МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Савченко А.С.

«___»_____2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: «ELK stack на базі хмарних технологій з автоматизацією Ansible і Terraform»

Виконав: Бурачинський Олег Едуардович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер з ЄСКД (ЄСПД): Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕСИТЕТ

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Спеціальність 122 “Комп’ютерні науки та інформаційні технології”

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Савченко А.С.

“ ” _____2019р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Бурачинського Олега Едуардовича

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи): «ELK stack на базі хмарних технологій з автоматизацією Ansible і Terraform» затверджена наказом ректора №2175/ст. від 14.10.2019р..

2. Термін виконання проекту (роботи): з 14.10.2019р. по 09.02.2020р.

3. Вихідні данні до проекту (роботи):

Зміст пояснювальної записки (перелік питань, що підлягають розробці): вступ, засоби побудови систем моніторингу, система логування та моніторингу на базі elk. використання автоматизації ansible і terraform, висновок.

4. Перелік обов’язкового графічного матеріалу:

Схема обробки даних засобами Logstash, схема процесу обробки моніторингових даних мікросервісної системи, аналізатор обробки даних.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Аналіз літератури та джерел за темою дипломного проекту.	13.10.19р.– 20.10.19	
2.	Розроблення та затвердження плану дипломного проекту.	21.10.19– 22.10.19	
3.	Проведення консультації з науковим керівником щодо створення першого розділу.	23.10.19 – 27.10.19	
4.	Розробка розділу 1	30.10.19 – 22.11.19	
5.	Розробка розділу 2	23.11.19 – 08.12.19	
6.	Розробка розділу 3	09.12.19 – 22.12.19	
7.	Висновки та оформлення пояснювальної записки дипломного проекту.	25.12.19 – 29.12.19	
8.	Підписання необхідних документів у встановленому порядку.	15.01.20-19.01.20	
9.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	22.01.20 – 31.01.20	

7. Дата видачі завдання: 13.10.2019р.

Керівник дипломного проекту _____ Холявкіна Т.В.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Бурачинський О.Е.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту роботи «ELK stack на базі хмарних технологій з автоматизацією Ansible і Terraform» викладена на 95с., містить 49 рис., табл.1, 4 літературних джерел.

Ключові слова: ELK , STACK, TERRAFOM , ANSIBLE, KIBANA , LOGSTASH . ENSIBLE, ХМАРНІ ТЕХНОЛОГІЇ , GCP.

***Об'єкт дослідження*:** Досліження в галузі безпеки і збору інформації у систем з повною автоматизацією.

Предмет дослідження: Розроблення безвідказної системи логування

Мета роботи: Розробити правильну систему логування на базі ELK stack з використанням хмарних технологій Google Cloud Platform і безвідказною автоматизацією.

Методи дослідження: Використання Terraform для створення віртуальних машин на Google Cloud Platform і Ansible для заповнення цих машин ELK стеком.

Отримані результати: Була розробленна Автоматична система логування ELK на GCP для компанії яка стабільно працює і досі.

Результати дипломної роботи використовується на підприємстві.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних;

ОС - операційна система;

ПЗ - програмне забезпечення;

ПК - персональний комп’ютер;

СЗІ – система захисту інформації

ELK – Elasticsearch Logstash Kibana

PBR - policy based routing;

DCE - Data Communication Equipment;

DTE - Data Terminal EquipmentC;

VTP - VLAN Trunking Protocol;

NAT - Network Address Translation;

DNS - Domain Name System;

TFTP - Trivial File Transfer Protocol;

FTP - File Transfer Protocol;

HTTP - HyperText Transfer Protocol;

ACL - Access Control List;

ВСТУП

Навіщо потрібні логи?

Список, або журнал дій користувача, допомагає тестувальникам зрозуміти, що означає помилка, а також - звідки вона взялася.

Вам не потрібні логи? Все падіння ручних тестів легко відтворите самі?

Ок. Випустили продукт в продакшен, дзвонить користувач - у мене впало!

Ви по його плутаним поясненням і так і сяк систему корячітся - немає проблеми. Адже користувачів не вчать локалізувати проблеми, запам'ятовувати ВСЕ свої кроки (а часом бага відтворюється тільки при обов'язковому виконанні якогось пункту). Він просто "ввів дані", і все.

Коли у вас система логірует ваші дії, і, що більш корисно, повідомлення про помилки - коли можна почитати стек, помилку відтворювати стає набагато простіше. Часто, прочитавши повідомлення про помилку, вже розумієш, у чому проблема, не просячи допомоги розробників ...

Навіяно обговоренням локалізації проблем на тренінгу Олексія.

В принципі, іноді вистачає навіть не стільки логів, скільки журналу дій користувача.

це:

- Потрібно користувачам. Наприклад, керівники зможуть відстежувати те, що зробили їхні підлеглі.
- Потрібно нам. Щоб, отримавши дзвінок від користувачів (у нас проблема!), Можна було просто відкрити бойової стенд (якщо ми говоримо про веб-додатку) і подивитися журнал - а що, власне кажучи, зараз відбувалося? Це допоможе навести на думку, куди копати далі.

РОЗДІЛ 1

СИСТЕМА ЛОГУВАННЯ ТА МОНІТОРИНГ ДАНИХ

1.1 Аналіз критичних задач у процесах моніторингу мікросервісних додатків

Процеси моніторингу мікросервісів набагато складніші, ніж у систем, де функціональні можливості не розподіляються по-різному залежно один від одного. А через наявність великої кількості окремих сервісів, які не залежать один від одного і можуть докорінно відрізнятися внутрішньою архітектурою, мовою програмування або навіть ядром операційної системи, важливо узагальнити критичні моменти загального моніторингу між окремими послугами (табл. 1). Давайте визначимо деякі ключові завдання, які можуть виникнути в процесі моніторингу:

1) Попередня обробка всіх даних. Завдяки різноманітності програмного забезпечення, що використовується, дані моніторингу також різноманітні. Тому, перш ніж шукати будь-яку інформацію щодо цих даних, необхідно привести їх до єдиного формату, виділивши ключові дані, необхідні для пошуку та індексації.

2) Збір даних для моніторингу. У розподіленій системі є окремі частини, і іноді їх кількість може сягати сотень і навіть тисяч. Однак всі вони можуть бути незалежними в архітектурі, мовах програмування, операційних системах. Збір даних з усіх систем - непросте завдання.

3) Візуалізація оброблених даних. Проіндексовані та оброблені дані ще не готові до моніторингу. Вони доступні для пошуку, аналізу, але відсутність дружнього графічного призводить до труднощів для нетехнічного персоналу. Розробником цих продуктів є одна і та ж компанія і, відповідно, ці продукти комплексно працюють один з одним.

4) Пошук у великій кількості даних. Сотні сервісів можуть генерувати гігабайти моніторингу різної інформації щодня. Тому пошук таких обсягів є особливим завданням, якщо не застосовується оптимальна індексація.Джерело{https://serveradmin.ru/ustanovka-i-nastroyka-elasticsearch-logstash-kibana-elk-stack/}

Таблиця 1.1

Критичні задачі інтеграції моніторингу мікросервісних додатків та методи їх усунення

№ п/п	Задача	Існуючі методи та засоби вирішення	Методи та засоби, що пропонуються
1	Попередня обробка даних	Ручна обробка даних	Автоматизована обробка даних з Logstash
2	Збір (логів)	Обробка даних у кожному сервісі	Збір даних за допомогою Filebeat
3	Візуалізація прийнятих даних	Свої адміністративні панелі	Панель і моніторинг Kibana у режимі реального часу
4	Пошук великих обсягів даних	Індексовані поля NoSql/MySQL, grep	Індексація даних з Elasticsearch

В результаті аналізу критичних завдань, що виникають в процесі моніторингу мікросистем, можна визначити, що головним правилом моніторингу є зручність та швидкість виявлення проблем. Чим швидше буде виявлена помилка, тим менше шкоди та витрат буде витрачено на її локалізацію. У той же час ви можете стежити за ходом бізнес-процесів у системі мікросервісу в режимі реального часу.

1.2 Огляд сучасних засобів моніторингу мікросервісних додатків

Одним з найпопулярніших засобів моніторингу мікросервісної системи є впровадження процесів логування за допомогою програмного стеку ELK

(Elasticsearch Logstash Kibana). Цей програмний стек складається з трьох програмних продуктів:

1. Elasticsearch – це індексована база даних та рушій пошуку;

2. Logstash – це препроцессор вхідної інформації;

3. Kibana – це графічний інтерфейс відображення моніторингових даних.

Інструментом моніторингу є утиліта Logstash. Це дозволяє збирати дані будь-яких форматів та протоколів, фільтрувати ці дані та подавати їх далі для зберігання, обробки та індексації (рис.1.1).

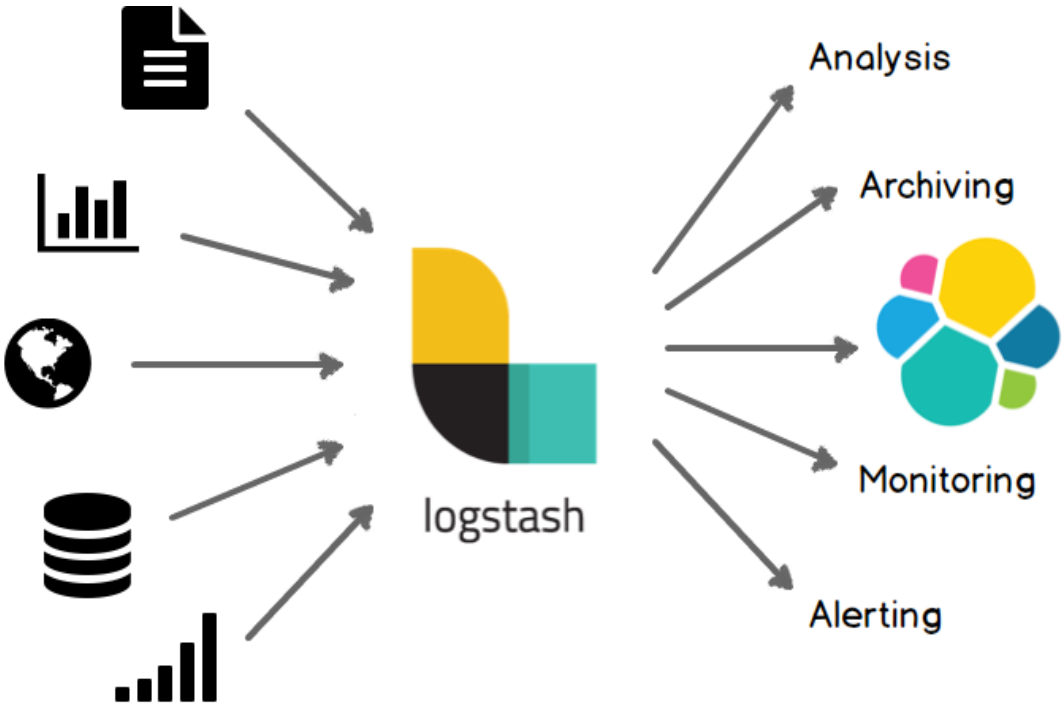


Рис.1.1 – Схема обробки даних засобами Logstash

Logstash підтримує понад 300 шаблонів введення. Подібним рішенням є утиліта Fluentd, але вона поступається універсальності Logstash та підтримці більшості заздалегідь підготовлених типів введення шаблону. Основні дані, які Logstash здатний отримувати, - це

наступні:

– логи мережевих пристроїв, фаєрволів, Windows event logs, syslog;

– логи веб-серверів Nginx , Apache та додатків на прикладі log4j для Java;

– метрики таких додатків як Ganglia, JMX, NetFlow, Collectd;

– обробка будь-яких пристроїв та додатків інфраструктури через UDP та TCP.

Logstash оптимізований для передачі даних в Elasticsearch для подальшої індексації та архівації. Зокрема, Elasticsearch є повноцінним механізмом пошуку та зберігання даних. Це легко масштабується та настроюється.

З самого розвитку цього продукту було здійснено повнотекстовий пошук, заснований на пошуковій системі Apache Lucene, одночасно забезпечуючи просте масштабування, тиражування тощо. Таким чином, цей продукт був інструментом для проектів даних з великим обсягом даних.

Зі збільшенням популярності та універсальності цього програмного забезпечення інженери почали використовувати його не тільки для повнотекстового пошуку, але і для централізованого зберігання різних даних аналітики та моніторингу.

Головною особливістю Elasticsearch є його проста конфігурація. Це програмне забезпечення працює відразу після простого налаштування та встановлення. Крім того, навколо цього програмного забезпечення було успішно сформовано спільноту розробників, яка може вирішувати складні кластери в кожному конкретному випадку.

Графічна панель Kibana використовується для відображення даних. Це один з основних елементів стека ELK, який відповідає за графічне відображення будь-якого типу інформації. Kibana призначений для роботи спільно з базою даних Elasticsearch і з її допомогою можна переглядати, шукати, взаємодіяти з даними, що зберігаються в базі даних Elastic. Цей програмний інструмент дозволяє обробляти величезну кількість даних за допомогою інтерфейсу та в режимі реального часу. Можна розширити, щоб використовувати нові функції.

1.3 Модель моніторингу даних мікросервісної системи

Ми пропонуємо графічне зображення та опис моделі обробки даних моніторингу мікросистеми (рис. 1.2)

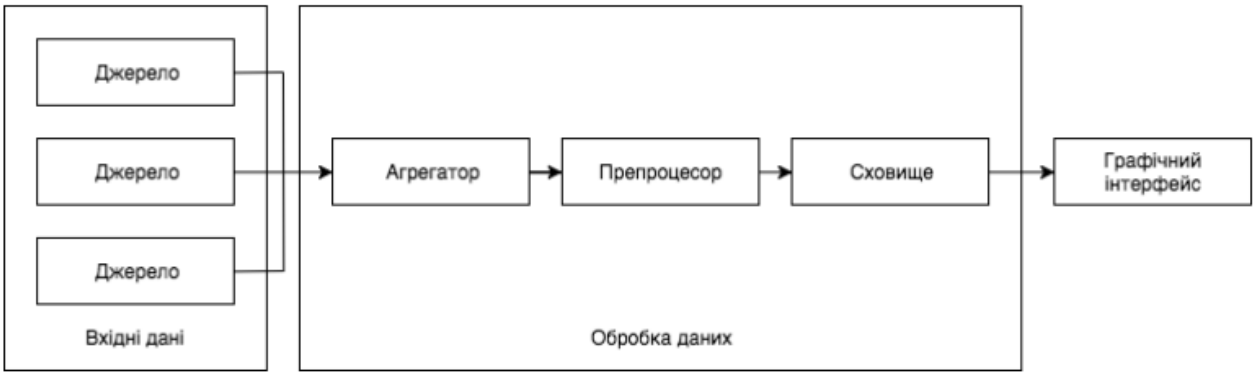


Рис.1.2 – Схема процесу обробки моніторингових даних мікросервісної системи

Взято з джерела {https://serveradmin.ru/ustanovka-i-nastroyka-elasticsearch-logstash-kibana-elk-stack/}

Схема процесу обробки моніторингових даних мікросервісної системи містить такі компоненти:

1) Джерело - сервіси, що генерують необроблені дані (Syslog). Тут створюються журнали. Багато програмного забезпечення, програм, середовищ генерують журнали різних форматів та розширень. Ці дані передаються агрегатору, встановленому в кожній службі, і цей агрегатор потрібно дотримуватися.

2) Агрегатор необроблених даних. Даний компонент є вхідною точкою для збору (агрегування) моніторингових даних. Він відповідає за транспортування моніторингових даних до препроцесора Logstash.

3) Агрегатний препроцесор даних (Logstash). Після входу в цю службу дані розбиваються на структурні частини з подальшими необхідними даними, які виводяться на вихід у єдиному форматі для подальшого збереження, індексації та аналізу в Elasticsearch.

4) Сховище оброблених даних (Elasticsearch). Централізоване сховище відповідає за збереження моніторингових даних. Тут надається можливість повнотекстового пошуку у великих обсягах даних з використанням пошукового механізму Apache Lucene.

5) Графічний інтерфейс взаємодії з даними (Kibana). Кінцевою точкою роботи моделі є графічне відображення проаналізованої інформації у вигляді

різноманітних графіків, схем, діаграм.

Останні три компоненти, сам стек ELK, повинні бути встановлені окремо від інших системних сервісів і повинні бути доступними для всіх служб. Після встановлення програмного забезпечення ELK наступним кроком є налаштування даних про продукт. Спочатку потрібно визначити вхід і вихід Logstash. Конфігурація за замовчуванням складається з трьох розділів –

“input”, “filter”, “output”. В секції “input” необхідно визначитись тим, звідки будуть надходити дані.

```
input {
 stdin {
 type => "rails"
 }
 syslog {}
}
```

Секція “filter” відповідає за розбиття моніторингових даних на структурні складові, які потім будуть передані на вихід.

```
filter {
 grok {
 match => ["message", "%{RAILS}"]
 }
 date {
 match => [ "timestamp", "YYYY-MM-dd HH:mm:ss Z"]
 target => "@timestamp"
 remove_field => "timestamp"
 }
```

Згодом після обробки дані потрапляють на вихід у централізоване сховище.

```
output {
 elasticsearch {
 host => “localhost:9200”
}
```

}

Далі вам потрібно скористатися Elasticsearch. Коли система готова імпортувати журнали, служби надсилають дані для обробки за запропонованою моделлю, вам потрібно налаштувати панель Kibana. На цьому кроці весь процес налаштування зводиться до коригування графіків та агрегації даних через саму панель. (рис. 3).

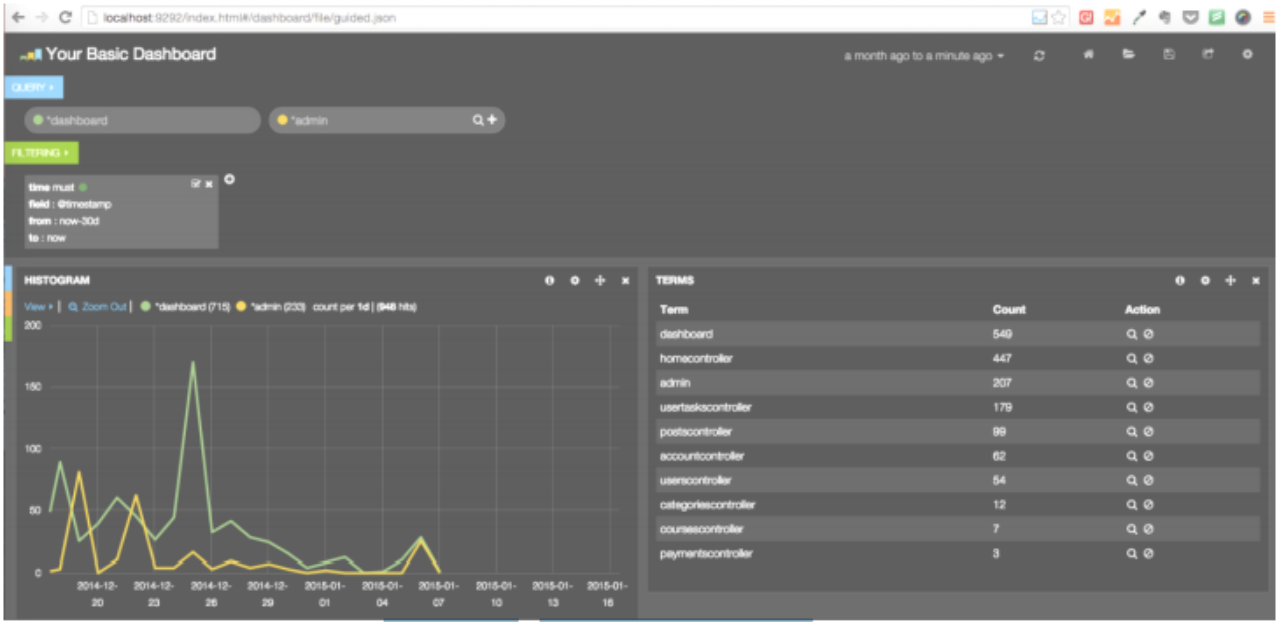


Рис.1.3 – Графічний інтерфейс Kibana

Наступна модель обробки даних моніторингу - найпростіший приклад використання програмного забезпечення ELK. Звичайно, ці можливості набагато ширші, ніж описано в цій роботі. Ця модель здатна обробляти дані сотень чи тисяч послуг, агрегувати дані різних програмних стеків, технологій та форматів. Однак слід зазначити, що представлена модель моніторингу є ефективною лише у проектах, де доцільно використовувати її у проектах з великою кількістю даних моніторингу.

