Szkolenie Kubernetes w praktyce

Spis treści

Opis

Szkolenie Kubernetes w praktyce obejmuje naukę i praktyczne doświadczenie związane z zarządzaniem kontenerami. Uczestnicy zdobędą umiejętności w zakresie instalacji, konfiguracji i utrzymania klastra Kubernetes, co obejmuje zarządzanie zasobami, skalowalność i dostępność. W trakcie szkolenia praktycznego uczestnicy bedą pracować nad rzeczywistymi przypadkami użycia, implementować aplikacje w kontenerach i nauczą się jak skutecznie monitorować oraz debugować środowisko. Ponadto szkolenie obejmuje tematy związane z bezpieczeństwem, takie jak zarządzanie dostępem, autentykacja i szyfrowanie. Kurs ten umożliwi uczestnikom zrozumienie pełnego ekosystemu Kubernetes i przygotuje ich do efektywnego wdrażania i utrzymania aplikacji w kontenerach w środowisku produkcyjnym.

O trenerze

Imię i nazwisko: Juliusz Marciniak

Zawód: Programista, DevOps, TechLead

Doświadczenie: 10 lat

Technologie:

- Java, Spring
- Jenkins, GitLab, Nexus
- Docker, Kubernetes

Jestem programistą Java z 10-letnim stażem. W codziennej pracy zajmuję się tematami cloudowymi i Kubernetesowymi. Oprócz tego doglądam architektury aplikacji w swoich projektach. Jestem entuzjastą nowych rozwiązań i technologii. Zwracam dużą uwagę, by rzeczy wdrażane przeze mnie były przemyślane i spójne. Czasami też programuję… i rekrutuję.

Agenda

1. Docker
2. Architektura Kubernetes
3. Podstawy konfiguracji
4. Uwierzytelnianie oraz autoryzacja
5. Sieci
6. Storage
7. Dodatkowe funkcjonalności
8. Logowanie oraz Monitoring
9. Bezpieczeństwo
10. Dystrybucje oraz użyteczne narzędzia

11. Helm jako natywny menedżer pakietów aplikacji w Kubernetes

Zasady

- Staramy się nie spóźniać
- Telefon odbieramy poza salą
- Jesteśmy aktywni
- Jeśli utknęliśmy na problemie podczas realizacji zadań — natychmiast informujemy
- Nie śmiejemy się z innych i nie krytykujemy pomysłów innych
- Każdy ma prawo do wyrażania swojej opinii
- Przerwa na kawę co 1-1,5h
- Przerwa obiadowa o 13:00

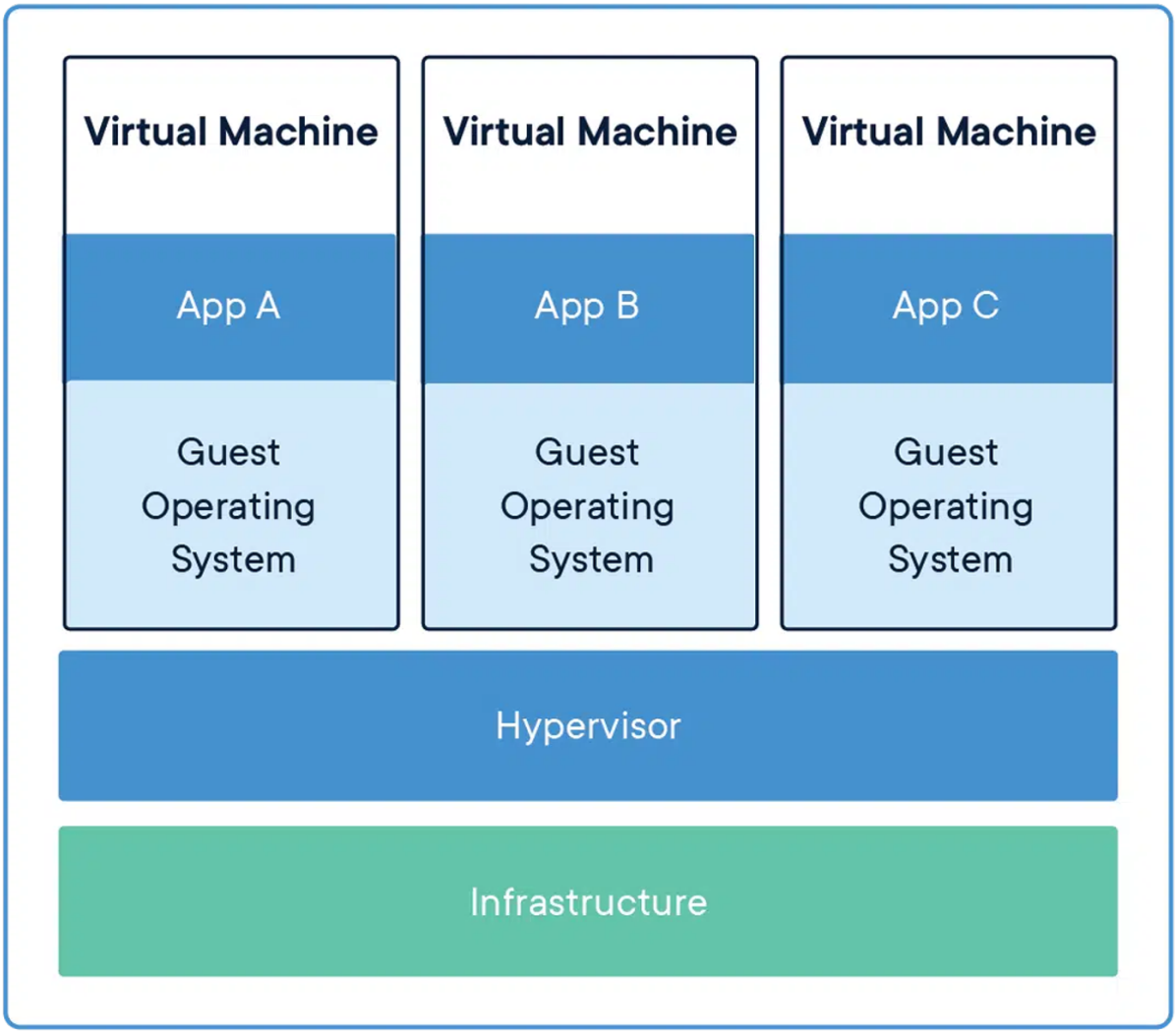
Podstawy teoretyczne

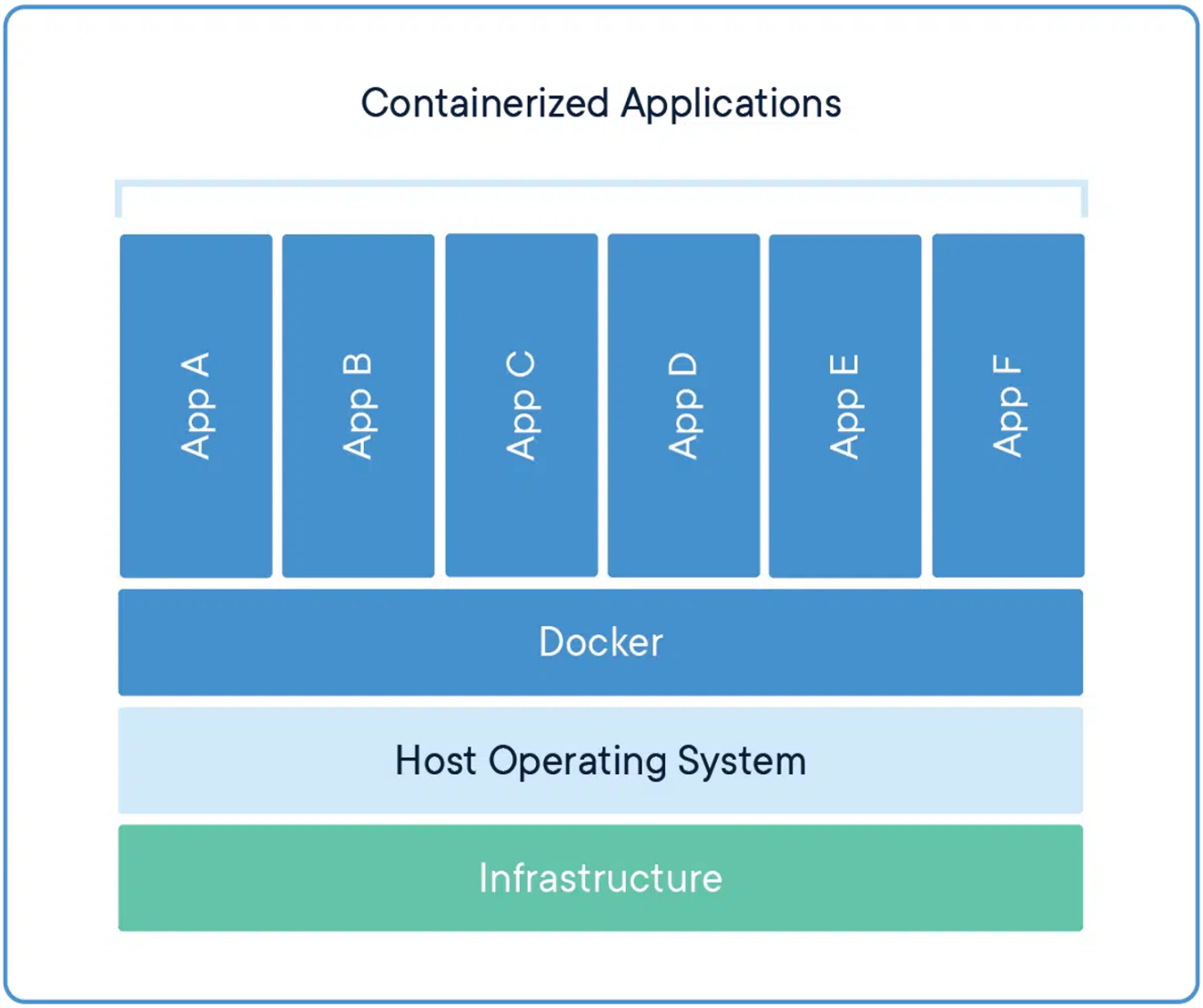
10 powodów, dla których warto konteneryzować aplikacje

1. Izolacja i jednolitość: Kontenery izolują aplikacje od środowiska, co eliminuje konflikty zależności i zapewnia jednolitość między różnymi środowiskami.
2. Łatwość przenoszenia: Kontenery zawierają wszystkie niezbędne zależności, co ułatwia przenoszenie aplikacji między różnymi platformami i infrastrukturami.
3. Szybkość wdrażania: Kontenery uruchamiają się błyskawicznie, co przyspiesza proces wdrażania i umożliwia elastyczne skalowanie w zależności od obciążenia.
4. Elastyczność skalowania: Kontenery umożliwiają dynamiczne dodawanie lub usuwanie instancji aplikacji w zależności od potrzeb, co wspiera elastyczne skalowanie.
5. Łatwe zarządzanie zasobami: Kontenery pozwalają precyzyjnie zarządzać zasobami, co prowadzi do efektywnego wykorzystania infrastruktury i redukcji kosztów.
6. Łatwość utrzymania: Aktualizacje i poprawki mogą być łatwo wdrażane w kontenerach, co minimalizuje czas niedostępności aplikacji.
7. Łatwa replikacja środowiska deweloperskiego: Kontenery pozwalają na dokładne replikowanie środowiska deweloperskiego na produkcji, co eliminuje problemy wynikające z różnic między środowiskami.
8. Bezpieczeństwo: Kontenery posiadają mechanizmy bezpieczeństwa, takie jak izolacja procesów, co pomaga w zabezpieczeniu aplikacji i danych przed nieautoryzowanym dostępem.
9. Automatyzacja: Kontenery integrują się łatwo z narzędziami do automatyzacji, co ułatwia zarządzanie cyklem życia aplikacji.
10. Podział na mikroserwisy: Kontenery wspierają architekturę mikroserwisów, co ułatwia rozwijanie, utrzymanie i skalowanie poszczególnych komponentów aplikacji niezależnie.

Porównanie Dockera i VM

1. Rozmiar i Wydajność:
    - Docker: Kontenery Dockerowe są lżejsze od maszyn wirtualnych, ponieważ dzielą jądro systemowe hosta i współdzielą zasoby z systemem operacyjnym hosta. To sprawia, że są bardziej wydajne i uruchamiają się szybciej.
    - Maszyny wirtualne: VM są bardziej obciążające dla systemu host, ponieważ każda z nich wymaga własnego systemu operacyjnego, co prowadzi do większego zużycia zasobów.
2. Izolacja:
    - Docker: Kontenery Dockerowe używają przestrzeni użytkownika do izolacji procesów, dzięki czemu są lżejsze. Izolacja jest mniej ścisła niż w przypadku maszyn wirtualnych.
    - Maszyny wirtualne: VM oferują pełną izolację, ponieważ każda z nich działa na własnym systemie operacyjnym, ale kosztem większego zużycia zasobów.


Virtual Machine
Virtual Machine
Virtual Machine
App A
App B
App C
Guest Operating System
Guest Operating System
Guest Operating System
Hypervisor
Infrastructure


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure

Docker

Co to jest Docker?

