Workshop Docker -> K8s

Docker images for a K8s environment

Brief introduction

- Marco Verleun
    - Older than the internet
    - Even older than Unix… ;-)
- Engineer at SUE
- Currently working as DevOps engineer for a government agency:
    - Gitlab
    - Kubernetes clusters
    - Many legacy applications

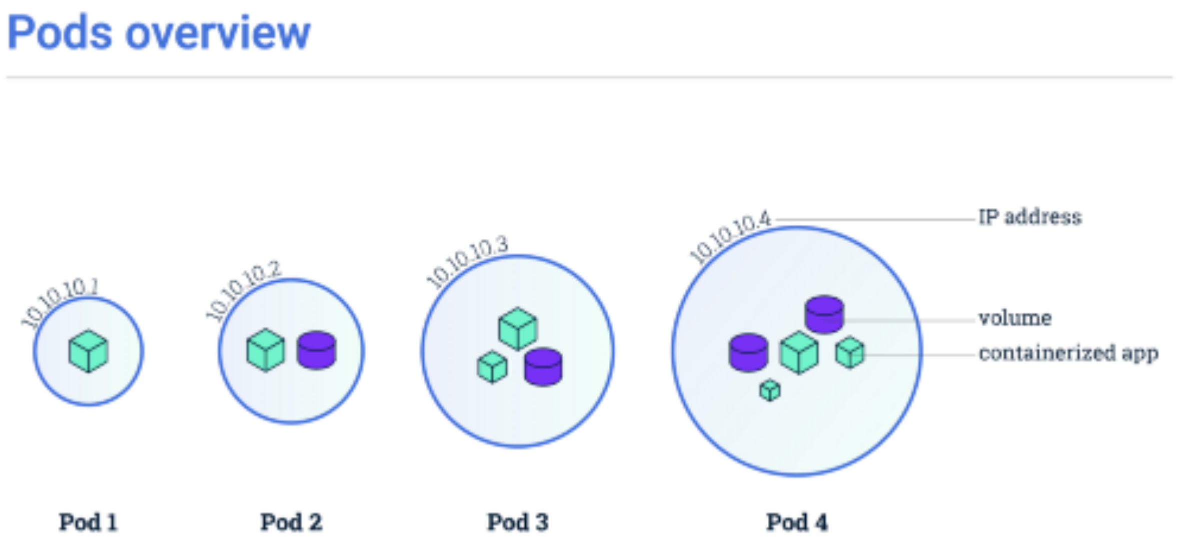
Topics

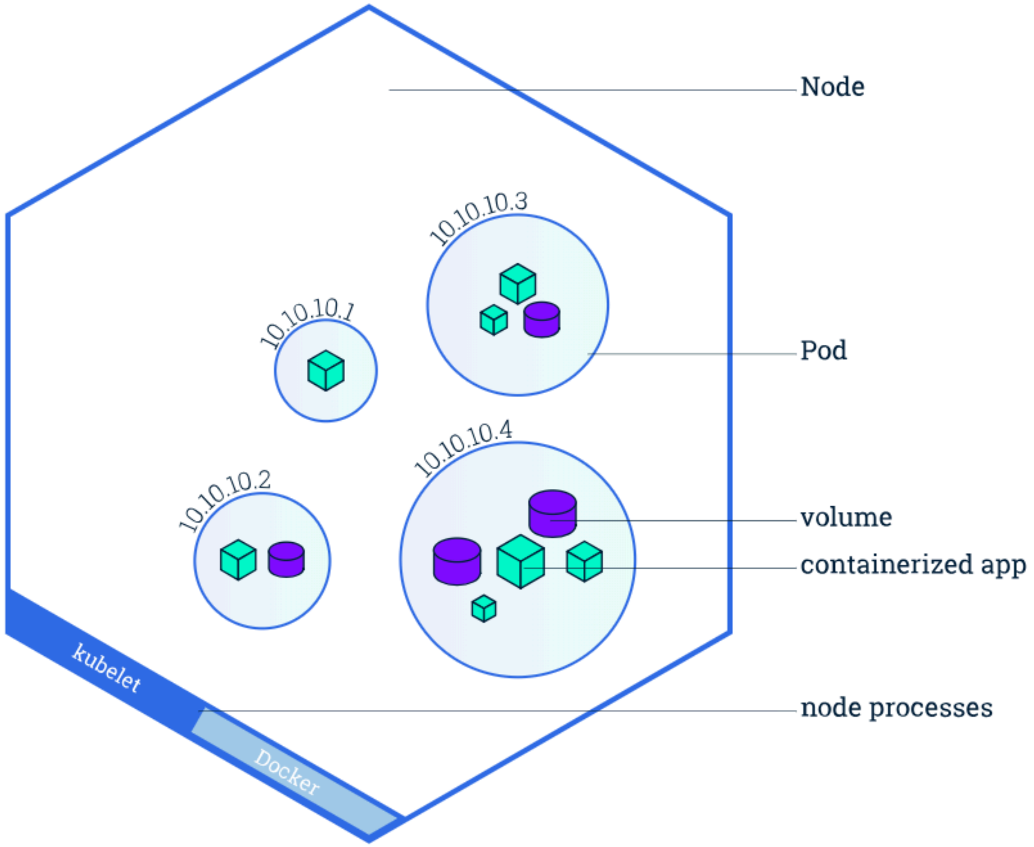
- Disclaimer
- Basics of a Pod and nodes
- Basics of ingress traffic, services and pods.
- K8s upgrade strategies and scaling
- Basics of a container
- Best practices according to Google
- Process(es) inside a container
- Logging via stdout
- User ID’s and non-root images
- http(s) redirects

Disclaimer

- In this workshop we will use older ‘official’ images of Nextcloud.
  In no way this workshop is ment to embarrass the creators of these images.
  Many official images are comparable and the objective of this workshop is to illustrate how ‘official’ images not always reflect the best practices for K8s images.
- We will see certain aspects that could affect the functioning of a K8s cluster. I’m pretty sure that the rules are not 100% complete and are not always avoidable. Use your own judgement.