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі були показані та проаналізовані інструменти та методи моніторингу мікросервісних програм. Зокрема, було проведено огляд сучасних засобів моніторингу програм, збору та обробки інформації з сервісів окремих систем для її аналітики. Ця модель моніторингу мікросервісної системи дозволяє швидко обробляти статистику інформації, реагувати на критичні події в роботі мікросервісів, а також аналізувати нову інформацію, щоб передбачити можливі проблеми у вашій компанії. Представлені та використані в роботі інструменти на достатньому рівні відповідають усім потребам моніторингу великих мікросистем. Також аналізуються різні типи завдань, які виникають під час здійснення процесів моніторингу, що пропонує різні способи їх уникнення.

.

РОЗДІЛ 2
ЗАСОБИ ПОБУДОВИ СИСТЕМ МОНІТОРИНГУ

2.1 Elasticsearch

Незважаючи на те що XXI століття принесло в наші будинки (не в усі, але все ж) гігабітні канали зв'язку і котиків на YouTube в дозволі 4K і 60 FPS, основою Мережі, наймасовішою його частиною, все ще залишаються текстові дані. Реферати та курсові роботи, технічні драми на Хабре і Stack Overflow, що сидить скалкою в ... Роскомнадзора Луркмор - все це текст, все це слова. А оскільки каталогізація в реальному світі очевидно кульгає, то без хорошого повнотекстового пошуку нікуди.

Що нам дає Elasticsearch?

Масштабованість і відмовостійкість. Elasticsearch легко масштабується. До вже існуючої системи можна на ходу додавати нові сервери, і пошуковий движок зможе сам розподілити на них навантаження. При цьому дані будуть розподілені таким чином, що при відмові будь-то з нод вони не будуть загублені і сама пошукова система продовжить роботу без збоїв.

Насправді воно навіть працює. У хіпстерском стилі «чувак, ось тобі три команди - користуйся ними і, будь ласка, не баріться, яке пекло відбувається всередині». І часто це прокатує. Нові Ноди підключаються буквально парою рядків в конфіги, майже як у Redis. Головне, майстри зі слейв не плутати, а то він візьме і мовчки потре всі дані :). При випаданні будь-яких серверів з кластера, якщо правильно були розподілені репліки даних, коректно налаштоване додаток продовжить пошук, як ніби нічого не сталося. Після того як сервер підніметься, він сам повернеться в кластер і підтягне останні зміни в даних.

Мультиарендність (англ. Multitenancy) - можливість організувати кілька різних пошукових систем в рамках одного об'єкта Elasticsearch. Причому

організувати їх можна абсолютно динамічно. Дуже цікава особливість, яка в окремих випадках стає визначальною при виборі пошукової системи. На перший погляд може здатися, що необхідності в цій особливості немає. Класичні системи пошуку типу Sphinx зазвичай індексують якусь одну базу з певним колом даних. Це форуми, інтернет-магазини, чати, різні каталоги. Всі ті місця, де пошук для всіх відвідувачів повинен бути ідентичним. Але насправді досить часто виникають ситуації, коли систем пошуку повинно бути більше однієї. Це або мультимовні системи, або системи, де є певна кількість користувачів, яким потрібно надавати можливість пошуку по їх персональних даних.

У першому випадку нам потрібно будувати окремі індекси по різних мов, окремо налаштовувати морфологію, стемінг, параметри нечіткого пошуку для того, щоб отримати максимально якісні результати для кожного з мов. У другому випадку в якості гіпотетичного прикладу можна взяти який-небудь корпоративний аналог Dropbox'a. Приходить клієнт, реєструється, заливає свої документи. Система їх аналізує, вгадує мову, парсить, заливає в окремий індекс пошукової системи, налаштовує параметри під потрібну мову. І далі клієнт може користуватися пошуком за своїми документами. Пошук буде працювати досить швидко, тому що даних в індексі окремого клієнта завжди буде менше, ніж в одному великому загальному, буде можливість динамічно такі індекси створювати, встановлювати різні пошукові параметри. Ну і дані клієнтів будуть ізольовані один від одного.

Операційна стабільність - на кожну зміну даних в сховищі ведеться логирование відразу на декількох осередках кластера для підвищення відмовостійкості і збереження даних у разі різного роду збоїв.

Відсутність схеми (schema-free) - Elasticsearch дозволяє завантажувати в нього звичайний JSON-об'єкт, а далі він вже сам все проіндексує, додасть в базу пошуку. Дозволяє не морочитися занадто сильно над структурою даних при швидкому прототіпіруванії.

RESTful api - Elasticsearch практично повністю управляється по HTTP за

допомогою запитів в форматі JSON.

2.1.1 Установка і використання

Встановити Elasticsearch простіше простого. Є готові репозиторії і для RHEL / Centos, і для Debian. Можна окремо встановити з тарбола.

```
 # rpm --import https://packages.elasticsearch.org/GPG-KEY-elasticsearch
# echo '
[elasticsearch-1.5]
name=Elasticsearch repository for 1.5.x packages
baseurl=http://packages.elasticsearch.org/elasticsearch/1.5/centos
gpgcheck=1
gpgkey=http://packages.elasticsearch.org/GPG-KEY-elasticsearch
enabled=1
' > /etc/yum.repos.d/elastic.repo
# yum install elasticsearch
# curl -XGET http://localhost:9200/
{
   "status" : 200,
   "name" : "Franklin Hall",
   "cluster_name" : "elasticsearch",
   "version" : {
      "number" : "1.5.0",
      "build_hash" : "544816042d40151d3ce4ba4f95399d7860dc2e92",
      "build_timestamp" : "2015-03-23T14:30:58Z",
      "build_snapshot" : false,
      "lucene_version" : "4.10.4"
   },
   "tagline" : "You Know, for Search"
}
```

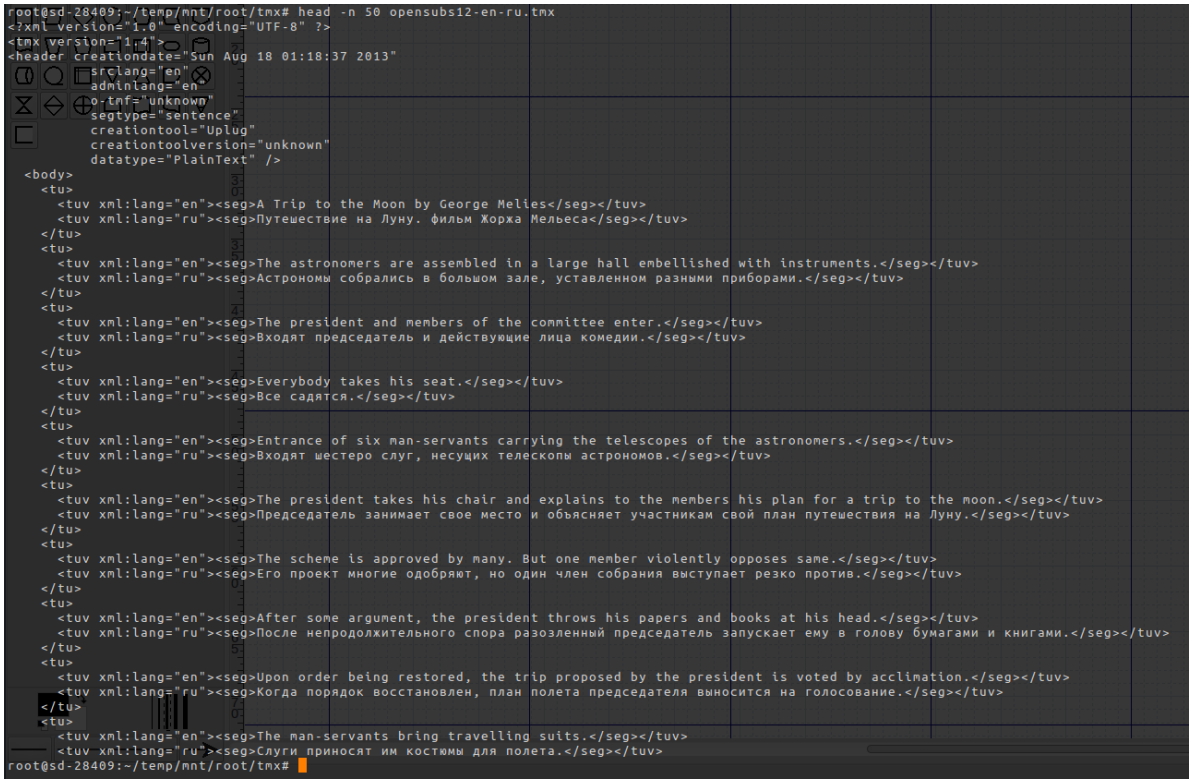


Рис.2.1 – HTTP запит

І вся подальша робота з ним відбувається за допомогою HTTP-запитів в JSON-форматі. Давай, наприклад, створимо новий індекс і заб'ємо в нього якісь тестові дані. Я взяв звідси англо-російський паралельний корпус, зібраний з даних OpenSubtitles.org. Формат TMX досить простий, описувати його окремо не стану. Напишу невеликий парсер на Python, який би розбирав файл і заливав дані в новий індекс:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import sys
from elasticsearch import Elasticsearch
import xml.etree.ElementTree as ET

TMX = sys.argv[1]

es = Elasticsearch()
```

```
with open(TMX) as source:
    context = iter(ET.iterparse(source, events=('start', 'end')))
    _, root = next(context)
    num = 1
    for event, elem in context:
        if event == 'end' and elem.tag == 'tu':
            doc = {
                'eng': elem[0][0].text.encode('utf-8'),
                'rus': elem[1][0].text.encode('utf-8')
            }
            res = es.index(index="demon-index", doc_type='tmx1', id=num, body=doc)
            num += 1
            elem.clear()
            root.clear()
```

На VPS'ке з чотирма гігамі пам'яті у флопс заливка чотирьох з половиною мільйонів документів (трохи більше 900 Мб даних в текстовому форматі) займає приблизно півтори години. В цілому дуже навіть непогано. Тепер накидали невеликий скриптик для зручного пошуку:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

from __future__ import unicode_literals
import sys
from elasticsearch import Elasticsearch

SEARCH_DATA = sys.argv[1]

es = Elasticsearch()

res = es.search(index="demon-index", body={'fields': ['eng', 'rus'], 'query': {'match': {'eng': SEARCH_DATA}}})

for item in res['hits']['hits']:
    print "%s - %s - %s" % (item['_score'], item['fields']['eng'][0], item['fields']['rus'][0])
```

І перевіряємо, що у нас вийшло:

 # python search.py "What the hell"

4.5126777 - What... what the hell? - Что... что за черт?

4.368808 - What the hell? What the hell? - Махнем не глядя?

4.149931 - What the hell Mae! - Что за спешка, Мэй?

4.149931 - What the hell? - It.... - Что за черт? - Это...

4.149931 - What the hell happened? - Давай вернемся назад.

 # python search.py "God for our right"

3.0920234 - (several) For our God. - (некоторые) Для нашего Бога.

3.0859475 - For our God. - Для нашего Бога.

2.7642622 - Proceed, troops, trusting God for our right, for the Romanian Independence. - Идите вперед, ибо Господь на нашей стороне, во имя независимости Румынии. Карл.

2.654899 - Our fathers were our models for God. - Заткнись.

2.4985123 - God, he liveS right in our neighborhood. - Ничего себе, он живет прямо в нашем районе.

Перша колонка - вага отриманого значення, інші дві - знайдені результати. А тепер шукаємо по-російськи:

 # python ./search.py "Ой все"

4.835098 - - Oh, come on. - - Ой, все, хватит.

4.835098 - Oops, it's okay. - Ой, все в порядке.

4.8212147 - Oh, that's okay. - Ой, все в порядке.

4.8188415 - - I'm a bit rusty. - - Ой, как все запущено.

4.8188415 - Oh, it's over? [sighs] - Ой, уже все закончилось?

 # python ./search.py "Чот приуныл"

4.629143 - The whole sad act. - Приуныл.

2.3740823 - Choate, then yale, - Чот, потом Йель,

2.3145716 - Hey, why so glum, William? - Эй, чего приуныл, Уильям.

1.8609953 - Now that the Booroos are missing, he has lost track of life, you see? - Когда Бурусы не пришли, он совсем приуныл.

1.573388 - (Shivers) (Leah) Erm, that's a good look, Andy. - Ээ... что-то он у тебя приуныл, Энди.

Як бачиш, непогано шукає вже прямо з коробки, для якогось блогу або невеликого форуму цілком підійде. А якщо якість видачі здасться недостатньо

високим (а до такої думки рано чи пізно приходять майже всі), то Elasticsearch надає велику кількість можливостей для подальшого тюнінгу аналізаторів і пошукових алгоритмів.

2.1.2 Аналізатори

Вибір правильного аналізатора для обробки своїх даних - це щось майже на межі мистецтва. Зсередини кожен аналізатор являє собою своєрідний конвеєр, що складається з декількох обробників:

- символьної фільтрації;
- токенізації;
- фільтрації отриманих токенов.
- Головна мета будь-якого аналізатора - з довгого речення, переобтяженого непотрібними деталями, вичавити основну суть і отримати список токенов, які б її відбивали.

Рис.2.2 – аналізатор обробки даних

Приблизну схему роботи конвеєра можна побачити на зображенні поблизу. Аналіз починається з опціональних символьних фільтрів. Це, наприклад, переклад тексту в нижній регістр або підстановка слів. Отриманий результат передається токенізатору, головному і єдиному обов'язковому елементу аналізатора. Тут пропозиція очищається від розділових знаків, розбивається на окремі слова-маркери, які можуть або зберігати наявну форму, або обрізатися тільки до основи слова, або оброблятися ще якимось чином в залежності від токенізатора. Після токенізатора отримані дані відправляються на подальшу фільтрацію, якщо вже виконаних маніпуляцій буде недостатньо.

Elasticsearch з коробки надає відразу кілька різних аналізаторів. Якщо їх буде мало, то нестандартні аналізатори можна буде додати за допомогою

спеціального API. Ось базовий приклад нестандартного аналізатора:

```
PUT /demon-index/_settings
{
 "index": {
  "analysis": {
   "analyzer": {
    "customHTMLSnowball": {
     "type": "custom",
      "char_filter": [
        "html_strip"
      ],
      "tokenizer": "standard",
      "filter": [
        "lowercase",
        "stop",
        "snowball"
      ]
    }}}}}
```

Що робить цей аналізатор:

1. Прибирає HTML-теги з вихідного тексту за допомогою символьного фільтра html_strip.
2. Ділить текст на слова і прибирає пунктуацію за допомогою токенізатора standart.
3. Перекладає всі токени в нижній регістр.
4. Прибирає токени, що знаходяться в списку стоп-слів.
5. Проводить стемінг залишилися токенов за допомогою фільтра snowball.

І дивись, як це виглядає на живому прикладі. Візьмемо пропозицію «Мама мила раму, поки собака доїдала сосиску» і розберемо його по пунктах (рис. «Мама мила-мила ...»).

Детальніше про надані разом з Elasticsearch анализаторах і фільтрах можна прочитати в офіційній документації. Тут описувати не візьмуся, так як деталей там дуже багато.

2.1.3 Нечіткий пошук

Обробка природних мов - це робота з постійними неточностями. Здебільшого пошукові движки намагаються аналізувати граматичні структури різних мов, освоювати певні патерни, характерні для тієї чи іншої мови. Але пошукова система постійно стикається з запитами, що виходять за рамки усталених правил орфографії та морфології. Найчастіше це або помилки, або банальна безграмотність. Найпростіший приклад нечіткого пошуку - це знамените «Можливо, Ви мали на увазі ...» в Гуглі. Коли людина шукає «Пгод Ві Кутського», а йому показують погоду в Іркутську.

Основою нечіткого пошуку є відстань Дамерау - Левенштейна - кількість операцій вставки / видалення / заміни / транспозиції для того, щоб один рядок збіглася з іншого. Наприклад, для перетворення «Пгод Ві Кутського» в «погода в Іркутську» така відстань було б дорівнює трьом - дві вставки і одна заміна.

Відстань Дамерау - Левенштейна - це модифікація класичної формули Левенштейна, в якій спочатку була відсутня операція транспозиції (перестановки двох сусідніх символів). Elasticsearch підтримує можливість використання в нечіткому пошуку обох варіантів, за замовчуванням включено використання відстані Дамерау - Левенштейна.

При роботі з нечітким пошуком також не варто забувати і про те, як Elasticsearch (та й будь-який інший пошуковий движок в принципі) працює зсередини. Всі дані, які в індекс, спершу проходять обробку аналізатором, лематизації, стемінг. В індекс вже складаються тільки «обривки» вихідних даних, що містять максимум сенсу при мінімумі знакового обсягу. І вже по цих самих уривків згодом проводиться пошук, що при використанні нечіткого пошуку може давати досить курйозні результати.

Наприклад, при використанні аналізатора snowball під час нечіткого пошуку по слову running воно після проходу через стемінг перетвориться в run, але при цьому по ньому не знайдеться слово runninga, так як для збігу з ним потрібно більше двох правок. Тому для підвищення якості роботи нечіткого пошуку краще використовувати найпростіший Стеммер і відмовитися від пошуку по синонімів.

Elasticsearch підтримує кілька різних способів нечіткого пошуку:

- match query + fuzziness option. Додавання параметра нечіткості до звичайного запиту на збіг. Аналізує текст запиту перед пошуком;
- fuzzy query. Нечіткий запит. Краще уникати його використання. Більше схожий на пошук по Стемм. Аналіз тексту запиту перед пошуком не проводиться;
- fuzzy_like_this / fuzzy_like_this_field. Запит, аналогічний запит more_like_this, але підтримує нечіткість. Також підтримує можливість аналізу ваг для кращого ранжирування результатів пошуку;
- suggesters. Припущення - це не зовсім тип запиту, швидше за інша операція, яка працює зсередини на нечітких запитах. Може використовуватися як спільно зі звичайними запитами, так і самостійно.

2.1.4 CJK

CJK - це три літери болю західних систем повнотекстового пошуку і людей, які хочуть ними скористатися. CJK - це скорочення для Chinese, Japanese, Korean. Три основних східних мови, що становлять разом майже 10% сучасного інтернету. Вони відрізняються від звичних західних мов практично всім - і писемністю, і морфологією, і синтаксисом. Все це, зрозуміло, викликає деякі проблеми при розробці різних систем обробки природних мов, в тому числі і пошукових систем.

У Elasticsearch в цій області справи теж йдуть непогано. Є вбудований аналізатор CJK зі стемінг, є можливість використовувати нечіткий пошук. Ось тільки якщо за текстами на корейською та японською мовами ще хоч якось можна шукати «за класичними правилами» (тобто ділимо на слова, відкидаємо спілки / приводи, що залишилися слова токенізіруем і заганяємо в індекс), то ось з китайським, в якому слова в реченні не прийнято розділяти пробілами, все куди складніше.

Для пошукової системи все пропозицію на китайському залишається однією цілою одиницею, за якими проводиться пошук. Наприклад, пропозиція «Мері і я гуляємо по Пекіну» виглядає ось так:

玛丽和我走北京周边.

Дев'ять символів без пробілів, 18 байт в UTF-8. У нормальній всесвіту це прокатали б за одне слово, але не тут. Якщо стратегічно розставити прогалини в потрібних місцях, то пропозиція стане виглядати ось так:

玛丽 和 我 走 北京 周边.

Шість слів. З цим вже можна було б працювати. Ось тільки прогалини в китайському ніхто не використовує. Можна намагатися розділяти пропозиції на слова в автоматичному режимі (вже навіть існує пара готових рішень), але і тут тебе чекатимуть неприємності. Деякі склади, стоять у реченні поряд, можуть, в залежності від того, як їх розділити пробілами, складатися в різні слова і різко міняти зміст речення. Візьмемо для прикладу речення 我 想到 纽约:

2.1.5 Безпека

У Elasticsearch немає ніякої вбудованої системи авторизації та обмеження прав доступу. Після установки він за замовчуванням вішається на порт 9200 на всі доступні інтерфейси, що робить можливим не тільки повністю відвести у тебе все, що знаходиться в пошуковій базі, але і, чисто теоретично, через виявлену дірку залізти в систему і там начудуватися. До версії 1.2 така можливість була доступна прямо з коробки (см. CVE-2014-3120) і напружуватися не було взагалі ніякої потреби. В 1.2 за замовчуванням виконання скриптів в пошукових запитах відключено, але поки що і це не рятує.

Зовсім недавно ми спостерігали ботнет на еластиком версій в тому числі і 1.4 і вище. Судячи з усього, використовувалася уразливість CVE-2015-1427. У версії 1.4.3 її ніби як закрили, але, сам розумієш, покладатися на удачу в таких справах не варіант (насправді так, поки писалася ця стаття, свіжопоставленний еластік версії 1.5.0 на тестових віртуалкою у мене встигли поламати уже на другий день. Вішай сервіс тільки на локальні IP, всі необхідні підключення ззовні обмежуй тільки довіреними адресами, фільтруй пошукові запити, своєчасно оновлювати. Порятунок потопаючих - справа рук самих потопаючих.

