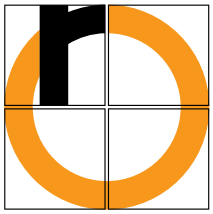
Hochschule **Rosenheim**
University of Applied Sciences



Fakultät für Informatik

Studiengang Informatik B.Sc.

Attack surfaces and security measures in Platform-as-a-Service solutions

Bachelor Thesis

von

Lukas Grams

Datum der Abgabe: TT.MM.JJJJ

TODO Erstprüfer: Prof. Dr. Reiner Hüttl

Zweitprüfer: Prof. Dr. Gerd Beneken

ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim, den XX.XX.2019 TODO

Lukas Grams

Abstract (german)

Die wachsende Zahl von Platform-as-a-Service (PaaS) Lösungen, Cloud-Umgebungen und Microservice-Architekturen bieten neue Angriffsszenarios. Dies erhöht den Bedarf an neuen Verteidigungs-Strategien in der IT-Sicherheit von Entwicklungs- und Produktiv-Umgebungen. Gerade Lösungen auf Basis von (Kubernetes-konformen) Container-Orchestrierung sind sowohl stark gefragt, as auch fundamental anders aufgebaut als andere, etabliertere Lösungen. Dieser Umstand legt eine nähere Untersuchung dieses Teilbereichs nahe.
Aus diesem Grund ist das Ziel dieser Arbeit, folgende Fragestellungen zu beantworten:

- Was für Sicherheits-Risiken existieren für den Anbieter und/oder Nutzer einer mandantenfähigen PaaS-Lösung, wenn jeder Mandant über eigene Entwicklungs-, Verteilungs- und Laufzeit-Umgebungen für seine Anwendungen verfügt?

- Wie kann ein Anbieter von PaaS-Lösungen an interne und/oder externe Nutzer diese Risiken Eindämmen?

- Was spricht in diesem Kontext aus Sicht eines PaaS-Anbieters jeweils für und gegen Vor-Ort- bzw. Cloud-Lösungen?

Ein weiteres Ziel ist es, Maßnahmen für die unterschiedlichen Implementierungsmöglichkeiten zu empfehlen. Falls der festgelegte Zeitrahmen es erlaubt, soll darüber hinaus folgende Fragestellung erörtert werden:

- Bietet eine selbstverwaltete Lösung in der Cloud (Einkauf von IaaS, Anbieten von PaaS) Vorteile im Vergleich zu den obigen Lösungen?
  (Sollten signifikante Unterschiede vorliegen, werden darüber hinaus weitere Lösungs-spezifische Empfehlungen zur Absicherung angeführt)

Unter den etablierten PaaS-Lösungen in unterschiedlichen Umgebungen finden sich OpenShift als Vor-Ort-Lösung und Azure Kubernetes Service für Cloud-Umgebungen. Zur Betrachtung einer selbstverwalteten Lösung in der Cloud kann OpenShift auf selbstverwalteten Instanzen in Azure dienen.

Um die genannten Ziele zu erreichen, grenzt diese Arbeit den Problembereich zuerst ein, indem sie sich auf spezifische Komponenten von zwei bis drei gängiger und Kubernetes-zertifizierter PaaS-Lösungen konzentriert. Das Hauptaugenmerk liegt hierbei auf den Kubernetes-konformen Teilen, da hier die Risiken und Maßnahmen den weitreichensten Gültigkeitsbereich haben, unabhängig von der betrachteten Lösung. Unter Betrachtung von drei gängigen Angriffsszenarien werden Schwachstellen und Angriffsvektoren identifiziert, ihr Schadenspotential evaluiert und das Risiko eingeschätzt:

- Angriff von böswilligen Dritten auf die Infrastruktur von innerhalb des LAN und/oder dem Internet
- Angriff von böswlligen Dritten aus einem Container heraus, über den die Kontrolle übernommen wurde. Ein Beispiel wäre das Asführen von Code oder Befehlen per Zugriff von außen.
- Fahrlässiger, übernommener oder böswilliger Nutzer bzw. Nutzer-Identität, was ein Kompromittierungsrisiko für diesen und/oder weitere Mandanten und deren Anwendungen darstellt.

