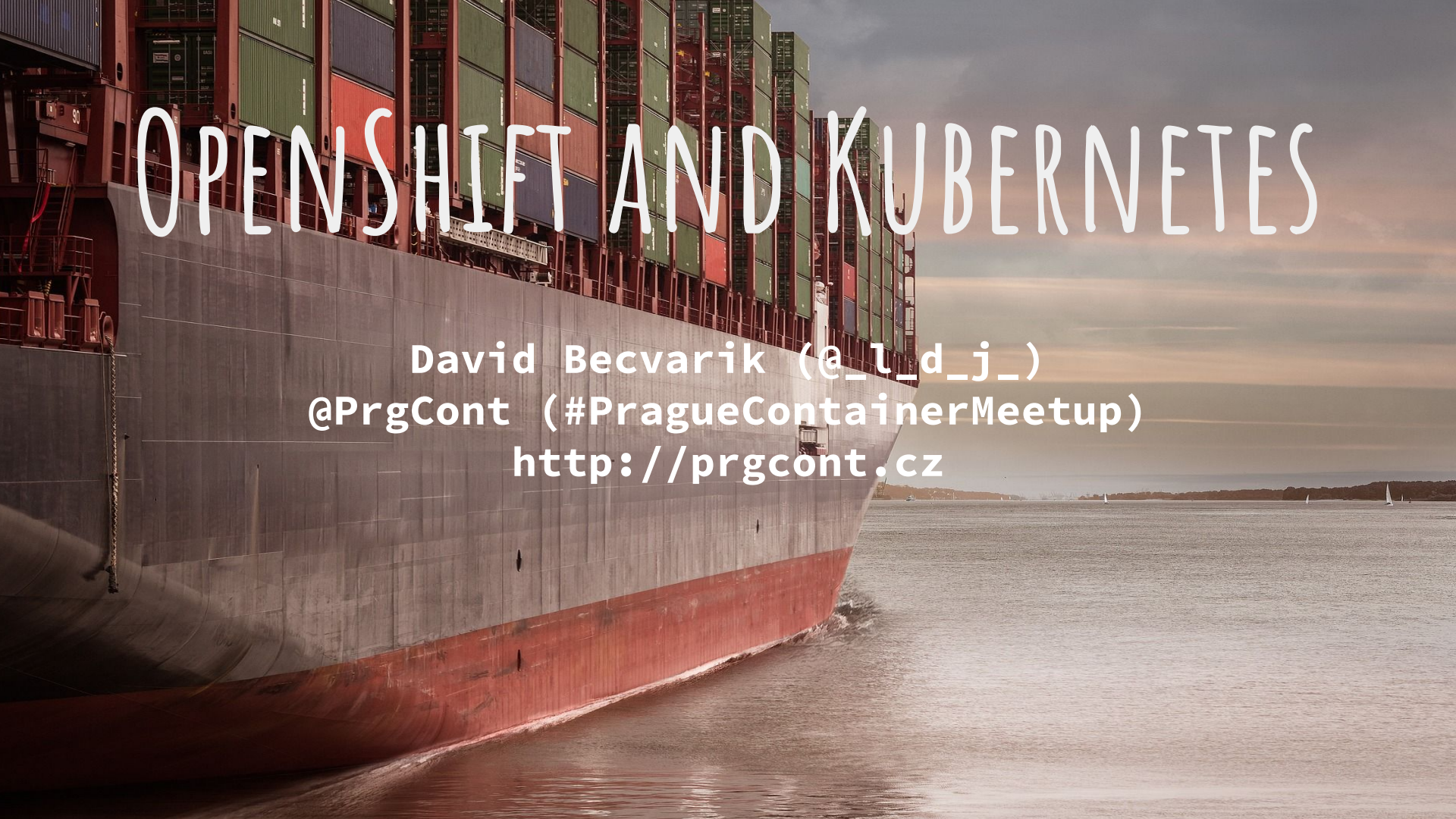


OpenShift and Kubernetes

David Becvarik (@_l_d_j_)