До теми збереження даних також варто згадати про бекапи. Можливості резервного копіювання та відновлення вбудовані в сам *Elasticsearch*, причому

досить цікаво. Перед початком створення резервних копій потрібно еластиком повідомити, куди вони будуть складатися. У місцевих термінах це називається «створити репозиторій»:

```
$ curl -XPUT 'http://localhost:9200/_snapshot/my_backup' -d '{
  "type": "fs",
  "settings": {
     "location": "/es-backup",
     "compress": true
  }
}'
```

тут

type - тип сховища, куди вони будуть складатися. З коробки є тільки файлова система. За допомогою додаткових плагінів можна реалізувати заливку на *AWS S3, HDFS і Azure Cloud;*

location - куди зберігати нові бекапи;

compress - стискати чи бекапи. Толку не дуже багато, так як за фактом стискає тільки метаінформацію, а файли даних залишаються незжатими. За замовчуванням включено.

Після того як створений репозиторій, можна почати бекапить:

```
$ curl -XPUT "localhost:9200/_snapshot/my_backup/snapshot_1?wait_for_completion=true"
```

Такий запит створює бекап з назвою snapshot_1 в репозиторії my_backup.

Відновити дані можна наступним чином:

```
$ curl -XPOST "localhost:9200/_snapshot/my_backup/snapshot_1/_restore"
```

Причому знімки статки створюються інкрементальні. Тобто в перший раз створюється повний бекап, а далі при наступних бекапить фіксується тільки різниця стану між поточним моментом і моментом попереднього бекапа. Якщо у тебе кластер з декількома майстрами, то сховище сховища має бути відкрите між усіма майстрами (тобто, при зберіганні на файлової системі, це повинен бути будь-якого роду мережевий диск, доступний всім майстрам). Файли сховища я б теж з диска куди-небудь бекапить про всяк випадок.

2.2 Logstash

Logstash - це конвеєр обробки даних, який отримує сирі дані (наприклад, логи) з одного або декількох джерел, обробляє їх і покращує фільтрами, а потім відправляє результат одному або декільком одержувачам. Elastic рекомендує в якості одержувача використовувати Elasticsearch, але насправді можна використовувати все, що душі завгодно: STDOUT, WebSocket, звичайні сокети, черги повідомлень - вибір величезний.

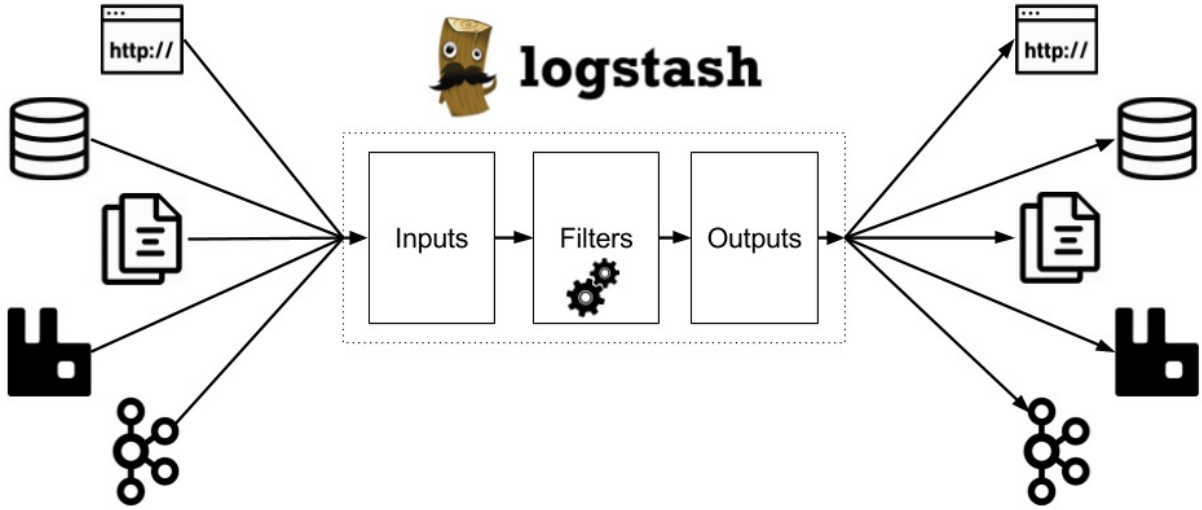


Рис.2.3 – робота Logstash

2.2.1 Установка

Якщо на хості встановлена Java, то можна просто завантажити архів, розпакувати його і запустити bin / logstash -f logstash.conf. Для запуску, правда, знадобиться файл конфігурації, але для простих прикладів зійде щось на input {STDIN {}} output {STDOUT {}}.

Так, любителям контейнерів можна не обтяжувати себе скачкою архівів і використовувати заводський Docker образ:

docker run -it logstash -e 'input { stdin { } } output { stdout { } }'

2.2.2 Конфігурація, кодеки і «Hello World!»

Отже, архів викачаний, що далі? Як я згадав вище, для запуску Logstash потрібен файл конфігурації. Наприклад, такий:

input {

stdin { }

}

output {

stdout { }

}

З ним Logstash буде брати дані з консолі, робити свою магію за замовчуванням, і видавати результат назад в консоль.

Hello world

Я знайшов у себе на хості трохи логів від Apache2, так що чому б не згодувати їх Logstash і не подивитися, що з цього вийде?

bin/logstash -e 'input { stdin { } } output { stdout {} }'

#....

#05:38:59.948 [Api Webserver] INFO logstash.agent - Successfully started Logstash API endpoint {:port=>9600}

$ 172.17.0.1 - - [11/Feb/2017:04:41:22 +0000] "GET / HTTP/1.1" 200 3525 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"

#2017-02-13T05:39:12.684Z 269a27a16415 172.17.0.1 - - [11/Feb/2017:04:41:22 +0000] "GET / HTTP/1.1" 200 3525 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"

А вийде багато нудного тексту. Logstash запустився, йому на вхід увійшла рядок (4) з балкою від Apache2, і вона тут же пішла на вихід з двома новими полями: часовою міткою 2017-02-13T05: 39: 12.684Z і ім'ям хоста 269a27a16415.

2.2.3 Кодеки

Але ми можемо зробити висновок більш читабельним. Logstash дозволяє підключати до вводу та виводу різні кодеки, які форматують потік

тексту, але не змінюють його значення: додають відступи, архівують, переводять в JSON, і тд. У нашому випадку ми можемо використовувати rubydebug для того, щоб він відформатував результат і показав, як саме Logstash «бачить» дані, з якими працює.

Конфігурація зміниться зовсім трохи, але результат стане набагато людяніше:

```
{
  stdout {
    codec => rubydebug
  }
}
$ bin/logstash -e 'input { stdin { } } outut { stdout {codec => rubydebug} }'
#....
$ 172.17.0.1 - -] "GET / HTTP/1.1" 2030 32525 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_14_2) AppleWebKit/52357.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36"
# {
#    "@timestamp" => 2017-02-12T05:24:21.270Z,
#      "@version" => "1",
#          "host" => "31190306c1eb",
#          "message" => "172.17.0.1 - - [11/Feb/2017:04:41:22 +0000] \"GET / HTTP/1.1\" 200 3525 \"-\" \" (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like 0.2924.87 Safari/537.36\""
#}
```

2.2.4 Вивід результатів в Elasticsearch

Перш ніж рушити далі, варто спробувати щось ще. На початку посту я сказав, що одержувачів даних може бути більше одного, і взагалі ми можемо писати результати прямо в Elasticsearch. Це як раз той випадок, коли можна одним рухом тіла підтвердити обидва пункти.

Так вже виявилося, що в сусідньому з Logstash контейнером у мене

влаштувався чистий Elasticsearch з айпішка 172.19.0.2 і портом 9200. Щоб підключити їх один до одного, потрібно додати всього пару рядків в конфігурацію Logstash:

```
...
output {
  stdout {
    codec => rubydebug
  }
  elasticsearch {
    hosts => ["142.18.0.2:9200"]
  }
}
```

Тепер, якщо перезапустити Logstash і знову згодувати йому Apache2 логів, в Elasticsearch з'явиться дещо щодо цікаве:

```
$ curl 142.18.0.2:9200/_cat/indices
#yellow open logstash-2017.02.12 rgQub7hsS0qq-FBj3HA2Rg 5 1 5 0 20.1kb 20.1kb
```

По-перше, там з'явився новий індекс - logstash-2017.02.12. По-друге, запустивши за цим індексом пошук, ми отримаємо ті ж дані, що Logstash писав в STDOUT:

```
$ curl 142.18.0.2:9200/logstash-2017.02.12/_search?pretty
#{
#   "hits" : [
#     {
#       "_index" : "logstash-2017.02.12",
#       "_type" : "logs",
#       "_source" : {
#         "@timestamp" : "2017-02-12T05:24:21.272Z",
#         "@version" : "1",
#         "host" : "31190306c1eb",
#             "message" : "172.17.0.1 - - [11/Feb/2017:04:41:22 +0000] \"GET
```

/icons/ubuntu-logo.png HTTP/1.1\" 200 3623 \"http://localhost/\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36\""

#}

2.2.5 Оброблення данних фільтрами

Поки Logstash справлявся з переправленням логів з точки А в точку Б, але не робив нічого для того, щоб це дані стали хоч трохи корисніше. Прийшов час змін. Крім INPUT і OUTPUT в конфігурацію можна додати FILTER, і ось там-то основна магія і почнеться.

Фільтри є практично для всього: для агрегації метрик, маскування чутливих даних (на кшталт імен та номерів кредитних карт), додавання і видалення полів, знаходження IP по доменному імені, знаходження адреси по IP (тримайтеся, коментатори!), І т.п. Але для нас добре б почати з розбору рядка Apache2 балки на її компоненти: IP, шлях, user agent, і так далі. Фільтр, який займається подібною роботою, називається grok.

2.3 GROK

Grok - це основний інструмент для додання форми і структури сполшному масиву тексту. Працює це так: з попередньо встановлених патернів (IP, Number, Word, ...) ми збираємо рядок-шаблон, яка за структурою повторює ту, яку ми збираємося парсити. Наприклад, якби мої логи виглядали так:

0 127.0.0.1 /default.html

1 172.1.0.9 /

То я б використовував такий шаблон для їх розбору:

%{NUMBER:sequence} %{IP:client} ${URIPATHPARAM:target}

Після обробки фільтром в логах з'явилися б нові поля: sequence, client і target. Ті самі, які вказали в шаблоні.

Крім елементарних паттернів начебто Number є більш складні, які описують відомі формати логів цілком. Наприклад, COMBINEDAPACHELOG описує рядок логу від Apache2. Ось її-то і можна спробувати в дії.

Нова конфігурація:

```
input {
  stdin {}
}
filter {
  grok {
    match => {
      "message" => "%{COMBINEDAPACHELOG}"
    }
  }
}
output {
  stdout {
    codec => rubydebug
  }
  elasticsearch {
    hosts => ["142.18.0.2:9200"]
  }
}
```

І реакція Logstash на неї:

```
{
        "request" => "/",
          "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36\"",
           "auth" => "-",
          "ident" => "-",
           "verb" => "GET",
        "message" => "172.17.0.1 - - [11/Feb/2017:04:41:22 +0000] \"GET / HTTP/1.1\" 200 3525 \"-\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36\"",
        "referrer" => "\"-\"",
```

"@timestamp" => 2017-02-12T05:41:18.687Z,

"response" => "200",

"bytes" => "3525",

"clientip" => "172.17.0.1",

"@version" => "1",

"host" => "31190306c1eb",

"httpversion" => "1.1",

"timestamp" => "11/Feb/2017:04:41:22 +0000"

}

2.3.1 Додаткові налаштування GROK

Но обработку можно сделать еще лучше. В середине распаршеного лога затесалось поле message, которое дублирует всё, что мы только что распарсили. Хорошо бы его убрать. А в grok как раз есть дополнительные настройки, одна из которых называется remove_field — как раз то, что нам нужно:

filter {

grok {

match => {

"message" => "%{COMBINEDAPACHELOG}"

}

remove_field => ["message"]

{

"request" => "/icons/ubuntu-logo.png",

"agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36\"",

"auth" => "-",

"ident" => "-",

"verb" => "GET",

"referrer" => "\"http://localhost/\"",

"@timestamp" => 2017-02-12T05:42:55.999Z,

"response" => "200",

"bytes" => "3623",

"clientip" => "172.17.0.1",

"@version" => "1",

"host" => "31190306c1eb",

"httpversion" => "1.1",

"timestamp" => "11/Feb/2017:04:41:22 +0000"

}

Тепер результат просто чудовий. Але ми можемо зробити його ще краще.

2.3.2 geoip фільтр

Як випливає з назви, geoip конвертує IP адреса в координати на карті і супутній вуличний адресу. Я замінив докеровскі айпішка одного з логів на свою зовнішню, включив geoip, і пропустив лог через Logstash ще раз:

filter {

grok {

...

}

geoip {

source => "clientip"

І сам вивід

{

"request" => "/",

"agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/602.4.8 (KHTML, like Gecko) Version/10.0.3 Safari/602.4.8\"",

"geoip" => {

"timezone" => "America/Toronto",

"latitude" => 43.4464,

"continent_code" => "NA",

"city_name" => "Oakville",

"country_code2" => "CA",

```
        "country_name" => "Canada",
      "country_code3" => "CA",
       "region_name" => "Ontario",
          "location" => [
       [0] -79.7593,
       [1] 43.4464
     ],
       "postal_code" => "L6M",
        "longitude" => -79.7593,
       "region_code" => "ON"
  },
       "auth" => "-",
      "ident" => "-",
```

geoip взяв clientip поле, яке створив grok, і потім додав кілька власних полів. Мені навіть не довелося щось викачувати з ваших інтернетів, щоб це запрацювало.

А тепер, коли оброблені дані стікаються в Elasticsearch, ми можемо використовувати його пошук на повну котушку і почати ставити питання. Чи були відвідувачі з Oakville?

```
curl -s 142.18.0.2:9200/logstash-2017.02.12/_search?q=oakville | json_pp
#{
#  ...
#    "hits" : [
#      {
#        "_id" : "AVow3zOW6fU5oFCNC7kH",
#        "_score" : 1.5404451,
#        "_type" : "logs",
#        "_index" : "logstash-2017.02.12",
#        "_source" : {
#          "geoip" : {
```

 # ...

 # "city_name" : "Oakville",

 # },

 # "response" : "200",

 # "@timestamp" : "2017-02-12T05:50:18.333Z",

 # "agent" : "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/602.4.8 (KHTML, like Gecko) Version/10.0.3 Safari/602.4.8\"",

 # "request" : "/",

#...

 # }

 # }

 #],

Звичайно були. А скільки HTTP помилок було вчора? Чи намагався хтось зайти в адмінку блогу без попиту? Скільки раз? Де він живе?

2.4 Kibana

Kibana - це веб-вікно в дані Elasticsearch. Вона допоможе скласти запити, побудувати графіки і дашборда, і навіть покаже трохи статистики. Разом з плагіном X-Pack в неї можна додати інструменти моніторингу, повідомлень і пару інших корисних речей.

Як і з будь-яким іншим виключно UI інструментом, розповідати про кожну кнопку і картинку було б дивно, тому ми просто пройдемося по основним фічам. Але спочатку Kibana потрібно якось встановити.

Установка

Як і всі інші компоненти ELK стека, Kibana вимагає Java. Як тільки вимога задоволено, процес установки зводиться до банального скачай-і-розпакуйте. Після цього можна запускати bin / kibana і гештальт, можна сказати, завершений.

Але! Без Elasticsearch, що живе неподалік, Kibana дивно марна. Якщо Elasticsearch все-таки є поруч, його URL потрібно покласти в config / kibana.yml (опція elasticsearch.url), і після цього Kibana буде повністю готова до трудових

подвигів.

1. Правда, є трохи більш простий спосіб отримати працездатну Кібанов. Якщо запустити її і Elasticsearch в Docker контейнерах,

2. Назвати контейнер з Elasticsearch - elasticsearch,

3. І покласти їх в одну мережу,

то вони знайдуть один одного і будуть спокійно взаємодіяти. Всі три умови може виконати всього один docker-compose файл:

```
version: '2'
services:
  elasticsearch:
    image: elasticsearch:5.2
    ports:
      - "9200:9200"
  kibana:
    image: kibana:5.2
    ports:
      - "5601:5601"
```

Запускаємо його через docker-compose up, даємо elasticsearch секунд 10-15 доробити свої справи, і можна радіти Кібанов за адресою 127.0.0.1:5601.

2.4.1 Добавлення текстових данних

Ще одна міні-проблема пов'язана з установкою - тестові дані. Що не кажи, Kibana - інструмент для візуалізації та роботи з даними. Немає даних - немає візуалізації. На превелике щастя для нас в офіційному Гайд є кілька тестових датасета і інструкція, як їх встановлювати. Інструкція дуже проста: скачати, розпакувати, імпортувати.

2.4.2 Индекс-паттерн

В хранении логов есть такая традиция — группировать их по датам и хранить в отдельных файлах. Logstash эту традицию поддерживает и потому будет писать логи в разные Elasticsearch индексы, например mylogs-2017-01-

01, mylogs-2017-01-02, mylogs-2017-01-03, и т.д. Но для визуализации удобнее всё-таки работать с одним индексом, а не с коллекцией.

Для этого в Кибане есть такая штука как индекс-паттерн. Это всего лишь строка-шаблон, охватывающая несколько индексов. Например, шаблон для примера выше — mylogs-*.

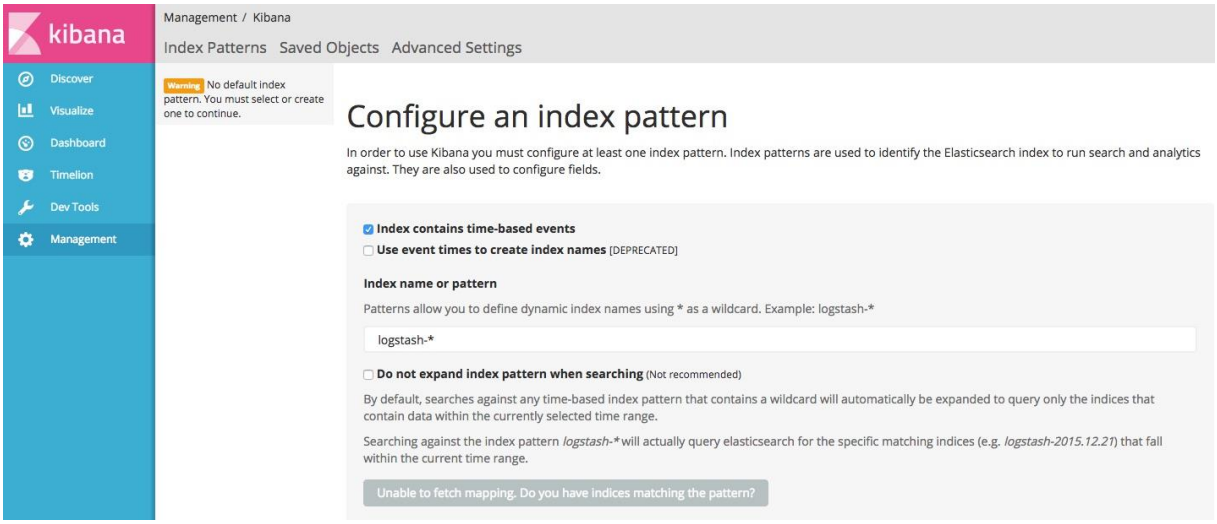


Рис.2.4 – відображення менеджера Kibana

Як тільки ми його збережемо, mylogs- * можна буде використовувати як самий звичайний індекс.

2.4.3 Перегляд даних

Найперший пункт основного меню Кіба - «Discover» - одночасно і найкорисніше місце для ознайомлення зі своїми даними. З ходу там буде і рядок пошуку, яка розуміє Lucene мову запитів, і гістограма розподілу даних індексу за часом, і навіть найпростіша статистика по деяких полях.

