Vademecum DevOps

Opracowanie i tłumaczenie: **Kacper Rychel**

Uwaga od autora streszczenia:

Jeśli nie posiadasz praw do oryginałów podanych poniżej, ów streszczenia również nie powinieneś posiać.

Część: Docker

Tłumaczenie i streszczenie **dokumentacji Docker**.

Część: Kubernetes, Helm

Strzeszeniem książki:
„**Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud**”
ISBN: 978-83-283-6928-3
autorstwa **John Arundel** i **Justin Domingus**; tłumaczenia **Łukasza Wójcika**

Część: Prometheus

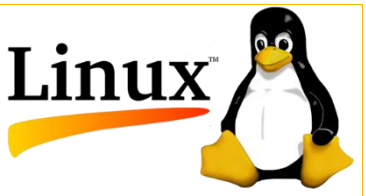
Tłumaczenie i streszczenie **wpisu z bloga** autorstwa **Ryana Harrisona.**

Część: OpenShift

Tłumaczenie **wpisu Gliada Maayana.**

Część: Linux

Opracowanie i tłumaczenie:



- wideokursu **Linux Directories Explained in 100 Seconds** autorstwa **Fireship,**
- serii wideokursów **Linux Essentials For Hacker** autorstwa **HackerSploit,**
- **Linuxa man pages** ze strony **linux.die.net**
- postu **Linux Capabilities: Hardening Linux binaries by removing setuid** autorstwa **Michaela Boelena**

Część: Maven

Tłumaczenie i streszczenie z **dokumentacji Maven.**

Część: OpenShift

Opracowanie i tłumaczenie:

- wideokursu **Understanding AWS Core Services** autorstwa **Davida Tuckera,**
- wideokursu **Deploying Basic Infrastructure Services on AWS** autorstwa **Davida Tuckera**
- **dokumentacji AWS.**

Spis treści

Rozdział 2. Docker

Czym jest Docker?

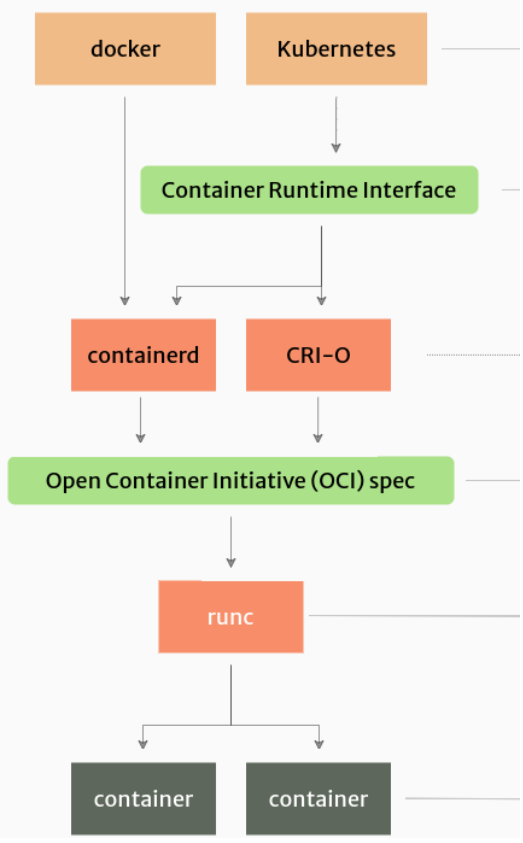
Docker to projekt typu "open source" służący do automatyzowania wdrażania aplikacji jako przenośne, samoobsługowe kontenery, które można uruchamiać w chmurze lub lokalnie.

Docker (*https://docs.docker.com/*) to kilka różnych, ale powiązanych rzeczy:

1. **format obrazu kontenera**,
2. **biblioteka środowiska *wykonawczego kontenera (container runtime)***, która zarządza cyklem życia kontenerów,
3. **narzędzie wiersza polecenia** do pakowania i uruchamiania kontenerów
4. **interfejs API** do zarządzania kontenerami.

Docker to również firma , która wspiera i zmienia tę technologię, pracując w współpracy z dostawcami chmury, Linux i Windows, w tym z firmą Microsoft.

Docker a Kubernetes – standardy konteneryzacji



To są narzędzia używane przez użytkowników do uruchamiania kontenerów.

Kubernetes API – określają sposób, w jaki Kubernetes komunikuje się z *container runtime*.

Ponieważ jest to ustandaryzowane w specyfikacji, możesz wybrać implementację CRI (lub napisać własną).

Publikuje specyfikacje dla obrazów i kontenerów. Dostarcza również specyfikację *runc*. Jest to narzędzie do tworzenia i uruchamiania procesów kontenera.

Narzędzie zgodne z OCI do tworzenia i uruchamiania procesów kontenera.

Nareszcie. Skonteneryzowany proces.

Container

Kontener to znormalizowany pakiet zawierający oprogramowanie wraz z zależnościami, konfiguracją, danymi itd., czyli wszystko, czego potrzebuje do uruchomienia.

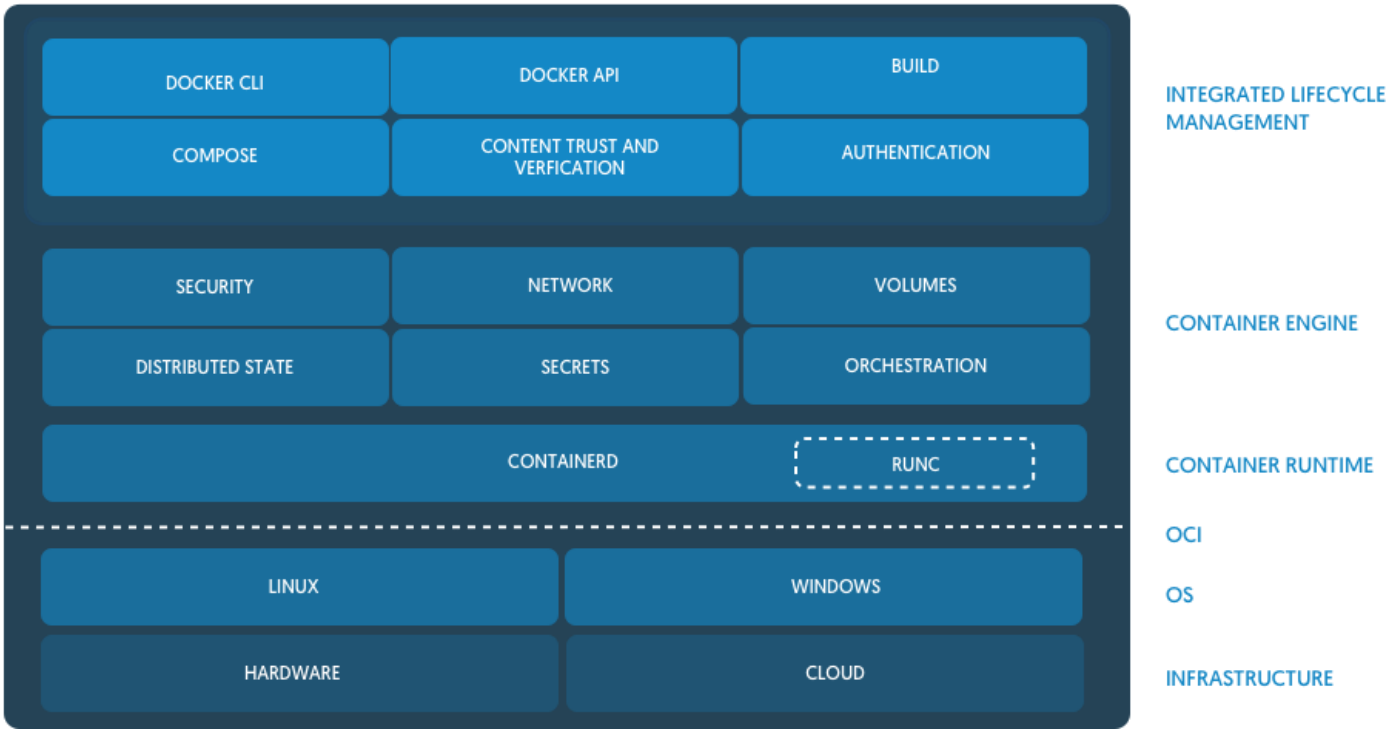
Z punktu widzenia systemu operacyjnego, kontener reprezentuje izolowany proces (lub grupę procesów), który istnieje we własnej przestrzeni nazw. Procesy wewnątrz kontenera nie widzą procesów poza nim i vice versa. Kontener nie może uzyskać dostępu do zasobów należących do innego kontenera ani przetwarzać zasobów poza kontenerem.

Kontener ma również punkt wejścia (ang. entrypoint), czyli polecenie uruchamiane podczas startu kontenera. Zwykle powoduje to utworzenie jednego procesu do uruchomienia polecenia, chociaż niektóre aplikacje często uruchamiają kilka podprocesów, które działają jako pomocnicy lub pracownicy. Aby uruchomić wiele oddzielnych procesów w kontenerze, musisz napisać skrypt, który będzie działał jako punkt wejścia i uruchomi pożądane procesy.

Docker image

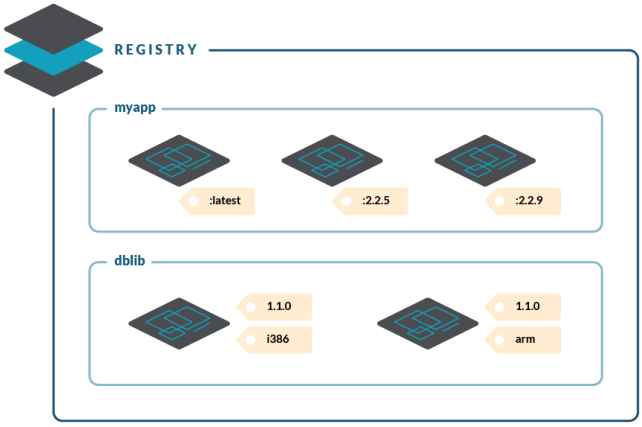
Obrazy są zasadniczo warstwami systemów plików, zwykle opartymi na obrazie bazowym pod warstwą zapisywalną i budowanymi z warstwami różnic w stosunku do obrazu bazowego. To minimalizuje ślad obrazu i umożliwia współdzielenie rozwoju.

Architektura Dockera



Słownik terminów Dockera

registry, repository, tag,



Rejestr (***registry***) jest usługą, której używamy do przechowywania i pobierania (push/pull) naszych obrazów kontenerów.

Obrazy wewnątrz rejestru są pogrupowane w **repozytoria** (***repository***),
a repozytoria mogą mieć dodatkowe podgrupy, takie jak organizacje, projekty itp. Repozytorium zawiera wiele **znaczników** (***tags***).

```
$ docker run my-registry.com/some-/-organization/repository:tag
```

Kiedy tworzymy kontener, pobieramy obraz z rejestru, wybierając konkretny tag z repozytorium. Tag ten odpowiada obrazowi i uruchamia izolowane wykonanie procesu w systemie plików zdefiniowanym przez ten obraz.

base image

Obraz bazowy nie ma określonego obrazu nadrzędnego w swoim pliku Dockerfile. Jest on tworzony przy użyciu pliku Dockerfile z dyrektywą FROM scratch.

build

Proces budowania obrazów Dockera przy użyciu pliku Dockerfile. Do budowy używa się pliku **Dockerfile** i "kontekstu" (***context***). Kontekst jest zbiorem plików w katalogu, w którym budowany jest obraz.

cgroups

Funkcjonalność jądra Linuksa, która ogranicza, rozlicza i izoluje wykorzystanie zasobów (CPU, pamięć, I/O dysku, sieć, itp.) przez zbiór procesów. Docker opiera się na *cgroups* do kontroli i izolowania limitów zasobów.

compose (docker-compose)

To narzędzie do definiowania i uruchamiania złożonych aplikacji z wieloma kontenerami w jednym pliku, którą uruchamiasz jednym poleceniem.

copy-on-write

Docker wykorzystuje technikę ***copy-on-write*** oraz ***union file system*** dla obrazów i kontenerów w celu optymalizacji zasobów i zwiększenia wydajności. Wiele kontenerów współdzieli tę samą instancję i każda z nich dokonuje zmian tylko w swojej unikalnej warstwie, która jest usuwana, gdy kontener jest usuwany.

docker-machine

Maszyna pozwala na tworzenie hostów Dockera lokalnie, u dostawców usług w chmurze i we własnym centrum danych. Tworzy serwery, instaluje na nich Dockera, a następnie konfiguruje klienta Dockera, aby mógł się z nimi łączyć.

entypoint / cmd

Jeśli chcesz, aby Twój plik Dockerfile mógł być uruchamiany bez podawania dodatkowych argumentów do polecenia docker run, musisz podać ENTRYPOINT, CMD lub oba.

CMD jest instrukcją, która jest uruchamiana domyślnie podczas uruchamiania kontenera, ale może być łatwo nadpisana, jeśli kontener jest uruchamiany za pomocą komendy.

ENTRYPOINT jest ustawiany na pojedynczą komendę. Nawet jeśli nie określisz ENTRYPOINT, możesz odziedziczyć go z obrazu bazowego. Nie jest pomijany, nawet jeśli kontener jest uruchamiany za pomocą komendy. Aby nadpisać ENTRYPOINT można użyć opcji --entrypoint.

Instrukcje są sprowadzane do uruchomienia komendy zgodnie z wzorcem:
`$ <ENTYPOINT> <CMD>`
dlatego **CMD** może być używane do przekazywania domyślnych argumentów dla komendy w **ENTRYPOINT.**

image

Jest uporządkowaną **kolekcją zmian głównego systemu plików** i odpowiadających im **parametrów wykonania** w celu użycia ich w czasie uruchamiania kontenera. Obraz zazwyczaj zawiera ***union file system*** ułożone jeden na drugim. Obraz nie posiada stanu i nigdy się nie zmienia.

manifest

Manifest dostarcza podstawowe dostępne pola do pracy z formatem obrazu w rejestrze:

- **name** *string* – nazwa repozytorium obrazu,
- **tag** *string* – tag obrazu,
- **architecture** *string* – architektura hosta, na którym ten obraz ma być uruchomiony (informacyjnie),
- **fsLayers** *array* – lista deskryptorów warstw (***digest***) pod postacią skrótów sha256,
- **history** *array* – lista warstw obrazu oraz warstwy warstw nadrzędnych,
- **schemaVersion** *int* – wersja schematu manifestu, do którego odnosi się ten obraz,
- **signatures** *array* – lista podpisów generowanych przez ***libtrust***.

layer

Warstwa jest modyfikacją obrazu, reprezentowana przez instrukcję w pliku Dockerfile. Warstwy są kolejno nakładane na obraz bazowy, aby utworzyć obraz końcowy. Kiedy obraz jest aktualizowany lub przebudowywany, tylko warstwy, które się zmieniają, muszą być aktualizowane, a niezmienione warstwy są buforowane lokalnie.

namespace

Przestrzeń nazw Linuksa jest funkcjonalnością jądra Linuksa, która izoluje i wirtualizuje zasoby systemowe. Procesy mogą wchodzić w interakcję tylko z zasobami lub procesami, które są częścią tej samej przestrzeni nazw. Przestrzenie nazw istnieją dla każdego typu zasobów, w tym:

- net (sieci),
- mnt (storage),
- pid (procesy),
- uts (kontrola nazw hostów),
- user (mapowanie UID).

overlay network driver

Sterownik sieciowy Overlay zapewnia łączność sieciową multi-host dla kontenerów w klastrze.

overlay storage driver

OverlayFS jest usługą systemu plików dla Linuksa, która implementuje montowanie unii dla innych systemów plików. Jest obsługiwany przez demona Docker jako sterownik pamięci masowej.

persistent storage

Pamięć trwała lub wolumenowa pozwala użytkownikowi na dodanie trwałej warstwy do systemu plików kontenera. Ta trwała warstwa może znajdować się na hoście kontenera lub na urządzeniu zewnętrznym. Cykl życia tej trwałej warstwy nie jest związany z cyklem życia kontenera, co pozwala użytkownikowi na zachowanie stanu.

tag

Tag to etykieta przypisana do obrazu Dockera w repozytorium. Tagi są sposobem na odróżnienie różnych obrazów w repozytorium od siebie jako wskaźnik na plik manifestu. Jest zmienny (*mutable*) w odróżnieniu od blob sha256 (*immutable*).

union file system

W komputerowych systemach operacyjnych, montowanie unii jest sposobem łączenia wielu katalogów w jeden, który wydaje się zawierać ich połączoną zawartość.

volumen

Wolumen jest specjalnie zaprojektowanym katalogiem w ramach jednego lub więcej kontenerów, który omija system plików Union. Wolumeny są zaprojektowane do przechowywania danych, niezależnie od cyklu życia kontenera.

Istnieją trzy rodzaje woluminów:

- **host** volumen – żyje na systemie plików hosta Dockera i może być dostępny z wnętrza kontenera;
- **named** volumen – jest wolumenem, który Docker zarządza, gdzie jest tworzony na dysku, ale otrzymuje nazwę;
- **anonymous** volume – jest podobny do nazwanego wolumenu, jednak może to być trudne, aby odnieść się do tego samego wolumenu w czasie. To Docker zarządza, gdzie pliki są przechowywane.

Budowanie obrazu z Dockerfile – dobre praktyki

Twórz efemeryczne kontenery

Obraz zdefiniowany przez twój plik Docker powinien generować kontenery, które mogą być zatrzymane i zniszczone, a następnie odbudowane i zastąpione z absolutnie minimalną konfiguracją.

Przesyłaj (*pipe*) Dockerfile przez *stdin*

Docker posiada możliwość budowania obrazów poprzez przesyłanie pliku Dockerfile przez *stdin*. Może to być przydatne do wykonania jednorazowych buildów bez zapisywania Dockerfile na dysk lub gdy Dockerfile jest generowany i nie powinien być zachowywany.

```
$ echo -e 'FROM busybox\nRUN echo "hello world"' | docker build -
```

Gdy myślnik - zajmuje miejsce PATH, instruuje Dockera, aby odczytał kontekst budowania (który zawiera tylko Dockerfile) ze *stdin*, zamiast z katalogu. Pominięcie kontekstu budowania może być przydatne w sytuacjach, gdy Dockerfile nie wymaga kopiowania plików do obrazu i poprawia szybkość budowania, ponieważ żadne pliki nie są wysyłane do demona.

Linting – czyli czysty Dockerfile

Przykładowym linterem do Dockerfile jest Hadolint.

```
$ docker run --rm -i hadolint/hadolint < Dockerfile
```

Jeżeli Hadolint nie wykryje żadnych błędów w Twoim Dockerfile, nie zobaczymy żadnych błędów i zostanie zwrócony exit code 0.

Dostępna jest też wersja online.

Minimalizuj liczbę warstw

Tworzenie kolejnych warstw

Instrukcje, które tworzą kolejne warstwy:

- **FROM** – tworzy warstwę z obrazu rodzica,
- **COPY** / **ADD** – dodaje pliki z bieżącego katalogu klienta Docker,
- **RUN** – buduje twoją aplikację za pomocą make.
- **CMD** – określa jakie polecenie ma zostać uruchomione wewnątrz kontenera.

Inne instrukcje tworzą puste obrazy pośrednie i nie zwiększają rozmiaru kompilacji, lecz ich kolejność ma wpływ na budowanie obrazu z użyciem *cache'u* Dockera.

Kiedy uruchamiasz obraz i generujesz kontener, dodajesz nową zapisywalną warstwę ("warstwa kontenera") na wierzchu warstw bazowych. Wszystkie zmiany dokonywane w działającym kontenerze, takie jak zapisywanie nowych plików, modyfikowanie istniejących plików i usuwanie plików, są zapisywane w tej zapisywalnej warstwie kontenera.

Aby sprawdzić jakie warstwy z jakimi poleceniami składają się na dany obraz, użyj polecenia **docker image history**.

Wyodrębnienie wspólnych warstw do obrazu bazowego

Docker musi załadować wspólne warstwy tylko raz i są one buforowane. Oznacza to, że twoje pochodne obrazy wykorzystują pamięć na hoście Dockera bardziej efektywnie i ładują się szybciej.

Debugowanie lekkiego obrazu produkcyjnego

Aby utrzymać obraz produkcyjny lekki, ale umożliwić debugowanie, rozważ użycie obrazu produkcyjnego jako obrazu bazowego dla obrazu debugowania.

Minimalizuj rozmiar obrazu

Korzystaj z cache build

Gdy każda instrukcja jest sprawdzana, Docker szuka istniejącego obrazu w swojej pamięci podręcznej, który może ponownie wykorzystać, zamiast tworzyć nowy (duplikat) obrazu.

Uporządkuj swoje kroki od najmniej do najczęściej zmieniających się kroków, aby zoptymalizować *cache'owanie*.

Jeśli z jakiegoś powodu nie chcesz w ogóle korzystać z pamięci podręcznej, możesz użyć opcji --no-cache=true w poleceniu Docker build. Jeśli jednak pozwolisz korzystać z pamięci podręcznej, ważne jest, aby zrozumieć, podstawowe zasady, którymi kieruje się Docker:

- Zaczynając od obrazu rodzica, który jest już w pamięci podręcznej, następna instrukcja jest porównywana ze wszystkimi obrazami potomnymi pochodzącymi z tego obrazu bazowego pod kątem użycia dokładnie tej samej instrukcji.
- Dla instrukcji ADD i COPY, badana jest suma kontrolna każdego pliku.
- Dla reszty instrukcji, W tym przypadku tylko sam łańcuch polecenia jest używany do znalezienia dopasowania.

Buduj źródła w spójnym środowisku

Zamiast budować kod źródłowy gdzieś poza obrazem i kopiowania tylko jego wyniku do obrazu, zbuduj kod źródłowy w samym obrazie, aby zapewnić spójność między środowiskiem, na którym został wybudowany, i środowiskiem, na którym ma być uruchamiany. Może to być ten sam obraz.

Pobierz zależności w osobnym kroku

Myśląc o *cache'u* budowania, możemy zdecydować, że pobieranie zależności jest oddzielną warstwą, która musi zależeć tylko od zmian w pom.xml, a nie od kodu źródłowego.

Krok RUN pomiędzy dwoma krokami COPY mówi **Mavenowi**, aby pobierał tylko zależności:

```
COPY pom.xml
RUN mvn -e -B dependency:resolve
COPY src ./src
RUN mvn -e -B package
```

Flaga -e służy do pokazywania błędów, a -B do uruchamiania w trybie nie-interaktywnym.

Nie instaluj zbędnych pakietów

Aby zredukować złożoność, zależności, rozmiary plików i czas kompilacji, unikaj instalowania niepotrzebnych pakietów. Narzędzia do debugowania można zainstalować później w razie potrzeby.

Niektóre menedżery pakietów, takie jak **apt**, automatycznie instalują pakiety, które są zalecane przez pakiet określony przez użytkownika, niepotrzebnie zwiększając ślad. apt install posiada flagę -no-install-recommends, która zapewnia, że zależności, które nie były w rzeczywistości potrzebne, nie są instalowane. Jeśli są one potrzebne, dodaj je jawnie.

Menedżery pakietów utrzymują swój własny cache, który może wylądować w obrazie. Jednym ze sposobów radzenia sobie z tym jest usunięcie pamięci podręcznej w tej samej instrukcji RUN, która zainstalowała pakiety. Usunięcie go w innej instrukcji RUN nie zmniejszyłoby rozmiaru obrazu.

Używaj budowania wieloetapowego – *multi-stage builds*

Wieloetapowe kompilacje pozwalają drastycznie zmniejszyć rozmiar końcowego obrazu, bez konieczności walki o zmniejszenie liczby warstw pośrednich i plików.

Ponieważ obraz jest budowany podczas ostatniego etapu procesu kompilacji, możesz zminimalizować warstwy obrazu poprzez wykorzystanie pamięci podręcznej kompilacji.

Na przykład, jeśli twój build zawiera kilka warstw, możesz uporządkować je od rzadziej zmienianych do częściej zmienianych:

1. zainstaluj narzędzia potrzebne do zbudowania aplikacji,
2. zainstaluj lub zaktualizuj zależności bibliotek,
3. wygeneruj swoją aplikację z ponownym użyciem dyrektywy FROM lub jeśli to możliwe FROM scratch.

```
FROM maven:3.6-jdk-8-alpline AS builder
RUN mvn -e -B package
…
FROM openjdk:8-jre-alpine
COPY –from=builder /app.target.app.jar /
```

Aby podejrzeć jakie warstwy i przez co zostały utworzone, skorzystaj z komendy:

```
docker image history getting-started
```

Utrzymanie obrazów

Sortuj argumenty wieloliniowe

Jeśli to tylko możliwe, ułatw późniejsze zmiany sortując alfanumerycznie argumenty wieloliniowe. Pomoże to uniknąć duplikacji argymentów i sprawi, że lista będzie łatwiejsza do aktualizacji.

Rozdziel aplikacje

Każdy kontener powinien mieć tylko jeden wątek. Rozdzielenie aplikacji na wiele kontenerów ułatwia skalowanie horyzontalne i ponowne wykorzystanie kontenerów. Ograniczenie każdego kontenera do jednego procesu jest dobrą zasadą, ale nie jest to sztywna reguła. Niektóre programy same tworzą dodatkowe procesy. Jeśli kontenery zależą od siebie nawzajem mogą się komunikować.

Bezpieczeństwo obrazów

Używaj oficjalnych obrazów, kiedy tylko to możliwe

Oficjalne obrazy mogą zaoszczędzić wiele czasu poświęconego na utrzymanie, ponieważ wszystkie kroki instalacji są już wykonane i zastosowane są najlepsze praktyki. Do tego są znacznie bezpieczniejsze i w większym stopniu wolne od podatności.

Jeśli już musisz użyć niestandardowego obrazu, sprawdź źródło obrazu i plik Dockerfile, a następnie zbuduj własny obraz bazowy. Nie ma gwarancji, że obraz opublikowany w publicznym rejestrze jest naprawdę zbudowany z danego pliku Dockerfile. Nie ma też pewności, że jest on aktualizowany.

Czasami oficjalne obrazy mogą nie być lepszym wyborem, jeśli chodzi o bezpieczeństwo i minimalizm.
Np. porównując oficjalny obraz *node* z obrazem *bitnami/node*, ten drugi oferuje dostosowane wersje na bazie dystrybucji *minideb*. Są one często aktualizowane z najnowszymi poprawkami błędów, podpisane ***Docker Content Trust*** oraz przechodzą skanowanie bezpieczeństwa w celu śledzenia znanych luk.

Skanowanie pod kątem bezpieczeństwa

Dobrą praktyką bezpieczeństwa jest zastosowanie paradygmatu "*shift left security*" poprzez bezpośrednie skanowanie obrazów, zaraz po ich zbudowaniu, w potokach CI przed wypchnięciem do rejestru. Dobrą praktyką jest przeskanowanie go pod kątem luk w zabezpieczeniach za pomocą polecenia **docker scan**. Docker współpracuje z firmą Snyk w celu zapewnienia usługi skanowania podatności. Powinieneś wykonywać skanowanie na różnych etapach cyklu życia obrazu. Dotyczy to również komputera dewelopera.

Zeskanowany obraz może być "bezpieczny" teraz. Ale w miarę starzenia się i odkrywania nowych podatności, może stać się niebezpieczny. Należy okresowo przeprowadzać ponowną ocenę pod kątem nowych podatności.

Pliki wykonywalne tylko do odczytu z własnością *roota*

Każdy plik wykonywalny jest własnością użytkownika *root*, nawet jeśli jest wykonywany przez użytkownika nie będącego *rootem*, nie powinien być zapisywalny. To zablokuje użytkownikowi wykonującemu modyfikację istniejących binarek lub skryptów, co mogłoby umożliwić różne ataki. Kontenery niezmienne nie aktualizują swojego kodu automatycznie w czasie działania.

Nie wiąż się z określonym UID

Uruchom kontener jako użytkownik nie będący *rootem*, ale nie wymagaj podania UID tego użytkownika.

OpenShift domyślnie używa losowych identyfikatorów UID podczas uruchamiania kontenerów.

Wymuszenie konkretnego UID (np. pierwszego standardowego użytkownika z UID 1000) wymaga dostosowania uprawnień do wszelkich montowanych bindów, takich jak folder hosta dla persystencji danych. Ewentualnie, jeśli uruchomisz kontener (docker run -u) z UID hosta, może to spowodować błąd usługi podczas próby odczytu lub zapisu z folderów wewnątrz kontenera.

Drop capabilities

Również podczas wykonywania, możesz ograniczyć możliwości aplikacji do minimalnego wymaganego zestawu używając flagi --cap-drop w Docker lub securityContext.capabilities.drop w Kubernetes. W ten sposób w przypadku, gdy Twój kontener zostanie naruszony, zakres działań dostępnych dla atakującego jest ograniczony.

Nie umieszczaj sekretów w obrazie

Nigdy nie umieszczaj żadnych sekretów lub danych uwierzytelniających w instrukcjach Dockerfile.

Zachowaj szczególną ostrożność z plikami, które są kopiowane do kontenera. Nawet jeśli plik zostanie usunięty w późniejszej instrukcji w pliku Dockerfile, nadal będzie można uzyskać do niego dostęp na poprzednich warstwach, ponieważ tak naprawdę nie został usunięty, tylko "ukryty" w końcowym systemie plików.

Często aktualizuj swoje obrazy

Używaj obrazów bazowych, które są często aktualizowane, i na ich podstawie przebudowuj swoje. Ponieważ ciągle odkrywane są nowe luki w zabezpieczeniach, najlepszą praktyką jest stosowanie najnowszych poprawek bezpieczeństwa.

Nie ma potrzeby, aby zawsze przechodzić do najnowszej wersji, która może zawierać przełomowe zmiany, ale należy zdefiniować strategię wersjonowania:

- Trzymaj się wersji stabilnych lub wersji z długoterminowym wsparciem, które dostarczają poprawki bezpieczeństwa szybko i często.
- Planuj z wyprzedzeniem. Bądź gotowy do porzucenia starych wersji i migracji zanim Twoja podstawowa wersja obrazu osiągnie koniec swojego życia i przestanie otrzymywać aktualizacje.
- Przebuduj również swoje własne obrazy okresowo i z podobną strategią, aby uzyskać najnowsze pakiety z bazowego środowiska uruchomieniowego (np. Node, Python).

Instrukcje Dockerfile

- FROM

Kiedy to tylko możliwe, używaj aktualnych oficjalnych obrazów jako podstawy dla swoich obrazów. Polecamy obraz Alpine, ponieważ jest on ściśle kontrolowany i ma mały rozmiar (obecnie poniżej 6 MB), a jednocześnie jest pełną dystrybucją Linuksa.

- LABEL

Etykiety (metadane) można dodawać do obrazów, aby ułatwić porządkowanie obrazów według projektów, zapisywać informacje o licencjach, pomagać w automatyzacji lub z innych powodów.

Możesz spojrzeć na predefiniowane adnotacje ze specyfikacji obrazu OCI.

- RUN

Podziel długie lub złożone instrukcje **RUN** na wiele linii oddzielonych \, aby Twój plik Docker był bardziej czytelny, zrozumiały i łatwy w utrzymaniu.

Zawsze łącz apt-get update update z apt-get install w tym samym poleceniu **RUN**. Użycie samego apt-get update w poleceniu **RUN** powoduje problemy z cache'owaniem i kolejne instrukcje apt-get install kończą się niepowodzeniem.

Używaj potoku | (*pipe*), aby przesyłać dane dane z jednego polecenia do drugiego w ramach jednej instrukcji **RUN**.

- CMD

CMD powinno być używane do uruchamiania aplikacji w obrazie wraz z wszelkimi argumentami. **CMD** powinno być prawie zawsze używane w formie CMD ["executable", "param1", "param2"...]. W istocie, ta forma instrukcji jest zalecana dla każdego obrazu opartego na usługach.

W większości innych przypadków, **CMD** należy podać interaktywnej powłoce, takiej jak *bash*, *python* czy *perl*. Na przykład: CMD ["python"] lub CMD ["/bin/bash"]. Użycie tej formy oznacza, że kiedy wykonasz coś takiego jak docker run -it python, zostaniesz zrzucony do użytecznej powłoki, gotowej do działania.

CMD powinno być rzadko używane w sposób CMD ["param", "param"] w połączeniu z **ENTRYPOINT**

- ETRYPOINT

Najlepszym zastosowaniem ENTRYPOINT jest ustawienie głównego polecenia obrazu, pozwalającego na uruchomienie obrazu tak, jakby to było to polecenie (a następnie użycie CMD jako domyślnych flag); np.:

```
ENTRYPOINT ["s3cmd"]
CMD ["--help"]
```

Uruchomienie kontenera bez parametrów uruchomi sekcję *help*, a podanie argumentów na końcu polecenia docker run zastąpi instrukcję **CMD**.

Instrukcja **ENTRYPOINT** może być także używana w połączeniu ze skryptem pomocniczym, zwłaszcza jeśli uruchomienie narzędzia wymaga więcej niż jednego kroku: ENTRYPOINT ["/docker-entrypoint.sh"].
Dzięki temu można uruchamiać kontener na różne sposoby przewidziane w skrypcie.

- formy *shell* i *exec* (RUN / CMD / ENTRYPOINT)

Instrukcje RUN, CMD i ENTRYPOINT mają2 formy:

- ***shell*** **CMD <command>**

Uruchamia zawartość podaną w <command> w domyślnej powłoce (Linux: /bin/sh /-c, Windows: cmd /S /C).

- ***exec*** **CMD [“executable”, “param1”, “param 2”]**

W przeciwieństwie do formy powłoki, forma exec nie wywołuje powłoki poleceń. Oznacza to, że nie zachodzi normalne przetwarzanie powłoki. Na przykład, CMD ["echo", "$HOME"] nie wykona podstawiania zmiennych na $HOME. To właśnie powłoka wykonuje interpretację zmiennych środowiskowych, a nie docker.

Jeśli chcesz przetwarzania powłoki, to albo użyj formularza powłoki, albo wykonaj powłokę bezpośrednio, na przykład: CMD ["sh", "-c", "echo $HOME"].

Jeśli chcesz uruchomić swoją <command> bez powłoki, musisz wyrazić polecenie jako tablicę JSON i podać pełną ścieżkę do pliku wykonywalnego. Ta forma tablicowa jest preferowanym formatem **CMD**. Wszelkie dodatkowe parametry muszą być indywidualnie wyrażone jako łańcuchy w tablicy: CMD ["/usr/bin/wc","--help"]. Składnia JSON przekazuje argumenty do jądra bezpośrednio jako wywołanie **exec syscall**. W wywołaniu **exec** nie ma oddzielania polecenia od argumentów spacjami, uciekania od cudzysłowów, przekierowania IO, podstawiania zmiennych, przesyłania między poleceniami, uruchamiania wielu poleceń, itd. Syscall pobiera tylko program wykonywalny do uruchomienia i listę argumentów do przekazania temu programowi, i uruchamia go.

Forma exec umożliwia uniknięcie mungingu* łańcuchów powłoki i wykonywanie poleceń przy użyciu obrazu bazowego, który nie zawiera określonego pliku wykonywalnego powłoki.

* Mung

Temin używany w żargonie komputerowym dla serii potencjalnie destrukcyjnych lub nieodwracalnych zmian w kawałku danych lub pliku. Jest czasami używany dla niejasnych kroków transformacji danych. Powszechne operacje mungingu obejmują usuwanie interpunkcji lub znaczników HTML, parsowanie danych, filtrowanie i transformację.

- ADD / COPY
  - poprawne zastosowanie

Mimo podobieństw **ADD** i **COPY**, preferowane jest **COPY**, ponieważ jest bardziej przejrzysty. **COPY** obsługuje tylko podstawowe kopiowanie lokalnych plików do kontenera, podczas gdy **ADD** posiada funkcje, które nie są od razu oczywiste, takie jak lokalna ekstrakcja plików tar i obsługa zdalnych adresów URL, np. **ADD** rootfs.tar.xz /.

Jeśli wiele kroków Dockerfile używają różnych plików z kontekstu, kopiuj je pojedynczo. Zapewnia to, że cache’owanie każdego kroku jest unieważniane (zmuszając krok do ponownego uruchomienia) tylko wtedy, gdy konkretnie wymagane pliki ulegną zmianie.

Ze względu na rozmiar obrazu, używanie **ADD** do pobierania pakietów ze zdalnych adresów URL jest zdecydowanie odradzane; powinieneś użyć **curl** lub **wget** zamiast tego. W ten sposób możesz usunąć pliki, których już nie potrzebujesz po ich wyodrębnieniu.

- kontekst

Parametr "." jest kontekstem budowania. Używanie "." jako kontekstu jest niebezpieczne, ponieważ możesz skopiować poufne lub niepotrzebne pliki do kontenera

Dobrą praktyką Dockerfile jest utworzenie podkatalogu zawierającego pliki, które muszą być skopiowane wewnątrz kontenera, użycie go jako kontekstu budowania, a kiedy to możliwe, jawne instrukcje **COPY**, unikaj symboli wieloznacznych * (*wildcard*).

- .dockerignore

Należy również utworzyć plik **.dockerignore**, aby jawnie wykluczyć pliki i katalogi.

Nawet jeśli zachowasz szczególną ostrożność z instrukcjami **COPY**, cały kontekst budowania jest wysyłany do demona Dockera przed rozpoczęciem budowania obrazu. Oznacza to, że posiadanie mniejszego i ograniczonego kontekstu kompilacji przyspieszy kompilację.

- EXPOSE

Instrukcja **EXPOSE** informuje o portach, na których kontener nasłuchuje połączeń. Każdy otwarty port w twoim kontenerze to otwarte drzwi do twojego systemu. Eksponuj tylko te porty, których potrzebuje twoja aplikacja i unikaj eksponowania portów takich jak SSH (22). Powinieneś używać zwykłego, tradycyjnego portu dla swojej aplikacji: np. obraz serwera Apache użyje EXPOSE 80. Następnie trzymaj się tych portów podczas publikowania lub eksponowania w czasie wykonywania.

Odsłonięcie portu nie powoduje automatycznego zezwolenia na połączenia dla wszystkich portów podczas uruchamiania kontenera (chyba że użyjesz ~~$ docker run --publish-all~~).

- VOLUME

Instrukcja **VOLUME** powinna być używana do wystawienia dowolnego obszaru bazy danych, przechowywania konfiguracji lub plików/folderów utworzonych przez twój kontener docker. Zaleca się używanie **VOLUME** dla wszystkich części obrazu, które mogą ulec zmianie i/lub być obsługiwane przez użytkownika.

