

Master „Data Science and Business Analytics“
Fakultät für Information und Kommunikation
Hochschule der Medien, Stuttgart
Modul „BI- and Big-Data-Architectures“
von Prof. Dr. Peer Küppers

Projekt „PyctureStream“

Prototypische Implementierung einer Architektur für verteiltes
Image-Processing in Micro-Batches

von Holger Büch und Marcus Fixel
Februar/März 2018

Inhaltsverzeichnis

Einleitung	2
Projektplanung	3
Business Case	4
Showcase	4
Fachliche Anforderungen	5
Entwicklung der Architektur	6
Leistungsanforderungen	6
Big Data vs. traditionelle BI Architektur	6
Horizontale vs. vertikale Skalierbarkeit	6
Cloud vs. On-Premise	7
Lambda vs. Kappa	8
Softwareauswahl	9
Beschreibung der Referenzarchitektur	10
Proof of Concept	12
Vereinfachung der Architektur	12
Einrichtung der Infrastruktur	12
Setup der Software-Komponenten	12
Implementierung	13
Fazit	16
Quellenverzeichnis	17
Anhang	19

Einleitung

Die vorliegende Dokumentation beschreibt die Entwicklung eines Fallbeispiels aus dem Bereich Data Analytics und der dazugehörigen Big Data Architektur. Ziel des knapp achtwöchigen Projekts war es, dieses Fallbeispiel prototypisch End-To-End abzubilden.

Zunächst werden kurz die nötige Projektplanung, Rollenverteilung und Methoden der virtuellen Zusammenarbeit thematisiert. Anschließend wird die Geschäftsidee skizziert und ein exemplarischer Showcase aus dem Bereich der Bildverarbeitung beschrieben, der die Idee in einem Projektvorhaben konkretisiert. Aus diesem Showcase werden die für eine Implementierung nötigen fachlichen Anforderungen hergeleitet.

Diese fachlichen Anforderungen dienen als Grundlage der Big Data Architektur des Projekts. Auf ihrer Basis werden die verschiedenen Architekturentscheidungen getroffen und ausführlich begründet. Die so hergeleitete Referenzarchitektur wird anschließend detailliert beschrieben.

Die Verprobung der entwickelten Architektur in der Praxis bildet den letzten Teil der Arbeit. Hierin wird die Implementierung des exemplarischen Showcases in Form eines Proof of Concept (PoC) beschrieben, der trotz nötiger Vereinfachungen in der Architektur den Showcase End-to-End abbildet.

Projektplanung

Nach der Methode des Projektstrukturplans (Aichele 2006, S.75-79) wurde das Projekt in Arbeitspakete (Ebene 1.x) gegliedert (Abb. 1) und in die erforderlichen Aufgaben unterteilt (Ebene 1.x.x). Aufgaben, die im Rahmen des Moduls als nicht essenziell angesehen und eingespart wurden, sind grau hinterlegt.

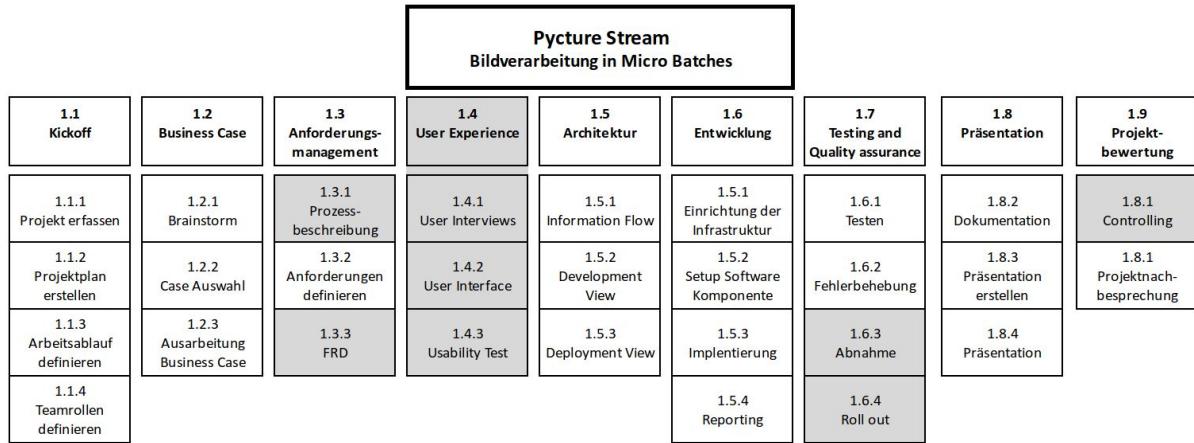


Abbildung 1: Projektstrukturplan

Der zeitliche Aufbau des Projekts wurde mit dem Dienst agantty.com als Gantt-Chart (s. Anhang 1) geplant und verfolgt. Zusätzlich wurden Google Drive und Github für die virtuelle Zusammenarbeit genutzt. Wöchentliche Skype-Meetings dienten dem Austausch zum Projektfortschritt. Besonders hilfreich waren zusätzliche mehrstündige Face-to-Face-Meetings, in denen die PoC-Implementierung durchgegangen und aktuelle Schwierigkeiten besprochen wurden.

Marcus Fixel fokussierte sich in seiner Rolle auf die Projektplanung und Business-Aspekte, entwickelte daher maßgeblich das Geschäftsmodell und die damit verbundenen fachlichen Anforderungen. Für den PoC übernahm er zusätzlich die Implementierung des Reporting.

Holger Büch übernahm im Projekt die Rolle des Entwicklers der Referenzarchitektur. Er übersetzte die fachlichen in technische Anforderungen und war bei der Entwicklung und Implementierung des PoC federführend.

Business Case

Die Businessidee ist die Gründung einer Beratungsfirma, die Unternehmenskunden im Big-Data-Umfeld unterstützt. Durch eine Spezialisierung auf Architekturen für Bildanalyse ist eine Differenzierung von Wettbewerbern vorgesehen. Um effizient unterstützen zu können, wird auf eine generische Referenzarchitektur gesetzt, die an individuelle Kundenanforderungen angepasst werden kann. Als Showcase wurde aus einem Pool mehrerer Ideen (s. Anhang 2) der Use-Case „Assistenz-App für Blinde“ ausgewählt. An ihm können alle funktionalen Aspekte der Lösung demonstriert werden. Außerdem ist der Use-Case im vorgegebenen Rahmen des Kurses umsetzbar und komplex genug, um zahlreichende Lernmöglichkeiten zu bieten. Nicht zuletzt erzeugt der Wohltätigkeits-Aspekt der App eine gewisse Aufmerksamkeit sowie positive Emotionen und birgt das Potenzial, tatsächlich nützlich zu sein.

Showcase

Die Sinnesorgane geben dem Menschen die Möglichkeit, seine Umgebung wahrzunehmen. Dem Fernsinn Sehen wird hierbei eine besondere Bedeutung beigemessen (Mallot 2006, S. 127). Die Idee des Showcase ist es, blinden Menschen die Möglichkeit zu geben, sich ihre Umgebung beschreiben zu lassen. Dazu wird eine Smartphone-App bereitgestellt, mit der Bilder des Umfelds aufgenommen werden können. Unsere Lösung detektiert die darauf erkennbaren Objekte und Personen und stellt den Nutzern eine verständliche Beschreibung der Umgebung per Sprachausgabe zur Verfügung.¹ Um diesen Showcase realisieren zu können, müssen große Datenmengen in kurzer Zeit verarbeitet werden. Dies bietet die ideale Gelegenheit, um die Leistungsfähigkeit der Referenzarchitektur zu demonstrieren.

Aus Business-Sicht wird angenommen, dass die Kosten für diese Lösung nicht allein von den Endkunden getragen werden können. Aus diesem Grund wird das Projekt unter einem sozialen und gemeinnützigen Aspekt aufgebaut und mithilfe von Sponsoren und Partnern finanziert. Eine zusätzliche Abo-Gebühr wird eher als Steuerungsmittel für die Nutzungsintensität eingesetzt: Durch sie kann eine zweckfremde Nutzung, etwa durch nicht sehbehinderte Menschen, und die damit verbundenen Kosten verringert werden. Die verschiedenen Business-Aspekte können mit der Methode des Business Model Canvas (BMI 2018) übersichtlich dargestellt werden (s. Abb. 2).

¹ Durch die Automatisierung skaliert unsere Lösung besser als Ansätze vergleichbarer Apps wie etwa Be My Eyes (2019), die für die Bildbeschreibungen auf Crowdsourcing und Chat-Funktionalität setzen.



Abbildung 2: Business Model Canvas

Fachliche Anforderungen

Aus Business- und Showcase wurden folgende fachlichen Anforderung (FA) abgeleitet:

- [FA1] Die Architektur soll generisch genug sein, um verschiedene Use-Cases abbilden zu können, aber andererseits flexibel genug, um eine gewisse Anpassbarkeit gewährleisten zu können.
- [FA2] Im ersten Jahr wird für einen durchschnittlichen Use-Case von 720 Stunden Videoaufnahme pro Tag in HD-Qualität ausgegangen.²
- [FA3] In den darauffolgenden Jahren wird stetiges Wachstum antizipiert, die Architektur soll daher möglichst stufenlos skalierbar sein.
- [FA4] Das Ergebnis der Bildanalyse soll den Kunden innerhalb von maximal drei Sekunden bereitgestellt werden. Eine längere Zeitspanne würde Use-Cases wie eine Überwachung von Gefahrenbereichen gefährden (s. Anhang 2).
- [FA5] Der Datenschutz spielt eine sehr wichtige Rolle: Insbesondere die Videoaufnahmen sollen bestmöglich vor Zugriffen Dritter geschützt sein.
- [FA6] Aus diesem Grund ist die Speicherung und nachträgliche Verarbeitung des Bildmaterials für einen Standard Use-Case nicht vorgesehen.
- [FA7] Das beauftragende Unternehmen soll befähigt werden, die Lösung eigenständig zu betreiben und weiterzuentwickeln. Dies inkludiert die Vermeidung eines Vendor-Lock-Ins.