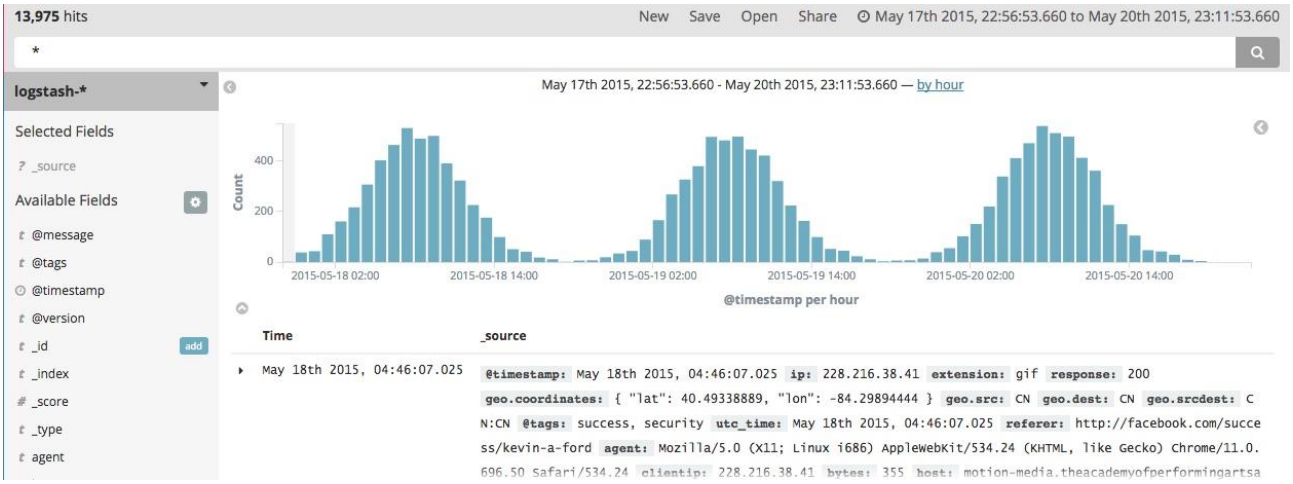


Рис.2.5 – відображення роботи логів

Синтаксис Lucene дивно простий. * - це знайти все. 13 - це знайти 13 по всім полям. response: 404 AND request: *. css - знайти всі запити до CSS файлів, які нічого не знайшли (404). Все просто.

Ще одна зручність полягає в тому, що список полів зліва дуже навіть інтерактивний і старанно корисний. Він буде намагатися показати статистику по полях, якщо це має сенс, і пропонувати намалювати хоч який-небудь підходить під тип даних графік.

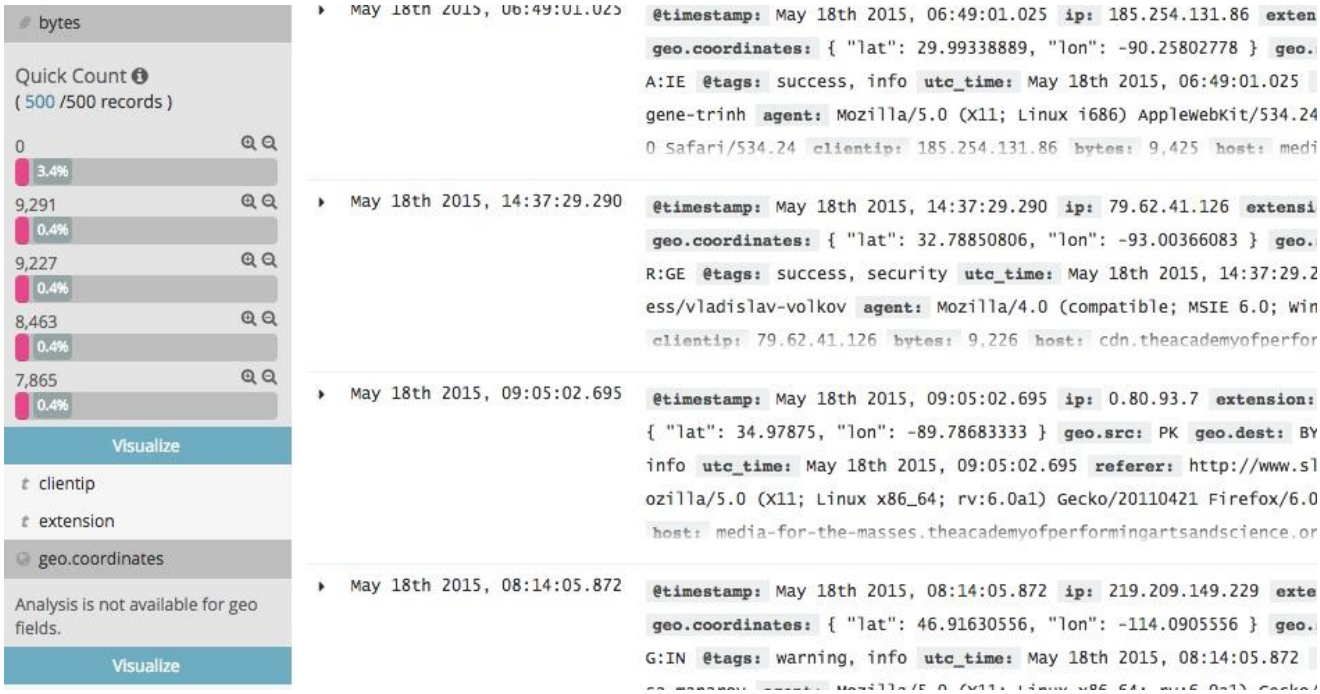


Рис.2.6 – інтерактивний пошук

Наприклад, кнопка «visualize» для числових даних покаже звичайну гістограму. Для географічних ж координат вона постарається трохи більше і відразу відкриє карту.

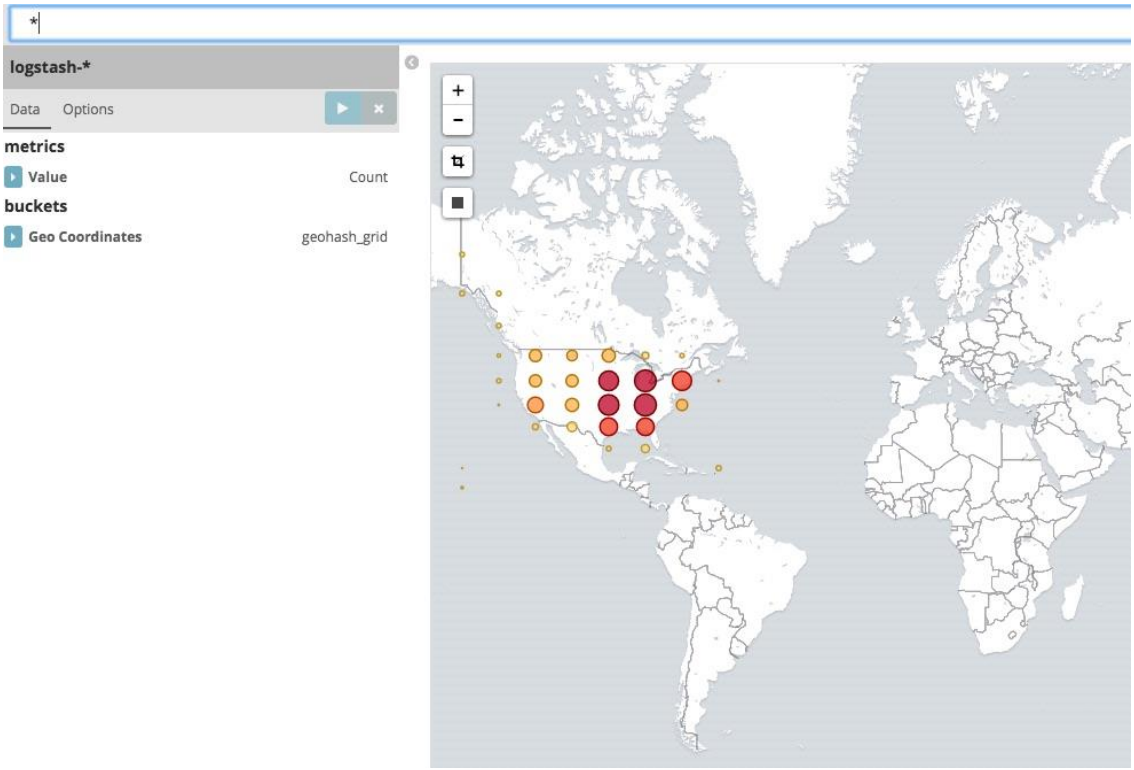


Рис.2.7 – відображення робти геолокації

2.4.4 Графіки та інші візуалізації

У Kibana повно і інших способів ці дані відобразити. Від лічильників і графіків, до шматків Markdown і хмар тегів. Більшість з них ховаються на сторінці «Visualize»:

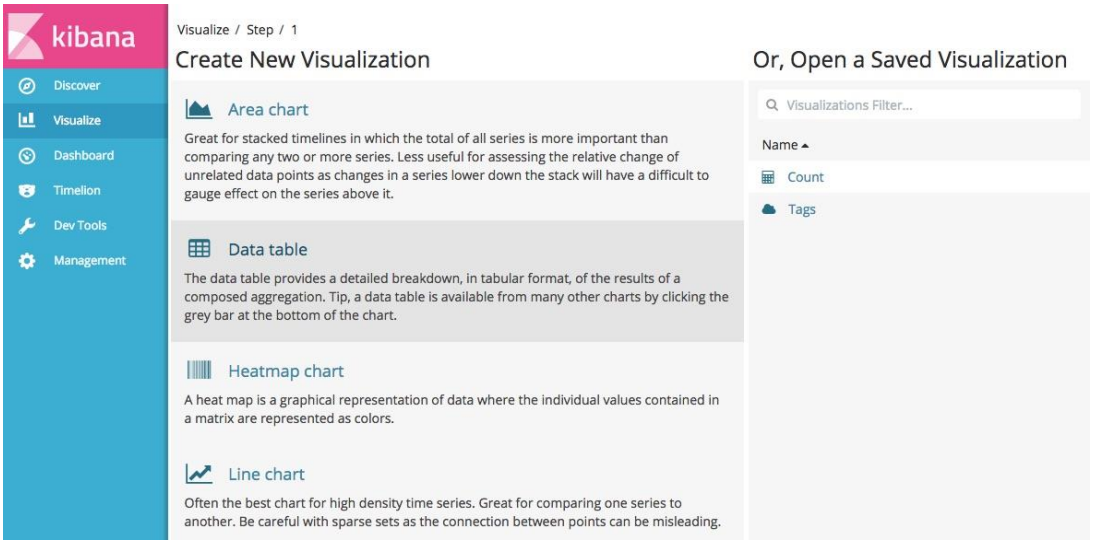


Рис.2.8 – віалізація в Kibana

Нічого незвичайного в цих компонентах немає, але я б відзначив пару моментів. По-перше, практично все в компонентах можна клікнути, і тоді обраний елемент піде прямо пошуковий запит. Для будь-якого самотнього графіка це навряд чи буде корисним, але для дашборда цілком може стати в нагоді.

По-друге, той же налаштований графік можна зберегти, і потім перевикористати в тих же дашборда.

2.4.5 Дашборда

Сторінка дашборда дозволяє зібрати збережені графіки та інші компоненти з даними в одному місці.

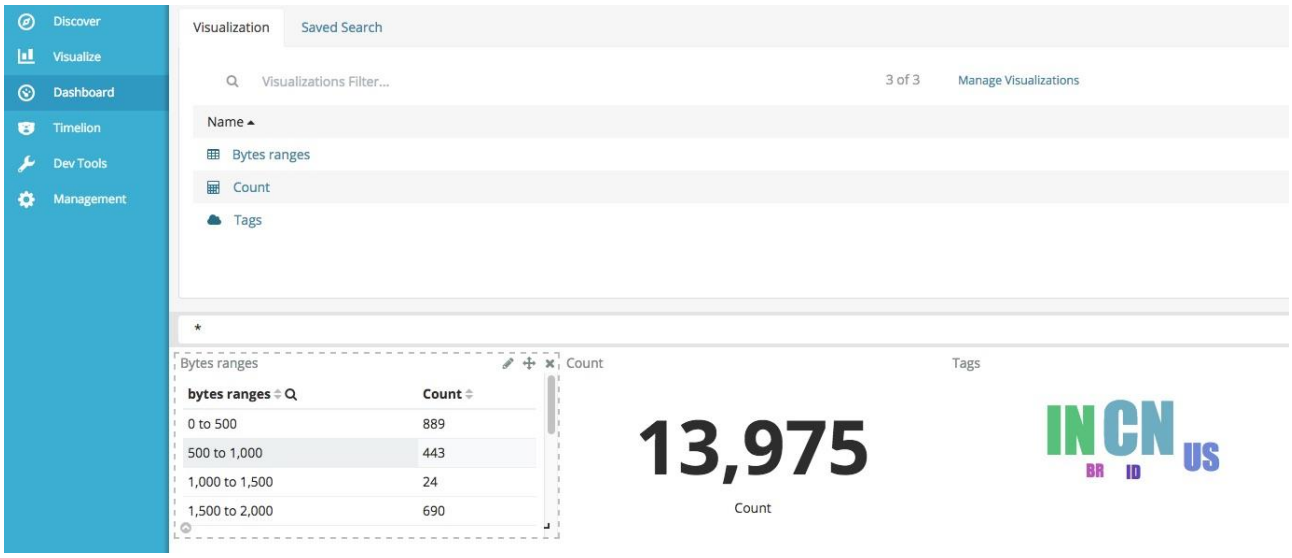


Рис. 2.9 - дашборла Kibana

Можливість клікнути практично на будь-якому шматку даних тут зберігається, але тепер це перетворюється в корисний інструмент. Наприклад, можна клікнути по чому-небудь в хмарі тегів, наприклад по US, і дашборда цілком буде показувати дані тільки по US.

До того ж рядок пошуку тут теж є, і вибірку можна уточнити ще більше:

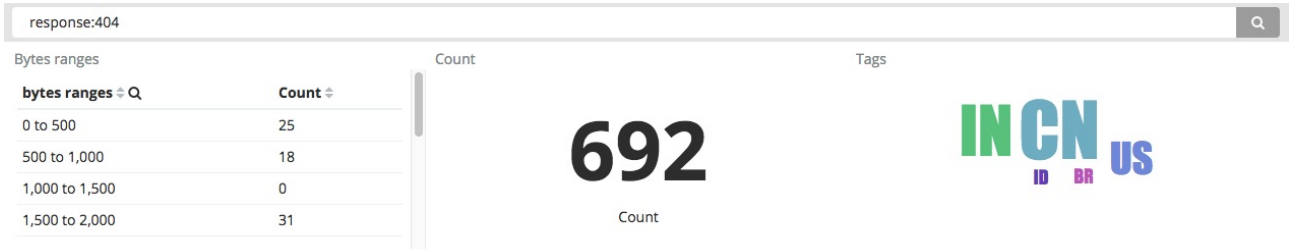


Рис. 2.10 – пошук у дашборді Kibana

2.4.6 Timelion

Timelion - дуже цікава штука. Це інструмент для аналізу часових рядів з власною мовою для вибірки і маніпуляції цих самих рядів. На практиці за допомогою цієї мови можна зробити запит в кілька індексів, пропустити результат через функцію, вибрати колір і заголовки, і покласти на один графік. Є короткий і цілком зрозумілий блог пост, яка пояснювала б навіщо Timelion взагалі існує.

Рис. 2.11 – Робота Timelion Kibana

ВИСНОВОК ДО РОЗДІЛУ 2

Logstash належить до класу інструментів, які не створюють особливого враження своїм описом, але трохи закохує в себе після того, як з ними пограти. Варто трохи покомбінірувать входи, виходи і фільтри, як стає дико цікаво, що ж ще можна буде вилита зі своїх даних.

Так, Logstash прекрасно працює з Elasticsearch, але і джерелами і одержувачами даних може бути величезний набір сервісів, починаючи з черг повідомлень і закінчуючи TCP сокетами.

Kibana - це приємний інструмент для практичної роботи з даними, які ми встигли зібрати в Elasticsearch. Вона дозволяє робити запити, малювати графіки і навіть намагається вгадати наступний крок і показати статистику по полях, або посилання на відповідний графік або карту.

На відміну від Grafana, Кіба може працювати тільки з Elasticsearch. Але так як вона дуже добре інтегрується з Elastic стеком цілком, то це, в принципі, не недолік.

Але в цій картині вселенського щастя пропущений один елемент - призначений для користувача інтерфейс. Що не кажи, аналізувати логи з командного рядка - не продуктивне заняття. Щоб це виправити, в наступний раз ми подивимося в бік останнього компонента ELK стека - Kibana.

РОЗДІЛ 3
СИСТЕМА ЛОГУВАННЯ ТА МОНІТОРИНГУ НА БАЗІ ELK. ВИКОРИСТАННЯ АВТОМАТИЗАЦІЇ ANSIBLE І TERRAFORM

3.1.1 Підготовка та встановлення Java

Всі компоненти системи, за винятком агентів на серверах, написані на java, тому нашу настройку почнемо з установки Oracle Java 8. Ця версія обрана, тому що підтримується всіма версіями Elasticsearch. Детальніше про сумісність програмних продуктів дивіться в окремій таблиці з документації.

Ubuntu

Підключаємо репозиторій з Java 8.

```
# add-apt-repository -y ppa:webupd8team/java
```

Оновлюємо список пакетів і встановлюємо Oracle Java 8.

```
# apt update
# apt install oracle-java8-installer
```

Перевіряємо версію

```
# java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

3.1.2 Підготовка та встановлення Elasticsearch

Встановлюємо ядро системи зі збору логів - Elasticsearch. Його установка дуже проста за рахунок готових пакетів під всі популярні платформи.

Копіюємо собі публічний ключ сховища:

```
# wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | apt-key add -
```

Якщо у вас немає пакета apt-transport-https, то треба встановити:

```
# apt install apt-transport-https
```

Додаємо репозиторій Elasticsearch в систему:

```
# echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | tee -a /etc/apt/sources.list.d/elastic-6.x.list
```

Встановлюємо Elasticsearch на Debian або Ubuntu:

```
# apt update && apt-get install elasticsearch
```

Після установки додаємо elasticsearch в автозавантаження і запускаємо.

```
# systemctl daemon-reload
# systemctl enable elasticsearch.service
# systemctl start elasticsearch.service
```

Перевіряємо, запустився він:

```
# systemctl status elasticsearch.service
```

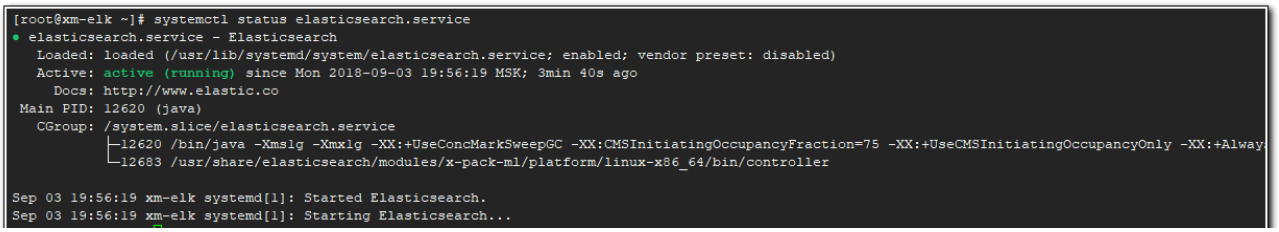


Рис.3.1 – відображення роботи Elasticsearch

Якщо все в порядку, то переходимо до налаштування Elasticsearch.

3.1.3 Налаштування Elasticsearch

Налаштування Elasticsearch знаходяться в файлі /etc/elasticsearch/elasticsearch.yml. На початковому етапі нас будуть цікавити наступні параметри:

```
path.data: /var/lib/elasticsearch # директория для хранения данных
network.host: 127.0.0.1 # слушаем только локальный интерфей
```

За замовчуванням Elasticsearch слухає все наявні мережеві інтерфейси. Нам це не потрібно, так як дані в нього буде передавати logstash, який буде встановлений локально. Звертаю окрему увагу на параметр для директорії з даними. Найчастіше вони будуть займати значне місце, інакше навіщо нам Elasticsearch. Подумайте заздалегідь, де ви будете зберігати логи. Всі інші настройки я залишаю дефолтними.

Після зміни налаштувань, треба перезапустити службу:

```
# systemctl restart elasticsearch.service
```

Дивимося, що вийшло:

```
# netstat -tulnp | grep 9200
tcp6       0      0 127.0.0.1:9200          :::*                    LISTEN      14130/java
```

Elasticsearch повис на локальному інтерфейсі. Причому я бачу, що він слухає ipv6, а про ipv4 ні слова. Але його він теж слухає, так що все в порядку. Переходимо до установки kibana.

3.1.4 Підготовка та встановлення Kibana

Далі встановлюємо web панель Kibana для візуалізації даних, отриманих з Elasticsearch. Тут теж нічого складного, репозиторій і готові пакети є під всі популярні платформи. Репозиторії і публічний ключ для установки Kibana будуть такими ж, як в установці Elasticsearch. Але я ще раз все повторю для тих, хто буде встановлювати тільки Kibana, без всього іншого. Це продукт закінчений і використовується не тільки в зв'язці з Elasticsearch.

Установка Kibana на Debian або Ubuntu така ж, як і на центос - підключаємо репозиторій і ставимо з deb пакета. Додаємо публічний ключ:

```
# wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | apt-key add -
```

Додаємо репозиторій Kibana:

```
# echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | tee -a /etc/apt/sources.list.d/elastic-6.x.list
```

Запускаємо установку

```
# apt update && apt install kibana
```

Додаємо Kibana в автозавантаження і запускаємо:

```
# systemctl daemon-reload
# systemctl enable kibana.service
# systemctl start kibana.service
```

Перевіряємо стан запущеного сервісу:

```
# systemctl status kibana.service
```

За замовчуванням, Kibana слухає порт 5601. Тільки не поспішайте його перевіряти після запуску. Кіба стартує довго. Зачекайте приблизно хвилину і перевіряйте.