Docker to platforma open-source do konteneryzacji, umożliwiająca pakowanie, dostarczanie i uruchamianie aplikacji w izolowanych środowiskach zwanych kontenerami. Kontenery Docker zawierają aplikację oraz wszystkie zależności, co eliminuje problemy związane z różnicami w środowisku wykonawczym. Platforma pozwala na konsystentne wdrażanie aplikacji na różnych środowiskach, od środowiska deweloperskiego po produkcję. Docker używa obrazów, które są szablonami do tworzenia kontenerów, a te z kolei oparte są na warstwowej strukturze plików, co ułatwia efektywne dzielenie zasobów. Pozwala na szybkie skalowanie aplikacji, zarządzanie zależnościami i izolację procesów. Docker umożliwia prostą integrację z narzędziami do automatyzacji i dostarcza narzędzi takich jak Docker Compose do definiowania i zarządzania wielokontenerowymi aplikacjami. Popularność Dockera wynika z jego elastyczności, wydajności i zdolności do jednolitego zarządzania aplikacjami w kontenerach na różnych platformach. Platforma ta jest szeroko stosowana w dziedzinie wdrażania mikroserwisów i tworzenia skalowalnych i elastycznych architektur aplikacji.

Architektura

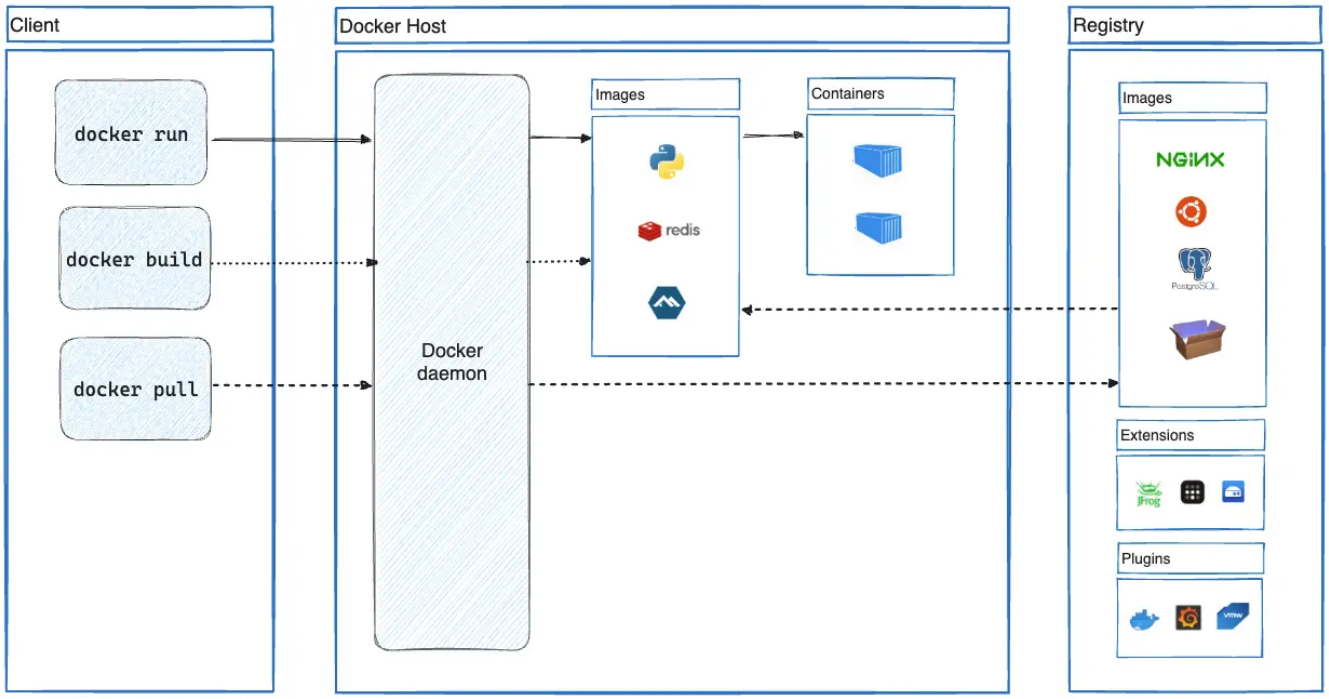


Figure 1. https://docs.docker.com/get-started/overview/

Docker Daemon

Docker Daemon, znany jako `dockerd`, to centralny serwer Dockera, odpowiedzialny za zarządzanie cyklem życia kontenerów, alokację zasobów, obsługę żądań klienta Docker oraz kontrolę dostępu do systemowych zasobów. Działa w tle jako proces, umożliwiając interakcję z klientami poprzez API REST lub interfejs Unix.

Docker client

Docker Client to interfejs użytkownika, umożliwiający interakcję z Docker Daemon poprzez linię poleceń. Działa jako narzędzie wiersza poleceń, pozwalając użytkownikowi wysyłać żądania do serwera Dockera, takie jak tworzenie, uruchamianie i zarządzanie kontenerami.

Docker Desktop

Docker Desktop to narzędzie umożliwiające łatwe korzystanie z platformy Docker na komputerach stacjonarnych. Dostarcza kompletną instalację Dockera, w tym Docker Daemon, Docker CLI, narzędzia do wizualnego zarządzania kontenerami oraz integrację z systemem operacyjnym, umożliwiając szybkie tworzenie, testowanie i uruchamianie kontenerów na platformie Windows lub macOS.

Docker registries

Docker Registries to repozytoria przechowujące i udostępniające obrazy kontenerów używane przez platformę Docker. Mogą być publiczne, takie jak Docker Hub, lub prywatne, umożliwiając organizacjom przechowywanie i udostępnianie własnych obrazów kontenerów.

Docker objects

Docker objects to kluczowe elementy w ekosystemie Dockera, obejmujące kontenery, obrazy, wolumeny, sieci i inne składniki używane do definiowania i zarządzania środowiskiem kontenerowym. Te obiekty pozwalają na przenośność aplikacji, izolację zasobów oraz efektywne zarządzanie cyklem życia aplikacji w kontenerach.

Images

Obrazy (Images) w Dockerze są szablonami, które definiują zawartość i konfigurację środowiska potrzebnego do uruchomienia aplikacji w kontenerze. Są to jednokierunkowe, niezmienne struktury, które zawierają system plików z wszystkimi zależnościami i konfiguracjami niezbędnymi dla aplikacji. Obrazy są tworzone na podstawie plików Dockerfile, które zawierają instrukcje dotyczące instalacji aplikacji, konfiguracji środowiska oraz dodawania danych do obrazu. Po utworzeniu obrazu może on być przechowywany w Docker Registry, a następnie używany do uruchamiania wielu kontenerów, co umożliwia jednolite wdrażanie aplikacji na różnych środowiskach.

Containers

Kontenery w Dockerze są lekkimi, przenośnymi jednostkami wykonawczymi, które zawierają wszystko, co jest niezbędne do uruchomienia aplikacji, włącznie z kodem, zależnościami i konfiguracją. Kontenery opierają się na izolacji procesów, dzięki czemu mogą działać niezależnie od siebie na jednym systemie operacyjnym.

Przykładem uruchomienia kontenera może być uruchomienie kontenera NGINX, popularnego serwera HTTP, za pomocą następującego polecenia w terminalu:

```
docker run -d -p 8080:80 nginx
```

To polecenie powoduje, że Docker Daemon pobiera obraz NGINX z Docker Hub (jeśli nie jest już pobrany), a następnie tworzy i uruchamia instancję kontenera NGINX. Opcja `-d` oznacza, że kontener będzie działał w tle (detached mode), a `-p 8080:80` przekierowuje port 8080 na lokalnej maszynie do portu 80 w kontenerze NGINX. Po wykonaniu tego polecenia serwer NGINX będzie dostępny lokalnie pod adresem http://localhost:8080, obsługując żądania przez uruchomiony kontener.

Instalacja

https://docs.docker.com/engine/install/

Komendy

```
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.11.2-desktop.5)
  compose*    Docker Compose (Docker Inc., v2.23.0-desktop.1)
  container   Manage containers
  context     Manage contexts
  dev*        Docker Dev Environments (Docker Inc., v0.1.0)
  extension*  Manages Docker extensions (Docker Inc., v0.2.20)
  image       Manage images
  init*       Creates Docker-related starter files for your project (Docker Inc.,
v0.1.0-beta.9)
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
```

```
  sbom*       View the packaged-based Software Bill Of Materials (SBOM) for an image
(Anchore Inc., 0.6.0)
  scan*       Docker Scan (Docker Inc., v0.26.0)
  scout*      Docker Scout (Docker Inc., v1.0.9)
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Swarm Commands:
  swarm       Manage Swarm

Commands:
  attach      Attach local standard input, output, and error streams to a running
container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Global Options:
      --config string      Location of client config files (default
"/Users/jmarciniak/.docker")
  -c, --context string     Name of the context to use to connect to the daemon
(overrides DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket to connect to
  -l, --log-level string   Set the logging level ("debug", "info", "warn", "error",
"fatal") (default "info")
```

```
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
"/Users/jmarciniak/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default
"/Users/jmarciniak/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default
"/Users/jmarciniak/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Run 'docker COMMAND --help' for more information on a command.

For more help on how to use Docker, head to https://docs.docker.com/go/guides/
```

docker pull

```
Usage:  docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Download an image from a registry

Aliases:
  docker image pull, docker pull

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
      --platform string         Set platform if server is multi-platform capable
  -q, --quiet                   Suppress verbose output
```

docker build

```
Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build

Aliases:
  docker buildx build, docker buildx b

Options:
      --add-host strings              Add a custom host-to-IP mapping (format:
"host:ip")
      --allow strings                 Allow extra privileged entitlement (e.g.,
"network.host", "security.insecure")
      --attest stringArray            Attestation parameters (format:
"type=sbom,generator=image")
      --build-arg stringArray         Set build-time variables
      --build-context stringArray     Additional build contexts (e.g., name=path)
      --builder string                Override the configured builder instance
```

```
(default "desktop-linux")
      --cache-from stringArray        External cache sources (e.g., "user/app:cache",
"type=local,src=path/to/dir")
      --cache-to stringArray          Cache export destinations (e.g.,
"user/app:cache", "type=local,dest=path/to/dir")
      --cgroup-parent string          Optional parent cgroup for the container
  -f, --file string                   Name of the Dockerfile (default:
"PATH/Dockerfile")
      --iidfile string                Write the image ID to the file
      --label stringArray             Set metadata for an image
      --load                          Shorthand for "--output=type=docker"
      --metadata-file string          Write build result metadata to the file
      --network string                Set the networking mode for the "RUN"
instructions during build (default "default")
      --no-cache                      Do not use cache when building the image
      --no-cache-filter stringArray   Do not cache specified stages
  -o, --output stringArray            Output destination (format:
"type=local,dest=path")
      --platform stringArray          Set target platform for build
      --progress string               Set type of progress output ("auto", "plain",
"tty"). Use plain to show container output (default "auto")
      --provenance string             Shorthand for "--attest=type=provenance"
      --pull                          Always attempt to pull all referenced images
      --push                          Shorthand for "--output=type=registry"
  -q, --quiet                         Suppress the build output and print image ID on
success
      --sbom string                   Shorthand for "--attest=type=sbom"
      --secret stringArray            Secret to expose to the build (format:
"id=mysecret[,src=/local/secret]")
      --shm-size bytes                Size of "/dev/shm"
      --ssh stringArray               SSH agent socket or keys to expose to the build
(format: "default|<id>[=<socket>|<key>[,<key>]]")
  -t, --tag stringArray               Name and optionally a tag (format: "name:tag")
      --target string                 Set the target build stage to build
      --ulimit ulimit                 Ulimit options (default [])
```

