

# **Building Scalable Applications using Docker and Kubernetes**

ΓΑΛΑΝΗΣ ΑΧΙΛΛΕΑΣ ΑΛΕΞΑΝΔΡΟΣ ΒΑΣΙΛΕΙΟΣ - 02941 and ΓΑΛΑΝΗΣ  
ΚΩΝΣΤΑΝΤΙΝΟΣ ΟΡΕΣΤΗΣ ΒΑΣΙΛΕΙΟΣ - 03074

## **Lab**

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών  
Πανεπιστήμιο Θεσσαλίας, Βόλος  
{acgalanis, kogalanis}@e-ce.uth.gr

## Table of Contents

Building Scalable Applications using Docker and Kubernetes .....	1
<i>ΓΑΛΑΝΗΣ ΑΧΙΛΛΕΑΣ ΑΛΕΞΑΝΔΡΟΣ ΒΑΣΙΛΕΙΟΣ - 02941 and</i>	
<i>ΓΑΛΑΝΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ ΟΡΕΣΤΗΣ ΒΑΣΙΛΕΙΟΣ - 03074</i>	
1 Προαπαιτούμενα .....	4
2 Εργαστήριο κατανόησης Docker .....	4
2.1 Δοχεία και Εικόνες .....	4
Εκτελώντας δοχεία .....	4
Κατασκευή περιβάλλοντος δοχείου .....	11
Κατασκευή εικόνων .....	19
Χρήση μητρώων εικόνων .....	27
2.2 Εφαρμογές πολλαπλών δοχείων .....	33
Docker Compose .....	33
Μοντελοποίηση εφαρμογών με Compose .....	40
Δημιουργία εφαρμογών με το Compose .....	48
Περιορισμοί του Compose .....	54
2.3 Multi-stage builds .....	61
Multi-stage builds .....	61
Δικτύωση δοχείων .....	66
Κατανόηση της ενορχήστρωσης .....	72
3 Εργαστήριο κατανόησης Kubernetes .....	78
3.1 Τα βασικά του Kubernetes .....	78
Nodes .....	78
Pods .....	79
Services .....	83
Deployments .....	91
3.2 Μοντελοποίηση Εφαρμογών .....	98
ConfigMaps .....	98
Secrets .....	108
PersistentVolumes .....	121
Namespaces .....	134
3.3 Προχωρημένη Μοντελοποίηση Εφαρμογών .....	149
Role-based Access Control (RBAC) .....	149
DaemonSets .....	158
Ingress .....	167
Jobs and CronJobs .....	184
StatefulSets .....	196
3.4 Λειτουργώντας το Kubernetes .....	207
Ετοιμότητα παραγωγής .....	207
Παρακολούθηση .....	225
Logging .....	242
3.5 Continuous Integration και Continuous Deployment (CI/CD) .....	254

BuildKit .....	254
Helm .....	255
Rollouts .....	255
Jenkins .....	255
GitOps .....	256
3.6 Advanced Kubernetes .....	257
NetworkPolicy .....	257
Admission .....	269
Χειριστές .....	284
3.7 Λειτουργίες Παραγωγής .....	297
Clusters .....	297
Affinity .....	297
Tools .....	298
4 Λύσεις εργαστηριακών ασκήσεων .....	298
4.1 Εργαστηριακή άσκηση 1 .....	298
4.2 Εργαστηριακή άσκηση 2 .....	299
4.3 Εργαστηριακή άσκηση 3 .....	299
4.4 Εργαστηριακή άσκηση 4 .....	300
4.5 Εργαστηριακή άσκηση 4 .....	301
4.6 Εργαστηριακή άσκηση 6 .....	302
4.7 Εργαστηριακή άσκηση 7 .....	303
4.8 Εργαστηριακή άσκηση 8 .....	304
4.9 Εργαστηριακή άσκηση 9 .....	305
4.10 Εργαστηριακή άσκηση 10 .....	305
4.11 Εργαστηριακή άσκηση 11 .....	307
4.12 Εργαστηριακή άσκηση 12 .....	308
4.13 Εργαστηριακή άσκηση 13 .....	312
4.14 Εργαστηριακή άσκηση 14 .....	313
4.15 Εργαστηριακή άσκηση 15 .....	319
4.16 Εργαστηριακή άσκηση 16 .....	320
4.17 Εργαστηριακή άσκηση 17 .....	322
4.18 Εργαστηριακή άσκηση 18 .....	325
4.19 Εργαστηριακή άσκηση 19 .....	326
4.20 Εργαστηριακή άσκηση 20 .....	328
4.21 Εργαστηριακή άσκηση 21 .....	329
4.22 Εργαστηριακή άσκηση 22 .....	332
4.23 Εργαστηριακή άσκηση 23 .....	334
4.24 Εργαστηριακή άσκηση 24 .....	337
4.25 Εργαστηριακή άσκηση 25 .....	339
4.26 Εργαστηριακή άσκηση 26 .....	342
4.27 Εργαστηριακή άσκηση 27 .....	346
5 Πηγές και βοηθητικό υλικό .....	348

## 1 Προαπαιτούμενα

Για την εγκατάσταση του Docker και του Kubernetes (εγκαταστήσαμε το minikube - μια ελαφρύτερη έκδοση του Kubernetes) ακολουθήσαμε το tutorial που βρίσκεται [εδώ](#) (το tutorial είναι έγκυρο και για Windows 11). Επίσης πρέπει να δημιουργήσετε λογαριασμό [Docker Hub](#) και να συνδεθείτε σε αυτόν (εκτελώντας την εντολή docker login).

## 2 Εργαστήριο κατανόησης Docker

### 2.1 Δοχεία και Εικόνες

#### Εκτελώντας δοχεία

##### – Εκτελώντας δοχεία

Τα δοχεία είναι ένα εικονικό υπολογιστικό περιβάλλον που δημιουργεί το Docker. Όταν εκτελείτε ένα δοχείο, συνήθως εκτελείται μια διαδικασία και μπορείτε να εκτελέσετε αυτήν τη διαδικασία με πολλούς τρόπους:

- [Docker command line reference](#)
- [Container commands](#)
- [docker run](#)

Το Docker command line στέλνει τις εντολές στο Docker API. Οι εντολές ομαδοποιούνται κατά τύπους αντικειμένων (π.χ. δοχεία ή δίκτυα). Μπορείτε πάντα να εκτυπώσετε τις διαθέσιμες εντολές και τις επιλογές τους:

```
# CLI Reference
docker --help
docker container --help
docker container run --help
```

##### – Εκτελώντας one-off δοχεία

Τα δοχεία Docker λειτουργούν όσο εκτελείται η διαδικασία μέσα στο δοχείο. Όταν τελειώσει η διαδικασία, το δοχείο τερματίζει.

- Μπορείτε να εκτελέσετε ένα απλό δοχείο που εκτελεί μία μόνο εντολή Linux (Πολλές εντολές Docker έχουν παραλλαγές - το docker run είναι το ίδιο με το docker container run):

```
docker run alpine hostname
```

```

C:\Users\Axilleas>docker run alpine hostname
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
661ff4d9561e: Pull complete
Digest: sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
Status: Downloaded newer image for alpine:latest
0ca40020a039

C:\Users\Axilleas>docker run alpine hostname
615a9bbce1ad

C:\Users\Axilleas>docker run alpine hostname
3457d3a8535b

C:\Users\Axilleas>

```

**Fig. 1.** Έξοδος εντολής "docker run alpine hostname"

Θα δείτε πολλά διαφορετικά αποτελέσματα όταν εκτελέσετε αυτή την εντολή. Αυτό γιατί το Docker τραβάει την εικόνα του Alpine Linux από το Docker Hub ώστε να μπορεί να εκτελεστεί τοπικά, ξεκινά ένα δοχείο με αυτή την εικόνα, εκτελεί την εντολή 'hostname' και εκτυπώνει την έξοδο. Επομένως κάθε φορά η εικόνα ανατίθεται σε διαφορετικό δοχείο άρα και διαφορετική εικονική μηχανή με δικό της hostname. Εκτελώντας την ίδια εντολή παραπάνω φορές παρατηρείτε ότι δεν τραβάμε ξανά την εικόνα αλλά εκτυπώνετε μόνο ένα διαφορετικό αποτέλεσμα.

- Εκτυπώστε μια λίστα με όλα τα δοχεία σας:



```

# print running containers:
docker container ls

# or use ps
docker ps

# the -a flag shows all statuses
docker ps -a

```

<input type="checkbox"/>		alpine:latest	alpine	Exited	N/A	6 minutes ago
<input type="checkbox"/>		alpine:latest	alpine	Exited	N/A	6 minutes ago

**Fig. 2.** Προβολή των δοχείων στο Docker Desktop



```

C:\Users\Avilios>docker container ls -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS         PORTS         NAMES
006024040e     kiama/cert-generator                "/bin/sh -c './start...' 4 hours ago    Exited (130) 4 hours ago    festive_ptolemy
022604040e     kiama/cert-generator               "/bin/sh -c './start...' 4 days ago     Exited (13) 7 hours ago    odious_lamarr
7aa07200f01    kiama/cert-generator               "/bin/sh -c './start...' 4 days ago     Exited (13) 7 hours ago    strange_walton
006024040e     kiama/cert-generator               "/bin/sh -c './start...' 12 days ago    Exited (137) 9 hours ago    #010404

C:\Users\Avilios>docker container logs 006024040e
Generating certs - hostname: kiama.local; IP: 127.0.0.1; SAN: DNS:hello.kiama.local,DNS:web.kiama.local,DNS:todo.kiama.local,DNS:todo2.kiama.local,DNS:pi.kiama.local, expiry days: 730
Generating RSA private key, 4096 bit long modulus (2 primes)
.....
Generating RSA private key, 4096 bit long modulus (2 primes)
.....
.....
# is ready (006024040e)
Signature ok
subject=C=kiama.local
Getting CA Private key
Certs generated
#
C:\Users\Avilios>

```

Fig. 5. Έξοδος εντολών για εκτύπωση logs

Τα container logs είναι η έξοδος από τη διαδικασία του δοχείου - Το Docker τα αποθηκεύει κατά τον τερματισμό του δοχείου.

Η εικόνα του cert-generator συσκευάζει τη βιβλιοθήκη OpenSSH με ένα script για τη δημιουργία πιστοποιητικών HTTPS για ένα σύνολο τομέων, με προεπιλεγμένες ρυθμίσεις. Θα χρειαστεί να εγκαταστήσετε τις βιβλιοθήκες και να κάνετε λήψη του script για να έχετε το ίδιο αποτέλεσμα χωρίς τη χρήση του Docker.

#### – Εκτελώντας διαδραστικά δοχεία

Τα one-off δοχεία εκτελούνται και μετά τερματίζουν. Μπορείτε να εκτελέσετε μια μακροχρόνια διαδικασία σε ένα δοχείο και να συνδέσετε το τερματικό σας στο τερματικό του δοχείου.

- Αυτό είναι σαν να συνδέσετε σε ένα απομακρυσμένο μηχάνημα - οποιεσδήποτε εντολές εκτελείτε, εκτελούνται πραγματικά μέσα στο δοχείο:

```
docker run -it alpine
```

Η επιλογή -it σημαίνει ότι εκτελείται διαδραστικά, με συνδεδεμένο το τοπικό τερματικό σας και η προεπιλεγμένη εντολή για το κοντέινερ Alpine είναι να τρέξει το Linux kernel.

- Τώρα που είστε συνδεδεμένοι στο δοχείο, μπορείτε να εξερευνήσετε το περιβάλλον του:

```
docker run -it alpine
```

```

C:\Users\Axilleas>docker run -it alpine
/ # ls /
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
/ # whoami
root
/ # hostname
7add0dbf8bb8
/ # cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.19.0
PRETTY_NAME="Alpine Linux v3.19"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"
/ # exit
C:\Users\Axilleas>

```

**Fig. 6.** Έξοδος εντολών στο περιβάλλον του δοχείου

Μπορείτε να διαπιστώσετε ότι το δοχείο συμπεριφέρεται σαν ένα σύστημα Linux, που εκτελεί την έκδοση Alpine.

Τα δοχεία λειτουργικού συστήματος Linux για Docker έχουν συνήθως εγκατεστημένο το ελάχιστο σύνολο εργαλείων.

Η ομάδα του Ubuntu δημοσιεύει ένα πακέτο για τον διακομιστή Ubuntu αλλά δεν έχει εγκατεστημένα τα συνηθισμένα εργαλεία. Για παράδειγμα, δεν υπάρχει curl, επομένως δεν μπορείτε να πραγματοποιήσετε κλήσεις HTTP, αλλά το δοχείο εκτελείται ως χρήστης root, ώστε να έχετε δικαιώματα για να εγκαταστήσετε οτιδήποτε χρειάζεστε:

```

docker run -it ubuntu

curl https://docker.courselabs.co # command not found

apt-get install -y curl # this doesn't work - there's no
install cache

apt-get update # update the cache

apt-get install -y curl # now this works

curl https://docker.courselabs.co

exit

```

Οι αλλαγές που πραγματοποιήσατε αφορούν μόνο αυτό το ένα κοντέινερ.

- Εκτελέστε ένα άλλο κοντέινερ Ubuntu και δείτε αν μπορεί να χρησιμοποιήσει το curl:

```

# you can do this as a one-off container - it won't work:
docker run ubuntu bash -c curl https://docker.courselabs.co

```



```
C:\Users\Axilleas>docker run ubuntu bash -c curl https://docker.courselabs.co
https://docker.courselabs.co: line 1: curl: command not found

C:\Users\Axilleas>
```

Fig. 7. Έξοδος εντολής "docker run ubuntu bash -c curl https://docker.courselabs.co"

- Τα διαδραστικά δοχεία μπορούν να κάνουν διασκεδαστικά πράγματα με την οθόνη - αυτό είναι χρήσιμο για να εντυπωσιάσετε τους ανθρώπους με τις δεξιότητές σας στο hacking :)

```
docker run -it sixeyed/hollywood

# Ctrl-C / Cmd-C to stop the process

# and exit the container
exit
```

#### – Εκτέλεση ενός background δοχείου

Τα διαδραστικά δοχεία είναι εξαιρετικά για τη δοκιμή εντολών και την εξερεύνηση της οργάνωσης του δοχείου, αλλά κυρίως θα θέλετε να εκτελείτε αποσπώμενα δοχεία.

Αυτά εκτελούνται στο background, επομένως το δοχείο συνεχίζει να λειτουργεί μέχρι να τερματιστεί η διαδικασία εφαρμογής ή να σταματήσετε το δοχείο.

- Θα χρησιμοποιούσατε background δοχεία για διακομιστές ιστού, διεργασίες batch, ουρές μηνυμάτων βάσεων δεδομένων και οποιαδήποτε άλλη διαδικασία διακομιστή:

```
docker run -d -P --name nginx1 nginx:alpine
```

- Η επιλογή -d εκτελεί ένα αποσπασμένο background δοχείο.
- Το -P δημοσιεύει θύρες δικτύου, ώστε να μπορείτε να στέλνετε αντικείμενα στο δοχείο.
- Το --name δίνει στο δοχείο ένα όνομα, ώστε να μπορείτε να το προσθέσετε σε άλλες εντολές.

Τώρα έχετε έναν απλό διακομιστή ιστού που τρέχει στο δοχείο που ονομάζεται nginx1

- Μάθετε σε ποια θύρα ακούει το κοντέινερ και δοκιμάστε να περιηγηθείτε στον διακομιστή ιστού.

```
# print the container's port mapping
docker container port nginx1

# browse to the port with curl or your browser
curl localhost:<container-port>
```

#### Welcome to nginx!

If you see this page, the nginx web server is successfully installed and  
working. Further configuration is required.  
For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).  
Thank you for using nginx.



**Fig. 8.** Σύνδεση στο port του localhost

Το Docker ακούει την εισερχόμενη κίνηση στη θύρα δικτύου του υπολογιστή σας και την προωθεί στο δοχείο.

### – Εργαστηριακή Άσκηση 1

Έχουμε εκτελέσει δοχεία χρησιμοποιώντας Alpine, Linux και Ubuntu, καθώς και με εγκατεστημένο το Nginx. Είναι όλα δημόσια πακέτα διαθέσιμα στο Docker Hub.

Σε αυτή την άσκηση θα εργαστείτε με δοχεία Java:

- Βρείτε ένα πακέτο στο Docker Hub το οποίο μπορείτε να χρησιμοποιήσετε για να εκτελέσετε μια εφαρμογή Java.
- Εκτελέστε ένα δοχείο Java και επιβεβαιώστε ποια έκδοση της Java είναι εγκατεστημένη χρησιμοποιώντας την εντολή `java -version`.
- Τέλος βρείτε μια μικρή εικόνα για Java 8, με εγκατεστημένο μόνο το JRE.

### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## Κατασκευή περιβάλλοντος δοχείου

- **Κατασκευή περιβάλλοντος δοχείου** Το Docker δημιουργεί το εικονικό περιβάλλον για ένα δοχείο. Μπορείτε να καθορίσετε τον τρόπο ρύθμισης των τμημάτων του περιβάλλοντος, κάτι που σας επιτρέπει να προετοιμάσετε το δοχείο σας για τις εφαρμογές σας.

Τα περιβάλλοντα δοχείων είναι στατικά - τα χαρακτηριστικά του είναι σταθερά για τη διάρκεια ζωής του δοχείου. Μπορείτε να διαμορφώσετε το περιβάλλον με την εντολή `docker run`.

Αναφορές:

- [Setting environment variables](#)
- [Mounting volumes](#)
- [Applying resource constraints](#)

Υπάρχουν δεκάδες επιλογές για τη λειτουργία ενός δοχείου:

```
docker run --help
```

Οι επιλογές που θα χρησιμοποιήσετε είναι:

- **-e** για να ορίσετε την τιμή για μια μεμονωμένη μεταβλητή περιβάλλοντος.
- **--env-file** για να ορίσετε πολλές μεταβλητές περιβάλλοντος από δεδομένα σε αρχείο κειμένου.
- **-v** για να προσαρτήσετε έναν τοπικό κατάλογο στο σύστημα αρχείων του δοχείου.
- **--cpus** και **--memory** για τον καθορισμό των υπολογιστικών πόρων που είναι διαθέσιμοι στο δοχείο.

## – Μεταβλητές περιβάλλοντος

Οι μεταβλητές περιβάλλοντος είναι ζεύγη κλειδιών-τιμών που ορίζονται στο λειτουργικό σύστημα. Οι εφαρμογές μπορούν να τα διαβάσουν και χρησιμοποιούνται συχνά για ρυθμίσεις διαμόρφωσης.

Το `printenv` είναι η εντολή Linux για να τις εκτυπώσετε:

```
docker run alpine printenv
```

```
docker run openjdk:8-jre-alpine printenv
```

```

C:\Users\Axilleas>docker run alpine printenv
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
661ffad9561e: Pull complete
Digest: sha256:51b07209f354137895d43f2b3d910bfacd3945438e94dc5ac55fdac340352f48
Status: Downloaded newer image for alpine:latest
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=db22c849834d
HOME=/root

C:\Users\Axilleas>docker run openjdk:8-jre-alpine printenv
Unable to find image 'openjdk:8-jre-alpine' locally
8-jre-alpine: Pulling from library/openjdk
e7c96db7181b: Pull complete
f91ba506b6cb: Pull complete
b6abafe89f63: Pull complete
Digest: sha256:f362b165b870ef129cbe738f29065ff37399c8aa8bcab3e44b51c302938c9193
Status: Downloaded newer image for openjdk:8-jre-alpine
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/bin
HOSTNAME=f0f80316d9a
LANG=C.UTF-8
JAVA_HOME=/usr/lib/jvm/java-1.8-openjdk/jre
JAVA_VERSION=8u12
JAVA_ALPINE_VERSION=8.212.04-r0
HOME=/root

C:\Users\Axilleas>

```

**Fig. 9.** Έξοδος εντολών για προβολή μεταβλητών περιβάλλοντος

Θα δείτε μεταβλητές που ορίζονται από το λειτουργικό σύστημα και στο πακέτο κοντέινερ.

- Εκτελέστε ένα νέο δοχείο - ορίστε τη δική σας μεταβλητή περιβάλλοντος και εκτυπώστε όλες τις μεταβλητές.

```

# -e adds a new environment variable
docker run -e COURSELABS=env alpine printenv

```

```

C:\Users\Axilleas>docker run -e COURSELABS=env alpine printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=56903817403a
COURSELABS=env
HOME=/root

C:\Users\Axilleas>

```

**Fig. 10.** Έξοδος εντολής "docker run -e COURSELABS=env alpine printenv"

Οι μεταβλητές περιβάλλοντος δεν μπορούν να αλλάξουν, έχουν οριστεί για τη διάρκεια ζωής του δοχείου.

- Μπορείτε να προσθέσετε όσες μεταβλητές περιβάλλοντος χρειάζεστε και μπορείτε επίσης να παρακάμψετε τις προεπιλεγμένες μεταβλητές για το δοχείο:

```

docker run -e COURSELABS=env -e LANG=C.UTF-16 openjdk:8-jre-alpine printenv

```

```
C:\Users\Axilleas>docker run -e COURSELABS=env -e LANG=C.UTF-16 openjdk:8-jre-alpine printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/bin
HOSTNAME=a76f62b96666
COURSELABS=env
LANG=C.UTF-16
JAVA_HOME=/usr/lib/jvm/java-1.8-openjdk/jre
JAVA_VERSION=8u212
JAVA_ALPINE_VERSION=8.212.04-r0
HOME=/root
C:\Users\Axilleas>
```

**Fig. 11.** Έξοδος εντολής "docker run -e COURSELABS=env -e LANG=C.UTF-16 openjdk:8-jre-alpine printenv"

Αυτό παρακάμπτει την προεπιλεγμένη ρύθμιση γλώσσας στο δοχείο Java.

Εάν ορίζετε πολλές μεταβλητές, είναι πιο εύκολο να τις αποθηκεύσετε όλες σε ένα αρχείο όπως το var.env και να τις μεταβιβάσετε στο δοχείο ως αρχείο περιβάλλοντος.

- Εκτελέστε το δοχείο περνώντας ένα αρχείο μεταβλητών περιβάλλοντος:

```
# save these in a local file named var.env:
LOG_LEVEL=debug
FEATURES_CACHE_ENABLED=true

# run a container loading that file as environment variables:
docker run --env-file path/to/var.env alpine printenv
```

```
C:\Users\Axilleas>docker run --env-file E:\Users\Achilleas\Desktop\var.env alpine printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=a1126ffd44a5
LOG_LEVEL=debug
FEATURES_CACHE_ENABLED=true
HOME=/root
C:\Users\Axilleas>
```

**Fig. 12.** Έξοδος εντολής "docker run --env-file path/to/var.env alpine printenv"

Τα περιεχόμενα των .env αρχείων αντικαθιστούν τις προεπιλεγμένες τιμές, αλλά μπορείτε επίσης να τις αντικαταστήσετε χρησιμοποιώντας την επιλογή -e.

#### – Σύστημα αρχείων δοχείου

Τα συστήματα αρχείων ενός δοχείου είναι εικονικοί δίσκοι, δημιουργημένοι από τον Docker. Κάθε δοχείο ξεκινά με τα περιεχόμενα του δίσκου που έχουν ρυθμιστεί από το δοχείο.

- Τρέξτε ένα background δοχείο Nginx που ονομάζεται nginx:

```
# alpine is the smallest variant but any will do:
docker run -d --name nginx nginx:alpine
```

- Μπορείτε να συνδεθείτε σε ένα απομονωμένο δοχείο και να εκτελέσετε εντολές σε αυτό για την εξερεύνηση του συστήματος αρχείων:

```
docker exec -it nginx sh

ls /usr/share/nginx/html/

cat /usr/share/nginx/html/index.html

exit
```

```
C:\Users\Axilleas>docker run -d --name nginx nginx:alpine
d43e36f42ceaf95dc46d0e9cb868d71a4dab2fb4a7bbebb42dc041aebfd59f33

C:\Users\Axilleas>docker exec -it nginx sh
/ # ls /usr/share/nginx/html/
50x.html    index.html
/ # cat /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
/ # exit

C:\Users\Axilleas>|
```

**Fig. 13.** Έξοδος εντολών εξερεύνησης συστήματος αρχείων

Το δοχείο έχει εγκατεστημένο τον διακομιστή ιστού Nginx και ορισμένες προεπιλεγμένες σελίδες HTML.

- Μπορείτε να συνδεθείτε σε ένα απομονωμένο δοχείο και να εκτελέσετε εντολές σε αυτό για την εξερεύνηση του συστήματος αρχείων:

```
docker exec -it nginx sh

ls /usr/share/nginx/html/

cat /usr/share/nginx/html/index.html

exit
```

```
C:\Users\Axilleas>docker run -d --name nginx nginx:alpine
d43e36f42ceaf95dc46d0e9cb868d71a4dab2fb4a7bbebb42dc041aebfd59f33

C:\Users\Axilleas>docker exec -it nginx sh
/ # ls /usr/share/nginx/html/
50x.html    index.html
/ # cat /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
/ # exit

C:\Users\Axilleas>|
```

**Fig. 14.** Έξοδος εντολών εξερεύνησης συστήματος αρχείων

Το δοχείο έχει εγκατεστημένο τον διακομιστή ιστού Nginx και ορισμένες προεπιλεγμένες σελίδες HTML.

Μπορείτε να προσαρτήσετε έναν κατάλογο από τον τοπικό σας υπολογιστή στο σύστημα αρχείων του δοχείου. Μπορείτε να το χρησιμοποιήσετε για να προσθέσετε νέο περιεχόμενο ή για να παρακάμψετε υπάρχοντα αρχεία.

- Εκτελέστε ένα δοχείο προσθέτοντας του ένα καινούργιο index.html για να εμφανίσετε τη νέα ιστοσελίδα - πρέπει να χρησιμοποιήσετε την πλήρη διαδρομή ως πηγή για τον τόμο (η επιλογή -v προσαρτά έναν τοπικό κατάλογο στο δοχείο - οι μεταβλητές αποθηκεύουν την πλήρη διαδρομή και σημαίνουν ότι μπορούμε να χρησιμοποιήσουμε την ίδια εντολή Docker σε οποιοδήποτε λειτουργικό σύστημα):

```
# create this index.html file and put it in a index.html
  folder
<!DOCTYPE html>
<html>
<head>
  <title>Docker Course</title>
</head>
<body>
  <h1>Docker Course</h1>
</body>
</html>

# put the full local path in a variable - on macOS/Linux:
htmlPath="${PWD}/html"

# OR with PowerShell:
$htmlPath="${PWD}/html"

# run a container mounting the local volume to the HTML
  directory:
docker run -d -p 8081:80 -v ${htmlPath}:/usr/share/nginx/html
  --name nginx2 nginx:alpine
```



**Fig. 15.** Ιστοσελίδα

Περιηγηθείτε στο `http://localhost:8081` και θα δείτε την καινούργια απόκριση HTML.

Το δοχείο διαβάζει δεδομένα από το τοπικό σας μηχάνημα. Μπορείτε να επεξεργαστείτε το `index.html` και όταν ανανεώσετε το πρόγραμμα περιήγησής σας θα δείτε αμέσως τις αλλαγές σας.

#### – Υπολογισμός πόρων

Μπορείτε να περιορίσετε την ποσότητα υπολογιστικής ισχύος των δοχείων. Από προεπιλογή δεν υπάρχει περιορισμός, επομένως το δοχείο βλέπει όλες τις CPU και τη μνήμη που διαθέτει το μηχάνημά σας.

- Εκτελέστε μια εφαρμογή υψηλής υπολογιστικής ισχύος χωρίς περιορισμούς (η επιλογή `-m` μεταβιβάζεται στην εφαρμογή μέσα στο δοχείο, δεν αφορά τη διαμόρφωση του δοχείου.):

```
docker run -d -p 8031:80 --name pi kiamol/ch05-pi -m web
```

Περιηγηθείτε στο `http://localhost:8031/pi?dp=50000` - αυτό υπολογίζει το  $\Pi$  σε 50K δεκαδικά ψηφία. Στο μηχάνημά μου παίρνει  $< 0.5$  δευτερόλεπτα.

Δεν μπορείτε να αλλάξετε τους πόρους της CPU που χρησιμοποιεί ένα δοχείο, είναι προκαθορισμένοι.

- Επιθεωρήστε τα χαρακτηριστικά του δοχείου:

```
docker inspect pi
```

```
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
```

**Fig. 16.** Όρια CPU

Όλα τα όρια CPU και μνήμης έχουν οριστεί στο 0, που σημαίνει ότι δεν είναι περιορισμένα.

- Δοκιμάστε να εκτελέσετε την ίδια εφαρμογή σε ένα κοντέινερ με μόνο 200 Mb μνήμης και με το 1/4 του πυρήνα της CPU:

```
docker rm -f pi

docker run -d -p 8031:80 --name pi --memory 200m --cpus 0.25
kiamol/ch05-pi -m web
```

Ανανεώστε το <http://localhost:8031/pi?dp=50000> και θα διαπιστώσετε ότι θα χρειαστεί πολύ περισσότερος χρόνος.

- Μπορείτε να εκτυπώσετε συγκεκριμένα μέρη της διαμόρφωσης δοχείου χρησιμοποιώντας [formatting](#):

```
docker container inspect --format='Memory: b, CPU: n' pi
```

Η μνήμη εκτυπώνεται σε byte και η CPU σε νανο-πύρηνες (1 δισεκατομμυριοστό του πυρήνα!).

## – Εργαστηριακή Άσκηση 2

Στην προηγούμενη ενότητα εκτελέσατε ένα δοχείο για τη δημιουργία πιστοποιητικών TLS σ, αλλά τα πιστοποιητικά δημιουργήθηκαν μέσα στο σύστημα αρχείων του δοχείου.

Σε αυτή την άσκηση η δουλειά σας είναι να αντιγράψετε το πιστοποιητικό TLS και το κλειδί από το δοχείο στον τοπικό σας υπολογιστή.

Ξεκινήστε δημιουργώντας πιστοποιητικά σε ένα νέο δοχείο - εκτελέστε το στο background και το δοχείο θα παραμείνει ενεργό:

```
docker run -d --name tls kiamol/ch15-cert-generator
```

Τώρα αντιγράψτε τα αρχεία server-cert.pem και server-key.pem από το φάκελο /certs του δοχείου στον υπολογιστή σας.

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

### Κατασκευή εικόνων

#### – Κατασκευή εικόνων δοχείων

Οι εικόνες είναι τα πακέτα τα οποία εκτελούν τα δοχεία. Θα δημιουργήσετε μια εικόνα για κάθε στοιχείο της εφαρμογής σας και η εικόνα έχει όλα τις προαπαιτούμενες εξαρτήσεις εγκατεστημένες και διαμορφωμένες, έτοιμη προς εκτέλεση. Μπορείτε να σκεφτείτε τις εικόνες ως το σύστημα αρχείων για το δοχείο, καθώς και κάποια μεταδεδομένα που λένε στον Docker ποια εντολή θα εκτελεστεί κατά την εκκίνηση του.

Αναφορές:

- [Dockerfile syntax](#)
- [Image build command](#)

Χρησιμοποιείτε τις εντολές "image" για να εργαστείτε με εικόνες. Η εντολή build έχει επίσης παραλλαγές:

```
docker image --help
```

```
docker build --help
```

```
docker history --help
```

#### – Εικόνες βάσης

Οι εικόνες μπορούν να δημιουργηθούν σε μια ιεραρχία - μπορείτε να ξεκινήσετε με μια εικόνα λειτουργικού συστήματος που ρυθμίζει μια συγκεκριμένη διανομή Linux και, στη συνέχεια, δημιουργήστε πάνω από αυτήν το runtime της εφαρμογής σας.

- Πριν δημιουργήσουμε οποιεσδήποτε εικόνες, θα ρυθμίσουμε το Docker να χρησιμοποιεί την αρχική μηχανή κατασκευής:

```
# on macOS or Linux:
export DOCKER_BUILDKIT=0

# OR with PowerShell:
$env:DOCKER_BUILDKIT=0
```

Το **BuildKit** είναι η νέα μηχανή κατασκευής εικόνων Docker. Παράγει τις ίδιες εικόνες με τον αρχικό engine, αλλά η εκτυπωμένη έξοδος δεν δείχνει την εκτέλεση των οδηγιών Dockerfile. Χρησιμοποιούμε το αρχικό engine, ώστε να μπορείτε να δείτε όλα τα βήματα.

- Θα δημιουργήσουμε μια πολύ απλή εικόνα βάσης (Το `-t` ή το `-tag` δίνει στην εικόνα ένα όνομα. Τερματίζετε πάντα την εντολή `build` με τη διαδρομή προς το φάκελο που περιέχει το Dockerfile):

```
# write the base image in a Dockerfile and save it in a
  folder called base
FROM ubuntu

# build the image
docker build base ./base
```

```
PS E:\Users\Achilleas\Desktop> docker build -t labs/base ./base
[+] Building 8.3s (6/6) FINISHED
=> [internal] load .dockerignore                                docker:default 0.1s
=> => transferring context: 2B                                  0.0s
=> [internal] load build definition from Dockerfile             0.1s
=> => transferring dockerfile: 48B                               0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 1.9s
=> [auth] library/ubuntu:pull token for registry-1.docker.io   0.0s
=> [1/1] FROM docker.io/library/ubuntu@sha256:6e42508cf4b44023ea1894effe7899666b0c5c7871ed83a97c36c76ae560bb9b 6.1s
=> => resolve docker.io/library/ubuntu@sha256:6e42508cf4b44023ea1894effe7899666b0c5c7871ed83a97c36c76ae560bb9b 0.0s
=> => sha256:a486411936734b0d1d201c8a0ed8e9d449a64d50833fdc33411ec95bc26460efb 29.55MB / 29.55MB 4.9s
=> => sha256:6042508cf4b44023ea1894effe7899666b0c5c7871ed83a97c36c76ae560bb9b 1.13kB / 1.13kB 0.0s
=> => sha256:bbf3d1baa208b7649d1d0264ef7d522e1dc0deeeaa6f6085bf8e4618867f93494 424B / 424B 0.0s
=> => sha256:174c8c134b2a94b5bb0b37d9a2b6ba0663d82d23ebf62bd51f74a2fd457333da 2.30kB / 2.30kB 0.0s
=> => extracting sha256:a486411936734b0d1d201c8a0ed8e9d449a64d50833fdc33411ec95bc26460efb 0.9s
=> => exporting to image                                          0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:1eb4201ffe5e015911b098c4f89f5e94ce42d4d8c9d35eabcbe5f1cea657b22 0.0s
=> => naming to docker.io/labs/base                             0.0s

View build details: docker--desktop:///dashboard/build/default/default/inqz7q2xtjd10y23elwky6q

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Users\Achilleas\Desktop>
```

Fig. 17. Έξοδος εντολής "docker build base ./base"

- Καταγράψτε όλες τις εικόνες που έχετε - στη συνέχεια φιλτράρετε τις για εικόνες που ξεκινούν με "labs":

```
# list all local images:
docker image ls
```

```
# and filter for the courselabs images:
docker image ls 'labs/*'
```

```
PS E:\Users\Achilleas\Desktop> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
axilleas7/myapp-backend	1.0	165c783f364a	13 days ago	918MB
axilleas7/myapp-frontend	1.0	3c3b215b6ef8	2 weeks ago	42.9MB
labs/base	latest	1eb4281fffe5e	4 weeks ago	77.9MB
gcr.io/k8s-minikube/kicbase	v0.8.42	0bc648f75a85	2 months ago	1.2GB
hubproxy.docker.internal:5555/docker/desktop-kubernetes	kubernetes-v1.28.2-cni-v1.3.0-critools-v1.28.0-cri-dockerd-v0.3.4-1-debian	1d7e8203bd09	3 months ago	430MB
registry.k8s.io/kube-apiserver	v1.28.2	cdcad12b2dd1	3 months ago	126MB
registry.k8s.io/kube-scheduler	v1.28.2	7a5d9d9fa13f	3 months ago	68.1MB
registry.k8s.io/kube-controller-manager	v1.28.2	55f13c92defb	3 months ago	122MB
registry.k8s.io/kube-proxy	v1.28.2	c128fed2be08	3 months ago	73.1MB
registry.k8s.io/etcd	3.5.9-0	72d6d9a3f792	7 months ago	294MB
docker/desktop-vpnkit-controller	dc31cb22858be8cdd97c84a9cfecaf44a1af6e	55d698875b3d	7 months ago	36.2MB
registry.k8s.io/coredns/coredns	v1.10.1	ead8a4a53df8	11 months ago	53.6MB
registry.k8s.io/etcd	3.5.7-0	88b6a7f6d652	11 months ago	296MB
docker/getting-started	latest	3e439ef6072f	12 months ago	47MB
registry.k8s.io/pause	3.9	e6f181688397	15 months ago	744KB
kiamol/ch15-cert-generator	latest	f4b13f6ed177	18 months ago	55.2MB
kiamol/ch06-pi	latest	6c4e4fcd4ee5	18 months ago	183MB
sixeyed/hollywood	latest	a5f8a5c6a789	18 months ago	524MB
docker/desktop-storage-provisioner	v2.0	99f89d71f470	2 years ago	41.9MB
openjdk	0-jre-alpine	f7a292b0078c	4 years ago	84.9MB

```
PS E:\Users\Achilleas\Desktop> docker image ls 'labs/*'
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
labs/base	latest	1eb4281fffe5e	4 weeks ago	77.9MB

```
PS E:\Users\Achilleas\Desktop>
```

Fig. 18. Έξοδος εντολών καταγραφής

Αυτές είναι οι εικόνες που είναι αποθηκευμένες στην τοπική μνήμη του Docker engine.

Η νέα βασική εικόνα δεν προσθέτει τίποτα στην επίσημη εικόνα του Ubuntu, η οποία είναι διαθέσιμη σε πολλές διαφορετικές εκδόσεις.

- Τραβήξτε την κύρια εικόνα του Ubuntu και, στη συνέχεια, τραβήξτε την εικόνα για την έκδοση 22.04 του Ubuntu:

```
docker pull ubuntu
```

```
# image versions are set in the tag name:
docker pull ubuntu:22.04
```

- Καταχωρίστε όλες τις εικόνες σας στο Ubuntu και τη δική σας βασική εικόνα:

```
docker image ls --filter reference=ubuntu --filter reference=
labs/base
```

Θα δείτε ότι όλα έχουν το ίδιο ID - στην πραγματικότητα όλες είναι παραλλαγές για μια μεμονωμένη εικόνα.

#### – Εντολές και σημεία εισόδου

Η σύνταξη ενός Dockerfile είναι απλή στην εκμάθηση:

- κάθε εικόνα πρέπει να έχει ως βάση μια άλλη εικόνα (FROM),

- χρησιμοποιείτε το RUN για να εκτελέσετε εντολές ως μέρος της κατασκευής,
  - και το CMD ορίζει την εντολή που εκτελείται κατά την εκκίνηση του δοχείου
- Ακολουθεί ένα απλό παράδειγμα που εγκαθιστά το εργαλείο curl.

- Δημιουργήστε μια εικόνα που ονομάζεται labs/curl από το ακόλουθο Dockerfile:

```
# write the base image config in a Dockerfile and save it in
a folder called curl

# Alpine is a tiny Linux distro
FROM alpine

# apk is the package manager for Alpine
# this installs curl into the container image
RUN apk add curl

# tells Docker to run curl when it starts a container
CMD curl

#build image
docker build -t labs/curl ./curl
```

```
PS E:\Users\Achilleas\Desktop> docker build -t labs/curl ./curl
[+] Building 5.6s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 250B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/alpine:latest
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [1/2] FROM docker.io/library/alpine@sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
=> => resolve docker.io/library/alpine@sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
=> => sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48 1.64kB / 1.64kB
=> => sha256:13b7e62e8df88264dbb747995705a986aa530415763a6c58f84a3ca8af9a5bcd 528B / 528B
=> => sha256:f8c20f8bbcb64055b4fea470fdd169c86e87786940b3262335b12ec3adef418 1.47kB / 1.47kB
=> => sha256:661ff4d9561e3fd050929ee5097067c34baf523ee60f5294a37fd08056a73ca 3.41MB / 3.41MB
=> => extracting sha256:661ff4d9561e3fd050929ee5097067c34baf523ee60f5294a37fd08056a73ca
=> [2/2] RUN apk add curl
=> => exporting to image
=> => exporting layers
=> => writing image sha256:da0f7a695b772d2ccad8d86606195155986bbdcbe3dcb782ce58f6bc68bdaf6e
=> => naming to docker.io/labs/curl

View build details: docker-desktop://dashboard/build/default/default/jeenobghvgnwvvl0j2g47zscg

What's Next?
View a summary of image vulnerabilities and recommendations -> docker scout quickview
PS E:\Users\Achilleas\Desktop>
```

Fig. 19. Έξοδος εντολής docker build -t labs/curl ./curl

- Τώρα μπορείτε να εκτελέσετε ένα δοχείο από την εικόνα, αλλά μπορεί να μην συμπεριφέρεται όπως περιμένετε:

```
# just runs curl:
docker run labs/curl
```



```

PS E:\Users\Achilleas\Desktop> docker build -t labs/curl:v2 -f curl/Dockerfile.v2 curl
2024/01/09 16:45:15 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 2.7s (6/6) FINISHED
=> [internal] load build definition from Dockerfile.v2
=> => transferring dockerfile: 279B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/alpine:3.16
=> CACHED [1/2] FROM docker.io/library/alpine:3.16@sha256:e4cdb7d47b66ba0a062ad2a97a7d154967c8f83934594d9f2bd3efa89292996b
=> [2/2] RUN apk add --no-cache curl
=> exporting to image
=> exporting layers
=> writing image sha256:2951ad6600633e3d408f8e0a36837ea7f7c0e7a99ad99fddd130a1749a19588d
=> naming to docker.io/labs/curl:v2

View build details: docker-desktop:///dashboard/build/default/default/7mj7slj2gerwzu0ko1dyhl3s

What's Next?
View a summary of image vulnerabilities and recommendations + docker scout quickview
PS E:\Users\Achilleas\Desktop> |

```

**Fig. 21.** Έξοδος εντολής "docker build -t labs/curl:v2 -f curl/Dockerfile.v2 curl"

- Μπορείτε να εκτελέσετε τώρα δοχεία από αυτήν την εικόνα με πιο λογική σύνταξη:

```
docker run courselabs/curl:v2 --head youtube.com
```

```

PS E:\Users\Achilleas\Desktop> docker run labs/curl:v2 --head youtube.com
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0HTTP/1.1 301 Moved Permanently
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0
Content-Type: application/binary
X-Content-Type-Options: nosniff
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: Mon, 01 Jan 1990 00:00:00 GMT
Date: Tue, 09 Jan 2024 14:53:57 GMT
Location: https://youtube.com/
Content-Length: 0
Server: ESF
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Connection: close

PS E:\Users\Achilleas\Desktop> |

```

**Fig. 22.** Έξοδος εντολής "docker run labs/curl:v2 --head youtube.com"

Η επιλογή -head και η διεύθυνση URL στην εντολή του δοχείου μεταφέρονται στο σημείο εισόδου.

- Καταγράψτε όλες τις labs/curl τις εικόνες για να συγκρίνετε μεγέθη:

```
docker image ls labs/curl
```



```
PS E:\Users\Achilleas\Desktop> docker image ls labs/curl
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
labs/curl     v2        2951ad660063   12 minutes ago 7.78MB
labs/curl     latest    da0f7a695b77   31 minutes ago 14MB
PS E:\Users\Achilleas\Desktop>
```

**Fig. 23.** Έξοδος εντολής "docker image ls labs/curl"

Η εικόνα v2 είναι μικρότερη - πράγμα που σημαίνει ότι έχει λιγότερα πράγματα στο σύστημα αρχείων.

#### – Ιεραρχία εικόνων

Συνήθως δεν χρησιμοποιείτε εικόνες λειτουργικού συστήματος ως βάση στην εικόνα FROM. Θέλετε να έχετε ήδη εγκατεστημένες όσες περισσότερες από τις προϋποθέσεις της εφαρμογής σας γίνεται.

Θα πρέπει να χρησιμοποιείτε **επίσημες εικόνες**, οι οποίες είναι εικόνες εφαρμογής και runtime που διατηρούνται από ομάδες project.

- Αυτό το Dockerfile χρησιμοποιεί κάποιο προσαρμοσμένο περιεχόμενο HTML πάνω από την επίσημη εικόνα Nginx (η διαδρομή φακέλου ονομάζεται context - περιέχει το Dockerfile και τυχόν αρχεία στα οποία αναφέρεται, το αρχείο index.html σε αυτήν την περίπτωση):

```
# write the image cofig in a Dockerfile and save it in a
  folder called web
# also place the index.html file inside the folder

# use a specific version of Nginx:
FROM nginx:1.23.0-alpine

# overwrite the default HTML doc:
COPY index.html /usr/share/nginx/html/

#build image
docker build -t labs/web ./web
```

```

PS E:\Users\Achilleas\Desktop> docker build -t labs/web ./web
[+] Building 4.5s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 172B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:1.23.0-alpine
=> [internal] load build context
=> => transferring context: 169B
=> [1/2] FROM docker.io/library/nginx:1.23.0-alpine@sha256:4a846cc240449c53c8ae24269ba6bcae5167d8ad75cd2a8d8ba422b7c726979
=> => resolve docker.io/library/nginx:1.23.0-alpine@sha256:4a846cc240449c53c8ae24269ba6bcae5167d8ad75cd2a8d8ba422b7c726979
=> => sha256:87a9dea80b01f6b82538e15828ecf9b21a52953ca6ef073913921bc1a7129b11 8.89kB / 8.89kB
=> => sha256:29e4e20b7210a8dd7e779fa527ce217e4ed0514d31d30e6b35f601f65a16a2d 604B / 604B
=> => sha256:4a846cc240449c53c8ae24269ba6bcae5167d8ad75cd2a8d8ba422b7c726979 1.65kB / 1.65kB
=> => sha256:1d08e644bcaf116218f25250ec8a85022d3035a8b8dca2745199f86bb3cf85d1 1.57kB / 1.57kB
=> => sha256:530afca65e2ea04227638ae746e0c85b2bd1a179379cbf2b6501b49c4cab2ccc 2.80MB / 2.80MB
=> => sha256:6a89def8063152acad33aa7ebalcd2d9c5388fa6fee6d3a6e7c8bed1ee0546aa 7.38MB / 7.38MB
=> => extracting sha256:530afca65e2ea04227638ae746e0c85b2bd1a179379cbf2b6501b49c4cab2ccc
=> => sha256:2b101aad7024d10982d455bf1138b39fd1c0fb5f1460fa2350a16b231076afd 665B / 665B
=> => sha256:0b3cca595de12677cd14cc43d1fcf9832bf1263e771b66ba782b06240ad15589 894B / 894B
=> => sha256:f3c5ffbedfb2b0855073e67368f620507671052390d09d70174dece1e8c514ud 1.39kB / 1.39kB
=> => extracting sha256:6a89def8063152acad33aa7ebalcd2d9c5388fa6fee6d3a6e7c8bed1ee0546aa
=> => extracting sha256:29e4e20b7210a8dd7e779fa527ce217e4ed0514d31d30e6b35f601f65a16a2d
=> => extracting sha256:0b3cca595de12677cd14cc43d1fcf9832bf1263e771b66ba782b06240ad15589
=> => extracting sha256:2b101aad7024d10982d455bf1138b39fd1c0fb5f1460fa2350a16b231076afd
=> => extracting sha256:f3c5ffbedfb2b0855073e67368f620507671052390d09d70174dece1e8c514ud
=> [2/2] COPY index.html /usr/share/nginx/html/
=> => exporting to image
=> => exporting layers
=> => writing image sha256:42c1698b1eb518d4f8a52b5d745999d6bfcf9798ff4c6e159b7e6a149cad7785
=> => naming to docker.io/labs/web

View build details: docker-desktop:///dashboard/build/default/default/s8ut955azukku49gs6cgei9gy

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Users\Achilleas\Desktop>

```

Fig. 24. Εξοδος εντολής "docker build -t labs/web ./web"

- Εκτελέστε ένα δοχείο από τη νέα σας εικόνα, δημοσιεύοντας τη θύρα 80, και περιηγηθείτε σε αυτό.

```

# use any free local port, e.g. 8090:
docker run -d -p 8090:80 labs/web

curl localhost:8090

```

**Fig. 25.** Ιστοσελίδα

Το δοχείο εμφανίζει το HTML έγγραφό σας, χρησιμοποιώντας τη ρύθμιση Nginx που έχει διαμορφωθεί στην επίσημη εικόνα.

### – Εργαστηριακή άσκηση 3

Σειρά σας να γράψετε ένα Dockerfile.

Υπάρχει μια απλή εφαρμογή Java σε αυτόν τον [φάκελο](#) που έχει ήδη μεταγλωττιστεί στο αρχείο java/HelloWorld.class.

Δημιουργήστε μια εικόνα Docker που συσκευάζει αυτήν την εφαρμογή και εκτελέστε ένα δοχείο για να επιβεβαιώσετε ότι λειτουργεί. Η εντολή που πρέπει να εκτελέσει το δοχείο σας είναι η java HelloWorld.

### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## Χρήση μητρώων εικόνων

### – Πρόσβαση σε εικόνες από μητρώα

Τα μητρώα είναι διακομιστές που αποθηκεύουν εικόνες Docker. Το Docker Hub είναι το πιο δημοφιλές, αλλά το API μητρώου είναι αμετάβλητο και μπορείτε να εκτελέσετε το δικό σας μητρώο στο cloud ή τοπικά ως το ιδιωτικό σας κατάστημα εικόνων.

Οι οργανισμοί διαχειρίζονται τα δικά τους μητρώα, ώστε να μπορούν να έχουν ένα μακροπρόθεσμο κατάστημα εκδόσεων από το build pipeline ή να κάνουν διαθέσιμες εικόνες στην ίδια περιοχή δικτύου με το περιβάλλον παραγωγής.

Αναφορές:

- [Pulling images](#)
- [Pushing images](#)
- [Registry API spec](#)
- [Docker Registry](#) - για να εκτελέσετε του δικό σας μητρώο σε ένα δοχείο :)

Χρησιμοποιείτε εντολές "image" για εργασία με μητρώα. Οι πιο δημοφιλείς εντολές έχουν παραλλαγές:

```
docker image --help
docker pull --help
docker push --help
docker tag --help
```

#### – Pushing images

Οποιοσδήποτε εικόνες δημιουργείτε αποθηκεύονται μόνο στον υπολογιστή σας.

Για να κάνετε κοινή χρήση εικόνων, πρέπει να τις προωθήσετε σε ένα μητρώο - όπως το Docker Hub. Για το Docker Hub το όνομα της εικόνας πρέπει να περιλαμβάνει το όνομα χρήστη σας, το οποίο χρησιμοποιεί το Docker Hub για να προσδιορίσει την ιδιοκτησία.

- Βεβαιωθείτε ότι έχετε εγγραφεί στο Docker Hub. Στη συνέχεια, αποθηκεύστε το Docker ID σας σε μια μεταβλητή, ώστε να μπορούμε να το χρησιμοποιήσουμε σε μεταγενέστερες εντολές:

```
# on Linux or macOS:
dockerId='<your-docker-hub-id>'

# OR with PowerShell:
$dockerId='<your-docker-hub-id>'
```

Αυτό είναι το όνομα χρήστη του Hub και όχι η διεύθυνση email σας. Για το δικό μου χρησιμοποιώ: \$dockerId='ailleas7'

- Τώρα μπορείτε να δημιουργήσετε μια εικόνα, συμπεριλαμβανομένου του Docker ID σας στο όνομα:

```
docker build -t ${dockerId}/curl:21.06 -f curl/Dockerfile.v2
curl

docker image ls '*/curl'
```

```
PS E:\Users\Achilleas\Desktop> docker build -t ${dockerId}/curl:21.06 -f curl/Dockerfile.v2 curl
[+] Building 0.8s (6/6) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile.v2         0.0s
=> => transferring dockerfile: 279B                             0.0s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [internal] load metadata for docker.io/library/alpine:3.16  0.6s
=> [1/2] FROM docker.io/library/alpine:3.16@sha256:e4c0b7d47b06ba0a062ad2a97a7d154967c8f83934594d9f2bd3efa892929 0.0s
=> CACHED [2/2] RUN apk add --no-cache curl                    0.0s
=> exporting to image                                          0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:2951ad6600633e3d408f8e0a36837ea7f7c0e7a99ad99fddd130a1749a195884 0.0s
=> => naming to docker.io/axilleas7/curl:21.06                 0.0s

View build details: docker-desktop://dashboard/build/default/default/t24mgkr6ec6i9vt9bvwiifgxo

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS E:\Users\Achilleas\Desktop> docker image ls '*/curl'
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
axilleas7/curl      21.06       2951ad660063  6 hours ago    7.78MB
PS E:\Users\Achilleas\Desktop> |
```

Fig. 26. Έξοδος εντολών

- Τώρα σπρώξτε τη δική σας εικόνα curl:21.06 στο Docker Hub.

```
# log in if you haven't already:
docker login -u ${dockerId}

# push your image:
docker push ${dockerId}/curl:21.06
```

```
PS E:\Users\Achilleas\Desktop> docker login -u ${dockerId}
Password:
Login Succeeded
PS E:\Users\Achilleas\Desktop> docker push ${dockerId}/curl:21.06
The push refers to repository [docker.io/axilleas7/curl]
be5d135611dc: Pushed
e6f74d769c02: Mounted from library/alpine
21.06: digest: sha256:b949b3f81ec2108e69d6bf967fd833cc07e48e6a8345fda9dbdbbe6525d1bac8 size: 738
PS E:\Users\Achilleas\Desktop>
```

Fig. 27. Έξοδος εντολών για ανέβασμα εικόνας στο Docker hub

- Οι εικόνες του Docker Hub είναι δημόσια διαθέσιμες (μπορείτε επίσης να δημιουργήσετε ιδιωτικές εικόνες). Εκτελέστε αυτήν την εντολή και περιηγηθείτε στην εικόνα σας στο Docker Hub:

```
echo "https://hub.docker.com/r/${dockerId}/curl/tags"
```

### – Ετικέτες και αναφορές

Τα ονόματα εικόνων (που ονομάζονται σωστά αναφορές) αποτελούνται από τρία μέρη:

- το domain του μητρώου του κοντέινερ
- το όνομα του αποθετηρίου - το οποίο προσδιορίζει την εφαρμογή και τον ιδιοκτήτη
- η ετικέτα - η οποία μπορεί να είναι οτιδήποτε, αλλά συνήθως χρησιμοποιείται για την έκδοση

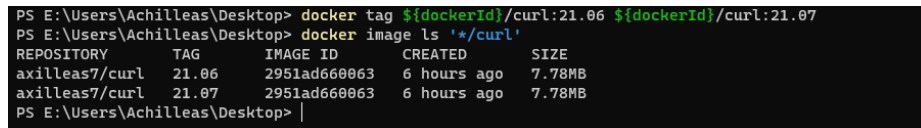
Μια μεμονωμένη εικόνα μπορεί να έχει πολλαπλές αναφορές, οι οποίες είναι σαν παραλλαγές - δεν είναι αντίγραφα της εικόνας, είναι διαφορετικά ονόματα για την ίδια εικόνα.

Οι ετικέτες χρησιμοποιούνται συνήθως για την έκδοση, επομένως μπορείτε να υποθέσετε ότι το sixeyed/curl:20.09 και το sixeyed/curl:21.06 είναι διαφορετικές εκδόσεις της ίδιας εφαρμογής, που δημοσιεύονται από τον ίδιο κάτοχο.

- Add a new reference for your image using the tag command:

```
docker tag ${dockerId}/curl:21.06 ${dockerId}/curl:21.07

docker image ls '*/curl'
```



```
PS E:\Users\Achilleas\Desktop> docker tag ${dockerId}/curl:21.06 ${dockerId}/curl:21.07
PS E:\Users\Achilleas\Desktop> docker image ls '*/curl'
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
axilleas7/curl	21.06	2951ad660063	6 hours ago	7.78MB
axilleas7/curl	21.07	2951ad660063	6 hours ago	7.78MB

```
PS E:\Users\Achilleas\Desktop> |
```

**Fig. 28.** Έξοδος εντολών για αλλαγή tag

Τώρα έχετε πολλές εικόνες curl, αλλά όλες οι παραλλαγές έχουν το ίδιο αναγνωριστικό εικόνας

- Σπρώξτε όλες τις ετικέτες εικόνων curl στο Docker Hub:

```
# you can push individual tags:
docker push ${dockerId}/curl:21.07

# or all local tags for the repository:
docker push --all-tags ${dockerId}/curl
```

```
PS E:\Users\Achilleas\Desktop> docker push ${dockerId}/curl:21.07
The push refers to repository [docker.io/axilleas7/curl]
be5d135611dc: Layer already exists
e6f74d769c02: Layer already exists
21.07: digest: sha256:b949b3f81ec2108e69d6bf967fd833cc07e48e6a8345fda9dbdbb6525d1bac8 size: 738
PS E:\Users\Achilleas\Desktop> docker push --all-tags ${dockerId}/curl
The push refers to repository [docker.io/axilleas7/curl]
be5d135611dc: Layer already exists
e6f74d769c02: Layer already exists
21.06: digest: sha256:b949b3f81ec2108e69d6bf967fd833cc07e48e6a8345fda9dbdbb6525d1bac8 size: 738
be5d135611dc: Layer already exists
e6f74d769c02: Layer already exists
21.07: digest: sha256:b949b3f81ec2108e69d6bf967fd833cc07e48e6a8345fda9dbdbb6525d1bac8 size: 738
PS E:\Users\Achilleas\Desktop>
```

**Fig. 29.** Έξοδος εντολών για ανέβασμα στο Docker hub

θα δείτε πολλά Layer already exists - τα μητρώα έχουν την ίδια προσέγγιση προσωρινής αποθήκευσης επιπέδου με το Docker Engine.

– Εκτέλεση τοπικού μητρώου

Οι εικόνες Docker για πραγματικές εφαρμογές μπορεί να είναι μεγάλες - εκατοντάδες megabyte ή ακόμα και gigabyte. Η λήψη μεγάλων εικόνων προσθέτει στον χρόνο εκκίνησης των κοντείνερ, επομένως οι οργανισμοί συνήθως εκτελούν το δικό τους μητρώο στο cloud ή στο κέντρο δεδομένων.

Όλα τα μητρώα λειτουργούν με τον ίδιο τρόπο, επομένως είναι σχεδόν εναλλάξιμα.

Θα χρησιμοποιούσατε το [Azure Container Registry](#) ή το [Amazon Elastic Container Registry](#) εάν εκτελούσατε σε Azure ή AWS.

- Σε ένα τοπικό περιβάλλον, μπορείτε να φιλοξενήσετε το δικό σας μητρώο, που εκτελείται σε ένα δοχείο, χρησιμοποιώντας την επίσημη εικόνα του [Μητρώου Docker](#):

```
docker run -d -p 5000:5000 --name registry registry:2.7.1
```

```
docker logs registry
```

[illegible]

**Fig. 30.** Έξοδος εντολών για τοπικό μητρώο

Ο τομέας του τοπικού μητρώου σας είναι ο localhost:5000, ώστε να μπορείτε να τον συμπεριλάβετε στις αναφορές εικόνας για να προωθήσετε και να τραβήξετε τοπικά.

```
[registry-domain]/[repository-name]:[tag]
```

- Προσθέστε ετικέτα στην εικόνα Alpine και σπρώξτε την στο τοπικό σας μητρώο:

```
# the tag command creates an alias, which can include the
# registry domain:
docker tag alpine localhost:5000/alpine

# pushing a tag with a domain in the reference tells Docker
# which registry to use:
docker push localhost:5000/alpine
```

Το Docker Desktop έχει ρυθμιστεί ώστε να επιτρέπει μητρώα localhost, αλλά δεν το έχουν όλες οι ρυθμίσεις του Docker. Εάν λάβετε ένα σφάλμα κατά την ώθηση, πρέπει να επιτρέψετε τα [insecure Registries](#).

- Το μητρώο ανοιχτού κώδικα δεν διαθέτει διεπαφή ιστού όπως το Docker Hub, αλλά μπορείτε να εργαστείτε μαζί του χρησιμοποιώντας το REST API:

```
curl --head localhost:5000/v2/

curl localhost:5000/v2/alpine/tags/list
```

Κάθε εικόνα έχει μια ετικέτα - αν δεν την παρέχετε, το Docker χρησιμοποιεί μια προεπιλογή.

- Μπορείτε να αφαιρέσετε την τοπική σας εικόνα και να την τραβήξετε ξανά από το μητρώο

```
docker image rm localhost:5000/alpine

docker pull localhost:5000/alpine
```

```
PS E:\Users\Achilleas\Desktop> docker image rm localhost:5000/alpine
Untagged: localhost:5000/alpine:latest
Untagged: localhost:5000/alpine@sha256:13b7e62e8df80264dbb747995705a986aa530415763a6c58f84a3ca8af9a5bcd
PS E:\Users\Achilleas\Desktop> docker pull localhost:5000/alpine
Using default tag: latest
latest: Pulling from alpine
Digest: sha256:13b7e62e8df80264dbb747995705a986aa530415763a6c58f84a3ca8af9a5bcd
Status: Downloaded newer image for localhost:5000/alpine:latest
localhost:5000/alpine:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview localhost:5000/alpine
PS E:\Users\Achilleas\Desktop> |
```

**Fig. 31.** Έξοδος εντολών



- Δημιουργήστε και σπρώξτε μερικές ακόμη ετικέτες για την εικόνα Alpine - ελέγξτε ότι είναι διαθέσιμες στο μητρώο.