```
# netstat -tulnp | grep 5601
tcp        0      0 127.0.0.1:5601          0.0.0.0:*               LISTEN      27401/node
```

3.1.5 Налаштування Kibana

Файл з настройками Кіба розташовується по шляху - /etc/kibana/kibana.yml. На початковому етапі можна взагалі нічого не чіпати і залишити все як є. За замовчуванням kibana слухає тільки localhost і не дозволяє підключатися віддалено. Це нормальна ситуація, якщо у вас буде на цьому ж сервері встановлений nginx в якості reverse proxy, який буде приймати підключення і проксювати їх в Kibana . Так і треба робити в продакшені, коли системою будуть користуватися різні люди з різних місць. За допомогою nginx можна буде розмежовувати доступ, використовувати сертифікат, налаштовувати нормальне доменне ім'я і т.д.

Якщо ж у вас це тестова установка, то можна обійтися без nginx. Для цього треба вирішити Кіба слухати зовнішній інтерфейс і приймати підключення. Змініть параметр server.host, вказавши ip адресу сервера, наприклад ось так:

```
server.host: "10.1.4.114"
```

Якщо хочете, щоб вона слухала все інтерфейси, вкажіть в якості адреси 0.0.0.0. Після цього Kibana треба перезапустити:

```
# systemctl restart kibana.service
```

Тепер можна зайти в веб інтерфейс за адресою http://10.1.4.114:5601.

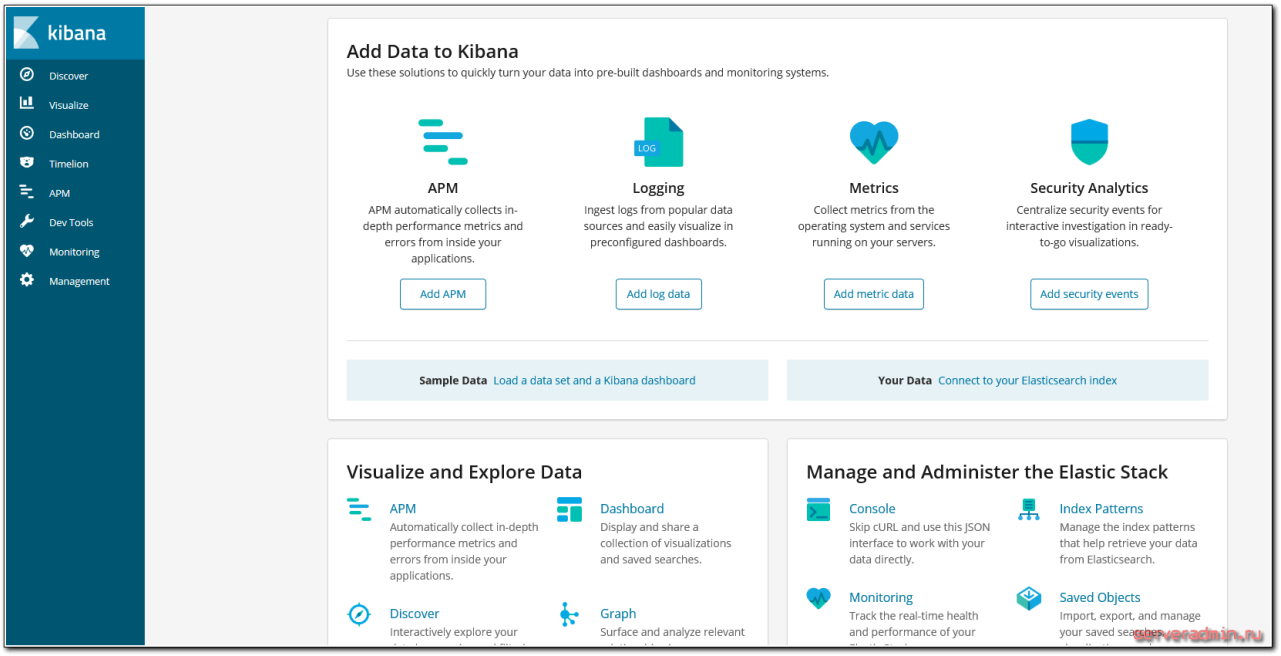


Рис.3.2 – інтерфейс Kibana

Можна продовжувати настройку і тестування, а коли все буде закінчено, запустити nginx і налаштувати проксінг. Я настройку nginx залишу на самий кінець. В процесі настройки буду підключатися безпосередньо до Kibana.

3.1.6 Встановлення і налаштування Logstash

Logstash встановлюється так само просто, як Elasticsearch і Kibana, з того ж сховища. Не буду ще раз показувати, як його додати. Просто встановимо його і додамо в автозавантаження.

```
# apt install logstash
```

Додаємо logstash в автозавантаження:

```
# systemctl enable logstash.service
```

Запускати поки не будемо, треба його спочатку налаштувати. Основний конфиг logstash лежить за адресою /etc/logstash/logstash.yml. Я його чіпати не буду, а всі налаштування буду за змістом розділяти за різними конфігураційним файлів в директорії /etc/logstash/conf.d. Створюємо перший конфиг input.conf, який буде описувати прийом інформації з beats агентів.

```
input {
  beats {
    port => 5044
  }
}
```

Тут все просто. Вказую, що приймаємо інформацію на 5044 порт. Цього достатньо. Якщо ви хочете використовувати ssl сертифікати для передачі логів по захищеним з'єднанням, тут додаються параметри ssl. Я буду збирати дані з закритого периметра локальної мережі, у мене немає необхідності використовувати ssl.

Тепер зазначимо, куди будемо передавати дані. Тут теж все відносно

просто. Малюємо конфиг output.conf, який описує передачу даних в Elasticsearch.

```
output {
      elasticsearch {
        hosts    => "localhost:9200"
        index    => "nginx-%{+YYYY.MM.dd}"
      }
#stdout { codec => rubydebug }
}
```

Тут все просто - передавати всі дані в elasticsearch під зазначеним індексом з маскою у вигляді дати. Наскільки я зрозумів, розбивка індексів по днях і по типам даних зручна з точки зору управління даними. Потім легко буде виконувати очистку даних по цих індексах. Я закоментувавши останній рядок. Вона відповідає за логування. Якщо її включити, то всі, хто вступає дані logstash буде відправляти додатково в системний лог. У centos це / var / log / messages. Використовуйте тільки під час налагодження, інакше лог швидко розростеться дублями даних, що надходять.

Залишається останній конфіг з опис обробки даних. Тут з'являється невелика вулична магія, в якій я розбирався деякий час. Розповім нижче. Малюємо конфіг filter.conf.

```
filter {
 if [type] == "nginx_access" {
   grok {
      match => { "message" => "%{IPORHOST:remote_ip} - %{DATA:user} \[%{HTTPDATE:access_time}\] \"%{WORD:http_method} %{DATA:url} HTTP/%{NUMBER:http_version}\" %{NUMBER:response_code} %{NUMBER:body_sent_bytes} \"%{DATA:referrer}\" \"%{DATA:agent}\"" }
    }
  }
  date {
      match => [ "timestamp" , "dd/MMM/YYYY:HH:mm:ss Z" ]
  }
  geoip {
       source => "remote_ip"
       target => "geoip"
       add_tag => [ "nginx-geoip" ]
  }
}
```

Перше, що робить цей фільтр, парсит логи nginx за допомогою grok, якщо вказаний відповідний тип логів, і виділяє з логу ключові дані, які записує в певні поля, щоб потім з ними було зручно працювати. Спочатку я не зрозумів, навіщо це потрібно. У документації до filebeat добре описані модулі, що йдуть в комплекті, які все це і так вже вміють робити з коробки, потрібно тільки підключити відповідний модуль.

Виявилося, що модулі filebeat працюють тільки в тому випадку, якщо ви відправляєте дані безпосередньо в elasticsearch. На нього ви теж ставите відповідний плагін і отримуєте відформатовані дані. Але у нас працює проміжну ланку logstash, який приймає дані. З ним, як я зрозумів, плагіни filebeat не працюють, тому доводиться окремо в logstash парсити дані. Це не дуже складно, але тим не менше. Як я зрозумів, це плата за зручності, які дає logstash. Якщо у вас багато розрізнених даних, то відправляти їх безпосередньо в elasticsearch не так зручно, як з використанням предобработки в logstash. Якщо я не правий, прошу мене поправити. Я так зрозумів цей момент.

Для фільтра grok, який використовує logstash, є зручний дебагер, де можна подивитися, як будуть Парс ваші дані. Покажу на прикладі одного рядка з конфіга nginx. Наприклад, візьмемо такий рядок з логу nginx:

```
180.163.220.100 - travvels.ru [05/Sep/2018:14:45:52 +0300] "GET /assets/galleries/26/1.png HTTP/1.1" 304 0 "https://travvels.ru/ru/glavnaya/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36"
```

І подивимося, як її розпарсити правило grok, яке я використовував в конфіги вище.

```
%{IPORHOST:remote_ip} - %{DATA:user} \[%{HTTPDATE:access_time}\] \"%{WORD:http_method} %{DATA:url} HTTP/%{NUMBER:http_version}\" %{NUMBER:response_code} %{NUMBER:body_sent_bytes} \"%{DATA:referrer}\" \"%{DATA:agent}\"
```

Власне, результат ви можете самі побачити в дебаггера. Фільтр розпарсити лог і на виході сформує json, де кожному значенню буде присвоєно своє поле, по якому потім зручно буде в Еластік будувати звіти і робити вибірки. Тільки не забувайте про формат логів. Наведене мною правило відповідає дефолтного формату main логів в nginx. Якщо ви якимось чином модифікували формат логів, внесіть зміни в grok фільтр.

Сподіваюся зрозуміло пояснив роботу цього фільтра. Ви можете таким чином парсити будь логи і передавати їх в еластікс. Потім на основі цих даних будувати звіти, графіки, дашборда. Я планую розпарсити як мені потрібно поштові логи postfix і dovecot.

Далі використовується модуль date для того, щоб виділяти дату з вступників логів і використовувати її в якості дати документа в elasticsearch. Робиться це для того, щоб не виникало плутанини, якщо будуть затримки з доставкою логів. В системі повідомлення будуть з однією датою, а всередині балки буде інша дата. Незручно розбирати інциденти.

В кінці я використовую geoip фільтр, який на основі ip адреси, який ми отримали раніше за допомогою фільтра grok і записали в поле remote_ip, визначає географічне розташування. Він додає нові мітки і записує туди географічні дані. Для його роботи мені було досить описати його в конфіги і завантажити базу адрес в папку / etc / logstash.

```
# wget http://geolite.maxmind.com/download/geoip/database/GeoLite2-City.mmdb.gz
# gunzip GeoLite2-City.mmdb.gz
```

Закінчили настройку logstash. Запускаємо його:

```
# systemctl start logstash.service
```

Можете перевірити на всяк випадок лог /var/log/logstash/logstash-plain.log, щоб переконатися в тому, що все в порядку. Тепер налаштуємо агенти для відправки даних.

3.1.7 Встановлення Filebeat для відправки логів в Logstash

Встановимо першого агента Filebeat на сервер з nginx для відправки логів веб сервера на сервер з ELK. Ставити можна як із загального сховища, який ми підключали раніше, так і окремо пакети. Як ставити - вирішувати вам. У першому випадку доведеться на всі хости додавати репозиторій, але зате потім зручно оновлювати пакунки. Якщо підключати репозиторій не хочеться, можна просто завантажити пакет і встановити його.

```
# curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.4.0-amd64.deb
# dpkg -i filebeat-6.4.0-amd64.deb
# apt install filebeat
```

Після установки малюємо приблизно такий конфіг /etc/filebeat/filebeat.yml для відправки логів в logstash.

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
     - /var/log/nginx/*-access.log
  fields:
    type: nginx_access
  fields_under_root: true
  scan_frequency: 5s
- type: log
  enabled: true
  paths:
     - /var/log/nginx/*-error.log
  fields:
    type: nginx_error
  fields_under_root: true
  scan_frequency: 5s
output.logstash:
  hosts: ["10.1.4.114:5044"]
xpack.monitoring:
  enabled: true
  elasticsearch:
    hosts: ["http://10.1.4.114:9200"]
```

Деякі пояснення до конфігу, так як він не зовсім дефолтний і мінімалістичний. Я його трохи модифікував для зручності. По-перше, я розділив логи access і error за допомогою окремого поля type, куди записую відповідний тип балки: nginx_access або nginx_error. Залежно від типу змінюються правила обробки в logstash. Плюс, я включив моніторинг і для цього вказав адресу elastichsearch, куди filebeat передає дані моніторингу безпосередньо. Показую це для вас просто з метою продемонструвати можливість. У мене всюди окремо працює моніторинг на zabbix, так що великого сенсу в окремому моніторингу немає. Але ви подивіться на нього, можливо вам він стане в нагоді. Щоб моніторинг працював, його треба активувати в відповідному розділі в Kibana - Monitoring. І не забудьте запустити elasticsearch на зовнішньому інтерфейсі. У початковому налаштуванні я вказав слухати тільки локальний інтерфейс.

Запускаємо filebeat і додаємо в автозавантаження.

```
# systemctl start filebeat
# systemctl enable filebeat
```

Перевіряйте лог за адресою / var / log / filebeat / filebeat. Він дуже інформатівет. Якщо все в порядку, побачите список всіх логів в директорії / var / log / nginx, які знайшов filebeat і почав готувати до відправки. Якщо все зробили правильно, то дані вже потекли в elasticsearch. Ми їх можемо подивитися в Kibana. Для цього відкриваємо web інтерфейс і переходимо в розділ Discover. Так як там ще немає індексу, нас перенаправить в розділ Managemet, де ми зможемо додати.

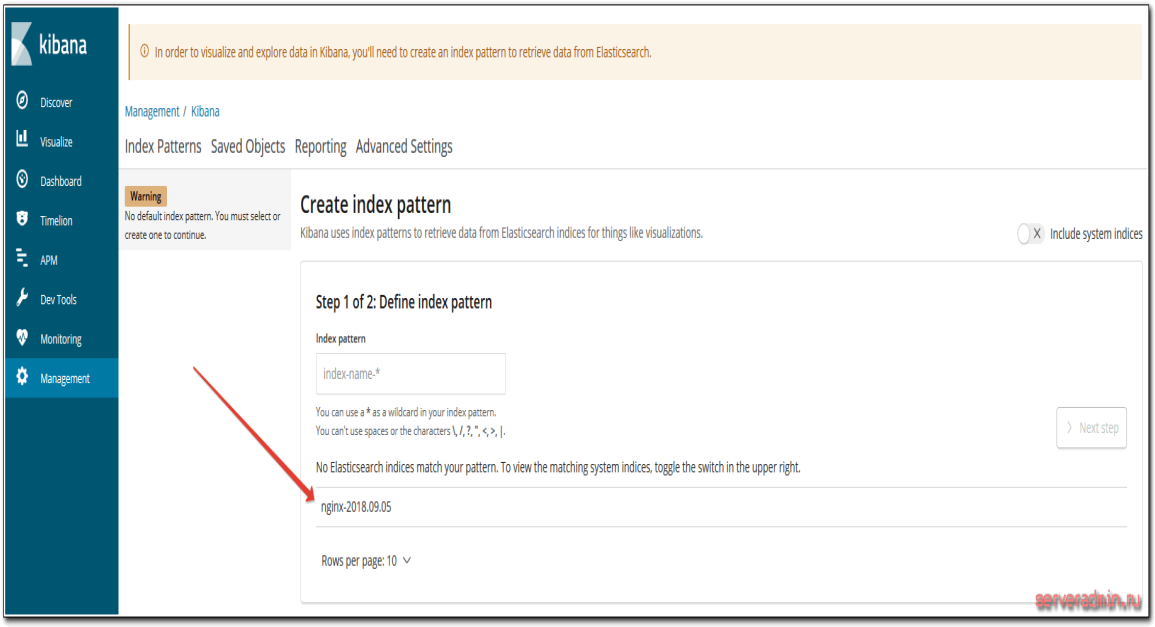


Рис.3.3 – підключення Logstash до Kibana

Ви повинні побачити індекс, який почав заливати logstash в elasticsearch. В поле Index pattern введіть nginx- * і натисніть Next Step. На наступному етапі виберіть ім'я поля для тимчасового фільтра. У вас буде тільки один варіант - @timestamp, вибирайте його і тисніть Create Index Pattern.

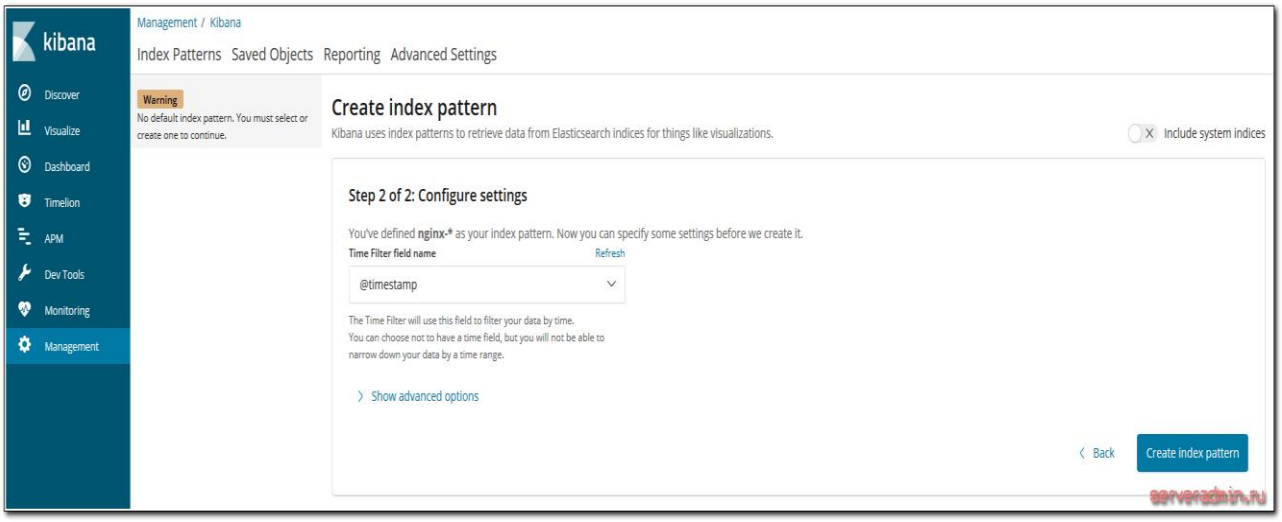


Рис.3.4 – підключення Logstash до Kibana

Новий індекс доданий. Тепер при переході в розділ Discover, він буде відкриватися за замовчуванням з усіма даними, які в нього надходять.

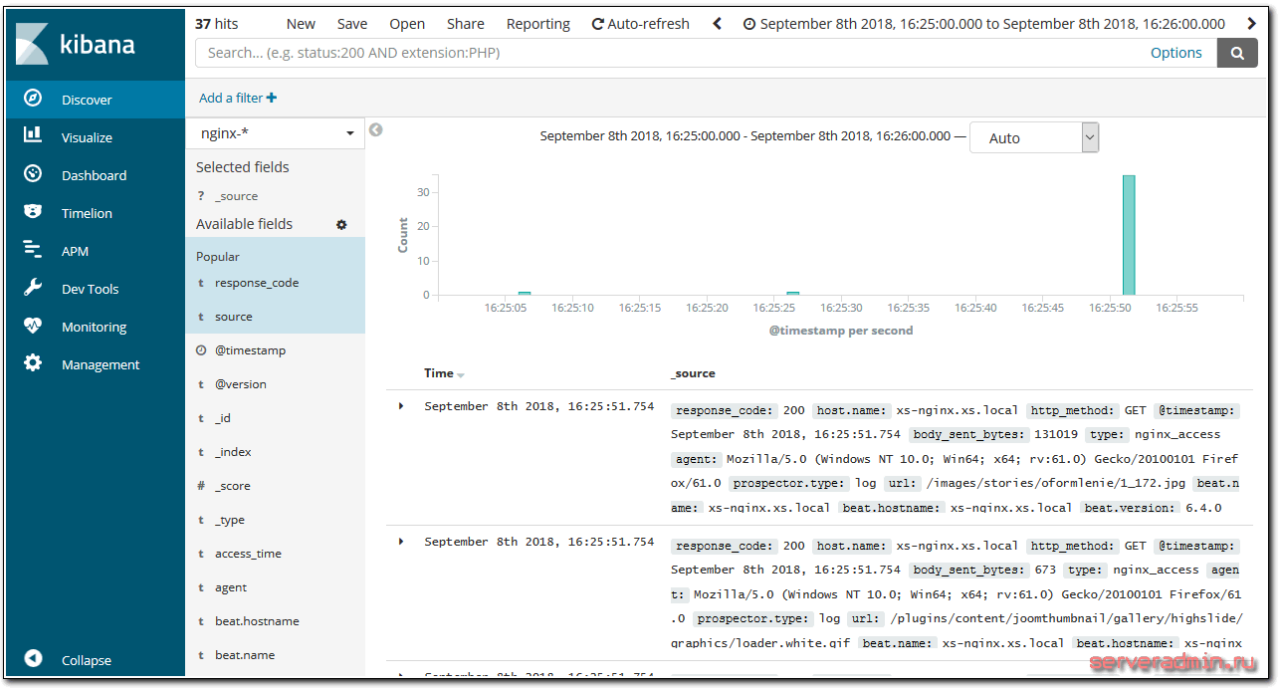


Рис.3.5 – відображення роботи Logstash в Kibana

Отриманням логів з linux серверів налаштували. Тепер зробимо те ж саме для журналів windows. Використанні джерела {https://codeguida.com/post/1415}

3.1.8 Проксінг підключень до Kibana через Nginx

Я не буду детально розповідати про те, що таке проксінг в nginx. У мене є окрема стаття на цю тему - настройка proxy_pass в nginx. Наведу просто приклад конфіга для передачі запитів з nginx в kibana. Я рекомендую використовувати ssl сертифікати для доступу до Kibana. Навіть якщо в цьому немає об'єктивної необхідності, набридають повідомлення браузерів про те, що у вас небезпечне підключення. Детальна інструкція по установці, налаштуванню і використанню ssl в nginx так само є у мене на сайті - настройка web сервера nginx. Всі подробиці налаштування nginx дивіться там.

