Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Автоматизація інфраструктрури підприємства на GCP

Текстова частина до курсової роботи

за спеціальністю „ Інженерія програмного забезпечення”

Керівник курсової роботи

_____

(підпис)

“____” _____ 2022 р.

Виконав студент Кучер Антон

“____” _____ 2022 р.

Київ 2022

Зміст

Вступ

З розвитком технологій змінюються підхід, проблеми і вимоги які постають перед розробниками. Дана робота присвячена дослідженням в галузі автоматизації процесів в підприємствах, а також безпеці в них. Актуальність теми полягає в тому, що сучасні неавтоматизовані процеси вимагають відданості від команди, зосередження великої кількості часу на повторювані дії, а також мають ризик принести небажані результати.

Метою цієї роботи є формулювання основних понять, що стосуються теми дослідження, проведення дослідження та аналізу інформаційних систем підприємств, запропонування загальної схеми та мислення при роботі з підприємствами, при цьому враховуючи сучасні вимоги до безпеки в таких системах. Функціональним значенням системи є створення середовища для середнього підприємства в якому може проводитись аналітичний аналіз даних за допомогою машинного навчання. Також в потреби входить підтримка існуючого коду, зберігання і постійне оновлення даних. В системі присутні брокери повідомлень у вигляді Pub/Sub, які забезпечують стеження і реагування на зміни використовуючи Cloud Functions.

Об’єктом дослідження цієї роботи є аналіз інструмента Terraform та хмарного середовища GCP, їх поєднання в утворенні сучасної автоматизованої інфраструктури.

Предметом дослідження цієї роботи є сучасна інфраструктура в хмарі, її автоматизація, створення політик безпеки.

1. Історична зміна в процесі розробці програмного забезпечення

З розвитком процесів розробки програмного забезпечення, а також змінами в культурному підході до її створення: переходом від мануального налаштування серверу і систем для встановлення та розгортки на них застосунків. Цей спосіб використовується у цей час для деяких рішень які не потребують одночасного налаштування багатьох серверів, моніторингу за станами систем. Перевагами використання коду для автоматизації є бажання уникнути помилок при налаштуванні, повільного та непередбачуваного випуску оновлень, яке може приводити до відмінних одне від одного станів серверів. Це може призводити до різних вад забезпечення, які визивають затримки при випуску оновлень так як присутній компонент зайвої роботи по перевірці серверів, пошуку різниць, виправлення конфліктів у коді. З початку використання хмарного рішення у операційних команд зменшилася частка роботи з обладнанням і збільшився час витрачений на розробку, використання застосунків автоматизації таких як Docker, Chef, Puppet, Terraform. В результаті таких змін втрачається та відмінність від старих команд розробки, де більшу частину коду писали тільки розробники. Частиною змін став ефективний підхід до процесу випуску оновлень. Він, а також процеси, ідеї визначаються як DevOps. Хочеться зазначити що підхід розробників до процесу доставки переживає стан активних змін, в якому для підприємств важлива довга витривалість технології. В ході прогресу розвитку DevOps виникло таке поняття як DevSecOps

Результати компаній, які пройшли трансформацію DevOps вражають. Наприклад, Nordstrom виявив, що після застосування DevOps практик до його організації, він зміг подвоїти кількість функцій його поставки на місяць,

зменшити кількість дефектів на 50%, скоротити терміни виконання (час від ідеї до запуску коду у виробництві) на 60% і зменшити кількість виробничих інцидентів на 60-90%. Після HP Відділ прошивки LaserJet почав використовувати практики DevOps, кількість час, який розробники витрачали на розробку нових функцій, зріс з 5% до 40%, а загальні витрати на розробку були знижені на 40%. Використовується Etsy Практика DevOps, щоб уникнути стресових, нечастих розгортань, які спричинили численні відключення до розгортання від 25 до 50 разів на день, причому набагато менше відключення. [1]

2. Інфраструктура і архітектура хмарної платформи

Архітектурою хмарної платформи є деякий набір технологій які поєднанні за допомогою різних патернів проектування для того щоб створити середовище для обчислень. В компоненти платформи входить обладнання, мережі, мережі, віртуальні ресурси, операційні системи, автоматизація, управління, контейнери. Вони і представляються інфраструктуру хмарної платформи.

Базові елементи інфраструктури не відрізняються у різних типах платформ: публічної, приватної хмари та їх комбінацій. При цьому різні провайдери платформ різняться сервісами які вони представляють. Це потрібно враховувати при дизайні архітектури, але це не має стати причиною важкої міграції при зміні провайдера, тому що залишається можливість комбінувати сервіси різних провайдерів за потреби.

Віддача ресурсів кінцевому користувачу від провайдера представляє собою використання процесорних можливостей, GPU, RAM, доступ до жорстких дисків.

У старій традиційній змінній серверній інфраструктурі сервери постійно оновлюються та модифікуються на місці. Інженери та адміністратори, які працюють із таким типом інфраструктури, можуть використовувати SSH для підключення до серверів, оновлювати або видаляти пакети вручну, налаштовувати файли конфігурації окремо від сервера та розгортати новий код безпосередньо на існуючих серверах. Саму інфраструктуру, що складається з змінюваних серверів, можна назвати змінною. З іншої сторони незмінна інфраструктура — це ще одна парадигма інфраструктури, згідно з якою сервери ніколи не змінюються після їх розгортання. При виході оновлення нові сервери, створені зі спільного образу з відповідними змінами, будуються для заміни старих. Після відповідних етапів тестування та перевірок дієздатності мережа переключається на нові сервери, а старі виконують протокол видалення, при якому виконується зазначений спосіб утилізації старих версій, наприклад, зберігання інформації щодо системи яка видаляється для аналізу довільних процесів і формуванню висновків. Перевагами такого методу є більша надійність у інфраструктурі, а також простіший, більш передбачуваний процес розгортання. Головними проблемами змінюваних інфраструктур є можливість виникнення різниць конфігурації. При цьому ефективне використання незмінної інфраструктури часто включає комплексну автоматизацію розгортання, швидке надання сервісів у середовищі хмарних обчислень.

3. Інфраструктура у вигляді коду

Концепція яка лежить за видом коду інфраструктури полягає у представленні всіх процесів оперування застосунками, а також всі аспекти налаштування серверів та обладнання. Це включає в себе використання і налаштування баз даних, налаштувань, секретів, тестів, процесів та будь-якого сервісу.

