
KubeCon
CloudNativeCon
Europe 2021
Virtual
Forward Together »

Lessons Learned Deploying Traditional Web Applications on Top of Kubernetes

Marcos Bjoerkelund (He/Him)

KubeCon | CloudNativeCon
Europe 2021
Virtual

Agenda

Introduction

General Workflow

From Build to Deploy

Tips and Good Practices

To Create Better Container Images

Challenges You May Find

And Ways to Solve Them

An Example

Deploying WordPress on Kubernetes

Introduction

Bitnami is a Catalog of Free Open Source Software

10+ years building and maintaining software packages

Over 180 applications, runtimes, frameworks and more, including:

Any Environment

Local

Cloud

Data Center

Any Format

Virtual Machines

Containers

Deployment Templates

Any Platform

kubernetes vmware aws

Azure Google IBM Cloud

Traditional Web Applications

Traditional Web Applications

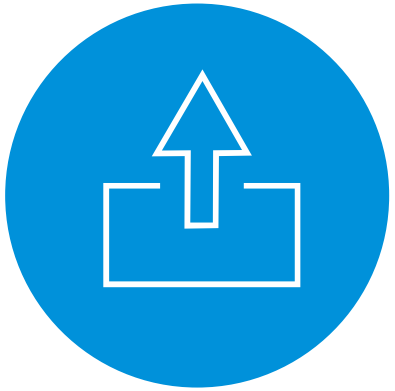
Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data, lack of external input methods (i.e. env vars)



Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Traditional Web Applications

Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data, lack of external input methods (i.e. env vars)

Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Traditional Web Applications

Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data and config, lack of external input methods (i.e. env vars)

Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Traditional Web Applications

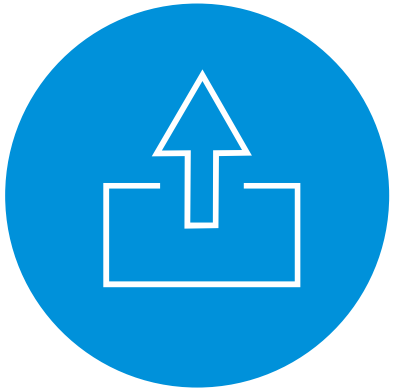
Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data, lack of external input methods (i.e. env vars)



Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Traditional Web Applications

Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data, lack of external input methods (i.e. env vars)

Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Traditional Web Applications

Do not follow key requirements for cloud-native apps (based on https://12factor.net):

State & Processes

Processes are stateless and share-nothing

Config mgmt.

No separation with code/data, lack of external input methods (i.e. env vars)

Disposability

Can be started/stopped at any moment, with minimal startup time

Concurrency

Application must be able to scale horizontally

Dependencies

Reproducible environments

Examples

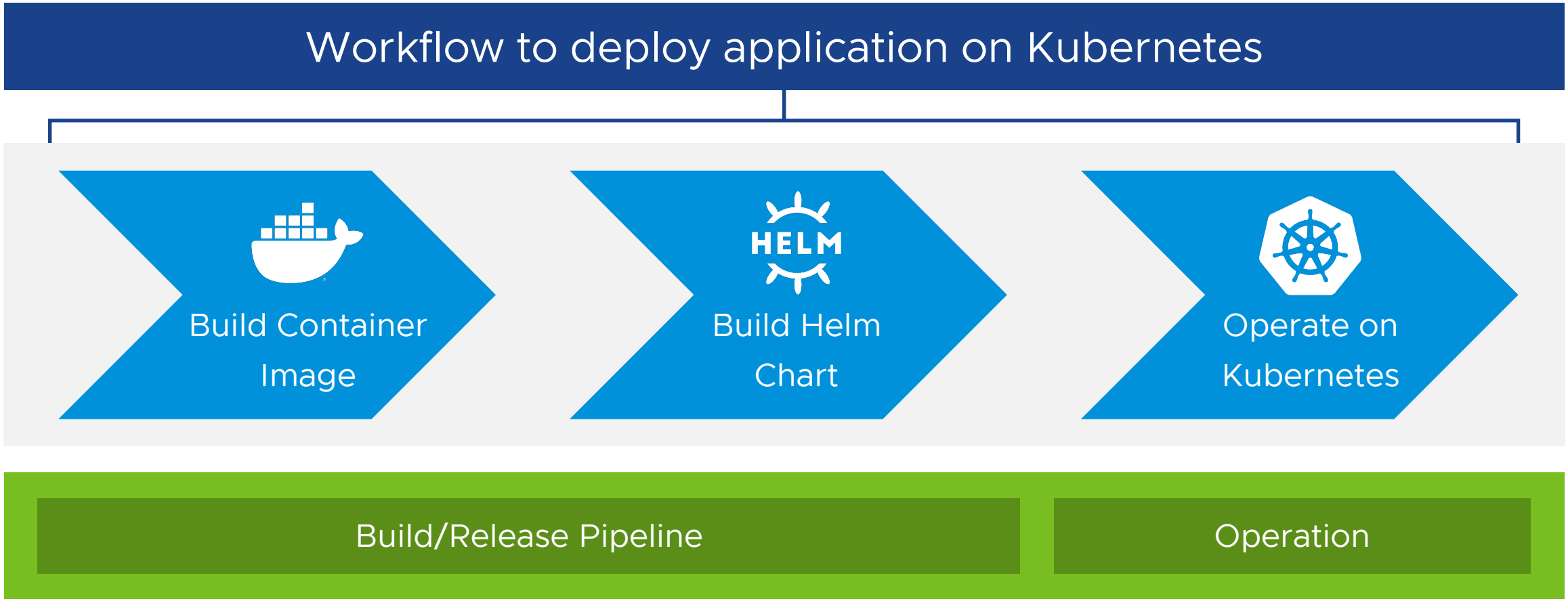
github.com/bitnami-labs/deploy-web-apps-on-kubernetes

