basysKom

Continuous Deployment nach Microsoft Azure

Fallstricke und Herausforderungen – Ein Erfahrungsbericht

basysKom

Agenda

Über basysKom

Motivation und Projektszenario

Release-Szenario

Continuous Integration Systeme

Konfiguration der Cloud-Dienste

Arten der App-Bereitstellung

Zusammenfassung und Fazit

basysKom

Jeremias Bosch Dipl. Inf (FH)

- Seit 2009 bei der basysKom GmbH
- Senior Developing Project Manager (Technischer Projektleiter)
- IoT, Azure, Cloud, Kubernetes,…
- C++, Qt, Qt Quick

basysKom

Facts

Gegründet 2004
von Prof. Dr.-Ing. Eva Brucherseifer

30 Angestellte
davon 25 Entwickler

2 Standorte, 1 Team
Hauptstandort in Darmstadt, Büro in Nürnberg

- Wir setzen auf Open Source-Innovation, modernstes Engineering und ein agiles Projektmanagement
- Die Schwerpunkte unserer Arbeit sind die Realisierung von Kundenapplikationen in C++, Qt, Web- und Cloud-basierten Technologien
- Wir setzen seit Jahren auf eine vertrauensvolle und intensive Zusammenarbeit mit unseren Kunden und übernehmen volle Verantwortung für Ihre Aufgabenstellungen
- Seit 2019 sind wir ein Teil der UX Gruppe

basysKom

Motivation: Warum Continuous Deployment / Delivery

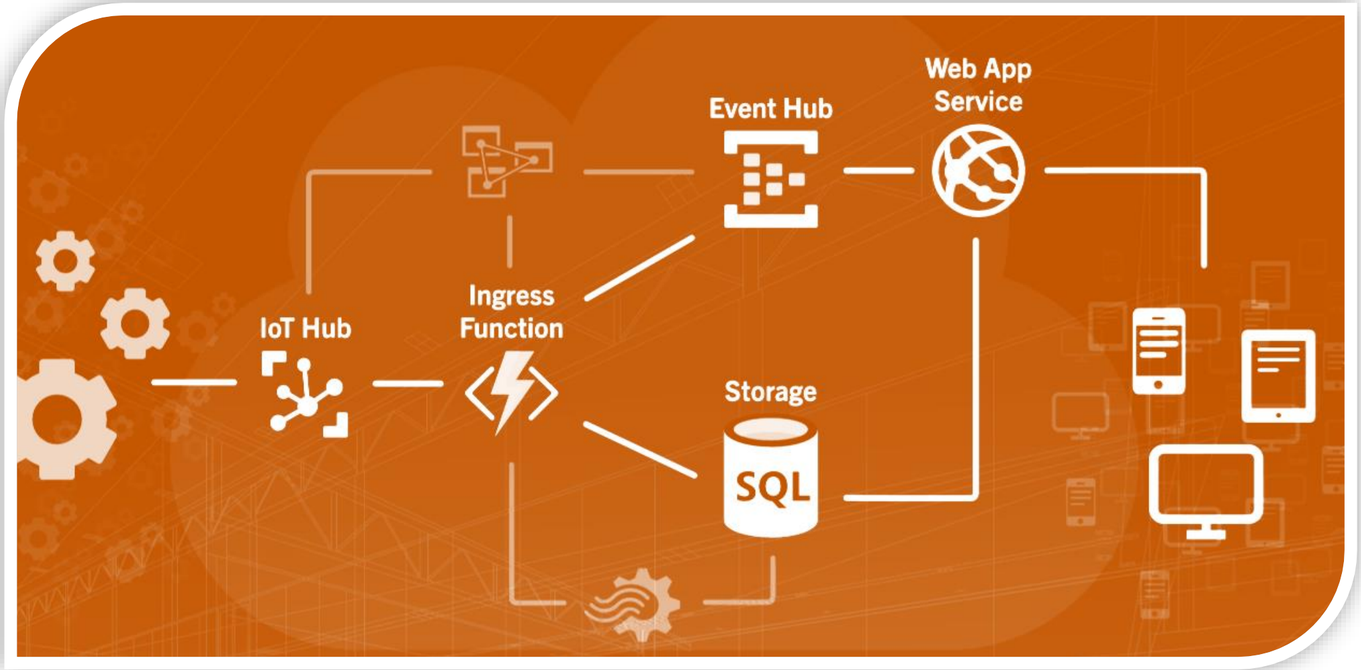
- **Das generelle Ziel eines Continuous Deployments ist es die Bereitstellung von Software:**
  - Schmerzlos zu gestallten
  - Mit geringem Risiko auszuführen
  - Zu Jederzeit (auf Wunsch) durchführen zu können

