Informatik

Geodatenverarbeitung mit Budget Instanzen

Semesterarbeit

Abbildung 1: Eine Tour mit Budget Instanzen [Web10].

Departement:	Informatik
Kurs:	CAS CLD FS20 – Cloud Computing
Autor:	Tobias Reber
Experte:	Jörg Thomann
Ansprechpartner:	Christoph Böcklin
Datum:	01. 09. 2020

Perplexity
is the beginning of knowledge.

- Kahlil Gibran

[AD19, S. 33]

1 Management Summary

Wie aus dem Titel der Arbeit *Geodatenverarbeitung mit Bugdet Instanzen* zu entnehmen ist, geht es bei dieser Semesterarbeit darum, Geodaten mittels Budget Instanzen zu verarbeiten: Geodaten sind Daten mit einem räumlichen Bezug, sie haben Koordinaten. Bugdet Instanzen sind günstige virtuelle Server vom Amazon AWS EC2 Angebot, sogenannte Spot Instanzen, die den Nachteil haben, dass sie innerhalb einer 2 minütigen Vorankündigung entfernt werden können.

Aus der Problemstellung geht hervor, dass die swisstopo die Publikationsvoränge von Geodaten fortlaufend optimieren möchte. Zurzeit erweist sich die Publikation von 3D Geodaten als besonders aufwändig, weil die manuelle Publikation spezielle Tools benötig, die auf performanten und somit teuren Rechnern laufen müssen. Ausserdem erfordert die Bereitstellungen einen hohen Koordinationsaufwand zwischen IT (Infrastruktur), Entwicklung und Auftraggeber. Mit dieser Arbeit wird mittels Prototyp aufgezeigt, wie der Automatisierungsgrad erhöht werden könnte und wie bei der Verarbeitung Budget Instanzen zum Einsatz kommen könnten.

Um den Automatisierungsgrad wesentlich zu erhöhen, wurde die Datenverarbeitung mittels Automatisierungswerkzeug Ansible in einem deklarativen Code beschrieben. Die Spot Instanzen können über eine sogenannte Flottenafrage gesteuert werden. Die Flottenanfrage wurde so konfiguriert, dass sie dafür sorgt, dass immer eine Spot Instanz mit vordefinierten Mindestanforderungen zur Verfügung gestellt werden soll. Sobald eine Spot Instanz hochgefahren wird, wird das Ansible Skript automatisch gestartet. Die einzelnen Verarbeitungsschritte werden auf einem separaten Laufwerk festgehalten. Falls eine Spot Instanz beendet werden sollte, kann die nächste von der Flottenanfrage automatisch gestartete Instanz die Verarbeitung beim letzten festgehaltenen Schritt fortführen.

Mit dem Prototypen konnte gezeigt werden, dass sich Spot Instanzen gut eignen, um Geodaten zu verarbeiten. Durch die Automatisierung würden Personalstunden wegfallen und die Fehleranfälligkeit kleiner werden. Beim Use Case wurden alle erfassten 3D Gebäude der Schweiz auf Spot Instanzen innerhalb von 18 Stunden verarbeitet. So konnte gezeigt werden, dass das relative Sparpotential beim Einsatz von Bugdet Instanzen gegenüber gewöhnlichen (On-Demand) Instanzen enorm ist, beim gewählten Use Case 76%. Jedoch in absoluten Zahlen macht die Einsparung gerade einmal 15$ aus. Dennoch sind Spot Instanzen aus wirtschaftlicher Sicht ein interessantes Angebot und ein vermehrter Einsatz in der Geodatenverarbeitung würde sich lohnen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

2 Einführung

Ich arbeite als GIS-Spezialist bei der swisstopo[1], dem Bundesamt für Landestopografie, in Wabern. Wir machen Karten. Unser Team macht Internetkarten - wie Google Maps[2], jedoch von der Schweiz für die Schweiz; und für alle anderen auch. Unsere Internetkarte, der Viewer, erfreut sich relativ grosser Beliebtheit und beinhaltet ca. 800 Themen wie Wanderwege, Solarkataster und Luftfahrthindernisse. Lieber Leser[3], falls dir map.geo.admin.ch noch kein Begriff sein sollte, kann ich dir wärmstens empfehlen, darin zu schmökern. Es gibt viel zu entdecken und es ist gratis - ein Service Public.

Abbildung 2: Internetkarte des Bundes *map.geo.admin.ch*. Hier ein Ausschnitt der Saane bei Kleinbösingen, das Luftaufnahmen von 1946 mit heute vergleicht.

2.1 GIS - Geografische Informationssysteme

Wie erwähnt, arbeite ich als *GIS-Spezialist*. Wobei mir der Titel *Geoinformatiker* besser gefällt: weil er die Begriffe *Geografie* und *Informatik* vereint. *Geografie* kommt aus dem Griechischen und bedeutet Erdbeschreibung [SE04, S. 14]. *Informatik* ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen [10c].

GIS ist ein Akronym für Geografische Informationssysteme. Es bedeutet im engsten Sinn eine Ansammlung von Computerprogrammen, die zur Bearbeitung von Karten und Geodaten verwendet werden. Geodaten sind nichts weiter als Daten mit einem räumlichen Bezug[4]. In einem weiteren Sinn deckt der Begriff GIS ein ganzes Fachgebiet ab, das sich mit Karten und Geodaten auskennt. Es ist also nicht nur ein Werkzeug, sondern ein Fachgebiet, das Kenntnisse über Datensammlung, Speicherung, Analyse und Darstellung innerhalb von vielen verschiedenen Themen mit einem räumlichen Bezug abdeckt. Typische Geodaten sind digitale Karten, Inventare und Register von Parzellen, Umweltfaktoren, Grenzen, Entwicklung, Planung etc., die einen räumlichen Bezug haben und dadurch in einem geografischen Zusammenhang analysiert und dargestellt werden können [BJB06, S. 15].

Es ist zwar weit von der klassischen Geografie zu Zeiten Alexanders von Humbold[5] entfernt, aber es liegt auf der Hand, dass auch die Geografie Cloud Computing nutzt.

[1] www.swisstopo.ch
[2] maps.google.com
[3] Im vorliegenden Dokument wird durchwegs der männliche Singular (Leser, Benutzer) als Ansprache verwendet. Diese Ansprache bezieht sich auf beide Geschlechter sowie gegebenenfalls mehrere Personen. Sie dient lediglich der leichteren Lesbarkeit der Semesterarbeit.
[4] Mit Koordinaten (Nord/Ost, x/y).
[5] Ein Forschungsreisender des 19 Jh. mit einem weit über Europa hinausreichenden Wirkungsfeld. Mehrjährige Forschungsreisen führten ihn nach Lateinamerika, USA und nach Zentralasien [Keh05].

3 Vorgehen

In diesem Kapitel wird das Vorgehen beschrieben, wie die Arbeit geplant und realisiert wurde.

3.1 Arbeitsmethodik

Da die Arbeit von einem einzigen Autor umgesetzt wurde, musste bezüglich Arbeitsmethodik nur wenig definiert und koordiniert werden.

Um zu starten, hat dem Autor geholfen, dass der Experte Jörg Thomann bereits anfangs Sommer (16. Juli) einen ersten Besprechungstermin angesetzt hat. Um eine Diskussionsbasis für diesen Termin zu haben, musste sich der Autor ernsthaft mit der Semesterarbeit auseinandersetzen. Dazu hat er aus einer BFH Vorlage [6] ein Gerüst der Arbeit erstellt und stichwortartig abgefüllt. Nach gründlicher Diskussion mit dem Experten wurde dieses Gerüst etwas angepasst und diente von da an als Basis für die Arbeit; wo sich die Kapitel zunehmend von losen Notizen zum finalen Zustand festigten.

Um die Übersicht der anstehenden Aufgaben nicht zu verlieren, wurde beschlossen, nach Kanban zu arbeiten. Weil für die Arbeit sowieso Repository auf *github.com*[7] angelegt wurde, konnte dort auch gleich ein Kanban Board erstellt werden. Das Kanban Board wurde vor allem auch dafür genutzt, fortlaufend Aufgaben zu erfassen und um die Menge der angefangenen Aufgaben zu kontrollieren. Es wurde darauf geachtet, die Menge der angefangenen Aufgaben möglichst klein zu halten.

Dadurch konnte im Alltag (wenn gerade etwas Zeit zur Verfügung stand) eine solche Aufgabe genommen und erledigt werden. Es wurde nicht strikte nach Kanban gearbeitet, aber einige Prinzipien daraus wurden verwendet. Zum Beispiel wurde das erste Grundprinzip angewendet *"Beginne mit dem, was du gerade tust: In diesem Grundprinzip stecken zwei Dinge. Indem man mit dem beginnt, was man gerade tut, beendet man die aktuelle Arbeit, bevor etwas Neues begonnen wird. Genauso ist hier aber auch die Aussage enthalten, dass Kanban einfach eingeführt werden kann."*[10d]. Eine Momentaufnahme des Kanbans befindet sich im Anhang B.

3.2 Projektplan

Um den übergeordneten Rahmen der Arbeit nicht aus dem Fokus zu verlieren und um sich bewusst zu machen, wieviel Zeit generell bis zur Abgabe zur Verfügung steht, war es hilfreich, einen Projektplan mit Meilensteinen konsultieren zu können. Durch die Berner Fachhochschule waren die grundlegenden Meilensteine bereits vorgegeben. Der Projektplan befindet sich im Anhang C.