General Workflow

From Build to Deploy

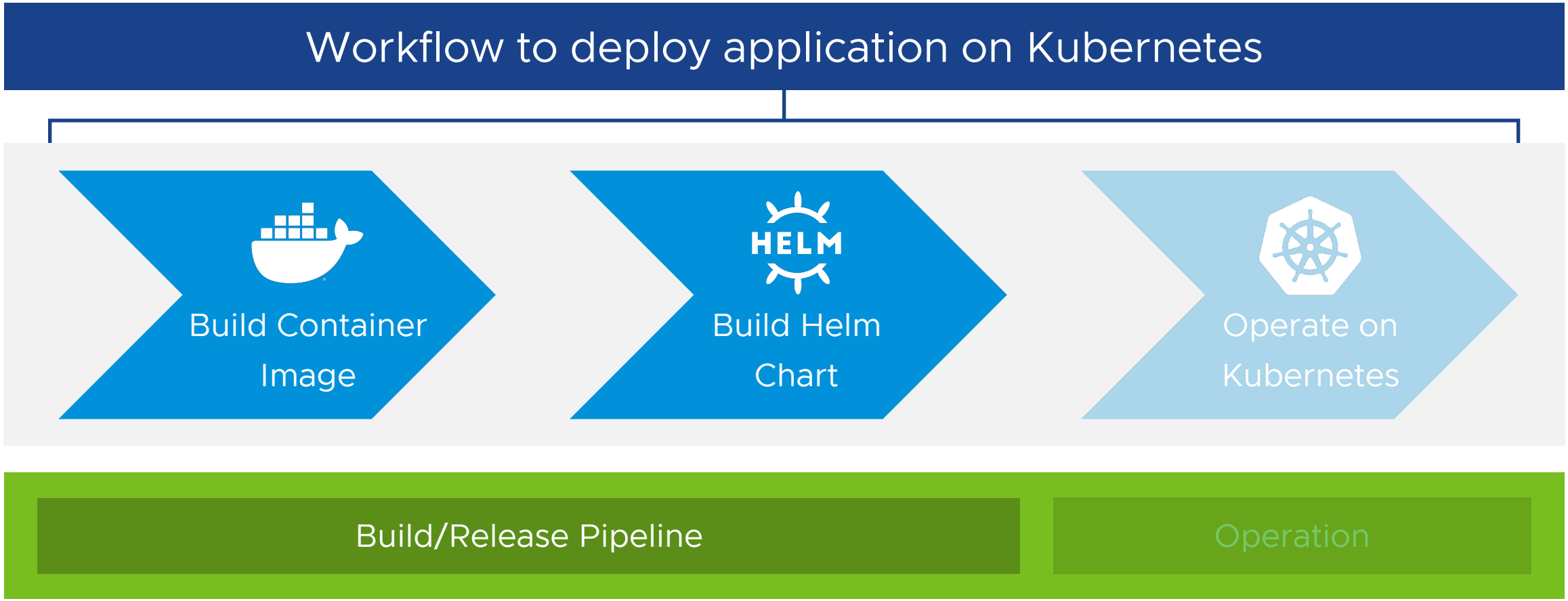
General Workflow for a Traditional Web Application