² Zum Beispiel 30 Kameras mit 24 Stunden Laufzeit pro Tag in einem Überwachungs-Use-Case oder 2880 Kameras mit 15 Minuten pro Tag bei der Assistenz-App für blinde Menschen.

Entwicklung der Architektur

Leistungsanforderungen

Für die Abschätzung der Anforderungen bezüglich Datenvolumen und Verarbeitungsgeschwindigkeit in einem kleineren Use-Case wie der Assistenz-App für blinde Menschen wurden folgende Annahmen getroffen:

- 1 Bild einer Full-HD-Kamera hat im JPEG-Format eine Größe ca. 300 kB (s. FA2).
- Die Verarbeitung eines Bildes alle 3 Sekunden ist pro Kamera ausreichend.
- Insgesamt streamen die Kameras ca. 720 Stunden pro Tag.³ (s. FA2)
- Metadaten der erkannten Objekte werden zum Zweck des Monitorings temporär gespeichert. Die Datenmenge ist bei ca. 200 b pro Bild vernachlässigbar.⁴
- Das Reporting wird auf einer durchschnittlichen Virtuellen Maschine (VM) betrieben.
- Ein existierendes Deep-Learning-Model zur Objekterkennung wird vorausgesetzt. Die Beschaffung durch Erwerb oder Entwicklung wird hier nicht berücksichtigt.

Hieraus lässt sich ein benötigter Datendurchsatz für Netzwerk, Message Queue und Processing von rund 260 GB pro Tag kalkulieren, sowie eine Rechenleistung abschätzen, die durch etwa 180 gängige CPUs erbracht werden kann (s. Anhang 3).

Big Data vs. traditionelle BI Architektur

Allein die zu prozessierende Datenmenge („Volume“) von 260 GB pro Tag erfordert zumindest in der Anfangsphase nicht unbedingt eine Big Data Architektur. Doch die geplante Skalierung, die erforderliche hohe Verarbeitungsgeschwindigkeit („Velocity“), die Heterogenität aus Bild-, Log- und relationalen Daten („Variety“), sowie die Unschärfe („Veracity“) der Daten aus der Objekterkennung⁵ indizieren den Bedarf einer Big Data Architektur.

Horizontale vs. vertikale Skalierbarkeit

Um die Leistungsanforderungen an das Processing (s. FA2, FA4) erfüllen zu können, aber gleichzeitig eine möglichst stufenlose Skalierung (s. FA3) zu ermöglichen, ist eine horizontal skalierbare Big Data

³ Es wird zur Vereinfachung eine gleichmäßige Verteilung der Last angenommen.

⁴ Die Bilder selbst werden aus Datenschutzgründen nicht archiviert.

⁵ Ein Modell für visuelle Objekterkennung wird Objekte nie mit Sicherheit detektieren, sondern immer Wahrscheinlichkeitswerte für einzelne Objekte zurückliefern.

Architektur einer vertikale Skalierbaren vorzuziehen. Denn durch die Nutzung eher günstiger Commodity-Hardware oder gar einer Cloud-Infrastruktur ist eine flexiblere Anpassung der Infrastruktur möglich, als bei einer tendenziell teureren und nur beschränkt erweiterbaren monolithischen Hardware-Lösung (e.g. SAP HANA), die zusätzlich als Single point of Failure ein schwer einschätzbares Risiko bedeuten würde.

Cloud vs. On-Premise

Kostenabschätzung

Im Cloud-Betrieb⁶ ist zunächst eine VM für Reporting sowie einfache Tests und Monitoring nötig. Auf der Google Cloud Platform (GCP) erscheint hierfür der Service „Cloud Compute“ in gemäßigter Ausstattung ausreichend und ist mit 166 \$ pro Monat inklusive Storage kein Kostentreiber. Als Streaming Service erscheint „Cloud Dataflow“ des Typs „Streaming“ sinnvoll. Die benötigte Konfiguration ist nicht ganz klar: Angaben zum Datendurchsatz in Abhängigkeit von Anzahl und Typ der Worker Nodes konnten nicht zuverlässig recherchiert werden. Wir halten drei Worker Nodes mit mittlerer Ausstattung für insgesamt 761 \$ pro Monat für realistisch, in der Praxis würden wir Autoscaling nutzen und während Last-Tests Erfahrungswerte sammeln. Verschiedene Optionen gibt es für den Funktionsbereich Analytics bzw. Objekterkennung. Exemplarisch wurden drei verschiedene Konfiguration mit den Services „Machine Learning“, „Vision API“ oder „Cloud Dataproc“ betrachtet. Die Variante mit „Machine Learning“ erreicht vermutlich nicht die geforderte Verarbeitungsgeschwindigkeit und bleibt daher unberücksichtigt (s. Anhang 6). Die beiden anderen Services sind aus technischer Sicht denkbar und in unserem Use-Case etwa gleich teuer. Auf drei Jahre gerechnet ist für den Cloud-Betrieb mit Betriebskosten von insgesamt rund \$78.000 zu rechnen.

Die Hardwarekosten im On-Premise-Betrieb sind von sehr vielen Faktoren abhängig. Wir führen an dieser Stelle eine beispielhafte Rechnung auf Basis von Empfehlungen von Dell (2017) an, die sich als Partner für die Cloudera-Distribution positionieren. Demnach ist ein Setup mit elf Dell PowerEdge Servern in einer Konfiguration für je 4.180 \$ denkbar (s. Anhang 7). Die Anschaffungskosten für die Server belaufen sich auf ca. 50.000 \$. Hinzu kämen laufende Kosten von etwa 700 \$ pro Monat für die Unterbringung in einem europäischen Datencenter inklusive Strom- und Netzwerkversorgung (vgl. Leaseweb 2018). Lizenzkosten für Software können durch die Verwendung von Software mit

⁶ Für den Betrieb in der Cloud wurden exemplarisch Dienste der Google Cloud Platform betrachtet, idealerweise müssten die Überlegungen für alternative Anbieter analog durchgeführt werden.

geeigneten Open Source Lizzenzen⁷ vermieden werden. Auf drei Jahre gerechnet bedeutet dies Kosten in Höhe von 75.200 \$.

Bei der Bewertung⁸ der genannten Optionen für den geschilderten eher kleinen Use-Case wiegt der preisliche Vorteil der Infrastruktur im On-Premise-Betrieb nicht die zu erwartenden erhöhten Personalkosten für die Einrichtung und den Betrieb der Hardware auf. Im Cloud-Betrieb können durch dynamische Skalierung (Autoscaling) Kosten gespart werden und das Investitionsrisiko ist geringer. Dennoch wird ersichtlich, dass die Kosten für Cloud-Computing nicht zu vernachlässigen sind: Wäre ein größeres Cluster notwendig, so würden die Kosten der Service-Nodes, die Personal- und Unterbringungskosten weniger stark ins Gewicht fallen. Die On-Premise-Variante würde so schnell an Attraktivität gewinnen.

Datenschutz & Sicherheit

Der geforderte höchste Schutz der Daten (s. FA5) kann nur in einer vom Internet getrennten Umgebung, also einer On-Premise-Lösung erreicht werden. Diese ist daher vorzuziehen. Einige Use-Cases, etwa die Assistenz-App für Blinde, benötigen jedoch eine Verbindung zum Internet. In solchen Fällen muss abgewägt werden, ob die Unternehmens-IT on-premise einen höheren Schutz als in der Cloud erreicht. Von der Nutzung von Services wie „Vision API“, bei denen das Bildmaterial zur Verarbeitung an ein anderes Unternehmen übertragen wird, ist wegen unzureichender Transparenz und fehlender Kontrollmöglichkeiten abzuraten.

Lambda vs. Kappa

Um eine anpassbare, generische Architektur anbieten zu können (s. FA1) und mit Blick auf die Befähigung des Kunden, den Betrieb und die Weiterentwicklung vollständig zu übernehmen (s. FA7), empfiehlt sich der Rückgriff auf erprobte und bekannte Architekturmödelle wie Lambda oder Kappa.

Für eine Lösung mit den beschriebenen Anforderungen ist die Kappa-Architektur geeigneter: Beide Architekturen ermöglichen zwar eine Stream-Verarbeitung (s. FA4), jedoch wird bei der Kappa-Architektur der zusätzliche Layer zur Batch-Bearbeitung der Lambda-Architektur eingespart. Dieser ist auch nicht nötig, da das Bildmaterial nicht gespeichert wird (s. FA6). Dadurch sinkt die Komplexität des Technologie-Stacks, was letztendlich den Aufwand und damit die Kosten der Entwicklung und des Betriebs verringert. Wäre die Verwerfung des Bildmaterials nicht vorgesehen,

⁷ Denkbar ist die Verwendung der Apache Hadoop Distribution unter Apache License 2.0.

⁸ Diese Kostenbewertung kann lediglich eine Tendenz aufzeigen, da zahlreiche Einflussfaktoren wie Preisschwankungen, Inflation, Risiko, Wert der Hardware oder steuerliche Aspekte in keiner Weise berücksichtigt wurden!

etwa wenn das Modell zur Objekterkennung mit Hilfe der erfassten Bilder nachtrainiert werden sollte, so wäre angesichts der Datenmenge eine Lambda-Architektur mit ihrer Möglichkeit zur tendenziell günstigeren Batchverarbeitung neu zu prüfen.