[6] Herzlichen Dank an Andreas Habegger und Lukas Studer fürs Bereitstellen der LaTex Vorlage.
[7] Projekt auf github.com.

4 Ausgangslage

Hier werden das Arbeitsumfeld und die Aufgaben beschrieben, aus denen sich die Problemstellung dieser Semesterarbeit ergeben hat.

4.1 swisstopo bei AWS

Es liegt auf der Hand, dass die swisstopo in ihrer Rolle als *Geoinformationszentrum* auf Cloud Computing setzt. Die swisstopo nutzt Cloud Computing mit AWS[8] seit mehr als 10 Jahren für den Betrieb des Geoportal des Bundes.

"Mit der BGDI[9] unter AWS können wir derzeit ca. eine Million Internetbenutzer pro Monat bedienen. Dank AWS können wir die zur Zuordnung neuer Server benötigte Zeit erheblich verkürzen und unseren Fokus auf echte Kundenanforderungen verstärken." [Chr20].

4.2 Publikation von Geodaten

Wie bereits erwähnt, können auf dem Viewer ca. 800 Themen wie Wanderwege, Solarkataster und Luftfahrthindernisse angesehen werden. Unser Team publiziert diese Daten. Der Nachführungszyklus wie auch der Aufwand zur Aufbereitung der Daten fürs Web sind unterschiedlich. Einige Daten werden manuell aufwändig aufbereitet, andere stündlich automatisch nachgeführt.

4.3 Die Web Services geben den Technologie Stack vor

Nebst der Publikation der Daten ist unser Team für den Betrieb und der Weiterentwicklung der Web Services und des Viewers verantwortlich. Der ganze Technologie Stack wurde schon länger nicht mehr grundlegend erneuert. Zurzeit wird die gesamte Architektur analysiert und überarbeitet, um eine gestaffelte Migration auf eine neue Lösung zu ermöglichen. Einige Rahmenbedingungen dieser zukünftigen Architektur sind bereits klar: Das Geoportal des Bundes wird weiterhin in der AWS Cloud betrieben werden, freie Software auf Linux[10], die Migration wird vor allem über Microservices gestaffelt erfolgen, diese Services werden als Docker Container laufen, Amazon Elsatic Kubernetes Service wird die Orchestrierung der Container übernehmen; und für Continuous Integration wird AWS Codebuild/Pipeline zum Einsatz kommen.

[8] Amazon Web Services
[9] Bundesgeodateninfrastruktur: Viewer und andere Services.
[10] Wann immer möglich, freie Software [Sta10]: OpenLayers, PostGIS, Debian, Mapserver, Python Frameworks, Kubernetes etc.

4.4 Exkurs 3D Daten

Die swisstopo erfasst und aktualisiert Daten mit einem räumlichen Bezug. Diese Geodaten sind die Basis für die Ableitung in eine Vielzahl von Produkten, wie die Landeskarten 1:25'000. Nebst Karten gibt es unter anderem die Produktpalette der Landschaftsmodelle. Diese geben die Objekte der Landschaft im flexiblen Vektorformat wieder. Sie bestehen aus thematischen Ebenen (Bsp. Gebäude). Jede Ebene umfasst georeferenzierte Punkt-, Linien-, Flächen- oder 3D Objekte. Jedes Objekt enthält Attribute und Beziehungen [10e].

Zu den Lanschaftsmodellen gehören Produkte wie swissTLM3D und swissBuildings3D. Im Viewer wird eine Auswahl von Themen aus eben diesen Landschaftsmodellen dargestellt: Zurzeit Gebäude, Bäume, Seilbahnen, Namen und das Terrain. Vor wenigen Jahren wurden diese 3D Daten mit einem grossen Effort medienwirksam publiziert.

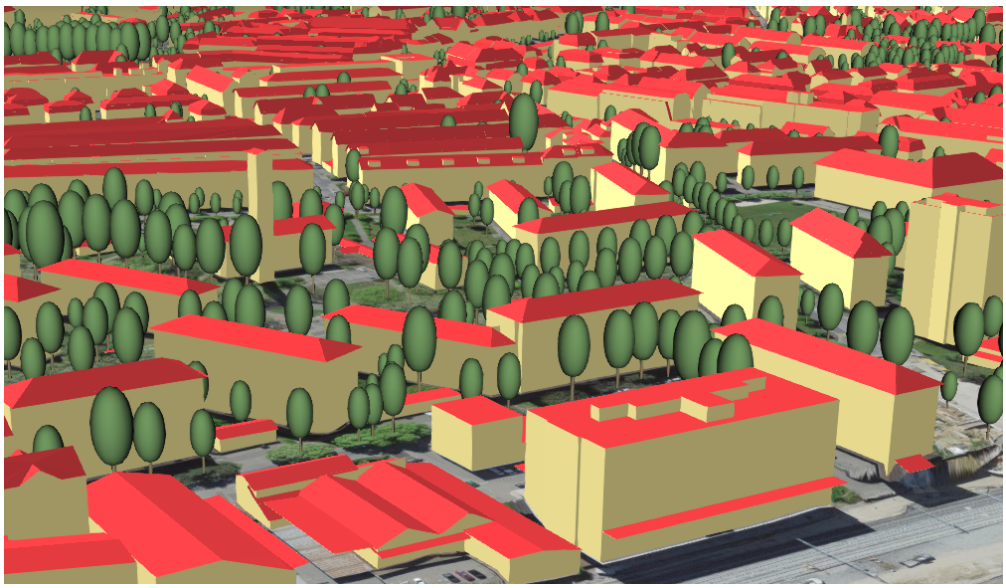

Abbildung 3: Im Viewer werden zurzeit Gebäude, Bäume, Seilbahnen, Namen und das Terrain dargestellt. Um aktuell zu bleiben, müssen diese 3D Daten regelmässig nachgeführt werden.

Seit der Erstpublikation ist inzwischen etwas Zeit vergangen. Als die ersten Aktualisierungen der Daten anstanden, wurde den Beteiligten bewusst, dass sich diese nicht einfach so *auf Knopfdruck* realisieren lässt: Seit der Erstpublikation hat es personelle Wechsel gegeben und punkto Dokumentation und Automatisierungsgrad wurden Lücken identifiziert.

Es gibt immer gute Gründe für *technische Schulden*, wie in diesem Fall für positive medienwirksame Reaktionen[11]. Früher oder später müssen diese abgebaut werden, weil es einen direkten Einfluss auf die Wartbarkeit des Produktes hat [10f].

[11] Wie Bsp. auf watson.ch oder Twitter [18].

5 Der Use Case

5.1 Problemstellung

Es ist der swisstopo schon lange ein Anliegen, die Publikationvorgänge von Geodaten zu optimieren. Wann immer möglich, soll der Automatisierungsgrad erhöht werden.

Besonders aufwändig erweist sich zurzeit die Publikation von 3D Daten. Die manuelle Publikation der 3D Daten benötigt eigene Tools, die auf einem performanten und somit teuren Rechner laufen müssen. Ausserdem erfordert die Bereitstellung einen hohen Koordinationsaufwand zwischen der Infrastruktur und der Entwicklung. Dabei passiert es, dass Mängel in den 3D Daten erst nach beendeter Webpublikation bemerkt werden; und der ganze Publikationsvorgang muss wieder von vorne gestartet werden. Auch dem Hersteller der 3D Daten (dem Bereich Topografie) wäre es ein Anliegen, wenn er diese Daten selbst automatisch publizieren und prüfen könnte.

Einerseits soll in dieser Arbeit mittels Prototyp aufgezeigt werden, wie der Automatisierungsgrad erhöht werden könnte. Andererseits soll untersucht werden, ob für die Verarbeitung Budget Instanzen[12] anstelle von On-Demand Instanzen[13] verwendet werden können und wie deren Einsatz aussehen könnte.

5.1.1 Budget Instanzen

Amazon preist Budget Instanzen folgendermassen an: *"Mit Amazon EC2 Spot-Instances können Sie die Vorteile nicht genutzter EC2-Kapazitäten in der AWS Cloud nutzen. Spot-Instances sind mit einem Rabatt von bis zu 90% im Vergleich zum On-Demand-Preis verfügbar. Sie können Spot-Instances für diverse statuslose, fehlertolerante und flexible Anwendungen verwenden. Dazu zählen unter anderem Big-Data-Anwendungen, auf Containern ausgeführte Workloads, CI/CD, Webserver-Anwendungen, HPC-Anwendungen (High-Performance Computing) sowie Test- und Entwicklungs-Workloads."* [10a].

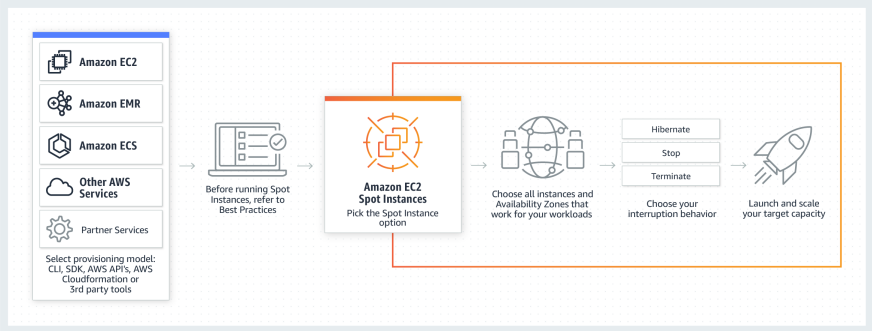

Abbildung 4: So funktioniert das Ausleihen von Bugdet (SPOT) Instanzen

Das Verkaufsargument *90%* Rabatt ist eine Ansage: Eine Preis-Aktion, ein Budget Produkt, damit AWS nicht genutzte EC2-Kapazitäten doch noch verkaufen kann. Der Konsument gibt viel weniger aus, mit dem Nachteil, dass einem die Instanz innerhalb von 2 minütiger Vorankündigung weggenommen werden kann.