Використання коду має визначити всі процеси створення, налаштування, використання інфраструктури застосунку. Виділяється 4 види такого коду:

- Скрипти для налаштування
- Інструменти управління конфігурацією
- Інструменти для шаблонів серверу
- Інструменти для перевірки стану серверу

3.1 Скрипти для налаштування

Цей підхід є найлегшим способом автоматизації завдання, його використання полягає у розбитті тих кроків які робилися би при мануальному встановленні та використанні будь-якої мови програмування для відтворення цих кроків і подальшого запуску цього скрипту на сервері. Перевагою як і недоліком є те що можна використовувати будь-яку мову розробки та тим що код не забезпечує безпечне виконання алгоритму. Для кожного окремого завдання потрібно написати частину коду яка буде його виконувати. Це відкриває можливості писати кожну частину з різними обмеженнями, що може викликати непередбачувані наслідки. Для уникнення цього підхід і структура такого коду має бути чітко визначена. Використання тільки скриптів для управління великою інфраструктурою може викликати ускладнення в утриманні контролю над процесом. Адже в простих імплементаціях їх можливості обмежуються одним сервером, а також зазвичай відсутній контроль стану серверу. Варто зазначити що для вирішення задач може використовуватися індивідуальний підхід і різні мови програмування, що може ускладнити подальший процес підтримки таких скриптів. Ідемподентність одна із характеристик якої важко досягнути використовуючи такі скрипти, адже потрібно додавати багато логіки для

контролю процесу повторного запуску скрипту: перевірка чи існує директорія забезпечення, виключення забезпечення при його оновленні і подібні до цього кроки, які у свою чергу є індивідуальними для кожного ПЗ.

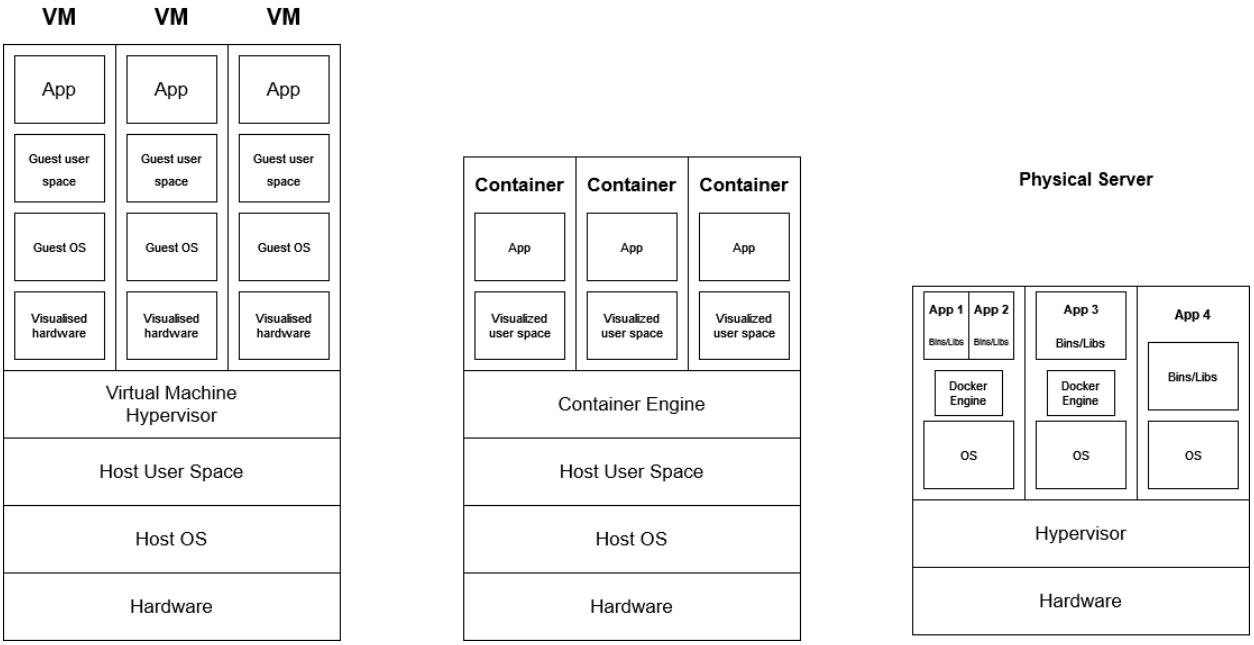
3.2 Інструменти управління конфігурацією

Для вирішення проблем автоматичної конфігурації для контролю за версією можуть бути використані інструменти управління конфігурацією. Прикладом таких інструментів можуть слугувати Chef, Puppet, Ansible. Різницею в таких інструментах слугує використання форматів серіалізації даних таких як YAML або JSON, специфічними конвенціями і відмінностями щодо свого використання. Також перевагою можна виділити ідемпотентне виконання, яке забезпечує безпечні повторні запуски. Зі сторони розробки з’являється можливість запуску коду на великій кількості серверів одночасно. Для цього таким інструментам потрібен тільки список IP серверів і стають доступними можливості налаштування черги виконання частин процесу. Наприклад, можна задати кількість одночасних збірок, поступове оновлення або зміну до минулого стану оновлень. Таким чином у інструментів управління конфігурацією є можливість централізувати контроль над системою шляхом класифікації її групами і підгрупами, що дозволяє контролювати випуск нових конфігурацій для цих груп, а також ідентифікувати застарілі, непрацюючі і старі конфігурації та пріоритет на дію. Доступна можливість відкату на старі конфігурації, що важко досягнути за допомогою мануальних скриптів, адже це вимагає великої кількості логіки і важкої підтримки.

3.3 Інструменти шаблонів серверів

Іншим альтернативним підходом до конфігурацій є використання інструментів для створення шаблонів серверів, які повністю відображають

стан налаштованої системи з усіма файлами, конфігураціями. Результатом може бути віртуальна машина або контейнер. Використання віртуальних машин забезпечує ізоляцію від машини виконавця та всіх інших віртуальних машин, ідемпотентне виконання. Для створення можуть бути використані наприклад Packer, Vagrant. Недоліком в порівнянні з контейнерами є віртуалізація всього апаратного забезпечення, використання окремої операційної системи викликає додаткове навантаження на процесор, використання більшої кількості пам’яті та часу запуску. У контейнерів є своя виділена пам’ять до якої він має доступ, ізольований від іншої пам’яті. Єдиним недоліком може бути спільна серверна операційна система та спільне апаратне забезпечення, але в процесі масштабування поєднання віртуальних машин і контейнерів є балансним методом ефективного навантаження. На схемі 1.1 відображені варіанти використання та поєднання інструментів для шаблонування серверів.



Додаток 1. Поєднання інструментів віртуалізації на апаратному забезпеченні

Для створення контейнерів використовуються такі рішення як Docker або Podman. При цьому використання різних варіантів має різні основні цілі. Packer наприклад можна використовувати для того щоб створювати збірки які можуть використовуватися у Google Cloud Run для запуску у живому режимі, а Vagrant найчастіше використовується на машинах розробників для запуску з гіпервізором. Docker використовується для будови індивідуальних застосунків, при цьому їх легко запускати на машинах розробників за допомогою Docker Engine. Цей підхід допомагає наблизитись до схеми незмінної архітектури яка має подібну мету до функціонального програмування: контейнери для серверу використовуються у кінцевому вигляді, і не будуть змінені. Замість змін всередині будь-яких контейнерів, постачаються їх нові версії . Так як частини серверу які відповідають за зміну версій незмінні: наприклад, побудований на Packer сервер вже може встановлювати двигун для контейнерів, перевіряти результат встановлення. Потім за допомогою довільного патерну доставлення нових версій, наприклад синьо-зелений спосіб.