Softwareauswahl

Die Diversität der Daten⁹ legt eine NoSQL-Umgebung nahe. Präferiert der Kunde eine Cloud-Lösung, so sollten aus Effizienzgründen die beim jeweiligen Anbieter vorhandenen Services für Streaming und Processing verwendet werden. Solange die Komplexität der Architektur ein gewisses Maß nicht überschreitet, schätzen wir die damit verbundene Gefahr eines Vendor-Lock-In als gering, da die zu nutzenden Services funktional bei allen konkurrierenden Cloud-Anbietern vorhanden sind. Für die On-Premise-Variante, die aus Datenschutzgründen auch für den PoC des Showcases präferiert wurde, ist der Einsatz der Kern-Komponenten Kafka, Hadoop und Spark aus dem Big-Data-Ökosystem der Apache Foundation eine solide Wahl. Zum einen ist die Weiterentwicklung der Software durch die im Apache-Projekt engagierten Unternehmen über mehrere Jahre gesichert (s. FA3), zum anderen erleichtert ihre Verbreitung den Aufbau oder die Anwerbung von entsprechenden Experten (s. FA7). Zudem sind der hohe Reifegrad und die Flexibilität der Plattform (s. FA1) durch den Einsatz in vielen unterschiedlichen Unternehmen belegt. Falls gewünscht, bieten zahlreiche kommerzielle Distributionen Unternehmenssupport (s. FA7).

Dennoch wäre es sinnvoll, alternative Softwareprodukte zu evaluieren. Insbesondere sollte überprüft werden, ob ein Message Queue System wie Apache ActiveMQ oder RabbitMQ die Anforderungen besser abdecken könnte, als das Publisher/Subscriber Framework Apache Kafka. Auch der Einsatz von Apache Storm als Alternative zu Apache Spark sollte in Betracht gezogen werden. Im Rahmen dieser Arbeit konnten diese Evaluationen jedoch nicht durchgeführt werden.

⁹ Im PoC werden Bilddaten, Event-Logs sowie tabellarische Daten verarbeitet, in einer Erweiterung könnten zusätzlich Videomaterial für lückenlose Erkennung und Textinformationen für die Generierung der Beschreibungen genutzt werden.

Beschreibung der Referenzarchitektur

Wie bereits vorwiegend, basiert die für den PoC präferierte Architektur auf dem verbreiteten Schichtmodell der Kappa-Architektur (s. auch Anhang 4). Abbildung 3 zeigt die Softwarekomponenten, die in den verschiedenen Layern zum Einsatz kommen.

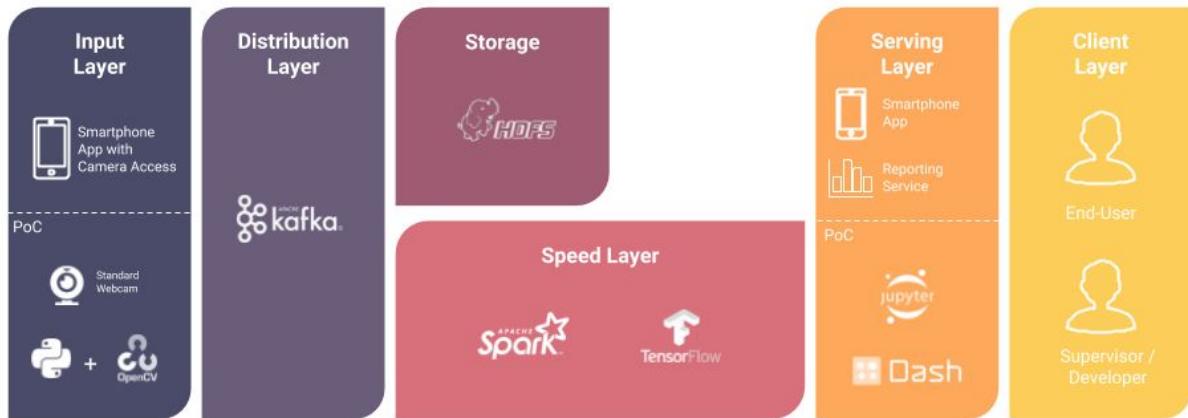


Abbildung 3: Component View

Die Erfassung des Bildmaterials (Input Layer) soll über eine App auf dem Smartphone des Endnutzer erfolgen. Im PoC wird dies durch ein Python-Programm simuliert, das mithilfe der OpenCV Bibliothek eine angeschlossene Webcam ausliest. Als Streaming Plattform (Distribution Layer) wird Apache Kafka eingesetzt, das Processing (Speed Layer) geschieht mit Apache Spark. Die Objekterkennung kann hierbei mit TensorFlow erfolgen, das sich in einer verteilten Spark-Umgebung einsetzen lässt. Ob dies die optimale Lösung ist, müsste genauer evaluiert werden. Zur Persistierung der Daten (Storage) ist HDFS vorgesehen, auf das im PoC jedoch verzichtet wurde. Nach der Verarbeitung werden die Ergebnisse ausgeliefert (Serving Layer): Individuelle Ergebnisse werden zurück an die App des Endnutzers gesendet, in der das jeweilige Bildmaterial erfasst wurde. Ein nicht näher spezifizierter Reporting Service dient der aggregierten Sicht auf die Nutzung des Dienstes und seine Erkennungsleistung. Im PoC wird dieser durch ein auf Python und Plotly Dash basierendes Dashboard simuliert.¹⁰

¹⁰ Die Verteilung der genannten Komponenten auf physischen Maschinen ist in Anhang 2 beschrieben.

Der Informationsfluss zwischen den Komponenten ist in einem zweiten Diagramm (Abb. 4) beschrieben.

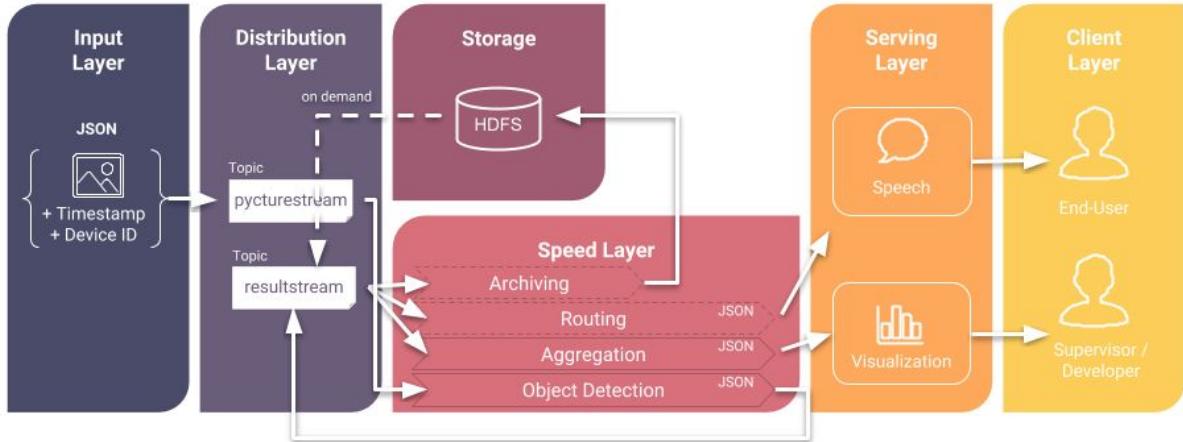


Abbildung 4: Information Flow

Das vom Kunden aufgenommene binäre Bildmaterial wird zunächst serialisiert und mit Metadaten in ein JSON-Objekt transformiert (Input Layer). Diese Vorgehensweise erleichtert die Verarbeitung im Stream, da alle zusammengehörigen Informationen im selben JSON-Objekt übermittelt werden. Jedoch bedeutet die für JSON nötige Serialisierung der Binärdaten einen erhöhten Speicherbedarf, der sich auf die Bandbreite im gesamten System auswirkt. Die produzierten JSON-Objekte werden in das Kafka-Topic „pycturestream“ geschrieben (Distribution Layer) und dort vom Objekterkennungs-Service abgeholt und prozessiert (Speed Layer). Die Ergebnisse der Objekterkennung werden in ein zweites Kafka-Topic „resultstream“ geschrieben. Dieses Topic ist von verschiedenen Services abonniert: Ein Archivierungs-Service schreibt die Ergebnisse in das HDFS, ein Aggregations-Service bereitet die Ergebnisse auf und übermittelt sie dem Reporting-Service zur Visualisierung. Ein Routing-Service sorgt zusätzlich für die Übermittlung an den Endkunden, dem die erkannten Objekte per Sprachausgabe vorgelesen werden.

Im PoC wurde auf den Archivierungs-Service komplett verzichtet. Auch der Routing-Service wurde eingespart, stattdessen wird die Anwendung des Endkunden durch ein weiteres Python-Programm simuliert, das unmittelbar das Topic „resultstream“ abonniert.

Proof of Concept

Vereinfachung der Architektur

Die Einrichtung der vollständigen vorgeschlagenen Architektur ist im zur Verfügung stehenden Zeitrahmen nicht leistbar. Daher wurde die Architektur für den PoC vereinfacht, um trotz der Komplexität einen einfachen Use-Case End-to-End abbilden zu können. Im Wesentlichen beinhaltet dies die Reduzierung des Kafka-Clusters und Hadoop-Clusters auf Single-Node Instanzen, die in einer VM betrieben werden. Es sei jedoch darauf hingewiesen, dass trotz der Vereinfachung alle funktionalen Bestandteile der vorgeschlagenen Architektur vorhanden sind.