Basics of a Pod and Node





Images from: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/

Ingress and services to connect to Pods

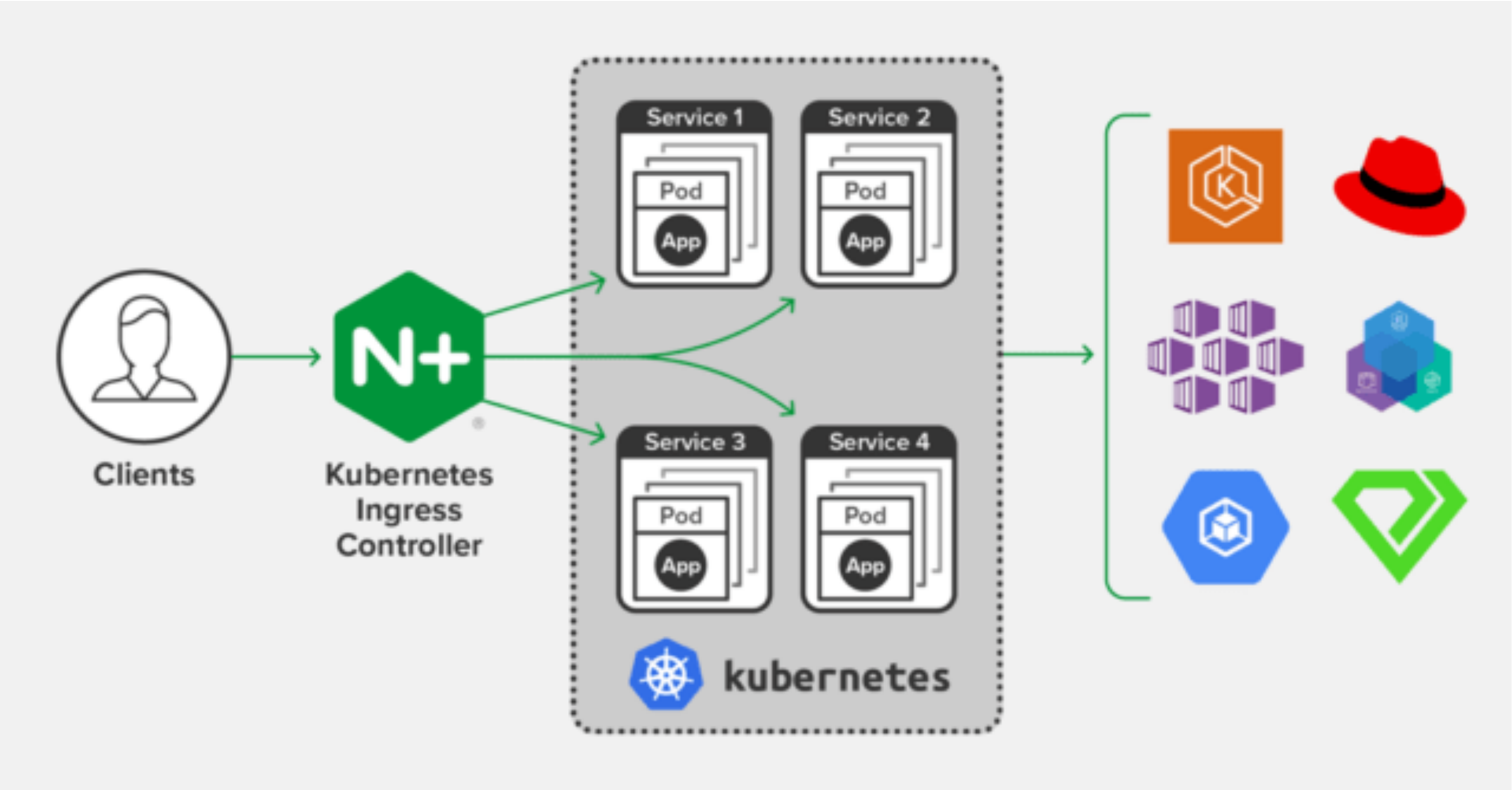
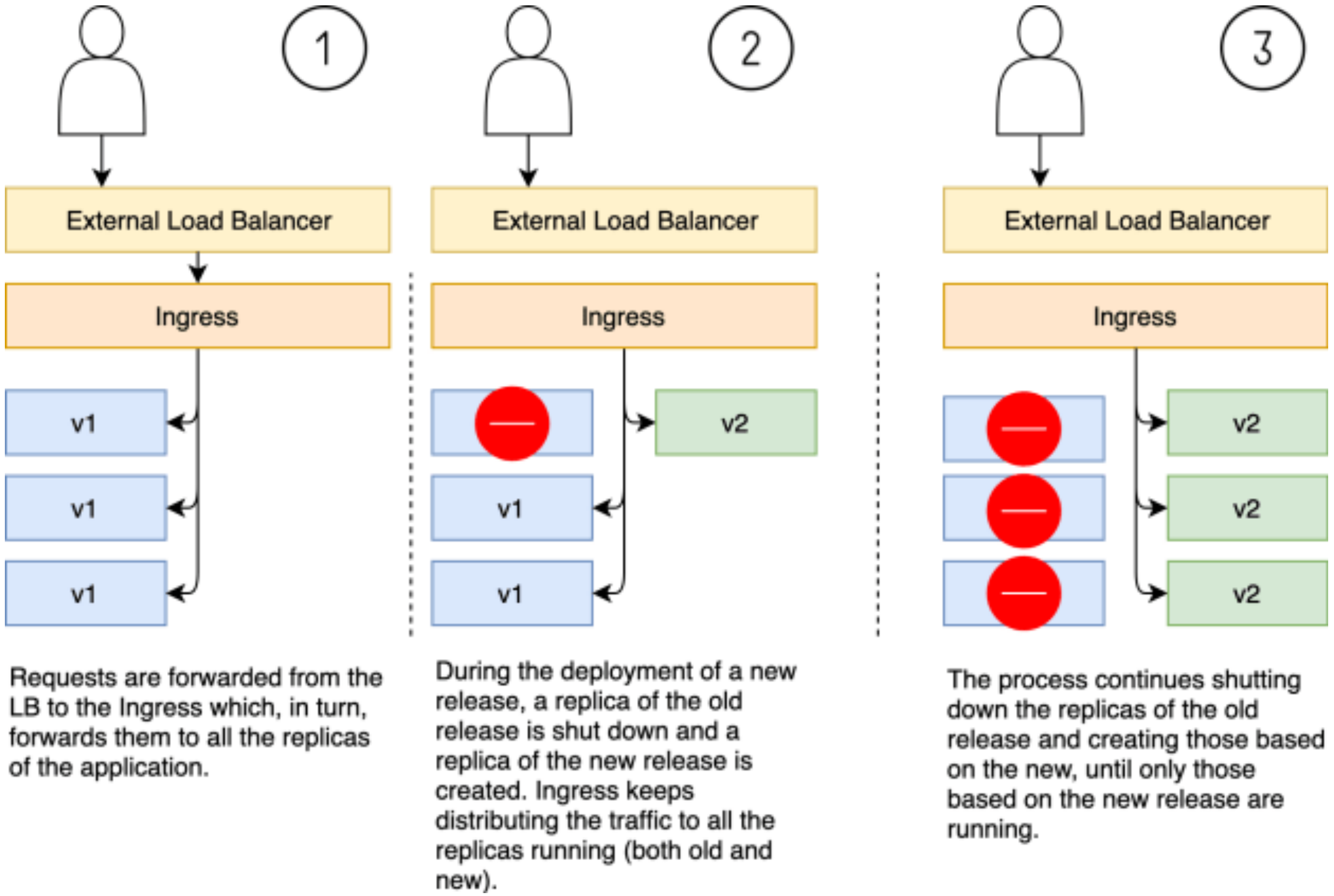