- WORKDIR

Dla przejrzystości i niezawodności, powinieneś zawsze używać bezwzględnych ścieżek do swojego **WORKDIR**.

Należy też używać **WORKDIR** zamiast ~~**RUN cd ... &&** do-something~~.

- ENV

Używanie **ENV** pozwala na zmianę pojedynczej zmiennej np. w celu automatycznego podniesienia wersji oprogramowania w kontenerze, czy też do dostarczania wymaganych zmiennych środowiskowych specyficznych dla skonteneryzowanej aplikacji.

Każda linia **ENV** tworzy nową warstwę pośrednią**!** Oznacza to, że nawet jeśli usuniesz zmienną środowiskową w przyszłej warstwie, to nadal pozostaje ona w tej warstwie:

```
FROM alpine
ENV ADMIN_USER="mark"
RUN echo $ADMIN_USER > ./mark
RUN unset ADMIN_USER
```

```
$ docker run --rm test sh -c 'echo $ADMIN_USER'
```

resp: mark

Aby do tego nie dopuścić i naprawdę usunąć zmienną środowiskową uruchom wszystko w jednej instrukcji **RUN**:

```
FROM alpine
RUN export ADMIN_USER="mark" \
    && echo $ADMIN_USER > ./mark \
    && unset ADMIN_USER
```

- USER

Użyj **USER**, aby zmienić i nie używać użytkownika *root*, jeśli usługa może działać bez przywilejów. Utwórz użytkownika i grupę w Dockerfile:

```
RUN groupadd -r postgres && useradd --no-log-init -r -g postgres postgres
```

(--no-log-init chroni przed rozrośnięciem się pliku /var/log/lastlog do ogromnych rozmiarów.)

Jeśli jednak kontener musi uruchomić specyficzne polecenie jako *root*, może użyć ***sudo*** albo specjalnego użytwnika **gosu** lub **su-exec**.

Aby zredukować warstwy i złożoność, unikaj częstego przełączania USER tam i z powrotem.

- HEALTHCHECK

Kiedy używasz zwykłego Dockera lub Docker Swarm, dołącz instrukcję HEALTHCHECK do swojego pliku Dockera, kiedy tylko jest to możliwe. Jest to krytyczne dla długo działających lub trwałych usług, aby upewnić się, że są zdrowe i zarządzać ponownym uruchomieniem usługi w przeciwnym razie.

- ONBUILD

ONBUILD wykonuje się w każdym obrazie potomnym pochodzącym z bieżącego obrazu po zakończeniu budowania bieżącego pliku Dockerfile. Docker wykonuje polecenia **ONBUILD** przed jakimikolwiek poleceniami w pliku potomnym Dockerfile.

Jest to przydatne, jeśli budujesz obraz, który będzie używany jako podstawa do budowania innych obrazów, na przykład środowiska budowania aplikacji lub demona, który może być dostosowany do konfiguracji użytkownika.

Na przykład, *jeśli twój obraz jest konstruktorem aplikacji Pythona wielokrotnego użytku, będzie wymagał dodania kodu źródłowego aplikacji w określonym katalogu, a następnie może wymagać wywołania skryptu budującego.*

Obrazy zbudowane za pomocą **ONBUILD** powinny otrzymać osobny tag, na przykład:
*ruby:1.9-onbuild lub ruby:2.0-**onbuild***.

Dobre praktyki poza budowaniem obrazu

Chroń porty TCP i socket

Socket Dockera to duże uprzywilejowane drzwi do systemu hosta, które, mogą być wykorzystane do włamania i użycia złośliwego oprogramowania. Upewnij się, że twój /var/run/docker.sock ma odpowiednie uprawnienia, a jeśli docker jest wystawiony przez TCP (co nie jest w ogóle zalecane), upewnij się, że jest odpowiednio chroniony.

Izolacja kontenerów w przestrzeni nazw użytkownika na hoście

Linuksowe przestrzenie nazw zapewniają izolację dla uruchomionych procesów, ograniczając ich dostęp do zasobów systemowych bez świadomości ograniczeń przez uruchomiony proces. Najlepszym sposobem zapobiegania atakom wykorzystującym przywileje z wnętrza kontenera jest skonfigurowanie aplikacji kontenera do uruchamiania jako nieuprzywilejowani użytkownicy.

W przypadku kontenerów, których procesy muszą działać jako użytkownik root w kontenerze, można przemapować tego użytkownika na mniej uprzywilejowanego użytkownika na hoście Dockera. Zmapowany użytkownik otrzymuje zakres UID, które funkcjonują w przestrzeni nazw jak normalne UID od 0 do 65536, ale nie mają żadnych przywilejów na hoście.

Samo przemapowanie jest obsługiwane przez dwa pliki: /etc/subuid i /etc/subgid. Oba pliki działają tak samo, ale **subuid** zajmuje się zakresem ID użytkownika, a **drugi** zakresem ID grupy. Rozważmy następujący wpis w /etc/subuid:

```
testuser:231072:65536
```

Oznacza to, że *testuser* otrzymuje podrzędny identyfikator użytkownika z zakresu *231 072* i kolejnych *65 536* liczb całkowitych. *UID 231072* jest mapowany w przestrzeni nazw (w tym przypadku w kontenerze) jako *UID 0* (*root*). *UID 231073* jest mapowany jako *UID 1*, i tak dalej. Jeśli proces próbuje eskalować przywileje poza przestrzenią nazw, proces ten działa jako nieuprzywilejowany UID o wysokim numerze na hoście, który nawet nie mapuje się na prawdziwego użytkownika. Oznacza to, że proces nie ma żadnych przywilejów w systemie hosta.

Bardzo ważne jest, aby zakresy nie nakładały się na siebie, tak aby proces nie mógł uzyskać dostępu do innej przestrzeni nazw.

Kiedy konfigurujesz Dockera do użycia mechanizmu **userns-remap**, możesz opcjonalnie podać istniejącego użytkownika i/lub grupę, lub możesz podać **default**. Jeśli podasz *default*, użytkownik i grupa *dockremap* są tworzone i używane do tego celu.

Podpisywanie obrazów i weryfikacja podpisów

Jedną z dobrych praktyk jest używanie *docker content trust*, *Docker notary*, *Harbor notary* lub podobne narzędzia do cyfrowego podpisywania obrazów, a następnie weryfikowania ich podczas runtime.

Komendy Dockera dot. kontenerów

Basic Commands - Life Cycle

- `docker create` utwórz kontener, nie uruchamiając go
- `docker rename` zmień nazwę kontenera
- `docker run` utwórz i uruchom kontener w jednej operacji
- `docker rm` skasuj kontener
- `docker update` zaktualizuj limity zasobów kontenera

Start and Stop

- `docker start` uruchom kontener (running)
- `docker stop` zatrzymaj uruchomiony (running) kontener
- `docker restart` zatrzymaj I uruchom kontener
- `docker pause` wstrzymaj uruchomiony kontener
- `docker unpause` wznów wstrzymany kontener
- `docker kill` wyślij SIGKILL do uruchomionego kontenera
- `docker attach` połącz lokalny input, output, error stream do uruchomionego kontenera

Information

- `docker ps` pokaż uruchomione kontenery
- `docker ps -a` pokaż uruchomione i zatrzymane kontenery
- `docker logs` pokaż logi kontenera (ze strumienia)
- `docker inspect` pokaż wszystkie informacje o kontenerze (w tym adresy IP)
- `docker events` pokaż zdarzenia z kontenera
- `docker port` pokaż publiczne porty kontenera
- `docker top` pokaż uruchomione procesy wewnątrz kontenera
- `docker stats` pokaż statystyki użycia zasobów uruchomionych kontenerów
- `docker stats --all` pokaż statystyki użycia zasobów wszystkich kontenerów
- `docker diff` pokaż zmodyfikowane pliki w systemie plików kontenera

Import / Export

- `docker cp` skopiuj pliki/foldery pomiędzy systemami plików kontenerów
- `docker export` wyeksportuj system plików konteners do archiwum tar

Kubernetes to system operacyjny świata cloud native, zapewniający niezawodną i skalowalną platformę do uruchamiania zadań kontenerowych. Kubernetes łączy wiele serwerów w *klaster.*

Kubernetes obsługuje różne container runtimes: Docker, containerd, CRI-O oraz każdą implementację zgodną z Kubernetes CRI (Container Runtime Interface).

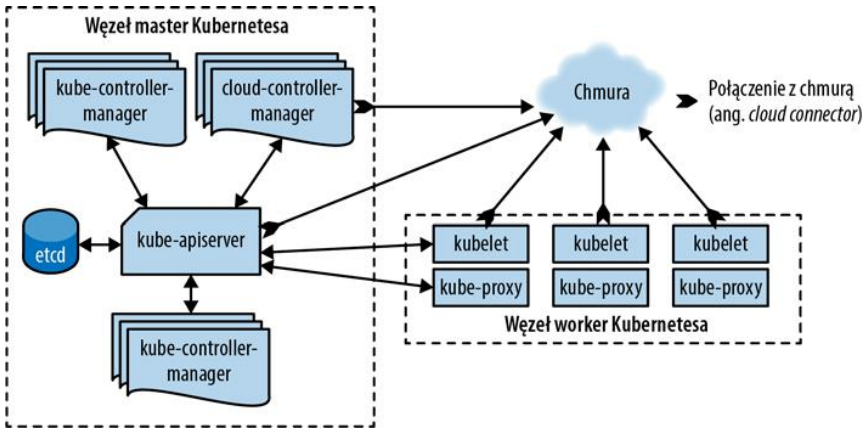
Architektura klastrowa

Warstwa sterowania – *control plane*

Mózg klastra nazwany został *warstwą sterowania* (ang. ***control plane***) i wykonuje wszystkie zadania wymagane przez Kubernetes do jego pracy, takie jak:

planowanie kontenerów, zarządzanie usługami, obsługa żądań API, itp.

Elementy klastra, które uruchamiają komponenty warstwy sterowania, nazywane są ***węzłami master***.
Warstwa sterowania składa się w rzeczywistości z kilku elementów.



- kube-apiserver

Serwer front-end warstwy sterowania, obsługujący **żądania API**.

- etcd

Baza danych, w której Kubernetes przechowuje wszystkie **informacje o klastrze**:

jakie węzły istnieją, jakie zasoby istnieją w klastrze, itd.

- kube-scheduler

Decyduje, gdzie uruchomić nowo utworzone **Pody**.

- kube-controller-manager

Element odpowiedzialny za uruchamianie **kontrolerów zasobów**, takich jak Deployments.

- cloud-controller-manager

Współdziała z dostawcą chmury (w klastrach opartych na chmurze), **zarządzając zasobami**, takimi jak usługi równoważenia obciążenia i woluminy dyskowe.

Elementy węzła

Elementy klastra, które uruchamiają zadania użytkowników, nazywane są ***węzłami worker***.

Poza uruchomieniem różnych komponentów oprogramowania, nie ma istotnej różnicy między węzłami master a węzłami worker. Węzły master zwykle jednak nie uruchamiają zadań użytkowników, z wyjątkiem bardzo małych klastrów (takich jak Docker Desktop lub Minikube).

Każdy węzeł worker w klastrze Kubernetes uruchamia następujące komponenty:

- kubelet

Odpowiedzialny za uruchomienie środowiska wykonawczego kontenera w celu uruchomienia zadań zaplanowanych w węźle
i monitorowania ich stanu.

- kube-proxy

Komponent odpowiedzialny za obsługę routingu żądań między Podami w różnych węzłach oraz między Podami a internetem.

- container runtime (środowisko wykonawcze kontenera)

Komponent, który uruchamia i zatrzymuje kontenery oraz obsługuje ich komunikację. Zazwyczaj to Docker, ale Kubernetes obsługuje inne środowiska wykonawcze, np. CRI-O.

System wysokiej niezawodności (ang. high availability)

Awaria warstwy sterowania

Wysoka niezawodność warstwy sterowania ma kluczowe znaczenie dla prawidłowo działającego klastra.

Prawidłowo skonfigurowana warstwa sterowania Kubernetes posiada wiele węzłów master.
Jeśli dowolny pojedynczy węzeł master ulegnie awarii, zostanie zamknięty lub węzły master nie mogą komunikować się z innymi z powodu awarii sieci, klaster nadal będzie działał poprawnie.

Baza danych ***etcd*** jest replikowana na wielu węzłach i może przetrwać awarię poszczególnych węzłów, o ile kworum przekraczające połowę liczby replik *etcd* jest nadal dostępne.

Awaria węzła worker

Ufaj, ale sprawdzaj:
W czasie zaplanowanej konserwacji lub poza godzinami szczytu spróbuj zresetować węzeł worker i sprawdź, co się stanie.

Koszty samodzielnego hostingu Kubernetes

- Czy warstwa sterowania jest wysoko niezawodna?
- Czy pula węzłów worker jest wysoce niezawodna?
- Czy klaster jest zabezpieczony?
- Czy wszystkie usługi w klastrze są bezpieczne?
- Czy klaster został *skonfigurowany* zgodnie ze standardami?
- Czy węzły klastra są w pełni zarządzane przez konfigurację (ang. *config-managed*)?
- Czy w klastrze są prawidłowo wykonywane kopie zapasowe?
- Jak już będziesz mieć działający klaster, to jak w miarę upływu czasu będzie wyglądał proces jego utrzymania?

Zarządzane usługi Kubernetes

Zarządzane usługi Kubernetes zwalniają Cię z prawie wszystkich kosztów administracyjnych związanych z konfigurowaniem
i uruchamianiem klastrów Kubernetes, zwłaszcza warstwy sterowania. Korzystanie z usług zarządzanych może być znacznie bardziej opłacalne niż samodzielne hostowanie klastrów Kubernetes.

Przykady:

- Google Kubernetes Engine (GKE)
- Amazon Elastic Container Service for Kubernetes (EKS)
- Azure Kubernetes Service (AKS)
- OpenShift
- IBM Cloud Kubernetes Service

Uruchom mniej oprogramowania – „run less software”

Istnieją trzy filary filozofii „run less software”. Wszystkie pomogą optymalizować czas i pokonać konkurencję.

1. Wybierz standardową technologię.
2. Zlecaj na zewnątrz „podnoszenie banalnych ciężarów".
3. Stwórz trwałą przewagę konkurencyjną.

— Rich Archbold

Bezklastrowe usługi kontenerowe

Jeśli chcesz zminimalizować obciążenie związane z uruchamianiem zadań kontenerowych, masz do wyboru jeszcze tzw. usługi *bezklastrowe*.
Chociaż pod spodem znajduje się klaster, nie masz do niego dostępu. Zamiast tego określasz obraz kontenera do uruchomienia i kilka parametrów, takich jak wymagania CPU i pamięci aplikacji. Usługa zajmie się resztą.

Dostępne usługi:

- Amazon Fargate
- Azure Container Instances (ACI)

Rozdział 4. Praca z obiektami Kubernetes

Deployment

Deployment to deklaracja pożądanego stanu.

Dla każdego programu, który Kubernetes musi nadzorować, tworzony jest odpowiedni **obiekt Deployment**; obiekt rejestruje pewne informacje o programie:

- nazwę obrazu kontenera,
- liczbę replik,
- oraz cokolwiek innego, co musi wiedzieć, aby uruchomić kontener.

Współdziałanie z zasobami Deployment kontrolowane jest przez **kontroler** (***controller***).
Kontrolery obserwują zasoby, za które są odpowiedzialne, upewniając się, że:

- są obecne i działają,
- stan rzeczywisty odpowiada stanowi żądanemu.

Deployment nie zarządza bezpośrednio replikami. Zamiast tego automatycznie tworzy powiązany obiekt o nazwie **ReplicaSet**, który to obsługuje.

Pody

Pod to obiekt Kubernetes, który reprezentuje **grupę** jednego lub wielu **kontenerów**.

Komenda `$ kubectl run` nie tworzy Poda bezpośrednio. Został utworzony Deployment, a następnie Deployment uruchomił Pod.

ReplicaSet

ReplicaSet jest odpowiedzialny za **grupę** identycznych **Podów** lub ***replik*** (***replicas***). Jeśli jest za mało (lub zbyt wiele) Podów
w porównaniu ze specyfikacją, kontroler ReplicaSet uruchomi (lub zatrzyma) niektóre Pody w celu naprawy sytuacji.

Kiedy aktualizujesz Deployment, tworzony jest nowy obiekt ReplicaSet w celu zarządzania nowymi Podami.

Utrzymanie pożądanego stanu - *reconciliation loop*

Proces ten nazywa się pętlą uzgadniania (*reconciliation loop*).

Scheduler

Scheduler jest komponentem odpowiedzialnym za **uruchomie** wymaganych **Podów**.

Gdy Deployment (za pośrednictwem ReplicaSet) zdecyduje, że potrzebna jest nowa replika, tworzy zasób Pod w bazie danych Kubernetes. Jednocześnie ten Pod jest dodawany do kolejki, która działa jak skrzynka odbiorcza komponentu scheduler.

Zadaniem Schedulera jest obserwowanie kolejki nieprzydzielonych Podów, wybranie z niego kolejnego Poda i znalezienie węzła, na którym można go uruchomić.

Po przydzieleniu Poda do węzła proces **kubelet** działający w tym węźle pobiera go i dba o faktyczne uruchomienie kontenerów.

Serwis

Serwis podaje pojedynczy, niezmienny **adres IP** lub **nazwę DNS**, które zostaną automatycznie przekierowane do dowolnego pasującego Poda. Możesz myśleć o zasobie Serwis jak o serwerze *proxy* lub *load balancer*.
Serwis może przekazywać ruch z dowolnego portu do dowolnego innego portu, zgodnie z opisem w części specyfikacji *ports*.

Pole *selector* informuje Serwis, jak kierować żądania do poszczególnych Podów. Żądania będą przekazywane do dowolnych Podów zgodnych z określonym zestawem **etykiet** (***label***).

Manifesty zasobów w formacie YAML (JSON)

Zasoby są danymi

Wszystkie **zasoby** Kubernetes, takie jak Deployment lub Pod, są reprezentowane **jako rekordy** w wewnętrznej **bazie danych**. **Pętla uzgadniania** śledzi bazę danych pod kątem zmian w tych rekordach i podejmuje odpowiednie działania.

Rozdział 5. Zarządzanie zasobami

Utrzymuj małe kontenery

Utrzymywanie jak najmniejszych obrazów kontenera, z wielu powodów, jest dobrym pomysłem.

- Małe kontenery budują się szybciej.
- Obrazy zajmują mniej miejsca.
- Krócej trwa ładowanie obrazów.
- Są mniejsze możliwości ataku.

Zarządzanie cyklem życia kontenera

Sondy żywotności – liveness probes

Jeśli aplikacja odpowiada kodem stanu HTTP 2xx lub 3xx, Kubernetes uważa ją za działającą. Jeśli zareaguje inaczej lub w ogóle nie zareaguje, kontener będzie uznany za martwy i zostanie ponownie uruchomiony.

Sondy gotowości – readiness probes

Kontener, który nie przejdzie testu sondy gotowości, zostanie usunięty z dowolnych Serwisów obsługujących dany Pod. To jest jak usunięcie uszkodzonego węzła z puli load balancera: żaden ruch nie zostanie wysłany do Poda, dopóki jego sonda gotowości nie przejdzie pozytywnego testu.

Kontener, który nie jest gotowy, nadal będzie miał status Running, ale w kolumnie READY pojawi się co najmniej jeden nieprzygotowany kontener w Podzie.

Uwaga:

Sondy gotowości powinny zwracać tylko status HTTP 200 OK. Chociaż Kubernetes uważa oba kody stanu 2xx i 3xx za poprawne, to usługi load balancera — nie. Jeśli korzystasz z zasobu Ingress w połączeniu z modułem load balancera w chmurze, a Twoja sonda gotowości zwraca np. kod 301 (przekierowanie), moduł load balancera może oznaczyć Pody jako niesprawne. Upewnij się, że Twoje sondy gotowości zwracają tylko kod statusu 200.

Inne typy sond

- tcpSocket
- exec: command

Sonda *exec* kończy się powodzeniem, jeśli komenda kończy działanie statusem 0.

- gRPC

*Modern open-source high performance **Remote Procedure Call** (**RPC**) framework that can run in any environment.*

Protokoły tego typu mają na celu ułatwienie komunikacji pomiędzy komputerami.

minReadySeconds

Kontener lub Pod nie będą uważane za gotowe, dopóki sonda gotowości nie będzie gotowa przez *minReadySeconds* sekund (domyślnie 0).

Korzystanie z przestrzeni nazw – namespace

Przestrzeń nazw Kubernetes to sposób na podzielenie klastra na osobne poddziały (zależnie od przeznaczenia). Nazwy w jednej przestrzeni nazw nie są widoczne z innej przestrzeni nazw.

Adresy serwisów

Nazwy usług DNS zawsze mają następujący format:

SERVICE.NAMESPACE.svc.cluster.local

Część ***.svc.cluster.local*** jest opcjonalna, podobnie jak przestrzeń nazw. Jeśli jednak np. chcesz skomunikować się z serwisem demo w przestrzeni nazw prod, możesz użyć nazwy **demo.prod**.

Przydziały zasobów – Resource Quota

Możesz (i powinieneś – best practices) ograniczyć wykorzystanie zasobów w danej przestrzeni nazw. Sposobem na to jest utworzenie **ResourceQuota** w przestrzeni nazw.

Domyślne żądania zasobów i limity – LimitRange

Za pomocą zasobu **LimitRange** możesz ustawić domyślne żądania i limity dla wszystkich kontenerów w przestrzeni nazw. Każdy kontener w przestrzeni nazw, który nie określa limitu zasobów ani żądania, odziedziczy wartość domyślną z **LimitRange**.

Rozdział 6. Operacje na klastrach

Rozmiar i skalowanie klastra

Najmniejszy klaster

W celu zapewnienia wysokiej niezawodności klastry Kubernetes wymagają co najmniej trzech węzłów master - być może potrzeba będzie więcej do obsługi pracy większych klastrów.
Dwa węzły worker to minimum wymagane do tego, aby obciążenia były odporne na awarie pojedynczego węzła, a lepiej posiadać trzy takie węzły.

Największy klaster

Aby zapewnić maksymalną niezawodność, w klastrze Kubernetes powinno być mniej niż 5000 węzłów i 150 000 Podów (nie jest to problem dla większości użytkowników).
Jeśli potrzebujesz więcej zasobów, uruchom wiele klastrów.

Klastry sfederowane

Jeśli chcesz zreplikować obciążenia w wielu klastrach, być może ze względu na opóźnienia wynikające z różnych lokalizacji geograficznych, wykorzystaj federacje. Jednak większość użytkowników nie musi tego stosować.

Czy potrzebuję wielu klastrów?

Użyj pojedynczego klastra produkcyjnego i pojedynczego klastra do rozwoju oprogramowania, jeśli potrzebujesz pełnej izolacji jednego zestawu obciążeń lub zespołów od drugiego.
Jeśli chcesz po prostu podzielić klaster na partycje, aby ułatwić zarządzanie, skorzystaj z przestrzeni nazw.

Węzły i instancje

Wybór odpowiedniego rozmiaru węzła

Użyj najbardziej opłacalnego typu węzła, który oferuje Twój dostawca. Często większe węzły są tańsze, ale jeśli masz tylko kilka węzłów, możesz dodać kilka mniejszych, aby pomóc w nadmiarowości.

Typy instancji chmurowych

Węzeł główny dla małych klastrów (do około pięciu węzłów) powinien mieć co najmniej jeden wirtualny procesor (vCPU) i 3 – 4 GiB pamięci, przy czym większe klastry wymagają więcej pamięci i procesorów dla każdego węzła master. Jest to odpowiednik instancji n1-standard-1 w Google Cloud, m3.medium w AWS i Standard DS1 v2 na Azure.

Instancja pojedynczego procesora 4 GiB jest również rozsądnym minimum dla węzła worker, chociaż — jak widzieliśmy — czasem może być bardziej opłacalny zakup większych węzłów. Przykładowo domyślny rozmiar węzła w Google Kubernetes Engine, który w przybliżeniu spełnia tę specyfikację, to n1-standard-1.

W przypadku większych klastrów, które mogą mieć kilkadziesiąt węzłów, sensowne może być zapewnienie dwóch lub trzech różnych rozmiarów instancji.

Węzły heterogeniczne

Większość kontenerów jest zbudowanych dla systemu Linux, więc prawdopodobnie będziesz chciał uruchamiać głównie węzły oparte na systemie Linux. Konieczne może być jednak dodanie jednego lub dwóch specjalnych typów węzłów dla określonych wymagań, np. zawierających GPU lub Windows.

Serwery bare-metal

Jeśli posiadasz własne serwery lub nie jesteś jeszcze gotowy do pełnej migracji do rozwiązania chmurowego, użyj Kubernetes, aby uruchomić obciążenia kontenerów na istniejących komputerach.

Skalowanie klastra

Grupy instancji

Narzędzie **kops** obsługuje pojęcie grupy instancji, która jest zbiorem węzłów danego typu instancji (np. m3.medium). Można skalować grupy instancji lub grupy węzłów, zmieniając minimalny i maksymalny rozmiar grupy, określony typ instancji lub oba te elementy.

Skalowanie w dół

Możesz nakazać Kubernetes opróżnienie węzłów, które chcesz usunąć, co spowoduje stopniowe zamykanie lub przenoszenie działających Podów na inne węzły. Komenda:

```
$ kubectl drain
```

pod warunkiem, że w pozostałej części klastra jest wystarczająca ilość wolnej pojemności.

Aby uniknąć zbytniego zmniejszania liczby replik Podów dla danej usługi, możesz skorzystać z PodDisruptionBudgets, aby określić minimalną liczbę dostępnych Podów lub maksymalną liczbę Podów, które mogą być niedostępne w dowolnym momencie.

Automatyczne skalowanie

Kubernetes ma dodatek o nazwie **Cluster Autoscaler**, który może korzystać z narzędzi do zarządzania klastrami, takich jak **kops** — w celu umożliwienia zastosowania funkcji skalowania w chmurze.

Uwaga:

Nie włączaj opcji automatycznego skalowania klastra tylko dlatego, że tam jest. Prawdopodobnie nie będziesz jej potrzebować, aż obciążenia nie staną się bardzo zmienne. Dopóki nie zorientujesz się, jak potrzeby zmieniają się w czasie, skaluj klaster ręcznie.

Walidacja i audyt

Każdy klaster produkcyjny z pewnością powinien być mieć zweryfikowaną **zgodność** (***conformant***).
Istnieje jednak wiele typowych problemów z konfiguracjami Kubernetes i obciążeniami, których nie sprawdzą testy zgodności, takie jak **Sonobuoy**.

Oto przykłady:

- używanie zbyt dużych obrazów kontenerów może zmarnować dużo czasu i zasobów klastra;
- obiekty Deployment, które posiadają tylko jedną replikę Poda, nie zapewniają wysokiej niezawodności;
- uruchamianie procesów w kontenerach jako *root* jest potencjalnym zagrożeniem dla bezpieczeństwa.

Oto niektóre narzędzia i techniki, które mogą pomóc w znalezieniu problemów związanych z klastrem:

- K8Guard,
- Copper,
- kube-bench,
- dziennik kontroli Kubernetes (ang. Kubernetes audit log).
  Po włączeniu dziennika kontroli wszystkie żądania do interfejsu API klastra będą rejestrowane razem ze znacznikiem czasu. Uzyskasz informacje o tym, kto wysłał żądanie (które konto usługi), szczegóły żądania, takie jak kwerenda, której dotyczy zapytanie, i jaka była odpowiedź.

Rozdział 7. Narzędzia Kubernetes

kubectl

Narzędzie kubectl jest podstawowym narzędziem służącym do interakcji z klastrem Kubernetes i może być używane:

- *imperatywnie* (aby np. uruchomić publiczny obraz kontenera i utworzyć niezbędne zasoby Kubernetes)
- *deklaratywnie*, aby zastosować plik konfiguracyjny Kubernetes w formacie YAML.

Aliasy powłoki

W pliku ***.bash_profile*** ustawiony następujący alias:

```
$ alias k = kubectl
```

Teraz, zamiast wpisywać kubectl w całości dla każdego polecenia, możemy po prostu użyć k:

```
$ k get pods
```

Jeśli często używasz komend kubectl, również możesz dla nich utworzyć aliasy, np.

```
$ alias kgdep=kubectl get deployment
```

Automatyczne uzupełnianie poleceń kubectl

```
$ kubectl completion -h
```

Uzyskiwanie pomocy na temat zasobów Kubernetes

```
$ kubectl explain pods
```

```
$ kubectl explain deploy.spec.template.spec.containers.livenessProbe.exec
```

jq

```
$ kubectl get pods -o json --all-namespaces | jq '.items | group_by(.spec.nodeName) |
map({"nodeName": .[0].spec.nodeName, "count": length}) | sort_by(.count) | reverse'
```

Jeśli nie masz dostępu do jq, narzędzie **kubectl** obsługuje również zapytania **JSONPath**.

```
$ kubectl get pods -o=jsonpath={.items[0].metadata.name}
```

Oglądanie obiektów

```
$ kubectl get pods –watch
```

Opisywanie obiektów

Szczegółowe informacje o obiektach Kubernetes.

```
$ kubectl describe pods demo-d94cffc44-gvgzm
```

Imperatywne polecenia kubectl

Np.
```
$ kubectl create namespace my-new-namespace
```

```
$ kubectl edit deployments my-deployment
```

Nie używaj poleceń imperatywnych kubectl, takich jak ~~create~~ lub ~~edit~~, w klastrach produkcyjnych.
Zamiast tego zawsze zarządzaj zasobami za pomocą manifestów **YAML**, weryfikowanych przez systemy kontroli wersji, stosowanych razem z poleceniem $ kubectl **apply** (lub za pomocą wykresów **Helm**).

Generowanie manifestów zasobów

Zamiast wpisywać powtarzające się słowa kluczowe do pustego pliku, skorzystaj z kubectl, które wygeneruje manifest **YAML** za Ciebie:

```
$ kubectl create deployment demo --image=cloudnatived/demo:hello --dry-run=client -o yaml > deployment.yaml
```

```
$ kubectl get deployments newdemo -o yaml > deployment.yaml
```

Dane wyjściowe będą zawierać dodatkowe informacje, jak np. sekcję status, które możesz bezpiecznie usunąć przed zapisaniem ich razem z innymi manifestami.

Śledzenie różnic w zasobach

Zanim zastosujesz manifesty Kubernetes za pomocą polecenia kubectl apply, zobacz dokładnie, co by się zmieniło w klastrze. Komenda kubectl diff Ci w tym pomoże.

```
$ kubectl diff -f deployment.yaml
```

Uwaga:

Przed zastosowaniem jakichkolwiek aktualizacji w klastrze produkcyjnym użyj polecenia kubectl diff, aby sprawdzić, co by się zmieniło.

Przekierowanie portu kontenera

```
$ kubectl port-forward demo-54f4458547-vm88z 9999:8888
```

Teraz port **9999 na Twoim komputerze** lokalnym zostanie **przekierowany do** portu **8888 na kontenerze** i możesz np. połączyć się z nim za pomocą przeglądarki internetowej.

Wykonywanie poleceń w kontenerach

Za pomocą polecenia kubectl exec możesz uruchomić określone polecenie w dowolnym kontenerze, w tym w powłoce:

```
$ kubectl exec -it -c container2 <POD_NAME>/bin/sh
```

BusyBox

Obraz **busybox** jest szczególnie przydatny. Zawiera wiele najczęściej używanych komend uniksowych, takich jak:

cat, echo, find, grep i kill. Pełną listę poleceń BusyBox możesz zobaczyć na stronie internetowej.

BusyBox zawiera również lekką powłokę podobną do bash, zwaną ash, która jest kompatybilna ze standardowymi skryptami powłoki /bin/sh. Aby uzyskać interaktywną powłokę w klastrze, możesz uruchomić:

```
$ kubectl run busybox --image=busybox:1.28 --rm -it --restart=Never /bin/sh
```

Wzorzec uruchamiania poleceń z obrazu BusyBox jest zawsze taki sam, można więc utworzyć dla niego alias powłoki

```
$ alias bb=kubectl run busybox --image=busybox:1.28 --rm -it --restart=Never --command --
$  bb nslookup demo
```

gdzie:

- `--rm`

Flaga informuje Kubernetes, że ma usunąć zasoby stworzone przez to polecenie dla powiązanych kontenerów po zakończeniu działania, aby nie zaśmiecał lokalnej pamięci Twoich węzłów.

- `--restart=Never`

Chcemy uruchomić kontener tylko raz, możemy wyłączyć domyślną zasadę restartu.

- `--command`

Określa polecenie do uruchomienia zamiast domyślnego punktu wejścia kontenera.

- `--`

Wszystko, co następuje po --, zostanie przekazane do kontenera jako linia poleceń wraz z argumentami.

Jeśli nie ma /bin/sh w kontenerze, najprostszym sposobem na łatwe debugowanie kontenerów, przy zachowaniu bardzo małych obrazów, jest skopiowanie do nich pliku wykonywalnego **busybox** podczas kompilacji.
To tylko 1 MiB, czyli niewielka cena za posiadanie użytecznej powłoki i zestawu narzędzi uniksowych.

```
COPY --from=busybox:1.28 /bin/busybox /bin/busybox
```

Teraz nadal masz bardzo mały kontener, ale możesz również na nim skorzystać z powłoki:

```
$ kubectl exec -it <POD_NAME> /bin/busybox sh
```

Konteksty i przestrzenie nazw

kubectl stosuje pojęcie **kontekstów** (***contexts***). Kontekst jest kombinacją:

- klastra,
- użytkownika,
- przestrzeni nazw.

```
$ kubectl config set-context myapp --cluster=gke --namespace=myapp
$ kubectl config get-contexts
$ kubectl config use-context context-name
```

Jeśli zapomnisz, jaki jest Twój obecny kontekst, podpowie Ci polecenie:

```
$ kubectl config current-context
```

Przydatne narzędzia Kubernetes

kubectx i kubens

Narzędzia do szybkiego przełączania kontekstów (**kubectx**) i przestrzeni nazw (**kubens**).

kube-shell

Wyświetla możliwe uzupełnienia dla każdego polecenia *kubectl*.

Click

Z pomocą **Click** możesz wybrać dowolny zasób z listy, wpisując jego numer (np. 1 dla pierwszego elementu). To jest teraz bieżący zasób, a następne polecenie Click domyślnie będzie działać na tym zasobie, które wyświetlają się w wierszu pleceń:

```
[context] [namespace] [object] > #
```

Aby ułatwić znalezienie żądanego obiektu, Click obsługuje wyszukiwanie według wyrażeń regularnych.

Stern

Bardziej wyrafinowane narzędzie do śledzenia dzienników pozwala określić grupę Podów za pomocą wyrażenia regularnego czy zestawu etykiet, a dodatkowo obserwacja dzienników nie jest zakłócona nawet po ponownym uruchomieniu poszczególnych kontenerów.

Rozdział 8. Uruchamianie kontenerów

Co należy do kontenera?

Jeśli procesy nie powinny o sobie nic wiedzieć, nie muszą działać w tym samym kontenerze. Dobrą zasadą dotyczącą kontenera jest to, że powinien on robić jedną rzecz. Każdy kontener powinien uruchamiać tylko jeden główny proces. Jeśli prowadzisz dużą grupę niepowiązanych procesów w kontenerze, powinieneś pomyśleć o podzieleniu aplikacji na wiele komunikujących się kontenerów.

Co należy do Poda?

Wszystkie kontenery w Podzie powinny współpracować, aby wykonać jedną pracę. Jeśli potrzebujesz tylko jednego kontenera do wykonania tej pracy, użyj jednego kontenera. Jeśli potrzebujesz dwóch lub trzech, w porządku. Jeśli masz więcej, możesz zastanowić się, czy kontenery można podzielić na osobne Pody.

Ogólnie rzecz biorąc, właściwe pytanie brzmi:
„Czy te kontenery będą działać poprawnie, kiedy wylądują na różnych maszynach?”.
Jeśli odpowiedź brzmi: „Nie”, Pod zapewni prawidłowe grupowanie kontenerów.
Jeśli odpowiedź brzmi „Tak”, zastosowanie wielu Podów jest tutaj prawdopodobnie właściwym rozwiązaniem.

Polityka pobierania obrazu

Pole imagePullPolicy w kontenerze

określa, jak często Kubernetes będzie pobierać obraz z odpowiedniego rejestru kontenera.

- Always

Obraz jest wyciągany za każdym razem, gdy kontener jest uruchamiany.

Uwaga:

Zakładając, że określisz **tag**, jest to niepotrzebne, marnuje czas i przepustowość.

- IfNotPresent (default)

Jeśli obraz nie jest już obecny w węźle, zostanie pobrany. Zapisany obraz będzie używany przy każdym uruchomieniu kontenera (chyba że zmienisz specyfikację obrazu).

- Never

Nigdy nie pobierze obrazu z rejestru.
Jeśli jest już obecny w węźle, zostanie użyty; jeśli nie, kontener nie uruchomi się.

Zmienne środowiskowe

Jeśli obraz kontenera określa zmienne środowiskowe (np. ustawione w Dockerfile), wówczas ustawienia **env** Kubernetes zastąpią je. Może to być przydatne do zmiany domyślnej konfiguracji kontenera.

Bardziej elastycznym sposobem przekazywania danych konfiguracyjnych do kontenerów jest użycie obiektów Kubernetes **ConfigMap** lub **Secret**.

Bezpieczeństwo kontenerów i podów

Uruchamianie kontenerów jako użytkownik inny niż *root*

Oto przykład specyfikacji kontenera, która każe Kubernetes uruchomić kontener jako konkretnego użytkownika:

```
containers:
- name: demo
  securityContext:
     runAsUser: 1000
```

Wartością **runAsUser** jest **UID** (numeryczny identyfikator użytkownika). W wielu systemach Linux identyfikator UID 1000 jest przypisany do pierwszego użytkownika, innego niż *root*, utworzonego w systemie. Jeśli zostanie określony UID za pomocą **runAsUser**, zastąpi on dowolnego użytkownika skonfigurowanego w obrazie kontenera.
Jeśli w manifeście lub obrazie nie zostanie określony żaden użytkownik, kontener będzie działał jako *root.*

Blokowanie kontenerów z uprawnieniami root

Kubernetes pozwala zablokować uruchamianie kontenerów, gdyby działały one jako użytkownik root.

```
containers:
- name: demo
  securityContext:
    runAsNonRoot: true
```

Ustawianie systemu plików tylko do odczytu

readOnlyRootFilesystem uniemożliwia zapisy kontenera we własnym systemie plików. Wiele kontenerów nie musi zapisywać niczego we własnym systemie plików, więc to ustawienie nie będzie im przeszkadzało.

```
containers:
- name: demo
  securityContext:
    readOnlyRootFilesystem: true
```

Wyłączanie eskalacji uprawnień

Zwykle pliki binarne systemu Linux działają z tymi samymi uprawnieniami, co użytkownik, który je wykonuje. Istnieje jednak wyjątek: pliki binarne korzystające z mechanizmu **setuid** mogą tymczasowo uzyskać uprawnienia użytkownika, który jest właścicielem pliku binarnego (zwykle *root*).

