Internet of Things

Bräutigam / Kraul

2021
ISBN 978-3-406-74898-1
C.H.BECK

deren Beseitigung erfordert zumeist einen geringeren Aufwand. Neben diesen beiden Standardvarianten sind in der Praxis auch weitere Ansätze geläufig, etwa Customizing mit ergänzender Programmierung von Sonderfunktionen.[135]

bb) Open-Source-Software. Auch im Rahmen von Apps spielt der Einsatz von **OSS** eine erhebliche Rolle und es stellen sich die damit verbundenen Fragen (→ Rn. 65 ff.). Eine besondere Akzentuierung ergibt sich allerdings im Hinblick auf die Beurteilung, inwieweit der Anbieter oder der Nutzer für die Einhaltung der OSS-Bedingungen verantwortlich ist. Maßgeblich hierfür ist, dass beim Einsatz von Apps der Programmcode nicht zwingend auf der Plattform, sondern etwa bestimmte Skripte auch auf den Systemen des Nutzers ausgeführt werden können. 92

b) Deployment und Aktualisierung

Der Betrieb einer App auf der Plattform erfordert deren Installation und Konfiguration. Beides wird als **Deployment** (Softwareverteilung) bezeichnet. Das Deployment umfasst in einem weiten Verständnis neben der Installation der Software auch deren Aktualisierung und ggf. auch das Entfernen der Software von der Plattform. Der Betrieb von komplexen IoT-Plattformen, zumal solchen, die (auch) als Ökosysteme für Third Party Apps fungieren, stellt hohe technische und operative Anforderungen an das Deployment. Es werden verschiedene Formen des Deployments unterschieden: 93

- *Manuell:* Möglich, wegen der damit verbundenen Nachteile jedoch in der Praxis eher unüblich ist ein vollständig **manuelles Deployment.** Hierbei muss die Applikation manuell auf den zuvor aufgesetzten Server transportiert und dort ausgeführt werden, wobei diverse Anpassungen an das jeweilige Betriebssystem auf dem Server erforderlich sein können. Gerade das Aufsetzen eines Servers kann mit erheblichem Arbeitsaufwand verbunden sein. Das manuelle Deployment wird häufig mithilfe einer umfangreichen Dokumentation oder eines Handbuchs ausgeführt, da zahlreiche Einzelschritte erforderlich sind.
- *Skripte:* Daneben ist ein Deployment durch **Skripte,** also dedizierte Softwareprogramme möglich, die Arbeitsanweisungen für die einzelnen Unterschritte des Deployments beinhalten. Diese Skripte können manuell oder durch andere Skripte aufgerufen werden.[136] So können einige der Arbeitsschritte, die beim manuellen Deployment anfallen, eingespart werden.
- *Virtuelle Maschine:* Darüber hinaus kann für den Deployment-Prozess eine **virtuelle Maschine** genutzt werden. Eine virtuelle Maschine ist ein virtuelles Computersystem, bei dem die Hardware nachgebildet wird, auf der wiederum verschiedene Betriebssysteme laufen können. In diesem Fall sind keine Skripte zur Ausführung notwendig, da die Einstellungen der virtuellen Maschine über Tools justiert werden können. Die Software befindet sich hier in der virtuellen Maschine, in der auch das Betriebssystem simuliert wird.[137] Die virtuelle Maschine wird auf den Server transportiert und dort ausgeführt.
- *Container:* Am geläufigsten ist wohl das **Container-Deployment.** Hierbei werden sog. Container, also standardisierte Softwareeinheiten erstellt, die die konkrete Anwendung enthalten, um diese auf dem Server auszuführen (→ Rn. 106 ff.). Hier kann jeder Container im Prinzip als einzelne, voll funktionsfähige virtuelle Maschine betrachtet werden. In jedem Container wird ein Betriebssystem simuliert, auf dem mehrere Anwendungen laufen können. Es kann aber anders als bei der virtuellen Maschine nur ein Betriebssystem simuliert werden. Trotz der vielfältigen Einsatzmöglichkeiten eines Containers benötigt er weniger Speicherplatz als eine virtuelle Maschine.[138]

[135] Vgl. *Sarre* in Auer-Reinsdorff/Conrad IT-R-HdB § 1 Rn. 56.
[136] *Link* S. 23.
[137] *Link* S. 24.
[138] www.docker.com/resources/what-container (11. 12. 2019).