Image from: https://www.nginx.com/products/nginx/kubernetes-ingress-controller/

K8s upgrade strategies and scaling



Requests are forwarded from the LB to the Ingress which, in turn, forwards them to all the replicas of the application.

During the deployment of a new release, a replica of the old release is shut down and a replica of the new release is created. Ingress keeps distributing the traffic to all the replicas running (both old and new).

The process continues shutting down the replicas of the old release and creating those based on the new, until only those based on the new release are running.

Image from: https://technologyconversations.com/2019/09/05/exploring-deployment-strategies-in-kubernetes/

Best practices (according to Google)

- https://cloud.google.com/solutions/best-practices-for-building-containers and https://www.weave.works/blog/kubernetes-best-practices
- A few examples:
  - Use a non-root user inside the container
  - Make the file system read-only
  - One process per container
  - Don’t restart on failure. Crash cleanly instead
  - Logging via stdout and stderr
  - Readiness and Liveness Probes are your friend, no docker health

Docker images

- Many well known products provide ‘official’ images.
- These images do not always adhere to the best practices.
- Sometimes it’s really challenging to properly deploy images in a K8s environment.
- Don’t simply do it because you can, take some time to properly investigate the pro’s and con’s of running an app in K8s.
- Especially legacy applications can be challenging (We deployed pods that can take up to two hours to startup and that require 6 Gb of RAM each…)

Dockerfile

- Let’s have a look at a Dockerfile from an ‘official’ image.
- It is taken from: https://github.com/nextcloud/docker
- The full version is found here: https://github.com/nextcloud/docker/blob/master/17.0/apache/Dockerfile, we’ll explore snippets

