Основи оркестрації контейнерів

Оркестрація контейнерів — це процес, який автоматизує управління життєвим циклом контейнерів. Він необхідний, коли кількість контейнерів у проєкті стає занадто великою для ручного управління.

Навіщо потрібна оркестрація?

Коли проєкт зростає, кількість контейнерів може сягнути сотень або тисяч. Це створює труднощі з керуванням, масштабуванням і підключенням, що легко може вийти з-під контролю. Оркестрація вирішує ці проблеми, автоматизуючи ключові завдання.

Основні переваги:

- **Автоматизація:** Автоматизує розгортання, масштабування, мережування та забезпечення доступності.
- **Спрощення складності:** Полегшує управління великими та динамічними середовищами.
- **Швидкість та ефективність:** Збільшує швидкість розгортання та ефективність використання ресурсів, інтегруючись у робочі процеси **CI/CD** та **DevOps**.
- **Висока доступність:** Забезпечує доступність додатків, переносячи контейнери на інші хости у разі збою.

Функції інструментів оркестрації

Інструменти для оркестрації контейнерів мають широкий спектр можливостей, які допомагають автоматизувати процес:

- **Управління життєвим циклом:** Автоматично розгортає нові контейнери та керує їх життєвим циклом на основі конфігураційних файлів.
- **Розподіл ресурсів:** Ефективно розподіляє системні ресурси, такі як **CPU** та пам'ять.
- **Безпека та мережа:** Забезпечує безпечне мережеве з'єднання між контейнерами.
- **Масштабування:** Автоматично масштабує контейнери відповідно до попиту та навантаження.
- **Оновлення та відкат:** Дозволяє виконувати плавні оновлення та швидкі відкати.
- **Перевірка працездатності:** Проводить регулярні перевірки працездатності та вживає необхідних заходів у разі збоїв.

Інструменти оркестрації

Існує кілька відомих інструментів для оркестрації контейнерів, серед яких:

- **Marathon:** Фреймворк для **Apache Mesos**, який автоматизує управління та моніторинг.
- **Nomad:** Гнучкий інструмент від **HashiCorp** для управління та планування кластерів.

- **Docker Swarm:** Інструмент, розроблений спеціально для **Docker**, що робить його популярним вибором у Docker-середовищах.
- **Kubernetes:** Фактичний стандарт для оркестрації контейнерів, розроблений Google. Він автоматизує широкий спектр завдань, включаючи балансування навантаження, забезпечення сховища та самовідновлення (здатність перезапускати або замінювати несправні контейнери).

Введення в Kubernetes

Що таке Kubernetes?

Kubernetes — це відкрита платформа для автоматизації **розгортання, масштабування та управління контейнерними додатками**. Розроблений компанією Google і підтримуваний **Cloud Native Computing Foundation (CNCF)**, він став фактичним стандартом для **оркестрації контейнерів**.

Чим не є Kubernetes?

Важливо розуміти, що Kubernetes не є універсальним інструментом. Він не надає:

- **Повний набір платформи як послуги (PaaS):** Це не готова платформа "все в одному".
- **Інструменти CI/CD:** Він не створює конвеєри безперервної інтеграції та розгортання.
- **Системи моніторингу та логування:** Користувачі повинні самі інтегрувати сторонні інструменти.
- **Вбудовані бази даних або проміжне ПЗ:** Ці сервіси слід налаштовувати окремо.

Ключові концепції Kubernetes

- **Поди (Pods):** Найменший об'єкт, який можна розгорнути. Він може містити один або кілька контейнерів, які працюють разом.
- **Сервіси (Services):** Дозволяють надавати доступ до додатків, що працюють у подах.
- **Сховище:** Підтримує як тимчасове, так і постійне сховище для даних додатків.
- **Планування та витіснення (Scheduling & Preemption):** Автоматично розподіляє поди між вузлами та може витісняти поди з нижчим пріоритетом, щоб звільнити місце для критично важливих робочих навантажень.
- **Конфігурація та секрети:** Дозволяє керувати конфігураціями та конфіденційною інформацією (наприклад, паролями) без необхідності перезбирати образи контейнерів.

Можливості Kubernetes

- **Автоматизоване розгортання та відкат:** Автоматизує випуск змін, а також може відкотити їх, якщо щось піде не так.
- **Оркестрація сховища:** Може автоматично підключати обрані системи зберігання, включаючи локальне сховище, мережеве сховище або хмарні сервіси.
- **Горизонтальне масштабування:** Масштабує робочі навантаження автоматично на основі метрик або за допомогою команд.
- **Самовідновлення:** Автоматично замінює несправні або не відповідаючі контейнери.
- **Балансування навантаження:** Розподіляє трафік для кращої продуктивності та високої доступності.
- **Управління секретами:** Безпечно керує конфіденційною інформацією.