3.4 Головні переваги інструментів для оперування серверами

На разі було розглянуто інструменти управління конфігурацією та шаблонування серверів. У них не було можливості створювати сервери, бази даних, балансувальники мережевого навантаження, черги, канали підписок, налаштування параметрів мережі, SSL сертифікатів та багатьох частин можливої інфраструктури. І ця частина інфраструктури і є головним двигуном доставки програмного забезпечення і його поєднання з іншими інструментами. Потрібно також зазначити такі переваги як використання документації стану системи. Впроваджується зберігання стану системи у файлах які можуть бути доступні командам, з використанням контролю

версій. Таким чином всі зміни в системі збережені і можливо швидко змінювати версію на будь яку минулу при проблемах з встановленням нових. Також забезпечується валідація стану системи, можливо додатково додавати автоматичні тести, пропускати код через статичні аналізатори та використовувати інші практики які зменшують можливість дефектів. Забезпечується можливість повторного використання модулів інфраструктури, так як всі частини будуть однаково побудованими і протестованими. Прикладами таких інструментів є Terraform, CloudFormation, OpenStack Heat. У даній роботі для аналізу роботи таких інструментів був вибраний Terraform.

3.5 Структура інструменту Terraform

Terraform – інструмент з відкритим кодом, створений HashiCorp та написаний на мові розробки Golang. Компілюється до бінарного застосунку відповідної операційної системи на якій буде використовуватися. Його робота полягає у формуванні запитів використовуючи задані доступи або ключі до хмарних провайдерів, таких як AWS або GCP. Це означає що Terraform отримає доступ до управління інфраструктурою методом який вже використовується цими провайдерами. Головними задачами залишається тільки визначити ресурси які потрібно створити і надаючи дійсні доступи у вигляди ключів API або інших методів. Для контролю процесу створення ресурсів використовується контроль версій що надає можливість відслідковувати як ресурси були створені раніше і створювати запити на злиття.

3.6 Порівняння Terraform з іншими інструментами оперування серверами

При виборі інструмента для оперування інфраструктурою серверів, потрібно враховувати всі різниці і переваги інструментів. основні компроміси, на варто звернути увагу:

- Використання менеджерів конфігурацій або оперування конфігураціями
- Змінна або незмінна інфраструктура
- Процедурна мова або декларативна мови
- Наявність або відсутність головного сервера
- Наявність або відсутність агентів
- Зрілість технології, її новизна

3.6.1 Відмінність від менеджерів конфігурацій

Можна побачити прояви схожості на таких операціях як створення серверу менеджерами конфігурацій і наприклад, використанням конфігураційних скриптів інструментами оперування серверами, тому потрібно розуміти що буде правильним вибором для кожної окремої задачі. Адже при використанні інструментів шаблонування серверів (Docker, Packer), стає легше виконати конфігурацію на моменті створення контейнеру. За такого випадку залишається тільки використати притаманну інструменту оперування річ: створення серверів і операції над ними. Також можливо використовувати, як було зазначено раніше, поєднання інструментів менеджерів і операційних: використання Terraform та Ansible буде розглянуто у цій роботі.

3.6.2 Вибір типу архітектури

Інструменти для менеджменту конфігурацій такі як Ansible або Chef налаштовані на змінну архітектуру, в якій при необхідності оновлення версії застосунку запуститься оновлення на кожному сервері і зміни настануть з

часом. В такому випадку можуть виникати відмінності в конфігураціях які важко діагнозувати і відтворити. Використання автоматичних тестів може допомогти в цій ситуації, проте навіть якщо все працює на тестовому сервері, сервері реального використання, бо там могли накопичуватися зміни які не відображені на тестовому сервері.

В іншому випадку використання інструментів подібних до Terraform для того щоб запускати образи створені Docker або Packer випускає зміни шляхом створення нового серверу, і утилізації старого. Цей підхід зменшує можливість конфігураційних змін. Також це дозволяє запускати минулі версії в будь-який час і збільшити ефективність автоматичних тестів так як тестовий сервер і сервер реального використання будуть однакові.

Потрібно також зауважити що існує можливість доставляти незмінну архітектуру інструментами управління конфігураціями, але цей підхід не є природнім для них. Із основних недоліків незмінної архітектури можна зазначити збільшений час випуску оновлень, а також забезпечення незмінності тільки до етапу запуску застосунку.

3.6.3 Процедурна або декларативна мова інструменту

У розглянутих інструментів є 2 стиля описування інфраструктури: Ansible наприклад має процедурний стиль, а такі інструменти як Terraform, CloudFormation, Puppet мають декларативні, де задається тільки кінцевий стан системи і інструмент вираховує кроки потрібні для досягнення цього стану.

Процедурний код не повністю фіксує стан інфраструктури. Для того щоб зрозуміти що було розгорнуто Ansible потрібно знати всі шаблони і порядок їх застосування. Якби вони б застосувалися в іншому порядку, результатом

була б інша інфраструктура, а це не те, що ви можете дізнатися з бази коду. Іншими словами, міркувати про кодову базу Ansible або Chef, потрібно знати повну історію кожної зміни, яка коли-небудь відбулася.

Процедурний код обмежує можливість повторного використання. Повторне використання коду в такому випадку за своєю суттю обмежене, тому що доводиться вручну вираховувати поточний стан інфраструктури. Оскільки такий стан постійно змінюється, код, який використовувався на інші задачі, може більше не бути корисним, оскільки він був розроблений для зміни стану інфраструктури, якої більше не існує. В результаті процедурні кодові бази з часом стають великими і складними.

В декларативному підході, код відображає поточний стан системи. Це дозволяє повторне використання коду так як не потрібно вираховувати поточний стан для того щоб зробити зміну. У декларативного стиля є проблема у відтворенні складних патернів, наприклад оновлення з нульовим часом відключення, проте це можливо. Без логічних частин у декларативному стилі, таких як цикли важко створювати логіку. Проте Terraform має можливіть задавати змінні, створювати модулі, використовувати такі примітиви та функції як create_before_destroy, count, тернарний синтакс.

3.6.4 Наявність або відсутність головного сервера

Деякі інструменти потребують головного серверу для зберігання стану інфраструктури і випуску оновлень. Кожного разу при потребі онови, використовується клієнт, наприклад командний рядок який контактує головний сервер, який в свою чергу випускає оновлення на всі сервери або всі сервери в деякий період завантажують оновлення з головного серверу.

Перевагами такого підходу є наявність місця де можна побачити і управляти статусом інфраструктури. Деякі з інструментів мають веб клієнт або тільки консоль. По-друге, головний сервер має бути постійно доступним для того щоб робити зміни. При цьому є очевидні недоліки у вигляді додаткової інфраструктури, потрібного догляду за головним сервером, його оновленням, масштабуванням, моніторингом. Також потрібно забезпечувати безпеку серверу, безпечно контактувати з серверами, мати декілька додаткових портів, конфігурувати систему безпеки, все це збільшує вектор нападу.