94 **aa) Herausforderungen beim Deployment.** Beim Deployment stellen sich unabhängig von der auf der Plattform zu betreibenden Anwendung typische Herausforderungen, denen durch die Auswahl der geeigneten **Deployment-Strategie** begegnet werden kann. Diese typischen Problemfelder betreffen insbesondere Kompatibilität, Einfachheit und Sicherheit des Deployment-Vorgangs.

95 **(1) Kompatibilität.** Zentrales Ziel des Deployment ist es, dass die auf einem lokalen Gerät entwickelte Anwendung auch auf dem Server fehlerfrei abläuft. Probleme können insofern auftreten, wenn die Software nur an die lokale Entwicklungsebene angepasst war, auf dem Server hingegen nicht funktionsfähig ist. Zur Gewährleistung der **Kompatibilität** eignet sich besonders das Container-Deployment, da im Container die Rechnerarchitektur eines real existierenden oder hypothetischen Rechners nachgebildet wird und der Container so auf jeder Umgebung funktionsfähig ist. Dafür wird häufig die als Open-Source-Software verfügbare Container-Lösung Docker Engine eingesetzt.

96 Um die Kompatibilität einer Anwendung mit der Plattform zu gewährleisten, stehen ferner bestimmte Deployment-Strategien zur Verfügung, die nicht nur beim Container-Deployment anwendbar sind. Allerdings ergeben sich gerade bei einer **kombinierten Anwendung** weitere Vorteile. Erwähnenswert sind dabei **Blue/Green-Deployment** und Canary-Deployment. Darüber hinaus gibt es den veralteten Ansatz vom „Big Bang"-Deployment, auf den hier wegen der mangelnden praktischen Relevanz nicht näher eingegangen werden soll.

97 Beim sog. Blue/Green-Deployment werden Anwendungen auf zwei separaten, aber ansonsten inhaltsgleichen Systemen entwickelt. Wird im Rahmen der **kontinuierlichen Auslieferung** (→ Rn. 107) ein Update bereitgestellt, erfolgt die Implementierung zunächst nur auf einem der beiden Systeme. Da immer nur eine Umgebung im System aktiv genutzt wird, kann die andere als Test-Umgebung eingesetzt werden. Ein Wechsel auf die aktive Umgebung erfolgt erst dann, wenn sichergestellt ist, dass die Anwendung fehlerfrei läuft. Falls es doch zu Anwendungsfehlern kommt, ist zudem jederzeit ein Wechsel auf die vorherige Version möglich, die noch in der Test-Umgebung zur Verfügung steht.

98 Die Funktionsweise des Canary-Deployment ähnelt der des Blue/Green-Deployment. Dieser Ansatz ist jedoch noch **risikoärmer,** da der Wechsel zwischen den verschiedenen Systemen nicht in einem einzigen Schritt erfolgt. Die Software wird vielmehr nach und nach auf den Servern implementiert (sog. Rolling-Deployment), wenn zuvor keine Fehler aufgetreten sind.

99 **(2) Einfachheit.** Eine weitere Herausforderung für ein erfolgreiches Deployment ist die Schaffung eines einfach handhabbaren **Gesamtprozesses,** mittels dessen die Fehleranfälligkeit des manuellen Deployment-Verfahrens reduziert werden kann. So kann die Überführung einer Anwendung von der Entwicklungs- in die davon verschiedene Produktivumgebung Probleme bei der Softwareausführung mit sich bringen. Zudem kann der Deployment-Prozess auch aus mehreren Einzelschritten bestehen, die zwingend in einer besonderen Reihenfolge vorgenommen werden müssen. Je mehr der Einzelschritte des Deployments automatisiert ablaufen, desto weniger aufwändig und mithin fehleranfällig ist dieses. Mittel zur Automatisierung sind beispielsweise kontinuierliche Integration und kontinuierliche Auslieferung (→ Rn. 107 ff.).