docker run

```
Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image

Aliases:
  docker container run, docker run

Options:
      --add-host list                  Add a custom host-to-IP mapping (host:ip)
      --annotation map                 Add an annotation to the container (passed
through to the OCI runtime) (default map[])
```

```
  -a, --attach list                    Attach to STDIN, STDOUT or STDERR
      --blkio-weight uint16            Block IO (relative weight), between 10 and
1000, or 0 to disable (default 0)
      --blkio-weight-device list       Block IO weight (relative device weight)
(default [])
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
      --cgroup-parent string           Optional parent cgroup for the container
      --cgroupns string                Cgroup namespace to use (host|private)
                                       'host':    Run the container in the Docker
host's cgroup namespace
                                       'private': Run the container in its own private
cgroup namespace
                                       '':        Use the cgroup namespace as
configured by the
                                                  default-cgroupns-mode option on the
daemon (default)
      --cidfile string                 Write the container ID to the file
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler)
period
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler) quota
      --cpu-rt-period int              Limit CPU real-time period in microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
      --cpus decimal                   Number of CPUs
      --cpuset-cpus string             CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
  -d, --detach                         Run container in background and print container
ID
      --detach-keys string             Override the key sequence for detaching a
container
      --device list                    Add a host device to the container
      --device-cgroup-rule list        Add a rule to the cgroup allowed devices list
      --device-read-bps list           Limit read rate (bytes per second) from a
device (default [])
      --device-read-iops list          Limit read rate (IO per second) from a device
(default [])
      --device-write-bps list          Limit write rate (bytes per second) to a device
(default [])
      --device-write-iops list         Limit write rate (IO per second) to a device
(default [])
      --disable-content-trust          Skip image verification (default true)
      --dns list                       Set custom DNS servers
      --dns-option list                Set DNS options
      --dns-search list                Set custom DNS search domains
      --domainname string              Container NIS domain name
      --entrypoint string              Overwrite the default ENTRYPOINT of the image
  -e, --env list                       Set environment variables
      --env-file list                  Read in a file of environment variables
      --expose list                    Expose a port or a range of ports
      --gpus gpu-request               GPU devices to add to the container ('all' to
```

```
pass all GPUs)
      --group-add list                  Add additional groups to join
      --health-cmd string               Command to run to check health
      --health-interval duration        Time between running the check (ms|s|m|h)
(default 0s)
      --health-retries int              Consecutive failures needed to report unhealthy
      --health-start-period duration    Start period for the container to initialize
before starting health-retries countdown (ms|s|m|h) (default 0s)
      --health-timeout duration         Maximum time to allow one check to run
(ms|s|m|h) (default 0s)
      --help                            Print usage
  -h, --hostname string                 Container host name
      --init                            Run an init inside the container that forwards
signals and reaps processes
  -i, --interactive                     Keep STDIN open even if not attached
      --ip string                       IPv4 address (e.g., 172.30.100.104)
      --ip6 string                      IPv6 address (e.g., 2001:db8::33)
      --ipc string                      IPC mode to use
      --isolation string                Container isolation technology
      --kernel-memory bytes             Kernel memory limit
  -l, --label list                      Set meta data on a container
      --label-file list                 Read in a line delimited file of labels
      --link list                       Add link to another container
      --link-local-ip list              Container IPv4/IPv6 link-local addresses
      --log-driver string               Logging driver for the container
      --log-opt list                    Log driver options
      --mac-address string              Container MAC address (e.g., 92:d0:c6:0a:29:33)
  -m, --memory bytes                    Memory limit
      --memory-reservation bytes        Memory soft limit
      --memory-swap bytes               Swap limit equal to memory plus swap: '-1' to
enable unlimited swap
      --memory-swappiness int           Tune container memory swappiness (0 to 100)
(default -1)
      --mount mount                     Attach a filesystem mount to the container
      --name string                     Assign a name to the container
      --network network                 Connect a container to a network
      --network-alias list              Add network-scoped alias for the container
      --no-healthcheck                  Disable any container-specified HEALTHCHECK
      --oom-kill-disable                Disable OOM Killer
      --oom-score-adj int               Tune host's OOM preferences (-1000 to 1000)
      --pid string                      PID namespace to use
      --pids-limit int                  Tune container pids limit (set -1 for
unlimited)
      --platform string                 Set platform if server is multi-platform
capable
      --privileged                      Give extended privileges to this container
  -p, --publish list                    Publish a container's port(s) to the host
  -P, --publish-all                     Publish all exposed ports to random ports
      --pull string                     Pull image before running ("always", "missing",
"never") (default "missing")
  -q, --quiet                           Suppress the pull output
```

```
      --read-only                     Mount the container's root filesystem as read
only
      --restart string                Restart policy to apply when a container exits
(default "no")
      --rm                            Automatically remove the container when it
exits
      --runtime string                Runtime to use for this container
      --security-opt list             Security Options
      --shm-size bytes                Size of /dev/shm
      --sig-proxy                     Proxy received signals to the process (default
true)
      --stop-signal string            Signal to stop the container
      --stop-timeout int              Timeout (in seconds) to stop a container
      --storage-opt list              Storage driver options for the container
      --sysctl map                    Sysctl options (default map[])
      --tmpfs list                    Mount a tmpfs directory
  -t, --tty                           Allocate a pseudo-TTY
      --ulimit ulimit                 Ulimit options (default [])
  -u, --user string                   Username or UID (format:
<name|uid>[:<group|gid>])
      --userns string                 User namespace to use
      --uts string                    UTS namespace to use
  -v, --volume list                   Bind mount a volume
      --volume-driver string          Optional volume driver for the container
      --volumes-from list             Mount volumes from the specified container(s)
  -w, --workdir string                Working directory inside the container
```

Cheat sheet

https://docs.docker.com/get-started/docker_cheatsheet.pdf

Zadania

Zadanie 1. Pobranie obrazu

1. Pobierz obraz `nginx`

Zadanie 2. Uruchomienie obrazu

1. Uruchom kontener z obrazu `httpd`

Zadanie 3. Sprawdzenie listy obrazów

1. Sprawdź listę obrazów komendą `docker image ls`

Zadanie 4. Sprawdzenie działających kontenerów

1. Sprawdź listę działających kontenerów komendą `docker ps`

Zadanie 5. Uruchomienie kontenera na konkretnym porcie

1. Uruchom kontener z obrazu `nginx`
2. Wskaż port 8000 na hoście, na którym ma działać ten kontener
3. Sprawdź, czy `nginx` działa

Zadanie 6. Sprawdzanie logów

1. Sprawdź logi kontenera z poprzedniego zadania za pomocą `docker logs`

Zadanie 7. Uruchomienie kontenera ze zmiennymi środowiskowej

1. Uruchom kontener z obrazu `nginx`
2. Zmień port, na którym nasłuchuje `nginx` za pomocą zmiennej środowiskowej `NGINX_PORT` na 81
3. Wskaż port 8001 na hoście, na którym ma działać ten kontener
4. Sprawdź, czy `nginx` działa

Zadanie 8. Uruchomienie bazy danych

1. Uruchom kontener z obrazu `postgres`
2. Wskaż odpowiednie porty
3. Przekaż odpowiednie zmienne środowiskowe
4. Ustaw odpowiednie volumeny
5. Sprawdź, czy baza danych działa (możesz za pomocą logów)

Zadanie 9. Budowanie obrazu

1. Stwórz `Dockerfile`
2. Stwórz obraz z `nginx`
3. Zbuduj obraz i otaguj go jako `my-nginx`
4. Uruchom kontener z tego obrazu
5. Sprawdź, czy kontener działa (możesz za pomocą logów)

Kubernetes

Co to jest Kubernetes?

Kubernetes to otwarte oprogramowanie do automatycznego wdrażania, skalowania i zarządzania kontenerami aplikacyjnymi. Jest często używane do orkiestrowania kontenerów Dockera, chociaż obsługuje również inne kontenery. Kubernetes dostarcza narzędzi i mechanizmów do zarządzania mikroserwisowymi aplikacjami w sposób zdecentralizowany. Pozwala na definiowanie, uruchamianie i skalowanie aplikacji składających się z wielu kontenerów, umożliwiając elastyczne zarządzanie zasobami. Kubernetes obsługuje automatyczne przywracanie zgodnie z zdefiniowanym stanem oraz umożliwia równomierne rozłożenie obciążenia między różnymi instancjami aplikacji. Dzięki deklaratywnemu podejściu do konfiguracji Kubernetes umożliwia programistom skupienie się na opisie zamierzonego stanu systemu, pozostawiając platformie zadanie samodzielnego dostosowywania rzeczywistego stanu do deklaratywnie określonego stanu. Kubernetes jest szeroko stosowany w środowiskach chmurowych oraz w środowiskach lokalnych do budowy skalowalnych i odpornych na awarie aplikacji opartych na kontenerach.

Architektura

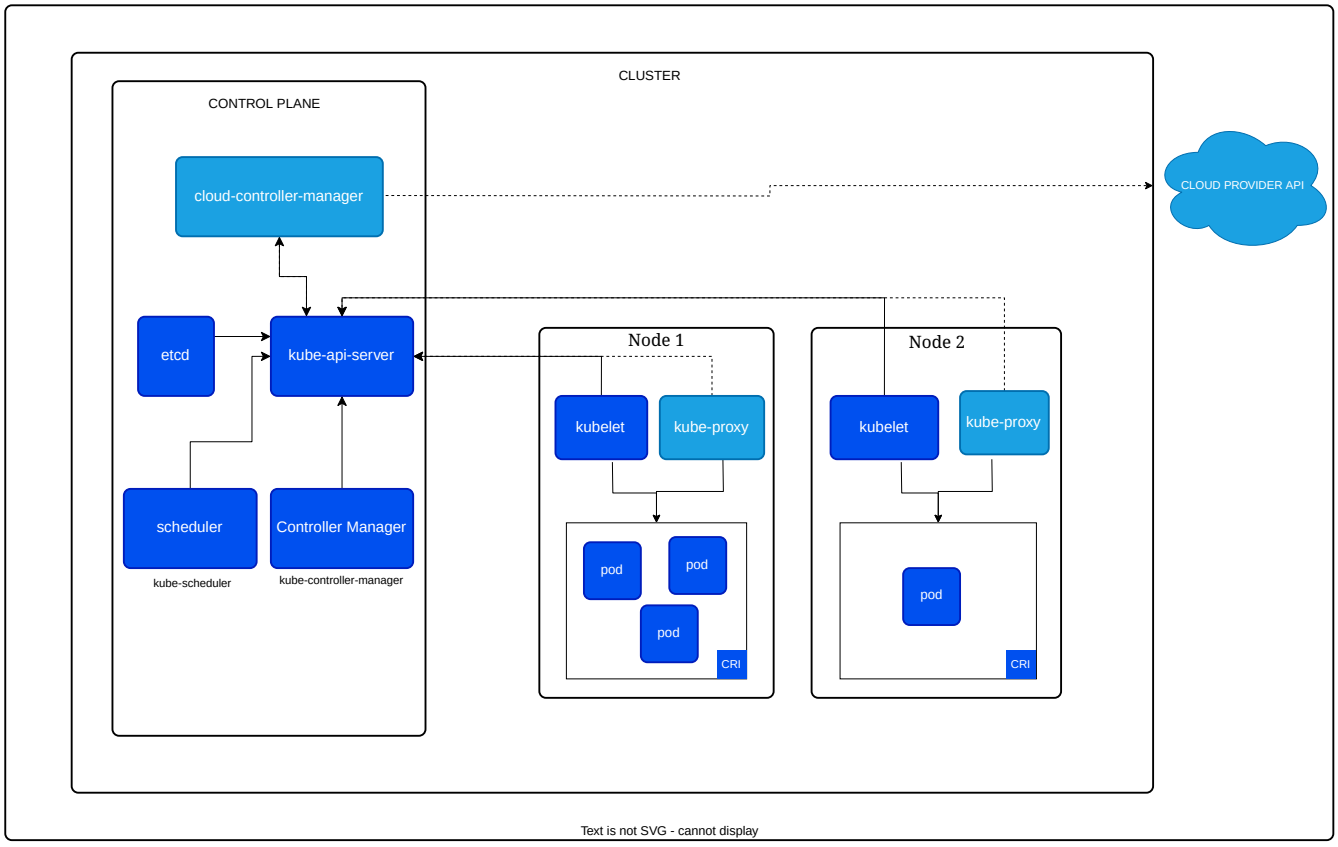


Figure 2. https://kubernetes.io/docs/concepts/architecture/

etcd

`etcd` pełni kluczową rolę wewnątrz klastra Kubernetes jako rozproszony i niezawodny magazyn danych, przechowujący konfigurację klastra, stan zasobów oraz inne ważne informacje w formie kluczy i wartości. To rozproszone repozytorium przechowuje dane w sposób spójny, co pozwala na

synchronizację i dostęp do informacji pomiędzy różnymi węzłami klastra. `etcd` obsługuje automatyczne przywracanie, co sprawia, że jest niezawodnym źródłem prawdy, a jego spójność danych jest kluczowa dla poprawnego działania wszystkich komponentów Kubernetes. Działa również jako mechanizm blokujący, umożliwiając koordynację i rozwiązanie problemów związanych z jednoczesnym dostępem do danych przez różne komponenty klastra.

kube-apiserver

`kube-apiserver` jest komponentem kluczowym w klastrze Kubernetes, pełniącym rolę interfejsu API dla zarządzania klastrami. Jego głównym zadaniem jest wystawianie interfejsu RESTful API, który umożliwia komunikację pomiędzy różnymi komponentami klastra oraz interakcję z użytkownikami i narzędziami zewnętrznymi. `kube-apiserver` odpowiedzialny jest za przyjmowanie żądań API, autentykację, autoryzację, a także przekazywanie tych żądań do odpowiednich komponentów systemu. To centralne miejsce, w którym konfigurowane są obiekty Kubernetes, takie jak Pod, Service czy Deployment, co umożliwia jednolite zarządzanie zasobami klastra.

kube-scheduler

`kube-scheduler` w klastrze Kubernetes odpowiada za decyzje dotyczące tego, na którym węźle klastra należy uruchomić nowo tworzone kontenery. Jego rola polega na analizowaniu dostępnych zasobów w klastrze, takich jak CPU i pamięć, oraz uwzględnianiu wymagań dotyczących zasobów i polityk wdrażania określonych przez użytkownika. Kiedy nowy pod jest tworzony, `kube-scheduler` decyduje, na którym węźle powinien zostać uruchomiony, co pozwala na równomierne rozłożenie obciążenia w klastrze. Ten komponent jest integralną częścią procesu wdrażania i skalowania aplikacji w Kubernetes, zapewniając optymalne wykorzystanie dostępnych zasobów w klastrze.

kube-controller-manager

`kube-controller-manager` pełni kluczową rolę wewnątrz klastra Kubernetes, zarządzając różnymi kontrolerami, które monitorują i utrzymują stan klastra zgodnie z określonymi specyfikacjami. Kontrolery te obejmują różne obszary, takie jak zarządzanie replikami, wdrażanie aplikacji, dostawcy chmury oraz zarządzanie woluminami. Działa w tle, automatyzując zadania administracyjne i zapewniając, że deklaratywny stan klastra jest zawsze zgodny z zamierzonymi specyfikacjami. W ten sposób `kube-controller-manager` wspomaga utrzymanie stabilności, niezawodności i zgodności z oczekiwaniami użytkownika w środowisku Kubernetes.

cloud-controller-manager (opcjonalny)

`cloud-controller-manager` zajmuje się zarządzaniem kontrolerami specyficznymi dla dostawcy chmury. Jego głównym zadaniem jest umożliwienie integracji z zasobami chmurowymi dostawcy, takimi jak AWS, GCP lub Azure, zapewniając optymalne wykorzystanie funkcji oferowanych przez danego dostawcę. `cloud-controller-manager` deleguje kontrolery chmury do zewnętrznego dostawcy, co umożliwia modularność i elastyczność w zarządzaniu różnymi aspektami klastra, takimi jak równoważenie obciążenia, routing sieciowy czy dostosowywanie rozmiaru węzłów. Dzięki temu komponentowi Kubernetes może skutecznie dostosować się do specyfiki środowiska chmurowego, co jest kluczowe dla efektywnego i spójnego zarządzania zasobami w klastrze.

kube-proxy

`kube-proxy` to komponent wewnątrz klastra Kubernetes odpowiedzialny za zarządzanie ruchem sieciowym między różnymi podami. Jego głównym zadaniem jest utrzymanie reguł przekierowań (routingu) na poziomie węzła, umożliwiając komunikację między różnymi podami w klastrze. `kube-proxy` implementuje funkcje takie jak Network Address Translation (NAT) i Load Balancing, umożliwiając dostęp do aplikacji działających w kontenerach na różnych węzłach klastra. Działa zarówno w trybie użytkownika (user space) jak i w trybie jądra (kernel space), dostosowując się do specyfiki środowiska.

