УДК 004.272.3
А. М. Луцків, канд .техн. наук, доц., М. П.Голубовський
Тернопільський національний технічний університет ім. Івана Пулюя, Україна

КРИТЕРІЇ ВИБОРУ ІНСТРУМЕНТІВ IAC

A.M. Lutskiv, Ph.D., Assoc. Prof, M.P. Holubovskyi

IAC TOOLS SELECTION CRITERIA

Процес вибору необхідного інструменту для опису інфраструктури у вигляді коду є комплексною задачею. Усі популярні рішення дозволяють для описати інфраструктуру та з використанням будь-якого можна реалізувати поставлене завдання, але це не означає що вибір буде оптимальним і не приведе до непередбачуваних наслідків у майбутньому, не потребуватиме додаткових витрат часу на дослідження нюансів роботи та підводних каменів. До популярних інструментів належать Chef, Puppet, Ansible, SaltStack, CloudFormation, Deployment Manager, Terraform.

IaC рішення поділяються на рішення з відкритим вихідним кодом та пропрієтарні. З наведених вище продуктів CloudFormation і Deployment Manager є пропрієтарними інструментами Amazon та Google відповідно і можуть працювати тільки з їх хмарними середовищами. Решта – працюють з усіма популярними постачальниками хмарних послуг.

Першою вагомою відмінністю між описаними інструментами є те, що Chef, Puppet, Ansible і SaltStack є інструментами для керування конфігурацією (configuration management tools), і призначені для встановлення та налаштування програмного забезпечення на існуючому апаратному забезпеченні (віртуальних машинах, серверах). CloudFormation і Terraform – інструменти для створення (provisioning) інфраструктури. Вони розроблені для створення ресурсів машин та інших елементів інфраструктури: баз даних, сховищ інформації, налаштування мережі тощо, залишаючи роботу по налаштуванню для інших рішень. Ці категорії не є взаємовиключними: більшість інструментів керування конфігурацією можуть забезпечити певні етапи створення інфраструктури, так само як інструменти створення інфраструктури можуть виконувати налаштування. Проте, акценти закладені розробниками приводять то того, що конкретні інструменти краще підходять для виконання конкретних завдань.

Інструменти керування конфігурацією Chef, Puppet, Ansible та SaltStack працюють за парадигмою змінної інфраструктури. Наприклад, якщо вказати такому інструменту оновити версію програмного пакету, він виконає оновлення програмного забезпечення на існуючих, робочих серверах і зміни одразу будуть застосовані. У ході такого процесу не гарантується конкретний стан системи, інструмент відповідає тільки за виконання процедури, не враховуючи стан службових, тимчасових системних файлів та інших програм. Якщо з часом буде виконуватися більше і більше оновлень кожна машина буде містити унікальну історію змін. Це може призвести до такого феномену як зсув конфігурації (configuration drift), при якому кожен сервер стає трохи іншим за інші, що призводить до тонких помилок конфігурації, які важко діагностувати і майже неможливо відтворити. При використанні інструментів створення інфраструктури, для внесення змін у конфігурацію при кожному оновленні буде створюватися нова машина, а існуюча – видалятися. Такий підхід зменшує ризик проблем пов’язаних з конфігураційним зсувом, дозволяє точно знати яке програмне забезпечення запущене та дозволить за потреби легко розгорнути попередню версію у будь-який час.

Chef і Ansible використовують процедурний стиль, де код крок-за-кроком описується послідовність дій потрібну для досягнення бажаного стану конфігурації.

Для Terraform, CloudFormation, SaltStack і Puppet є характерним декларативний стиль, коли код описує бажаний кінцевий стан системи, а інструмент сам відповідальний для виконання дій, потрібних для його досягнення. Недоліком процедурного підходу є те, що стан такої системи може не повністю відповідати коду, щоб з'ясувати кінцевий стан системи потрібно знати порядок, з яким вносились та застосовувалися зміни. Повторне використання процедурного коду є обмеженим, тому що код написаний раніше може бути більше не придатним до використання, оскільки він був призначений для модифікації стану інфраструктури яка більше не існує. З декларативним підходом код завжди відповідає останньому стану інфраструктури. Таким чином, описаний підхід забезпечує самодокументованість розгорнутої інфраструктури й дає змогу здійснювати аналіз історії її змін у системі контролю версій.

Недоліком декларативного підходу є те, що деякі типи інфраструктурних змін важко описати у чисто декларативних термінах, уникнувши логічних операторів, циклів. Тому, зазвичай такі інструменти надають примітиви, зокрема: змінні, модулі, цикли, які дають змогу створювати функціональний, багаторазовий код навіть з використанням декларативної мови.

За замовчуванням для роботи Chef, Puppet, SaltStack потребують запуску мастер сервера для зберігання інформації про стан інфраструктури та розповсюдження змін. Для внесення зміни використовуються клієнт (CLI) який передає команди на мастер сервер, а він уже виконує потрібні дії. Перевагою є те, що мастер є єдиним, центральним місцем звідки можна переглянути та керувати станом інфраструктури. Він постійно працює у фоновому режимі і відстежує стан конфігурації. Таким чином, якщо відбувається внесення змін вручну, головний сервер може скасувати цю зміну, щоб запобігти дрейфу конфігурації на цільовій системі. Ansible, CloudFormation, Heat, Terraform за-замовчуванням не потребують використання мастер сервера. Такий функціонал уже вбудований у інструмент і не потребує додаткового налаштування. Ansible, наприклад, працює, під'єднуючись до кожної машини через SSH, тому не потребує додаткової інфраструктури для запуску чи налаштування інших механізмів автентифікації.

Chef, Puppet, SaltStack вимагають встановлення програмного забезпечення агента на кожну машину яка буде налаштовуватися. Для цього можуть використовуватися образи віртуальних машин, де вже буде потрібне ПЗ, скрипти які виконуються при створенні ресурсу, доступ до віддалених машин по SSH. Недоліком є необхідність моніторити стан, оновлювати та підтримувати роботу програми-агента на цільових машинах. Використання агентів впливає на налаштування безпеки, викликаючи необхідність відкривати необхідні мережеві порти для доступу чи інші механізми, що можуть бути вразливими до дій зловмисників. Ansible, CloudFormation, Heat і Terraform не вимагають встановлення агентів для своєї роботи. Точніше, агенти, зазвичай, вже встановлені. Для прикладу, постачальники хмарних послуг встановлюють свої агенти на кожен фізичний сервер. Користувач Terraform виконує команди, а агент постачальника виконує необхідні дії.

При виборі інструменту IaC також відіграє роль спільнота та зрілість технології. Ці фактори визначають, скільки людей бере участь у розвитку проекту, скільки доступно інтеграцій та розширень, наскільки легко знайти документацію, літературу та вирішення проблем. Для вибору варто врахувати кількість та активність розробки продукту, активність контриб'юторів щодо повідомлення про помилки та їх виправлення, повноту документації.

Література

1. Brikman Y. Terraform: Up & Running: Writing Infrastructure as Code / Yevgeniy Brikman. – Sebastopol, CA: O'Reilly Media, 2019. – 368 с.