@PrgCont (#PragueContainerMeetup)

http://prgcont.cz

What should I know

- Linux administration (ssh, bash, basics diagnostics)
- What is a container (cgroups, namespaces, …)
- Basic networking (TCP, UDP, OSI model, …)

PaaS - is a **platform** which **enables** IT professional to **develop**, **build** and **run** application **without** taking care about **infrastructure**

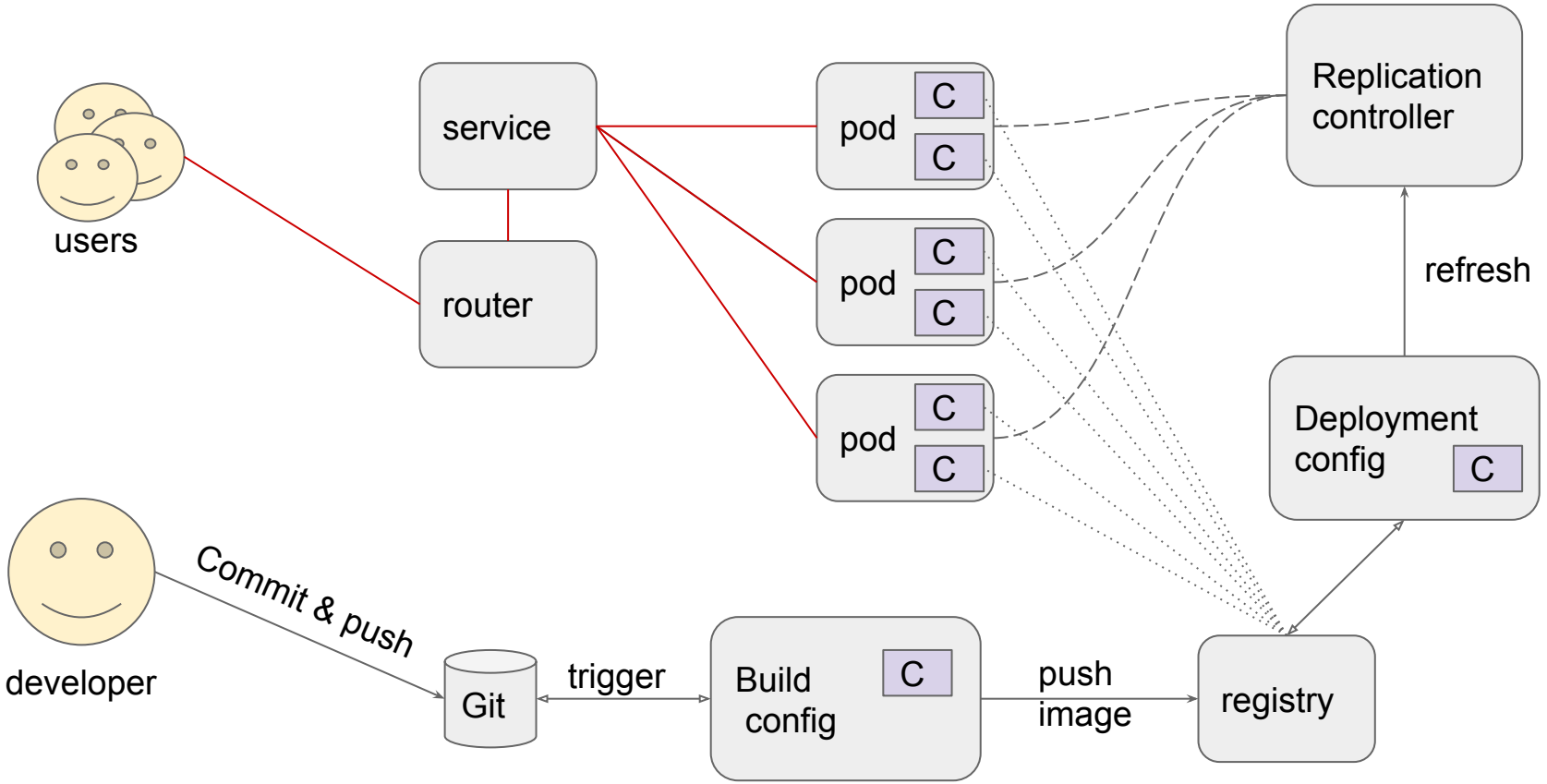
What we will speak about

Kubernetes

Open-Source platform for **automation** of an **application deployment**, scaling and operations in a **containerized environment**