Einrichtung der Infrastruktur

Voraussetzung für den Betrieb der PoC-Implementierung ist ein Computer mit Internetanschluss, eingerichtetem 64-bit Betriebssystem und ausreichenden Systemressourcen für den Betrieb einer VM. Wir empfehlen mindestens 8 GB Arbeitsspeicher¹¹, 15 GB freien Festplattenspeicher und einen 4-Kern-Prozessor.

Für die VM wird Oracle VirtualBox genutzt, das vom Hersteller mit Installationshinweise bereitgestellt wird (Oracle 2018). Die von Cloudera bereitgestellte Quick-Start-Appliance CDH 5.12 (Cloudera 2018) bildet die Basis der VM. Nach deren Installation wird das virtuelle Netzwerk konfiguriert. Die hierfür nötigen Schritte sind in der Datei /README.md im Abschnitt „Install Cloudera Quickstart in VirtualBox“ detailliert beschrieben (Büch 2018).

Setup der Software-Komponenten

Um die Reproduzierbarkeit der Umgebung zu verbessern und die Entwicklung im Team zu vereinfachen, ist ein möglichst hoher Automatisierungsgrad des Setups sinnvoll. Deshalb wurde das Installations-Skript /setup_cloudera_vm.sh erstellt und dessen Nutzung sowie weitere nötige Schritte in /README.md im Abschnitt „Setup Software Components“ beschrieben (Büch 2018).

¹¹ Für die Appliance werden 4 GB als Mindestgröße angegeben (Cloudera 2018), für unseren Anwendungsfall mit zusätzlicher Kafka-Installation ist das zu wenig.

Das Installations-Skript ist ausführlich kommentiert und begründet, hier seien nur die wesentlichen Schritte aufgezählt:

- Installation & Konfiguration von Apache Kafka
- Anlegen der Kafka-Topics „pycuturestream“ und „resultstream“
- Installation von Anaconda mit Python und zusätzlichen Bibliotheken
- Installation des JupyterLab¹² als Entwicklungsumgebung
- Bereitstellung des für TensorFlow benötigten Modells (s. folgendes Kapitel)

Implementierung¹³

Video Capturing (Client)

Die Aufgabe dieser Komponente ist die frequentielle Aufnahme von Einzelbildern mit der an das Host-System angeschlossenen Kamera, die Serialisierung des Bildmaterials und seine Übersendung mit zugehörigen Metadaten an einen Kafka-Endpoint.

Dieser Kafka-Producer wurde als Python-Script implementiert, das auf dem Host-System läuft. Für den Zugriff auf die Kamera und die Konvertierung des Bildes in das komprimierende JPEG-Format wird OpenCV verwendet, eine bekannte Bibliothek für Bildverarbeitung. Metadaten, die neben dem Bild an Kafka gesendet werden, sind ein Zeitstempel vom Aufnahmedatum und eine konfigurierbare ID, welche die Producer-Instanz bzw. die von ihr genutzte Kamera identifizierbar macht.

Als Austauschformat wählten wir JSON. Es erlaubt zum einen, Bild und Metadaten in einem Event kombiniert zu verschricken, zum anderen vereinfacht die weite Verbreitung des Formats die Wiederverwendung des Codes, auch außerhalb des Kafka-Kontexts. Allerdings erfordert das Format einen zusätzlichen Verarbeitungsschritt der Bilddaten: Das binäre JPEG-Format muss in einen JSON-konformen String kodiert werden. Hierfür wurde der ebenfalls gängige Base64-encoder genutzt. Für die Übermittlung des JSON-Objekts an Kafka wird die Bibliothek `kafka-python` verwendet, die sofort problemlos funktionierte.

Für Debugging bzw. Testing wird zusätzlich das jeweils aktuellste Bild im lokalen Dateisystem gespeichert. Mit der HTML-Datei `/client/monitor_imagestream.html` kann so der Bilder-Stream beobachtet werden.

¹² JupyterLab ist der kürzlich freigegebene Nachfolger des Jupyter Notebooks (Project Jupyter 2018). Es wird noch nicht für produktive Szenarien empfohlen, wir wollten es uns aber nicht nehmen lassen, die Software im Rahmen dieses Projektes auszuprobieren.

¹³ Der Betrieb der PoC-Implementierung wurde exemplarisch als Bildschirmvideo festgehalten und im Repository zur Verfügung gestellt (Büch 2018). Es sei ergänzend zur Lektüre dieses Kapitel empfohlen.

Objekterkennung (Processing)

Für die Implementierung des Stream-Processing wählten wir die Spark Streaming Bibliothek. In diesem verteilten In-Memory-System wird das zur Objekterkennung benötigte Deep-Learning-Model zu Beginn der Verarbeitung einmalig auf alle Knoten des Cluster geladen. Anschließend werden die einzelnen Events im Cluster verteilt und mit TensorFlow analysiert. Die Analyseergebnisse werden in das Kafka-Topic „resultstream“ geschrieben.

Für die Implementierung der Objekterkennung im Rahmen des PoC wurde TensorFlow in Verbindung mit bereits trainierten Modellen (Frozen Network Models) in Kombination mit Spark Streaming verwendet. Dabei wird im Wesentlichen die sich in der Entwicklung befindenden Object Detection API von TensorFlow (2018a) genutzt. Die dortigen Beispielcodes, Bibliotheken und Dokumentationen thematisieren jedoch nicht die Nutzung auf verteilten Systemen. Die Nutzung von TensorFlow mit Spark behandelt dafür Hunter (2016), allerdings nicht im Kontext von Objekterkennung und ohne Streaming. Die Verwendung von Spark Streaming in Kombination mit Kafka demonstriert hingegen Moffatt (2017). Unser Prototyp vereint also die Erkenntnisse aus den genannten Quellen.

Für die Auswahl des Modells wurde von uns ein Subset der vortrainierten Modelle auf ihre Performance bezüglich Geschwindigkeit und Erkennung getestet. Zwar sind bereits entsprechende Kennzahlen auf der Übersichtsseite verfügbarer Modelle veröffentlicht (TensorFlow 2018b), aber es war uns wichtig, die Eignung an eigenem Bildmaterial und mit unserer Hardware, zu überprüfen. Dazu führten wir das Notebook /notebooks/object_detection_model_evaluation.ipynb (Büch 2018) auf der VM mit jeweils drei Test-Bildern aus. Anschließend wurde das annotierte Bildmaterial, die Analysedauer auf unserem PoC-System und die subjektive Bewertung der Erkennungsqualität miteinander verglichen (s. Anhang 5).

Anhand der Ergebnisse wurde deutlich, welche Abwägung zu treffen war: Die Netzwerke, die mit dem Datensatz Common Object in Context (COCO) trainiert wurden, arbeiten deutlich schneller. Allerdings sind hier lediglich 90 verschiedene Labels trainiert, es können also maximal 90 verschiedene Objektkategorien erkannt werden. Das mit dem Open Image Dataset (OID) trainierte Netzwerk verfügt über 450 Objektkategorien, arbeitet jedoch deutlich langsamer. Auch ist die Erkennung von Details und Personen auf unseren Testbildern bei diesem Modell weniger erfolgreich.

Für den Produktiveinsatz ist eine hohe Anzahl von Objektkategorien wünschenswert, 90 Kategorien erscheinen nicht ausreichend. Eine sinnvolle Geschwindigkeit sollte durch entsprechende Skalierung gesichert werden. Für den PoC wählten wir dennoch das schnellste COCO-basierte Modell, um eine höhere Bildfrequenz zu erreichen, wodurch wir näher an der Idee des Streamings blieben.

Ausgabe der Ergebnisse (Client)

Diese Komponente ist ein Kafka-Consumer des Topics „resultstream“ und demonstriert die Weiterleitung der Ergebnisse an den Client, der das Bildmaterial lieferte. Im PoC empfängt dieser Consumer die Messages *aller* Clients und filtert lediglich die Ergebnisse auf Basis der mitgesendete `device_id`. Produktiv wäre hier zwingend eine sichere Authentifizierungsmethode notwendig, auf deren Grundlage nur die Messages des jeweiligen Clients zugänglich gemacht werden sollten.

Die Ergebnisse werden zum einen als Konsolenausgabe dargestellt. Zum anderen werden sie in natürlicher Sprache aufbereitet und mittels Text-To-Speech vorgelesen, etwa in folgender Form:

„Person at middle right. Two Books at top right. Bottle at middle center.“

Zusätzlich wird das aktuellste annotierte Bild für Demonstrations- und Debugging-Zwecke als JPEG gespeichert und kann ebenfalls in `/client/monitor_imagestream.html` angezeigt werden.

Dashboard (Reporting)

Um die Entwicklung des Dashboards und der Processing-Pipeline parallel starten zu können, wurde bereits zu Projektbeginn ein einfaches Datenmodell im JSON-Format entworfen. Auf Basis dieser JSON-Datei wurde mit Python und Plotly Dash¹⁴ ein exemplarisches Dashboard implementiert, das die Anzeige der Analyseergebnisse in aggregierter Form demonstriert. Produktiv sollten hierin Live-Daten angezeigt werden, im PoC wird lediglich der letzte Stand der JSON-Datei als Datengrundlage genommen.

Diese JSON-Datei wird von einem zweiten Kafka-Consumer des Topics „resultstream“ generiert, der die Ergebnisdaten lediglich in die zuvor festgelegte Struktur transformiert und in die JSON-Datei schreibt.

¹⁴ Plotly Dash ist ein Python Framework unter MIT License zur Darstellung von Charts in Form interaktiver Single-Page-Webseiten (Plotly 2018).