Dockerfile first lines

```
FROM php:7.3-apache-buster

# entrypoint.sh and cron.sh dependencies
RUN set -ex; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        rsync \
        bzip2 \
        busybox-static \
    ; \
    rm -rf /var/lib/apt/lists/*; \
    \
    mkdir -p /var/spool/cron/crontabs; \
    echo '*/5 * * * * php -f /var/www/html/cron.php' > /var/spool/cron/crontabs/www-data
```

Dockerfile first lines

```
FROM php:7.3-apache-buster

# entrypoint.sh and cron.sh dependencies
RUN set -ex; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        rsync \
        bzip2 \
        busybox-static \
    ; \
    rm -rf /var/lib/apt/lists/*; \
    \
    mkdir -p /var/spool/cron/crontabs; \
    echo '*/5 * * * * php -f /var/www/html/cron.php' > /var/spool/cron/crontabs/www-data
```

Dockerfile snippets

```
RUN set -ex; \
    \
    curl -fsSL -o nextcloud.tar.bz2 \
        "https://download.nextcloud.com/server/releases/nextcloud-${NEXTCLOUD_VERSION}.tar.bz2"; \
    tar -xjf nextcloud.tar.bz2 -C /usr/src/; \
    mkdir -p /usr/src/nextcloud/data; \
    mkdir -p /usr/src/nextcloud/custom_apps; \

COPY *.sh upgrade.exclude /
COPY config/* /usr/src/nextcloud/config/
```

Dockerfile ENTRYPOINT and CMD

```
ENTRYPOINT ["/entrypoint.sh"]
CMD [“apache2-foreground”]
```

- The script entrypoint.sh is quite extensive and is worth reading online: https://github.com/nextcloud/docker/blob/master/17.0/apache/entrypoint.sh
- In short, it installs from /usr/src to /var/www/html during the first time a container is started. It checks versions to see if an upgrade is necessary.
- The last line is great. It replaces the process of the current PID (1) with the content of the CMD line, in this case apache will run as process 1 by default.
- See also: https://success.docker.com/article/use-a-script-to-initialize-stateful-container-data

```
exec "$@"
```

Download demo files from GitHub

1. Clone the repo: git clone https://github.com/mverleun/Meetup-Docker-K8s-Images.git

2. Download as zip: https://github.com/mverleun/Meetup-Docker-K8s-Images/archive/master.zip

Explore behaviour with docker-compose example.

1. `cd Demo-1`
2. `cat docker-compose.yml`
3. `docker-compose up db`
4. `docker-compose up nc-12`
5. Open a browser for http://127.0.0.1:8012 and configure Nextcloud
6. `docker-compose up nc-13`
7. Open a browser for http://127.0.0.1:8013 and watch the logging
8. Refresh the browser in the Nextcloud 12 session. What is happening? And more important why? What are the implications?
9. Cleanup with `docker-compose down -v`

   Note: Take it slow and experience everything. It’s not about the destination but also the journey.

non-root containers

- Are best practices and strongly recommended
- Could require a bit of rebuilding using a Dockerfile
- Let’s explore it step by step
    - `docker run --rm -it --name nc-13 nextcloud:13.0.1`
    - `docker exec -it nc-13 bash`
    - Execute the following commands: `id; ss -antp`
    - It shows that you’ve got root permissions and that apache is listening on port 80
    - Now try: `apt update`
- Ohhh… That’s scary…
- Stop the container by pressing Ctrl-C

non-root containers attempt 1

- This **should** fail and it **will**, so don’t try it in production. ;-)
- Start as a user with UID 33:

  -
    ```
    docker run --rm -it --name nc-13 --user 33 nextcloud:13.0.1
    ```
  - Notice that the container will not start for obvious reasons.
  - Port 80 is reserved for root

non-root containers Dockerfile

- Check the content of Demo-2 directory
- Dockerfile:

```
FROM nextcloud:13.0.1
USER root
COPY root /
USER 33
```

- root/etc/apache2/ports.conf