```
# you can use any string in the image tag:
docker tag alpine localhost:5000/alpine:21.07

docker tag alpine localhost:5000/alpine:local

docker push --all-tags localhost:5000/alpine

curl localhost:5000/v2/alpine/tags/list
```

Θα πρέπει να δείτε να επιστραφούν η προεπιλεγμένη ετικέτα και οι νέες ετικέτες σας.

#### – Εργαστηριακή άσκηση 4

Οι αναφορές εικόνων έχουν αρκετές λεπτομέρειες ώστε το Docker να βρει το μητρώο και να προσδιορίσει μια συγκεκριμένη έκδοση της εφαρμογής.

Το Docker χρησιμοποιεί προεπιλογές για το μητρώο και την ετικέτα. Ποιες είναι αυτές οι προεπιλογές; Ποια είναι η πλήρης αναφορά για την εικόνα kiamol/ch05-pi;

Δεν είναι όλες οι επίσημες εικόνες στο Docker Hub. Η Microsoft χρησιμοποιεί το δικό της μητρώο εικόνων MCR στη διεύθυνση [mcr.microsoft.com](https://mcr.microsoft.com). Ποια εντολή θα χρησιμοποιούσατε για να τραβήξετε την έκδοση 6.0 της εικόνας dotnet/runtime από το MCR;

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## 2.2 Εφαρμογές πολλαπλών δοχείων

### Docker Compose

#### – Docker Compose

Το Docker Compose είναι μια προδιαγραφή για την περιγραφή εφαρμογών που εκτελούνται σε δοχεία και ένα εργαλείο γραμμής εντολών που λαμβάνει αυτές τις προδιαγραφές και τις εκτελεί στο Docker.

Είναι μια προσέγγιση επιθυμητής κατάστασης, όπου μοντελοποιείτε τις εφαρμογές σας σε YAML και η γραμμή εντολών Compose δημιουργεί ή αντικαθιστά στοιχεία για να φτάσετε στην επιθυμητή κατάσταση.

Αναφορές:

- [Docker Compose manual](#)
- [Compose specification - GitHub](#)
- [Docker Compose v3 syntax](#)

Το αρχικό Docker Compose CLI είναι ένα ξεχωριστό εργαλείο:

```
docker-compose --help

docker-compose up --help
```

Οι πιο πρόσφατες εκδόσεις του Docker έχουν ενσωματωμένη την εντολή Compose. Οι εντολές είναι ίδιες, μείον την παύλα, οπότε το docker-compose γίνεται docker compose:

```
docker compose --help

docker compose up --help
```

Αυτή είναι μια νέα λειτουργικότητα και δεν είναι 100% συμβατή με το αρχικό Compose CLI. Για αυτό το εργαστήριο θα πρέπει να μπορείτε να χρησιμοποιήσετε ένα από τα δύο, αλλά αν έχετε προβλήματα, συνεχίστε με το docker-compose.

#### – Εφαρμογές πολλαπλών δοχείων

Το Docker μπορεί να εκτελέσει οποιοδήποτε είδος εφαρμογής - η εικόνα του δοχείου θα μπορούσε να είναι για μια ελαφριά μικρουπηρεσία ή ένα παλαιού τύπου monolith. Όλα λειτουργούν με τον ίδιο τρόπο, αλλά τα δοχεία είναι ιδιαίτερα κατάλληλα για κατανεμημένες εφαρμογές, όπου κάθε στοιχείο εκτελείται σε ξεχωριστό δοχείο.

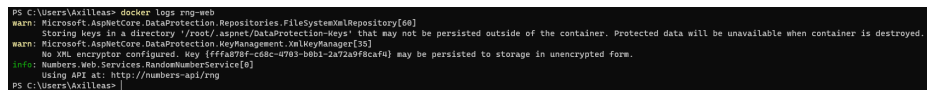
- Δοκιμάστε να εκτελέσετε μια εφαρμογή - μια γεννήτρια τυχαίων αριθμών:

```
docker run -d -p 8088:80 --name rng-web courselabs/rng-web
:21.05
```

Περιηγηθείτε στο <http://localhost:8088> και προσπαθήστε να αιτηθείτε έναν τυχαίο αριθμό. Μετά από λίγα δευτερόλεπτα θα αποτύχει και θα εμφανιστεί το μήνυμα σφάλματος "RNG service unavailable!".

- Ελέγξτε τα logs της εφαρμογής για να δείτε τι συμβαίνει:

```
docker logs rng-web
```



```
PS C:\Users\Avileas> docker logs rng-web
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[0]
      Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key (ffffa78f-c6dc-4703-b0b1-2a72a9f8cafd) may be persisted to storage in unencrypted form.
info: Numbers.Api.Services.RandomNumberService[0]
      Using API at: http://numbers-api/rng
PS C:\Users\Avileas>
```

Fig. 32. Έξοδος εντολής "docker logs rng-web"

Η εφαρμογή Ιστού είναι απλώς η διεπαφή, προσπαθεί να βρει μια υπηρεσία API υποστήριξης στη διεύθυνση `http://numbers-api/rng` - αλλά δεν εκτελείται τέτοια υπηρεσία.

Θα μπορούσατε να ξεκινήσετε ένα δοχείο API με την εντολή "docker run", αλλά θα πρέπει να γνωρίζετε το όνομα της εικόνας, τις θύρες που θα χρησιμοποιήσετε και τη ρύθμιση δικτύου ώστε τα δοχεία να επικοινωνούν μεταξύ τους.

Αντίθετα, μπορείτε να χρησιμοποιήσετε το Docker Compose για να μοντελοποιήσετε και τα δύο δοχεία.

### – Εφαρμογή Compose

Το Docker Compose μπορεί να ορίσει τις υπηρεσίες της εφαρμογής σας - οι οποίες εκτελούνται σε δοχεία - και τα δίκτυα που ενώνουν τα δοχεία μεταξύ τους.

Μπορείτε να χρησιμοποιήσετε το Compose ακόμη και για απλές εφαρμογές - αυτό απλώς ορίζει ένα κοντέινερ Nginx:

```
# save it in a folder called compose and name it docker-
compose.yml

services:
  nginx:
    image: nginx:1.21-alpine
    ports:
      - "8082:80"
```

Γιατί να μπείτε στον κόπο να το βάλετε σε ένα αρχείο Compose; Καθορίζει μια έκδοση εικόνας και τις θύρες που θα χρησιμοποιηθούν. λειτουργεί ως το documentation του project σας, καθώς και ως εκτελέσιμη προδιαγραφή για την εφαρμογή.

- Το Docker Compose έχει τη δική του γραμμή εντολών - αυτή η εντολή σας λέει τις διαθέσιμες εντολές:

```
docker-compose
```

- εκτελέστε αυτήν την εφαρμογή χρησιμοποιώντας το docker-compose CLI:

```
# run 'up' to start the app, pointing to the Compose file
docker-compose -f ./compose/docker-compose.yml up
```



```
image: courselabs/rng-web:21.05
environment:
  - Logging__LogLevel__Default=Debug
ports:
  - "8090:80"
networks:
  - app-net

networks:
  app-net:

# run the app:
docker-compose -f ./compose/v1.yml up -d

# use compose to show just this app's containers:
docker-compose -f ./compose/v1.yml ps

# and this app's logs:
docker-compose -f ./compose/v1.yml logs
```

[illegible]

**Fig. 34.** Έξοδος εντολών "docker-compose"

Αυτά είναι απλώς τυπικά δοχεία - το Compose CLI στέλνει εντολές στο Docker Engine με τον ίδιο τρόπο που κάνει το συνηθισμένο Docker CLI.

- Μπορείτε επίσης να διαχειριστείτε δοχεία που έχουν δημιουργηθεί με το Compose χρησιμοποιώντας το Docker CLI:

```
docker ps
```

- Περιηγηθείτε στη νέα εφαρμογή Ιστού στη διεύθυνση <http://localhost:8090> και προσπαθήστε να βρείτε έναν τυχαίο αριθμό.

Εξακολουθεί να μη δουλεύει! Υποθέτω ότι πρέπει να κάνουμε debug στην εφαρμογή Ιστού.

### – Διαχείριση εφαρμογών με το Compose

Η εφαρμογή Ιστού χρησιμοποιεί το API για τη λήψη τυχαίων αριθμών. Υπάρχουν μόνο δύο λόγοι για τους οποίους μπορεί να αποτύχει:

- το API δεν λειτουργεί
- Η εφαρμογή Ιστού δεν μπορεί να συνδεθεί στο API.

Το API δημοσιεύει μια θύρα, ώστε να μπορούμε να ελέγξουμε τη λειτουργία της ανεξάρτητα.

- Βρείτε τη δημοσιευμένη θύρα για το API και περιηγηθείτε στο /rng endpoint.

```
# the API is listening on port 8089 - you can see that in the
# Compose file or use the CLI:
docker-compose -f ./compose/v1.yml port rng-api 80

curl localhost:8089/rng
```

Θα πρέπει να δείτε έναν τυχαίο αριθμό πίσω, οπότε το API λειτουργεί σωστά.

Φαίνεται ότι η εφαρμογή Ιστού δεν συνδέεται με το API. Το δοχείο Ιστού είναι συσκευασμένο με το εργαλείο nslookup το οποίο μπορούμε να χρησιμοποιήσουμε για να ελέγξουμε το DNS και να δούμε εάν ο τομέας API είναι προσβάσιμος.

- Ελέγξτε τα logs του δοχείου Ιστού για να δείτε τη διεύθυνση API που χρησιμοποιεί και χρησιμοποιήστε το nslookup για να λάβετε τη διεύθυνση IP αυτού του τομέα:

```
docker logs compose-rng-web-1

# the web app is using the domain 'numbers-api'

# run the nslookup command in the container:
docker exec compose-rng-web-1 nslookup numbers-api
```

```
PS E:\Users\Achilleas\Desktop> docker logs compose-rng-web-1
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
      Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be unavailable when container is destroyed.
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key (a6e1dd5-2ab0-4f6a-a837-8b0cda231cc) may be persisted to storage in unencrypted form.
info: Numbers.Web.Services.RandomNumberService[0]
      Using API at: http://numbers-api/rng
PS E:\Users\Achilleas\Desktop> docker exec compose-rng-web-1 nslookup numbers-api
Server:      129.0.0.11
Address:     129.0.0.11:53
.; connection timed out; no servers could be reached
PS E:\Users\Achilleas\Desktop>
```

Fig. 35. Έξοδος εντολών

Υπάρχει ένα σφάλμα DNS - ο τομέας API δεν είναι προσβάσιμος.

Το όνομα μιας υπηρεσίας στο αρχείο Compose γίνεται το ονόμα DNS που μπορούν να χρησιμοποιήσουν τα δοχεία για να έχουν πρόσβαση σε αυτήν την υπηρεσία. Το όνομα της υπηρεσίας API είναι RNG-API, όχι Numbers-API.

Θα μπορούσαμε να αλλάξουμε την υπηρεσία API στο αρχείο Compose, αλλά η εφαρμογή Ιστού υποστηρίζει μια ρύθμιση διαμόρφωσης για τη διεύθυνση URL του API (ορίζει την τιμή config και αυξάνει επίσης το επίπεδο καταγραφής για το API):

```
# save it in a folder called compose and name it v2.yml

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
    ports:
      - "8089:80"
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://rng-api/rng
    ports:
      - "8090:80"
    networks:
      - app-net

networks:
  app-net:
```

Εδώ θα δούμε την προσέγγιση της επιθυμητής κατάστασης. Εάν πρέπει να αλλάξετε την αίτησή σας, αλλάζετε το YAML και τρέχετε ξανά. Το Compose κοιτάζει τι τρέχει και τι ζητάτε να τρέξει και κάνει ό,τι αλλαγές χρειάζεται.

- Αναπτύξτε τα ενημερωμένα spec compose στον φάκελο /compose/v2.yml και χρησιμοποιήστε το Compose για να παρακολουθήσετε όλα τα logs των δοχείων.

```
docker-compose -f ./compose/v2.yml up -d

docker-compose -f ./compose/v2.yml logs -f
```

```

PS E:\Users\Achilleas\Desktop> docker-compose -f ./compose/v2.yml up -d
[*] Running ./v2
Container compose-rng-api-1  Running 0.0s
Container compose-rng-web-1  Started 0.5s
PS E:\Users\Achilleas\Desktop> docker-compose -f ./compose/v2.yml logs -f
rng-web-1 warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
rng-web-1 warn: Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be unavailable when container is destroyed.
rng-web-1 warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
rng-web-1 warn: No XML encryptor configured. Key (6060123-9139-4bde-b4de-dc7f12ab377) may be persisted to storage in unencrypted form.
rng-api-1 info: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Random number generator initialized
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 1. Returning random number: 97, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 2. Returning random number: 92, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 3. Returning random number: 56, from min: 0, max: 100
rng-web-1 info: Numbers.Web.Services.RandomNumbersService[0]
rng-web-1 debug: Using API at: http://rng-api/rng
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 4. Returning random number: 63, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 5. Returning random number: 35, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 6. Returning random number: 16, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 7. Returning random number: 45, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 8. Returning random number: 0, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 9. Returning random number: 11, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 10. Returning random number: 66, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 11. Returning random number: 22, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 12. Returning random number: 35, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 13. Returning random number: 50, from min: 0, max: 100
rng-api-1 debug: Numbers.Api.Controllers.RngController[0]
rng-api-1 debug: Call: 14. Returning random number: 87, from min: 0, max: 100

```

Fig. 36. Έξοδος εντολών

Θα δείτε το δοχείο ιστού να αναδημιουργείται επειδή οι προδιαγραφές έχουν αλλάξει. Δοκιμάστε την εφαρμογή τώρα στη διεύθυνση <http://localhost:8090> και θα λειτουργήσει - και θα δείτε τα logs της εφαρμογής από το Compose.

### – Εργαστηριακή άσκηση 5

Το Compose χρησιμοποιείται για τον καθορισμό εφαρμογών που εκτείνονται σε πολλά δοχεία, αλλά οι υπηρεσίες σχετίζονται μόνο μέσω δικτύων δοχείων.

Προσθέστε ένα δοχείο Nginx και ένα άλλο δίκτυο στον ορισμό της εφαρμογής RNG. Διαμορφώστε την υπηρεσία Nginx στο νέο δίκτυο και στο αρχικό δίκτυο.

Αναπτύξτε το ενημερωμένο spec. Τι διευθύνσεις IP έχει το δοχείο Nginx; Μπορείτε να συνδέσετε τα δοχεία μεταξύ τους - είναι το δοχείο Nginx προσβάσιμο από το διαδικτυακό δοχεία RNG, παρόλο που δημιουργήθηκε εκ των υστέρων;

### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## Μοντελοποίηση εφαρμογών με Compose

### – Μοντελοποίηση εφαρμογών με Compose

Μπορείτε να σκεφτείτε το Compose ως τεκμηρίωση που αντικαθιστά την εκτέλεση docker - όλες οι επιλογές που θα βάζατε στις εντολές εκτέλεσης καθορίζονται στο αρχείο Compose που γίνεται ζωντανή τεκμηρίωση. Αυτό περιλαμβάνει τη ρύθμιση του περιβάλλοντος δοχείου με μεταβλητές και τόμους.



Το Compose περιλαμβάνει επίσης μοντελοποίηση υψηλότερου επιπέδου για αρχεία διαμόρφωσης, χρησιμοποιώντας secret objects. Αυτό βοηθά να καταστεί σαφές ότι αναπτύσσετε τη διαμόρφωση παραμέτρων αντί για τον αποθηκευτικό χώρο.

Τα μοντέλα Compose μπορούν να χωριστούν σε πολλά αρχεία, γεγονός που καθιστά εύκολη την εκτέλεση της ίδιας εφαρμογής σε διαφορετικές διαμορφώσεις σε ένα μόνο μηχάνημα.

Αναφορές:

- [Environment variables in Compose](#)
- [Compose secrets](#)
- [Merging multiple Compose files](#)

Το Docker Compose CLI είναι το ίδιο:

```
docker-compose --help
```

Το κείμενο βοήθειας δεν είναι εξαιρετικά σαφές, αλλά μπορείτε να χρησιμοποιήσετε την επιλογή -f πολλές φορές για να δημιουργήσετε ένα μοντέλο από πολλά αρχεία YAML.

#### – Διαμόρφωση μοντελοποίησης στο Compose

Η διαμόρφωση της εφαρμογής εφαρμόζεται σε δοχεία με μεταβλητές περιβάλλοντος και προσαρτήσεις συστήματος αρχείων. Το Compose υποστηρίζει τις ίδιες επιλογές με την τυπική εντολή docker run:

- Εκτελέστε την προδιαγραφή v1 της προηγούμενης ενότητας και δοκιμάστε την εφαρμογή. στη συνέχεια εκτελέστε την προδιαγραφή v2 - τι πιστεύετε ότι θα συμβεί με το δοχείο v1;

```
# save it in a folder called compose (delete the previous one
) and name it v1.yml
services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
    ports:
      - "8089:80"
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://rng-api/rng
    ports:
      - "8090:80"
```

```

    networks:
      - app-net

networks:
  app-net:

# save it in a folder called compose and name it v2.yml

services:
  rng-api:
    image: courselabs/rng-api:21.05
    env_file: ./config/logging.env
    ports:
      - "8089:80"
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    env_file: ./config/logging.env
    environment:
      - RngApi__Url=http://rng-api/rng
    ports:
      - "8090:80"
    networks:
      - app-net

networks:
  app-net:

```

Η αναβάθμιση δεν αλλάζει τίποτα - τα αρχεία περιβάλλοντος επεκτείνονται από το CLI και οι τιμές χρησιμοποιούνται για τον ορισμό μεμονωμένων μεταβλητών περιβάλλοντος στα δοχεία.

- Εκτυπώστε τις λεπτομέρειες της μεταβλητής περιβάλλοντος από το νέο δοχείο ιστού για να δείτε ότι τα περιεχόμενα του αρχείου env έχουν επεκταθεί:

```
docker container inspect --format='{{ .Config.Env }}' compose-rng-web-1
```

Οι μεταβλητές περιβάλλοντος είναι ένας συνηθισμένος τρόπος ρύθμισης παραμέτρων εφαρμογής, αλλά δεν είναι τόσο ευέλικτες όσο τα αρχεία διαμόρφωσης.

#### – Volume mounts και secrets

- Το Compose σας επιτρέπει να μοντελοποιείτε προσαρτήσεις τόμου, ώστε να μπορείτε να φορτώνετε τοπικά αρχεία στο σύστημα αρχείων του δοχείου. Αυτό χρησιμοποιεί την ίδια προσέγγιση με την εκτέλεση docker - καθορίζοντας μια διαδρομή πηγής και στόχου. Το Compose έχει επίσης ένα abstraction που ονομάζεται Secrets:

```

# save it in a folder called compose and name it v3.yml

# uses a volumes to load the logging config file, and secrets
  for the api config and web config files

# logging config file:
{
  "Logging" : {
    "LogLevel" : {
      "Default" : "Debug"
    }
  }
}

# api config file:
{
  "Rng" : {
    "Range": {
      "Min": 0,
      "Max": 50
    }
  }
}

# web config file:
{
  "RngApi" : {
    "Url" : "http://rng-api/rng"
  }
}

services:
  rng-api:
    image: courselabs/rng-api:21.05
    volumes:
      - ./config/dev/logging.json:/app/config/logging.json
    secrets:
      - source: rng-api
        target: /app/config/override.json
    ports:
      - "8089:80"
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    volumes:

```

```

- ./config/dev/logging.json:/app/config/logging.json
secrets:
- source: rng-web
  target: /app/config/override.json
ports:
- "8090:80"
networks:
- app-net

networks:
  app-net:

secrets:
  rng-api:
    file: ./config/dev/api/override.json
  rng-web:
    file: ./config/dev/web/override.json

```

Πλατφόρμες όπως το Kubernetes χρησιμοποιούν παρόμοιο μηχανισμό για τα secrets, αλλά η πηγή των δεδομένων είναι ένα αντικείμενο που είναι αποθηκευμένο στο Kubernetes, όχι ένα τοπικό αρχείο στον διακομιστή.

- Εκτελέστε την προδιαγραφή v3 και δείτε τη διαμόρφωση του τόμου για το δοχείο Ιστού. Πώς αντιμετωπίζονται διαφορετικά οι τόμοι και τα μυστικά;

```

# deploy v3
docker-compose -f compose/v3.yml up -d

docker inspect compose-rng-web-1

```

Οι τόμοι και τα secrets έχουν οριστεί ως bind mounts.. Η προδιαγραφή τόμου είναι προεπιλεγμένη για εγγραφή ("RW": true), αλλά τα secrets είναι μόνο για ανάγνωση.

Θα χρειαστεί να κατανοήσετε τις διαφορετικές επιλογές για τη διαμόρφωση των ρυθμίσεων της εφαρμογής, επειδή όλες οι εφαρμογές τείνουν να έχουν διαφορετικές ιδέες σχετικά με τη διαμόρφωση.

#### – Εκτέλεση πολλαπλών περιβαλλόντων

Όλα τα δοχεία και τα δίκτυα για την εφαρμογή τυχαίων αριθμών έχουν δημιουργηθεί με το πρόθεμα rng. Το Compose χρησιμοποιεί ένα όνομα project για να αναγνωρίσει όλα τα αντικείμενα που δημιουργεί και ορίζει από προεπιλογή το όνομα του φακέλου στον οποίο βρίσκονται τα αρχεία YAML.

Μπορείτε να καθορίσετε ένα προσαρμοσμένο όνομα project και να αναπτύξετε ξανά τον ίδιο ορισμό μοντέλου για να δημιουργήσετε ένα ξεχωριστό αντίγραφο

της εφαρμογής σας.

- Αναπτύξτε την προδιαγραφή v3 με το προσαρμοσμένο όνομα project rng-dev - ξεκινά σωστά; Αναπτύξτε την προδιαγραφή v3 με το προσαρμοσμένο όνομα έργου rng-dev - ξεκινά σωστά;

```
# use the -p flag to set a project name:
docker-compose -p rng-dev -f compose/v3.yml up -d

# this will fail because ports cannot be used more than once
```

Το Compose δημιουργεί δοχεία με το νέο πρόθεμα rng-dev, αλλά αποτυγχάνουν να ξεκινήσουν - προσπαθούν να χρησιμοποιήσουν θύρες που χρησιμοποιούνται ήδη από τα δοχεία rng.

Εδώ μπαίνουν τα [αρχεία παράκαμψης](#) - μπορείτε να χωρίσετε το μοντέλο σας σε πολλά αρχεία Compose, χρησιμοποιώντας διαφορετικές ρυθμίσεις διαμόρφωσης για διαφορετικά περιβάλλοντα:

```
1) save it in a folder called compose and name it core.yml

# specifies the core application model, with settings which
  are the same for all environments. This is not a complete
  model, so you can't deploy this YAML on its own

services:
  rng-api:
    image: courselabs/rng-api:21.05
    secrets:
      - source: dotnet-logging
        target: /app/config/logging.json
      - source: rng-api
        target: /app/config/override.json
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    secrets:
      - source: dotnet-logging
        target: /app/config/logging.json
      - source: rng-web
        target: /app/config/override.json
    networks:
      - app-net

networks:
  app-net:
```

2) save it in a folder called compose and name it dev.yml

```
# sets the ports and configuration file sources for the dev
environment
```

```
services:
  rng-api:
    image: courselabs/rng-api:21.05
    secrets:
      - source: dotnet-logging
        target: /app/config/logging.json
      - source: rng-api
        target: /app/config/override.json
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    secrets:
      - source: dotnet-logging
        target: /app/config/logging.json
      - source: rng-web
        target: /app/config/override.json
    networks:
      - app-net
```

```
networks:
  app-net:
```

3) save it in a folder called compose and name it test.yml

```
# sets the ports and configuration file sources for the test
environment
```

```
services:
  rng-api:
    ports:
      - "8289:80"

  rng-web:
    ports:
      - "8290:80"

secrets:
  dotnet-logging:
    file: ./config/test/logging.json
  rng-api:
    file: ./config/test/api/override.json
  rng-web:
```

```
file: ./config/test/web/override.json
```

- Εκτελέστε την εφαρμογή στη διαμόρφωση dev συγχωνεύοντας τα αρχεία Compose και χρησιμοποιώντας το όνομα project rng-dev:

```
# use -p to set a custom project name, and join files
# starting
# with the core spec and then adding the dev override:
docker-compose -p rng-dev -f compose/core.yml -f compose/dev.
yml up -d
```

Συγχωνεύετε πολλαπλά αρχεία Compose με την επιλογή -f - τα αρχεία στα δεξιά προσθέτουν ή παρακάμπτουν ρυθμίσεις από τα αρχεία προς τα αριστερά.

Δοκιμάστε την εφαρμογή ιστού στη διεύθυνση <http://localhost:8190>, θα δείτε τυχαίους αριθμούς στην περιοχή 0-50.

- Ελέγξτε τα logs API για το περιβάλλον προγραμματισμού και επιβεβαιώστε ότι εκτυπώνονται:

```
# the project name isn't enough for Compose to find the
# container:
docker-compose -p rng-dev logs rng-api

# you need to include the project name and all the override
# files in any Compose commands:
docker-compose -p rng-dev -f compose/core.yml -f compose/dev.
yml logs rng-api
```

Ένα μειονέκτημα των παρακάμψεων και των ονομάτων project είναι ότι πρέπει να τα παρέχετε με την ίδια σειρά για κάθε εντολή Compose.

Τώρα έχετε δύο εκδόσεις της εφαρμογής που εκτελούνται. Τα κοντέινερ χρησιμοποιούν πολύ λίγη υπολογιστική ισχύ εκτός εάν είναι υπό φορτίο, επομένως το Compose είναι ιδανικό για την εκτέλεση πολλών μη παραγωγικών περιβαλλόντων σε ένα μόνο μηχάνημα.

- Αναπτύξτε την εφαρμογή στη διαμόρφωση test. Δοκιμάστε το - σε τι διαφέρει από το dev;

```
# make sure to use a new project name and the correct files:
docker-compose -p rng-test -f compose/core.yml -f compose/
test.yml up -d

# try the app at http://localhost:8290
```

```
# print the API logs:
docker-compose -p rng-test -f compose/rng/core.yml -f compose
/test.yml logs rng-api
```

Οι τυχαίοι αριθμοί βρίσκονται στην περιοχή 0-5000 και τα αρχεία καταγραφής API μειώνονται σε επίπεδο πληροφοριών στη διαμόρφωση δοκιμής.

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

### Δημιουργία εφαρμογών με το Compose

#### – Δημιουργία εφαρμογών με το Compose

Τα μοντέλα Docker Compose μπορούν να περιλαμβάνουν λεπτομέρειες χρόνου κατασκευής, ώστε να μπορείτε να δημιουργήσετε πολλές εικόνες δοχείων χρησιμοποιώντας μία εντολή Docker Compose. Η χρήση αρχείων παράκαμψης σας επιτρέπει να προσαρμόσετε τις αναφορές εικόνας και να ορίσετε ετικέτες για να παρακολουθείτε την εικόνα πίσω στον πηγαίο κώδικα.

Αναφορές:

- [Compose build spec](#)
- [docker-compose push --help](#)

Το Docker Compose έχει εντολές για να συνεργαστεί με εικόνες:

```
docker-compose build --help
```

```
docker-compose push --help
```

Αυτές υποστηρίζουν πολλαπλά αρχεία YAML με τον ίδιο τρόπο όπως και οι άλλες εντολές.

#### – Ανάπτυξε με το Compose

Αρχείο Compose:

```
# save it in a folder called compose-build and name it docker
-compose.yml

# it includes the context and Dockerfile paths

services:
  rng-api:
    image: rng-api:21.05-local
    networks:
```



```

    - app-net
  build:
    context: .
    dockerfile: docker/api/Dockerfile

rng-web:
  image: rng-web:21.05-local
  environment:
    - RngApi__Url=http://rng-api/rng
  ports:
    - 8090:80
  networks:
    - app-net
  build:
    context: .
    dockerfile: docker/web/Dockerfile

networks:
  app-net:

#create two folder in compose-build to save the Dockerfiles
#docker/web

FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS builder
ARG BUILD_VERSION=1.0.0

WORKDIR /src
COPY src/Numbers.Web/Numbers.Web.csproj .
RUN dotnet restore

COPY src/Numbers.Web/ .
RUN dotnet publish -c Release /p:Version=$BUILD_VERSION -
o /out Numbers.Web.csproj

# app image
FROM mcr.microsoft.com/dotnet/aspnet:6.0-alpine

ENV RngApi__Url=http://numbers-api/rng

ENTRYPOINT ["dotnet", "/app/Numbers.Web.dll"]

WORKDIR /app
COPY --from=builder /out/ .

ARG BUILD_TAG=local
ARG COMMIT_SHA=local
LABEL build_tag=${BUILD_TAG}
LABEL commit_sha=${COMMIT_SHA}

#docker/api

```

```

FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS builder
ARG BUILD_VERSION=1.0.0

WORKDIR /src
COPY src/Numbers.Api/Numbers.Api.csproj .
RUN dotnet restore

COPY src/Numbers.Api/ .
RUN dotnet publish -c Release /p:Version=$BUILD_VERSION -
o /out Numbers.Api.csproj

# app image
FROM mcr.microsoft.com/dotnet/aspnet:6.0-alpine

ENTRYPOINT ["dotnet", "/app/Numbers.Api.dll"]

WORKDIR /app
COPY --from=builder /out/ .

ARG BUILD_TAG=local
ARG COMMIT_SHA=local
LABEL build_tag=${BUILD_TAG}
LABEL commit_sha=${COMMIT_SHA}

```

- Μεταβείτε στο φάκελο `compose-build` και δημιουργήστε τις εικόνες:

```

cd compose-build

docker-compose -f docker-compose.yml build

```

Η εντολή `build` δημιουργεί όλες τις εικόνες με μια ενότητα κατασκευής στην προδιαγραφή. Θα δείτε την έξοδο από τη διαμορφωμένη μηχανή κατασκευής σας - μπορείτε να χρησιμοποιήσετε το πρωτότυπο ή το BuildKit για αυτό το εργαστήριο.

Αυτές οι εικόνες έχουν την ετικέτα `21.05-local`. Μπορείτε να χρησιμοποιήσετε το ίδιο αρχείο Compose για να εκτελέσετε την εφαρμογή από τις τοπικές σας εικόνες.

- Εκτελέστε την εφαρμογή χρησιμοποιώντας τις νέες σας εικόνες και δοκιμάστε ότι λειτουργεί:

```

docker-compose -f docker-compose.yml up -d

# try the app at http://localhost:8090

```

Η προδιαγραφή Compose έχει όλες τις λεπτομέρειες για την εκτέλεση και τη δημιουργία της εφαρμογής.

– **Δημιουργήστε arguments και ετικέτες εικόνων**

Ένα μεμονωμένο αρχείο Compose για τη δημιουργία και την εκτέλεση της εφαρμογής σας είναι πολύ ελκυστικό, αλλά οι επιλογές εκτέλεσης και δημιουργίας είναι πολύ διαφορετικές και είναι συνήθως πιο εύκολο να τις χωρίσετε για να διατηρήσετε τις προδιαγραφές Compose ευκολότερες στην ανάγνωση και τη διατήρηση:

```
# save it in a folder called compose-build and name it core.
  yml

# defines the core services and networks for the random
  number app

services:
  rng-api:
    image: rng-api:21.05
    networks:
      - app-net

  rng-web:
    image: rng-web:21.05
    environment:
      - RngApi_Url=http://rng-api/rng
    networks:
      - app-net

networks:
  app-net:

# save it in a folder called compose-build and name it build.
  yml

# defines the build options for the services, including the
  path to the build context and the path to the Dockerfile

services:
  rng-api:
    build:
      context: .
      dockerfile: docker/api/Dockerfile

  rng-web:
    build:
      context: .
      dockerfile: docker/web/Dockerfile
```

Και με κάποιες πρόσθετες ρυθμίσεις, μπορείτε να προσθέσετε χρήσιμο έλεγχο στις εικόνες σας:

```
#the rng API Dockerfile uses ARG instructions - which are
#values you can set as build arguments - to add metadata
#to the image, using labels to record the build version
#and Git commit ID

FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS builder
ARG BUILD_VERSION=1.0.0

WORKDIR /src
COPY src/Numbers.Api/Numbers.Api.csproj .
RUN dotnet restore

COPY src/Numbers.Api/ .
RUN dotnet publish -c Release /p:Version=$BUILD_VERSION -
o /out Numbers.Api.csproj

# app image
FROM mcr.microsoft.com/dotnet/aspnet:6.0-alpine

ENTRYPOINT ["dotnet", "/app/Numbers.Api.dll"]

WORKDIR /app
COPY --from=builder /out/ .

ARG BUILD_TAG=local
ARG COMMIT_SHA=local
LABEL build_tag=${BUILD_TAG}
LABEL commit_sha=${COMMIT_SHA}

# save it in a folder called compose-build and name it args.
# yml

# overrides the image name and sets default values for the
# build arguments. All the ${VARIABLES} can be overridden
# by environment variables on the machine running the build
# .

services:
  rng-api:
    image: ${REPOSITORY:-courselabs}/rng-api:${RELEASE
:-21.05}-${BUILD_NUMBER:-0}
    build:
      args:
        BUILD_VERSION: ${RELEASE:-21.05}.${BUILD_NUMBER:-0}
        BUILD_TAG: ${GITHUB_WORKFLOW:-local}-${BUILD_NUMBER
:-0}-${GITHUB_REF:-local}
        COMMIT_SHA: ${GITHUB_SHA:-local}
```

```

rng-web:
  image: ${REPOSITORY:-courselabs}/rng-web:${RELEASE}
        :-21.05}-${BUILD_NUMBER:-0}
  build:
    args:
      BUILD_VERSION: ${RELEASE:-21.05}.${BUILD_NUMBER:-0}
      BUILD_TAG: ${GITHUB_WORKFLOW:-local}-${BUILD_NUMBER}
                :-0}-${GITHUB_REF:-local}
      COMMIT_SHA: ${GITHUB_SHA:-local}

```

- Ενώστε όλα αυτά τα αρχεία για να δημιουργήσετε την εφαρμογή και, στη συνέχεια, ελέγξτε τις ετικέτες για την εικόνα API.

```

# join all the files to get the full build spec:
docker-compose -f core.yml -f build.yml -f args.yml build

# this output shows label values:
docker image inspect --format '{{ courselabs/rng-api:21.05-0 }}'

```

Μπορείτε να ορίσετε μεταβλητές περιβάλλοντος στο μηχανήμά σας που θα αντικαταστήσουν τις προεπιλογές στα αρχεία Compose:

- Ενώστε όλα αυτά τα αρχεία για να δημιουργήσετε την εφαρμογή και, στη συνέχεια, ελέγξτε τις ετικέτες για την εικόνα API.

```

# macOS or Linux:
export RELEASE=2021.07
export BUILD_NUMBER=121

# OR with PowerShell:
$env:RELEASE='2021.07'
$env:BUILD_NUMBER='121'

```

- Επαναλάβετε την κατασκευή. Ποιες είναι οι νέες ετικέτες εικόνας; Και οι τιμές της ετικέτας στην εικόνα του API;

```

# it's the same set of files:
docker-compose -f core.yml -f build.yml -f args.yml build

# the tag is 2024.01-125

# show the new label values:
docker image inspect --format '{{ courselabs/rng-api }}'
:2024.01-125

```

Διαφορετικές τιμές ορίσματος κατασκευής διαγράφουν τη μνήμη cache για αυτήν την έκδοση, επομένως θα δείτε τις οδηγίες Dockerfile να εκτελούνται ξανά.

Αυτές οι τιμές ετικετών δεν είναι πολύ χρήσιμες όταν δημιουργείτε τοπικά - αλλά σε ένα Continuous Integration build, θα ορίζονται με τις σωστές τιμές από την υπηρεσία κατασκευής.

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

### Περιορισμοί του Compose

#### – Περιορισμοί του Compose

Το Docker Compose είναι ένα εργαλείο από την πλευρά του πελάτη - σας δίνει έναν εύκολο τρόπο να μοντελοποιήσετε και να αναπτύξετε εφαρμογές πολλαπλών δοχείων, αλλά δεν είναι ενορχηστρωτής δοχείων.

Ο κύριος περιορισμός του Compose είναι ότι εκτελείται σε έναν μόνο διακομιστή. Δεν υπάρχει τρόπος να διαχειριστείτε πολλούς διακομιστές με το Compose, επομένως δεν μπορείτε να κάνετε τις εφαρμογές σας πλήρως επεκτάσιμες και αξιόπιστες.

Αναφορές:

- [Compose in production](#)
- [Understanding container orchestration - YouTube](#)

#### – Προσθήκη αξιοπιστίας

Οι εφαρμογές αποτυγχάνουν και αυτό μπορεί να προκαλέσει τον τερματισμό των δοχείων. Κατά την παραγωγή, θα θέλατε η πλατφόρμα σας να δει ότι το δοχείο έχει τερματίσει και να ξεκινήσει μια αντικατάσταση, αλλά αυτή δεν είναι η προεπιλεγμένη συμπεριφορά στο Compose.

```
# save it in a folder called compose-limits and name it v1.
  yml

# is the same random number app we've used already, but with
  a configuration which causes the API to fail after 3
  calls

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Rng__FailAfter__Enabled=true
      - Rng__FailAfter__CallCount=3
```

```

    - Rng__FailAfter__Action=Exit
  networks:
    - app-net

rng-web:
  image: courselabs/rng-web:21.05
  environment:
    - Logging__LogLevel__Default=Debug
    - RngApi__Url=http://rng-api/rng
  ports:
    - "8088:80"
  networks:
    - app-net

networks:
  app-net:

```

- Εκτελέστε την εφαρμογή από το αρχείο `compose-limits/v1.yml`:

```
docker-compose -f compose-limits/v1.yml up -d
```

Θα δείτε τα δύο δοχεία να ξεκινούν. Περιηγηθείτε στην εφαρμογή στη διεύθυνση `http://localhost:8088` - κάντε κλικ στο κουμπί 3 φορές και θα λάβετε έναν τυχαίο αριθμό. Μετά από αυτό αποτυγχάνει και θα δείτε το `RNG service unavailable!` μήνυμα λάθους.

- Κοιτάξτε τα API logs και, στη συνέχεια, ελέγξτε την κατάσταση των κοντέινερ:

```
docker logs compose-limits-rng-api-1
```

```
docker ps -a
```

```

PS E:\Users\Achilleas\Desktop> docker logs compose-limits-rng-api-1
[info: Numbers.Api.Controllers.RngController[0]
      Random number generator initialized
[error: Numbers.Api.Controllers.RngController[0]
[error: FailAfter enabled. Call: 4. Exiting.
PS E:\Users\Achilleas\Desktop> docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS
f4c71877dc99   courselabs/rng-api:21.05            "dotnet /app/Numbers..." 2 minutes ago   Exited (100) About a minute ago
4e4ef64c3141   courselabs/rng-web:21.05            "dotnet /app/Numbers..." 2 minutes ago   Up 2 minutes       0.0.0.0:8088->80/tcp
462696a94bee   axilleas7/myapp-frontend:1.0        "/docker-entrypoint..." 10 days ago    Exited (0) 2 days ago
3aa872339f91   axilleas7/myapp-backend:1.0         "/docker-entrypoint..." 10 days ago    Exited (0) 2 days ago
a458f6602f88   gcr.io/985-minikube/kicbase:v0.0.42 "/usr/local/sbin/entr..." 2 weeks ago    Exited (137) 2 days ago
PS E:\Users\Achilleas\Desktop>

```

Fig. 37. Έξοδος εντολών καταγραφής

Το κοντέινερ API έχει σταματήσει επειδή ο διακομιστής API τερμάτισε.

- Μπορείτε να επανεκκινήσετε ένα τερματισμένο δοχείο με:

```
docker start compose-limits-rng-api-1
```

Το κοντέινερ API έχει σταματήσει επειδή ο διακομιστής API τερμάτισε.

Μπορείτε να διαμορφώσετε τα δοχεία για αυτόματη επανεκκίνηση εάν τερματίσουν:

```
# save it in a folder called compose-limits and name it v2.
  yml

# adds the restart: always setting to both services, so the
  containers restart if the application process exits.

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Rng__FailAfter__Enabled=true
      - Rng__FailAfter__CallCount=3
      - Rng__FailAfter__Action=Exit
    restart: always
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://rng-api/rng
    restart: always
    ports:
      - "8088:80"
    networks:
      - app-net

networks:
  app-net:
```

- Ενημερώστε την ανάπτυξή σας με το αρχείο Compose compose-limits/v2.yml.

```
docker-compose -f compose-limits/v2.yml up -d
```

Και τα δύο δοχεία αναδημιουργούνται. Το παλιό δοχείο ιστού ήταν καλά, αλλά δεν ταιριάζει με το νέο spec.



- Τώρα, όταν χρησιμοποιείτε την εφαρμογή, θα εξακολουθεί να αποτυγχάνει μετά από 3 αριθμούς, αλλά στη συνέχεια το Docker επανεκκινεί το κοντέινερ API. Η εφαρμογή δεν θα λειτουργεί κατά την εκκίνηση του δοχείου, αλλά στη συνέχεια θα λειτουργήσει ξανά:

```
# try the app until it fails, then check the container status
:
docker ps -a
```

Η κατάσταση για το δοχείο API εμφανίζει μια χρονική διαφορά μεταξύ του πό-τε δημιουργήθηκε και του χρόνου λειτουργίας - αυτή είναι η πιο πρόσφατη ώρα έναρξης.

Η επιλογή επανεκκίνησης εκκινεί επίσης τα δοχεία κατά την εκκίνηση του Docker engine. Μπορείτε να επανεκκινήσετε το Docker Desktop και η εφαρμογή σας θα επανέλθει στο διαδίκτυο, αλλά δεν θα είναι διαθέσιμη κατά την εκκίνηση του Docker.

#### – Εκτέλεση σε κλίμακα

- Μπορείτε να αυξήσετε την αξιοπιστία και το μέγεθος του φορτίου που μπορούν να χειριστούν οι εφαρμογές σας εκτελώντας πολλά δοχεία:

```
# save it in a folder called compose-limits and name it v3.
  yml

# adds the scale setting to both services, so Compose will
  try to run 3 API containers and 2 web containers.

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Rng__FailAfter__Enabled=true
      - Rng__FailAfter__CallCount=10
      - Rng__FailAfter__Action=Exit
    restart: always
    scale: 3
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://rng-api/rng
    restart: always
    scale: 2
    ports:
      - "8088:80"
```

```
networks:
  - app-net
```

```
networks:
  app-net:
```

- Αναπτύξτε την ενημέρωση και ελέγξτε την έξοδο από το CLI:

```
docker-compose -f compose-limits/v3.yml up -d
```

Θα δείτε σφάλματα σχετικά με την κατανομή θυρών. Η υπηρεσία Ιστού δημοσιεύει μια θύρα - μόνο ένα δοχείο μπορεί να χρησιμοποιήσει τη συγκεκριμένη θύρα, επομένως δεν μπορείτε να εκτελέσετε δημόσια στοιχεία σε κλίμακα με το Compose.

- Παρουσιάστε τα δοχεία που τρέχουν και μετά όλα τα δοχεία. Λειτουργεί η εφαρμογή;

```
docker ps
```

```
docker ps -a
```

```
PS E:\Users\Achilleas\Desktop> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
95d9e681e09e   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-3
953d040d0eb1b   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-1
f8114da09ea3   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-2
a87d3da2a1f0   courselabs/rng-web:21.05           "dotnet /app/Numbers..." 9 minutes ago  Up 9 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-web-1

PS E:\Users\Achilleas\Desktop> docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
95d9e681e09e   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-3
953d040d0eb1b   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-1
f8114da09ea3   courselabs/rng-api:21.05           "dotnet /app/Numbers..." 3 minutes ago  Up 3 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-api-2
72ac897078c1b   courselabs/rng-web:21.05           "dotnet /app/Numbers..." 3 minutes ago  Created                                compose-limits-rng-web-2
a87d3da2a1f0   courselabs/rng-web:21.05           "dotnet /app/Numbers..." 9 minutes ago  Up 9 minutes  0.0.0.0:8080->80/tcp               compose-limits-rng-web-1
462096a94bee   axilleas7/myapp-frontend:1.0       "/docker-entrypoint..." 10 days ago   Exited (0) 2 days ago                  upbeat_lanarr
3aa872398f01   axilleas7/myapp-backend:1.0        "/docker-entrypoint.s..." 10 days ago   Exited (0) 2 days ago                  strange_shannon
a459f6662f08   pcr.io/80s-minikube/kicbase:v0.0.02 "/usr/local/bin/entr..." 2 weeks ago   Exited (130) 7 minutes ago             minikube
```

Fig. 38. Έξοδος εντολών

Θα δείτε πολλά δοχεία API να εκτελούνται και ένα επιπλέον δοχείο ιστού που έχει κολλήσει στην κατάσταση Created. Η εφαρμογή λειτουργεί και μπορείτε να λάβετε πολύ περισσότερους αριθμούς προτού αποτύχει κατά την επανεκκίνηση των δοχείων API.

Υπάρχουν πολλά δοχεία API, οπότε το Docker προσθέτει όλες τις διευθύνσεις IP τους στην απόκριση DNS. Όταν το δοχείο Ιστού πραγματοποιεί μια κλήση στο όνομα τομέα API, εξισορροπείται μεταξύ των δοχείων API.

- Μπορείτε να δείτε όλες τις IP των δοχείων API στην απόκριση DNS στο δοχείου ιστού:

```
docker exec -it compose-limits-rng-web-1 nslookup rng-api
```

```
PS E:\Users\Achilleas\Desktop> docker exec -it compose-limits-rng-web-1 nslookup rng-api
Server:      127.0.0.11
Address:     127.0.0.11:53

Non-authoritative answer:
Non-authoritative answer:
Name:   rng-api
Address: 192.168.96.3
Name:   rng-api
Address: 192.168.96.5
Name:   rng-api
Address: 192.168.96.4
PS E:\Users\Achilleas\Desktop> |
```

**Fig. 39.** Έξοδος εντολής "docker exec -it compose-limits-rng-web-1 nslookup rng-api"

Θα δείτε πολλά δοχεία API να εκτελούνται και ένα επιπλέον δοχείο ιστού που έχει κολλήσει στην κατάσταση Created. Η εφαρμογή λειτουργεί και μπορείτε να λάβετε πολύ περισσότερους αριθμούς προτού αποτύχει κατά την επανεκκίνηση των δοχείων API.

#### – Κατανομή υπολογιστικών πόρων

Ο τελευταίος περιορισμός με το Compose είναι ότι μπορείτε να χρησιμοποιήσετε μόνο τους διαθέσιμους πόρους σε έναν διακομιστή. Εάν διαθέτετε μια εφαρμογή που απαιτεί πόρους, ενδέχεται να μην μπορείτε να την εκτελέσετε καθόλου:

```
# save it in a folder called compose-limits and name it v4.
  yml

# adds CPU and memory allocations to the containers. It
  requests 4 CPU cores and 8GB memory for each API
  container and 32 cores and 64GB memory for the web
  container.

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Rng__FailAfter__Enabled=true
      - Rng__FailAfter__CallCount=10
      - Rng__FailAfter__Action=Exit
    restart: always
    scale: 3
    cpus: 4
    mem_limit: 8g
```

```

networks:
  - app-net

rng-web:
  image: courselabs/rng-web:21.05
  environment:
    - Logging__LogLevel__Default=Debug
    - RngApi__Url=http://rng-api/rng
  restart: always
  scale: 1
  cpus: 32
  mem_limit: 64g
  ports:
    - "8088:80"
  networks:
    - app-net

networks:
  app-net:

```

- Αναπτύξτε την έκδοση v4 και ελέγξτε τα δοχεία. Λειτουργεί η εφαρμογή;

```

docker-compose -f compose-limits/v4.yml up -d

docker ps -a

```

```

PS E:\Users\Achilleas\Desktop> docker-compose -f compose-limits/v4.yml up -d
time="2024-01-18T16:48:15+02:00" level=warning msg="'scale' is deprecated. Use the 'deploy.replicas' element"
[+] Running 0/0
 - Container compose-limits-rng-web-1   Recreate
 - Container compose-limits-rng-api-3   Recreate
 - Container compose-limits-rng-api-2   Recreate
 - Container compose-limits-rng-api-1   Recreate
Error response from daemon: Range of CPUs is from 0.01 to 16.00, as there are only 16 CPUs available
PS E:\Users\Achilleas\Desktop>

```

**Fig. 40.** Έξοδος εντολής "docker-compose -f compose-limits/v4.yml up -d"

Εάν δεν διαθέτετε ένα μηχάνημα με υψηλές δυνατότητες, θα λάβετε σφάλματα που λένε ότι δεν υπάρχουν αρκετοί διαθέσιμοι πόροι.

Το Compose καταργεί το λειτουργικό δοχείο ιστού για να το αντικαταστήσει και, στη συνέχεια, διαπιστώνει ότι δεν μπορεί να δημιουργήσει το δοχείο αντικατάστασης - έτσι τώρα δεν υπάρχει δοχείο ιστού και η ενημέρωση χάλασε την εφαρμογή σας.

## – Εργαστηριακή Άσκηση 6

Αυτό είναι απλώς ένα εργαστήριο σκέψης :) Έχουμε δει τους περιορισμούς του Docker Compose - ποια πρακτικά προβλήματα θα είχατε αν προσπαθούσατε να χρησιμοποιήσετε το Compose στην παραγωγή;

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## 2.3 Multi-stage builds

### Multi-stage builds

#### – Multi-Stage Builds

Τα Multi-Stage Builds χρησιμοποιούν την τυπική σύνταξη dockerfile, με πολλαπλά στάδια να διαχωρίζονται με εντολές FROM. Σας δίνουν μια επαναλαμβανόμενη κατασκευή με ελάχιστες εξαρτήσεις.

Δεν θα δείτε Multi-Stage Builds να χρησιμοποιούνται παντού, αλλά είναι ένας πολύ καλός τρόπος για να κεντράρετε το σύνολο εργαλείων σας - οι προγραμματιστές και οι διακομιστές κατασκευής χρειάζονται απλώς το Docker και τον πηγαίο κώδικα, όλα τα εργαλεία είναι συσκευασμένα σε εικόνες Docker, έτσι ώστε όλοι να χρησιμοποιούν την ίδια εκδόση.

- [Multi-stage build docs](#)

Χρησιμοποιείται η τυπική εντολή docker build για Multi-Stage Builds. Η σύνταξη Dockerfile χρησιμοποιεί πολλαπλές εντολές FROM. Τα μοτίβα είναι τα ίδια για όλες τις γλώσσες, αλλά οι μεμονωμένες λεπτομέρειες είναι συγκεκριμένες.

Δείγματα στις κύριες γλώσσες:

- [Java app using Maven build](#)
- [Go application using Go modules](#)
- [.NET Core app using Alpine images](#)

#### – Multi-Stage Dockerfiles

Θα ξεκινήσουμε χρησιμοποιώντας το αρχικό Engine κατασκευής, ώστε να είναι ξεκάθαρο τι συμβαίνει στην κατασκευή - αργότερα θα μεταβούμε στο BuildKit που έχει καλύτερη απόδοση:

```
# on macOS or Linux:
export DOCKER_BUILDKIT=0

# OR with PowerShell:
$env:DOCKER_BUILDKIT=0
```

Ακολουθεί ένα απλό Multi-Stage Dockerfile:

- το στάδιο base χρησιμοποιεί Alpine και προσομοιώνει την προσθήκη ορισμένων εξαρτήσεων

- το στάδιο build βασίζεται στο base και προσομοιώνει μια κατασκευή εφαρμογής
- το στάδιο test ξεκινά από την έξοδο του σταδίου build και προσομοιώνει την αυτοματοποιημένη δοκιμή
- το τελικό στάδιο ξεκινά από το base και αντιγράφει την έξοδο του σταδίου build
- Δημιουργήστε μια εικόνα που ονομάζεται simple από το multi-stage/simple/Dockerfile:

```
# just a normal build:
docker build -t simple ./multi-stage/simple/
```

Όλα τα στάδια εκτελούνται, αλλά η τελική εικόνα της εφαρμογής έχει μόνο περιεχόμενο που έχει προστεθεί ρητά από προηγούμενα στάδια.

- Δημιουργήστε μια εικόνα που ονομάζεται simple από το multi-stage/simple/Dockerfile:

```
docker run simple
```

Η τελική εικόνα δεν έχει το πρόσθετο περιεχόμενο από το στάδιο της δοκιμής.

#### – BuildKit and build targets

Το BuildKit είναι μια εναλλακτική μηχανή κατασκευής στο Docker. Είναι πολύ βελτιστοποιημένο για εκδόσεις πολλών σταδίων, παράλληλη εκτέλεση σταδίων και παράλειψη σταδίων εάν δεν χρησιμοποιείται η έξοδος τους.

- Χρησιμοποιήστε στο BuildKit ορίζοντας μια μεταβλητή περιβάλλοντος:

```
# on macOS or Linux:
export DOCKER_BUILDKIT=1

# OR with PowerShell:
$env:DOCKER_BUILDKIT=1
```

- Τώρα επαναλάβετε την κατασκευή για το απλό Dockerfile - αυτή τη φορά το Docker θα χρησιμοποιήσει το BuildKit:

```
docker build -t simple:buildkit ./multi-stage/simple/
```

Θα δείτε αποτελέσματα από διαφορετικά στάδια ταυτόχρονα - και αν κοιτάξετε προσεκτικά, θα δείτε ότι το στάδιο test παραλείπεται.

- Εκτελέστε ένα δοχείο από τη νέα εικόνα. Η έξοδος είναι ίδια; Συγκρίνετε τις λεπτομέρειες της εικόνας:

```
# run a container - the output is the same:
docker run simple:buildkit

# list images - they're the same size but not the same image:
docker image ls simple
```

- Το BuildKit παρακάμπτει το στάδιο test επειδή κανένα από τα αποτελέσματα δεν χρησιμοποιείται σε μεταγενέστερα στάδια. Μπορείτε να δημιουργήσετε ρητά μια εικόνα μέχρι ένα συγκεκριμένο στάδιο με τη επιλογή target (το Docker θα δημιουργήσει όλα τα στάδια μέχρι και το ονομαζόμενο):

```
docker build -t simple:test --target test ./labs/multi-stage/
simple/
```

Αυτή η εικόνα είναι η έξοδος του σταδίου test και όχι του τελικού σταδίου.

- Εκτελέστε ένα δοχείο από τη έκδοση test, εκτυπώνοντας τα περιεχόμενα του αρχείου build.txt.

```
# no output here - the test stage has no CMD instruction
docker run simple:test

# run the cat command to see the output
docker run simple:test cat /build.txt
```

Η έξοδος είναι από το στάδιο build και το στάδιο test.

#### – Απλή Εφαρμογή Go

Τα πραγματικά Multi-Stage Builds χρησιμοποιούν μια εικόνα SDK (Software Development Kit) για τη μεταγλώττιση της εφαρμογής στο στάδιο build και μια μικρότερη εικόνα χρόνου εκτέλεσης (χωρίς εργαλεία κατασκευής) για τη συσκευασία της μεταγλωττισμένης εφαρμογής.

Οι εικόνες που χρησιμοποιείτε και οι εντολές που εκτελείτε είναι διαφορετικές για κάθε γλώσσα, αλλά θα βρείτε [επίσημες εικόνες στο Docker Hub για όλες τις μεγάλες πλατφόρμες](#), όπως:

- [maven](#) και [gradle](#) για εφαρμογές Java - χρησιμοποιούν το openjdk για την εικόνα χρόνου εκτέλεσης
- [python](#) - έχει εγκατεστημένο το Pip
- [node](#) για εφαρμογές Node.js - έχει εγκατεστημένο το NPM ώστε να κατεβάζετε και να τρέχετε πακέτα
- [golang](#) για εφαρμογές Go - δεν χρειάζονται εικόνα χρόνου εκτέλεσης οπότε η τελική εικόνα μπορεί να προέλθει από το [scratch](#)

- [dotnet/sdk](#) για εφαρμογές .NET Core/6, χρησιμοποιώντας [dotnet/runtime](#) ή [dotnet/aspnet](#) για την τελική εικόνα της εφαρμογής.

Δεν θα καλύψουμε λεπτομερώς διαφορετικές γλώσσες. Το [Whoami Dockerfile](#) δείχνει πώς λειτουργεί το μοτίβο, χρησιμοποιώντας μια εφαρμογή Go:

```
# save it in a folder called compose-limits/whoami

FROM golang:1.16.4-alpine AS builder
ENV CGO_ENABLED=0

RUN apk --no-cache --no-progress add \
    ca-certificates \
    git \
    tzdata \
    && update-ca-certificates

WORKDIR /go/whoami
COPY go.mod .
RUN go mod download

COPY app.go .
RUN go build -o /out/whoami

# app
FROM scratch

ENTRYPOINT ["/app/whoami"]
EXPOSE 80

COPY --from=builder /usr/share/zoneinfo /usr/share/zoneinfo
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/
COPY --from=builder /out/whoami /app/
```

- τα στάδια του builder ξεκινά από την εικόνα Go SDK
- εγκαθιστά τα πακέτα λειτουργικού συστήματος που απαιτούνται για τη δημιουργία της εφαρμογής
- στη συνέχεια αντιγράφει τη λίστα της βιβλιοθήκης και εκτελεί το go mod download για να εγκαταστήσει τις εξαρτήσεις της εφαρμογής
- έπειτα αντιγράφει τον πηγαίο κώδικα και μεταγλωττίζει την εφαρμογή
- η τελική εικόνα της εφαρμογής ρυθμίζει το περιβάλλον του δοχείου
- τότε αντιγράφει στο μεταγλωττισμένο την έξοδο από τον builder.
- Δημιουργήστε μια εικόνα που ονομάζεται whoami από το φάκελο multi-stage/whoami:

```
docker build -t whoami ./multi-stage/whoami/
```

Θα δείτε όλη την έξοδο των σταδίων από το BuildKit.



Οι εικόνες SDK είναι συνήθως πολύ μεγάλες, έχοντας ολόκληρο το σύνολο εργαλείων δημιουργίας. Δεν θέλετε να χρησιμοποιήσετε μια εικόνα SDK στο τελικό σας στάδιο, διαφορετικά θα έχετε όλα αυτά τα πράγματα στην εικόνα της εφαρμογής σας.

- Συγκρίνετε τα μεγέθη των εικόνων whoami και golang.

```
docker pull golang:1.16.4-alpine
docker image ls -f reference=whoami -f reference=golang
```

Η εικόνα SDK είναι πάνω από 300 MB. η εικόνα της εφαρμογής είναι κάτω από 10 MB.

- Η εφαρμογή είναι ένας απλός διακομιστής ιστού. Εκτελέστε ένα δοχείο που δημοσιεύει μια τυχαία θύρα και βρείτε τη θύρα:

```
docker run -d -P --name whoami1 whoami
docker port whoami1
```

Η εντολή EXPOSE λέει στο Docker τη θύρα προορισμού για το δοχείο. Όταν χρησιμοποιείτε τη επιλογή -P το Docker δημοσιεύει όλες τις εκτεθειμένες θύρες.

- Τώρα μπορείτε να χρησιμοποιήσετε την εφαρμογή:

```
curl localhost:<port>
```

Ο διακομιστής απλώς εκτυπώνει ορισμένες λεπτομέρειες σχετικά με το περιβάλλον και το αίτημα.

## – Εργαστηριακή Άσκηση 7

Οι εφαρμογές χρειάζονται ειδικά δικαιώματα Linux για ακρόαση στις τυπικές θύρες HTTP - ακόμα και μέσα σε ένα δοχείο.

Η εφαρμογή whoami υποστηρίζει μια επιλογή για τη διαμόρφωση της θύρας στην οποία ακούει, ώστε να μπορείτε να χρησιμοποιήσετε μια μη τυπική θύρα και ενδεχομένως να εκτελείται με αυστηρότερη ασφάλεια.

Ο στόχος σας για αυτό το εργαστήριο είναι να εκτελέσετε την εφαρμογή whoami σε ένα δοχείο - χρησιμοποιώντας το όρισμα εφαρμογής -port για ακρόαση σε μια συγκεκριμένη θύρα. Τι συμβαίνει όταν εκτελείτε ένα δοχείο με την επιλογή -P (-publish-all); Αντιστοιχίζει σωστά το Docker τη νέα θύρα;

Τι πρέπει να κάνετε για να εκτελέσετε ένα δοχείο που λειτουργεί;

### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## Δικτύωση δοχείων

### – Δικτύωση δοχείων

Το Docker διαχειρίζεται τη δικτύωση - τα δοχεία μπορούν να συνδεθούν σε δίκτυα Docker και τα δοχεία στο ίδιο δίκτυο μπορούν να επικοινωνήσουν μεταξύ τους μέσω διεύθυνσης IP.

Το Docker εκτελεί επίσης έναν διακομιστή DNS, επομένως τα δοχεία στο ίδιο δίκτυο μπορούν να βρουν το ένα το άλλο χρησιμοποιώντας το όνομα του δοχείου ως όνομα DNS host.

Αναφορές:

- [Docker networking overview](#)
- Container network options in Compose: [DNS](#), [hostname aliases](#), [IP addresses](#)
- [Network configuration in Compose](#)

Τα δίκτυα είναι ξεχωριστά αντικείμενα, τα οποία μπορείτε να διαχειριστείτε από το Docker CLI:

```
docker network --help
docker network ls
```

ο Docker χρησιμοποιεί ένα [plugin μοντέλο για δικτύωση](#), έτσι ώστε τα δοχεία να μπορούν να μοντελοποιηθούν ώστε να ταιριάζουν με διαφορετικές φυσικές αρχιτεκτονικές δικτύου.

### – Δικτύωση δοχείων

Τα δίκτυα Docker είναι σημαντικά. Τα χρησιμοποιούμε ήδη και το Docker Compose τα δημιουργεί όταν αναπτύσσουμε μια εφαρμογή:

```
# save it in a folder called networking and name it compose.
  yml

# defines a network with no special configuration

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
    networks:
```

```

    - app-net

rng-web:
  image: courselabs/rng-web:21.05
  environment:
    - Logging__LogLevel__Default=Debug
    - RngApi__Url=http://rng-api/rng
  ports:
    - "8090:80"
  networks:
    - app-net

networks:
  app-net:

```

- Εκτελέστε την εφαρμογή, απαριθμήστε όλα τα δίκτυα και εκτυπώστε τις λεπτομέρειες του νέου δικτύου εφαρμογών.

```

docker-compose -f networking/compose.yml up -d

# networks are a top-level object in the Docker CLI:
docker network ls

# compose adds the project name as a prefix to the network
  name:
docker network inspect networking_app-net

```

Θα δείτε πολλές λεπτομέρειες σχετικά με το δίκτυο - συμπεριλαμβανομένων των δοχείων που είναι συνδεδεμένα σε αυτό. Το API και τα δοχεία ιστού υπάρχουν, με διευθύνσεις IP στην ίδια περιοχή δικτύου, επομένως θα πρέπει να μπορούν να συνδεθούν.

Η εικόνα δοχείου Ιστού έχει εγκατεστημένα ορισμένα εργαλεία δικτύωσης, τα οποία μπορούμε να χρησιμοποιήσουμε για την αντιμετώπιση προβλημάτων:

- nslookup για να ρωτήσετε τον διακομιστή DNS και να λάβετε μια διεύθυνση IP
- ping για να δοκιμάσετε τη σύνδεση δικτύου σε απομακρυσμένο μηχάνημα
- wget για να κάνετε αιτήματα HTTP.
- Χρησιμοποιήστε τα για να ελέγξετε τη ρύθμιση του δικτύου:

```

# run a DNS lookup for the API service name:
docker exec networking-rng-web-1 nslookup rng-api

# ping the IP address of the API container :
docker exec networking-rng-web-1 ping -c3 <rng-api-ip>

# try calling the API inside the web container:
docker exec networking-rng-web-1 wget -qO- rng-api/rng

```

Ο διακομιστής Docker DNS επιστρέφει τη διεύθυνση IP για ονόματα δοχείων και παραλλαγές δικτύου. Το δοχείο Ιστού μπορεί να έχει πρόσβαση στο δοχείο API στην τυπική θύρα HTTP 80.

#### – Δημοσίευση θυρών

Τα δοχεία που είναι συνδεδεμένα στο ίδιο δίκτυο μπορούν να έχουν πρόσβαση μεταξύ τους με διεύθυνση IP ή όνομα. Δεν υπάρχει περιορισμός στην κυκλοφορία - όλες οι θύρες μπορεί να χρησιμοποιηθούν μεταξύ των δοχείων, δεν χρειάζεται να δημοσιευθούν.

Η κυκλοφορία εκτός του δικτύου δοχείων είναι περιορισμένη και ο μόνος τρόπος αποστολής της κίνησης σε ένα δοχείο είναι η δημοσίευση θυρών. Το Docker ακούει στη δημοσιευμένη θύρα και στέλνει την κίνηση στη θύρα του δοχείου προορισμού.

- Δοκιμάστε να εκτελέσετε έναν απλό διακομιστή web, δημοσιεύοντας σε μια συγκεκριμένη θύρα

```
docker run -d -p 8080:80 sixeyed/whoami:21.04
curl localhost:8080
```

Οι θύρες είναι πόροι μιας χρήσης, μόνο μία διεργασία μπορεί να τις ακούσει. Εάν επαναλάβετε την εντολή, θα λάβετε ένα μήνυμα αποτυχίας γιατί έχει ήδη εκχωρηθεί.

- Εκτελέστε μερικά ακόμη δοχεία από την ίδια εικόνα, χρησιμοποιώντας διαφορετικές θύρες (οι 8081, 8082 και 8083 θα πρέπει να είναι ελεύθερες).