Fazit

Das Projekt konnte wie geplant konzipiert und End-to-end umgesetzt werden, jedoch nicht ohne zahlreiche Schwierigkeiten zu überwinden.

Eine bewusst gewählte Herausforderung war die Konfiguration der PoC-Umgebung: die Sicherstellung der Kompatibilität zwischen den Komponenten Kafka und Spark, sowie Spark und Tensorflow durch Wahl der geeigneten Versionen und entsprechenden Einstellungen hat viel Zeit in Anspruch genommen, aber deutlich zum besseren Verständnis der Komponenten und ihrem Zusammenspiel beigetragen.

Bei der Implementierung der Analytics-Pipeline wurde deutlich, wie stark sich die Herangehensweise mit Stream-Processing von der uns eher geläufigen Batch-Verarbeitung unterscheidet. Vermutlich wären hier Kenntnisse entsprechender Programmier-Paradigmen und mehr praktische Erfahrung hilfreich. Angesichts der wachsenden Bedeutung von Real-Time-Analytics ist dies sicher ein lohnendes Forschungs- und Lernfeld.

Etwas getrübt wurde das Endergebnis durch die Geschwindigkeitseinbußen in der PoC- Umgebung: Der Overhead durch den Betrieb der VM, des Single-Node Kafka Clusters und des Single-Node Spark Clusters auf einem einzelnen Notebook war zwar erwartet, fiel jedoch deutlich höher aus, als befürchtet. Die erreichte Geschwindigkeit von einem Bild in etwa 9 Sekunden war etwas enttäuschend. Der Gedanke daran, dass die Architektur jedoch prinzipiell skaliert, sodass die gewünschte Performance auf geeigneter Infrastruktur erzielt werden könnte, sorgte für einen gewissen Trost.

Die Hauptschwierigkeit bei den verschiedenen zu treffenden Architekturentscheidungen war die Wahl zwischen einer On-Premise und einer Cloud-Lösung, besonders aufgrund der schwer einzuschätzenden Leistungsanforderungen und den somit entstehenden Kosten. Das Argument „Datenschutz und -sicherheit“ mag zwar für viele Unternehmen und Use-Cases ausschlaggebend sein, dennoch wäre es als Fortsetzung des Projekts sicherlich lohnenswert, die komplette Architektur des PoC noch einmal in einer Cloud-Umgebung umzusetzen, um Aufwand, Leistung und Kosten adäquat mit der On-Premise-Variante vergleichen zu können.

Quellenverzeichnis

Aichele C. (2006): Intelligentes Projektmanagement. Stuttgart: Kohlhammer Verlag.

Be My Eyes (2018): Be My Eyes – Hilfe für Blinde und Sehbehinderte. Android App im Google Play Store. URL: <https://play.google.com/store/apps/details?id=com.bemyeyes.bemyeyes>. Zugriff: 25.03.2018.

Büch H., *alias* dynobo (2018). PyctureStream – PoC for image processing using Kafka, Spark Streaming & TensorFlow. GitHub repository. URL: <https://github.com/dynobo/PyctureStream>. Zugriff: 26.02.2018.

BMI – Business Models Inc. (2018): The Business Model Canvas – Tool to help you understand a business model. URL: <https://www.businessmodelsinc.com/tools-skills/tools/business-model-canvas>. Zugriff: 25.03.2018.

Cloudera Inc. (2018). QuickStarts for CDH 5.12 – Virtualized clusters for easy installation on your desktop. URL: https://www.cloudera.com/downloads/quickstart_vms/5-12.html. Zugriff: 26.02.2018.

Dell Inc. (2017). Solution Overview – Dell EMC Ready Bundles for Hadoop. URL: <https://www.emc.com/collateral/solution-overview/ready-bundles-for-hadoop-solution-overview.pdf>. Zugriff: 13.03.2018.

Google LLC (2018): Cloud Vision API – Quotas and Limits. URL: <https://cloud.google.com/vision/quotas>. Zugriff: 22.03.2018.

Hunter T. (2016): Deep Learning with Apache Spark and TensorFlow. URL: <https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html>. Zugriff: 28.02.2018.

LeaseWeb (2018): Private Rack Colocation – Set your business free with powerful private racks. URL: <https://www.leaseweb.com/colocation/private-rack>. Zugriff: 20.03.2018

Pathirage M. (2017): Kappa Architecture – Where everything is a stream. URL: <http://www.kappa-architecture.com>. Zugriff: 23.03.2018.

Mallot H. A. (2006): Visuelle Wahrnehmung. In J. Funke & P. A. Frensch (Hrsg.): Handbuch der Allgemeinen Psychologie – Kognition. Göttingen: Hogrefe Verlag.

Moffat R. (2017): Getting Started with Spark Streaming, Python and Kafka. URL: <https://www.rittmannmead.com/blog/2017/01/getting-started-with-spark-streaming-with-python-and-kafka/>. Zugriff: 05.03.2018.

Plotly Inc. (2018): Dash by Plotly – Build beautiful web-based interfaces in Python. URL: <https://plot.ly/products/dash/>. Zugriff: 20.03.2018.

Project Jupyter (2018, 20. Februar): JupyterLab is Ready for Users. *Jupyter Blog – All the latest about Project Jupyter*. URL: <https://blog.jupyter.org/jupyterlab-is-ready-for-users-5a6f039b8906>. Zugriff: 25.02.2018.

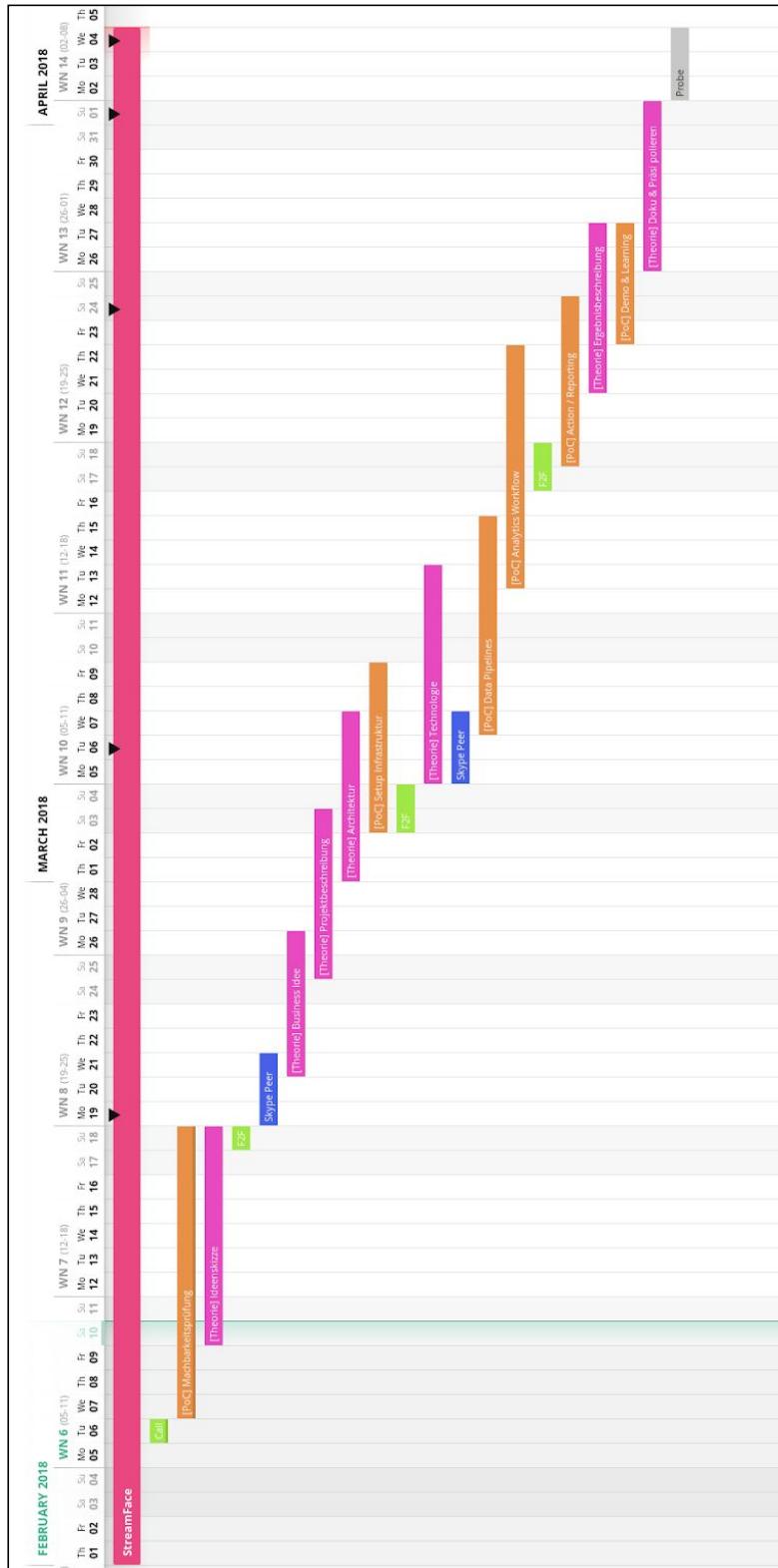
Oracle Corp. (2018). Oracle VM VirtualBox. URL: <https://www.virtualbox.org>. Zugriff: 26.02.2018.

TensorFlow (2018a): Tensorflow Object Detection API. GitHub repository. URL: https://github.com/tensorflow/models/tree/master/research/object_detection. Zugriff: 26.02.2018.

TensorFlow (2018b): Tensorflow detection model zoo. GitHub repository. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Zugriff: 26.02.2018.