Jest to potencjalny problem występujący w kontenerach, ponieważ nawet jeśli kontener działa jako zwykły użytkownik (np. UID 1000) i zawiera plik binarny **setuid**, ten plik binarny może domyślnie uzyskać uprawnienia *root*.

Aby temu zapobiec:

```
containers:
- name: demo
  securityContext:
    allowPrivilegeEscalation: false
```

Mechanizm właściwości (ang. capabilities)

Mechanizm właściwości systemu Linux definiuje rzeczy, które program może zrobić, takie jak:

- ładowanie modułów jądra,
- wykonywanie bezpośrednich operacji sieciowych,
- uzyskiwanie dostępu do urządzeń systemowych,
  itp.

Domyślny zestaw możliwości kontenerów Docker jest dość hojny. Z drugiej strony, zasada najmniejszego uprzywilejowania mówi, że kontener nie powinien mieć możliwości, których nie potrzebuje. Konteksty bezpieczeństwa Kubernetes pozwalają usunąć dowolne właściwości z zestawu domyślnego i dodawać te, które są potrzebne.

```
containers:
- name: demo
  securityContext:
    capabilities:
      drop: ["CHOWN", "NET_RAW", "SETPCAP"]
      add: ["NET_ADMIN"]
```

Aby zapewnić maksymalne bezpieczeństwo, należy usunąć dla każdego kontenera wszystkie właściwości i dodać tylko określone, jeśli są potrzebne:

```
containers:
- name: demo
  securityContext:
    capabilities:
      drop: ["all"]
      add: ["NET_ADMIN"]
```

Po usunięciu właściwości na poziomie kontenera nie można jej odzyskać, nawet w przypadku *roota*.

Konteksty bezpieczeństwa Poda

Niektóre ustawienia kontekstu zabezpieczeń można też ustawić na poziomie Poda.

```
apiVersion: v1
kind: Pod
spec:
  securityContext:
    runAsNonRoot: false
    allowPrivilegeEscalation: false
```

Zamiast określać wszystkie ustawienia zabezpieczeń dla każdego kontenera lub Poda, można je określić na poziomie klastra za pomocą zasobu **PodSecurityPolicy** (o ile ma się dostęp – **RBAC**).

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
```

Woluminy

Woluminy emptyDir

To efemeryczna pamięć, na początku pusta, która przechowuje dane w węźle (w RAM lub na dysku węzła). Utrzymuje się tak długo, jak długo Pod działa w tym węźle. Przydatny do udostępniania plików między kontenerami w Podzie.

```
kind: Pod
spec:
  volumes:
  - name: cache-volume
    emptyDir: {}
    containers:
    - name: demo
      volumeMounts:
      - mountPath: /cache
        name: cache-volume
```

Uwaga:

Zachowaj ostrożność, zapisując w udostępnionych woluminach. Kubernetes nie wymusza żadnego blokowania zapisów na dysku. Jeśli dwa kontenery spróbują jednocześnie zapisać w tym samym pliku, może to spowodować uszkodzenie danych.

Woluminy trwałe – PersistentVolumes & PersistentVolumeClaim

Najbardziej elastycznym sposobem korzystania z **PersistentVolume** w Kubernetes jest utworzenie obiektu **PersistentVolumeClaim**. Reprezentuje on żądanie do utworzenia **PersistentVolume** o określonym typie oraz wielkości, np. woluminu o pojemności 10 GiB z szybką pamięcią do odczytu i zapisu.

W swoim klastrze możesz utworzyć pulę PersistentVolumes.

```
kind: PersistentVolume
metadata:
  name: foo-pv
spec:
  storageClassName: ""
  claimRef:
    name: foo-pvc
    namespace: foo
```

```
kind: PersistentVolumeClaim
metadata:
  name: foo-pvc
  namespace: foo
spec:
  storageClassName: "" # Empty string must be explicitly set otherwise default StorageClass will be set
  volumeName: foo-pv
```

```
volumes:
- name: data-volume
  persistentVolumeClaim:
    claimName: foo-pvc
```

Alternatywnie można skonfigurować dynamiczne przydzielanie.

Rozdział 9. Zarządzanie Podami

Etykiety

Etykiety to pary klucz-wartość, które są dołączone do obiektów Kubernetes. Etykiety same w sobie określają atrybuty identyfikujące obiekty mające znaczenie tylko dla użytkowników.

```
spec:
  metadata:
    labels:
      app: demo
```

Selektory

Wyrażenie pasujące do etykiety lub zestawu etykiet. Jest to sposób na określenie grupy zasobów według ich etykiet.

```
spec:
  selector:
    app: demo
```

Dla zasobu Serwis dostępne są tylko selektory równości. Natomiast dla zapytań z użyciem kubectl lub bardziej wyrafinowanych zasobów, takich jak Deployment, istnieją inne opcje.

```
kubectl get pods -l app=demo,environment=production
```

```
kubectl get pods -l app!=demo
```

```
kubectl get pods -l 'environment in (staging, production)'
```

```
kubectl get pods -l 'environment notin (production)'
```

W formacie YAML:

```
selector:
  matchExpressions:
    - {key: environment, operator: NotIn, values: [staging, production]}
```

Etykiety vs adnotacje

Obie reprezentują zestaw par klucz-wartość, które określają metadane dla zasobów.
Etykiety identyfikują zasoby. Służą do wybierania grup powiązanych zasobów.
Adnotacje są używane przez narzędzia lub usługi poza Kubernetes.

Ponieważ etykiety są często używane w wewnętrznych zapytaniach, które mają krytyczne znaczenie dla Kubernetes, istnieją pewne dość ścisłe ograniczenia dotyczące prawidłowości etykiet.

Składnia etykiet

Prawidłowe klucze etykiet:

- składają się z dwóch segmentów: opcjonalnego **prefiksu** i **nazwy**
- segmenty oddzielone są ukośnikiem **/**
- **nazwa** (wymagana):
    - musi składać się z maksymalnie 63 znaków
    - musi zaczynać i kończyć się znakiem alfanumerycznym **[a-z0-9A-Z]**
    - może zawierać myślniki **-** podkreślniki **_** kropki **.** i znaki alfanumeryczne **[a-z0-9A-Z]**
- **prefiks** (opcjonalny) musi być subdomeną DNS, czyli serią etykiet DNS:
    - nie dłuższych niż 253 znaki
    - oddzielonych kropkami **.**
    - po których następuje ukośnik **/**

Koligacje

Koligacje węzłów

Koligacje węzłów **affinity** są używane, aby preferencyjnie uruchamiać Pody w określonych węzłach.

Reguły te mają zastosowanie podczas planowania, ale nie podczas wykonywania. Część `IgnoredDuringExecution` oznacza to, że gdy Pod zostanie przypisany do określonego węzła spełniającego koligację, pozostanie tam nawet, jeśli etykieta Noda zmieni się podczas w trakcie.

```
kind: Pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        # lub
      preferredDuringSchedulingIgnoredDuringExecution:
```

Koligacja twarda – *requiredDuringSchedulingIgnoredDuringExecution*

Koligacje twarde oznaczają, że Pod się nie uruchomi, dopóki nie zostaną spełnione warunki.

Koligacja miękka – *preferredDuringSchedulingIgnoredDuringExecution*

Koligacje miękkie są wyrażane w sposób, że każdej regule przypisuje się wagę liczbową od 1 do 100, a Pod może się uruchomić mimo niespełnionych warunków.

Koligacje Podów i antykoligacje

Czasami są takie pary Podów, które działają lepiej, gdy są razem w tym samym węźle, np. serwer WWW i baza danych. I odwrotnie, czasami lepiej, aby Pody unikały się nawzajem, np. w celu utrzymana zrównoważonych obciążeń.

Podobnie jak koligacje węzłów, koligacje Podów są wyrażane jako zbiór reguł: albo twarde wymagania, albo miękkie preferencje z zestawem wag.

Koligacja:

```
kind: Pod
spec:
  affinity:
    podAffinity:
      prefferedDuringSchedulingIgnoredDuringExecution:
```

Antykoligacja:

```
kind: Pod
spec:
  affinity:
    podAntiAffinity:
      prefferedDuringSchedulingIgnoredDuringExecution:
```

Kontrolery Podów

ReplicaSet / Deployment

Zarządza grupą replik konkretnego Poda. Działa w sposób ciągły, aby upewnić się, że zawsze jest określona liczba replik:

- uruchamia nowe – jeśli jest ich za mało,
- kończy ich pracę – jeśli jest zbyt wiele.

DaemonSet

Termin demon tradycyjnie odnosi się do długotrwałych procesów działających w tle na serwerze, które obsługują takie funkcje jak rejestrowanie, więc analogicznie **DaemonSet** uruchamia kontener demona na każdym węźle w klastrze.

Jeśli korzystasz z aplikacji, w której utrzymanie określonej liczby replik jest ważniejsze niż to, w którym węźle działają Pody, użyj zamiast tego obiektu Deployment.

StatefulSet

StatefulSet dodaje możliwość uruchamiania i zatrzymywania Podów w kreślonej numerowanej sekwencji i być w stanie rozpoznać je za pomocą numeru.

Jeśli np. utworzysz **StatefulSet** o nazwie *redis*, pierwszy uruchomiony Pod będzie miał nazwę *redis-0*, a przed uruchomieniem następnego, *redis-1*, Kubernetes poczeka, aż poprzedni Pod będzie gotowy.
Po zakończeniu StatefulSet repliki zostaną zamknięte w odwrotnej kolejności, czekając na zakończenie każdego Poda przed przejściem do następnego.

Aby można było adresować każdy z Podów za pomocą przewidywalnej nazwy DNS, takiej jak *redis-1*, musisz także utworzyć **Serwis** z polem **cluserIP** o wartości **None** (znany jako **usługa headless**).

ControlerJob

Podczas gdy Deployment uruchamia określoną liczbę Podów i restartuje je w sposób ciągły, Job uruchamia Pod tylko określoną liczbę razy. Następnie kończy pracę.

Istnieją dwa pola sterujące wykonywaniem zadania:

- **completions**

Ustala, ile razy określony Pod musi zadziałać, zanim zadanie zostanie uznane za ukończone.
Wartość completions określa tylko pozytywne zakończenia działań.

- **parallelism**

Określa, ile Podów powinno działać jednocześnie.

CronJob

Dwa ważne pola, występujące w manifeście CronJob to:

- **schedule**
- **jobTemplate**

Horizontal Pod Autoscaler

Horizontal Pod Autoscaler (**HPA**) obserwuje konkretny Deployment, stale monitorując dane, aby sprawdzić, czy konieczne jest zwiększenie lub zmniejszenie liczby replik.

Interesujące pola to:

- **scaleTargetRef** – określa Deployment do skalowania,
- **minReplicas** i **spec.maxReplicas** – określają granice skalowania,
- **metrics** – określające metryki, które będą używane do skalowania

PodPreset

PodPreset to rodzaj obiektu zwany **kontrolerem dostępu** (***admission controller***). Kontrolery dostępu obserwują budowanie Podów i podejmują pewne działania, gdy tworzone są Pody pasujące do ich selektora. Przykładowo niektóre kontrolery dostępu mogą blokować tworzenie Poda, jeśli narusza to politykę. Natomiast inne wprowadzają dodatkową konfigurację do Poda.

Jeśli Pod zostanie zmodyfikowany przez PodPreset, zobaczysz taką adnotację:

podpreset.admission.kubernetes.io/podpreset-add-cache: "<resource version>"

PodPreset nie może być używane do zastępowania własnej konfiguracji Poda, a jedynie do uzupełniania ustawień, których sam Pod nie określa.

Operatory i niestandardowe definicje zasobów – Operators & CustomResourceDefinitions (CRD)

Kubernetes umożliwia tworzenie własnych nowych typów obiektów za pomocą mechanizmu **CRD** w przypadku aplikacji, które wymagają bardziej skomplikowanego zarządzania, niż mogą zapewnić standardowe obiekty Kubernetes. Aby to zrobić, musisz napisać program, który komunikuje się z interfejsem API Kubernetes. Taki program nazywany jest **operatorem** (***operator***).

Ingress

Podczas gdy Serwis kieruje ruchem wewnętrznym w klastrze, **Ingress** udostępnia trasy **HTTP** i **HTTPS** z zewnątrz klastra do **serwisów** wewnątrz klastra. Sterowanie ruchem odbywa się za pomocą reguł zdefiniowanych na zasobie Ingress.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
spec:
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Ingress nie eksponuje dowolnych portów ani protokołów. Eksponowanie usług innych niż HTTP i HTTPS do Internetu zazwyczaj wykorzystuje usługę typu Service.Type=NodePort lub Service.Type=LoadBalancer.

Kontroler Ingress

Aby zasób Ingress mógł działać, w klastrze musi być uruchomiony **kontroler Ingress**. W zależności od miejsca, w którym działają klastry, używany kontroler może być inny.

W przeciwieństwie do innych typów kontrolerów, które działają jako część binarnego *kube-controller-manager*, kontrolery Ingress nie są uruchamiane automatycznie wraz z klastrem.

Kubernetes jako projekt wspiera i utrzymuje kontrolery ingress:

- AWS Load Balancer Controller
- GLBC (GCE L7 load balancer controller)
- NGINX Ingress Controller

choć dostępnych jest znacznie więcej implementacji *third party*.

Zarządzanie połączeniami TLS za pomocą Ingress

Ingress może obsługiwać bezpieczne połączenia za pomocą **protokołu TLS**, podając **Secret**, który zawiera klucz prywatny TLS (tls.key) i certyfikat (tls.crt).

Ingress obsługuje tylko jeden port TLS (**443**), i zakłada zakończenie TLS w punkcie wejściowym, czyli ruch do serwisu i jego Podów jest w nieszyfrowany.

Wiele różnych hostów w tej samej domenie mogą współużytkować certyfikat TLS, a pojedynczy Ingress może zarządzać tymi połączeniami na tym samym IP i porcie.

Uwaga:

Jeśli w klastrze uruchomisz **cert-manager**, automatycznie zostaną wykryte zasoby **Ingress TLS**, które nie mają certyfikatu. W wyniku tego zażąda on od określonego dostawcy (np. *LetsEncrypt*) jednego certyfikatu.

Reguły Ingress – *Ingress rules*

Każda reguła HTTP zawiera następujące informacje:

- **opcjonalny host**

Jeśli nie podano hosta, reguła dotyczy całego przychodzącego ruchu HTTP przez podany adres IP.
Jeśli podany jest host (np. foo.bar.com), reguły odnoszą się do tego hosta.

- **lista ścieżek**

Na przykład *„/testpath”*. Każda ścieżka ma przypisany **backend**. Zarówno host jak i ścieżka muszą być zgodne z treścią przychodzącego żądania zanim **load balancer** skieruje ruch do **serwisu**, do którego się odwołuje.

Każda ścieżka w Ingress musi posiadać odpowiadający jej typ ścieżki. Istnieją trzy obsługiwane typy ścieżek:

- ImplementationSpecific

Dopasowanie jest zależne od **IngressClass**.

- Exact (case sensitive)

Dokładne dopasowanie ścieżki.

- Prefix (case sensitive)

Dopasowanie odbywa się na zasadzie porównania elementów ścieżki po elemencie, które są oddzielonego **/**. Ignoruje końcowy ukośnik.

Kind	Path(s)	Request path(s)	Matches?
Prefix	/	(all paths)	Yes
Exact	/foo	/foo	Yes
Exact	/foo	/foo/	No
Prefix	/aaa/bbb	/aaa/bbb/	Yes
Prefix	/aaa/bbb	/aaa/bbb/ccc	Yes

- **backend**

W przypadku gdy **backend** wskazuje na **serwis**, jest on kombinacją nazwy usługi (service.name) i portu (service.port.name lub service.port.number). Można też używać **backendu** do niestandardowych zasobów CRD. Żądania

HTTP i HTTPS do Ingress, które pasują do hosta i ścieżki reguły są wysyłane do backendu z listy. Domyślny backend (**defaultBackend**) jest często konfigurowany, aby obsłużyć wszystkie żądania, które nie pasują do ścieżki w specyfikacji.

Istio

Istio to usługa **mesh**. Bardzo przydatna, gdy wiele aplikacji i usług komunikują się ze sobą.
Obsługuje routing i szyfrowanie ruchu sieciowego między usługami oraz dodaje ważne funkcje, takie jak metryki, dzienniki i równoważenie obciążenia.

Envoy

Envoy jest wysokowydajnym **rozproszonym proxy** zaprojektowanym dla pojedynczych usług i aplikacji, ale może być również używany jako część architektury usług **mesh**. Jest stosowany do równoważenia obciążenia w bardziej wyrafinowanego niż standardowe usługi równoważenia obciążenia w chmurze.

Rozdział 10. Konfiguracja i obiekty Secret

ConfigMap

Tworzenie ConfigMap

Tak wygląda manifest dla obiektu ConfigMap:

```
kind: ConfigMap
data:
  config.yaml: |
    autoSaveInterval: 60
    protocols:
    - http
    - https
```

Łatwiejszym sposobem jest wykorzystanie polecenia kubectl, tworząc ConfigMap bezpośrednio z pliku YAML:

```
$ kubectl create configmap demo-config --namespace=demo --from-file=config.yaml
```

Ustawianie zmiennych środowiskowych z obiektu ConfigMap – env-configMapKeyRef

```
spec:
  containers:
  - name: demo
    env:
    - name: GREETING
      valueFrom:
        configMapKeyRef:
          name: demo-config
          key: greeting
```

Ustawianie środowiska za pomocą ConfigMap – envFrom-configMapRef

```
spec:
  containers:
  - name: demo
    envFrom:
    - configMapRef:
      name: demo-config
```

env ma pierwszeństwo przed **envFrom** w przypadku takiej samej nazwy.

Używanie zmiennych środowiskowych w argumentach poleceń – $()

```
spec:
  containers:
  - name: demo
    args:
    - "-greeting"
    - "$(GREETING)"
    env:
    - name: GREETING
      valueFrom:
        configMapKeyRef:
          name: demo-config
          key: greeting
```

Tworzenie plików konfiguracji z ConfigMaps – volumes-configMap

```
spec:
  containers:
  - name: demo
    volumeMounts:
    - name: demo-config-volume
      mountPath: /config/
      readOnly: true
  volumes:
  - name: demo-config-volume
    configMap:
      name: demo-config
      items:
      - key: config
        path: demo.yaml
```

Aktualizacja Podów po zmianie konfiguracji

Jeśli korzystasz z wykresu Helm, dodaj tę adnotację do specyfikacji Deployment:

```
checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml") . | sha256sum }}
```

Po wykonaniu polecenia `helm upgrade` Helm wykryje, że specyfikacja Deployment uległa zmianie i ponownie uruchomi wszystkie Pody.

Secret

Używanie obiektów Secret jako zmiennych środowiskowych – env-valueFrom.secretKeyRef

```
spec:
  containers:
  - name: demo
    env:
    - name: MAGIC_WORD
      valueFrom:
        secretKeyRef:
          name: demo-secret
          key: magicWord
```

Zapisywanie obiektów Secret do plików – volumes-secret

```
spec:
  containers:
  - name: demo
    volumeMounts:
    - name: demo-secret-volume
      mountPath: "/secrets/"
      readOnly: true
  volumes:
  - name: demo-secret-volume
    secret:
      secretName: demo-secret
```

Pole `mountPath` ma wartość */secrets*. Kubernetes utworzy w tym katalogu **jeden plik dla każdej pary klucz-wartość** zdefiniowanej w obiekcie **Secret**.

Dostęp do obiektów Secret

Jest to kontrolowane przez mechanizm kontroli dostępu Kubernetes, RBAC.

Szyfrowanie w stanie spoczynku – *encryption at rest*

Tajne dane w bazie danych **etcd** są przechowywane zaszyfrowane na dysku i nieczytelne nawet dla osób, które mają bezpośredni dostęp do bazy danych. Tylko serwer API Kubernetes ma klucz do odszyfrowania tych danych.

Możesz sprawdzić, czy szyfrowanie w stanie spoczynku jest włączone w klastrze:

```
$ kubectl describe pod -n kube-system -l component=kube-apiserver | grep encryption
```

resp: `--experimental-encryption-provider-config=...`

Trwałe przechowywanie obiektów Secret

Za pomocą adnotacji specyficznej dla narzędzia Helm możesz zapobiec usunięciu zasobu:

```
kind: Secret
metadata:
  annotations:
    "helm.sh/resource-policy": keep
```

Strategie zarządzania obiektami Secret

Bez względu na to, jakie narzędzie lub jaką strategię wybierzesz do zarządzania tajnymi danymi w swoich aplikacjach, będziesz potrzebować odpowiedzi na następujące pytania.

- Gdzie przechowujesz obiekty Secret, zachowując wysoką niezawodność?
- W jaki sposób udostępniasz obiekty Secret działającym aplikacjom?
- Co musi się stać z działającymi aplikacjami, gdy zmieniasz Secret?

Możliwe są następujące opcje:

- szyfrowanie Secret w systemach kontroli wersji,
- zdalne przechowywanie Secret,
- dedykowane narzędzie do zarządzania obiektami Secret.

Szyfrowanie Secret w systemach kontroli wersji

Przechowywanie obiektów **Secret** w postaci zaszyfrowanej bezpośrednio w kodzie, w repozytoriach kontroli wersji i odszyfrowanie ich w czasie wdrażania. Przykładowe narzędzie de/szyfrowania: **Sops, helm-secrets**.

\+ Pozwala przeglądać i śledzić zmiany w nich wykonywane.

\+ Obiekty Secret są wysoce niezawodne tak długo, jak repozytoria kontroli wersji.

\+ Strategia ta jest łatwa do wdrożenia i nie ma żadnych zależności z wyjątkiem klucza i narzędzia de/szyfrowania.

– Jeśli ten sam klucz jest używany przez wiele aplikacji, wszystkie potrzebują jego kopii w kodzie źródłowym. Przy aktualizacji klucza musisz upewnić się, że zmieniłeś wszystkie jego wystąpienia.

– **Poważne ryzyko przypadkowego przekazania obiektów w postaci zwykłego tekstu do systemu kontroli wersji**.

Zdalne przechowywanie Secret

Przechowywanie ich w plikach w zdalnym bezpiecznym magazynie plików (np. **AWS S3**). Podczas wdrażania aplikacji pliki zostaną pobrane, odszyfrowane i dostarczone do aplikacji. Jest to podobne rozwiązanie do opcji szyfrowania Secret w systemie kontroli wersji; z tym wyjątkiem, że obiekty Secret są przechowywane centralnie.

\+ To rozwiązuje problem duplikowania Secret w wielu repozytoriach kodu.

– **Potrzeba trochę dodatkowej inżynierii i koordynacji, aby ściągnąć odpowiedni plik Secret w czasie wdrażania.**

– W związku z tym, że Secret nie znajdują się w systemie kontroli wersji, potrzebny jest proces do obsługi aktualizacji tych obiektów w uporządkowany sposób — najlepiej z logowaniem oraz pewnego rodzaju równoważnej procedury kontroli zmian

Dedykowane narzędzie do zarządzania obiektami Secret

Np. **Vault Hashicorp** i **AWS Secrets Manager**. Narzędzia te obsługują bezpieczne przechowywanie wszystkich obiektów Secret aplikacji w jednym centralnym miejscu w wysoce niezawodny sposób.

W jaki sposób aplikacje pobierają dane z narzędzia do zarządzania Secret?

Jednym z powszechnych pobierania sekretów przez aplikacje jest użycie konta usługi z dostępem tylko do odczytu, dzięki czemu każda aplikacja może tylko odczytywać potrzebne mu Secret.

\+ Kontrolują, którzy użytkownicy i jakie konta usług mają uprawnienia do dodawania, usuwania, zmiany lub przeglądania Secret.

\+ Wszystkie działania są kontrolowane i weryfikowane, co ułatwia analizę naruszeń bezpieczeństwa i wykazanie zgodności z przepisami.

\+ Niektóre z tych narzędzi zapewniają także możliwość automatycznej zmiany Secret w regularnych odstępach czasu.

– Dodaje znaczną złożoność do infrastruktury. Wymaga konfigurowania i uruchamiania magazynu Secret.

– **Konieczne jest dodanie narzędzi lub oprogramowania pośredniego do każdej aplikacji i usługi wykorzystującej Secret.** Aplikacje można ponownie przebudować lub przeprojektować w celu uzyskania bezpośredniego dostępu do magazynu obiektów Secret, ale może to być droższe i bardziej czasochłonne niż zwykłe dodanie warstwy.

Rozdział 11. Bezpieczeństwo i kopia zapasowa + RBAC

Zarządzanie dostępem przez klaster

Jeśli żaden zespół nie powinien mieć dostępu do innego zespołu i procesu wdrażania (np. dev/prod), każdy zespół może mieć własny dedykowany klaster, ale nie powinien posiadać poświadczeń w klastrach drugiego zespołu.

Jest to z pewnością najbezpieczniejsze podejście, ale posiadanie dodatkowych klastrów generuje dodatkowe problemy. Każdy z nich musi być aktualizowany i monitorowany. Wiele małych klastrów działa również mniej wydajnie niż większe klastry.

Kontrola dostępu oparta na rolach (RBAC)

RBAC ma na celu przyznawanie określonych uprawnień konkretnym użytkownikom lub kontom usług, które są kontami użytkowników powiązanymi z automatycznymi systemami.

Bez RBAC każdy, kto ma dostęp do klastra, jest w stanie zrobić wszystko.

Uwaga:

```
$ kubectl describe pod -n kube-system -l component=kube-apiserver
```

resp:

```
...
Containers:
  kube-apiserver:
    Command:
      kube-apiserver
      ...
      --authorization-mode=Node,RBAC
```

Jeśli **`--authorization-mode`** nie zawiera RBAC, to RBAC nie jest włączony dla Twojego klastra.

Role

Wszyscy użytkownicy mogą potencjalnie mieć różne zestawy uprawnień. Są one regulowane przez role Kubernetes. Rola opisuje określony zestaw uprawnień. Można zdefiniować role na poziomie przestrzeni nazw (za pomocą obiektu ***Role***) lub w całym klastrze (za pomocą obiektu ***ClusterRole***).

Aby zobaczyć, jakie uprawnienia ma dana rola, użyj polecenia:

```
$ kubectl describe clusterrole/edit
```

Predefiniowane role

Wstępnie zdefiniowane role **cluster-admin**, **admin**, **edit** oraz **view**, prawdopodobnie spełnią większość wymagań.

- **cluster-admin**

Umożliwia dostęp super-usera, przeznaczony do wykonania dowolnej akcji na dowolnym zasobie. W przypadku użycia w **ClusterRoleBinding** daje pełną kontrolę nad każdym zasobem w klastrze i we wszystkich przestrzeniach nazw. Gdy użyte w **RoleBinding**, daje pełną kontrolę nad każdym zasobem w przestrzeni nazw wiązania roli, włączając w to samą przestrzeń nazw.

- **admin**

Umożliwia dostęp administratora, przeznaczony do przyznawania ról w przestrzeni nazw za pomocą **RoleBinding**. Jeśli użyte w RoleBinding, pozwala na odczyt/zapis do większości zasobów w przestrzeni nazw, w tym możliwość tworzenia ról i powiązań ról w przestrzeni nazw. Nie pozwala na zapis do zasobów lub do samej przestrzeni nazw.

- **edit**

Umożliwia dostęp do odczytu i zapisu większości obiektów w przestrzeni nazw. Nie pozwala na przeglądanie lub modyfikację ról lub powiązań ról.

- **view**

Pozwala na dostęp tylko do odczytu, aby zobaczyć większość obiektów w przestrzeni nazw. Nie pozwala na przeglądanie ról i powiązań ról. Nie pozwala na przeglądanie sekretów.

Tworzenie własnych roli

Przykładową rolą, której brakuje, jest rola do odczytu sekretów (np. aby móc wykonać `$ helm list`).
Jej definicja może wyglądać np.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
  rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "watch", "list"]
```

Wiązanie ról z użytkownikami

Użytkownika można skojarzyć z daną rolą za pomocą powiązania roli (**role binding**). Można utworzyć obiekt **RoleBinding**, który odnosi się do przestrzeni nazw, lub **ClusterRoleBinding**, który działa na poziomie klastra.

```
$ kubectl create rolebinding sam-edit
    --clusterrole edit \
    --user sam \
    --namespace foo
```

W Kubernetes uprawnienia są addytywne; użytkownicy zaczynają bez uprawnień i można je dodawać za pomocą ***Role*** i ***RoleBinding***. Nie możesz ograniczyć uprawnień osobie, która już je ma.

Ochrona dostępu do cluster-admin

Bądź bardzo ostrożny, kiedy przydzielasz rolę **cluster-admin**. Może zrobić wszystko. Nigdy nie przydzielaj tej roli użytkownikom, którzy nie są operatorami klastrów, a zwłaszcza nie przydzielaj ich do kont serwisów dla aplikacji, które mogą mieć dostęp do Internetu.

Aplikacje i wdrażanie

Aplikacje działające w Kubernetes zwykle nie potrzebują żadnych uprawnień RBAC. **Domyślnie** wszystkie **Pody** będą działać jako **konto** usługi **default** w obszarze nazw, z którym nie są związane żadne role.

Do **wdrażania aplikacji** w klastrze nadaje się rola **edit**. Użytkownicy z rolą edit mogą tworzyć i niszczyć zasoby w przestrzeni nazw, ale nie mogą tworzyć nowych ról ani udzielać uprawnień innym użytkownikom.

Skanowanie bezpieczeństwa

Jeśli w klastrze jest oprogramowanie innych firm, warto sprawdzić, czy nie występują problemy związane z bezpieczeństwem.

Niektóre skanery bezpieczeństwa:

- Clair

Statycznie analizuje obrazy kontenerów, zanim zostaną faktycznie uruchomione, aby sprawdzić, czy zawierają oprogramowanie lub wersje, które nie są uważane za bezpieczne.

- Aqua

Umożliwiająca skanowanie kontenerów w poszukiwaniu luk, złośliwego oprogramowania i podejrzanych działań.

Oferuje również bezpłatne narzędzie o nazwie *MicroScanner*, które można dodawać do obrazów kontenerów w celu skanowania zainstalowanych pakietów pod kątem znanych luk.

- Anchore Engine

Skanuje obrazy kontenerów pod kątem znanych słabych punktów oraz identyfikuje wszystkie składniki, które zawiera kontener — w tym bibliotek, plików konfiguracyjnych i uprawnień do plików. Można to wykorzystać do weryfikacji kontenerów pod kątem zasad zdefiniowanych przez użytkownika.

Kopie zapasowe

Jeśli posiadasz własne węzły master, jesteś odpowiedzialny za zarządzanie **etcd**, **replikację** i tworzenie **kopii zapasowych**.

Oprócz awarii **etcd** istnieje także kwestia zapisu stanu indywidualnych zasobów, np. Deployment.

W praktyce nie wszystko, co masz w repozytorium, działa teraz w klastrze.

Przykładowym darmowym narzędziem do tworzenia i przywracania backupów jest **Velero**.

Uwaga:

Automatycznie i regularnie twórz kopie zapasowe stanu klastra i trwałych danych, przynajmniej co noc.

Uruchamiaj test przywracania co najmniej raz w miesiącu.

Powinieneś napisać szczegółową procedurę opisującą sposób przywracania danych z kopii zapasowych i upewnić się, że wszyscy pracownicy wiedzą, gdzie znaleźć ten dokument.

Twoja procedura powinna być tak jasna i precyzyjna, że będzie mógł ją przeprowadzić ktoś, kto nie zna narzędzia lub nawet Kubernetes.

Monitorowanie statusu klastra

Stan control plane

```
$ kubectl get componentstatuses
```
lub
```
$ kubectl get cs
```

res:
```
NAME                 STATUS  MESSAGE             ERROR
controller-manager   Healthy                     ok
scheduler            Healthy                     ok
etcd-0               Healthy                     {"health": "true"}
```

Stan węzłów

```
$ kubectl get nodes
```

res:
```
NAME                 STATUS   ROLES          AGE   VERSION
docker-for-desktop   Ready    master         5d    v1.10.0
worker-on-desktop    Ready    <none>         5d    v1.10.0
```

W przypadku usług zarządzanych nie ma bezpośredniego dostępu do węzła master.

`<none>` wskazuje na worker.

Obciążenia na Podach

```
$ kubectl get pods --all-namespaces
```

res:
```
NAMESPACE     NAME                            READY   STATUS             RESTARTS   AGE
cert-manager  cert-manager-cert-manager-55    1/1     Running            1          10d
pa-test       permissions-auditor-15281892    0/1     CrashLoopBackOff   1720       6d
```

Wykorzystanie procesora i pamięci

```
$ kubectl top nodes
```

res:
```
NAME                          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
gke-k8s-cluster-1-n1-...dwtv  155m         8%     3449Mi          61%
gke-k8s-cluster-1-n1-...67ch  580m         30%    3172Mi          56%
```

```
$ kubectl top pods
```

res:
```
NAME                                CPU(cores)   MEMORY(bytes)
event-exporter-v0.1.9-85bb4fd64d-2zjng  0m       27Mi
fluentd-gcp-scaler-7c5db745fc-h7ntr     10m      27Mi
fluentd-gcp-v3.0.0-5m627                11m      171Mi
```

Pulpit Kubernetes – *Kubernetes Dashboard*

Kubernetes Dashboard to przeglądarkowy interfejs użytkownika dla klastrów Kubernetes.

Uwaga:

Ponieważ pulpit udostępnia wiele informacji o klastrze i obciążeniach, bardzo ważne jest, aby odpowiednio go zabezpieczyć i nigdy nie udostępniać publicznie w Internecie. Musisz więc ściśle kontrolować dostęp do pulpitu. Jeśli nie musisz uruchamiać pulpitu Kubernetes (np. masz już konsolę na **OpenShift**), nie uruchamiaj go.

node-problem-detector

Dodatek node-problem-detector może wykrywać i raportować kilka rodzajów problemów na poziomie węzłów. Mogą to być problemy ze sprzętem, takie jak m.in.:

- błędy procesora lub pamięci,
- uszkodzenie systemu plików,
- błędy związane z środowiskiem wykonawczym kontenera.

Rozdział 12. Wdrażanie aplikacji Kubernetes – Helm

Helm – menadżer pakietów Kubernetes

Narzędzie **helm** udostępnia wiersz poleceń, za pomocą którego możesz **instalować** i **konfigurować aplikacje** (własne lub dowolne inne), a także tworzyć pakiety zwane **wykresami (chart)** Helm, które całkowicie określają zasoby potrzebne do uruchomienia aplikacji, jej zależności oraz konfigurowalne ustawienia.

Chart nie zawiera samego obrazu kontenera. Zamiast tego zawiera metadane dotyczące tego, gdzie można znaleźć obraz i jak go wdrożyć.

```
$ helm install demo
```

```
$ helm update --install demo
```

https://helm.sh/docs/helm/helm_upgrade/

Co znajduje się w wykresie narzędzia Helm? – struktura Helm Chart

Każdy wykres Helm (**Helm chart**) ma standardową strukturę.

wordpress/	# [DIR]	Chart jest zawarty w katalogu o tej samej nazwie
| **Chart.yaml**	#	Zawiera informacje o wykresie (chart)
| **values.yaml**	#	Domyślne wartości konfiguracji wykresu
| values.schema.json	# [OPCJA]	Schemat JSON narzucający strukturę pliku values.ymal
| requirements.yaml	# [OPCJA]	wymagane zależności od innych wykresów
| LICENSE	# [OPCJA]	Informacje o licencji wykresy (plain text)
| README.md	# [OPCJA]	Plik README w formacie Markdown
| **charts/**	# [DIR]	Wykresy zależne od głównego wykresu (Chart.yaml)
| **crds/**	# [DIR]	Definicje customowych zasobów
| **templates/**	# [DIR]	Szablony wypełniane wartościami generujące manifesty Kubernetes
| | NOTES.txt	# [OPCJA]	Krótka instrukcja obsługi (plain text)

Chart.yaml

Zawiera informacje o wykresie. W pliku możesz zdefiniować wiele opcjonalnych pól, w tym link do kodu źródłowego projektu.

Jedynymi **wymaganymi** informacjami są:

- **nazwa,**
- **wersja**.

Określanie zależności – requirements.yaml

Jeśli wykres opiera się na wartościach innych wykresów, należy określić zależności w pliku ***requirements.yaml***, aby Helm je pobrał.

```
dependencies:
- name: redis
  version: 1.2.3
```

Szablony – Helm, Go, Sprig – templates/, values.yaml

Szablony wykresów Helm chart są napisane w języku szablonów Go, z dodatkiem około 50 dodatkowych funkcji szablonów z biblioteki Sprig i kilku innych wyspecjalizowanych funkcji.

Wszystkie pliki szablonów przechowywane są w folderze **templates/**. Kiedy Helm renderuje wykresy, przepuszcza każdy plik z tego katalogu przez silnik szablonów.

Składnia GO

Szablony (**templates**) manifestów Kubernetes zawierają symbol zastępczy, który Helm zastąpi rzeczywistą wartością z pliku **values.yaml**.

Nawiasy klamrowe `{{ }}` są częścią składni szablonu Go.

Format szablonu Go jest bardzo rozbudowany i obsługuje:

- zastępowanie zmiennych (zmienne interpolacyjne),
- pętle,
- wyrażenia,
- instrukcje warunkowe,
- funkcje.

Funkcje Sprig

Przykładowe funkcje:

- String Functions: **trim**, **wrap**, **randAlpha**, **plural,**
  - String List Functions: **splitList**, **sortAlpha**,
- Integer Math Functions: **add**, **max**, **mul**,
  - Integer Slice Functions: **until**, **untilStep,**
- Float Math Functions: **addf**, **maxf**, **mulf**,
- Date Functions: **now**, **date**,
- Defaults Functions: **default**, **empty**, **coalesce**, **fromJson**, **toJson**, **toPrettyJson**, **toRawJson**, **ternary**,
- Encoding Functions: **b64enc**, **b64dec**,
- Lists and List Functions: **list**, **first**, **uniq**,
- Dictionaries and Dict Functions: **get**, **set**, **dict**, **hasKey**, **pluck**, **dig**, **deepCopy**,
- Type Conversion Functions: **atoi**, **int64**, **toString**,
- Path and Filepath Functions: **base**, **dir**, **ext**, **clean**, **isAbs**, **osBase**, **osDir**, **osExt**, **osClean**, **osIsAbs,**
- Flow Control Functions: **fail**.