Es werden potentielle Maßnahmen zur Risiko-Eindämmung gesucht, evaluiert und (soweit möglich) exemplarisch in der Praxis umgesetzt. Hierzu werden die PaaS-Lösungen im Problembereich genutzt. Mithilfe der Ergebnissewerden verschiedene Implementierungs-Empfehlungen verglichen, anhand der verschiedenen Anwendungsfälle differenziert und jeweils Maßnahmen empfohlen.

Schlagworte:
Platform-as-a-Service, PaaS, Cloud, Security, Container, Kubernetes, Docker, Risikoanalyse, OpenShift, Azure Kubernetes Service, AKS

Abstract (english)

The increasing amount of Platform-as-a-Service (PaaS) solutions, cloud-hosted environments and microservice architectures introduces new attack scenarios. This creates the need for new defense strategies in both Development and Operations. Especially solutions providing (Kubernetes conformant) container orchestration are identifiably different and in high demand compared to long established solutions. This calls for a more detailed, focused examination.
This thesis aims to answer the following questions:

- What generic security risks emerge when providing or using a multi-tenant PaaS solution, with each tenant developing, deploying and running their own applications?
- How can a PaaS provider (serving internal and/or external users) mitigate those risks?
- In this scope and from a PaaS provider viewpoint, how does an on-premise solution compare to a public cloud solution?

Another goal is to recommend security measures for different implementation use cases. If achievable within the provided time frame, it will additionally try and answer this question:

- Does a cloud-hosted, self-managed solution (buy IaaS, provide PaaS) offer benefits in contrast to the solutions compared above?
  (In case there are sufficient differences, another set of security measures will be recommended for this solution)

Examples for widely used PaaS solutions in different environments include OpenShift as an on- premise solution and Azure Kubernetes Service as a public cloud solution. To include a cloud-hosted, self-managed solution, OpenShift running on self-managed instances in Azure could serve as an example. To achieve these goals, the thesis will first limit the view on the problem to a manageable scope by focusing on specific components of a few (2-3) commonly used PaaS solutions, specifically Certified Kubernetes solutions. Components providing Kubernetes conformity will be the main focus, as these bear the most significance across all Kubernetes Certified solutions.

Looking at three common attack scenarios, it will then determine vulnerabilities and attack vectors, as well as their potential damage and rate those risks:

- Malicious third party attacking the underlying infrastructure from within the LAN and/or the internet
- Malicious third party attacking from inside a hijacked container, i.e. remotely executing code or commands
- Bad User, i.e. a negligent, hijacked or malicious developer (account) risking compromise of his own and/or other applications

Possible measures to mitigate those risks will also be explored, evaluated and (if possible) put to use in practical examples, leveraging the PaaS solutions within scope. With the results gathered, the thesis will compare different best practice implementations for different use cases and recommend measures for each.

Keywords:
Platform-as-a-Service, PaaS, Cloud, Security, Container, Kubernetes, Docker, Risikoanalyse, OpenShift, Azure Kubernetes Service, AKS

Contents

List of Figures

List of Tables

Listings

Abkürzungsverzeichnis

LED Light Emitting Diode

1 Introduction

With this chapter, the reader should be able to comprehend why this thesis was written, what it tries to accomplish and which topics are considered in this work.

1.1 Motivation

The increasing amount of Platform-as-a-Service (PaaS) solutions, cloud-hosted environments and microservice architectures introduces new attack scenarios. This creates the need for new defense strategies in both Development and Operations. Especially solutions providing (Kubernetes conformant) container orchestration are identifiably different and in high demand compared to long established solutions. This calls for a more detailed, focused examination.

1.2 Objective

This thesis aims to answer the following questions:

- What generic security risks emerge when providing or using a multi-tenant PaaS solution, with each tenant developing, deploying and running their own applications?
- How can a PaaS provider (serving internal and/or external users) mitigate those risks?
- In this scope and from a PaaS provider viewpoint, how does an on-premise solution compare to a public cloud solution?