Ось приблизний конфиг nginx для проксінг запитів до Kibana з обмеженням доступу по паролю:

```
server {
    listen 443;
    server_name kibana.site.ru;
    ssl_certificate /etc/letsencrypt/live/kibana.site.ru/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/kibana.site.ru/privkey.pem;
    location / {
                auth_basic "Restricted Access";
                auth_basic_user_file /etc/nginx/htpasswd.kibana;
                proxy_pass http://localhost:5601;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
}
```

Створюємо файл для користувачів і паролів:

```
# htpasswd -c /etc/nginx/htpasswd.kibana kibanauser
```

Якщо утиліти htpasswd немає в системі, то встановіть її:

```
# yum install -y httpd-tools
```

Після цього вийде запит на введення пароля. За допомогою наведеної вище команди ми створили файл з інформацією про користувача і пароль

kibanauser для обмеження доступу до web панелі Kibana .

3.2 Встановлення і Налаштування автоматизації

3.2.1 Google Cloud Platform (compute instance) и Terraform в Unix/Linux

Google Cloud Platrorm - це платформа виду «інфраструктура як сервіс» (IaaS), що дозволяє клієнтам створювати, тестувати і розгортати власні додатки на інфраструктурі Google, в високопродуктивних віртуальних машинах.

Google Compute Engine надає віртуальні машини, що працюють в інноваційних центрах обробки даних Google і всесвітньої мережі.

Установка terraform в Unix / Linux

Щоб отримати допомогу по використанню команд, виконайте:

sh 36 lines

```
$ terraform --help
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a Terraform working directory
    output             Read an output from a state file
    plan               Generate and show an execution plan
    providers          Prints a tree of the providers used in the configuration
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version
    workspace          Workspace management

All other commands:
    debug              Debug output management (experimental)
    force-unlock       Manually unlock the terraform state
    state              Advanced state management
```

Рис.3.6 – Відображення команд Terraform

Робота з Google Cloud Platform (compute instance) і Terraform в Unix / Linux

Перше що потрібно зробити - це налаштувати «Cloud Identity». За допомогою сервісу Google Cloud Identity ви зможете надавати доменів, користувачам і акаунтів в організації доступ до ресурсів Cloud, а також централізовано керувати користувачами і групами через консоль адміністратора Google.

Наведу приклад налаштування аккаунта, для цього відкриваємо консоль з гуглом і переходимо в «IAM і адміністрування» -> »Сервісні акаунти»:

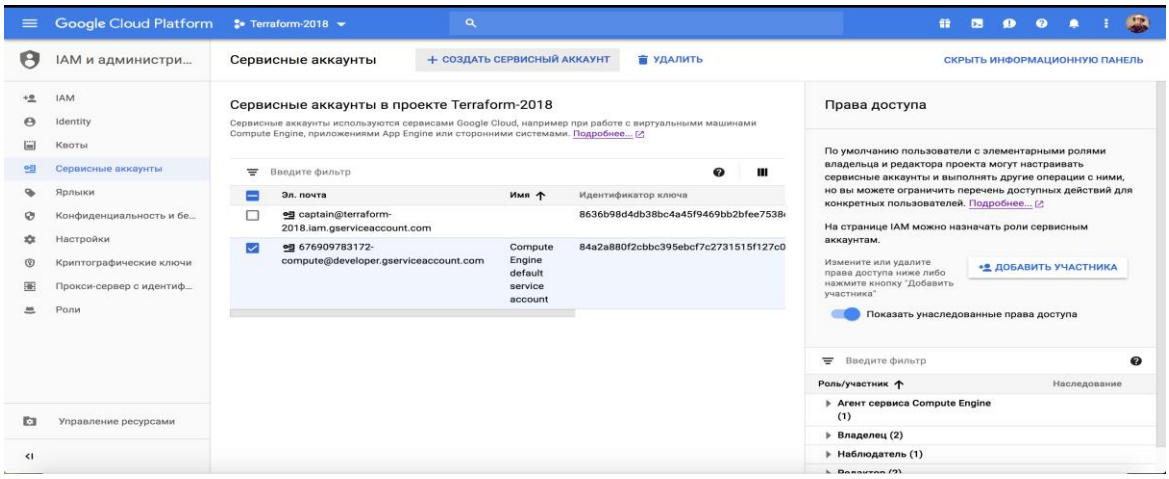


Рис.3.7 – підключення сервісного аккаунта GCP

Натискаємо на «Створити сервісний акаунт»:

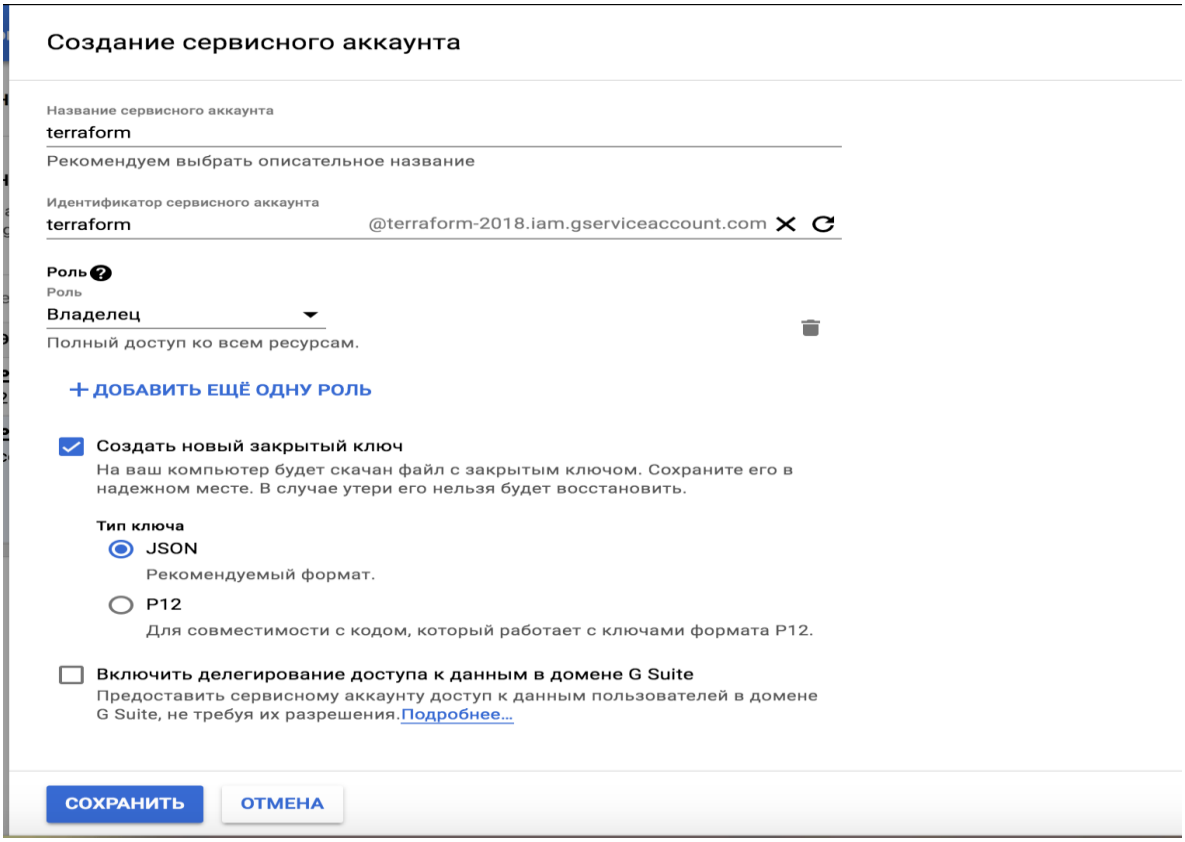


Рис.3.8 – створення сервісного аккаунта GCP

Створити сервісний акаунт в Google Cloud

Прописуємо необхідні дані і натискаємо на «Зберегти». Створити то створили, але потрібно ще і дозволити використання необхідних ресурсів:

PS: Можна все це справа наклацать в консолі, в такий спосіб:

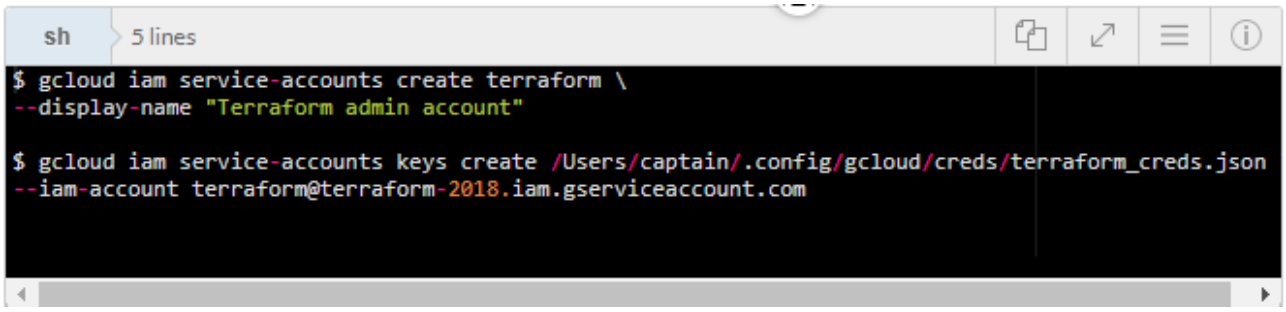


Рис.3.9 - створення сервісногого аккаунта через консоль gcloud

де:

terraform - Назва юзера. Даний користувач буде використовуватися для подальшого використання.

/Users/captain/.config/gcloud/creds/terraform_creds.json - Шлях куди

збережеться файл.

terraform-2018 - назва проекту. Даний проект буде фігурувати і в наступних моїх статтях.

Надайте дозвіл облікового запису служби для перегляду проекту Admin і управління хмарним сховищем:

```sh
$ gcloud projects add-iam-policy-binding terraform-2018 \
--member serviceAccount:terraform@terraform-2018.iam.gserviceaccount.com \
--role roles/owner

$ gcloud projects add-iam-policy-binding terraform-2018 \
--member serviceAccount:terraform@terraform-2018.iam.gserviceaccount.com \
--role roles/storage.admin
```

Рис.3.10 - створення проекту через консоль gcloud

Будь-які дії, які виполняються через Terraform, вимагають, щоб API був включений. У цьому керівництві Terraform вимагає наступне:

```sh
$ gcloud services enable cloudresourcemanager.googleapis.com
$ gcloud services enable cloudbilling.googleapis.com
$ gcloud services enable iam.googleapis.com
$ gcloud services enable compute.googleapis.com
```

Рис.3.11 - підключення політик через консоль gcloud

Я не впевнений що це вся політика яка необхідна мені для подальшої роботи. Але будемо дивитися за обставинами, у мене не багато досвіду з Google Cloud, з цього - вникати доводиться відразу. Іноді, матеріал дуже застарілий, навіть не офіційному сайті - це возмутімо!

У мене є папка terraform, в ній у мене будуть лежати провайдери з якими я буду працювати. Т.к в цьому прикладі я буду використовувати google_cloud_platform, то створю дану папку і перейду в неї. Далі, в цій папці, варто створити:

```sh
$ mkdir examples modules
```

Рис.3.12 – створення папки examples-modules

В папці examples, я буду зберігати так звані «плейбук» для розварачіванія різних служб, наприклад - zabbix-server, grafana, web-сервери і так далі. У modules директорії, я буду зберігати всі необхідні модулі.

Почнемо писати модуль, але для цього завдання, я створю папку:

```
$ mkdir modules/compute_instance
```

Рис.3.13 створення папки computer_instance

Переходим в неї і відкриваємо файл

$ vim compute_instance.tf

І в даний файл копіюємо:

```
#----------------------------------------------------
# Create compute instance
#----------------------------------------------------
resource "google_compute_instance" "compute_instance" {
    #count           = "${var.number_of_instances}"
    count           = "${var.enable_attached_disk ? 0 : var.number_of_instances}"

    project         = "${var.project_name}"
    name            = "${lower(var.name)}-ce-${lower(var.environment)}-${count.index+1}"
    zone            = "${var.zone}"
    machine_type    = "${var.machine_type}"

    allow_stopping_for_update   = "${var.allow_stopping_for_update}"
    can_ip_forward              = "${var.can_ip_forward}"
    timeouts                    = "${var.timeouts}"
    description                 = "${var.description}"
    deletion_protection         = "${var.deletion_protection}"
    min_cpu_platform            = "${var.min_cpu_platform}"

    #scratch_disk {
    #    #interface   = "${var.scratch_disk_interface}"
    #}

    boot_disk {
        auto_delete             = "${var.boot_disk_auto_delete}"
        device_name             = "${var.boot_disk_device_name}"
        disk_encryption_key_raw = "${var.disk_encryption_key_raw}"
        initialize_params {
            size    = "${var.boot_disk_initialize_params_size}"
            type    = "${var.boot_disk_initialize_params_type}"
            image   = "${var.boot_disk_initialize_params_image}"
        }
    }
```

Рис.3.14 конфігурація instance.tr

```
#attached_disk {
#    source                 = "testua666"
#    device_name            = "testua666"
#    mode                   = "READ_WRITE"
#    disk_encryption_key_raw = "${var.disk_encryption_key_raw}"
#}

network_interface {
    network            = "${var.network}"
    subnetwork         = "${var.subnetwork}"
    subnetwork_project = "${var.subnetwork_project}"
    address            = "${var.address}"

    #alias_ip_range {
    #    ip_cidr_range            = "10.138.0.0/20"
    #    subnetwork_range_name    = ""
    #}

    access_config {
        nat_ip                 = "${var.nat_ip}"
        public_ptr_domain_name = "${var.public_ptr_domain_name}"
        network_tier           = "${var.network_tier}"
    }
}

labels {
    name            = "${lower(var.name)}-ce-${lower(var.environment)}-${count.index+1}"
    environment     = "${lower(var.environment)}"
    orchestration   = "${lower(var.orchestration)}"
}

metadata {
    ssh-keys = "${var.ssh_user}:${file("${var.public_key_path}")}"
    #shutdown-script = "${file("${path.module}/scripts/shutdown.sh")}"
}
```

Рис.3.15 конфігурація instance.tr

```
metadata_startup_script = "${file("${path.module}/${var.install_script_src_path}")}"
#
#metadata_startup_script = "echo hi > /test.txt"
#metadata_startup_script = "${file("startup.sh")}"
#metadata_startup_script = <<SCRIPT
#    ${file("${path.module}/scripts/install.sh")}
#SCRIPT

#provisioner "file" {
#    source      = "${var.install_script_src_path}"
#    destination = "${var.install_script_dest_path}"
#
#    connection {
#        type             = "ssh"
#        user             = "${var.ssh_user}"
#        port             = "${var.ssh_port}"
#        private_key      = "${file("${var.private_key_path}")}"
#        agent            = false
#        timeout          = "5m"
#        agent_identity   = false
#        insecure         = true
#    }
#}

#provisioner "remote-exec" {
#    connection {
#        type         = "ssh"
#        user         = "${var.ssh_user}"
#        port         = "${var.ssh_port}"
#        private_key = "${file("${var.private_key_path}")}"
#        agent        = false
#        timeout      = "5m"
#    }
#
#    inline = [
#        "chmod +x ${var.install_script_dest_path}",
#        "sudo ${var.install_script_dest_path} ${count.index}",
#    ]
#}
```

Рис.3.16 конфігурація instance.tr

```
    tags = [
        "${lower(var.name)}",
        "${lower(var.environment)}",
        "${lower(var.orchestration)}"
    ]

    service_account {
        email  = "${var.service_account_email}"
        scopes = "${var.service_account_scopes}"
    }

    scheduling {
        preemptible         = "${var.scheduling_preemptible}"
        on_host_maintenance = "${var.scheduling_on_host_maintenance}"
        automatic_restart   = "${var.scheduling_automatic_restart}"
    }

    #Note: GPU accelerators can only be used with on_host_maintenance option set to TERMINATE.
    guest_accelerator {
        type  = "${var.guest_accelerator_type}"
        count = "${var.guest_accelerator_count}"
    }

    lifecycle {
        ignore_changes = [
            "network_interface",
        ]
        create_before_destroy = true
    }

}
#-----------------------------------------------
# Create compute instance with attached disk
#-----------------------------------------------
resource "google_compute_instance" "compute_instance_with_attached_disk" {
    count            = "${var.enable_attached_disk && length(var.attached_disk_source) > 0 ? var.num

    project          = "${var.project_name}"
    name             = "${lower(var.name)}-ce-${lower(var.environment)}-${count.index+1}"
    zone             = "${var.zone}"
    machine_type     = "${var.machine_type}"
```

Рис.3.17 конфігурація instance.tr

```
    allow_stopping_for_update    = "${var.allow_stopping_for_update}"
    can_ip_forward               = "${var.can_ip_forward}"
    timeouts                     = "${var.timeouts}"
    description                  = "${var.description}"
    deletion_protection          = "${var.deletion_protection}"
    min_cpu_platform             = "${var.min_cpu_platform}"

    #scratch_disk {
    #    #interface   = "${var.scratch_disk_interface}"
    #}

    boot_disk {
        auto_delete              = "${var.boot_disk_auto_delete}"
        device_name              = "${var.boot_disk_device_name}"
        disk_encryption_key_raw  = "${var.disk_encryption_key_raw}"
        initialize_params {
            size     = "${var.boot_disk_initialize_params_size}"
            type     = "${var.boot_disk_initialize_params_type}"
            image    = "${var.boot_disk_initialize_params_image}"
        }
    }

    attached_disk {
        source                   = "${var.attached_disk_source}"
        device_name              = "${var.attached_disk_device_name}"
        mode                     = "${var.attached_disk_mode}"
        disk_encryption_key_raw  = "${var.disk_encryption_key_raw}"
    }

    network_interface {
        network            = "${var.network}"
        subnetwork         = "${var.subnetwork}"
        subnetwork_project = "${var.subnetwork_project}"
        address            = "${var.address}"
```

Рис.3.18 конфігурація instance.tr

```
        #alias_ip_range {
        #    ip_cidr_range               = "10.138.0.0/20"
        #    subnetwork_range_name       = ""
        #}

        access_config {
            nat_ip                  = "${var.nat_ip}"
            public_ptr_domain_name  = "${var.public_ptr_domain_name}"
            network_tier            = "${var.network_tier}"
        }
    }

    labels {
        name            = "${lower(var.name)}-ce-${lower(var.environment)}-${count.index+1}"
        environment     = "${lower(var.environment)}"
        orchestration   = "${lower(var.orchestration)}"
    }

    metadata {
        ssh-keys = "${var.ssh_user}:${file("${var.public_key_path}")}"
        #shutdown-script = "${file("${path.module}/scripts/shutdown.sh")}"
    }

    metadata_startup_script = "${file("${path.module}/${var.install_script_src_path}")}"
    #
    #metadata_startup_script = "echo hi > /test.txt"
    #metadata_startup_script = "${file("startup.sh")}"
    #metadata_startup_script = <<SCRIPT
    #    ${file("${path.module}/scripts/install.sh")}
    #SCRIPT

    #provisioner "file" {
    #    source      = "${var.install_script_src_path}"
    #    destination = "${var.install_script_dest_path}"
    #
    #    connection {
    #        type            = "ssh"
    #        user            = "${var.ssh_user}"
    #        port            = "${var.ssh_port}"
    #        private_key     = "${file("${var.private_key_path}")}"
    #        agent           = false
```