100 Zudem besteht auch der Bedarf, eine Anwendung vor dem Deployment auf der Plattform durch andere Entwickler **testen** zu lassen. Ist dafür erst eine langwierige Installation notwendig, erschwert dies den Arbeitsalltag erheblich. Hierfür ist der Einsatz einer virtuellen Maschine nur bedingt geeignet, denn diese beinhaltet ein Gastbetriebssystem, das häufig mehrere Gigabytes an Speicherplatz in Anspruch nimmt. Dies kann die Festplattenkapazitäten eines Entwicklungsrechners überstrapazieren. Auch dafür bietet das Container-Deployment Lösungen. So beinhaltet oder erfordert der Docker-Container kein eigenes Betriebssystem, was die Größe der Anwendung erheblich reduziert und das Starten und

Beenden des Containers beschleunigt. Ferner lassen sich mit geeigneten Containerlösungen komplexe Anwendungen schnell erstellen und veröffentlichen.[139]

Das Container-Deployment dient der **Vereinfachung** weiterhin dadurch, dass kein eigener Server aufgesetzt werden muss. Das Aufsetzen eines Servers ist uU mit erheblichem zeitlichen Aufwand verbunden. 101

(3) Sicherheit. IoT-Plattformen stellen hohe Anforderungen an die **IT-Sicherheit.** Diese betreffen insbesondere auch das Deployment von Anwendungen auf der Plattform, gerade dann, wenn diese die Integration von Third Party-Apps ermöglicht. Hierzu sind Sicherheitstools- und -techniken anzuwenden. Besondere Synergieeffekte können entstehen, wenn Sicherheitstests bereits in die kontinuierliche Integration und kontinuierliche Auslieferung integriert werden. 102

Sicherheitsrisiken beim Internet der Dinge ergeben sich insbesondere daraus, dass auf die Plattform regelmäßig sensible Daten übermittelt werden, deren Geheimhaltung von hoher Bedeutung ist. Dabei muss sichergestellt werden, dass nur tatsächlich berechtigte Personen über Anwendungen auf der Plattform Zugriff auf diese Daten erhalten. Weiterhin ist zu vermeiden, dass Anwendungen potentiell schädliche Software enthalten, die auf andere auf dem Server liegende Daten zugreifen oder die Plattform anderweitig kompromittieren könnte. Zur Vermeidung dieser Szenarien werden verschiedene Strategien eingesetzt. Eine Möglichkeit ist das vorherige Testen der Software vor deren Deployment auf den Server (sog. Quality Gate). Dies kann entweder automatisiert mithilfe entsprechender Software erfolgen oder es kann ein sog. Review Team eingesetzt werden, welches Anwendungen vorab kontrolliert. Letzteres ist allerdings mit hohem zusätzlichen Arbeits- und Personalaufwand verbunden. Daher kommt dies aus Praktikabilitätsgründen nur bei offensichtlich kritischer Software in Betracht und kann in der Regel nicht generell vor Deployment auf sämtliche Anwendungen erstreckt werden. 103

Die Anforderungen an ein vorheriges Testing sind vor allem dann hoch, wenn nicht auf Container-Deployment gesetzt wird. Dieses bietet eine technische **Sicherheitsschranke** dadurch, dass Container als geschlossene Einheit von einem direkten Plattform-Zugriff abgekoppelt sind. Zudem kann der Plattformbetreiber eigenständig entscheiden, welche Zugriffsrechte er für den Container einräumt. Erhält dieser keine sog. Administratorenrechte, ist über den Container üblicherweise kein Zugriff auf andere Plattforminhalte möglich. Insbesondere kann (ohne eine spezifische Berechtigung) nicht auf diejenigen Inhalte zugreifen, die sich in anderen Containern befinden.[140] 104

(4) Verantwortlichkeit für das Deployment. Üblicherweise erfolgt der Betrieb von Anwendungen auf einer IoT-Plattform und damit auch deren Deployment durch den **Betreiber** der Plattform. Der Plattformanbieter kann sich aber auch dazu entschließen, seine Plattform für Third Party-Apps zu öffnen. Dabei kann den Drittanbietern der Zugang in unterschiedlich weitem Umfang eingeräumt werden. Beispielsweise kann der Drittanbieter selbst berechtigt sein, seine Anwendungen zu implementieren. Dies stellt allerdings besondere Anforderungen an Sicherheitsmaßnahmen für die Plattform. In diesem Fall ist die Plattform strikt gegen unberechtigte Zugriffe durch und Schadsoftware als Teil der Third Party-Apps abzuschirmen, insbesondere durch den Einsatz von Container-Lösungen. 105

bb) Deployment-Prozesse. Wesentliche Aspekte des **Deployment-Prozesses** ist neben der eigentlichen Implementierung der Anwendung auf der Plattform deren Testing und Aktualisierung. 106

[139] *Link* S. 24.
[140] docs.docker.com/engine/security/ (20.11.2019).