- **Dabei wünscht man sich:**
  - Keine Ausfallzeiten der Systeme während oder aufgrund des deployments
  - Schnelleren Marktzugang mit neuen Features
  - Geringere Kosten & Risiken bei der Einführung neuer Features

basysKom

Projektszenario

- **Einsatz von Microsoft Azure Services**
- **Generelles Szenario**
  - Geräte befinden sich außerhalb der Cloud
  - Geräte authentifizieren sich gegen die Cloud
  - Geräte senden Daten in die Cloud
  - Daten werden in der Cloud abgelegt und aufbereitet
  - Daten werden über die Cloud bereitgestellt/dargestellt



- Doch wie wird die beschriebene Lösung kontinuierlich bereitgestellt?

basysKom

Verwendete Cloud-Dienste

- Azure IoT Hub
- Azure Device Provisioning Service (DPS)
- Azure Servicebus
- **Azure Function Apps**
- **Azure Web Apps**
- Azure SQL Server/Datenbank (MS SQL)
- Azure ADB2C
- Azure Container Registry (ACR)
- Azure Key Vault
- Azure Storage Account
- Azure Linux App Service Plan
- Azure Application Insights
- SendGrid

basysKom

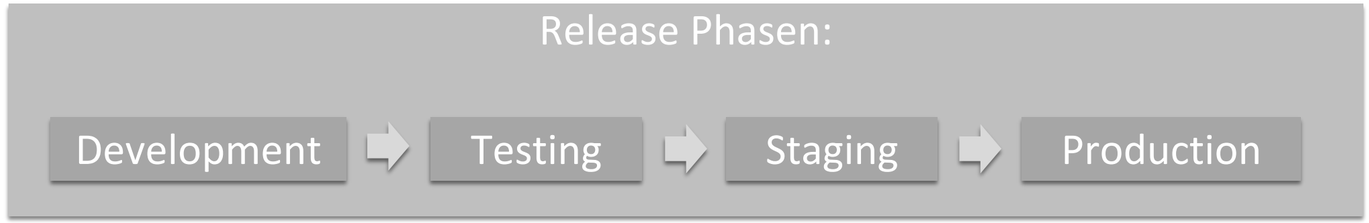
Erfahrungen beim Releasemanagement

- **Die Konzeption des Releasemanagement hat großen Einfluss auf die spätere Gestaltung des Deployment Prozess**
- **Wir empfehlen die Umsetzung von verschiedenen Techniken und Konzepten**
  - Git-Flow Konzept und die verschiedene Phasen bis zum Release (Development, Testing, Staging, Production)
  - Bereitstellung über Slots (Zero Downtime Bereitstellung)