Möchte man die EC2 Spot Instanzen für die Geodatenverarbeitung einsetzen, muss also ein Weg gefunden werden, um mit diesen Unterbrüchen umgehen zu können.

Wie auf der Abbildung 4 dargestellt, kann das Verhalten der Instanz bei einem Interrupt[14] definiert werden. Es besteht dabei sogar die Möglichkeit eines *Hibernate*, dass der Zustand des RAMs auf der Festplatte persistiert

[12] Amazon Spot Instanzen
[13] Herkömmliche EC2 Instanzen.
[14] Wenn Amazon die Spot Instanz für einen anderen Zweck beanspruchen möchte, die wegnimmt.

wird.

Diese 2 minütige Vorankündigung eines Intterupts kann auch via RESTful Abfrage[15] von der Instanz aus abgefragt werden.

Diese Vorankündigung lässt sich auch via AWS CLI abfragen. In einem grösseren Setup könnte das Signal der Vorankündigung auch mit dem AWS Überwachungstool *CloudWatch* verarbeitet werden.

5.2 Ist-Zustand der 3D Datenpublikation

5.2.1 3D Datenpublikation

Das Aufzeigen des Ist-Zustandes der 3D Datenpublikation soll helfen sich einen Überblick, eine Ausgangslage, zu verschaffen. Es bildet die Grundlage, um die Frage zu beantworten, welche Arbeiten erledigt werden müssen, um die 3D Daten im Web zu publizieren? Welche Arbeitsschritte könnten automatisiert werden?

Eine Datenpublikation läuft folgendermassen ab: Sobald der Auftrag für eine 3D Datenpublikation erteilt wurde[16], müssen zurzeit folgende Schritte, die in der Abbildung 6 referenziert sind, erledigt werden:

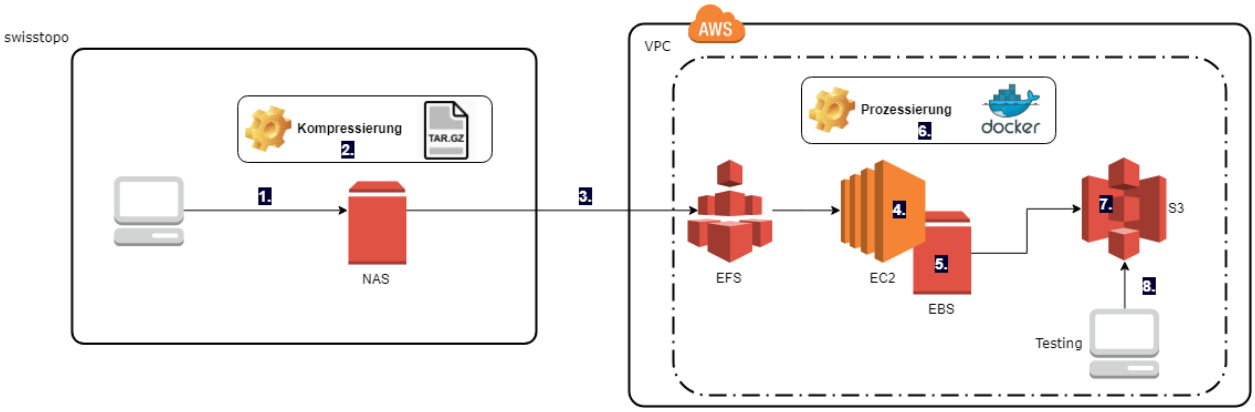

Abbildung 5: Arbeitsschritte, die es braucht, um die 3D Daten zu Publizieren.

1. Die Rohdaten[17] werden vom Auftraggeber (Bereich Topografie) auf einem NAS bereitgestellt.
2. Da es sich um Millionen Dateien handelt, werden diese je Kartenblatt[18] erst einmal gezippt, um so (weil bedeutend weniger Dateien) schneller kopiert werden zu können.
3. Kopieren der komprimierten Dateien vom swisstopo Netzwerk in die Amazon Cloud (AWS VPC[19]) kopiert[20].
4. Parallel dazu wird die IT via Ticket gebeten einen Server mit dem entsprechenden Image[21] bereitzustellen.
5. Kopieren und Entzippen der Rodaten auf die gemountete Festplatte[22] des Servers.
6. Die Daten auf dem Server via Docker verarbeiten[23].

[15] EC2 Spot Instance Interruption Notice, eine HTTP Abfrage: Im Anhang D.2 hat es ein Beispiel dazu.
[16] Vom Bereich Topografie.
[17] Das Format der Rohdaten ist KML/COLLADA (beides XML).
[18] Blatteinteilung: Die swisstopo arbeitet viel nach Blatteinteilung nach Kartenblättern, hier ein Beispiel: map.geo.admin.ch.
[19] AWS: Amazon Web Services, VPC: Virtual Private Network. Ein Kopieren.
[20] Via *rsync*.
[21] Eine EC2 Instanz *eine m4.10xlarge* aus einem bereits vorhandenes AMI.
[22] Ein EBS Volumen.
[23] Umwandeln in das Web-Format *Cesium3DTiles*.

7. Kopieren der Web-optimierten Daten auf S3.
8. Die Daten visualisieren, um inhaltlich testen zu können. Ein Codepen Projekt[24], dass auf die 3D Tiles zugreift.

5.2.2 Aufwand der Verarbeitung

Folgende Schritte sind besonders aufwendig:

- **Abb. 6, Schritt 2. und 5.**: Das Kopieren / Komprimieren (Packen und Entpacken) der Rohdaten dauert lange.
- Es kommt immer mal wieder vor, dass Daten korrupt sind, was zu einer Nachlieferung führt, mit der Gefahr, dass es mit Versionen der Lieferung zu einem Durcheinander kommen könnte.
- **Abb. 6, Schritt 4.**:Unsere IT muss für die Verarbeitung eine EC2 Instanz mit EFS bereitzustellen. Um laufende Kosten zu verringern, wird diese Instanz nach getaner Arbeit[25] wieder gestoppt. Falls mit den Daten etwas nicht in Ordnung ist, muss dieser Schritt von der IT wiederholt werden. Nebst der Bemühung der IT, muss auf der Instanz selber anschliessend das eine und andere manuell installiert und konfiguriert werden.

5.2.3 Technische Komponenten

Auflistung der technischen Komponenten:

- **Abb. 6, Schritt 2., 3. und 5.**: Das Komprimieren und Kopieren der Rohdaten erfolgt mit Linux Bordmitteln (*cp*, *rsync*, *tar*)
- **Abb. 6, Schritt 4. und 7.**: Erfolgen via AWS CLI
- **Abb. 6, Schritt 6.**: Via Docker. Der Container wurde von der Firma Analytical Graphics Inc. [10b] bereitgestellt. Das Tool, das die Rohdaten in ein Web-Format umwandelt, wird mit Node.js ausgeführt.
- **Abb. 6, Schritt 7.**: Ein Projekt, um die Daten im Browser betrachten und inhaltlich Testen zu können, erfolgt über die Webseite codepen.io.

[24] SaaS: Eine Webseite, um Front-End Code zu schreiben, zu testen, und bereitzustellen (codepen.io).
[25] Der Verarbeitung.

6 Architektur

6.1 Analyse des Ist-Zustandes

6.1.1 Bereitstellung der Rohdaten und Sicherstellung dessen Qualität

Basierend auf der Aufwandeinschätzung des Ist-Zustandes des Kapitels 5.2.2 geht hervor, dass vor allem das Bereitstellen der Rohdaten und das Sicherstellen dessen Qualität aufwändig ist.

Das Bereitstellen der Rohdaten nimmt aufgrund der Datenmenge[26] viel Zeit in Anspruch. Hier gäbe es folgende zwei Lösungsansätze:

1. Der Datenlieferant könnte direkt aufs EFS schreiben (Abb. 6, Nr. 5)
2. Automatischer Prozess via Cronjob oder Trigger, der die Kopierschritte (Abb. 6, Schritte Abb. 6) regelmässig bereitstellt.

Wobei der Lösungsansatz 1. simpler und weniger fehleranfällig zu sein scheint. Dieser Ansatz könnte, sobald die swisstopo eine Hybrid Cloud hat, weiterverfolgt werden.

Sicherung der Qualität: Weil es dem Lieferanten an Tools fehlt, um die bereitgestellten Daten inhaltlich zu Prüfen, werden Fehler häufig erst nach der Publikation entdeckt. Dies sicherlich auch, weil die Daten im Web an ein breites Publikum gelangen. Da keine inhaltliche Prüfung der Rohdaten gemacht werden kann, wäre es sicher hilfreich, wenn der Datenlieferant die Iterationen der Publizierung gleich selber machen könnte und dann nur noch Bescheid gibt, wenn sie seines Erachtens in Ordnung sind.

6.1.2 Bereitstellung der Infrastruktur für die Verarbeitung

Die Rohdaten werden via Docker Image verarbeitet. Jedes Mal wenn das Image für die Verarbeitung ausgeführt werden soll, muss die IT gebeten werden, die nötige Infrastruktur bereitzustellen. Idealerweise findet sich eine Lösung, die diesen Schritt überflüssig macht.