Openshift

Platform based on kubernetes which encapsulates your **application lifecycle** from your **developer** commit to delivering your application to a cluster


users
service
router
pod
C
C
pod
C
C
pod
C
C
Replication controller
refresh
Deployment config
C
developer
Commit & push
Git
trigger
Build config
C
push image
registry

How it works


POD

POD is a one or more **container sharing** the same **host.**

They can **share network**
(One IP address, same localhost)

They can **share /dev/shm**
(OpenShift 1.5 and newer)

POD

How do I prevent my pod from eating my node resources?

Resource limits:

CPU

Memory (RAM)

This is cgroup limit - you need to tune your app too!

POD

POD - Health Check

Liveness probe

- Check if **application** is **alive**
- It means **listens** to on **port**
- It should really check only if app is running - OCP will wait for Readiness probe to succeed

Readiness probe

- Check if **application** is **ready**
- **Ping transaction**
- It should really check that app is **ready for customers**

In OpenShift we persist application state via **Persistent Volumes.** They are represented by network storage like NFS, CEPH, …

Persistent volumes are **created** in batch by OpenShift **cluster admin** and then **application** can **consume** them.

If there is **not enough** of Persistent Volumes - application deployment **will not finish**


Service

Service is a way how you can make multiple apps to communicate inside cluster

Service can be viewed as a load balancer object for pods

Its defined by an **IP address** and a **Port**

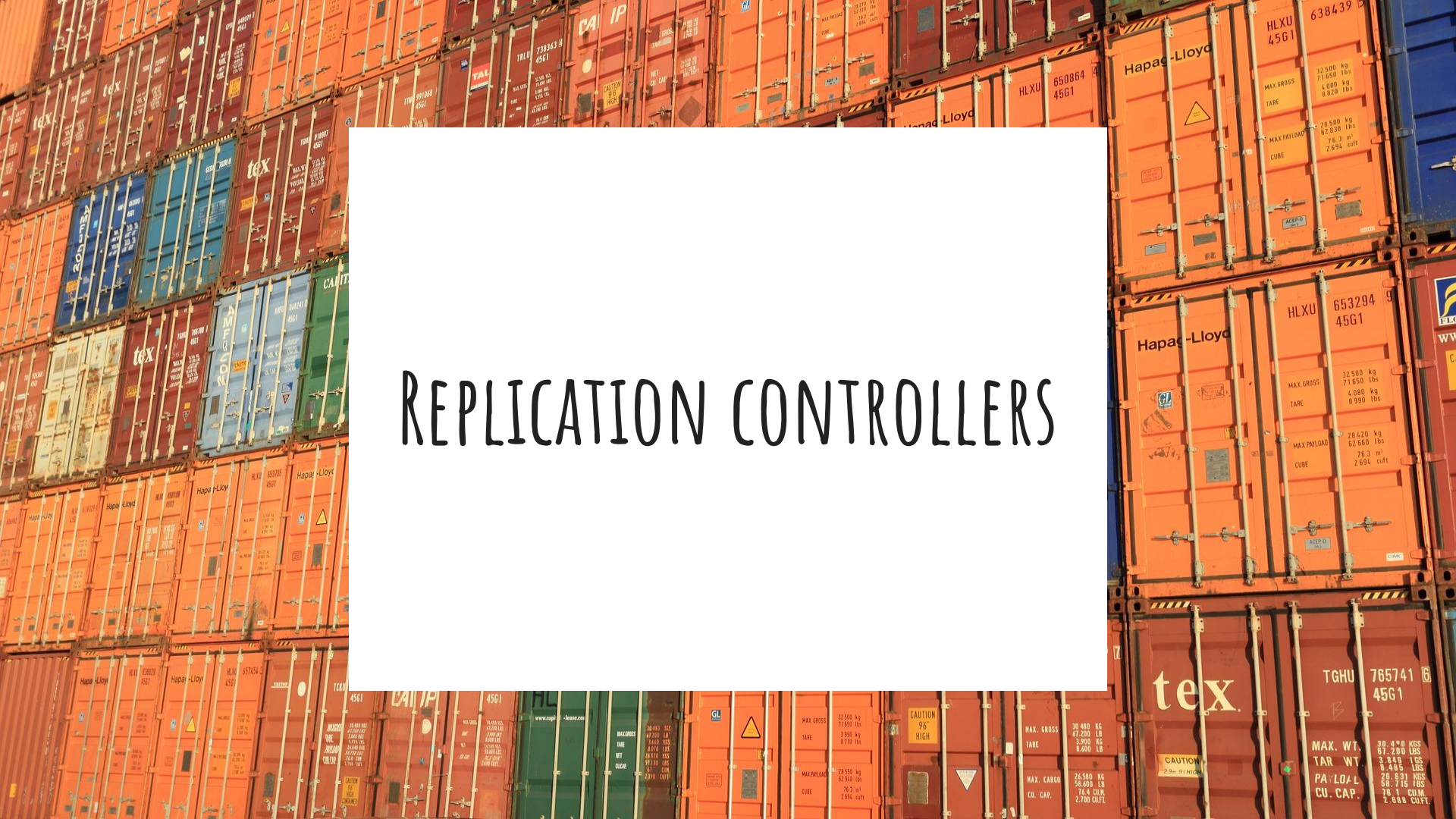
Service


Router

Enables external traffic to be routed to an OpenShift