107 **(1) Bau des Containers und dessen Verteilung.** Wegen der hohen Praxisrelevanz soll das Container-Deployment näher erläutert werden. Hierbei wird nicht eine statische Entwicklungsumgebung auf der Plattform eingesetzt, sondern mittels einer Software, die sich zwischen Plattform und Anwendung befindet, **virtuell** ein Server repliziert. Der einzelne Container muss kein vollständiges Betriebssystem enthalten, sondern benötigt nur die für die Ausführung relevanten Daten. Als Standardsoftware für das Container-Deployment hat sich die Open Source-Lösung Docker etabliert.[141]

108 Im Wesentlichen kann bei der Nutzung von Docker zwischen zwei **Schritten** unterschieden werden. Zunächst muss ein Container auf einem lokalen Rechner ‚gebaut' werden, bevor er in Schritt zwei auf den Server transportiert und dort ausgeführt werden kann.[142] Ein Container ist eine standardisierte Softwareeinheit. Jeder einzelne Container enthält sämtliche Software, die für die Ausführung der Anwendung auf der Plattform notwendig ist (beispielsweise Quellcode, Tools, Einstellungen und System Libraries). Es handelt sich um eigenständige Software-Pakete, die mit hoher Verlässlichkeit auf eine fremde Umgebung transportiert und dort ausgeführt werden können. Die Container sind nicht von einem Server oder einer bestimmten Umgebung abhängig und es können auch mehrere Container auf einem Server laufen, ohne sich gegenseitig bei der Ausführung zu behindern.

109 **(2) Testing.** Um die Funktionsfähigkeit, Kompatibilität und Sicherheit des Betriebs einer Anwendung auf der Plattform sicherzustellen, sind vor jedem erstmaligen Deployment wie auch vor Updates Tests der Anwendung **(Quality Gate)** durchzuführen (→ Rn. 103). Hierbei werden zunehmend die Prinzipien der kontinuierlichen Auslieferung (Continuous Delivery) angewandt. Darunter wird eine Sammlung von Techniken, Prozessen und Tools verstanden, die das Deployment verbessern. Im Hintergrund werden kontinuierliche Überprüfungen jedes Entwicklungsstands der Anwendung einschließlich aller Änderungen und Erweiterungen sowie automatisierte Tests für eine jederzeit lauffähige Software durchgeführt. Durch die sofortige Überprüfung der Änderungen entsteht eine Kette der Qualitätssicherung, die schnellstmöglich Fehler aufzeigen kann. Wegen der Kleinteiligkeit der Prüfungsschritte lassen sich Fehler genau identifizieren. Anders als bei klassischen Entwicklungssystemen kann der Nutzer auch beinahe jederzeit Softwareupdates anfordern. Diese Vorgehensweise beschleunigt Prozesse, da nicht erst separate Tests der Überarbeitungen notwendig sind, und bietet daher monetäre Anreize.

110 **(3) Aktualisierung.** Nach der Bereitstellung werden Anwendungen durch **Updates** aktualisiert. Hierdurch können etwa Anwendungsfunktionen verbessert oder nachträglich aufgetretene Fehler korrigiert werden.[143] Sicherheitsprobleme können ebenfalls Updates erforderlich machen. Die Bereitstellung von Daten und das Einspielen von Updates lässt sich schneller und zuverlässiger realisieren, wenn möglichst viele Prozesse automatisiert werden. Daher sind automatisierte Prozesse zur Implementierung von Updates manuellen in der Regel vorzuziehen. Dabei sind die Begriffe kontinuierliche Integration (Continuous Integration) und ebenfalls kontinuierliche Auslieferung zu nennen.