Екосистема Kubernetes

Екосистема Kubernetes — це величезна і швидкозростаюча мережа продуктів, сервісів і провайдерів. Вона включає:

- **Публічні хмарні провайдери:** Google Cloud, AWS, IBM.
- **Фреймворки з відкритим кодом:** Red Hat, VMware, Docker.
- **Інструменти та провайдери:** JFrog, Datadog, Grafana.
- **Рішення для безпеки та моніторингу.**

Цей документ допоможе вам швидко ознайомитися з ключовими аспектами Kubernetes. Якщо у вас є додаткові питання або ви хочете заглибитися в якусь конкретну тему, я з радістю допоможу.

Архітектура Kubernetes

Кластер Kubernetes складається з одного або більше вузлів (nodes), де кожен вузол — це машина (фізична або віртуальна), на якій працюють контейнерні додатки. Кожен кластер має **площину управління** та одну або більше **робочих площин**.

Площина управління (Control Plane)

Площина управління є "мозком" кластера. Вона відповідає за підтримку бажаного стану кластера, приймає рішення та реагує на події.

Ключові компоненти:

- **API-сервер (API server):** Це фронтенд для площини управління, який надає Kubernetes API. Вся комунікація в кластері відбувається через цей сервер.
- **etcd:** Розподілене сховище ключ-значення з високою доступністю, яке містить усі дані кластера. Воно зберігає бажаний стан кластера, і система прагне привести його фактичний стан у відповідність до бажаного.
- **Планувальник (Scheduler):** Призначає нові поди (pods) вузлам. Він визначає, де

мають працювати ваші робочі навантаження, на основі доступних ресурсів та конфігурації.
- **Менеджер контролерів (Controller Manager):** Запускає процеси, які відстежують стан кластера та забезпечують відповідність між фактичним та бажаним станом.
- **Менеджер хмарних контролерів (Cloud Controller Manager):** Керує контролерами, які взаємодіють з API хмарного провайдера, роблячи Kubernetes більш незалежним від конкретного хмарного середовища.

Робоча площина (Worker Plane)

Робоча площина складається з вузлів, на яких виконуються ваші додатки. Ці вузли управляються площиною управління.

Ключові компоненти:

- **Kubelet:** Найважливіший компонент вузла. Він спілкується з API-сервером, щоб отримувати специфікації подів і забезпечувати їх виконання. Він також звітує площині управління про стан подів.
- **Середовище виконання контейнерів (Container Runtime):** Відповідає за завантаження образів та запуск контейнерів. Kubernetes підтримує різні середовища, включаючи **Docker**, **Podman** та **CRI-O**.
- **Kube-proxy:** Мережевий проксі, який працює на кожному вузлі. Він підтримує мережеві правила, що дозволяють комунікацію з подами та всередині кластера.

Об'єкти Kubernetes

Об'єкти Kubernetes — це **постійні сутності**, які представляють стан вашого кластера. Кожен об'єкт має два основні поля: `spec` **(специфікація)**, яке визначає бажаний стан, і `status` **(статус)**, яке відображає поточний стан.

Основні поняття

- **Мітки (Labels):** Це пари "ключ-значення", що використовуються для ідентифікації та групування об'єктів. За допомогою селекторів міток можна знаходити набори об'єктів, які мають однакові мітки.
- **Простори імен (Namespaces):** Забезпечують механізм для ізоляції груп ресурсів в одному кластері. Вони ідеально підходять для великих команд або проектів, що спільно використовують один кластер, допомагаючи уникнути конфліктів імен.
- **Поди (Pods):** Найпростіша одиниця в Kubernetes. Під представляє один екземпляр додатка або процес, що працює в кластері, і зазвичай містить один або кілька контейнерів.

Набори реплік (Replica Sets)

Набір реплік створює і керує певною кількістю ідентичних копій подів, забезпечуючи їх горизонтальне масштабування. Він автоматично створює або видаляє поди, щоб підтримувати бажану кількість реплік. Однак, створювати їх безпосередньо не рекомендується.

Розгортання (Deployments)

Розгортання — це об'єкт вищого рівня, який керує **наборами реплік** і подами. Воно надає додаткові можливості, яких немає в наборах реплік, зокрема **плавні оновлення (rolling updates)**.

Плавне оновлення — це ключова особливість розгортань. Під час оновлення воно поступово масштабує нову версію додатка, одночасно зменшуючи стару версію до нуля, забезпечуючи безперебійну роботу.

Таким чином, розгортання є кращим вибором для керування вашими додатками, оскільки вони забезпечують більший контроль і гнучкість, ніж набори реплік.

Сервіси (Services)

Сервіс є логічною абстракцією для набору подів. Він вирішує проблему мінливості подів, надаючи єдину, постійну IP-адресу або DNS-ім'я. Це дозволяє додаткам всередині кластера легко знаходити та спілкуватися з подами, незважаючи на те, що IP-адреси самих подів можуть змінюватися.