6.2 Bewertungskriterien

Kosten einsparen	Änderungen des Prozesses sollen nicht teurer sein, als die bisherige Lösung.
Automatisierbar	Die Lösung soll den Grad der Automatisierung möglichst weit vorantreiben und dadurch so wenig Personalaufwand wie möglich beanspruchen.
Einfach	Ein neuer Mitarbeiter soll die Lösung rasch verstehen und warten können.
Bestehende Technologie	Obwohl der Anwendungsfall anders ist, soll der Technologie Stack möglichst demjenigen der Microservices entsprechen.

[26] Weil es sich um mehrere Millionen Dateien handelt.

6.3 Architekturentscheid

Für den Architekturentscheid wurde auf eine Entscheidungsmatrix verzichtet. Dies Aufgrund der grossen Auswahl und der vielen Entscheidungen, die getroffen werden mussten. Die oben erwähnten Kriterien wurden in der Entscheidungsfindung mit einbezogen und weitere Argumente werden ausgeführt:

6.3.1 Verarbeitung auf Spot Instanzen

Der Autor ist auf drei Möglichkeiten gestossen, um Geodaten mit AWS Spot Instanzen zu verarbeiten: *Direkt auf der Spot Instanz*, *Kubernetes (EKS)* und *AWS Batch*. Gerne hätte der Autor alle drei Lösungen als Prototyp weiterverfolgt, aber die Zeit dazu reichte leider nicht. Aus diesem Grund beschränkt sich der Autor lediglich auf eine Beschreibung der drei Möglichkeiten. Wobei die erste Möglichkeit als Prototyp umgesetzt wurde.

Direkt auf einer Spot Instanz: Diese Möglichkeit bedingt nur geringe Anpassungen der bisherigen Verarbeitung, weil die Daten im Prinzip wie bisher auf einer EC2 Instanz verarbeitet werden; jedoch mit dem Nachteil, dass diese jederzeit durch eine andere Instanz ersetzt werden könnte.

Der Autor hat sich bei der Auswahl der Images[27] für *Ubuntu Server 18.04 LTS* entschieden. Dies weil die Verarbeitung bisher auf Ubuntu gelaufen ist und weil er keinen Grund sieht, etwas Funktionierendes zu ändern[28]. Dasselbe Argument gilt für die Wahl der Dimensionierung des Servers: Ein Server mit 200 GB SSD Festplatte, 16 CPUs und 60 GByte Arbeitspeicher konnte die Verarbeitung der 3D Daten[29] gut stemmen.

Bei der Provisionierung[30] des Servers wurde der *"Cloud-init"* Ansatz[31] gewählt: Dieser Ansatz provisioniert den Server der Startup Phase. Dies hat gegenüber einem eigenen AMI den Vorteil, dass die Anforderungen in einer Textdatei festgehalten und somit jederzeit nachvollziehbar sind. Ausserdem muss bei einem Update des Basis Images kein neues Image gebaut werden. Die Initialisierungsskript wurde auf ein Minimum beschränkt und die eigentliche Initialisierung wird an *Ansible* übergeben.

Kubernetes (EKS) auf Spot Instanzen erweitern TODO: Weil die Microservices in Zukunft auf EKS laufen werden, wäre es sicher ein Möglichkeit, abzuklären, inwiefern sich eine Datenverarbeitung in dieser Umgebung einrichten lässt. In der Literatur hat der Autor folgende Elemente gefunden.

- Job
- CronJob
- Eigenes ReplicaSet?

EBook auf meinem Reader: Job und CronJob aber auch Kubernetes on AWS S. 88

AWS Batch auf Spot Instanzen Diese Möglichkeit wäre wohl die Naheliegendste, weil AWS Batch für genau diese Art von Aufgaben gebaut wurde. Im Internet hat der Autor einen schönes Beispiel gefunden, wie sich AWS Batch einsetzen lässt. https://aws.amazon.com/de/blogs/compute/creating-a-simple-fetch-and-run-aws-batch-job/

[27] AMI: Amazon Machine Image.
[28] Security Maintenance bis 2028 [20d].
[29] Gebäude, Bäume und Namen.
[30] Bereitstellung
[31] Eigentlich ein Bash Skript. Unter AWS als *User Data* bezeichnet.

7 Realisierung des Prototyps

Als Umgebung, um die Infrastruktur für den Prototypen bereitzustellen, hat der Autor das Kennenlernangebot von AWS, ein einjähriges kostenloses Kontingent an Services und Produkten [20a], gewählt.

Nachdem der erwähnte *AWS Account* erstellt war, mussten erste Schritte, wie grundlegende Konfigurationen des Identity Access Management (IAM) gemacht und das Steuern von Spot Anfragen, das Einrichten eines Container Repositories (ECR), das Aufsetzen eines öffentlich zugänglichen S3 Buckets und anderes aus dem AWS Produktekatalog, erlernt werden.

Weitere Schritte folgten. Die eigentliche Realisierung des Prototypen wird in den folgenden Kapiteln beschrieben werden. Das Endresultat wird jedoch der Nachvollziehbarkeit halber schon einmal hier abgebildet:

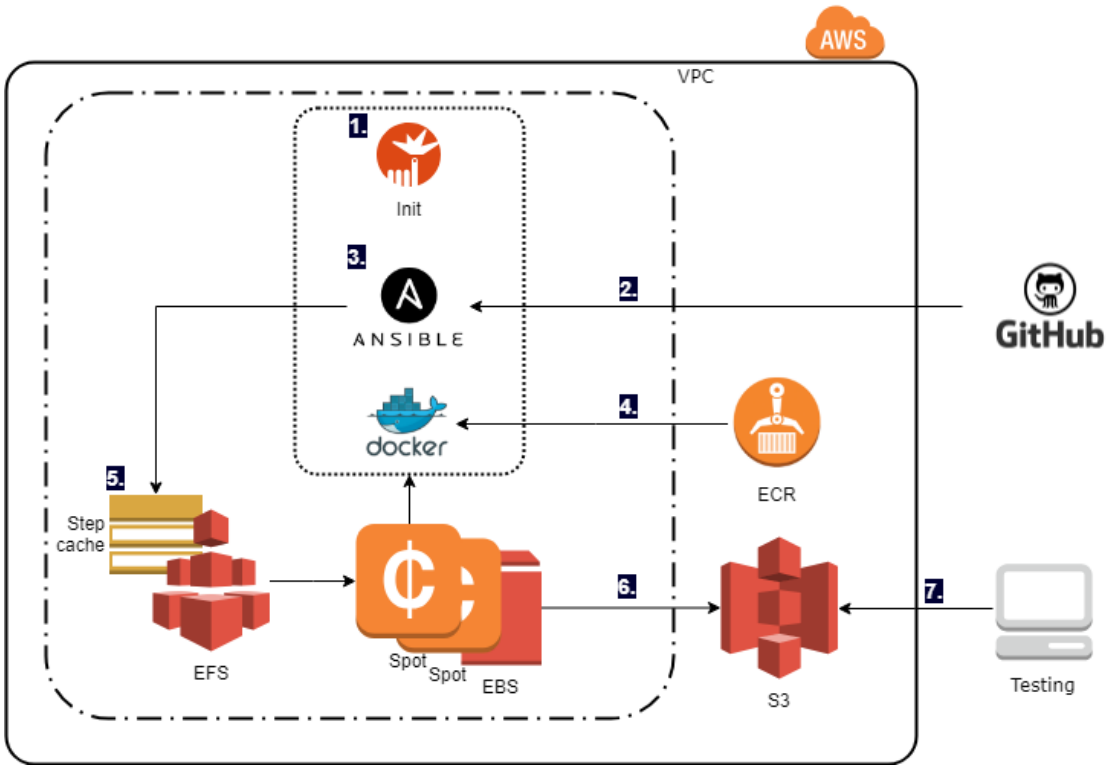

Abbildung 6: Geodatenverarbeitung mit SPOT Instanzen.

7.1 Spot Flottenanfrage

Als Setup für die Spot Flottenanfrage wurde ein sogenanntes Launch Template mit den wichtigsten Konfigurationen wie Sicherheitsgruppe, Wahl des AMI und User Data angelegt. Anschliessend wurde eine Spot Flottenanfrage basierend auf diesem Template erstellt. Wobei als Mindestanforderung an den Rechner 16 CPUs und 60 GB Memory waren. Als Option wurde gewählt, dass die Zielkapazität aufrechterhalten bleiben soll[32]: Somit wird nach jedem Interrupt automatisch eine neue Instanz mit der definierten Konfiguration zur Verfügung gestellt.

[32] Die Zielkapazität soll immer 1 sein.

7.2 Handling der Interrupts

Wie im Kapitel 5.1.1 beschrieben, sind Spot Instanzen so günstig, weil sie einem jederzeit weggenommen werden können, um einem anderen Kunden zur Verfügung zu stellen. Also muss die Datenverarbeitung mit dem Handling von Interrupts umgehen können. Der Einfachheit halber hat der Autor gänzlich auf das *Horchen eines bevorstehenden Interrupts* via RESTful Abfrage (In Anhang D.2 beschrieben) oder *CloudWatch* verzichtet und hat stattdessen die Datenverarbeitung als Serie betrachtet und in Schritte unterteilt. Sobald ein Schritt erledigt wird, wird dies auf dem EFS Volumen in einer Textdatei festgehalten. Falls es zu einem Interrupt kommen sollte, wird von der Spot Flotte die nächste Instanz bereitgestellt und die Publizierung macht bei dem Schritt weiter, der zuletzt in der Textdatei festgehalten wurde (Abbildung 6, Nr. 5).