```
docker run -d -p 8081:80 sixeyed/whoami:21.04
docker run -d -p 8082:80 sixeyed/whoami:21.04
docker run -d -p 8083:80 sixeyed/whoami:21.04
```

- Το port mapping σάς επιτρέπει να ακούσετε σε οποιαδήποτε ελεύθερη θύρα στο μηχάνημά σας, παρόλο που τα δοχεία ακούνε όλα εσωτερικά στη θύρα 80. Ελέγξτε τα δοχεία που εκτελούνται:

```
docker ps
```

- Το καθένα δημοσιεύει σε διαφορετικές θύρες, ώστε να μπορείτε να δείτε την απάντηση από κάθε δοχείο:

```
curl localhost:8081
curl localhost:8082
curl localhost:8083
```

### – Προσαρμογή δικτύων δοχείων

Το Docker ορίζει τη διεύθυνση IP, τον διακομιστή DNS και άλλες επιλογές δικτύου για ένα δοχείο - και μπορείτε επίσης να τα διαμορφώσετε.

- Υπάρχει ένα script το οποίο μπορούμε να προσαρτήσουμε σε ένα δοχείο για να εκτυπώσουμε τις ρυθμίσεις δικτύου - network-info.sh. Αποθηκεύστε τη διαδρομή προς το script σε μια μεταβλητή για να διευκολύνετε την προσάρτηση σε ένα δοχείο:

```
# save it in the folder networking/scripts and name it
network-info.sh

#!/bin/sh

echo ''

echo "*** Hostname ***"
echo $(hostname)
echo ''

echo "*** IP address ***"
echo $(hostname -i)
echo ''

echo "*** DNS server: ***"
cat /etc/resolv.conf | grep nameserver | cut -c 12-
echo ''

# on macOS/Linux:
scriptsPath="${PWD}/networking/scripts"
chmod +x "${PWD}/networking/scripts/network-info.sh"

# OR with PowerShell:
$scriptsPath="${PWD}/networking/scripts"
```

- Τώρα εκτελέστε ένα βασικό δοχείο Alpine για να εκτελέσετε το script:

```
docker run -v ${scriptsPath}:/scripts alpine sh /scripts/
network-info.sh
```

Όλες αυτές οι ρυθμίσεις δικτύου έχουν δημιουργηθεί από το Docker

- Επαναλάβετε την εντολή εκτέλεσης για ένα νέο δοχείο με επιπλέον ρυθμίσεις για να καθορίσετε έναν διακομιστή DNS 1.1.1.1 και hostname alpine1.

```
docker run --dns 1.1.1.1 --hostname alpine1 -v ${scriptsPath}
:/scripts alpine sh /scripts/network-info.sh
```

Αυτό είναι χρήσιμο για εφαρμογές που χρειάζονται συγκεκριμένη διαμόρφωση.

- Μπορείτε επίσης να ορίσετε μια διεύθυνση IP - αλλά πρώτα πρέπει να δημιουργήσετε ένα δίκτυο με ένα συγκεκριμένο εύρος διευθύνσεων IP:

```
docker network create --subnet=10.10.0.0/16 courselabs

docker run --network courselabs --ip 10.10.0.100 -v ${
  scriptsPath}:/scripts alpine sh /scripts/network-info.sh
```

Αν δεν τις διαμορφώσετε, οι διευθύνσεις IP ορίζονται τυχαία από το Docker - θα χρειαστεί να χρησιμοποιήσετε μια προσαρμοσμένη περιοχή υποδικτύου εάν η διεύθυνση IP του Docker συγκρούεται με το δίκτυό σας ή το VPN σας.

- Μπορείτε επίσης να διαμορφώσετε προσαρμοσμένη δικτύωση στο Docker Compose:

```
# save it in a folder called networking and name it compose-
network.yml

# the random number app using custom IP addresses with a
named network

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
    volumes:
      - ./scripts:/scripts
    networks:
      app-net:
        ipv4_address: 10.218.0.10

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://10.218.0.10/rng
    ports:
      - "8090:80"
    networks:
      - app-net

networks:
  app-net:
    name: courselabs-rng
```

```
ipam:
  driver: default
  config:
    - subnet: 10.218.0.0/16
```

- Εκτελέστε την εφαρμογή με το Compose και δοκιμάστε την. Το δοχείο API προσαρτά επίσης το test script δικτύου - εκτελέστε το για να ελέγξετε τη διεύθυνση IP.

```
# update the app:
docker-compose -f networking/compose-network.yml up -d

# try it out at http://localhost:8090

# print the container's network details:
docker exec networking-rng-api-1 sh /scripts/network-info.sh
```

- Ελέγξτε τα logs του δοχείου ιστού και θα δείτε ότι χρησιμοποιεί την προσαρμοσμένη διεύθυνση IP για το δοχείο API:

```
docker logs networking-rng-web-1
```

Δεν χρειάζεται συχνά να προσαρμόζετε τη δικτύωση όπως εδώ - τα δοχεία υποτίθεται ότι εκτελούνται σε ένα δυναμικό περιβάλλον. Αλλά αυτό είναι πολύ χρήσιμο για τη μεταφορά παλαιότερων εφαρμογών σε δοχείο, εάν έχουν σταθερές προσδοκίες για το περιβάλλον.

## – Εργαστηριακή Άσκηση 8

Ο διακομιστής DNS του Docker επιστρέφει τη διεύθυνση IP για ένα δοχείο, χρησιμοποιώντας το όνομα ως όνομα τομέα.

Όταν εκτελείτε εφαρμογές με το Compose, δημιουργεί ονόματα δοχεία όπως networking-rng-web-1 - πώς λοιπόν τα δοχεία ανακαλύπτουν το ένα το άλλο χρησιμοποιώντας το όνομα της υπηρεσίας Compose;

Κλιμακώστε το API τυχαίων αριθμών για την εκτέλεση πολλών δοχείων και ελέγξτε ότι ο ιστότοπος εξισορροπεί τα αιτήματα φόρτωσης σε όλα. Στη συνέχεια, κοιτάξτε τη ρύθμιση του δικτύου των δοχείων για να δείτε πώς λειτουργεί.

## – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker rm -f $(docker ps -aq)
```

## Κατανόηση της ενορχήστρωσης

### – Ενορχήστρωση με τον Docker Swarm

Το [Swarm Mode](#) είναι η πλατφόρμα ενορχήστρωσης που είναι ενσωματωμένη στο Docker Engine. Είναι εύκολο να ξεκινήσετε, επειδή χρησιμοποιεί την προδιαγραφή Docker Compose για τη μοντελοποίηση εφαρμογών.

Το Swarm δεν χρησιμοποιείται πλέον ευρέως, αλλά είναι μια πλήρως εξοπλισμένη πλατφόρμα, επομένως αποτελεί μια καλή εισαγωγή στις έννοιες της ενορχήστρωσης.

Αναφορές:

- [Swarm Mode introduction](#)
- [Key swarm concepts](#)
- [Overlay networks](#)

Η λειτουργία Swarm χρησιμοποιεί το ίδιο Docker CLI, αλλά προσθέτει νέα αντικείμενα για διαχείριση:

```
docker swarm --help
docker node
docker stack
docker service
```

Ένα σύνολο μηχανών σε ένα Swarm ονομάζεται σύμπλεγμα - cluster και κάθε μηχανή είναι ένας κόμβος - node. Μπορείτε να αναπτύξετε εφαρμογές από τις προδιαγραφές του Compose, όπου κάθε εφαρμογή είναι μια στοίβα - stack, η οποία έχει μία ή περισσότερες υπηρεσίες - services.

### – Μετάβαση σε λειτουργία Swarm

- Δεν υπάρχουν επιπλέον εξαρτήσεις για τη λειτουργία Swarm - απλώς εκτελείτε μια εντολή για να αρχικοποιήσετε το Swarm, το οποίο μετατρέπει αυτό το μηχάνημα σε διαχειριστή Swarm:

```
docker swarm init
```

Η έξοδος σάς δίνει ένα token που θα χρησιμοποιήσετε για άλλους διακομιστές για να ενταχθούν στο Swarm ως εργαζόμενοι. Δεν θα προσθέσουμε άλλους, αλλά μπορείτε να εκτελέσετε Swarms με χιλιάδες κόμβους.

- Εκτυπώστε μια λίστα με όλους τους κόμβους στο Swarm.

```
# `node` commands are available on the manager:
docker node ls
```



Αυτή η μία εντολή `init swarm` δημιούργησε όλη την υποδομή για μια κλιμακούμενη πλατφόρμα δοχείων - συμπεριλαμβανομένης μιας ανθεκτικής βάσης δεδομένων που χρησιμοποιεί το `Swarm` για την αποθήκευση μοντέλων εφαρμογών.

#### – Αποθήκευση αρχείων διαμόρφωσης

Η βάση δεδομένων του cluster αναπαράγεται μεταξύ κόμβων διαχειριστή όταν εκτελείτε σε κλίμακα, επομένως τα δεδομένα είναι πάντα διαθέσιμα. Χρησιμοποιείται επίσης για την αποθήκευση στατικών δεδομένων για αρχεία διαμόρφωσης.

Τα [αντικείμενα διαμόρφωσης](#) αποθηκεύουν κάθε είδους δεδομένα, τα οποία μπορείτε να εμφανίσετε σε συστήματα αρχείων δοχείων. Δημιουργείτε ρυθμίσεις παραμέτρων από ένα τοπικό αρχείο και τα δεδομένα αποθηκεύονται στη βάση δεδομένων του cluster.

- Δημιουργήστε ένα αντικείμενο διαμόρφωσης που ονομάζεται `rng-logging` από την τοπική διαδρομή αρχείου `orchestration/config/logging.json`:

```
# create logging.json and save it in the orchestration/
# config folder

{
  "Logging" : {
    "LogLevel" : {
      "Default" : "Information"
    }
  }
}

docker config create rng-logging orchestration/config/logging
.json
```

- Καταχωρίστε όλα τα αντικείμενα διαμόρφωσης και εκτυπώστε τις λεπτομέρειες της νέας διαμόρφωσης `logging`.

```
# config is a top-level object in Swarm mode:
docker config ls

# inspecting with the pretty flag prints the contents:
docker config inspect --pretty rng-logging
```

Τα αντικείμενα διαμόρφωσης αποθηκεύονται σε απλό κείμενο και είναι ορατά από οποιονδήποτε έχει πρόσβαση στον cluster.

Ορισμένες ρυθμίσεις διαμόρφωσης έχουν ευαίσθητες λεπτομέρειες - όπως κωδικούς πρόσβασης ή κλειδιά API. Μπορείτε να τα αποθηκεύσετε στο `swarm`, αλλά κρατήστε τα προστατευμένα χρησιμοποιώντας [secret](#).

Έχουμε ένα αντικείμενο διαμόρφωσης με ρυθμίσεις `logging` και τώρα θα χρησιμοποιήσουμε `secrets` για τα άλλα αρχεία διαμόρφωσης της εφαρμογής τυχαίων αριθμών.

- Δημιουργήστε ένα μυστικό που ονομάζεται `rng-api` από το αρχείο στο `orchestration/config/api.json` και ένα που ονομάζεται `rng-web` από το αρχείο στο `orchestration/config/web.json`:

```
# secrets are top-level objects like configs:
docker secret create rng-api orchestration/config/api.json

docker secret create rng-web orchestration/config/web.json
```

Εργάζεστε με secrets με παρόμοιες εντολές για τη διαμόρφωση αντικειμένων. Η διαφορά είναι ότι είναι κρυπτογραφημένα στη βάση δεδομένων και αποκρυπτογραφούνται μόνο μέσα στο σύστημα αρχείων δοχείων.

- Καταγράψτε τα secret στο Swarm και εκτυπώστε τις λεπτομέρειες του API secret:

```
docker secret ls

docker secret inspect --pretty rng-api
```

Ακόμη και οι διαχειριστές cluster δεν μπορούν να δουν το περιεχόμενο των secrets. Τα config και τα secrets είναι ξεχωριστά από τα δοχεία, επομένως είναι ιδανικά για ροές εργασίας όπου μια ομάδα διαχείρισης διαμόρφωσης δημιουργεί τα αρχεία διαμόρφωσης πριν από την ανάπτυξη της εφαρμογής.

#### – Ανάπτυξη εφαρμογών

Η λειτουργία swarm χρησιμοποιεί την προδιαγραφή Compose για να μοντελοποιήσει εφαρμογές:

```
# save it in a folder called orchestration and name it rng-v1
# .yaml

# it is the random number app, loading the config and secret
# objects into the container filesystem

version: "3.7"

services:
  rng-api:
    image: courselabs/rng-api:21.05
    configs:
      - source: rng-logging
        target: /app/config/logging.json
    secrets:
      - source: rng-api
        target: /app/config/override.json
    ports:
      - "8089:80"
```

```

    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    configs:
      - source: rng-logging
        target: /app/config/logging.json
    secrets:
      - source: rng-web
        target: /app/config/override.json
    ports:
      - "8090:80"
    networks:
      - app-net

networks:
  app-net:

configs:
  rng-logging:
    external: true

secrets:
  rng-api:
    external: true

  rng-web:
    external: true

```

Οι εφαρμογές αναπτύσσονται ως **stacks** στη λειτουργία Swarm. Οι στοίβες είναι ένας μηχανισμός ομαδοποίησης για υπηρεσίες - services (που λειτουργούν ως δοχεία) και δίκτυα (που εκτείνονται σε ολόκληρο το swarm).

- Αναπτύξτε μια στοίβα που ονομάζεται rng από το μοντέλο v1 και απαριθμήστε όλες τις υπηρεσίες.

```

# stack deploy uses a desired-state approach:
docker stack deploy -c orchestration/rng-v1.yml rng

# you'll see services for the API and web components:
docker service ls

```

Δοκιμάστε την εφαρμογή στο <http://localhost:8090>

Θα μπορούσατε να αναπτύξετε αυτήν την ίδια προδιαγραφή Compose σε ένα swarm με 100 κόμβους και θα λειτουργούσε με τον ίδιο τρόπο. Η δικτύωση είναι σε επίπεδο

cluster, επομένως οποιοσδήποτε κόμβος μπορεί να λάβει ένα αίτημα στη θύρα 8090 και να το κατευθύνει σε ένα δοχείο, ακόμα κι αν αυτό εκτελείται σε διαφορετικό κόμβο.

#### – Διαχείριση εφαρμογών

Οι υπηρεσίες είναι μια αφαίρεση στο μοντέλο Compose - με το Docker Compose δημιουργούν δοχεία, αλλά στη λειτουργία swarm οι υπηρεσίες είναι ξεχωριστά αντικείμενα.

- Μπορείτε να χρησιμοποιήσετε υπηρεσίες για τη διαχείριση της εφαρμογής σας:

```
# list the containers running the API service:
docker service ps rng_rng-api

# print the API container logs:
docker service logs rng_rng-api
```

Οι υπηρεσίες είναι η μονάδα κλίμακας - και σε αντίθεση με το Docker Compose, μπορείτε να έχετε πολλά δοχεία που ακούν στην ίδια δημοσιευμένη θύρα. Το swarm φροντίζει να λαμβάνει την κίνηση και να την κατευθύνει σε ένα από τα δοχεία.//

```
# save it in a folder called orchestration and name it rng-v2
.yml

# increases the scale of the app with multiple replicas

version: "3.7"

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - Rng__FailAfter__Enabled=true
      - Rng__FailAfter__CallCount=10
      - Rng__FailAfter__Action=Exit
    configs:
      - source: rng-logging
        target: /app/config/logging.json
    secrets:
      - source: rng-api
        target: /app/config/override.json
    ports:
      - "8089:80"
    networks:
      - app-net
    deploy:
      replicas: 4
```

```

rng-web:
  image: courselabs/rng-web:21.05
  configs:
    - source: rng-logging
      target: /app/config/logging.json
  secrets:
    - source: rng-web
      target: /app/config/override.json
  ports:
    - "8090:80"
  networks:
    - app-net
  deploy:
    replicas: 2

networks:
  app-net:

configs:
  rng-logging:
    external: true

secrets:
  rng-api:
    external: true

  rng-web:
    external: true

```

Η νέα προδιαγραφή ενεργοποιεί επίσης τη λειτουργία αποτυχίας για το API, επομένως πολλαπλές κλήσεις θα έχουν ως αποτέλεσμα την έξοδο από τα δοχεία. Θα το χρησιμοποιήσουμε για να δούμε πώς η λειτουργία swarm διατηρεί την εφαρμογή online.

- Ενημερώστε τη στοίβα στο μοντέλο v2 και βεβαιωθείτε ότι οι υπηρεσίες εκτελούνται σε κλίμακα.

```

# stack deploy is for updates and new releases:
docker stack deploy -c orchestration/rng-v2.yml rng

# you'll see multiple replicas for each service:
docker service ls

```

Η λίστα υπηρεσιών δείχνει τον αριθμό των αντιγράφων - κάθε αντίγραφο είναι ένα δοχείο που εκτελείται σε έναν κόμβο στο Swarm. Με πολλούς κόμβους, τα αντίγραφα θα απλώνονταν σε αυτούς για αξιοπιστία.

Δοκιμάστε την εφαρμογή τώρα, λαμβάνοντας επανειλημμένα νέους τυχαίους αριθμούς. Τα δοχεία API θα αποτύχουν και το swarm θα τα αντικαταστήσει. Η εφαρμογή

θα εξακολουθεί να εμφανίζει αποτυχίες κατά την επανεκκίνηση των δοχείου API, αλλά σε ένα πιο σύνθετο μοντέλο θα προσθέσετε υγειονομικούς ελέγχους για να το αποφύγετε.

#### – Καθάρισμα

Καθαρίστε αφαιρώντας όλα τα δοχεία:

```
docker swarm leave -f
```

## 3 Εργαστήριο κατανόησης Kubernetes

### 3.1 Τα βασικά του Kubernetes

#### Nodes

#### – Εξέταση κόμβων με Kubectl

Το Kubectl είναι η γραμμή εντολών για εργασία με ένα Kubernetes cluster.

Έχει [εντολές](#) για την ανάπτυξη εφαρμογών και την εργασία με αντικείμενα στο cluster.

#### – Εργασία με κόμβους

Δύο από τις πιο κοινές εντολές του Kubectl είναι το "get" and "describe".

- Μπορείτε να τα χρησιμοποιήσετε με διαφορετικά αντικείμενα. Δοκιμάστε να βρείτε πληροφορίες σχετικά με τους κόμβους στο cluster σας:

```
kubectl get nodes
```

Οι κόμβοι είναι οι διακομιστές στο cluster. Η εντολή "get" εκτυπώνει έναν πίνακα με βασικές πληροφορίες.

```
kubectl describe nodes
```

Υπάρχουν πολλές περισσότερες πληροφορίες για να δείτε και το "describe" τις μετατρέπει σε αναγνώσιμη μορφή.

#### – Βοήθεια

Το Kubectl έχει ενσωματωμένη βοήθεια, μπορείτε να τη χρησιμοποιήσετε για να παραθέσετε όλες τις εντολές ή να απαριθμήσετε τις λεπτομέρειες μιας εντολής:

```
kubectl --help
```

```
kubectl get --help
```

Και μπορείτε να μάθετε για τους πόρους ζητώντας από την Kubectl να τους εξηγήσει:

```
kubectl explain node
```

#### – Querying and formatting

Θα περάσετε πολύ χρόνο με την Kubectl. Πρέπει να εξοικειωθείτε με ορισμένες δυνατότητες από νωρίς, όπως το querying.

- Το Kubectl μπορεί να εκτυπώσει πληροφορίες σε διαφορετικές μορφές, δοκιμάστε να εμφανίσετε τις λεπτομέρειες του κόμβου σας σε JSON:

```
kubectl get node <your-node> -o json
```

Ελέγξτε τη βοήθεια για να δείτε ποιες άλλες μορφές εξόδου μπορείτε να χρησιμοποιήσετε.

- Το ένα είναι το [JSON Path](#), το οποίο είναι μια γλώσσα query που μπορείτε να χρησιμοποιήσετε για να εκτυπώσετε συγκεκριμένα πεδία:

```
kubectl get node <your-node> -o jsonpath='{.status.capacity.  
cpu}'
```

Αυτό σας λέει τον αριθμό των πυρήνων CPU που βλέπει το Kubernetes για αυτόν τον κόμβο.

Τι συμβαίνει εάν δοκιμάσετε την ίδια εντολή χωρίς να καθορίσετε όνομα κόμβου;

#### – Εργαστηριακή Άσκηση 9

Κάθε αντικείμενο στο Kubernetes μπορεί να έχει ετικέτες - είναι ζεύγη κλειδιών-τιμών που χρησιμοποιούνται για την καταγραφή πρόσθετων πληροφοριών σχετικά με το αντικείμενο.

Χρησιμοποιήστε το Kubectl για να βρείτε ετικέτες για τον κόμβο σας, οι οποίες θα επιβεβαιώνουν την αρχιτεκτονική της CPU και το λειτουργικό σύστημα που χρησιμοποιεί.

## Pods

#### – Εκτελώντας δοχεία σε pods

Το [Pod](#) είναι η βασική μονάδα υπολογισμού στο Kubernetes. Τα Pods τρέχουν δοχεία - είναι δουλειά τους να διασφαλίσουν ότι το δοχείο συνεχίζει να λειτουργεί.

Οι προδιαγραφές του pod είναι πολύ απλές. Το ελάχιστο YAML χρειάζεται κάποια μεταδεδομένα και το όνομα της εικόνας του δοχείου για να εκτελεστεί.

## – API specs

- Pod

Δεν γίνεται πιο απλά για ένα Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: whoami
spec:
  containers:
    - name: app
      image: sixeyed/whoami:21.04
```

Κάθε πόρος Kubernetes απαιτεί αυτά τα τέσσερα πεδία:

- apiVersion - οι πόροι έχουν εκδοθεί για να υποστηρίζουν συμβατότητα προς τα πίσω
- kind - ο τύπος του αντικειμένου
- metadata - συλλογή πρόσθετων δεδομένων αντικειμένων
- name - το όνομα του αντικειμένου

Η μορφή του πεδίου προδιαγραφών είναι διαφορετική για κάθε τύπο αντικειμένου.

Για τα Pods, αυτό είναι το ελάχιστο που χρειάζεστε:

- containers - λίστα δοχείων για εκτέλεση στο Pod
- name - το όνομα του δοχείου
- image - η εικόνα Docker για εκτέλεση

Η εσοχή είναι σημαντική στο YAML - τα πεδία αντικειμένων είναι ένθετα με κενά.

## – Εκτελώντας ένα απλό Pod

- Το Kubectl είναι το εργαλείο για τη διαχείριση αντικειμένων. Δημιουργείτε οποιοδήποτε αντικείμενο από το YAML χρησιμοποιώντας την εντολή εφαρμογής.

```
# create a folder called pods/specs and save the YAML file
  under the name whoami-pod.yaml

# it is the Pod spec for a simple web app

apiVersion: v1
kind: Pod
metadata:
  name: whoami
spec:
  containers:
    - name: app
      image: sixeyed/whoami:21.04
```

- Αναπτύξτε την εφαρμογή από το τοπικό σας μηχανημα:



```
kubectl apply -f pods/specs/whoami-pod.yaml
```

- Η η διαδρομή προς το αρχείο YAML μπορεί να είναι μια διεύθυνση ιστού:

```
kubectl apply -f https://kubernetes.courselabs.co/labs/pods/specs/whoami-pod.yaml
```

Η έξοδος σάς δείχνει ότι τίποτα δεν έχει αλλάξει. Το Kubernetes λειτουργεί στην επιθυμητή κατάσταση.

- Αναπτύξτε την εφαρμογή από το τοπικό σας μηχανημα:

```
kubectl apply -f pods/specs/whoami-pod.yaml
```

- Τώρα μπορείτε να χρησιμοποιήσετε τις γνωστές εντολές για να εκτυπώσετε πληροφορίες:

```
kubectl get pods
```

```
kubectl get po -o wide
```

Η δεύτερη εντολή χρησιμοποιεί το σύντομο όνομα PO και προσθέτει επιπλέον στήλες, συμπεριλαμβανομένης της διεύθυνσης IP POD

Ποιες επιπλέον πληροφορίες βλέπετε στη δεύτερη έξοδο και πώς θα εκτυπώνετε όλες τις πληροφορίες Pod σε αναγνώσιμη μορφή;

#### – Εργασία με Pods

Σε ένα cluster παραγωγής το Pod θα μπορούσε να λειτουργεί σε οποιονδήποτε node. Το διαχειρίζεστε χρησιμοποιώντας το Kubectl, ώστε να μην χρειάζεστε απευθείας πρόσβαση στον διακομιστή.

- Εκτυπώστε τα logs του δοχείου.

```
kubectl logs whoami
```

- Συνδέθείτε στο δοχείο μέσα στο Pod:

```
# this will fail:
kubectl exec -it whoami -- sh
```

Αυτή η εικόνα δοχείου δεν έχει εγκατεστημένο shell!

- Ας δοκιμάσουμε μια άλλη εφαρμογή:

```
# create a folder called pods/specs and save the YAML file
  under the name sleep-pod.yaml

# it runs an app which does nothing

apiVersion: v1
kind: Pod
metadata:
  name: sleep
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
```

- Αναπτύξτε τη νέα εφαρμογή από το pods/specs/sleep-pod.yaml και ελέγξτε ότι εκτελείται:

```
kubectl apply -f pods/specs/sleep-pod.yaml

kubectl get pods
```

- Αυτό το δοχείο Pod έχει ένα shell και έχει εγκατεστημένα μερικά χρήσιμα εργαλεία.

```
kubectl exec -it sleep -- sh
```

- Τώρα είστε συνδεδεμένοι μέσα στο δοχείο. Μπορείτε να εξερευνήσετε το περιβάλλον του δοχείου:

```
hostname

whoami
```

- Και το δίκτυο Kubernetes:

```
nslookup kubernetes

# this will fail:
ping kubernetes
```

Ο διακομιστής Kubernetes API είναι διαθέσιμος για χρήση σε δοχείο σε Pod, αλλά οι εσωτερικές διευθύνσεις δεν υποστηρίζουν ping

## – Σύνδεση μεταξύ pods

- Τερματίστε το κέλυφος στο Sleep Pod:

```
exit
```

- Εκτυπώστε τη διεύθυνση IP του αρχικού Whoami Pod.

```
kubectl get pods -o wide whoami
```

Αυτή είναι η εσωτερική διεύθυνση IP του Pod - κάθε άλλο Pod στον cluster μπορεί να συνδεθεί σε αυτήν τη διεύθυνση

- Κάντε ένα αίτημα στον διακομιστή HTTP στο Pod Whoami από το Sleep Pod:

```
kubectl exec sleep -- curl -s <whoami-pod-ip>
```

Η έξοδος είναι η απάντηση από τον διακομιστή whoami - περιλαμβάνει το hostname και τις διευθύνσεις IP

#### – Εργαστηριακή άσκηση 10

Τα pods είναι μια αφαίρεση πάνω από δοχεία. Παρακολουθούν το δοχείο και αν βγει από το Pod επανεκκινείται, δημιουργώντας ένα νέο δοχείο για να συνεχίσει να λειτουργεί η εφαρμογή σας. Αυτό είναι το πρώτο επίπεδο υψηλής διαθεσιμότητας που παρέχει το Kubernetes.

Μπορείτε να το δείτε σε δράση με μια κακώς διαμορφωμένη εφαρμογή, όπου το δοχείο συνεχίζει να εξέρχεται. Γράψτε μια προδιαγραφή Pod για να εκτελέσετε ένα δοχείο από την εικόνα του Docker Hub `courselabs/bad-sleep`. Αναπτύξτε τις προδιαγραφές σας και παρακολουθήστε το Pod - τι συμβαίνει μετά από περίπου 30 δευτερόλεπτα; Και μετά από λίγα λεπτά;

Το Kubernetes θα συνεχίσει να προσπαθεί να κάνει το Pod να λειτουργεί, επομένως θα πρέπει να το αφαιρέσετε όταν τελειώσετε.

#### – Καθάρισμα

Θα καθαρίσουμε πριν προχωρήσουμε, διαγράφοντας όλα τα Pods που δημιουργήσαμε:

```
kubectl delete pod sleep whoami
```

## Services

#### – Δικτύωση Pods με Υπηρεσίες (Services)

Κάθε POD έχει μια διεύθυνση IP που μπορεί να χρησιμοποιηθεί για να επικοινωνήσει με άλλα Pods στο Cluster, αλλά αυτή η διεύθυνση IP ισχύει μόνο όσο το POD είναι

ενεργό.

Οι **υπηρεσίες - services** παρέχουν μια συνεπή διεύθυνση IP που συνδέεται με ένα όνομα DNS και θα χρησιμοποιείτε πάντα Υπηρεσίες για τη δρομολόγηση εσωτερικής και εξωτερικής κίνησης σε Pods.

Οι υπηρεσίες και τα Pods συνδέονται χαλαρά: μια Υπηρεσία βρίσκει Pods προορισμού χρησιμοποιώντας έναν **επιλογή ετικετών**.

#### – API Spec

- **Service**

Οι ορισμοί υπηρεσιών έχουν τα συνηθισμένα μεταδεδομένα. Η προδιαγραφή πρέπει να περιλαμβάνει τις θύρες δικτύου και τον επιλογή ετικετών:

```
apiVersion: v1
kind: Service
metadata:
  name: whoami
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 80
      targetPort: 80
```

Οι θύρες είναι εκεί όπου ακούει η Υπηρεσία και ο επιλογέας ετικετών μπορεί να αντιστοιχίσει κανένα ή περισσότερα Pods.

- selector - λίστα ετικετών για εύρεση pods - στόχων
- ports - λίστα θυρών για ακρόαση
- name - όνομα θύρας μέσα στο Kubernetes
- port - θύρα που ακούει η Υπηρεσία
- targetPort - θύρα στο Pod όπου αποστέλλεται η κίνηση.

#### – Pod YAML

Τα Pods πρέπει να περιλαμβάνουν αντίστοιχες ετικέτες για να λαμβάνουν κίνηση από την Υπηρεσία.

- Οι ετικέτες καθορίζονται στα μεταδεδομένα:

```
apiVersion: v1
kind: Pod
metadata:
  name: whoami
  labels:
    app: whoami
spec:
  # ...
```

Οι ετικέτες είναι αυθαίρετα ζεύγη κλειδιού-τιμής. Το app, το component και το version χρησιμοποιούνται συνήθως για τα Pods εφαρμογών.

#### – Εκτελέστε Pods

- Ξεκινήστε δημιουργώντας μερικά απλά Pods από ορισμούς που περιέχουν ετικέτες:

```
# save it in a folder called services/specs/pods and name it
whoami.yaml

apiVersion: v1
kind: Pod
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: services
    app: whoami
spec:
  containers:
    - name: app
      image: sixeyed/whoami:21.04

# save it in a folder called services/specs/pods and name it
sleep.yaml

apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: services
    app: sleep
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
```

```
kubectl apply -f services/specs/pods
```

Μπορείτε να εργαστείτε με πολλά αντικείμενα και να αναπτύξετε πολλαπλές εκδηλώσεις YAML με το Kubectl.

- Ελέγξτε την κατάσταση για όλα τα Pods, εκτυπώνοντας όλες τις διευθύνσεις IP και τις ετικέτες.

```
kubectl get pods -o wide --show-labels
```

- Το όνομα POD δεν έχει καμία επίδραση στη δικτύωση, τα pods δεν μπορούν να επικοινωνήσουν μεταξύ τους με το όνομα:

```
kubectll exec sleep -- nslookup whoami
```

#### – Αναπτύξτε μια εσωτερική Υπηρεσία

Το Kubernetes παρέχει διαφορετικούς τύπους Υπηρεσιών για εσωτερική και εξωτερική πρόσβαση στα Pods.

Το **επιλογή ετικετών** είναι η προεπιλογή και σημαίνει ότι η Υπηρεσία λαμβάνει μια διεύθυνση IP η οποία είναι προσβάσιμη μόνο εντός του cluster - για τα στοιχεία επικοινωνίας εσωτερικά.

```
# save it in a folder called services/specs/services and name
  it whoami-clusterip.yaml

# defines a ClusterIP service which routes traffic to the
  whoami Pod

apiVersion: v1
kind: Service
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: services
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 80
      targetPort: 80
  type: ClusterIP
```

- Αναπτύξτε την Υπηρεσία από τον φάκελο services/specs/services/whoami-clusterip.yaml και εκτυπώστε τα στοιχεία της.

```
kubectll apply -f services/specs/services/whoami-clusterip.
  yaml
```

- Εκτυπώστε τις λεπτομέρειες:

```
kubectll get service whoami

kubectll describe svc whoami
```

Οι εντολές "get" και "describe" είναι ίδιες για όλα τα αντικείμενα. Οι υπηρεσίες έχουν το ψευδώνυμο svc.

Η Υπηρεσία έχει τη δική της διεύθυνση IP, η οποία είναι στατική για όλη τη διάρκεια της Υπηρεσίας.

– **Χρησιμοποιήστε το DNS για να βρείτε την Υπηρεσία**

- Το Kubernetes εκτελεί έναν διακομιστή DNS μέσα στο cluster και κάθε Υπηρεσία λαμβάνει μια καταχώρηση, που συνδέει τη διεύθυνση IP με το όνομα της Υπηρεσίας.

```
kubectl exec sleep -- nslookup whoami
```

Αυτό λαμβάνει τη διεύθυνση IP της Υπηρεσίας από το όνομα DNS της. Η πρώτη γραμμή είναι η διεύθυνση IP του ίδιου του Kubernetes DNS διακομιστή.

- Τώρα τα Pods μπορούν να επικοινωνούν χρησιμοποιώντας ονόματα DNS:

```
kubectl exec sleep -- curl -s http://whoami
```

Δημιουργήστε ξανά το Whoami Pod και η αντικατάσταση θα έχει μια νέα διεύθυνση IP - αλλά η υπηρεσία με DNS εξακολουθεί να λειτουργεί.

- Ελέγξτε την τρέχουσα διεύθυνση IP και, στη συνέχεια, διαγράψτε το Pod:

```
kubectl get pods -o wide -l app=whoami
```

```
kubectl delete pods -l app=whoami
```

Μπορείτε να χρησιμοποιήσετε επιλογείς ετικετών και στο Kubectl - οι ετικέτες είναι ένα ισχυρό εργαλείο διαχείρισης.

- Δημιουργήστε ένα άλλο Pod και ελέγξτε τη διεύθυνση IP του:

```
kubectl apply -f services/specs/pods
```

```
kubectl get pods -o wide -l app=whoami
```

- Η διεύθυνση IP της υπηρεσίας δεν αλλάζει, επομένως εάν οι πελάτες αποθηκεύσουν προσωρινά αυτήν την IP, θα εξακολουθούν να λειτουργούν. Τώρα η Υπηρεσία δρομολογεί την κυκλοφορία στο νέο Pod:

```
kubectl exec sleep -- nslookup whoami
```

```
kubectl exec sleep -- curl -s http://whoami
```

### – Κατανόηση εξωτερικών υπηρεσιών

Υπάρχουν δύο τύποι Υπηρεσίας στους οποίους μπορείτε να έχετε πρόσβαση εκτός του cluster: [LoadBalancer](#) και [NodePort](#).

Και οι δύο ακούν την κίνηση που έρχεται στο cluster και το δρομολογούν στο Pods, αλλά λειτουργούν με διαφορετικούς τρόπους. Οι LoadBalancers είναι πιο εύκολοι στην κατανόηση, αλλά δεν τα υποστηρίζει κάθε cluster Kubernetes.

Σε αυτό το εργαστήριο θα αναπτύξουμε τόσο LoadBalancers όσο και NodePorts για όλες τις εφαρμογές μας.

- Οι Υπηρεσίες LoadBalancer ενσωματώνονται με την πλατφόρμα στην οποία εκτελούνται για να λάβουν μια πραγματική διεύθυνση IP. Σε μια διαχειριζόμενη υπηρεσία Kubernetes στο cloud, θα λάβετε μια μοναδική δημόσια διεύθυνση IP για κάθε Υπηρεσία, ενσωματωμένη με έναν εξισορροπητή φόρτου cloud για να κατευθύνει την κυκλοφορία στους κόμβους σας. Στο Docker Desktop η διεύθυνση IP θα είναι localhost. Σε k3s θα είναι διεύθυνση τοπικού δικτύου.
- Τα NodePort δεν χρειάζονται εξωτερική ρύθμιση, επομένως λειτουργούν με τον ίδιο τρόπο σε όλα τα Kubernetes clusters. Κάθε κόμβος στο cluster ακούει στην καθορισμένη θύρα και προωθεί την κυκλοφορία σε Pods. Ο αριθμός εξωτερικής θύρας πρέπει να είναι  $\geq 30000$  - μια επιλογή για ενισχυμένη ασφάλεια, ώστε τα στοιχεία του Kubernetes να μην χρειάζεται να εκτελούνται με αυξημένα δικαιώματα στον κόμβο.

Platform	LoadBalancer	NodePort	—	—	—	Docker Desktop	Y	Y	K3s	Y
Y K3d	Y	Y	AKS, EKS, GKE etc.	Y	Y	Kind	N	Y	Minikube	N
Y	Microk8s	N	Y	Bare-metal	N	Y				

Εάν δεν έχετε υποστήριξη LoadBalancer, μπορείτε να την προσθέσετε με το [MetalLB](#), αλλά δεν θα ασχοληθούμε με αυτό το ζήτημα στο εργαστήριο :)

### – Αναπτύξτε μια εξωτερική Υπηρεσία

Υπάρχουν δύο ορισμοί Υπηρεσίας για να γίνει η εφαρμογή whoami διαθέσιμη εκτός του cluster:

```
# save it in a folder called services/specs/services and name
it whoami-nodeport.yaml

# for clusters which don't support LoadBalancer Services

apiVersion: v1
kind: Service
metadata:
  name: whoami-np
  labels:
    kubernetes.courselabs.co: services
    app: whoami
```



```
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8010
      targetPort: 80
      nodePort: 30010
      type: NodePort

# save it in a folder called services/specs/services and name
# it whoami-loadbalancer.yaml

# for clusters which do

apiVersion: v1
kind: Service
metadata:
  name: whoami-lb
  labels:
    kubernetes.courselabs.co: services
    app: whoami
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8080
      targetPort: 80
      type: LoadBalancer
```

- Αναπτύξτε και τα δύο:

```
kubectl apply -f services/specs/services/whoami-nodeport.yaml
-f services/specs/services/whoami-loadbalancer.yaml
```

- Εκτυπώστε τις λεπτομέρειες για τις υπηρεσίες - και οι δύο έχουν την ετικέτα app=whoami.

```
kubectl get svc -l app=whoami
```

Εάν το cluster σας δεν διαθέτει υποστήριξη LoadBalancer, το πεδίο EXTERNAL-IP θα παραμείνει στο <pending> για πάντα.

Οι εξωτερικές υπηρεσίες δημιουργούν επίσης ένα ClusterIP, ώστε να μπορείτε να έχετε πρόσβαση σε αυτές εσωτερικά.

- Πρέπει πάντα να χρησιμοποιείτε τη θύρα υπηρεσίας για επικοινωνία:

```
kubectl exec sleep -- curl -s http://whoami-lb:8080
kubectl exec sleep -- curl -s http://whoami-np:8010
```

Όλες οι Υπηρεσίες έχουν τον ίδιο επιλογέα ετικετών, επομένως όλες κατευθύνουν την κίνηση στο ίδιο Pod

- Τώρα μπορείτε να καλέσετε την εφαρμογή whoami από τον τοπικό σας υπολογιστή:

```
# either
curl http://localhost:8080

# or
curl http://localhost:30010
```

Εάν το σύμπλεγμα σας δεν εκτελείται τοπικά, χρησιμοποιήστε τη διεύθυνση IP του κόμβου για πρόσβαση στο NodePort ή το πεδίο διεύθυνσης EXTERNAL-IP για το LoadBalancer.

### Εργαστηριακή άσκηση 11

Οι υπηρεσίες είναι μια αφηρημένη έννοια δικτύωσης - είναι σαν δρομολογητές που ακούν την εισερχόμενη κίνηση και την κατευθύνουν στα Pods.

Τα Target Pods προσδιορίζονται με έναν επιλογέα ετικετών και μπορεί να υπάρχουν μηδέν ή περισσότερες αντιστοιχίσεις.

Δημιουργήστε νέες υπηρεσίες και pods Whoami για να δοκιμάσετε τα παρακάτω σενάρια:

- κανένα Pod δεν αντιστοιχεί στις προδιαγραφές της ετικέτας
- πολλά Pods ταιριάζουν με τις προδιαγραφές της ετικέτας

Τι συμβαίνει; Πώς μπορείτε να βρείτε τα Target Pods για μια υπηρεσία;

#### – Καθάρισμα

Κάθε προδιαγραφή YAML για αυτό το εργαστήριο προσθέτει μια ετικέτα `kubernetes.courselabs.co=services`.

Αυτό καθιστά εξαιρετικά εύκολο τον καθαρισμό, διαγράφοντας όλους αυτούς τους πόρους:

```
kubectl delete pod,svc -l kubernetes.courselabs.co=services
```

## Deployments

### – Κλιμάκωση και διαχείριση Pods με Deployments

Δεν δημιουργείτε συχνά Pods απευθείας, επειδή αυτό δεν είναι ευέλικτο - δεν μπορείτε να ενημερώσετε τα Pods για να κυκλοφορούν ενημερώσεις εφαρμογών και μπορείτε να τα κλιμακώσετε μόνο αναπτύσσοντας εσείς νέα Pods.

Αντίθετα, θα χρησιμοποιήσετε έναν **ελεγκτή - controller** - ένα αντικείμενο Kubernetes που διαχειρίζεται άλλα αντικείμενα. Ο ελεγκτής που θα χρησιμοποιήσετε περισσότερο για τα Pods είναι το Deployment, το οποίο διαθέτει δυνατότητες για υποστήριξη αναβαθμίσεων και κλίμακας.

Οι αναπτύξεις χρησιμοποιούν ένα πρότυπο για τη δημιουργία Pods και έναν επιλογέα ετικετών για να προσδιορίσουν τα Pods που κατέχουν.

### – API specs

- **Deployment**

Οι ορισμοί των Deployments έχουν τα συνηθισμένα μεταδεδομένα.

- Η προδιαγραφή είναι πιο ενδιαφέρουσα - περιλαμβάνει έναν επιλογέα ετικέτας αλλά και μια προδιαγραφή Pod. Η προδιαγραφή Pod είναι η ίδια μορφή που θα χρησιμοποιούσατε για να ορίσετε ένα Pod στο YAML, με τη διαφορά ότι δεν συμπεριλαμβάνετε όνομα.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
spec:
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04.01
```

Οι ετικέτες στα μεταδεδομένα Pod πρέπει να περιλαμβάνουν τις ετικέτες στον επιλογέα για το Deployment, διαφορετικά θα λάβετε ένα σφάλμα όταν προσπαθείτε να εφαρμόσετε το YAML.

- spec.selector- λίστα ετικετών για εύρεση Pods
- spec.template - το πρότυπο που θα χρησιμοποιηθεί για τη δημιουργία Pods

- `spec.template.metadata` - μεταδεδομένα για το πεδίο Pods - χωρίς πεδίο ονόματος
- `spec.template.metadata.labels` - οι ετικέτες για εφαρμογή σε Pods, πρέπει να περιληφθούν αυτές στον επιλογή
- `spec.template.spec` - πλήρης προδιαγραφή Pod

– **Δημιουργήστε ένα Deployment για την εφαρμογή whoami**

Το cluster σας θα πρέπει να είναι άδειο εάν διαγράψατε το τελευταίο εργαστήριο. Αυτή η προδιαγραφή περιγράφει ένα Deployment για τη δημιουργία ενός Whoami Pod:

```
# save it in a folder called deployments/specs/deployments
# and name it whoami-v1.yaml

# the same Pod spec you've seen, wrapped in a Deployment

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: deployments
spec:
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
        version: v1
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
```

- Δημιουργήστε το Deployment και θα δημιουργήσει το Pod:

```
kubectl apply -f deployments/specs/deployments/whoami-v1.yaml
```

```
kubectl get pods -l app=whoami
```

Τα Deployments εφαρμόζουν το δικό τους σύστημα ονομασίας όταν δημιουργούν Pods, και τελειώνουν με μια τυχαία συμβολοσειρά.

Τα Deployments είναι σημαντικά αντικείμενα, εργάζεστε μαζί τους στο Kubectl με τον συνηθισμένο τρόπο.

- Εκτυπώστε τις λεπτομέρειες του Deployment.

```
kubectl get deployments

kubectl get deployments -o wide

kubectl describe deploy whoami
```

Τα events μιλούν για ένα άλλο αντικείμενο που ονομάζεται ReplicaSet - θα το φτάσουμε σύντομα.

#### – Κλιμάκωση Deployments

Το Deployment ξέρει πώς να δημιουργεί Pods από το πρότυπο στην προδιαγραφή. Μπορείτε να δημιουργήσετε όσα αντίγραφα θέλετε - διαφορετικά Pods που δημιουργήθηκαν από την ίδια προδιαγραφή Pod - όσα μπορεί να χειριστεί το cluster σας.

- Μπορείτε να κλιμακώσετε επιτακτικά με το Kubectl:

```
kubectl scale deploy whoami --replicas 3

kubectl get pods -l app=whoami
```

Αλλά τώρα το εκτελούμενο αντικείμενο Deployment είναι διαφορετικό από τις προδιαγραφές που έχετε στον έλεγχο πηγής. Αυτό είναι κακό.

- Επομένως, είναι καλύτερο να κάνετε τις αλλαγές δηλωτικά στο YAML.

```
# save it in a folder called deployments/specs/deployments
# and name it whoami-v1-scale.yaml

# sets a replica level of 2

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: deployments
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
        version: v1
    spec:
```

```
containers:
  - name: app
    image: sixeyed/whoami:21.04
```

- Ενημερώστε το Deployment χρησιμοποιώντας αυτήν την προδιαγραφή και ελέγξτε ξανά τα Pods.

```
kubectl apply -f deployments/specs/deployments/whoami-v1-scale.yaml
```

```
kubectl get pods -l app=whoami
```

Το Deployment καταργεί ένα Pod, επειδή η τρέχουσα κατάσταση (3 αντίγραφα) δεν ταιριάζει με την επιθυμητή κατάσταση στο YAML (2 αντίγραφα)

#### – Εργασία με διαχειριζόμενα Pods

- Επειδή τα ονόματα pod είναι τυχαία, θα τα διαχειριστείτε στο Kubectl χρησιμοποιώντας ετικέτες. Το κάναμε αυτό με το "get" και λειτουργεί και για logs:

```
kubectl logs -l app=whoami
```

- Και αν χρειάζεται να εκτελέσετε εντολές στο Pod, μπορείτε να χρησιμοποιήσετε το exec στο επίπεδο Deployment:

```
# this will fail
kubectl exec deploy/whoami -- hostname
```

Δεν υπάρχει shell σε αυτήν την εικόνα κοντέινερ :)

Η προδιαγραφή Pod στο πρότυπο Deployment εφαρμόζει μια ετικέτα.

- Εκτυπώστε λεπτομέρειες - συμπεριλαμβανομένων της διεύθυνσης IP και των ετικετών - για όλα τα Pods με την ετικέτα app=whoami.

```
kubectl get pods -o wide --show-labels -l app=whoami
```

- Ο επιλογέας ετικέτας σε αυτές τις Υπηρεσίες ταιριάζει και με αυτήν την ετικέτα:

```
# save it in a folder called deployments/specs/services and
  name it whoami-loadbalancer.yaml
```

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

```

name: whoami-lb
labels:
  kubernetes.courselabs.co: deployments
  app: whoami
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8080
      targetPort: 80
    type: LoadBalancer

# save it in a folder called deployments/specs/services and
# name it whoami-nodeport.yaml

apiVersion: v1
kind: Service
metadata:
  name: whoami-lb
  labels:
    kubernetes.courselabs.co: deployments
    app: whoami
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8080
      targetPort: 80
    type: LoadBalancer

```

- Αναπτύξτε τις Υπηρεσίες και ελέγξτε τα endpoints IP του Pod:

```

kubectl apply -f labs/deployments/specs/services/
kubectl get endpoints whoami-np whoami-lb

```

- Έτσι, μπορείτε ακόμα να έχετε πρόσβαση στην εφαρμογή από το μηχανήμά σας:

```

# either
curl http://localhost:8080

# or
curl http://localhost:30010

```

### – Ενημέρωση της εφαρμογής

Οι ενημερώσεις εφαρμογών συνήθως σημαίνουν μια αλλαγή στην προδιαγραφή Pod - μια νέα εικόνα δοχείου ή μια αλλαγή διαμόρφωσης. Δεν μπορείτε να αλλάξετε τις προδιαγραφές ενός Pod που εκτελείται, αλλά μπορείτε να αλλάξετε τις προδιαγραφές του Pod σε ένα Deployment. Κάνει την αλλαγή ξεκινώντας νέα Pods και τερματίζοντας τα παλιά.

```
# save it in a folder called deployments/specs/deployments
and name it whoami-loadbalancer.yaml

# changes a configuration setting for the app. It's an
environment variable update, but those are fixed for the
life of a Pod container, so this change means new Pods.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: deployments
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
        version: v2
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q
```

```
# open a new terminal to monitor the Pods:
kubectl get po -l app=whoami --watch

# apply the change:
kubectl apply -f labs/deployments/specs/deployments/whoami-v2
.yaml
```



Δοκιμάστε ξανά την εφαρμογή - θα δείτε μικρότερη έξοδο και αν επαναλάβετε τα αιτήματά σας είναι εξισορροπημένα.

- Τα Deployments αποθηκεύουν προηγούμενες προδιαγραφές στη βάση δεδομένων Kubernetes και μπορείτε εύκολα να μεταβείτε σε παλαιότερη έκδοση αν η καινούργια δεν λειτουργεί:

```
kubectl rollout history deploy/whoami
kubectl rollout undo deploy/whoami
kubectl get po -l app=whoami
```

Δοκιμάστε ξανά την εφαρμογή και θα δείτε ότι επιστρέφουμε σε μεγαλύτερη έξοδο.

## – Εργαστηριακή άσκηση 12

Οι κυλιόμενες ενημερώσεις δεν είναι πάντα αυτό που θέλετε - σημαίνουν ότι η παλιά και η νέα έκδοση της εφαρμογής σας εκτελούνται ταυτόχρονα, ενώ και οι δύο επεξεργάζονται αιτήματα.

Μπορεί να θέλετε μια blue-green Deployment, όπου εκτελούνται και οι δύο εκδόσεις, αλλά μόνο η μία λαμβάνει κίνηση.

Γράψτε Deployments and Services για να δημιουργήσετε μια blue-green ενημέρωση για την εφαρμογή whoami. Ξεκινήστε εκτελώντας δύο αντίγραφα για το v1 και δύο για το v2, αλλά μόνο τα v1 Pods θα πρέπει να λαμβάνουν κίνηση.

Στη συνέχεια, πραγματοποιήστε την ενημέρωσή σας για να αλλάξετε την κίνηση στα v2 Pods χωρίς αλλαγές στα Deployments.

## – EXTRA: Κατανόηση ReplicaSets

Παρατηρήσατε κάποιο μοτίβο στα ονόματα των Pod στην άσκηση επαναφοράς; Όταν επαναφέρατε την ενημέρωσή σας, μπορεί να έχετε δει ότι τα νέα Pods είχαν το ίδιο πρόθεμα με το προηγούμενο σύνολο Pods.//

Τα Deployments δημιουργούν τα ονόματα Pod, αλλά δεν είναι εντελώς τυχαία - το μοτίβο είναι [deployment-name]-[template-hash]-[random-suffix]. Μπορείτε να ενημερώσετε μια προδιαγραφή Deployment χωρίς να αλλάξετε την προδιαγραφή Pod (π.χ. να ορίσετε αντίγραφα) και αυτό δεν προκαλεί αντικατάσταση του Pod.

- Όταν αλλάζετε την προδιαγραφή Pod στο πρότυπο, αυτό σημαίνει νέα Pods - και το Deployment τοποθετεί την ευθύνη για τη δημιουργία Pod στα ReplicaSets:

```
kubectl get replicaset
```

Το όνομα είναι το όνομα του Deployment και τον κατακερματισμό του template.

- Τα Deployments διαχειρίζονται ενημερώσεις δημιουργώντας ReplicaSets και διαχειρίζοντας τον αριθμό των επιθυμητών Pods για το αντίγραφο. Οι αντικατασταθείσες προδιαγραφές μειώνονται σε 0, αλλά αν ένα νέο UPDATE ταιριάζει με ένα παλιό spec, το αρχικό αντίγραφο επαναχρησιμοποιείται.

```
# in a new terminal:
kubectl get rs --watch

kubectl apply -f deployments/specs/deployments/whoami-v2.yaml
```

Θα δείτε την κυλιόμενη ενημέρωση σε δράση - το νέο ReplicaSet κλιμακώνεται σταδιακά, ενώ το παλιό μειώνεται

#### – Καθάρισμα

Καθαρίστε αφαιρώντας αντικείμενα με την ετικέτα αυτού του εργαστηρίου:

```
kubectl delete deploy,svc -l kubernetes.courselabs.co=
  deployments
```

## 3.2 Μοντελοποίηση Εφαρμογών

### ConfigMaps

#### – Διαμόρφωση εφαρμογών με ConfigMaps

Υπάρχουν δύο τρόποι αποθήκευσης ρυθμίσεων διαμόρφωσης στο [ConfigMaps](#) - είτε ως ζεύγη κλειδιού-τιμής, τα οποία θα εμφανίσετε ως μεταβλητές περιβάλλοντος ή ως δεδομένα κειμένου που θα εμφανίσετε ως αρχεία στο σύστημα αρχείων του δοχείου.

#### – API Specs

- [ConfigMaps](#)

#### – ConfigMap και Pod YAML - χρησιμοποιώντας μεταβλητές περιβάλλοντος

Τα ζεύγη κλειδιών-τιμών ορίζονται στο YAML ως εξής:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
data:
  Configurable__Environment: uat
```

Τα μεταδεδομένα είναι τυπικά - θα αναφέρετε το όνομα του ConfigMap στην προδιαγραφή Pod για να φορτώσετε τις ρυθμίσεις.

- **data** - λίστα ρυθμίσεων ως ζεύγη κλειδιού-τιμής, διαχωρισμένα με άνω και κάτω τελείες.

Στην προδιαγραφή Pod προσθέτετε μια αναφορά:

```
spec:
  containers:
    - name: app
      image: sixeyed/configurable:21.04
      envFrom:
        - configMapRef:
            name: configurable-env
```

- **envFrom** - φορτώστε όλες τις τιμές στην πηγή ως μεταβλητές περιβάλλοντος.

#### – ConfigMap και Pod YAML - χρησιμοποιώντας το σύστημα αρχείων του δοχείου

Τα αρχεία κειμένου ορίζονται στην ίδια δομή YAML, μία καταχώρηση για κάθε αρχείο:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-override
data:
  override.json: |-
    {
      "Configurable": {
        "Release": "21.04.01"
      }
    }
```

Προσοχή με το κενό διάστημα - τα δεδομένα του αρχείου πρέπει να έχουν μια εσοχή περισσότερο από το όνομα του αρχείου

Η προδιαγραφή API είναι η ίδια, αλλά σε αυτήν τη μορφή:

- **data** - λίστα αρχείων, με το σύνολο ονομάτων αρχείου και τα περιεχόμενα να ακολουθούν το διαχωριστικό |-

Στην προδιαγραφή Pod μπορείτε να φορτώσετε όλες τις τιμές στο σύστημα αρχείων του δοχείου ως προσαρτήσεις τόμου:

```
spec:
  containers:
    - name: app
```

```

    image: sixeyed/configurable:21.04
    volumeMounts:
      - name: config-override
        mountPath: "/app/config"
        readOnly: true
    volumes:
      - name: config-override
        configMap:
          name: configurable-override

```

Οι τόμοι ορίζονται σε επίπεδο Pod - είναι μονάδες αποθήκευσης που αποτελούν μέρος του περιβάλλοντος Pod. Φορτώνετε τη μονάδα αποθήκευσης στο σύστημα αρχείων του δοχείου χρησιμοποιώντας μια προσάρτηση.

- volumes - λίστα μονάδων αποθήκευσης προς φόρτωση, μπορεί να είναι ConfigMaps, Secrets ή άλλοι τύποι
- volumeMounts - λίστα τόμων για προσάρτηση στο σύστημα αρχείων του δοχείου
- volumeMounts.name - ταιριάζει με το όνομα του τόμου
- volumeMounts.mountPath - η διαδρομή καταλόγου όπου εμφανίζεται ο τόμος
- volumeMounts.readOnly - επισήμανση εάν ο τόμος είναι μόνο για ανάγνωση ή επεξεργασίμος

#### – Εκτελέστε demo με δυνατότητα διαμόρφωσης

Η εφαρμογή demo για αυτό το εργαστήριο έχει τη λογική να συγχωνεύει τις ρυθμίσεις παραμέτρων από πολλές πηγές.

- Οι προεπιλογές είναι ενσωματωμένες στο αρχείο appsettings.json μέσα στην εικόνα Docker - εκτελέστε ένα Pod χωρίς να έχουν εφαρμοστεί ρυθμίσεις για να δείτε τις προεπιλογές:

```

kubectl run configurable --image=sixeyed/configurable:21.04
  --labels='kubernetes.courselabs.co=configmaps'

kubectl wait --for=condition=Ready pod configurable

kubectl port-forward pod/configurable 8080:80

```

Αυτές είναι χρήσιμες εντολές για γρήγορη δοκιμή ή εντοπισμό σφαλμάτων, αλλά στην πραγματικότητα όλα είναι YAML.

Ελέγξτε την εφαρμογή στη διεύθυνση <http://localhost:8080> (ή τη διεύθυνση IP του κόμβου σας εάν έχετε απομακρυσμένο cluster).

Βλέπετε τις προεπιλεγμένες ρυθμίσεις διαμόρφωσης από το αρχείο JSON στην εικόνα του δοχείου. Οι μεταβλητές περιβάλλοντος προέρχονται από το Dockerfile, συν το λειτουργικό σύστημα του δοχείου και κάποιες που έχουν οριστεί από την Kubernetes.

- Τερματίστε το port-forward και αφαιρέστε το Pod.

```
# Ctrl-C to exit the command

kubectl delete pod configurable
```

- Τερματίστε το port-forward και αφαιρέστε το Pod.

```
# Ctrl-C to exit the command

kubectl delete pod configurable
```

#### – Ρύθμιση Config με μεταβλητές περιβάλλοντος στην προδιαγραφή του Pod

- Η προδιαγραφή του Pod είναι αυτή όπου εφαρμόζετε τη διαμόρφωση:

```
docker pull java
```

```
# save it in a folder called configmaps/specs/configurable/
# and name it whoami-loadbalancer.yaml

# it adds a config setting with an environment variable in
# the template Pod spec.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: configmaps
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          env:
            - name: Configurable__Release
              value: 24.01.1
```

- Αναπτύξτε την εφαρμογή από το φάκελο configmaps/specs/configurable/

```
kubectl apply -f configmaps/specs/configurable/
```

- Μπορείτε να ελέγξετε ότι η μεταβλητή περιβάλλοντος έχει οριστεί εκτελώντας το `printenv` μέσα στο δοχείο του Pod:

```
kubectl exec deploy/configurable -- printenv | grep __
```

Θα πρέπει να εμφανίζεται το `Configurable__Release=24.01.1`

- Επιβεβαιώστε το κάνοντας περιήγηση στην εφαρμογή από την Υπηρεσία σας.

```
# print the Service details:
kubectl get svc -l app=configurable
```

#### – Ρύθμιση Config με μεταβλητές περιβάλλοντος στα ConfigMaps

- Οι μεταβλητές περιβάλλοντος στις προδιαγραφές του Pod είναι καλές για μεμονωμένες ρυθμίσεις όπως οι επιλογές χαρακτηριστικών. Συνήθως θα έχετε πολλές ρυθμίσεις και θα χρησιμοποιήσετε ένα ConfigMap:

```
# save it in a folder called configmaps/specs/configurable/
# config-env/ and name it configmap-env.yaml

# a ConfigMap with multiple environment variables

apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
  labels:
    kubernetes.courselabs.co: configmaps
    app: configurable
data:
  Configurable__Release: 24.01.2
  Configurable__Environment: uat

# save it in a folder called configmaps/specs/configurable/
# config-env/ and name it deployment-env.yaml

# a Deployment which loads the ConfigMap into environment
# variables

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: configmaps
```

```

    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - configMapRef:
                name: configurable-env

```

```
kubectl apply -f configmaps/specs/configurable/config-env/
```

Αυτό δημιουργεί ένα νέο ConfigMap και ενημερώνει το Deployment. Θυμάστε ποιο αντικείμενο χρησιμοποιεί το Deployment για τη διαχείριση των Pods;

- Εκτυπώστε τις μεταβλητές περιβάλλοντος που έχουν οριστεί στο ενημερωμένο Pod.

```
kubectl exec deploy/configurable -- printenv | grep __
```

Θα πρέπει να δείτε ότι η έκδοση είναι τώρα 24.01.2 και υπάρχει μια νέα ρύθμιση Configurable\_\_Environment=uat

#### – Ρύθμιση Config με αρχεία στο ConfigMaps

Οι μεταβλητές του περιβάλλοντος είναι επίσης περιορισμένες. Είναι ορατές σε όλες τις διαδικασίες, επομένως υπάρχει πιθανότητα διαρροής ευαίσθητων πληροφοριών. Μπορεί επίσης να υπάρξουν συγκρούσεις εάν τα ίδια κλειδιά χρησιμοποιούνται από διαφορετικές διεργασίες.

Το σύστημα αρχείων είναι ένα ο πιο αξιόπιστος χώρος αποθήκευσης για τη διαμόρφωση. Μπορούν να οριστούν δικαιώματα για αρχεία και επιτρέπει πιο σύνθετα config με ένθετες ρυθμίσεις.

- Η εφαρμογή demo μπορεί να χρησιμοποιεί διαμόρφωση JSON καθώς και μεταβλητές περιβάλλοντος και υποστηρίζει τη φόρτωση πρόσθετων ρυθμίσεων από ένα αρχείο παράκαμψης:

```

\begin{lstlisting}[language=bash]
# save it in a folder called configmaps/specs/configurable/
# config-json/ and name it configmap-env.yaml

# it stores the config settings as a JSON data item

apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-override
  labels:
    kubernetes.courselabs.co: configmaps
data:
  override.json: |-
    {
      "Configurable": {
        "Release": "21.04.03"
      },
      "Features" : {
        "Caching" : true
      }
    }

# save it in a folder called configmaps/specs/configurable/
# config-json/ and name it deployment-env.yaml

# it loads the JSON as a volume mount and loads environment
# variables

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: configmaps
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - configMapRef:

```



```

        name: configurable-env
      volumeMounts:
        - name: config-override
          mountPath: "/app/config"
          readOnly: true
      volumes:
        - name: config-override
          configMap:
            name: configurable-override

```

```
kubectl apply -f configmaps/specs/configurable/config-json/
```

Ανανεώστε την εφαρμογή Ιστού και θα δείτε νέες ρυθμίσεις που προέρχονται από το αρχείο config/override.json

- Ελέγξτε το σύστημα αρχείων μέσα στο δοχείο για να δείτε το αρχείο που έχει φορτωθεί από το ConfigMap στη διαδρομή /app/config.

```
kubectl exec deploy/configurable -- ls /app/
```

```
kubectl exec deploy/configurable -- ls /app/config/
```

```
kubectl exec deploy/configurable -- cat /app/config/override.json
```

Το πρώτο αρχείο JSON προέρχεται από την εικόνα του δοχείου, το δεύτερο από την προσάρτηση τόμου ConfigMap.