Рис.3.19 конфігурація instance.tr

```
    #        timeout         = "5m"
    #        agent_identity  = false
    #        insecure        = true
    #    }
    #}

    #provisioner "remote-exec" {
    #    connection {
    #        type        = "ssh"
    #        user        = "${var.ssh_user}"
    #        port        = "${var.ssh_port}"
    #        private_key = "${file("${var.private_key_path}")}"
    #        agent       = false
    #        timeout     = "5m"
    #    }
    #
    #    inline = [
    #        "chmod +x ${var.install_script_dest_path}",
    #        "sudo ${var.install_script_dest_path} ${count.index}",
    #    ]
    #}
    #
    #

    tags = [
        "${lower(var.name)}",
        "${lower(var.environment)}",
        "${lower(var.orchestration)}"
    ]

    service_account {
        email    = "${var.service_account_email}"
        scopes   = "${var.service_account_scopes}"
    }

    scheduling {
```

Рис.3.20 конфігурація instance.tr

```
    scheduling {
        preemptible         = "${var.scheduling_preemptible}"
        on_host_maintenance = "${var.scheduling_on_host_maintenance}"
        automatic_restart   = "${var.scheduling_automatic_restart}"
    }

    #Note: GPU accelerators can only be used with on_host_maintenance option set to TERMINATE.
    guest_accelerator {
        type  = "${var.guest_accelerator_type}"
        count = "${var.guest_accelerator_count}"
    }

    lifecycle {
        ignore_changes = [
            "network_interface",
        ]
        create_before_destroy = true
    }

}
```

Рис.3.21 конфігурація instance.tr

Відкриваємо файл:

$ vim variables.tf

І прописуємо:

```
variable "name" {
    description = "A unique name for the resource, required by GCE. Changing this forces a new reso
    default     = "TEST"
}

variable "zone" {
  description = "The zone that the machine should be created in"
  default     = "us-east1-b"
}

variable "environment" {
    description = "Environment for service"
    default     = "STAGE"
}

variable "orchestration" {
    description = "Type of orchestration"
    default     = "Terraform"
}

variable "createdby" {
    description = "Created by"
    default     = "Vitaliy Natarov"
}

variable "project_name" {
    description = "The ID of the project in which the resource belongs. If it is not provided, the
    default     = ""
}

variable "machine_type" {
    description = "The machine type to create"
    default     = "f1-micro"
}

variable "allow_stopping_for_update" {
    description = "If true, allows Terraform to stop the instance to update its properties. If you
    default     = true
}
```

Рис.3.22 конфігурація змінних проекту Terraform

```
variable "can_ip_forward" {
    description = "Whether to allow sending and receiving of packets with non-matching source or de
    default     = false
}

variable "timeouts" {
    description = "Configurable timeout in minutes for creating instances. Default is 4 minutes. Ch
    default     = 4
}

variable "description" {
    description = "A brief description of this resource."
    default     = ""
}

variable "deletion_protection" {
    description = "Enable deletion protection on this instance. Defaults to false. Note: you must d
    default     = false
}

variable "min_cpu_platform" {
    description = "Specifies a minimum CPU platform for the VM instance. Applicable values are the
    default     = "Intel Haswell"
}

variable "boot_disk_auto_delete" {
    description = "Whether the disk will be auto-deleted when the instance is deleted. Defaults to
    default     = true
}

variable "boot_disk_device_name" {
    description = "Name with which attached disk will be accessible under /dev/disk/by-id/"
    default     = ""
}
```

Рис.3.23 конфігурація змінних проекту Terraform

```
variable "disk_encryption_key_raw" {
    description = "A 256-bit customer-supplied encryption key, encoded in RFC 4648 base64 to encryp
    default     = ""
}

variable "boot_disk_initialize_params_size" {
    description = "The size of the image in gigabytes. If not specified, it will inherit the size o
    default     = "10"
}

variable "boot_disk_initialize_params_type" {
    description = "The GCE disk type. May be set to pd-standard or pd-ssd."
    default     = "pd-standard"
}

variable "boot_disk_initialize_params_image" {
    description = "The image from which to initialize this disk. This can be one of: the image's se
    default     = "centos-7"
}

variable "number_of_instances" {
    description = "Number of instances to make"
    default     = "1"
}

#variable "scratch_disk_interface" {
#    description = "The disk interface to use for attaching this disk; either SCSI or NVME. Default
#    default     = "SCSI"
#}

variable "network" {
    description = "The name or self_link of the network to attach this interface to. Either network
    default     = "default"
}
```

Рис.3.24 конфігурація змінних проекту Terraform

```
variable "subnetwork" {
    description = "The name or self_link of the subnetwork to attach this interface to. The subnetw
    default     = ""
}

variable "subnetwork_project" {
    description = "The project in which the subnetwork belongs. If the subnetwork is a self_link, t
    default     = ""
}

variable "address" {
    description = "The private IP address to assign to the instance. If empty, the address will be
    default     = ""
}

variable "nat_ip" {
    description = "The IP address that will be 1:1 mapped to the instance's network ip. If not give
    default     = ""
}

variable "public_ptr_domain_name" {
    description = "The DNS domain name for the public PTR record. To set this field on an instance,
    default     = ""
}

variable "network_tier" {
    description = "The networking tier used for configuring this instance. This field can take the
    default     = "PREMIUM"
}

variable "service_account_email" {
    description = "The service account e-mail address. If not given, the default Google Compute Eng
    default     = ""
}

variable "service_account_scopes" {
    description = "A list of service scopes. Both OAuth2 URLs and gcloud short names are supported.
    default     = []
}
```

Рис.3.25 конфігурація змінних проекту Terraform

```
variable "scheduling_on_host_maintenance" {
    description = "Describes maintenance behavior for the instance. Can be MIGRATE or TERMINATE"
    default     = "TERMINATE"
}

variable "scheduling_automatic_restart" {
    description = "Specifies if the instance should be restarted if it was terminated by Compute En
    default     = "true"
}

variable "guest_accelerator_type" {
    description = "The accelerator type resource to expose to this instance. E.g. nvidia-tesla-k80.
    default     = ""
}

variable "guest_accelerator_count" {
    description = "The number of the guest accelerator cards exposed to this instance."
    default     = "0"
}

variable "ssh_user" {
    description = "User for connection to google machine"
    default     = "captain"
}

variable "ssh_port" {
    description = "Port for connection to google machine"
    default     = "22"
}

variable "public_key_path" {
    description = "Path to file containing public key"
    default     = "~/.ssh/gcloud_id_rsa.pub"
}

variable "private_key_path" {
    description = "Path to file containing private key"
    default     = "~/.ssh/gcloud_id_rsa"
}
```

Рис.3.26 конфігурація змінних проекту Terraform

```
variable "install_script_src_path" {
    description = "Path to install script within this repository"
    default     = "scripts/install.sh"
}

variable "install_script_dest_path" {
    description = "Path to put the install script on each destination resource"
    default     = "/tmp/install.sh"
}

variable "enable_attached_disk" {
    description = "Enable attaching disk to node"
    default     = "false"
}

variable "attached_disk_source" {
    description = "The name or self_link of the disk to attach to this instance."
    default     = ""
}

variable "attached_disk_device_name" {
    description = "Name with which the attached disk will be accessible under /dev/disk/by-id/"
    default     = ""
}

variable "attached_disk_mode" {
    description = "Either 'READ_ONLY' or 'READ_WRITE', defaults to 'READ_WRITE' If you have a persi
    default     = "READ_WRITE"
}
```

Рис.3.27 конфігурація змінних проекту Terraform

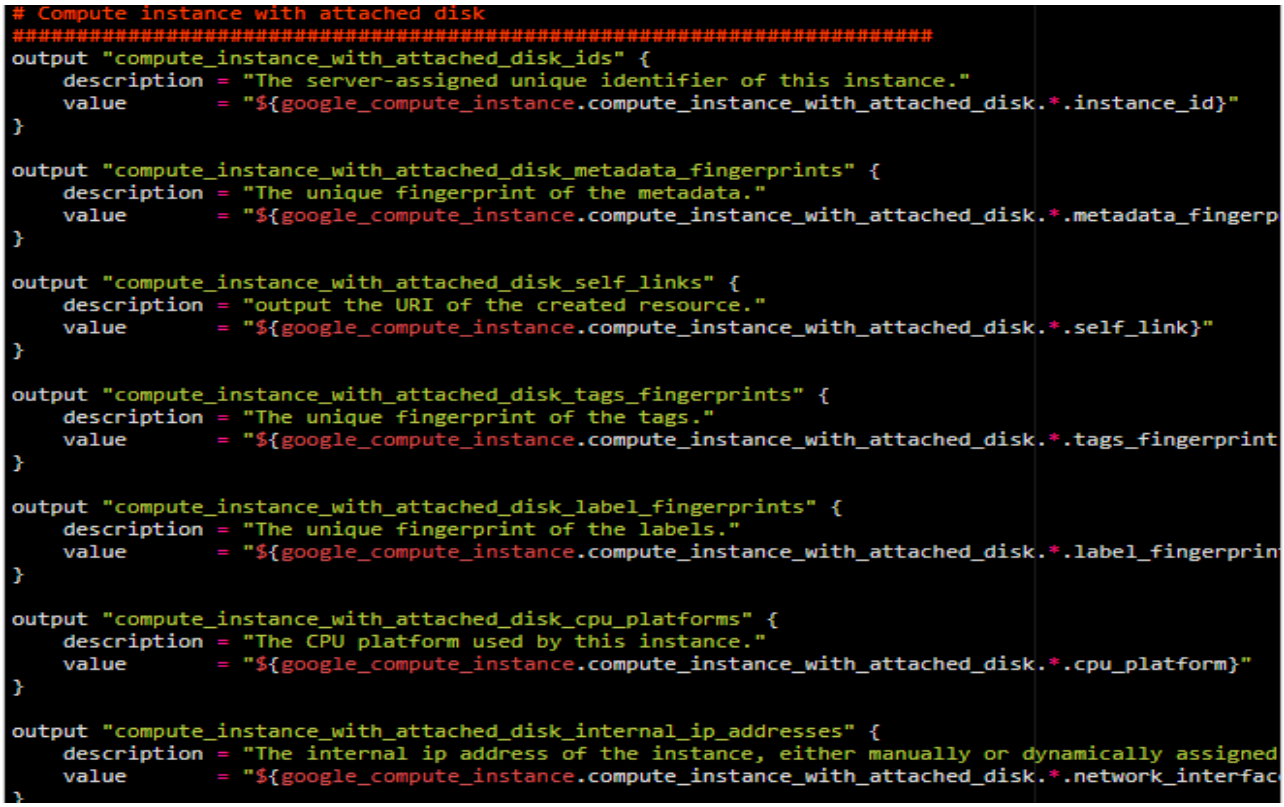
Відкриваємо останній файл:

$ vim outputs.tf

І вставляємо туди код:

```
# Compute instance
#####################################################################
output "compute_instance_ids" {
    description = "The server-assigned unique identifier of this instance."
    value       = "${google_compute_instance.compute_instance.*.instance_id}"
}

output "compute_instance_metadata_fingerprints" {
    description = "The unique fingerprint of the metadata."
    value       = "${google_compute_instance.compute_instance.*.metadata_fingerprint}"
}

output "compute_instance_self_links" {
    description = "output the URI of the created resource."
    value       = "${google_compute_instance.compute_instance.*.self_link}"
}

output "compute_instance_tags_fingerprints" {
    description = "The unique fingerprint of the tags."
    value       = "${google_compute_instance.compute_instance.*.tags_fingerprint}"
}

output "compute_instance_label_fingerprints" {
    description = "The unique fingerprint of the labels."
    value       = "${google_compute_instance.compute_instance.*.label_fingerprint}"
}

output "compute_instance_cpu_platforms" {
    description = "The CPU platform used by this instance."
    value       = "${google_compute_instance.compute_instance.*.cpu_platform}"
}

output "compute_instance_internal_ip_addresses" {
    description = "The internal ip address of the instance, either manually or dynamically assigned
    value       = "${google_compute_instance.compute_instance.*.network_interface.0.address}"
}
```

Рис.3.28 конфігурація зовнішніх змінних проекту Terraform



```
# Compute instance with attached disk
################################################################################
output "compute_instance_with_attached_disk_ids" {
    description = "The server-assigned unique identifier of this instance."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.instance_id}"
}

output "compute_instance_with_attached_disk_metadata_fingerprints" {
    description = "The unique fingerprint of the metadata."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.metadata_fingerp
}

output "compute_instance_with_attached_disk_self_links" {
    description = "output the URI of the created resource."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.self_link}"
}

output "compute_instance_with_attached_disk_tags_fingerprints" {
    description = "The unique fingerprint of the tags."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.tags_fingerprint
}

output "compute_instance_with_attached_disk_label_fingerprints" {
    description = "The unique fingerprint of the labels."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.label_fingerprin
}

output "compute_instance_with_attached_disk_cpu_platforms" {
    description = "The CPU platform used by this instance."
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.cpu_platform}"
}

output "compute_instance_with_attached_disk_internal_ip_addresses" {
    description = "The internal ip address of the instance, either manually or dynamically assigned
    value       = "${google_compute_instance.compute_instance_with_attached_disk.*.network_interfac
}
```

Рис.3.29 конфігурація зовнішніх змінних проекту Terraform

В каталозі створюємо папук “scripts”:

$ mkdir scripts

Створюємо там Bash скрипт:

$ vim scripts/install.sh

sudo yum update -y

sudo yum upgrade -y

sudo yum install epel-release -y

sudo yum install nginx -y

sudo service nginx restart

Переходимо тепер в папку *google_cloud_platform / examples* і створюємо ще одну папку для перевірки написаного дива:

$ mkdir compute && cd $_

Відкриваємо файл:

$ vim main.tf

І вставляємо в нього наступний код:

Рис.3.30 конфігурація проекту Terraform

Тепер все написано і можна тестувати.

Ініціалізуємо проект:

$ terraform init

Встановлення модулів:

$ terraform get

І для обновлення всіх модулів :

$ terraform get -update

Перевіряємо валідацію:

$ terraform validate

Потім дивимся які будуть зміні приміненні командою:

$ terraform plan

і далі примінення цього плану:

$ terraform apply

Можемо зайти на Gcloud і перевірити як примінилися змінні і встановлення віртуальних машин. Далі Буде використовуватися Ansible для заповнення цих машин.

3.2.2 Налаштування Ansible в Unix/Linux

Установка і налаштування Ansible в Unix/Linux

Ansible - безкоштовне ПО для конфігурації і управління комп'ютерами, поєднує в собі розгортання багатовузлового програмного забезпечення, виконання різних завдань і управління конфігураціями. Він керує нодамі (які повинні мати підтримку Python 2.4 або більш пізньої версії) через SSH. Модулі працюють над JSON і стандартний висновок може бути написаний будь-якою мовою програмування. Система використовує YAML для написання функцій (завдань виконання).

На Debian/Ubuntu

```
# apt-key update && apt-get update && apt-get -y upgrade && apt-get -y install python-software-properties && apt-get -y install software-properties-common && apt-add-repository -y ppa:rquillo/ansible && apt-get update && apt-get -y install ansible
```

Встановлення Ansible через Git:

 # cd /usr/local/src && git clone git://github.com/ansible/ansible.git

Провіряємо чи встановились модулі:

cd ansible && git submodule update --init --recursive

Провіряємо версію Ansible:

```
$ ansible --version
ansible 2.1.1.0
 config file = /etc/ansible/ansible.cfg
 configured module search path = Default w/o overrides
```

Рис.3.31 – Перевірка версії Ansible

Як працює Ansible?

Головна мета Ansible - це з одного (можна зробити і пару) серверів з ансіблом можна було керувати всіма іншими нодамі. З такого сервера можна відправляти різні команди (набір деяких інструкцій, так званих playbooks) на будь-яку з підключених нод на сервері. Виконання команд виконується шляхом підключення сервера до ноді через публічний ключ по SSH. Внизу представлена наочна схема того як все на ньому влаштовано:

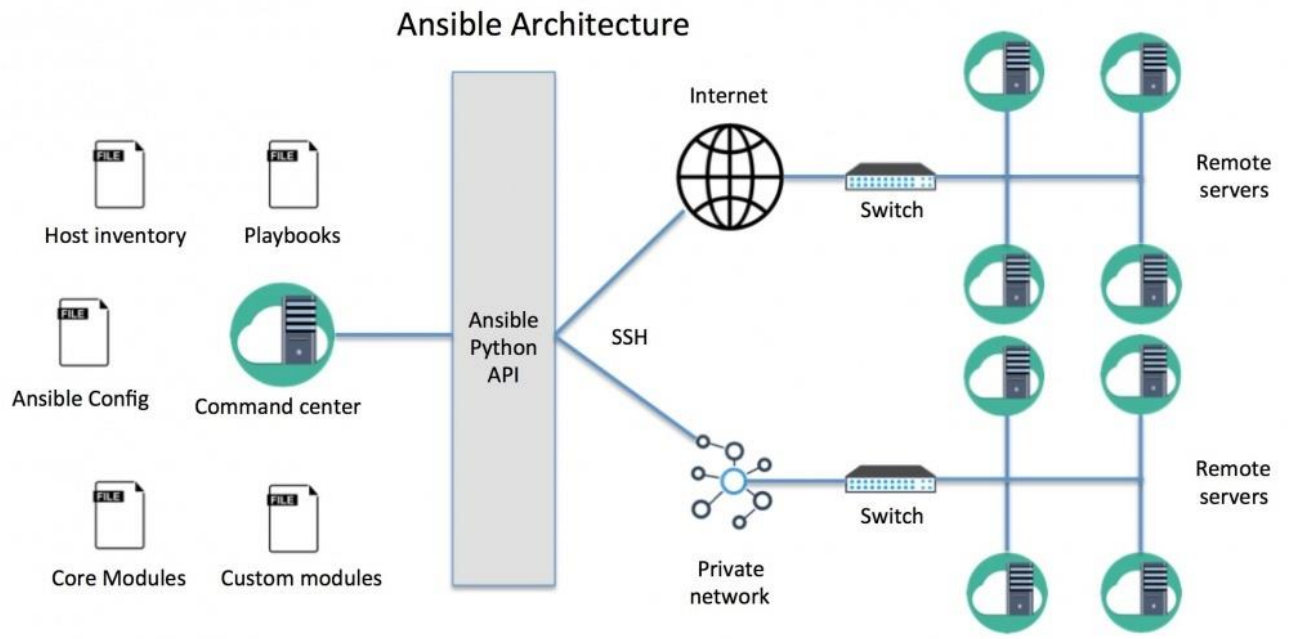


Рис.3.32 – принцип роботи Ansible

У файлі «Host inventory» міститься інформація про обслуговуваних нодах (де власне все команди будуть виконані). Конфігураційний файл з Ansible може бути використаний для настройки змінного оточення. Інформацію за основу взято з джерела {https://linux-notes.org/ustanovka-i-nastrojka-ansible-v-unix-linux/}

У ansible, використовуються playbooks - це набір інструкцій і вони складаються з завдань. Плейбук описуються за допомогою функціональності модуля ядра самого Ansible (так само, можуть використовувати сторонні модулі). Playbooks - послідовність або набір команд в яких так само, може використовуватися різні перевірки: якщо умова не виконується, то певні команди можуть пропускатися. Дані кукбукі, описуються в форматі YAML.

З Ansible API - ви можете запускати скрипти. Якщо скрипту-обгортці (wrapper) може знадобиться запуск playbook, то це можна зробити через API. Налаштування Ansible в Unix / Linux

Сам конфігураційний файл описується в форматі INI. Так само, є можливість перевизначення частини конфігурації (можна навіть всього конфіга) в параметрах playbook або в змінних оточення.

При виконанні команд, Ansible виконує перевірку і шукає наявності файлу

з конфігураціями в наступних розташуваннях:

Параметр ANSIBLE_CONFIG перевіряється змінна оточення, який може бути вказувати на файл конфігурації.

- ./ansible.cfg - в цій папці.
- ~ / .Ansible.cfg - в домашній директорії користувача.
- /etc/ansible/ansible.cfg - в папці, смого ansible.
- Налаштування через змінні оточення

Багато опцій, можна задати за допомогою змінного оточення, для цього варто використовувати префікс ANSIBLE_ перед назвою самого параметра з конфігурацією (великими літерами). наприклад:

export ANSIBLE_SUDO_USER=root

після чого *ANSIBLE_SUDO_USER* зможе користуватися змінними *Playbook.*

Налаштування в ansible.cfg

Самих параметрів в Ansible конфігурації багато, але основні я перерахую:

- hostfile: Дана опція вказує на шлях до inventory file. У ньому зберігається список ваших хостів, до яких Ansible зможе виконати підключення.

Наприклад: hostfile = / etc / ansible / hosts

- library: Шлях до папки. У ній збережені всі модулі Ansible.

Наприклад: library = / usr / share / ansible

- forks: Здається кількість процесів, які може породити Ansible. За замовчуванням параметр встановлений в 5.