Represented by **HAproxy** of **F5 BIG-IP**

ROUTER



Replication controllers

It is **responsible** to guard that **desired** number of **pod** replicas are **running** in a cluster

If pod **dies** - it is automatically **rescheduled**

If there is **more pod replicas** than **configured** - they get **killed**

Replication controler


Deployment

Its **OpenShift** addition make Replication Controller pattern more friendly with development/operation needs.

It consists:

Replication Controller

Triggers

Update strategy

lifecycle

ImageChange

When image changes in internal registry application is redeployed

Configuration Change trigger

Application is automatically redeployed when BuildConfig changes

Deployment - triggers

Rollout/Rollback

Pods are recreated **one by one by**. This strategy continues only when **readiness check** passes on new pod.

This is the safest option - it guarantee **no downtime** for end users. But requires your application to be **horizontally scalable** and to provide a way how to run **two versions** at once

Recreate

Old replication controller is scaled down and new pods are spinned up.

Custom

You are on your own.

Deployment - update strategy


BuildConfig

Build is a process of transforming resource into runnable object.

Typically it transform source code from git into docker image runnable by openshift

We can define multiple strategies

BuildConfig

Docker build strategy is plain docker build command.

It expect a repository with a Dockerfile as a parameter.

BuildConfig - Docker

Custom

It's completely on your own - you can even use jenkins and jenkins pipelines integration.