- Ωστόσο, κάτι δεν πάει καλά - η ρύθμιση έκδοσης εξακολουθεί να προέρχεται από τη μεταβλητή περιβάλλοντος:

```
kubectl exec deploy/configurable -- cat /app/config/override.json
```

```
kubectl exec deploy/configurable -- printenv | grep __
```

Η ιεραρχία ρυθμίσεων σε αυτήν την εφαρμογή βάζει τις μεταβλητές περιβάλλοντος μπροστά από τις ρυθμίσεις στα αρχεία, ώστε να υπερκαλύπτονται. Θα πρέπει να κατανοήσετε την ιεραρχία προκειμένου οι εφαρμογές σας να μοντελοποιηθούν σωστά τη διαμόρφωση.

### – Εργαστηριακή άσκηση 13

Η διαμόρφωση αντιστοίχισης στο ConfigMap YAML λειτουργεί καλά και σημαίνει ότι μπορείτε να αναπτύξετε ολόκληρη την εφαρμογή σας με το "kubectl apply".

Αλλά δεν ταιριάζει σε κάθε οργανισμό και το Kubernetes υποστηρίζει επίσης τη δημιουργία ConfigMaps απευθείας από τιμές και αρχεία διαμόρφωσης - χωρίς κανένα YAML.

Δημιουργήστε δύο νέα ConfigMaps για να υποστηρίξετε το Deployment στο deployment-lab.yaml και ορίστε αυτές τις τιμές:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: configmaps
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - configMapRef:
                name: configurable-env-lab
          volumeMounts:
            - name: config-override
              mountPath: "/app/config"
              readOnly: true
      volumes:
        - name: config-override
          configMap:
            name: configurable-override-lab
```

- Μεταβλητή περιβάλλοντος Configuration\_\_Release=21.04-lab
- ρύθμιση JSON Features.DarkMode=true

– **EXTRA: Να είστε προσεκτικοί με τις προσαρτήσεις τόμων**

Η φόρτωση των ConfigMaps σε προσαρτήσεις τόμων είναι πολύ έξυπνη, αλλά υπάρχουν μερικά πράγματα που πρέπει να γνωρίζετε:

- Η ενημέρωση του ConfigMap δεν ενεργοποιεί την αντικατάσταση του Pod. Τα περιεχόμενα του νέου αρχείου φορτώνονται στην προσάρτηση τόμου, αλλά η εφαρμογή στο δοχείο μπορεί να το αγνοήσει εάν διαβάζει μόνο αρχεία διαμόρφωσης κατά την εκκίνηση.

- Οι τόμοι δεν συγχωνεύονται στη διαδρομή προορισμού για μια προσάρτηση τόμου - εάν ο κατάλογος υπάρχει ήδη, η προσάρτηση τόμου τον αντικαθιστά με τα περιεχόμενα του τόμου.

Μπορείτε εύκολα να χαλάσετε την εφαρμογή σας εάν κάνετε λάθος τις προσαρτήσεις τόμων:

```
# save it in a folder called configmaps/specs/configurable/
# config-broken/ and name it deployment-env.yaml

# it mounts a ConfigMap into the /app directory, which
# overwrites the actual app folder from the image

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: configmaps
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
        broken: bad-mount
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          volumeMounts:
            - name: config-override
              mountPath: "/app"
              readOnly: true
      volumes:
        - name: config-override
          configMap:
            name: configurable-override
            items:
              - key: override.json
                path: appsettings.json
```

```
kubectl apply -f configmaps/specs/configurable/config-broken/
```

```
kubectl get pods -l app=configurable --watch
```

Δημιουργείται ένα νέο pod, χαλάει και πηγαίνει σε κατάσταση CrashLoopBackoff.

```
# Ctrl-C
```

```
kubectl logs -l app=configurable,broken=bad-mount
```

Η προσάρτηση αντικαθιστά ολόκληρο τον φάκελο της εφαρμογής, επομένως δεν υπάρχει εφαρμογή για εκτέλεση :)

#### – Καθάρισμα

Καθαρίστε αφαιρώντας τα αντικείμενα με την ετικέτα αυτού του εργαστηρίου:

```
kubectl delete configmap,deploy,svc,pod -l kubernetes.
  courselabs.co=configmaps
```

- Αλλά το αρχικό Pod δεν αντικαθίσταται:

```
kubectl get replicaset -l app=configurable
```

- Εκτελέστε ένα δοχείο για να ελέγξετε την έκδοση Java:

Το Deployment δεν θα μειώσει το παλιό ReplicaSet έως ότου το νέο αποκτήσει την επιθυμητή χωρητικότητα. Η χρήση ενός Deployment προστατεύει την εφαρμογή σας από ζητήματα όπως αυτό.

#### – Καθάρισμα

Καθαρίστε αφαιρώντας αντικείμενα με την ετικέτα αυτού του εργαστηρίου:

```
kubectl delete configmap,deploy,svc,pod -l kubernetes.
  courselabs.co=configmaps
```

### Secrets

#### – Διαμόρφωση εφαρμογών με Secrets

Τα ConfigMaps είναι ευέλικτα για σχεδόν οποιοδήποτε σύστημα διαμόρφωσης εφαρμογής, αλλά δεν είναι κατάλληλα για ευαίσθητα δεδομένα. Τα περιεχόμενα του ConfigMap είναι ορατά σε απλό κείμενο σε οποιονδήποτε έχει πρόσβαση στο cluster σας.

Για ευαίσθητες πληροφορίες, το Kubernetes έχει μυστικά. Το API είναι πολύ παρόμοιο - μπορείτε να εμφανίσετε τα περιεχόμενα ως μεταβλητές περιβάλλοντος ή αρχεία στο δοχείο Pod - αλλά υπάρχουν πρόσθετες δικλίδες ασφαλείας σχετικά με το Secrets.

## – API Specs

- [Secrets](#)

## – Secrets και Pod YAML - μεταβλητές περιβάλλοντος

Οι τιμές Secrets μπορούν να κωδικοποιηθούν με base-64 και να οριστούν σε δεδομένα YAML:

```
apiVersion: v1
kind: Secret
metadata:
  name: configurable-secret-env
data:
  Configurable__Environment: cHJlLXByb2QK
```

Τα μεταδεδομένα είναι στάνταρ - θα αναφέρετε το όνομα του Secret στην προδιαγραφή Pod για να φορτώσετε τις ρυθμίσεις.

- data - λίστα ρυθμίσεων ως ζεύγη κλειδιού-τιμής, διαχωρισμένα με άνω και κάτω τελεία και με κωδικοποιημένες τιμές σε base-64.

Στην προδιαγραφή του Pod προσθέτετε μια αναφορά:

```
spec:
  containers:
    - name: app
      image: sixeyed/configurable:21.04
      envFrom:
        - secretRef:
            name: configurable-secret-env
```

- envFrom - φορτώστε όλες τις τιμές στην πηγή ως μεταβλητές περιβάλλοντος.

## – Δημιουργία Secrets από κωδικοποιημένο YAML

Η μοντελοποίηση της εφαρμογής σας για χρήση του Secrets είναι η ίδια όπως και με το ConfigMaps - φόρτωση μεταβλητών περιβάλλοντος ή τοποθέτηση τόμων.

Στο περιβάλλον του δοχείου, οι μυστικές τιμές παρουσιάζονται ως απλό κείμενο.

- Ξεκινήστε με την ανάπτυξη της διαμορφώσιμης εφαρμογής χρησιμοποιώντας το ConfigMaps:

```
# save it in a folder called secrets/specs/configurable and
name it configmaps.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
```

```

labels:
  kubernetes.courselabs.co: secrets
  app: configurable
data:
  Configurable__Release: 24.01.2
  Configurable__Environment: uat
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-override
data:
  override.json: |-
    {
      "Features" : {
        "Caching" : true
      }
    }

# save it in a folder called secrets/specs/configurable and
# name it deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - configMapRef:
                name: configurable-env
          volumeMounts:
            - name: config-override
              mountPath: "/app/config"
              readOnly: true
      volumes:
        - name: config-override

```

```

        configMap:
          name: configurable-override

# save it in a folder called secrets/specs/configurable and
# name it services.yaml

apiVersion: v1
kind: Service
metadata:
  name: configurable-np
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    app: configurable
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30010
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: configurable-lb
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    app: configurable
  ports:
    - name: http
      port: 8010
      targetPort: 80
    type: LoadBalancer

```

```
kubectl apply -f secrets/specs/configurable
```

- Ελέγξτε τις λεπτομέρειες ενός ConfigMap και μπορείτε να δείτε όλες τις τιμές σε απλό κείμενο.

```
kubectl get configmaps
```

```
kubectl describe cm configurable-env
```

Γι' αυτό δεν θέλετε ευαίσθητα δεδομένα εκεί.

- Αυτό το YAML δημιουργεί ένα Secret από μια κωδικοποιημένη τιμή και το φορτώνει σε μεταβλητές περιβάλλοντος σε μια ανάπτυξη:

```
# save it in a folder called secrets/specs/configurable/
# secrets-encoded and name it secret-encoded.yaml

# it uses data with encoded values

apiVersion: v1
kind: Secret
metadata:
  name: configurable-env-encoded
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
data:
  Configurable__Environment: cHJlLXByb2Q=

# save it in a folder called secrets/specs/configurable/
# secrets-encoded and name it deployment-env.yaml

# it loads the Secret into environment variables

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - secretRef:
                name: configurable-env-encoded
```



```
kubectl apply -f secrets/specs/configurable/secrets-encoded
```

Περιηγηθείτε στον ιστότοπο και μπορείτε να δείτε την τιμή απλού κειμένου για το Configurable:Environment

#### – Δημιουργία Secrets από απλό κείμενο YAML

Η κωδικοποίηση σε base-64 είναι δύσκολη και σας δίνει την ψευδαίσθηση ότι τα δεδομένα σας είναι ασφαλή. Η κωδικοποίηση δεν είναι κρυπτογράφηση και μπορείτε εύκολα να αποκωδικοποιήσετε τη βάση-64.

- Εάν θέλετε να αποθηκεύσετε ευαίσθητα δεδομένα σε απλό κείμενο YAML, μπορείτε να το κάνετε. Θα το κάνετε αυτό μόνο όταν το YAML σας είναι κλειδωμένο:

```
# save it in a folder called secrets/specs/configurable/
# secrets-encoded and name it secret-encoded.yaml

# it uses stringData with values in plain text

apiVersion: v1
kind: Secret
metadata:
  name: configurable-env-plain
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
stringData:
  Configurable__Environment: staging

# save it in a folder called secrets/specs/configurable/
# secrets-encoded and name it deployment-env.yaml

# it loads the Secret into environment variables

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
```

```

    app: configurable
  spec:
    containers:
    - name: app
      image: sixeyed/configurable:21.04
      envFrom:
      - secretRef:
          name: configurable-env-plain

```

```
kubectl apply -f secrets/specs/configurable/secrets-plain
```

Ανανεώστε τον ιστότοπο και θα δείτε την ενημερωμένη τιμή config.

#### – Εργασία με τιμές Secret σε base-64

Τα Secrets εμφανίζονται πάντα ως απλό κείμενο μέσα στο περιβάλλον του δοχείου.

Αλλά αυτή δεν είναι η προεπιλεγμένη ρύθμιση. Μπορείτε επίσης να ενσωματώσετε το Kubernetes με ασφαλής εφαρμογές, όπως το Hashicorp Vault και το Azure Key-Vault.

Το Kubectl δείχνει πάντα τα μυστικά κωδικοποιημένα ως base-64, αλλά αυτό είναι απλώς ένα βασικό μέτρο ασφαλείας.

- Τα Windows δεν έχουν εντολή "base64", επομένως εκτελέστε αυτό το script σε PowerShell εάν χρησιμοποιείτε Windows:

```

Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
  Process -Force

. ./scripts/windows-tools.ps1

```

Αυτό επηρεάζει μόνο την τρέχουσα περίοδο λειτουργίας PowerShell, δεν κάνει μόνιμες αλλαγές στο σύστημά σας.

- Μπορείτε να ανακτήσετε το στοιχείο δεδομένων από ένα Secret και να το αποκωδικοποιήσετε σε απλό κείμενο:

```

kubectl describe secret configurable-env-plain

kubectl get secret configurable-env-plain -o jsonpath="{.data
  .Configurable__Environment}"

kubectl get secret configurable-env-plain -o jsonpath="{.data
  .Configurable__Environment}" | base64 -d

```

Στην παραγωγή θα πρέπει να καταλάβετε πώς το cluster σας προστατεύει τα μυστικά. Θα χρησιμοποιήσετε επίσης το Role-Based Access Control για να περιορίσετε ποιος μπορεί να εργαστεί με τα Secrets στο Kubectl.

#### – Δημιουργία μυστικών από αρχεία

Ορισμένοι οργανισμοί έχουν ξεχωριστές ομάδες διαχείρισης διαμόρφωσης. Έχουν πρόσβαση στα ακατέργαστα ευαίσθητα δεδομένα και στο Kubernetes θα κατέχουν τη διαχείριση του Secrets.

Η ομάδα παραγωγής θα κατέχει το Deployment YAML το οποίο αναφέρεται στα Secrets και στα ConfigMaps. Η ροή εργασίας είναι αποσυνδεδεμένη, επομένως η ομάδα DevOps μπορεί να αναπτύξει και να διαχειριστεί την εφαρμογή χωρίς να έχει πρόσβαση στα ευαίσθητα δεδομένα.

- Υποδυθείτε την ομάδα διαχείρισης config με πρόσβαση σε Secrets στον τοπικό σας δίσκο:

```
# save it in a folder called secrets/secrets and name it
configurable.env

# a .env file for loading environment variables

Configurable__Release=21.04-secret

# save it in a folder called secrets/secrets and name it
secret.json

# a JSON file for loading as a volume mount

{
  "Features" : {
    "SecretApiKey" : "123APIKEY"
  }
}
```

- Δημιουργήστε Secrets από τα αρχεία

```
kubectl create secret generic configurable-env-file --from-
env-file ./secrets/secrets/configurable.env

kubectl create secret generic configurable-secret-file --from
-file ./secrets/secrets/secret.json
```

- Τώρα υποδυθείτε την ομάδα DevOps, αναπτύσσοντας την εφαρμογή χρησιμοποιώντας τα secrets που υπάρχουν ήδη:

```
# save it in a folder called secrets/specs/configurable/
secret-file and name it deployment.yaml

# it references those Secrets

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          envFrom:
            - secretRef:
                name: configurable-env-file
          volumeMounts:
            - name: secret
              mountPath: "/app/secrets"
              readOnly: true
      volumes:
        - name: secret
          secret:
            secretName: configurable-secret-file
```

```
kubectl apply -f ./secrets/specs/configurable/secrets-file
```

Περιηγηθείτε στην εφαρμογή και τώρα μπορείτε να δείτε μια άλλη πηγή διαμόρφωσης - το αρχείο secret.json

#### – Εργαστηριακή άσκηση 14

Η διαχείριση της διαμόρφωσης που έχει φορτωθεί σε προσαρτήσεις τόμου γίνεται από την Kubernetes. Εάν αλλάξει η πηγή ConfigMap ή Secret, το Kubernetes ωθεί την αλλαγή στο σύστημα αρχείων του δοχείου.//

Ωστόσο, η εφαρμογή μέσα στο Pod ενδέχεται να μην ελέγχει τη βάση για ενημερώσεις αρχείων, επομένως, ως μέρος μιας αλλαγής διαμόρφωσης, θα πρέπει να αναγκάσετε

την κυκλοφορία του Deployment για να δημιουργήσετε ξανά τα Pod και να φορτώσετε τη νέα διαμόρφωση.

Αυτή δεν είναι μια εξαιρετική επιλογή - δημιουργεί μια διαδικασία ενημέρωσης πολλών σταδίων, με κίνδυνο να ξεχαστούν τα βήματα. Βρείτε μια εναλλακτική προσέγγιση, ώστε όταν εφαρμόζετε αλλαγές σε ένα Secret στο YAML, η κυκλοφορία του Deployment να γίνεται ως μέρος της ίδιας ενημέρωσης.

#### – EXTRA: Η μεταβλητή περιβάλλοντος υπερισχύει

Θα έχετε συχνά πολλές πηγές διαμόρφωσης στις προδιαγραφές του Pod σας. Το Config εξαπλώνεται γρήγορα και είναι λογικό να το κεντράρετε όσο το δυνατόν περισσότερο - εάν όλες οι εφαρμογές σας χρησιμοποιούν την ίδια διαμόρφωση καταγραφής, τότε αποθηκεύστε το σε ένα ConfigMap και χρησιμοποιήστε το σε όλες τις αναπτύξεις.

Η ανάλυση της διαμόρφωσης καθιστά ευκολότερη τη διαχείριση, αλλά πρέπει να κατανοήσετε πώς συγχωνεύονται διαφορετικές πηγές, ώστε να γνωρίζετε τη σειρά προτεραιότητας. Η λογική της εφαρμογής σας καθορίζει την προτεραιότητα διαφορετικών πηγών, αλλά το Kubernetes αποφασίζει την προτεραιότητα για επικαλυπτόμενες μεταβλητές περιβάλλοντος.

Εάν το ίδιο κλειδί εμφανίζεται στα env και envFrom, αυτή είναι η σειρά προτεραιότητας:

- env in Pod spec > envFrom Secrets > envFrom ConfigMaps

```
# save it in a folder called secrets/specs/configurable/
# secrets-overlapping and name it configmap-env.yaml

# it has two settings, one will be replaced by

apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
data:
  Configurable__Release: from-configmap
  Configurable__Extra: from-configmap

# save it in a folder called secrets/specs/configurable/
# secrets-overlapping and name it secret-plain.yaml

# it has two settings, one will be replaced by

apiVersion: v1
kind: Secret
metadata:
```

```

name: configurable-env-plain
labels:
  kubernetes.courselabs.co: secrets
  app: configurable
stringData:
  Configurable__Environment: from-secret
  Configurable__Release: from-secret

# save it in a folder called secrets/specs/configurable/
# secrets-overlapping and name it deployment-env.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          env:
            - name: Configurable__Environment
              value: from-podspec
          envFrom:
            - configMapRef:
                name: configurable-env
            - secretRef:
                name: configurable-env-plain

```

```

kubectl apply -f ./labs/secrets/specs/configurable/secrets-
overlapping

```

Περιηγηθείτε και θα δείτε τη σειρά προτεραιότητας σε δράση.

#### – EXTRA: Διαχείριση ενημερώσεων ρυθμίσεων

Ορισμένες εφαρμογές υποστηρίζουν επανάληψη φόρτωσης της διαμόρφωσης - παρακολουθούν τα αρχεία διαμόρφωσης και αν αλλάξουν τα περιεχόμενα, επαναφορτώνουν αυτόματα

τις ρυθμίσεις.

Άλλοι φορτώνουν τις ρυθμίσεις μόνο κατά την εκκίνηση και αν αλλάξετε τα περιεχόμενα του αρχείου σε ConfigMap ή Secret, η εφαρμογή δεν θα επαναφορτωθεί.

Αυτό ισχύει μόνο για ρυθμίσεις που φορτώνετε με προσαρτήσεις τόμων - οι μεταβλητές περιβάλλοντος είναι στατικές για όλη τη διάρκεια ζωής του Pod.

Εάν γνωρίζετε ότι η εφαρμογή σας κάνει επανάληψη φόρτωσης, τότε η διαδικασία ενημέρωσης είναι απλή, απλώς εφαρμόστε το τροποποιημένο ConfigMap ή Secret και περιμένετε.

Το Kubernetes αποθηκεύει προσωρινά τα περιεχόμενα, οπότε θα χρειαστούν μερικά λεπτά για να λάβουν όλοι οι κόμβοι το πιο πρόσφατο περιεχόμενο και η εφαρμογή να δει την αλλαγή στο σύστημα αρχείων.

- Αναπτύξτε την εφαρμογή Ιστού με μια νέα ρύθμιση:

```
# save it in a folder called secrets/specs/configurable/
# secrets-update and name it deployment-file.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          volumeMounts:
            - name: secret
              mountPath: /app/secrets
              readOnly: true
      volumes:
        - name: secret
          secret:
            secretName: configurable-secret-v1
```

```
# save it in a folder called secrets/specs/configurable/
# secrets-update and name it secret-plain.yaml

apiVersion: v1
kind: Secret
metadata:
  name: configurable-secret-v1
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
stringData:
  secret.json: |-
    {
      "Configurable__ConfigVersion": "v1"
    }
```

```
kubectl apply -f secrets/specs/configurable/secrets-update
```

Ελέγξτε την τιμή στην εφαρμογή ιστού σας - στην ενότητα Secrets.json θα πρέπει να δείτε Configurable\_\_ConfigVersion=v1

- Τώρα αναπτύξτε την ενημερωμένη διαμόρφωση v1-update:

```
# save it in a folder called secrets/specs/configurable/
# secrets-update/v1-update and name it secret-plain.yaml

apiVersion: v1
kind: Secret
metadata:
  name: configurable-secret-v1
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable
stringData:
  secret.json: |-
    {
      "Configurable__ConfigVersion": "v1-update"
    }
```

Ανανεώστε την εφαρμογή και θα εξακολουθεί να δείχνει την παλιά τιμή. Ο χρόνος που απαιτείται για την ενημέρωση εξαρτάται από την πολιτική της **κρυφής μνήμης cache** του Kubernetes και από οποιαδήποτε προσωρινή αποθήκευση κάνει η εφαρμογή.

- Ελέγξτε εάν τα περιεχόμενα του αρχείου ενημερώνονται στο Pod:



```
kubectl exec deploy/configurable -- cat /app/secrets/secret.json
```

- Εάν τα περιεχόμενα του αρχείου ενημερωθούν αλλά η εφαρμογή δεν αλλάξει, ενδέχεται να μην υποστηρίζει επανάληψη φόρτωσης ή να κάνει πολύ επιθετικό caching. Μπορείτε να αναγκάσετε μια ενημέρωση σε όλα τα Pods σε μια ανάπτυξη με την εντολή "rollout restart":

```
kubectl rollout restart deploy/configurable
```

Τώρα ο ιστότοπος θα εμφανίσει την πιο πρόσφατη έκδοση

### – Καθάρισμα

```
kubectl delete all,cm,secret -l kubernetes.courselabs.co=secrets
```

## PersistentVolumes

### – Αποθήκευση δεδομένων εφαρμογής με PersistentVolumes

Το Kubernetes δημιουργεί το σύστημα αρχείων του δοχείου και μπορεί να προσαρτήσει πολλές πηγές. Έχουμε δει ConfigMaps και Secrets που είναι συνήθως προσαρτήσεις μόνο για ανάγνωση, τώρα θα χρησιμοποιήσουμε εγγράψιμους [τόμους](#).

Το storage στο Kubernetes μπορεί να συνδεθεί, επομένως υποστηρίζει διαφορετικούς τύπους - από τοπικούς δίσκους στους κόμβους έως κοινόχρηστα συστήματα αρχείων δικτύου.

Αυτές οι λεπτομέρειες διατηρούνται μακριά από το μοντέλο εφαρμογής χρησιμοποιώντας μια αφηρημένη έννοια - το [PersistentVolumeClaim](#), το οποίο χρησιμοποιεί μια εφαρμογή για να ζητήσει χώρο αποθήκευσης.

### – API Specs

- [PersistentVolumeClaim](#)

Το απλούστερο PersistentVolumeClaim (PVC) μοιάζει με αυτό:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: small-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

Όπως και με τα ConfigMaps και Secrets, χρησιμοποιείτε το όνομα PVC για να αναφέρετε έναν τόμο στην προδιαγραφή Pod σας. Η προδιαγραφή PVC καθορίζει τις απαιτήσεις του:

- AccessModes - περιγράφει εάν ο αποθηκευτικός χώρος είναι μόνο για ανάγνωση ή για ανάγνωση και εγγραφή και εάν είναι αποκλειστικός σε έναν κόμβο ή μπορεί να προσπελαστεί από πολλούς
- resources - η ποσότητα αποθήκευσης που χρειάζεται το PVC

Στην προδιαγραφή Pod μπορείτε να συμπεριλάβετε έναν τόμο PVC για προσάρτηση στο σύστημα αρχείων του δοχείου:

```
volumes:
- name: cache-volume
  persistentVolumeClaim:
    claimName: small-pvc
```

#### – Δεδομένα στο επίπεδο εγγραφής του δοχείου

Πριν φτάσουμε στα PVC, θα εξετάσουμε άλλες επιλογές για τη σύνταξη δεδομένων εφαρμογών στο Kubernetes.

Κάθε δοχείο έχει ένα επίπεδο εγγραφής που μπορεί να χρησιμοποιηθεί για τη δημιουργία και την ενημέρωση αρχείων.

Η εφαρμογή επίδειξης για αυτό το εργαστήριο είναι ένας ιστότοπος υπολογισμού Pi, ο οποίος βρίσκεται μπροστά από έναν διακομιστή μεσολάβησης Nginx. Ο διακομιστής μεσολάβησης αποθηκεύει προσωρινά απαντήσεις από τον ιστότοπο για να βελτιώσει την απόδοση.

- Αναπτύξτε και δοκιμάστε την εφαρμογή:

```
# save it in a folder called persistentvolumes/specs/pi and
# name it nginx.yaml

apiVersion: v1
kind: Service
metadata:
  name: pi-proxy-lb
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-proxy
spec:
  ports:
    - port: 8010
      targetPort: http
      name: http
  selector:
    app: pi-proxy
  type: LoadBalancer
---
```

```

apiVersion: v1
kind: Service
metadata:
  name: pi-proxy-np
  labels:
    app: pi-proxy
    kubernetes.courselabs.co: persistentvolumes
spec:
  ports:
    - port: 8010
      targetPort: http
      name: http
      nodePort: 30010
  selector:
    app: pi-proxy
    type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-proxy
  labels:
    app: pi-proxy
    kubernetes.courselabs.co: persistentvolumes
spec:
  selector:
    matchLabels:
      app: pi-proxy
  template:
    metadata:
      labels:
        app: pi-proxy
        storage: container
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          ports:
            - containerPort: 80
              name: http
          volumeMounts:
            - name: config
              mountPath: "/etc/nginx/"
              readOnly: true
      volumes:
        - name: config
          configMap:
            name: nginx-configmap

```

```

# save it in a folder called persistentvolumes/specs/pi and
  name it nginx-configMap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
  labels:
    kubernetes.courselabs.co: persistentvolumes
data:
  nginx.conf: |-
    user  nginx;
    worker_processes  1;

    error_log  /var/log/nginx/error.log warn;
    pid        /var/run/nginx.pid;

    events {
      worker_connections  1024;
    }

    http {
      proxy_cache_path /tmp levels=1:2 keys_zone=STATIC:10m
        inactive=24h  max_size=1g;

      gzip  on;
      gzip_proxied any;

      map $sent_http_content_type $expires {
        default                off;
        ~image/                 6M;
      }

      server {
        listen 80 default_server;
        listen [::]:80 default_server;

        location / {
          proxy_pass          http://pi-web-internal;
          proxy_set_header    Host $host;
          proxy_cache          STATIC;
          proxy_cache_valid   200 1d;
          proxy_cache_use_stale error timeout
            invalid_header updating
            http_500 http_502 http_503
            http_504;
          add_header           X-Cache
$upstream_cache_status;
          add_header           X-Host      $hostname;

```

```

    }
  }
}

# save it in a folder called persistentvolumes/specs/pi and
# name it web.yaml

apiVersion: v1
kind: Service
metadata:
  name: pi-web-internal
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-web
spec:
  ports:
    - port: 80
      name: http
  selector:
    app: pi-web
    type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  labels:
    app: pi-web
    kubernetes.courselabs.co: persistentvolumes
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - containerPort: 80
              name: http

```

```
kubectl apply -f persistentvolumes/specs/pi
```

Περιηγηθείτε στη διεύθυνση `http://localhost:30010/pi?dp=30000` ή `http://localhost:8010/pi?dp=30000`, θα δείτε ότι χρειάζεται πάνω από ένα δευτερόλεπτο για να υπολογίσετε την απάντηση και να την στείλετε.

- Ανανεώστε και η απόκριση θα είναι άμεση - ελέγξτε την κρυφή μνήμη απόκρισης στο Nginx, μπορείτε να τη δείτε στον φάκελο `/tmp`.

```
kubectl exec deploy/pi-proxy -- ls /tmp
```

- Τώρα σταματήστε τη διαδικασία του δοχείου, η οποία αναγκάζει την επανεκκίνηση του Pod:

```
kubectl exec deploy/pi-proxy -- kill 1
```

```
kubectl get po -l app=pi-proxy
```

Ελέγξτε το φάκελο `/tmp` στο νέο δοχείο και θα δείτε ότι είναι κενό. Ανανεώστε την εφαρμογή Pi και θα χρειαστεί άλλο ένα δευτερόλεπτο για να φορτώσει, επειδή η κρυφή μνήμη είναι κενή, οπότε υπολογίζεται ξανά.

Τα δεδομένα στο επίπεδο εγγραφής δοχείου έχουν τον ίδιο κύκλο ζωής με το δοχείο. Όταν αντικατασταθεί το δοχείο, τα δεδομένα χάνονται.

#### – Αποθηκευτικός χώρος Pod σε τόμους EmptyDir

Οι τόμοι προσαρτούν την αποθήκευση στο σύστημα αρχείων του δοχείου από μια εξωτερική πηγή. Ο απλούστερος τύπος τόμου ονομάζεται EmptyDir - δημιουργεί έναν κενό κατάλογο σε επίπεδο Pod, τον οποίο μπορούν να προσαρτήσουν τα δοχεία Pod.

- Μπορείτε να το χρησιμοποιήσετε για δεδομένα που δεν είναι μόνιμα, αλλά τα οποία θα θέλατε να επιβιώσετε από την επανεκκίνηση. Είναι ιδανικό για τη διατήρηση μιας τοπικής κρυφής μνήμης δεδομένων.

```
# save it in a folder called persistentvolumes/specs/caching-
  proxy-emptydir and name it nginx.yaml
```

```
# it uses an EmptyDir volume, mounting it to the /tmp
  directory
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-proxy
```

```

labels:
  kubernetes.courselabs.co: persistentvolumes
  app: pi-proxy
spec:
  selector:
    matchLabels:
      app: pi-proxy
  template:
    metadata:
      labels:
        app: pi-proxy
        storage: emptydir
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          ports:
            - containerPort: 80
              name: http
          volumeMounts:
            - name: config
              mountPath: "/etc/nginx/"
              readOnly: true
            - name: cache-volume
              mountPath: /tmp
      volumes:
        - name: config
          configMap:
            name: nginx-configmap
        - name: cache-volume
          emptyDir: {}

```

- Αυτή είναι μια αλλαγή στην προδιαγραφή Pod, οπότε θα λάβετε ένα νέο Pod με έναν νέο κενό τόμο καταλόγου:

```

kubectl apply -f persistentvolumes/specs/caching-proxy-
emptydir

kubectl wait --for=condition=Ready pod -l app=pi-proxy,
storage=emptydir

```

Ανανεώστε τη σελίδα σας για να δείτε ξανά τον υπολογισμό Pi - το αποτέλεσμα αποθηκεύεται στην προσωρινή μνήμη και θα δείτε τον φάκελο /tmp να γεμίζει.

- Σταματήστε τη διαδικασία Nginx και το Pod θα επανεκκινηθεί. Ελέγξτε το φάκελο tmp στο νέο δοχείο για να δείτε εάν τα παλιά δεδομένα είναι ακόμα διαθέσιμα.

```
kubectl exec deploy/pi-proxy -- kill 1

kubectl get pods -l app=pi-proxy,storage=emptydir

kubectl exec deploy/pi-proxy -- ls /tmp
```

Ανανεώστε τον ιστότοπο με το νέο δοχείο και φορτώνεται αμέσως.

Εάν διαγράψετε το Pod, τότε το Deployment θα δημιουργήσει μια αντικατάσταση - με έναν νέο τόμο EmptyDir που θα είναι κενός.

Τα δεδομένα σε τόμους EmptyDir έχουν τον ίδιο κύκλο ζωής με το Pod. Όταν αντικατασταθεί το Pod, τα δεδομένα χάνονται.

#### – Εξωτερική αποθήκευση με PersistentVolumeClaims

Η μόνιμη αποθήκευση αφορά τη χρήση τόμων που έχουν ξεχωριστό κύκλο ζωής από την εφαρμογή - έτσι τα δεδομένα παραμένουν όταν αντικαθίστανται τα δοχεία και τα Pods.

Ο χώρος αποθήκευσης στο Kubernetes μπορεί να συνδεθεί και τα cluster παραγωγής θα έχουν συνήθως πολλούς τύπους σε προσφορά, που ορίζονται ως [Κατηγορίες Αποθήκευσης - Storage Classes](#):

```
kubectl get storageclass
```

Θα δείτε ένα μεμονωμένο StorageClass στο Docker Desktop και το k3d, αλλά σε μια υπηρεσία cloud όπως το AKS θα δείτε πολλά, το καθένα με διαφορετικές ιδιότητες (π.χ. ένα γρήγορο SSD που μπορεί να συνδεθεί σε έναν κόμβο ή μια κοινόχρηστη τοποθεσία αποθήκευσης δικτύου που μπορεί να χρησιμοποιηθεί από πολλούς κόμβους).

- Μπορείτε να δημιουργήσετε μια PersistentVolumeClaim με μια StorageClass με όνομα ή να παραλείψετε την κλάση για να χρησιμοποιήσετε την προεπιλογή.

```
# save it in a folder called persistentvolumes/specs/caching-
proxy-pvc and name it pvc.yaml

# it requests 100MB of storage, which a single node can mount
for read-write access

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pi-proxy-pvc
  labels:
    kubernetes.courselabs.co: persistentvolumes
spec:
```



```
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 100Mi
```

```
kubectl apply -f persistentvolumes/specs/caching-proxy-pvc/
pvc.yaml
```

Κάθε StorageClass έχει έναν προμηθευτή που μπορεί να δημιουργήσει τη μονάδα αποθήκευσης κατά παραγγελία.

- Καταγράψτε τα persistent volumes and claims.

```
kubectl apply -f labs persistentvolumes/specs/caching-proxy-
pvc/pvc.yaml
```

```
kubectl get pvc
```

```
kubectl get persistentvolumes
```

Ορισμένοι προμηθευτές δημιουργούν χώρο αποθήκευσης μόλις δημιουργηθεί το PVC - άλλοι περιμένουν να διεκδικηθεί το PVC από ένα Pod.

- Αυτή η προδιαγραφή Deployment ενημερώνει τον διακομιστή μεσολάβησης Nginx για χρήση του PVC:

```
# save it in a folder called persistentvolumes/specs/caching-
proxy-pvc and name it nginx.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-proxy
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-proxy
spec:
  selector:
    matchLabels:
      app: pi-proxy
  template:
    metadata:
      labels:
```

```

    app: pi-proxy
    storage: pvc
  spec:
    containers:
      - image: nginx:1.18-alpine
        name: nginx
        ports:
          - containerPort: 80
            name: http
        volumeMounts:
          - name: config
            mountPath: "/etc/nginx/"
            readOnly: true
          - name: cache-volume
            mountPath: /tmp
    volumes:
      - name: config
        configMap:
          name: nginx-configmap
      - name: cache-volume
        persistentVolumeClaim:
          claimName: pi-proxy-pvc

```

```

kubectl apply -f persistentvolumes/specs/caching-proxy-pvc/

kubectl wait --for=condition=Ready pod -l app=pi-proxy,
  storage=pvc

kubectl get pvc,pv

```

Τώρα το PVC είναι προσαρτημένο και το PersistentVolume υπάρχει με το ζητούμενο μέγεθος και λειτουργία πρόσβασης στο PVC

Το PVC ξεκινάει άδειο. Ανανεώστε την εφαρμογή και θα δείτε τον φάκελο /tmp να γεμίζει.

- Κάντε επανεκκίνηση και, στη συνέχεια, αντικαταστήστε το Pod και επιβεβαιώστε ότι τα δεδομένα στο PVC επιβιώνουν και στα δύο.

```

# force the container to exit
kubectl exec deploy/pi-proxy -- kill 1

kubectl get pods -l app=pi-proxy,storage=pvc

kubectl exec deploy/pi-proxy -- ls /tmp

```

```
# force a rollout to replace the Pod
kubectl rollout restart deploy/pi-proxy

kubectl get pods -l app=pi-proxy,storage=pvc

kubectl exec deploy/pi-proxy -- ls /tmp
```

- Δοκιμάστε ξανά την εφαρμογή και το νέο Pod εξακολουθεί να εξυπηρετεί την απόκριση από την προσωρινή μνήμη, οπότε θα είναι εξαιρετικά γρήγορο.

Τα δεδομένα στο PersistentVolumes έχουν τον δικό τους κύκλο ζωής. Επιβιώνει μέχρι να αφαιρεθεί το PersistentVolumes.

#### – Εργαστηριακή άσκηση 15

Υπάρχει ένας ευκολότερος τρόπος για να αποκτήσετε μόνιμη αποθήκευση, αλλά δεν είναι τόσο ευέλικτο όσο η χρήση ενός PVC και συνοδεύεται από ορισμένες ανησυχίες για την ασφάλεια.

Εκτελέστε ένα απλό Sleep Pod με διαφορετικό τύπο τόμου, που σας δίνει πρόσβαση στη δίσκο root στον κόμβο host όπου εκτελείται το Pod.

Μπορείτε να χρησιμοποιήσετε το sleep Pod για να βρείτε τα αρχεία προσωρινής μνήμης από το Nginx Pod;

#### – EXTRA: Χειροκίνητη διαχείριση PVC με PV

Ορισμένοι πάροχοι διαγράφουν ένα PV όταν διαγραφεί το PVC που το χρησιμοποιεί:

```
kubectl delete -f labs/persistentvolumes/specs/caching-proxy-
pvc/

kubectl get pods -l app=pi-proxy

kubectl get pvc

kubectl get pv
```

Ανάλογα με τον προμηθευτή αποθήκευσής σας, το PV μπορεί να εξακολουθεί να υπάρχει με τα δεδομένα να υπάρχουν ακόμα, επομένως μπορεί να χρησιμοποιηθεί σε άλλο PVC. Ή μπορεί να έχει διαγραφεί.

Όταν χρειάζεστε περισσότερο έλεγχο, μπορείτε να διαχειριστείτε χειροκίνητα τον κύκλο ζωής των PV:

```
# save it in a folder called persistentvolumes/specs/caching-
proxy-pv and name it persistentVolume.yaml
```

```

# it defines a PV which uses local storage, and attaches to a
  node which has a label labs-pvc

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pi-proxy-pv
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-proxy
spec:
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  local:
    path: /volumes/pi-proxy
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: labs-pvc
              operator: Exists

# save it in a folder called persistentvolumes/specs/caching-
  proxy-pv and name it pvc.yaml

# it requests a volume by name, instead of using a Storage
  Class

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pi-proxy-pv
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-proxy
spec:
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  local:
    path: /volumes/pi-proxy
  nodeAffinity:
    required:
      nodeSelectorTerms:

```

```

      - matchExpressions:
        - key: labs-pvc
          operator: Exists

# save it in a folder called persistentvolumes/specs/caching-
# proxy-pv and name it nginx.yaml

# it updates the proxy deployment to use the new PVC

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-proxy
  labels:
    kubernetes.courselabs.co: persistentvolumes
    app: pi-proxy
spec:
  selector:
    matchLabels:
      app: pi-proxy
  template:
    metadata:
      labels:
        app: pi-proxy
        storage: local
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          ports:
            - containerPort: 80
              name: http
          volumeMounts:
            - name: config
              mountPath: "/etc/nginx/"
              readOnly: true
            - name: cache-volume
              mountPath: /tmp
      volumes:
        - name: config
          configMap:
            name: nginx-configmap
        - name: cache-volume
          persistentVolumeClaim:
            claimName: pi-proxy-pvc-local
  tolerations:
    - key: "node-role.kubernetes.io/master"
      operator: "Exists"

```

- Το PV χρησιμοποιεί τον τοπικό τύπο τόμου, που σημαίνει ότι δημιουργείται ως κατάλογος στο δίσκο του κόμβου. Χρησιμοποιεί έναν NodeSelector για να καθορίσει τον κόμβο που πρέπει να χρησιμοποιήσει.

```
kubectl apply -f labs/persistentvolumes/specs/caching-proxy-
pv

kubectl get pvc,pv -l app=pi-proxy

kubectl get pod -l app=pi-proxy,storage=local
```

Το PV και το PVC υπάρχουν - μπορεί να είναι δεσμευμένα ή σε εκκρεμότητα, ανάλογα με το πόσο γρήγορα κάνετε αντιγραφή και επικόλληση, αλλά το Pod θα παραμείνει σε κατάσταση Pending.

- Τα γεγονότα μας λένε γιατί:

```
kubectl describe pod -l app=pi-proxy,storage=local
```

Το PV δεν μπορεί να βρει έναν κόμβο που να ταιριάζει με τον επιλογέα ετικέτας και το μη βοηθητικό μήνυμα Pod μας λέει ότι το Pod δεν μπορεί να προγραμματιστεί επειδή δεν υπάρχει κόμβος όπου μπορεί να ακολουθήσει το PV.

- Προσθέστε μια ετικέτα σε έναν κόμβο και όλα θα επιδιορθωθούν:

```
kubectl label node $(kubectl get nodes -o jsonpath='{.items
[0].metadata.name}') labs-pvc=1

kubectl get nodes --show-labels

kubectl describe pod -l app=pi-proxy,storage=local
```

Τώρα το Pod έχει προγραμματιστεί - αλλά υπάρχει ένα άλλο σφάλμα... Αυτό πρέπει να σκεφτείτε εσείς (αυτό που κάνατε στο εργαστήριο θα έχει βοηθήσει) :

#### – Καθάρισμα

```
kubectl delete all,cm,pvc,pv -l kubernetes.courselabs.co=
persistentvolumes
```

## Namespaces

#### – Απομόνωση των Workloads με Namespaces

Ένα από τα σπουδαία χαρακτηριστικά του Kubernetes είναι ότι μπορείτε να εκτελέσετε οποιονδήποτε τύπο εφαρμογής - πολλοί οργανισμοί θέλουν να μεταφέρουν ολόκληρο το τοπίο εφαρμογών τους στο Kubernetes. Αυτό θα μπορούσε να κάνει τις λειτουργίες

δύσκολες αν δεν υπήρχε τρόπος να διαχωριστεί το cluster, επομένως το Kubernetes δημιούργησε τα [Namespaces](#).

Τα Namespaces είναι πόροι Kubernetes που αποτελούν δοχεία για άλλους πόρους. Μπορείτε να τα χρησιμοποιήσετε για να απομονώσετε φόρτους εργασίας και το πώς θα κάνετε την απομόνωση εξαρτάται από εσάς. Μπορεί να έχετε ένα cluster παραγωγής με διαφορετικό Namespace για κάθε εφαρμογή και ένα cluster μη παραγωγικό με Namespaces για κάθε περιβάλλον (dev, test, UAT).

Εισάγετε κάποια πολυπλοκότητα χρησιμοποιώντας τα Namespaces, αλλά σας παρέχουν πολλές διασφαλίσεις, ώστε να μπορείτε να εκτελείτε με σιγουριά πολλούς φόρτους εργασίας σε ένα μόνο σύμπλεγμα χωρίς να διακυβεύεται η κλίμακα ή η ασφάλεια.

#### – API specs

- [Namespace](#)

Το βασικό YAML για ένα Namespaces είναι εξαιρετικά απλό:

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

Αυτό ήταν :) Το Namespace χρειάζεται ένα όνομα και για κάθε πόρο που θέλετε να δημιουργήσετε μέσα στον χώρο ονομάτων, προσθέτετε του Namespaces στα μεταδεδομένα αυτού του αντικειμένου:

```
apiVersion: v1
kind: Pod
metadata:
  name: whoami
  namespace: whoami
```

Τα Namespace δεν μπορούν να είναι ένθετοι, είναι μια ιεραρχία ενός επιπέδου που χρησιμοποιείται για την κατάτμηση του cluster.

#### – Δημιουργία και χρήση Namespace

- Τα βασικά στοιχεία του ίδιου του Kubernetes εκτελούνται σε Pods και Services - αλλά δεν τα βλέπετε στο Kubectl επειδή βρίσκονται σε ξεχωριστό Namespace:

```
kubectl get pods

kubectl get namespaces

kubectl get pods -n kube-system
```

Η επιλογή `-n` λέει στο `Kubectl` ποιο `Namespace` να χρησιμοποιήσει. Εάν δεν το συμπεριλάβετε, οι εντολές χρησιμοποιούν το προεπιλεγμένο `Namespace`.

Όλα όσα έχουμε αναπτύξει μέχρι στιγμής έχουν δημιουργηθεί στο προεπιλεγμένο `Namespace`.

Αυτό που θα δείτε στο `kube-system` εξαρτάται από τη διανομή `Kubernetes`, αλλά θα πρέπει να περιλαμβάνει ένα διακομιστή `DNS Pod`.

Μπορείτε να εργαστείτε με τους πόρους του συστήματος με τον ίδιο τρόπο όπως οι δικές σας εφαρμογές, αλλά πρέπει να συμπεριλάβετε το `Namespace` στην εντολή `Kubectl`.

- Εκτυπώστε τα logs του διακομιστή `DNS` συστήματος.

```
kubectl logs -l k8s-app=kube-dns
kubectl logs -l k8s-app=kube-dns -n kube-system
```

- Η προσθήκη `Namespace` σε κάθε εντολή είναι χρονοβόρα και το `Kubectl` διαθέτει περιβάλλοντα που σας επιτρέπουν να ορίσετε το προεπιλεγμένο `Namespace` για εντολές:

```
kubectl config get-contexts
cat ~/.kube/config
```

Τα `Contexts` είναι επίσης ο τρόπος εναλλαγής μεταξύ `cluster`. Οι λεπτομέρειες διακομιστή `API` του `cluster` αποθηκεύονται στο αρχείο `kubeconfig`.

Μπορείτε να δημιουργήσετε ένα νέο περιβάλλον για να οδηγείτε σε ένα απομακρυσμένο `cluster` ή ένα συγκεκριμένο `Namespace` σε ένα `cluster`. Τα `contexts` περιλαμβάνουν λεπτομέρειες ελέγχου ταυτότητας, επομένως θα πρέπει να αντιμετωπίζονται προσεκτικά.

- Μπορείτε να ενημερώσετε τις ρυθμίσεις για το περιβάλλον σας για να αλλάξετε το `Namespace`:

```
kubectl config set-context --current --namespace kube-system
```

Όλες οι εντολές `Kubectl` λειτουργούν ενάντια στο `cluster` και το `Namespace` στο τρέχον περιβάλλον.

- Εκτυπώστε ορισμένες λεπτομέρειες `Pod` από το `Namespace` του συστήματος και τον προεπιλεγμένο `Namespace`.



```
kubectl get po
kubectl logs -l k8s-app=kube-dns
kubectl get po -n default
```

- Επαναφέρετε το περιβάλλον σας στο προεπιλεγμένο Namespace, ώστε να μην κάνουμε κατά λάθος κάτι επικίνδυνο.

```
kubectl config set-context --current --namespace default
```

#### – Ανάπτυξη αντικειμένων σε Namespaces

Οι προδιαγραφές του αντικειμένου μπορούν να περιλαμβάνουν το Namespace προορισμού στο YAML. Εάν δεν έχει καθοριστεί, μπορείτε να ορίσετε το Namespace με το Kubectl.

```
# save it in a folder called namespaces/specs and name it
sleep-pod.yaml

# it defines a Pod with no namespace, so Kubectl decides the
namespace - using the default for the context, or an
explicit namespace

apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: namespaces
    app: sleep
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
```

- Αναπτύξτε την προδιαγραφή Pod στο namespace/specs/sleep-pod.yaml στο προεπιλεγμένο Namespace και στο Namespace συστήματος.

```
kubectl apply -f namespaces/specs/sleep-pod.yaml -n default
kubectl apply -f namespaces/specs/sleep-pod.yaml -n kube-
system
kubectl get pods -l app=sleep --all-namespaces
```

Η πρόσβαση στο Namespace μπορεί να περιοριστεί με στοιχεία ελέγχου πρόσβασης, αλλά στο περιβάλλον προγραμματιστή σας θα έχετε δικαιώματα διαχειριστή cluster, ώστε να μπορείτε να βλέπετε τα πάντα.

Εάν χρησιμοποιείτε Namespaces για την απομόνωση εφαρμογών, θα συμπεριλάβετε την προδιαγραφή Namespace με το μοντέλο και θα καθορίσετε το Namespace σε όλα τα αντικείμενα:

```
# save it in a folder called namespaces/specs/whoami and name
it 01-namespace.yaml

# it defines the namespace

apiVersion: v1
kind: Namespace
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: namespaces

# save it in a folder called namespaces/specs/whoami and name
it deployment.yaml

# it defines a Deployment for the namespace

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  namespace: whoami
spec:
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04

# save it in a folder called namespaces/specs/whoami and name
it services.yaml

# it defines Services; the label selectors only apply to Pods
in the same namespace as the Service
```

```

apiVersion: v1
kind: Service
metadata:
  name: whoami-lb
  namespace: whoami
  labels:
    kubernetes.courselabs.co: namespaces
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8010
      targetPort: 80
    type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: whoami-np
  namespace: whoami
  labels:
    kubernetes.courselabs.co: namespaces
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8010
      targetPort: 80
      nodePort: 30010
    type: NodePort

```

Το Kubectl μπορεί να αναπτύξει όλο το YAML σε έναν φάκελο, αλλά δεν ελέγχει τα αντικείμενα για εξαρτήσεις και δεν τα δημιουργεί με τη σωστή σειρά. Κυρίως αυτό είναι εντάξει λόγω της χαλαρά συζευγμένης αρχιτεκτονικής - Οι υπηρεσίες μπορούν να δημιουργηθούν πριν από μια ανάπτυξη και αντίστροφα.

- Αλλά τα Namespaces πρέπει να υπάρχουν για να μπορέσουν να δημιουργηθούν αντικείμενα σε αυτούς, επομένως ο χώρος ονομάτων YAML ονομάζεται 01\_namespaces.yaml για να διασφαλιστεί ότι θα δημιουργηθεί πρώτος (το Kubectl επεξεργάζεται τα αρχεία με τη σειρά ανά όνομα αρχείου).

```

kubectl apply -f namespaces/specs/whoami

kubectl get svc -n whoami

```

Η χρήση Namespace για την ομαδοποίηση εφαρμογών ή περιβαλλόντων σημαίνει ότι τα αντικείμενα ανώτατου επιπέδου (Deployments, Υπηρεσίες, ConfigMaps) δεν χρειάζονται τόσες πολλές ετικέτες. Θα εργαστείτε μαζί τους μέσα σε ένα Namespace, ώστε να μην χρειάζεστε ετικέτες για φιλτράρισμα.

Εδώ είναι μια άλλη εφαρμογή όπου όλα τα στοιχεία θα απομονωθούν στον δικό τους Namespace:

```
# save it in a folder called namespaces/specs/configurable
  and name it 01-namespace.yaml

# the new namespace

apiVersion: v1
kind: Namespace
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: namespaces

# save it in a folder called namespaces/specs/configurable
  and name it configmap.yaml

# ConfigMap with app settings

apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
  namespace: configurable
data:
  Configurable__Release: 24.01.2
  Configurable__Environment: uat

# save it in a folder called namespaces/specs/configurable
  and name it deployment.yaml

# Deployment which references the ConfigMap. Config objects
  need to be in the same namespace as the Pod.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  namespace: configurable
spec:
  selector:
    matchLabels:
      app: configurable
```

```
template:
  metadata:
    labels:
      app: configurable
  spec:
    containers:
      - name: app
        image: sixeyed/configurable:21.04
        envFrom:
          - configMapRef:
              name: configurable-env
```

- Αναπτύξτε την εφαρμογή και χρησιμοποιήστε το Kubectl για να παραθέσετε τα Deployments σε όλα τα Namespaces.

```
kubectl apply -f namespaces/specs/configurable

kubectl get deploy -A --show-labels
```

- Μπορείτε να χρησιμοποιήσετε το Kubectl μόνο με ένα Namespace ή όλα τα Namespaces, επομένως μπορεί να θέλετε πρόσθετες ετικέτες για αντικείμενα όπως οι Υπηρεσίες, ώστε να μπορείτε να κάνετε λίστα με όλα τα Namespaces και να φιλτράρετε ανά ετικέτα:

```
kubectl get svc -A -l kubernetes.courselabs.co=namespaces
```

## – Namespaces και DNS Υπηρεσίας

Η δικτύωση στο Kubernetes είναι επίπεδη, επομένως κάθε Pod σε οποιονδήποτε Namespace μπορεί να έχει πρόσβαση σε άλλο Pod μέσω της διεύθυνσης IP του.

Οι υπηρεσίες έχουν πNamespace, επομένως εάν θέλετε να βρείτε τη διεύθυνση IP για μια Υπηρεσία χρησιμοποιώντας το όνομά της DNS, μπορείτε να συμπεριλάβετε τον χώρο ονομάτων:

- το whoami-np είναι ένα τοπικό όνομα τομέα, επομένως θα αναζητήσει μόνο την Υπηρεσία whoami-np στο ίδιο Namespace όπου εκτελείται η αναζήτηση
- το whoami-np.whoami.svc.cluster.local είναι ένα πλήρως πιστοποιημένο όνομα τομέα (FQDN), το οποίο θα αναζητήσει την Υπηρεσία whoami-np στο Namespace whoami
- Εκτελέστε ορισμένα ερωτήματα DNS μέσα στο sleep Pod:

```
# this won't return an address - the Service is in a
  different namespace:
kubectl exec pod/sleep -- nslookup whoami-np

# this includes the namespace, so it will return an IP
  address:
```

```
kubectl exec pod/sleep -- nslookup whoami-np.whoami.svc.
cluster.local
```

Ως βέλτιστη πρακτική θα πρέπει να χρησιμοποιείτε FQDN για την επικοινωνία μεταξύ των στοιχείων. Κάνει την ανάπτυξή σας λιγότερο ευέλικτη, επειδή δεν μπορείτε να αλλάξετε τον Namespace χωρίς να αλλάξετε επίσης τις ρυθμίσεις της εφαρμογής, αλλά αφαιρεί ένα δυνητικά μπερδεμένο σημείο αποτυχίας.

#### – Εφαρμογή limit σε πόρους

Τα Namespace δεν προορίζονται μόνο για τη λογική ομαδοποίηση στοιχείων, αλλά μπορείτε επίσης να επιβάλετε όρια - limits σε ένα Namespace για να περιορίσετε τους διαθέσιμους πόρους.

Αυτό διασφαλίζει ότι οι εφαρμογές δεν χρησιμοποιούν όλη την επεξεργαστική ισχύ του cluster, εξαλείφοντας άλλες εφαρμογές. Τα **όρια πόρων** και τα **εύρη ορίων** σε επίπεδο Namespace συνεργάζονται με τα **όρια πόρων και τα αιτήματα** σε επίπεδο Pod.

Η εφαρμογή Ιστού υπολογισμού Πι έχει υψηλή υπολογιστική ισχύ, επομένως για να διατηρήσουμε το cluster μας χρησιμοποιήσιμο για άλλες εφαρμογές, θα το αναπτύξουμε σε ένα νέο χNamespace με εφαρμοσμένο όριο CPU:

```
# save it in a folder called namespaces/specs/pi and name it
02-cpu-limit-quota.yaml

# it defines a quota which sets a total limits of 4 CPU cores
across all Pods in the namespace

apiVersion: v1
kind: ResourceQuota
metadata:
  name: cpu-limit
  namespace: pi
spec:
  hard:
    requests.cpu: 3500m
    limits.cpu: 4000m

# save it in a folder called namespaces/specs/pi and name it
web-deployment.yaml

# it defines a Deployment with one Pod which has a limit of
125 millicores (0.125 of one core)

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
```

```

namespace: pi
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
        cpu: min
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - containerPort: 80
              name: http
          resources:
            requests:
              cpu: 125m
              memory: 250Mi
            limits:
              cpu: 125m
              memory: 500Mi

```

Τα αιτήματα πόρων καθορίζουν πόση μνήμη ή CPU θα ήθελε να εκχωρηθεί το Pod όταν δημιουργηθεί. Τα όρια πόρων καθορίζουν τη μέγιστη μνήμη ή CPU στην οποία μπορεί να έχει πρόσβαση το Pod.

- Δεν υπάρχει διακομιστής μεσολάβησης Nginx για αυτήν την έκδοση της εφαρμογής Ρί και η κατανομή της CPU είναι πολύ μικρή, επομένως οι υπολογισμοί θα είναι αργοί.

```
kubectl apply -f namespaces/specs/pi
```

```
kubectl -n pi get quota
```

```
kubectl -n pi get po
```

Δοκιμάστε την εφαρμογή στη διεύθυνση <http://localhost:30030/pi?dp=30000>.

Ας την επιταχύνουμε:

```
# save it in a folder called namespaces/specs/pi/mid-cpu and
  name it web-deployment.yaml
```

```
# it bumps the processing power to 2.5 CPU cores

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  namespace: pi
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
        cpu: mid
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - containerPort: 80
              name: http
          resources:
            requests:
              cpu: 2500m
              memory: 500Mi
            limits:
              cpu: 2500m
              memory: 500Mi
```

- Ενημερώστε την εφαρμογή και ελέγξτε τους πόρους που έχουν οριστεί στο Pod.

```
kubectl apply -f namespaces/specs/pi/mid-cpu
```

```
kubectl describe po -l app=pi-web,cpu=mid -n pi
```

Ανανεώστε την <http://localhost:30030/pi?dp=30000>.

Δοκιμάστε και μεταβείτε στο μέγιστο της CPU - max-cpu/web-deployment.yaml  
 θέτει ένα όριο 4,5 πυρήνων CPU, το οποίο είναι μεγαλύτερο από το όριο για το Namespace:

```
# save it in a folder called namespaces/specs/pi/max-cpu and
  name it web-deployment.yaml
```



```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  namespace: pi
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
        cpu: max
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - containerPort: 80
              name: http
          resources:
            requests:
              cpu: 3500m
              memory: 750Mi
            limits:
              cpu: 4500m
              memory: 1000Mi

```

```

kubectl apply -f labs/namespaces/specs/pi/max-cpu

kubectl -n pi get rs -l app=pi-web

kubectl -n pi describe rs -l app=pi-web,cpu=max

```

Το νέο ReplicaSet δεν κλιμακώνεται ποτέ στον επιθυμητό αριθμό. Θα δείτε ένα ωραίο, σαφές σφάλμα που θα σας λέει ότι το όριο έχει ξεπεραστεί. Το Kubernetes θα συνεχίσει να προσπαθεί, σε περίπτωση που αλλάξει το όριο

#### – Εργαστηριακή άσκηση 16

Αυτή η υπηρεσία Pi διαρκεί πολύ για να εκτελεστεί, αποδίδει καλύτερα όταν την εκτελείτε με έναν αντίστροφο διακομιστή μεσολάβησης για να αποθηκεύσετε τις απαντήσεις στην κρυφή μνήμη.

Προσθέστε έναν διακομιστή μεσολάβησης προσωρινής αποθήκευσης μπροστά από την εφαρμογή Pi και να γνωρίζετε ότι η ομάδα λειτουργιών θέλει όλους τους διακομιστές μεσολάβησης σε ένα Namespace που ονομάζεται front-end.

Μπορείτε να χρησιμοποιήσετε τη ρύθμιση αντίστροφου διακομιστή μεσολάβησης από εδώ ως σημείο εκκίνησης, αλλά οι προδιαγραφές δεν περιλαμβάνουν Namespace:

```
# save it in a folder called namespaces/specs/reverse-proxy
and name it nginx.yaml

# it is the configuration, using http://pi-web-np.pi.svc.
cluster.local:8030 as the proxy_pass setting

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-configmap
data:
  nginx.conf: |-
    user  nginx;
    worker_processes  1;

    error_log  /var/log/nginx/error.log warn;
    pid        /var/run/nginx.pid;

    events {
        worker_connections  1024;
    }

    http {
        proxy_cache_path /tmp levels=1:2 keys_zone=STATIC:10m
        inactive=24h  max_size=1g;

        gzip  on;
        gzip_proxied any;

        map $sent_http_content_type $expires {
            default                off;
            ~image/                 6M;
        }

        server {
            listen 80 default_server;
            listen [::]:80 default_server;

            location / {

                proxy_pass          http://pi-web-internal;
```

```

        proxy_set_header    Host $host;
        proxy_cache          STATIC;
        proxy_cache_valid   200 1d;
        proxy_cache_use_stale error timeout
    invalid_header updating
                                http_500 http_502 http_503

    http_504;
        add_header          X-Cache
    $upstream_cache_status;
        add_header          X-Host      $hostname;
    }
}
}
---
apiVersion: v1
kind: Service
metadata:
  name: pi-proxy-np
spec:
  ports:
    - port: 8040
      targetPort: http
      name: http
      nodePort: 30040
  selector:
    app: pi-proxy
    type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-proxy
spec:
  selector:
    matchLabels:
      app: pi-proxy
  template:
    metadata:
      labels:
        app: pi-proxy
        storage: container
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          ports:
            - containerPort: 80
              name: http
          volumeMounts:

```

```

      - name: config
        mountPath: "/etc/nginx/"
        readOnly: true
    volumes:
      - name: config
        configMap:
          name: nginx-configmap

```

Περιηγηθείτε στο `http://localhost:30040` και θα βρείτε ένα σφάλμα - θα πρέπει να διορθώσετε τη διαμόρφωση για να λειτουργήσει.

#### – EXTRA: Εναλλαγή Context

Όταν εργάζεστε με πολλά Kubernetes clusters το καθένα με πολλά Namespace, είναι πολύ δύσκολο να τα διαχειριστείτε.

Υπάρχει ένα εξαιρετικό εργαλείο που ονομάζεται `kubectx` που βοηθάει σε αυτό - είναι cross-platform και σας επιτρέπει εύκολα να κάνετε εναλλαγή μεταξύ clusters, μαζί με το συνεργαζόμενο εργαλείο "kubens" για εναλλαγή Namespaces.

- Είναι πολύ χρήσιμα εργαλεία καθώς χρησιμοποιείτε περισσότερο το Kubernetes, μπορείτε να τα εγκαταστήσετε από τα επίσημα [releases](#) ή με:

```

# Windows
choco install kubectx kubens

# macOS
brew install kubectx kubens

```

- Στη συνέχεια, χρησιμοποιήστε το για να διαχειριστείτε Namespaces όπως αυτό:

```

# list all namespaces
kubens

# switch to pi
kubens pi

# toggle back to the previous namespace:
kubens -

```

- Χρησιμοποιώ aliases σε όλα μου τα shells:

```

alias d="docker"
alias k="kubectl"
alias kx="kubectx"
alias kn="kubens"

```

- Έτσι, η τυπική ροή εργασίας μου είναι:

```
kx <client-cluster>
kn <namespace>
k etc.

kx -
```

#### – Καθάρισμα

```
# deleting a namespace deletes everything inside it:
kubectl delete ns -l kubernetes.courselabs.co=namespaces

# which just leaves the sleep Pods:
kubectl delete po -A -l kubernetes.courselabs.co=namespaces
```

### 3.3 Προχωρημένη Μοντελοποίηση Εφαρμογών

#### Role-based Access Control (RBAC)

##### – Role-Based Access Control

Το Kubernetes υποστηρίζει λεπτομερή έλεγχο πρόσβασης, ώστε να μπορείτε να αποφασίσετε ποιος έχει άδεια να εργαστεί με πόρους στο cluster σας και τι μπορεί να κάνει με αυτούς.

Υπάρχουν δύο μέρη στο [RBAC](#), τα δικαιώματα αποσύνδεσης και ποιος έχει τα δικαιώματα - που σας επιτρέπει να μοντελοποιήσετε την ασφάλεια με έναν διαχειρίσιμο αριθμό αντικειμένων:

- τα Roles ορίζουν δικαιώματα πρόσβασης για πόρους (όπως Pods και Secrets), επιτρέποντας συγκεκριμένες ενέργειες (όπως δημιουργία και διαγραφή)
- τα RoleBindings εκχωρεί τα δικαιώματα σε ένα role πάνω σε ένα θέμα, το οποίο θα μπορούσε να είναι ένας χρήστης του Kubectl ή μια εφαρμογή που εκτελείται σε ένα Pod.

Τα Roles και τα RoleBindings ισχύουν για αντικείμενα σε συγκεκριμένο Namespace. Υπάρχουν επίσης ClusterRole και ClusterRoleBindings που έχουν παρόμοιο API και ασφαλή πρόσβαση σε αντικείμενα σε όλα τα Namespace.

##### – API specs

- [Role](#)
- [RoleBinding](#)
- [ClusterRole](#)
- [ClusterRoleBinding](#)

##### – Role και RoleBinding API spec

Τα Roles περιέχουν ένα σύνολο δικαιωμάτων ως κανόνες:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-viewer
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]

```

- **apiGroups** - το Namespace του πόρου, τον οποίο χρησιμοποιείτε στο **apiVersion** στα μεταδεδομένα αντικειμένου. Τα **pods** είναι απλώς **v1**, επομένως το **apiGroup** είναι κενό, τα **Deployments** είναι **apps/v1**, επομένως το **apiGroup** θα είναι **apps**
- **resources** - ο τύπος του πόρου, τον οποίο χρησιμοποιείτε ως είδος στα μεταδεδομένα αντικειμένου
- **verbs** - οι ενέργειες που επιτρέπει το **Role** να εκτελεστούν στον πόρο

Αυτό το **Role** ισοδυναμεί με το **Kubectl** να πάρει **pods** και να περιγράψει τα δικαιώματα **pod** στον προεπιλεγμένο **Namespace**.