111 **Kontinuierliche Integration** beschreibt eine Software-Entwicklungstechnik, bei der Entwickler ihre Arbeit in regelmäßigen Abständen in einem zentralen Repository zusammenfügen. Das Repository ist das Verzeichnis, in dem die Software liegt. Ein bekanntes Repository ist beispielsweise GitHub. Die Software wird zunächst lokal geschrieben und dann manuell ins Repository geschoben. Diese Integration sollte mindestens auf einer täglichen Basis erfolgen. Nach jeder Integration läuft ein automatischer Test ab, der Fehler so schnell wie möglich erkennt. Der Umfang der zu testenden Software wird dabei vom Ent-

[141] www.docker.com/resources/what-container (20.11.2019).
[142] www.docker.com/resources/what-container (25.10.2019).
[143] *Pruß/Sarre* in Auer-Reinsdorff/Conrad IT-R-HdB Technisches Glossar: Update.

wickler selbst festgelegt. Hierdurch kann Aufwand vermieden werden, der durch vollumfängliche Tests der gesamten Software auch nach nur geringen Aktualisierungen entstehen würde. Durch das Testen der aktualisierten Software lassen sich Integrationsfehler reduzieren und schnell eine geschlossene Software entwickeln. Gerade bei großen Softwareprojekten lassen sich so lange Integrationsprozesse mit ungewisser Dauer vermeiden.

Eine **Aktualisierung** kann auch erforderlich sein, wenn sich die Bedingungen auf der 112
Plattform verändern, etwa APIs geändert oder neu implementiert werden. Davon unberührt bleiben jedoch regelmäßig Container, da diese unabhängig von den jeweiligen Plattformbedingungen agieren. Eine Anpassung wäre lediglich bei einem manuellen oder durch Skripte unterstützen Deployment erforderlich. Diese Unabhängigkeit von Änderungen der Plattform ist ein weiterer Grund, warum verstärkt Container-Deployment eingesetzt wird.

c) KI und Algorithmen

Zunehmend werden auch im IoT-Umfeld Anwendungen eingesetzt, die auf Methoden 113
der künstlichen Intelligenz basieren. Dabei handelt es sich um voll funktionsfähige Anwendungen, die **Machine Learning** einsetzen. Die technischen Grundlagen der KI werden in → § 11 erläutert (→ § 11 Rn. 12 ff.).

4. User-Interfaces

Der Zugang zu und die Bedienung von IoT-Plattformen erfolgt über Benutzerschnittstel- 114
len, sog. **User Interfaces.** Begrifflich sind diese abzugrenzen. Zum einen werden Hardware-Interfaces eingesetzt, um Geräte miteinander zu verbinden. Weiterhin dienen sog. Application Programming Interfaces dem Austausch von Daten zwischen verschiedenen Software-Komponenten (→ Rn. 25). Hier soll es ausschließlich um User Interfaces gehen. Sie ermöglichen es dem Nutzer, mit einem Gerät oder einer Software zu kommunizieren. Ihre Qualität hat wesentliche Bedeutung für die User Experience. Dabei werden vor allem graphische Benutzeroberflächen und textbasierte User Interfaces eingesetzt. Graphische Benutzeroberflächen enthalten neben Text auch graphische Bedienelemente wie Icons. Mit dem Klicken auf ein Icon oder einen Text wird ein bestimmter Teil des Programms ausgeführt, der für die durch den Benutzer ausgewählte Aktion zuständig ist.[144]

User Interfaces erleichtern dem Nutzer die Verwendung und den **Zugriff** auf die IoT- 115
Plattform und die hierauf von ihm verwendeten Anwendungen. So kann der Nutzer Einstellungen ändern und seinen Account verwalten, etwa Devices hinzufügen und entfernen, Admin- und User-Rollen zuweisen und diesen Berechtigungen erteilen oder entziehen (→ Rn. 55 ff.). Über User Interfaces können ferner Benutzerdaten eingegeben und diese der Programmlogik zur Verfügung gestellt werden.[145] Ein besonders nutzerfreundliches User Interface zeichnet sich dadurch aus, dass der Nutzer durch minimale Eingaben das gewünschte Resultat erzielt und die Schnittstelle leicht und bestenfalls intuitiv zu bedienen ist.

User Interfaces basieren letztlich auf Programmcode, wobei verschiedene Arten der Be- 116
reitstellung unterschieden werden. Geläufig sind einerseits **Mobile Apps,** die etwa auf Smartphones und Tablets installiert werden können (und die begrifflich von Apps im Sinne von gegenüber Nutzern im Rahmen von SaaS-Angeboten zur Verfügung gestellten Anwendungen abzugrenzen sind (→ Rn. 88 ff.)). Andererseits werden User Interfaces in Form von Clients bereitgestellt. Diese können einerseits auf der Plattform betrieben und lediglich der Zugriff über einer Browser ermöglicht (sog. Web Clients) oder aber zur lokalen Installation angeboten werden.