Anhang

1.) Projektplanung - GANTT Chart



2.) Projektideen

Projekttitel	Idee	Beschreibung
Assistenz-App für Blinde	Eine Idee ist, durch eine Video- / Bildanalyse den blinden Menschen automatisiert die Umgebung zu beschreiben und damit den Menschen einen zusätzlichen Zugang zu ihrer Umgebung zu ermöglichen.	Das Projekt bedarf einer Architektur, die innerhalb Sekunden ein Bild analysiert und das Ergebnis dem User ausgibt. Der benötigte Algorithmus müsste Open Source sein, da einen eigenen Algorithmus zu trainieren dem Projektrahmen überschreitet. Die Videoaufnahmen können selbst generiert werden.
Stiller Alarm	In Banken, Tankstellen etc. werden die Angestellten überfallen. Dabei haben die Angestellte manchmal keine Möglichkeit, die Polizei zu alarmieren. Die Idee ist, automatisch einen Überfall zu erkennen: Wenn die Bildanalyse eine Waffe erkennt, könnte ein stiller Alarm automatisiert abgesendet werden.	Das Projekt bedarf einer Architektur, die in wenigen Sekunden einen Raubüberfall erkennt. Es müsste eigenständig ein Algorithmus trainiert werden, der Waffen (Großkaliber, Kleinkaliber, Messer, etc.) automatisch erkennt. Die Datengrundlage könnten z. B. Videos von Youtube sein. Label müssen manuell vergeben werden.
Aufsichtspflicht	Dieser Projektvorschlag deckt Fälle ab, in dem Menschen ein großes Gebiet überwachen und schnell handeln müssen. Als Praxisbeispiel könnte ein Algorithmus einen Bademeister unterstützen, indem eine Person in Notlage automatisch erkannt wird. Ein anderes Beispiel wäre, im Stadion oder auf dem Schulhof eine Schlägerei zu prognostizieren.	Das Projekt bedarf einer Architektur, die schnell eine bevorstehende Schlägerei oder Tumulte prognostiziert. In diesem Fall müsste der Algorithmus einige Parameter berücksichtigen. Auf einem Bild einen Gegenstand zu erkennen, reicht nicht aus. Mehrere Bilder mit den Bewegungen müssten zu einer Entscheidung des Algorithmus führen. Die Datengrundlage könnte über Youtube Videos und manuelle Annotierung generiert werden.
Überwachung Gefahrenbereich	In einer Produktionsumgebung wie einer Fabrik gibt es häufig Gefahrenbereiche, etwa im Einflussbereich großer Maschinen. Die ist, solche Bereiche per Video zu überwachen und beim Eindringen einer Person in den Gefahrenbereich eine Warnung oder ggf. eine Notabschaltung auszulösen.	Die Architektur muss innerhalb weniger Sekunden mit großer Sicherheit Personen detektieren können. Hierzu könnten frei verfügbare Modelle genutzt werden. Möchte man eine Notabschaltung simulieren, wäre ein Demo-Aufbau einer Maschine mit Gefahrenbereich möglich.

3.) Herleitung des Leistungsbedarfs

Datendurchsatz

- Bilder pro Kamera und Stunde: $20 * 60 = 1.200$
- Bilder aller Kameras pro Tag: $1.200 * 720 = 864.000$
- Ein Bild entspricht einem Datenvolumen von 300 kB
- **Der Datendurchsatz pro Tag beträgt also rund 260 GB**

Rechenleistung

- Im PoC teilen sich Kafka und Spark eine VM mit 4 CPUs und 20 GB RAM.
- Im PoC erreichen wir eine Erkennungsleistung von 1 Bild in maximal 9 Sekunden.
- Um einen Durchsatz von 30 Bildern in 3 Sekunden erreichen zu können (s. FA4), muss das Processing um das 90-fache schneller sein.
- Laut Systemmonitor wurden für das Spark-Processing in der VM 2 CPUs beansprucht.
- **Die benötigte Rechenleistung sollte also durch 180 CPUs erreicht werden.**

Arbeitsspeicher

- Der RAM-Bedarf wird hier wegen seiner geringen Rolle nicht berücksichtigt: Die zu verarbeitenden Bilder werden nur einmalig in den Speicher geladen und dort nicht lange vorgehalten.
- Auch in unseren Tests wurde nur ein Bruchteil des standardmäßig verfügbaren RAM genutzt.

4.) Weitere Architektur-Diagramme

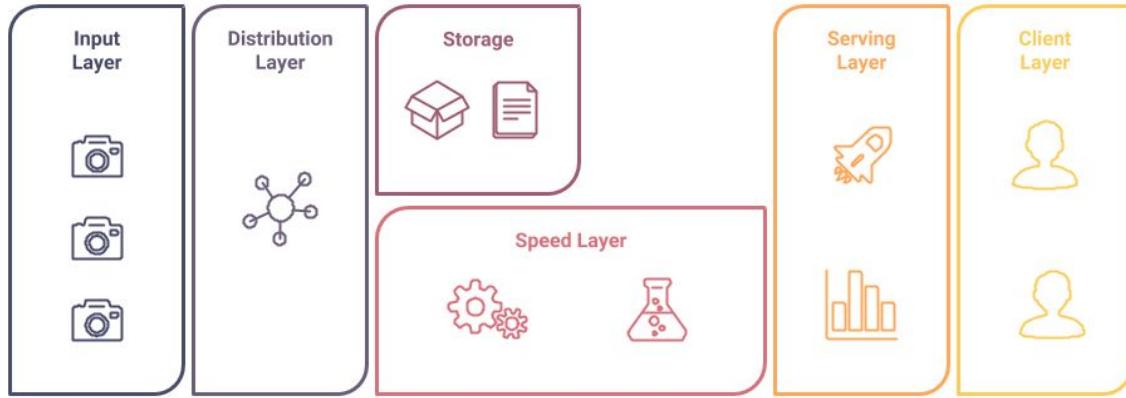


Abbildung 5: Layer View

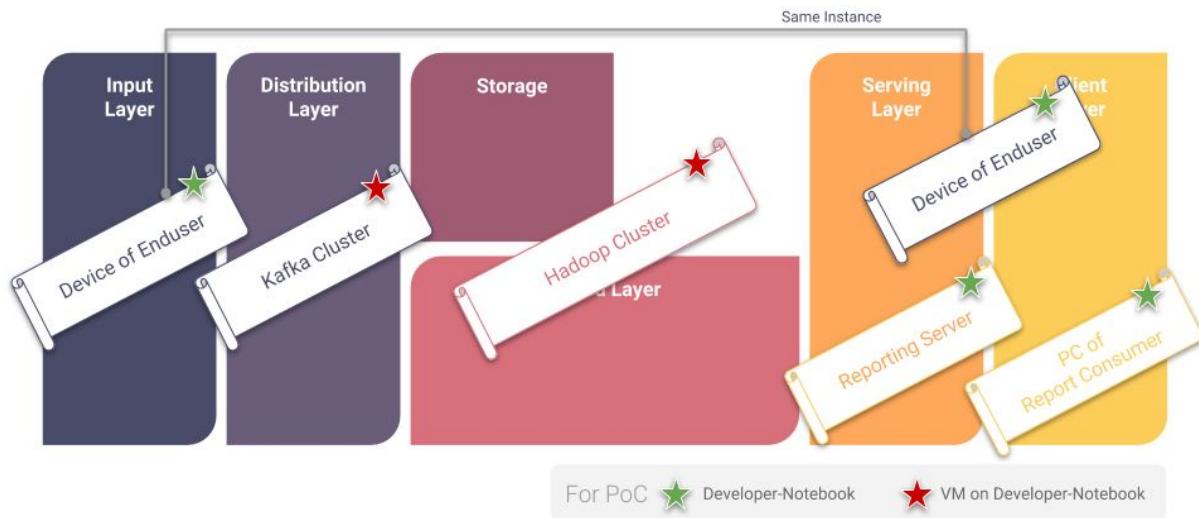
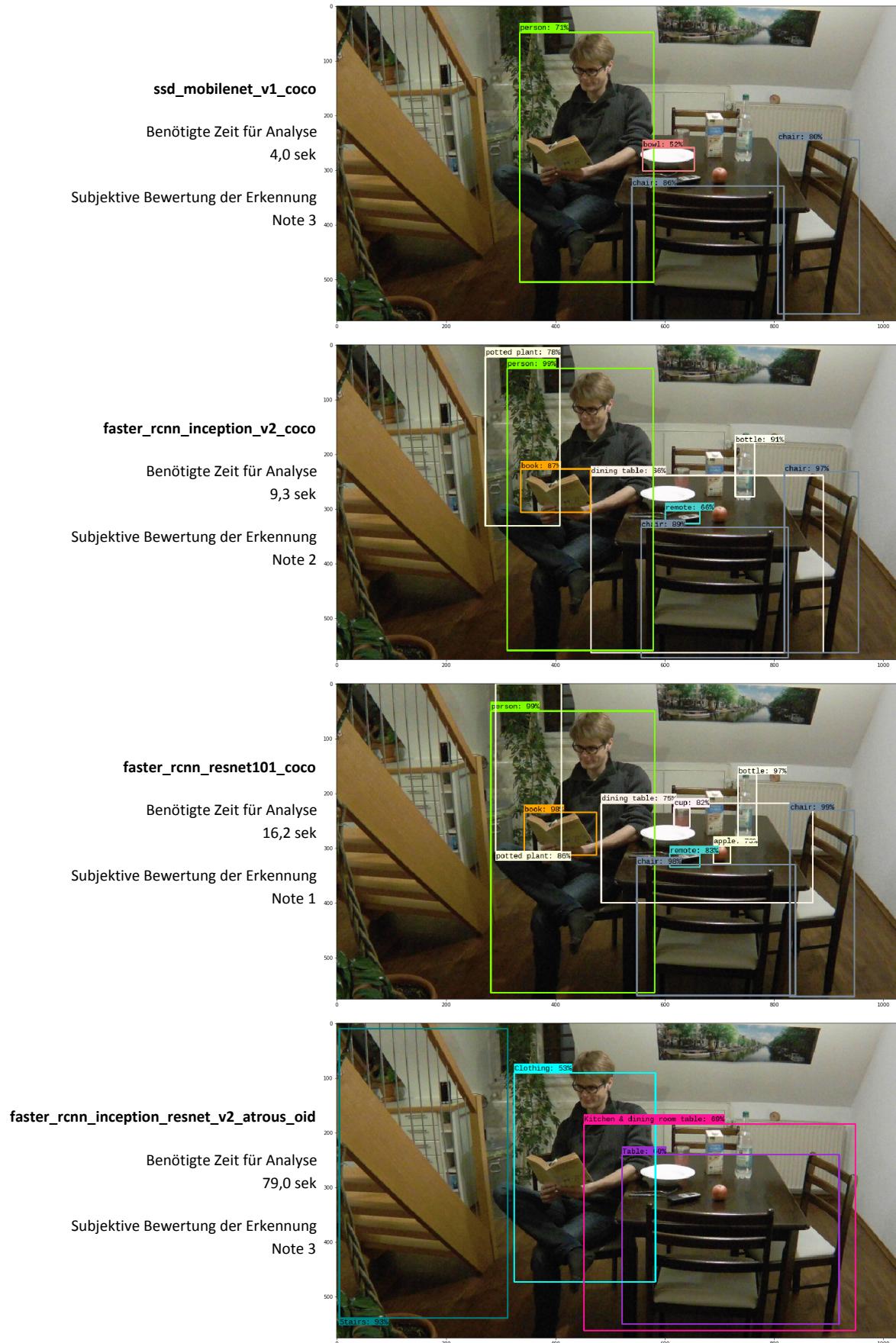