Another goal is to recommend security measures for different implementation use cases. If achievable within the provided time frame, it will additionally try and answer this question:

- Does a cloud-hosted, self-managed solution (buy IaaS, provide PaaS) offer benefits in contrast to the solutions compared above?
  (In case there are sufficient differences, another set of security measures will be recommended for this solution)

1.3 Scope limitation

To achieve the aforementioned goals, the view on the problem is limited to a manageable scope by focusing on specific components of a few commonly used (Certified Kubernetes) PaaS solutions. These components will be OpenShift Container Platform for On-Premise use cases and Azure Kubernetes Service as a public cloud solution. To include a cloud-hosted and self-managed solution, OpenShift running on self-managed instances in Azure serve as an example implementation.
Components providing Kubernetes conformity will be the main focus, as these bear the most significance across all Kubernetes Certified solutions.
A look at popular tools and frameworks used in such clusters will be avoided in order to keep the scope manageable, though some might be recommended as a mitigation.
This thesis aims to provide insight to the risks of providing a PaaS solution and mitigations thereof. As such, it will look at the capabilities a potential provider has to (mis-)configure such solutions - inherent risks of the technologies themselves are only explored when measures to mitigate them are accessible from a provider standpoint.
In short, the goal is to improve the security of your Kubernetes cluster, not Kubernetes itself.
Looking at three common attack scenarios, it will then determine vulnerabilities and attack vectors, as well as their potential damage and rate those risks:

- Malicious third party attacking the underlying infrastructure from within the LAN and/or the internet
- Malicious third party attacking from inside a hijacked container, i.e. remotely executing code or commands
- Bad user, i.e. a negligent, hijacked or malicious developer (account) risking compromise of his own and/or other applications

Possible measures to mitigate those risks will also be explored, evaluated and (if possible) put to use in the practical examples. With the results gathered, the thesis will compare different best practice implementations for different use cases and recommend measures for each.

TODO: Versioning Freeze! Openshift, AKS (not possible...), Docker engine, Kubernetes versions.

2 Theory

With this chapter, a reader with foundational knowledge of topics regarding Computer Science and/or Informatics should be able to grasp the specialized technologies discussed within the thesis and familiarize themselves with the definitions and terminology used throughout this thesis.

2.1 Infrastructure-as-a-Service

In Infrastructure-as-a-Service (IaaS) environments, a consumer trusts his IaaS provider with the management and control of the infrastructure needed to deploy his applications. The provided service ends at provisioning or processing, storage, networks and other computing resources **nistcloud** (TODO Quelle: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf). Therefore consumers do not need to manage their own data centers or topics like system uplink availability. As shown in Figure 2.1, consumers are responsible for the operating system (OS) layer and everything above it.

(TODO: you manage <-> other manages https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/)