7.3 Die Datenverarbeitung als Code

Mit der Entscheidung, die Spot Instanz via *Cloud-Init*-Ansatz (Abb. 6, Nr. 1) mit Ansible zu Provisionieren[33], lag es auf der Hand, auch gleich dieselbe Technologie für die Realisierung der Datenverarbeitung zu verwenden (Nr. 3). Viele Annehmlichkeiten, wie die Nähe zur Shell und Python, das Logging und deklarativer Code zeigten sich in der Realisierung als hilfreich. Eines der ersten Kommandos welches das Initialisierungsskript ausführt, ist das klonen des Codes von github.com (Abb. 6, Nr. 2). Das Init-Skript wie auch das Ansible Playbook können auf github.com/bfh-semesterarbeit eingesehen werden.

7.4 Testen der Datenstruktur

Wie in Kapitel 6.1.1 erwähnt, könnte die Datenverarbeitung bereits mit einfachen Bordmitteln vereinfacht werden. Es ist auch schon vorgekommen, dass die gelieferten Rohdaten selbst fehlerhaft waren. Da die alle gelieferten Rohdaten (KML und Collada) in XML geschrieben sind, wurde beschlossen einen kleinen Test mit grosser Wirkung in den automatischen Publikationsprozess mit einzubeziehen. Dieser Test macht nichts anderes, als die Dateien auf well-formed XML zu testen. Dieser Test schreibt den Namen aller fehlerhaften XML Dateien in eine Liste. Nach dem Testen wird die Länge dieser Liste überprüft und falls diese nicht leer ist, wird der gesamte Datenverarbeitung gestoppt. Diese Liste kann dann dem Auftraggeber zur Nachbearbeitung der fehlerhaften Daten ausgehändigt werden. Dieser Schritt passiert innerhalb Schritt Nr. 3 der Abbildung 6.

7.5 Datenverarbeitung

Die eigentliche Datenverarbeitung, der Formatumbau[34], wird mittels Docker Container ausgeführt. Um nicht von externen Services wie dockerhub.io abhängig zu sein und aus Neugier, wurde dafür ein AWS Container Repository[35] eingerichtet (Abbildung 6, Nr. 4.). Das Kopieren der fertig verarbeiteten Webdaten auf S3 wird über die AWS CLI gemacht (Abb. 6, Nr. 6).

7.6 Monitoring

Bisher wird die Datenverarbeitung lediglich geloggt. Die verwendeten Tools[36] wie auch Ansible loggen in ein Verzeichnis im EFS, das gut zugänglich ist. Vor allem die Logdatei von Ansible ist ausführlich und macht eine Fehlersuche einfach.

Um die Last des Servers zu überprüfen, wurde lediglich die Standardüberwachung der Amazon Webconsole verwendet.

[33] Und nicht mit einem neuen AMI.
[34] Rohdaten in das Web Format Cesium Tiles.
[35] ECR
[36] Der well-formed XML Test und der Output des Containers.

Bei einer allfälligen Integration des Prototypen in den Betrieb, müsste das Format und der Datenfluss dieser Logs an die Vorgaben der produktiven Infrastruktur angepasst werden.

7.7 Inhaltliche Kontrolle der publizierten Daten

Um dem Datenlieferanten, dem Hersteller der 3D, zu ermöglichen, die 3D Daten inhaltlich zu prüfen, wurde ein sogenanntes codepen.io Projekt angelegt, dass auf den S3 Bucket des Prototypen zugreift und man so direkt die verarbeiteten 3D Gebäude inhaltlich überprüfen kann. Das Resultat, des Belastungstests (Kap. 7.8.2), kann vorläufig[37] betrachtet und somit, zumindest theoretisch, inhaltlich überprüft werden. Es zeigt alle erfassten 3D Gebäude der Schweiz: codepen.io/rebert/pen/ExKZmmE. Wobei zu erwähnen ist, dass nur die Gebäude vom S3 Testbucket der Prototypen kommen (Abb. 6, Nr. 7).

7.8 Testen

Da es sich um eine Datenverarbeitung und nicht um einen öffentlich zugänglichen Web Service handelt, wurde auf End-to-end Testing und Unit-Tests verzichtet. Es wurden lediglich Überprüft, ob die Datenverarbeitung auf den Spot Instanzen funktioniert. Hierzu wurden zwei Arten von Tests gemacht:

7.8.1 Unerwartete Interrupts

Da es in der ganzen Entwicklungsphase zu keinem einzigen Interrupt einer Spot Instanz gekommen ist, musste die Stabilität des Datenverarbeitungsprozesses auf unerwartete Interrupts manuell getestet werden. Um die Testphasen zu verkürzen, wurde die Menge der Eingangsdaten beschränkt[38].

Dieser Test wurde iterativ umgesetzt und der Code wurde bei Fehlern fortlaufend angepasst.

7.8.2 Belastungstest

Der komplette Rohdatensatz, alle (erfassten) Gebäude der Schweiz, wurde verarbeitet. Dieser Test war sozusagen die Probe aufs Exempel. Bezüglich Anforderungen an die Rechnerleistung konnte auf Erfahrungswerte aufgebaut werden. Die Anforderungen an die Instanz mit dem gemounteten SSD Volumen haben gereicht. Dieser Test wurde auch dafür genutzt, den Output des Prototypen inhaltlich begutachten zu können, wie im vorhergehenden Kapitel (Kap. 7.7 erwähnt. Aber auch die Erkenntnisse zur Wirtschaftlichkeit Kap. 8.2 wurden aus diesem Test abgeleitet.

[37] Bis am 1. November 2020
[38] Auf ein Schweizerischer Kartenblatt 1:25'000.

8 Evaluation

8.1 Erfahrungen

Die Anregungen aus dem Studium[39] und das günstige Angebot von AWS haben den Autor motiviert, die komplette Infrastruktur für den Prototypen selber in einem eigenen AWS Testaccount aufzubauen. Da die Infrastruktur und die Verarbeitung deklarativ festgehalten wurden, kann ein Grossteil dieses Codes auch im Account des Betriebes ohne grosse Anpassungen wiederverwendet werden.

Im Betrieb hätte das Team und die IT Abteilung das nötige Know-How gehabt, um mit dieser Arbeit weiter zu kommen, als es jetzt der Fall ist. Aber der Autor hat sich bewusst dazu entschieden, die Arbeit möglichst selbständig zu realisieren und diese Ressource zu schonen, indem er sie so wenig wie möglich genutzt hat. Es war für den Autor schon nur eine Hilfe, zu Wissen, dass die Möglichkeit einer Unterstützung da war. Dies führte dazu, dass der Autor Momente der Ratlosigkeit erleben musste. Was nicht weiter schlimm war, da *"Perplexity is the beginning of knowledge"* [AD19, S. 33].

Bei der Auswahl der Spot Instanz Aufgabe hat der Autor *Flexible Workloads* gewählt. Der Autor war darüber erstaunt, dass ihm während der gesamten Entwicklungs- und Testphase nicht ein einziges Mal eine Instanz weggenommen wurde. Die Spot Flotte musste aufgrund eines gemounteten EBS-Volumens auf eine bestimmte Availability-Zone beschränkt werden. Nicht einmal mit dieser Einschränkung ist es zu einem Interrupts gekommen. Um das Verhalten bei Interrupts testen zu können, mussten die Spot-Instanzen somit mutmasslich terminiert werden.

8.2 Wirtschaftlichkeit

Personalstunden Der Grad der Automatisierung konnte erheblich erhöht werden. Dadurch fallen Personalstunden[40] weg. Vor allem diese Kosten rechnen sich. Die Verarbeitungsschritte sind im Code abgebildet, was die Fehleranfälligkeit kleiner macht, als wenn Kommandos aus der Prozessdokumentation kopiert werden müssen. Um die IT Abteilung gänzlich zu Entlasten, müsste noch eine Rolle eingerichtet werden, die eine vordefinierte Spot Flottenanfrage steuern kann.

Einsparungen Spot im Vergleich zu On-Demand Um die Kosten von On-Demand mit Spot Instanzen zu vergleichen, werden hier die Ergebnisse des Prototyps aufgelistet. Verarbeitung der Gebäude mit der Mindesanforderung: 16 CPUs und 60 GByte Memory. Die Gesamtrechenzeit war ca. 18 Stunden und es konnten 76% der Kosten eingespart werden.

Rechenzeit	**On-Demand**	**Spot**
Pro Stunde	1.19$	0.29$
Total: 18 h	21.42$	5.22$

Tabelle 1: Relativ betrachtet ist das Sparpotential enorm: 76%.

Schaut man die Kosten relativ an, ist das Sparpotential enorm: 76%! In absoluten Zahlen erscheint das Sparpotential für einen einzelnen Verarbeitungsauftrag nicht riesig: ca. 15$. Dazu muss allerdings ergänzt werden, dass die Verarbeitungszeit durch die Automatisierung verkürzt wurde. Ausserdem werden jährlich mehrere 3D Updates in Auftrag gegeben. Der aufwändigste Auftrag davon ist das Update des 3D Terrains, das eine wesentlich grössere Instanz über eine längere Zeitspanne[41] braucht.

[39] CAS in Cloud Computing an der BFH Bern: Bsp. die Vorlesungen Architektur, IaaS, PaaS, Docker und Kubernetes.
[40] In unserem Team.
[41] Ca. 1 Woche.