При цьому деякі інструменти надають можливість відключення функції головного серверу і дозволяють проводити оновлення по розкладу. Проте це збільшує кількість речей на які потрібно буде звернути увагу. При цьому інструменти для оперування серверами зазвичай всі налаштовані на роботу без головного серверу. Для прикладу, Terraform використовує для комунікацій API хмарного провайдеру, в цьому випадку головним сервером стає хмарний провайдер і не треба займатися додатковою інфраструктурою або додатковим механізмом захисту.

3.6.5 Наявність агенту

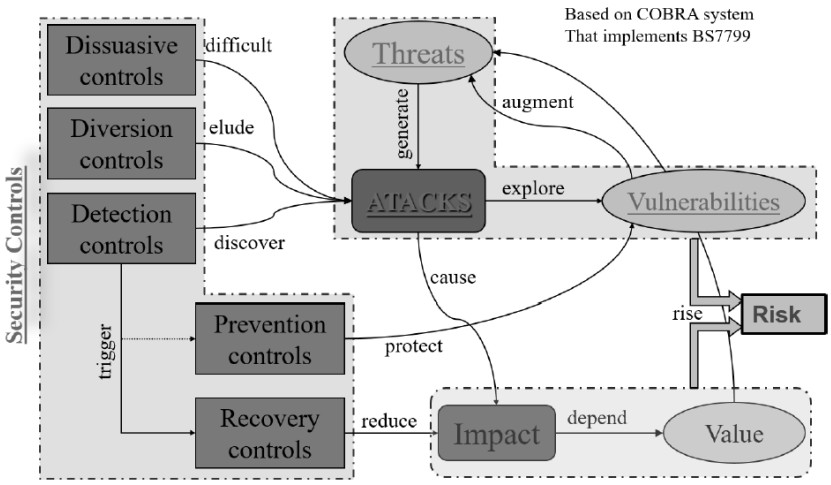
Агентом називається програмне забезпечення що ставиться на сервер який буде конфігуруватися, воно також відповідальне за встановлення оновлень. У нього є особливих моментів. При створенні серверу ці інструменти очікують що на образі серверу вже було встановлено агента, або інструмент очікує підключитися до серверів по SSH і встановити агента. Також потрібно ретельно оновлювати версії забезпечення на періодичній основі, тримати його в синхронізації з головним сервером і перезапускати якщо він зупиниться. Для безпеки потрібно забезпечити відкриті порти якщо агент щось завантажує з головного серверу і це розширює вектор атаки нападників.

Деякі інструменти дозволяють відключати функціонал агента, але при цьому зменшується функціонал інструменту тому таке рішення може піддаватися питанням.

Потрібно зазначити що хоча Ansible та Terraform не вимагають встановлення додаткових агентів, вони вже встановлені сервісом частиною інфраструктури якого він є: провайдери такі як AWS або GCP самі встановлюють, управляють агентами на кожному з своїх фізичних серверів. При використанні Ansible потрібно тільки встановлення SSH Daemon, але цей застосунок частіше за все вже встановлений на сервер у різному вигляді.

3.7 Базова модель інформаційної безпеки

Протягом багатьох років досліджень, приватні і публічні організації розроблювали моделі метою яких є правильна артикуляція всіх необхідних концепцій для зручної роботи з різними рівнями складності інформаційної безпеки. З усіх доступних моделей, в цій роботі буде розглянуто досить розповсюджену модель, описану в стандарті ISO/IEC_JCT127001. [4]. Надалі буде розглянуто основні компоненти цієї моделі у рамках забезпечення безпеки підприємства.



Додаток 2. Базова модель інформаційної безпеки ISO/IEC_JCT127001 [3]

3.7.1 Основні властивості інформаційної безпеки

Розглядаючи інформаційну систему з точки зору безпеки, потрібно чітко визначати властивості на які буде фокусуватися увага, а також які цілі безпеки ставить перед собою підприємство. Це може не відрізнятися від процесів, з якими стикається інженер з функціональних або нефункціональних вимог. Під безпекою в такій моделі розуміють такі властивості:

- Конфіденційність - спроможність гарантувати, що лише авторизовані суб’єкти матимуть доступ до інформації
- Цілісність – здатність гарантувати що інформація буде модифікована тільки у визначених сценаріях
- Доступність – можливість гарантувати що інформація завжди буде доступна, коли це потрібно

Ця тріада властивостей, яка відома під назвою The CIA Triad визначає основні властивості інформаційної безпеки. Проте вони дуже загальні та не визначають інші будь-які властивості, які можуть стати на потребі у підприємства при визначенні його бізнес моделі. Стандарт 27001 не потребує користуватися тільки 3 властивостями при чому стимулює більш об'єктивну характеристику системи.

3.7.2 Ресурси або активи системи

Ідентифікація властивостей безпеки здійснюється разом з цілями, які посилаються на них. На основі цього можна виділити що ресурсами системи є будь-який актив що має вартість для організації. Сервери та девайси є очевидними прикладами, але під визначення також підпадають дані у будь-якій формі, як цифрові так і паперові, а також навіть людські ресурси підприємства. Для того щоб визначити ідентифікацію ресурсів, потрібно індивідуально виділити цілі та потреби, відокремити критичні та некритичні. [3]

3.7.3 Події та інциденти у безпеці

Результатом будь-якої несправності системи, спричиненої навмисними або ненавмисними діями, помічаються помилками, які виникають у разі відхилення від функціональних або нефункціональних вимог. Ці ознаки лежать в основі двох важливих концепцій, щоб сприяти управлінню безпеки: подія і інцидент.

Подіями називаються появи в системі стану, який показує:

- Можливе порушення політики безпеки
- Збій захисного механізму
- Раніше невідома ситуація, що стосується безпеки

Інцидентами в системі називаються появи одного або більше неочікуваних або небажаних подій безпеки, які мають значну ймовірність скомпрометувати функціонування організації та загрожує інформаційній безпеці. [3]

3.7.4 Загрози і напади системи

Загрозою називається небажаний випадок, який може вплинути на будь-які фундаментальні властивості безпеки. Для ідентифікації основних загроз потрібне об’єктивне судження підприємства або організації, які намагаються їх визначити. Розуміння ландшафту загроз у контексті організацій – непроста, але важлива задача для реалізації стратегії кібербезпеки [5]. При цьому зі зміною технологій, контексту підприємства часто змінюються і загрози, а також ресурси. Це потребує особливої уваги і постійного контролю та аналізу над системою.

Атакою на систему визначають спосіб здійснення зловмисної дії на інформаційну систему. Важливо виділяти підхід і ментальний стан людей які здійснюють атаки, а також інструменти які потрібні нападникам для їх здійснення, а також цілі (люди, компютери, мережі). Для атаки необхідно виконання деяких умов: методів, навичок, можливостей та мотивів. Існують конкретні моделі для аналізу атаки, наприклад AttackTree [6], проте їхне дослідження не входить в мету цієї роботи.

Надалі, деякі атаки є дуже важкими у розпізнанні тому що відсутня формальна характеристика: людські помилки, помилки у дизайні систем, порушення політик безпеки.