citation test 1**servicecomparison**
cit test 3 **servicecomparison**
cit

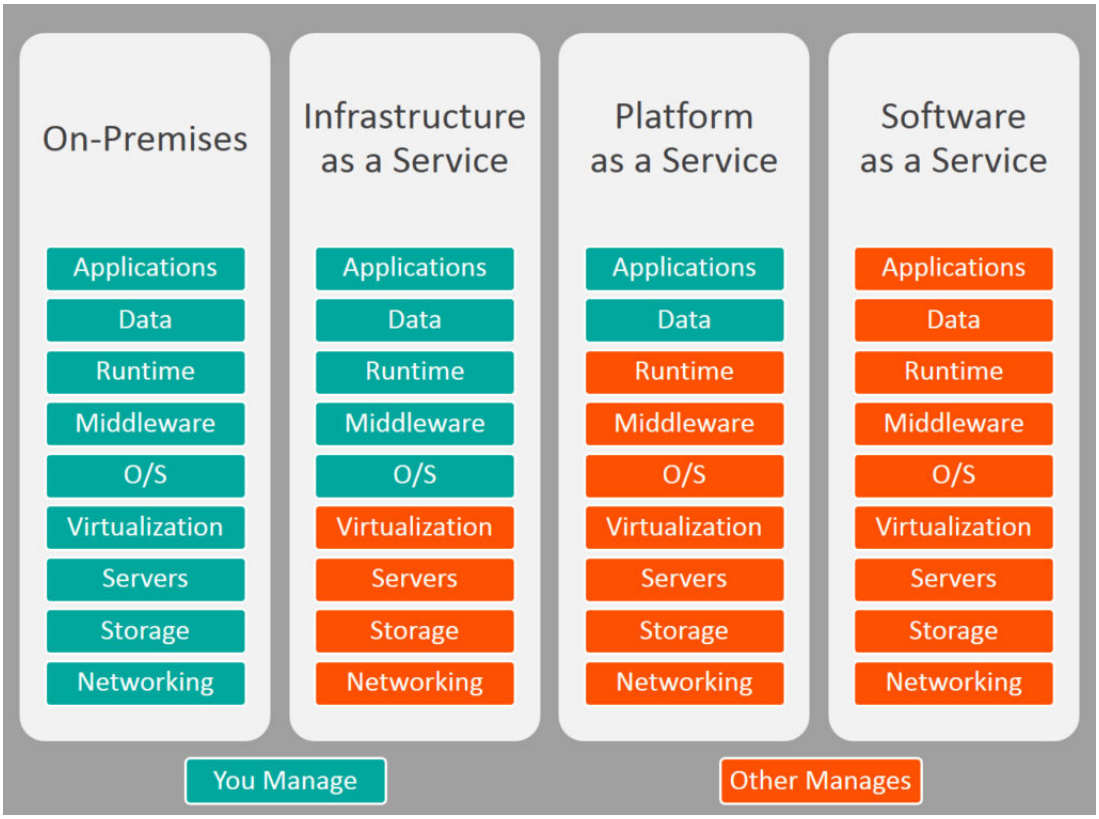


Figure 2.1 captionright[1]

2.2 Platform-as-a-Service

In Platform-as-a-Service (PaaS) environments, a consumer trusts his PaaS provider with the management and control of even more resources needed to deploy his applications beyond those covered by IaaS. (TODO Quelle: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf) In an ideal scenario, this leads to a consumer not having to concern himself with the underlying network, hardware, servers, operating systems, storage or even common middleware like log data collection and analysis (TODO Quelle: https://docs.microsoft.com/en-us/azure/data-explorer/) and allows him to focus on other tasks, i.e. application development. As a downside to this, a consumer might only have limited control on, among others, the software installed on the machines provisioned. Although this shifts some of the responsibility burden towards the provider, the situation isn't as clear-cut as one might think. Figure X (TODO: Picture link) shows middleware and runtime as provided, but there is no clear standard on what capabilities or tools are included. A consumer might require capabilities which aren't provided or wishes to avoid provider lock-in through proprietary

1 Source: **servicecomparison**

tools, again resulting in middleware responsibilities for the consumer. A consumer might also have to extensively configure the application-hosting environment for compliance or security purposes. Even some low-level tasks aren't completely managed, i.e. Virtual Machine (VM) reboots to apply security updates. (TODO Quelle: https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-security?view=azuremgmtbilling-1.1.0-preview#process-node-updates-and-reboots-using-kured)

2.3 Containers, Docker

TODO

concept of container vs. VM, upsides vs. downsides (?), docker-specific b/c highest market share automation & load elasticity -> need for orchestration

https://www.docker.com/resources/what-container
https://www.docker.com/sites/default/files/d8/2018-11/docker-containerized-and-vm-transparent-bg.png
Citations: TODO Look for books!