8.3 Kritische Punkte

8.3.1 Security

Bezüglich IAM hat der Prototyp noch technische Schulden. Dem Autor ist bewusst, dass folgende zwei Sicherheitslücken bezüglich Zugängen vorhanden sind:

github.com: Der Einfachheit halber wurde das Github Repository, auf welchem sich das Init-Skript und das Ansible Playbook befinden, öffentlich zugänglich gemacht (Abb. 6, Nr. 2). Dies ist nicht weiter problematisch, da der Code auf Github keine Passwörter etc. bekannt gibt. Aber es sind darin Informationen über die AWS Infrastruktur enthalten. Sauber wäre eine Implementation, die Github Credentials eines privaten Repositories verwaltet. Geeignet dazu wäre z.B. der AWS Secrets Manager [20b].

AWS Key für S3: Eigens für das Kopieren auf S3, dem Zugang für die Container Registry (ECR) und für das Mounten eines SSD Volumen wurde eine IAM Rolle angelegt, die nicht mehr darf, als die eben erwähnten Tätigkeiten (Abb. 6, Nr. 4, Nr. 6 und das Mounten des Volumens). Der Zugangsschlüssel dazu wurden auf dem EFS Volumen abgelegt. Dies ist sicher nicht ideal. Zwar ist die Sicherheitsgruppe des EFS ist zwar so konfiguriert, dass nur die Sicherheitsgruppe der EC2 Instanzen darauf Zugriff haben sollten, aber dennoch sollten keine unverschlüsselten Keys auf dem Filesystem herumliegen. Um diese Sicherheitslücke zu schliessen, könnte eigens dafür eine IAM Rolle für die EC2 Instanz angelegt werden [20c].

Sollte der Prototyp in den Betrieb überführt werden, müsste man das IAM der swisstopo übernehmen; und dieses Kapitel hätte sich erübrigt.

8.3.2 Datenverarbeitung als Blackbox

Idealerweise wird die Datenverarbeitung in kleine parallelisierbare und in sich selber geschlossene Schritte unterteilt. Nach einem Interrupt der Instanz können dann die noch nicht abgeschlossenen Schritte fortgeführt werden.

Beim gewählten Use Case der 3D Datenverarbeitung handelt es sich um eine Blackbox in einem Container, die entweder Alles oder Nichts verarbeitet. Was in diesem Fall alle (erfassten) Gebäude der Schweiz bedeutet. Damit der Prototyp funktioniert, müssen die Spot Instanzen mindestens 10 Stunden ohne Interrupt laufen. Zum Glück verhalten sich die Spot Instanzen in der Regel stabil genug, um die einzelnen langwierigen Schritte seriell verarbeiten zu können.

Der Test, der die Rohdaten auf well-formed XML testet, hilft Fehler, vor dem langwierigen Verarbeiten in der Blackbox, abzufangen. Jedoch leider nicht alle Fehler. Wenn man nun davon ausgeht, dass es in den ca. 8 Millionen Gebäuden, die geliefert wurden, immer mal wieder fehlerhafte Daten hat, welche die Blackbox ins Straucheln bringen, kann die 3D Datenverarbeitung trotz aller Automaistation zu einem mühseligen Unterfangen werden.

9 Ausblick

Bald steht das nächste 3D Update vor der Tür. Bei dieser Gelegenheit könnte der Automatisierungsteil dieser Semesterarbeit übernommen werden. Da es sich dabei um ein Ansible Playbook in einer deklarativen Sprache handelt, sind die einzelnen Verarbeitungsschritte auch gleich dokumentiert. Anpassungen sind einfach zu machen. Falls diese Nachführung auf einer herkömmlichen On-Demand EC2 Instanz gemacht werden müsste, könnte das Skript dennoch verwendet werden.

Mittelfristig wäre es schon nur aus Gründen des Kostensparens interessant, wenn rechenintensive Geodatenverarbeitungen auf Spot Instanzen laufen könnten. Hier nochmals: Relative Einsparungen von mehr als 70%, was sich mit der Zeit auch in absoluten Kosteneinsparungen ($) sehen lassen würde.
Um den Koordinationsaufwand mit der IT zu verringern, könnte die Person, die eine Spot Instanz braucht, mit den nötigen Rechten versehen werden: Bsp. könnte sie, wie bei einem Load-Balancer, die Anzahl Instanzen der Spot Flotte von 0 auf 1 ändern dürfen, um nach getaner Arbeit die Anzahl Instanzen wieder auf 0 zu setzen.

Sobald die swisstopo über eine Hybrid Cloud verfügt, könnte die Topografie neue Daten direkt auf das EFS schreiben. Dieser Event könnte getriggert werden, um eine Spot Instanz für die 3D Geodatenverarbeitung zu starten. Jeweils ca. 18 Stunden später wären die Daten im Web und die Topografie könnte diese auf der Testumgebung einsehen (Abb. 6, Nr. 7). Diesen Vorgang könnte die Topografie solange wiederholen, bis sie mit dem Resultat zufrieden ist.
Ohne Hybrid Cloud müssten die Daten immer noch vom Intranet in die AWS VPC kopiert werden. Auch dieser Vorgang liesse sich einfach automatisieren.

Persönlich hat dem Autor die Tour mit Spot Instanzen Spass gemacht. Die auf der Tour gemachten Erfahrungen waren ein guter Einstieg, um Services der AWS Cloud besser verstehen und nutzen zu können. Es wird sicher nicht bei dieser Tour bleiben.

Literaturverzeichnis

[10a] (2010). Amazon EC2-Spot-Instances, Adresse: https://aws.amazon.com/de/ec2/spot/ (besucht am 08. 07. 2010).

[10b] (2010). Analytical Graphics Inc., Adresse: https://www.agi.com (besucht am 09. 08. 2020).

[10c] (2010). Informatik, Adresse: https://de.wikipedia.org/wiki/Informatik (besucht am 16. 07. 2010).

[10d] (2010). Kanban (Softwareentwicklung), Adresse: https://de.wikipedia.org/wiki/Kanban_(Softwareentwicklung) (besucht am 19. 07. 2010).

[10e] (2010). swisstopo Onlineshop, Adresse: https://shop.swisstopo.admin.ch (besucht am 17. 07. 2020).

[10f] (2010). Technische Schulden, Adresse: https://de.wikipedia.org/wiki/Technische_Schulden (besucht am 18. 07. 2010).

[18] (2018). Diese 3D-Landkarte gibt dir einen völlig neuen Blick auf die Schweiz, Adresse: https://www.watson.ch/digital/schweiz/674619561-diese-3d-landkarte-gibt-dir-einen-voellig-neuen-blick-auf-die-schweiz (besucht am 18. 07. 2010).

[20a] (2020). AWS Free Tier, Adresse: https://aws.amazon.com/de/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc (besucht am 21. 08. 2020).

[20b] (2020). AWS Secrets Manager, Adresse: https://aws.amazon.com/secrets-manager/ (besucht am 28. 08. 2020).

[20c] (2020). IAM roles for Amazon EC2, Adresse: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html?icmpid=docs_ec2_console (besucht am 28. 08. 2020).

[20d] (2020). The Ubuntu lifecycle and release cadence, Adresse: https://ubuntu.com/about/release-cycle (besucht am 28. 08. 2020).

[AD19] J. Arundel und J. Domingus, *Cloud Native DevOps with Kubernetes*. O'RELLY, 2019.

[BJB06] T. Balstrøm, O. Jacobi und L. Bodum, *GIS og geodata*. Forlaget GIS og Geodata, 2006.

[Chr20] H. Christ. (9. Juli 2020). swisstopo Fallstudie, Adresse: https://aws.amazon.com/de/solutions/case-studies/swisstopo-hpc/.

[Keh05] D. Kehlmann, *Die Vermessung der Welt*. Rowohlt Verlag GmbH, 23. Sep. 2005, 304 S., ISBN: 3498035282. Adresse: https://www.ebook.de/de/product/3563778/daniel_kehlmann_die_vermessung_der_welt.html.

[SE04] M.-H. Schertenleib und H. Egli-Broz, *Grundlagen Geografie: Aufgaben des Fachs, Erde als Himmelskörper und Kartografie : Lerntext, Aufgaben mit Lösungen und Kurztheorie*. Zürich: Compendio Bildungsmedien, 2004, ISBN: 3715591714.

[Sta10] R. Stallman. (2010). What is Free Software?, Adresse: http://www.gnu.org/philosophy/free-sw.html (besucht am 28. 08. 2020).

[Web10] Webpage. (2010). Keep calm and make your choice..., Adresse: https://www.grenke-40-one.de/excursions-detail/ (besucht am 11. 07. 2010).