```
Listen 8080
<IfModule ssl_module>
        Listen 8443
</IfModule>
<IfModule mod_gnutls.c>
        Listen 843
</IfModule>
```

Build image and test

- Once more the same commands
  - `docker build -t nextcloud:13.0.1-non-root .`
  - `docker run --rm -it --name nc-13 nextcloud:13.0.1-non-root`
  - `docker exec -it nc-13 bash`
  - Execute the following commands: `id; ss -antp`
  - Now try: `apt update`
- Ohhh… That's nice.
- Note when you start with the proper port mapping you can do the same as before.

Logging via stdout and stderr

- Let’s extend the Dockerfile slightly, see files in Demo-3

```
FROM nextcloud:13.0.1-non-root
USER root
COPY root /
USER 33
CMD ["/init-script.sh", “apache2-foreground"]
```

Logging via stdout and stderr

- The script contains:

```
#!/bin/bash
function stdout() {
    echo "${1}" > /proc/1/fd/1
}
function stderr() {
    echo "${1}" > /proc/1/fd/2
}
echo This is a init script with a long delay.
for i in {1..30}; do
    stdout "${i}"
    sleep 1
done
stdout "We could have done something usefull as well"
exec "$@"
```

Logging via stdout and stderr

- Build a new image and run it:
  - `docker build -t nextcloud:13.0.1-non-root-logging .`
  - `docker run --rm -it --name nc-13 nextcloud:13.0.1-non-root-logging`
- For proc #1 not always necessary, for other processes a nice way to write to stdout and stderr.
- File permissions of `/proc/1/fd/*` apply!

Readiness and liveness probes and more

- Instead of using the Docker health check K8s uses several probes.
  - Liveness probe performs checks to decide if a container restart is required
  - Readiness probe is used to configure the ingress controller
  - Other probes are available as well
- These probes are often http requests to specific webpages (/healthz etc)
- Shell scripts can be used as well. Either add scripts using configmaps or add them to an image.
- This is one of the reasons that Docker health checks are discouraged.
- Check the files in Demo-4

Downgrading Nextcloud? And what happend?

- Back to Demo-1, repeat the steps but then stop Nextcloud version 12 and start it again…
- Outch…. No canary deployments or rolling updates….
  This would require a green/blue deployment with two databases etc. if a fallback is required…
  But even then, the volumes have to be separated as well…

  Bottom line: Nextcloud installs php code during the first run and configures the database.
  The php code is installed in a shared volume, otherwise collaboration between containers will fail…
  When upgrading Nextcloud the code in de the volume is updated and Nextcloud 12 is serving Nextcloud 13 code, but still has Nextcloud 12 vulnerabilities…
- Please put version dependent code (PHP, Python etc) in the image during a docker build if possible.

Resource limits

- Is one of the best practices for K8s.
  - If you hit a CPU limit things will slow down.
  - If you hit a memory limit **things** will crash.
- No limit settings means that the node determines the limits…
- Java requires tuning with the Xmx and Xms parameters, you have to make sure that the amount of memory allowed is greater than the the Xmx value + the overhead from other processes…
- `docker stats, docker run -m … -c …, docker update`

Pudding time

- What about these best practices?:
  - One process per container
  - Don’t restart on failure. Crash cleanly instead

```
docker stats
docker run -m 64M  - -name nc-13 --rm -it -p 8080:80 nextcloud:13.0.4
docker exec -it nc-13 top
siege -c 10 -r 10 http://127.0.0.1:8080
siege -c 100 -r 10 http://127.0.0.1:8080
```

- Please explain…
- Siege: https://www.joedog.org/siege-home/.
  `(brew install siege, apt-get install -y siege)`

This one is easier (I think)

-
  ```
  docker stats
  ```

-
  ```
  docker run -m 512M --rm -it --name demo  busybox
         tail -f /dev/zero
  ```

Versus:

-
  ```
  docker run -m 512M --rm -it --name demo  busybox tail -f /dev/zero
  ```

- Where is the crash?