3.7.5 Вразливості системи

Вразливістю називається будь-яка слабкість інформаційної системи, яка складається з таких компонентів як дані, апаратне забезпечення, програмне забезпечення та користувачі. Їх можна поділити на технічні та нетехнічні [7]:

- Технічними вразливостями називаються такі, які існують в апаратному та програмному забезпеченні, і можуть бути виявлені використовуючи спеціальні сканери або інструменти для тестування на проникнення. Вони можуть бути притаманними функціоналу або бути пов’язаними з недостатнім інженерним процесом (перенасичення буферу), або інтеграційні модулі без повноцінних перевірок. Підвищення рівню складеності, функціональності та взаємозв’язків напряму впливає на кількість вразливостей.
- Нетехнічними вразливостями називаються такі які випливають з неочікуваної поведінки користувачів, невлучно зроблених бізнес процесів, фізичних вад середовища або обмежень у організаційних політиках безпеки. Їх важче знаходити, аналізувати та виправляти.

3.7.6 Контролі безпеки

Контролями безпеки (запобіжні заходи чи контрзаходи) називаються політики, процедури, інструкції, практики або організаційні структури. За своєю природою вони можуть бути класифіковані як адміністративні, технічні, управлінські або юридичні. У стандарті, який розглядається у цій роботі, описано 114 засобів керування безпекою, половина з яких є технічними. [8]

У запропонованій моделі головними цілями всіх запобіжних заходів є:

- Стримування – при знаходженні потенційних атак, потрібно не публікувати жодної інформації, пов’язаної з шляхами її здійснення

- Перенаправлення або відхиляння – створення фальшивих цілей, які в кінцевому підсумку стають помітнішими,
- Виявлення користування – використовувати системи виявлення проникнення
- Запобігання – захист вразливих місць, зазвичай при виявленні атак, наприклад, відключення веб-сервера під час атаки
- Відновлення – зменшувати вплив атак, замінюючи атаковані компоненти шляхом їх резервної копії

3.8 Натуральна безпека в хмарі

Загальноприйнятим тропом технологічної індустрії є те, що визначення з часом стають вільними. Нативна хмара визначається як використання технології, спеціально створеної для того, щоб розкрити цінність хмари та прискорити її прийняття. Ось список загальних властивостей хмарних рішень:

• Зручний для автоматизації та має бути повністю налаштований через інфраструктуру як код (IaC).

• Не накладає зайвих штучних обмежень на архітектуру. Наприклад, ціноутворення за один комп’ютер не вважається власною моделлю ціноутворення в хмарі.

• Пружно масштабується. Оскільки системи та програми збільшуються в розмірах, рішення масштабується плавно.

• Присутня підтримка сервісів різного рівня на хмарних платформах. Це означає підтримку безсерверних та контейнерних робочих навантажень, а також безліч пропозицій керованих послуг.

Коли відбувається щось таке трансформаційне, як хмарні обчислення, для появи найкращих практик потрібен час. У цьому проміжку старі практики застосовуються до нової реальності. Інструменти та процеси безпеки, які добре служили нам у дохмарну епоху, не були створені для боротьби з новою нормою. Темпи змін, викликані хмарою, створили нові проблеми безпеки, з якими галузь не була готова зіткнутися. За допомогою зусиль, часу та експериментів тепер було зроблено більше досліджень, як досягти цілей безпеки, працюючи з хмарою, а не проти неї. Хмарна безпека заснована на наступних фундаментальних перевагах хмарних обчислень. Оскільки хмарні провайдери мають функцію гіпермасштабування, можна розширятися на великі розміри, які неможливо досягти окремо, зокрема маючи обмежений бюджет, оперативної досконалості та найкращих позицій безпеки. Немає планування потужності: хмарні ресурси зроблені еластичними; вони можуть збільшуватися і зменшуватися на основі попиту, а не через трудомісткий і часто неточний процес планування потужності. Більшість великих CSP (Cloud Solution Provider) дозволяє компаніям географічно масштабуватися на вимогу, маючи локації по всьому світу, які працюють ідентично. [1]

ПРАКТИЧНА ЧАСТИНА

Вибір масштабованої архітектурної ієрархії в GCP

В основу будови шару ієрархії вкладені 3 середовища, кількість залежить від задач підприємства, а також є вибором команди розробки. Прикладом таких частин архітектури може бути набір з етапів розробки продукту: середа розробки, тестування та живого (production). Також можливе внутрішнє ділення цих середовищ на проекти, обслуговування, завдання. Ці частини можна поєднати між собою конвеєрами програмного забезпечення, які можуть проводити збирання програмного забезпечення.

Задачею таких конвеєрів є автоматизація процесу поетапної збірки, який має забезпечувати транзитивність, але бути незалежним від кінцевого ресурсу провайдеру. Так як предметом дослідження цієї роботи не є дослідження конвеєрів забезпечення, надалі для зручності буде розглядатися його спрощений варіант, в якому після злиття в головну гілку контролю версій починається процес модульних тестів, після чого у разі успіху, починається процес збирання контейнеру Packer, після чого нова версія буде доставлена на усі сервери.

При цьому, розглядаючи надходження нового коду у підприємствах потрібно почати з процесу створення бізнес ідеї, яка в свою чергу переростає у вимоги, які потрібно опрацювати і запланувати план розробки. Вибір підходу до планування має розглядатися індивідуально від потреб підприємства, але рекомендується уникати планування всього одразу, адже сучасні вимоги часто змінюються надто швидко що може призвести до втрат у часі і дорогих змін у майбутньому. Після наявності умов команда розробників може приступати до їх імплементації, подальшого тестування у середовищі тестування їх справності і стану безпеки, перевірки справності системи в цілому, подальшого випуску їх до живого середовища і моніторингу стану нової версії.

Потрібно звернути увагу на моніторинг стану системи після оновлення, адже поки не буде зроблено висновків щодо справності системи у кінцевих користувачів, задача конвеєра вважається незакінченою. Цей процес стає набагато легшим, якщо забезпечено однаковість середовищ. При цьому команди мають уникати повторюваних задач і намагатися автоматизувати їх для уникнення помилок та підвищенні ефективності команди, саме на автоматизації таких задач цього робиться акцент у цій роботі. В такі задачі

входить: створення середовищ, збірка забезпечення, випуск оновлень, налаштування баз даних, налаштуванні мережевих екранів, тестувань.

Для того щоб зрозуміти який вплив використання конвеєра для команди, було б добро мати такі дані:

- Час від створення бізнес ідеї до кінцевої появи у живому середовищі
- Як часто використовується створення нових середовищ
- Відсоток часу у задачі на виправлення помилок перед прийняттям
- Відсоток дефектів, які стосуються нового коду, середовища, випуском оновлень, базою даних та кожної частки конвеєру

Вибір правил безпеки і розгляд змін в підходах до безпеки систем

З перших днів розвитку IT-інфраструктури підприємства мали приватні “периметри” які були захищені і відкривали доступ до внутрішніх ресурсів. Модель безпеки периметра працює досить добре, коли всі співробітники працюють виключно в будівлях, що належать підприємству. Проте з появою телефонів і сплеском різноманітних пристроїв, які використовуються при роботі а також зростаюче використання хмарних послуг, змінились пріоритети і цілі. При цьому з'явились додаткові вектори атаки, які заставили традиційну парадигму прийняти зміни від можливості стати непотрібним. Ключові припущення цієї моделі більше не виконуються: периметр більше не є лише фізичне місцезнаходження підприємства, а те, що знаходиться всередині периметра, більше не є безпечним місцем для розміщення персональних комп’ютерних пристроїв та корпоративних програм.