A Fachbegriffe und Abkürzungen

Abkürzung	Definition
Ansible	Ein Open-Source Automatisierungswerkzeug zur Orchestrierung und allgemeinen Konfiguration und Administration von Computern.
AMI	Amazon Machine Images: Images für virtuelle Server.
Availability Zone	Jeder Amazon Rechenzentrumstandort (Region) besteht aus mehreren isolierten Zonen, den Availability Zones.
AWS	Amazon Web Services
AWS CLI	Das Command Line Interface, um AWS Ressourcen zu verwalten.
BGDI	Bundesgeodateninfrastruktur.
Budget Instanzen	Im Kontext dieser Arbeit ein Synonym für AWS Spot Instanzen.
EBS	Block Storage: Speicher (für eine Instanz).
EC2	Amazon Elastic Compute Cloud: Rechenkapazität, Speicher (und mehr).
EFS	Amazon Elastic File System: Cloud NFS-Dateisystem.
Hybrid Cloud	Eine Computerinfrastruktur, die Public Cloud und Private Cloud kombiniert nutzt.
IaaS	Cloud Infrastructure as a Service: Infrastruktur *"Pay as you go"* beziehen.
IAM	Identity and Access Management: Verwaltung der Zugänge und der dazugehörenden Rechte.
On-Demand Instanzen	Herkömmliche EC2 Instanzen. Der Begriff wird in dieser Arbeit manchmal verwendet, um von EC2 Spot Instanzen unterscheiden zu können.
Kartenblatt	In der swisstopo wird viel in der Einheit von Kartenblättern gearbeitet, welche die ganze Schweiz abdecken. Hier ein Beispiel: map.geo.admin.ch.
PaaS	Plattform as a Service
POC	Proof of Concept: Die Machbarkeit eines Produktes oder einer Idee aufzeigen.
S3	Amazon S3 (Simple Storage Service): Ein Filehosting-Dienst dessen Zugriff über HTTP/HTTPS erfolgt.
SaaS	Software as a Service
SSD	Solid-State-Disk - Halbleiterlaufwerk: Festplatte ohne bewegliche Teile, dafür mit kurzen Zugriffszeiten.
SSH	Secure Shell: Netzwerkprotokoll, um auf auf einen entfernten Rechner zuzugreifen.
S3	Simple Storage Service: Ein Objektspeicherservice von Amazon.

Tabelle 2: In der Arbeit verwendete Fachbegriffe und Abkürzungen

B Kanban

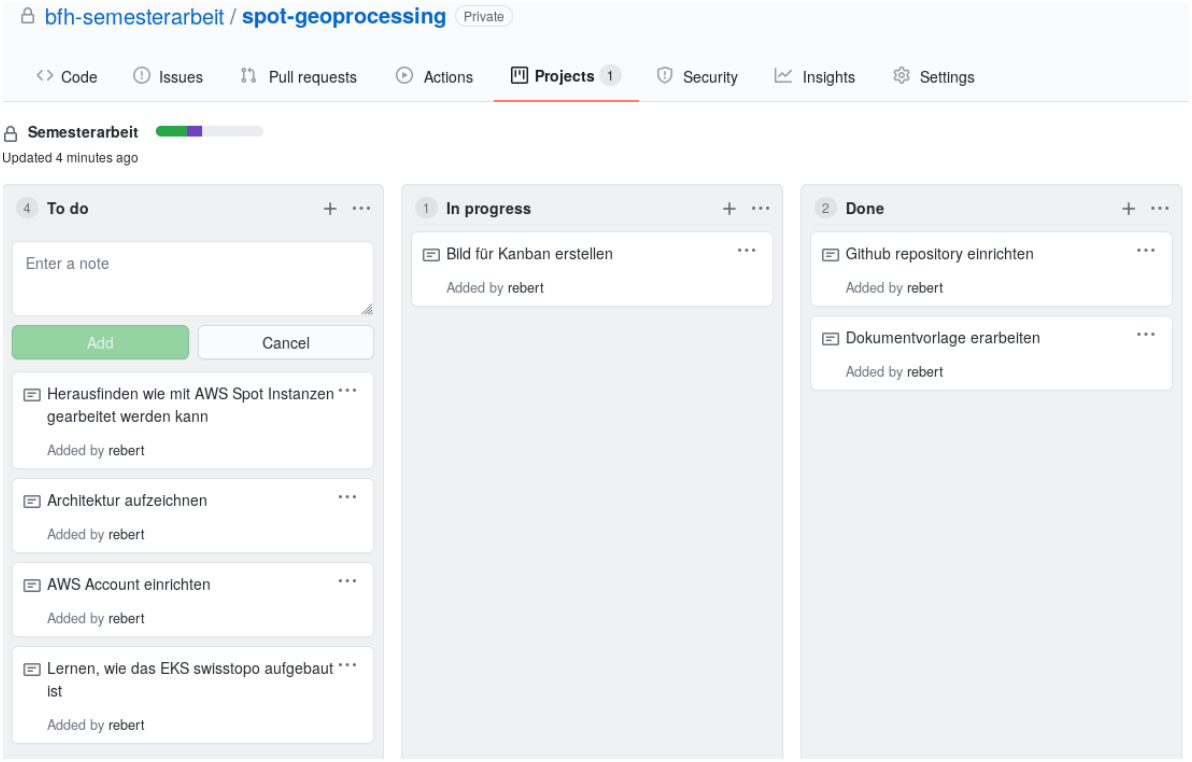

Abbildung 7: Klassisches Kanban auf *github.com*

C Projektplan

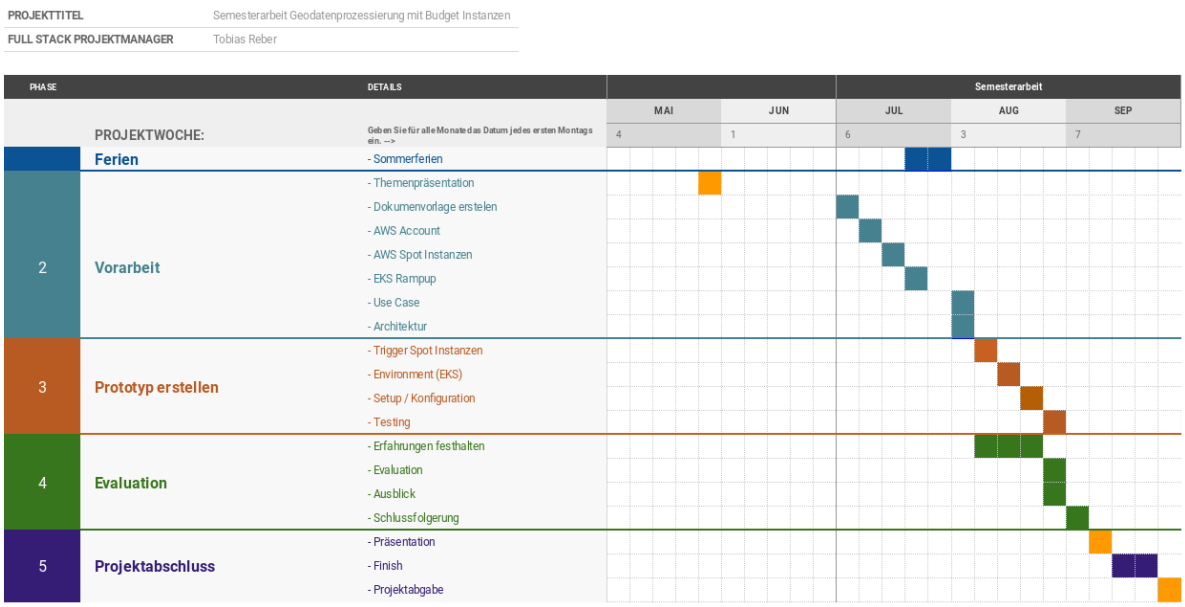

Abbildung 8: Projektplan. Die orangen Meilensteine wurden von der BFH vorgegeben.

D Konfigurationen und Kommandos

D.1 EFS auf EC2-Instanz mounten

Anhand einer Anleitung, einem sogenannten Walktrough, wurde via AWS CLI[42] ein EFS an eine EC2-Instanz gemountet und die Rohdaten wurden schon einmal dorthin kopiert. In der Verarbeitung bildet dieses EFS den Ausgang der Datenverarbeitung.

Listing 1: EFS auf EC2-Instanz mounten.

```
#!/bin/bash
# from https://docs.aws.amazon.com/efs/latest/ug/wt1-create-ec2-resources.html

# security group 4 ec2
aws ec2 create-security-group \
--region eu-west-1 \
--group-name efs-dataprocessing1-ec2-sg \
--description "Amazon EFS dataprocessing 1, SG for EC2 instance" \
--vpc-id vpc-87ad55fe


# security group 4 efs
aws ec2 create-security-group \
--region eu-west-1 \
--group-name efs-dataprocessing1-mt-sg \
--description "Amazon EFS dataprocessing 1, SG for mount target" \
--vpc-id vpc-87ad55fe

# access to ec2 instance group from everywhere
aws ec2 authorize-security-group-ingress \
--group-id sg-098669727548dcedd \
--protocol tcp \
--port 22 \
--cidr 0.0.0.0/0 \
--region eu-west-1

# describe security group
aws ec2 describe-security-groups \
--region eu-west-1 \
--group-id sg-098669727548dcedd

# access to efs storrage from ec2 group
aws ec2 authorize-security-group-ingress \
--group-id sg-02778494bc39601d4 \
--protocol tcp \
--port 2049 \
--source-group sg-098669727548dcedd \
--region eu-west-1

# get subnet id
aws ec2 describe-subnets \
--region eu-west-1 \
--filters "Name=vpc-id,Values=vpc-87ad55fe"

# run ec2 instance
aws ec2 run-instances \
--image-id ami-047bb4163c506cd98 \
--count 1 \
--instance-type t2.micro \
--associate-public-ip-address \
--key-name bfh_root.pem \
--security-group-ids sg-098669727548dcedd \
--subnet-id subnet-f66512ac \
--region eu-west-1
```

