Folie 2 Dieter / Uwe

- **Dieter**

- **Uwe**

- **Our passion is to develop high quality SW for our customers. We work toghether within agile teams. Due to the composition of our teams we find a balance between experience and innovation“**
- **One of the different principles within SW-development which guids us to develop high quality cloud applications is the 12 Factor App manifesto. We are glad to be allowed to talk about this manifesto with you today**

Folie 3 „What is the 12-Factor App manifesto talking about?“ (Dieter)

- **The manifesto was initially written by Adam Wiggins, who is the cofounder of the Heroku-Cloud-Platform**
- **He and the other contributers where directly involved in the development of the apps for the platform.**
- **The experience they gained during the development, the operation and scaling of these applications, was transfered to this manifesto.**
- **The manifesto is a triangulation on ideal practices for app development.**
- **It is agnostic to any programming language; it is rather a methodology for architectural principles like (Punkte von Folie).**
- **The contributers of this manifesto found 12 factors which supports these principles**

Today we will show you an example application which was designed with focus on these factors. But before we go into details of this application we‘d like to give you a rough overview of the 12 factors.

Folie 4 „Vorstellung der 12 Faktoren“ (abwechselnd)

We start with the factor ‚Codebase‘ which is the first factor in the manifesto. But actually the order of the factors doesn‘t matter at all.

Start your engine Uwe :-)

Factor I: Codebase (Uwe)

- **This factor basically says, that the code of one app is tracked with a version control system and there is a 1-1 relation between the app and the codebase.**

- **Each app has one codebase but may have many deploys.**

- **If you have multiple codebases, you talk about a distributed system but not about a app. Each of the apps within a distributed system would be developed on the base of the 12 factors and shared functionality would be managed by the dependency management.**

Dieter:

Which leads to the next factor…

Factor II: Dependencies

- **A 12 factor app never relies on implicit existence of system-wide dependencies.**

- **It declares all dependencies with a dependency declaration manifest and it uses dependeny isolation to ensure that no implicit dependency leaks in**

Uwe: Factor III: Config

To be deployed and executed on different stages a 12 factor app needs to be independent from environment specific configurations.

Everything which is likely to vary between deploys must be able to change without building the application. That‘s why a 12 factor app stores such configs in the environment variables which are independently managed for each stage.

The advantage of env-vars is that such a configuration can‘t be checked in accidentally and they are a language- and OS-agnostic standard.

Dieter: Factor IV: Backing services

Now we are independent about environment specific config but most likely a app will interact with services like a DB, a messaging system or other apps.

- **Any service a app consumes over the network as part of its normal operation is called a ‚Backing service‘**
- **A Backing service should be possible to be swapped out without changing the applications code**
- **For example, it should be possible to swapp out a local MySQL-Database with a Amazon RDS managed MySQL-Database without codechange**

Uwe: Factor V: Build, release, run

If we have developed our application, it must be transformed into a deploy. This is achieved through the three stages build, release und run, which must be strictly separated.

- **The build stage creates an executable bundle based on a specified commit by fetching dependencies and compiling binaries.**
- **The release stage takes the produced build and combines it with the current config of a specific environment and publishes it with a unique release id like a timemstamp. It is important to note, that once a release is created, it cannot be mutated anymore. Any change to the app or the config must create a new release.**
- **The run stage launches the published release in the releated environment.**

Dieter: Factor IX Disposability

Not only because a release of a 12 factor app is described as the combination of the apps version plus the configuration of a stage, it is neccassary to be able to stop and start the app at a moments notice.

So a process of a app can be disposed in any time to maximize robustness.

The factor ‚Disposability‘ leads to a crash-only design because it says that a application should always be able to graceful start even if it died unexpectedly.

Moreover the short startup time helps to improve the scalability of the application.

Uwe: Factor VI: Processes

Within the run stage the application is executed as one or more statless processes.

This means, that the application never assumes a specific local state for any request

For that reason required data, like session data must be stored in a backing service.

Therefore you can scale your application with just starting more processes which leads us to other factors like concurrency and disposability.

Dieter: Factor VIII: Concurrency

Threading within one process is even allowed within a 12 factor app.

Hower vertical scale is limited to the size of one VM, a 12 factor app scales by starting a process multiple times on one or more machines.

If you combine this factor with other paradigmns, like stream processing or actor model, you can completely avoid threading and avoid typical concurrency problems.

Uwe: Factor VII: Port binding

A 12 factor app is completely self-contained and does not rely on runtime injection of a webserver.

Because of that and the earlier mentioned factors, a routing layer might be neccessary to route requests from a public-facing hostname to the port-bound processes.

Dieter: Factor X: Dev/prod parity

To minimize the risk of unexpected behaviour of the app in production stage, it is essential to keep the different stages as similar as possible.

This means as a developer you use in dev the same backing services as in production.

Moreover any codechange should be deployd as early as possible, which leads to the need of designing a 12 factor app for continous deployment.

Uwe: Factor XI: Logs

Now the app is developed, released and deployd but there is still the need to keep track of the health status of it.

Typically this is done by monitoring logs.

Logs are a aggregated collection of time ordered events of a whole distributed system.

Any app of the system does not care about storing or routing its events. It rather writes the events to STDOUT and another system must take care of the collection and its representation.

A well known implementation of this concept is the ELK stack.

Dieter: Factor XII: Admin processes

If it is really neccassary to run one-off tasks for a app, this task must be deployed together with the app.

Such admin tasks must be stored in the same codebase and use the same configuration as the app. This leads to the fact, that a admin task ist released together with the app.