basyskom

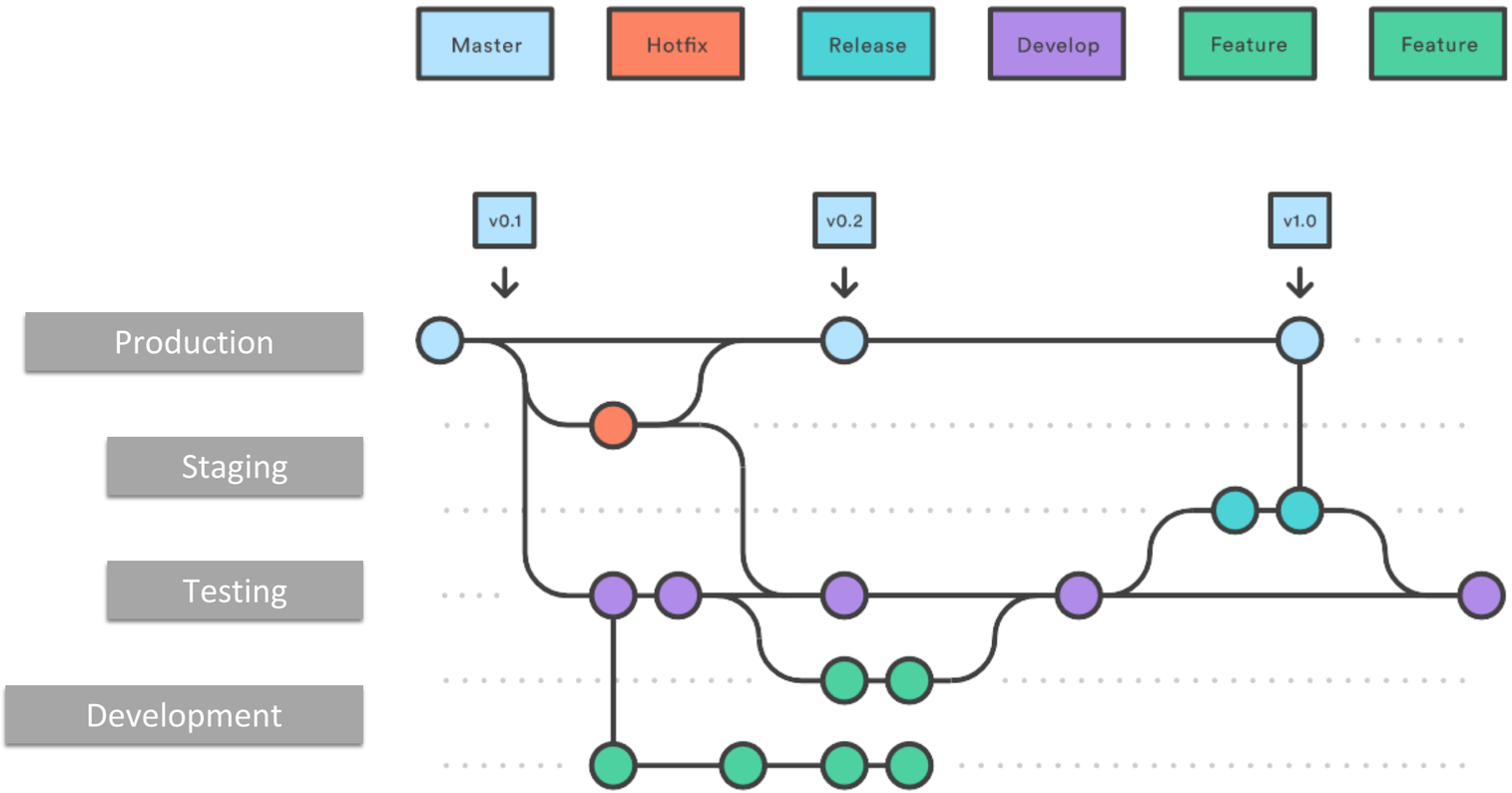
Aufbau der Deploymentumgebung

- Development
- Testing
- Staging
- Production



basysKom

Git-Flow Konzept und die verschiedene Phasen bis zum Release



Quelle: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

basysKom

Auswahl des Countinuous Integration System

- **Azure DevOps**
  - Volle Integration in das Azure Ökosystem
  - Continuous Deployment Beschreibung im Azure DevOps Portal
- **Jenkins**
  - Verschiedene Plugins für das Azure Deployment vorhanden
  - Continuous Deployment Beschreibung als Datei (Jenkinsfile)