Τα **RoleBindings** εφαρμόζουν έναν **Role** σε ένα σύνολο θεμάτων:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: student-pod-viewer
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-viewer
subjects:
- kind: User
  name: student@courselabs.co
  namespace: default

```

- το **Namespace** πρέπει να ταιριάζει με τον **Namespace** στο **Role**
- το **roleRef** αναφέρεται στο **Role** με το όνομα
- τα **subjects** είναι ή **ServiceAccounts**, ή **Groups** ή **User** που έχουν το **Role** που εφαρμόζεται - μπορούν να βρίσκονται σε διαφορετικά **Namespace**.

– **\*Κάντε αυτό πρώτα εάν χρησιμοποιείτε Docker Desktop \***

Υπάρχει ένα [σφάλμα στην προεπιλεγμένη ρύθμιση RBAC](#) σε παλαιότερες εκδόσεις του **Docker Desktop**, πράγμα που σημαίνει ότι τα δικαιώματα δεν εφαρμόζονται σωστά. Εάν χρησιμοποιείτε το **Kubernetes** στο **Docker Desktop** έκδοση 4.2 ή παλαιότερη έκδοση, εκτελέστε αυτό για να διορθώσετε το σφάλμα:

```
# on Docker Desktop for Mac (or WSL2 on Windows):
sudo chmod +x ./scripts/fix-rbac-docker-desktop.sh
./scripts/fix-rbac-docker-desktop.sh

# OR on Docker Desktop for Windows (PowerShell):
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
    Process -Force
./scripts/fix-rbac-docker-desktop.ps1
```

To Docker Desktop 4.3.0 διορθώνει το πρόβλημα, επομένως, εάν εκτελέσετε την εντολή και δείτε Σφάλμα από τον διακομιστή (NotFound): clusterrolebindings.rbac.authorization.k8s.io "docker-for-desktop-binding" not found - αυτό σημαίνει ότι η έκδοση σας δεν έχει το σφάλμα και είστε έτοιμοι.

#### – Εξασφάλιση πρόσβασης API με λογαριασμούς υπηρεσίας

Ο έλεγχος ταυτότητας για πρόσβαση end-user διαχειρίζεται εκτός του Kubernetes, επομένως θα χρησιμοποιήσουμε το RBAC για εσωτερική πρόσβαση στο cluster - εφαρμογές που εκτελούνται στο Kubernetes.

Θα χρησιμοποιήσουμε μια απλή εφαρμογή Ιστού που συνδέεται με τον διακομιστή Kubernetes API για να λάβουμε μια λίστα με τα Pods, τα εμφανίζει και σας επιτρέπει να τα διαγράψετε.

- Δημιουργήστε το παρακάτω:

```
# save it in a folder called rbac/specs and name it sleep.
yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: rbac
spec:
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
    spec:
      containers:
        - name: sleep
          image: kiamol/ch03-sleep
```

```
kubectl apply -f rbac/specs/sleep.yaml
```

- Η αρχική προδιαγραφή για την εφαρμογή Ιστού δεν περιλαμβάνει κανόνες RBAC, αλλά περιλαμβάνει έναν συγκεκριμένο λογαριασμό ασφαλείας για το Pod:

```
# save it in a folder called rbac/specs/kube-explorer and
# name it deployment.yaml

# it creates a ServiceAccount and sets the Pod spec to use
# that ServiceAccount

apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-explorer
  labels:
    kubernetes.courselabs.co: rbac
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-explorer
  labels:
    kubernetes.courselabs.co: rbac
spec:
  selector:
    matchLabels:
      app: kube-explorer
  template:
    metadata:
      labels:
        app: kube-explorer
    spec:
      serviceAccountName: kube-explorer
      containers:
        - image: kiamol/ch17-kube-explorer
          name: web
          ports:
            - containerPort: 80
              name: http
          env:
            - name: ASPNETCORE_ENVIRONMENT
              value: Development
```

- Αναπτύξτε τους πόρους

```
kubectl apply -f rbac/specs/kube-explorer
```



Περιηγηθείτε στην εφαρμογή στη διεύθυνση `http://localhost:8010` ή `http://localhost:30010`

Θα δείτε ένα σφάλμα. Η εφαρμογή προσπαθεί να συνδεθεί στο Kubernetes REST API για να λάβει μια λίστα με Pods, αλλά λαμβάνει ένα μήνυμα σφάλματος 403 Forbidden.

Το Kubernetes συμπληρώνει αυτόματα ένα διακριτικό ελέγχου ταυτότητας στο Pod, το οποίο χρησιμοποιεί η εφαρμογή για να συνδεθεί στον διακομιστή API.

- Εκτυπώστε όλες τις λεπτομέρειες σχετικά με το kube-explorer Pod.

```
# you can get the Pod ID or use the label
kubectl describe pod -l app=kube-explorer
```

Θα δείτε έναν τόμο τοποθετημένο στο `/var/run/secrets/kubernetes.io/serviceaccount` - δεν περιλαμβάνεται στις προδιαγραφές του Pod, είναι προεπιλογή του Kubernetes για να τον προσθέσετε.

```
kubectl exec deploy/kube-explorer -- cat /var/run/secrets/
kubernetes.io/serviceaccount/token
```

Επομένως, η εφαρμογή έχει πιστοποιηθεί και επιτρέπεται να χρησιμοποιεί το API, αλλά ο λογαριασμός δεν είναι εξουσιοδοτημένος να καταχωρεί τα Pods. Οι αρχές ασφαλείας - ServiceAccounts, Groups and Users - ξεκινούν χωρίς δικαιώματα και πρέπει να τους παραχωρηθεί πρόσβαση σε πόρους.

- Μπορείτε να ελέγξετε τα δικαιώματα ενός χρήστη με την εντολή "auth can-i":

```
kubectl auth can-i get pods -n default --as system:
serviceaccount:default:kube-explorer
```

Η εντολή του λειτουργεί για Users και ServiceAccounts - το Αναγνωριστικό των ServiceAccounts περιλαμβάνει το Namespace και το όνομα.

Οι κανόνες RBAC εφαρμόζονται όταν υποβάλλεται αίτημα στον διακομιστή API, επομένως μπορούμε να διορθώσουμε αυτήν την εφαρμογή αναπτύσσοντας ένα Role και ένα RoleBinding:

```
# save it in a folder called rbac/specs/kube-explorer/rbac-
namespace and name it role-pod-manager.yaml

# it creates a Role with permissions to list and delete Pods
in the default namespace
```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-manager
  namespace: default # this is the scope where the role
    applies
  labels:
    kubernetes.courselabs.co: rbac
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "delete"]

# save it in a folder called rbac/specs/kube-explorer/
# rolebinding-pod-manager-sa.yaml

# it creates a RoleBinding applying the new Role to the app's
# Service Account

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kube-explorer-pod-manager
  namespace: default # needs to match the ns in the role
  labels:
    kubernetes.courselabs.co: rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-manager
subjects:
- kind: ServiceAccount
  name: kube-explorer
  namespace: default # the subject can be in a different ns

```

- Αναπτύξτε την προδιαγραφή βεβαιωθείτε ότι ο ServiceAccounts έχει πλέον άδεια.

```

kubectl apply -f rbac/specs/kube-explorer/rbac-namespace

kubectl auth can-i get pods -n default --as system:
  serviceaccount:default:kube-explorer

```

Τώρα η εφαρμογή έχει τα δικαιώματα που χρειάζεται. Ανανεώστε τον ιστότοπο και θα δείτε μια λίστα Pod. Μπορείτε να διαγράψετε το Sleep Pod και, στη συνέχεια, να επιστρέψετε στην κύρια σελίδα και θα δείτε ένα ανταλλακτικό Pod που δημιουργήθηκε από το ReplicaSet.

– **Εκχώρηση αδειών σε όλο το σύμπλεγμα**

Η δέσμευση ρόλων περιορίζει την πρόσβαση στον προεπιλεγμένο Namespace, ο ίδιος ServiceAccount δεν μπορεί να δει τα Pods στο Namespace του συστήματος.//

- Ελέγξτε εάν ο λογαριασμός kube-explorer μπορεί να λάβει Pods στο Namespace του kube-system.

```
kubectl auth can-i get pods -n kube-system --as system:serviceaccount:default:kube-explorer
```

Μπορείτε να παραχωρήσετε πρόσβαση σε Pods σε κάθε Namespace με περισσότερα Roles και RoleBindings, αλλά αν θέλετε τα δικαιώματα να ισχύουν σε όλους τα Namespace, μπορείτε να χρησιμοποιήσετε ένα ClusterRole και ένα ClusterRoleBinding:

```
# save it in a folder called rbac/specs/kube-explorer/rbac-cluster and name it clusterrole-pod-reader.yaml

# it sets Pod permissions for the cluster; note there is no namespace in the metadata

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-reader
  labels:
    kubernetes.courselabs.co: rbac
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]

# save it in a folder called rbac/specs/kube-explorer/rbac-cluster and name it clusterrolebinding-pod-reader-sa.yaml

# it applies the role to the app's ServiceAccount

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-explorer-pod-reader
  labels:
    kubernetes.courselabs.co: rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pod-reader
subjects:
- kind: ServiceAccount
  name: kube-explorer
```

```
namespace: default # can be any ns
```

- Αναπτύξτε τις προδιαγραφές στα και επαληθεύστε ότι η SA μπορεί να λάβει Pods στο Namespace του συστήματος, αλλά δεν μπορεί να τα διαγράψει.

```
kubectl apply -f rbac/specs/kube-explorer/rbac-cluster/

kubectl auth can-i get pods -n kube-system --as system:
serviceaccount:default:kube-explorer

kubectl auth can-i delete pods -n kube-system --as system:
serviceaccount:default:kube-explorer
```

Περιηγηθείτε στην εφαρμογή με Namespace στη συμβολοσειρά ερωτημάτων, π.χ. <http://localhost:8010/?ns=kube-system> ή <http://localhost:30010/?ns=kube-system>

Η εφαρμογή μπορεί να δει τα Pods σε άλλα Namespaces τώρα.

Τα δικαιώματα RBAC ελέγχονται λεπτομερώς. Η εφαρμογή έχει πρόσβαση μόνο σε πόρους Pod - εάν κάνετε κλικ στη σύνδεση Λογαριασμοί υπηρεσίας, η εφαρμογή εμφανίζει ξανά το σφάλμα 403 Forbidden.

#### – Εργαστηριακή άσκηση 17

Πρέπει να είστε εξουσιοδοτημένοι με το RBAC. Σίγουρα θα έχετε περιορισμένα δικαιώματα σε cluster παραγωγής και αν χρειάζεστε νέα πρόσβαση, θα την αποκτήσετε πιο γρήγορα εάν δώσετε στον διαχειριστή ένα Role και ένα RoleBinding για ό,τι χρειάζεστε.

Εξασκηθείτε με την ανάπτυξη νέων κανόνων RBAC, ώστε η προβολή ServiceAccount στην εφαρμογή kube-explorer να λειτουργεί σωστά, για αντικείμενα στον προεπιλεγμένο Namespace.

Ω - κάτι ακόμα :) Η τοποθέτηση του διακριτικού ServiceAccount στο Pod είναι προεπιλεγμένη συμπεριφορά, αλλά οι περισσότερες εφαρμογές δεν χρησιμοποιούν τον διακομιστή API Kubernetes. Είναι ένα πιθανό ζήτημα ασφάλειας, επομένως μπορείτε να τροποποιήσετε το Sleep Pod ώστε να μην έχει τοποθετηθεί διακριτικό.

#### – EXTRA: Διαχείριση δικαιωμάτων end-user

Το Kubernetes ενσωματώνεται με άλλα συστήματα για έλεγχο ταυτότητας end-user, αλλά σε μια ρύθμιση προγραμματιστή μπορείτε να δημιουργήσετε πιστοποιητικά για χρήστες και να εφαρμόσετε κανόνες RBAC για αυτούς. Αυτή δεν είναι καθημερινή δουλειά, αλλά αν σας ενδιαφέρει μπορείτε να εργαστείτε με τις ασκήσεις στο [RBAC για χρήστες](#).

#### – EXTRA: Εφαρμογή ClusterRoles σε συγκεκριμένα Namespace

Υπάρχουν ενσωματωμένα ClusterRoles που παρέχουν ένα καλό σημείο εκκίνησης για γενική πρόσβαση - συμπεριλαμβανομένης του view, edit and admin.

Τα ClusterRoles μπορούν να συνδεθούν σε θέματα σε όλο το cluster με ClusterRoleBinding ή σε συγκεκριμένα Namespaces με RoleBinding:

```
# save it in a folder called rbac/specs/user and name it
rolebinding-edit-default.yaml

# it applies the edit ClusterRole to the new user, restricted
to the default namespace

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: reader-edit-default
  namespace: default # this is the scope where the binding
    applies
  labels:
    kubernetes.courselabs.co: rbac
roleRef:
  kind: ClusterRole
  name: edit
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: User
  name: reader@courselabs.co
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl apply -f labs/rbac/specs/user/

kubectl auth can-i delete po/user-cert-generator --as
  reader@courselabs.co

kubectl delete pod user-cert-generator --context labreader
```

Ο χρήστης μπορεί τώρα να διαγράψει Pods στο προεπιλεγμένο Namespace. Εάν υπήρχαν άλλοι χρήστες στην ίδια ομάδα, τότε δεν θα είχαν αυτήν την άδεια - δεσμεύεται ειδικά για τον χρήστη.

- Αλλά το ClusterRole περιορίζεται στο προεπιλεγμένο Namespace:

```
kubectl delete pod --all -n kube-system --context labreader
```

## – Καθάρισμα

```
kubect1 delete pod,deploy,svc,serviceaccount,role,rolebinding
,clusterrole,clusterrolebinding -A -l kubernetes.
courselabs.co=rbac
```

## DaemonSets

### – Εκτέλεση Replicas σε κάθε κόμβο με DaemonSets

Οι περισσότεροι ελεγκτές Pod επιτρέπουν στον προγραμματιστή Kubernetes να καθορίσει ποιος κόμβος πρέπει να εκτελέσει ένα Pod. Τα DaemonSets είναι διαφορετικά - εκτελούν ακριβώς ένα Pod σε κάθε κόμβο.

Είναι για φόρτους εργασίας όπου θέλετε υψηλή διαθεσιμότητα σε πολλούς κόμβους, αλλά δεν χρειάζεστε υψηλά επίπεδα κλίμακας. Ή όπου η εφαρμογή πρέπει να λειτουργεί με κάθε κόμβο, π.χ. συλλογή logs.

Τα Deployments ταιριάζουν καλύτερα στις περισσότερες εφαρμογές και τα DaemonSets είναι λιγότερο κοινά, αλλά θα τα δείτε να χρησιμοποιούνται και δεν είναι πολύπλοκο να εργαστείτε μαζί τους.

### – API specs

- [DaemonSet \(apps/v1\)](#)
- Το DaemonSet είναι ένας ελεγκτής Pod, επομένως όλες οι σημαντικές λεπτομέρειες περιλαμβάνονται στις προδιαγραφές του Pod - το οποίο είναι ακριβώς το ίδιο Pod API που χρησιμοποιείτε με τα Deployments:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      # Pod spec
```

- selector - οι ετικέτες που χρησιμοποιούνται για την αναγνώριση των Pods που ανήκουν στο DaemonSet
- template.metadata - ετικέτες pod, οι οποίες πρέπει να ταιριάζουν ή να αποτελούν υπερσύνολο του επιλογέα
- template.spec - τυπική προδιαγραφή Pod.

– **Αναπτύξτε ένα DaemonSet με ένα HostPath**

Μια έγκυρη περίπτωση χρήσης για το DaemonSets είναι όπου η εφαρμογή πρέπει να χρησιμοποιεί πόρους συγκεκριμένους για τον κόμβο. Πολλαπλά Pods που εκτελούνται στον κόμβο ενδέχεται να συγκρούονται σχετικά με τους πόρους, επομένως ένα DaemonSet το αποτρέπει.

```
# save it in a folder called daemonsets/specs/nginx and name
it daemonset.yaml

# it defines an Nginx app where logs are written to a
HostPath volume, directly using the node's disk

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: daemonsets
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          volumeMounts:
            - name: logs
              mountPath: /var/log/nginx
      volumes:
        - name: logs
          hostPath:
            path: /volumes/nginx-logs
            type: DirectoryOrCreate

# save it in a folder called daemonsets/specs/nginx and name
it services.yaml

# it defines Services to route to the Nginx Pod; the label
selector mechanism is the same as with Deployments

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
```

```

labels:
  kubernetes.courselabs.co: daemonsets
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          volumeMounts:
            - name: logs
              mountPath: /var/log/nginx
      volumes:
        - name: logs
          hostPath:
            path: /volumes/nginx-logs
            type: DirectoryOrCreate

```

```
kubectl apply -f daemonsets/specs/nginx
```

```
kubectl get daemonset
```

Το επιθυμητό πλήθος ταιριάζει με τον αριθμό των κόμβων στο cluster σας. Σε ένα cluster ενός κόμβου θα λάβετε ένα Pod με 10 κόμβους, 10 Pods.

Οι υπηρεσίες δρομολογούνται στα Pods με τον ίδιο τρόπο, είτε διαχειρίζονται από DaemonSet είτε από ReplicaSet.

- Επιβεβαιώστε ότι η διεύθυνση IP του Pod έχει καταχωριθεί στην Υπηρεσία.

```
kubectl get po -l app=nginx -o wide
```

```
kubectl get endpoints nginx-np
```

Περιηγηθείτε στην εφαρμογή στη διεύθυνση <http://localhost:30010> ή <http://localhost:8010> και θα δείτε την τυπική σελίδα Nginx.

#### – Ενημέρωση DaemonSets

Τα DaemonSets εκτελούν ακριβώς ένα Pod σε κάθε κόμβο, επομένως η συμπεριφορά ενημέρωσης είναι να αφαιρέσετε τα Pod πριν ξεκινήσετε τις αντικαταστάσεις.



Αυτό διαφέρει από τα Deployments, τα οποία ξεκινούν από προεπιλογή νέα Pods και ελέγχουν ότι είναι υγιή πριν αφαιρέσουν τα παλιά. Οι ενημερώσεις του DaemonSet μπορούν να χαλάσουν την εφαρμογή σας:

```
# save it in a folder called daemonsets/specs/nginx/update-
  bad and name it daemonset-bad-command.yaml

# it has a misconfigured command in the Pod container - when
  the container runs, it will exit instead of running Nginx

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: daemonsets
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          command: ['sh', '-c', "echo '<!DOCTYPE html><html><
body><h1>Kubernetes Course Labs</h1></body></html>' > /
usr/share/nginx/html/index.html"]
          volumeMounts:
            - name: logs
              mountPath: /var/log/nginx
      volumes:
        - name: logs
          hostPath:
            path: /volumes/nginx-logs
            type: DirectoryOrCreate
```

```
kubectl apply -f daemonsets/specs/nginx/update-bad
```

```
kubectl get pods -l app=nginx --watch
```

Θα δείτε το παλιό Pod να τερματίζεται πρώτα και μετά να ξεκινά το νέο Pod και να αποτυγχάνει.

Δοκιμάστε την εφαρμογή - είναι χαλασμένη. Με ένα Deployment το παλιό Pod δεν θα είχε αφαιρεθεί μέχρι να είναι έτοιμο το νέο Pod, επομένως η εφαρμογή θα είχε παραμείνει συνδεδεμένη.

#### – Pods με δοχεία init

Η κακή ενημέρωση στην τελευταία άσκηση προσπάθησε να γράψει μια νέα σελίδα HTML για να εξυπηρετήσει το Nginx, αλλά η αλλαγή της εντολής στο δοχείο της εφαρμογής δεν είναι ο σωστός τρόπος.

Όλα τα Pods υποστηρίζουν δοχείο init που μπορείτε να χρησιμοποιήσετε για εργασίες εκκίνησης. Ένα δοχείο init μπορεί να μοιράζεται τόμους με το δοχείο της εφαρμογής και θα εκτελείται πριν από την εκκίνηση του δοχείο της εφαρμογής:

```
# save it in a folder called daemonsets/specs/nginx/update-
  good and name it daemonset-init-container.yaml

# it uses an init container to write a new HTML page for the
  Nginx app container to serve

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: daemonsets
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      initContainers:
        - name: init-html
          image: kiamol/ch03-sleep
          command: ['sh', '-c', "echo '<!DOCTYPE html><html><
body><h1>I was written by an init container</h1></body></
html>' > /html/index.html"]
      volumeMounts:
        - name: html
          mountPath: /html
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          volumeMounts:
            - name: html
```

```

        mountPath: /usr/share/nginx/html
      - name: logs
        mountPath: /var/log/nginx
    volumes:
      - name: html
        emptyDir: {}
      - name: logs
        hostPath:
          path: /volumes/nginx-logs
          type: DirectoryOrCreate

```

- Αναπτύξτε τη νέα ενημέρωση:

```

kubectl apply -f daemonsets/specs/nginx/update-good
kubectl get pods -l app=nginx --watch

```

Θα δείτε μερικές νέες καταστάσεις στην έξοδο, όπως Init και PodInitializing.

- Το δοχείο init εκτελείται και ολοκληρώνεται και, στη συνέχεια, ξεκινά το δοχείο της εφαρμογής. Όταν το Pod βρίσκεται σε κατάσταση Ready, εκτελείται μόνο το δοχείο της εφαρμογής.

```

kubectl logs -l app=nginx

kubectl exec daemonset/nginx -- cat /usr/share/nginx/html/
index.html

```

Το περιεχόμενο του HTML γράφτηκε από το δοχείο init στον κοινόχρηστο τόμο EmptyDir

Δοκιμάστε την εφαρμογή, λειτουργεί ξανά και έχει νέα αρχική σελίδα.

#### – Ανάπτυξη σε ένα υποσύνολο κόμβων

Το DaemonSets εκτελεί ένα Pod σε κάθε κόμβο - ο ελεγκτής παρακολουθεί την κατάσταση του κόμβου καθώς και την κατάσταση του Pod για να βεβαιωθεί ότι ο επιθυμητός αριθμός αντιγράφων είναι σωστός.

Μπορεί να θέλετε Pods μόνο σε ορισμένους από τους κόμβους και το DaemonSet το υποστηρίζει με έναν επιλογέα κόμβου:

```

# save it in a folder called daemonsets/specs/nginx/update-
subset and name it daemonset-node-selector.yaml

```

```

# it adds a node selector to the DaemonSet, so Pods will only
  run on nodes which have a matching label

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: daemonsets
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      initContainers:
        - name: init-html
          image: kiamol/ch03-sleep
          command: ['sh', '-c', "echo '<!DOCTYPE html><html><
body><h1>I will only run on nodes labelled ip=public</h1
></body></html>' > /html/index.html"]
          volumeMounts:
            - name: html
              mountPath: /html
      containers:
        - image: nginx:1.18-alpine
          name: nginx
          volumeMounts:
            - name: html
              mountPath: /usr/share/nginx/html
            - name: logs
              mountPath: /var/log/nginx
      volumes:
        - name: html
          emptyDir: {}
        - name: logs
          hostPath:
            path: /volumes/nginx-logs
            type: DirectoryOrCreate
      nodeSelector:
        kubernetes.courselabs.co.ip: public

```

- Εφαρμόστε την αλλαγή - τι πιστεύετε ότι θα συμβεί με το υπάρχον Nginx Pod;

```
kubectl apply -f daemonsets/specs/nginx/update-subset
```

```
kubectl get pods -l app=nginx --watch
```

Διαγράφηκε. Κανένας κόμβος δεν ταιριάζει με τα κριτήρια για το DaemonSet, επομένως το επιθυμητό πλήθος είναι 0. Το Pod αφαιρείται για να φτάσει στο επιθυμητό πλήθος.

Μόλις ένας κόμβος ταιριάζει με τον επιλογέα, προγραμματίζεται ένα Pod για αυτόν.

- Προσθέστε την ετικέτα που λείπει στον κόμβο σας και επιβεβαιώστε την έναρξη ενός Pod.

```
kubectl label node $(kubectl get nodes -o jsonpath='{.items[0].metadata.name}') kubernetes.courselabs.co.ip=public
kubectl get pods -l app=nginx --watch
```

Δημιουργείται ένα νέο Pod και η εφαρμογή λειτουργεί ξανά.

#### – Εργαστηριακή άσκηση 18

Υπάρχουν μόνο μερικές ακόμη δυνατότητες των DaemonSets - χρησιμοποιούνται σπάνια, αλλά είναι πολύ χρήσιμες όταν τις χρειάζεστε.

Πρώτα θέλουμε έναν τρόπο χειροκίνητου ελέγχου του πότε αντικαθίστανται τα Pods, ώστε να μπορούμε να ενημερώσουμε το DaemonSet αλλά το νέο Pod δεν θα δημιουργηθεί μέχρι να διαγράψουμε το παλιό.

Δεύτερον, θέλουμε να κάνουμε το αντίστροφο - να διαγράψουμε το DaemonSet αλλά να αφήσουμε το Pod ανέπαφο.

#### – EXTRA: Αναπτύξτε ένα Pod εντοπισμού σφαλμάτων σε έναν κόμβο Daemon-Set

- Το Nginx Pod γράφει logs σε έναν τόμο HostPath:

```
kubectl exec daemonset/nginx -- ls /var/log/nginx
```

Μπορείτε να αναπτύξετε ένα άλλο Pod με την ίδια προδιαγραφή τόμου HostPath και θα έχει κοινόχρηστο χώρο αποθήκευσης με το Nginx Pod.

- Σε ένα cluster πολλαπλών κόμβων, πρέπει να διασφαλίσετε ότι το νέο Pod προσγειώνεται στον ίδιο κόμβο με το Nginx Pod και μπορείτε να το κάνετε αυτό με τη συνάφεια Pod - Pod affinity:

```
# save it in a folder called daemonsets/specs and name it
sleep-with-hostPath.yaml

# it defines a sleep Pod with a HostPath volume and an
affinity rule, which means this Pod will be scheduled on
the same node as the Nginx Pod

apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: daemonsets
    app: sleep
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: node-root
          mountPath: /node-root
  volumes:
    - name: node-root
      hostPath:
        path: /
        type: Directory
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - nginx
          topologyKey: "kubernetes.io/hostname"
```

- Αναπτύξτε το νέο Pod και επαληθεύστε ότι προσγειώνεται στον ίδιο κόμβο.

```
kubectl apply -f daemonsets/specs/sleep-with-hostPath.yaml

kubectl get po -l app -o wide
```

Σε ένα cluster ενός κόμβου, κάθε Pod θα βρίσκεται σε αυτόν τον κόμβο - αλλά αυτό το παράδειγμα λειτουργεί με τον ίδιο τρόπο σε ένα cluster πολλών κόμβων

- Τώρα τα δύο Pods μοιράζονται ένα μέρος του συστήματος αρχείων του κόμβου host:

```
kubectl exec daemonset/nginx -- ls -l /var/log/nginx
kubectl exec pod/sleep -- ls -l /node-root/volumes/nginx-logs
```

Σε ένα cluster ενός κόμβου, κάθε Pod θα βρίσκεται σε αυτόν τον κόμβο - αλλά αυτό το παράδειγμα λειτουργεί με τον ίδιο τρόπο σε ένα cluster πολλών κόμβων

Ορισμένες εικόνες δοχείων δημιουργούνται "FROM scratch", πράγμα που σημαίνει ότι δεν υπάρχει λειτουργικό σύστημα και δεν υπάρχει shell για εκτέλεση. Αυτή είναι μια προσέγγιση για την εκκίνηση ενός δεύτερου Pod που μπορεί να βοηθήσει στον εντοπισμό σφαλμάτων με τα Pods εφαρμογών.

#### – Καθάρισμα

```
kubectl delete svc,ds,po -l kubernetes.courselabs.co=
daemonsets
```

## Ingress

#### – Ingress

Υπάρχουν δύο μέρη στο Ingress:

- ο ελεγκτής, ο οποίος είναι ένας αντίστροφος διακομιστής μεσολάβησης που λαμβάνει όλη την εισερχόμενη κίνηση
- τα αντικείμενα Ingress που ορίζουν τους κανόνες δρομολόγησης για τον ελεγκτή.

Μπορείτε να επιλέξετε από διαφορετικούς ελεγκτές. Θα χρησιμοποιήσουμε τον [Nginx Ingress Controller](#), αλλά το [Traefik](#) και το [Contour - a CNCF project](#) είναι δημοφιλείς εναλλακτικές λύσεις.

#### – API specs

- [Ingress \(networking.k8s.io/v1\)](#)

Οι κανόνες Ingress μπορούν να έχουν πολλαπλά mappings, αλλά είναι αρκετά απλοί.

Συνήθως έχετε ένα αντικείμενο ανά εφαρμογή και έχουν Namespace, ώστε να μπορείτε να τα αναπτύξετε στο ίδιο Namespace με την εφαρμογή:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami
spec:
  rules:
  - host: whoami.local
    http:
      paths:
```

```

- path: /
  pathType: Prefix
  backend:
    service:
      name: whoami-internal
      port:
        name: http

```

- rules - συλλογή κανόνων δρομολόγησης
- host - το όνομα DNS host της εφαρμογής Ιστού
- http - η δρομολόγηση εισόδου είναι μόνο για κίνηση ιστού
- paths - συλλογή διαδρομών αιτημάτων, αντιστοίχιση στις Υπηρεσίες Kubernetes
- path - η διαδρομή αιτήματος HTTP, μπορεί να είναι γενική "/" ή συγκεκριμένη "/admin"
- pathType - εάν η αντιστοίχιση διαδρομής είναι ως Prefix ή ως Exact
- backend.service - Υπηρεσία όπου ο ελεγκτής θα ανακτήσει περιεχόμενο

#### – Αναπτύξτε έναν ελεγκτή Ingress

Δεν είναι καλό όνομα, επειδή ένας ελεγκτής Ingress δεν είναι ένας συγκεκριμένος τύπος αντικειμένου Kubernetes - όπως το Deployment είναι ένας ελεγκτής Pod.

- Ένας ελεγκτής Ingress είναι ένα λογικό πράγμα, που αποτελείται από μια υπηρεσία, έναν ελεγκτή Pod και ένα σύνολο κανόνων RBAC:

```

# save it in a folder called ingress/specs/ingress-controller
  and name it 01_namespace.yaml

# it ingress controllers are shared for all apps, so they
  usuall have their own namespace

apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    kubernetes.courselabs.co: ingress

# save it in a folder called ingress/specs/ingress-controller
  and name it 02_rbac.yaml

# RBAC rules so the ingress controller can query the
  Kubernetes API for Service endpoints, Ingress objects and
  more

apiVersion: v1
kind: ServiceAccount
metadata:

```



```

    name: ingress-nginx
    namespace: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    kubernetes.courselabs.co: ingress
rules:
- apiGroups:
  - ''
  resources:
  - configmaps
  - endpoints
  - nodes
  - pods
  - secrets
  verbs:
  - list
  - watch
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ''
  resources:
  - services
  verbs:
  - get
  - list
  - update
  - watch
- apiGroups:
  - extensions
  - networking.k8s.io # k8s 1.14+
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ''
  resources:
  - events

```

```

    verbs:
      - create
      - patch
  - apiGroups:
      - extensions
      - networking.k8s.io    # k8s 1.14+
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - networking.k8s.io    # k8s 1.14+
    resources:
      - ingressclasses
    verbs:
      - get
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    kubernetes.courselabs.co: ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - namespaces
    verbs:
      - get
  - apiGroups:
      - ''
    resources:

```

```

    - configmaps
    - pods
    - secrets
    - endpoints
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - ''
  resources:
    - services
  verbs:
    - get
    - list
    - update
    - watch
- apiGroups:
  - extensions
  - networking.k8s.io    # k8s 1.14+
  resources:
    - ingresses
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - extensions
  - networking.k8s.io    # k8s 1.14+
  resources:
    - ingresses/status
  verbs:
    - update
- apiGroups:
  - networking.k8s.io    # k8s 1.14+
  resources:
    - ingressclasses
  verbs:
    - get
    - list
    - watch
- apiGroups:
  - ''
  resources:
    - configmaps
  verbs:
    - get
    - update
- apiGroups:
  - ''

```

```

    resources:
      - configmaps
    verbs:
      - create
  - apiGroups:
      - ''
    resources:
      - endpoints
    verbs:
      - create
      - get
      - update
  - apiGroups:
      - ''
    resources:
      - events
    verbs:
      - create
      - patch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx

# save it in a folder called ingress/specs/ingress-controller
# and name it configmap.yaml

# additional config for Nginx, to enable proxy caching

apiVersion: v1
kind: ConfigMap
metadata:
  name: ingress-nginx-controller
  namespace: ingress-nginx
data:
  http-snippet: |
    proxy_cache_path /tmp/nginx-cache levels=1:2 keys_zone=
    static-cache:2m max_size=100m inactive=7d use_temp_path=
    off;
    proxy_cache_key $scheme$proxy_host$request_uri;
```

```

    proxy_cache_lock on;
    proxy_cache_use_stale updating;

# save it in a folder called ingress/specs/ingress-controller
# and name it daemonset.yaml

# DaemonSet to run the ingress controller Pods; contains a
# few fields you haven't seen yet

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/component: controller
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/component: controller
    spec:
      containers:
        - name: controller
          image: k8s.gcr.io/ingress-nginx/controller:v1.2.0 #
            quay.io/kubernetes-ingress-controller/nginx-ingress-
            controller:0.33.0
          args:
            - /nginx-ingress-controller
            - --publish-service=ingress-nginx/ingress-nginx-
              controller-lb
            - --publish-service=ingress-nginx/ingress-nginx-
              controller-np
            - --election-id=ingress-controller-leader
            - --ingress-class=nginx
            - --controller-class=k8s.io/ingress-nginx
            - --configmap=ingress-nginx/ingress-nginx-
              controller
          securityContext:
            runAsUser: 101
            allowPrivilegeEscalation: true
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:

```

```

        fieldPath: metadata.name
      - name: POD_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
    ports:
      - name: http
        containerPort: 80
        protocol: TCP
      - name: https
        containerPort: 443
        protocol: TCP
    serviceAccountName: ingress-nginx

# save it in a folder called ingress/specs/ingress-controller
# and name it services.yaml

# Services for external access

apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx-controller-lb
  namespace: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 8000
      targetPort: http
    - name: https
      port: 8040
      targetPort: https
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx-controller-np
  namespace: ingress-nginx
spec:
  type: NodePort
  ports:
    - name: http
      port: 8080
      targetPort: http
      nodePort: 30000

```

```

- name: https
  port: 8040
  targetPort: https
  nodePort: 30040
selector:
  app.kubernetes.io/name: ingress-nginx
  app.kubernetes.io/instance: ingress-nginx
  app.kubernetes.io/component: controller

```

- Αναπτύξτε τον ελεγκτή:

```

kubectl apply -f ingress/specs/ingress-controller

kubectl get all -n ingress-nginx

kubectl wait --for=condition=Ready pod -n ingress-nginx -l
  app.kubernetes.io/name=ingress-nginx

```

Περιηγηθείτε στη διεύθυνση <http://localhost:8000> ή <http://localhost:30000>. Δεν υπάρχουν εφαρμογές που εκτελούνται στο cluster, αλλά θα λάβετε μια απάντηση 404, η οποία προέρχεται από τον ελεγκτή εισόδου

Ο ελεγκτής εισόδου τροφοδοτείται από το Nginx, αλλά δεν χρειάζεται να διαμορφώσετε τη δρομολόγηση μέσα στο Nginx - το αντιμετωπίζετε ως μαύρο κουτί και κάνετε όλη τη διαμόρφωση με αντικείμενα Ingress.

#### – Δημοσιεύστε μια εφαρμογή μέσω Ingress

Θα ξεκινήσουμε με μια εφαρμογή, η οποία θα είναι catch-all, έτσι ώστε οι χρήστες να μην βλέπουν ποτέ την απόκριση 404 από τον ελεγκτή εισόδου.

```

# save it in a folder called ingress/specs/default and name
  it deployment.yaml

# a simple Nginx deployment, using the standard Nginx image
  not the ingress controller

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: ingress
spec:
  selector:
    matchLabels:
      app: nginx

```

```

template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - image: nginx:1.18-alpine
        name: nginx
        volumeMounts:
          - name: content
            mountPath: /usr/share/nginx/html
    volumes:
      - name: content
        configMap:
          name: nginx-content

# save it in a folder called ingress/specs/default and name
# it configmap.yaml

# configuration containing HTML file for Nginx to show

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-content
  labels:
    kubernetes.courselabs.co: ingress
data:
  index.html: |-
    <!DOCTYPE html>
    <html>
      <body>
        <h1>
          Nothing to see here
        </h1>
      </body>
    </html>

# save it in a folder called ingress/specs/default and name
# it service.yaml

# ClusterIP Service

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-content
  labels:
    kubernetes.courselabs.co: ingress
data:

```



```
index.html: |-
  <!DOCTYPE html>
  <html>
  <body>
    <h1>
      Nothing to see here
    </h1>
  </body>
</html>
```

- Αναπτύξτε την εφαρμογή ιστού:

```
kubectl apply -f ingress/specs/default
```

- Δεν γίνεται τίποτα ακόμα. Οι υπηρεσίες δεν συνδέονται αυτόματα με τον ελεγκτή εισόδου - το κάνετε καθορίζοντας κανόνες δρομολόγησης σε ένα αντικείμενο Ingress:

```
# save it in a folder called ingress/specs/default/ingress
# and name it default.yaml

# Ingress rule with no host specified, so all requests will
# go here by default

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default
  labels:
    kubernetes.courselabs.co: ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /*
        pathType: Prefix
        backend:
          service:
            name: nginx-internal
            port:
              name: http
```

- Τώρα αναπτύξτε τον κανόνα εισόδου και απαριθμήστε όλους τους κανόνες:

```
kubectl apply -f ingress/specs/default/ingress
```

```
kubectl get ingress
```

Όταν περιηγηστείτε σε οποιαδήποτε διεύθυνση URL, θα δείτε την προεπιλεγμένη απάντηση:

Περιηγηθείτε στη διεύθυνση `http://localhost:8000/a/bc.php` ή `http://localhost:30000/a/bc.php`

Εκεί η απάντηση 404 ήρθε αρχικά από το Nginx. Μια εναλλακτική λύση για την εκτέλεση της δικής σας εφαρμογής είναι [να προσαρμόσετε το προεπιλεγμένο back-end](#) - αλλά αυτό ισχύει ειδικά για τον ελεγκτή εισόδου που χρησιμοποιείτε.

– **Δημοσιεύστε μια εφαρμογή σε μια συγκεκριμένη διεύθυνση host**

Για να δημοσιεύσετε όλες τις εφαρμογές σας μέσω του ελεγκτή εισόδου, είναι το ίδιο μοτίβο - έχετε μια εσωτερική Υπηρεσία πάνω από τα Pods της εφαρμογής και ένα αντικείμενο Ingress με κανόνες δρομολόγησης.

- Ακολουθεί η προδιαγραφή για την εφαρμογή whoami, η οποία θα δημοσιεύεται σε ένα συγκεκριμένο όνομα host:

```
# save it in a folder called ingress/specs/whoami and name it
  whoami.yaml

# Deployment and ClusterIP Service for the app, nothing
  ingress-specific

apiVersion: v1
kind: Service
metadata:
  name: whoami-internal
  labels:
    kubernetes.courselabs.co: ingress
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 80
      targetPort: 80
    type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: ingress
spec:
  replicas: 3
  selector:
    matchLabels:
```

```

    app: whoami
  template:
    metadata:
      labels:
        app: whoami
        version: v1
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q

# save it in a folder called ingress/specs/whoami and name it
# ingress.yaml

# Ingress which routes traffic with the host domain whoami.
# local to the ClusterIP Service

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: ingress
spec:
  rules:
    - host: whoami.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: whoami-internal
                port:
                  name: http

```

- Αναπτύξτε την εφαρμογή και ελέγξτε τους κανόνες εισόδου.

```
kubectl apply -f ingress/specs/whoami
```

```
kubectl get ingress
```

```

# using Powershell - your terminal needs to be running as
# Admin:
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
Process -Force

```

```
./scripts/add-to-hosts.ps1 whoami.local 127.0.0.1

# on macOS or Linux - you'll be asked for your sudo password:
sudo chmod +x ./scripts/add-to-hosts.sh
./scripts/add-to-hosts.sh whoami.local 127.0.0.1
```

Περιηγηθείτε στη διεύθυνση `http://whoami.local:8000` ή `http://whoami.local:30000` και θα δείτε τον ιστότοπο. Υπάρχουν πολλά α-ντίγραφα - ανανεώστε για να δείτε την εξισορρόπηση φορτίου μεταξύ τους.

#### – Χρησιμοποιήστε Ingress με response caching

Το Ingress API δεν υποστηρίζει όλες τις δυνατότητες κάθε ελεγκτή εισόδου, επομένως για να χρησιμοποιήσετε προσαρμοσμένες δυνατότητες ορίζετε τη διαμόρφωση σε σχολιασμούς.

- Θα δημοσιεύσουμε την εφαρμογή Ιστού Pi στο hostname "pi.local", χρησιμοποιώντας πρώτα ένα απλό Ingress χωρίς response cache:

```
# save it in a folder called ingress/specs/pi and name it pi.
  yaml

# Deployment and Service for the app

apiVersion: v1
kind: Service
metadata:
  name: pi-internal
  labels:
    kubernetes.courselabs.co: ingress
spec:
  ports:
    - port: 80
      targetPort: http
      name: http
  selector:
    app: pi-web
    type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  labels:
    kubernetes.courselabs.co: ingress
spec:
  replicas: 2
  selector:
    matchLabels:
```

```

    app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - name: http
              containerPort: 80

# save it in a folder called ingress/specs/pi and name it
# ingress.yaml

# Ingress which routes pi.local to the Service

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pi
  labels:
    kubernetes.courselabs.co: ingress
spec:
  rules:
    - host: pi.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: pi-internal
                port:
                  name: http

```

- Αναπτύξτε την εφαρμογή, ελέγξτε την κατάσταση και προσθέστε το pi.local στο αρχείο hosts σας.

```

kubectl apply -f ingress/specs/pi

kubectl get ingress

kubectl get po -l app=pi-web

# Windows:
./scripts/add-to-hosts.ps1 pi.local 127.0.0.1

```

```
# *nix:
./scripts/add-to-hosts.sh pi.local 127.0.0.1
```

Περηγηθείτε στη διεύθυνση `http://pi.local:8000/pi?dp=25000` / `http://pi.local:30000/pi?dp=25000`, θα χρειαστεί περίπου ένα δευτερόλεπτο για να δείτε την απάντηση. Κάντε ανανέωση και θα δείτε ότι το αίτημα είναι εξισορροπημένο και η απόκριση υπολογίζεται κάθε φορά.

- Μπορούμε να ενημερώσουμε το αντικείμενο Ingress για να χρησιμοποιήσουμε response caching - την οποία υποστηρίζει ο ελεγκτής εισόδου Nginx:

```
# save it in a folder called ingress/specs/pi/update and name
it ingress-with-cache.yaml

# Deployment and Service for the app

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pi
  labels:
    kubernetes.courselabs.co: ingress
  annotations:
    nginx.ingress.kubernetes.io/proxy-buffering: "on"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_cache static-cache;
      proxy_cache_valid 10m;
spec:
  rules:
    - host: pi.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: pi-internal
                port:
                  name: http
```

Τώρα μεταβείτε στη διεύθυνση `http://pi.local:8000/pi?dp=25000` / `http://pi.local:30000/pi?dp=25000` - θα βλέπετε το response caching με κάθε ανανέωση.

Μπορεί να έχετε διαφορετικούς κανόνες Ingress - έναν για όλο το στατικό περιεχόμενο που έχει τον σχολιασμό της cache και έναν άλλο για το δυναμικό περιεχόμενο.

#### – Εργαστηριακή άσκηση 19

Δύο μέρη σε αυτό το εργαστήριο. Πρώτα θέλουμε να πάρουμε την διαμορφώσιμη εφαρμογή web και να τη δημοσιεύσουμε μέσω του ελεγκτή εισόδου.

Η προδιαγραφή της εφαρμογής είναι ήδη σε ισχύ για να ξεκινήσετε, η δουλειά σας είναι να δημιουργήσετε και να αναπτύξετε τη δρομολόγηση Ingress:

```
# save it in a folder called ingress/specs/configurable and
  name it 01-namespace.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: ingress

# save it in a folder called ingress/specs/configurable and
  name it configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: configurable-env
  namespace: configurable
data:
  Configurable__Release: 24.01.2
  Configurable__Environment: prod

# save it in a folder called ingress/specs/configurable and
  name it deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  namespace: configurable
spec:
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
```

```

    containers:
      - name: app
        image: sixeyed/configurable:21.04
        envFrom:
          - configMapRef:
              name: configurable-env

# save it in a folder called ingress/specs/configurable and
# name it services.yaml

apiVersion: v1
kind: Service
metadata:
  name: configurable-web
  namespace: configurable
spec:
  selector:
    app: configurable
  ports:
    - name: config-app
      port: 8020
      targetPort: 80
  type: ClusterIP

```

```
kubectl apply -f ingress/specs/configurable
```

Το δεύτερο μέρος είναι ότι θα θέλαμε να αλλάξουμε τον ελεγκτή εισόδου ώστε να χρησιμοποιεί τις τυπικές θύρες - 80 για HTTP και 443 για HTTPS. Θα μπορείτε να το κάνετε μόνο εάν χρησιμοποιείτε το LoadBalancer.

#### – EXTRA: Ingress για HTTPS

Οι ελεγκτές Ingress μπορούν να εφαρμόσουν πιστοποιητικά TLS για την κρυπτογράφηση της κυκλοφορίας HTTPS, επομένως δεν χρειάζεστε αυτή τη λογική στις εφαρμογές σας. Το Ingress για HTTPS σας οδηγεί σε αυτό.

#### – Καθάρισμα

```
kubectl delete all,secret,ingress,clusterrolebinding,
clusterrole,ns,ingressclass -l kubernetes.courselabs.co=
ingress
```

## Jobs and CronJobs

### – Εκτέλεση One-off Pods με Jobs και Recurring Pods με CronJobs



Μερικές φορές θέλετε ένα Pod να εκτελέσει κάποια εργασία και μετά να σταματήσει. Θα μπορούσατε να αναπτύξετε μια προδιαγραφή Pod, αλλά αυτή έχει περιορισμένη υποστήριξη επανάληψης, εάν η εργασία αποτύχει, αλλά δεν μπορείτε να χρησιμοποιήσετε ένα Deployment, επειδή θα αντικαταστήσει το Pod εάν εξέλθει με επιτυχία.

Τα [Jobs](#) προορίζονται για αυτήν την περίπτωση - είναι ένας ελεγκτής Pod που δημιουργεί ένα Pod και διασφαλίζει ότι λειτουργεί μέχρι την ολοκλήρωσή του. Εάν το Pod αποτύχει, το Job θα ξεκινήσει μια αντικατάσταση, αλλά όταν το Pod πετύχει, η Job έχει ολοκληρωθεί.

Τα Jobs μπορούν να έχουν τον δικό τους ελεγκτή με ένα [CronJob](#) που περιέχει μια προδιαγραφή Job και ένα χρονοδιάγραμμα. Στο χρονοδιάγραμμα δημιουργεί ένα Job, το οποίο δημιουργεί και παρακολουθεί ένα Pod.

#### – API specs

- [Job \(batch/v1\)](#)
- [CronJob \(batch/v1beta1\)](#)

Η απλούστερη προδιαγραφή Job έχει απλώς μεταδεδομένα και ένα πρότυπο με τυπική προδιαγραφή Pod:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job
spec:
  template:
    spec:
      containers:
      - # container spec
      restartPolicy: Never
```

- `template.spec` - μια προδιαγραφή Pod που μπορεί να περιλαμβάνει τόμους, διαμόρφωση και οτιδήποτε άλλο υπάρχει σε ένα κανονικό Pod
- `restartPolicy` - η προεπιλεγμένη πολιτική επανεκκίνησης Pod είναι `Always`, η οποία δεν επιτρέπεται για τα Jobs. Πρέπει να την καθορίσετε ως `"Never"` ή `"OnFailure"`.

Το CronJobs αναδιπλώνει την προδιαγραφή Job, προσθέτοντας ένα χρονοδιάγραμμα με τη [μορφή έκφρασης \\*nix cron](#):

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: db-backup
spec:
  schedule: "0 9 * * *"
```

```
concurrencyPolicy: Forbid
jobTemplate:
  # job spec
  restartPolicy: Never
```

- `apiVersion` - το Kubernetes χρησιμοποιεί εκδόσεις beta για να υποδείξει ότι το API δεν είναι οριστικό. Τα CronJobs θα σε κανονική έκδοση στο Kubernetes 1.21
- `schedule` - έκφραση cron για το πότε πρόκειται να δημιουργηθούν εργασίες
- `concurrencyPolicy` - εάν θα επιτρέπεται (Allow) η δημιουργία νέων Job(s) όταν το προηγούμενο προγραμματισμένο Job εξακολουθεί να εκτελείται, εάν θα απαγορεύεται (Forbid) ή εάν θα αντικαθίσταται (Replace) η παλιά εργασία με μια νέα.

—

#### – Εκτελέστε μια one-off εργασία σε ένα Job

Έχουμε έναν ιστότοπο που μπορούμε να χρησιμοποιήσουμε για να υπολογίσουμε το Pi, αλλά η εφαρμογή μπορεί επίσης να εκτελέσει έναν one-off υπολογισμό:

```
# save it in a folder called jobs/specs/pi/one and name it pi
-job-50dp.yaml

# Job spec which uses the same Pi image with a new command to
run a one-off calculation

apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job-one
  labels:
    kubernetes.courselabs.co: jobs
    app: pi-one
    dp: '50'
spec:
  template:
    spec:
      containers:
        - name: pi
          image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "console",
"-dp", "50"]
          restartPolicy: Never
```

- Δημιουργήστε το Job:

```
kubectl apply -f jobs/specs/pi/one

kubectl get jobs
```

Τα Job εφαρμόζουν μια ετικέτα στα Pods που δημιουργούν (μαζί με τις ετικέτες στο πρότυπο Προδιαγραφής των Pods).

- Χρησιμοποιήστε την ετικέτα του Job για να λάβετε τις λεπτομέρειες του Pod και να εμφανίσετε τα logs του.

```
kubectl get pods --show-labels
kubectl get pods -l job-name=pi-job-one
kubectl logs -l job-name=pi-job-one
```

Θα δείτε υπολογισμένο το Pi. Αυτή είναι η μόνη έξοδος από αυτό το Pod.

- Όταν ολοκληρωθούν τα Jobs, δεν εκκαθαρίζονται αυτόματα:

```
kubectl get jobs
```

Η εργασία εμφανίζει 1/1 completions - που σημαίνει ότι 1 Pod εκτελέστηκε με επιτυχία.

- Δεν μπορείτε να ενημερώσετε τις προδιαγραφές Pod για ένα υπάρχον Job, τα Jobs δεν διαχειρίζονται τις αναβαθμίσεις Pod όπως τα Deployments.

```
# save it in a folder called jobs/specs/pi/one/update and
# name it pi-job-500dp.yaml

# it changes the Pod spec to calculate Pi to more decimal
# places

apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job-one
  labels:
    kubernetes.courselabs.co: jobs
    app: pi-one
    dp: '500'
spec:
  template:
    spec:
      containers:
        - name: pi
          image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "console",
"-dp", "500"]
          restartPolicy: Never
```

- Προσπαθήστε να αλλάξετε το υπάρχον Job και θα λάβετε ένα σφάλμα:

```
kubectl apply -f jobs/specs/pi/one/update
```

Για να αλλάξετε ένα Job θα πρέπει πρώτα να διαγράψετε το παλιό.

#### – Εκτελέστε ένα Job με πολλές ταυτόχρονες διεργασίες

Τα Jobs δεν είναι μόνο για μία διεργασία, σε ορισμένα σενάρια θέλετε να εκτελείται το ίδιο Pod για σταθερό αριθμό φορών.

Όταν έχετε ένα σταθερό σύνολο διεργασιών για επεξεργασία, μπορείτε να χρησιμοποιήσετε ένα Job για να εκτελέσει όλα τα κομμάτια παράλληλα:

```
# save it in a folder called jobs/specs/pi/many and name it
pi-job-random.yaml

# it defines a Job to run 3 Pods concurrently, each of which
calculates Pi to a random number of decimal places

apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job-one
  labels:
    kubernetes.courselabs.co: jobs
    app: pi-one
    dp: '500'
spec:
  template:
    spec:
      containers:
        - name: pi
          image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "console",
                    "-dp", "500"]
          restartPolicy: Never
```

- Εκτελέστε την τυχαία διεργασία Pi:

```
kubectl apply -f jobs/specs/pi/many
```

```
kubectl get jobs -l app=pi-many
```

Θα δείτε ένα Job, με 3 αναμενόμενες ολοκληρώσεις.

- Ελέγξτε την κατάσταση Pod και τα logs για αυτήν την εργασία.

```
kubectl get pods -l job-name=pi-job-many
kubectl logs -l job-name=pi-job-many
```

Θα λάβετε logs για όλα τα Pods - σελίδες με το Pi :)

- Το Job έχει λεπτομέρειες για όλα τα Pods που δημιουργεί:

```
kubectl describe job pi-job-many
```

Εμφανίζει συμβάντα δημιουργίας καταστάσεων pod.

#### – Προγραμματίστε διεργασίες με το CronJobs

Τα Jobs δεν εκκαθαρίζονται αυτόματα, για να μπορείτε να εργαστείτε με τα Pods και να δείτε logs.

- Η περιοδική εκτέλεση μιας εργασίας εκκαθάρισης είναι ένα σενάριο όπου χρησιμοποιείτε ένα CronJob:

```
# save it in a folder called jobs/specs/cleanup and name it
cronjob.yaml

# a CronJob which runs a shell script to delete jobs by
running Kubectl inside a Pod

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: job-cleanup
  namespace: default
  labels:
    kubernetes.courselabs.co: jobs
spec:
  schedule: "*/1 * * * *" # every minute - see https
                        ://crontab.guru
  concurrencyPolicy: Forbid
  jobTemplate:
    metadata:
      labels:
        app: job-cleanup
    spec:
      template:
        metadata:
          labels:
```

```

        app: job-cleanup
        status: successful
    spec:
        serviceAccountName: job-cleanup
        restartPolicy: Never
        containers:
        - name: app
          image: sixeyed/kubectl:1.21.0
          command: ['sh', '-c', '/scripts/run.sh']
          env:
          - name: CLEANUP_NAMESPACE
            value: default
        volumeMounts:
        - name: scripts
          mountPath: "/scripts"
        volumes:
        - name: scripts
          configMap:
            defaultMode: 0500      # executable, e.g.
            name: job-cleanup-script
            chmod +x

# save it in a folder called jobs/specs/cleanup and name it
rbac.yaml

# Service Account for the Pod and RBAC rules to allow it to
query and delete Jobs

apiVersion: v1
kind: ServiceAccount
metadata:
  name: job-cleanup
  namespace: default
  labels:
    kubernetes.courselabs.co: rbac
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: job-cleanup-role
  labels:
    kubernetes.courselabs.co: jobs
rules:
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["delete", "get", "list"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["delete", "get", "list"]
---
```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: job-cleanup-role-binding
  labels:
    kubernetes.courselabs.co: jobs
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: job-cleanup-role
subjects:
  - kind: ServiceAccount
    name: job-cleanup
    namespace: default

# save it in a folder called jobs/specs/cleanup and name it
# configmap.yaml

# ConfigMap which contains the shell script - this is a nice
# way to run scripts without having to build a custom
# Docker image

apiVersion: v1
kind: ConfigMap
metadata:
  name: job-cleanup-script
  namespace: default
  labels:
    kubernetes.courselabs.co: jobs
data:
  run.sh: |-
    #!/bin/sh
    kubectl config set-credentials user --token=$(cat /var/
run/secrets/kubernetes.io/serviceaccount/token) > /dev/
null

    echo '** Deleting Completed Jobs **'
    kubectl delete jobs -n $CLEANUP_NAMESPACE $(kubectl get
jobs -n $CLEANUP_NAMESPACE -o=jsonpath='{.items[?(@.
status.conditions[0].type=="Complete")].metadata.name}')
    || true

    if [ "$CLEANUP_STATUS" == "ALL" ]; then
      echo '** Deleting Failed Jobs **'
      kubectl delete jobs -n $CLEANUP_NAMESPACE $(kubectl get
jobs -n $CLEANUP_NAMESPACE -o=jsonpath='{.items[?(@.
status.conditions[0].type=="Failed")].metadata.name}') ||
      true
    fi

```

```
echo "*** Done cleaning namespace: $CLEANUP_NAMESPACE ***"
```

Το CronJob έχει ρυθμιστεί να εκτελείται κάθε λεπτό, οπότε σύντομα θα το δείτε να εργάζεται.

- Αναπτύξτε το CronJob και παρακολουθήστε όλα τα Jobs για να δείτε την κατάργησή τους.