Наприклад: forks = 5

- sudo_user: Користувач який буде використовуватися за замовчуванням і від якого Ansible власне виконуватиме запуск команд на віддалених нодах.

Наприклад: sudo_user = root

- remote_port: Порт для з'єднання з SSH до Нодаме (за замовчуванням використовує стандартний 22-й порт).

Наприклад: remote_port = 22

- host_key_checking: Дана опція дає можливість вимкнути перевірку SSH-ключа на хості, але за замовчуванням дана перевірка включена.

Наприклад: host_key_checking = False

- timeout: Дана опція задає таймаут (час спроби підключення по SSH).

Наприклад: timeout = 60

- log_path: Шлях для збереження лог-файлів. За замовчуванням Ansible не збереглося їх взагалі.

Наприклад: log_path = /var/log/ansible.log

Пишемо перший файл конфігурації Ansible

Зараз я створю перший конфігураційний файл для Ansible. Для цього потрібно виконати підключення по SSH до сервера де встановлений Ansible. Я створю папку для всіх моїх проектів і після чого, перейду в неї:

 # mkdir ~/ansible

 # cd ~/ansible

Також створюємо директорію для збереження всіх модулів для Ansible і так само, директорію для збереження всіх логів (куди ж без них):

 # mkdir ~/ansible/modules

 # mkdir ~/ansible/logs

Підійшов час до створення файл ansible.cfg:

$ vim ~/ansible/ansible.cfg

В нього записуєм:

[defaults]

hostfile = ~/ansible/inventory

sudo_user = root

log_path = ~/ansible/logs/ansible.log

У новій версії ansible поля «hostfile» немає, за місце його, є інше:

inventory = /etc/ansible/hosts

Вказуємо обслуговуються сервера в host inventory

Як я описував раніше, даний файл, служить для внесення IP адрес всіх наявних нод (ті, які будуть обслуговуватися) і так само, згрупуйте їх. Для цього створюємо файл inventory:

$ vim ~/ansible/inventory

І добавляємо IP всіх хостів:

[name_of_group]

alias ansible_ssh_node=IP_of_SERVER

some_your_host ansible_ssh_host=SERVER_IP

в мене це :

[experiments]

192.168.1.216

alias ansible_ssh_host=192.168.1.216

Генерація ключа для ansible

Як я говорив раніше, ansible використовує підключення SSH по ключу (для доступу до налаштованим серверів), з цього, потрібно згенерувати його:

 # ssh-keygen -t rsa -b 4096 -C admin@linux-notes.org

Заблокую доступ до SSH каталогу, що тільки ви смоглі читати або писати в цю папку:

 # chmod 700 ~/.ssh

Потрібно змінити дозволу, щоб зберегти цей файл у безпеки:

 # chmod 600 ~/.ssh/id_rsa && chmod 644 ~/.ssh/id_rsa.pub

Створюємо простий ключ, все пропускаємо (натискаємо enter на всі питання). Після того як ключ згенерували, потрібно його передати на все наявний Ноди, для цього є команда:

$ ssh-copy-id root@ip_адрес_настраиваемого_сервера

Виставляємо правильні права:

 # chmod 644 ~/.ssh/id_rsa.pub

Щоб перевірити чи працює все добре, можна підключиться на сервер по SSH і якщо не запитає пароль, то все працює добре (можна і виконати ping на

віддалений хост з сервера Ansible).

$ ssh captain@192.168.1.216

Опція «-m ping» - частина команди для анзібль яка дає можливість використовувати модуль «Ping». Ви можете перевірити роботу даної команди з опцій «-a» і перевірити як працює дана опція.

Модуль «shell» дозволяє нам послати команду на віддалений хост і отримати результати. Наприклад, щоб дізнатися використання пам'яті на нашій машині some_your_host_1, ми могли б використовувати:
$ ansible -m shell -a 'free -m' some_your_host_1

Або як в мене :
$ ansible -m command -a "df -h" 192.168.1.216

Управління ansible конфігураціями

Працюємо з playbook

На playbook-ах заснований власне весь ansible, так як вони служать (містять) для виконання різних завдань. Кожна написана завдання всередині самого Ansible використовує шматок коду-модуля. Плейбук, використовують формат YAML, але власне, будь-які інші ваші модулі можуть бути написані на будь-якій мові програмування.

Хочу зауважити, що дуже важливо, щоб формат повідомлень від модулів був в JSON. YAML

Кожен плейбук пишеться на мові YAML і для ансібл, майже кожен YAML файл починається з деякого списку, а цей елемент даного списку - список пар «ключ-значення» (іноді звуть словником).

Дані файли повинні починатися з «-» (такий формат властивий YAML) і означає початок вашого документа.

Так само, всі ваші списки повинні знаходиться з однаковим відступом від початку кожного рядка, і повинні починатися з пробілу або «-«. Щоб закомментировать, використовуйте «#».

наприклад:

#Message

- Administration

– Install

Словник представлений у вигляді «ключ:» (двокрапка і пробіл) «значення»:

#Message

site: linux-notes

blog: blog.linux-notes

При необхідності словники можуть бути представлені в скороченій формі:

#Comment

{site: linux-notes, blog: blog.linux-notes}

Можна вказати логічні значення (істина / неправда) так:

---

need_access: no

use_service: yes

file_conf: TRUE

read_value: True

kill_process: false

Цілком наш приклад YAML-файлу буде виглядати так:

#About blog

site: linux-notes

blog: blog

must_read: True

themes:

- hosting

- cloud

- it

- geeks

brands:

- infobox

- infoboxcloud

Для змінних Ansible використовує «{{var}}». Якщо значення після двокрапки починається з «{«, то YAML буде думати, що це Слован.

Для використання змінних потрібно укласти дужки в лапки: *word: "{{ variable }}"*

Цього достатньо для початку написання playbooks.

Пишемо наш перший playbook

Playbooks може складатися зі списку обслуговуються серверів, змінних користувача, завдань, обробників (хендлеров) і т.д. Більшість налаштувань конфігурації можна перевизначити в playbook. Кожен playbook складається з одного або більше дії (гри) в списку.

Мета гри - зв'язати групу хостів з зумовленими ролями, представленими як виклик завдань Ansible.

Як приклад давайте розглянемо процес установки nginx.

Створимо директорію, де будуть зберігатися playbooks:

$ mkdir ~/ansible/playbooks

$ vim ~/ansible/playbooks/setup_nginx.yml

```
---
- hosts: experiments
tasks:

- name: Install nginx package
apt: name=nginx update_cache=yes
sudo: yes

- name: Starting nginx service
service: name=nginx state=started
sudo: yes
```

Рис.3.33 - playbook

Давайте розглянемо вміст:

- hosts: Список хостів або група, на якій ви запускаєте завдання.

Це поле обов'язкове і кожен playbook повинен мати його, за винятком ролей. Якщо вказана хост-група, спочатку Ansible її шукає в playbook, а

потім у файлі inventory. Дізнатися, на яких хостах відбуватиметься робота, можна командою: ansible-playbook -list-host, де - шлях до вашого playbook (playbooks / setup_nginx.yml).

- tasks: Завдання. Все playbooks містять завдання. Завдання - це список дій, які ви хочете виконати. Поле завдання містить ім'я завдання (довідкова інформація про завдання для користувача playbook), модуль, який повинен бути виконаний і аргументи, необхідні для модуля. Параметр «name» опціональний, але рекомендований.

команди

playbook для ansible

Виконання плейбук з назвою *test_job.yml* для певного хоста або групи хостів (прописано в файлі):

 # ansible-playbook --check -i my_custom_hosts_file test_job.yml

Опція «-i» говорить Ansible використовувати довільні хости з файлу. Іноді ви можете захотіти виконати даний playbook (або частина його), тільки на певних хостах. Замість того, щоб робити різні хости в різні файли, можна спробувати ці методи:

Використовуйте опцію «-limit» щоб вказати на якому хості запустити вказаний плейбук (наприклад на host1):

 # ansible-playbook -i all_hosts.inventory some_playbook.yml --limit "your_host_1"

Використовуйте опцію «-limit» щоб вказати кілька хостів (розделенние запитом), наприклад для host1, host2, host3:
 # ansible-playbook -i all_hosts.inventory some_playbook.yml --limit "your_host_1,your_host_2,your_host_3"

Якщо ви не впевнені в тому, що опція «-limit» буде працювати, ви можете використовувати опцію «-list-hosts» і ви отримаєте список хостів, на яких буде виконуватися playbook.

Можна ставити так звані мітки для деяких завдань і потім виконувати їх за цими заданим мітках. Можна зробити задачу щоб вона виконувалася в будь-яких умовах, незалежно від міток. Це може бути зроблено шляхом додавання:

always_run: yes

Приклад:

```
tasks:

- name: Copy MY_FILE_HERE_1
copy: src=/root/work/MY_FILE_HERE_1 dest=/root/work/MY_FILE_HERE_1
tags:
- MY_FILE_HERE_1

- name: Copy MY_FILE_HERE_2
copy: src=/root/work/MY_FILE_HERE_2 dest=/root/work/MY_FILE_HERE_2
tags:
- MY_FILE_HERE_2

- name: Copy MY_FILE_HERE_2
copy: src=/root/work/MY_FILE_HERE_3 dest=/root/work/MY_FILE_HERE_3
tags:
- MY_FILE_HERE_3
```

Рис.3.34 – запис зміних в записах Ansible

Тепер, якщо хочете скопіривать тільки MY_FILE_HERE_1, виконайте**:**

 # ansible-playbook main.yml --tags "MY_FILE_HERE_1"

Приклад Хеш-цикла:

```
- name: vim files
synchronize:
src=/absolute/repo/path/roles/common/files/{{ item.src }}
dest={{ item.dst }}
with_items:
- {src: vim/bundle, dst: /home/jefdaj/.vim }
- {src: vim/vimrc , dst: /home/jefdaj/.vimrc}
- {src: vim/bundle, dst: /root/.vim }
- {src: vim/vimrc , dst: /root/.vimrc }

- name: vim permissions
file: path={{ item.pth }} owner={{ item.own }} group={{ item.grp }} recurse=yes
with_items:
- {pth: /root/.vim , own: root , grp: root }
- {pth: /root/.vimrc , own: root , grp: root }
- {pth: /home/jefdaj/.vim , own: jefdaj, grp: jefdaj}
- {pth: /home/jefdaj/.vimrc, own: jefdaj, grp: jefdaj}
```

Рис.3.35 – запис зміних маршрута в записах Ansible

 # ansible -i my_inventory all -a "/bin/uptime" -f 10

Запускаємо команду «-a« / bin / uptime »» на всіх хостах (опція «all») з певного файлу (опція «-i») з встановленим паралельним використанням в 10.

Запускаємо команди з використанням модуля «shell»:

 # ansible -i my_inventory some_host -m shell -a 'echo $TERM'

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділу було розглянуто докладно і зрозуміло про повну налаштуванні ELK Stack. Інформацію в основному почерпнув в офіційній документації. Мені не попалося більш чи менш докладних статей ні в рунеті, ні в буржунете, хоча шукав я грунтовно. Начебто такий популярний і ефективний інструмент, але статей більше ніж просто дефолтна установка я майже не бачив. Буквально одна на Хабре попалася з якоюсь більш чи менш кастомизацией.

Якісь перевірені моменти я не став описувати в роботі, так як вважав їх незручними і не став використовувати сам. Наприклад, якщо відмовитися від logstash і відправляти дані з beats безпосередньо в elasticsearch, то на перший погляд все стає простіше. Штатні модулі beats самі Парс висновок, встановлюють готові візуалізації і дашборда в Kibana. Вам залишається тільки зайти і милуватися красою :) Але на ділі все виходить не так красиво, як хотілося б. Кастомізація конфігурації ускладнюється. Зміна полів в логах призводить до більш складним налаштувань по введенню цих змін в систему. Всі настройки інформації, що надходить переносяться на кожен beats, змінюються в конфігах окремих агентів. Це не зручно.

У той же час, при використанні logstash, ви просто відправляєте дані зі всіх beats на нього і вже в одному місці всім керуєте, розподіляєте індекси, міняєте поля і т.д. Всі настройки переміщаються в одне місце. Це більш зручний підхід. Плюс, при великому навантаженні ви можете винести logstash на окрему машину.

ВИСНОВКИ

У данній дипломній роботі був проведений аналіз методів та засобів моніторину мікросервісних додатків. Зокрема, проведено огляд сучасних засобів моніторингу мікросервісних додатків, обробки та збирання інформації з окремих сервісів системи для подальшої її аналітики.

Підводячи підсумок скажу, що з цією системою зберігання логів потрібно дуже вдумливо і уважно розбиратися. З наскоку її не осилити. Щоб було зручно користуватися, потрібно багато всього налаштувати. Я описав тільки трохи кастомізованих збір даних, їх візуалізація - окрема розмова. Сам я в даний час вивчаю систему, тому буду радий будь-яким порадам, зауваженням, цікавим посиланням і всьому, що допоможе освоїти тему.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Встановлення ELK систем -https://serveradmin.ru/ustanovka-i-nastroyka-elasticsearch-logstash-kibana-elk-stack/
2. Логування - https://efsaver.com.ua/docs/loguvannya.html

Посібник з логування - https://codeguida.com/post/1415

встановлення і налаштуванн Ansible- https://linux-notes.org/ustanovka-i-nastrojka-ansible-v-unix-linux/

3. Terraform- https://habr.com/ru/company/piter/blog/351878/
4. Elasticsearch і його примінення - https://www.elastic.co/products/elasticsearch

Код Програм:

Terrafor:

Main.tf

```
###STARTUP SCRIPT FOR  kibana
data "template_file" "startup_script_kibana_server" {
 # count = "${var.total_nodes}"
  template = "${file("${path.module}/kibana-master-start-up-script.sh")}"
  vars = {
    ip_address      = "${var.ipv4_prefix}"
    name_server     = "${var.name}"
    git_key         = "${file("/Users/oleg/.ssh/id_rsa_git")}"
#    git_key         = "${data.local_file.foo.content}"
    target_size               = "${var.total_nodes}"
    #rabbitmq_username         = "admin"
    #rabbitmq_password         = "${var.elk_password}"
  }
}

data "google_compute_network" "env_network" {
  name = "${var.network}"
}

resource "google_compute_instance" "kibana"  {
  tags = ["kibana"]
  machine_type = "${var.machine_type}"
  name = "${var.name}-${count.index}"
  zone = "${var.zone}"
  count = "${var.total_nodes}"
  boot_disk {
    auto_delete  = true
```

```
    #sourse =
    #type        = "PERSISTENT"

    initialize_params {
      #size = "10"
      type = "pd-ssd"
      image = "${var.image_id}"
      }
  }

  network_interface {
    network = "${data.google_compute_network.env_network.self_link}"
    subnetwork    = "${var.subnetwork}"
    network_ip    = "${var.ipv4_prefix}${count.index}"

    #acces_config =
    #alias_ip_range =
  }
  metadata_startup_script =
"${data.template_file.startup_script_kibana_server.rendered}"

  lifecycle {
    create_before_destroy = false

  }
    service_account {
    email  = "${var.service_account_email}"
    scopes = "${var.service_account_scopes}"
  }

}
```

```
####FIREWALL for ELK
resource "google_compute_firewall" "kibana" {
  name    = "${var.name}-firewall-kibana"
  network = "${data.google_compute_network.env_network.self_link}"

  allow {
    protocol = "tcp"
    ports    = ["22" , "80" , "443"]
  }
  allow {
    protocol = "icmp"
  }
}

variable "name" {
  default= "kibana"
}
variable "zone" {
  default = "europe-west1-b"
}
variable "project" {
  default = "~/test-terraform/infra/modules/terraform-google-kibana"
}
variable "region" {
  default = "europe-west1"
}
variable "machine_type" {
  default = "g1-small"
}
variable "image_id" {
```

```
  default = "debian-cloud/debian-9"
}
variable "total_nodes" {
  default = "1"
}

variable "service_account_email" {
  default = "oleh.burachynskyi@.com"
}
variable "disk_size_gb" {
  default = "50"
}
variable "subnetwork" {
  default = "default"
}
variable "ipv4_prefix" {
  default = "10.0.0.10"
}
variable "network" {
  default = "default"
}
variable "service_account_scopes" {
  type    = "list"
  default = ["userinfo-email", "compute-rw", "storage-rw", "logging-write"]
}
```

ANSIBLE

Main.tf

```
---
#
# Installing Nginx
#
# Install Nginx
- name: Update repositories cache and install Nginx
  apt:
   name: nginx
   update_cache: yes
# Create /etc/nginx/conf.d/ directory
- name: Create nginx directory structure
  file:
   path: '/etc/nginx/conf.d/'
   state: 'directory'
   mode: '0755'
# Deploy kibana.conf with FQDN
- name: Add nginx ssl-params.conf
  template:
   src: 'ssl-params.conf'
   dest: '/etc/nginx/snippets/ssl-params.conf'
   owner: 'root'
   group: 'root'
   mode: '0644'

- name: delate /etc/nginx/sites-available/default
  file:
    path: /etc/nginx/sites-available/default
    state: absent
```

```
- name: Setup Nginx reverse proxy for kibana
  template:
   #sudo: yes
      src: 'kibana.conf.j2'
      dest: '/etc/nginx/sites-available/default'
      owner: 'root'
      group: 'root'
      mode: '0644'
  register: nginx_needs_restart

# Enable nginx service
- name: Enabling Nginx service
  systemd:
   name: nginx
   enabled: yes

# Creating certs directories for SSL
- name: Creates SSL directories
  file:
   path: /etc/pki/tls/certs
   state: directory
# Creating private directories for SSL
- name: Creates SSL directories
  file:
   path: /etc/pki/tls/private
   state: directory
# Update SSL to restrict outside access
- name: Updating the config file to restrict outside access
  lineinfile:
   destfile: /etc/ssl/openssl.cnf
```

```
    regexp: 'subjectAltName ='
    line: 'subjectAltName = IP: {{ elk_ip }}'
# Generate SSL certificates for Logstash
- name: Generate SSL certificates
  shell: "openssl req -config /etc/ssl/openssl.cnf -x509 -days 3650 -batch -nodes -newkey rsa:2048 -keyout /etc/pki/tls/private/{{server_name}}.key -out /etc/pki/tls/certs/{{server_name}}.crt"


# Start Nginx service
- name: Starting Nginx service
  systemd:
    name: nginx
    state: started
    daemon_reload: yes

# Install Pexpect to handle promts of the terminal
- name: Installing Python Pexpect
  apt:
    name: python-pexpect
    update_cache: yes
# Writes the create user script in the temp directory
- name: Create kibana admin user
  template:
    src: 'kibanaAdmin.j2'
    dest: '/tmp/createUser'
    owner: 'root'
    group: 'root'
    mode: '0744'
# Runs the script to create Kibana admin user
- name: Create Kibana admin user
```

```
  expect:
    command: bash /tmp/createUser
    responses:
      'Password:' : "{{kibana_password}}"
# Restart Nginx service
- name: Restart Nginx service
  systemd:
   name: nginx
   state: reloaded
   daemon_reload: yes
```

kibana.conf

```
upstream kibana {
    server 127.0.0.1:5601 fail_timeout=0;
}
server {
        listen {{nginx_kibana_port}};
        listen 443 ssl http2;
        server_name {{server_name}};

         auth_basic "Restricted Access";
         auth_basic_user_file /etc/nginx/htpasswd;

         ssl_certificate /etc/pki/tls/certs/{{server_name}}.crt; # managed by Certbot
         ssl_certificate_key /etc/pki/tls/private/{{server_name}}.key; # managed by
Certbot
         include /etc/nginx/snippets/ssl-params.conf;
```

```
        location / {
            proxy_pass        http://127.0.0.1:5601;
           # proxy_pass http://{{server_name}}:5601/app/kibana;
            proxy_set_header   Host $host;
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Connection 'upgrade';
            proxy_set_header   X-Forwarded-Host $server_name;
            proxy_read_timeout  1200s;

            # used for view/edit office file via Office Online Server
            client_max_body_size 0;

            access_log      /var/log/nginx/seahub.access.log;
            error_log       /var/log/nginx/seahub.error.log;
        }
}
```

SSL.conf

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

ssl_prefer_server_ciphers on;

#ssl_dhparam /opt/nginx/ssl/dhparam.pem;

ssl_ciphers ECDHE-RSA-AEW256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ERHE-RSA-ADF256-GCM-SHA384:DSE-RSA-AES256-GCM-SHA384:ESDDHE-RSA-AES256-SDA384;

ssl_ecdh_curve secp384r1; # Requires nginx >= 1.1.0

ssl_session_timeout 10m;

ssl_session_cache shared:SSL:10m;

ssl_session_tickets off; # Requires nginx >= 1.5.9

 # ssl_stapling on; # Requires nginx >= 1.3.7

 # ssl_stapling_verify on; # Requires nginx => 1.3.7

resolver 8.8.8.8 8.8.4.4 valid=300s;

resolver_timeout 5s;

add_header X-Frame-Options DENY;

add_header X-Content-Type-Options nosniff;