What processes and steps are required to deploy on Kubernetes



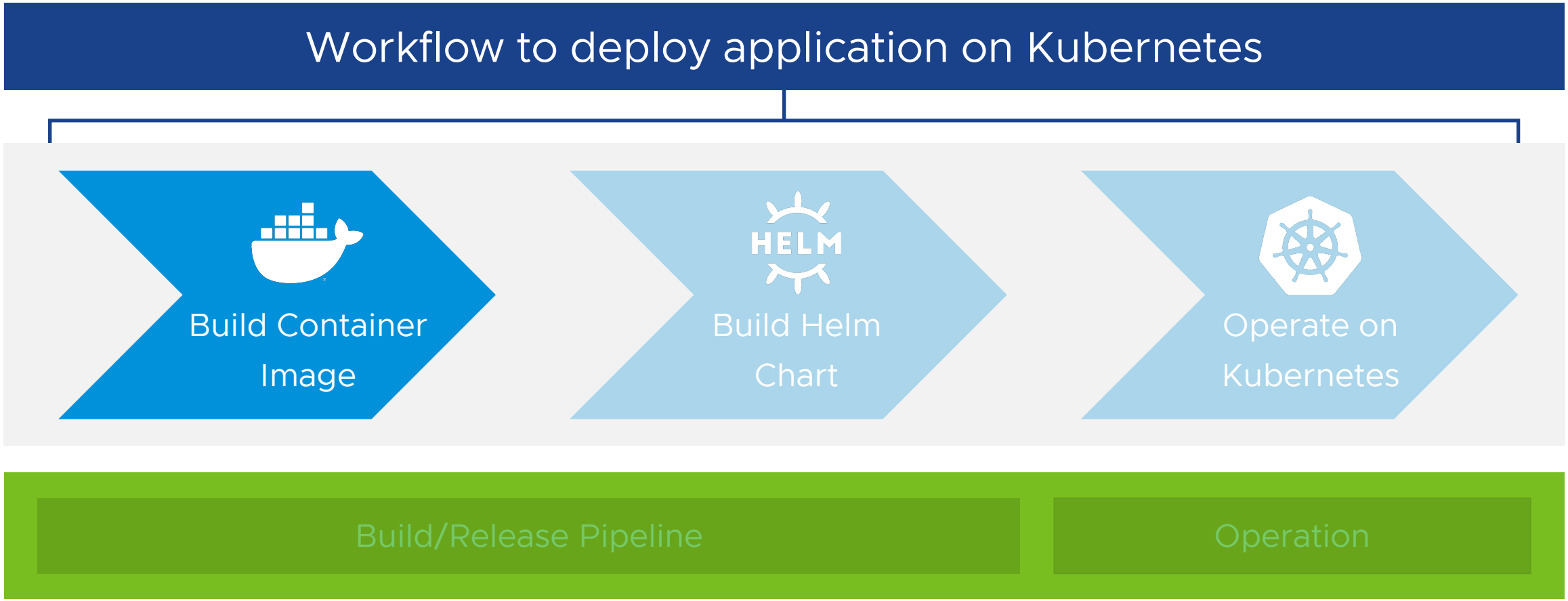
General Workflow for a Traditional Web Application

What processes and steps are required to deploy on Kubernetes



General Workflow for a Traditional Web Application

What processes and steps are required to deploy on Kubernetes



Build Container Image

Container images overview

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/1-container-image

- OCI compliant images (Docker, Podman)
- Contexts: Build-time & runtime
- Build-time (Dockerfile):
  - List of steps to build/run an image
  - Based on commands and layers
  - Limited input support (ARG)
- Runtime (entrypoint)
  - Logic to install and run the application
  - Inputs via env. vars (or mounted files)

Example: Dockerfile

```
FROM debian:buster

RUN apt-get update && \
    apt-get install -y python

COPY html /app

WORKDIR /app
EXPOSE 8080
USER www-data
CMD python -m SimpleHTTPServer 8080
```

Base image

Runtime deps

Install app

Runtime config

Example: Build image from Dockerfile

```
$ docker build -t image:tag .
```

Build Container Image

Dockerfile design

Must-have

Install dependencies and system package

Copy and prepare application files

Establish image security

Good to have

Focus on minimal image size

Design with runtime stability in mind, e.g.:

- CPU- and memory-intensive steps at build
- Pre-download files from the Internet

Runtime configuration defaults

Build Container Image

Entrypoint design

Must-have

Focus on minimal deployment time

Validation of inputs (env. vars / filemounts)

Persistence of application data

Start web application

Good to have

Configuration logic (dependent on inputs)