Ten komponent jest kluczowy dla zapewnienia komunikacji między podami w klastrze, zarządzając dostępem do usług oraz umożliwiając skalowanie aplikacji w sposób przezroczysty dla użytkownika. Dzięki `kube-proxy`, Kubernetes oferuje jednolity sposób zarządzania ruchem sieciowym, niezależnie od tego, czy aplikacje działają wewnątrz klastra, czy są dostępne publicznie.

kubelet

`kubelet` to agent działający na każdym węźle w klastrze Kubernetes, odpowiedzialny za zarządzanie i utrzymanie kontenerów na danym węźle. Jego główną rolą jest monitorowanie stanu podów (instancji kontenerów) i ich zarządzanie zgodnie z deklaratywną konfiguracją dostarczoną przez `kube-apiserver`. `Kubelet` komunikuje się z serwerem API Dockera (lub innej implementacji kontenerów) na danym węźle, inicjując operacje związane z cyklem życia kontenerów, takie jak ich uruchamianie, zatrzymywanie czy usuwanie. Dodatkowo, Kubelet sprawuje kontrolę nad zasobami węzła, raportując informacje na temat dostępności, zużycia pamięci czy CPU.

`Kubelet` jest kluczowym elementem umożliwiającym realizację deklaratywnego modelu zarządzania zasobami w klastrze Kubernetes. Działa w tle, utrzymując zgodność stanu rzeczywistego węzła z zamierzoną konfiguracją. Jego rola obejmuje także raportowanie stanu podów i węzła do `kube-apiserver`, co umożliwia koordynację działań na poziomie klastra.

Node

Opis

W Kubernetes, node to pojedyncza jednostka obliczeniowa w klastrze, na której uruchamiane są kontenery. Każdy node reprezentuje fizyczną maszynę lub wirtualną maszynę w infrastrukturze klastra. W skład jednego noda wchodzą kluczowe komponenty, takie jak Kubelet, który zarządza podami na danym węźle, oraz Kube-proxy, który odpowiada za przekierowywanie ruchu sieciowego między podami. Node zawiera także runtime kontenerów, na przykład Docker, który wykonuje kontenery na poziomie węzła. W klastrze Kubernetes, pody są uruchamiane na węzłach, tworząc jednostki wdrażania dla aplikacji. Etykietowanie węzłów umożliwia przypisywanie im różnych właściwości, ułatwiając selekcję węzłów do konkretnych zadań. Dynamiczne skalowanie pozwala na elastyczne uruchamianie i zarządzanie podami na różnych węzłach w zależności od dostępności zasobów. Węzły są zarządzane przez kontroler zarządzania węzłami, co obejmuje monitorowanie i utrzymanie ich stanu w klastrze Kubernetes.

Control plane

Control Plane w Kubernetes to centralny zestaw komponentów odpowiedzialnych za zarządzanie i

kontrolowanie działania klastra. Składa się z kluczowych elementów, takich jak API Server, etcd, Kube-scheduler i Controller Manager, które współpracują w celu utrzymania pożądanego stanu klastra. API Server pełni rolę interfejsu komunikacyjnego, etcd przechowuje trwałe dane konfiguracyjne, Kube-scheduler odpowiada za planowanie pracy, a Controller Manager monitoruje stan klastra i podejmuje działania w celu utrzymania spójności. Control Plane jest zarządzane przez główny węzeł w klastrze, który przyjmuje polecenia od użytkowników i koordynuje działanie pozostałych węzłów.

Worker

Worker node w klastrze Kubernetes to węzeł, który pełni rolę wykonawczą, uruchamiając kontenery i hostując pody. Jest to fizyczna maszyna lub wirtualna maszyna, na której działa agent Kubelet, odpowiadający za komunikację z głównym API serwerem kontrolnym i zarządzanie podami. Dodatkowo, na worker node działa Kube-proxy, który odpowiada za zarządzanie ruchem sieciowym między podami. Węzły robocze są kluczowym elementem klastra, zapewniającym miejsce do uruchamiania aplikacji i realizowania obliczeń.

CRI

Opis

CRI, czyli Container Runtime Interface, to standardowy interfejs w Kubernetesie, który umożliwia komunikację między Kubelet (komponentem działającym na węzłach w klastrze) a różnymi runtime'ami kontenerów. CRI pełni kluczową rolę w modularności Kubernetes, pozwalając na używanie różnych runtime'ów kontenerów wewnątrz klastra.

Wspierane CRI

- containerd
- CRI-O
- Docker Engine (cri-dockerd)
- Mirantis Container Runtime

dockershim

W grudniu 2020 roku Kubernetes ogłosił plany usunięcia wsparcia dla dockershim, który był interfejsem CRI używanym przez Docker w celu integracji z Kubeletem. W wydaniu Kubernetes 1.20 dockershim było oznaczone jako przestarzałe, ostatecznie usunięto wsparcie w wersji 1.24.

Cykl Wydań i Wsparcie w Kubernetes

Wydania Kubernetes są planowane i publikowane regularnie, co trzy miesiące. Cykl wydawniczy jest stosunkowo krótki, co oznacza, że nowa stabilna wersja jest publikowana kwartalnie.

Wsparcie dla poszczególnych wersji jest zarządzane zgodnie z zasadami określonymi w "Kubernetes Version Skew Policy". Te zasady obejmują dwie kwestie: wsparcie dla wersji stabilnej (version skew) oraz wsparcie dla konkretnego wydania (release support).

Wersja Stabilna (Version Skew): Kubernetes wspiera wersję stabilną, co oznacza, że nowszy Kubelet (węzeł) może komunikować się z starszym API Serverem (główny węzeł) i odwrotnie, ale z pewnym ograniczeniem. Zaleca się jednak aktualizację wszystkich komponentów klastra do najnowszej wersji.

Wsparcie dla Wydania (Release Support): Oficjalne wsparcie dla danego wydania Kubernetes trwa około 1 roku od daty wydania. W ciągu tego czasu dostarczane są nowe poprawki bezpieczeństwa i łatki dla danej wersji. Po upływie tego okresu wydanie traci oficjalne wsparcie.

Warto zaznaczyć, że dostawcy klastrów oraz narzędzi do zarządzania klastrami mogą oferować dłuższe wsparcie dla konkretnych wersji. Dlatego ważne jest sprawdzenie oficjalnej dokumentacji i polityki wsparcia dla danego dostawcy lub narzędzia, jeśli używasz Kubernetes w konkretnym środowisku.

Aktualizacje Kubernetes są wprowadzane często, aby dostarczać nowe funkcje, poprawiać bezpieczeństwo, zwiększać wydajność oraz dostosowywać platformę do zmieniających się wymagań. Dlatego zaleca się regularne aktualizacje klastra Kubernetes, aby korzystać z najnowszych korzyści i poprawek.

Wersje API

- Alfa
  - Nazwy wersji zawierają alfa (na przykład v1alpha1).
  - Wbudowane wersje API alfa są domyślnie wyłączone i muszą być jawnie włączone w konfiguracji kube-apiserver, aby zostały użyte.
  - Oprogramowanie może zawierać błędy. Włączenie funkcji może ujawnić błędy.
  - Wsparcie dla wersji alfa API może zostać usunięte w dowolnym momencie bez wcześniejszego powiadomienia.
  - API może ulec zmianie w sposób niekompatybilny w późniejszym wydaniu oprogramowania bez wcześniejszego powiadomienia.
  - Oprogramowanie jest zalecane do użytku tylko w krótkotrwałych klastrach testowych ze względu na zwiększone ryzyko błędów i brak długoterminowego wsparcia.
- Beta
  - Nazwy wersji zawierają beta (na przykład v2beta3).
  - Wbudowane wersje API beta są domyślnie wyłączone i muszą być jawnie włączone w konfiguracji kube-apiserver, aby zostały użyte (z wyjątkiem wersji beta API wprowadzonych przed Kubernetes 1.22, które były domyślnie włączone).
  - Wbudowane wersje API beta mają maksymalny okres życia 9 miesięcy lub 3 wydań (zależnie od tego, które jest dłuższe) od wprowadzenia do przestarzałości, a także 9 miesięcy lub 3 wydań (zależnie od tego, które jest dłuższe) od przestarzałości do usunięcia.
  - Oprogramowanie jest dobrze przetestowane. Włączenie funkcji uważane jest za bezpieczne.
  - Wsparcie dla funkcji nie zostanie usunięte, chociaż szczegóły mogą ulec zmianie.
  - Schemat i/lub semantyka obiektów mogą ulec zmianie w sposób niekompatybilny w

późniejszej wersji API beta lub stabilnej. W przypadku takich zmian udostępniane są instrukcje migracyjne. Przystosowanie do kolejnej wersji API beta lub stabilnej może wymagać edycji lub ponownego tworzenia obiektów API i może nie być prostą operacją. Migracja może wymagać czasu przestoju dla aplikacji korzystających z danej funkcji.

    - Oprogramowanie nie jest zalecane do użycia w środowisku produkcyjnym. Kolejne wydania mogą wprowadzać zmiany niekompatybilne. Użycie wersji API beta jest konieczne do przejścia do kolejnych wersji API beta lub stabilnych po tym, jak wersja API beta zostanie przestarzała i przestanie być obsługiwana.

- Stabilne
    - Nazwa wersji to vX, gdzie X to liczba całkowita.
    - Stabilne wersje API pozostają dostępne we wszystkich przyszłych wydaniach w ramach danej wersji głównej Kubernetes, i nie ma obecnych planów na rewizję głównej wersji Kubernetes, która usuwałaby stabilne API.

Wycofywanie API

Zasada #1

API elementy mogą być usuwane jedynie poprzez zwiększenie wersji grupy API.

Po dodaniu elementu API do grupy API w konkretnej wersji, nie można go usunąć z tej wersji ani istotnie zmienić jego zachowania, niezależnie od ścieżki.

Zasada #2

Obiekty API muszą być w stanie przemieszczać się między wersjami API w danym wydaniu bez utraty informacji, z wyjątkiem całych zasobów REST, które nie istnieją w niektórych wersjach.

Na przykład obiekt można zapisać jako v1, odczytać jako v2 i przekonwertować na v1, a rezultujący zasób v1 będzie identyczny z oryginalnym. Reprezentacja w v2 może różnić się od v1, ale system potrafi je konwertować w obie strony. Dodatkowo, każde nowe pole dodane w v2 musi być w stanie przemieszczać się między v1 a z powrotem, co oznacza, że v1 może musieć dodać równoważne pole lub reprezentować je jako adnotację.

Zasada #3

Wersji API w danej ścieżce nie wolno przestarzać na rzecz mniej stabilnej wersji API.

- Wersje GA mogą zastępować wersje beta i alfa.
- Wersje beta mogą zastępować wcześniejsze wersje beta i alfa, ale nie mogą zastępować wersji GA.
- Wersje alfa mogą zastępować wcześniejsze wersje alfa, ale nie mogą zastępować wersji GA ani beta.

Zasada #4a

Żywotność API jest określana przez poziom stabilności API.

- Wersje GA mogą być oznaczone jako przestarzałe, ale nie mogą być usuwane w ramach głównej wersji Kubernetes.
- Wersje beta są przestarzałe nie wcześniej niż 9 miesięcy lub 3 wydania po wprowadzeniu (zależnie od tego, co jest dłuższe) i przestają być obsługiwane 9 miesięcy lub 3 wydania po przestarzałości (zależnie od tego, co jest dłuższe).
- Wersje alfa mogą być usuwane w dowolnej wersji bez wcześniejszego ogłoszenia przestarzałości.

To zapewnia, że wsparcie dla wersji beta obejmuje maksymalne wspierane odchylenie wersji wynoszące 2 wydania, oraz że API nie utyka na niestabilnych wersjach beta, gromadząc użycie w produkcji, które zostanie zakłócone, gdy wsparcie dla wersji beta zakończy się.

Zasada #4b

"Preferowana" wersja API i "wersja przechowywania" dla danej grupy nie mogą się zmieniać, dopóki nie zostanie wydane wsparcie dla obu nowej i poprzedniej wersji.

Użytkownicy muszą być w stanie zaktualizować się do nowej wersji Kubernetes, a następnie cofnąć się do poprzedniej wersji, bez konwersji do nowej wersji API lub doświadczania awarii (chyba że używali wyłącznie funkcji dostępnych w nowszej wersji). Jest to szczególnie istotne w przechowywanej reprezentacji obiektów.

kubectl

Opis

`kubectl` to narzędzie wiersza poleceń używane do interakcji z klastrami Kubernetes. Pozwala użytkownikom na wysyłanie poleceń do serwera Kubernetes, zarządzanie zasobami klastra, wdrażanie aplikacji oraz monitorowanie stanu kontenerów i podów.