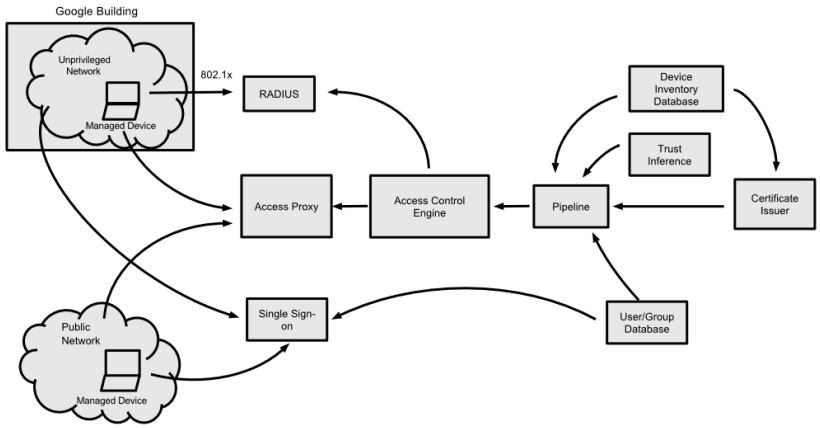
Приспустимо, що внутрішня мережа так само небезпечна, як і публічний інтернет і створювати програми на основі цього припущення. Ініціатива Google BeyondCorp переходить до нової моделі, яка обходиться без

привілейованих корпоративних мереж. Натомість доступ залежить виключно від пристрою та облікових даних користувача, беручи до уваги менше місцезнаходження користувача в мережі — будь то місце розташування підприємства, домашня мережа, готель чи кав'ярня. Весь доступ до ресурсів підприємства повністю аутентифікований, повністю авторизований і повністю зашифрований на основі стану пристрою та облікових даних користувача. Ми можемо забезпечити дрібнозернистий доступ до різних частин ресурсів підприємства. В результаті всі співробітники можуть працювати успішно з будь-якої мережі та без необхідності традиційного VPN-з’єднання в привілейованій мережа. Досвід користувача між локальним і віддаленим доступом до ресурсів підприємства фактично ідентичні, за винятком потенційних відмінностей у затримці (ping).

На рисунку представлено схему представлення доступу до інформаційної системи, основа якої лежить у управлінні керованими пристроями, які прив’язуються до користувачів та дані яких зберігаються у базі даних. Таким принципом за допомогою видачі сертифікатів доступу та імплементації авторизації SSO надається доступ до приватних ресурсів. Це де-факто є стандартом у хмарному провайдері, який розглядається у даній роботі, та і взагалі більшості провайдерів на даний час при створенні робочого середовища для підприємства або при налаштуванні акаунту до стандартів безпеки. Звісно використання VPN у будь-якому вигляді підприємством може включатися і використовуватися для обмеження та більш пильного контролю над ресурсами, адже можна спостерігати за всіма запитами, збирати важливу інформацію по ресурсам. Просто можна також ізолювати дані по сервісам, і робити їх подальшу агрегацію. Це рішення має

розглядатись у рамках запитів та цілей на безпеку, які визначаються стратегією безпеки.

BeyondCorp: A New Approach to Enterprise Security



Додаток 3. Представлення отримання доступу до інформаційної системи

Порівняння важливих характеристик масштабованих БД різних хмарних провайдерів

Головним аспектом користування хмарою розкриває можливість доступу до швидкої аналітики, яка збирає у себе дуже великі потоки даних. Доцільним буде провести порівняльну характеристику цих компонентів з метою виявити і дослідити оптимальність цінових затрат та оцінити операційні затрати на потрібні інтеграції та управління БД.

Таблиця 1. Порівняльна характеристика Google BigQuery, AWS Redshift, Snowflake

	Google BigQuery	**AWS Redshift**	**Snowflake**

Розмір і планування	Планування не потрібно	Потрібно зазначати точні потужності	Потрібно визначати віртуальний розмір індивідуального кластеру
Гнучкість	При потребі збільшити кількість слотів	Зміна розміру кластеру викликає відключення кластеру	При потребі додати новий віртуальних кластер
Управління місцем	Автоматична оптимізація розташування даних по оптимальній ціні	Мануальна міграція і старіння	Потрібно надавати даним категорії швидких та дорогих або економічного зберігання
Оновлення схеми, скрипти, запити	Потрібні невеликі зміни для більшості сервісів	Зміни не потрібні	Потрібні невеликі зміни для більшості рішень
Управління доступом, безпека	Simplified IAM доступи, рольові доступи з автоматичним оновленням даних.	Потрібно задавати користувачів, налаштовувати безпеку. Оновлення даних	Спрощена система доступу по ролям і дозволам з автоматичним

		в окремому часовому вікні	оновленням даних
Стрімінг, поглинання, обробка даних	Вільне поглинання даних, їх трансформація і опції потоків. Високі мережеві показники	Комплексні оброки і стрімінг даних потребують сервісу Kinesis. Поглинання даних потребує алокації додаткових ресурсів	Окреме сховище рекомендується для обробки нових даних. Використання Snowpipe тільки у рамках сервісів реального часу
Інтеграція з іншими сервісами	Спроектований з підтримкою багатьох сервісів GCP: DataFlow, DataProc, CloudDB, PubSub і потребує мінімального налаштування	Потребує додаткового налаштування на підримку різними сервісами	Нова інтеграція потребує управління і операційного навантаження, розробки
Цінова політика	Flat Rate Pricing	Бронювання на рік, знижки при попередній оплаті	Підписка, кредити
Оптимальний розмір (одна ціна)	2000 фіксованих слотів	16 x dc2.8large	44 кредити (3 використанням у 12 год / день)

Cloud Storage	200 TB Google Cloud Storage	20 TB + 80 S3 Storage	100 TB
Підтримка потоків	Streaming Inserts	AWS Kinesis	Large Data Warehouse (8 кредитів на 8 годин на день)
Розвинена підтримка Terraform (наявність готових модулів)	Присутня	Присутня	Присутня

За визначених критеріїв в оптимізованій вартості, використання BigQuery для підприємств малого та середнього розміру є найоптимальнішим варіантом. При чому маючи в доступі безкоштовні ресурси, яких вистачає на покриття базових потреб або початку. По критерію легкості інтеграції в кодову базу інструменту Terraform, хочеться зазначити що проте всі системи мають модулі, найбільш простим дизайном інтеграції сервісів я визначив BigQuery. Незалежно від цього у сервісів відбуваються активні зміни, що зумовлені перебудовою старих недоліків і розробкою інтеграцій, тож ситуація може бути відмінною у найближчому майбутньому.