2.4 Container Orchestration, Kubernetes

TODO

idea: control system, architecture (control plane vs. application(s), kubelet, master/non-master nodes, ...), standard (can have other implementations), certification requirements, uses in general and in AKS/Openshift specifically.

https://kubernetes.io/docs/concepts/#overview
https://kubernetes.io/docs/concepts/overview/components/

2.4.1 Kubernetes Objects

TODO higher-level idea behind objects Pods, Services, Volumes, Namespaces

2.4.2 Kubernetes Controllers

TODO

higher-level idea behind controllers ReplicaSets, Deployments, StatefulSets, DaemonSets, Jobs

2.4.3 Using Kubernetes

TODO

build from scratch, buy CaaS/PaaS/IaaS, cloud vs. on-prem, different scopes/features from different products

https://kubernetes.io/docs/setup/pick-right-solution/

2.5 TODO: Others?

TODO

3 Potential attack scenarios

TODO

3.1 Defining procedures and approach

TODO

3.2 Scenario 1: Third party over network

TODO

3.2.1 TODO: Vector one

TODO

3.2.2 TODO

TODO

3.3 Scenario 2: Third party inside container

TODO

3.3.1 TODO: Vector one

TODO

3.3.2 TODO

TODO

3.4 Scenario 3: Bad user

TODO

3.4.1 TODO: Vector one

TODO

3.4.2 TODO

TODO

4 PaaS solution risk analysis

TODO

4.1 Differentiating on-premise and public cloud environments

TODO

4.2 OPTIONAL: Comparison of cloud-hosted, self-managed environments

If this is omitted, the section above will become the chapter TODO

5 Best-Practice implementations

TODO

5.1 On-premise environment

TODO

5.2 Public cloud environment

TODO

5.3 OPTIONAL: cloud-hosted, self-managed environment

TODO

6 Summary

TODO

7 Einleitung

7.1 Anführungszeichen, Zitieren, Fußnoten

Quellen stehen in der Datei thesis.bib!
Da gibt es auch ein tag, um Bücher von Online-Quellen zu unterscheiden.

So wird zitiert**booktest**. Das ist eine Quelle für Bücher.
Das ist eine Online-Quelle**wikimoscow**. Um kleine Definitionen zu verlinken benutze ich Fußnoten[2]

So werden in Deutschland Anführungszeichen gemacht: „Sehr wichtig!!“

Die Abkürzungen stehen in der Datei abbreviations.tex!
Das abkürzungsverzeichnis wird natürlich automatisch erstellt.

So eine Light Emitting Diode (LED) ist schon toll.

2 `https://www.eclipse.org/4diac/` (Abgerufen am 07.02.2019)

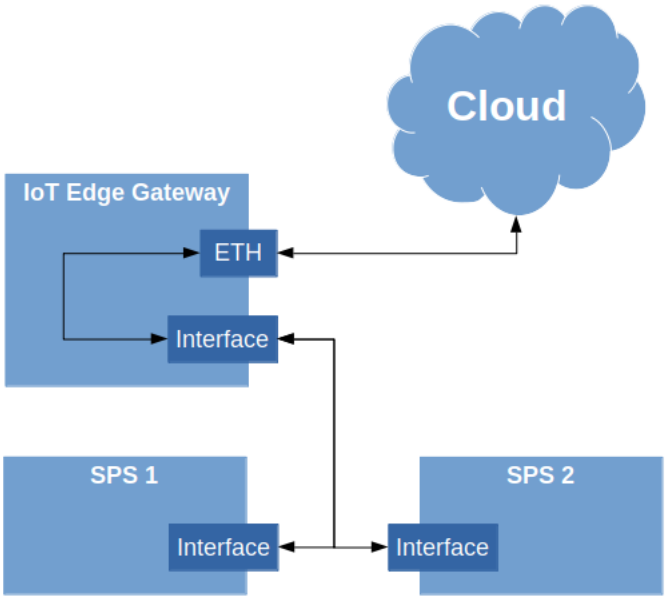


Figure 7.1 Bild wird automatisch im Abbildungsverzeichnis eingefügt!

8 Einführung und Analyse der Anforderungen

Die Abbildung 8.1 zeigt...