Wait for external services

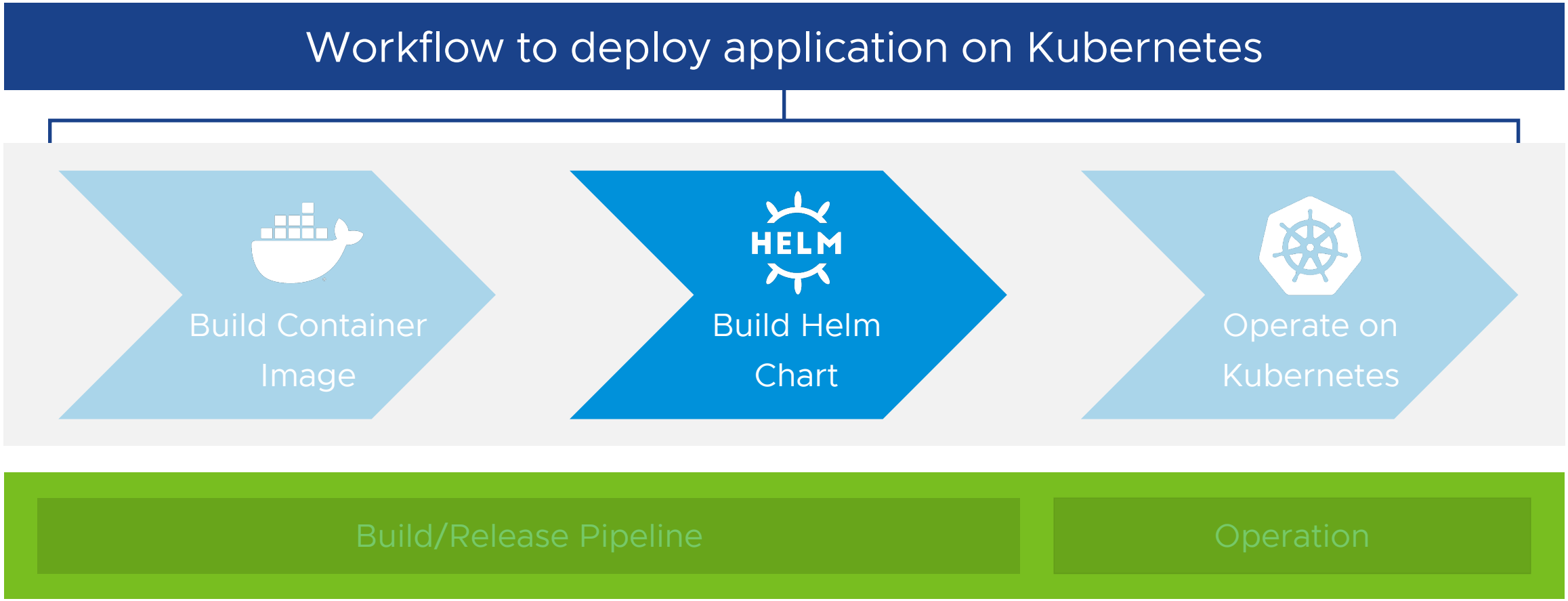
- Avoid potential app errors and warnings

Pre-/post-installation features, e.g.:

- Further extend functionality of the app
- Automatic installation in the database

General Workflow for a Traditional Web Application

What processes and steps are required to deploy on Kubernetes



Build Helm Chart

Helm charts overview

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/2-helm-chart

Introduction to Helm:

- Most popular for deploying web apps on K8s
- Defines how the application should operate
- Inputs via values.yaml or CLI (--set) 1
- Template of Kubernetes resources in YAML 2
- Sub-charts? 3

To deploy a Helm chart on Kubernetes:

```
$ helm install my-release myrepository/wordpress \
  --set option1=value1[,option2=value2]
```