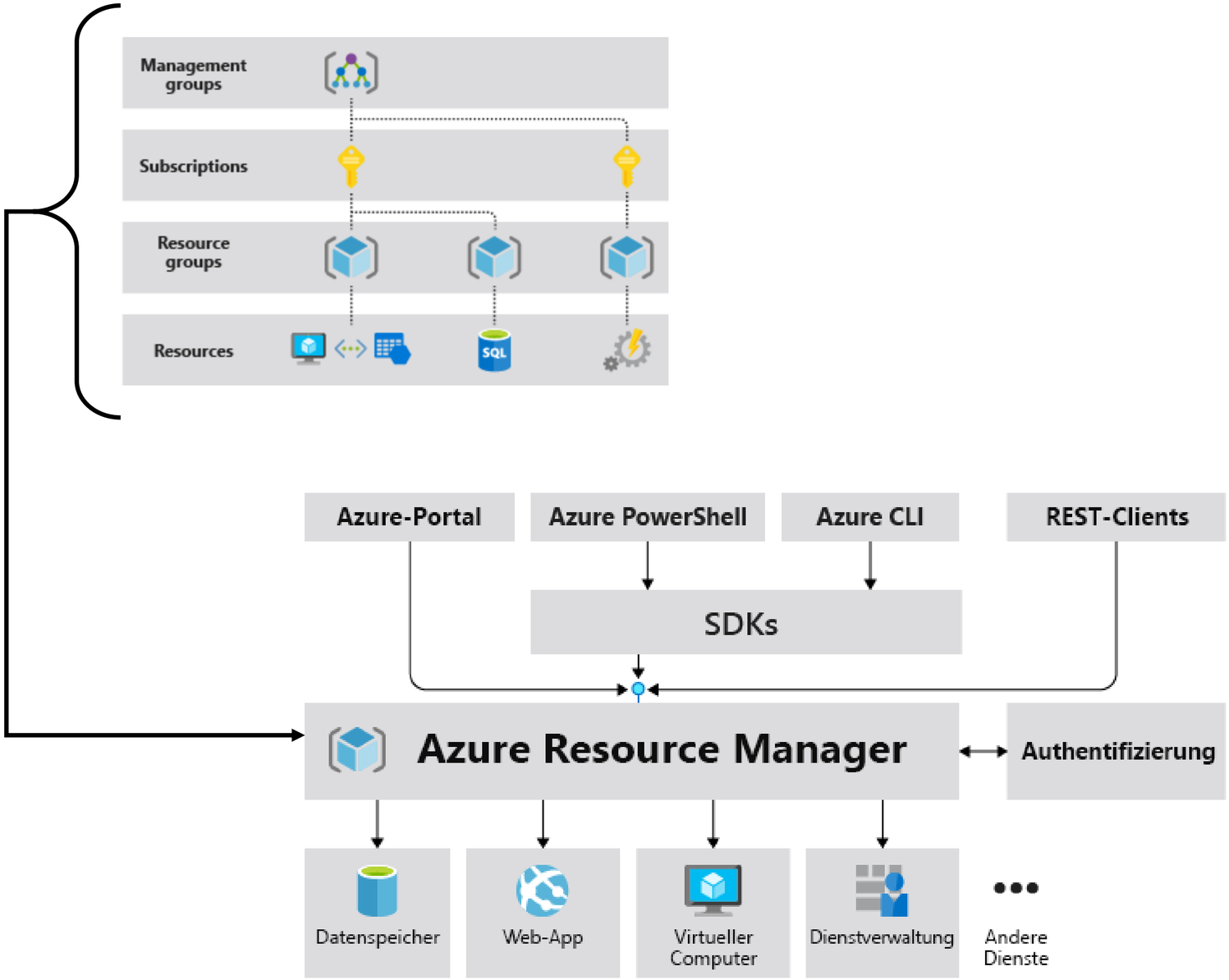
- **Warum wir Jenkins gewählt haben:**
  - bewährter Einsatz von Jenkins in Projekten
  - Jenkins auf eigenem Server

basysKom

Konfiguration der Cloud-Dienste

- **Auswahl der Konfigurationsschnittstellen**
  - Azure CLI
  - Azure Resource Manager (ARM) Templates
  - Azure REST API

- **Die Auswahl hängt an den Kriterien der Konfiguration**
  - Anzahl der Cloud-Dienste
  - Art der Cloud-Dienste
  - Komplexität der Konfigurierbarkeit (Abhängigkeiten)
  - Portabilität der Konfiguration (ARM vs. Shellscript)



Quelle: https://docs.microsoft.com/de-de/azure/azure-resource-manager/management/overview

basysKom

Erfahrungen mit den verschiedenen Konfigurationsschnittstellen

- **ARM Templates:**
  - Flexibel Konfigurierbar (Variablen, Funktionen, Abhängigkeitslogik)
  - Unübersichtlich ab einer gewissen Größe (Syntax-Fehler schwieriger zu finden)
  - Unserer Erfahrung nach ist dies der beste Weg um Azure Services zu deployen und zu konfigurieren. Z.B. den IoT Hub
- **Azure CLI**
  - Praktikabel zum Abfragen der Konfiguration der Cloud-Dienste
  - Einsatzgebiet in Shell Skripten (→ Mehraufwand beim Abbilden von Abhängigkeiten)
  - Repräsentiert nicht immer alle Konfigurationen eines Cloud Dienstes
- **REST API**
  - Einsatzgebiet in Applikationen
  - Nur in speziellen Fällen notwendig (→ Bereitstellung der Apps)