Instalacja

https://kubernetes.io/docs/tasks/tools/

Komendy

```
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
create          Create a resource from a file or from stdin
expose          Take a replication controller, service, deployment or pod and expose
it as a new Kubernetes service
```

```
run             Run a particular image on the cluster
set             Set specific features on objects

Basic Commands (Intermediate):
explain         Get documentation for a resource
get             Display one or many resources
edit            Edit a resource on the server
delete          Delete resources by file names, stdin, resources and names, or by
resources and label selector

Deploy Commands:
rollout         Manage the rollout of a resource
scale           Set a new size for a deployment, replica set, or replication
controller
autoscale       Auto-scale a deployment, replica set, stateful set, or replication
controller

Cluster Management Commands:
certificate     Modify certificate resources
cluster-info    Display cluster information
top             Display resource (CPU/memory) usage
cordon          Mark node as unschedulable
uncordon        Mark node as schedulable
drain           Drain node in preparation for maintenance
taint           Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
describe        Show details of a specific resource or group of resources
logs            Print the logs for a container in a pod
attach          Attach to a running container
exec            Execute a command in a container
port-forward    Forward one or more local ports to a pod
proxy           Run a proxy to the Kubernetes API server
cp              Copy files and directories to and from containers
auth            Inspect authorization
debug           Create debugging sessions for troubleshooting workloads and nodes
events          List events

Advanced Commands:
diff            Diff the live version against a would-be applied version
apply           Apply a configuration to a resource by file name or stdin
patch           Update fields of a resource
replace         Replace a resource by file name or stdin
wait            Experimental: Wait for a specific condition on one or many resources
kustomize       Build a kustomization target from a directory or URL

Settings Commands:
label           Update the labels on a resource
annotate        Update the annotations on a resource
completion      Output shell completion code for the specified shell (bash, zsh, fish,
or powershell)
```

```
Other Commands:
api-resources   Print the supported API resources on the server
api-versions    Print the supported API versions on the server, in the form of
"group/version"
config          Modify kubeconfig files
plugin          Provides utilities for interacting with plugins
version         Print the client and server version information

Usage:
kubectl [flags] [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
```

Cheat sheet

https://kubernetes.io/docs/reference/kubectl/cheatsheet/

kubeconfig

Opis

`kubeconfig` to plik konfiguracyjny używany przez narzędzie `kubectl` do skonfigurowania dostępu do klastra Kubernetes. Ten plik zawiera informacje takie jak adres serwera API klastra, dane uwierzytelniające użytkownika, konteksty, ustawienia klastra czy konfiguracje proxy. `kubeconfig` umożliwia użytkownikowi definiowanie różnych konfiguracji dla różnych klastrów lub kontekstów, co pozwala na łatwe przełączanie między różnymi środowiskami Kubernetes.

Zarządzanie przez kubectl

```
Modify kubeconfig files using subcommands like "kubectl config set current-context my-
context".

 The loading order follows these rules:

  1.  If the --kubeconfig flag is set, then only that file is loaded. The flag may
only be set once and no merging takes
place.
  2.  If $KUBECONFIG environment variable is set, then it is used as a list of paths
(normal path delimiting rules for
your system). These paths are merged. When a value is modified, it is modified in the
file that defines the stanza. When
a value is created, it is created in the first file that exists. If no files in the
chain exist, then it creates the
last file in the list.
  3.  Otherwise, ${HOME}/.kube/config is used and no merging takes place.
```

```
Available Commands:
  current-context   Display the current-context
  delete-cluster    Delete the specified cluster from the kubeconfig
  delete-context    Delete the specified context from the kubeconfig
  delete-user       Delete the specified user from the kubeconfig
  get-clusters      Display clusters defined in the kubeconfig
  get-contexts      Describe one or many contexts
  get-users         Display users defined in the kubeconfig
  rename-context    Rename a context from the kubeconfig file
  set               Set an individual value in a kubeconfig file
  set-cluster       Set a cluster entry in kubeconfig
  set-context       Set a context entry in kubeconfig
  set-credentials   Set a user entry in kubeconfig
  unset             Unset an individual value in a kubeconfig file
  use-context       Set the current-context in a kubeconfig file
  view              Display merged kubeconfig settings or a specified kubeconfig file

Usage:
  kubectl config SUBCOMMAND [options]
```

Przykładowy plik kubeconfig

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: fake-ca-file
    server: https://1.2.3.4
  name: development
- cluster:
    insecure-skip-tls-verify: true
    server: https://5.6.7.8
  name: test
contexts:
- context:
    cluster: development
    namespace: frontend
    user: developer
  name: dev-frontend
- context:
    cluster: development
    namespace: storage
    user: developer
  name: dev-storage
- context:
    cluster: test
    namespace: default
    user: experimenter
  name: exp-test
```

```
current-context: ""
kind: Config
preferences: {}
users:
- name: developer
  user:
    client-certificate: fake-cert-file
    client-key: fake-key-file
- name: experimenter
  user:
    # Documentation note (this comment is NOT part of the command output).
    # Storing passwords in Kubernetes client config is risky.
    # A better alternative would be to use a credential plugin
    # and store the credentials separately.
    # See https://kubernetes.io/docs/reference/access-authn-
authz/authentication/#client-go-credential-plugins
    password: some-password
    username: exp
```

Podstawowe obiekty

Obiekty w Kubernetes są abstrakcyjnymi reprezentacjami różnych elementów klastra, definiującymi jego stan i funkcjonalności. To kluczowy koncept w modelu deklaratywnego zarządzania zasobami, gdzie użytkownik opisuje zamierzony stan klastra za pomocą plików konfiguracyjnych YAML, a kontrolery Kubernetes są odpowiedzialne za utrzymanie zgodności rzeczywistego stanu z deklaratywnie zdefiniowanym.

Podstawowe obiekty to m.in. pody, które reprezentują jednostki uruchomieniowe, usługi, które definiują stałe punkty końcowe, i wdrożenia, które pozwalają na zarządzanie replikacją i aktualizacją podów. Dodatkowe obiekty obejmują konfiguracje, tajemnice, przestrzenie nazw i konta usług, które wspierają różne aspekty aplikacji i zarządzania zasobami w klastrze.

Namespace

Opis

W Kubernetes, przestrzenie nazw (Namespaces) to mechanizm, który umożliwia podział klastra na logiczne grupy zasobów. Przestrzenie nazw pozwalają na izolację i segmentację zasobów, co jest szczególnie przydatne w środowiskach, gdzie istnieje wiele aplikacji lub zespołów pracujących w jednym klastrze. Każda przestrzeń nazw posiada swoje unikalne zasoby, takie jak pody, usługi czy konfiguracje, co pozwala na organizację klastra w bardziej przejrzysty sposób. Przestrzenie nazw są używane do unikania konfliktów w nazwach oraz wspierają efektywne zarządzanie i skalowanie klastrów Kubernetes.

Domyślne namespace

1. **default:** Główna przestrzeń nazw, do której zasoby są dodawane, jeśli nie zostanie podana żadna inna przestrzeń nazw podczas tworzenia obiektów.

2. **kube-system:** Przestrzeń nazw zawierająca zasoby systemowe i komponenty samego klastra Kubernetes, takie jak `kube-dns`, `kube-proxy` czy `coredns`.
3. **kube-public:** Przestrzeń nazw dostępna do odczytu przez wszystkich użytkowników (read-only) i zawierająca zasoby, które mają być dostępne publicznie w klastrze.
4. **kube-node-lease:** Przestrzeń nazw zawierająca obiekty Lease, które służą do monitorowania i zarządzania stanem węzłów w klastrze.

Przykład imperatywny

```
kubectl create namespace NAMESPACE_NAME
```

Przykład deklaratywny

```
apiVersion: v1
kind: Namespace
metadata:
  name: NAMESPACE_NAME
```

Zapisz ten plik, na przykład jako `my-namespace.yaml`, a następnie zastosuj definicję za pomocą narzędzia `kubectl` poniższą komendą:

```
kubectl apply -f my-namespace.yaml
```

Listowanie przestrzeni nazw

```
kubectl get namespaces
```

Pobieranie obiektów z namespace

Aby pobrać obiekt z określonej przestrzeni nazw w Kubernetes, użyj poniższej komendy `kubectl`:

```
kubectl get OBJECT_TYPE -n NAMESPACE_NAME OBJECT_NAME
```

Gdzie:

- `OBJECT_TYPE` to rodzaj obiektu, np. `pods`, `services`, `deployments`, itp.
- `NAMESPACE_NAME` to nazwa przestrzeni nazw, z której chcesz pobrać obiekt.
- `OBJECT_NAME` to nazwa konkretnego obiektu w danej przestrzeni nazw.

Domyślnie, jeśli nie zostanie podana przestrzeń nazw, `kubectl` korzysta z przestrzeni nazw "default". Przykładowo, aby pobrać informacje o podzie o nazwie "moj-pod" z przestrzeni nazw "default", użyj:

```
kubectl get pods moj-pod
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#namespace-v1-core

Pod

Opis

Pody w Kubernetes (K8s) są najmniejszymi jednostkami uruchomieniowymi w klastrze. Reprezentują abstrakcję pojedynczego egzemplarza działającego kontenera lub grupy kontenerów, współdzielących przestrzeń sieciową i pamięci na jednym węźle klastra. Pody są tworzone, zarządzane i monitorowane przez Kubelet (agenta na węźle klastra), a każdy pod może zawierać jedną lub więcej aplikacji, które dzielą te same zasoby i mają dostęp do wspólnego środowiska. Pody mogą być dynamicznie skalowalne, a zarządzanie nimi obejmuje zarówno utrzymanie ich liczby, jak i rozdział ruchu sieciowego oraz równoważenie obciążenia.

Przykład imperatywny

```
kubectl create run moj-pod --image=nginx
```

Przykład deklaratywny

```
apiVersion: v1
kind: Pod
metadata:
  name: moj-pod
spec:
  containers:
  - name: kontener-nginx
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
    env:
    - name: MOJA_ZMIENNA
      value: "wartosc"
    ports:
    - containerPort: 80
      protocol: TCP
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#pod-v1-core

Deployment

Opis

Deployment w Kubernetes to obiekt, który definiuje i zarządza cyklem życia replik aplikacji w klastrze. Jest to zalecane narzędzie do wdrażania aplikacji, które oferuje deklaratywny sposób zarządzania replikami podów. Deployment umożliwia automatyczne skalowanie, aktualizacje wersji aplikacji, a także zapewnia równoważenie obciążenia i automatyczną obsługę awarii, co przyczynia się do nieprzerwanego dostarczania aplikacji w kontenerach. Przy użyciu Deployment, można zdefiniować i kontrolować stan aplikacji, a Kubernetes zadba o jego utrzymanie.

Strategie uaktualnienia

- RollingUpdate:
  - Jest to strategia domyślna.
  - Nowe pody są wdrażane stopniowo, jeden po drugim, minimalizując wpływ aktualizacji na dostępność aplikacji.
  - Kontrolowane jest, ile maksymalnie nowych podów może być wdrożonych jednocześnie (określane przez parametr `maxSurge`).
  - Jednocześnie kontrolowane jest, ile minimalnie starych podów musi być utrzymanych w trakcie aktualizacji (określane przez parametr `maxUnavailable`).

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
```

- Recreate:
  - Wszystkie stare pody są zatrzymywane jednocześnie, a następnie wdrażane są nowe pody.
  - W tym przypadku, przez krótki okres, aplikacja może być niedostępna.
  - Ta strategia jest prostsza, ale może być mniej zalecana w przypadku aplikacji wymagających ciągłej dostępności.

```
strategy:
  type: Recreate
```

Przykład imperatywny

```
kubectl create deployment moj-deployment --image=nginx
```

Przykład deklaratywny

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: moj-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: moja-aplikacja
  template:
    metadata:
      labels:
        app: moja-aplikacja
    spec:
      containers:
      - name: kontener-nginx
        image: nginx
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
        env:
        - name: MOJA_ZMIENNA
          value: "wartosc"
        ports:
        - containerPort: 80
          protocol: TCP
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#deployment-v1-apps

ReplicaSet

Opis

ReplicaSet w Kubernetes to obiekt, który definiuje żądaną liczbę replik (kopii) podów działających w klastrze. Jego głównym celem jest utrzymanie stałej liczby replik, co umożliwia skalowanie aplikacji. ReplicaSet monitoruje stany podów i automatycznie utrzymuje zdefiniowaną liczbę replik, wznawiając lub tworząc nowe podczas awarii lub innych zmian w klastrze. Jest używany wraz z Deploymentem, który dostarcza bardziej zaawansowanego zarządzania cyklem życia replik i wersjami aplikacji.

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#replicaset-v1-apps

DaemonSet

Opis

DaemonSet w Kubernetes to obiekt, który zapewnia uruchomienie jednej repliki poda na każdym węźle klastra. Jego głównym zadaniem jest utrzymanie identycznej kopii poda na wszystkich węzłach, co czyni go idealnym dla zadań, które muszą być uruchomione na każdym węźle, takich jak monitorowanie, zbieranie logów czy dostarczanie specyficznych usług. DaemonSet automatycznie tworzy i usuwa repliki w zależności od zmian w klastrze, umożliwiając skalowalne i jednolite wdrożenie na wszystkich węzłach. Jest często używany do instalacji oprogramowania infrastrukturalnego na każdym węźle, na przykład agentów monitoringu czy proxy.