Example: File tree of a chart

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml 1
  values.schema.json
  templates/ 2
  templates/NOTES.txt
  charts/ 3
  crds/
```

Build Helm Chart

Helm chart design

Essentials

Deployment of for app containers

Ingress with TLS/SSL

Persistent volume claims with app data

Secrets with credentials

...

Good to have

Database or external service,
either external or sub-chart

Metrics support (service, container, ...)

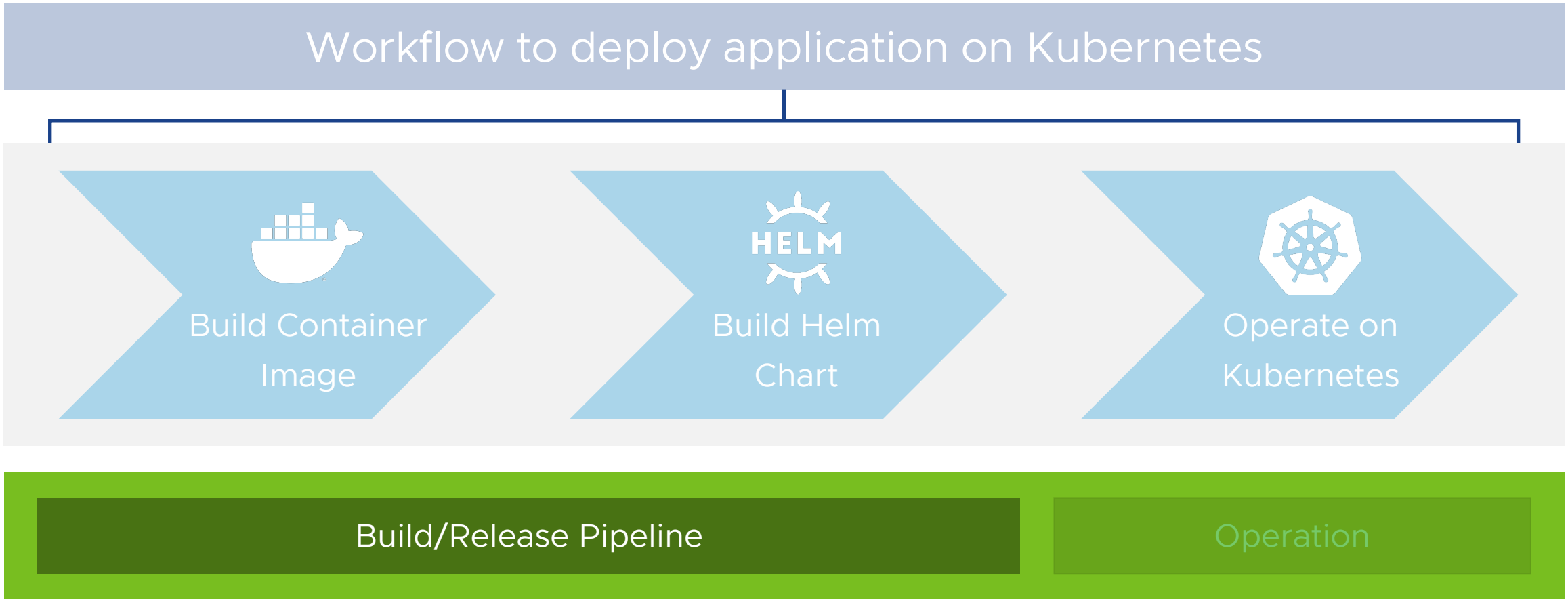
Cron jobs for temporary jobs

Custom resources, init scripts...

...

General Workflow for a Traditional Web Application

What processes and steps are required to deploy on Kubernetes



Build and Release Pipeline

What steps to perform on a build and release pipeline

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/3-pipeline

Basic steps:

- Build container image ("docker build")
- Build Helm chart ("helm package")
- Testing container image & chart
- Release: Publish images and charts

Multiple free (*) and simple CI/CD options available:

- GitHub Actions
- Travis CI
- Jenkins / Jenkins X
- GitLab CI
- CircleCI
- And many more

Bitnami's pipeline:

See: "App Testing at Scale: How Bitnami Tests Thousands of Releases Per Month" by Juan Jose Martos Castro

(KubeCon/CloudNativeCon North America 2020)

(*): There may be limits to the free tier, except from self-hosted options (like Jenkins or GitLab CE).

Tips and Good Practices

To Create Better Container Images and Helm Charts

Security

Non-root users

Non-root containers protect from **privilege escalations** and **access to non-writable files**

Existing non-root username (known UID)

- Incompatible with platforms that require non-root or arbitrary UID (i.e. OpenShift)
- Writable files: Explicit file/group ownership

Existing non-root username (arbitrary UID)

- Works in all environments
- Writable files: Explicit group ownership
- **Insecure:** Requires writable /etc/password

Link to OpenShift docs

Arbitrary UID (no username)

- Works in all environments
- Incompatible with some apps (workaround: `nss_wrapper`)
- Ugly prompts ("I have no name!")
- Writable files: `g+rwX` permissions (root owner)

Security

Non-root users

Non-root containers protect from **privilege escalations** and **access to non-writable files**

Existing non-root username (known UID)

- Incompatible with platforms that require non-root or arbitrary UID (i.e. OpenShift)
- Writable files: Explicit file/group ownership

Existing non-root username (arbitrary UID)

- Works in all environments
- Writable files: Explicit group ownership
- **Insecure:** Requires writable /etc/password

Link to OpenShift docs

Arbitrary UID (no username)

- Works in all environments