basyskom

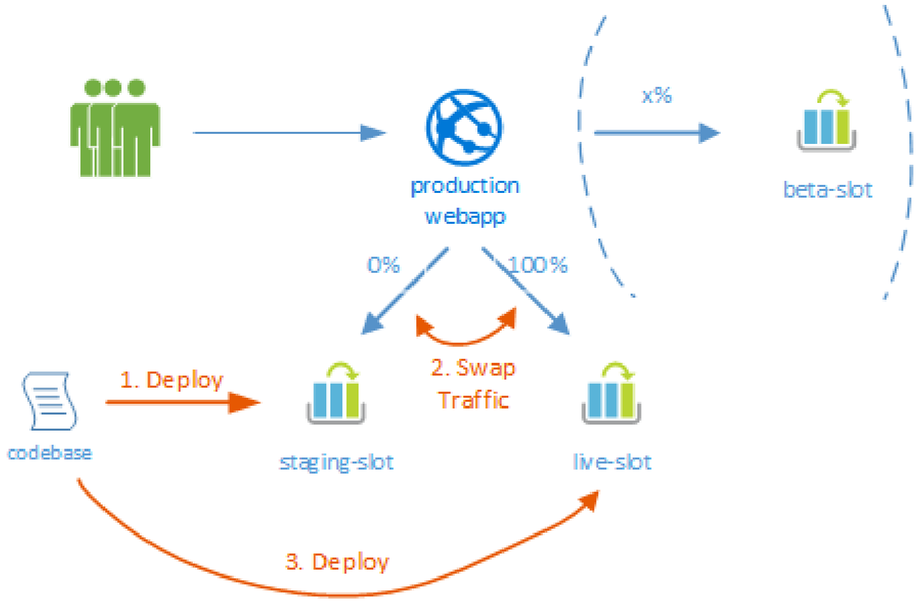
Erfahrungen bei der WebApp und FunctionApp Bereitstellung

- **Git Repository**
  - Pushen der Sourcen in ein Azure Git Repo
  - Automatischer Buildservice (Kudu) für Node.js, PHP, HTML, Python, Ruby, ASP.net, WebJobs und Azure Functions
  - *Fallstricke*:
    - Git Push Befehl blockiert bis App-Bereitstellung fertig ist
    - Buildprozess in der Cloud (Dauer u.a. abhängig von Serviceplan)
- **Docker Container**
  - Erstellen eines Docker-Images
  - Vorheriges Bauen der App notwendig (ggf. im Docker Container)
  - Upload des im Jenkins erzeugten Containers in eine Docker Container Registry notwendig
  - *Fallstricke*:
    - Konfigurieren des Docker-Image im Azure Web/Function Service
    - Man benötigt das ‚richtige' Base Image für den Container
- **Zip-Datei**
  - Vorheriges Bauen der App notwendig
  - Upload der (gebauten) App als Zip-Datei
  - Automatisches Entpacken in der Cloud (Kudu)
  - Upload über Azure-CLI oder REST API möglich
  - *Fallstricke*:
    - Deployment ist abhängig von Status der Azure Web/Function Service (Der Cloud Dienst muss „bereit" sein)
    - Azure CLI: Deployment dauerte öfters sehr lange (>25min) oder lief in Timeout
    - REST API: Asynchrones Deployment – zusätzliche Überprüfung, ob Deployment erfolgreich oder nicht

basysKom

Bereitstellung über Slots (Zero Downtime Bereitstellung)

- **Web und Function Apps stellen Slots zur Verfügung**
  - „Production“-Slot für das veröffentlichen der App
  - „Staging“-Slot zum Testen der App
  - „Staging“-Slot nach dem Testen mit dem „Production“-Slot tauschen (slot swapping)
- **Slots ermöglichen das zurückrollen auf vorherige Version**
- **„Staging“-Slot wird bei neuem Release immer mit der neusten Version bespielt**
- **Azure Services ohne Slot-Funktionalität müssen doppelt vorhanden sein**
  - z.B.: IoT Hubs, Servicebus (Topics, Queues), ...



https://onelittlespark.bartverdonck.be/testing-in-production/

basysKom

Zusammenfassung: Dies ist unserer Erfahrung nach ein stabil funktionierendes Setup:

- **CI System**
  - Jenkins
- **Konfiguration**
  - 10% Azure CLI
  - 90 % ARM Templates
- **App-Bereitstellung**
  - Docker Container
- **Release Konzepte**
  - Git-Flow
  - 4 Release/Deployment-Phasen
  - Bereitstellung über Slots

- **Takeaways:**
  - Deployment auf das Release-Szenario abstimmen
    - Analyse vor dem Aufbau des Deployment notwendig
  - Unterscheidung zwischen Service- und App-Bereitstellung
  - Deployment von WebApps und Functions über Docker Container hat sich bewährt
  - Deployment am Freitag Nachmittag ist keine gute Idee

Fragen?

basysKom GmbH

Robert-Bosch-Straße 7
64293 Darmstadt
Germany

Tel.: +49 6151 870 589 0

Email: sales@basyskom.com