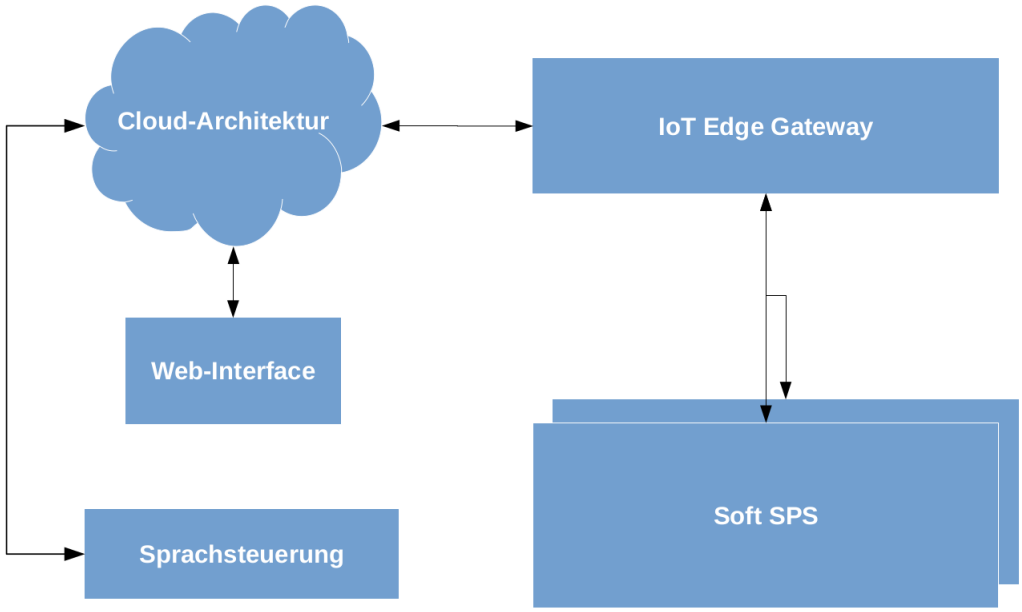


Figure 8.1 Bild wird automatisch im Abbildungsverzeichnis eingefügt!

8.1 Tabelle und Liste

Hier gibts eine Fancy Tabelle:

Table 8.1 Anforderungen an die Sprachsteuerung

Nr.	Anforderung	Erklärung	Priorisierung
(1a)	Einfache Installation	einfache „Out-of-the-box“ Installation und intuitives Anlegen von Nutzerkonten	Must

Liste um alles etwas in die Länge zu strecken:

- Blah
- Blah

Bilder:

Figure 8.2 Bild wird automatisch im Abbildungsverzeichnis eingefügt!