Wartości – values.yaml

Zawiera ustawienia modyfikowalne przez użytkownika, które udostępnił autor wykresu.
Plik ten ma całkowicie wolny format YAML, bez predefiniowanego schematu.

Można się do niego odwołać poprzez: `{{ .Values.<klucze>.<z>.<pliku> }}`

Wartości dla szablonów są dostarczane na dwa sposoby:

- poprzez plik o nazwie **values.yaml** wewnątrz wykresu. Plik ten może zawierać wartości domyślne.
- poprzez plik YAML, który zawiera wartości. Może on być dostarczony w linii poleceń z użyciem `$ helm install --values=my-values.yaml`

Kiedy użytkownik dostarczy niestandardowe wartości, będą one zastępować wartości w pliku values.yaml.

Wartości tekstowe w szablonach

Aby automatycznie otoczyć daną wartość znakiem cudzysłowu, możesz wykorzystać funkcję **quote**:

```
name: {{.Values.MyName | quote }}
```

Wartości predefiniowane

Poniższe wartości są predefiniowane, dostępne dla każdego szablonu i nie można ich zastąpić. Tak jak w przypadku wszystkich wartości, w nazwach rozróżniana jest wielkość liter.

- **Release.Name** – Nazwa release (nie wykresu).
- **Release.Namespace** – Namespace, na który należy wydać (*release*) wykres.
- **Release.Service** – Usługa, która wykonała wydanie.
- **Release.IsUpgrade** – Wartość ustawiana na *true*, jeśli operacja jest aktualizacją (***update***) lub cofnięciem (***rollback***).
- **Release.IsInstall** – Wartość ustawiana na *true*, jeśli operacja jest instalacją (***install***).
- **Chart** – Zawartość pliku ***Chart.yaml*** (wersja wykresu jest dostępna jako Chart.Version).
- **Capabilities** – Obiekt zawiera informacje o wersjach Kubernetes {{ .Capabilities.KubeVersion }} i obsługiwanych wersjach Kubernetes API {{ .Capabilities.APIVersions.Has "batch/v1" }}.
- **Files** – Obiekt zawiera wszystkie niespecjalne pliki w wykresie (o ile nie są wykluczone ***.helmignore***). Dostęp do plików można uzyskać za pomocą {{ index .Files "file.name" }} lub {{ .Files.Get name }}. Można też dostęp do zawartości pliku jako **[]byte** używając {{ .Files.GetBytes }}.

Zakresy wartości

values.yaml może deklarować wartości dla wykresu najwyższego poziomu, jak również dla każdego z wykresów, które są zawarte w katalogu **charts/** tego wykresu.

```
dependencies:
  - name: apache
    version: 1.2.3
    repository: https://example.com/charts
  - name: mysql
    version: 3.2.1
    repository: https://another.example.com/charts
```

```
title: "My WordPress Site" # Sent to the WordPress template

mysql:
  max_connections: 100    # Sent to templates/MySQL
  password: "secret"

apache:
  port: 8080              # Passed to templates/Apache
```

Wykresy na wyższym poziomie mają dostęp do wszystkich zmiennych zdefiniowanych poniżej. Tak więc wykres WordPress może uzyskać dostęp do hasła MySQL jako **.**Values.**mysql**.password. Ale wykresy niższego poziomu nie mogą uzyskać dostępu do rzeczy w wykresach nadrzędnych, więc MySQL nie będzie w stanie uzyskać dostępu do właściwości title. Nie może również uzyskać dostępu do apache.port.

Wartości mają przestrzeń nazw, ale przestrzenie nazw są przycinane. Tak więc dla wykresu WordPress, może uzyskać dostęp do pola hasła MySQL jako **.Values.mysql**.password, ale dla wykresu MySQL zakres wartości został zmniejszony, a prefiks przestrzeni nazw usunięty, więc będzie widział pole hasła po prostu jako .**Values**.password.

Wartości globalne

```
title: "My WordPress Site" # Sent to the WordPress template

global:
  app: MyWordPress

mysql:
  […]
```

Dostęp do takiej globalnej wartości uzyskuje się poprzez {{ .Values.**global**.app}}

Biblioteki wykresów

Biblioteka wykresów jest typem wykresu Helm, który definiuje:

- prymitywy wykresów,
- definicje,

które mogą być współdzielone przez szablony Helm w innych wykresach, unikając powtórzeń.

Wdrażanie wykresów Helm

Komendy helm

- install

Polecenie `$ helm install` pozwala określić w wierszu polecenia dodatkowe pliki wartości, które zastąpią wartości domyślne w pliku values.yaml.

Aby określić **dodatkowe opcje** w poleceniu `$ helm install`, użyj opcji `--values`, tak jak poniżej:

```
$ helm install --name demo-staging --values=./k8s/demo/staging-values.yaml ./k8s/demo ...
```

Za pomocą flagi `--set` lub `--set-string` możesz także określić wartości, które podaje się bezpośrednio przy poleceniu `$ helm install`.

- inspect

Aby wyświetlić listę wartości, które można ustawić dla wykresu, uruchom:

```
$ helm inspect values stable/Prometheus
```

- upgrade

Aby zastosować zmiany w istniejącym obiekcie Deployment, uruchom następującą komendę:

```
$ helm upgrade demo-staging --values=./k8s/demo/staging-values.yaml ./k8s/demo
```

Flaga `--install` pozwala zainstalować **release**, jeśli nie istnieje taki o podanej nazwie.

- create

Tworzenie nowego wykresu:

```
$ helm create mychart
```

res: `Created mychart/`

- package

Po edycji wykresu, helm może go spakować do archiwum wykresów:

```
$ helm package mychart
```

res: `Archived mychart-0.1.tgz`

- lint

Można również użyć **helm** do pomocy w znalezieniu problemów z formatowaniem wykresu lub informacjami:

```
$ helm lint mychart
```

res: `No issues found`

Powrót do poprzednich wersji – rollback

Polecenie `$ helm rollback` przywraca poprzednią wersję. Należy podać nazwę **releasu** i **numer wersji**:

```
$ helm rollback demo-staging 1
```

Automatyczny powrót za pomocą – *helm-monitor, Kuberbs*

Wtyczki **helm-monitor** i **Kuberbs** mogą automatycznie powrócić do poprzedniej wersji, bazując na odpowiednich metrykach.

Charts Hooks

Helm dostarcza mechanizm haków pozwalający na interwencję w określonych punktach cyklu życia wydania.

Na przykład można wykonać zadanie przed usunięciem wydania, aby zgrabnie wyłączyć usługę z rotacji przed jej usunięciem.

Zdefiniowane są następujące haki:

- pre-install,
- post-install,

- pre-delete,
- post-delete,

- pre-upgrade,
- post-upgrade,

- pre-rollback,
- post-rollback,

- test.

Repozytorium wykresów Helm

Aby utworzyć własne repozytorium wykresów Helm, wykresy muszą być dostępne przez **HTTP**.

Aby zainstalować wykresy z repozytorium, musisz przede wszystkim dodać repozytorium do listy Helm:

```
$ helm repo add myrepo http://myrepo.example.com
```

```
$ helm install myrepo/myapp
```

Repozytorium charakteryzuje się przede wszystkim obecnością specjalnego pliku o nazwie index.yaml, który zawiera listę wszystkich pakietów dostarczanych przez repozytorium, wraz z metadanymi, które umożliwiają pobieranie i weryfikację tych pakietów.

Zarządzanie obiektami Secret wykresów Helm – Sops + helm-secrets

Jeśli masz więcej niż jeden lub dwa obiekty Secret do zarządzania, łatwiejsze może być utworzenie jednego pliku zawierającego wszystkie te obiekty zamiast pojedynczych plików. A jeśli używasz wykresu Helm do wdrożenia aplikacji, możesz sprawić, by ten plik stał się plikiem wartości i zaszyfrować go za pomocą *Sops*.

Helm-secrets dostarcza dodatkowe możliwości do narzędzi szyfrujących takich jak *Sops* czy *Hashicorp Vault*. Domyślnie używa Sops.

Podstawowe komendy `$ helm secrets`:

- `enc` – zaszyfruj plik *secrets*,
- `dec` – deszufruj plik *secrets*,
- `view` – wydrukuj zaszyfrowany plik *secrets*,
- `edit` – edytuj plik *secrets* (odszufruj przed I and zaszyfruj po),
- `clean` – usuń plik ***.yaml.dec** w folderze (rekursywnie).

helm-wrapper pozwala na wywołanie tych komend `$ helm upgrade` i `$ helm install` z deszyfrowaniem w locie plików sekretów przekazanych jako argumenty `-f` lub `--values`. Należy wywołać:

`$ helm secrets install ...` zamiast `$ helm install ...`

Uwaga:
Jeśli istnieje zdeszyfrowany plik **secrets.yaml.dec** i jest on nowszy niż plik **secrets.yaml**, zostanie on użyty w poleceniu wrapped zamiast deszyfrowania pliku **secrets.yaml**.

Zarządzanie wieloma wykresami za pomocą Helmfile (alt. Landscaper, Helmsman)

O ile **Helm** umożliwia wdrażanie jednej aplikacji przy użyciu szablonów i zmiennych, **Helmfile** pozwala na wdrażanie wszystkiego, co powinno być zainstalowane w klastrze, za pomocą jednego polecenia.

W pliku **helmfire.yaml** każda z podanych wersji (***releases***) określa następujące metadane:

- **name** – nazwę wykresu Helm do wdrożenia,
- **namespace** – przestrzeń nazw, w której ma zostać wdrożony,
- **chart** – adres URL lub ścieżkę do samego wykresu,
- **values** – ścieżkę do pliku **values.yaml**, który ma być używany z obiektem **Deployment**,
- **set** – dodatkowe wartości oprócz zawartych w pliku wartości.

Aby uruchom plik **helmfire.yaml**

`$ helmfile sync`

Rozdział 13. Proces tworzenia oprogramowania

Narzędzia programistyczne

Skaffold

Automatycznie odbudowuje kontenery i **wdraża** te **zmiany w klastrze** lokalnym lub zdalnym. Gdy wprowadzasz zmiany do plików w lokalnym katalogu, **Skaffold** wznawia pracę, buduje nowy kontener ze zmianami, a następnie wdraża go automatycznie, oszczędzając Twój czas.

Draft

Podobnie jak Skaffold, może wykorzystywać Helm do **automatycznego wdrażania aktualizacji** w klastrze po zmianie kodu.

Jeśli dopiero zaczynasz pracę z nową aplikacją, a nie masz jeszcze Dockerfile lub wykresów Helm, **Draft** może być idealnym narzędziem **do szybkiego uruchomienia**.

Strategie wdrażania

W Kubernetes **strategia wdrażania** aplikacji jest zdefiniowana w manifeście Deployment.
Domyślnie wybrana jest opcja **RollingUpdate**.

```
kind: Deployment
spec:
  replicas: 1
  strategy:
    type: Recreate
```

Rolling Updates

W aktualizacjach typu **Rolling Updates** Pody są aktualizowane pojedynczo, dopóki wszystkie repliki nie zostaną zastąpione przez nową wersję.

Jeśli Twoje Pody mogą ulec awarii w krótkim czasie po stanie gotowości, skorzystaj z pola minReadySeconds, aby poczekać z kontynuacją aktualizacji do momentu, aż każdy Pod się ustabilizuje.

Recreate

W trybie **Recreate** wszystkie działające repliki są kończone jednocześnie, a następnie tworzone są nowe.

Jedną z zalet trybu **Recreate** jest to, że pozwala uniknąć sytuacji, w której dwie różne wersje aplikacji działają jednocześnie tak jak podczas *Rolling Updates*.

maxSurge i maxUnavailable

W miarę postępu przebiegu aktualizacji **Rolling Update** czasami będziesz mieć uruchomionych więcej lub mniej Podów in normalna wartość replik. Zachowanie to regulują dwie ważne opcje:

- maxSurge

Ustawia maksymalną liczbę nadmiarowych Podów.
Duże wartości przyspieszają wdrażanie kosztem dodatkowego obciążenia zasobów klastra.

- maxUnavailable

Ustawia maksymalną liczbę Podów, która może być niedostępna.
Duże wartości przyspieszają wdrażanie kosztem wydajności aplikacji.

blue-green deployment

We wdrożeniach niebiesko-zielonych zamiast unieruchamiać i zastępować Pody pojedynczo, tworzony jest zupełnie nowy Deployment, a Serwis jest przełączany dopiero, gdy nowy Deployment jest gotowy do przyjęcia ruchu.

\+ Zaletą rozwiązania jest to, że nie musisz jednocześnie obsługiwać zarówno starych, jak i nowych wersji aplikacji.

– Jednak klaster będzie musiał być wystarczająco duży, aby uruchomić dwukrotnie większą liczbę replik wymaganych dla aplikacji, co może być kosztowne i oznaczać dużą ilość niewykorzystanej pojemności przez większość czasu (chyba że skalujesz klaster, jeśli trzeba).

rainbow deployment

W niektórych rzadkich przypadkach może być konieczne jednoczesne utrzymanie trzech lub więcej wersji aplikacji w tym samym czasie.

Za każdym razem, gdy wdrażasz aktualizację, dostajesz nowy zestaw kolorów Podów. Gdy najstarsze Pody zamkną połączenia, będzie można je usunąć.

canary deployment

Podobnie jak w przypadku kanarka w kopalni węgla, niewielka grupa nowych Podów zostaje wystawiona w środowisku produkcyjnym, aby sprawdzić, co się z nimi stanie. Jeśli przeżyją, wdrożenie może być kontynuowane. Jeśli występuje problem, promień rażenia jest ściśle ograniczony.

Podobnie jak w przypadku wdrożeń niebiesko-zielonych, można to zrobić za pomocą etykiet.
Bardziej wyrafinowanym sposobem jest wykorzystanie **Istio**, która pozwala losowo kierować zmienną część ruchu do jednego lub większej ilości serwisów. Ułatwia to także wykonywanie takich czynności jak testowanie A/B.

Testy A/B

Metoda badawcza polegająca na porównaniu dwóch wersji strony internetowej celem wybrania tej wersji, która lepiej spełnia stawiane przed nią zadania.

Obsługa migracji za pomocą Helm –*hook function*

W przypadku bazy danych wdrażanie i aktualizacja są bardziej skomplikowana niż w przypadku aplikacji bezstanowych. Zmiany w schemacie bazy danych zwykle wymagają uruchomienia zadania migracji w określonym momencie wdrażania.

W tym celu w Kubernetes możesz:

- wykorzystać zasób **Job**;
- wykorzystać poleceń **kubectl** (jako części procesu aktualizacji);
- użyć wbudowanej funkcji Helm o nazwie **hook**.

Funkcje hook pozwalają kontrolować kolejność zdarzeń podczas wdrażania. Pozwalają także przerwać aktualizację, jeśli coś pójdzie nie tak.

Właściwości helm.sh/hook są zdefiniowane w sekcji annotations:

```
apiVersion: batch/v1
kind: Job
metadata:
  annotations:
    "helm.sh/hook": pre-upgrade
    "helm.sh/hook-delete-policy": hook-succeeded
```

Rozdział 14. Ciągłe wdrażanie (Continuous Delivery) w Kubernetes

Continuous Delivery (CD) to automatyczne wdrażanie udanych kompilacji do produkcji. CD jest to seria zautomatyzowanych działań, które przenoszą kod ze stacji roboczej programisty na produkcję, poprzez sekwencję testów i etapów akceptacji. Kluczową kwestią jest to, że artefakt nie jest kodem źródłowym, ale kontenerem. Testowanie kontenera zamiast kodu może pomóc w wychwyceniu wielu błędów wkradających się między kodem źródłowym a działającym plikiem binarnym.

Niektóre z popularnych narzędzi CD, które dobrze współpracują z Kubernetes:

- Jenkins,
- GitLab,
- Drone,
- Cloud Build,
- Spinnaker.

Istnieje również wiele nowszych narzędzi, które zostały utworzone specjalnie do automatyzacji wdrożeń w klastrach Kubernetes. Niektóre z nich to:

- Gitkube,
- Flux,
- Keel.

Definiowanie kroków pipeline'a za pomocą kodu (***Pipeline-as-a-Code***) umożliwia śledzenie i modyfikowanie tych kroków wraz z kodem aplikacji.

Kontenery wspomagają proces budowania komponentów za pośrednictwem środowiska testowego oraz ewentualnie produkcyjnego, bez konieczności przebudowywania nowego kontenera (patrz. Docker multi-stage builds i `$ docker build --target`).

Rozdział 15. Obserwowalność i monitorowanie

W systemach rozproszonych musimy założyć, że usługi, komponenty i połączenia zawodzą, mniej więcej przez cały czas, w sposób tajemniczy i sporadyczny. Odporny system może sobie z tym poradzić, nie zawodząc całkowicie.

Monitorowanie – *monitoring*

Zautomatyzowane monitorowanie polega na sprawdzeniu dostępności lub zachowania aplikacji w programowy sposób, zgodnie z regularnym harmonogramem oraz za pomocą zautomatyzowanego sposobu powiadamiania inżynierów o problemach.

Monitorowanie typu czarna skrzynka

Ja sama nazwa wskazuje, ten typ monitoringu obserwuje tylko **zewnętrzne zachowanie systemu**, bez żadnej próby zaobserwowania, co się z nim dzieje w środku.

Testowe zapytania HTTP – HTTP request

Najprostszym możliwym monitorowaniem jest sprawdzenie kodu stanu HTTP. Jednak może nie zgłaszać problemu w przypadku źle skonfigurowanego, ale działającego serwera WWW. Bardziej zaawansowany monitoring może szukać określonego tekstu na stronie.

W przypadku prostych stron internetowych odpowiedź typu „tak lub nie” na pytanie „Czy działa?” może wystarczyć. W bardziej złożonych stronach pytanie może obejmować np.:

- Czy moja aplikacja jest dostępna na całym świecie? Czy tylko w niektórych regionach?
- Jak długo trwa ładowanie strony dla większości moich użytkowników? Co z użytkownikami, którzy mogą mieć wolne łącze? Czy niektóre funkcje działają wolno, czy wcale, i ilu użytkowników to dotyczy?
- Czy wszystkie funkcje mojej witryny działają zgodnie z przeznaczeniem?
- Jeśli strona opiera się na usługach stron trzecich, co dzieje się z moją aplikacją, gdy ta usługa zewnętrzna działa źle lub jest niedostępna?
- Co się stanie, gdy mój dostawca usług chmurowych ma awarię?

Monitorowanie z zewnątrz infrastruktury

Wykonanie monitoringu wewnątrz infrastruktury nie wystarczy. Błąd może wynikać z różnego rodzaju problemów oraz awarii między użytkownikiem a zewnętrznymi elementami infrastruktury, np.:

- nieprawidłowych rekordów DNS,
- partycji sieciowych,
- utraty pakietów,
- źle skonfigurowanych routerów,
- brakujących lub złych reguł firewall,
- awarii dostawcy chmury.

Dlatego powinno się też monitorować dostępność usług z pewnego miejsca zewnętrznego w stosunku do infrastruktury.

Ograniczenia black-box

Istnieje jednak kilka ograniczeń takiego monitorowania typu black-box.

- Może wykryć tylko przewidywalne awarie (np. strona internetowa nie odpowiada).
- Sprawdza tylko zachowanie części systemu, tych które są wystawione na zewnątrz.
- Jest pasywny i reaktywny, tzn. informuje o problemie dopiero po jego wystąpieniu.
- Może odpowiedzieć na pytanie „co jest zepsute?”, ale nie na ważniejsze pytanie „dlaczego?”.

Zapisywanie logów

Jeśli korzystasz z logowania, to zamiast zapisywać rekordy w postaci zwykłego tekstu, powinieneś użyć pewnej formy danych strukturalnych, takich jak JSON, które można automatycznie przeanalizować.

Dzienniki mogą być przydatne, ale mają też swoje ograniczenia. Decyzję o tym, co należy zalogować, a czego nie, trzeba podjąć w momencie pisania aplikacji przez programistę. Dlatego, podobnie jak w przypadku czarnej skrzynki, dzienniki mogą tylko odpowiadać na pytania lub wykrywać problemy, które można przewidzieć z góry.

Ponieważ dzienniki muszą rejestrować wystarczającą ilość informacji, aby zdiagnozować każdy możliwy problem, zazwyczaj mają słaby stosunek sygnału do szumu.

Dzienniki również nie skalują się dobrze. Jeśli każde żądanie użytkownika generuje wiersz dziennika, który musi zostać wysłany do agregatora, musisz użyć dużej przepustowości sieci, która jest w związku z tym niedostępna dla użytkowników.

Podczas gdy scentralizowane agregowanie logów może być przydatne w aplikacjach Kubernetes, to nie zapewniają wszystkich informacji potrzebnych dla prawdziwej obserwowalności.

Metryki

Metryka jest liczbową miarą czegoś. W zależności od aplikacji odpowiednie wskaźniki mogą obejmować:

- liczbę aktualnie przetwarzanych żądań,
- liczbę obsługiwanych żądań na minutę,
- liczbę napotkanych błędów podczas obsługi żądań,
- średni czas potrzebny na obsługę żądań (lub wartości maksymalne albo 99 percentylów).

Przydatne jest również zebranie danych na temat infrastruktury i aplikacji, czyli:

- wykorzystanie procesora przez poszczególne procesy lub kontenery,
- aktywność dyskowa operacji we/wy węzłów i serwerów,
- przychodzący i wychodzący ruch sieciowy maszyn, klastrów lub modułów równoważenia obciążenia.

W przeciwieństwie do dzienników, metryki można łatwo przetwarzać na wiele użytecznych sposobów, np.:

- rysować wykresy,
- pobierać statystyki,
- powiadamiać o przekroczeniu zdefiniowanych progów.

Dane mogą również pomóc w odpowiedzi na pytanie „dlaczego?”.

Np. skoki długich czasów odpowiedzi pokrywa się z podobnym skokiem metryki wykorzystania procesora dla określonego komponentu. To daje wskazówkę, że komponent może być zaklinowany lub wielokrotnie ponawiać niektóre nieudane operacje albo jego węzeł może mieć problemy sprzętowe.

Metryki mogą pomóc też w przewidywaniu problemów.
Zanim problem zostanie zauważony przez Ciebie lub Twoich użytkowników, wzrost niektórych metryk może wskazywać, że problem wystąpi niebawem. Przykładowo metryka informująca o wykorzystaniu dysku na serwerze może powoli rosnąć.

Niektóre systemy do analizowania metryk wykorzystują nawet techniki uczenia maszynowego, aby wykryć anomalię i uzasadnić przyczynę jej wystąpienia. Jednak dla większości zastosowań zbieranie danych, tworzenie wykresów oraz alarmy generowane przez metryki są wystarczająco dobrym rozwiązaniem.

Śledzenie – *tracing*

Jest szczególnie ważne w systemach rozproszonych. Podczas gdy metryki i dzienniki informują o tym, co się dzieje z poszczególnymi komponentami systemu, śledzenie koncentruje się na jednym żądaniu użytkownika oraz całym cyklu jego życia.

Może się okazać, że czas obsługi żądania na każdym etapie jest normalny, z wyjątkiem przetwarzania przez bazę danych, mimo iż działa poprawnie, a jej wskaźniki nie wykazują problemów. Ewentualnie odkryjesz problem związany z nadmierną utratą pakietów na połączeniu pomiędzy serwerami aplikacji a serwerem bazy danych.

Niektóre popularne narzędzia do rozproszonego śledzenia: **Zipkin**, **Jaeger** i **LightStep.**

Obserwowalność – observability

Monitorowanie informuje, czy system działa, a obserwowalność zachęca do odpowiedzi na pytanie, dlaczego nie działa lub działa inaczej niż zakładano.

Obserwowalność dotyczy również danych. Musimy wiedzieć, jakie dane generować, co gromadzić, jak je agregować, na jakich wynikach się skupić oraz jak je wyszukiwać i wyświetlać.

Jeszcze bardziej ogólnie obserwowalność dotyczy kultury. To kluczowa zasada filozofii DevOps polegająca na zamknięciu pętli między tworzeniem kodu a uruchamianiem go na dużą skalę w środowisku produkcyjnym.

Potok obserwowalności – *observability pipeline*

Za pomocą **observability pipeline** oddzielamy źródła danych od miejsc docelowych i udostępniamy bufor. Wszystkie dane są przesyłane do potoku, który obsługuje ich **filtrowanie** i **dostarczanie** we właściwe miejsca. Dodanie nowego źródła danych, usługi wizualizacyjnej lub alerty to tylko kwestia podłączenia go do potoku.

Korzystanie z potoku obserwowalności wymaga standardowego formatu metryk. Zamiast emitować logi tekstowe i analizować je później przy użyciu delikatnych wyrażeń regularnych, korzystaj ze strukturyzowanych danych.

Monitorowanie w Kubernetes

Nie buduj własnej infrastruktury monitorowania

Większość tych usług jest na pewnym poziomie darmowa albo posiada niedrogie subskrypcje (jako niezbędny wydatek operacyjny).

Sprawdź, czy zewnętrzny dostawca monitorowania obsługuje następujące krytyczne funkcje:

- sprawdzanie HTTP/HTTPS,
- sprawdzanie, czy Twój certyfikat TLS jest nieważny lub wygasł,
- dopasowanie słowa kluczowego (ostrzeżenie, gdy brakuje słowa kluczowego lub gdy jest obecne),
- alerty przez e-mail, SMS, webhook lub inny prosty mechanizm,
- automatyczne tworzenie lub aktualizowanie rodzaju sprawdzeń za pośrednictwem interfejsu API (aby stosować *Infrastructure-as-a-Code*).

Rozbudowany test sondy – *health check*

To, że program działa, niekoniecznie oznacza, że działa poprawnie. Jeśli program musi komunikować się z bazą danych, bardziej zaawansowana może sprawdzić, czy ma prawidłowe i responsywne połączenie z bazą danych. Jeżeli zależy to od innych usług, może sprawdzić dostępność usług.

Wzorzec wyłącznika – *circuit breaker pattern*

Jak wiesz, jeśli sprawdzenie żywotności kontenera nie powiedzie się, Kubernetes uruchomi go ponownie. Nie jest to tak bardzo pomocne w sytuacji, gdy w kontenerze nie dzieje się nic złego, a zawiedzie tylko jeden składnik. Może to doprowadzić, że błąd jednego kontenera w ostateczności wyłączy front-end.

Wzorzec wyłącznika polega na tym, aby wyłącza się z działania komponent niższego poziomy, aby zapobiec wysyłaniu do niego kolejnych żądań, dopóki problem nie zostanie rozwiązany.

Wdzięczna degradacja – *degrade gracefully*

Należy tak zaprojektować swoje usługi, aby uniknąć awarii całego systemu w przypadku niedostępności jednej usług składowych. Wdzięczna degradacja oznacza, że nawet jeśli aplikacja nie może zrobić wszystkiego, nadal może zrobić coś.

Rozdział 16. Metryki w Kubernetes

Liczniki i mierniki – counter & gauge /geɪdʒ/

Licznik (counter) może tylko rosnąć lub resetować się do zera. Nadaje się do mierzenia n.in. liczby obsługiwanych żądań i liczby otrzymanych błędów.

Miernik (gauge) może się zmieniać w górę i w dół. Jest przydatne do pomiaru ciągle zmieniających się wartości, takich jak zużycie pamięci.

Odpowiedzi na niektóre pytania to prostu tak lub nie. W takim przypadku odpowiednią metryką będzie **gauge** o ograniczonym zakresie wartości do 0 i 1.

Wybór dobrych metryk

Usługi: wzorzec RED – *Requests-Errors-Duration*

Wzorzec RED oferuje użyteczne informacje na temat wydajności Twoich usług oraz ich wpływu na użytkowników.

- Liczba otrzymywanych *żądań* – Requests

  Ponieważ łączna liczba żądań stale rośnie, bardziej przydatne jest sprawdzenie częstotliwości żądań, np. liczby żądań na sekundę. Daje to sensowny obraz ruchu, jaki system obsługuje w danym przedziale czasu.

- *Liczba błędów* – Errors

  Liczba żądań, które zakończyły się niepowodzeniem na różne sposoby.

  Ponieważ poziom błędów jest powiązany z częstotliwością żądań, warto mierzyć błędy procentowo.

- Czas trwania każdego żądania – Duration

  Daje on wyobrażenie o tym, jak dobrze działa usługa i jak bardzo nieszczęśliwi mogą być użytkownicy.

Zasoby: wzorzec USE – *Utilization-Saturation-Errors*

Wzorzec USE ma pomóc w analizie problemów z wydajnością i w znalezieniu wąskich gardeł. Mierzenie tych danych dla kluczowych zasobów w systemie jest dobrym sposobem na wykrycie wąskich gardeł. Zasoby o niskim zużyciu, bez nasycenia i bez błędów są prawdopodobnie w porządku.

Wszystko, co odbiega od tego, jest warte uwagi. Jeśli np. jeden z linków sieciowych jest nasycony lub zawiera dużą liczbę błędów, może przyczyniać się do ogólnych problemów z wydajnością.

USE oznacza:

- wykorzystanie – Utilization

  Średni czas, w którym zasób był zajęty obsługą żądań, lub ilość aktualnie wykorzystywanej pojemności zasobu. Przykładowo dysk, który jest zapełniony w 90%, miałby wykorzystanie w 90%.

- nasycenie – Saturation

  Stopień przeciążenia zasobu lub długość kolejki żądań oczekujących na dostępność tego zasobu.
  Jeśli np. na uruchomienie czeka 10 procesów, wartość nasycenia wynosi 10.

- błędy – Errors

  Liczba niepowodzeń operacji na tym zasobie.
  Przykładowo dysk z niektórymi uszkodzonymi sektorami może mieć liczbę błędów jako 25 nieudanych odczytów.

Metryki biznesowe

Istnieją przydatne wskaźniki biznesowe, które mogą być generowane przez usługi. Np.:

- **Wskaźnik rejestracji i rezygnacji**
- **Przychody na klienta**
  przydatne do obliczania miesięcznych przychodów cyklicznych
- **Skuteczność stron pomocy i wsparcia**
  np. odsetek osób, które odpowiedziały twierdząco na pytanie: „czy ta strona rozwiązała problem?”
- **Ruch do strony informującej o stanie systemu**
  który często gwałtownie wzrasta w przypadku awarii lub pogorszenia jakości usług

Metryki Kubernetes

Wskaźniki kondycji klastra

Możesz monitorować kondycję i wydajność klastra na najwyższym poziomie tzn.

- czy ma wystarczającą pojemność,
- jak zmienia się jego użycie w czasie,
- czy musisz go rozszerzyć, czy też zmniejszyć.

Powinieneś spojrzeć przynajmniej na kwestie, takie jak:

- liczba węzłów
- liczba Podów na węzeł
- wykorzystanie/alokacja zasobów na węzeł

Metryki obiektu Deployment

W przypadku wszystkich obiektów Deployment ważne są:

- liczba obiektów Deployment
- liczba skonfigurowanych replik na Deployment
- liczba niedostępnych replik na Deployment

Warto zwrócić uwagę na dane dotyczące niedostępnych replik — ostrzegają o problemach związanych z pojemnością.

Metryki kontenera

Na poziomie kontenera ważne są:

- liczba kontenerów lub Podów na węzeł
- wykorzystanie zasobów dla każdego kontenera w stosunku do jego żądań czy limitów

  Jeśli kontener regularnie przekracza swoje limity zasobów, może to oznaczać błąd programu, a może po prostu musisz nieznacznie zwiększyć limit.

- żywotność (*liveness*) lub gotowość (*readiness*) kontenerów
- liczba restartów kontenera lub Poda

  Kubernetes automatycznie restartuje te kontenery, które uległy awarii lub przekroczyły swoje limity zasobów, dlatego musisz wiedzieć, jak często to się dzieje. Nadmierna liczba restartów może wskazywać na problem z konkretnym kontenerem.

- ruch sieciowy
- błędy dla każdego kontenera

Metryki aplikacji

Niezależnie od języka programowania, prawdopodobnie dostępna jest biblioteka lub narzędzie umożliwiające eksport niestandardowych danych.

Wybór metryk aplikacji zależy od tego, co ona robi. Są jednak pewne wspólne wzorce.

Jeśli np. usługa obsługuje wiadomości **kolejki** i je przetwarza je, możesz obserwować metryki, takie jak:

- liczba otrzymanych komunikatów,
- liczba pomyślnie przetworzonych komunikatów,
- liczba nieprawidłowych lub błędnych komunikatów,
- czas przetwarzania każdego komunikatu i reakcji na każdy z nich.

Jeśli aplikacja oparta jest głównie na **żądaniach**, można użyć wzorca RED (*Requests-Errors-Duration*)

Trudno określić, jakie metryki będą przydatne, gdy jesteś na wczesnym etapie rozwoju.
W razie wątpliwości rejestruj wszystko. Dane są tanie.

Metryki czasu wykonywania

Na poziomie środowiska wykonawczego większość bibliotek metryk będzie również raportować przydatne dane o tym, co robi program, takie jak:

- liczba procesów/wątków,
- wykorzystanie stosów (***stack***) i sterty (***heap***),
- ilość wykorzystywanej/pozostałej pamięci,
- pule buforów sieciowych.

Tworzenie wykresów metryk w pulpicie (*dashboard*)

Użyj standardowego układu graficznego dla wszystkich serwisów

Warto ułożyć pulpity w taki sam sposób dla każdego z serwisów, aby każdy mógł szybko zinterpretować dane, bez konieczności znajomości tego konkretnego serwisu.

Powinieneś regularnie przeglądać kluczowe pulpity (przynajmniej raz w tygodniu), sprawdzając dane z poprzedniego tygodnia, aby wszyscy wiedzieli, jaki jest normalny wygląd.

Najbardziej przydatne wykresy:

- **dla usług**: żądań, błędów i czasu trwania (**RED**),
- **dla zasobów**: stopień wykorzystania, nasycenie, błędy (**USE**).\

Zbuduj radiator informacji za pomocą dashboardu – *information radiator*

Jeśli masz sto serwisów, ważne jest, aby te informacje były dostępne. W tej skali potrzebujesz bardziej ogólnego przeglądu. Utwórz główny pulpit, który pokazuje żądania, błędy i czas trwania dla wszystkich serwisów łącznie i zrób z niego **radiator informacji**. Jest to **duży ekran** pokazujący **kluczowe dane**, które są **widoczne dla wszystkich** w odpowiednim zespole lub biurze.

Nie rób nic fantazyjnego: proste wykresy liniowe są łatwiejsze w interpretacji i lepiej wizualizują złożone wykresy. Często kuszące jest wrzucanie zbyt dużej ilości informacji. To nie może być celem.

Celem radiatora jest:

- szybkie pokazanie aktualnego stanu systemu,
- pokazanie jasnej informacji o tym, które wskaźniki zespół uważa za ważne,
- zapoznanie ludzi z tym, jak wygląda normalna sytuacja.

Umieszczanie na pulpicie rzeczy, które ulegają awarii

Przydatne są też pulpity dla określonych metryk, które ulegają awarii. Za każdym razem, gdy wystąpi jakieś zdarzenie lub awaria, poszukaj metryki lub kombinacji metryk, które z góry powiadomiłyby Cię o tym problemie.

Nie chodzi tu o problemy, które pojawiają się w ciągu kilku minut lub nawet godzin; one powinny być przechwytywane przez automatyczne alerty. Mowa tu o tendencjach, które zbliżają się w ciągu dni lub tygodni. Chociaż alerty informują o przekroczeniu ustalonego progu, nie zawsze z góry wiadomo, jaki powinien być.

Alarmy na podstawie metryk

Dla niektórych alarmy są istotą monitorowania. Uważamy, że takie podejście z wielu powodów musi się zmienić.

Jakie są problemy związane z alarmami?

Duże systemy rozproszone nigdy nie są w stanie *działa*. Mają tak wiele metryk, że jeśli zaalarmujesz za każdym razem, gdy niektóre z nich przekroczą normalne limity, wygenerujesz setki raportów dziennie, bez żadnego celu. Kiedy alarmy występują ciągle, ludzie stają się wobec nich obojętni. Fałszywe alarmy niebezpiecznie zmniejszają zaufanie do systemu.

Aby system monitorowania był użyteczny, musi mieć bardzo wysoki stosunek sygnału do szumu.
Ostrzeżenie powinno oznaczać jedną bardzo prostą rzecz: **osoba musi podjąć działanie.**

Jeśli nie jest wymagane żadne działanie, nie jest potrzebny alarm.
Jeśli działanie może podjąć automatyczny system, to zautomatyzuj je.

Pilne, ważne i przydatne alarmy

Jeśli problem ma rzeczywisty lub potencjalny wpływ na biznes oraz wymaga ludzkiego działania, może być potencjalnym kandydatem do wygenerowania alarmu.

Ostrzeżenia powinny być ograniczone tylko do alarmów:

- **pilnych**
  ORAZ
- **ważnych**
  ORAZ
- **możliwych do rozwiązania**.

Poza tym możesz wysyłać asynchroniczne powiadomienia (np. e-maile), które są widoczne w godzinach pracy.

Przeglądaj wszystkie pilne zgłoszenia, co najmniej raz w tygodniu i napraw lub wyeliminuj fałszywe lub niepotrzebne alarmy. Jeśli nie potraktujesz tego poważnie, ludzie nie będą traktować Twoich alertów poważnie.

Uwaga:

Śledź swoje alarmy, powiadomienia poza godzinami pracy i pobudki. Liczba alertów wysłanych w danym tygodniu jest dobrym wskaźnikiem ogólnego stanu zdrowia i stabilności Twojego systemu. Liczba pilnych zgłoszeń, a zwłaszcza liczba powiadomień wysyłanych poza godzinami pracy, w weekendy i podczas normalnego snu, jest dobrym wskaźnikiem ogólnego stanu zdrowia i morale Twojego zespołu.