Визначення вимог та моделювання архітектури

В даній роботі розглядається підхід до моделювання систем підприємств середнього або малого розміру.

Основною вимогою до розширення в хмари є перенесення або саме інтеграція існуючого стеку технологій в створену робочу середу

підприємства. Має відбуватися збір інформації по каналам Pub/Sub, який пізніше ретранслюється у відокремлені BigQuery (Operations + Marketing). Ці дані аналізуються за допомогою різних сервісів, вимогою було визначено можливість простої інтеграції сторонніх сервісів аналітики. Використовуючи моделі штучного інтелекту, розроблюються трігери, які мають бути налаштовані на спрацювання під час виявленої підробки даних.

У лівому кутку схеми визначені вимоги до підтримання існуючого програмного забезпечення, що виконується створенням Artifact & Container Registry. Таким чином всі зовнішні залежності будуть збережені у локальному сховищі, а контроль над оновленням версій перейде у політику конвеєру. При цьому вимогою до проходження конвеєру є сканер Container Analysis, який перевіряє на можливі вади з реєстрів. Механізмом цього процесу є Cloud Build, через який будуються всі образи. Варто зазначити що процес інтеграції Cloud Build з стороннім сервісом контролю версій, таким як, у цьому випадку наприклад GitHub є можливим, але не входить в контекст цієї роботи.

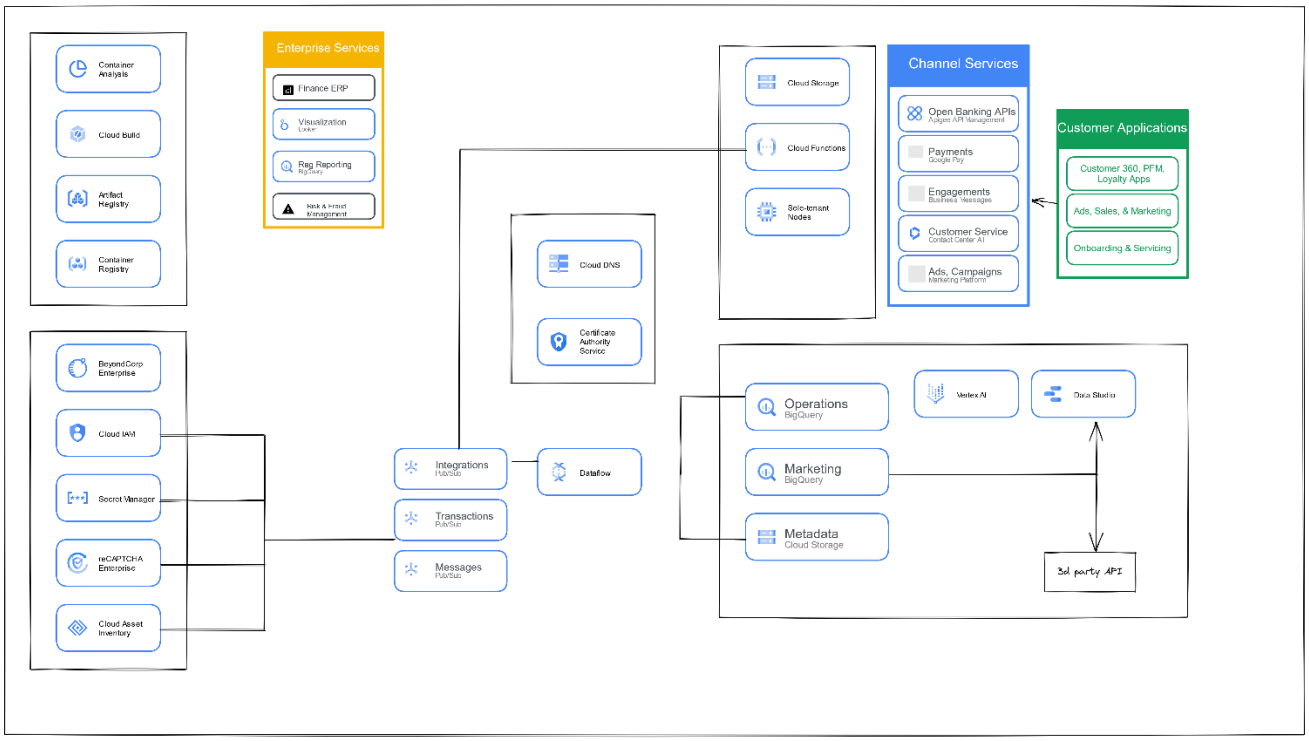
Для забезпечення потрібного контролю безпеки були використані BeyondCorp Enterprise, що дозволяє вести контроль і облік за активними девайсами у розглянутому Zero Trust. Використовуються Cloud IAM для створення потрібних ролей, сервісних аккаунтів. Cloud Asset Inventory використовується як канал по якому йде відстеження створення ресурсів. Secret Manager як сховище для секретів, які використовуються різними частинами інфраструктури, а також reCAPTCHA Enterprise для контролю над деякими ресурсами і захисту від ботів. Більшість сервісів безпеки має під’єднаний канал Pub/Sub, дані з якого потім потрапляють у сховище даних.

З точки зору бенефітів від підприємства використовуються Risk & Fraud Management, Finance ERP для додаткової безпеки і контролю над непередбачуваними витратами, а також Visualization і окремий Reg Reporting BigQuery для інвентаризації.

Для управління внутрішнім DNS та сертифікатів підприємства визначені вимогою Cloud DNS та Certificate Authority Service. Для всіх задач по відкриттю публічного доступу до ресурсу використовуються Cloud Load Balancing, для отримання приватного доступу Cloud NAT. Потрібно зазначити що дана схема припускає використання стандартного VPC. При використанні окремо створеної віртуальної хмари, потрібно зробити перенаправлення на потрібні адреси.