Przykład imperatywny

```
kubectl create daemonset moj-daemonset --image=nginx
```

Przykład deklaratywny

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: moj-daemonset
spec:
  selector:
    matchLabels:
      app: moja-aplikacja
  template:
    metadata:
      labels:
        app: moja-aplikacja
    spec:
      containers:
      - name: kontener-nginx
        image: nginx
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#daemonset-v1-apps

StatefulSets

Opis

StatefulSets w Kubernetes to obiekty używane do wdrażania i zarządzania aplikacjami, które wymagają trwałych, unikalnych identyfikatorów i stabilnych adresów sieciowych, takich jak bazy danych. Są one szczególnie przydatne dla aplikacji, które przechowują dane stanowe, ponieważ gwarantują, że instancje StatefulSet mają stabilne nazwy oraz indeksy, co ułatwia identyfikację i dostęp do nich. StatefulSets automatycznie zarządzają cyklem życia podów, zapewniając unikalne nazwy hostów, które pozostają niezmienne nawet po ponownym uruchomieniu. Są idealne do zastosowań, gdzie ważna jest kolejność uruchamiania, takich jak klastry baz danych z replikacją.

Przykład deklaratywny

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  clusterIP: None
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: moj-statefulset
spec:
  serviceName: "nginx"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: kontener-nginx
        image: nginx
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#statefulset-v1-apps

Job

Opis

W Kubernetes, Job to kontroler, który zarządza zadaniami wykonywalnymi jednorazowo w klastrze. Job zapewnia, że zadanie jest zakończone poprawnie przed zamknięciem. Jeśli zadanie kończy się błędem, Job może zrestartować go, aż osiągnie sukces. Jest to przydatne do zadań, które muszą być wykonane dokładnie raz, takich jak przetwarzanie danych czy obliczenia wsadowe. Job w K8s umożliwia definiowanie parametrów takich jak liczba replik zadania, strategia restartowania oraz obsługa błędów, co pozwala na skuteczne zarządzanie wykonywaniem zadań w klastrze.

Przykład imperatywny

```
kubectl create job moje-zadanie --image=obraz-kontenera --restart=OnFailure -- command
--argument1=value1 --argument2=value2
```

Przykład deklaratywny

```
apiVersion: batch/v1
kind: Job
metadata:
  name: moje-zadanie
spec:
  template:
    metadata:
      labels:
        app: moja-aplikacja
    spec:
      restartPolicy: OnFailure
      containers:
      - name: moj-kontener
        image: obraz-kontenera
        command: ["sh", "-c", "echo Hello Kubernetes! && sleep 3600"]
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
  backoffLimit: 2
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#job-v1-batch

CronJob

Opis

CronJob w Kubernetes to kontroler, który umożliwia planowanie i regularne wykonywanie zadań na podstawie składni czasu znanego z systemu Unixowego crona. Może być używany do automatyzacji powtarzających się operacji, takich jak wykonywanie kopii zapasowych, aktualizacje danych, czy czyszczenie zasobów. Definiuje się go za pomocą manifestu YAML, gdzie określa się harmonogram, obraz kontenera oraz inne parametry, takie jak limity czasowe czy strategię restartowania. CronJob w K8s zapewnia elastyczne rozwiązanie dla pracy w tle, automatyzując cykliczne czynności w klastrze.

Cron schedule syntax

```
# ┌──────────────────────────── minute (0 - 59)
# │ ┌────────────────────────── hour (0 - 23)
# │ │ ┌──────────────────────── day of the month (1 - 31)
# │ │ │ ┌────────────────────── month (1 - 12)
# │ │ │ │ ┌──────────────────── day of the week (0 - 6) (Sunday to
Saturday;
# │ │ │ │ │                                    7 is also Sunday on some systems)
# │ │ │ │ │                                    OR sun, mon, tue, wed, thu, fri,
sat
# │ │ │ │ │
# * * * * *
```

Przykład imperatywny

```
kubectl create cronjob moj-cronjob --image=obraz-kontenera --schedule="*/5 * * * *"
--restart=OnFailure -- command --argument1=value1 --argument2=value2
```

Przykład deklaratywny

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
```

```
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#cronjob-v1-batch

Service

Opis

W Kubernetes usługa (Service) to abstrakcja, która definiuje stały punkt dostępowy do jednego lub wielu podów w klastrze. Usługi pozwalają na komunikację między różnymi komponentami aplikacji, niezależnie od ich położenia czy dynamicznie przypisywanych adresów IP. Usługi Kubernetes obsługują równoważenie obciążenia, umożliwiając skalowanie aplikacji, oraz oferują mechanizmy odkrywania usług, dzięki czemu inne komponenty mogą dynamicznie odnajdywać usługi w klastrze.

Przykład imperatywny

```
kubectl create service clusterip moja-usluga --tcp=80:8080
```

Przykład deklaratywny

```
apiVersion: v1
kind: Service
metadata:
  name: moja-usluga
spec:
  selector:
    app: moja-aplikacja
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#service-v1-core

Labels and Selectors

Opis

Etykiety (labels) w Kubernetes to kluczowe elementy metadanych, które umożliwiają dodanie do obiektów informacji identyfikacyjnych i organizacyjnych. Są to pary klucz-wartość przypisane do zasobów Kubernetes, które pozwalają na ich jednoznaczne oznaczenie oraz kategoryzację. Etykiety są używane do różnych celów, takich jak identyfikacja, grupowanie, selekcja i organizacja zasobów w klastrze.

```
metadata:
  labels:
    app: frontend
    environment: production
```

Selektory w Kubernetes to mechanizm używany do identyfikowania i wybierania zasobów (takich jak pod, serwis, itp.) na podstawie ich etykiet. Selektory pozwalają na definiowanie kryteriów, które muszą być spełnione przez etykiety zasobów, aby zostały wybrane.

```
spec:
  selector:
    app: frontend
    environment: production
```

Well-Known Labels

https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/

https://kubernetes.io/docs/reference/labels-annotations-taints/

Annotation

Opis

W Kubernetes, annotations (adnotacje) to metadane, czyli klucz-wartość, które można przypisać do obiektów w klastrze, takich jak pod, usługa, czy przestrzeń nazw. Adnotacje nie mają bezpośredniego wpływu na działanie samego obiektu, ale mogą być używane do przechowywania dodatkowych informacji, opisów czy konfiguracji.

Adnotacje są elastycznym mechanizmem, który pozwala użytkownikom na dostosowanie obiektów według własnych potrzeb, bez ingerencji w ich podstawowe atrybuty. Przykłady użycia adnotacji to dodawanie informacji auditowych, zarządzanie konfiguracją, czy dostarczanie dodatkowych danych do zautomatyzowanego systemu zarządzania klastrami. Adnotacje można odczytywać i modyfikować zarówno za pomocą narzędzia `kubectl`, jak i interfejsu API Kubernetesa.

```
metadata:
```

```
  annotations:
    example.com/description: "To jest mój przykładowy pod."
    team: "operations"
```

Uwierzytelnianie

ServiceAccount

Opis

ServiceAccount w Kubernetes to zasób, który dostarcza jednoznaczną identyfikację dla podów w klastrze. Każdy ServiceAccount ma przypisany unikalny token uwierzytelniający, umożliwiający podom dostęp do API serwera Kubernetes i innych zasobów klastra. ServiceAccounty są powiązane z rolami i uprawnieniami, co umożliwia precyzyjne zarządzanie dostępem podów do różnych zasobów. Stanowią ważny element zarządzania bezpieczeństwem i kontroli dostępu w środowisku Kubernetes.

Przykład deklaratywny

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: moj-service-account
  namespace: moj-namespace
```

Wyciągnięcie tokena

```
kubectl create token moj-service-account -n moj-namespace
```

Role/ClusterRole

Opis

Role/ClusterRole to zasób, który definiuje zestaw uprawnień dla operacji na zasobach w klastrze. Każda rola jest przypisana do konkretnego namespace, a jej zastosowanie ogranicza się do jednego obszaru klastra. Role są wykorzystywane do precyzyjnego zarządzania dostępem do zasobów, takich jak pod, usługi czy konfiguracje. Rola klastrowa za to nie ma przypisanego namespace.

Przykład deklaratywny

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
```

```
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

RoleBinding/ClusterRoleBinding

Opis

W Kubernetes, RoleBinding/ClusterRoleBinding to zasób, który ustanawia powiązanie między konkretnym użytkownikiem, grupą użytkowników lub serwisem a rolą w określonym namespace (lub bez namespace dla ClusterRoleBinding). RoleBinding/ClusterRoleBinding definiuje, jakie uprawnienia są przyznane danemu podmiotowi w kontekście danego obszaru klastra.

Przykład deklaratywny

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "dave" to read secrets in the "development" namespace.
# You need to already have a ClusterRole named "secret-reader".
kind: RoleBinding
metadata:
  name: read-secrets
  #
  # The namespace of the RoleBinding determines where the permissions are granted.
  # This only grants permissions within the "development" namespace.
  namespace: development
subjects:
- kind: User
  name: dave # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

```
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to read secrets in
any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Sieć

Sterowniki CNI

Sterowniki CNI (Container Network Interface) w Kubernetes (K8s) są to elementy oprogramowania, które zarządzają konfiguracją i działaniem sieci pomiędzy kontenerami w klastrze. Służą one do ustanawiania, konfigurowania i utrzymania połączeń sieciowych pomiędzy różnymi jednostkami obliczeniowymi w klastrze Kubernetes.

Sterowniki CNI pełnią kluczową rolę w umożliwianiu komunikacji między kontenerami, zarządzaniu adresacją IP, a także wdrożeniu polityk bezpieczeństwa sieciowego. Wspierają różne technologie sieciowe, takie jak przełączniki programowalne, routery i protokoły komunikacyjne.

Ich głównym zadaniem jest zapewnienie spójności sieciowej oraz umożliwienie elastycznego dostosowywania konfiguracji sieciowej w miarę potrzeb. Sterowniki CNI umożliwiają integrację różnych rozwiązań sieciowych, co pozwala na dostosowanie infrastruktury sieciowej klastra do konkretnych wymagań aplikacji i środowiska.

Każdy kontener w klastrze K8s posiada swoje unikalne interfejsy sieciowe, a sterowniki CNI zapewniają, że te interfejsy są skonfigurowane zgodnie z zasadami i wymaganiami określonymi dla danego klastra. Ostatecznie, sterowniki CNI ułatwiają elastyczne i skalowalne zarządzanie siecią w środowisku Kubernetes.

Przykłady Sterowników CNI w Kubernetes

Flannel

Flannel to prosty i lekki sterownik CNI, który dostarcza warstwę abstrakcji dla sieci kontenerowej. Umożliwia konfigurację podsieci dla kontenerów.

Calico

Calico to otwarte oprogramowanie, które implementuje sieć opartą na protokole BGP (Border

Gateway Protocol). Zapewnia zaawansowane funkcje bezpieczeństwa, takie jak polityki sieciowe.

Weave

Weave to elastyczny sterownik CNI, który automatycznie tworzy sieć między kontenerami w klastrze. Posiada wbudowaną funkcjonalność routingu i wsparcie dla wielu chmur.

Cilium

Cilium to sterownik, który łączy funkcje sieciowe z bezpieczeństwem. Oferuje zaawansowane funkcje takie jak ochrona przed atakami, detekcja intruzów i kontrola dostępu.

Kube-router

Kube-router to lekki sterownik CNI zaimplementowany w Go, który oferuje obsługę wielu protokołów sieciowych, w tym BGP.

Antrea

Antrea to sterownik CNI rozwijany przez projekt CNCF (Cloud Native Computing Foundation), który dostarcza funkcje zarówno w zakresie sieci, jak i bezpieczeństwa dla klastrów Kubernetes.

Rodzaje Service

ClusterIP

ClusterIP to domyślny rodzaj usługi w Kubernetes. Przypisuje stały adres IP wewnętrzny do usługi, który jest dostępny tylko w obrębie klastra. Jest to użyteczne, gdy komponenty aplikacji w klastrze muszą komunikować się między sobą.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

NodePort

NodePort przypisuje stały port na każdym węźle w klastrze i przekierowuje ruch z tego portu do usługi. Pozwala to na dostęp do usługi z zewnątrz klastra.