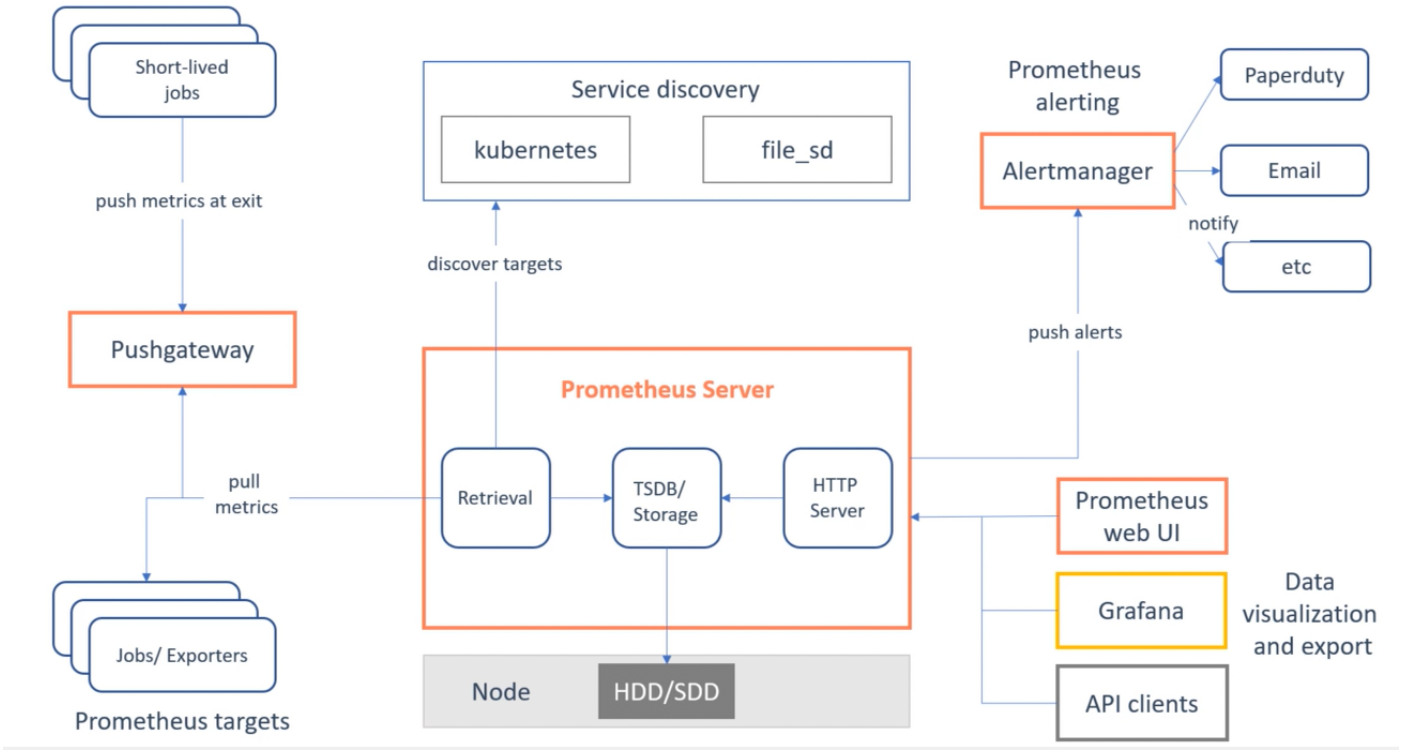
Prometheus

Tłumaczenie i streszczenie wpisu z bloga autorstwa Ryana Harrisona

Prometheus to zestaw narzędzi open source służący do monitorowania i ostrzegania o problemach w systemie na podstawie danych metryk w czasie.

Rdzeniem narzędzia Prometheus jest **serwer**, który gromadzi i przechowuje metryki. Zawiera też różne inne opcjonalne komponenty, takie jak narzędzie alarmujące (**Alertmanager**) oraz biblioteki dla języków programowania.

Architektura



- Główny serwer Prometheus (**Prometheus Server**) składa się z:
  - bazy danych szeregów czasowych (***time-series database*** - ***TSDB***) – przechowuje każdy z przechwyconych pomiarów;
  - scrapera (**Scraper/Retrieval**) – pobiera próbki z zewnętrznych aplikacji, hostów, platform;
  - serwera http (**HTTP Server**) – pozwala na wykonywanie operacji na metrykach przez zewnętrzne serwisy.

- **Prometheus** jest komponentem typu single-instance, wszystkie dane są przechowywane na lokalnym dysku węzła.
  - Jeśli potrzebujesz skalować, zalecane jest uruchomienie wielu oddzielnych instancji Prometheus.

- Prometheus domyślnie działa w modelu **pull**, w którym ustawiony jest do okresowego pobierania metryk ze wszystkich docelowych instancji aplikacji.
  - Dlatego musi wiedzieć o lokalizacji wszystkich aktywnych instancji poprzez **service discovery.**
  - Sama aplikacja nie posiada wiedzy o Prometheusie, udostępnia **endpoint** z najnowszym zrzutem metryk.

- Dla krótko żyjących instancji i procesów, od których metryki mogą nie zdążyć być pobrane (*scraped*) przewidziany jest komponent **Pushgateway** (jako *middle-man*), na który wysyłane są metryki przez same aplikacje/procesy.

- **Service discovery** integruje się z infrastrukturą (np. Kubernetes, AWS), aby wykryć aktualną topologię wszystkich węzłów docelowych (***target nodes***)

- Pobrana do *TSDB* metryka może zostać udostępniona poprzez UI/Grafana/API (***data visualization*** *and* ***export***)

- Na podstawie zestawu reguł odpytujących metryki **Alert Manager** generuje alerty systemowe, czyli:
  - wykonuje de-duplikację alertów, throttling, wyciszanie itp.,
  - informuje o alercie poprzez np. mail, Slack.

Konfiguracja

Cała konfiguracja znajduje się w pliku ***Prometheus.yml***.
Można przeładować konfigurację w czasie działania aplikacji (o ile nowa konfiguracja jest poprawna) poprzez wysłanie *HTTP POST* do endpointu /-/reload (o ile flaga --web.enable-lifecycle jest odblokowana).

```
global:                                  # Zastosowane do wszystkich celów, o ile nie nadpisane.
  scrape_interval: 15s                   # Jak często pobierać metryki.
  evaluation_interval: 15s               # Jak często stosować predefiniowane reguły (rules).

scrape_configs:                          # Zestaw zadań (jobs) określających skąd pobierać metryki.
- job_name: "prometheus"                 # Grupa celów, dodawana również jako etykieta do każdego pomiaru.
  metrics_path: '/metrics'               # Endpoint z metrykami (domyślnie '/metrics').
  static_configs:                        # Wpisany na sztywno host/port lub może to być service discovery.
  - targets: ["localhost:9090"]          # Host dla endpointa metryk (domyślnie http).
```

Service Discovery

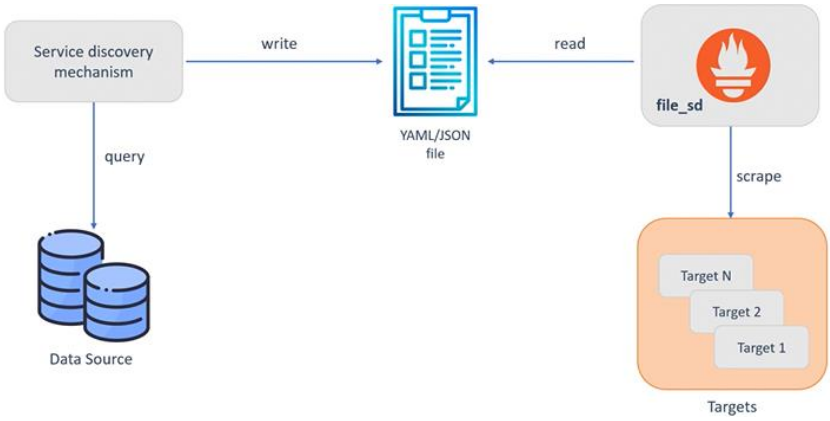
static_configs nie skaluje się do dynamicznych środowisk, gdzie instancje są często dodawane/usuwane.

Prometheus może zintegrować się z usługami ***service discovery***, aby automatycznie aktualizować widok działających instancji.

- Kiedy nowe instancje są dodawane, Prometheus rozpocznie scrapowanie.
- Kiedy zostaną utracone z discovery, seria czasowa również zostanie usunięta.

Plik JSON/YAML może być publikowany przez platformę, określając wszystkie cele, z których należy skrobać.

```
scrape_configs:
- job_name: 'node'
  file_sd_configs:
  - files:
    - 'targets.json'
```



Instrumenting

Istnieją dwa sposoby, w jakie metryki aplikacji mogą być wystawione dla Prometheusa:

- użycie **biblioteki** klienckiej bezpośrednio **w aplikacji** w celu utworzenia i udostępnienia endpointa dla Prometheusa (zazwyczaj /metrics),
- użycie pośredniej instancji proxy **exporter**, która instrumentuje aplikację docelową i konwertuje na format metryk Prometheusa.

Eksportery

Istnieje wiele bibliotek i serwerów, które pomagają w eksporcie istniejących metryk z systemów innych firm jako metryk Prometheusa. Jest to przydatne w przypadkach, gdy nie jest możliwe bezpośrednie instrumentowanie danego systemu metrykami Prometheusa (np. jądro Linux).

Konwencja i praktyki

- Nazwy metryk powinny zaczynać się od litery, po której może następować dowolna ilość liter, cyfr i podkreśleń: a_123_asd_23.
- Metryki muszą mieć unikalne nazwy, biblioteki klienckie powinny zgłaszać błąd, jeśli próbujesz zarejestrować tę samą metrykę dwa razy.
- Powinny posiadać przyrostek opisujący jednostkę w liczbie mnogiej (np. _bytes, _total, _seconds).
- Powinny reprezentować tę samą logiczną rzecz mierzoną we wszystkich wymiarach etykiety.
- Każda unikalna kombinacja par klucz-wartość etykiety reprezentuje nowy szereg czasowy, co może drastycznie zwiększyć ilość przechowywanych danych.
- Nie należy używać etykiet do przechowywania wymiarów o wysokiej kardynalności (wiele różnych wartości etykiet), takich jak identyfikatory użytkowników, adresy e-mail lub inne nieograniczone zestawy wartości.

Relabelling

Konfiguracje *przeetykietowania* pozwalają wybrać cele, które mają być poddane scrapingowi, oraz jakie będą etykiety celów. Więc jeśli chcesz powiedzieć, aby skrobać ten a nie inny typ maszyny, użyj **relabel_configs**.

Natomiast **metric_relabel_configs** są stosowane po wykonaniu scrape'u, ale zanim dane zostaną pobrane przez system przechowywania. Więc jeśli są jakieś ciężkie metryki, które chcesz porzucić, lub etykiety pochodzące z samego scrapa (np. ze strony /metrics), którymi chcesz manipulować, to właśnie tam stosuje się **metric_relabel_configs**.

W uprzoszczeniu:

- **relabel_config** dzieje się przed scrape'm,
- **metric_relabel_configs** dzieje się po scrape'ie (przed zapisem do TSDB).

Przykłady:

```
relabel_configs:
- source_labels: [__meta_kubernetes_service_label_app]
  regex: nginx
  action: keep
- source_labels: [__meta_ec2_public_ip]
  regex: '(.*)'
  replacement: '${1}:9100'
  target_label: __address__
```

```
- job_name: cadvisor
  metric_relabel_configs:
  - source_labels: [container_label_JenkinsId]
    regex: '.+'
    action: drop
```

Można wykonać następujące operacje (***action***):

- **keep** – zachowaj dopasowany cel lub serię, porzuć wszystkie inne;
- **drop** – usuń dopasowany cel lub serię, zachowaj wszystkie inne;
- **replace –** zmień nazwę dopasowanej etykiety na nową, zdefiniowaną przez parametry target_label i replacement**;**
- **labelkeep** – dopasuj regex do wszystkich nazw etykiet, usuń wszystkie etykiety, które nie pasują;
- **labeldrop** – dopasuj regex do wszystkich nazw etykiet, usuwa wszystkie etykiety, które pasują.

Federacje

Prometheus jest narzędziem typu stand-alone, co oznacza, że nie skaluje się horyzontalnie. Tu z pomocą przychodzi **federacja**. Powszechnie, jest ona używana do osiągnięcia skalowalnych konfiguracji monitorowania Prometheusa lub do przeciągania powiązanych metryk z jednej usługi Prometheusa do innej.

- Federacja hierarchiczna

W przypadku **federacji hierarchicznej** topologia przypomina drzewo, w którym serwery Prometheus wyższego poziomu zbierają zagregowane dane szeregów czasowych z większej liczby podległych serwerów.
Umożliwia to skalowanie do środowisk z dziesiątkami centrów danych i milionami węzłów.

Na przykład, konfiguracja może zapewniać zagregowany widok globalny i szczegółowe widoki lokalne.

- Federacja cross-service

W przypadku **federacji między usługami,** serwer Prometheus jednej usługi jest skonfigurowany tak, aby skrobać wybrane dane z serwera Prometheus innej usługi. Umożliwia to alertowanie i zapytania dotyczące obu zbiorów danych w ramach jednego serwera.

Na przykład

***Scheduler* klastra** uruchamiający wiele usług może ujawnić informacje o wykorzystaniu zasobów. Z drugiej strony, usługa działająca na tym klastrze będzie eksponować tylko metryki specyficzne dla aplikacji. Często te dwa zestawy metryk są zbierane przez oddzielne serwery Prometheus. Wykorzystując federację, serwer Prometheus zawierający metryki na poziomie usługi może pobierać metryki wykorzystania zasobów klastra dotyczące jego konkretnej usługi z Prometheusa klastra, tak aby oba zestawy metryk mogły być używane w ramach tego serwera.

- Configuracja federacji

Na dowolnym serwerze Prometheus, punkt końcowy `/federate` umożliwia pobranie bieżącej wartości dla wybranego zestawu serii czasowych w tym serwerze. Przynajmniej jeden parametr URL `match[]` musi być określony, aby wybrać serie do ekspozycji.

Aby sfederować metryki z jednego serwera na drugi, należy skonfigurować docelowy serwer Prometheusa tak, aby skrobał z punktu końcowego `/federate` serwera źródłowego, jednocześnie włączając opcję skrobania `honor_labels` (aby nie nadpisywać żadnych etykiet ujawnionych przez serwer źródłowy) i przekazując żądane parametry `match[]`.

Metryki

Biblioteki klienckie oferują cztery podstawowe typy metryk. Obecnie są one rozróżniane tylko w bibliotekach klienckich oraz w protokole sieciowym. Serwer Prometheus nie korzysta jeszcze z informacji o typach i spłaszcza wszystkie dane do postaci nieopisanych szeregów czasowych.

- Counter – licznik

Metryka kumulatywna, która reprezentuje pojedynczy monotonicznie rosnący licznik, którego wartość może tylko wzrosnąć lub zostać wyzerowana przy ponownym uruchomieniu.

Możesz użyć licznika do reprezentowania liczby obsłużonych żądań, ukończonych zadań lub błędów.

Nie używaj licznika do wyświetlania wartości, która może się zmniejszać.

```
# HELP http_requests_total the total number of HTTP requests.

http_requests_total{method="post",code="200"}       1027    1395066363000
http_requests_total{method="post",code="400"}     3         1395066363000
```

- Gauge – wskaźnik

Metryka reprezentująca pojedynczą wartość liczbową, która może arbitralnie wzrastać i spadać.

Wskaźniki są zwykle używane dla wartości mierzonych, takich bieżące wykorzystanie pamięci lub liczba jednoczesnych żądań.

- Histogram

Próbkuje obserwacje i liczy je w konfigurowalnych kubełkach (***bucket***). Dostarcza również sumę wszystkich zaobserwowanych wartości. Jest to metryka kumulatywna.

Przydatny w obserwacji rzeczy takich jak czas trwania żądania lub rozmiar odpowiedzi.

Histogram z nazwą metryki bazowej `<basename>` wyświetla wiele serii czasowych podczas skrobania:

- skumulowane liczniki dla ***observation buckets*** `<basename>_bucket{le="<upper inclusive bound>"}`
- całkowita suma wszystkich zaobserwowanych wartości `<basename>_sum`
- liczba zdarzeń, które zostały zaobserwowane `<basename>_count`

(identycznie jak `<basename>_bucket{le="+Inf"}`)

```
# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram

http_request_duration_seconds_bucket{le="0.1"}      33444
http_request_duration_seconds_bucket{le="0.5"}      129389
http_request_duration_seconds_bucket{le="1"}        133988
http_request_duration_seconds_bucket{le="+Inf"}     144320
http_request_duration_seconds_sum                   53423
http_request_duration_seconds_count                 144320
```

W celu obliczenia kwantyli z histogramów użyj funkcji `histogram_quantile()`.

- Summary – podsumowanie

Podobnie do histogramu, summary **próbkuje obserwacje**. Chociaż dostarcza również całkowitą liczbę obserwacji i sumę wszystkich zaobserwowanych wartości, to **oblicza konfigurowalne kwantyle** w przesuwanym oknie czasowym.

Podsumowanie z nazwą metryki bazowej `<basename>` wyświetla wiele serii czasowych podczas skrobania:

- **strumieniowe kwantyle** ϕ (0 ≤ ϕ ≤ 1) zdarzeń `<basename>{quantile="<ϕ>"}`
- całkowita suma wszystkich zaobserwowanych wartości `<basename>_sum`
- liczba zaobserwowanych zdarzeń `<basename>_count`

```
# HELP rpc_duration_seconds A summary of the RPC duration in seconds.
# TYPE rpc_duration_seconds summary

rpc_duration_seconds{quantile="0.01"}       3102
rpc_duration_seconds{quantile="0.5"}        4773
rpc_duration_seconds{quantile="0.9"}        9001
rpc_duration_seconds{quantile="0.99"}       76656
rpc_duration_seconds_sum                    1.7560473e+07
rpc_duration_seconds_count                  2693
```

Wybierz **histogramy**, jeśli musisz agregować.

Summary nada się w przypadku, gdy:

- masz pojęcie o zakresie i dystrybucji wartości, które będą obserwowane
- albo potrzebujesz dokładnego kwantyli.

PromQL

Typy języka wyrażeń – expression language

W zależności od przypadku użycia, tylko niektóre z tych typów są właściwe jako wynik wyrażenia zdefiniowanego przez użytkownika.

Np. wyrażenie zwracające wektor chwilowy jest jedynym typem, który może być bezpośrednio przedstawiony na wykresie.

- instant vector – *wektor chwilowy*

Zbiór szeregów czasowych zawierający pojedynczą próbkę dla każdego szeregu czasowego.
Wszystkie mają ten sam znacznik czasu.

```
http_requests_total{job="prometheus"}
```

- range vector – *wektor zakresu*

Zbiór szeregów czasowych zawierający zakres próbek danych dla każdego szeregu czasowego.
Czas trwania jest dołączony w nawiasach kwadratowych [ ] na końcu selektora wektora.

```
http_requests_total{job="prometheus"}[5m]
```

Czas trwania można określić:

- ms – milliseconds,
- s – seconds,
- m – minutes,
- h – hours,
- d – days – w założeniu, że dzień ma zawsze 24h,
- w – weeks – w założeniu, że tydzień ma zawsze 7d,
- y – years – w założeniu, że rok ma zawsze 365d.

Można je łączyć: np. 1h30m.

- scalar – *skalar*

Prosta wartość liczbowa zmiennoprzecinkowa.

- string

Prosta wartość łańcuchowa; obecnie nieużywana.

selectors & matchers

W najprostszej formie **selektora** określona jest tylko nazwa metryki. Wynikiem tego jest ***instant vector*** zawierający elementy dla wszystkich szeregów czasowych, które mają tę nazwę metryki:

`http_requests_total`

Możliwe jest dalsze filtrowanie tych szeregów czasowych poprzez dołączenie listy oddzielonych przecinkami dopasowań etykiet w nawiasach klamrowych `{}`.

`http_requests_total{job="prometheus",group="canary"}`

Powyższy przykład wybiera tylko te serie czasowe, które mają:

- nazwę metryki `http_requests_total`,
- etykietę `job` ustawioną na *"Prometheus"*,
- etykietę `group` ustawioną na *"canary"*.

Dostępne dla selektorów **operatory przyrównania** (***matcher***):

- **`=`** – dokładnie równe podanemu ciągowi znaków,
- **`!=`** – nierówne podanemu łańcuchowi,
- **`=~`** – dopasowane przy użyciu regex,
- **`!~`** – niedopasowane przy użyciu regex.

Selektory wektorowe muszą albo określać nazwę albo przynajmniej jeden dopasowujący etykietę, który nie pasuje do pustego łańcucha.

```
{job=~".*"}               # Bad!
{job=~".+"}               # Good!
{job=~".*",method="get"} # Good!
```

Dopasowywanie etykiet może być również zastosowane do nazw metryk `{__name__=~"job:.*"}`

PromQL Cheat Sheet

Na stronie **PtomLabs znajduje Cheat Sheet** z wyjaśnionymi podstawowymi elementami języka PromQL wraz z interaktywnym interpreterem zapytań na przykładowej bazie metryk.

Reguły – Rules

Prometheus obsługuje dwa rodzaje reguł, które mogą być konfigurowane, a następnie analizowane w regularnych odstępach czasu:

- **Recording Rules**
- **Alerting Rules**

Aby włączyć reguły, należy utworzyć plik zawierający deklaracje reguł wskazać plik w polu `rule_files` w konfiguracji.

Recording Rules

Recording Rules pozwalają na wstępne obliczenie często potrzebnych lub kosztownych obliczeniowo wyrażeń i zapisanie ich wyniku jako nowego zestawu szeregów czasowych z nazwą metryki podaną przez **'record'**.

Odpytanie już obliczonego wyniku będzie często znacznie szybsze niż wykonywanie oryginalnego wyrażenia za każdym razem, gdy jest ono potrzebne. Jest to szczególnie przydatne dla dashboardów, które muszą odpytywać to samo wyrażenie wielokrotnie przy każdym odświeżeniu.

Reguły powinny mieć ogólną postać **level:metric:operation**

- **level** – poziom agregacji metryki i etykiet wyjścia reguły
- **metric** – nazwa metryki poddawanej ocenie
- **operation** – lista operacji zastosowanych do ocenianej metryki

```
groups:
- name: example # Musi być unakalna w ramach pliku.
  rules:
  - record: job:http_inprogress_requests:sum              # Nazwa poprawnej metryki.
    expr: sum by (job) (http_inprogress_requests)         # Wyrażenie PromQL do ewaluacji.
```

Alerting Rules

Reguły alertów pozwalają na definiowanie warunków alertów oraz wysyłanie powiadomień o uruchomieniu alertów do usługi zewnętrznej.

Reguły alertów są konfigurowane w *Prometheusie* w taki sam sposób jak reguły rejestrowania:

```
groups:
- name: example
  rules:         # Alert dla dowolnej instancji, która ma medianę opóźnień żądań > 1s.
  - alert: APIHighRequestLatency
    expr: api_http_request_latencies_second{quantile="0.5"} > 1
    for: 10m
    labels:
      severity: page
    annotations:
      summary: "High request latency on {{ $labels.instance }}"
      description: "{{ $labels.instance }} has a median request latency above 1s (current value: {{ $value }}s)"
```

gdzie:

- `for` – odczekanie określonego czasu pomiędzy pierwszym napotkaniem nowego elementu *wektora wyrażenia* wyjściowego (`expr`) a uznaniem alertu za uruchomiony dla tego elementu;
- `labels` – określenie zestawu dodatkowych etykiet, które będą dołączane do alertu;
- `annotations` – etykiety informacyjne, które przechowują dłuższe dodatkowe informacje.

Prometheus jest skonfigurowany do cyklicznego wysyłania informacji o stanach alertów do instancji **AlertManager**, który dba o:

- deduplikowanie (***deduplication***),
- grupowanie,
- kierowanie

ich do właściwego odbiorcy.

Podsumowanie wszystkich rozdziałów

Rozdział 1. Rewolucja chmurowa

Zapoznałeś się z krótką prezentacją chmurowego środowiska DevOps. Mamy nadzieję, że wystarczy ona, abyś nabrał pewności, że będziesz sprawniej rozwiązywał problemy związane z środowiskiem chmurowym, kontenerami oraz Kubernetes.

Poniżej zamieściliśmy krótkie podsumowanie głównych tematów — potem przejdziesz do następnego rozdziału i zapoznasz się z Kubernetes.

- Przetwarzanie w chmurze uwalnia Cię od kosztów zarządzania własnym sprzętem, umożliwiając budowanie odpornych, elastycznych, skalowalnych systemów rozproszonych.
- DevOps zapewnia, że współczesne tworzenie oprogramowania nie kończy się w momencie napisania kodu: chodzi o utworzenie pętli sprzężenia zwrotnego między tymi, którzy piszą kod, a tymi, którzy go używają.
- DevOps wprowadza podejście zorientowane na kod oraz dobre praktyki inżynierii oprogramowania w świecie infrastruktury i operacji.
- Kontenery umożliwiają wdrażanie i uruchamianie oprogramowania w małych, znormalizowanych, samodzielnych jednostkach. To sprawia, że budowanie dużych, różnorodnych, rozproszonych systemów jest łatwiejsze i tańsze dzięki połączeniu mikrousług kontenerowych.
- Systemy orkiestracji zajmują się rozmieszczaniem kontenerów, planowaniem, skalowaniem, tworzeniem sieci i wszystkim, co zrobiłby dobry administrator systemu, ale w sposób zautomatyzowany i programowalny.
- Kubernetes to de facto standardowy system orkiestracji kontenerów — gotowy do użycia już dziś w produkcji.
- Cloud native jest skrótem przydatnym, kiedy mówimy o chmurowych, kontenerowych, rozproszonych systemach, złożonych ze współpracujących mikrousług, dynamicznie zarządzanych przez zautomatyzowaną infrastrukturę w postaci kodu.
- Umiejętności związane z operacjami oraz obsługą infrastruktury, które nie są uważane za przestarzałe w terminologii cloud native, są i będą ważniejsze niż kiedykolwiek.
- Nadal istnieje sens, aby centralny zespół budował i utrzymywał platformy oraz narzędzia, które umożliwiają skorzystanie z metodyki DevOps wszystkim pozostałym zespołom.
- Zaniknie wyraźne rozróżnienie między inżynierami oprogramowania a inżynierami operacji. Od teraz wszystko związane jest z oprogramowaniem i wszyscy jesteśmy inżynierami.

Rozdział 2. Pierwsze kroki z Kubernetes

Jeśli, podobnie jak my, szybko niecierpliwisz się długimi wypowiedziami na temat, dlaczego Kubernetes jest taki świetny, mamy nadzieję, że podobało Ci się nasze praktyczne podejście do tematu. Jeśli jesteś już zaawansowanym użytkownikiem narzędzi Docker lub Kubernetes, być może wybaczysz nam taki kurs przygotowawczy. Chcemy mieć pewność, że wszyscy czują się komfortowo w budowaniu i uruchamianiu kontenerów, oraz upewnić się, że posiadasz środowisko Kubernetes — zanim przejdziesz do bardziej zaawansowanych rzeczy.

Oto wiadomości, które powinieneś zapamiętac z tego rozdziału.

- Wszystkie przykłady kodu źródłowego (i wiele innych) są dostępne w repozytorium demonstracyjnym (https://github.com/cloudnativedevops/demo), powiązanym z tą książką.
- Narzędzie Docker pozwala budować kontenery lokalnie, dodawać je lub pobierać z rejestru kontenerów, takiego jak Docker Hub, oraz uruchamiać obrazy kontenerów lokalnie na komputerze.
- Obraz kontenera jest całkowicie określony w pliku Dockerfile; jest to plik tekstowy, który zawiera instrukcje dotyczące sposobu budowania kontenera.
- Docker Desktop pozwala na uruchomienie małego (jednowęzłowego) klastra Kubernetes na Twoim komputerze, który jest jednak zdolny do uruchamiania dowolnej aplikacji kontenerowej. Minikube to kolejna opcja.
- Narzędzie kubectl jest podstawowym narzędziem służącym do interakcji z klastrem Kubernetes i może być używane albo imperatywnie (aby np. uruchomić publiczny obraz kontenera i utworzyć niezbędne zasoby Kubernetes), albo deklaratywnie, aby zastosować plik konfiguracyjny Kubernetes w formacie YAML.

Rozdział 3. Opis Kubernetes

Kubernetes jest wszędzie! Nasza podróż przez rozległy krajobraz narzędzi, usług i produktów Kubernetes była z konieczności krótka, ale mamy nadzieję, że okaże się przydatna.

Chociaż nasze omówienie określonych produktów i funkcji jest tak aktualne, jak to możliwe, świat porusza się dość szybko i liczymy, że wiele się zmieni.

Uważamy jednak, że podstawową kwestią jest to, że nie warto samemu zarządzać klastrami Kubernetes, jeśli usługodawca może to zrobić lepiej i taniej.

Z naszego doświadczenia w doradztwie dla firm migrujących do Kubernetes często jest to zaskakujący pomysł dla wielu ludzi. Okazuje się, że organizacje, które zrobiły pierwsze kroki w kierunku własnego hostingu klastrów (używając narzędzi takich jak kops), nie zastanawiały się nad użyciem usługi zarządzanej, takiej jak GKE. Warto o tym pomyśleć.

Oto co powinieneś zapamietac z tego rozdziału.

- Klastry Kubernetes składają się z węzłów master, które obsługują warstwę sterowania, oraz węzłów worker, które obsługują zadania.
- Klastry produkcyjne muszą być wysoce niezawodne, co oznacza, że awaria węzła master nie spowoduje utraty danych ani nie wpłynie na działanie klastra.
- Jest długa droga od prostego klastra demonstracyjnego do takiego, który jest gotowy na krytyczne obciążenia produkcyjne; wysoka niezawodność, bezpieczeństwo i zarządzanie węzłami to tylko niektóre z problemów.
- Zarządzanie własnymi klastrami wymaga znacznej inwestycji czasu, wysiłku i wiedzy; nawet wtedy, kiedy wszystko opanujesz, możesz jeszcze popełniać błędy.
- Usługi zarządzane, takie jak Google Kubernetes Engine, wykonują za Ciebie wszystkie zadania, znacznie niższym kosztem niż samodzielny hosting.
- Usługi „pod klucz” stanowią dobry kompromis między własnym hostingiem a całkowicie zarządzanym Kubernetes; dostawcy „pod klucz” zarządzają za Ciebie węzłami master, podczas gdy uruchamiasz węzły worker na własnych komputerach.
- Jeśli musisz hostować własny klaster, kops jest dojrzałym i szeroko stosowanym narzędziem, które może udostępniać klastry produkcyjne w AWS oraz Google Cloud i zarządzać nimi.
- Jeśli możesz, powinieneś skorzystać z zarządzanego Kubernetes; jest to najlepsza opcja dla większości firm pod względem kosztów, nakładów pracy i jakości.
- Jeśli usługi zarządzane nie są dla Ciebie opcją, rozważ użycie usług „pod klucz” jako dobrego kompromisu.
- Nie hostuj swojego klastra bez uzasadnionych powodów biznesowych. Jeśli korzystasz z własnego hostingu, nie lekceważ czasu związanego z początkową konfiguracją i bieżącymi kosztami konserwacji.

Rozdział 4. Praca z obiektami Kubernetes

To nie jest książka o wewnętrznych elementach Kubernetes (przepraszamy, nie przyjmujemy zwrotów). Naszym celem jest pokazanie, co potrafi Kubernetes, i szybkie doprowadzenie do momentu, w którym możesz uruchomić rzeczywiste zadania produkcyjne. Warto jednak znać przynajmniej niektóre główne elementy maszyny, z którymi będziesz pracować, takie jak Pod czy Deployment. W tym rozdziale krótko przedstawiliśmy niektóre z najważniejszych.

Choć technologia ta jest fascynująca dla geeków takich jak my, jesteśmy również zainteresowani częścią praktyczną. Dlatego nie poruszyliśmy wszystkich informacji związanych z zasobami dostarczanymi przez Kubernetes — ponieważ jest ich wiele, a wielu z nich prawie na pewno nie będziesz potrzebować (przynajmniej jeszcze nie teraz).

Oto wiadomości, które powinieneś zapamietać z tego rozdziału.

Pod jest podstawową jednostką pracy w Kubernetes, określającą pojedynczy kontener lub grupę

komunikujących się kontenerów, wykonujących tę samą pracę.

- Deployment jest zasobem wysokiego poziomu Kubernetes, który deklaratywnie zarządza Podami — tworzy, uruchamia, aktualizuje i restartuje, jeśli trzeba.
- Serwis w Kubernetes jest odpowiednikiem load balancera lub serwera proxy; kieruje ruch do odpowiednich pasujących Podów poprzez jeden, znany i stały adres IP lub nazwę DNS.
- Scheduler w Kubernetes sprawdza, czy dany Pod nie jest jeszcze uruchomiony, znajduje odpowiedni węzeł i instruuje kubelet w tym węźle, aby uruchomił ten Pod.
- Zasoby, takie jak Deployment, są reprezentowane przez rekordy w wewnętrznej bazie danych Kubernetes; zewnętrznie zasoby te mogą być reprezentowane przez pliki tekstowe (znane jako manifesty) w formacie YAML; manifest jest deklaracją pożądanego stanu zasobu.
- kubectl jest głównym narzędziem służącym do interakcji z Kubernetes, umożliwiającym stosowanie manifestów, odpytywanie zasobów, wprowadzanie zmian, usuwanie zasobów i wykonywanie wielu innych zadań.
- Helm jest menadżerem pakietów Kubernetes; upraszcza konfigurowanie i wdrażanie aplikacji Kubernetes, umożliwiając korzystanie z jednego zestawu wartości (takich jak nazwa aplikacji lub port nasłuchiwania) oraz zestawu szablonów do generowania plików YAML — bez konieczności samodzielnego tworzenia plików YAML od zera.

Rozdział 5. Zarządzanie zasobami

Kubernetes radzi sobie z obciążeniami w niezawodny i wydajny sposób, bez faktycznej potrzeby ręcznej interwencji. Musisz tylko przekazać do komponentu scheduler dokładne szacunki zasobów kontenerów.

Dzięki temu czas poświęcony na rozwiązywanie problemów można znacznie lepiej wykorzystać — np. na tworzenie aplikacji. Dzięki, Kubernetes!

Zrozumienie, w jaki sposób Kubernetes zarządza zasobami, jest kluczem do prawidłowego budowania i uruchamiania klastra. Oto wiadomości do zapamiętania z tego rozdziału.

- Kubernetes przydziela zasoby procesora i pamięci do kontenerów na podstawie żądań i limitów.
- Żądania kontenera to minimalna ilość zasobów niezbędna do uruchomienia, jego limity określają maksymalną dozwoloną ilość.
- Minimalne obrazy kontenerów są szybsze w budowie, wdrażaniu i uruchamianiu; im mniejszy kontener, tym mniej potencjalnych luk w zabezpieczeniach.
- Sondy żywotności informują Kubernetes, czy kontener działa poprawnie; jeśli sonda żywotności kontenera zawiedzie, kontener zostanie zrestartowany.
- Sondy gotowości informują Kubernetes, że kontener jest gotowy i może obsłużyć żądania; jeśli sonda gotowości zawiedzie, kontener zostanie usunięty z Serwisów, które się do niego odwołują, i odłączony od ruchu użytkowników.
- Pod DisruptionBudgets pozwala ograniczyć liczbę Podów, które można zatrzymać jednocześnie podczas eksmisji, zachowując wysoką niezawodność Twojej aplikacji.
- Przestrzenie nazw to sposób logicznego podziału klastra na partycje; możesz utworzyć przestrzeń nazw dla każdej aplikacji lub grupy powiązanych aplikacji.
- Aby odwołać się do usługi w innej przestrzeni nazw, możesz użyć adresu DNS, takiego jak SERVICE.NAMESPACE.
- ResourceQuota pozwala ustawić ogólne limity zasobów dla danej przestrzeni nazw.
- LimitRanges określa domyślne żądania zasobów i limity dla kontenerów w przestrzeni nazw.
- Limity zasobów należy ustawiać tak, aby były wystarczające dla aplikacji, ale żeby nie były przekraczone w normalnym użytkowaniu.
- Nie należy przydzielać w chmurze więcej miejsca niż potrzeba oraz dużej przepustowości, chyba że ma to decydujące znaczenie dla wydajności Twojej aplikacji.
- Ustaw adnotacje właściciela na wszystkich swoich zasobach i regularnie skanuj klaster w poszukiwaniu zasobów nieposiadających przypisanego właściciela.
- Znajdź i wyczyść zasoby, które nie są używane (ale najpierw skontaktuj się z ich właścicielami).
- Instancje zastrzeżone mogą zaoszczędzić pieniądze, jeśli możesz zrobić plan w perspektywie długoterminowej.
- Instancje preemptible mogą zaoszczędzić pieniądze, ale bądź przygotowany na ich zniknięcie w krótkim okresie; skorzystaj z koligacji węzłów, aby trzymać wrażliwe na awarie Pody z dala od węzłów preemptible.

Rozdział 6. Operacje na klastrach

Naprawdę trudno ustalić rozmiar oraz konfigurację pierwszych klastrów Kubernetes. Istnieje wiele opcji i nie wiesz, czego będziesz potrzebować, dopóki nie zdobędziesz doświadczenia produkcyjnego.

Kilka pomocnych wskazówek do przemyślenia.

- Przed udostępnieniem produkcyjnego klastra Kubernetes zastanów się, ile węzłów oraz jakiego rozmiaru potrzebujesz.
- Potrzebujesz co najmniej trzech węzłów master (lub nie, jeśli korzystasz z usługi zarządzanej) i co najmniej dwóch (najlepiej trzech) węzłów worker. Tworzenie klastrów Kubernetes może się początkowo wydawać trochę drogie, zwłaszcza gdy wykonujesz tylko kilka małych zadań, ale nie zapomnij o zaletach wbudowanej odporności i skalowania.
- Klastry Kubernetes można skalować do wielu tysięcy węzłów i setek tysięcy kontenerów.
- Jeśli chcesz skalować do większych wartości, użyj wielu klastrów (czasem musisz to zrobić również ze względów bezpieczeństwa lub zgodności). Jeśli chcesz zreplikować obciążenia w różnych klastrach, możesz połączyć klastry za pomocą federacji.
- Typowy rozmiar instancji dla węzła Kubernetes to 1 procesor, 4 GiB RAM. Warto jednak mieszać kilka różnych rozmiarów węzłów.
- Kubernetes nie jest przeznaczony tylko do rozwiązań chmurowych; działa również na serwerach bare-metal. Jeśli masz takie serwery, dlaczego ich nie użyć?
- Możesz skalować klaster ręcznie w górę i w dół bez większych problemów i prawdopodobnie nie będziesz musiał robić tego zbyt często. Miło stosować automatyczne skalowanie, ale nie jest to ważne.
- Istnieje dobrze zdefiniowany standard dla dostawców i produktów Kubernetes: logo Certified Kubernetes. Jeśli go nie widzisz, zapytaj.
- Testowanie chaosu polega na losowym unieruchamianiu Podów i sprawdzaniu, czy aplikacja nadal działa. Jest to użyteczne, ale środowisko chmurowe i tak przeprowadza własne testy chaosu, bez pytania o Twoją zgodę.