- Incompatible with some apps (workaround: `nss_wrapper`)
- Ugly prompts ("I have no name!")
- Writable files: `g+rwX` permissions (root owner)

Security

Permissions

- Permissions should be set at build time, not at runtime:
  - **Limited** ability to set permissions at runtime by non-root users
  - **Slow**, affecting deployment time (at build time we have all the time in the world)
- Limit list of writable files to the bare minimum (i.e. only data directories).
- Advanced topics: Docker Security Cheat Sheet (OWASP)
  - **Read-only file systems**
  - Use Linux Security Modules: AppArmor, seccomp, SELinux
  - …

Optimize Container Image Size

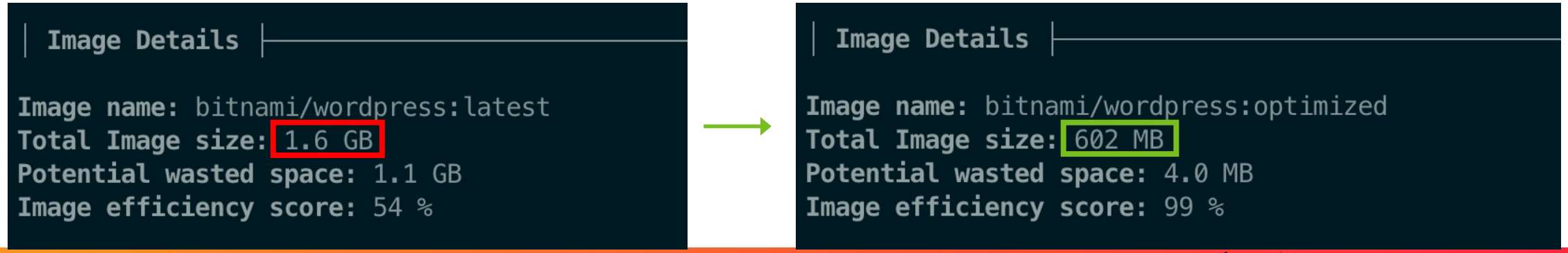
How to reduce your container image size

The **layer format** of containers images may have a **huge impact in their size**

Motivation: **Reduce bandwidth usage and deployment time** (*).

- Multi-stage image (especially for apps w/ build-time deps or must be compiled)
- Strategic Dockerfile design: Group commands + Remove unused files in each step + Location
- Docker experimental feature: `--squash` – Beware of its limitations

Useful tool: wagoodman/dive



Good practices

Create container images following good practices guidelines

- Best Practices Writing a Dockerfile
- 1 process per container!
- Run heavy and time-consuming tasks needed to bootstrap the application at build-time
  - E.g. build assets, install npm dependencies
- Reproducibility: Beware of network issues, distro package manager changes, etc.
- Maintenance: Periodic re-build of the image
  - Minimize response time in case of a security incident of an application dependency
  - Detect new build-time issues early before a re-build is urgent
- Use simple tools for initializing the app (e.g. Bash): Visibility, easy to debug, little memory usage
- Other: Multi-arch, distro election depending on your needs…

Good practices

Create Helm charts following good practices guidelines

- Best Practices for Creating Production-Ready Helm Charts
- Bitnami Helm chart template following good practices (charts/template)
- Deploy non-root containers by default.
- Extensibility and flexibility. For example:
  - Extend the app configuration.
  - Components and features to enable/disable via chart inputs.
  - Custom init containers, volumes, environment variables, annotations, resource limits, etc.
  - Custom additional Kubernetes resources to deploy.
- Integrate with logging and monitoring platforms.
- Diagnostics: Make it easier to debug deployments in case of a failure.

Challenges You May Find

And Ways to Solve Them

Persistence of Application Data

What application files to persist between runs

Only persist data files that must be writable at runtime. Separation of data & code.