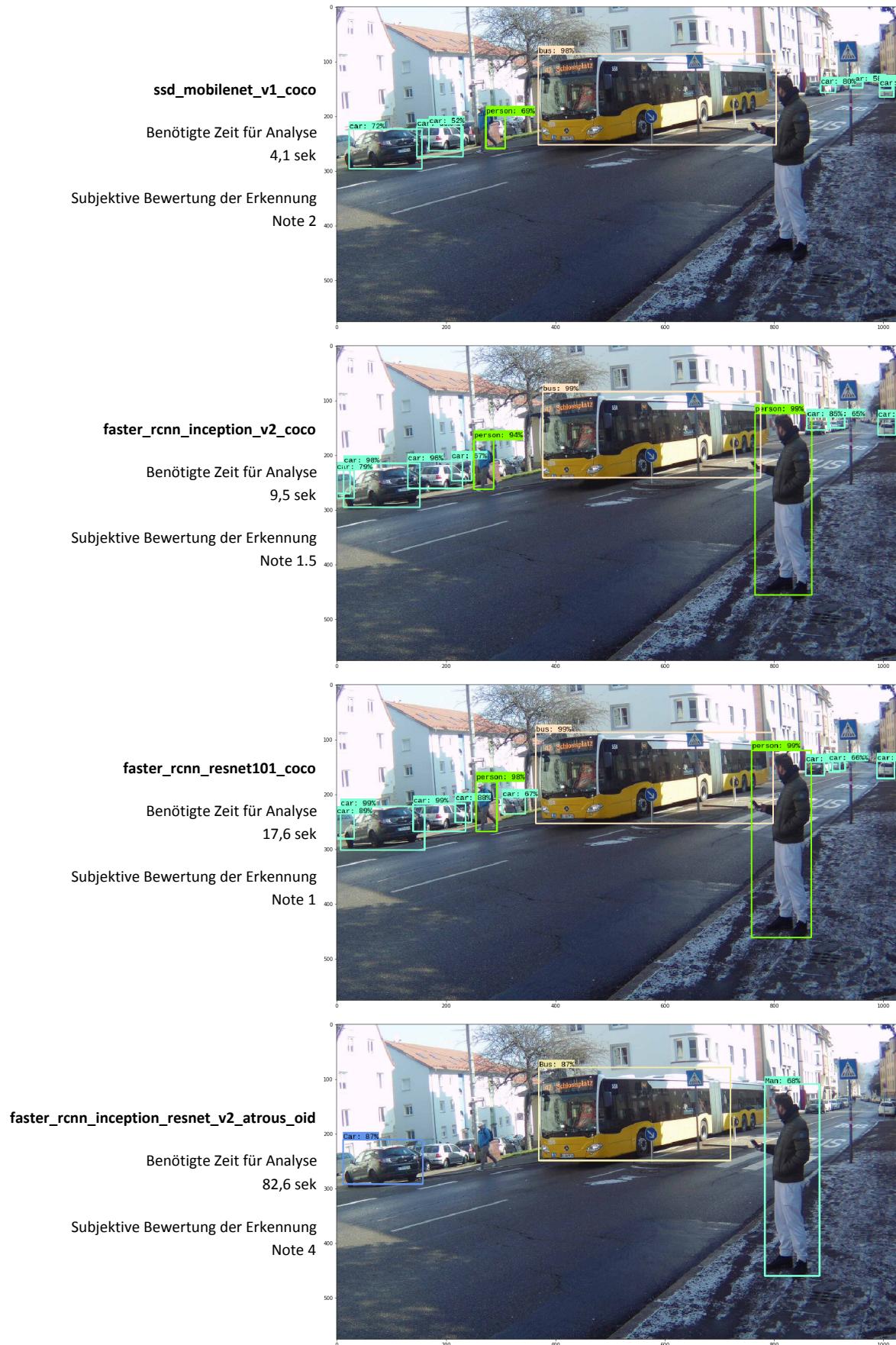
Abbildung 6: Deployment View

Das Schichtmodell (Abb. 5) stellt die verschiedenen Layer der Kappa-Architektur dar, welche bereits ausführlich beschrieben wurde (s. Referenzen von Pathirage 2018). Der Deployment View (Abb. 6) zeigt die Verteilung der Architektur auf die physischen Komponenten: Den Smartphones der Nutzer für Ein- und Ausgabe von Daten, einem Cluster für die Stream-Verteilung, einem Cluster für die Datenanalyse und ggf. Speicherung, sowie den Server für das Reporting, der über Mitarbeiter-PCs zugänglich gemacht wird.

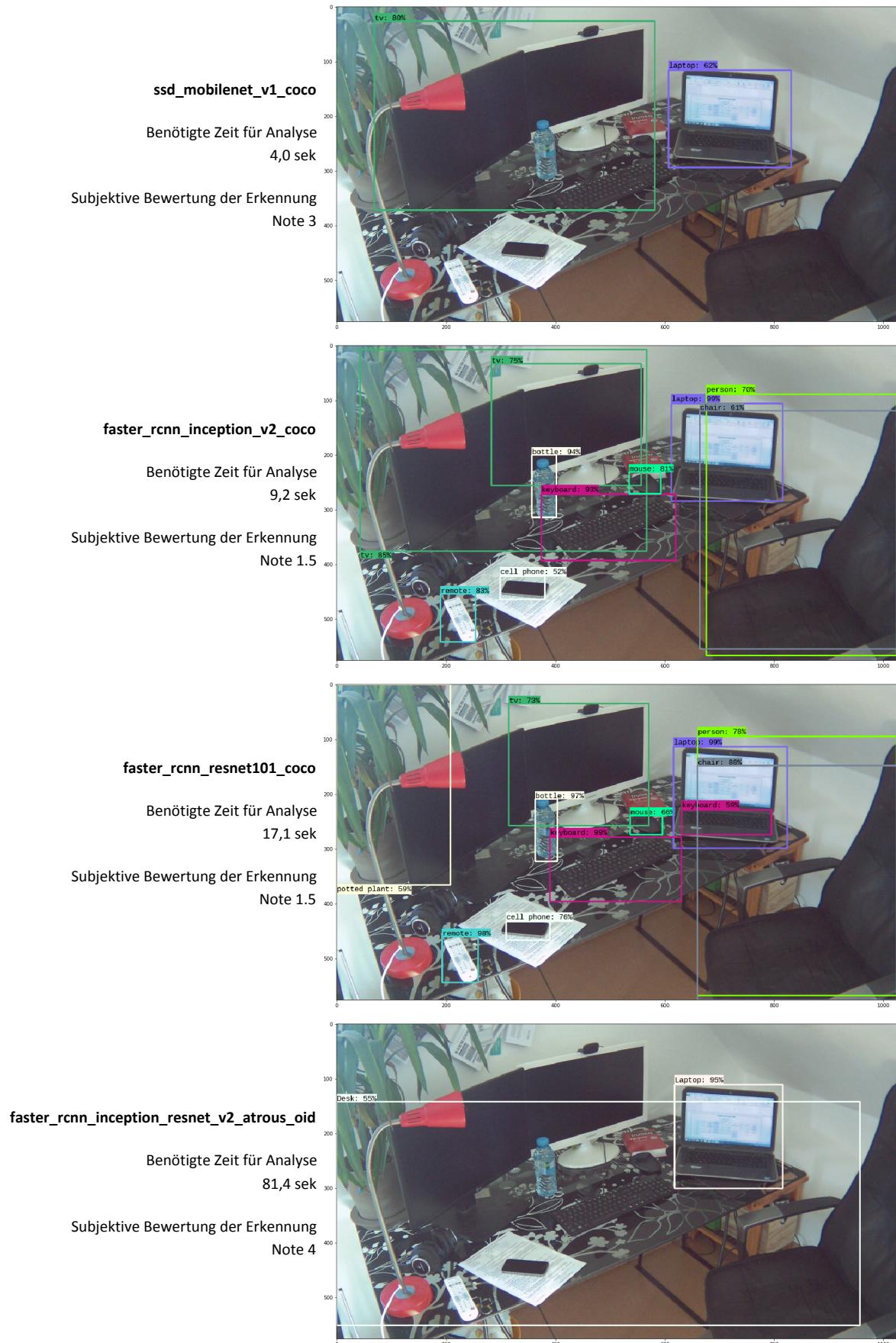
Die Legende beschreibt die für den PoC vorgenommenen Vereinfachungen: Alle Komponenten werden auf einem Entwickler-Notebook betrieben, die beiden Cluster als Single-Node Instanzen in einer hierfür eingerichteten VM.