---

[144] pythonbuch.com/gui.html (27. 11. 2019).
[145] *Wiebe* GRUR-Int. 1990, 21 (22).

a) Mobile Apps

117 Als Mobile Apps werden Anwendungen für **mobile Endgeräte** bezeichnet. Hierbei wird zwischen sog. Native Apps und Web Apps unterschieden.[146] Native Apps werden für ein bestimmtes Betriebssystem entwickelt und sind auf dessen Funktionen und Fähigkeiten optimal abgestimmt. Der Zugriff auf Native Apps erfolgt über einen in das Betriebssystem des mobilen Endgeräts integrierten App Store. Eine fertig entwickelte Native App ist daher erst dann verfügbar, wenn sie in den jeweiligen App Store eingestellt ist.[147] Dies erfolgt in der Regel nach Prüfung der App durch den App-Store-Betreiber. Native Apps sind plattformabhängig, das heißt sie müssen den jeweiligen technischen Rahmenbedingungen des App Stores entsprechen.[148] Daher ist es nicht möglich, etwa eine Android-App auf einem iPhone zu installieren.[149]

118 Eine **Mobile App** muss anderen Anforderungen genügen als beispielsweise ein Web Client. So muss eine Mobile App trotz eines in der Regel kleineren Bildschirms auf dem Device problemlos für den Nutzer ausführbar sein. Dadurch müssen die Symbole kleiner abgebildet werden und nicht immer wird eine Textbeschreibung beim jeweiligen Button aus Platzgründen möglich sein. Dennoch muss die Anwendung für den Nutzer verständlich und leicht nutzbar bleiben.

119 **Web Apps** werden hingegen mit Webtechnologien realisiert, weshalb sie unabhängig vom benutzten Betriebssystem sind und nicht über App Stores bereitgestellt werden.[150] Im Kontext von mobilen Endgeräten zählen auch sie zu den Mobile Apps. Da sie sich in ihrer Funktionsweise jedoch nicht von sonstigen Web Interfaces unterscheiden, werden sie im folgenden Abschnitt zu Web Clients erläutert.

b) Web Clients

120 **Web Clients**, auch Web Apps genannt, werden auf Basis von Webtechnologien wie HTML5 entwickelt.[151] Client-seitige Programme werden überwiegend über den Webbrowser von einem Webserver abgerufen und müssen nicht auf dem Endgerät installiert werden.[152] Die grafische Aufbereitung von Websites erfolgt dann auf der Client-Seite im Webbrowser, die Ausführung der Anwendung hingegen auf dem Server (sog. Thin-Client).[153]

121 Allerdings erfordern manche SaaS-Angebote zur Nutzung des Dienstes auch spezielle Clients, die auf dem Endgerät installiert werden müssen. Die Rechenleistung wird in diesem Falle weiterhin im Server des Anbieters erbracht, jedoch wird die Anwendung direkt auf dem Client ausgeführt (sog. **Fat-Client**).[154] Dies kann beispielsweise erforderlich sein, um lange Ladezeiten zu vermeiden.[155]

c) Einsatz von Open-Source-Software

122 Im Rahmen von User Interfaces kann auch auf **OSS** zurückgegriffen werden (→ Rn. 65 ff., 92). Für die Frage, ob der Anbieter oder der Nutzer für die Einhaltung der OSS-Bedingungen verantwortlich ist, kommt es wesentlich darauf an, ob die entsprechende Software auf der Plattform oder aber auf einem Device des Nutzers ausgeführt wird.

---

[146] *Pruß/Sarre* in Auer-Reinsdorff/Conrad IT-R-HdB Technisches Glossar: App.
[147] Vgl. *Baumgartner/Ewald* Apps Rn. 91.
[148] *Baumgartner/Ewald* Apps Rn. 91.
[149] *Abts/Mülder* S. 504 f.
[150] *Abts/Mülder* S. 505.
[151] *Pruß/Sarre* in Auer-Reinsdorff/Conrad IT-R-HdB Technisches Glossar: App.
[152] *Abts/Mülder* S. 128, 505.
[153] *Abts/Mülder* S. 123.
[154] Vgl. *Schorer* in Hilber Cloud Computing HdB Teil 1 C Rn. 34.
[155] *Küchler* in Bräutigam IT-Outsourcing und Cloud Computing S. 210.