App separates code and data

Persist certain application files (data)

Not supported for all applications

Faster as less data must be populated.

Automatic app update process support

No separation of code and data

Persist all application files

Supported for all applications

Slower as it requires volume population.

Limited maintenance & upgrade support

Application Updates

How to implement support for automatic application updates

App must separate code and data, or provide a UI interface for upgrading.

Automatic updates support

Backup DB before upgrading! Scale down!

App separates code and data

- Update version: Swap image version (tag).

App provides web UI update tool

- Force insecure permissions.
- No rollback in case of errors.

Requires manual updates

May be your only choice:

- Unfeasible to automate upgrade script.
- Code and data separation is not supported.

Multiple Application Services

How to support running multiple application services

Avoid running > 1 service per container

- Problem: Each service require shared data with the app (i.e. workers):
  - Does the app support external services for cache/data? (i.e. Redis)
  - Alternative: Synchronized / cached FS (i.e. NFS) if data needs to be available for all services
- Problem: Frontend and backend servers running the same code:
  - CDN for storing non-static assets (frontend), route all requests to backend
  - Shared / cached file systems (see above)
  - Varnish: Serve everything through backend, cache repeatable requests to reduce app load
- Other cases:
  - Periodic jobs (i.e. Crontab) to use Kubernetes `CronJob` resource with app volumes.
  - Init containers for pre-install steps (i.e. further customize your application), aside from container logic.

Horizontal Scaling

How to design a Helm chart to support scaling your application up/down

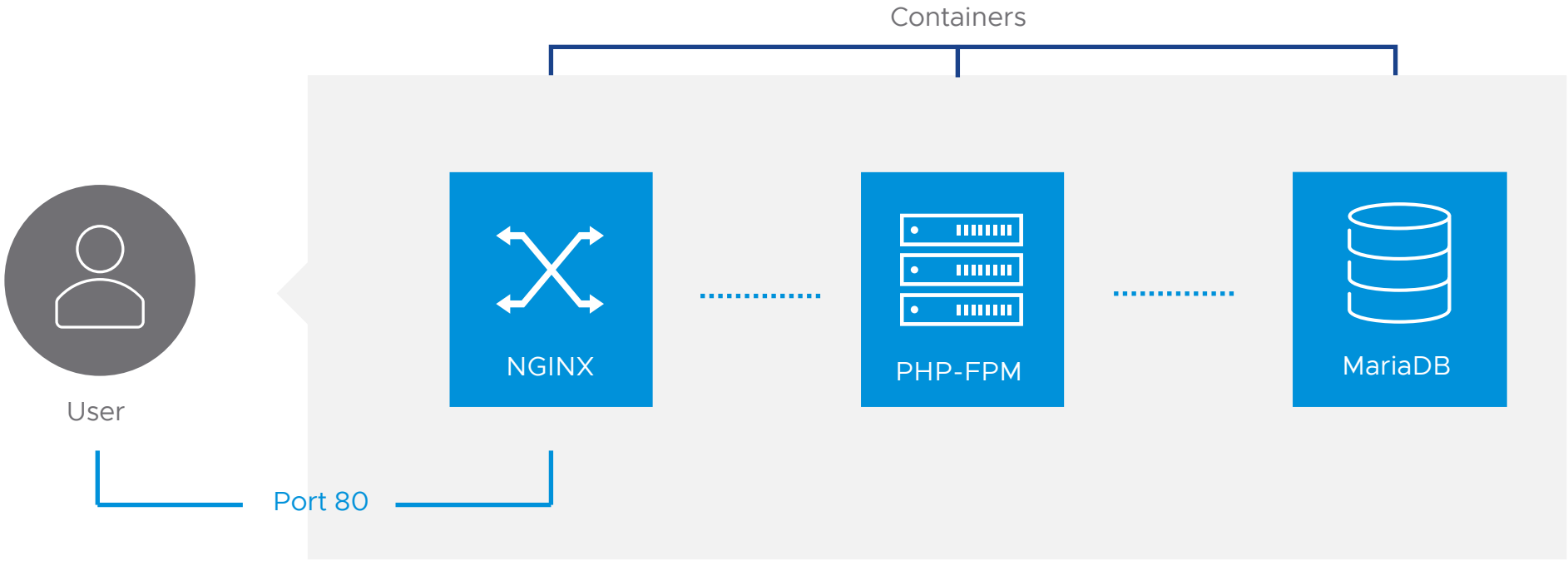
- Ideally: Rely on external services, no local data. For example:
  - Web server to run the application scripts.
  - Server for temporary files (i.e. Redis or Memcached): cache, sessions.
  - CDN: host assets, user uploads.
  - External load balancer as main entrypoint for the application. Headless services.
- Reality:
  - Limited scale support but require local data: Workaround via synchronized FS (previous slide).
  - May be impossible: Cache server (Varnish). Optimize application configuration. Scale vertically.

An Example

Deploying WordPress on Kubernetes

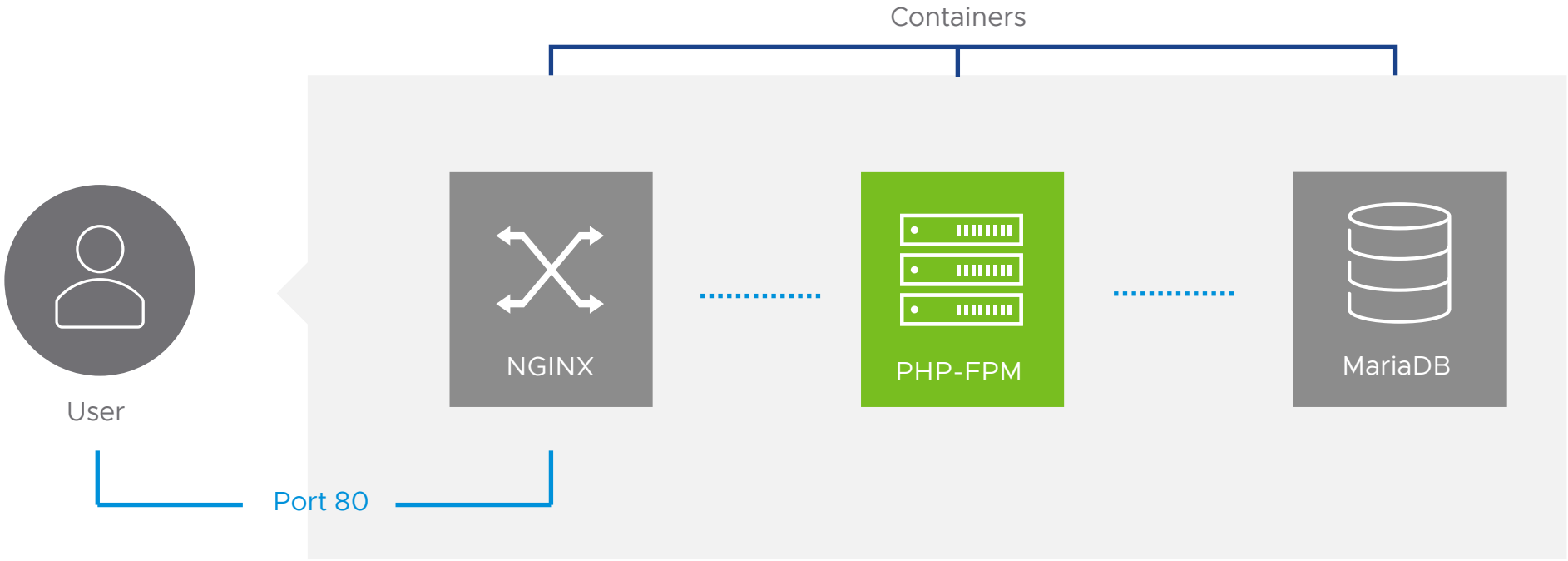
Service Diagram

How the different services will interact with each other



WordPress Deployment Service Diagram

How the different services will interact with each other



WordPress Dockerfile

How the Dockerfile will be structured