Rozdział 7. Narzędzia Kubernetes

Oto kilka najważniejszych rzeczy, o których warto wiedzieć.

- kubectl zawiera kompletną i wyczerpującą dokumentację, dostępną za pomocą polecenia kubectl -h, a o każdym zasobie, polu lub funkcji Kubernetes, za pomocą polecenia kubectl explain.
- Jeśli chcesz wykonać skomplikowane filtrowanie i transformacje na wyniku wyjściowym polecenia kubectl, np. w skryptach, wybierz format JSON z opcją -o json; do obróki danych w formacie JSON możesz wykorzystać narzędzia, takie jak jq (do wysyłania zapytań).
- Opcja kubectl --dry-run=client, w połączeniu z -o YAML (aby uzyskać format YAML na wyjściu), pozwala używać poleceń imperatywnych do generowania manifestów Kubernetes; podczas tworzenia plików manifestów dla nowych aplikacji zyskujemy dużą oszczędność czasu.
- Można również przekształcić istniejące zasoby do manifestów YAML, używając flagi -o w poleceniu kubectl get.
- Polecenie kubectl diff powie Ci, co by się zmieniło, gdybyś zastosował manifest, bez faktycznej jego zmiany.
- Za pomocą polecenia kubectl logs możesz zobaczyć komunikaty wyjściowe i komunikaty o błędach dla dowolnego kontenera, przesyłać je strumieniowo w sposób ciągły za pomocą flagi --follow lub śledzić dzienniki wielu Podów przy użyciu narzędzia Stern.
- Aby rozwiązać problemy z kontenerami, możesz się do nich przyłączyć za pomocą polecenia kubectl attach lub uzyskać dostęp do powłoki w kontenerze za pomocą polecenia kubectl exec -it ... /bin/sh.
- Aby rozwiązać problemy, możesz uruchomić dowolny publiczny obraz kontenera za pomocą narzędzia kubectl run — w tym narzędzia BusyBox, które zawiera wszystkie Twoje ulubione polecenia uniksowe.
- Konteksty Kubernetes są jak zakładki, oznaczające Twoje miejsce w określonym klastrze i przestrzeni nazw; możesz wygodnie przełączać się między kontekstami i przestrzeniami nazw za pomocą narzędzi kubectx i kubens.
- Click to potężna powłoka Kubernetes, która zapewnia funkcjonalność kubectl, ale z dodanym stanem: zapamiętuje aktualnie wybrany obiekt z jednego polecenia i przenosi do drugiego, więc nie musisz go podawać za każdym razem.
- Kubernetes został zaprojektowany do automatyzacji i kontrolowania go za pomocą kodu. Gdy potrzebujesz wyjść poza to, co zapewnia kubectl, biblioteka client-go Kubernetes daje Ci pełną kontrolę nad każdym aspektem klastra za pomocą języka Go.

Rozdział 8. Uruchamianie kontenerów

Aby zrozumieć Kubernetes, musisz najpierw zrozumieć działanie kontenerów. W tym rozdziale przedstawiliśmy podstawowe dane na temat kontenerów; napisaliśmy, jak współpracują w Podach oraz jakie opcje są dostępne, aby kontrolować sposób działania kontenerów w Kubernetes.

Oto podstawowe zagadnienia.

- Kontener Linux, na poziomie jądra, to izolowany zestaw procesów z wydzielonymi zasobami. Z wnętrza kontenera wygląda to tak, jakby ten kontener miał dla siebie maszynę z systemem Linux.
- Kontenery nie są maszynami wirtualnymi. Każdy kontener powinien uruchamiać jeden proces podstawowy.
- Pod zwykle zawiera jeden kontener z podstawową aplikacją oraz opcjonalne kontenery pomocnicze, które ją obsługują.
- Specyfikacje obrazu kontenera mogą obejmować nazwę hosta rejestru, przestrzeń nazw repozytorium, repozytorium obrazów i tag, np. docker.io/cloudnatived/demo:hello. Wymagana jest tylko nazwa obrazu.
- W przypadku powtarzalnych wdrożeń zawsze podawaj tag obrazu kontenera. W przeciwnym razie podczas aktualizacji otrzymasz wszystko oznaczone tagiem latest.
- Programy w kontenerach nie powinny działać jako użytkownik root. Zamiast tego przydziel im uprawnienia zwykłego użytkownika.
- Możesz ustawić pole runAsNonRoot: true na kontenerze, aby zablokować dowolny kontener, który chce działać jako root.
- Inne przydatne ustawienia zabezpieczeń w kontenerach obejmują readOnlyRootFilesystem: true i allowPrivilegeEscalation: false.
- Mechanizm właściwości Linuksa zapewnia precyzyjną kontrolę uprawnień, ale domyślne ustawienia kontenerów są zbyt duże. Zacznij od usunięcia wszystkich właściwości kontenerów, a następnie przydziel tylko te, których kontener potrzebuje.
- Kontenery działające w tym samym Podzie mogą udostępniać dane, odczytując i zapisując zamontowany wolumin. Najprostszym woluminem jest emptyDir, który jest pusty i zachowuje swoją zawartość tylko tak długo, jak długo działa Pod.
- Wolumin PersistentVolume zachowuje swoją zawartość tak długo, jak to konieczne. Pody mogą dynamicznie udostępniać nowe PersistentVolumes za pomocą obiektów PersistentVolumeClaims.

Rozdział 9. Zarządzanie Podami

Oto podstawowe zagadnienia do zapamiętania.

- Etykiety to pary klucz-wartość, które identyfikują zasoby i mogą być używane z selektorami w celu dopasowania do określonej grupy zasobów.
- Koligacje węzłów przyłączają lub odłączają Pody do lub z węzłów o określonych atrybutach. Przykładowo można określić, że Pod może działać tylko w węźle o określonej strefie dostępności.
- Podczas gdy koligacje węzłów twardych mogą blokować działanie Poda, koligacje węzłów miękkich działają bardziej jako sugestie dla schedulera. Możesz stosować wiele koligacji miękkich z różnymi wartościami wag.
- Koligacje Podów wyrażają preferencję, aby Pody były uruchamiane w tym samym węźle, co inne Pody. Przykładowo Pody, które działają w tym samym węźle, mogą wyrazić to, używając wzajemnej koligacji Podów.
- Antykoligacje Podów odłączają inne Pody, zamiast je dołączać. Przykładowo antykoligacja replik tego samego Poda może pomóc równomiernie rozłożyć repliki w klastrze.
- Skazy to sposób oznaczania węzłów konkretnymi informacjami; zwykle dotyczy problemów lub awarii węzłów. Domyślnie Pody nie będą uruchamiane na węzłach ze skazą.
- Tolerancje pozwalają na uruchomienie Poda w węzłach o określonej skazie. Za pomocą tego mechanizmu można uruchamiać niektóre Pody tylko w dedykowanych węzłach.
- Zasoby DaemonSet pozwalają zaplanować jedną kopię Poda na każdym węźle (np. agenta rejestrującego).
- StatefulSets uruchamiają i zatrzymują repliki Poda w określonej numerowanej sekwencji, umożliwiając adresowanie każdego z nich za pomocą przewidywalnej nazwy DNS. Jest to idealne rozwiązanie dla aplikacji klastrowych, takich jak bazy danych.
- Zasoby Job uruchamiają Pod jeden raz (lub określoną liczbę razy). Podobnie zasoby CronJob okresowo uruchamiają Pod w określonych godzinach.
- Horizontal Pod Autoscaler obserwuje zestaw Podów, próbując zoptymalizować daną metrykę (np. wykorzystanie procesora). Zwiększają lub zmniejszają żądaną liczbę replik, aby osiągnąć określony cel.
- PodPreset mogą wstrzykiwać fragmenty konfiguracji do wszystkich wybranych Podów w czasie ich tworzenia. Przykładowo PodPreset może zostać wykorzystany do zamontowania określonego woluminu na wszystkich pasujących Podach.
- Niestandardowe definicje zasobów (CRD) umożliwiają tworzenie własnych niestandardowych obiektów Kubernetes w celu przechowywania dowolnych danych. Operatory to programy klienckie Kubernetes, które mogą implementować orkiestrację dla określonej aplikacji (np. MySQL).
- Zasoby Ingress kierują żądania do różnych usług, w zależności od zestawu reguł, np. na podstawie adresu URL żądania. Mogą również obsługiwać połączenia TLS dla Twojej aplikacji.
- Istio to narzędzie, które zapewnia zaawansowane funkcje sieciowe dla mikroserwisów i może być instalowane, podobnie jak każda aplikacja Kubernetes, za pomocą Helm.
- Envoy oferuje bardziej wyrafinowane funkcje równoważenia obciążenia niż standardowe usługi rozwiązania chmurowego. Świadczy także obsługę usługi mesh.

Rozdział 10. Konfiguracja i obiekty Secret

Oto najważniejsze rzeczy, których się nauczyłeś.

- Oddziel swoje dane konfiguracyjne od kodu aplikacji i wdróż je za pomocą Kubernetes Config-Map i Secret. W ten sposób nie musisz ponownie wdrażać aplikacji za każdym razem, gdy zmienisz hasło.
- Możesz pobrać dane do ConfigMap, pisząc je bezpośrednio w pliku manifestu Kubernetes lub za pomocą polecenia kubectl, aby przekonwertować istniejący plik YAML na specyfikację ConfigMap.
- Gdy dane znajdą się w ConfigMap, możesz wstawić je do środowiska kontenera lub do argumentów polecenia. Alternatywnie możesz zapisać dane do pliku zamontowanego w kontenerze.
- Obiekty Secret działają podobnie jak ConfigMaps. Z tym wyjątkiem, że dane są szyfrowane w stanie spoczynku i w wynikach polecenia kubectl występują w postaci zakodowanej.
- Prostym, elastycznym sposobem zarządzania obiektami Secret jest przechowywanie ich bezpośrednio w repozytorium kodu źródłowego w postaci zaszyfrowanej za pomocą Sops lub innego narzędzia do szyfrowania opartego na tekście.
- Nie zastanawiaj się nad zarządzaniem oiektami Secret, zwłaszcza na początku. Zacznij od czegoś prostego, co jest łatwe do skonfigurowania dla programistów.
- Tam, gdzie wiele aplikacji współdzieli obiekty Secret, możesz je przechowywać (zaszyfrowane) w chmurze i pobierać w czasie wdrażania.
- Do zarządzania obiektami Secret na poziomie przedsiębiorstwa potrzebujesz dedykowanej usługi, takiej jak Vault. Jednak nie zaczynaj od Vault, ponieważ możesz jej nie potrzebować. Zawsze możesz przenieść się do Vault później.
- Sops to narzędzie do szyfrowania, które działa z plikami klucz-wartość, takimi jak YAML i JSON. Klucz szyfrujący możesz uzyskać za pomocą GnuPG lub usług zarządzania kluczami w chmurze, takich jak Amazon KMS i Google Cloud KMS.

Rozdział 11. Bezpieczeństwo i kopia zapasowa

Bezpieczeństwo nie jest produktem ani celem końcowym, ale ciągłym procesem wymagającym wiedzy, przemyślenia i uwagi. Jeśli przeczytałeś ten rozdział i zrozumiałeś informacje w nim zawarte, wiesz wszystko, co musisz wiedzieć, aby bezpiecznie skonfigurować swoje kontenery w Kubernetes.

Jesteśmy pewni, że rozumiesz, iż powinien to być początek, a nie koniec Twojego procesu zapewnienia bezpieczeństwa.

Oto najważniejsze rzeczy, o których należy pamiętać.

- Kontrola dostępu oparta na rolach (RBAC) zapewnia dokładne zarządzanie uprawnieniami w Kubernetes. Upewnij się, że jest włączona i użyj ról RBAC, aby przyznać określonym użytkownikom i aplikacjom tylko minimalne uprawnienia, których potrzebują do wykonywania swoich zadań.
- Kontenery nie są wolne od problemów związanych z bezpieczeństwem i złośliwym oprogramowaniem. Za pomocą skanera sprawdź wszystkie kontenery, które uruchomisz w produkcji.
- Kubernetes jest świetny, ale nadal potrzebujesz kopii zapasowych. Użyj narzędzia Velero, aby wykonać kopię zapasową danych i stanu klastra. Jest także przydatne do przenoszenia zasobów między klastrami.
- kubectl to potężne narzędzie do sprawdzania i raportowania wszystkich aspektów klastra i obciążeń. Zaprzyjaźnij się z kubectl. Spędzisz z nim dużo czasu.
- Skorzystaj z konsoli przeglądarkowej dostawcy Kubernetes i kube-ops-view, aby uzyskać graficzny przegląd tego, co się dzieje. Jeśli korzystasz z pulpitu Kubernetes, zabezpiecz go tak dokładnie, jak swoje dane uwierzytelniające w chmurze i klucze kryptograficzne.

Rozdział 12. Wdrażanie aplikacji Kubernetes

Chociaż możesz wdrażać aplikacje w Kubernetes przy użyciu samych manifestów YAML, jest to niewygodne. Helm to potężne narzędzie, które może Ci w tym pomóc — pod warunkiem, że wiesz, jak z niego korzystać.

Obecnie opracowywanych jest wiele nowych narzędzi, które znacznie ułatwią wdrożenia w Kubernetes w przyszłości. Z niektórych funkcji można także skorzystać za pomocą Helm. Tak czy inaczej ważne jest, aby zapoznać się z podstawami korzystania z wykresu Helm.

- Wykres jest specyfikacją pakietu Helm, obejmującą metadane dotyczące pakietu, niektóre jego wartości konfiguracyjne oraz szablony obiektów Kubernetes, które odwołują się do tych wartości.
- Zainstalowanie wykresu tworzy wersję Helm. Za każdym razem, gdy instalujesz instancję wykresu, tworzona jest nowa wersja. Gdy ją aktualizujesz, Helm zwiększa numer wersji.
- Aby dostosować wykres Helm do własnych wymagań, utwórz plik wartości niestandardowych przesłaniający tylko te ustawienia, które są dla Ciebie ważne, i dodaj go do polecenia helm install lub helm upgrade.
- Możesz użyć zmiennej (np. environment), aby wybrać różne zestawy wartości lub obiekty Secret w zależności od środowiska wdrażania, takiego jak produkcyjne, testowe itd.
- Za pomocą Helmfile można deklaratywnie określić zestaw wykresów Helma i wartości, które mają być zastosowane w klastrze, oraz zainstalować lub zaktualizować je wszystkie za pomocą jednego polecenia.
- Helm może być używany razem z Sops do obsługi konfiguracji obiektów Secret dla Twoich wykresów. Może także używać funkcji do automatycznego kodowania Secret za pomocą base64 — standardowo stosowanego w Kubernetes.
- Helm nie jest jedynym dostępnym narzędziem do zarządzania manifestami Kubernetes. Są dostępne narzędzia Tanka oraz Kapitan, które używają Jsonnet, innego języka szablonów; kustomize stosuje inne podejście i zamiast interpolować zmienne, używa nakładek YAML do konfigurowania manifestów.
- Do szybkiego testowania i sprawdzania manifestów służy kubeval. Narzędzie kubeval sprawdzi poprawną składnię i typowe błędy w manifestach.

Rozdział 13. Proces tworzenia oprogramowania

Tworzenie aplikacji Kubernetes może być żmudne, jeśli trzeba zbudować i wdrożyć obraz kontenera w celu przetestowania każdej małej zmiany kodu. Narzędzia, takie jak Draft, Skaffold i Telepresence, sprawiają, że proces ten jest znacznie szybszy i przyspiesza cały rozwój.

Szczególnie wprowadzanie zmian w produkcji jest znacznie łatwiejsze w Kubernetes niż w tradycyjnych serwerach, pod warunkiem, że rozumiesz podstawowe pojęcia oraz wiesz, jak możesz je dostosować do swojej aplikacji.

- Domyślna strategia wdrażania Rolling Update w Kubernetes aktualizuje kilka Podów jednocześnie, czekając przed zamknięciem starego, aż każdy nowy Pod będzie gotowy.
- Aktualizacje Rolling Updates zapobiegają przestojom kosztem wydłużenia procesu wdrażania. Oznacza to również, że zarówno stare, jak i nowe wersje aplikacji będą działały jednocześnie podczas wdrażania.
- Możesz ustawić odpowiednie wartości dla maxSurge i maxUnavailable, aby dostosować bieżące aktualizacje. W zależności od wersji interfejsu API Kubernetes, którego używasz, wartości domyślne mogą, ale nie muszą być odpowiednie dla Twojej sytuacji.
- Strategia Recreate zamyka jednocześnie wszystkie stare Pody i uruchamia nowe. Jest to szybka opcja, ale powoduje przestoje, więc nie nadaje się do aplikacji zorientowanych na użytkownika.
- W przypadku wdrożenia niebiesko-zielonego wszystkie nowe Pody są uruchamiane, ale bez odbierania ruchu od użytkowników. Następnie cały ruch jest przełączany na nowe Pody za jednym razem, przed wycofaniem starych Podów.
- Wdrożenia rainbow są podobne do wdrożeń niebiesko-zielonych, ale jednocześnie obsługują więcej niż dwie wersje.
- W Kubernetes możesz skorzystać z wdrożenia niebiesko-zielonego oraz rainbow, dostosowując etykiety na swoich Podach i zmieniając selektor w obiekcie Serwis, aby skierować ruch do odpowiedniego zestawu Podów.
- Funkcje hook i wykres Helm zapewniają sposób na zastosowanie niektórych zasobów Kubernetes (zwykle Job) na określonym etapie wdrożenia, np. w celu uruchomienia migracji bazy danych. Funkcje hook mogą określać kolejność, w jakiej zasoby powinny być stosowane podczas wdrażania, i powodować zatrzymanie wdrażania, jeśli coś się nie powiedzie.

Rozdział 14. Ciągłe wdrażanie w Kubernetes

Konfigurowanie ciągłego procesu wdrażania aplikacji umożliwia konsekwentne, niezawodne i szybkie wdrażanie oprogramowania. Programiści powinni mieć możliwość przesyłania kodu do repozytorium, a wszystkie fazy kompilacji, testowania i wdrażania będą odbywać się automatycznie w scentralizowanym potoku (ang. *pipeline*).

Ponieważ jest tak wiele opcji oprogramowania CD i technik, nie możemy dać Ci jednego przepisu, który zadziała dla wszystkich. Chcieliśmy pokazać, dlaczego potok CD może przynieść Ci korzyści oraz dać kilka ważnych tematów do przemyśleń, jeżeli zdecydujesz się wdrożyć go we własnej firmie.

- Decyzja o wyborze narzędzi CD jest ważnym procesem przy budowie nowego potoku. Wszystkie narzędzia, o których wspominamy w tej książce, mogą być prawdopodobnie wykorzystane w prawie każdym istniejącym narzędziu CD.
- Jenkins, GitLab, Drone, Cloud Build i Spinnaker to tylko niektóre z popularnych narzędzi CD, które dobrze współpracują z Kubernetes. Istnieje również wiele nowszych narzędzi, takich jak Gitkube, Flux i Keel, które zostały utworzone specjalnie do automatyzacji wdrożeń w klastrach Kubernetes.
- Definiowanie kroków potoku kompilacji za pomocą kodu umożliwia śledzenie i modyfikowanie tych kroków wraz z kodem aplikacji.
- Kontenery wspomagają proces budowania komponentów za pośrednictwem środowiska testowego oraz ewentualnie produkcjnego, bez konieczności przebudowywania nowego kontenera.
- Nasz przykładowy potok korzystający z Cloud Build powinien łatwo dostosowywać się do innych narzędzi i typów aplikacji. Ogólne kroki budowania, testowania i wdrażania są w dużej mierze takie same w każdym potoku CD, niezależnie od użytych narzędzi lub rodzaju oprogramowania.

Rozdział 15. Obserwowalność i monitorowanie

O monitorowaniu można powiedzieć wiele. Nie mieliśmy miejsca, by powiedzieć tyle, ile chcieliśmy, ale mamy nadzieję, że ten rozdział dostarczył kilku użytecznych informacji na temat tradycyjnych technik monitorowania; napisaliśmy w nim, co mogą zrobić i czego nie mogą zrobić, a także,

jak rzeczy muszą się zmienić w środowisku cloud native. Pojęcie obserwowalności wprowadza szersze spojrzenie na tradycyjne pliki logowania i kontrole typu czarna skrzynka. Metryki stanowią ważną część tego obszaru, a w następnym i ostatnim rozdziale bardziej szczegółowo się im przyjrzymy.

- Monitoring typu czarna skrzynka obserwuje zachowanie zewnętrzne systemu w celu wykrycia przewidywalnych awarii.
- Systemy rozproszone ujawniają ograniczenia tradycyjnego monitorowania, ponieważ nie dotyczy ich stan typu działa/nie działa: istnieją w stałym stanie częściowo zdegradowanej usługi. Innymi słowy, na pokładzie statku nic nie jest do końca w porządku.
- Logi mogą być przydatne do rozwiązywania problemów po incydencie, ale trudno je skalować.
- Metryki otwierają nowy wymiar poza zwykłą odpowiedzią działa/nie działa i dostarczają ciągłych wartości liczbowych, które dotyczą setek lub tysięcy aspektów Twojego systemu.
- Metryki mogą pomóc Ci odpowiedzieć na pytanie dlaczego, a także zidentyfikować problematyczne trendy, zanim doprowadzą do awarii.
- Śledzenie rejestruje zdarzenia z dokładnym pomiarem czasu w cyklu życia pojedynczego żądania, aby pomóc w debugowaniu problemów z wydajnością.
- Obserwowalność to połączenie tradycyjnego monitorowania, rejestrowania, pomiarów i śledzeniaoraz wszystkich innych sposobów rozumienia systemu.
- Obserwowalność oznacza także przejście w kierunku kultury zespołowej inżynierii opartej na faktach i opiniach.
- Należy sprawdzać, czy Twoje usługi skierowane do użytkowników działają, korzystając z zewnętrznych kontroli black-box. Nie próbuj tworzyć własnych rozwiązań; skorzystaj z usługi monitorowania innej firmy, np. takiej jak Uptime Robot.
- Dziewiątki nie mają znaczenia, jeśli użytkownicy nie są zadowoleni.

Rozdział 16. Metryki w Kubernetes

W świecie cloud native bez odpowiednich danych i danych obserwacyjnych bardzo trudno stwierdzić, co się dzieje. Z drugiej strony, gdy obserwujemy dużo metryk, zbyt wiele informacji może być tak samo bezużytecznych, jak zbyt mało.

Przede wszystkim sztuką jest zebranie odpowiednich danych, przetworzenie ich we właściwy sposób, wykorzystanie do odpowiedzi na właściwe pytania, wizualizacja w odpowiedni sposób i zastosowanie do ostrzeżenia właściwych ludzi we właściwym czasie.

Po przeczytaniu tego rozdziału musisz pamiętać o niżej spisanych zasadach.

- Skoncentruj się na kluczowych metrykach dla każdej usługi: żądaniach, błędach i czasie trwania (RED) i dla każdego zasobu: wykorzystanie, nasycenie i błędy (USE).
- Poinstruuj swoje aplikacje tak, aby wyświetlały niestandardowe metryki zarówno dla wewnętrznej obserwowalności, jak i dla biznesowych wskaźników KPI.
- Przydatne metryki Kubernetes, na poziomie klastra, obejmują liczbę węzłów, liczbę Podów na węzeł i zużycie zasobów przez węzły.
- Na poziomie obiektu Deployment śledź te obiekty Deployment i repliki, szczególnie niedostępne repliki, które mogą wskazywać na problem z pojemnością.
- Na poziomie kontenera śledź zużycie zasobów na kontener, stany żywotności lub gotowości, restarty, ruch sieciowy i błędy sieciowe.
- Zbuduj pulpit dla każdej usługi, korzystając ze standardowego layoutu i głównego radiatora informacji, który zgłasza istotne parametry całego systemu.
- Jeśli ostrzegasz o metrykach, alerty powinny być pilne, ważne i wykonalne. Alarmowy chaos powoduje zmęczenie i obniża morale zespołu.
- Śledź i przeglądaj liczbę pilnych zgłoszeń, które otrzymuje Twój zespół, szczególnie w trakcie snu i weekendu.
- W świecie cloud native standardowym rozwiązaniem związanym z metrykami jest Prometheus. Prawie wszystkie narzędzia stosują format danych Prometheus.
- Usługi zarządzane związane z metrykami to także Google Stackdriver, Amazon Cloudwatch, Datadog i New Relic.

Rozdział 17. OpenShift

OpenShift to platforma open-source do tworzenia, wdrażania i zarządzania aplikacjami kontenerowymi. OpenShift to implementacja modelu PaaS (Platform as a Service).

Umożliwia programistom budowanie i wdrażanie kontenerów w formacie Docker w zintegrowanym środowisku programistycznym (IDE), a następnie zarządzanie nimi za pomocą platformy Kubernetes. Platforma umożliwia również automatyczne lub ręczne skalowanie skonteneryzowanych aplikacji.

OpenShift vs. Kubernetes – najistotniejsze różnice

Pporównanie Kubernetesa z OpenShiftem może prowadzić do nieporozumień, ponieważ Kubernetes jest integralnym elementem OpenShifta. Poniżej przedstawiono kilka najważniejszych różnic między OpenShift i Kubernetes.

Poniższe różnice są przetłumaczone z wpisu Gliada Maayana.

Produkt vs. Projekt

Subskrypcja OpenShift umożliwia użytkownikom korzystanie z **płatnego wsparcia**. Subskrypcja obejmuje również **CloudForms**, który pomaga organizacjom zarządzać infrastrukturą prywatną, publiczną i wirtualną. Użytkownicy muszą okresowo odnawiać subskrypcję w oparciu o rozbudowę klastra.

Kubernetes oferuje model **samowsparcia**. W sytuacjach problematycznych użytkownicy mogą zwrócić się do **zewnętrznych ekspertów** i **społeczności**.

Bezbieczeństwo

OpenShift ma silniejsze polityki bezpieczeństwa niż Kubernetes. Polityka bezpieczeństwa OpenShift ogranicza możliwość uruchamiania prostych obrazów kontenerów, jak również wielu oficjalnych obrazów. OpenShift wymaga określonych uprawnień, aby utrzymać minimalny poziom bezpieczeństwa. W rezultacie, aby wdrożyć więcej aplikacji, trzeba nauczyć się polityk.

Procesy uwierzytelniania i autoryzacji są również różne. Konfiguracja i konfiguracja uwierzytelniania Kubernetes wymaga wiele wysiłku. Z kolei OpenShift oferuje zintegrowany serwer dla lepszego uwierzytelniania.

Obie platformy oferują autoryzację poprzez kontrolę dostępu opartą na rolach (RBAC). Kontekst bezpieczeństwa jest istotnym elementem w Kubernetes. OpenShift posiada oddzielne ograniczenie kontekstu bezpieczeństwa (SCC). Podejście OpenShift przewyższa podejście Kubernetes w aspekcie bezpieczeństwa.

Web-UI

Interfejs użytkownika oparty na przeglądarce internetowej (UI) jest ważny dla efektywnego administrowania klastrem.

Musisz zainstalować dashboard Kubernetes oddzielnie i użyć kube-proxy, aby przekierować port twojego lokalnego komputera do serwera administracyjnego klastra. Dodatkowo, musisz ręcznie utworzyć token na okaziciela, aby zapewnić autoryzację i uwierzytelnienie, ponieważ dashboard nie posiada strony logowania.

Konsola internetowa OpenShift posiada stronę logowania. Można łatwo uzyskać dostęp do konsoli i tworzyć lub zmieniać większość zasobów za pomocą formularza. Możesz również wizualizować serwery, projekty i role klastra.

Podejście do wdrożeń (deployment)

Wdrażanie Kubernetes odbywa się za pomocą obiektów deployment. Możesz wewnętrznie zaimplementować obiekty deployment przez kontrolery i używać ich do aktualizacji strąków. Obiekty deployment Kubernetes mogą obsługiwać wielokrotne i współbieżne aktualizacje.

Instalacja OpenShift jest wykonywana za pomocą polecenia DeploymentConfig. Nie można zaimplementować DeploymentConfig za pomocą kontrolerów; musisz użyć dedykowanych logik dla pod.

DeploymentConfig nie obsługuje wielokrotnych aktualizacji, tak jak obiekty Kubernetes. OpenShift DeploymentConfig ma jednak inne zalety, takie jak wersjonowanie i wyzwalacze, które napędzają zautomatyzowane wdrożenia.

CI/CD

Zarówno OpenShift jak i Kubernetes można wykorzystać do budowy rurociągów CI/CD. Jednak żadna z tych platform nie stanowi pełnego rozwiązania CI/CD. Musisz zintegrować obie platformy z dodatkowymi narzędziami, takimi jak zautomatyzowane testy i monitoring oraz serwery CI, aby zbudować pełny pipeline CI/CD.

Proces ten jest łatwiejszy w OpenShift, ponieważ oferuje on certyfikowany kontener Jenkinsa, którego można użyć jako serwera CI.

Zwykły Kubernetes nie oferuje oficjalnego rozwiązania integracyjnego CI/CD. Musisz zintegrować narzędzia innych firm, takie jak CircleCI, aby zbudować rurociąg CI/CD z Kubernetes.

Zintegrowany Rejestr Obrazów

Kubernetes umożliwia skonfigurowanie własnego rejestru Docker, ale nie otrzymujesz zintegrowanego rejestru obrazów.

OpenShift zapewnia zintegrowany rejestr obrazów, którego można używać z Red Hat lub Docker Hub. Rejestr obrazów ma konsolę, w której można wyszukiwać informacje o obrazach i strumieniach obrazów do projektów w klastrze.

Aktualizacje

Na obu platformach można aktualizować istniejące klastry Kubernetes zamiast odbudowywać je od podstaw. Jednak harmonogramy aktualizacji w OpenShift i Kubernetes są różne.

W OpenShift nie dostajemy powiadomień o konieczności aktualizacji do nowej wersji Kubernetes. Musisz użyć systemu zarządzania pakietami Red Hat Enterprise Linux, aby zaktualizować OpenShift do najnowszej wersji.

Kubernetes zazwyczaj używa polecenia kubeadm upgrade, aby zaktualizować się do nowszej wersji. Na obu platformach, przed aktualizacją należy wykonać kopię zapasową istniejącej instalacji.

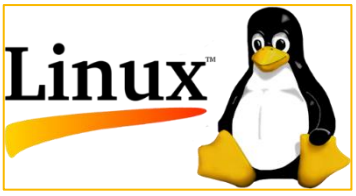
Instalacja

Zarówno Kubernetes jak i OpenShift pozwalają na łatwe wdrażanie i zarządzanie skonteneryzowanymi aplikacjami. Mają one jednak pewne różnice.

Instalacja Kubernetes jest skomplikowana i często wymaga rozwiązania innej firmy.

OpenShift ma wbudowaną platformę Kubernetes, która ułatwia proces instalacji, ale jest ograniczona do dystrybucji Red Hat Linux.

Rozdział 18. Podstawy Linuxa



Powłoka systemu – *shell*

Program komputerowy pośredniczący pomiędzy systemem operacyjnym lub aplikacjami a użytkownikiem, przyjmując jego polecenia i zwracając wyniki działania programów.
To pośrednictwo nie jest obowiązkowe (programy mogą być bardziej „samodzielne").

Powłoki systemowe możemy podzielić na:

- **tekstowe** – często same zawierają podstawowe polecenia, gdy jednak wydane przez użytkownika polecenie nie jest wbudowane, uruchamiany jest program zewnętrzny;
- **graficzne** – mają zwykle postać menedżera plików kontrolowanego przy pomocy myszy i pozwalającego w łatwy sposób wykonywać najczęstsze operacje.

System plików

Wszystkie katalogi **./bin** są grupowane pod zmienną środowiskową **$PATH**.
Jeśli potrzebujesz sprawdzić dokładną lokalizację binarki komendy użyj komendy **$ witch**.

- **/bin**

Zawiera pliki binarne (***binaries***) i wykonywalne (***executes***). Można je wywołać w komendach cały czas.

- **/sbin**

Zawiera systemowe **pliki binarne** uruchamialne tylko przez ***roota***.

- **/lib**

Biblioteki, z których korzystają binarki z katalogów /bin i /sbin.

- **/user**
  - **./bin, ./sbin**

Aplikacje i pliki **binarne** zbędne dla systemu, lecz przeznaczone dla końcowego użytkownika.

  - **./local/bin**

Zawiera pliki **binarne**, które sam możesz **skompilować manualnie**.
Z założenia jest to miejsce, które nie konfliktuje z innymi binarkami (np. systemowymi).

- **/etc**

Modyfikowalna konfiguracja tekstowa (***E**ditable **T**ext **C**onfiguration*).
Pliki konfiguracji nie tylko systemu, ale też zainstalowanego oprogramowania. Często mają rozszerzenie .conf.

- **/home**

Katalogi domowe użytkowników, np. /home/alice. Aby je modyfikować, trzeba być zalogowanym jako użytkownik danego katalogu domowego lub jako *root*. Jest to startowa lokacja po zalogowaniu i jest przypisana pod ~.

- **/boot**

Zawiera **pliki** potrzebne **do uruchomienia systemu operacyjnego**. Zawiera m.in. jądro Linuxa (***Linux kernel***).

- `/dev`

Zawiera **pliki sprzętowe** (***device files***).
Możesz tu komunikować się z urządzeniami i sterownikami tak jakby były zwykłymi plikami.

- `/opt`

Zawiera instalacje **pakietów dodatkowego oprogramowania**, czyli takie, które nie jest częścią systemu.

- `/var`

Zawiera zmienne **pliki**, które są **modyfikowane w trakcie** użytkowania systemu, takie jak **logi** i pliki podręczne (***cach***).

- `/temp`

Zawiera **pliki tymczasowe**, które nie są zachowanie pomiędzy restartem systemu.

- `/proc`

Wirtualny system plików, który jest tworzony w pamięci na bieżąco przez **jądro Linuxa**, aby **śledzić** uruchomione **procesy**.

Uprawnieni i własności plików i katalogów

Dla zapewnienia skutecznego bezpieczeństwa, Linux dzieli autoryzację na 2 poziomy:

- **własność**,
- **uprawnienie**.

Własność – *ownership*

Każdy plik i katalog w systemie Linux ma przypisane 3 rodzaje właścicieli.

- user

Użytkownik jest właścicielem pliku. Domyślnie osoba, która utworzyła plik staje się jego właścicielem.

- group

Grupa może zawierać wielu użytkowników. Wszyscy użytkownicy należący do danej grupy będą mieli dostęp do pliku z takimi samymi uprawnieniami grupy Linux.

- other

Każdy inny użytkownik, który ma dostęp do pliku. Osoba ta nie utworzyła pliku ani nie należy do grupy użytkowników, która mogłaby posiadać plik. W praktyce oznacza to wszystkich innych. Stąd, gdy ustawisz uprawnienia dla innych, jest to również określane jako uprawnienia dla świata.

Uprawnienia – *permissions*

Każdy plik i katalog w systemie Linux ma 3 uprawnienia zdefiniowane dla wszystkich 3 rodzajów właścicieli:

- **read**,
- **write,**
- **execute.**

- read

To uprawnienie daje Ci możliwość otwarcia i odczytania pliku.

Uprawnienie odczytu na katalogu daje Ci możliwość wylistowania jego zawartości.

- write

Uprawnienie zapisu daje Ci prawo do modyfikowania zawartości pliku.

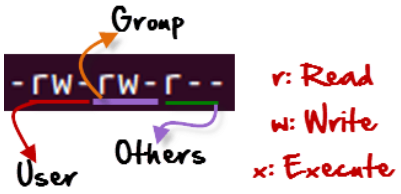
Uprawnienie zapisu w katalogu daje Ci możliwość dodawania, usuwania i zmiany nazw plików przechowywanych w katalogu.

Rozważmy scenariusz, w którym masz uprawnienia do zapisu na pliku, ale nie masz uprawnień do zapisu w katalogu, w którym plik jest przechowywany. Będziesz mógł modyfikować zawartość pliku. Ale nie będzie można zmienić nazwy, przenieść lub usunąć plik z katalogu.

- execute

W systemach Linux nie można uruchomić programu, jeśli nie jest ustawione uprawnienie ***execute***. Jeśli uprawnienie *execute* nie jest ustawione, możesz nadal być w stanie zobaczyć i zmodyfikować kod programu (pod warunkiem, że uprawnienia do odczytu i zapisu są ustawione), ale nie uruchomić go.

Zarządzanie własnościami i uprawnieniami



Komenda `$ ls -l` wyświetla własności i uprawnienia.

Nie możesz mieć dwóch grup posiadających ten sam plik.

- chown

Aby zmienić właściciela pliku/katalogu, możesz użyć następujących komand:

```
$ chown user somefile
```

lub

```
$ chown user:group somefile
```

- chgrp

chgrp zmienia grupę:

```
$ chgrp group somefile
```

- groups

Plik /etc/group zawiera wszystkie grupy zdefiniowane w systemie. Możesz użyć polecenia:

```
$ groups
```

aby znaleźć wszystkie grupy, których jesteś członkiem.

W Linuksie nie ma zagnieżdżonych grup. Jedna grupa nie może być podgrupą innej.

- newgrp

Zienia bieżący identyfikator rzeczywistej grupy na grupę o podanej nazwie lub na grupę domyślną wymienioną w /etc/passwd, jeśli nie podano nazwy grupy.

`$ newgrp` próbuje także dodać grupę do zestawu grup użytkownika.

- chmod

Używając tej komendy możemy ustawić uprawnienia:

```
$ chmod <permissions> <filename>
```

Istnieją 2 sposoby użycia tego polecenia:

- tryb bezwzględny (numeryczny) – ***absolute (numeric)***

W tym trybie uprawnienia do plików jako trzycyfrowa liczba ósemkowa:

NUMBER		PERMISSION TYPE	SYMBOL
0	0	no permission	---
1	**1**	**execute**	**--x**
2	**2**	**write**	**-w-**
3	2+1	write + execute	-wx
4	**4**	**read**	**r--**
5	4+1	read + execute	r-x
6	4+2	read +write	rw-
7	4+2+1	read + write + execute	rwx

gdzie `$ chmod 764 somefile` znaczy, że:

- **właściciel** może: *odczytywać, zapisywać, wykonywać;*
- **grupa** użytkowników może: *zapisywać;*
- **inni/świat** może: *odczytywać.*

- tryb symboliczny – symbolic

W trybie bezwzględnym zmieniasz uprawnienia dla wszystkich 3 właścicieli. W trybie symbolicznym, można modyfikować uprawnienia określonego właściciela. Wykorzystuje on symbole matematyczne do modyfikacji uprawnień plików uniksowych.