```
kubectl apply -f jobs/specs/cleanup
kubectl get cronjob
```

θα δείτε να δημιουργείται το Job καθαρισμού και, στη συνέχεια, η λίστα θα ενημερώνεται με ένα νέο Job καθαρισμού κάθε λεπτό.

- Επιβεβαιώστε ότι τα ολοκληρωμένα Pi Jobs και τα Pods τους έχουν αφαιρεθεί:

```
# Ctrl-C to exit the watch
kubectl get jobs
kubectl get pods -l job-name --show-labels
```

Η πιο πρόσφατη job εκκαθάρισης εξακολουθεί να υπάρχει επειδή το CronJobs δεν διαγράφει τις εργασίες όταν ολοκληρωθούν.

- Μπορείτε να ελέγξετε τα logs για να δείτε τι έκανε το script εκκαθάρισης:

```
kubectl logs -l app=job-cleanup
```

## Εργαστηριακή άσκηση 20

Τα αληθινά CronJobs δεν εκτελούνται κάθε λεπτό - χρησιμοποιούνται για εργασίες συντήρησης και εκτελούνται πολύ λιγότερο συχνά, ωριαία, ημερήσια ή εβδομαδιαία.

Συχνά θέλετε να εκτελέσετε ένα one-off Job από ένα CronJob χωρίς να περιμένετε να δημιουργηθεί το επόμενο σύμφωνα με το χρονοδιάγραμμα.

Η πρώτη διεργασία για αυτό το εργαστήριο είναι να επεξεργαστείτε το "job-cleanup" CronJob και να το θέσετε σε αναστολή, ώστε να μην εκτελούνται άλλα Jobs και να μπερδέψετε όταν δημιουργείτε το νέο σας Job. Δείτε αν μπορείτε να το κάνετε αυτό χωρίς να χρησιμοποιήσετε το "kubectl apply".

- Στη συνέχεια, αναπτύξτε αυτό το νέο CronJob:



```
# save it in a folder called jobs/specs/backup and name it
cronjob.yaml

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: db-backup
  namespace: default
  labels:
    kubernetes.courselabs.co: jobs
spec:
  schedule: "0 1 * * 0"          # 01:00 every Sunday
  concurrencyPolicy: Forbid
  jobTemplate:
    metadata:
      labels:
        app: db-backup
    spec:
      template:
        metadata:
          labels:
            app: db-backup
        spec:
          restartPolicy: Never
          containers:
            - name: app
              image: alpine:3.13
              command: ['sh', '-c', 'echo Done.']
```

```
kubectl apply -f jobs/specs/backup
```

Και η επόμενη διεργασία είναι να εκτελέσετε ένα Job από αυτήν την προδιαγραφή του CronJob. Δείτε αν μπορείτε επίσης να το κάνετε αυτό χωρίς να χρησιμοποιήσετε το "kubectl apply".

#### – EXTRA: Διαχειριστείτε τις αποτυχίες στα Jobs

Οι εργασίες στο παρασκήνιο στα Jobs θα μπορούσαν να εκτελούνται για μεγάλο χρονικό διάστημα και χρειάζεστε κάποιο έλεγχο σχετικά με τον τρόπο χειρισμού των αποτυχιών.

Η πρώτη επιλογή είναι να επιτραπεί η επανεκκίνηση του Pod, οπότε αν το δοχείο αποτύχει, τότε ξεκινά ένα νέο δοχείο στο ίδιο Pod:

```
# save it in a folder called jobs/specs/pi/one-failing and
name it pi-job.yaml
```

```
# a Job spec with a mistake in the container command; the
# restart policy is set so the Pod will restart when the
# container fails

apiVersion: batch/v1
kind: Job
metadata:
  name: pi-job-one-failing
  labels:
    kubernetes.courselabs.co: jobs
spec:
  template:
    spec:
      containers:
        - name: pi
          image: kiamol/ch05-pi
          command: ["donet", "Pi.Web.dll", "-m", "console", "-dp", "50"]
          restartPolicy: OnFailure
```

- Δοκιμάστε αυτό το Job:

```
kubectl apply -f jobs/specs/pi/one-failing

kubectl get jobs pi-job-one-failing
```

- Το Pod δημιουργείται αλλά το δοχείο θα τερματίσει αμέσως, προκαλώντας επανεκκίνηση του Pod:

```
kubectl get pods -l job-name=pi-job-one-failing --watch
```

Θα δείτε καταστάσεις `RunContainerError` και πολλές επανεκκινήσεις έως ότου το Pod μεταβεί στο `CrashLoopBackoff`.

- Ενδέχεται να μην θέλετε να επανεκκινήσει ένα Pod που αποτυγχάνει και το Job μπορεί να ρυθμιστεί ώστε να δημιουργεί αντικατάσταση Pod. Αυτό είναι καλό εάν μια αποτυχία προκλήθηκε από πρόβλημα σε έναν κόμβο, επειδή το Pod αντικατάστασης θα μπορούσε να εκτελεστεί σε διαφορετικό κόμβο:

```
# save it in a folder called jobs/specs/pi/one-failing and
# name it pi-job-restart.yaml

# it sets the restart policy so the Pod never restarts, and
# sets a backoff limit in the Job so if the Pod fails the
# Job will try with new Pods up to 4 times

apiVersion: batch/v1
```

```
kind: Job
metadata:
  name: pi-job-one-failing
  labels:
    kubernetes.courselabs.co: jobs
spec:
  completions: 1
  backoffLimit: 4
  template:
    spec:
      containers:
        - name: pi
          image: kiamol/ch05-pi
          command: ["donet", "Pi.Web.dll", "-m", "console", "-dp", "50"]
          restartPolicy: Never
```

- Δεν μπορείτε να ενημερώσετε το υπάρχον Job, επομένως θα πρέπει πρώτα να το διαγράψετε:

```
kubectl delete jobs pi-job-one-failing

kubectl apply -f jobs/specs/pi/one-failing/update
```

- Τώρα, όταν παρακολουθείτε τα Pods, δεν θα βλέπετε το ίδιο Pod να επανεκκινείται, θα δείτε να δημιουργούνται νέα Pod:

```
kubectl get pods -l job-name=pi-job-one-failing --watch
```

Θα δείτε την κατάσταση ContainerCannotRun, 0 επανεκκινήσεις και στο τέλος συνολικά 4 Pods.

- Ένα αρνητικό του Pod που βρίσκεται την κατάσταση ContainerCannotRun είναι ότι δεν θα βλέπετε κανένα log και για να βρείτε γιατί βρίσκεται σε αυτή την κατάσταση θα χρειαστεί να περιγράψετε το Pod:

```
kubectl logs -l job-name=pi-job-one-failing

kubectl describe pods -l job-name=pi-job-one-failing
```

## – Καθάρισμα

```
kubectl delete job,cronjob,cm,sa,clusterrole,
  clusterrolebinding -l kubernetes.courselabs.co=jobs
```

## StatefulSets

### – Μοντελοποίησης σταθερότητας με StatefulSets

Το Kubernetes είναι μια δυναμική πλατφόρμα όπου τα αντικείμενα δημιουργούνται συνήθως παράλληλα και με τυχαία ονόματα. Αυτό είναι το τι συμβαίνει με τα Pods όταν δημιουργείτε ένα Deployment και είναι ένα μοτίβο που κλιμακώνεται καλά.

Ωστόσο, ορισμένες εφαρμογές χρειάζονται ένα σταθερό περιβάλλον, όπου τα αντικείμενα δημιουργούνται με γνωστή σειρά και με σταθερά ονόματα. Σκεφτείτε ένα αναπαραγόμενο σύστημα όπως μια ουρά μηνυμάτων ή μια βάση δεδομένων - συχνά υπάρχει ένας πρωτεύων κόμβος και πολλοί δευτερεύοντες. Οι δευτερεύοντες εξαρτώνται από το πρωτεύον που ξεκινά πρώτα και πρέπει να ξέρουν πώς να το βρουν, ώστε να μπορούν να συγχρονίσουν δεδομένα. Εκεί χρησιμοποιείτε ένα StatefulSet.

Τα StatefulSets είναι ελεγκτές Pod που μπορούν να δημιουργήσουν πολλαπλά αντίγραφα σε ένα σταθερό περιβάλλον. Τα αντίγραφα έχουν γνωστά ονόματα, ξεκινούν διαδοχικά και μπορούν να διευθυνσιοδοτηθούν μεμονωμένα μέσα στο cluster.

### – API specs

- [StatefulSet \(apps/v1\)](#)

Η προδιαγραφή είναι παρόμοια με τα Deployments - μεταδεδομένα, έναν επιλογή και ένα πρότυπο για την προδιαγραφή Pod - αλλά με μια σημαντική προσθήκη:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: simple-statefulset
spec:
  selector:
    matchLabels:
      app: simple-statefulset
  serviceName: simple-statefulset
  replicas: 3
  template:
    # Pod spec
```

- selector - ετικέτες για την αναγνώριση Pods που ανήκουν στο StatefulSet
- replicas - αριθμός Pods, η διαχείριση των οποίων θα γίνεται με τη σειρά
- serviceName - όνομα μιας Υπηρεσίας που παρέχει πρόσβαση δικτύου σε Pods.

Οι υπηρεσίες αποσυνδέονται από άλλους ελεγκτές Pod, αλλά απαιτείται μια Υπηρεσία για κάθε StatefulSet. Η Υπηρεσία χρησιμοποιεί μια ειδική ρύθμιση χωρίς ClusterIP:

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

```

  name: simple-statefulset
spec:
  ports:
    - port: 8010
      targetPort: 80
  selector:
    app: simple-statefulset
  clusterIP: None

```

- selector - ταιριάζει με τις ετικέτες των Pod
- clusterIP - η χρήση None απαιτείται για StatefulSets

Το StatefulSet έχει έναν σύνδεσμο προς την Υπηρεσία επειδή διαχειρίζεται τα end-points της Υπηρεσίας. Κάθε Pod έχει τη διεύθυνση IP του που προστίθεται στην Υπηρεσία και δημιουργείται ένα ξεχωριστό όνομα DNS για κάθε Pod.

#### – Αναπτύξτε ένα απλό StatefulSet

Θα ξεκινήσουμε με ένα σχετικά απλό παράδειγμα που εκτελεί πολλαπλά Nginx Pods. Αυτή η εφαρμογή δεν χρειάζεται να χρησιμοποιεί StatefulSet, αλλά δείχνει το μοτίβο χωρίς να γίνεται πολύ περίπλοκο:

```
docker pull java
```

```

# save it in a folder called statefulsets/specs/simple and
# name it services.yaml

# the headless Service and external Services to access the
# app

apiVersion: v1
kind: Service
metadata:
  name: simple-statefulset
  labels:
    kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8010
      targetPort: 80
  selector:
    app: simple-statefulset
  clusterIP: None
---
apiVersion: v1
kind: Service
metadata:
  name: simple-statefulset-lb

```

```

    labels:
      kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8010
      targetPort: 80
  selector:
    app: simple-statefulset
    type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: simple-statefulset-np
  labels:
    kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8010
      targetPort: 80
      nodePort: 30010
  selector:
    app: simple-statefulset
    type: NodePort

# save it in a folder called statefulsets/specs/simple and
# name it configmap-scripts.yaml

# ConfigMap with shell scripts the app uses for
# initialization

apiVersion: v1
kind: ConfigMap
metadata:
  name: simple-statefulset-scripts
  labels:
    kubernetes.courselabs.co: statefulsets
data:
  wait-service.sh: |-
    #!/bin/sh
    if [ "$HOSTNAME" == "$PRIMARY_NAME" ]; then
      echo '** I am the primary **'
    else
      echo '** I am a secondary - waiting for primary **'
      until nslookup ${PRIMARY_FQDN}; do echo Waiting for ${
PRIMARY_FQDN}; sleep 1; done
    fi
  create-html.sh: |-
    #!/bin/sh

```

```

    echo "<!DOCTYPE html><html><body><h1>I am $HOSTNAME</h1>
    </body></html>" > /html/index.html

# save it in a folder called statefulsets/specs/simple and
  name it statefulset.yaml

# StatefulSet which uses the headless Service and the scripts
  ; init containers ensure the secondaries don't start
    until the primary is ready, and then create the HTML to
    serve

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: simple-statefulset
  labels:
    kubernetes.courselabs.co: statefulsets
    app: simple-statefulset
spec:
  selector:
    matchLabels:
      app: simple-statefulset
  serviceName: simple-statefulset
  replicas: 3
  template:
    metadata:
      labels:
        app: simple-statefulset
    spec:
      initContainers:
        - name: wait-service
          image: kiamol/ch03-sleep
          env:
            - name: PRIMARY_NAME
              value: simple-statefulset-0
            - name: PRIMARY_FQDN
              value: simple-statefulset-0.simple-statefulset.
      default.svc.cluster.local
      command: ['/scripts/wait-service.sh']
      volumeMounts:
        - name: scripts
          mountPath: "/scripts"
        - name: create-html
          image: kiamol/ch03-sleep
          command: ['/scripts/create-html.sh']
          volumeMounts:
            - name: scripts
              mountPath: "/scripts"
            - name: html
              mountPath: /html

```

```

containers:
  - name: app
    image: nginx:1.18-alpine
    volumeMounts:
      - name: html
        mountPath: /usr/share/nginx/html
volumes:
  - name: scripts
    configMap:
      name: simple-statefulset-scripts
      defaultMode: 0500
  - name: html
    emptyDir: {}

```

- Το Pod 0 ξεκινά, το πρώτο σενάριο εκτελείται επιβεβαιώνοντας ότι αυτό το Pod είναι το κύριο, μετά εκτελείται το δεύτερο σενάριο και δημιουργεί το HTML. τότε εκτελείται το της δοχείο εφαρμογής
- Το Pod 1 ξεκινά, το πρώτο σενάριο εκτελείται και ελέγχει την καταχώρηση DNS για το Pod 0 - εάν δεν υπάρχει, τότε το πρωτεύον δεν είναι έτοιμο, οπότε το σενάριο περιμένει. Όταν το πρωτεύον συνδεθεί, το επόμενο σενάριο γράφει HTML και η εφαρμογή ξεκινά.
- Το Pod 2 ξεκινά - η ίδια διαδικασία με το Pod 1.
- Ας το δούμε στην πράξη:

```
kubectl apply -f statefulsets/specs/simple
```

```
kubectl get po -l app=simple-statefulset --watch
```

Θα δείτε δύο διαφορές από ένα Deployment - τα Pod δεν έχουν τυχαία ονόματα και κάθε Pod δημιουργείται μόνο όταν έχει ξεκινήσει το προηγούμενο Pod.

Ελέγξτε τα logs για το δοχείο wait-service σε κάθε ένα από τα Pods.

- Στα Pods με πολλά δοχεία, μπορείτε να προβάλετε τα logs για συγκεκριμένα δοχεία με την επιλογή -c. Αυτά τα logs θα εμφανίσουν τη ροή εργασίας εκκίνησης:

```
kubectl logs simple-statefulset-0 -c wait-service
```

```
kubectl logs simple-statefulset-1 -c wait-service
```

Το Pod-0 γνωρίζει ότι είναι το κύριο, επειδή έχει το αναμενόμενο "-0" hostname. Το Pod 1 γνωρίζει ότι είναι δευτερεύον επειδή δεν έχει αυτό το hostname

#### – Επικοινωνία με StatefulSet Pods

Τα StatefulSets προσθέτουν τις διευθύνσεις IP του Pod στην Υπηρεσία.



- Ελέγξτε ότι όλα τα Pods είναι καταχωρημένα στην Υπηρεσία.

```
kubectl get endpoints simple-statefulset
```

Υπάρχει μία Υπηρεσία με 3 διευθύνσεις IP Pod, αλλά αυτές οι ομάδες μπορούν επίσης να προσεγγιστούν χρησιμοποιώντας μεμονωμένα ονόματα τομέα.

- Εκτελέστε ένα Sleep Pod και κάντε μια αναζήτηση DNS για simple-statefulset και simple-statefulset-2.simple-statefulset.default.svc.cluster.local.

```
# save it in a folder called statefulsets/specs and name it
sleep-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: statefulsets
    app: sleep
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
```

```
kubectl apply -f statefulsets/specs/sleep-pod.yaml
```

```
kubectl exec sleep -- nslookup simple-statefulset
```

```
kubectl exec sleep -- nslookup simple-statefulset-2.simple-
statefulset.default.svc.cluster.local
```

Η εσωτερική Υπηρεσία επιστρέφει όλα τα Pod IP, αλλά κάθε Pod έχει επίσης τη δική του καταχώρηση DNS χρησιμοποιώντας το όνομα του Pod -0, -1 κ.λπ.

Αυτή η εφαρμογή διαθέτει υπηρεσίες LoadBalancer και NodePort με τον ίδιο επιλογέα Pod. Αυτά καθιστούν την εφαρμογή διαθέσιμη εξωτερικά και φορτώνουν αιτήματα ισορροπίας με τον συνηθισμένο τρόπο.

Περιηγηθείτε στο <http://localhost:8010> / <http://localhost:30010> και στη συνέχεια κάντε "Ctrl-refresh", θα δείτε απαντήσεις από διαφορετικά Pods.

- Τα StatefulSet Pods έχουν το όνομά τους σε μια ετικέτα, οπότε αν θέλετε να αποφύγετε την εξισορρόπηση φορτίου (π.χ. να στείλετε όλη την κίνηση σε ένα δευτερεύον), μπορείτε να καρφιτσώσετε την εξωτερική Υπηρεσία σε ένα συγκεκριμένο Pod:

```
# save it in a folder called statefulsets/specs/simple/update
and name it services.yaml

# it adds to the selector for the external Services, so they'
ll only send requests to Pod-1

apiVersion: v1
kind: Service
metadata:
  name: simple-statefulset-lb
  labels:
    kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8010
      targetPort: 80
  selector:
    app: simple-statefulset
    statefulset.kubernetes.io/pod-name: simple-statefulset-1
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: simple-statefulset-np
  labels:
    kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8010
      targetPort: 80
      nodePort: 30010
  selector:
    app: simple-statefulset
    statefulset.kubernetes.io/pod-name: simple-statefulset-1
  type: NodePort
```

- Αναπτύξτε την αλλαγή της Υπηρεσίας:

```
kubectl apply -f statefulsets/specs/simple/update
```

- Τώρα περιηγηθείτε στην εφαρμογή και οι απαντήσεις θα προέρχονται πάντα από το ίδιο Pod

#### – Αναπτύξτε μια αναπαραγόμενη βάση δεδομένων SQL

Κατέχουμε την κύρια ιδέα των StatefulSets, οπότε τώρα μπορούμε να αναπτύξουμε μια εφαρμογή που πραγματικά τα χρειάζεται - μια βάση δεδομένων Postgres με πρωτεύοντες και δευτερεύοντες κόμβους, καθένας από τους οποίους χρειάζεται ένα

PersistentVolumeClaim (PVC) για την αποθήκευση δεδομένων.

Θα χρησιμοποιήσουμε μια εικόνα Postgres Docker που έχει όλα τα σενάρια προετοιμασίας, οπότε δεν χρειάζεται να ανησυχούμε γι' αυτό (αν σας ενδιαφέρει θα τη βρείτε στο αποθετήριο [sixeyed/widgetario](#)).

Τα StatefulSets έχουν μια ειδική σχέση με το PersistentVolumeClaims, επομένως μπορείτε να ζητήσετε ένα PVC για κάθε Pod που παραμένει συνδεδεμένο με το Pod. Το Pod-1 θα έχει το δικό του PVC και όταν αναπτύσσετε μια ενημέρωση, στο νέο Pod-1 θα προσαρτάται το ίδιο PVC με το προηγούμενο Pod-1:

```
# save it in a folder called statefulsets/specs/products-db
and name it service.yaml

# headless Service, no external Services needed for these
Pods

apiVersion: v1
kind: Service
metadata:
  name: products-db
  labels:
    kubernetes.courselabs.co: statefulsets
    app: products-db
spec:
  ports:
    - port: 5432
      targetPort: postgres
  selector:
    app: products-db
  clusterIP: None

# save it in a folder called statefulsets/specs/products-db
and name it secret.yaml

# Secret containing the password for the Postgres user

apiVersion: v1
kind: Secret
metadata:
  name: products-db-password
  labels:
    kubernetes.courselabs.co: statefulsets
type: Opaque
stringData:
  POSTGRES_PASSWORD: wldgetar!0
  PGPASSWORD: wldgetar!0
```

```

# save it in a folder called statefulsets/specs/products-db
# and name it statefulset-with-pvc.yaml

# StatefulSet which runs two replicas, Pod-0 will be the
# primary and Pod-1 the secondary; each will have its own
# PVC created from the volume claim template

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: products-db
  labels:
    kubernetes.courselabs.co: statefulsets
    app: products-db
spec:
  selector:
    matchLabels:
      app: products-db
  serviceName: products-db
  replicas: 2
  template:
    metadata:
      labels:
        app: products-db
    spec:
      containers:
        - name: app
          image: widgetario/products-db:postgres-replicated
          envFrom:
            - secretRef:
                name: products-db-password
          ports:
            - containerPort: 5432
              name: postgres
          volumeMounts:
            - name: data
              mountPath: /var/lib/postgresql/data
      volumeClaimTemplates:
        - metadata:
            name: data
            labels:
              app: products-db
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 50Mi

```

- Αναπτύξτε τη βάση δεδομένων και παρακολουθήστε τα PVC που δημιουργούνται:

```
kubectl apply -f statefulsets/specs/products-db
kubectl get pvc -l app=products-db --watch
```

Θα δείτε να δημιουργείται ένα PVC για Pod-0 και, στη συνέχεια, όταν εκτελείται το Pod-0 δημιουργείται ένα άλλο PVC για το Pod-1

- Ελέγξτε τα logs του Pod-0 και θα δείτε ότι έχει οριστεί ως κύριο.

```
kubectl logs products-db-0
```

- Ελέγξτε το Pod-1 και ορίζεται από μόνο του ως δευτερεύον, όταν η βάση δεδομένων Postgres είναι ενεργοποιημένη και εκτελείται:

```
kubectl logs products-db-1
```

- Και τα δύο Pods θα πρέπει να τελειώνουν με ένα log λέγοντας ότι η βάση δεδομένων είναι έτοιμη να δεχτεί συνδέσεις:

```
kubectl logs -l app=products-db --tail 3
```

## – Εργαστηριακή άσκηση 21

Τα StatefulSets είναι πολύπλοκα και όχι τόσο κοινά όσο άλλοι ελεγκτές, αλλά έχουν ένα μεγάλο πλεονέκτημα έναντι των Deployments - μπορούν να παρέχουν δυναμικά ένα PVC για κάθε Pod.

Τα Deployments δεν σας επιτρέπουν να το κάνετε αυτό ([ακόμα](#)), επομένως μπορείτε να χρησιμοποιήσετε ένα StatefulSet για να επωφεληθείτε από τα πρότυπα διεκδίκησης τόμου.

Το simple-proxy/deployment.yaml είναι μια προδιαγραφή για την εκτέλεση ενός διακομιστή μεσολάβησης Nginx στον ιστότοπο StatefulSet που εκτελούμε.

```
# save it in a folder called statefulsets/specs/simple-proxy
# and name it deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: simple-proxy
  labels:
    kubernetes.courselabs.co: statefulsets
  app: simple-proxy
```

```
spec:
  selector:
    matchLabels:
      app: simple-proxy
  template:
    metadata:
      labels:
        app: simple-proxy
    spec:
      containers:
        - name: app
          image: nginx:1.18-alpine
          volumeMounts:
            - name: config
              mountPath: "/etc/nginx/"
              readOnly: true
            - name: cache
              mountPath: /cache
      volumes:
        - name: config
          configMap:
            name: simple-proxy-configmap
        - name: cache
          emptyDir: {}
```

- Αναπτύξτε τον proxy:

```
kubectl apply -f statefulsets/specs/simple-proxy
```

Δοκιμάστε ότι λειτουργεί στο `http://localhost:8040` / `http://localhost:30040`.

Η εργασία σας είναι να αντικαταστήσετε το Deployment που χρησιμοποιεί έναν τόμο `emptyDir` για αρχεία cache με ένα StatefulSet που χρησιμοποιεί ένα PVC για τη μνήμη cache για κάθε Pod.

Ο proxy δεν χρειάζεται διαδοχική διαχείριση των Pods, επομένως η προδιαγραφή θα πρέπει να ρυθμιστεί ώστε να δημιουργείται παράλληλα.

#### – EXTRA: Δοκιμή της αναπαραγόμενης βάσης δεδομένων

Μπορείτε να εκτελέσετε μια βάση δεδομένων SQL στα cluster δοκιμής σας. Δεν θέλετε να είναι δημόσια διαθέσιμο, αλλά θέλετε να μπορείτε να συνδεθείτε και να εκτελέσετε ερωτήματα. Η [εκτέλεση ενός προγράμματος-πελάτη SQL στο Kubernetes](#) σας καθοδηγεί σε αυτό.

#### – Καθάρισμα

```
kubectl delete svc,cm,secret,statefulset,deployment,pod -l
kubernetes.courselabs.co=statefulsets
```

### 3.4 Λειτουργώντας το Kubernetes

#### Ετοιμότητα παραγωγής

##### – Προετοιμασία για Παραγωγή

Είναι εύκολο να μοντελοποιήσετε τις εφαρμογές σας στο Kubernetes και να τις θέσετε σε λειτουργία, αλλά πρέπει να κάνετε περισσότερη δουλειά πριν φτάσετε στην παραγωγή.

Το Kubernetes μπορεί να διορθώσει εφαρμογές που έχουν προσωρινές αποτυχίες, να αναβαθμίσει αυτόματα τις εφαρμογές που είναι υπό φόρτωση και να προσθέσει στοιχεία ελέγχου ασφαλείας γύρω από τα δοχεία.

Αυτά είναι τα πράγματα που θα προσθέσετε στα μοντέλα εφαρμογών σας για να ετοιμαστείτε για παραγωγή.

##### – API specs

- [ContainerProbe](#)
- [HorizontalPodAutoscaler \(autoscaling/v1\)](#)

Οι ανιχνευτές δοχείων αποτελούν μέρος της προδιαγραφής του δοχείου εντός της προδιαγραφής Pod:

```
spec:
  containers:
    - # normal container spec
      readinessProbe:
        httpGet:
          path: /health
          port: 80
          periodSeconds: 5
```

- ReadinessProbe - υπάρχουν διαφορετικοί τύποι ανιχνευτή, αυτός ελέγχει ότι η εφαρμογή είναι έτοιμη να λαμβάνει αιτήματα δικτύου
- httpGet - λεπτομέρειες για την κλήση HTTP που πραγματοποιεί το Kubernetes για να δοκιμάσει την εφαρμογή - κωδικός απόκρισης non-OK σημαίνει ότι η εφαρμογή δεν είναι έτοιμη
- periodSeconds - πόσο συχνά να εκτελείτε τον ανιχνευτή.

Οι HorizontalPodAutoscalers (HPA) είναι ξεχωριστά αντικείμενα που αλληλεπιδρούν με έναν ελεγκτή Pod και ενεργοποιούν συμβάντα κλιμάκωσης με βάση τη χρήση της CPU:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: whoami-cpu
spec:
```

```

scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: whoami
minReplicas: 2
maxReplicas: 5
targetCPUUtilizationPercentage: 50

```

- `scaleTargetRef` - το αντικείμενο του ελεγκτή Pod για εργασία
- `minReplicas` - ελάχιστος αριθμός αντιγράφων
- `maxReplicas` - μέγιστος αριθμός αντιγράφων
- `targetCPUUtilizationPercentage` - μέσος στόχος χρήσης CPU - κάτω από αυτό το HPA θα μειωθεί, πάνω από αυτό το HPA θα κλιμακωθεί.

#### – Εφαρμογές αυτοθεραπείας με ανιχνευτές ετοιμότητας

Γνωρίζουμε ότι το Kubernetes κάνει επανεκκίνηση του Pods όταν το δοχείο τερματίζει, αλλά η εφαρμογή μέσα στο δοχείο θα μπορούσε να εκτελείται αλλά να μην ανταποκρίνεται - όπως μια εφαρμογή Ιστού που επιστρέφει το 503 - και η Kubernetes δεν θα το γνωρίζει.

Η εφαρμογή `whoami` έχει μια ωραία δυνατότητα που μπορούμε να χρησιμοποιήσουμε για να ενεργοποιήσουμε μια τέτοια αποτυχία.

```

# save it in a folder called productionizing/specs/whoami and
# name it deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q

```



```
# save it in a folder called productionizing/specs/whoami and
  name it services.yaml

apiVersion: v1
kind: Service
metadata:
  name: whoami-np
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8010
      targetPort: 80
      nodePort: 30010
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: whoami-lb
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    app: whoami
  ports:
    - name: http
      port: 8010
      targetPort: 80
    type: LoadBalancer
```

- Ξεκινήστε με την ανάπτυξη της εφαρμογής:

```
kubectl apply -f productionizing/specs/whoami
```

- Τώρα έχετε δύο Whoami Pods - κάντε μια εντολή POST και ένα από αυτά θα μεταβεί σε κατάσταση αποτυχίας:

```
# if you're on Windows, run this to use the correct curl:
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
  Process -Force; . ./scripts/windows-tools.ps1

curl http://localhost:8010

curl --data '503' http://localhost:8010/health
```

```
curl -i http://localhost:8010
```

Επαναλάβετε την τελευταία εντολή curl και θα λάβετε μερικές OK απαντήσεις και μερικά σφάλματα 503 - το Pod με την χαλασμένη εφαρμογή δεν διορθώνεται από μόνο του.

- Μπορείτε να πείτε στο Kubernetes πώς να ελέγξει ότι η εφαρμογή σας είναι υγιής με ανιχνευτές δοχείων. Ορίζετε την ενέργεια για το probe και το Kubernetes την εκτελεί επανειλημμένα για να βεβαιωθεί ότι η εφαρμογή είναι υγιής:

```
# save it in a folder called productionizing/specs/whoami/
# update and name it deployment-with-readiness.yaml

# it adds a readiness probe, which makes an HTTP call to the
# /health endpoint of the app every 5 seconds

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      update: readiness
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q
          readinessProbe:
            httpGet:
              path: /health
              port: 80
              periodSeconds: 5
```

- Αναπτύξτε την ενημέρωση στο productionizing/specs/whoami/update και περιμένετε να είναι έτοιμα τα Pods με ετικέτα update=readiness.

```
kubectl apply -f productionizing/specs/whoami/update
kubectl wait --for=condition=Ready pod -l app=whoami,update=
  readiness
```

Περιγράψτε ένα Pod και θα δείτε τον έλεγχο ετοιμότητας να αναφέρεται στην έξοδο

- Αυτά είναι νέα Pods, ώστε η εφαρμογή να είναι υγιής και στα δύο. Μεταβείτε ένα Pod σε μια ανθυγιεινή κατάσταση και θα δείτε την κατάσταση να αλλάζει:

```
curl --data '503' http://localhost:8010/health
kubectl get po -l app=whoami --watch
```

To One Pod αλλάζει στη στήλη Ready - τώρα 0/1 δοχεία είναι έτοιμα.

- Αυτά είναι νέα Pods, ώστε η εφαρμογή να είναι υγιής και στα δύο. Μεταβείτε ένα Pod σε μια ανθυγιεινή κατάσταση και θα δείτε την κατάσταση να αλλάζει:

```
curl --data '503' http://localhost:8010/health
kubectl get po -l app=whoami --watch
```

To One Pod αλλάζει στη στήλη Ready - τώρα 0/1 δοχεία είναι έτοιμα.

Εάν αποτύχει ένας έλεγχος ετοιμότητας, το Pod αφαιρείται από την Υπηρεσία και δεν θα λάβει καθόλου κίνηση.

- Επιβεβαιώστε ότι η Υπηρεσία έχει μόνο ένα Pod IP και δοκιμάστε την εφαρμογή.

```
# Ctrl-C to exit the watch
kubectl get endpoints whoami-np
curl http://localhost:8010
```

Μόνο το υγιές Pod είναι εγγεγραμμένο στην Υπηρεσία, επομένως θα λαμβάνετε πάντα μια OK απάντηση.

Εάν αυτή ήταν μια πραγματική εφαρμογή, το 503 θα μπορούσε να συμβεί εάν η εφαρμογή είναι υπερφορτωμένη. Η κατάργησή της από την Υπηρεσία μπορεί να της δώσει χρόνο για ανάκτηση.

– **Εφαρμογές αυτοεπιδιόρθωσης με ανιχνευτές ζωντάνιας**

Οι ανιχνευτές ετοιμότητας απομονώνουν τα αποτυχημένα Pods από τον Εξισορροπητή φόρτου Υπηρεσίας, αλλά δεν προβαίνουν σε ενέργειες για την επιδιόρθωση της εφαρμογής.

- Για αυτό, μπορείτε να χρησιμοποιήσετε έναν ανιχνευτή ζωντάνιας που θα επανεκκινήσει το Pod με ένα νέο δοχείο, εάν ο ανιχνευτής αποτύχει:

```
# save it in a folder called productionizing/specs/whoami/
# update2 and name it deployment-with-liveness.yaml

# it adds a liveness check; this one uses the same test as
# the readiness probe

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
        update: liveness
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q
          readinessProbe:
            httpGet:
              path: /health
              port: 80
              periodSeconds: 5
          livenessProbe:
            httpGet:
              path: /health
              port: 80
              periodSeconds: 10
              failureThreshold: 3
```

Θα κάνετε συχνά τα ίδια τεστ για ετοιμότητα και ζωντάνια, αλλά ο έλεγχος ζωντάνιας έχει πιο σημαντικές συνέπειες, επομένως μπορεί να θέλετε να εκτελείται λιγότερο συχνά και να έχετε υψηλότερο όριο αποτυχίας.

- Αναπτύξτε την ενημέρωση στο `productionizing/specs/whoami/update2` και περιμένετε να είναι έτοιμα τα Pods με ετικέτα `update=liveness`.

```
kubectl apply -f productionizing/specs/whoami/update2

kubectl wait --for=condition=Ready pod -l app=whoami,update=
liveness
```

- Τώρα ενεργοποιήστε μια αποτυχία σε ένα Pod και παρακολουθήστε για να βεβαιωθείτε ότι θα επανεκκινηθεί.

```
curl --data '503' http://localhost:8010/health

kubectl get po -l app=whoami --watch
```

Ένα Pod θα είναι `ready 0/1` - μετά θα επανεκκινήσει και μετά θα γίνει ξανά `ready 1/1`.

Ελέγξτε το endpoint και θα δείτε ότι και οι δύο Pod IP βρίσκονται στη λίστα Service. Όταν το επανεκκινημένο Pod πέρασε τον έλεγχο ετοιμότητας, προστέθηκε ξανά.

- Υπάρχουν και άλλοι τύποι ανιχνευτών, επομένως αυτό δεν είναι μόνο για εφαρμογές HTTP. Αυτή η προδιαγραφή Postgres Pod χρησιμοποιεί έναν ανιχνευτή TCP και έναν ανιχνευτή εντολών:

```
# save it in a folder called productionizing/specs/products-
db and name it products-db.yaml

# it has a readiness probe to test Postgres is listening and
a liveness probe to test the database is usable

apiVersion: apps/v1
kind: Deployment
metadata:
  name: products-db
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    matchLabels:
      app: products-db
  template:
```

```

metadata:
  labels:
    app: products-db
spec:
  containers:
    - name: db
      image: widgetario/products-db:postgres
      readinessProbe:
        tcpSocket:
          port: 5432
        periodSeconds: 5
        initialDelaySeconds: 10
      livenessProbe:
        exec:
          command: ["pg_isready", "-h", "localhost"]
        periodSeconds: 30
        initialDelaySeconds: 20
        failureThreshold: 5

```

#### – Αυτόματη κλιμάκωση υπολογιστικών φόρτων εργασίας

Ένα Kubernetes cluster είναι μια δεξαμενή πόρων CPU και μνήμης. Εάν έχετε φόρτο εργασίας με διαφορετική ζήτηση, μπορείτε να χρησιμοποιήσετε ένα [HorizontalPodAutoscaler](#) για αυτόματη κλιμάκωση των Pods προς τα πάνω και προς τα κάτω, εφόσον το cluster σας έχει χωρητικότητα.

- Η βασική αυτόματη κλίμακα χρησιμοποιεί μετρήσεις CPU που τροφοδοτούνται από το έργο [metrics-server](#). Δεν το έχουν εγκαταστήσει όλα τα clusters, αλλά είναι εύκολο να ρυθμιστεί:

```

# save it in a folder called productionizing/specs/metrics-
# server and name it components-v0.6.1.yaml

apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    k8s-app: metrics-server
  rbac.authorization.k8s.io/aggregate-to-admin: "true"
  rbac.authorization.k8s.io/aggregate-to-edit: "true"
  rbac.authorization.k8s.io/aggregate-to-view: "true"
  name: system:aggregated-metrics-reader

```

```

rules:
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    k8s-app: metrics-server
  name: system:metrics-server
rules:
- apiGroups:
  - ""
  resources:
  - nodes/metrics
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server-auth-reader
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: extension-apiserver-authentication-reader
subjects:
- kind: ServiceAccount
  name: metrics-server
  namespace: kube-system
---

```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server:system:auth-delegator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
- kind: ServiceAccount
  name: metrics-server
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    k8s-app: metrics-server
  name: system:metrics-server
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:metrics-server
subjects:
- kind: ServiceAccount
  name: metrics-server
  namespace: kube-system
---
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server
  namespace: kube-system
spec:
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: https
  selector:
    k8s-app: metrics-server
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:

```



```

    k8s-app: metrics-server
  name: metrics-server
  namespace: kube-system
spec:
  selector:
    matchLabels:
      k8s-app: metrics-server
  strategy:
    rollingUpdate:
      maxUnavailable: 0
  template:
    metadata:
      labels:
        k8s-app: metrics-server
    spec:
      containers:
        - args:
            - --cert-dir=/tmp
            - --secure-port=4443
            - --kubelet-preferred-address-types=InternalIP,
ExternalIP,Hostname
            - --kubelet-use-node-status-port
            - --metric-resolution=15s
            - --kubelet-insecure-tls
          image: k8s.gcr.io/metrics-server/metrics-server:v0
.6.1
          imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /livez
            port: https
            scheme: HTTPS
            periodSeconds: 10
          name: metrics-server
        ports:
          - containerPort: 4443
            name: https
            protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /readyz
            port: https
            scheme: HTTPS
            initialDelaySeconds: 20
            periodSeconds: 10
        resources:
          requests:
            cpu: 100m

```

```

        memory: 200Mi
        securityContext:
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: true
          runAsNonRoot: true
          runAsUser: 1000
        volumeMounts:
        - mountPath: /tmp
          name: tmp-dir
      nodeSelector:
        kubernetes.io/os: linux
      priorityClassName: system-cluster-critical
      serviceAccountName: metrics-server
      volumes:
      - emptyDir: {}
        name: tmp-dir
---
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  labels:
    k8s-app: metrics-server
  name: v1beta1.metrics.k8s.io
spec:
  group: metrics.k8s.io
  groupPriorityMinimum: 100
  insecureSkipTLSVerify: true
  service:
    name: metrics-server
    namespace: kube-system
  version: v1beta1
  versionPriority: 100

```

```
kubectl top nodes
```

```
# if you see "error: Metrics API not available" run this:
```

```
kubectl apply -f productionizing/specs/metrics-server
```

```
kubectl top nodes
```

- Η εφαρμογή Pi απαιτεί υψηλό φόρτο εργασίας, επομένως είναι ένας καλός στόχος για ένα HPA:

```
# save it in a folder called productionizing/specs/pi and
  name it deployment.yaml
```

```
# Deployment which includes CPU resources
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
    spec:
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: web
          ports:
            - containerPort: 80
              name: http
          resources:
            limits:
              cpu: 250m
            requests:
              cpu: 125m

# save it in a folder called productionizing/specs/pi and
# name it hpa-cpu.yaml

# HPA which will scale the Deployment, using 75% utilization
# of requested CPU as the threshold

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: pi-cpu
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: pi-web
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 75

```

- Αναπτύξτε την εφαρμογή από το `productionizing/specs/pi`, ελέγξτε τις μετρήσεις για το Pod και εκτυπώστε τις λεπτομέρειες για το HPA.

```
kubectl apply -f productionizing/specs/pi
kubectl top pod -l app=pi-web
kubectl get hpa pi-cpu --watch
```

Αρχικά το Pod είναι στο 0% CPU. Ανοίξτε 2 καρτέλες του προγράμματος περιήγησης που δείχνουν στη διεύθυνση `http://localhost:8020/pi?dp=100000` - αυτή είναι αρκετή δουλειά για να μεγιστοποιήσετε τον φόρτο εργασίας του Pod και να ενεργοποιήσετε το HPA.

Εάν το cluster σας τηρεί τα όρια της CPU, το HPA θα ξεκινήσει περισσότερα Pods. Μετά την επεξεργασία των αιτημάτων, ο φόρτος εργασίας μειώνεται, οπότε ο μέσος όρος της CPU σε όλα τα Pods είναι κάτω από το όριο και, στη συνέχεια, το HPA μειώνεται.

Το Docker Desktop έχει επί του παρόντος ένα πρόβλημα με την αναφορά των μετρήσεων για τα Pods. Εάν εκτελέσετε το `"kubectl top pod"` και δείτε `"error: Metrics not available for pod..."` τότε το HPA δεν θα ενεργοποιηθεί. [Εδώ είναι το πρόβλημα](#) - αλλά δεν συνιστώ να ακολουθήσετε τη διαδικασία για να το διορθώσετε.

- Οι προεπιλεγμένες ρυθμίσεις περιμένουν μερικά λεπτά πριν την κλιμάκωση και μερικά ακόμη πριν την αποκλιμάκωση.

## – Εργαστηριακή άσκηση 22

Η προσθήκη ανησυχιών για την παραγωγή είναι συχνά κάτι που θα κάνετε αφού ολοκληρώσετε την αρχική μοντελοποίηση και ξεκινήσετε την εφαρμογή σας.

- Επομένως, η εργασία σας είναι να προσθέσετε ανιχνευτές δοχείων και ρυθμίσεις ασφαλείας στην διαμορφώσιμη εφαρμογή. Ξεκινήστε εκτελώντας το με μια βασική προδιαγραφή:

```
# save it in a folder called productionizing/specs/
# configurable and name it deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
```

```

    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          env:
            - name: Configurable__FailAfterCallCount
              value: "3"

# save it in a folder called productionizing/specs/
# configurable and name it services.yaml

apiVersion: v1
kind: Service
metadata:
  name: configurable-np
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    app: configurable
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30040
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: configurable-lb
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    app: configurable
  ports:
    - name: http
      port: 8040
      targetPort: 80
    type: LoadBalancer

```

Δοκιμάστε την εφαρμογή και θα δείτε ότι αποτυγχάνει μετά από 3 ανανεώσεις και δεν θα επιστρέψει ποτέ ξανά στο διαδίκτυο. Υπάρχει ένα endpoint /healthz που μπορείτε να χρησιμοποιήσετε για να το ελέγξετε. Οι στόχοι σας είναι:

- εκτελέστε 5 αντίγραφα και βεβαιωθείτε ότι η κίνηση αποστέλλεται μόνο σε υγιή Pods
- επανεκκινήστε τα Pods εάν αποτύχει η εφαρμογή στο δοχείων
- προσθέστε ένα HPA ως αντίγραφο ασφαλείας, κλιμακώνοντας έως το 10 εάν τα Pods χρησιμοποιούν περισσότερο από 50% CPU.

Αυτή η εφαρμογή δεν είναι CPU-intensive, επομένως δεν θα μπορείτε να ενεργοποιήσετε το HPA πραγματοποιώντας κλήσεις HTTP. Πώς αλλιώς μπορείτε να δοκιμάσετε ότι το HPA κλιμακώνεται και αποκλιμακώνεται σωστά; //

#### – EXTRA: Ασφάλεια Pod

Τα όρια πόρων των δοχείων είναι απαραίτητα για τα HPA, αλλά θα πρέπει να τα έχετε σε όλες τις προδιαγραφές του Pod σας επειδή παρέχουν ένα επίπεδο ασφάλειας. Η εφαρμογή ορίων CPU και μνήμης προστατεύει τους κόμβους και σημαίνει ότι ο φόρτος εργασίας δεν μπορεί να μεγιστοποιήσει τους πόρους και να τερματίσει άλλα Pods.

Η ασφάλεια είναι ένα πολύ μεγάλο θέμα στα δοχεία, αλλά υπάρχουν μερικά χαρακτηριστικά που πρέπει να στοχεύσετε να συμπεριλάβετε σε όλες τις προδιαγραφές σας:

- αλλαγή του χρήστη για να διασφαλιστεί ότι η διαδικασία του δοχείου δεν εκτελείται ως root
- μην προσαρτήσετε το διακριτικό API λογαριασμού υπηρεσίας εκτός και αν το χρειάζεται η εφαρμογή σας
- προσθέστε ένα [πλαίσιο ασφαλείας](#) για να περιορίσετε τις δυνατότητες του λειτουργικού συστήματος που μπορεί να χρησιμοποιήσει η εφαρμογή.
- Το Kubernetes δεν τα εφαρμόζει από προεπιλογή, επειδή μπορεί να προκαλέσουν καταστροφικές αλλαγές στην εφαρμογή σας.

```
kubectl exec deploy/pi-web -- whoami

kubectl exec deploy/pi-web -- cat /var/run/secrets/kubernetes
.io/serviceaccount/token

kubectl exec deploy/pi-web -- chown root:root /app/Pi.Web.dll
```

Η εφαρμογή εκτελείται ως root, έχει ένα διακριτικό για τη χρήση του διακομιστή Kubernetes API και έχει ισχυρά δικαιώματα λειτουργικού συστήματος.

- Αυτή η εναλλακτική προδιαγραφή διορθώνει αυτά τα ζητήματα ασφαλείας:

```

# save it in a folder called productionizing/specs/pi-secure
  and name it deployment.yaml

# it sets a non-root user, doesn't mount the SA token and
  drops Linux capabilities

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-secure-web
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    matchLabels:
      app: pi-secure-web
  template:
    metadata:
      labels:
        app: pi-secure-web
    spec:
      automountServiceAccountToken: false
      securityContext:
        runAsUser: 65534
        runAsGroup: 3000
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: web
          ports:
            - containerPort: 80
              name: http
          resources:
            limits:
              cpu: 250m
            requests:
              cpu: 125m
          securityContext:
            allowPrivilegeEscalation: false
          capabilities:
            drop:
              - all

```

```
kubectl apply -f productionizing/specs/pi-secure/
```

```
kubectl get pod -l app=pi-secure-web --watch
```

Η προδιαγραφή είναι πιο ασφαλής, αλλά η εφαρμογή αποτυγχάνει. Ελέγξτε τα logs και θα δείτε ότι δεν έχει άδεια ακρόασης στη θύρα.

- Η θύρα 80 είναι προνομαϊκή μέσα στο δοχείο, επομένως οι εφαρμογές δεν μπορούν να την ανταποκριθούν ως χρήστες με λιγότερα προνόμια χωρίς δυνατότητες Linux. Αυτή είναι μια εφαρμογή .NET που μπορεί να χρησιμοποιήσει μια προσαρμοσμένη θύρα:

```
# save it in a folder called productionizing/specs/pi-secure/
# update and name it deployment-custom-port.yaml

# it configures the app to listen on non-privileged port 5001

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-secure-web
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  selector:
    matchLabels:
      app: pi-secure-web
  template:
    metadata:
      labels:
        app: pi-secure-web
        update: ports
    spec:
      automountServiceAccountToken: false
      securityContext:
        runAsUser: 65534
        runAsGroup: 3000
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: web
          env:
            - name: ASPNETCORE_URLS
              value: http://+:5001
          ports:
            - containerPort: 5001
              name: http
          resources:
            limits:
              cpu: 250m
            requests:
              cpu: 125m
            securityContext:
```



```
allowPrivilegeEscalation: false
capabilities:
  drop:
    - all
```

- Αναπτύξτε την ενημέρωση και ελέγξτε ότι διορθώνει αυτές τις τρύπες ασφαλείας.

```
kubectl apply -f productionizing/specs/pi-secure/update
kubectl wait --for=condition=Ready pod -l app=pi-secure-web,
update=ports
```

- Το δοχείο Pod εκτελείται, επομένως η εφαρμογή ανταποκρίνεται και τώρα είναι πιο ασφαλής:

```
kubectl exec deploy/pi-secure-web -- whoami
kubectl exec deploy/pi-secure-web -- cat /var/run/secrets/
kubernetes.io/serviceaccount/token
kubectl exec deploy/pi-secure-web -- chown root:root /app/Pi.
Web.dll
```

Αυτό δεν είναι το τέλος της ασφάλειας - είναι μόνο η αρχή. Η ασφάλιση δοχείων είναι μια προσέγγιση πολλαπλών επιπέδων που ξεκινά με την ασφάλιση των εικόνων σας, αλλά αυτό είναι ένα καλό βήμα παραπάνω από την προεπιλεγμένη ασφάλεια Pod.

## – Καθάρισμα

```
kubectl delete all,hpa -l kubernetes.courselabs.co=
productionizing
```

## Παρακολούθηση

### – Παρακολούθηση με Prometheus και Grafana

Το Kubernetes μπορεί να τρέξει εκατοντάδες δοχεία σε δεκάδες διακομιστές. Για να τα παρακολουθείτε, εκτελείτε έναν διακομιστή παρακολούθησης, όπως το [Prometheus](#), ο οποίος συλλέγει μετρήσεις από όλα τα δοχεία σας. Ο Prometheus αποθηκεύει τα δεδομένα και σας επιτρέπει να τα αναζητήσετε - και μπορείτε να οπτικοποιήσετε τους πίνακες χρησιμοποιώντας το [Grafana](#).

Αναφορές:

- [Helm chart for installing Prometheus](#)

- [Configuring Kubernetes service discovery](#)
- [Prometheus client libraries](#) και [exporters](#)

Ο διακομιστής Prometheus εκτελείται σε ένα Pod και συνδέεται με το Kubernetes API για να βρει άλλα Pod. Μπορείτε να διαμορφώσετε την ανακάλυψη υπηρεσίας ώστε να είναι opt-in (δεν παρακολουθούνται Pods από προεπιλογή) ή opt-out (όλα τα Pods παρακολουθούνται από προεπιλογή). Τα Application Pods χρησιμοποιούν σχολιασμούς για να διαμορφώσουν τον τρόπο με τον οποίο πρέπει να παρακολουθούνται.

- Έτσι μοιάζει η διαμόρφωση YAML του Prometheus:

```
scrape_configs:
  - job_name: 'app'
    kubernetes_sd_configs:
      - role: pod

    relabel_configs:
      - source_labels:
          - __meta_kubernetes_namespace
        regex: my-app
        action: keep

      - source_labels:
          - __meta_kubernetes_pod_annotationpresent_prometheus_io_scrape
          - __meta_kubernetes_pod_annotation_prometheus_io_scrape
        regex: true;true
        action: keep
```

- το `kubernetes_sd_configs` θέτει τον Prometheus να αναζητά όλα τα Pods
- το `relabel_configs` εξετάζει τα μεταδεδομένα Pod και περιλαμβάνει μόνο Pods για scrape εάν βρίσκονται στο Namespace της εφαρμογής μου και έχει εφαρμοστεί ο σχολιασμός.
- Αυτή η προδιαγραφή Pod θα συμπεριληφθεί στη διαμόρφωση του scrape:

```
kind: Pod
metadata:
  name: whoami
  namespace: my-app
  annotations:
    prometheus.io/scrape: 'true'
spec:
  containers:
    - name: app
      image: sixeyed/whoami:21.04
```

Μπορούν να χρησιμοποιηθούν άλλοι σχολιασμοί για τη διαμόρφωση της θύρας και της διαδρομής HTTP που θα πρέπει να χρησιμοποιεί ο Prometheus για τη συλλογή μετρήσεων.

– **Αναπτύξτε μια εφαρμογή που δημοσιεύει μετρήσεις**

Αυτή είναι μια εφαρμογή επεξεργασίας που λειτουργεί παρασκήνιο και έχει δημιουργηθεί με την υποστήριξη του Prometheus:

```
# save it in a folder called monitoring/specs/fulfilment-processor and name it deployment.yaml

# it includes the Prometheus annotations

apiVersion: apps/v1
kind: Deployment
metadata:
  name: fulfilment-processor
  labels:
    kubernetes.courselabs.co: monitoring
spec:
  selector:
    matchLabels:
      app: fulfilment
      component: processor
  template:
    metadata:
      labels:
        app: fulfilment
        component: processor
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '9110'
    spec:
      containers:
        - name: app
          image: courselabs/fulfilment-processor
          env:
            - name: Observability__Metrics__IncludeRuntime
              value: 'true'
          ports:
            - containerPort: 9110
              name: metrics
```

• **Αναπτύξτε την εφαρμογή:**

```
kubectl apply -f monitoring/specs/fulfilment-processor

kubectl get all -l kubernetes.courselabs.co=monitoring
```

Όταν το Pod είναι έτοιμο, περιηγηθείτε στο endpoint των μετρήσεων στη διεύθυνση `http://localhost:9110/metrics` ή `http://localhost:30910/metrics`.

Αυτή δεν είναι μια εφαρμογή Ιστού - εκτελεί έναν διακομιστή HTTP καθαρά για να παρέχει τις μετρήσεις. Ανανεώστε τη σελίδα και θα δείτε μερικούς από τους αριθμούς να αλλάζουν:

- Το `fulfilment_in_flight_total` είναι ένας μετρητής, ο αριθμός μπορεί να αυξηθεί ή να μειωθεί
- Το `process_cpu_seconds_total` είναι ένας μετρητής, ο αριθμός μπορεί μόνο να αυξηθεί (ή να παραμείνει ο ίδιος).

Αυτά είναι τα ακατέργαστα δεδομένα παρακολούθησης που θα συλλέξει ο Prometheus.

#### – Αναπτύξτε τη στοίβα παρακολούθησης

Θα χρησιμοποιήσουμε ξεχωριστό Namespace για την παρακολούθηση:

```
# save it in a folder called monitoring/specs/monitoring and
# name it prometheus.yaml

# it specs out Prometheus to run in a Deployment, with
# Services to access the web UI and RBAC rules to give
# access to the Kubernetes API

apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring
spec:
  selector:
    app: prometheus
  type: LoadBalancer
  ports:
    - port: 9090
      targetPort: prometheus
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus-np
  namespace: monitoring
spec:
  selector:
    app: prometheus
  type: NodePort
  ports:
    - port: 9090
      targetPort: prometheus
```

```

        nodePort: 30019
    ---
    apiVersion: apps/v1
    kind: Deployment
    metadata:
        name: prometheus
        namespace: monitoring
    spec:
        selector:
            matchLabels:
                app: prometheus
        template:
            metadata:
                labels:
                    app: prometheus
            spec:
                serviceAccountName: prometheus
                containers:
                    - name: prometheus
                      image: courselabs/prometheus:v2.28.1
                      args:
                          - "--config.file=/config/prometheus.yml"
                      ports:
                          - name: prometheus
                            containerPort: 9090
                      volumeMounts:
                          - name: prometheus-config
                            mountPath: /config/
                volumes:
                    - name: prometheus-config
                      configMap:
                          name: prometheus-config
    ---
    apiVersion: v1
    kind: ServiceAccount
    metadata:
        name: prometheus
        namespace: monitoring
    ---
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
        name: prometheus
        labels:
            kubernetes.courselabs.co: monitoring
    rules:
    - apiGroups: ["" ]
      resources:
        - nodes
        - services

```

```

- endpoints
- pods
verbs: ["get", "list", "watch"]
- apiGroups: [""]
resources:
- configmaps
verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
  labels:
    kubernetes.courselabs.co: monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: monitoring

# save it in a folder called monitoring/specs/monitoring and
# name it prometheus-config.yaml

# it has the Prometheus configuration for discovering Pods

apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s

    scrape_configs:
      - job_name: 'app'
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels:
              - __meta_kubernetes_namespace
            action: keep
            regex: default
          - source_labels:

```

```

-
__meta_kubernetes_pod_annotationpresent_prometheus_io_scrape
-
__meta_kubernetes_pod_annotation_prometheus_io_scrape
  regex: true;true
  action: keep
- source_labels:
-
__meta_kubernetes_pod_annotationpresent_prometheus_io_path
-
__meta_kubernetes_pod_annotation_prometheus_io_path
  regex: true;(.*)
  target_label: __metrics_path__
- source_labels:
-
__meta_kubernetes_pod_annotationpresent_prometheus_io_target
-
__meta_kubernetes_pod_annotation_prometheus_io_target
  regex: true;(.*)
  target_label: __param_target
- source_labels:
-
__meta_kubernetes_pod_annotationpresent_prometheus_io_port
-
  - __address__
-
__meta_kubernetes_pod_annotation_prometheus_io_port
  action: replace
  regex: true;([^\:]+)(?::\d+)?;(\d+)
  replacement: $1:$2
  target_label: __address__
- source_labels:
  - __meta_kubernetes_pod_labelpresent_component
  - __meta_kubernetes_pod_label_component
  regex: true;(.*)
  target_label: job
- source_labels:
  - __meta_kubernetes_pod_name
  target_label: instance

```

Τι ρύθμιση χρειάζονται τα Pods ώστε να συμπεριληφθούν στην ανακάλυψη του Prometheus;

Αυτή είναι μια ρύθμιση παραμέτρων επιλογής. Οι λοβοί πρέπει να:

- εκτέλεση στον προεπιλεγμένο Namespace

- έχουν τον σχολιασμό `prometheus.io/scrape: 'true'`

Ο επεξεργαστής εκπλήρωσης Pod ταιριάζει με τη διαμόρφωση εντοπισμού, επομένως ο Prometheus θα αρχίσει να ξύνει μετρήσεις όταν βρει το Pod.

Αναπτύξτε τα στοιχεία παρακολούθησης και περιηγηθείτε στη διαδρομή `/targets` για την Υπηρεσία Prometheus.

- Αναπτύξτε τις προδιαγραφές:

```
kubectl apply -f monitoring/specs/monitoring
```

- Ελέγξτε τη ρύθμιση της υπηρεσίας:

```
kubectl get svc -n monitoring
```

Περιηγηθείτε στο `http://localhost:9090/targets` ή `http://localhost:30990/targets`.

Θα δείτε το Pod του επεξεργαστή εκπλήρωσης στη λίστα - θα πρέπει να βρίσκεται σε κατάσταση Up.

Μεταβείτε στη σελίδα Graph και δείτε ορισμένες μετρήσεις:

- πληκτρολογήστε `"fulfilment_requests_total"` στον πίνακα Expression για να δείτε τα τρέχοντα δεδομένα. Στη συνέχεια μεταβείτε στην προβολή γραφήματος για να δείτε τα δεδομένα να αλλάζουν με την πάροδο του χρόνου
- κάντε το ίδιο για το `"fulfilment_in_flight_total"`.

Θα δείτε ένα σύνολο γραμμών γραφήματος να αυξάνεται πάντα και το άλλο να ανεβαίνει και προς τα κάτω.

Οι εκφράσεις χρησιμοποιούν μια προσαρμοσμένη γλώσσα ερωτημάτων που ονομάζεται PromQL. Είναι χρήσιμο να δοκιμάσετε τα ερωτήματά σας στο Prometheus UI, αλλά δεν είναι ένα πλήρες εργαλείο οπτικοποίησης - για πίνακες εργαλείων θα χρησιμοποιήσετε το Grafana.

#### – Φορτώστε τον πίνακα ελέγχου της εφαρμογής στο Grafana

Το Grafana συνδέεται με τον Prometheus - η ανάπτυξη χρησιμοποιεί ConfigMaps και Secrets, ώστε το περιβάλλον χρήστη να έχει ήδη διαμορφωθεί:

```
# save it in a folder called monitoring/specs/monitoring and
# name it grafana-config.yaml

# it sets up Prometheus as a data source which Grafana can
# query

apiVersion: v1
```



```

kind: ConfigMap
metadata:
  name: grafana-config
  namespace: monitoring
data:
  datasources.yaml: |-
    apiVersion: 1
    datasources:
    - name: Prometheus
      type: prometheus
      access: proxy
      url: http://prometheus:9090
      basicAuth: false
      version: 1
      editable: true
      isDefault: true

# save it in a folder called monitoring/specs/monitoring and
# name it grafana.yml

# it contains a Secret for the admin user credentials.

apiVersion: v1
kind: Service
metadata:
  name: grafana-lb
  namespace: monitoring
spec:
  selector:
    app: grafana
  ports:
    - name: grafana
      port: 3000
      targetPort: grafana
      type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: grafana-np
  namespace: monitoring
spec:
  selector:
    app: grafana
  ports:
    - name: grafana
      port: 3000
      targetPort: grafana
      nodePort: 30030
      type: NodePort

```

```

---
apiVersion: v1
kind: Secret
metadata:
  name: grafana-creds
  namespace: monitoring
stringData:
  GF_SECURITY_ADMIN_USER: admin
  GF_SECURITY_ADMIN_PASSWORD: labs
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - image: courselabs/grafana:8.0.5
          name: grafana
          ports:
            - containerPort: 3000
              name: grafana
          envFrom:
            - secretRef:
                name: grafana-creds
          env:
            - name: GF_USERS_DEFAULT_THEME
              value: "light"
          volumeMounts:
            - name: config-datasources
              mountPath: "/etc/grafana/provisioning/
datasources"
              readOnly: true
      volumes:
        - name: config-datasources
          configMap:
            name: grafana-config
            items:
              - key: datasources.yaml
                path: datasources.yaml

```

Περιηγηθείτε στη διεπαφή χρήστη Grafana και συνδεθείτε ως διαχειριστής.

- Ελέγξτε την Υπηρεσία για να βρείτε τη θύρα:

```
kubectl get svc -n monitoring
```

- Εκτυπώστε τις λεπτομέρειες απλού κειμένου του Secret:

```
# find the variable names:
kubectl describe secret -n monitoring grafana-creds

# print admin username:
kubectl get secret -n monitoring grafana-creds -o jsonpath="{.data.GF_SECURITY_ADMIN_USER}" | base64 -d

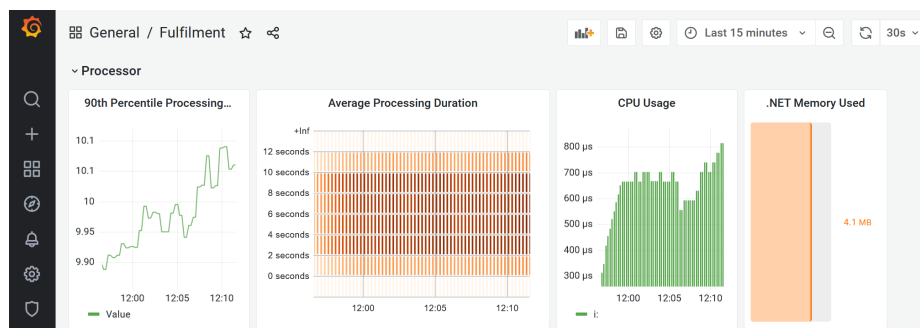
# print admin password:
kubectl get secret -n monitoring grafana-creds -o jsonpath="{.data.GF_SECURITY_ADMIN_PASSWORD}" | base64 -d
```

Μεταβείτε στη διεύθυνση <http://localhost:3000> ή <http://localhost:30300> και συνδεθείτε στα τα εργαστήρια διαχείρισης με το όνομα χρήστη και κωδικό πρόσβασης.

Τώρα είμαστε έτοιμοι να φορτώσουμε τον πίνακα ελέγχου της εφαρμογής - έχουμε έναν έτοιμο που μπορούμε να εισαγάγουμε. Κάντε κλικ στο εικονίδιο + στο αριστερό μενού πλοήγησης και επιλέξτε "Import".

Κάντε κλικ στην επιλογή "Upload JSON file" και περιηγηθείτε στους φακέλους σας, για να φορτώσετε το αρχείο `monitoring/dashboards/fulfilment-processor.json`.

Κάντε κλικ στο "Import" και ο πίνακας εργαλείων θα φορτώσει. Μοιάζει κάπως έτσι:



Όλα αυτά τα γραφήματα τροφοδοτούνται από τις απλές αριθμητικές μετρήσεις από το Pod. Το Grafana δεν είναι εξαιρετικά φιλικό προς το χρήστη. Μπορείτε να κάνετε

κλικ στη γραμμή τίτλου οποιουδήποτε από τα γραφήματα και να επιλέξετε Edit για να δείτε το ερώτημα PromQL.

### – Εργαστηριακή άσκηση 23

Οι μετρήσεις εφαρμογών είναι μόνο ένα μέρος της παρακολούθησης - θα χρειαστεί επίσης να δείτε ένα χαμηλότερο επίπεδο λεπτομέρειας για να δείτε την απόδοση του cluster.

Το [cAdvisor](#) και το [kube-state-metrics](#) είναι εργαλεία ανοιχτού κώδικα για την εξαγωγή μετρήσεων δοχείων και cluster για το Prometheus. Θέλουμε να εκτελέσουμε αυτά τα στοιχεία μετρήσεων και να δούμε τα δεδομένα στο Grafana.

- Ξεκινήστε με την ανάπτυξη των διακομιστών μετρήσεων:

```
# save it in a folder called monitoring/specs/cluster-metrics
and name it cadvisor.yaml

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cadvisor
  namespace: kube-system
  labels:
    kubernetes.courselabs.co: monitoring
spec:
  selector:
    matchLabels:
      app: cadvisor
  template:
    metadata:
      labels:
        app: cadvisor
        kubernetes.courselabs.co: monitoring
    spec:
      containers:
        - name: cadvisor
          image: gcr.io/cadvisor/cadvisor:v0.45.0
          volumeMounts:
            - name: rootfs
              mountPath: /rootfs
              readOnly: true
            - name: var-run
              mountPath: /var/run
              readOnly: true
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: docker
              mountPath: /var/lib/docker
```

```

        readOnly: true
      - name: disk
        mountPath: /dev/disk
        readOnly: true
    ports:
      - name: metrics
        containerPort: 8080
        protocol: TCP
    automountServiceAccountToken: false
    volumes:
      - name: rootfs
        hostPath:
          path: /
      - name: var-run
        hostPath:
          path: /var/run
      - name: sys
        hostPath:
          path: /sys
      - name: docker
        hostPath:
          path: /var/lib/docker
      - name: disk
        hostPath:
          path: /dev/disk

# save it in a folder called monitoring/specs/cluster-metrics
# and name it kube-state-metrics.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    kubernetes.courselabs.co: monitoring
spec:
  selector:
    matchLabels:
      app: kube-state-metrics
  template:
    metadata:
      labels:
        app: kube-state-metrics
        kubernetes.courselabs.co: monitoring
    spec:
      serviceAccountName: kube-state-metrics
      containers:

```

```

      - image: k8s.gcr.io/kube-state-metrics/kube-state-
metrics:v2.2.0
      name: kube-state-metrics
      ports:
      - containerPort: 8080
        name: metrics
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-state-metrics
  namespace: kube-system
  labels:
    kubernetes.courselabs.co: monitoring
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kube-state-metrics
  labels:
    kubernetes.courselabs.co: monitoring
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - secrets
  - nodes
  - pods
  - services
  - resourcequotas
  - replicationcontrollers
  - limitranges
  - persistentvolumeclaims
  - persistentvolumes
  - namespaces
  - endpoints
  verbs:
  - list
  - watch
- apiGroups:
  - extensions
  resources:
  - daemonsets
  - deployments
  - replicaset
  - ingresses
  verbs:
  - list
  - watch

```

```
- apiGroups:
  - apps
  resources:
  - statefulsets
  - daemonsets
  - deployments
  - replicaset
  verbs:
  - list
  - watch
- apiGroups:
  - batch
  resources:
  - cronjobs
  - jobs
  verbs:
  - list
  - watch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - list
  - watch
- apiGroups:
  - authentication.k8s.io
  resources:
  - tokenreviews
  verbs:
  - create
- apiGroups:
  - authorization.k8s.io
  resources:
  - subjectaccessreviews
  verbs:
  - create
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  verbs:
  - list
  - watch
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - list
```

```

- watch
- apiGroups:
  - storage.k8s.io
  resources:
  - storageclasses
  - volumeattachments
  verbs:
  - list
  - watch
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - list
  - watch
- apiGroups:
  - networking.k8s.io
  resources:
  - networkpolicies
  verbs:
  - list
  - watch
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-state-metrics
  labels:
    kubernetes.courselabs.co: monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-state-metrics
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: kube-system

```

```
kubectl apply -f monitoring/specs/cluster-metrics
```



- Ελέγξτε τις ομάδες μετρήσεων:

```
kubectl get pods -l kubernetes.courselabs.co=monitoring -A
```

Δεν γίνεται scrape ακόμα από τον Προμηθέα επειδή δεν ταιριάζουν με τους κανόνες ανακάλυψης. Δεν θέλουμε να αλλάξουμε τον ορισμό του Pod επειδή είναι στάνταρ, αντίθετα θα επεκτείνουμε τη ρύθμιση του Prometheus. Για να φορτώσετε τα δεδομένα θα χρειαστείτε:

- Ενημερώστε τη διαμόρφωση ανακάλυψης υπηρεσίας Prometheus για να συμπεριλάβει αυτά τα νέα Pods.
- Φορτώστε ξανά τον διακομιστή Prometheus για να παραλάβετε τη νέα διαμόρφωση.
- Εισαγάγετε τον πίνακα εργαλείων cluster στο Grafana από το αρχείο monitoring/dashboards/cluster.json.