```
FROM bitnami/php-fpm:8.0 AS builder
ARG WORDPRESS_VERSION=5.7
RUN apt-get update && apt-get install -y ca-certificates curl
RUN curl -s https://wordpress.org/wordpress-${WORDPRESS_VERSION}.tar.gz \
  | tar xz -C / --no-same-owner --no-same-permissions
# RUN curl http://.../plugin.zip | tar xz -C /wordpress/wp-content

FROM bitnami/php-fpm:8.0
RUN sed -Ei 's/;(clear_env)/\1/' /opt/bitnami/php/etc/php-fpm.d/www.conf
RUN apt-get update && apt-get install -y --no-install-recommends vim \
  && rm -r /var/lib/apt/lists /var/cache/apt/archives
COPY rootfs /
RUN mkdir /wp-content && chmod g+rwX /wp-content
COPY --from=builder /wordpress /wordpress
WORKDIR /wordpress
ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "php-fpm", "-F" ]
VOLUME [ "/wp-content" ]
USER 1001
...
```

Multistage build (same image to save space)

Install build dependencies (for cURL)

Extract + permissions/ownership (security)

Build/customize WordPress installation

Minimal image

Adapt base image to our needs

Install runtime dependencies & clean cache

Install WordPress in final container

Runtime configurations

Only persist “wp-content”

Run as non-root user (security)

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/1-container-image

WordPress Entrypoint Script

How the different services will interact with each other

```
#!/bin/bash
Set -euo pipefail
. /functions.sh
if [[ "$1" = "php-fpm" ]]; then
    # Persistence of application data
    if is_dir_empty /wp-content; then
        # Validate inputs
        assert_not_empty WORDPRESS_DB_HOST
        ...
        assert_check
        info "Persisting WordPress content"
        cp -r /wordpress/wp-content/. /wp-content
    else
        info "Deploying WordPress with persisted content"
    fi
fi
info "Starting WordPress server"
exec "$@"
```

Bash strict mode and load functions

Determine if app is being initialized

Validate inputs

Persist application data files (1st run only)

Already persisted data: Quick start (+ upgrade?)

Start app server

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/1-container-image

WordPress deployment on Kubernetes

Apache + mod_php example: bitnami/wordpress chart:

```
$ helm install my-wordpress bitnami/wordpress --set image.name=marcosbc/wordpress-modphp-example:5.7
```

- Parameters
- MariaDB sub-chart, ability to specify an external DB
- Optional Memcached support

NGINX + PHP-FPM example:

github.com/bitnami-labs/deploy-web-apps-on-kubernetes/tree/main/2-helm-chart

Thank You!

KubeCon | CloudNativeCon

Europe 2021

Virtual