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: NodePort
```

LoadBalancer

LoadBalancer automatycznie przydzielana jest zewnętrznemu równoważnikowi obciążenia, który przekierowuje ruch do serwisu. Jest to przydatne, gdy potrzebujesz publicznego dostępu do usługi.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: LoadBalancer
```

ExternalName

ExternalName to rodzaj usługi, który działa jako alias dla zewnętrznego usługodawcy. Pozwala na dostęp do zewnętrznego serwisu przez DNS.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ExternalName
  externalName: my.database.example.com
```

Ingress

Opis

Ingress w Kubernetes to mechanizm umożliwiający zarządzanie dostępem zewnętrznym do usług

w klastrze. Działa jako kontroler obsługujący żądania HTTP i HTTPS, umożliwiając konfigurację reguł routingu na podstawie ścieżek URL, hostów i innych atrybutów zapytania. Ingress obsługuje również terminację TLS/SSL, umożliwiając dekodowanie zaszyfrowanych żądań HTTPS. Pozwala na elastyczne definiowanie reguł przekierowywania ruchu do różnych usług w klastrze.

Przykład deklaratywny

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /app
        pathType: Prefix
        backend:
          service:
            name: myapp-service
            port:
              number: 80
  tls:
  - hosts:
    - myapp.example.com
    secretName: myapp-tls-secret
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#ingress-v1-networking-k8s-io

Ingress Controller

Opis

Ingress Controller w Kubernetes to komponent odpowiedzialny za implementację i obsługę zasobów Ingress. Jest to oprogramowanie lub moduł, który działa jako proxy HTTP/HTTPS, przetwarzając ruch zewnętrzny i kierując go do odpowiednich usług w klastrze na podstawie zdefiniowanych reguł Ingress. Ingress Controller może obsługiwać różne aspekty, takie jak zarządzanie ścieżkami URL, routingiem na podstawie hostów DNS oraz terminacją TLS/SSL. Istnieje wiele dostępnych Ingress Controllerów, a ich wybór zależy od konkretnych wymagań i preferencji konfiguracyjnych klastra.

Przykłady Ingress Controllerów

Nginx Ingress Controller

Nginx Ingress Controller to popularny kontroler Ingress oparty na serwerze proxy Nginx. Zapewnia zaawansowane funkcje routingu i obsługi TLS.

Traefik Ingress Controller

Traefik to dynamiczny kontroler Ingress, który automatycznie odkrywa usługi w klastrze. Posiada intuicyjny interfejs konfiguracyjny i obsługuje wiele backendów.

HAProxy Ingress Controller

HAProxy Ingress to kontroler Ingress, który wykorzystuje silnik HAProxy do przekierowywania ruchu. Posiada zaawansowane funkcje i obsługuje wiele trybów równoważenia obciążenia.

Contour Ingress Controller

Contour to kontroler Ingress rozwijany przez projekt Contour, bazujący na Envoy Proxy. Posiada zaawansowane funkcje routingu i obsługi TLS.

NetworkPolicy

Opis

NetworkPolicy w Kubernetes to zasób, który definiuje zasady bezpieczeństwa dla ruchu sieciowego pomiędzy różnymi podami w klastrze. Umożliwia precyzyjną kontrolę dostępu do usług i zasobów, definiując, które pody mogą komunikować się ze sobą, a które nie. NetworkPolicy pozwala na określenie reguł, takich jak blokowanie lub umożliwianie ruchu na podstawie etykiet podów, portów, protokołów i innych atrybutów.

Przykład deklaratywny

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
spec:
  podSelector:
    matchLabels:
      role: backend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
```

```
      port: 80
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database
    ports:
    - protocol: TCP
      port: 5432
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#networkpolicy-v1-networking-k8s-io

Storage

ConfigMap

Opis

ConfigMap to to zasób służący do przechowywania konfiguracji aplikacji. Jest to sposób na odseparowanie konfiguracji od kodu aplikacji, umożliwiający elastyczne zarządzanie ustawieniami bez konieczności modyfikowania kodu źródłowego. ConfigMap może być używany do przekazywania informacji, takich jak zmienne środowiskowe, pliki konfiguracyjne czy inne dane, do kontenerów uruchamianych w klastrze Kubernetes. Aplikacje mogą dynamicznie odczytywać wartości z ConfigMap, co ułatwia dostosowywanie konfiguracji bez konieczności ponownego uruchamiania kontenerów.

Przykład deklaratywny

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#configmap-v1-core

Secret

Opis

Secret to zasób służący do bezpiecznego przechowywania informacji poufnych, takich jak hasła, klucze API czy certyfikaty. Sekrety są używane do separacji i zarządzania wrażliwymi danymi w klastrze. Mogą być wykorzystywane przez aplikacje jako zabezpieczony sposób przechowywania danych uwierzytelniających lub innych poufnych informacji, a dostęp do nich jest kontrolowany na poziomie klastra. Sekrety mogą być używane w kontenerach jako zmienne środowiskowe lub zamontowane jako systemy plików, umożliwiając aplikacjom dostęp do poufnych danych bez konieczności trzymania ich w kodzie źródłowym.

Typy sekretów

1. **Opaque (Nieustrukturyzowany)**
   - `type: Opaque`
   - Jest to domyślny typ Secret, który traktuje dane jako nieustrukturyzowane bity. Oznacza to, że Secret może przechowywać dowolne dane binarne.
2. **kubernetes.io/tls (TLS/SSL)**
   - `type: kubernetes.io/tls`
   - Służy do przechowywania certyfikatu SSL/TLS oraz odpowiadającego mu klucza prywatnego.
3. **kubernetes.io/dockerconfigjson (Docker Registry Credentials)**
   - `type: kubernetes.io/dockerconfigjson`
   - Wykorzystywany do przechowywania danych uwierzytelniających potrzebnych do dostępu do prywatnych repozytoriów Docker.
4. **kubernetes.io/basic-auth (Basic Authentication)**
   - `type: kubernetes.io/basic-auth`
   - Zawiera dane do uwierzytelniania Basic Auth, takie jak nazwa użytkownika i hasło.
5. **bootstrap.kubernetes.io/token (Bootstrap Token)**
   - `type: bootstrap.kubernetes.io/token`
   - Używany do dostarczania tokena, który jest używany podczas inicjalizacji klastra Kubernetes.
6. **kubernetes.io/service-account-token (Service Account Token)**
   - `type: kubernetes.io/service-account-token`
   - Automatycznie generowany Secret, który zawiera token dostępu dla konta usługi w klastrze.

Przykład deklaratywny

```
apiVersion: v1
kind: Secret
metadata:
  name: moj-secret
type: Opaque
data:
  haslo: cGFzc3dvcmQxMjM=
```

```
apiVersion: v1
kind: Secret
metadata:
  name: moj-tls-secret
type: kubernetes.io/tls
data:
  tls.crt: BASE64_ENCODED_CERTIFICATE
  tls.key: BASE64_ENCODED_PRIVATE_KEY
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#secret-v1-core

EmptyDir

Opis

EmptyDir w Kubernetes to rodzaj zasobu pamięci masowej, który jest używany jako wolumin tymczasowy do współdzielenia danych między kontenerami w ramach tego samego poda. EmptyDir jest tworzony, gdy pod jest uruchamiany na węźle, a jego zawartość jest pusta. Podczas działania poda, kontenery w tym podzie mogą zapisywać i odczytywać dane z tego woluminu.

Przykład

```
apiVersion: v1
kind: Pod
metadata:
  name: moj-pod
spec:
  containers:
  - name: kontener-1
    image: busybox
    volumeMounts:
    - name: moj-emptydir
      mountPath: /data
  - name: kontener-2
    image: busybox
    volumeMounts:
```

```
    - name: moj-emptydir
      mountPath: /data
  volumes:
  - name: moj-emptydir
    emptyDir: {}
```

HostPath

Opis

hostPath w Kubernetes to typ woluminu, który umożliwia zamontowanie ścieżki z hosta bezpośrednio do poda. Oznacza to, że dane dostępne na węźle fizycznym mogą być udostępniane dla kontenerów wewnątrz poda. Jednak korzystanie z hostPath może być ryzykowne, ponieważ narusza izolację kontenerów i może prowadzić do problemów z bezpieczeństwem oraz przenośnością aplikacji.

Przykład

```
apiVersion: v1
kind: Pod
metadata:
  name: moj-pod
spec:
  containers:
  - name: kontener-1
    image: busybox
    volumeMounts:
    - name: moj-hostpath
      mountPath: /data
  volumes:
  - name: moj-hostpath
    hostPath:
      path: /ścieżka/na/węźle
```

CSI

Opis

CSI, czyli Container Storage Interface, to standardowy interfejs dla systemów zarządzania pamięcią masową w kontenerach w środowisku Kubernetes. Działa jako most między klastrem Kubernetes a różnymi rozwiązaniami do przechowywania danych, umożliwiając dynamiczne przydzielanie i zwalnianie przestrzeni dyskowej dla kontenerów. CSI pozwala na integrację różnorodnych rozwiązań do przechowywania danych, co umożliwia elastyczne dostosowanie do potrzeb aplikacji w kontenerach. Dzięki CSI wdrażanie i zarządzanie przechowywaniem danych w klastrze Kubernetes staje się bardziej interoperacyjne i zgodne ze standardami.

Przykłady sterowników

1. **AWS EBS (Elastic Block Store)**
   - Dostawca: Amazon EBS
   - Pozwala na dynamiczne przydzielanie woluminów EBS do kontenerów w klastrze Kubernetes.
2. **Azure Disk**
   - Dostawca: Microsoft Azure
   - Umożliwia dynamiczne przydzielanie dysków Azure dla kontenerów.
3. **GCE PD (Google Compute Engine Persistent Disk)**
   - Dostawca: Google Cloud
   - Pozwala na dynamiczne przydzielanie persistentnych dysków dla kontenerów w klastrze Kubernetes.
4. **Ceph**
   - Dostawca: Ceph
   - Integruje Kubernetes z rozproszonym systemem plików Ceph, umożliwiając dynamiczne przydzielanie zasobów pamięci masowej.
5. **NFS (Network File System)**
   - Dostawca: NFS
   - Umożliwia podłączanie woluminów NFS do kontenerów, co jest przydatne, gdy potrzebujesz współdzielonego dostępu do danych.
6. **vSphere**
   - Dostawca: VMware vSphere
   - Integruje Kubernetes z infrastrukturą wirtualizacyjną vSphere, umożliwiając dynamiczne zarządzanie woluminami.

StorageClass

Opis

StorageClass to zasób definiujący dynamiczny sposób przydzielania pamięci masowej dla aplikacji w klastrze. Określa typy pamięci masowej dostępne dla klastra oraz parametry z nimi związane, takie jak dostawca i konfiguracja. StorageClass umożliwia dynamiczne tworzenie woluminów na żądanie, co eliminuje konieczność ręcznego zarządzania dostępnym magazynem danych. Jest kluczowym elementem w ułatwianiu elastycznego i efektywnego zarządzania przechowywaniem w klastrze Kubernetes.

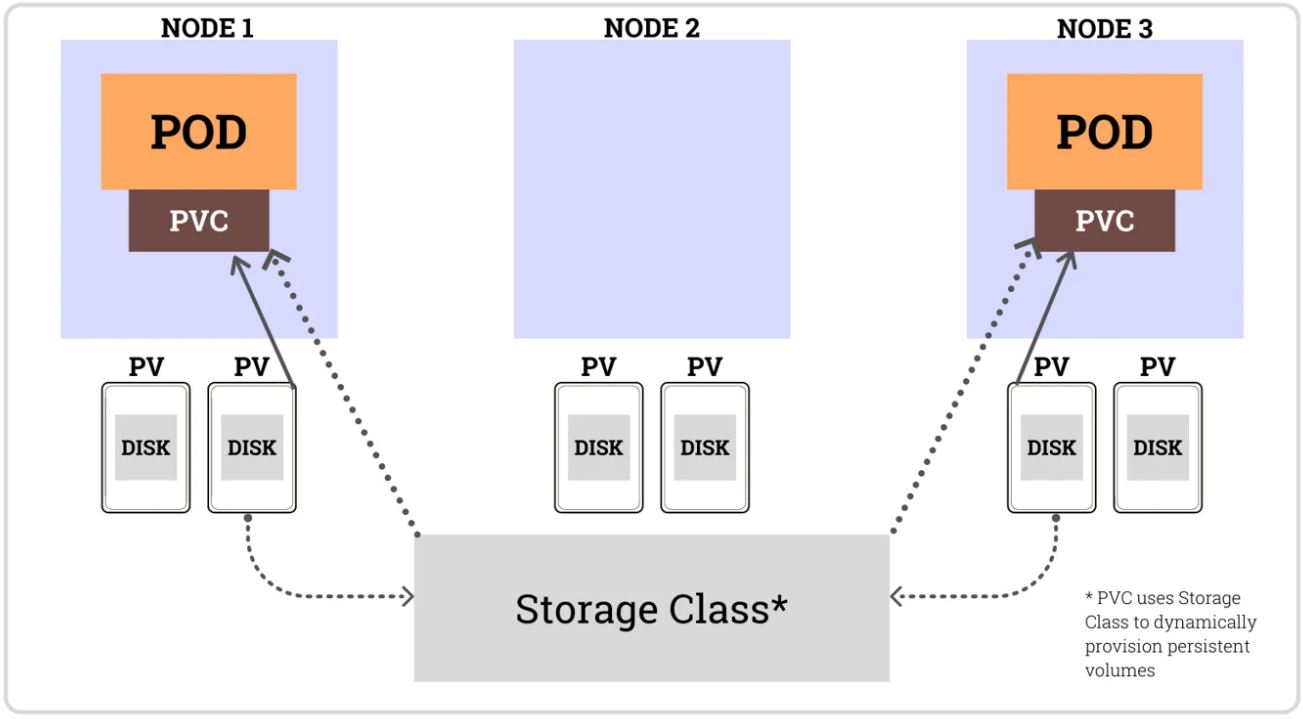


Figure 3. https://blog.mayadata.io/kubernetes-storage-basics-pv-pvc-and-storageclass

Przykład deklaratywny

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#storageclass-v1-storage-k8s-io

PV

Opis

PV (Persistent Volume) w Kubernetes to abstrakcyjna reprezentacja zasobu pamięci masowej w klastrze. PV reprezentuje fizyczne lub wirtualne zasoby pamięci masowej, takie jak dyski, partycje dyskowe, czy woluminy sieciowe, które mogą być udostępniane dla aplikacji działających w klastrze. PV umożliwia odseparowanie dostępu do pamięci masowej od aplikacji, co pozwala na elastyczne zarządzanie przechowywaniem danych w klastrze.

Przykład deklaratywny

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  nfs:
    path: /my/nfs/path
    server: nfs-server.example.com
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#persistentvolume-v1-core

PVC

Opis

PVC (Persistent Volume Claim) w Kubernetes to abstrakcyjny sposób, który pozwala aplikacjom wnioskować o dostęp do pamięci masowej w klastrze. PVC jest zasobem, który pozwala programom na żądanie określonej ilości pamięci masowej o określonych cechach, takich jak pojemność, tryb dostępu czy klasa pamięci masowej.