[42] AWS Command Line Interface: Ein kommandozeilenorientiertes Werkzeug.

```
aws ec2 describe-instances \
--instance-ids i-09cb26ed64cdde683 \
--region eu-west-1

# ec2-54-75-53-151.eu-west-1.compute.amazonaws.com
# EFS ========================================
aws efs create-file-system \
--creation-token FileSystemForDataprocessing1 \
--tags Key=Name,Value=Dataprocessing1 \
--region eu-west-1


# create mount target
aws efs create-mount-target \
--file-system-id fs-5ace1d90 \
--subnet-id subnet-f66512ac \
--security-group sg-02778494bc39601d4 \
--region eu-west-1


# On instance amazon linux (ec2-user)
sudo yum -y update
sudo reboot # dont know why... like windows
sudo yum -y install nfs-utils

# On instance ubuntu (ubuntu)
sudo apt-get update
sudo apt-get install nfs-common

mkdir ~/efs-mount-point
cd ~/efs-mount-point
sudo chmod go+rw .

sudo mount -t nfs \
-o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport
    fs-5ace1d90.efs.eu-west-1.amazonaws.com:/ \
~/efs-mount-point

# CP from home to ec2 instance
scp -i bfh_root.pem \
/media/saibot/vortrag/bfh/buildings/*.tar \
ec2-user@54.75.53.151:/home/ec2-user/efs-mount-point/buildings/
```

D.2 Von der SPOT Instanz aus abfragen, was ihr Status ist

Von der Instanz aus können via RESTful API Metadaten der Instanz abgefragt werden. Bezüglich Interrupt einer Spot Instanz kann der Zustand *none*, *hibernate*, *stop* oder *terminate* sein. *none*, wenn nichts ansteht. Von da an, wo klar ist, dass die Instanz abgestellt werden wird, kann der Zeitpunkt ausgelesen werden.

Listing 2: Status der Instanz abfragen

```
TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H
    "X-aws-ec2-metadata-token-ttl-seconds: 21600") && curl -H "X-aws-ec2-metadata-token: $TOKEN"
    http://169.254.169.254/latest/meta-data/instance-action
```

D.3 XML Testen

Einfaches Skript, um zu testen, ob alle Dateien well-formed XML sind.

Listing 3: XML testen

```
import os
from lxml import etree
import logging
import threading

BASEPATH = '/home/ubuntu/processing/input/'
LOGGING_FILE = '/home/ubuntu/data/log/bad_xml.log'

logger = logging.getLogger('not_good')
logger.setLevel(logging.DEBUG)
ch = logging.StreamHandler()
fh = logging.FileHandler(LOGGING_FILE)
logger.addHandler(fh)
logger.addHandler(ch)

def try_xml(my_f):
   try:
      tree = etree.parse(my_f)
   except Exception as e:
      logger.info(my_f)

# r=root, d=directories, f=files
for r, d, f in os.walk(BASEPATH):
      for file in f:
         if '.kml' in file or '.dae' in file:
            my_f = os.path.join(r, file)
            x = threading.Thread(target=try_xml, args=(my_f,))
            x.daemon = True
            x.start()
```

D.4 Publikationsschritte in Ansible

Der ganze Code zum hier aufgelisteten Ausschnitt kann auf github.com eingesehen, resp. verwendet, werden:

Listing 4: Nebst dem Setup werden die drei Publikationsschritte mit Ansible gesteuert.

```
---
- name: set processing file
  vars:
    processing_step: 1
  template:
    src: ../templates/processing_step.txt.tpl
    dest: '{{ processing_step_file }}'
    mode: '0660'
    force: no



# ============================================
# First step
# ============================================
- name: first step
  debug:
    msg: "Step 1: Uncompressing and validating data"
  when: lookup('file', processing_step_file)|int <= 1

- name: create input folder
  file:
    path: /home/{{ my_user }}/processing/input
    state: directory
    mode: '0700'
  when: lookup('file', processing_step_file)|int <= 1

- name: unzip files
  shell: |
    for i in /home/{{ my_user }}/data/buildings/*.tar;
      do echo extracting ${i};tar -xzvf ${i} -C /home/{{ my_user }}/processing/input/;
    done
  when: lookup('file', processing_step_file)|int <= 1

- name: test if xml are wellformed
  command:
    cmd: python ../tests/test_xml_wellformed.py
  when: lookup('file', processing_step_file)|int <= 1

- name: check how many xml are false
  shell: wc -l /home/{{ my_user }}/data/log/bad_xml.log | cut -d " " -f1
  register: not_wellformed

- debug:
    var: not_wellformed

- name: end play when xml not valid
  fail:
    msg: there is a xml, that is not wellformed
  when: not_wellformed.stdout|int > 0 and lookup('file', processing_step_file)|int <= 1

- name: first step accomplished
  vars:
    processing_step: 2
  template:
    src: ../templates/processing_step.txt.tpl
    dest: '{{ processing_step_file }}'
    mode: '0660'
    force: yes
  when: lookup('file', processing_step_file)|int <= 1
# ============================================

# ============================================
```

```
# Second step
# ============================================
- name: Second step
  debug:
    msg: "Step 2: Processing raw data to web format (Cesium tiles)"
  when: lookup('file', processing_step_file)|int == 2

- name: create output folder
  file:
    path: /home/{{ my_user }}/processing/output
    state: directory
    mode: '0700'
    force: no
  when: lookup('file', processing_step_file)|int == 2

- name: create db folder
  file:
    path: /home/{{ my_user }}/processing/db
    state: directory
    mode: '0700'
    force: no
  when: lookup('file', processing_step_file)|int == 2

- name: create agi log folder
  file:
    path: /home/{{ my_user }}/data/log
    state: directory
    mode: '0700'
    force: no
  when: lookup('file', processing_step_file)|int == 2

- name: get ecr pw
  shell: |
    aws ecr get-login-password
  register: my_pw
  when: lookup('file', processing_step_file)|int == 2

- name: Log into private registry and force re-authorization
  docker_login:
    registry: https://483277333869.dkr.ecr.eu-west-1.amazonaws.com
    username: AWS
    password: "{{ my_pw.stdout }}"
    reauthorize: yes
  when: lookup('file', processing_step_file)|int == 2

- name: process to web format
  docker_container:
    name: analyticalgraphicsinc-swayze
    image:
        483277333869.dkr.ecr.eu-west-1.amazonaws.com/semesterarbeit:analyticalgraphicsinc-swayze
    volumes:
      - /home/{{ my_user }}/processing/input/:/var/app/input/
      - /home/{{ my_user }}/processing/output/:/var/app/output/
      - /home/{{ my_user }}/processing/db/:/var/app/db/
      - /home/{{ my_user }}/data/log/:/var/app/log/
    command: ["/root/.nvm/versions/node/v8.11.2/bin/node", "--max-old-space-size=20000",
        "/var/app/node_modules/.bin/roadhouse", "-i", "/var/app/input", "-o", "/var/app/output",
        "--db", "/var/app/db/database", "--clear-normals", "--face-normals", "--max-tiles",
        "1000", "-r", "UUID", "-r", "DATUM_AENDERUNG","-r","DATUM_ERSTELLUNG", "-r",
        "ERSTELLUNG_JAHR","-r", "ERSTELLUNG_MONAT","-r", "REVISION_JAHR","-r",
        "REVISION_MONAT","-r", "GRUND_AENDERUNG", "-r", "HERKUNFT","-r", "HERKUNFT_JAHR","-r",
        "HERKUNFT_MONAT","-r", "OBJEKTART","-r", "ORIGINAL_HERKUNFT","-r", "GEBAEUDE_NUTZUNG",
        "-r", "Longitude", "-r", "Latitude", "-r", "Height"]
    detach: false
  register: docker_output
  when: lookup('file', processing_step_file)|int == 2

- name: log dockeroutput
  shell: echo {{ docker_output }} > /home/{{ my_user }}/data/log/agi_log.log
  when: lookup('file', processing_step_file)|int == 2
```

```
- name: second step accomplished
  vars:
    processing_step: 3
  template:
    src: ../templates/processing_step.txt.tpl
    dest: '{{ processing_step_file }}'
    mode: '0660'
    force: yes
  when: lookup('file', processing_step_file)|int == 2
# ============================================

# ============================================
# Third step
# ============================================
- name: Third step
  debug:
    msg: "Step 3: Publishing data on S3"
  when: lookup('file', processing_step_file)|int == 3

- name: copy to s3
  command:
    cmd: aws s3 cp --recursive --content-encoding gzip /home/{{ my_user }}/processing/output
        s3://3d-tiles/preview/ch.swisstopo.swisstlm3d.3d/current/
  when: lookup('file', processing_step_file)|int == 3

- name: third step accomplished
  vars:
    processing_step: 4
  template:
    src: ../templates/processing_step.txt.tpl
    dest: '{{ processing_step_file }}'
    mode: '0660'
    force: yes
  when: lookup('file', processing_step_file)|int == 3
# ============================================
#
# ============================================
# Fourth step
# ============================================
- name: Fourth step
  debug:
    msg: "Step 4: Cleanup"
  when: lookup('file', processing_step_file)|int == 4

- name: remove folders
  file:
    path: /home/{{ my_user }}/processing/{{ item }}
    state: absent
  with_items:
  - db
  - input
  - output
  when: lookup('file', processing_step_file)|int == 4

- name: fourth step accomplished
  vars:
    processing_step: 5
  template:
    src: ../templates/processing_step.txt.tpl
    dest: '{{ processing_step_file }}'
    mode: '0660'
    force: yes
  when: lookup('file', processing_step_file)|int == 4
# ============================================
```