Insofern ist zu unterscheiden. Während das sog. Backend stets auf der Plattform betrieben wird, befinden sich die Skripte zur Ausführung der OSS als Teil von Mobile Apps sowie Clients auf dem lokalen Device des Nutzers.

§ 5 Ökosysteme

Übersicht

Literatur:

Auer-Reinsdorff/Conrad (Hrsg.), Handbuch IT- und Datenschutzrecht, 3. Aufl., München 2019; *Borges/Meents* (Hrsg.), Rechtshandbuch Cloud Computing, München 2016; *Bräutigam/Rücker* (Hrsg.), Rechtshandbuch E-Commerce, München 2017; *Degmair,* Apps – Die schwierige Suche nach dem Vertragspartner, K&R 2013, 213; *Hoeren,* Datenbesitz statt Dateneigentum, MMR 2019, 5; *Höppner/Schulz,* Die EU-Verordnung („PZB-Verordnung“) für Fairness und Transparenz von Online-Vermittlungsdiensten 2019/1150, ZIP 2019, 2329; *Karbaum/Schulz,* Einstweilige Maßnahmen der Kartellbehörden – wirksamer Wettbewerbsschutz im Zeitalter digitaler Ökosysteme, NZKart 2019, 407; *Langer/Appt,* Unternehmen „App“ – Datenschutz- und Vertragsrechtliche Herausforderungen rund um die App-Entwicklung und den App-Vertrieb aus der Inhouse Perspektive, DSRITB 2013, 69; *Martini/Botta,* Iron Man am Arbeitsplatz? – Exoskelette zwischen Effizienzstreben, Daten- und Gesundheitsschutz, NZA 2018, 625; *Nolte/Hecht,* Plattformverträge, ITRB 2006, 188; *Pieper/Schneider,* Die Besonderheiten bei Entwicklungsverträgen für „smarte“ Systeme, DSRITB 2015, 805; *Sassenberg/Faber* (Hrsg.), Rechtshandbuch Industrie 4.0 und Internet of Things, 2. Aufl., München 2020; *Söbbing,* Plattform as a Service, ITRB 2016, 140; *Solmecke/Taeger/Feldmann* (Hrsg.), Mobile Apps, Berlin/Boston 2013; *Taeger/Pohle* (Hrsg.), Kilian/Heussen, Computerrechts-Handbuch, 35. Erg.-Lfg., München 2020; *Vocke/Kunkel/Hilbert,* Industrie 4.0 aus rechtlicher Sicht, WPg 2018, 1039.

I. Der Begriff „Ökosystem“

1 Der wirtschaftlich sinnvolle Einsatz sog. „smarter“ Endgeräte setzt ein mehrschichtiges Zusammenwirken verschiedener physischer bzw. softwareseitiger Ebenen voraus. Dabei besteht eine vollständige „IoT-Umgebung“ (zu den Begrifflichkeiten bereits → § 2) für die Nutzer „smarter“ Endgeräte in aller Regel aus

- grundlegender technischer Infrastruktur (dazu → § 3),
- den technischen Lösungen für die Anbindung der jeweiligen Endgeräte (Stichwort: Konnektivität, dazu → § 3),
- und schlussendlich (proprietären) Leistungen, gestützt auf **Softwareanwendungen** (**Softwareapplikationen** bzw. **„Apps“**).

2 Letztere werden – schon den hardwareseitig engen Grenzen diverser Endgeräte geschuldet – oft als sog. „as a Service-Lösung“ zur Verfügung gestellt (dazu grundlegend → § 7 Rn. 430 ff.).[1] Ein derartiges Wirkungsgefüge verschiedener Leistungsebenen und der beteiligten Akteure lässt sich auch als Ökosystem bezeichnen.[2]

II. Bedeutung und Ausblick

3 In der digitalen Ökonomie haben sich längst wirtschaftlich bedeutsame Ökosysteme etabliert. Dabei ist das Ökosystem aber stets mehr als nur die Summe der einzelnen beschriebenen Teilleistungen oder Akteure. Es werden vielmehr einzelne physische wie digitale

[1] S. auch *Vöcke/Kunkel/Hilbert* WPg 2018, 1039 (1043) und zum „as a Service“-Begriff *Schmidt/Pruß* in Auer-Reinsdorff/Conrad IT-R-HdB § 3 Rn. 327.

[2] Zum Begriff auch *Bräutigam* in Bräutigam/Rücker E-Commerce-HdB 1. Teil D Rn. 2.