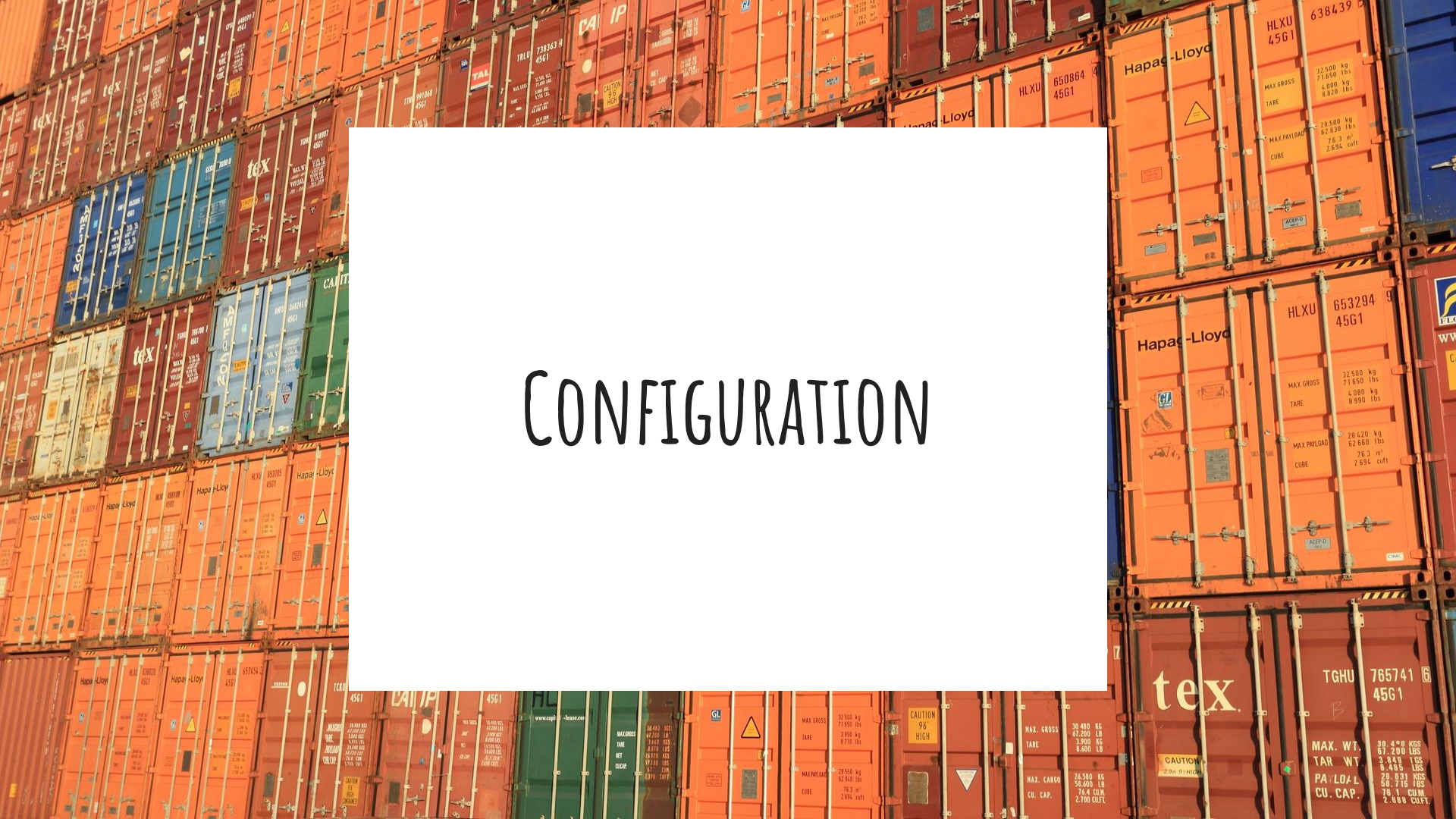
Pipeline are very powerful, they can contain builds, multiple deployments, approvements and really widely describe your app lifecycle - its most advanced and beneficial usage of OpenShift builds.

BuildConfig


S21

Source to image is a **tool** which enables OpenShift to **clone** a git **repository**, build sources, collect artifacts, **build image** with your **application**, push it to internal **registry**.

S2I


Configuration

The most **common way** to pass data to your application in a OpenShift origin cluster is via **environment variables**.

It's **not** very **secure** - anyone who has access to pod or openshift cluster can read it.

Environment variables

Key/value data object which can be used to store your application configuration.

It can be consumed via volumes, environment and so.