Існує чотири основних типи сервісів:

- **ClusterIP:** Тип за замовчуванням. Сервіс отримує внутрішню IP-адресу кластера, доступну лише всередині нього.
- **NodePort:** Розширює ClusterIP, виставляючи сервіс на статичному порту на кожному вузлі. Це дозволяє отримувати доступ до сервісу ззовні кластера.
- **LoadBalancer:** Розширює NodePort, автоматично створюючи зовнішній балансувальник навантаження, який розподіляє зовнішній трафік на вузли кластера. Цей тип зазвичай використовується для виставлення додатків в Інтернет.
- **ExternalName:** Сервіс, який не має селектора і мапується на DNS-ім'я. Це дозволяє легко звертатися до зовнішніх ресурсів (наприклад, до зовнішньої бази даних) як до внутрішнього сервісу.

Ingress

Ingress — це об'єкт API, який надає правила маршрутизації для управління доступом зовнішніх користувачів до декількох сервісів у кластері. Він, як правило, використовується в продакшн-середовищах для відкриття додатків в Інтернет через стандартні порти 80 (HTTP) і 443 (HTTPS), замінюючи дорогі зовнішні балансувальники

навантаження для кожного сервісу.

Інші об'єкти

- **DaemonSet:** Забезпечує, щоб копія поду завжди була запущена на кожному вузлі в кластері. Це ідеально підходить для завдань, які повинні працювати на кожному вузлі, наприклад, для збору логів або моніторингу.
- **StatefulSet:** Управляє додатками, що зберігають стан, забезпечуючи унікальну ідентичність і постійне сховище для кожного поду. На відміну від Deployment, він підтримує стійку ідентифікацію для кожного поду, що є критично важливим для баз даних або інших stateful-додатків.
- **Job:** Створює поди та відстежує їхнє завершення. Якщо под завершується з помилкою, Job буде повторно запускати його, доки завдання не буде успішно виконано. Ідеально підходить для одноразових або запланованих завдань. CronJob використовується для регулярного запуску Jobs.

Використання Kubectl

Kubectl — це інтерфейс командного рядка Kubernetes (CLI), який дозволяє користувачам розгортати додатки, перевіряти та керувати ресурсами кластера, переглядати логи та багато іншого.

Загальна структура команд **Kubectl** виглядає так:

`kubectl [команда] [тип] [ім'я] [прапори]`

- **Команда:** Операція, яку ви хочете виконати (`create`, `get`, `apply`, `delete`).
- **Тип:** Тип ресурсу (`pod`, `deployment`, `replica set`).
- **Ім'я:** Ім'я ресурсу (якщо застосовується).
- **Прапори:** Додаткові опції, які змінюють поведінку команди.

Типи команд Kubectl

Існує три основних типи команд:

1. Імперативні команди (Imperative Commands)

Ці команди дозволяють створювати, оновлювати та видаляти об'єкти безпосередньо, вказуючи всі параметри в командному рядку.

Переваги:

- Прості та швидкі у використанні.
- Ідеально підходять для розробки та тестування.

Недоліки:

- Немає **аудиторського сліду**, що ускладнює відстеження змін.
- Немає гнучкості.

2. Імперативна конфігурація об'єктів (Imperative Object Configuration)

Цей підхід використовує файли конфігурації у форматі **YAML** або **JSON**, де ви вказуєте повну специфікацію об'єкта. Використовуйте `kubectl create -f` для створення ресурсів із файлу.

Переваги:

- Забезпечує **аудиторський слід**, оскільки файли можуть зберігатися в системі контролю версій (наприклад, **Git**).
- Результати розгортання є ідентичними в різних середовищах.

Недоліки:

- Потрібно вручну вказувати всі операції (`create`, `update`, `delete`).
- Вимагає розуміння схеми об'єктів.

3. Декларативна конфігурація об'єктів (Declarative Object Configuration)

Це найбільш рекомендований метод для продакшн-систем. Він також використовує файли, але ви просто вказуєте бажаний стан об'єктів. Kubernetes автоматично визначає, які операції потрібно виконати, щоб привести поточний стан у відповідність до бажаного.

Переваги:

- Автоматизований процес.
- Не вимагає ручного вказування операцій.
- Єдине джерело істини для конфігурації.

Приклад:

Команда kubectl apply -f my-directory/ застосує конфігурацію до всіх файлів у вказаній директорії, автоматично створюючи або оновлюючи ресурси.

Поширені команди Kubectl

- `kubectl get`: Переглянути ресурси (наприклад, `kubectl get pods -A` щоб побачити поди в усіх просторах імен).
- `kubectl apply`: Створити або оновити ресурси з файлу.
- `kubectl delete`: Видалити ресурс.
- `kubectl scale`: Змінити кількість реплік.

ЛАБОРАТОРНА ПО KUBECTL

Перша лаба

Друга лаба