Θα πρέπει να δείτε στόχους όπως αυτός στο Prometheus:

Prometheus	Alerts	Graph	Status ▾	Help	Classic UI
------------	--------	-------	----------	------	------------

## Targets

All
Unhealthy
Collapse All

**app (1/1 up)**
show less

Endpoint	State	Labels
<a href="http://10.1.1.146:9110/metrics">http://10.1.1.146:9110/metrics</a>	UP	instance="fulfilment-processor-7d545b4d59-hjgt5" job="processor"

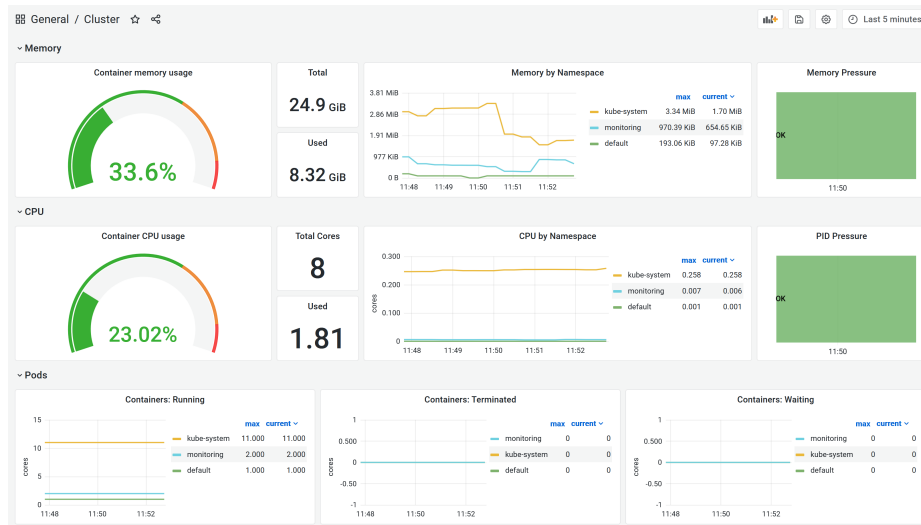
**cadvisor (1/1 up)**
show less

Endpoint	State	Labels
<a href="http://10.1.1.160:8080/metrics">http://10.1.1.160:8080/metrics</a>	UP	instance="10.1.1.160:8080" job="cadvisor"

**kube-state-metrics (1/1 up)**
show less

Endpoint	State	Labels
<a href="http://10.1.1.159:8080/metrics">http://10.1.1.159:8080/metrics</a>	UP	instance="10.1.1.159:8080" job="kube-state-metrics"

Και μετά θα δείτε κάτι σαν αυτό στο Grafana:



## – Καθάρισμα

```
kubectl delete ns,deploy,svc -l kubernetes.courselabs.co=monitoring

kubectl delete ds,deploy -n kube-system -l kubernetes.courselabs.co=monitoring

kubectl delete clusterrole,clusterrolebinding -l kubernetes.courselabs.co=monitoring
```

## Logging

### – Συγκεντρωτική καταγραφή με Elasticsearch, Fluentd και Kibana (EFK)

Είναι δύσκολο να εργαστείτε με logs Pod σε κλίμακα - το Kubectl δεν σας επιτρέπει να κάνετε αναζήτηση ή να φιλτράρετε logs. Η επιλογή ετοιμότητας παραγωγής είναι η εκτέλεση ενός κεντρικού υποσυστήματος καταγραφής, το οποίο συλλέγει όλα τα logs των Pod και τα αποθηκεύει σε μια κεντρική βάση δεδομένων. Το EFK είναι το συνηθισμένο σύστημα για να το κάνετε αυτό στο Kubernetes.

Αναφορές:

- [Kubernetes logging architecture](#)
- [Fluent Bit configuration for Kubernetes](#)

Το Fluent Bit είναι ένας βελτιωμένος συλλέκτης log που εξελίχθηκε από το Fluentd. Θα εκτελείται ως Pod σε κάθε κόμβο, συλλέγοντας τα logs των δοχείων των κόμβων. Το Fluent Bit χρησιμοποιεί μια διοχέτευση για την επεξεργασία logs. Αυτό

το μπλοκ εισόδου διαβάζει logs των δοχείων από τους κόμβους:

```
[INPUT]
Name          tail
Tag           kube.<namespace_name>.<container_name>.<
  pod_name>.<container_id>-
Tag_Regex     (?<pod_name>[a-z0-9] (?:[-a-z0-9]*[a-z0-9])?) (?<
  namespace_name>[^_]+) (?<container_name>.+)- (?<
  container_id>[a-z0-9]{64})/.log$
Path          /var/log/containers/*.log
```

- το tail διαβάζει αρχεία, παρακολουθώντας τα για νέο περιεχόμενο
- το path είναι όπου αποθηκεύονται τα logs
- το Tag\_Regex εξάγει μεταδεδομένα από το όνομα του log.

Αυτό το μπλοκ εξόδου αποθηκεύει κάθε log ως έγγραφο στο Elasticsearch:

```
[OUTPUT]
Name          es
Match         kube.default.*
Host          elasticsearch
Index         app-logs
Generate_ID   On
```

- το Match επιλέγει logs από τα Pods στο προεπιλεγμένο Namespace
- το Host είναι το όνομα DNS του διακομιστή Elasticsearch
- το Index είναι το όνομα του ευρετηρίου όπου δημιουργούνται τα έγγραφα.

#### – Εύρεση logs των Pod

Θα ξεκινήσουμε βλέποντας πώς το Kubernetes αποθηκεύει logs των δοχείων στον κόμβο όπου εκτελείται το Pod.

- Το fulfilment API είναι ένα απλό REST API που εγγράφει καταχωρήσεις log - δεν είναι τίποτα ιδιαίτερο:

```
# save it in a folder called logging/specs/fulfilment-api and
  name it deployment.yaml

# it runs a single Pod with no extra logging configuration

apiVersion: apps/v1
kind: Deployment
metadata:
  name: fulfilment-api
  labels:
    kubernetes.courselabs.co: logging
spec:
  selector:
```

```

matchLabels:
  app: fulfilment
  component: api
template:
  metadata:
    labels:
      app: fulfilment
      component: api
  spec:
    containers:
      - name: app
        image: courselabs/fulfilment-api
        ports:
          - containerPort: 80
            name: http

```

- Αναπτύξτε την εφαρμογή και ελέγξτε τα logs που εκτυπώνει κατά την εκκίνηση

```
kubectl apply -f logging/specs/fulfilment-api
```

```
kubectl logs -l app=fulfilment,component=api
```

Αυτή είναι μια εφαρμογή Java Spring Boot - θα δείτε ένα σύνολο logs εκκίνησης.

- Τα logs αποθηκεύονται στο σύστημα αρχείων του κόμβου που εκτελεί το Pod. Μπορεί να μην έχετε απευθείας πρόσβαση στο σύστημα αρχείων, αλλά μπορείτε να το αποκτήσετε χρησιμοποιώντας έναν τόμο:

```

# save it in a folder called logging/specs/jumpbox and name
  it pod.yaml

# it mounts the log paths from the node into the container
  filesystem

apiVersion: v1
kind: Pod
metadata:
  name: jumpbox
  labels:
    kubernetes.courselabs.co: logging
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: varlibdockercontainers

```

```

        mountPath: /var/lib/docker/containers
        readOnly: true
volumes:
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibdockercontainers
    hostPath:
      path: /var/lib/docker/containers

```

- Εκτελέστε το Pod και χρησιμοποιήστε το για να εξετάσετε τα logs στον κόμβο:

```

kubectl apply -f logging/specs/jumpbox

kubectl exec -it jumpbox -- sh

ls /var/log/containers

# use cat to read the contents of API log

exit

```

Κάθε δοχείο στον κόμβο έχει ένα αρχείο .log

Τα αρχεία ονομάζονται με ένα μοτίβο:

- fulfilment-processor-7695b895d7-h878q\_default\_app-260923b9ceb2c8223ebff38c2c3d81c2cd6301edfb3ae88c
- pod-name [namespace]\_[container\_name]\_[container-id]

Αυτό είναι το μοτίβο που θα χρησιμοποιήσει το Fluent Bit για να προσθέσει μεταδεδομένα σε κάθε log.

#### – Εκτελέστε τη στοίβα EFK

Το logging μια ανησυχία σε όλο το cluster και θα την εκτελέσουμε σε ξεχωριστό Namespace:

```

# save it in a folder called logging/specs/logging and name
  it fluentbit.yaml

# it runs Fluent Bit in a DaemonSet so there's a Pod on each
  node, mounting the log path volumes

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: logging

```

```

spec:
  selector:
    matchLabels:
      app: fluent-bit
  template:
    metadata:
      labels:
        app: fluent-bit
    spec:
      serviceAccountName: fluent-bit
      containers:
      - name: fluent-bit
        image: fluent/fluent-bit:1.8.3
        volumeMounts:
        - name: fluent-bit-config
          mountPath: /fluent-bit/etc/
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      volumes:
      - name: fluent-bit-config
        configMap:
          name: fluent-bit-config
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluent-bit
  namespace: logging
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluent-bit
rules:
- apiGroups: [""]
  resources:
  - namespaces
  - pods
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1

```

```

kind: ClusterRoleBinding
metadata:
  name: fluent-bit
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: fluent-bit
subjects:
- kind: ServiceAccount
  name: fluent-bit
  namespace: logging

# save it in a folder called logging/specs/logging and name
  it fluentbit-config.yaml

# it sets the Fluent Bit pipeline in a ConfigMap; logs are
  collected from two namespaces

apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  namespace: logging
data:
  fluent-bit.conf: |
    [SERVICE]
      Flush          5
      Log_Level      error
      Daemon         off
      Parsers_File   parsers.conf

    @INCLUDE input.conf
    @INCLUDE filter.conf
    @INCLUDE output.conf

  input.conf: |
    [INPUT]
      Name            tail
      Tag             kube.<namespace_name>.<
        container_name>.<pod_name>.<container_id>-
      Tag_Regex       (?<pod_name>[a-z0-9] (?:[-a-z0-9]*[a-
        -z0-9])?(?::\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)_(?<
        namespace_name>[^_]+)_(?<container_name>.+)-(?<
        container_id>[a-z0-9]{64})\.log$
      Path            /var/log/containers/*.log
      Parser           docker
      Refresh_Interval 10

  filter.conf: |
    [FILTER]

```

```

        Name      kubernetes
        Match      kube.*
        Kube_Tag_Prefix kube.
        Regex_Parser kube-tag

output.conf: |
[OUTPUT]
    Name      es
    Match      kube.default.*
    Host      elasticsearch
    Index      app-logs
    Generate_ID On

[OUTPUT]
    Name      es
    Match      kube.kube-system.*
    Host      elasticsearch
    Index      sys-logs
    Generate_ID On

parsers.conf: |
[PARSER]
    Name      docker
    Format      json
    Time_Key    time
    Time_Format %Y-%m-%dT%H:%M:%S.%L
    Time_Keep   On

[PARSER]
    Name      kube-tag
    Format      regex
    Regex      ^(?<namespace_name>[_]+)\.(?<container_name>
>.+)\.(?<pod_name>[a-z0-9](?:[-a-z0-9]*[a-z0-9])?(?:\[a-
z0-9]([-a-z0-9]*[a-z0-9])?)*)\.(?<container_id>[a-z0
-9]{64})-$

# save it in a folder called logging/specs/logging and name
# it elasticsearch.yaml

# it runs the latest OSS version of Elasticsearch

apiVersion: v1
kind: Service
metadata:
  name: elasticsearch
  namespace: logging
spec:
  selector:
    app: elasticsearch

```



```

    ports:
      - name: elasticsearch
        port: 9200
        targetPort: 9200
        type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
  namespace: logging
spec:
  selector:
    matchLabels:
      app: elasticsearch
  template:
    metadata:
      labels:
        app: elasticsearch
    spec:
      containers:
        - image: courselabs/elasticsearch:7.10
          name: elasticsearch
          ports:
            - containerPort: 9200
              name: elasticsearch

# save it in a folder called logging/specs/logging and name
# it kibana.yaml

# it runs the latest OSS version of Kibana

apiVersion: v1
kind: Service
metadata:
  name: kibana-lb
  namespace: logging
spec:
  selector:
    app: kibana
  ports:
    - name: kibana
      port: 5601
      targetPort: 5601
      type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: kibana-np

```

```

    namespace: logging
spec:
  selector:
    app: kibana
  ports:
    - name: kibana
      port: 5601
      targetPort: 5601
      nodePort: 30016
      type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana
  namespace: logging
spec:
  selector:
    matchLabels:
      app: kibana
  template:
    metadata:
      labels:
        app: kibana
    spec:
      containers:
        - image: courselabs/kibana:7.10
          name: kibana
          ports:
            - containerPort: 5601
              name: kibana

```

Σε ποια Namespaces θα συλλέγονται logs και σε ποιους δείκτες θα αποθηκευτούν τα logs;

- Υπάρχουν δύο μπλοκ εξόδου στο ConfigMap:

```

[OUTPUT]
  Name      es
  Match     kube.default.*
  Host      elasticsearch
  Index     app-logs
  Generate_ID On

[OUTPUT]
  Name      es
  Match     kube.kube-system.*
  Host      elasticsearch
  Index     sys-logs

```

Generate_ID	On
-------------	----

Το "Match" χρησιμοποιεί μεταδεδομένα ετικέτας που περιλαμβάνει το Namespace. Τα logs από το "default" Namespace θα αποθηκευτούν στο ευρετήριο "app-logs" και τα logs από το kube-system θα αποθηκευτούν στο ευρετήριο "sys-logs".

- Αναπτύξτε την εφαρμογή και περιμένετε να είναι έτοιμα τα Pods:

```
kubectl apply -f logging/specs/logging
kubectl get all -n logging
```

Το Elasticsearch χρησιμοποιεί ένα REST API στη θύρα 9200 για την εισαγωγή και αναζήτηση δεδομένων. Μπορούμε να το χρησιμοποιήσουμε από το jumpbox Pod.

- Δημιουργήστε ορισμένα logs εφαρμογών πραγματοποιώντας μια κλήση στο REST API εκκλήρωσης:

```
# you can use the NodePort address:
curl http://localhost:30018/documents

# or the LoadBalancer
curl http://localhost:8011/documents
```

Συνδεθείτε στο jumpbox Pod και κάντε ένα αίτημα HTTP με curl, στη διαδρομή /\_cat/indices στο Elasticsearch Pod.

- Πρώτη εκτέλεση σε shell στο Pod:

```
kubectl exec -it jumpbox -- sh
```

- Η εικόνα δοχείου έχει curl εγκατεστημένο - πρέπει να χρησιμοποιήσετε το πλήρως πιστοποιημένο όνομα τομέα για την Υπηρεσία Elasticsearch και τη θύρα:

```
curl http://elasticsearch.logging.svc.cluster.local:9200/_cat/indices

exit
```

Η έξοδος εμφανίζει μια λίστα δεικτών, η οποία περιλαμβάνει πού αποθηκεύονται τα logs:

```
yellow open app-logs 85auSZIAQ2SYpf1MN7NYGQ 5 1 12984 0
      3.4mb 3.4mb
yellow open sys-logs aKQA15XvQWac30upiwo71Q 5 1 106 0
      658.9kb 658.9kb
green open .kibana_1 bEezIodMQ_6FoJ8cQ5mP5A 1 0 0 0
      230b 230b
```

Μπορείτε να κάνετε τα πάντα με το REST API, αλλά το Kibana UI είναι πολύ πιο εύκολο στη χρήση.

#### – Προβολή logs εφαρμογών στο Kibana

Περιηγηθείτε στο Kibana στη διεύθυνση <http://localhost:5601> ή στο <http://localhost:30016>.

Από το αριστερό μενού:

- Κάντε κλικ στο Stack Management
- Μετά στο Index Patterns
- Κάντε κλικ στο Create index pattern, χρησιμοποιήστε το όνομα app-logs, και επιλέξτε το πεδίο ώρας @timestamp.

Τώρα από το αριστερό μενού:

- Κάντε κλικ στο Discover.

Μπορείτε να δείτε όλα τα logs των δοχείων, καθώς και τα μεταδεδομένα (Namespaces, pod, εικόνα κ.λπ.)

- Πραγματοποιήστε μια άλλη κλήση στο Spring Boot API:

```
curl localhost:30018/documents
```

Κάντε κλικ στην "Refresh" στο Kibana και θα δείτε ένα log που καταγράφει το αίτημα HTTP που μόλις υποβάλατε.

#### – Προσθήκη logs συστήματος στο Kibana

Τα μοτίβα ευρετηρίου στο Kibana χρησιμοποιούνται για την αναζήτηση δεδομένων στο Elasticsearch. Τα logs στοιχείων συστήματος αποθηκεύονται σε διαφορετικό ευρετήριο, επομένως χρειάζεστε ένα νέο μοτίβο ευρετηρίου.

Από το αριστερό μενού:

- Κάντε κλικ στο Stack Management
- Μετά στο Index Patterns
- Κάντε κλικ στο Create index pattern, χρησιμοποιήστε το όνομα "sys-logs", και επιλέξτε το πεδίο ώρας @timestamp.

Μεταβείτε στην καρτέλα Discover και επιλέξτε το νέο μοτίβο ευρετηρίου. Το Kibana είναι αρκετά φιλικό προς το χρήστη και αυτό είναι ένα καλό μέρος για να εξερευνήσετε τα logs.

Φιλτράρετε τις εγγραφές για εμφάνιση logs από τον διακομιστή API Kubernetes.

Κάντε κλικ στο πεδίο "kubernetes.labels.component" και θα δείτε όλες τις τιμές.

Κάντε κλικ στο "+" δίπλα στο "kube-apiserver" για να δείτε τα logs API.

Θα δείτε τα logs σχετικά με τις βασικές διεργασίες του συστήματος.

#### – Εργαστηριακή άσκηση 24

Αυτό είναι ένα γενικό σύστημα συλλογής logs το οποίο θα ανακτά logs από κάθε Pod - αλλά δεν δημιουργούν logs όλα τα Pods :

```
# save it in a folder called logging/specs/fulfilment-processor and name it deployment.yaml

# it runs a background processor which doesn't generate any logs

apiVersion: apps/v1
kind: Deployment
metadata:
  name: fulfilment-processor
  labels:
    kubernetes.courselabs.co: logging
spec:
  selector:
    matchLabels:
      app: fulfilment
      component: processor
  template:
    metadata:
      labels:
        app: fulfilment
        component: processor
    spec:
      containers:
        - name: app
          image: courselabs/fulfilment-processor
          env:
            - name: Observability__Logging__File
              value: 'true'
            - name: Observability__Logging__Console
              value: 'false'
            - name: Observability__Logging__LogFilePath
```

```
value: /app/logs/fulfilment-processor.log
```

- Ξεκινήστε εκτελώντας την εφαρμογή:

```
kubectl apply -f logging/specs/fulfilment-processor
```

- Ελέγξτε τα logs με το Kubectl και το Kibana - δεν υπάρχει κανένα για αυτό το νέο στοιχείο:

```
kubectl logs -l app=fulfilment,component=processor
```

- Η εφαρμογή γράφει logs σε ένα αρχείο στο σύστημα αρχείων του δοχείου:

```
kubectl exec deploy/fulfilment-processor -- cat /app/logs/fulfilment-processor.log
```

Ο στόχος σας είναι να επεκτείνετε τις προδιαγραφές Pod, ώστε αυτά τα logs να αφαιρούνται και να δημοσιεύονται ως logs από Pod.

#### – Καθάρισμα

```
kubectl delete ns,deploy,svc,po -l kubernetes.courselabs.co=logging
```

### 3.5 Continuous Integration και Continuous Deployment (CI/CD)

#### BuildKit

##### – Δημιουργία εικόνων Docker με το BuildKit

Το BuildKit είναι η μηχανή δημιουργίας εικόνων μέσα στο Docker. Δεν είναι μέρος άλλων runtime δοχείων - όπως το containerd - αλλά μπορείτε να εκτελέσετε έναν διακομιστή BuildKit σε ένα δοχείο.

Αυτό τροφοδοτεί μια διοχέτευση CI/CD που εκτελείται στο Kubernetes, όπου μπορείτε να έχετε ένα BuildKit Pod να δημιουργεί εικόνες από την πηγή. Η διαχείριση των builds γίνεται από έναν διακομιστή αυτοματισμού - όπως ο Jenkins - ο οποίος στέλνει εντολές στον διακομιστή BuildKit.

Αναφορές:

- [BuildKit on GitHub](#)
- [Running BuildKit in Kubernetes](#)
- [Storing registry credentials in Kubernetes Secrets](#)

## Helm

### – Συσκευασία και ανάπτυξη εφαρμογών με Helm

Το Helm προσθέτει μια γλώσσα προτύπου πάνω από το τυπικό Kubernetes YAML. Μετατρέπετε τις προδιαγραφές του αντικειμένου σας σε πρότυπα με μεταβλητές για τιμές που πρέπει να αλλάζουν μεταξύ εκδόσεων ή περιβαλλόντων - όπως η ετικέτα εικόνας που θα χρησιμοποιήσετε ή ο αριθμός των αντιγράφων. Το Helm έχει το δικό του CLI το οποίο χρησιμοποιείτε για την εγκατάσταση και την αναβάθμιση εφαρμογών, αλλά τα αναπτυγμένα αντικείμενα είναι τυπικοί πόροι Kubernetes που μπορείτε να διαχειριστείτε με το Kubectl.

Τα πακέτα εφαρμογών στο Helm ονομάζονται γραφήματα και μπορείτε να εγκαταστήσετε ένα γράφημα από έναν τοπικό φάκελο, ένα συμπιεσμένο αρχείο ή από ένα απομακρυσμένο χώρο αποθήκευσης γραφημάτων (παρόμοιο με το Docker Hub, αλλά για εφαρμογές). Τα γραφήματα περιέχουν απλώς τα πρότυπα YAML, ώστε να είναι μικρές λήψεις - οι εικόνες δοχείων εξακολουθούν να λαμβάνονται από το μητρώο εικόνων.

Αναφορές:

- [Helm documentation](#)
- [Helm CLI commands](#)
- [Chart structure and contents](#)
- [Template functions and pipelines](#)

## Rollouts

### – Ενημερώσεις με staged Rollouts

Οι ελεγκτές pod διαχειρίζονται τα Pod για εσάς - όταν ενημερώνετε την προδιαγραφή Pod, ο ελεγκτής εμφανίζει την αλλαγή αφαιρώντας τα παλιά Pod και δημιουργώντας νέα. Αυτό θα το κάνετε συνέχεια - κάθε ενημέρωση λειτουργικού συστήματος, ενημέρωση βιβλιοθήκης και νέα δυνατότητα θα είναι ενημέρωση. Ανάλογα με την εφαρμογή σας, οι αλλαγές διαμόρφωσης ενδέχεται να χρειάζονται επίσης rollout.

Μπορείτε να διαμορφώσετε τον ελεγκτή ώστε να τροποποιεί τον τρόπο με τον οποίο γίνεται η διάθεση - μπορείτε να επιλέξετε μια αργή αλλά ασφαλή ενημέρωση για ένα κρίσιμο στοιχείο. Τα Deployments είναι ο τυπικός ελεγκτής Pod, αλλά όλοι οι ελεγκτές έχουν επιλογές rollout.

Αναφορές:

- [Rollouts for Deployments](#)
- [Blue/green deployments in Kubernetes](#)
- [Helm upgrade command](#)

## Jenkins

### – CI/CD with Jenkins

Το Jenkins είναι ένας δημοφιλής και ισχυρός διακομιστής αυτοματισμού. Μπορείτε να εκτελέσετε το Jenkins στο Kubernetes cluster ως μέρος μιας υποδομής ανάπτυξης για να δημιουργήσετε και να προωθήσετε εικόνες Docker και να ενημερώσετε τις εφαρμογές που εκτελούνται στο cluster σας.

Αυτές οι ασκήσεις εκτελούν μια πλήρη εγκατάσταση τοπικής κατασκευής στο Kubernetes:

- [Gogs](#) - έναν διακομιστή Git για να φιλοξενήσει τον πηγαίο κώδικα και τις προδιαγραφές του έργου σας
- [BuildKit](#) - για τη δημιουργία εικόνων δοχείων από τα Dockerfiles
- [Jenkins](#) - για ανάκτηση κώδικα από το Gogs και δημιουργία εικόνων με το BuildKit.

Συνήθως δεν τα εκτελείτε όλα αυτά μόνοι σας, αλλά είναι πολύ χρήσιμο να δείτε πώς ταιριάζουν όλα μαζί και να το έχετε ως εφεδρική επιλογή εάν οι άλλες υπηρεσίες σας πέφτουν.

Αναφορές:

- [Using images from a private registry in Pods](#)
- [Getting started with Jenkins pipelines](#)
- [Using Declarative Jenkins Pipelines](#)

## GitOps

### – GitOps με ArgoCD

Το GitOps ανατρέπει την προσέγγιση CI/CD - αντί ένας εξωτερικός διακομιστής αυτοματισμού να σπρώχνει τις αλλαγές στο cluster, ένα εργαλείο μέσα στο cluster παρακολουθεί ένα αποθετήριο Git και πραγματοποιεί οποιεσδήποτε αλλαγές.

Σημαίνει ότι το cluster σας μπορεί να ασφαλιστεί πιο καθαρά, επειδή δεν χρειάζεστε χρήστη διαχειριστή cluster για τον διακομιστή αυτοματισμού. Σημαίνει επίσης ότι οι ορισμοί της εφαρμογής σας - ακόμα και η ρύθμιση της υποδομής για υπηρεσίες cloud - μπορούν να αποθηκευτούν οριστικά στα repos του Git. Μπορείτε να αναδημιουργήσετε ολόκληρη την ανάπτυξή σας από την αρχή.

Αναφορές:

- [GitOps](#) - περιγράφοντας την προσέγγιση της
- [ArgoCD](#) - το έργο GitOps στο CNCF
- [ArgoCD command line](#)
- [Application CRD spec](#)
- [GitOps with Kubernetes and Argo](#)



### 3.6 Advanced Kubernetes

#### NetworkPolicy

##### – Εξασφάλιση κίνησης με πολιτικές δικτύου

Η δικτύωση στο Kubernetes είναι επίπεδη - Τα Pods και οι Υπηρεσίες έχουν διευθύνσεις IP σε όλο το cluster και μπορούν να επικοινωνούν σε διαφορετικά Namespaces και κόμβους.

Αυτό διευκολύνει τη μοντελοποίηση κατανεμημένων εφαρμογών, αλλά σημαίνει ότι δεν μπορείτε να έχετε διαχωρισμένα δίκτυα εντός του cluster ή να σταματήσετε τις εφαρμογές σε Pods να φτάνουν εκτός του cluster.

Αυτά είναι κενά ασφαλείας που συμπληρώνει το API πολιτικής δικτύου, επιτρέποντάς σας να μοντελοποιήσετε την πρόσβαση στο δίκτυο ως μέρος της ανάπτυξης της εφαρμογής σας. Το API είναι απλό, αλλά το Kubernetes έχει ένα μοντέλο προσθήκης για το στοιχείο δικτύωσης του και δεν εφαρμόζουν πολιτικές όλες οι προσθήκες δικτύου.

- [NetworkPolicy API spec](#)
- [Network plugins \(CNI\)](#)
- [Common NetworkPolicy recipes](#)
- [Creating a k3d cluster with Calico CNI.](#)

##### – Αναπτύξτε την εφαρμογή APOD

Ας ξεκινήσουμε με την ανάπτυξη μιας απλής κατανεμημένης εφαρμογής. Τίποτα ιδιαίτερο στις προδιαγραφές - μόνο Deployments και υπηρεσίες:

```
# save it in a folder called networkpolicy/specs/apod and
  name it api.yaml

# a REST API which provides information about NASA's daily
  astronomy picture; used internally by the web app

apiVersion: v1
kind: Service
metadata:
  name: apod-api
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  ports:
    - port: 80
      targetPort: api
  selector:
    app: apod-api
    type: ClusterIP
---
```

```

kind: Deployment
metadata:
  name: apod-api
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  selector:
    matchLabels:
      app: apod-api
  template:
    metadata:
      labels:
        app: apod-api
    spec:
      containers:
        - name: api
          image: kiamol/ch14-image-of-the-day
          ports:
            - containerPort: 80
              name: api

# save it in a folder called networkpolicy/specs/apod and
# name it log.yaml

# a REST API which logs information about users of the app;
# used internally by the web app

apiVersion: v1
kind: Service
metadata:
  name: apod-log
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  ports:
    - port: 80
      targetPort: api
  selector:
    app: apod-log
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apod-log
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  selector:

```

```

    matchLabels:
      app: apod-log
  template:
    metadata:
      labels:
        app: apod-log
    spec:
      containers:
        - name: api
          image: kiamol/ch14-access-log
          ports:
            - containerPort: 80
              name: api

# save it in a folder called networkpolicy/specs/apod and
# name it web.yaml

# web application which consumes the REST APIs and shows the
# picture of the day; published with a NodePort Service

apiVersion: v1
kind: Service
metadata:
  name: apod-web
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  ports:
    - port: 8016
      targetPort: web
      nodePort: 30016
  selector:
    app: apod-web
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apod-web
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  selector:
    matchLabels:
      app: apod-web
  template:
    metadata:
      labels:
        app: apod-web

```

```
spec:
  containers:
    - name: web
      image: kiamol/ch14-image-gallery
      ports:
        - containerPort: 80
          name: web
```

- Δημιουργήστε τους πόρους:

```
kubectl apply -f networkpolicy/specs/apod
```

Περιμένετε να είναι έτοιμα όλα τα Pods και, στη συνέχεια, περιηγηθείτε στο <http://localhost:30016>, θα δείτε την εφαρμογή που λειτουργεί.

- Τώρα θα επιβάλουμε μια deny-all πολιτική δικτύου:

```
# save it in a folder called networkpolicy/specs/deny-all and
# name it default-deny.yaml

# the selector is empty so this will be enforced for all Pods

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```

Αυτό θα σταματήσει όλη την κίνηση δικτύου προς και από όλα τα Pods. Γιατί;

Οι κανόνες πολιτικής είναι πρόσθετοι - όπως το RBAC - επομένως τα θέματα ξεκινούν χωρίς δικαιώματα.

Δεν υπάρχουν δικαιώματα εισόδου ή εξόδου στους κανόνες, επομένως αυτή είναι μια πολιτική που δεν επιτρέπει τίποτα - αποκλείει αποτελεσματικά όλες τις εξερχόμενες και εισερχόμενες επικοινωνίες σε όλα τα Pods.

- Εφαρμόστε την πολιτική deny-all:

```
kubectl apply -f networkpolicy/specs/deny-all
```

```
kubectl get netpol
```

Αυτό θα πρέπει να αποκλείει την κίνηση, επομένως η εφαρμογή ιστού δεν μπορεί να επικοινωνήσει με τα API. Αλλά το Kubernetes cluster πιθανότατα δεν επιβάλλει την πολιτική δικτύου, επομένως η πολιτική δημιουργείται αλλά δεν εφαρμόζεται.

Ανανεώστε το <http://localhost:30016> την εφαρμογή (μάλλον) εξακολουθεί να λειτουργεί.

Θα μεταβούμε σε ένα νέο cluster και θα το δημιουργήσουμε με έναν πάροχο δικτύου που επιβάλλει την πολιτική.

- Καταργήστε την υπάρχουσα εφαρμογή για να ελευθερώσετε πόρους:

```
kubectl delete -f networkpolicy/specs/apod
```

- **\*\* Εάν χρησιμοποιείτε ήδη το K3d, σταματήστε το cluster σας για να μην έχουμε σύγκρουση θύρας:\*\***

```
k3d cluster stop k8s
```

#### – Install k3d CLI

Το **k3d** είναι ένα εργαλείο για τη δημιουργία τοπικών Kubernetes cluster, όπου κάθε κόμβος εκτελείται μέσα σε ένα δοχείο Docker. Δεν είναι τόσο φιλικό προς το χρήστη όσο το Docker Desktop, αλλά σας δίνει προηγμένες επιλογές για τη διαμόρφωση του cluster σας.

Χρησιμοποιήστε τις οδηγίες εγκατάστασης στο <https://k3d.io/v5.0.0/installation> Ή

- Ο απλός τρόπος, εάν έχετε εγκαταστήσει έναν διαχειριστή πακέτων:

```
# On Windows using Chocolatey:
```

```
choco install k3d
```

```
# On MacOS using brew:
```

```
brew install k3d
```

```
# On Linux:
```

```
curl -s https://raw.githubusercontent.com/rancher/k3d/main/
install.sh | bash
```

- Δοκιμάστε ότι το CLI λειτουργεί:

```
k3d version
```

Οι ασκήσεις χρησιμοποιούν k3d v5. Οι επιλογές έχουν αλλάξει πολύ σε σχέση με παλαιότερες εκδόσεις, οπότε αν είστε σε έκδοση 4 ή παλαιότερη έκδοση, θα χρειαστεί να κάνετε αναβάθμιση.

Το k3d απαιτεί Docker - Docker Engine σε Linux ή Docker Desktop σε Mac/Windows, επομένως δεν μπορείτε να το χρησιμοποιήσετε με οποιοδήποτε άλλο runtime δοχείων.

#### – Δοκιμάστε ένα νέο cluster με υποστήριξη NetworkPolicy

Μπορείτε να δημιουργήσετε απλά cluster με το k3d για να αναπαραστήσετε διαφορετικά περιβάλλοντα ή έργα και μετά να τα ξεκινήσετε και να τα σταματήσετε όταν χρειάζεται. Η δημιουργία cluster k3d έχει πολλές επιλογές.

Τα cluster k3d χρησιμοποιούν την προσθήκη Flannel CNI από προεπιλογή (όπως τα περισσότερα cluster), αλλά μπορείτε να διαμορφώσετε ένα νέο cluster χωρίς καμία προσθήκη δικτύου.

- Δημιουργήστε ένα νέο cluster χωρίς δικτύωση:

```
k3d cluster create labs-netpol -p "30000-30040:30000-30040
@server:0" --k3s-arg '--flannel-backend=none@server:0' --
k3s-arg '--disable=service@server:0' --k3s-arg '--
disable=traefik@server:0' --k3s-arg '--disable=metrics-
server@server:0'
```

- Αυτό δημιουργεί ένα cluster μονού κόμβου χωρίς εγκατεστημένο το Flannel CNI
- Οι θύρες δημοσιεύονται στο τοπικό μηχάνημα, ώστε να μπορείτε να χρησιμοποιήσετε το localhost με τις υπηρεσίες NodePort
- Οι επιπλέον λειτουργίες του k3d, όπως η υποστήριξη LoadBalancer και οι μετρήσεις, είναι απενεργοποιημένες

Το k3d θα αλλάξει το περιβάλλον Kubectl για να δείχνει στο νέο cluster

- Θα πρέπει να έχετε έναν κόμβο στο cluster - είναι στην πραγματικότητα ένα δοχείο Docker:

```
docker ps
```

Ελέγξτε την κατάσταση του νέου σας cluster. Είναι έτοιμο να τρέξει εφαρμογές τώρα;

- Το cluster δεν είναι έτοιμο:

```
kubectl get nodes
```

Ο κόμβος βρίσκεται στην κατάσταση NotReady, επειδή δεν υπάρχει εγκατεστημένο δίκτυο.

- Ψάξτε λίγο πιο βαθιά και θα δείτε ότι ο διακομιστής DNS δεν λειτουργεί:

```
kubectl get deploy -n kube-system
```

Το DNS απαιτεί πρόσθετο δίκτυο.

Το [Calico](#) είναι ένα πρόσθετο δικτύου που υποστηρίζει το NetworkPolicy. Είναι ανοιχτού κώδικα και χρησιμοποιείται πολύ συχνά όπου απαιτείται πολιτική δικτύου.

Το πρόσθετο δικτύου εκτελείται ως DaemonSet, αλλά χρησιμοποιεί επίσης προνομιακά δοχεία init για να τροποποιήσει τη διαμόρφωση δικτύου του λειτουργικού συστήματος στον κόμβο. Γι' αυτό χρησιμοποιούμε το k3d, ώστε να μην επηρεάζουμε τη δικτύωση στο κύριο cluster σας:

```
# save it in a folder called daemonsets/specs and name it
calico.yaml

# it is from the Calico docs, it includes the network plugin
and RBAC rules

https://kubernetes.courselabs.co/labs/networkpolicy/specs/k3d
/calico.yaml
```

- Εγκαταστήστε το Calico και περιμένετε να ετοιμαστούν τα Pods:

```
kubectl apply -f networkpolicy/specs/k3d
```

```
kubectl get pods -n kube-system --watch
```

Θα δείτε διάφορα Calico Pods να ξεκινούν στο kube-system Namespace.

- Είναι έτοιμο το σύμπλεγμα τώρα;

```
kubectl get nodes
```

Ο κόμβος σας θα πρέπει να είναι έτοιμος τώρα και το CoreDNS Deployment θα πρέπει να είναι σε πλήρη κλίμακα.

- Τώρα έχουμε ένα cluster επιβολής πολιτικής δικτύου, μπορούμε να δοκιμάσουμε ξανά την εφαρμογή APOD:

```
kubectl apply -f networkpolicy/specs/apod
```

Αυτό εκτελεί την ίδια εφαρμογή στο νέο cluster. Όταν εκτελούνται τα Pods, περιηγηθείτε στο <http://localhost:30016> και ελέγξτε ότι όλα λειτουργούν

- Εφαρμόστε την ίδια πολιτική deny-all:

```
kubectl apply -f networkpolicy/specs/deny-all
```

```
kubectl get netpol
```

Η Calico θα επιβάλει αυτήν την πολιτική. Ανανεώστε το <http://localhost:30016>, η εφαρμογή λήγει

- Δεν υπάρχει πολιτική εξόδου που να επιτρέπει την επικοινωνία από την εφαρμογή Ιστού στο API ή ακόμα και στον διακομιστή DNS.

```
# this will fail with a bad address message:
kubectl exec deploy/apod-web -- wget -O- http://apod-api/
image
```

Ελέγξτε ότι αυτό δεν είναι απλώς ένα ζήτημα DNS, βρίσκοντας τη διεύθυνση IP του API Pod και καλώντας το με ένα exec από το web Pod.

- Εκτυπώστε τη διεύθυνση IP με μια ευρεία λίστα Pod:

```
# this will fail with a bad address message:
kubectl exec deploy/apod-web -- wget -O- http://apod-api/
image
```

- Τώρα μπορείτε να χρησιμοποιήσετε τη διεύθυνση IP στην εντολή:

```
# this will fail with a timeout
kubectl exec deploy/apod-web -- wget -O- -T2 http://<pod-ip-
address>/image
```

#### – Ανάπτυξη πολιτικών για στοιχεία εφαρμογής

Θα δείτε συχνά μια προεπιλεγμένη πολιτική deny-all, για την αποφυγή τυχαίας επικοινωνίας δικτύου. Σε αυτήν την περίπτωση, πρέπει να μοντελοποιήσετε ρητά όλες τις γραμμές επικοινωνίας μεταξύ των στοιχείων σας:



```

# save it in a folder called networkpolicy/specs/apod/network
  -policies and name it apod-log.yaml

# it allows ingress from the web Pod to the API port; no
  egress as this component doesn't make any outgoing calls

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-log
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  podSelector:
    matchLabels:
      app: apod-log

  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: apod-web
        ports:
          - port: api

  egress: [] # deny all

# save it in a folder called networkpolicy/specs/apod/network
  -policies and name it apod-web.yaml

# it allows ingress from any location; egress to the two API
  Pods, and the DNS server (so the app can get the IP
  addresses of the API Pods)

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-web
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  podSelector:
    matchLabels:
      app: apod-web

  ingress:
    - {} # allow all

  egress:
    - to:

```

```

- podSelector:
  matchLabels:
    app: apod-api
- podSelector:
  matchLabels:
    app: apod-log
ports:
- port: api

- to:
- namespaceSelector:
  matchLabels:
    kubernetes.io/metadata.name: kube-system
  podSelector:
    matchLabels:
      k8s-app: kube-dns
  ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53

# save it in a folder called networkpolicy/specs/apod/network
# policies and name it apod-api.yaml

# it allows ingress from the web Pod; egress to the DNS
# server and to the IP address ranges where the 3rd-party
# API is hosted

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-api
  labels:
    kubernetes.courselabs.co: networkpolicy
spec:
  podSelector:
    matchLabels:
      app: apod-api

  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: apod-web
      ports:
        - port: api

  egress:
    - to: # DNS

```

```

- namespaceSelector:
  matchLabels:
    kubernetes.io/metadata.name: kube-system
  podSelector:
    matchLabels:
      k8s-app: kube-dns
  ports:
  - protocol: TCP
    port: 53
  - protocol: UDP
    port: 53

- to: # NASA APIs - api.nasa.gov
  - ipBlock:
      cidr: 15.200.69.242/24
  - ipBlock:
      cidr: 3.30.35.101/24
  ports:
  - protocol: TCP
    port: 443

```

Εάν θέλετε να περιορίσετε την πρόσβαση σε μπλοκ IP όπως αυτό, οι υπηρεσίες που χρησιμοποιείτε πρέπει να έχουν στατικές διευθύνσεις (μπορείτε να τις βρείτε χρησιμοποιώντας π.χ. `dig api.nasa.gov`)

- Εφαρμόστε τις νέες πολιτικές:

```

kubectl apply -f networkpolicy/specs/apod/network-policies
kubectl get netpol

```

- Επιβεβαιώστε ότι το web Pod μπορεί να χρησιμοποιήσει το API:

```

kubectl exec deploy/apod-web -- wget -O- -T2 http://apod-api/
image

```

- Το API ανακτά δεδομένα από τα API της NASA:

```

kubectl describe netpol apod-api

```

Ανανεώστε το `http://localhost:30016` και η εφαρμογή θα λειτουργήσει ξανά

### – Εργαστηριακή άσκηση 25

Έχουμε την εφαρμογή APOD που εκτελείται με ένα καλά ασφαλές δίκτυο, αλλά οι πολιτικές δεν ελέγχονται τόσο αυστηρά όσο θα μπορούσαν.

- Επιτρέπεται η κίνηση βάσει επιλογέων ετικετών και ένας κακόβουλος χρήστης θα μπορούσε να αναπτύξει ένα Pod με τις αναμενόμενες ετικέτες και να αποκτήσει πρόσβαση:

```
# save it in a folder called networkpolicy/specs/apod-hack
and name it sleep.yaml

# a basic sleep Pod with the apod-web label

apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    app: apod-web
    kubernetes.courselabs.co: networkpolicy
spec:
  containers:
  - name: sleep
    image: kiamol/ch03-sleep
```

- Αναπτύξτε αυτό το Pod και μπορείτε να το χρησιμοποιήσετε για να παρακάμψετε την ασφάλεια και να χρησιμοποιήσετε το API:

```
kubectl apply -f networkpolicy/specs/apod-hack

kubectl exec sleep -- wget -O- http://apod-api/image
```

Αυτό μπορεί να αποφευχθεί με την ανάπτυξη της εφαρμογής σε ένα προσαρμοσμένο Namespace, ο οποίος θα μπορούσε να ασφαλιστεί με RBAC. Δύο εργασίες για εσάς:

- αλλάξτε την εφαρμογή για να χρησιμοποιήσετε το Namespace του apod
- αλλάξτε τις πολιτικές δικτύου για να περιορίσετε την κίνηση εισόδου στα Pods από τα apod ns.

(Θα θέλετε να διαγράψετε την υπάρχουσα εφαρμογή APOD για αρχή).

### – Καθάρισμα

Εάν θέλετε να χρησιμοποιήσετε ξανά το k3d cluster, μπορείτε να διαγράψετε όλους τους πόρους άσκησης:

```
kubectl delete ns apod

kubectl delete all,netpol -A -l kubernetes.courselabs.co=
network-policy
```

Ή εάν θέλετε να διαγράψετε αυτό το cluster και να επιστρέψετε στο παλιό σας:

```
k3d cluster delete labs-netpol

kubectl config use-context docker-desktop # OR your old
cluster name

kubectl delete all,netpol -A -l kubernetes.courselabs.co=
network-policy
```

Και αν αρχικά χρησιμοποιούσατε ένα σύμπλεγμα K3d το οποίο σταματήσατε, θα πρέπει να το ξεκινήσετε ξανά:

```
k3d cluster start k8s
```

## Admission

### – Έλεγχος Εισόδου

Ο έλεγχος Εισόδου είναι η διαδικασία που επιτρέπει - ή αποκλείει - την εκτέλεση φόρτου εργασίας στο cluster. Μπορείτε να το χρησιμοποιήσετε για να επιβάλλετε τους δικούς σας κανόνες. Μπορεί να θέλετε να αποκλείσετε όλα τα δοχεία εκτός εάν χρησιμοποιούν μια εικόνα από ένα εγκεκριμένο μητρώο ή να αποκλείσετε τα Pods που δεν περιλαμβάνουν όρια πόρων στην προδιαγραφή.

Μπορείτε να το κάνετε αυτό με τα webhook ελεγκτών Εισόδου - διακομιστές HTTP που εκτελούνται μέσα στο cluster και καλούνται από το API για την εφαρμογή κανόνων κατά τη δημιουργία αντικειμένων. Οι ελεγκτές εισδοχής μπορούν να χρησιμοποιήσουν τη δική σας λογική ή μπορούν να χρησιμοποιήσουν ένα τυπικό εργαλείο όπως το [Open Policy Agent](#).

Αναφορές:

- [Using admission controllers](#)
- [Creating self-signed SSL certs for webhook servers](#)
- [OPA Gatekeeper](#)
- [Gatekeeper policy library](#)

### – Διακομιστές webhook σε cluster μέσω HTTPS

Τα webhook του ελεγκτή Εισόδου σας δίνουν τη μεγαλύτερη ευελιξία, επειδή μπορείτε να εκτελέσετε όποιον κώδικα θέλετε. Συνήθως θα τα εκτελείτε μέσα στο cluster

χρησιμοποιώντας τυπικά αντικείμενα Deployment και υπηρεσίας.

Ωστόσο, ο διακομιστής Kubernetes API θα καλεί webhook μόνο εάν εξυπηρετούνται μέσω HTTPS χρησιμοποιώντας ένα αξιόπιστο πιστοποιητικό. Ένας ωραίος τρόπος για να το κάνετε αυτό είναι με το [cert-manager](#), ένα έργο CNCF που δημιουργεί πιστοποιητικά TLS και τα δημιουργεί ως μυστικά.

```
# save it in a folder called admission/specs/cert-manager and
  name it 1.5.3.yaml

# it is from the cert-manager docs, it includes custom
  resources to create Certificates and all the RBAC,
  Services and Deployments to run the manager. It's complex
  - but definitely better than managing certs yourself

https://kubernetes.courselabs.co/labs/admission/specs/cert-
  manager/1.5.3.yaml
```

- Ανάπτυξη διαχειριστή πιστοποιητικών:

```
kubectl apply -f admission/specs/cert-manager
```

Ο διαχειριστής πιστοποιητικών χρησιμοποιεί τους εκδότες για να καθορίσει πώς δημιουργούνται τα πιστοποιητικά. Μπορείτε να το διαμορφώσετε έτσι ώστε να χρησιμοποιείτε έναν πραγματικό πάροχο πιστοποιητικών - π.χ. Ας κρυπτογραφήσουμε - αλλά θα χρησιμοποιήσουμε έναν αυτο-υπογεγραμμένο εκδότη:

```
# save it in a folder called admission/specs/cert-manager/
  issuers and name it self-signed.yaml

# it creates the issuer; this is a custom resource but this
  spec doesn't need any special config

apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned
spec:
  selfSigned: {}
```

Δημιουργήστε τον εκδότη και εκτυπώστε τα στοιχεία.

- Είναι ένας προσαρμοσμένος πόρος, αλλά είναι όλα YAML στο Kubernetes:

```
kubectl apply -f admission/specs/cert-manager/issuers
```

- Και εργάζεστε με αυτό με τον συνηθισμένο τρόπο:

```
kubectl get issuers
kubectl describe issuer selfsigned
```

Θα δείτε πολλά αποτελέσματα, συμπεριλαμβανομένης της κατάστασης που δείχνει ότι ο εκδότης είναι έτοιμος για χρήση.

Ο ελεγκτής αποδοχής μας είναι μια εφαρμογή ιστού NodeJS, κατασκευασμένη για να ταιριάζει με τις προδιαγραφές του webhook API ([πηγαίος κώδικας στο GitHub](#)):

```
# save it in a folder called admission/specs/webhook-server
and name it admission-webhook.yaml

# it defines a Deployment and Service. There's no RBAC or
special permissions, this is a standalone web server -
but it does run under HTTPS, expecting to find the TLS
cert in a Secret

apiVersion: v1
kind: Service
metadata:
  name: admission-webhook
  labels:
    kubernetes.courselabs.co: admission
spec:
  type: ClusterIP
  ports:
    - port: 443
      targetPort: https
  selector:
    app: admission-webhook
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: admission-webhook
  labels:
    kubernetes.courselabs.co: admission
spec:
  selector:
    matchLabels:
      app: admission-webhook
  template:
    metadata:
      labels:
        app: admission-webhook
    spec:
      containers:
```

```

- name: admission-webhook
  image: kiamol/ch16-admission-webhook
  env:
    - name: USE_HTTPS
      value: "true"
    - name: PORT
      value: "8443"
  ports:
    - name: https
      containerPort: 8443
  volumeMounts:
    - name: certs
      mountPath: /run/secrets/tls
      readOnly: true
  volumes:
    - name: certs
      secret:
        secretName: admission-webhook-cert

```

Αναπτύξτε τον διακομιστή webhook και επιβεβαιώστε ότι δημιουργείται το πιστοποιητικό Secret.

- Οι προδιαγραφές βρίσκονται στο φάκελο `webhook-server`:

```
kubectl apply -f admission/specs/webhook-server
```

- Ελέγξτε τα αντικείμενα πιστοποιητικού:

```
kubectl get certificates
```

- Θα πρέπει να δείτε ότι το πιστοποιητικό είναι Ready και η έξοδος δείχνει το Secret name όπου είναι αποθηκευμένο:

```
kubectl describe secret admission-webhook-cert
```

Το Secret περιέχει το πιστοποιητικό και το κλειδί TLS και το πιστοποιητικό CA για τον εκδότη.

- Αυτός είναι απλώς ένας τυπικός διακομιστής ιστού, ώστε να μπορούμε να δοκιμάσουμε τη ρύθμιση HTTPS εκτελώντας ένα Sleep Pod:

```

kubectl apply -f admission/specs/sleep

# you'll get a security error here:
kubectl exec sleep -- curl https://admission-webhook.default.svc

```



Το σφάλμα σημαίνει ότι το πιστοποιητικό έχει εφαρμοστεί, αλλά το curl δεν εμπιστεύεται τον εκδότη

#### – Επικύρωση Webhooks

Ο ελεγκτής εισόδου λειτουργεί, αλλά δεν κάνει τίποτα ακόμα. Πρέπει να διαμορφωθεί ως webhook για να καλεί ο διακομιστής Kubernetes API:

```
# save it in a folder called admission/specs/validating-
# webhook and name it validatingWebhookConfiguration.yaml

# it configures Kubernetes to call the webhook on the /
# validate path when Pods are created or updated; the
# annotation is there to configure the self-signed cert as
# trusted.

apiVersion: v1
kind: Service
metadata:
  name: admission-webhook
  labels:
    kubernetes.courselabs.co: admission
spec:
  type: ClusterIP
  ports:
    - port: 443
      targetPort: https
  selector:
    app: admission-webhook
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: admission-webhook
  labels:
    kubernetes.courselabs.co: admission
spec:
  selector:
    matchLabels:
      app: admission-webhook
  template:
    metadata:
      labels:
        app: admission-webhook
    spec:
      containers:
        - name: admission-webhook
          image: kiamol/ch16-admission-webhook
          env:
            - name: USE_HTTPS
```

```

        value: "true"
      - name: PORT
        value: "8443"
    ports:
      - name: https
        containerPort: 8443
    volumeMounts:
      - name: certs
        mountPath: /run/secrets/tls
        readOnly: true
    volumes:
      - name: certs
        secret:
          secretName: admission-webhook-cert

```

Αυτό είναι ένα webhook επικύρωσης - η λογική στον διακομιστή θα αποκλείσει τη δημιουργία οποιωνδήποτε Pods, όπου η προδιαγραφή δεν ορίζει το πεδίο `auto-mountServiceAccountToken` σε `false`.

- Εφαρμόστε το webhook επικύρωσης:

```
kubectl apply -f admission/specs/validating-webhook
```

- Ελέγξτε τις λεπτομέρειες και θα δείτε ότι ο διαχειριστής πιστοποίησης έχει εφαρμόσει το πιστοποιητικό CA από το πιστοποιητικό που δημιούργησε:

```
kubectl describe validatingwebhookconfiguration
servicetokenpolicy
```

Τώρα το webhook εκτελείται, το Kubernetes δεν θα εκτελεί κανένα Pods που δεν πληροί τους κανόνες - όπως αυτό σε αυτήν την [Ανάπτυξη εφαρμογής whoami](#).

- Δημιουργήστε τα αντικείμενα της εφαρμογής:

```
kubectl apply -f admission/specs/whoami
```

Η εφαρμογή δεν θα εκτελεστεί. Διορθώστε το για να βρείτε το μήνυμα σφάλματος που δημιουργήθηκε από τον ελεγκτή αποδοχής.

- Ελέγξτε το Deployment:

```
kubectl get deploy whoami
kubectl describe deploy whoami
```

Θα πρέπει να υπάρχουν δύο Pods, αλλά κανένα δεν είναι έτοιμο. Τα συμβάντα δείχνουν ότι το ReplicaSet έχει δημιουργηθεί και κλιμακωθεί, επομένως δεν υπάρχουν σφάλματα εδώ.

- Ελέγξτε το RS:

```
kubectl describe rs -l app=whoami
```

Εδώ βλέπετε το μήνυμα από τον ελεγκτή αποδοχής: Error creating: admission webhook "servicetokenpolicy.courselabs.co" denied the request: automountServiceAccountToken must be set to false.

Αυτή η εφαρμογή δεν θα διορθωθεί από μόνη της - το ReplicaSet θα συνεχίσει να προσπαθεί να δημιουργήσει Pods και θα συνεχίσει να απορρίπτονται από τον ελεγκτή αποδοχής.

- Για να εκτελεστεί, πρέπει να αλλάξετε την προδιαγραφή Pod - μπορείτε να επεξεργαστείτε την Ανάπτυξη ή να εφαρμόσετε μια [νέα προδιαγραφή](#) που πληρεί τους κανόνες επικύρωσης:

```
kubectl apply -f admission/specs/whoami/fix
```

```
kubectl get po -l app=whoami --watch
```

Τώρα δημιουργούνται τα Pods.

Η επικύρωση των webhook είναι ένας ισχυρός τρόπος για να διασφαλίσετε ότι οι εφαρμογές σας πληρούν τις πολιτικές σας - όλα τα αντικείμενα μπορούν να στοχευθούν και ολόκληρη η προδιαγραφή αποστέλλεται στο webhook, ώστε να μπορείτε να το χρησιμοποιήσετε για κανόνες ασφάλειας, απόδοσης ή αξιοπιστίας.

#### – Μεταλλαγμένα Webhooks

Η επικύρωση των webhook είτε επιτρέπει τη δημιουργία ενός αντικειμένου είτε το μπλοκάρει. Ο άλλος τύπος ελέγχου εισόδου είναι η αθόρυβη επεξεργασία των προδιαγραφών εισερχόμενου αντικειμένου χρησιμοποιώντας ένα μεταλλαγμένο webhook.

- Ο διακομιστής webhook που εκτελούμε έχει επίσης λογική μετάλλαξης:

```
# save it in a folder called admission/specs/mutating-webhook
  and name it mutatingWebhookConfiguration.yaml

# it operates when Pods are created or updated, and calls the
  /mutate endpoint on the server.
```

```

apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: nonrootpolicy
  labels:
    kubernetes.courselabs.co: admission
  annotations:
    cert-manager.io/inject-ca-from: default/admission-webhook-cert
webhooks:
- name: nonrootpolicy.courselabs.co
  admissionReviewVersions:
    - v1
  sideEffects: None
  rules:
    - operations: [ "CREATE", "UPDATE" ]
      apiGroups: [ "" ]
      apiVersions: [ "v1" ]
      resources: [ "pods" ]
  clientConfig:
    service:
      name: admission-webhook
      namespace: default
      path: "/mutate"

```

- Αναπτύξτε το νέο webhook:

```

kubectl apply -f admission/specs/mutating-webhook
kubectl describe mutatingwebhookconfiguration nonrootpolicy

```

Δεν υπάρχουν πληροφορίες για το τι κάνει στην πραγματικότητα αυτή η πολιτική...

- Δοκιμάστε να εκτελέσετε μια άλλη εφαρμογή - χρησιμοποιώντας αυτήν την προδιαγραφή για τον ιστότοπο Pi:

```

# save it in a folder called admission/specs/pi and name it pi.yaml

apiVersion: v1
kind: Service
metadata:
  name: pi-internal
  labels:
    kubernetes.courselabs.co: admission
spec:

```

```

ports:
  - port: 80
    nodePort: 30031
    targetPort: http
    name: http
  selector:
    app: pi-web
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  labels:
    kubernetes.courselabs.co: admission
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
    spec:
      automountServiceAccountToken: false
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: pi-web
          ports:
            - name: http
              containerPort: 80

```

```
kubectl apply -f admission/specs/pi
```

Ούτε αυτή η εφαρμογή θα εκτελεστεί. Ελέγξτε τα αντικείμενα και τις προδιαγραφές για να προσπαθήσετε να μάθετε τι πήγε στραβά.

- Δείτε τα Pods:

```
kubectl get po -l app=pi-web
```

- Θα δείτε ότι η κατάσταση είναι CreateContainerConfigError. Ελέγξτε τις λεπτομέρειες του Pod:

```
kubectl describe po -l app=pi-web
```

Θα δείτε ένα μήνυμα σφάλματος στα συμβάντα: Error: container has runAsNon-Root and image will run as root.

Αυτό σημαίνει ότι η εικόνα του δοχείου χρησιμοποιεί τον χρήστη root από προεπιλογή, αλλά η προδιαγραφή Pod έχει οριστεί με περιβάλλον ασφαλείας, ώστε να μην εκτελούνται δοχεία ως root.

Η προδιαγραφή Pod στο Deployment δεν λέει τίποτα για χρήστες που δεν είναι root, αυτό έχει εφαρμοστεί από το μεταλλαγμένο webhook.

- Μπορείτε να εκτελέσετε την εφαρμογή εφαρμόζοντας αυτήν την ενημερωμένη προδιαγραφή:

```
# save it in a folder called admission/specs/pi/fix and name
it pi.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pi-web
  labels:
    kubernetes.courselabs.co: admission
spec:
  selector:
    matchLabels:
      app: pi-web
  template:
    metadata:
      labels:
        app: pi-web
    spec:
      automountServiceAccountToken: false
      securityContext:
        runAsUser: 65534
        runAsGroup: 3000
      containers:
        - image: kiamol/ch05-pi
          command: ["dotnet", "Pi.Web.dll", "-m", "web"]
          name: web
          env:
            - name: ASPNETCORE_URLS
              value: http://+:5001
          ports:
            - name: http
              containerPort: 5001
```

```
kubectl apply -f admission/specs/pi/fix
```

### – OPA Gatekeeper

Τα προσαρμοσμένα webhook έχουν δύο μειονεκτήματα: πρέπει να γράψετε τον κωδικό μόνοι σας, ο οποίος προσθέτει στην περιουσία συντήρησης και οι κανόνες τους δεν μπορούν να εντοπιστούν μέσω του cluster, επομένως θα χρειαστείτε εξωτερική τεκμηρίωση.

Το OPA Gatekeeper είναι μια εναλλακτική που εφαρμόζει τον έλεγχο Εισόδου χρησιμοποιώντας γενικές περιγραφές κανόνων (σε μια γλώσσα που ονομάζεται [Rego](#)).

- Θα αναπτύξουμε κανόνες αποδοχής με το Gatekeeper - πρώτα διαγράψτε όλα τα προσαρμοσμένα webhook (τα δικά μας και του διαχειριστή πιστοποίησης):

```
kubectl delete ns,all,ValidatingWebhookConfiguration,
MutatingWebhookConfiguration -l kubernetes.courselabs.co=
admission

kubectl delete crd,ValidatingWebhookConfiguration,
MutatingWebhookConfiguration -l app.kubernetes.io/
instance=cert-manager
```

- Το OPA Gatekeeper είναι ένα άλλο πολύπλοκο στοιχείο, όπου ανταλλάσσετε τα overhead διαχείρισής του με τα ζητήματα της λειτουργίας των δικών σας ελεγκτών:

```
# save it in a folder called admission/specs/opa-gatekeeper
and name it 3.5.yaml

# it deploys custom resources to describe admission rules,
RBAC for the controller and a Service and Deployment to
run it
```

- Αναπτύξτε το OPA:

```
kubectl apply -f admission/specs/opa-gatekeeper
```

Ποιους προσαρμοσμένους τύπους πόρων εγκαθιστά το Gatekeeper;

- Ελέγξτε τα CustomResourceDefinitions:

```
kubectl get crd
```

Θα δείτε μερικά - το κύριο με το οποίο δουλεύουμε είναι το ConstraintTemplate.

Υπάρχουν δύο μέρη για την εφαρμογή κανόνων με το Gatekeeper:

- Δημιουργήστε ένα `ConstraintTemplate` που ορίζει έναν γενικό περιορισμό (π.χ. δοχεία σε ένα δεδομένο `Namespace` μπορούν να χρησιμοποιούν μόνο ένα δεδομένο μητρώο εικόνας)
- Δημιουργήστε έναν περιορισμό από το πρότυπο (π.χ. τα δοχεία στο `Namespace` `apod` μπορούν να χρησιμοποιούν μόνο εικόνες από τα repos `courselabs` στο `Docker Hub`).
- Ο ορισμός του κανόνα γίνεται με τη γενική γλώσσα πολιτικής `Rego`:

```
# save it in a folder called admission/specs/opa-gatekeeper/
# templates and name it requiredLabels-template.yaml

# it defines a simple (!) template to require labels on an
# object

apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: requiredlabels
spec:
  crd:
    spec:
      names:
        kind: RequiredLabels
      validation:
        openAPIV3Schema:
          properties:
            labels:
              type: array
              items: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package requiredlabels

        violation[{"msg": msg, "details": {"missing_labels":
missing}}] {
          provided := {label | input.review.object.metadata.
labels[label]}
          required := {label | label := input.parameters.
labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [
missing])
        }

# save it in a folder called admission/specs/opa-gatekeeper/
# templates and name it resourceLimits-template.yaml
```



```

# it defines a more complex template requiring container
  objects to have resources set

apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: policyresourcelimits
spec:
  crd:
    spec:
      names:
        kind: PolicyResourceLimits
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package policyresourcelimits

        missing(obj, field) = true {
          not obj[field]
        }

        missing(obj, field) = true {
          obj[field] == ""
        }

        violation[{"msg": msg}] {
          general_violation[{"msg": msg, "field": "containers
" }]
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_ ]
          not container.resources
          msg := sprintf("container <%v> has no resource
limits", [container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_ ]
          not container.resources.limits
          msg := sprintf("container <%v> has no resource
limits", [container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_ ]
          missing(container.resources.limits, "cpu")
          msg := sprintf("container <%v> has no cpu limit", [
container.name])
        }

```

```

        general_violation[{"msg": msg, "field": field}] {
            container := input.review.object.spec[field][_]
            missing(container.resources.limits, "memory")
            msg := sprintf("container <%v> has no memory limit"
, [container.name])
        }

```

- Δημιουργήστε τα πρότυπα:

```
kubectl apply -f admission/specs/opa-gatekeeper/templates
```

- Ελέγξτε ξανά τους προσαρμοσμένους πόρους. Πώς πιστεύετε ότι το Gatekeeper αποθηκεύει περιορισμούς στο Kubernetes;

```
kubectl get crd
```

- Βλέπετε νέα CRD για τα πρότυπα περιορισμών:

```
policyresourcelimits.constraints.gatekeeper.sh
```

```
requiredlabels.constraints.gatekeeper.sh
```

Το Gatekeeper δημιουργεί ένα CRD για κάθε πρότυπο περιορισμών, οπότε κάθε περιορισμός γίνεται ένας πόρος Kubernetes.

- Εδώ είναι οι περιορισμοί που χρησιμοποιούν τα πρότυπα:

```

# save it in a folder called admission/specs/opa-gatekeeper/
# constraints and name it requiredLabels.yaml

# it requires app and version labels on Pods, and a
# kubernetes.courselabs.co label on namespaces

apiVersion: constraints.gatekeeper.sh/v1beta1
kind: RequiredLabels
metadata:
  name: requiredlabels-ns
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Namespace"]
    parameters:
      labels: ["kubernetes.courselabs.co"]
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: RequiredLabels

```

```

metadata:
  name: requiredlabels-pods
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    parameters:
      labels: ["app", "version"]

# save it in a folder called admission/specs/opa-gatekeeper/
# constraints and name it resourceLimits.yaml

# it requires resources to be specified for any Pods in the
# apod namespace

apiVersion: constraints.gatekeeper.sh/v1beta1
kind: PolicyResourceLimits
metadata:
  name: resource-limits
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - "apod"

```

- Εφαρμόστε τους περιορισμούς:

```
kubectl apply -f admission/specs/opa-gatekeeper/constraints
```

Εκτυπώστε τις λεπτομέρειες του απαιτούμενου περιορισμού Namespace ετικετών. Είναι σαφές τι επιβάλλει;

- Ο τύπος περιορισμού είναι CRD, ώστε να μπορείτε να απαριθμήσετε αντικείμενα με τον συνήθη τρόπο:

```
kubectl get requiredlabels
```

```
kubectl describe requiredlabels requiredlabels-ns
```

Θα δείτε όλες τις υπάρχουσες παραβιάσεις του κανόνα και θα πρέπει να είναι σαφές τι απαιτείται - την ετικέτα σε κάθε Namespace.

## – Εργαστηριακή άσκηση 26

Τώρα έχουμε το OPA Gatekeeper στη θέση του, μπορούμε να δούμε πώς λειτουργεί.

- Δοκιμάστε να αναπτύξετε την εφαρμογή APOD από τις προδιαγραφές αυτού του εργαστηρίου:

```
kubectl apply -f admission/specs/apod
```

Θα αποτύχει επειδή οι πόροι δεν πληρούν τους περιορισμούς που έχουμε. Η δουλειά σας είναι να διορθώσετε τις προδιαγραφές και να ξεκινήσετε την εφαρμογή - χωρίς να κάνετε αλλαγές στις πολιτικές :)

#### – Καθάρισμα

Καταργήστε όλα τους Namespace του εργαστηρίου:

```
kubectl delete ns -l kubernetes.courselabs.co=admission
```

Και τα CRD:

```
kubectl delete crd -l gatekeeper.sh/system
```

```
kubectl delete crd -l gatekeeper.sh/constraint
```

## Χειριστές

#### – Διαχείριση εφαρμογών με χειριστές

Ορισμένες εφαρμογές χρειάζονται πολλές γνώσεις - όχι μόνο η πολυπλοκότητα της μοντελοποίησης της εφαρμογής στο Kubernetes, αλλά οι συνεχείς εργασίες συντήρησης. Σκεφτείτε μια κρατική εφαρμογή όπου θέλετε να δημιουργήσετε αντίγραφα ασφαλείας δεδομένων. Η ανάπτυξη της εφαρμογής μοντελοποιείται σε πόρους Kubernetes και θα ήταν υπέροχο να μοντελοποιήσουμε και τις λειτουργίες στο Kubernetes.

Αυτό κάνει το [μοτίβο χειριστή](#). Είναι ένας χαλαρός ορισμός για μια προσέγγιση όπου εγκαθιστάτε μια εφαρμογή που επεκτείνει το Kubernetes. Έτσι, σε αυτήν την εφαρμογή κατάστασης, ο χειριστής σας θα αναπτύξει την εφαρμογή και θα δημιουργούσε επίσης έναν προσαρμοσμένο πόρο DataBackup στο cluster. Κάθε φορά που θέλετε να δημιουργήσετε αντίγραφο ασφαλείας, αναπτύσσετε ένα αντικείμενο αντιγράφου ασφαλείας, ο χειριστής βλέπει το αντικείμενο και εκτελεί όλες τις εργασίες δημιουργίας αντιγράφων ασφαλείας.

Αναφορές:

- [Custom Resources](#) - επέκταση του Kubernetes με τον δικό σας τύπο αντικειμένου
- [NATS Operator](#) - ένα δείγμα χειριστή για την ανάπτυξη ουρών μηνυμάτων
- [Bitpoke MySQL Operator](#) - χειριστής για τη διαχείριση βάσεων δεδομένων MySQL

### – Προσαρμοσμένοι πόροι

Οι χειριστές συνήθως εργάζονται προσθέτοντας CustomResourceDefinitions (CRD) στο cluster.

- Μπορείτε να ξεκινήσετε κάνοντας λήψη του πακέτου Java:

```
# save it in a folder called operators/specs/crd and name it
student-crd.yaml

# it defines a v1 Student resource, with email and company
fields; the rest of the spec tells Kubernetes to store
objects in its database, and print out specific fields
when objects are shown in Kubectl

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: students.kubernetes.courselabs.co
  labels:
    kubernetes.courselabs.co: operators
spec:
  group: kubernetes.courselabs.co
  scope: Namespaced
  names:
    plural: students
    singular: student
    kind: Student
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                email:
                  type: string
                company:
                  type: string
  additionalPrinterColumns:
    - name: Email
      type: string
      description: Email Address
      jsonPath: .spec.email
    - name: Company
      type: string
      description: Company
```

```
jsonPath: .spec.company
```

- Καταχωρίστε όλους τους προσαρμοσμένους πόρους στο cluster και εκτυπώστε τις λεπτομέρειες του νέου CRD Student.