`$ chmod u=rwx somefile` `$ chmod g+x somefile` `$ chmod o-w somefile`

UŻYTKOWNICY	
u	user (owner)
g	group
o	other (world)
a	all

OPERATORY	
+	Dodaje uprawnienie.
-	Usuwa uprawnienia.
=	Nadpisuje wcześniej ustawione uprawnienia.

Zabezpieczenie możliwości – *capability sets*

Normalnie systemy Unixowe używają dwóch rodzajów procesów:

- **uprzywilejowanych**,
- **nieuprzywilejowanych**.

Pierwsza kategoria jest zwykle używana do celów administracyjnych, takich jak uruchamianie i zatrzymywanie innych procesów, dostrajanie jądra i otwieranie gniazd sieciowych (*socket*).

setuid – co to i dlaczego nie?

Flaga **setuid** jest ustawiana na plikach wykonywalnych na poziomie właściciela pliku. Plik wykonywalny z ustawionym bitem **setuid** jest wykonywany przez zwykłych użytkowników z takimi przywilejami jakie posiada właściciel pliku.

```
$ ls -l /usr/bin/su
```

res: `-rwSr-xr-x. 1 root root 32072 2016-08-02  /usr/bin/su`

Możemy przeszukać wszystkie pliki naszego systemu w poszukiwaniu plików posiadających ustawiony bit setuid:

```
$ find / -perm -4000
```

res: /usr/bin/fusermount
/usr/bin/mount

Zazwyczaj sensowne jest pozwolenie zaufanej binarce na korzystanie z uprawnień *roota* do wykonywania, jednak zawsze istnieje szansa, że może ona zawierać błędy. Tak więc nawet najmniejszy błąd w binarnce z ustawionym **setuid** może spowodować złamanie zabezpieczeń systemu.

capability sets

Użycie **zbioru możliwości** (***capability sets***) daje binarce uprawnienia *roota* tylko do ograniczonego zestawu wywołań systemowych. Więc nawet jeśli dojdzie do wycieku oprogramowania, może ono nie zostać nadużyte.

Każdy wątek procesu posiada trzy zbiory ***capability***

- efektywny – ***effective***

Możliwości używane przez jądro do sprawdzania uprawnień dla danego wątku.

- uprzywilejowany – ***permited***

Jeśli wątek porzuci jakąś możliwość (***drop capability***) ze swego dozwolonego zestawu, to nigdy nie może jej ponownie uzyskać (chyba, że wykona `exec()` program `set-user-ID-root`).

- dziedziczony – ***inheritable***

Dziecko utworzone przez `fork()` dziedziczy kopie zestawów właściwości swojego rodzica.

Oznacza to, że dla zwykłego programu binarnego, który nie będzie tworzył procesów potomnych, wystarczą ***permited***. Dla procesów, które rozwidlają (`fork`) inne procesy, należy użyć zestawu ***inheritable***.

Ustawianie *capability*

Zakładamy, że chcemy, aby zwykli użytkownicy nadal używali komendy `$ ping`, ale nie chcemy uruchamiać jego procesu jako uprzywilejowanego. Dodajmy więc możliwość (***capability***), która pozwoli na otwarcie gniazda sieciowego.

Zajrzyjmy do strony $ man i znajdźmy **capability**:

```
CAP_NET_RAW
* use RAW and PACKET sockets;
* bind to any address for transparent proxying.
```

Teraz możemy zaplikować tę wiedzę:

```
$ sudo setcap cap_net_raw+p /bin/ping
```

```
$ sudo getcap /bin/ping
```

res: `/bin/ping = cap_net_raw+p`

Zarządanie procesami

Przeglądanie uruchomionych procesów

- top

Najprostszym sposobem, aby dowiedzieć się, jakie **procesy** są **uruchomione** na serwerze, jest polecenia `$ top`:

```
top - 15:14:40 up 46 min,  1 user,  load average: 0.00, 0.01, 0.05
Tasks:  56 total,   1 running,  55 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1019600k total,   316576k used,   703024k free,     7652k buffers
Swap:        0k total,        0k used,        0k free,   258976k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      20   0 24188 2120 1300 S  0.0  0.2   0:00.56 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/0
```

- htop

Ulepszoną wersją top jest `$ htop`:

```
  Mem[||||||||||||           49/995MB]     Load average: 0.00 0.03 0.05
  CPU[                         0.0%]     Tasks: 21, 3 thr; 1 running
  Swp[                        0/0MB]     Uptime: 00:58:11

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1259 root       20   0 25660  1880  1368 R  0.0  0.2  0:00.06 htop
    1 root       20   0 24188  2120  1300 S  0.0  0.2  0:00.56 /sbin/init
  311 root       20   0 17224   636   440 S  0.0  0.1  0:00.07 upstart-udev-brid
```

- ps

Powyższe narzędzia nie zawsze są wystarczająco elastyczne, aby odpowiednio pokryć wszystkie scenariusze. Rozbudowane polecenie `$ ps` jest często odpowiedzią na te problemy. Narzędzie wyświetla **snapshot** procesów.

Bez argumentów wyświetla tylko wszystkie procesy związane z bieżącym użytkownikiem i sesją terminala:

```
$ ps
  PID TTY          TIME CMD
 1017 pts/0    00:00:00 bash
 1262 pts/0    00:00:00 ps
```

Opcje `aux` pokazują procesy należące do wszystkich użytkowników (niezależnie od ich powiązania z terminalem):

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  24188  2120 ?        Ss   14:28   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?        S    14:28   0:00 [kthreadd]
. . .
```

Opcje `axjf` pokazują widok drzewa, gdzie zilustrowane są relacje hierarchiczne:

```
$ ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S        0   0:00  \_ [ksoftirqd/0]
    2     6     0     0 ?           -1 S        0   0:00  \_ [migration/0]
. . .
```

Jak widać, proces `kthreadd` jest rodzicem procesu `ksoftirqd/0` i pozostałych.

- free

Wyświetla całkowitą ilość wolnej i używanej**:**

  - **pamięci fizycznej**,
  - **pamięci wymiany** (***swap***) w systemie,
  - **bufory** używane przez jądro.

- df

Wyświetla ilość **miejsca** dostępnego **na dysku** w systemie plików zawierającym każdy argument nazwy pliku. Jeżeli nie podano nazwy pliku, wyświetlane jest miejsce dostępne na wszystkich aktualnie zamontowanych systemach plików. Domyślnie przestrzeń dyskowa wyświetlana jest w blokach 1K.

- systemctl

Może być użyty do przeglądania i kontrolowania stanu systemu ***systemd*** i menedżera usług.
Serwisy wykorzystają do pracy przynajmniej jeden proces.

`$ systemctl` może wykonywać operacje na **serwisach,** m.in.:

- uruchamiać (**start**),
- przeładowywać (**reload**),
- zatrzymywać (**stop**).

Automatyczne uruchamianie serwisów przy starcie systemu może być przez `$ systemctl`:

- ustawiane (**enable**),
- usuwane (**disable**),
- sprawdzane (**is-enabled**).

Identyfikatory procesów

W Linuksie i systemach unikso-podobnych każdy proces ma przypisany identyfikator procesu, czyli **PID**.
Jest to sposób, w jaki system operacyjny identyfikuje i śledzi procesy.

Szybkim sposobem na uzyskanie **PID** procesu jest polecenie `pgrep`:

```
$ pgrep bash
```

res: `1017`

Pierwszy proces zrodzony przy starcie systemu, zwany **init**, otrzymuje **PID 1**.
Proces ten jest następnie odpowiedzialny za wywołanie każdego innego procesu w systemie.
Późniejsze procesy otrzymują większe numery **PID**.

Rodzicem procesu jest proces, który był odpowiedzialny za jego wywołanie. Procesy macierzyste mają **PPID**, który można zobaczyć w nagłówkach kolumn w wielu aplikacjach do zarządzania procesami.

Zależności rodzic-dziecko

Tworzenie **procesu potomnego (*child process*)** odbywa się w dwóch krokach:

- `fork()` – tworzy nową przestrzeń adresową i kopiuje zasoby należące do rodzica poprzez **copy-on-write**, aby były dostępne dla procesu potomnego;
- `exec()` – ładuje plik wykonywalny do przestrzeni adresowej i wykonuje go.

W przypadku, gdy proces potomny umiera przed swoim rodzicem, proces ten staje się **zombie**, dopóki rodzic nie zbierze informacji o nim lub nie poinformuje jądra, że nie potrzebuje tych informacji. Zasoby procesu potomnego zostaną wtedy uwolnione.

Jeśli jednak proces rodzica umrze przed dzieckiem, to zostanie ono przejęte przez **init**, choć może też zostać przypisane do innego procesu.

Sygnały do procesów – *kill & killall*

Sygnały są sposobem informowania programów na poziomie systemu operacyjnego, aby zakończyły działanie lub zmodyfikowały swoje zachowanie. Wszystkie procesy w Linuksie odpowiadają na sygnały.
Każdy sygnał ma przypisany numer, który może być przekazany zamiast nazwy.

Najbardziej powszechnym sposobem przekazywania sygnałów do programu jest polecenie **`kill`.**

```
$ kill [-s sigspec | -n signum | -sigspec] pid
```

Lista dostępnych nazw **sygnałów** (do użycia bez prefiksu `SIG`):

```
$ kill -l
```

Aby wysłać **sygnał zabicia do wszystkich** instancji procesu według nazwy (musi to być dokładna nazwa), użyj:

```
$ killall
```

Wszystkie sygnały oprócz **SIGKILL** i **SIGSTOP** mogą być przechwycone przez proces, pozwalając na czyste wyjście.

- TERM – *terminate*

`$ kill 1017` lub `$ kill -TERM 1017` lub `$ kill -15 1017`

Jest to specjalny sygnał, który nie jest wysyłany do programu. Zamiast tego, jest on przekazywany do jądra systemu operacyjnego, które **zamyka** proces **czysto** (***gracefully***). Jest używany do omijania programów, które ignorują wysyłane do nich sygnały.

- KILL – *kill*

Sygnalizuje systemowi operacyjnemu natychmiastowe zakończenie programu, który nie ma szans na przechwycenie sygnału, tzw. **zamykanie brutalne** (***forcefully***).

- HUP – *hang up*

Wiele *daemonów* **przeładowuje** się zamiast kończyć pracę.
Np. Apache przeładowuje swój plik konfiguracyjny i wznawia serwowanie treści.

- INT – *interrupt*

Ten sygnał jest zwykle inicjowany przez użytkownika naciskającego **[Ctrl]** + **[C]**.

- STOP & CONT – *stop & continue*

SIG**STOP** sygnalizuje systemowi operacyjnemu wstrzymanie programu do momentu otrzymania sygnału SIG**CONT**.

Przechwytywanie sygnałów – *trap*

`$ trap` wykonuje automatycznie polecenia po otrzymaniu sygnałów od procesów lub systemu operacyjnego. Może być użyte do wykonania czyszczenia w przypadku przerwania przez użytkownika lub innych działań.

Ustaw pułapkę na wykonywanie poleceń, gdy wykryty zostanie jeden lub więcej sygnałów:

```
$ trap 'echo "Caught signal"' SIGHUP SIGINT
```

Lista aktywnych pułapek dla bieżącej powłoki:

```
$ trap -p
```

Usuń aktywne pułapki:

```
$ trap - SIGHUP SIGINT
```

Priorytety procesów

Linux kontroluje priorytet poprzez wartość zwaną *uprzejmością* (***niceness***).

Zadania o wysokim priorytecie są uważane za mniej uprzejme, ponieważ nie dzielą się tak dobrze zasobami. Procesy o niskim priorytecie są miłe, ponieważ nalegają na pobieranie tylko minimalnych zasobów.

W odpowiedzi na polecenie `$ top` pojawiła się kolumna oznaczona jako **`NI`**. Jest to wartość uprzejmości (***nice* value**) procesu.

Wartości ***niceness*** zawierają się w przedziale:

- od **-20** (najbardziej korzystne dla procesu)
- do **19** (najmniej korzystne dla procesu).

Aby uruchomić program z określoną wartością **nice**, możemy użyć polecenia:

```
$ nice -n 15 bin/bash some-script.sh
```

Aby zmienić wartość **nice** programu, który jest już wykonywany, używamy narzędzia:

```
$ renice 0 1017
```

Podczas gdy `$ nice` z konieczności operuje na **nazwie** polecenia, `$ renice` operuje przez wywołanie procesu **PID.**

Rozdział 19. Podstawy Maven

Czym jest Maven

Maven jest zasadniczo **narzędziem do zarządzania projektu** i jako taki zapewnia sposób na pomoc w zarządzaniu:

- buildami,
- dokumentacją,
- raportowaniem,
- zależnościami,
- wydaniami,
- dystrybucją

poprzez zastosowanie standardowych konwencji i praktyk przyspieszających cykl rozwoju.

Do stworzenia projektu Maven wykorzystamy mechanizm **archetypów**. Jest to **szablon projektu**, który w połączeniu z danymi wprowadzonymi przez użytkownika tworzy działający projekt Maven dostosowany do wymagań, razem z plikiem **pom.xml**.

POM (*Project Object Model*) jest podstawową jednostką pracy w Maven.
Jest jest w zasadzie miejscem, gdzie znajdują się wszystkie informacje o projekcie.

Pluginy – *<plugins>*

Jeśli chcesz dostosować kompilację projektu Maven do swoich potrzeb, możesz to zrobić dodając lub zmieniając konfigurację pluginów w ***pom.xml***

Plugin zostanie automatycznie pobrany i użyty.
Jeśli nie zostanie podana jego wersja, zostanie u użyta najnowsza dostępna.

Znacznik `<configuration>` stosuje podane parametry do każdego celu (***goal***) z kompilatora wtyczki.
Możliwe jest również dodawanie nowych celów do procesu oraz konfigurowanie konkretnych celów.

Zewnętrzne zależności – *<dependencies>*

Dla każdej zewnętrznej zależności należy zdefiniować co najmniej 4 rzeczy: `groupId`, `artifactId`, `version`, `scope`.

`groupId`, `artifactId` i `version` są takie same jak te podane w ***pom.xml*** dla projektu, który zbudował daną zależność.

Element `scope` wskazuje, w jaki sposób nasz projekt używa tej zależności i może zawierać wartości takie jak:

- `compile`,
- `test`,
- `runtime`.

Cykl życia budowania – *build lifecycle*

Wbudowane cykle życia

Maven opiera się na centralnym pojęciu cyklu życia budowania. Oznacza to, że proces budowania i dystrybucji danego artefaktu (projektu) jest jasno zdefiniowany.

Istnieją trzy wbudowane cykle życia budowania:

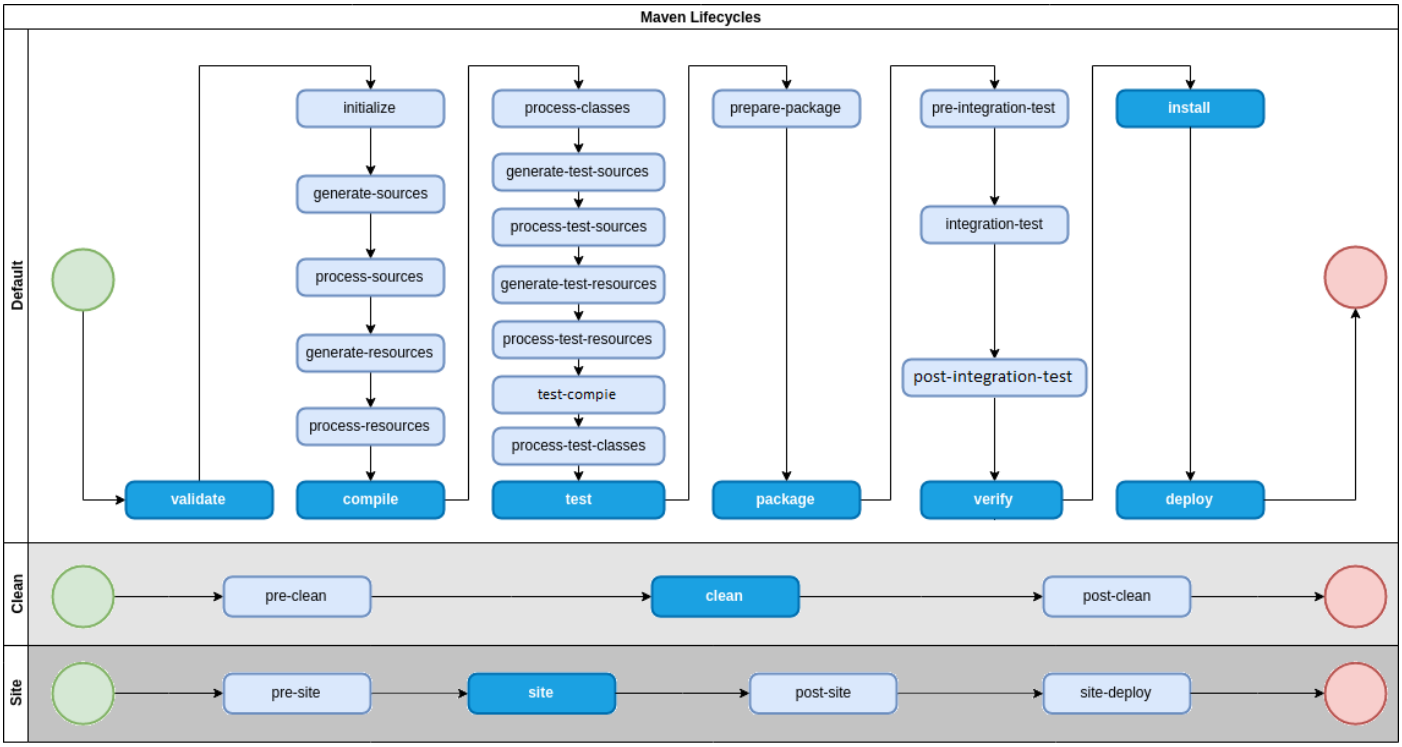
- **default** – zajmuje się wdrażaniem projektu,
- **clean** – zajmuje się czyszczeniem projektu,
- **site** – zajmuje się tworzeniem strony internetowej projektu.

Fazy budowania – *build phases*

Każdy z tych cykli życia budowania jest zdefiniowany przez inną listę **faz budowania** (***build phase***).
Na przykład, cykl życia ***default*** składa się m.in. z następujących faz:

- `validate` - sprawdź, czy projekt jest poprawny i czy dostępne są wszystkie niezbędne informacje;
- `compile` - skompiluj kod źródłowy projektu;
- `test` - przetestuj skompilowany kod źródłowy używając odpowiedniego frameworka;
- `package` - weź skompilowany kod i zapakuj go do formatu umożliwiającego dystrybucję, np. JAR;
- `verify` - sprawdź wyniki testów integracyjnych;
- `install` - zainstaluj pakiet w lokalnym repozytorium, aby mógł być używany jako zależność w innych lokalnych projektach;
- `deploy` - wykonaj w środowisku budowania, skopiuj końcowy pakiet do zdalnego repozytorium w celu udostępnienia go innym programistom i projektom.

Fazy cyklu życia są wykonywane sekwencyjnie, aby wykonać pełen wybrany cykl lub zakończyć na wybranej fazie.



Fazy nie wykonywane bezpośrednio z linii komend

Fazy nazwane **z myślnikami** (np. *pre-**, *post-** lub *process-**) nie są zwykle wywoływane bezpośrednio z linii poleceń. Fazy te sekwencjonują kompilację, produkując pośrednie wyniki, które nie są użyteczne poza kompilacją. W przypadku wywołania *integration-test*, środowisko może być pozostawione w stanie zawieszenia.

Przykład:

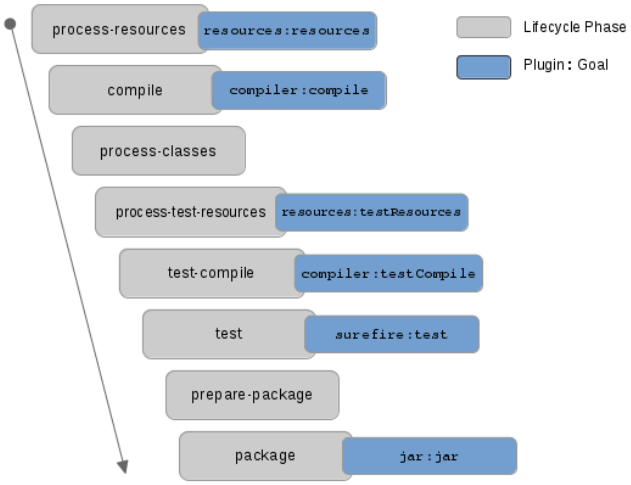
W przypadku wywołania integration-test, środowisko może być pozostawione w stanie zawieszenia. Narzędzia pokrycia kodu takie jak *Jacoco* wiąże cele z fazą *pre-integration-test*, aby przygotować środowisko kontenera testów integracyjnych. Wtyczkata wiąże również cele z fazą *post-integration-test*, aby zebrać statystyki pokrycia lub wycofać z użytku kontener testu integracyjnego.

Cele pluginów – *plugin goals*

Można modyfikować działanie poszczególnych **faz**.
Aby to zrobić, należy zdeklarować **cele** (***goal***) pluginów.

Cel pluginu reprezentuje konkretne **zadanie** (***task***) (drobniejsze niż faza budowania), które uczestniczy w budowaniu i zarządzaniu projektem.

Cel może wyć związane z wieloma **fazami** lub z żadną. Jeśli *zadanie* nie jest związane z **fazą**, może być wykonane poza **cyklem życia** budowania poprzez bezpośrednie wywołanie. Jeśli ***cel*** jest związany z jedną lub więcej **fazami**, to ten **cel** będzie wywoływany we wszystkich tych **fazach**.



Wiązanie celi z fazami

Każde opakowanie (***packaging***) zawiera **listę celów**, które należy **powiązać z** konkretną **fazą**.
Na przykład, opakowanie ***jar*** powiąże cele z fazami budowania w cyklu życia ***default***.

Wtyczki mogą zawierać informacje, które wskazują, do której fazy cyklu życia należy przypisać dany cel. Samo dodanie wtyczki nie jest wystarczającą informacją - musisz również określić cele, które chcesz uruchomić jako część swojego builda. Skonfigurowane cele zostaną dodane do celów już powiązanych z cyklem życia z wybranego opakowania. Cele pakowania są wykonywane przed celami pluginów.

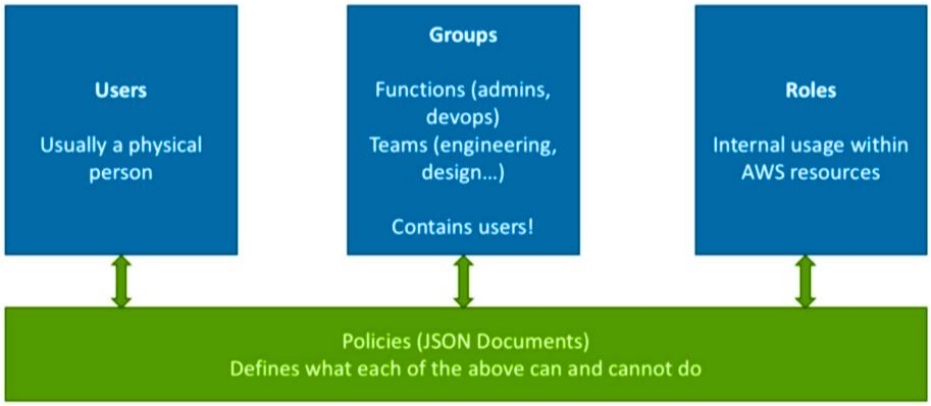
Można uruchomić ten sam cel wiele razy z różną konfiguracją, jeśli zajdzie taka potrzeba.

Gdy podanych jest **wiele wykonań** pasujących do danej fazy, są one wykonywane w kolejności określonej w POM, przy czym **najpierw wykonywane są wykonania dziedziczone**.

[AWS] 1. IAM – AWS Identities and Access Management

AWS Identity and Access Management (IAM) umożliwia bezpieczne zarządzanie dostępem do usług i zasobów AWS. Korzystając z IAM, można tworzyć i zarządzać użytkownikami i grupami AWS oraz używać uprawnień, aby zezwalać i odmawiać im dostępu do zasobów AWS.

IAM jest funkcją konta AWS oferowaną bez dodatkowych opłat. Zostaniesz obciążony jedynie za korzystanie z innych usług AWS przez Twoich użytkowników.



IAM User

Użytkownik IAM reprezentuje osobę lub usługę, która używa użytkownika IAM do interakcji z AWS.

Podstawowym zastosowaniem użytkowników IAM jest umożliwienie ludziom zalogowania się do konsoli zarządzania AWS w celu wykonywania interaktywnych zadań oraz wykonywania programowych żądań do usług AWS za pomocą API lub CLI.

Użytkownik w AWS składa się z nazwy, hasła do zalogowania się do konsoli zarządzania AWS oraz maksymalnie dwóch kluczy dostępu, które mogą być używane z API lub CLI.

Kiedy tworzysz użytkownika IAM, nadajesz mu uprawnienia poprzez uczynienie go członkiem grupy użytkowników, która ma dołączone odpowiednie polityki uprawnień (zalecane), lub poprzez bezpośrednie dołączenie polityk do użytkownika.

Sfederowani użytkownicy

Federacja tożsamości to system zaufania pomiędzy dwoma stronami w celu uwierzytelniania użytkowników i przekazywania informacji potrzebnych do autoryzacji ich dostępu do zasobów.

Możesz włączyć federację tożsamości, aby umożliwić istniejącym tożsamościom (użytkownikom, grupom i rolom) w przedsiębiorstwie dostępy bez konieczności tworzenia użytkownika IAM dla każdej tożsamości.

IAM User Group

Grupa IAM jest zbiorem użytkowników IAM. Możesz użyć grup użytkowników, aby określić uprawnienia dla zbioru użytkowników, co może ułatwić zarządzanie tymi uprawnieniami dla tych użytkowników.

IAM Role

Role IAM pozwalają na delegowanie dostępu dla użytkowników lub usług, które normalnie nie mają dostępu do zasobów AWS Twojej organizacji. Użytkownicy IAM lub usługi AWS mogą przyjąć rolę, aby uzyskać tymczasowe poświadczenia bezpieczeństwa, które mogą być używane do wywołań AWS API. Dzięki temu nie trzeba dzielić się długoterminowymi poświadczeniami lub definiować uprawnień dla każdego podmiotu, który wymaga dostępu do zasobu.

IAM permission policies

Uprawnienia pozwalają określić dostęp do zasobów AWS. Uprawnienia nadawane są podmiotom IAM (użytkownikom, grupom i rolom) i domyślnie podmioty te rozpoczynają pracę bez żadnych uprawnień.

Aby nadać podmiotom uprawnienia, możesz dołączyć politykę, która określa:

Akcje

Które akcje usług AWS dopuszczasz. Wszelkie akcje, na które nie zezwolisz są odrzucane.

Na przykład, możesz zezwolić użytkownikowi na wywołanie akcji Amazon S3 ListBucket.

Zasoby

Na które zasoby AWS zezwalasz na akcję. Użytkownicy nie mają dostępu do żadnych zasobów, do których nie przyznasz im jawnie uprawnień.

Na przykład, możesz określić, na które kubełki Amazon S3 pozwolisz użytkownikowi wykonać akcję ListBucket.

Efekt

Czy zezwolić lub odmówić dostępu. Ponieważ dostęp jest domyślnie odrzucany, zazwyczaj piszemy polityki, w których efektem jest zezwolenie.

Warunki

Jakie warunki muszą być spełnione, aby polityka zaczęła obowiązywać.

Na przykład, możesz zezwolić na dostęp do określonych kubełków S3, jeśli użytkownik łączy się z określonego zakresu IP lub użył wieloczynnikowego uwierzytelniania przy logowaniu.

Best practices

Oto zalecenia Amazon dotyczące usługi AWS Identity and Access Management (IAM):

- Zablokuj klucze dostępu użytkownika root do konta AWS.
- Utwórz indywidualnych użytkowników IAM.
- Użyj grup użytkowników, aby przypisać uprawnienia do użytkowników IAM.
- Przyznaj najmniejsze uprawnienia.
- Zacznij używać uprawnień z politykami zarządzanymi przez AWS.
- Waliduj swoje polityki.
- Używaj polityk zarządzanych przez klienta zamiast polityk inline.
- Wykorzystaj poziomy dostępu do przeglądu uprawnień IAM.
- Skonfiguruj politykę silnych haseł dla swoich użytkowników.
- Włącz MFA.
- Użyj ról dla aplikacji, które działają na instancjach Amazon EC2.
- Użyj ról do delegowania uprawnień.
- Nie udostępniaj kluczy dostępu.
- Regularnie rotuj dane uwierzytelniające.
- Usuń niepotrzebne dane uwierzytelniające.
- Używaj warunków polityki dla dodatkowego bezpieczeństwa.
- Monitoruj aktywność na swoim koncie AWS.

Szczegółowe zalecenia znajdują się na stronie AWS: Security best practices in IAM.

[AWS] 2. Computing services

EC2 – Elastic Compute Cloud

EC2 jest to serwis sieciowy (*web service*), który dostarcza zasoby obliczeniowe o zmiennej ilości w chmurze. Jest zaprojektowany, aby ułatwić skalowalność obliczeniową w sieci.

Typy instancji

Typ instancji EC2 definiuje:

- procesor,
- pamięć podręczną,
- pamięć masową,

które są dostępne dla wszystkich serwerów, które są uruchamiane z tym typem instancji. Nie mogą być zmienione bez przestoju.

Typy instancji dostępne są kilku kategoriach:

- ogólnego przeznaczenia,
- optymalizacji pod kątem obliczeń, pamięci i przechowywania (np. bazy danych),
- akceleracji obliczeń (np. uczenia maszynowego).

Ceny zależą od typu instancji.

Typy urządzeń źródłowych – *root devices types*

Istnieją dwa rodzaje urządzeń root:

- **Instance Store** – efemeryczna pamięć masowa, która jest faktycznie fizycznie podłączona do hosta, na którym działa serwer wirtualny.
- **EBS – Elastic Block Store** –trwała (*persistent*) pamięć masowa, która istnieje niezależnie od hosta, na którym działa serwer wirtualny.

Do większości zadań, do których potrzeba EC2, lepiej nadaje się EBS. Kluczową różnicą jest perzystencja, która pozwala na zachowanie danych po ponownym uruchomieniu hosta. W przypadku EBS można m.in. zrobić snapshot, skopiować i uruchomić inną instancję EC2 z tymi samymi danymi.

AMI – Amazon Machine Image

AMI to szablon dla instancji EC2, który zawiera konfigurację, system operacyjny i dane, które faktycznie znajdą się na tej konkretnej instancji. AWS dostarcza wiele różnych AMI.

AMI mogą być również współdzielone między kontami. Jeśli więc twoja organizacja ma określoną wersję, powiedzmy, Ubuntu Linux, którą chcesz zmodyfikować w określony sposób dla celów bezpieczeństwa i chcesz, aby miała jeden dodatkowy dysk, to są to przykłady konfiguracji, które możesz wykonać w ramach Amazon Machine Image.

Istnieje także rynek komercyjnych AMI. W **AWS Marketplace**, możesz znaleźć różne AMI dostarczane przez komercyjnych dostawców.

Opcje kupna

On-Demand (domyślnie)

Instancje na żądanie pozwalają Ci płacić za moc obliczeniową za godzinę lub sekundę (minimum 60 sekund) bez żadnych długoterminowych zobowiązań. Uwalnia Cię to od kosztów i zawiłości związanych z planowaniem, zakupem i utrzymaniem sprzętu, a także przekształca zwykle duże koszty stałe w znacznie mniejsze koszty zmienne.

Reserved Instances

Zapewnia zniżki w stosunku do modelu On-Demand, gdy możesz zobowiązać się do określonego okresu (1 lub 3 lata). Dodatkowo, gwarantuje on również rezerwację pojemności dla określonego typu instancji. na cały wykupiony okres.

Dostępne są 3 plany rezerwacji instancji:

- Standard

Daje najwyższy rabat i to działa dobrze tylko dla stałych obciążeń roboczych. Oznacza to, że nie można zmienić instancji w ciągu zamówionego okresu.

- Convertible

Pozwala na konwersję niektórych atrybutów, jeśli będzie to miało taką samą wartość. Jest to również świetne rozwiązanie dla stabilnych obciążeń.

- Scheduled

Scheduled Reserved Instance opłaca się w przypadku obciążeń zmiennych, ale przewidywalnych, np. wzmożony ruch w okresie świątecznym.

- Savings Plan

Podobny do instancji zarezerwowanych, z tą różnicą, że nie ogranicza się tylko do EC2. Obsługuje również **Fargate** i **AWS Lambda**. W przeciwieństwie do Reserved Instances nie rezerwuje pojemności.

Spot

Pozwala na wykorzystanie nadmiaru mocy obliczeniowej EC2, które mogą istnieć w strefie dostępności (***availability zone***). Pozwala uzyskać nawet 90% zniżki. Istnieje cena rynkowa dla typów instancji na strefę dostępności i jest to tzw. cena spot (***spot prise***). Jeśli twoja oferta jest wyższa niż cena spot, będziesz mógł uruchomić swoje instancje. Jeśli cena spot przekroczy twoją ofertę, wtedy twoja instancja zostanie zamknięta w ciągu 2 minut. Z tego powodu nadaje się w przypadku obciążeń, które mogą się uruchamiać i zamykać bez negatywnego wpływu na system.

Dedicated Host

Dostarcza dedykowany serwer fizyczny w centrum danych (***data centre***). Jest to najdroższa opcja.

Przykładowym przypadkiem jego zapotrzebowania jest pewność przestrzegania warunków licencji per-serwer.

Elastic Beanstalk

Elastic Beanstalk automatyzuje proces wdrażania i skalowania obciążeń na EC2, z tą różnicą, że zamiast zajmować się serwerami bezpośrednio (IaaS), jest to bardziej podejście typu Platforma jako usługa (PaaS). W przeciwieństwie do EC2, gdzie można zrobić wszystko, jeśli tylko można uruchomić to na serwerze, Elastic Beanstalk działa w ramach zestawu technologii.

W tym przypadku, nadal teoretycznie uruchomia się wszystkie obliczenia na EC2, ale proces zarządzania tymi serwerami i obsługi rzeczy - takich jak dostarczanie i równoważenie obciążenia, skalowanie i monitorowanie - są obsługiwane automatycznie poprzez pracę Elastic Beanstalk.

Wspierane platformy i języki programowania, m.in. Java, .NET, Node.js, Python, Docker.

AWS Lambda

Lambda pozwala uruchomić kod bez dostarczania i zarządzania serwerami, a płaci się tylko za czas obliczeń, które zużywa. Można uruchomić kod dla prawie każdego rodzaju aplikacji lub usługi back-end, wszystko z zerową administracją.

Integruje się z wieloma usługami AWS out-of-the-box, np. S3, DynamoDB. Dzięki temu może umożliwić przepływ pracy sterowany zdarzeniami.

Prawdziwą zmienną, od której zależy cena to ilość pamięci dostępnej dla uruchomionej funkcji.

Niektóre zalety AWS Lambda:

- Zmniejszone są wymagania konserwacyjne, więc nie musisz się martwić o te bazowe serwery i utrzymywanie ich na bieżąco.
- Umożliwia on odporność na awarie (***fault tolerance***) bez konieczności implementowania jej. Dzieje się to za sprawą działania w wielu strefach dostępności.
- Skaluje się w oparciu o zapotrzebowanie, a wycena jest oparta na rzeczywistym użyciu

[AWS] 3. Network & Content Delivery services

Amazon VPC

Amazon VPC to wirtualna chmura prywatna, która jest logicznie odizolowana od publicznej części. Można na w niej uruchamiać usługi i zasoby AWS.

Umożliwia to posiadanie wirtualnej sieci (VPN), którą definiujesz i możesz konfigurować. VPC obsługuje zarówno IPv4 jak i IPv6. Można skonfigurować:

- zakres adresów IP (***IP adderess range***),
- podsieci (***subnets***),
- tabele tras (***route tables***),
- bramy sieciowe (***naetwork gateways***).

VPC oferuje m.in.:

- wystawianie pewnych obszarów dostępnych z Internetu, np. serwer WWW,
- translacje adresów sieciowych lub NAT dla tych prywatnych podsieci,
- łączenie VPC między sobą,
- prywatne połączenia z wieloma usługami AWS.

Możesz mieć pewność, że wrażliwa aplikacja nie musi wysyłać ruchu przez Internet, tylko może pozostać w obrębie VPC, nawet jeśli korzysta z konkretnych usług AWS.

Internet Gateway

Internet Gateway to skalowany horyzontalnie i wysoce dostępny komponent VPC, który umożliwia komunikację między VPC a Internetem.

Internet Gateway służy do dwóch celów:

- zapewnienia docelowego miejsca w tablicach trasowania VPC dla ruchu trasowalnego przez Internet,
- wykonywania translacji adresów sieciowych (NAT) dla instancji, którym przypisano publiczne adresy IPv4.

Internet Gateway obsługuje ruch IPv4 i IPv6. Nie ogranicza przepustowości ruchu sieciowego.

Posiadanie bramy internetowej na koncie nie wiąże się z żadnymi dodatkowymi opłatami.

EC2 jest świadoma tylko prywatnej (wewnętrznej) przestrzeni adresowej IP zdefiniowanej w ramach VPC i podsieci. Internet Gateway logicznie zapewnia NAT jeden-do-jednego dla instancji, więc kiedy ruch opuszcza podsieć VPC i kieruje się do Internetu, pole adresu odpowiedzi jest ustawione na publiczny adres IPv4 lub adres Elastic IP instancji. I odwrotnie, ruch kierowany na publiczny adres IPv4 lub Elastic IP ma tłumaczony adres docelowy na prywatny adres IPv4 instancji, zanim zostanie dostarczony do VPC.

NAT Gateway & NAT Instance

Network Address Translation (translacja adresów sieciowych) jest to technika przesyłania ruchu sieciowego poprzez router, która wiąże się ze zmianą źródłowych lub docelowych adresów IP, zwykle również numerów portów TCP/UDP pakietów IP podczas ich przepływu. Zmieniane są także sumy kontrolne (zarówno w pakiecie IP, jak i w segmencie TCP/UDP), aby potwierdzić wprowadzone zmiany.

Możesz użyć urządzenia NAT, aby umożliwić instancjom w prywatnych podsieciach łączenie się z internetem, innymi VPC lub sieciami lokalnymi. Urządzenie NAT zamienia źródłowy adres IPv4 instancji na adres urządzenia NAT. Podczas wysyłania ruchu odpowiedzi do instancji, urządzenie NAT tłumaczy adresy z powrotem na oryginalne źródłowe adresy IPv4.

Instancje te mogą komunikować się z usługami spoza VPC, ale nie mogą otrzymywać niechcianych żądań połączenia.

Możesz użyć zarządzanego urządzenia NAT oferowanego przez AWS, zwanego **NAT Gateway** lub możesz stworzyć własne urządzenie NAT na instancji EC2, zwanej **NAT Instance**. Zalecamy korzystanie z bram NAT, ponieważ zapewniają one lepszą dostępność i przepustowość oraz wymagają mniej nakłady prac.

Kiedy tworzysz bramę NAT, określasz jeden z następujących **typów łączności**:

- **prywatny** - (domyślnie) Instancje w prywatnych podsieciach mogą łączyć się z Internetem przez publiczny NAT Gateway, ale nie mogą otrzymywać niechcianych połączeń przychodzących z Internetu. Należy przypisać Elastic IP do bramy NAT podczas jej tworzenia.
- **prywatny** - Instancje w podsieciach prywatnych mogą łączyć się z innymi VPC lub siecią lokalną poprzez prywatną bramę NAT. Ruch z bramy NAT może być kierowany przez bramę tranzytową lub wirtualną bramę prywatną. Nie można powiązać elastycznego adresu IP z prywatną bramą NAT.

Elastic IP

Elastic IP to statyczny adres IPv4 osiągalny z Internetu przeznaczony do dynamicznego przetwarzania w chmurze. Statyczność oznacza, że nie zmienia się w czasie. Elastic IP jest przypisany do Twojego konta AWS i jest Twój, dopóki go nie zwolnisz.

Używając adresu Elastic IP, możesz zamaskować awarię instancji lub oprogramowania poprzez szybkie przemapowanie adresu na inną instancję na Twoim koncie. Alternatywnie, możesz podać adres Elastic IP w rekordzie DNS dla swojej domeny, tak aby Twoja domena wskazywała na Twoją instancję.

Obecnie AWS nie świadczy usługi Elastic IP dla IPv6.

Aby skorzystać z Elastic IP, należy przypisać go do konta, a następnie skojarzyć z **instancją** lub **interfejsem sieciowym**. Jeśli jest skojarzony z instancją, jest on również skojarzony z jej głównym interfejsem. Jeśli jest skojarzony z interfejsem sieciowym, który jest dołączony do instancji, jest on również powiązany z tą instancją.

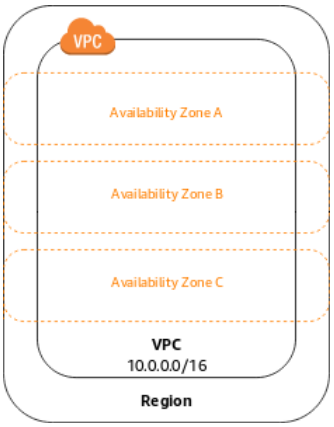
Elastic network interface

Elastyczny interfejs sieciowy jest logicznym komponentem sieciowym w VPC, który reprezentuje wirtualną kartę sieciową. Możesz tworzyć i konfigurować interfejsy sieciowe oraz dołączać, odłączać i przepinać je do instancji w tej samej strefie dostępności.

Twoje konto może również posiadać interfejsy sieciowe zarządzane przez requestera (*requester-managed*), które są tworzone i zarządzane przez usługi AWS, aby umożliwić Ci korzystanie z innych zasobów i usług. Nie możesz samodzielnie zarządzać tymi interfejsami sieciowymi.

Każda instancja ma domyślny interfejs sieciowy, zwany podstawowym interfejsem sieciowym. Nie możesz odłączyć podstawowego interfejsu sieciowego od instancji. Możesz tworzyć i dołączać dodatkowe interfejsy sieciowe. Maksymalna liczba interfejsów sieciowych, których możesz używać, różni się w zależności od typu instancji.

VPC Subnet – *podsieci*

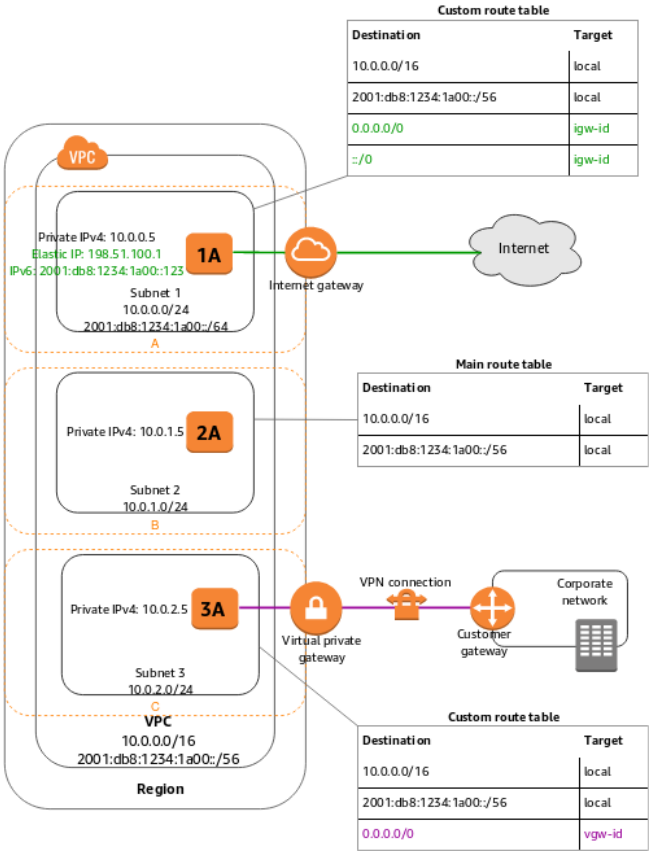


Podczas tworzenia VPC należy określić zakres adresów IPv4 dla VPC w postaci bloku Classless Inter-Domain Routing (CIDR), na przykład 10.0.0.0/16. Poniższy diagram przedstawia nowe VPC z blokiem CIDR IPv4.

W głównej tabeli tras znajdują się następujące trasy:

Destination	Target
10.0.0.0/16	local

VPC rozciąga się na wszystkie **strefy dostępności** w regionie.



Po utworzeniu VPC można dodać jedną lub więcej **podsieci** w każdej strefie dostępności.

Podczas tworzenia podsieci określa się blok **CIDR dla podsieci**, który jest **podzbiorem bloku CIDR VPC**. Każda podsieć musi znajdować się w całości w jednej strefie dostępności i nie może rozciągać się na inne strefy. Strefy dostępności są odrębnymi lokalizacjami, które zostały zaprojektowane w taki sposób, aby były odizolowane od awarii w innych strefach dostępności.

Opcjonalnie można dodać podsieci w **strefie lokalnej**. **Local Zone** jest wdrożeniem infrastruktury AWS, które lokuje obliczenia, pamięć masową, bazę danych i inne wybrane usługi bliżej użytkowników końcowych. Strefa lokalna umożliwia użytkownikom końcowym uruchamianie aplikacji, które wymagają opóźnień rzędu pojedynczych milisekund.

Opcjonalnie można przypisać blok IPv6 CIDR do swojego VPC i do podsieci.

Jeśli podsieć jest skojarzona z tabelą trasowania, która ma trasę do **Internet Gateway**, jest to tzw. **podsieć publiczna**. Jeśli podsieć jest skojarzona z tabelą trasowania, która nie zawiera trasy do bramy internetowej, jest to **podsieć prywatna**.

Route table – subnet routing

Każda podsieć musi być skojarzona z **tablicą trasowania (route table)**, która określa dozwolone trasy dla **ruchu wychodzącego** z danej podsieci. Każda utworzona podsieć jest automatycznie kojarzona z główną tabelą dla VPC. Można zmienić to skojarzenie oraz zawartość głównej tabeli trasowania.

Instancja 2A (patrz przykład z *VPC Subnet*) nie może dotrzeć do Internetu, ale może dotrzeć do innych instancji w VPC. Można pozwolić instancji w VPC na inicjowanie połączeń wychodzących do Internetu przez IPv4, ale uniemożliwić niechciane połączenia przychodzące z Internetu za pomocą bramy translacji adresów sieciowych (NAT) lub instancji. Ponieważ można przydzielić ograniczoną liczbę adresów Elastic IP, zaleca się korzystanie z urządzenia NAT, w przypadku większej liczny instancji wymagających statycznego publicznego adresu IP.

Network ACL – Access Control List (for subnets)

Sieciowe listy ACL kontrolują ruch przychodzący do i wychodzący z **podsieci**.

W większości przypadków grupy zabezpieczeń zaspokajają potrzeby, ale można również użyć sieciowych list ACL w celu uzyskania dodatkowej warstwy zabezpieczeń dla VPC.

Każda podsieć z założenia musi być **powiązana z listą sieciową ACL**. Każda utworzona podsieć jest automatycznie kojarzona z domyślnym ACL sieciowym VPC. Można zmienić to skojarzenie, a także zawartość domyślnej listy ACL sieciowej.

Można utworzyć **dziennik przepływów** (**flow log**) w VPC lub podsieci w celu przechwytywania ruchu przychodzącego i wychodzącego z interfejsów sieciowych w VPC lub podsieci. Można również utworzyć dziennik przepływu na pojedynczym interfejsie sieciowym. Dzienniki przepływów są publikowane w CloudWatch Logs lub Amazon S3.

Security group (for instances)

Grupy zabezpieczeń kontrolują ruch przychodzący do i wychodzący z **instancji**.

Jedna instancja w sieci VPC może mieć przypisanych do pięciu grup zabezpieczeń. Każda instancja w podsieci w VPC może być przypisana do innego zestawu grup zabezpieczeń. W przypadku niepodania grupy zabezpieczeń przy tworzeniu instancji, zostaje ona automatycznie przypisana do domyślnej grupy zabezpieczeń dla VPC.

Direct Connection

AWS Direct Connect ułatwia ustanowienie dedykowanego prywatnego połączenia między AWS a swoim centrum danych lub biurem. Może to zwiększyć przepustowość pasma i zapewnić bardziej spójne wrażenia sieciowe niż połączenia internetowe.

Route 53 – DNS

Route 53 jest to usługa nazw domen (**DNS**). DNS to proces mapowania nazwy domen do konkretnych adresów, które są potrzebne do identyfikacji usług komputerowych i urządzeń w sieci. Musisz jednak wiedzieć, że zmiany DNS nie są natychmiastowe, gdyż muszą zostać rozpropagowane w sieci serwerów DNS na całym świecie, co może potrwać nawet kilka godzin.

Jest to jedna z dwóch usług, które wykorzystują lokalizacje brzegowe AWS w ramach ich globalnej infrastruktury. Oznacza to, że wszelkie zmiany, które wprowadzasz, są stosowane globalnie. Jest wysoce dostępna, co oznacza, że będzie miała minimalny czas przestoju oraz umożliwia tworzenie wysoce dostępnych usług. Umożliwia globalny routing zasobów, więc pozwala odsyłać zgłoszenia do konkretnego serwera na podstawie kraju, z którego przychodzą czy też do serwera, który odpowiada najszybciej. Pozwala też przekierować ruch z wyłączonego lub uszkodzonego regionu w sposób niezauważalny dla użytkowników.

Elastic Load Balancing

Jednym z aspektów skalowalności (elastycznoći) jest sposób, w jaki kierujemy użytkowników do odpowiedniej infrastruktury, za co odpowiedzialny jest Elastic Load Balancing. ELB umożliwia dystrybucję ruchu do wielu miejsc docelowych, np. oparciu o obciążenie każdego z dostępnych serwerów. Obsługuje jedną lub więcej stref dostępności w ramach regionu.

Domyślnie integruje się z EC2, ECS (***Elastic Container Services***).

Istnieją trzy rodzaje load balancerów:

- Application Load Balancer (**ALB**),
- Network Load Balancer (**ELB**),
- Classic Load Balancer (czasem określane po prostu jako **ELB**).

Skalowanie w ramach ELB

Skalowanie dzieli się na:

- pionowe – **vertical** (**scaling up**)

Skalujemy nasz typ instancji do większego typu instancji z dodatkowymi zasobami. Jednak, jeśli to zrobimy, będziemy musieli wyłączyć nasz serwer. To zazwyczaj nienajlepszy sposób.

- **poziome** – **horizontal** (**scaling out**)

To jest to dodanie dodatkowych instancji przy użyciu ELB, aby faktycznie obsłużyć proces przekierowywania naszych użytkowników do właściwego serwera.

CloudFront

Jest to sieć dostarczania treści, która wykorzystuje lokalizacje brzegowe w ramach globalnej infrastruktury AWS. Oznacza to, że na całym świecie znajdują się serwery, do których można wysyłać treści, aby umożliwić użytkownikom otrzymywanie ich z serwera, który jest najbliżej nich, co zwiększy wydajność.

CloudFront obsługuje zarówno statyczne, jak i dynamiczne treści.

Zawiera również kilka zaawansowanych funkcji bezpieczeństwa, a są to takie rzeczy jak:

- **AWS Shield**, który obsługuje ataki distributed denial of service (D**DoS**),
- **Web Application Firewall**.

API Gateway

Jest to w pełni zarządzana usługa zarządzania API. Oznacza to, że możesz tworzyć i udostępniać API, które są po prostu usługami sieciowymi, które następnie inne aplikacje mogą wywoływać. Można dystrybuować je za pośrednictwem CloudFront.

API Gateway bezpośrednio integruje się z wieloma usługami AWS. Dostarcza również monitorowanie i metryki swoich wywołań API.

Można również zintegrować to zarówno z VPC jak i z prywatnymi aplikacjami on-premise, więc nie musi być używany tylko przez publiczne wywołania API.

AWS Global Accelerator

Jest to usługa sieciowa, która wysyła ruch użytkownika przez globalną infrastrukturę sieciową AWS, znaczmoe zwiększając wydajność użytkowników Internetu. Kiedy Internet jest przeciążony, Globalny Akcelerator AWS optymalizuje ścieżkę do aplikacji, aby utrzymać straty pakietów, jitter (jest to zazwyczaj zakłócenie w normalnej sekwencji wysyłania pakietów danych) i opóźnienia na stałym niskim poziomie.

Z Global Accelerator, jesteś wyposażony w dwa globalne statyczne publiczne IP, które działają jako stały punkt wejścia do aplikacji, poprawiając dostępność. Na zapleczu można dodawać lub usuwać punkty końcowe aplikacji AWS, takie jak **Application Load Balancers**, **Network Load Balancers**, instancje **EC2** i **elastyczne IP** bez wprowadzania zmian dla użytkowników. Globalny akcelerator automatycznie przekierowuje ruch do najbliższego zdrowego, dostępnego punktu końcowego, aby ograniczyć awarię punktu końcowego.

[AWS] 4. Storage services

S3 – *Simple Storage Service*

Amazon S3 poziomie pozwala na przechowywanie plików jako obiektów wewnątrz kubełków (***bucket***). Kubełki są jednostką organizacyjną w S3. Łatwe w użyciu funkcje zarządzania pozwalają uporządkować dane i skonfigurować precyzyjnie dostrojone kontrole dostępu, w tym dostęp do plików poprzez URL.

Amazon S3 oferuje szereg klas pamięci masowej przeznaczonych dla różnych przypadków użycia. Klasy przechowywania S3 mogą być konfigurowane na poziomie kubła oraz obiektu.

Można również używać polityk S3 Lifecycle do automatycznego przenoszenia obiektów pomiędzy klasami pamięci masowej bez konieczności wprowadzania zmian w aplikacji.

S3 wpiera SSL dla danych w tranzycie oraz szyfrowanie danych w spoczynku.

- ### S3 Standard

Do ogólnego przechowywania często dostępnych danych. S3 Standard oferuje wysoką trwałość, dostępność i wydajność przechowywania często używanych danych. Ponieważ zapewnia niskie opóźnienia i wysoką przepustowość, jest odpowiedni dla szerokiej gamy przypadków użycia , w tym aplikacji w chmurze, dynamicznych stron internetowych, dystrybucji treści, aplikacji mobilnych i gier oraz analityki big data.

- ### S3 Intelligent-Tiering

Jest jedyną klasą pamięci masowej w chmurze, która zapewnia automatyczną oszczędność kosztów poprzez przenoszenie obiektów pomiędzy czterema poziomami dostępu, gdy zmieniają się wzorce dostępu.

Poziomy zoptymalizowanych pod kątem dostępu:

  - częstego **frequent**,
  - nieczęstego **infrequent**,
  - archiwalnego **archive**,
  - głęboko archiwalnego **deep archive.**

Brak kosztów operacyjnych, gdy obiekty są przenoszone między poziomami dostępu w ramach klasy pamięci masowej S3 Intelligent-Tiering. Jest za to niewielka miesięczna opłata za monitorowanie i tworzenie warstw (auto-tiering).

- ### S3 Standard-Infrequent Access

Do danych o długim czasie przechowywania, ale rzadziej dostępnych.

S3 Standard-IA jest przeznaczony dla danych, które są udostępniane rzadziej, ale wymagają szybkiego dostępu w razie potrzeby. S3 Standard-IA oferuje wysoką trwałość, wysoką przepustowość i niskie opóźnienia S3 Standard, z niską ceną za GB pamięci masowej i opłatą za pobranie GB. To połączenie niskich kosztów i wysokiej wydajności sprawia, że S3 Standard-IA jest idealny do długoterminowego przechowywania, tworzenia kopii zapasowych i jako magazyn danych dla plików odzyskiwania po awarii.

- ### S3 One Zone-Infrequent Access

W przeciwieństwie do innych klas S3 Storage, które przechowują dane w co najmniej trzech strefach dostępności (AZ), S3 One Zone-IA przechowuje dane w jednej AZ i kosztuje 20% mniej niż S3 Standard-IA.

Jest to dobry wybór do przechowywania drugorzędnych kopii zapasowych danych znajdujących się w siedzibie firmy lub danych, które można łatwo odtworzyć.

- S3 Glacier

S3 Glacier to bezpieczna, trwała i tania klasa pamięci masowej do archiwizacji danych. Możesz niezawodnie przechowywać dowolną ilość danych przy kosztach, które są konkurencyjne lub tańsze niż w przypadku rozwiązań lokalnych.

Aby utrzymać koszty na niskim poziomie, a jednocześnie dostosować je do różnych potrzeb, S3 Glacier udostępnia trzy opcje odzyskiwania danych w czasie od kilku minut do kilku godzin.

Możesz przesyłać obiekty bezpośrednio do S3 Glacier lub korzystać z polityk S3 Lifecycle**.**

- S3 Glacier Deep Archive

S3 Glacier Deep Archive to najtańsza klasa pamięci masowej Amazon S3, która umożliwia długoterminowe przechowywanie i cyfrową ochronę danych, do których dostęp można uzyskać raz lub dwa razy w roku. Przeznaczony jest dla klientów - szczególnie z branż podlegających ścisłym regulacjom, takich jak usługi finansowe, opieka zdrowotna i sektor publiczny - którzy przechowują zbiory danych przez 7-10 lat lub dłużej, aby spełnić wymogi zgodności z przepisami.

Wszystkie obiekty przechowywane w S3 Glacier Deep Archive są replikowane i przechowywane w co najmniej trzech geograficznie rozproszonych strefach dostępności.

- S3 on Outposts

AWS Outposts jest fizycznym sprzętem dostarczanym do użytkownika. Przydatne, jeśli masz wymagania dotyczące rezydencji danych, które nie mogą być spełnione przez istniejący region AWS.

AWS Outposts świadczy lokalnie poprzez interfejsy API niektóre usługi AWS z kategorii:

- Compute
- ***Storage***
- Networking
- Database
- Containers
- Data Processing.

EBS – *Elastic Block Store*

EBS to pamięć blokowa, która została zaprojektowana do podłączenia do pojedynczej instancji EC2**.**

Może skalować się do obsługi petabajtów danych, a także obsługiwać wiele typów wolumenów w zależności od potrzeb. Może się przydać w przypadku okresowego tworzenia kopi zapasowych danych z dysku podłączonym do instancji EC2. Oferuje również szyfrowanie wolumenów.

EFS – *Elastic File System* (for Liniux)

EFS jest to w pełni zarządzany system plików **NFS**, który został zaprojektowany specjalnie dla obciążeń Linux.

Obsługuje skalowanie do petabajtów danych. Umożliwia również przechowywanie danych w wielu strefach dostępności, dzięki czemu domyślnie uzyskuje się pewną trwałość.

Udostępnia dwie różne klasy pamięci masowej:

  - Standard
  - Infrequent-Access.

FSx for Windows File Server

Jest to w pełni zarządzany natywny system plików Windows. Zawiera on natywne funkcje Windows, w tym takie rzeczy jak obsługa SMB, integracja z Active Directory i obsługa Windows NTFS.

[AWS] 5. Database and Utilities services

RDS – Relational Database Service

RDS to w pełni zarządzane serwisy dla relacyjnych baz danych. Obsługują zaopatrzenia (provisioning), patchowanie, kopie bezpieczeństwa i ich przywracanie baz danych. Wspierają wdrażania w wielu strefach dostępności (multi-AZ) oraz skalowanie baz danych. Istnieje możliwość uruchamiania wewnątrz VPC

Dostępne platformy RDS

- MySQL
- PostgresSQL
- MariaDB
- Oracle Database
- SQL Server
- Amazon Aurora

Amazon Aurora jest relacyjną bazą danych kompatybilną z MySQL i PostgreSQL, która została stworzona do zastosowań chmurowych. Łączy wydajność i dostępność baz danych typu enterprise z prostotą i efektywnością kosztową baz danych open-source.

DMS – Database Migration Service

Umożliwia migrację danych do AWS z już istniejących baz danych. Wspiera migrację danych jednorazową (one-time) oraz ciągłą (continual). Wspiera wiele popularnych baz danych, zarówno komercjalnych jak i open-source.

Płaci się tylko za moc obliczeniową zużywaną podczas migracji.

DynamoDB

DynamoDB jest to w pełni zarządzana usługa NoSQL. Została stworzona z myślą o skalowalności, dzięki czemu udało osiągnąć bardzo dużą wydajność w bardzo dużej skali. Daje możliwość konfiguracji autoskalowania. DynamoDB obsługuje pamięć podręczną (cache) dzięki usłudze DynamoDB Accelerator (DAX).

Elasticache

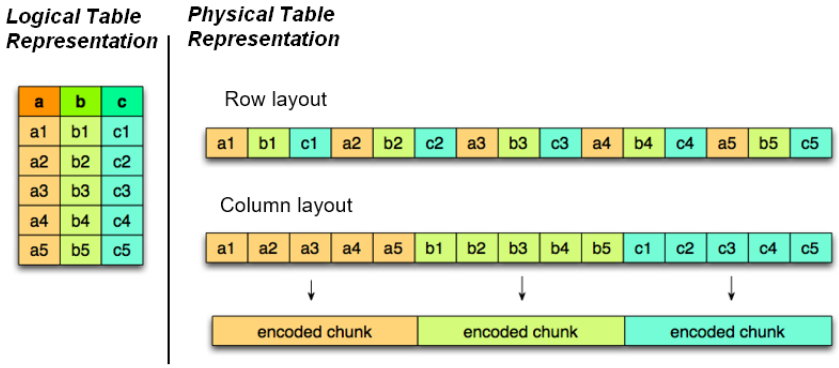
W pełni zarządzana usługa pamięci podręcznej. Wspiera *Memcached* and *Redis*.

Gwarantuje niski czas odpowiedzi, skalowanie i replikację, aby sprostać żądaniom aplikacji.

Obsługuje powszechne przypadki użycia, w tym cache dla baz danych i pamięć sesji.

Redshift

Jest to skalowalna hurtownia danych, która potrafi skalować się w petabajtowej skali. Łączy wysokowydajną pamięć dyskową z kolumnową bazą danych (*Column Oriented Database* - **CODB**). Oferuje pełne szyfrowanie zawartośći oraz izolację w ramach VPC.



[AWS] 6. App Integration services

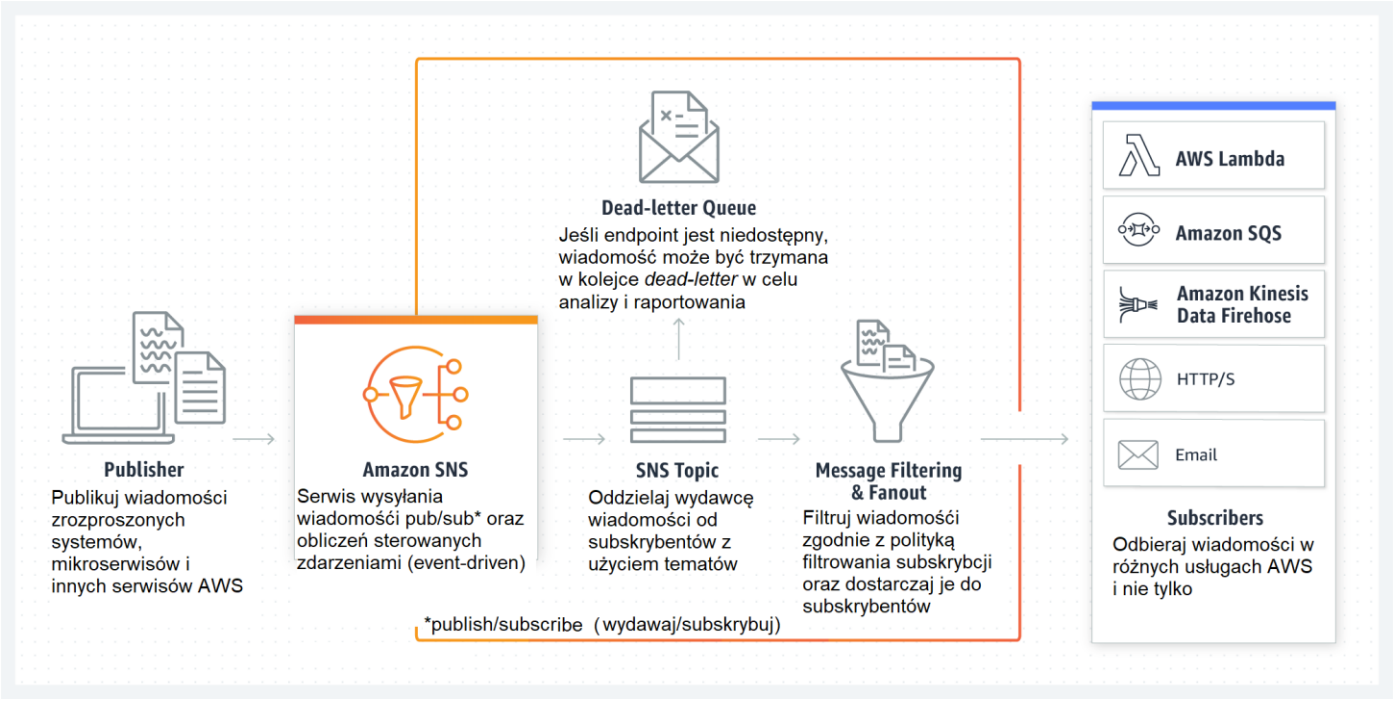
AWS Messaging Services

SNS – Simple Notification service

SNS to w pełni zarządzana usługa przesyłania wiadomości zarówno w komunikacji aplikacja-aplikacja (A2A), jak i aplikacja-osoba (A2P). Umożliwia wysyłanie wiadomości poprzez:

- **Pub/Sub** (publish/subscribe)
- **SMS**
- **Mobile Push**

Pub/Sub:



SQS – Sompple Queue Service

(SQS to w pełni zarządzana usługa kolejkowania wiadomości, która umożliwia oddzielanie i skalowanie mikroserwisów, systemów rozproszonych i aplikacji **serverless**. SQS eliminuje złożoność i koszty związane z zarządzaniem i obsługą **middlewate** zorientowanego na komunikaty. Korzystając z SQS, można wysyłać, przechowywać i odbierać wiadomości między komponentami oprogramowania o dowolnej objętości, bez utraty wiadomości lub wymagania dostępności innych usług.

SQS oferuje dwa rodzaje kolejek komunikatów:

- standardowe (**standard queue**) – maksymalna przepustowość, zamawianie w trybie best-effort i dostarczanie co najmniej raz,
- **FIFO** – zaprojektowane tak, aby zagwarantować, że wiadomości są przetwarzane dokładnie raz, dokładnie w takiej kolejności, w jakiej zostały wysłane.

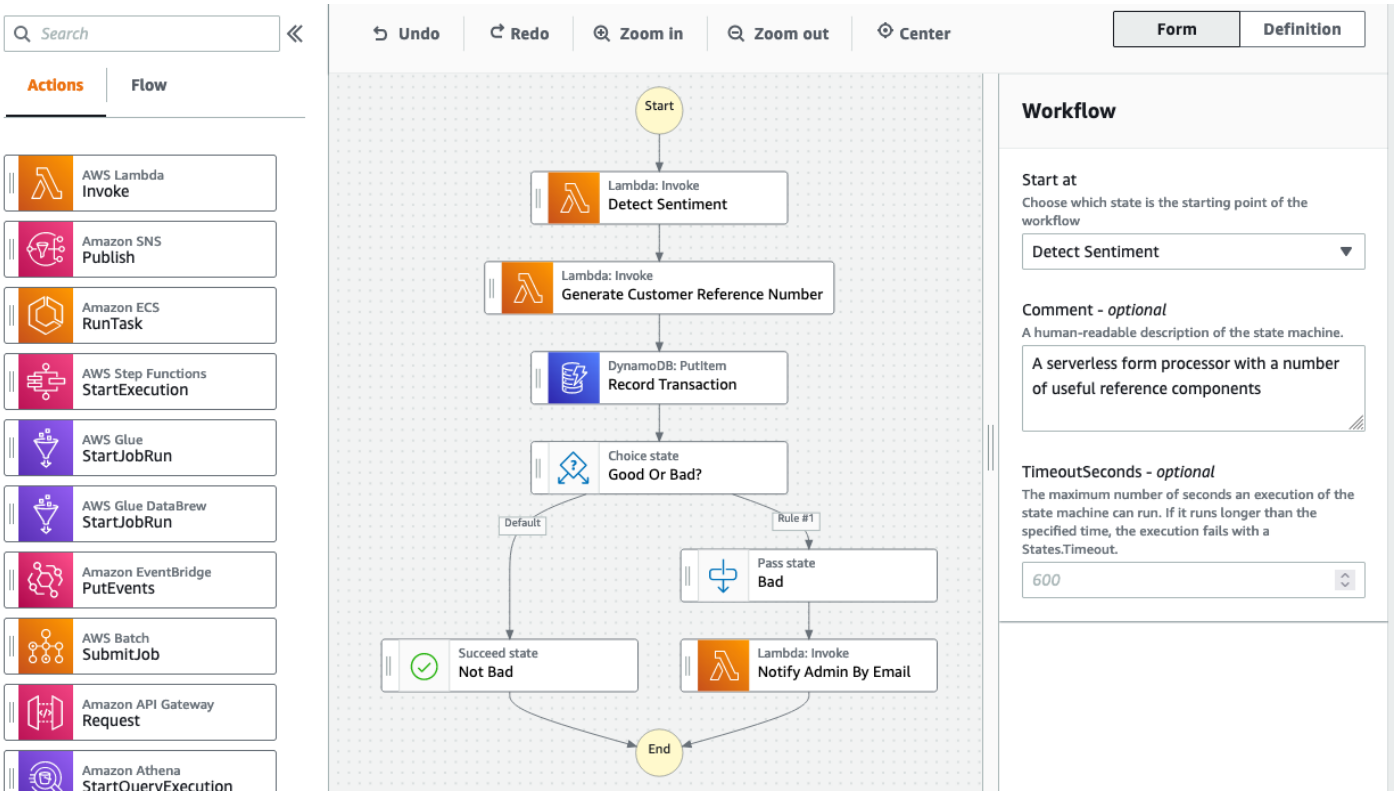
AWS Step Functions

AWS Step Functions to usługa wizualizacji przepływu pracy o niewielkim kodzie, służąca do orkiestracji usług AWS, automatyzacji procesów biznesowych i budowania aplikacji serverless.

Przepływy pracy zarządzają:

- awariami i ponawianiem prób,
- współbieżnością,
- integracją usług,
- obserwowalnością.

Przykład:



[AWS] 7. Management and Governance Services

AWS CloudTrail

Dzięki CloudTrail można rejestrować, stale monitorować i przechowywać aktywność konta związaną z działaniami w całej infrastrukturze AWS. CloudTrail zapewnia historię zdarzeń związanych z aktywnością konta AWS, w tym działań podejmowanych za pomocą konsoli zarządzania AWS, zestawów SDK AWS, narzędzi wiersza poleceń i innych usług AWS.

Historia zdarzeń upraszcza analizę bezpieczeństwa, śledzenie zmian w zasobach i rozwiązywanie problemów. Ponadto można używać CloudTrail do wykrywania nietypowej aktywności na kontach AWS.

Dobre praktyki nakazują, aby CloudTrail był włączony dla każdego konta.

Zarządzanie Infrastrukturą

CloudWatch

Amazon CloudWatch to usługa monitorowania i obserwacji. Gromadzi dane w postaci:
logów, **metryk** i **zdarzeń**,
zapewniając ujednolicony widok zasobów AWS, aplikacji i usług działających na serwerach AWS i lokalnych.

Zapewnia dane i wgląd w działania w postaci dashboardów w celu monitorowania aplikacji, reagowania na zmiany wydajności w całym systemie, optymalizacji wykorzystania zasobów i alarmowania.

AWS Config

AWS Config jest usługą, która stale monitoruje i rejestruje konfiguracje zasobów AWS i pozwala na zautomatyzowanie walidacji zarejestrowanych konfiguracji w stosunku do pożądanych konfiguracji.

Za pomocą Config można:

- przeglądać zmiany w konfiguracjach i relacjach pomiędzy zasobami AWS,
- przeglądać szczegółowe historie konfiguracji zasobów,
- walidować zgodność z konfiguracjami i zasadami określonymi w wewnętrznych wytycznych
- dokonywać korekcji (***remediation steps***) infrastruktury nie spełniającej kryteria.

AWS System Manager

Menedżer systemów zapewnia ujednolicony interfejs użytkownika, dzięki czemu można śledzić i rozwiązywać problemy operacyjne w swoich aplikacjach i zasobach AWS z jednego centralnego miejsca. Systems Manager upraszcza zarządzanie zasobami i aplikacjami, skraca czas wykrywania i rozwiązywania problemów operacyjnych oraz ułatwia obsługę i zarządzanie infrastrukturą w skali.

AWS CloudFormation

Dzięki AWS CloudFormation możesz w prosty sposób modelować kolekcję powiązanych zasobów AWS i innych firm, dostarczać je szybko i spójnie oraz zarządzać nimi przez cały cykl życia zgodnie z paradygmatem **Infrastructure as a Code.**

Szablon CloudFormation (**YAML** lub **JSON**) opisuje pożądane zasoby i ich zależności, dzięki czemu można je uruchomić i skonfigurować razem jako jeden stos. Za pomocą szablonu można tworzyć, aktualizować i usuwać cały stos jako pojedynczą jednostkę, zamiast zarządzać zasobami indywidualnie.

Dostarcza wykrywanie zmian w czasie (***drift detection***) w infrastrukturze.

Nie pobiera dodatkowych opłat za używanie szablonów względem tego, co jest naliczane za samo użycie uruchamianych usług.

AWS OpsWorks

AWS OpsWorks to usługa, która dostarcza instancje Chef i Puppet. OpsWorks pozwala używać Chef i Puppet do automatyzacji konfiguracji, wdrażania i zarządzania serwerami na instancjach Amazon EC2 lub w lokalnych środowiskach obliczeniowych.

OpsWorks ma trzy oferty:

- AWS OpsWorks for Chef Automate

Pakiet narzędzi automatyzacji do zarządzania konfiguracją, zgodnością i bezpieczeństwem oraz ciągłym wdrażaniem. OpsWorks utrzymuje serwer Chef poprzez automatyczne łatanie, aktualizowanie i tworzenie kopii zapasowych.

- AWS OpsWorks for Puppet Enterprise

Pakiet narzędzi automatyzacji od Puppet do zarządzania infrastrukturą i aplikacjami.
OpsWorks utrzymuje główny serwer Puppet poprzez automatyczne łatanie, aktualizowanie i tworzenie kopii zapasowych serwera.

- AWS OpsWorks Stacks

Dzięki OpsWorks Stacks możesz modelować swoją aplikację jako stos zawierający różne warstwy, takie jak load balancing, baza danych i serwer aplikacji. W ramach każdej warstwy można dostarczać instancje Amazon EC2, włączać automatyczne skalowanie i konfigurować instancje za pomocą receptur Chef przy użyciu Chef Solo.

Zarządzanie kontami AWS

AWS Organizations

AWS Organizations pomagają centralnie zarządzać i kierować środowiskiem w miarę wzrostu i skalowania zasobów AWS.

Korzystając z usługi AWS Organizations, można programowo tworzyć nowe konta AWS i przydzielać zasoby, grupować konta w celu uporządkowania przepływów pracy, stosować polityki do kont lub grup w celu zarządzania oraz upraszczać rozliczenia dzięki zastosowaniu jednej metody płatności dla wszystkich kont.

Ponadto AWS Organizations jest zintegrowane z innymi usługami AWS, dzięki czemu można definiować centralne konfiguracje, mechanizmy bezpieczeństwa, wymagania audytowe i współdzielenie zasobów pomiędzy kontami w organizacji.

AWS Contol Tower

AWS Control Tower zapewnia najprostszy sposób konfiguracji i zarządzania bezpiecznym środowiskiem AWS z wieloma kontami, zwanym **landing zone**. AWS Control Tower tworzy **landing zone** przy użyciu **AWS Organizations**, zapewniając bieżące zarządzanie kontami.

Dzięki AWS Control Tower konstruktorzy mogą dostarczać nowe konta AWS bazując na szablonach za pomocą kilku kliknięć, gwarantując, że te konta są zgodne z zasadami obowiązującymi w całej firmie.

Posiada też dashboard zawierający informacje operacyjne w jednym miejscu.