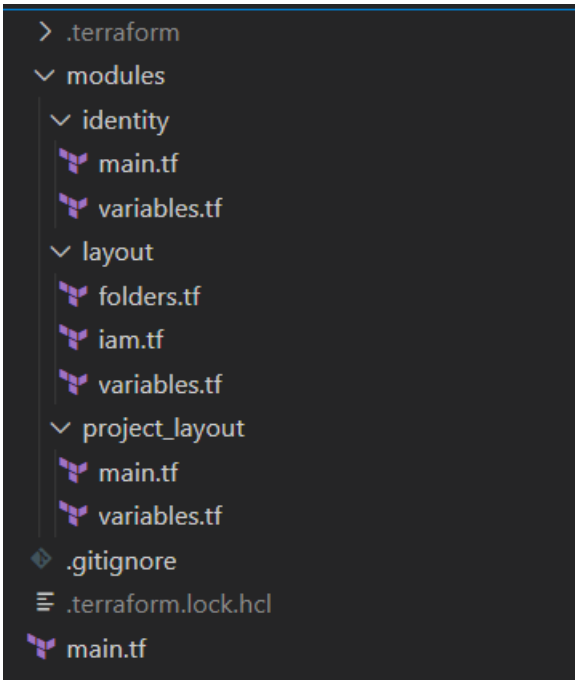
Визначаючи порядок створення архітектури в загальному потрібно орієнтуватися на такі ресурси, які статичні, стосуються надання допусків. Насправді, створювати ресурси виявилось краще за сценаріями застосування: у їх подальшому розвитку і збільшенню кодової бази, також важливо дотримуватися ізоляції ресурсів, по можливості краще створити окрему роль по користуванню, а не користуватися спільними ролями. Це забезпечить відкриту незалежність груп компонентів одне від одного і дозволить керувати компонентами. Розглянемо це на прикладі створення необхідних компонентів для BigQuery в Terraform. По-перше, потрібно створити ресурс датасету, в який потрібно передати дані, по-друге, потрібно створити роль якій надати їй потрібні можливості у перегляді, модифікації таблиці, дозволити використання деяких сервісів, створити канали Pub/Sub до який буде приєднаний вивід з їх каналу змін. Після цього налаштувати роботи Dataflow, у які налаштувати вивід каналів. Таким чином буде створена

частина архітектури відповідальна за моніторинг та аналітику системи. Після цього можна створити іншу незалежну частину проекту:



Додоток 5. Моделювання архітектури підприємства

Створення модулів для організації структури проекту



Додаток 6. Структура проекту

Для легшого майбутнього догляду за архітектурою можемо розбити її на модулі, які будуть відповідати тільки за свої частини інфраструктури. Маючи у функціоналі такі мета аргументи як depends_on, можна з легкістю досягти дуже часто необхідної пріоритетності у створенні чогось послідовно, визначивши залежності. Були також використані умовні оператори, шаблонування стрічок. Файловою структурою проекту є головний файл main.tf та папка modules, в якій розбито всі частини на логічні компоненти. В кожному модулі є файл з змінними які потрібні для конфігурації компонентів. Такими часто є кількість потрібного масштабування, назви кластерів, ідентифікатори вже готових компонентів, з якими присутня деяка залежність. Відповідно потрібні модулі потрібно включити в основному файлі main.tf, який включатиме в себе всі потрібні модулі. При цьому такого підходу вистачає при роботі з підприємствами малого і середнього розміру, проте при роботі з підприємствами великого масштабу рекомендується

застосування агрегації та винесенню рівня інфраструктури на окремий конвеєр.

Для перевірки планування ресурсів, використовується влучний функціонал terraform plan, при використанні якого можна зручно впевнитись на етапі розробки, які саме сервіси та з якими параметрами будуть створені. Використовується велика кількість output, тому що зручним є подальше використання змінних для моніторингу стану мережі.

Для контролю над станом системи використовується бекенд від GCP, в основі якого лежить зашифроване сховище з контролем версій Cloud Storage. Це дозволяє системі управляти і слідкувати за своїм станом, не будучи залежним від локального стану. Такий спосіб є звичайно кращим за локальний стан адже забезпечує додаткове шифрування і автоматичний контроль версій.

Створений шаблон тригеру, який є частою вимогою кожного сервісу, адже це один з механізмів захисту від неочікуваних проблем, перебоїв, визначення необхідності масштабування. Застосовані сервіси з класу підприємств: автоматична інвентаризація девайсів, а також ресурсів, з покращеним доглядом за фінансами і можливим моніторингом помилками.

Висновок

В роботі було розглянуто різні підходи в роботі з сучасними інструментами інфраструктури у вигляді коду. Проведено порівняння інструментів і способів роботи з інфраструктурою, розглянуто історичний перехід від управління деякої кількості машин до управління і створення віртуальних серверів, пошуків вирішення проблеми з дрифтами конфігурацій, циклічному витраченню часу на створення і управління, пошуку до зменшення шансу на виникнення помилки. При цьому інфраструктура від коду не є панацеєю сучасного програмування, вона активно розвивається і може мати певні обмеження, які краще розглядати наперед. Для налаштування системи CI/CD розглянуто застосування Terraform для створення ресурсів у процесі тестування, збірки образів за допомогою інструментів шаблонізаторів Docker або Packer. Проведено моделювання системи для малих та середніх підприємств. У критерії вибору застосування і існування компонентів в системі були широкість застосування доданого функціоналу, безпека застосунку від зовнішніх факторів, підтримка існуючого функціоналу, а також можливе розширення в майбутньому, оптимальність ціни, використання безкоштовних лімітів.

Завжди головною метою введення таких процесів є зменшення часу на доставку оновлень та тестуванню кожної частини окремо і вдосталь, так щоб бути впевненим в справності системи і створені трігери використовувати як сповіщення про несправність і подальші кроки автоматичного відновлення.

Список використаної літератури

1. Josh Armitage, Cloud Native Security Cookbook, pages 30-40, 150-170, 2022
2. Terraform: up & running, 2nd edition, O'Reilly Media, Inc., 2019
3. Henrique M. D. Santos, Cybersecurity: A practical Engineering approach, 2022 https://www.oreilly.com/library/view/cloud-native-security/9781098106294/
4. Teresa Susana Mendes Pereira and Henrique Santos. A Security Framework for Audit and Manage Information System Security. In 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, volume 3, pages 29 -32, Toronto, Ontario Canada, 2010. IEEE Computer Society.
5. Charles P. Pfliger, Shari Lawrence Pfliger and Jonathan Margulis. Security In the field of computer technology, Fifth Edition. Prentis Hall, 2015.
6. William Stollingsend and Lori Brown. Computer Security: Principles and Practice, 3rd Edition. Pearson Education, 2015.
7. ISO.ISO / IEC27005 Information technology — Security — Information security management, 2018.
8. ISO.ISO / IEC27002: 2013 Information technology - safety - Code of Practice for Information Security Control. https://www.iso.org/obp/ui/n#iso:std:iso-iec:27002:ed-2:v1:en .
9. Ardi Benusi, An Identity Management Survey on Cloud Computing, 2014 url: http://www.m-hikari.com/ijco/ijco2014/ijco1-4-2014/benusiIJCO1-4-2014.pdf
10. J. Peck, B. Beyer, C. Beske, M. Saltonstall, Migrating to BeyondCorp, 2017 https://www.usenix.org/system/files/login/articles/login_summer17_10_peck.pdf

11. Gary Gruver, Starting and Scaling DevOps in the Enterprise, 2016
12. Zahid Ahmed, S.Askari and S.Md.Firewall rule anomal Detection: A Survey. International Journal of Computational Intelligence & IoT, 2 (4), 2018.
13. Matt Bishop. Computer Security:Art and Science (2nd edition). Addison-Wesley Professional,2019.
14. Cornelius Diekmann,Lars Hupel,Julius Michaelis,Maximilian Haslbeck,and Georg Carle.Verifiediptablesfirewallanalysisand Verification. Journal of automatedreasoning, 61(1-4):191–242,2018.