Przykład deklaratywny

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi
```

API

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#persistentvolumeclaim-v1-core

Dodatkowe funkcjonalności

NodeSelector

Opis

nodeSelector w Kubernetes to funkcja, która umożliwia ograniczenie, na których węzłach klastra powinny być uruchamiane konkretne pody. Działa to poprzez dostarczenie specyficznych etykiet (labels) węzłom, a następnie określenie tych etykiet w definicji poda przy użyciu nodeSelector.

Podczas tworzenia poda, nodeSelector jest używane do wskazania etykiet węzła, na którym chcesz uruchomić dany pod. Jeśli etykieta na węźle odpowiada etykiecie w nodeSelector, pod może być uruchomiony na tym węźle. Jeśli nie ma dopasowania, pod nie zostanie uruchomiony na tym węźle.

Przykład

```
apiVersion: v1
kind: Pod
metadata:
  name: moj-pod
spec:
  containers:
  - name: moj-kontener
    image: nginx
  nodeSelector:
    disktype: ssd
```

Maintenance

1. **kubectl get events**
   - Polecenie `kubectl get events` umożliwia przeglądanie zdarzeń w klastrze, co jest przydatne do diagnostyki problemów i monitorowania różnych operacji w klastrze.
2. **kubectl logs**
   - Polecenie `kubectl logs` pozwala na przeglądanie logów kontenerów wewnątrz poda, co jest użyteczne do debugowania i analizy błędów w aplikacjach.
3. **kubectl top**
   - Polecenie `kubectl top` dostarcza informacje na temat zasobów zużywanych przez pody w klastrze, takie jak CPU i pamięć. Pomaga to w monitorowaniu wydajności klastra.
4. **kubectl drain**
   - Polecenie `kubectl drain` jest używane do przygotowania węzła do wyłączenia, przenosząc pody z tego węzła na inne węzły w klastrze.
5. **kubectl cordon**
   - Polecenie `kubectl uncordon` wyłącza dostępność węzła

6. **kubectl uncordon**
    - Polecenie `kubectl uncordon` przywraca dostępność węzła

Zadania

Zadanie 1: Instalacja dashboardu na klastrze

Wprowadzenie

Zarządzać i monitorować klaster można w bardzo różny sposób. Providerzy cloudowi dostarczają często swoje narzędzia. Mamy również do dyspozycji narzędzia konsolowe, dzisiaj jednak skupimy się na w miarę prostym graficznym narzędziu, które pozwoli nam przeglądać zasoby klastra. Zainstalujemy Kubernetes Dashboard za pomocą plików yaml. Dokumentacja znajduje się tutaj: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Kroki

1. Zainstaluj dashboard za pomocą komendy:

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.y
aml
```

2. Sprawdź, czy dashboard się uruchomił za pomocą komendy `kubectl get pods -n kubernetes-dashboard`
3. Rezultat powinien być następujący:

```
NAME                                         READY   STATUS    RESTARTS   AGE
dashboard-metrics-scraper-7bc864c59-rphjk    1/1     Running   0          100s
kubernetes-dashboard-6c7ccbcf87-lkx8s        1/1     Running   0          100s
```

4. Utwórz plik `dashboard-adminuser.yaml` z zawartością:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
```

```
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

5. Wrzuć pliki na klaster komendą `kubectl apply -f dashboard-adminuser.yaml`
6. Wygeneruj token, który będzie potrzebny do zalogowania `kubectl -n kubernetes-dashboard create token admin-user`
7. Powinieneś otrzymać token na kształt tego poniżej. Zapisz go na boku, przyda się za chwilę

Przykładowy token

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IlZCYXdWMmJUZ3ozVEpKbHpIRVdFeHppOUV3T2llVDIxbmZ5UWpvOVQxT2
8ifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwI
joxNjg1MTI0ODc5LCJpYXQiOjE2ODUxMjEyNzksImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN
2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib
2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiZDU2MTJjZDktNjM5NS0
0ZWFhLTgxY2UtZGRkM2M0ODgzNDljIn19LCJuYmYiOjE2ODUxMjEyNzksInN1YiI6InN5c3RlbTpzZXJ2aWNlY
WNjb3VudDprdWJlcm5ldGVzLWRhc2hib2FyZDphZG1pbi11c2VyIn0.d2Kab8lyfTb08cJS5wjoW_OEzE66UJV
HCcqgIghQ6lWLDOnO4MaudwskTNhMIpUorJaMwDy3ifH0OwTo2P6WZ3Y_oywdj8T-
EFws22Ot7sFvvXK1C1B8wTJYQnx4SviaqhixKnLoy1nWTMmUbtn2O7NPXB40_RvCt3ehoYzrleNUb3Cdp5nUfY
YWHoyExSblQYvBVK98tj7eozGOeVMuYjvSemJVCdEmjzAXdqr7uCanzspw7I6DQDJCfgG9Hl7D-
UdxiK9ZaiZjPt8xMLmmFkJ0FFNO9HuEx26HQe_1Dwim-yEcpRUmBbccEFZHioPCLW2eXElQxRX0xGHHE4OQIg
```

8. Uruchom proxy `kubectl proxy`
9. Wejdź na http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/
10. Zaloguj się za pomocą wcześniej wygenerowanego tokena
11. Obejrzyj zasoby swojego klastra

Zadanie 2: Utworzenie namespace

1. Utwórz imperatywnie namespace `szkolenie-imp`
2. Utwórz deklaratywnie namespace `szkolenie-dek`

Zadanie 3. Usunięcie namespace

1. Usuń namespace `szkolenie-imp` i `szkolenie-dek` w odpowiadający im sposób

Zadanie 4. Stworzenie poda

1. Utwórz namespace `szkolenie`
2. Utwórz imperatywnie poda z obrazu `nginx` o nazwie `nginx-imp` w powyższym namespace
3. Utwórz deklaratywnie poda z obrazu `nginx` o nazwie `nginx-dek` w powyższym namespace

Zadanie 5. Sprawdzenie poda

1. Wylistuj wszystkie pody w namespace `szkolenie`
2. Za pomocą komendy `kubectl describe` sprawdź pody `nginx-imp` i `nginx-dek`

Zadanie 6. Usunięcie poda

1. Usuń pody `nginx-imp` i `nginx-dek` w odpowiadający im sposób

Zadanie 7. Stworzenie deploymentu

1. Stwórz deployment z obrazu `nginx` o nazwie `nginx` w namespace `szkolenie`
2. Zrób 3 repliki

Zadanie 8. Sprawdzenie deploymentu

1. Wylistuj wszystkie deploymenty w namespace `szkolenie`
2. Wylistuj wszystkie pody w namespace `szkolenie`
3. Za pomocą komendy `kubectl describe` sprawdź deployment `nginx`

Zadanie 9. Skalowanie deploymentu

1. Zeskaluj deployment do 2 replik

Zadanie 10. Usunięcie deploymentu

1. Usuń któregoś z podów
2. Co się stało?
3. Usuń cały deployment

Zadanie 11. DaemonSet

1. Korzystając ze snippetu ze strony https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/
2. Utwórz DaemonSet dla `fluentd`
3. Powtórz kroki z Zadania 8. i 9. 10. dla DaemonSet
4. Czy któryś krok był niemożliwy?

Zadanie 12. StatefulSets

1. Utwórz StatefulSets z aplikacją `nginx`
2. Możesz skorzystać z pomocy z tej strony https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
3. Pamiętaj o utworzeniu service

4. Zrób 3 repliki
5. Powtórz kroki z Zadania 8. i 9. 10. dla StatefulSets

Zadanie 13. Init container

1. Stwórz poda, który będzie zawierać InitContainer
2. InitContainer ma wypisać na konsolę jakiś komunikat
3. Możesz skorzystać z pomocy tutaj: https://kubernetes.io/docs/concepts/workloads/pods/init-containers/
4. Sprawdź logi inicjalnego kontenera i głównego

Zadanie 14. Wiele kontenerów

1. Stwórz deployment, który będzie miał 2 kontenery, np. `nginx` i `busybox`
2. Sprawdź logi obu konternerów

Zadanie 15. Tworzenie joba

1. Stwórz joba, który wypisze coś na konsolę
2. Sprawdź logi joba
3. Sprawdź za pomocą `describe` co jest wewnątrz

Zadanie 16. Tworzenie CronJob

1. Stwórz takiego samego joba jak w Zadaniu 15.
2. Spraw, żeby uruchamiał się co 2 minuty

Zadanie 17. Service

1. Stwórz deployment z `nginx`
2. Stwórz Service, który będzie spinał pody utworzone przez Deployment

Zadanie 18. Utworzenie bazy danych

1. Na podstawie https://hub.docker.com/_/postgres utwórz Deployment z bazą danych
2. NIE twórz volumenu i sekertów

Zadanie 19. Utworzenie ServiceAccount

1. Stwórz ServiceAccount o nazwie `example-sa` w namespace `szkolenie-rbac`

Zadanie 20. Utworzenie roli

1. Stwórz rolę, która umożliwia listowanie namespace

Zadanie 21. Utworzenie roli binding

1. Stwórz role binding, który wiąże ServiceAccount i Rolę

Zadanie 22. Utworzenie tokenu i zalogowanie do dashboardu

1. Wygeneruj token dla tego ServiceAccount
2. Zaloguj się tokenem do dashboadu
3. Sprawdź, do czego masz dostęp

Zadanie 23. Utwórz użytkownika w K8s

TIP	Możesz do pomocy użyć tutoriala https://medium.com/@HoussemDellai/rbac-with-kubernetes-in-minikube-4deed658ea7b

1. Stwórz certyfikat dla użytkownika `-subj /CN=user1/O=group1`
2. Podpisz go `-CA ~/.minikube/ca.crt -CAkey ~/.minikube/ca.key`
3. Stwórz w pliku `kubeconfig` definicje użytkownika i contextu
4. Użyj stworzonego kontekstu
5. Spróbuj tym użytkownikiem stworzyć namespace

Zadanie 24. Nadanie uprawnień

1. Dodaj użytkownikowi uprawnienia do czytania podów i namespace
2. Spróbuj pobrać pody
3. Spróbuj stworzyć namespace

Zadanie 25. Tworzenie service typu NodePort

1. Stwórz deployment z `httpd`
2. Stwórz Service typu NodePort, który łączy pody z tym service
3. Sprawdź IP Nodów za pomocą `kubectl get nodes -o wide`
4. Sprawdź za pomocą `kubectl get svc` na jakim porcie wystawił się service
5. Za pomocą curla uderz do każdego node i sprawdź czy nginx odpowiada

Zadanie 26. Tworzenie Ingressu

1. Włącz obsługę ingressów za pomocą `minikube addons enable ingress`
2. Utwórz Ingress, który będzie wskazywał na Service z zadania 25
3. Zainstaluj na maszynie lokalnie nginx `apt install nginx`
4. Ustaw reverse proxy, w pliku `/etc/nginx/nginx.conf` dodaj

```
stream {
  server {
      listen 8888;
      #TCP traffic will be forwarded to the specified server
      proxy_pass IP_KLASTRA:NODE_PORT_INGRESSU;
  }
}
```

5. IP klastra możesz sprawdzić za pomocą `minikube ip`
6. Node port ingressu możesz sprawdzić za pomocą `kubectl get svc -n ingress-nginx`
7. Dodaj lokalnie w hosts (na swoim komputerze) wpis, który łączy domenę, której użyłeś z IP_MASZYNY
8. Sprawdź w przeglądarce wchodząc na IP_MASZYNY:8888, czy dostajesz odpowiednią odpowiedź

Zadanie 27. Utwórz Ingress dla Dashboardu

1. Utwórz Ingress dla dashboardu

Zadanie 28. Service ExternalName

1. Stwórz service typu ExternalName
2. Niech wskazuje na google.com
3. Utwórz dla niego Ingress
4. Sprawdź, czy działa

Zadanie 29. Service LoadBalancer

1. Stwórz service typu LoadBalancer
2. Co się stało?

Zadanie 30. Blokada ruchu za pomocą NetworkPolicy

1. Stwórz namespace `ns1`
2. Stwórz namespace `ns2`
3. W `ns1` stwórz bazę danych
4. W `ns2` stwórz instancję Hyperona
5. Połącz je ze sobą
6. Za pomocą NetworkPolicy ogranicz ruch pomiędzy namespaceami
7. Zrestartuj Hyperona
8. Sprawdź w logach czy faktycznie komunikacja została ucięta

Zadanie 31. Wystawienie Ingressa z Hyperonem

1. Usuń NetworkPolicy z poprzedniego zadania
2. Zrestartuj Hyperona
3. Utwórz Ingressa dla Hyperona
4. Sprawdź, czy działa

Zadanie 32. Przerobienie zmiennej środowiskowej z hasłem do Postgresa na Secret

1. Dodaj sekret, w którym będzie zapisane hasło do bazy danych
2. Dodaj obsługę sekretu w Postgresie, żeby wartość zmiennej środowiskowej była z niego brana

Zadanie 33. Użycie emptyDir

1. Skonfiguruj 2 kontenery `busybox`
2. Niech współdzielą wspólny katalog zdefiniowany jako `emptyDir`
3. Niech 1 kontener zapisze coś do tego katalogu
4. Sprawdź na drugim kontenerze czy coś jest w tym samym katalogu

Zadanie 34. Użycie hostPath

1. Skonfiguruj kontener z `nginx`
2. Podmountuj katalog `/etc` z hosta do `/etc-host`
3. Sprawdź, czy są pliki

Zadanie 35. Instalacja NFS CSI

1. Zainstaluj NFS CSI ze strony https://github.com/kubernetes-csi/csi-driver-nfs/blob/master/docs/install-csi-driver-v4.5.0.md
2. Dodaj storageclass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-csi
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: nfs.csi.k8s.io
parameters:
  server: 10.135.156.148
  share: /k8s
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:
```

```
  - nfsvers=4
```

Zadanie 36. Wolumen dla bazy danych

1. Dodaj PVC
2. Zmodyfikuj Deployment z bazą danych, żeby obsługiwał PVC
3. Zrób 2 repliki

Zadanie 37. Baza danych jako statefulset

1. Przerób bazę danych na StatefulSet