy=automated --set ingress.enabled=true --set ingress.host=production.example.com --set helm.release-name=production
```

- acend gmbh  
Commit and push the changes to your repository.

▼ Hint

```
git commit -m "Production stage"
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

```
cat <<EOF | kubectl apply -f -
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-helm-prod-$USER
spec:
  syncPolicy:
    automated: true
    prune: true
  source:
    repoURL: https://github.com/acend/argo-cd
    targetRevision: HEAD
  destination:
    namespace: argo-cd
    server: https://kubernetes.default.svc
EOF
```

And verify the deployment:

```
argo cd --name argo-helm-prod-$USER --server https://kubernetes.default.svc
```

Tell the Argo CD app to use the `values-production.yaml` values file

▼ Hint

```
argo cd --name argo-helm-prod-$USER --server https://kubernetes.default.svc --update
```

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

```
argo delete --name argo-helm-prod-$USER --server https://kubernetes.default.svc
```

## 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`.

- acend gmbh

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
- `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:

```
-----
```

## 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.

```
-----
```

Sync the application

### ▼ Hint

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

or execute the following command:

```
-----
```

And verify the deployment:

- acend gmbh

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

▼ Hint

## Task 5.2.2: Set a configuration parameter

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

And take a look at the application in the web UI or using the command line tool

▼ Hint

### 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

And verify the deployment:

## Task 5.2.4: Delete the Applications

- acend gmbh

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

▼ Hint



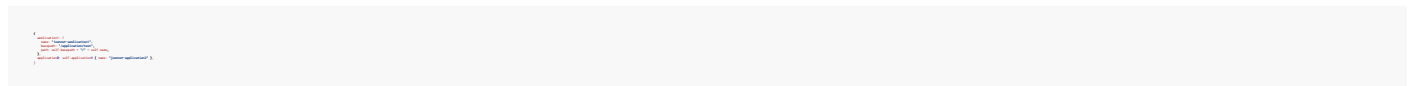
## 5.3. Jsonnet (Optional)

This lab explains how to use [jsonnet](#) as manifest format together with Argo CD.

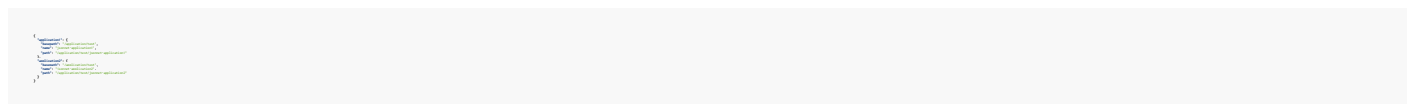
### Jsonnet

[Jsonnet](#) is a templating language which adds the possibility to programmatically work with the underlying data. It basically is a simple extension of [JSON](#).

Let's have a look at an example first. The following jsonnet file



will render into:



Among many other features, jsonnet can help to reduce duplications.

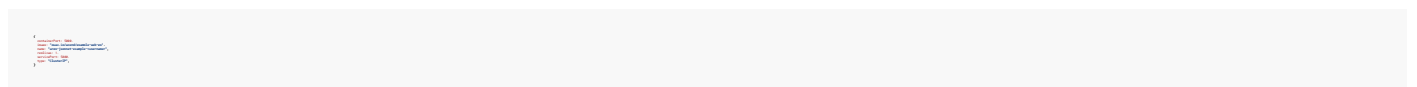
### Further Docs

Read more about the jsonnet integration in the [official documentation](#)

## Task 5.3.1: Deploy the simple-example with jsonnet

Let's first explore the files in your local repository under `jsonnet`.

Similar to `helm`, `jsonnet` allows us to extract parameters into a separate file. In our example we extracted all values to the `params.libsonnet` file:



And the actual template file, containing the kubernetes service and deployment definitions, `simple-application.jsonnet` file:



- acend gmbh

```
1 #!/bin/sh
2 # This script is used to create a new repository
3 # and push the changes to your repository.
4 # It is a simple script that uses git to
5 # create a new repository and push the
6 # changes to your repository.
```

Commit and push the changes to your repository.

▼ Hint

```
git commit -m "Initial commit"
git push
```

And verify the result in the ArgoCD Ui or by using the following command, this might take a little while to happen, depending on how many trainees are currently working on the labs. Hint: hit refresh to speed up the process.

```
git push
```

## Task 5.3.3: Delete the Applications

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

▼ Hint

```
git push
```