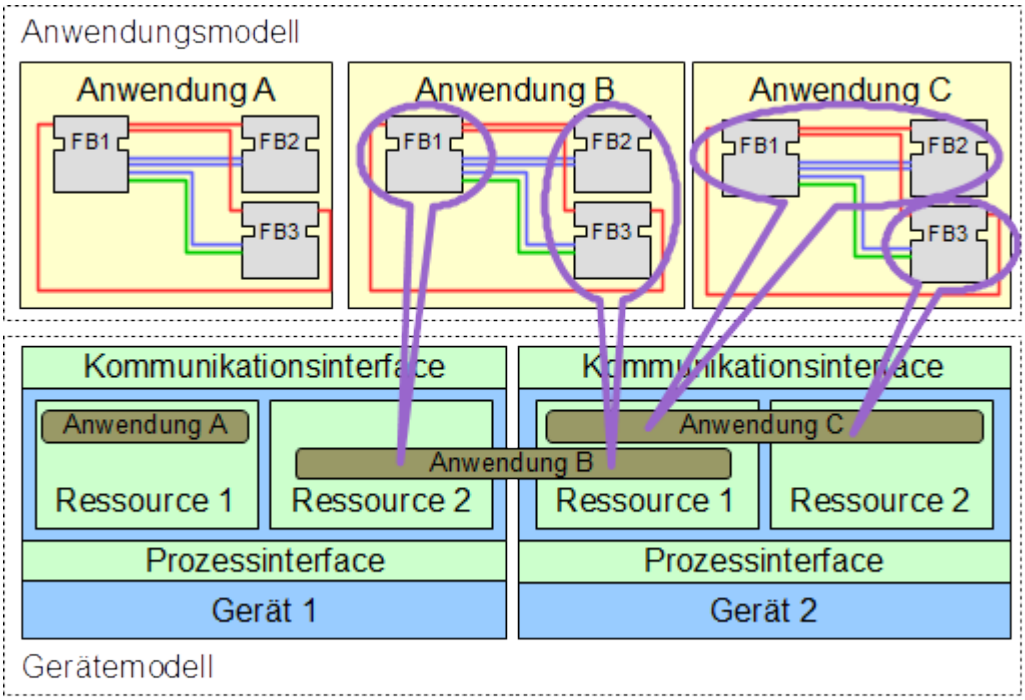


Figure 8.3 Bild wird automatisch im Abbildungsverzeichnis eingefügt!

9 Automat mit tikz

Falls jemand bock hat einen Automaten zu basteln, hier ist eine kleine Vorlage:

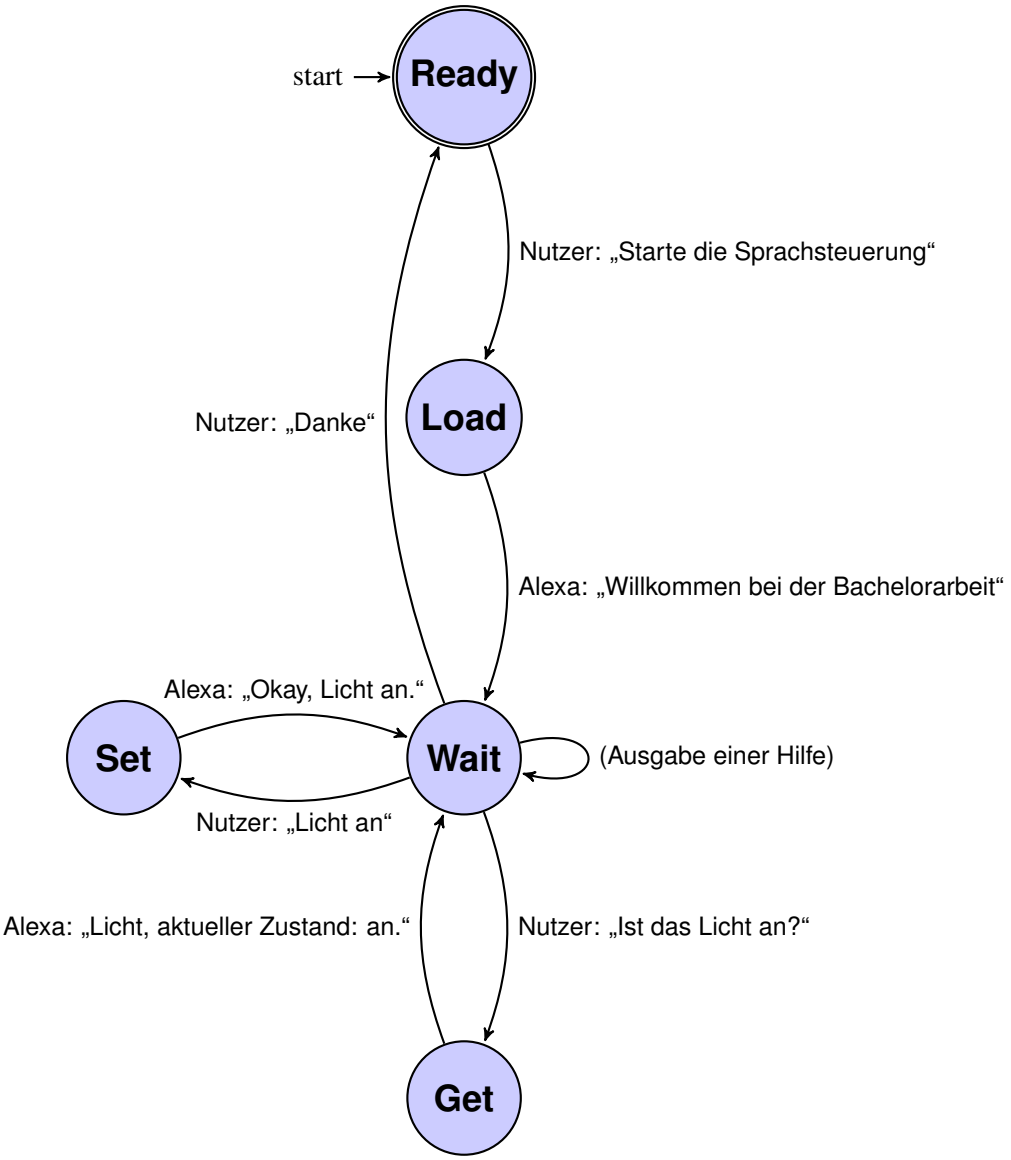


Figure 9.1 Automat des dialogbasierten Alexa Skill

9.0.1 verlinken

Verlinken mit Abbildungsnummer und Seite:
In der Abbildung 9.1 auf Seite 19 wird gezeigt wie nice tikz sein kann.

10 Listings

Immer nice, Listings mit Synthax-Highlighting!

Listing 10.1 Datei: setup-posix-mqtt.sh, Zeilen 31-48

```
1   cd "./$forte_bin_dir"
2
3   cmake -G "Unix Makefiles" \
4       -DFORTE_ARCHITECTURE=Posix \
5       -DFORTE_COM_ETH=ON \
6       -DFORTE_COM_FBDK=ON \
7       -DFORTE_COM_LOCAL=ON \
8       -DFORTE_TESTS=OFF \
9       -DFORTE_TESTS_INC_DIRS=${forte_boost_test_inc_dirs} \
10      -DFORTE_TESTS_LINK_DIRS=${forte_boost_test_inc_dirs} \
11      -DFORTE_MODULE_CONVERT=ON \
12      -DFORTE_MODULE_IEC61131=ON \
13      -DFORTE_MODULE_UTILS=ON ../../ \
14      -DFORTE_MODULE_SysFs=ON \
15      -DFORTE_COM_PAHOMQTT_INCLUDE_DIR=/home/pi/paho.mqtt.c/src \
16      -DFORTE_COM_PAHOMQTT=ON \
17      -DFORTE_COM_PAHOMQTT_LIB=/home/pi/ ... /libpaho-mqtt3a.so \
18      -DFORTE_COM_PAHOMQTT_LIB_DIR=/home/pi/paho.mqtt.c/build/output
```

So wird verlinkt:

Das Listing 10.1 auf Seite 21 zeigt in den Zeilen 34 bis 56 wie hässlich ein Shellscript werden kann. Hier wurde der Style custombash verwendet. Im Listing 10.2 auf Seite 21 wird der customc-Style verwendet.

Listing 10.2 IntentHandler für den LaunchRequest

```
1   handle(handlerInput) {
```

```
2
3       var userID = handlerInput.requestEnvelope['session']['user']['userId'];
4       var autorisationsDaten = require('./autorisation.json');
5       var juconid = getJuconnIDbyAlexaID(userID, autorisationsDaten);
6
7       if(juconid == 0) {
8
9           // Tell User how to authorize, and exit Skill
10          ...
11      }else{
12
13          // Welcome User, start Interaction according to state-machine
14          const requestAttributes = handlerInput.attributesManager.
                getRequestAttributes();
15          const sessionAttributes = handlerInput.attributesManager.
                getSessionAttributes();
16          const speakOutput = requestAttributes.t('WELCOME_MESSAGE',
17                                  requestAttributes.t('SKILL_NAME'));
18          const repromptOutput = requestAttributes.t('WELCOME_REPROMPT');
19          handlerInput.attributesManager.setSessionAttributes(sessionAttributes);
20          return handlerInput.responseBuilder
21              .speak(speakOutput)
22              .reprompt(repromptOutput)
23              .getResponse();
24      }
25  }
```

Bibliography