5.) Modell-Evaluation mit Testbildern









6.) Beispielkonfiguration für Google Cloud Plattform Konfiguration

Stream Processing: Cloud Dataflow Reporting: Compute Engine + Persistent Disk		
Analytics: Vision API	Analytics: Machine Learning	Analytics: Cloud Dataproc
<p>Die „Vision API“ ist aufgrund der Erkennungsqualität und dem Preis attraktiv.¹⁵ Allerdings würden die Quota-Limits von 600 Requests per Minute hart touchiert (Google 2018).</p>	<p>Für Stream-Predictions stehen nur single-core Maschinen ohne GPU zur Verfügung, eine Erkennung innerhalb der geforderten zwei Sekunden ist daher fragwürdig.</p>	<p>Diese Variante basiert unter anderem auf Apache Spark und ist am ehesten mit der On-Premise- Variante vergleichbar.</p>
<p>Compute Engine</p> <p>1 x </p> <p>730 total hours per month</p> <p>VM class: regular</p> <p>Instance type: n1-standard-4</p> <p>Region: Frankfurt</p> <p><u>Sustained Use Discount:</u> 30% </p> <p><u>Effective Hourly Rate:</u> \$0.171</p> <p>Estimated Component Cost: \$125.09 per 1 month</p>	<p>Compute Engine</p> <p>1 x </p> <p>730 total hours per month</p> <p>VM class: regular</p> <p>Instance type: n1-standard-4</p> <p>Region: Frankfurt</p> <p><u>Sustained Use Discount:</u> 30% </p> <p><u>Effective Hourly Rate:</u> \$0.171</p> <p>Estimated Component Cost: \$125.09 per 1 month</p>	<p>Compute Engine</p> <p>1 x </p> <p>730 total hours per month</p> <p>VM class: regular</p> <p>Instance type: n1-standard-4</p> <p>Region: Frankfurt</p> <p><u>Sustained Use Discount:</u> 30% </p> <p><u>Effective Hourly Rate:</u> \$0.171</p> <p>Estimated Component Cost: \$125.09 per 1 month</p>
<p>Persistent Disk</p> <p>Frankfurt </p> <p>Storage: 200 GB</p> <p>\$9.60</p>	<p>Persistent Disk</p> <p>Frankfurt </p> <p>Storage: 200 GB</p> <p>\$9.60</p>	<p>Persistent Disk</p> <p>Frankfurt </p> <p>Storage: 200 GB</p> <p>\$9.60</p>
<p>Cloud Dataflow</p> <p> </p> <p>3 x n1-standard-4 workers in Streaming Mode</p> <p>Region: Europe</p> <p>Total vCPU Hours: 8,640</p> <p>Total Memory Hours: 32,400 GB/Hours</p> <p>PD Local Storage: 64,800 GB/hours</p> <p>\$760.75</p>	<p>Cloud Dataflow</p> <p> </p> <p>3 x n1-standard-4 workers in Streaming Mode</p> <p>Region: Europe</p> <p>Total vCPU Hours: 8,640</p> <p>Total Memory Hours: 32,400 GB/Hours</p> <p>PD Local Storage: 64,800 GB/hours</p> <p>\$760.75</p>	<p>Cloud Dataflow</p> <p> </p> <p>3 x n1-standard-4 workers in Streaming Mode</p> <p>Region: Europe</p> <p>Total vCPU Hours: 8,640</p> <p>Total Memory Hours: 32,400 GB/Hours</p> <p>PD Local Storage: 64,800 GB/hours</p> <p>\$760.75</p>
<p>Vision API </p> <p>Label Detection: 864,000</p> <p>OCR: 0</p> <p>Explicit Content Detection: 0</p> <p>Facial Detection: 0</p> <p>Landmark Detection: 0</p> <p>Logo Detection: 0</p> <p>Image Properties: 0</p> <p>Web Detection: 0</p> <p>Document Text Detection: 0</p> <p>\$1,294.50</p> <p>Total Estimated Cost: \$2,189.94 per 1 month</p>	<p>Cloud Machine Learning </p> <p>Region: United States</p> <p>ML Training Units: 2.473</p> <p>Job run time: 0 minutes</p> <p>Prediction Mode: online</p> <p># of Predictions: 2,591,000</p> <p>Total Inference time: 5,182,000 seconds</p> <p>\$431.83</p> <p>Total Estimated Cost: \$1,327.28 per 1 month</p>	<p>Cloud Dataproc </p> <p>Cluster size: 176 vCPUs</p> <p>Time used: 720 hours</p> <p>\$1,267.20</p> <p>Total Estimated Cost: \$2,162.64 per 1 month</p>

Quelle: <https://cloud.google.com/products/calculator/>

¹⁵ Der Service „Video Intelligence“ würde die Anforderungen unseres Use-Cases übertreffen und ist deutlich kostenintensiver.

7.) Beispielkonfiguration eines Dell PowerEdge R730xd Rack Server

Ermittelter Bedarf für Demo-Use-Case: **11 Server**

- 8 Server für Hadoop-Cluster (3 Name Nodes, 1 Edge Node, 4 Worker Nodes)
- 3 Server Kafka-Cluster (1 Zookeeper Node, 2 Broker Nodes)
- Das Reporting wird auf dem Edge Node betrieben

Dell Price \$4,187.45

Starting at Price \$6,690.00
Total Savings \$2,502.55
 Standard Delivery Free

Dell Business Credit
 As low as \$126/mo.^ | Apply
[View Delivery Dates](#)



Customize

Option	Selection	SKU / Product Code	Quantity
Base	PowerEdge R730xd Server	[210-ADBC][329-BC2K] / R73DX	1
Trusted Platform Module (TPM)	No Trusted Platform Module	[461-AADZ] / NTPM	1
Chassis	Chassis with up to 12, 3.5" Hard Drives	[350-BBEU] / 3512HD	1
Processor	Intel® Xeon® E5-2640 v4 2.4GHz,25M Cache,8.0GT/s QPI,Turbo,HT,10C/20T (90W) Max Mem 2133MHz	[338-BJDL] / 26404	1
Additional Processor	Intel® Xeon® E5-2640 v4 2.4GHz,25M Cache,8.0GT/s QPI,Turbo,HT,10C/20T (90W) Max Mem 2133MHz	[338-BJDN] / E5264V	1
Processor Thermal Configuration	2 CPU Standard	[370-ABWE][374-BBHM][374-BBHM] / 2CPU	1
Memory DIMM Type and Speed	2400MT/s RDIMMs	[370-ACPH] / 2400MT	1
Memory Configuration Type	Fault Resilient Memory-Vmware	[379-BBGK] / VMFRM	1
Memory	4GB RDIMM, 2400MT/s, Single Rank, x8 Data Width	[370-ACOG] / 4GBIMM	4
RAID	RAID 0 for H330/H730/H730P (1-24 HDDs or SSDs)	[780-BBLI] / R0H	1
RAID Controller	PERC H330 RAID Controller	[405-AAEF] / H330	1
Hard Drive	1TB 7.2K RPM SATA 6Gbps 3.5in Hot-plug Hard Drive	[400-AEZ] / 1TA35	1
Embedded Systems Management	iDRAC8 Express, integrated Dell Remote Access Controller, Express	[385-BBHN] / IBEXP	1
Internal SD Module	None		1
Rack Rails	No Rack Rails or Cable Management Arm	[770-BBBS] / NORAIL	1
Bezel	No Bezel	[350-BBBW] / NOBEZL	1
Power Management BIOS Settings	Power Saving Dell Active Power Controller	[750-AABF] / DAPC	1
Power Cords	NEMA 5-15P to C13 Wall Plug, 125 Volt, 15 AMP, 10 Feet (3m), Power Cord, North America	[450-AALV] / 125V10	1
Power Supply	Single, Hot-plug Power Supply (1+0), 495W	[450-ADWP] / 495NR	1
System Documentation	No Systems Documentation, No OpenManage DVD Kit	[631-AACK] / NODOCS	1
Operating System	No Operating System	[619-ABVR] / NOOS	1
OS Media Kits	No Media Required	[421-5736] / NOMED	1
OS Partitions	None		1
Virtualization Software	None		1
Enabled Virtualization	None		1
Database Software	None		1
Additional Software Offerings	None		1
Network Daughter Card	Broadcom 5720 QP 1Gb Network Daughter Card	[540-BBBW] / 5720QP	1
PCIe Riser	Risers with up to 1,x8 PCIe Slots + 2,x16 PCIe Slots	[330-BBCO][374-BBHT] / RSR12	1

Quelle: <http://www.dell.com/en-us/work/shop/povw/poweredge-r730xd>