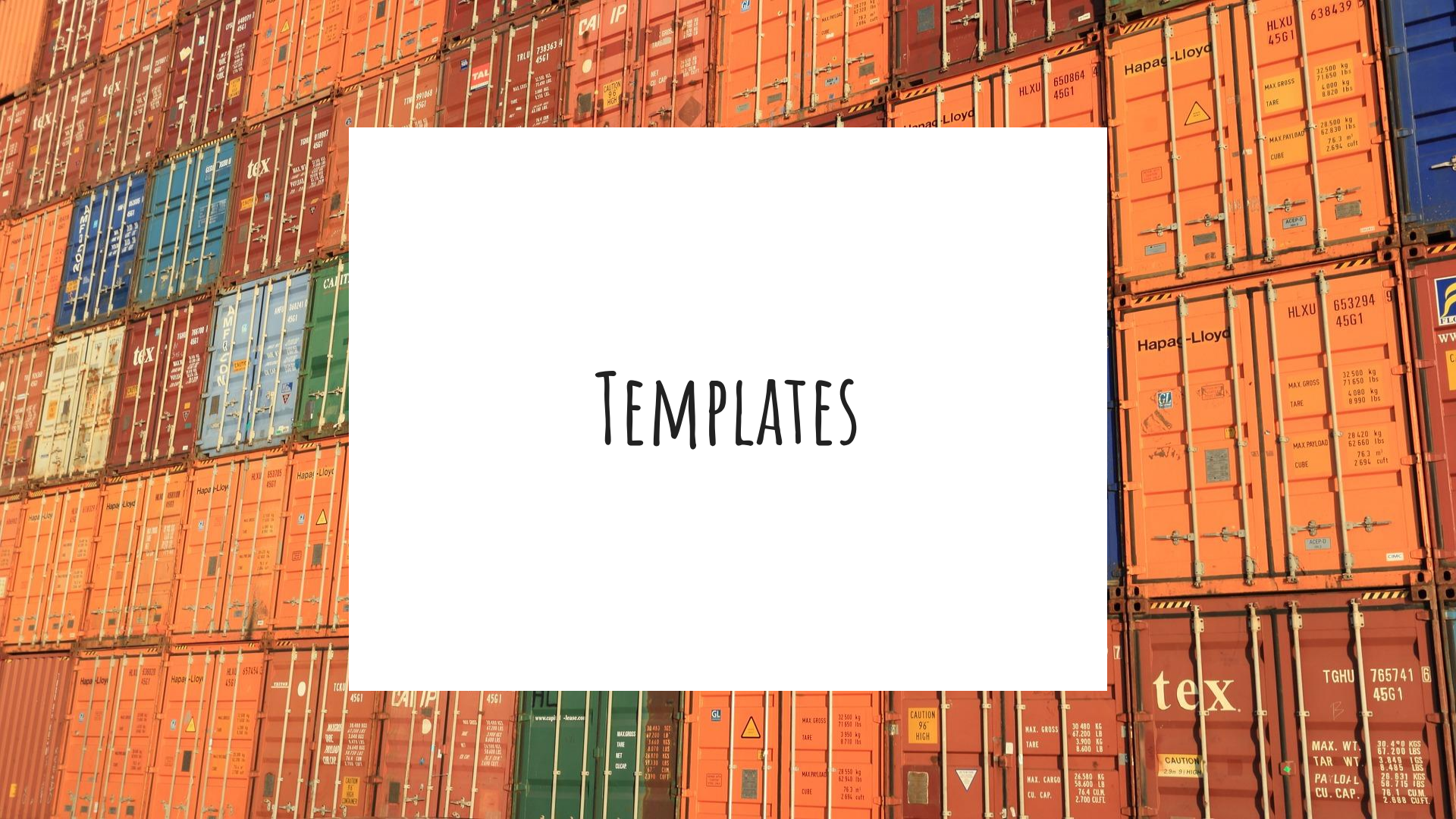
It's the most convenient **way** to insert **configuration files** into your **container**.

ConfigMaps

Secrets provides a way how to put some sensitive data to an application. They are not **encrypted** (only **base64**) and are **exposed** as **volumes** in pods.

Again any user with pod access can read it.

Secrets


Templates

Templates provide a way how to package complex application into an openshift and instantiate it multiple times with one click.

Templates

EAP pod,service, secrets with certificates

MySQL pod, service, persistent volumes

Build config - git repo + webhook

Deployment

Route

JBOSS EAP -MySQL template

Resources

- OpenShift documentation
  - https://docs.openshift.com/container-platform/3.5/welcome/index.html
- Kubernetes documentation
  - https://kubernetes.io/docs/home/
- Openshift source code
  - https://github.com/openshift/origin