```
kubectl get crd
```

```
kubectl describe crd students
```

Το CRD είναι απλώς το σχήμα αντικειμένου. Το Kubernetes αποθηκεύει αντικείμενα μόνο εάν κατανοεί τον τύπο, που είναι αυτό που περιγράφει το CRD.

- Τώρα το Kubernetes cluster κατανοεί τους πόρους Student, μπορείτε να τους ορίσετε με το YAML:

```
# save it in a folder called operators/specs/students and
  name it student-crd.yaml

# it describes a student who works at Microsoft

apiVersion: kubernetes.courselabs.co/v1
kind: Student
metadata:
  name: edwin
spec:
  email: edwin@microsoft.com
  company: Microsoft

# save it in a folder called operators/specs/students and
  name it priti.yaml

# a student who works at Google

apiVersion: kubernetes.courselabs.co/v1
kind: Student
metadata:
  name: priti
spec:
  email: priti@google.com
  company: Google
```

Δημιουργήστε όλους τους Students στο φάκελο operators/specs/students, απαριθμήστε τους και εκτυπώστε τις λεπτομέρειες για το Priti.

- Αυτά τα αντικείμενα είναι τυπικά YAML:

```
kubectl apply -f operators/specs/students
```

- Και οι πόροι μπορούν να προσπελαστούν χρησιμοποιώντας το όνομα CRD:

```
kubectl get students
```

```
kubectl describe student priti
```

Εάν προσπαθήσετε να εφαρμόσετε αυτό το YAML σε ένα cluster που δεν έχει εγκατεστημένο το Student CRD, θα λάβετε ένα σφάλμα.

Τα τυπικά ρήματα Kubectl (get, delete, describe) λειτουργούν για όλα τα αντικείμενα, συμπεριλαμβανομένων των προσαρμοσμένων πόρων.

### – Χειριστής NATS

Το ίδιο το CRD δεν κάνει τίποτα, απλώς σας επιτρέπει να αποθηκεύετε πόρους στο cluster. Οι χειριστές συνήθως εγκαθιστούν CRD και εκτελούν επίσης ελεγκτές στο cluster. Ένας ελεγκτής είναι απλώς μια εφαρμογή που εκτελείται σε ένα Pod που συνδέεται με το Kubernetes API και παρακολουθεί τη δημιουργία ή την ενημέρωση των CRD.

Όταν δημιουργείτε έναν νέο προσαρμοσμένο πόρο, ο ελεγκτής το βλέπει και αναλαμβάνει δράση - κάτι που θα μπορούσε να σημαίνει δημιουργία Deployment, Services και ConfigMaps ή εκτέλεση οποιουδήποτε προσαρμοσμένου κώδικα χρειάζεστε.

Το [NATS](#) είναι μια ουρά μηνυμάτων υψηλής απόδοσης που είναι πολύ δημοφιλής για ασύγχρονα μηνύματα σε κατανεμημένες εφαρμογές. Ο χειριστής NATS εκτελείται ως Deployment:

```
# save it in a folder called operators/specs/nats/operator
# and name it 10-deployment.yaml

# it runs a single operator Pod which installs some CRDs and
# runs the controller

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nats-operator
  # Change to the name of the namespace where to install NATS
  # Operator.
  # Alternatively, change to "nats-io" to perform a cluster-
  # scoped deployment in supported versions.
  namespace: default
spec:
  replicas: 1
  selector:
```

```

matchLabels:
  name: nats-operator
template:
  metadata:
    labels:
      name: nats-operator
  spec:
    serviceAccountName: nats-operator
    containers:
      - name: nats-operator
        image: courselabs/nats-operator:v0.8.3
        imagePullPolicy: IfNotPresent
        args:
          - nats-operator
          # Uncomment to perform a cluster-scoped deployment in
          # supported versions.
          #- --feature-gates=ClusterScoped=true
        ports:
          - name: readyz
            containerPort: 8080
        env:
          - name: MY_POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: MY_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
        readinessProbe:
          httpGet:
            path: /readyz
            port: readyz
            initialDelaySeconds: 15
            timeoutSeconds: 3

```

- Εγκαταστήστε τον χειριστή και δείτε προσεκτικά την έξοδο:

```
kubectl application -f operators/specs/nats/operator
```

#### Ο χειριστής εγκαθιστά ορισμένα αντικείμενα RBAC και το Deployment

- Καταχωρίστε τώρα τους προσαρμοσμένους τύπους πόρων στο cluster σας. Θα δείτε ορισμένους τύπους NATS - πώς δημιουργήθηκαν;

```
kubectl get crd
```



Εμφανίζει τον προσαρμοσμένο πόρο Student, καθώς και τους πόρους NatsCluster και NatsServiceRole.

Δεν υπάρχει YAML για αυτά τα CRD, επομένως ο ελεγκτής NATS που εκτελείται στο Pod πρέπει να τα έχει δημιουργήσει χρησιμοποιώντας το Kubernetes API στον κώδικα.

- Μπορείτε να επιβεβαιώσετε ότι η ρύθμιση RBAC δίνει στον ελεγκτή ServiceAccount άδεια να το κάνει:

```
kubectl auth can-i create crds --as system:serviceaccount:
  default:nats-operator
```

- Μπορούμε να χρησιμοποιήσουμε ένα αντικείμενο NatsCluster για να δημιουργήσουμε μια ουρά μηνυμάτων cluster, υψηλής διαθεσιμότητας για χρήση από εφαρμογές:

```
# save it in a folder called operators/specs/nats/operator
  and name it 10-deployment.yaml

# it defines a cluster with 3 NATS servers, running version
  2.5

apiVersion: nats.io/v1alpha2
kind: NatsCluster
metadata:
  name: msgq
spec:
  size: 3
  version: "2.5.0"
```

- Δημιουργήστε τον πόρο του συμπλέγματος:

```
kubectl application -f operators/specs/nats/cluster
```

Δημιουργείται ένα αντικείμενο.

Εκτυπώστε τις λεπτομέρειες της νέας ουράς μηνυμάτων και δείτε τα άλλα αντικείμενα που εκτελούνται στο προεπιλεγμένο Namespace. Τα logs χειριστή θα δείχνουν πώς δημιουργήθηκαν τα Pods.

- Η έξοδος από το CRD δεν δείχνει πολλά:

```
kubectl get natscluster -o wide
```

- Αλλά ο χειριστής έχει δημιουργήσει Pods και υπηρεσίες:

```
kubectl get all --show-labels
```

- Ελέγξτε τα logs και θα δείτε ότι ο χειριστής διαχειρίζεται τα Pods - δεν υπάρχει Deployment ή ReplicaSet για την ουρά μηνυμάτων Pods:

```
kubectl logs -l name=nats-operator
```

Ο χειριστής NATS είναι ασυνήθιστος επειδή λειτουργεί ως ελεγκτής Pod. Συνήθως οι χειριστές χτίζουν πάνω από πόρους Kubernetes, επομένως θα χρησιμοποιούσαν Deployments για να διαχειριστούν τα Pods.

- Εκτυπώστε τις λεπτομέρειες ενός από τα NATS Pods και θα δείτε ότι το διαχειρίζεται ο χειριστής:

```
kubectl describe po msgq-1
```

Θα δείτε "Controlled By: NatsCluster/msgq"

- Ο χειριστής NATS εξακολουθεί να παρέχει υψηλή διαθεσιμότητα. Διαγράψτε ένα από τα Pods της ουράς μηνυμάτων και επιβεβαιώστε ότι έχει ξαναδημιουργηθεί.

```
kubectl delete po msgq-2
```

```
kubectl get po -l app=nats
```

```
kubectl logs -l name=nats-operator
```

Θα δείτε να δημιουργείται ένα νέο Pod που ονομάζεται msgq-2 και τα logs του χειριστή δείχνουν ότι έρχεται στο διαδίκτυο.

Δεν υπάρχουν πολλά περισσότερα που μπορείτε να κάνετε με τον τελεστή NATS, επομένως θα δοκιμάσουμε έναν που έχει κάποιες περισσότερες δυνατότητες.

## – Χειριστής MySQL

Υπάρχει ένα διάγραμμα Helm για τον [χειριστή Presslabs MySQL](#):

```
# save it in a folder called operators/specs/mysql/operator/
# and name it values.yaml

# it defines the default values for the operator; there is a
# lot you can tweak here

apiVersion: nats.io/v1alpha2
kind: NatsCluster
```

```

metadata:
  name: msgq
spec:
  size: 3
  version: "2.5.0"

```

- Εγκαταστήστε τον χειριστή (θα χρειαστείτε εγκατεστημένο το [Helm CLI](#)):

```

helm install mysql-operator operators/specs/mysql/operator/

# the operator pod might restart and take a few minutes to be
# ready:
kubectl get po -l app.kubernetes.io/name=mysql-operator -w

```

Ποιους πόρους χρειάζεται να δημιουργήσετε για να αναπτύξετε ένα cluster βάσης δεδομένων MySQL χρησιμοποιώντας τον τελεστή;

Η έξοδος Helm σας δίνει ένα παράδειγμα του τι χρειάζεστε:

- ένα αντικείμενο MysqlCluster - αυτό είναι ένα CRD που έχει εγκαταστήσει ο χειριστής
- ένα secret που περιέχει τον κωδικό πρόσβασης χρήστη διαχειριστή για τη βάση δεδομένων
- Μπορείτε να δημιουργήσετε ένα αναπαραγόμενο cluster βάσεων δεδομένων χρησιμοποιώντας αυτές τις προδιαγραφές:

```

# save it in a folder called operators/specs/nats/operator
# and name it 01-secret.yaml

# the database password

apiVersion: v1
kind: Secret
metadata:
  name: db-password
type: Opaque
stringData:
  ROOT_PASSWORD: op3rtorlab$

# save it in a folder called operators/specs/nats/operator
# and name it db.yaml

# the cluster set to use two MySQL servers

apiVersion: mysql.presslabs.org/v1alpha1
kind: MysqlCluster
metadata:
  name: db

```

```
spec:
  replicas: 2
  secretName: db-password
  # specify MySQL version
  image: percona:5.7.35
```

- Δημιουργήστε τη βάση δεδομένων:

```
kubectl apply -f operators/specs/mysql/database
```

Τα Pods βάσης δεδομένων χρειάζονται λίγο χρόνο για να ξεκινήσουν. Τι ελεγκτή χρησιμοποιεί ο χειριστής και ποια είναι η διαμόρφωση του δοχείου στα Pods;

- Καταχωρίστε τα Pods και θα δείτε το db-mysql-0. Αυτό το όνομα θα πρέπει να υποδηλώνει ότι διαχειρίζεται ένα StatefulSet:

```
kubectl get statefulset
```

- Εκτυπώστε τις λεπτομέρειες του Pod και θα δείτε πολλά δοχεία:

```
kubectl describe po db-mysql-0
```

Η ρύθμιση του δοχείου είναι αρκετά περίπλοκη:

- δύο δοχεία init που μοιάζουν σαν να ρυθμίζουν το περιβάλλον της βάσης δεδομένων και τη διαμόρφωση της MySQL
- το κύριο δοχείο βάσης δεδομένων που εκτελεί τη MySQL
- τρία δοχεία sidecar που εξάγουν μετρήσεις βάσης δεδομένων και εκτελούν έλεγχο heartbeat μεταξύ των διακομιστών της
- Ελέγξτε τα logs του κύριου διακομιστή βάσης δεδομένων στο Pod 0:

```
kubectl logs db-mysql-0 -c mysql
```

Θα δείτε το "mysqld: ready for connections showing the database server is running successfully"

- Και τα logs του δευτερεύοντος διακομιστή βάσης δεδομένων στο Pod 1:

```
kubectl logs db-mysql-1 -c mysql
```

Θα δείτε "replication@db-mysql-0.mysql.default:3306', replication started showing the secondary is replicating data from the primary".

Ο χειριστής παρέχει μια ανάπτυξη της MySQL σε επίπεδο παραγωγής και δημιουργεί επίσης ένα CRD για τη δημιουργία αντιγράφων ασφαλείας βάσης δεδομένων και την αποστολή τους στο χώρο αποθήκευσης cloud.

### – Εργαστηριακή άσκηση 27

Θα χρησιμοποιήσουμε τους χειριστές για να εγκαταστήσουμε στοιχεία υποδομής για μια εφαρμογή demo.

- Ξεκινήστε διαγράφοντας την υπάρχουσα ουρά μηνυμάτων και τα clusters βάσης δεδομένων:

```
kubectl delete natscluster,mysqlcluster --all
```

Οι χειριστές παρακολουθούν τους πόρους που θα διαγραφούν και θα αφαιρέσουν όλα τα αντικείμενα που δημιούργησαν.

- Τώρα αναπτύξτε μια απλή εφαρμογή to-do list:

```
# save it in a folder called aoperators/specs/todo-list and
  name it todo-list-configMap.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: todo-list-config
  labels:
    kubernetes.courselabs.co: operators
data:
  config.json: |-
    {
      "Database" : {
        "Provider" : "MySQL"
      },
      "MessageQueue": {
        "Url": "nats://todo-list-queue:4222"
      }
    }
  }
```

```
# save it in a folder called aoperators/specs/todo-list and
  name it todo-list-secret.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: todo-list-secret
  labels:
    kubernetes.courselabs.co: operators
type: Opaque
stringData:
  secrets.json: |-
    {
      "ConnectionStrings": {
```

```

        "ToDoDb": "server=todo-db-mysql;database=todo;user=
        root;password=op3rtorlab$"
    }
}

# save it in a folder called aoperators/specs/todo-list and
# name it todo-save-handler.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-save-handler
  labels:
    kubernetes.courselabs.co: operators
spec:
  selector:
    matchLabels:
      app: todo-list
      component: save-handler
  template:
    metadata:
      labels:
        app: todo-list
        component: save-handler
    spec:
      containers:
        - name: web
          image: kiamol/ch20-todo-save-handler
          volumeMounts:
            - name: config
              mountPath: "/app/config"
              readOnly: true
            - name: secret
              mountPath: "/app/secrets"
              readOnly: true
      volumes:
        - name: config
          configMap:
            name: todo-list-config
            items:
              - key: config.json
                path: config.json
        - name: secret
          secret:
            secretName: todo-list-secret
            defaultMode: 0400
            items:
              - key: secrets.json
                path: secrets.json

```

```

# save it in a folder called aoperators/specs/todo-list and
  name it todo-web.yaml

apiVersion: v1
kind: Service
metadata:
  name: todo-web
  labels:
    kubernetes.courselabs.co: operators
spec:
  ports:
    - port: 8020
      targetPort: 80
      nodePort: 30028
  selector:
    app: todo-list
    component: web
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-web
  labels:
    kubernetes.courselabs.co: operators
spec:
  selector:
    matchLabels:
      app: todo-list
      component: web
  template:
    metadata:
      labels:
        app: todo-list
        component: web
    spec:
      containers:
        - name: web
          image: kiamol/ch20-todo-list
          volumeMounts:
            - name: config
              mountPath: "/app/config"
              readOnly: true
            - name: secret
              mountPath: "/app/secrets"
              readOnly: true
      volumes:
        - name: config
          configMap:
            name: todo-list-config

```

```

      items:
      - key: config.json
        path: config.json
- name: secret
  secret:
    secretName: todo-list-secret
    defaultMode: 0400
    items:
    - key: secrets.json
      path: secrets.json

```

```
kubectl apply -f operators/specs/todo-list
```

Η εφαρμογή διαθέτει έναν ιστότοπο που ακούει στο `http://localhost:30028`, ο οποίος δημοσιεύει μηνύματα σε μια ουρά όταν δημιουργείτε ένα νέο αντικείμενο εργασίας. Ένας χειριστής μηνυμάτων ακούει στην ίδια ουρά και δημιουργεί στοιχεία στη βάση δεδομένων.

Περιηγηθείτε στην εφαρμογή τώρα και θα δείτε ένα σφάλμα - τα στοιχεία που χρειάζεται δεν υπάρχουν ακόμα.

Θα χρειαστεί να δημιουργήσετε αντικείμενα `NatsCluster` και `MysqlCluster` που να ταιριάζουν με τις ρυθμίσεις παραμέτρων στην εφαρμογή για να λειτουργούν όλα σωστά.

### – Καθάρισμα

Διαγράψτε τα βασικά αντικείμενα και τα CRD:

```

kubectl delete crd natsclusters.nats.io natsserviceroles.nats
.io
kubectl delete all,cm,secret,crd -l kubernetes.courselabs.co=
operators

```

Η σειρά είναι σημαντική, η διαγραφή CRD διαγράφει προσαρμοσμένους πόρους - βεβαιωθείτε ότι ο ελεγκτής εξακολουθεί να υπάρχει για να τακτοποιήσετε

- Διαγράψτε τον τελεστή NATS:

```
kubectl delete -f operators/specs/nats/operator
```

- Διαγράψτε το MySQL CRD και τον τελεστή:



```
kubect1 delete crd -l app.kubernetes.io/name=mysql-operator
helm uninstall mysql-operator
```

### 3.7 Λειτουργίες Παραγωγής

#### Clusters

##### – Kubernetes Clusters

Ένα cluster ενός κόμβου είναι καλό για ένα τοπικό περιβάλλον, αλλά ένα πραγματικό cluster θα είναι πάντα πολυκόμβος για υψηλή διαθεσιμότητα και κλίμακα.

Σε ένα cluster παραγωγής, τα βασικά στοιχεία Kubernetes - που ονομάζεται επίπεδο ελέγχου - λειτουργούν σε αποκλειστικούς κόμβους. Δεν θα εκτελέσετε κανένα από τα στοιχεία της εφαρμογής σας σε αυτούς τους κόμβους, επομένως είναι αφιερωμένοι στο Kubernetes. Το επίπεδο ελέγχου αναπαράγεται συνήθως σε τρεις κόμβους. Στη συνέχεια, έχετε όσους κόμβους εργαζομένων χρειάζεστε για να εκτελέσετε τις εφαρμογές σας, οι οποίοι μπορεί να είναι δεκάδες ή εκατοντάδες.

Μπορείτε να εγκαταστήσετε το Kubernetes σε διακομιστές ή εικονικά μηχανήματα χρησιμοποιώντας το εργαλείο [Kubeadm](#). Στο cloud θα χρησιμοποιούσατε τη γραμμή εντολών ή τη διεπαφή ιστού για την πλατφόρμα σας (π.χ. το [az aks create](#) για το Azure, το [eksctl create cluster](#) για το AWS και το [gcloud container clusters create](#) για το GCP). Θα χρησιμοποιήσουμε το k3d για να δημιουργήσουμε τοπικά περιβάλλοντα πολλών κόμβων.

Αναφορές:

- [kubeadm init](#) - για να αρχικοποιήσετε ένα νέο cluster
- [kubeadm join](#) - για να ενώσετε έναν νέο κόμβο σε ένα υπάρχον cluster
- [k3d cluster create](#) - δημιουργία τοπικών συμπλεγμάτων με το k3d
- [Taints and tolerations](#) - κόμβοι σήμανσης
- [Deprecated API Migration Guide](#) - διαχείριση αναβαθμίσεων εκδόσεων API

#### Affinity

##### – Προγραμματισμός με Pod και Node Affinity

Το Affinity είναι μια δυνατότητα όπου μπορείτε να ζητήσετε να προγραμματιστούν τα Pods σε σχέση με άλλα Pods ή κόμβους. Μπορεί να θέλετε να εκτελέσετε πολλά Pods σε ένα Deployment και να εκτελούνται όλα σε διαφορετικούς κόμβους ή μπορεί να έχετε μια εφαρμογή Ιστού όπου θέλετε τα Web Pods να εκτελούνται στον ίδιο κόμβο με τα API Pods.

Προσθέτετε κανόνες συνάφειας στην προδιαγραφή Pod σας. Αυτοί μπορεί να είναι κανόνες required, πράγμα που σημαίνει ότι λειτουργούν ως περιορισμοί και εάν δεν μπορούν να εκπληρωθούν, τότε το Pod παραμένει σε κατάσταση εκκρεμότητας. Ή

μπορεί να είναι κανόνες *preferred*, που σημαίνει ότι το Kubernetes θα προσπαθήσει να τους τηρήσει, αλλά αν δεν τα καταφέρει θα προγραμματίσει τα Pods ούτως ή άλλως.

- [Affinity and anti-affinity](#) - εφαρμογή σε κόμβους και Pods
- [Affinity API spec](#) - μέρος της προδιαγραφής Pod
- [Standard node labels & taints](#) - που μπορείτε να χρησιμοποιήσετε για τη συγγένεια κόμβων

## Tools

## 4 Λύσεις εργαστηριακών ασκήσεων

### 4.1 Εργαστηριακή άσκηση 1

Αναζητήστε `java` στο Docker Hub και η επίσημη εικόνα του Java είναι το πρώτο αποτέλεσμα της αναζήτησης.

- Μπορείτε να ξεκινήσετε κάνοντας λήψη του πακέτου Java:

```
docker pull java
```

- Εκτελέστε ένα δοχείο για να ελέγξετε την έκδοση Java:

```
docker run java java -version
```

- Και τέλος το build JRE για το Alpine είναι η μικρότερη εικόνα:

```
docker pull java:8-jre-alpine
```

```
docker image ls java
```

Η επίσημη σελίδα Java στο Docker Hub στην πραγματικότητα σε ανακατευθύνει στο πακέτο OpenJDK, το οποίο είναι περιβάλλον εκτέλεσης Java ανοιχτού κώδικα.

- Αλλά το πακέτο `java` δεν είναι το ίδιο με το `openjdk` - και θα πρέπει να χρησιμοποιήσετε το `openjdk` επειδή είναι πιο ενημερωμένο:

```
docker pull openjdk
```

```
docker pull openjdk:8-jre-alpine
```

Η προεπιλεγμένη ετικέτα εικόνας είναι πολύ πιο πρόσφατη. Επίσης και οι δύο εικόνες είναι μικρότερες:

```
docker image ls openjdk
```

## 4.2 Εργαστηριακή άσκηση 2

- Μπορείτε να εκτελέσετε μια εντολή μέσα στο δοχείο για να παρουσιαστεί μία λίστα αρχείων:

```
docker exec tls ls /certs
```

Η εντολή "docker cp" αντιγράφει αρχεία μεταξύ δοχείου και του τοπικού συστήματος αρχείων.

Για να αντιγράψετε από το δοχείο που ονομάζεται tls, χρησιμοποιήστε:

```
docker cp tls:/certs/server-cert.pem .
```

```
docker cp tls:/certs/server-key.pem .
```

Μπορείτε επίσης να αντιγράψετε από το τοπικό μηχάνημα στο δοχείο, αλλά η διαδρομή προορισμού πρέπει να υπάρχει:

```
docker exec tls mkdir /certs-backup
```

```
docker cp server-cert.pem tls:/certs-backup/
```

```
docker cp server-key.pem tls:/certs-backup/
```

```
docker exec tls ls /certs-backup
```

## 4.3 Εργαστηριακή άσκηση 3

- Μία λύση:

```
FROM openjdk:8-jre-alpine
# or :11-jre-slim

WORKDIR /app
COPY java/HelloWorld.class .

CMD java HelloWorld
```

Απλώς δημιουργεί έναν κατάλογο εργασίας που ονομάζεται /app στο σύστημα αρχείων του δοχείου και αντιγράφει στο αρχείο κλάσης Java.

- Το αρχείο κλάσης και το Dockerfile βρίσκονται σε διαφορετικούς καταλόγους, επομένως πρέπει να χρησιμοποιήσετε ένα περιβάλλον όπου το Docker μπορεί να έχει πρόσβαση και στα δύο αρχεία:

```
| - Desktop    <- this is the context
|-- java      <- so Docker can get the class file from here
|-- Dockerfile <- and the Dockerfile from here
```

- Δημιουργήστε την εικόνα χρησιμοποιώντας αυτό το πλαίσιο και προσδιορίζοντας τη διαδρομή προς το αρχείο Docker (δημιουργήστε και τα δύο αρχεία στο Desktop):

```
docker build -t java-hello-world -f Dockerfile/Dockerfile .
```

- Εκτελέστε ένα δοχείο από την εικόνα:

```
| - Desktop    <- this is the context
|-- java      <- so Docker can get the class file from here
|-- Dockerfile <- and the Dockerfile from here
```

Η έξοδος θα πρέπει να λέει Hello, World.

#### 4.4 Εργαστηριακή άσκηση 4

Το προεπιλεγμένο μητρώο είναι το Docker Hub - χρησιμοποιώντας τον τομέα `docker.io` και η προεπιλεγμένη ετικέτα εικόνας είναι πιο πρόσφατη.

- Έτσι, το `kiamol/ch05-pi` είναι η σύντομη μορφή του `docker.io/kiamol/ch05-pi:latest`:

```
docker pull kiamol/ch05-pi
docker pull docker.io/kiamol/ch05-pi:latest
```

- Ελέγξτε τη λίστα εικόνων και θα δείτε μόνο μία - αυτά δεν είναι παραλλαγές, είναι απλώς διαφορετικές μορφές του ίδιου ονόματος:

Να είστε προσεκτικοί με τη χρήση των πιο πρόσφατων εικόνων - είναι ένα συγκεχυμένο όνομα επειδή μπορεί να μην είναι η πιο πρόσφατη έκδοση.

- Για άλλα μητρώα πρέπει να συμπεριλάβετε τον τομέα στην αναφορά - έτσι οι εικόνες στο MCR πρέπει να έχουν το πρόθεμα `mcr.microsoft.com/`:

```
docker pull mcr.microsoft.com/dotnet/runtime:6.0
```

#### 4.5 Εργαστηριακή άσκηση 4

Η ενδεικτική λύση είναι το παρακάτω αρχείο:

```
# save it in a folder called compose and name it compose.yml

services:
  rng-api:
    image: courselabs/rng-api:21.05
    environment:
      - Logging__LogLevel__Default=Debug
    ports:
      - "8089:80"
    networks:
      - app-net

  rng-web:
    image: courselabs/rng-web:21.05
    environment:
      - Logging__LogLevel__Default=Debug
      - RngApi__Url=http://rng-api/rng
    ports:
      - "8090:80"
    networks:
      - app-net

  nginx:
    image: nginx:alpine
    networks:
      - app-net
      - front-end

networks:
  app-net:
  front-end:
```

και προσθέτει:

- ένα δίκτυο που ονομάζεται front-end χωρίς επιλογές, επομένως θα δημιουργηθεί με τις προεπιλογές του Docker
- μια υπηρεσία που ονομάζεται nginx που χρησιμοποιεί την εικόνα Nginx και συνδέεται με τα δίκτυα front-end και app-net.
- Αναπτύξτε την ενημέρωση:

```
docker-compose -f ./labs/compose/rng/lab.yml up -d
```

Θα δείτε τη δημιουργία του νέου δικτύου και του δοχείου, αλλά τα δοχεία ιστού RNG και API θα παραμείνουν αμετάβλητα. Οι προδιαγραφές δεν έχουν αλλάξει για αυτές τις υπηρεσίες, επομένως τα δοχεία αντιστοιχούν στην επιθυμητή κατάσταση.

- Επιθεωρήστε το νέο δοχείο για να εμφανιστούν οι λεπτομέρειες του δικτύου:

```
docker inspect compose-nginx-1
```

Θα δείτε ότι έχει δύο διευθύνσεις IP στην ενότητα δικτύου στο τέλος της εξόδου. Αυτή είναι μια IP από κάθε δίκτυο - όπως ένα μηχάνημα με δύο κάρτες δικτύου.

- Δοκιμάστε τη συνδεσιμότητα μεταξύ του δοχείου Nginx και του δοχείου Ιστού:

```
docker exec compose-nginx-1 nslookup rng-web
```

Το νέο δοχείο μπορεί να ανιχνεύσει τις διευθύνσεις IP για το αρχικό δοχείο.

- Και από το δοχείο ιστού στο Nginx:

```
docker exec compose-rng-web-1 ping -c3 nginx
```

Τα παλιά δοχεία μπορούν να ανιχνεύσει το νέο.

#### 4.6 Εργαστηριακή άσκηση 6

Όλα έχουν να κάνουν με το γεγονός ότι πρέπει να τρέξετε τα πάντα σε έναν μόνο διακομιστή:

- όταν ο διακομιστής βγαίνει εκτός σύνδεσης - προγραμματισμένο ή μη - χάνετε όλες τις εφαρμογές σας
- θα ενημερώνετε τακτικά τις εικόνες του δοχείου σας (τουλάχιστον κάθε μήνα) για να λαμβάνετε ενημερώσεις λειτουργικού συστήματος και βιβλιοθήκης, καθώς και νέες δυνατότητες. Η ενημέρωση της εφαρμογής σας σημαίνει αντικατάσταση δοχείου, επομένως θα υπάρχει χρόνος διακοπής λειτουργίας, όσο συμβαίνει η ενημέρωση, επειδή δεν υπάρχουν άλλοι διακομιστές που να εκτελούν επιπλέον δοχεία
- το ίδιο και με το config, οποιεσδήποτε αλλαγές αναπτύσσονται με την αντικατάσταση των δοχείων, πράγμα που σημαίνει ότι η εφαρμογή σας μπορεί να είναι εκτός σύνδεσης - και μπορεί να χαλάσει όταν επανέλθει, εάν υπάρχει κάποιο λάθος διαμόρφωσης
- οι επιλογές κλιμάκωσης περιορίζονται στις θύρες CPU, μνήμης και δικτύου του μηχανήματος. Το πιο σημαντικό είναι η θύρα - μόνο ένα δοχείο μπορεί να ακούσει σε μια θύρα, επομένως δεν μπορείτε να εκτελέσετε πολλά αντίγραφα ενός δημόσιου στοιχείου.

## 4.7 Εργαστηριακή άσκηση 7

- Η εκτέλεση της εφαρμογής με μια προσαρμοσμένη θύρα απαιτεί απλώς να μεταβιβάσετε τα arguments στην εφαρμογή:

```
# run the app listening on port 5000 inside the container:
docker run -d -P --name whoami2 whoami -port 5000
```

- Μπορείτε να ελέγξετε τη θύρα που χρησιμοποιεί η εφαρμογή - αλλά αν προσπαθήσετε να την καλέσετε, δεν θα λάβετε απάντηση:

```
docker port whoami2
curl localhost:<port>
```

- Ελέγξτε τα logs και θα δείτε γιατί - το Docker κατευθύνει την κυκλοφορία στη θύρα 80, η οποία είναι η θύρα που εκθέτει η εικόνα. Αλλά η εφαρμογή δεν ακούει σε αυτήν τη θύρα:

```
docker logs whoami2
```

- Πρέπει να αντιστοιχίσετε ρητά τη θύρα εάν διαμορφώνετε την εφαρμογή ώστε να χρησιμοποιεί μια θύρα που δεν είναι εκτεθειμένη στην εικόνα:

```
docker run -d -p 8050:5000 --name whoami3 whoami -port 5000
curl localhost:8050
```

- Ή εάν θέλετε το Docker να ορίσει μια τυχαία θύρα host, καθορίστε μόνο τη θύρα προορισμού:

```
docker run -d -p :5000 --name whoami4 whoami -port 5000
docker port whoami4
curl localhost:<port>
```

#### 4.8 Εργαστηριακή άσκηση 8

- Επιστρέψτε στην προηγούμενη ανάπτυξη:

```
docker-compose -f networking/compose.yml up -d
```

- Και μπορείτε να αυξήσετε την κλίμακα με το Compose CLI:

```
docker-compose -f networking/compose.yml up -d --scale rng-api=3
```

Θα δείτε να δημιουργούνται δύο νέα δοχεία API.

- Ακολουθήστε τα logs από όλα τα κοντέινερ API:

```
docker-compose -f networking/compose.yml logs -f rng-api
```

Χρησιμοποιήστε την εφαρμογή στη διεύθυνση <http://localhost:8090> και θα δείτε διαφορετικά δοχεία API που δημιουργούν αριθμούς.

- Επιθεωρήστε τα δοχεία API και θα δείτε σύνολα σύνθεσης με την ίδια παραλλαγή δικτύου για καθένα από αυτά:

```
docker inspect networking-rng-api-1
```

- περιλαμβάνει το hostname του δοχείου και το όνομα της υπηρεσίας Compose στην ενότητα `.NetworkSettings.Networks`, π.χ.

```
"Aliases": [
  "rng-api",
  "b382ce0e8ffc"
]
```

Τυχόν δοχεία με το ίδιο ψευδώνυμο θα επιστραφούν στην απόκριση DNS για αυτό το όνομα τομέα.



#### 4.9 Εργαστηριακή άσκηση 9

- Ο ευκολότερος τρόπος για να δείτε ετικέτες για ένα αντικείμενο είναι με την επιλογή `show-labels` στην εντολή `get`:

```
kubectl get nodes --show-labels
```

- Για να δείτε συγκεκριμένες τιμές ετικετών, μπορείτε να εκτυπώσετε τις ετικέτες ως στήλες:

```
kubectl get nodes --label-columns kubernetes.io/arch,kubernetes.io/os
```

- `kubectl get node <your-node> -o jsonpath='.metadata.labels'`

```
kubectl get nodes --show-labels
```

- Ή μπορείτε να κάνετε ερώτημα για συγκεκριμένες τιμές με ένα πρότυπο Go:

```
# with Bash:
kubectl get node <your-node> -o go-template='${'

# or with PowerShell:
kubectl get node docker-desktop -o go-template=''
```

((JSONPath doesn't like the forward slash in the label key))

#### 4.10 Εργαστηριακή άσκηση 10

- Ή μπορείτε να κάνετε ερώτημα για συγκεκριμένες τιμές με ένα πρότυπο Go:

```
\begin{lstlisting}[language=bash]
# create a folder called pods/solution and save the YAML file
  under the name solution/lab.yaml

apiVersion: v1
kind: Pod
metadata:
  name: sleep-lab
spec:
  containers:
  - name: app
    image: courselabs/bad-sleep
```

- Αναπτύξτε το με τον συνηθισμένο τρόπο με το Kubectl:

```
kubectl apply -f pods/solution/lab.yaml
```

- Τώρα παρακολουθήστε την κατάσταση του Pod:

```
kubectl get pod sleep-lab --watch
```

- Μετά από περίπου 30 δευτερόλεπτα, η εφαρμογή στο δοχείο τελειώνει, οπότε το δοχείο τερματίζει - στη συνέχεια το Kubernetes επανεκκινεί το Pod. Θα δείτε μια νέα γραμμή στην έξοδο, με τον αριθμό επανεκκίνησης να έχει αυξηθεί σε 1:

```
kubectl get pod sleep-lab --watch
```

Τα pod επανεκκινούν δημιουργώντας ένα νέο δοχείο και όχι με επανεκκίνηση του υπάρχοντος δοχείου.

Το νέο δοχείο εκτελείται μέχρι να τερματίσει η εφαρμογή μετά από 30 δευτερόλεπτα. Το Kubernetes επανεκκινεί το Pod - αλλά εάν τα δοχεία του Pod συνεχίσουν να τερματίζουν, το Kubernetes προσθέτει μια αυξανόμενη καθυστέρηση πριν από την επανεκκίνηση.

Η κατάσταση αλλάζει σε Completed και στη συνέχεια σε Running, αλλά τελικά το Pod εισέρχεται σε κατάσταση CrashLoopBackOff:

NAME	READY	STATUS	RESTARTS	AGE
sleep-lab	1/1	Running	0	3s
sleep-lab	0/1	Completed	0	33s
sleep-lab	1/1	Running	1	35s
sleep-lab	0/1	Completed	1	64s
sleep-lab	0/1	CrashLoopBackOff	1	79s
sleep-lab	1/1	Running	2	80s
sleep-lab	0/1	Completed	2	110s
sleep-lab	0/1	CrashLoopBackOff	2	2m4s
sleep-lab	1/1	Running	3	2m17s

- Μπορείτε να διαγράψετε το Pod με τη χρήση του ονόματός του:

```
kubectl delete pod sleep-lab
```

- Ή χρησιμοποιώντας την εντολή delete με το αρχείο YAML σας:

```
kubectl delete -f pods/solution/lab.yaml
```

#### 4.11 Εργαστηριακή άσκηση 11

- Μπορείτε να παραθέσετε όλα τα Pods για μια Υπηρεσία χρησιμοποιώντας:

```
kubectl describe svc whoami

# OR

kubectl get endpoints whoami
```

Τα endpoints είναι αντικείμενα Kubernetes, αλλά συνήθως διαχειρίζονται από τις Υπηρεσίες και δεν τα δημιουργείτε μόνοι σας.

#### Υπηρεσίες χωρίς endpoints

- Μπορείτε να δημιουργήσετε μια Υπηρεσία χωρίς ταιριαστά Pods προσθέτοντας μια ετικέτα:

```
# save it in a folder called services/solution and name it
  whoami-svc-zero-matches.yaml

apiVersion: v1
kind: Service
metadata:
  name: whoami-zero-matches
  labels:
    kubernetes.courselabs.co: services
spec:
  selector:
    app: whoami
    version: v2 # no Pod has this label
  ports:
    - name: http
      port: 8010
      targetPort: 80
```

- Δεν υπάρχουν Pods που να ταιριάζουν επειδή το Whoami Pod δεν έχει ετικέτα έκδοσης:

```
kubectl apply -f services/solution/whoami-svc-zero-matches.yaml

kubectl get endpoints whoami-zero-matches

kubectl exec sleep -- nslookup whoami-zero-matches

kubectl exec sleep -- curl -v -m 5 http://whoami-zero-matches
```

Υπάρχει μια διεύθυνση IP για την Υπηρεσία, αλλά δεν υπάρχουν endpoints, επομένως κλήσης curl κάνει time out.

### Υπηρεσίες με πολλαπλά endpoints

- Πολλά Pods μπορούν να τρέξουν με τις ίδιες ετικέτες. Αναπτύξτε ένα δεύτερο Whoami Pod με τις ίδιες προδιαγραφές με το πρώτο - μόνο το όνομα πρέπει να αλλάξετε:

```
kubectl apply -f services/solution/whoami-pod-2.yaml
kubectl get po -o wide -l app=whoami
kubectl get endpoints whoami
```

Και οι δύο διευθύνσεις IP του Pod καταχωρούνται ως endpoints Υπηρεσίας.

```
kubectl exec sleep -- curl -v http://whoami
```

Η IP στην απόκριση είναι η Pod IP, η IP που ζητήθηκε είναι η Υπηρεσία. Επαναλάβετε την κλήση και το Pod IP στην απόκριση αλλάζει - οι υπηρεσίες εξισσοροπούν τα αιτήματα μεταξύ των Pods.

## 4.12 Εργαστηριακή άσκηση 12

Για blue-green ενημερώσεις χρειάζεστε δύο Deployments - κάθε ένα διαχειρίζεται Pods για μια διαφορετική έκδοση της εφαρμογής σας.

```
# save it in a folder called deployments/solution and name it
  solution/whoami-deployments.yaml

# it has two Deployments defined in the same YAML, so you can
  easily compare them.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami-lab-v1
  labels:
    kubernetes.courselabs.co: deployments
spec:
  replicas: 2
```

```

selector:
  matchLabels:
    app: whoami-lab
    version: v1
template:
  metadata:
    labels:
      app: whoami-lab
      version: v1
  spec:
    containers:
      - name: app
        image: sixeyed/whoami:21.04
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami-lab-v2
  labels:
    kubernetes.courselabs.co: deployments
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami-lab
      version: v2
  template:
    metadata:
      labels:
        app: whoami-lab
        version: v2
    spec:
      containers:
        - name: app
          image: sixeyed/whoami:21.04
          env:
            - name: WHOAMI_MODE
              value: q

```

– Το Kubectl υποστηρίζει και αυτό, χρησιμοποιώντας — για να διαχωρίσει αντικείμενα.

```

kubectl apply -f labs/deployments/solution/whoami-deployments.
yaml

kubectl get pods -l app=whoami-lab,version=v1

kubectl get pods -l app=whoami-lab,version=v2

```

Εκτελούνται τέσσερα Pods, αλλά δεν υπάρχουν υπηρεσίες που να στοχεύουν τις ετικέτες τους

### Αναπτύξτε την υπηρεσία v1

Ο blue-green διακόπτη γίνεται με την αλλαγή του επιλογέα ετικετών για την υπηρεσία.

```
# save it in a folder called deployments/solution and name it
solution/whoami-service-v1.yaml

# it has a LoadBalancer and NodePort Service defined - each uses
the same selector to pick the v1 Pods.

apiVersion: v1
kind: Service
metadata:
  name: whoami-lab-np
  labels:
    kubernetes.courselabs.co: deployments
spec:
  selector:
    app: whoami-lab
    version: v1
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30020
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: whoami-lab-lb
  labels:
    kubernetes.courselabs.co: deployments
spec:
  selector:
    app: whoami-lab
    version: v1
  ports:
    - name: http
      port: 8020
      targetPort: 80
    type: LoadBalancer
```

– Ανάπτυξη και τεστάρετε το v1:

```
kubectl apply -f deployments/solution/whoami-service-v1.yaml

kubectl get endpoints whoami-lab-np whoami-lab-lb

curl localhost:8020 # OR curl localhost:30020
```

### Μεταβείτε στο v2

```
# save it in a folder called deployments/solution and name it
  whoami-service-v1.yaml

# it has the same Service definitions with just a change to the
  selector.

apiVersion: v1
kind: Service
metadata:
  name: whoami-lab-np
  labels:
    kubernetes.courselabs.co: deployments
spec:
  selector:
    app: whoami-lab
    version: v2
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30020
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: whoami-lab-lb
  labels:
    kubernetes.courselabs.co: deployments
spec:
  selector:
    app: whoami-lab
    version: v2
  ports:
    - name: http
      port: 8020
      targetPort: 80
    type: LoadBalancer
```

- Το Kubernetes το αναπτύσσει ως ενημέρωση στις υπάρχουσες Υπηρεσίες, έτσι οι διευθύνσεις IP δεν αλλάζουν, αλλά μόνο τα endpoints που βρίσκουν οι Υπηρεσίες:

```
kubectl apply -f deployments/solution/whoami-service-v2.yaml
kubectl get endpoints whoami-lab-np whoami-lab-lb
curl localhost:8020 # OR curl localhost:30020
```

Μπορείτε να κάνετε εναλλαγή μεταξύ των deployments αλλάζοντας την προδιαγραφή υπηρεσίας.

### 4.13 Εργαστηριακή άσκηση 13

Χρησιμοποιείτε την εντολή "kubectl create" για να δημιουργήσετε επιτακτικά αντικείμενα. Μην το χρησιμοποιείτε για Pods ή Deployments - το YAML είναι μια πολύ καλύτερη επιλογή - αλλά μπορεί να λειτουργήσει καλά για ConfigMaps.

#### Δημιουργήστε ConfigMap από το literal

- Η πιο εύκολη επιλογή είναι να καθορίσετε το κλειδί και την τιμή για τις ρυθμίσεις μεταβλητής περιβάλλοντος ως literals:

```
kubectl apply -f deployments/solution/whoami-service-v2.yaml
kubectl get endpoints whoami-lab-np whoami-lab-lb
curl localhost:8020 # OR curl localhost:30020
```

#### Δημιουργήστε ConfigMap από αρχείο env

Εναλλακτικά, αποθηκεύστε τις τιμές σε ένα αρχείο .env - όπως το configurable.env. Αυτό δεν είναι το ίδιο με την αποθήκευση της διαμόρφωσης στο YAML, επειδή είναι η εγγενής μορφή και μπορεί να χρησιμοποιηθεί εκτός του Kubernetes.

```
Configurable__Release=21.04-secret
```

- Θα χρειαστεί να διαγράψετε το κυριολεκτικό ConfigMap για να το δοκιμάσετε - γι' αυτό η επιθυμητή κατάσταση στο YAML είναι καλύτερη επιλογή:

```
kubectl delete configmap configurable-env-lab
kubectl create configmap configurable-env-lab --from-env-file=
  labs/configmaps/solution/configurable.env
kubectl describe cm configurable-env-lab
```



### Δημιουργήστε ConfigMap από αρχείο config

Το `override.json` έχει τις απαιτούμενες ρυθμίσεις JSON. Το όνομα αρχείου είναι το ίδιο με το αναμενόμενο όνομα αρχείου που θα διαβάσει η εφαρμογή.

```
{
  "Features" : {
    "DarkMode" : true
  }
}
```

```
kubectl create configmap configurable-override-lab --from-file=
  labs/configmaps/solution/override.json
```

```
kubectl describe cm configurable-override-lab
```

### Αναπτύξτε την εφαρμογή

- Το `deployment-lab.yaml` αναμένει τα ίδια ονόματα ConfigMap που χρησιμοποιήσαμε, ώστε να μπορέσουμε να αναπτύξουμε:

```
kubectl apply -f configmaps/specs/configurable/lab/
```

Περιηγηθείτε στην Υπηρεσία σας και θα δείτε τις διαμορφωμένες ρυθμίσεις από τις αναμενόμενες πηγές.

## 4.14 Εργαστηριακή άσκηση 14

Ο τρόπος με τον οποίο διαθέτετε τις αλλαγές διαμόρφωσης εξαρτάται σε μεγάλο βαθμό από τη δομή του οργανισμού και τη ροή εργασίας σας.

Αυτή η λύση εμφανίζει δύο έγκυρες επιλογές, αλλά συχνά η διαδικασία αυτοματοποιείται με [Kustomize](https://helm.sh/) ή <https://helm.sh/>.

### Χρήση νέων ονομάτων για αντικείμενα διαμόρφωσης

Εάν θέλετε να ενεργοποιήσετε μια διάθεση ανάπτυξης κατά την αλλαγή των παραμέτρων, μπορείτε να δημιουργήσετε νέα αντικείμενα διαμόρφωσης με νέα ονόματα και να ενημερώσετε την προδιαγραφή Pod στο Deployment για να χρησιμοποιήσετε τα νέα αντικείμενα.

- Αναπτύξτε ένα σύνολο ρυθμίσεων `v1`:

```

# save it in a folder called secret/solution and name it
  deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable-lab
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
spec:
  selector:
    matchLabels:
      app: configurable-lab
  template:
    metadata:
      labels:
        app: configurable-lab
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          volumeMounts:
            - name: secret
              mountPath: /app/secrets
              readOnly: true
      volumes:
        - name: secret
          secret:
            secretName: configurable-lab-secret

# save it in a folder called secret/solution and name it secret.
  yaml

apiVersion: v1
kind: Secret
metadata:
  name: configurable-lab-secret
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
stringData:
  secret.json: |-
    {
      "Configurable__LabVersion": "v1"
    }

# save it in a folder called secret/solution and name it
  services.yaml

```

```

apiVersion: v1
kind: Service
metadata:
  name: configurable-lab-np
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
spec:
  selector:
    app: configurable-lab
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30020
    type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: configurable-lab-lb
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
spec:
  selector:
    app: configurable-lab
  ports:
    - name: http
      port: 8020
      targetPort: 80
    type: LoadBalancer

```

```
kubectl apply -f secrets/solution
```

Περηγηθείτε στην εφαρμογή στο :30020 ή :8020

– Η ενημέρωση γίνεται με ένα νέο Secret και μια ενημέρωση για το Deployment:

```

# save it in a folder called secrets/solution/v2-name and name
  it secret-v2.yaml

# it is a whole new Secret object with the new settings

apiVersion: v1

```

```

kind: Secret
metadata:
  name: configurable-lab-secret-v2
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
stringData:
  secret.json: |-
    {
      "Configurable__LabVersion": "v2"
    }

# save it in a folder called secrets/solution/v2-name and name
  it deployment.yaml

# it updates the existing Deployment object with the new Secret
  name in the Pod spec

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable-lab
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
spec:
  selector:
    matchLabels:
      app: configurable-lab
  template:
    metadata:
      labels:
        app: configurable-lab
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          volumeMounts:
            - name: secret
              mountPath: /app/secrets
              readOnly: true
      volumes:
        - name: secret
          secret:
            secretName: configurable-lab-secret-v2

```

```
kubectl apply -f secrets/solution/v2-name
```

Ανανεώστε την εφαρμογή και θα δείτε τη νέα διαμόρφωση μόλις ολοκληρωθεί η διάθεση του Pod.

- Αυτή η επιλογή έχει το πλεονέκτημα της διατήρησης των παλιών ρυθμίσεων διαμόρφωσης, ώστε να μπορείτε να κάνετε επαναφορά εάν υπάρχει πρόβλημα:

```
kubectl get secrets -l app=configurable-lab

kubectl rollout undo deploy/configurable-lab
```

### Χρήση σχολιασμών με εκδόσεις διαμόρφωσης

Οι σχολιασμοί είναι η άλλη δημοφιλής επιλογή. Οι σχολιασμοί είναι στοιχεία μεταδεδομένων όπως οι ετικέτες, αλλά χρησιμοποιούνται για την καταγραφή επιπλέον πληροφοριών, ενώ οι ετικέτες χρησιμοποιούνται εσωτερικά από την Kubernetes.

- Επαναφέρετε το εργαστήριο αφαιρώντας το και εκ νέου αναπτύξτε το:

```
kubectl delete deployment,secret -l app=configurable-lab

kubectl apply -f secrets/solution
```

Περιηγηθείτε στα :30020 ή :8020 - η διαμόρφωση επιστρέφει στην v1

- Τώρα αναπτύξτε την ενημέρωση - είναι μια αλλαγή στο υπάρχον αντικείμενο Secret και ένας νέος σχολιασμός στην προδιαγραφή Pod:

```
# save it in a folder called secrets/solution/v2-annotation and
  name it secret-v2.yaml

# it is an update to the data in the existing Secret object

apiVersion: v1
kind: Secret
metadata:
  name: configurable-lab-secret
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
stringData:
  secret.json: |-
    {
      "Configurable__LabVersion": "v2"
```

```

    }

# save it in a folder called secrets/solution/v2-annotation and
# name it deployment.yaml

# it updates the existing Deployment object, still with the same
# Secret name but adding an annotation to store the config
# version

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable-lab
  labels:
    kubernetes.courselabs.co: secrets
    app: configurable-lab
spec:
  selector:
    matchLabels:
      app: configurable-lab
  template:
    metadata:
      labels:
        app: configurable-lab
      annotations:
        com.configurable.configversion: v2
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          volumeMounts:
            - name: secret
              mountPath: /app/secrets
              readOnly: true
      volumes:
        - name: secret
          secret:
            secretName: configurable-lab-secret

```

```
kubectl apply -f secrets/solution/v2-annotation
```

Οι αλλαγές μεταδεδομένων στις προδιαγραφές του Pod ενεργοποιούν μια διάθεση (αλλά όχι στα μεταδεδομένα για το ίδιο το Deployment). Θα δείτε τη ρύθμιση διαμόρφωσης v2 όταν κάνετε ανανέωση.

- Αυτή η επιλογή δεν διατηρεί το ιστορικό ρυθμίσεων, επομένως για να αναιρέσετε τις αλλαγές θα πρέπει να εφαρμόσετε ξανά το v1 Secret και μετά να επαναφέρετε:

```
kubectl get secrets -l app=configurable-lab

kubectl apply -f secrets/solution/secret.yaml

kubectl rollout undo deploy/configurable-lab
```

Ποιο από αυτά λειτουργεί για ένα project εξαρτάται από τον οργανισμό σας, τον τρόπο αποθήκευσης των παραμέτρων και εάν θέλετε τα deployments να είναι πλήρως αυτοματοποιημένες ή εάν πρέπει να αποσυνδέσετε τη διαχείριση ρυθμίσεων από τις αναπτύξεις εφαρμογών.

#### 4.15 Εργαστηριακή άσκηση 15

Για να αποκτήσετε πρόσβαση στο δίσκο του κόμβου, δημιουργήστε ένα νέο Pod με τόμο HostPath που στοχεύει τη διαδρομή root `"/`.

##### Δημιουργήστε ένα Pod με τόμο HostPath

```
# save it in a folder called persistentvolumes/solution and name
it sleep-with-hostpath.yaml

# it uses a HostPath volume in the Pod spec.

apiVersion: v1
kind: Pod
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: persistentvolumes
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: node-root
          mountPath: /node-root
  volumes:
    - name: node-root
      hostPath:
        path: /
        type: Directory
```

– Αναπτύξτε το Pod:

```
kubectl apply -f persistentvolumes/solution
```

Το δοχείο Pod τοποθετεί το root του δίσκου του κόμβου στο /node-root μέσα στο δοχείο και εκτελείται ως root.

- Αυτό σημαίνει ότι μπορείτε να κάνετε σχεδόν οτιδήποτε στο δίσκο:

```
kubectl exec pod/sleep -- ls /node-root

kubectl exec pod/sleep -- mkdir -p /node-root/secret/hacker/
tools

kubectl exec pod/sleep -- ls -l /node-root/secret/hacker
```

Γι' αυτό δεν είναι ασφαλές :) Εάν χρειάζεστε πρόσβαση στο δίσκο του κόμβου, θα πρέπει να χρησιμοποιήσετε ένα hostPath με πιο περιοριστικό εύρος, όχι ολόκληρο το δίσκο root.

#### 4.16 Εργαστηριακή άσκηση 16

Η λύση που προτείνουμε δημιουργεί ένα νέο Namespace για την εκτέλεση του Nginx και χρησιμοποιεί ένα FQDN στη διαμόρφωση Nginx για να διαμεσολαβήσει την εφαρμογή Ιστού Pi - και καθορίζει τη σωστή θύρα:

```
# save it in a folder called namespaces/solution and name it 01-
namespace.yaml

# it is the front-end namespace

apiVersion: v1
kind: Namespace
metadata:
  name: front-end
  labels:
    kubernetes.courselabs.co: namespaces
```

- Αναπτύξτε πρώτα το Namespace:

```
kubectl apply -f namespaces/solution/01-namespace.yaml
```



- Στη συνέχεια, την αρχική ρύθμιση του διακομιστή μεσολάβησης:

```
kubectl apply -n front-end -f namespaces/specs/reverse-proxy
```

Περιηγηθείτε στο <http://localhost:30040> - θα λάβετε ένα σφάλμα από το πρόγραμμα περιήγησής σας.

- Ελέγξτε τα logs και θα δείτε ότι ο διακομιστής μεσολάβησης δεν θα εκτελείται εάν δεν μπορεί να βρεθεί ο διακομιστής "upstream":

```
kubectl logs -n front-end -l app=pi-proxy
```

Θα δείτε nginx: [emerg] host not found in upstream "pi-web-internal" in /etc/nginx/nginx.conf:28

- Ενημερώστε το ConfigMap με το σωστό FQDN, χρησιμοποιώντας την εφαρμογή στο υπάρχον Namespace pi και, στη συνέχεια, θα χρειαστεί να διαθέσετε νέα Pods για να επιλέξετε την αλλαγή διαμόρφωσης:

```
kubectl apply -f namespaces/solution/nginx-configMap.yaml
```

```
kubectl rollout restart -n front-end deploy/pi-proxy
```

Περιηγηθείτε στο <http://localhost:30040/pi?dp=40000> - τώρα ο διακομιστής μεσολάβησης φορτώνει το περιεχόμενο από την εφαρμογή Pi. η απάντηση θα αργήσει μερικά δευτερόλεπτα.

- Επιβεβαιώστε ότι η cache χρησιμοποιείται:

```
kubectl exec -n front-end deploy/pi-proxy -- ls /tmp
```

Ανανεώστε την εφαρμογή Ιστού και η απόκριση θα είναι άμεση.

#### 4.17 Εργαστηριακή άσκηση 17

##### Δημιουργήστε ένα Pod με τόμο HostPath

Πρώτα επιβεβαιώστε ότι η εφαρμογή δεν έχει πρόσβαση σε αντικείμενα service-account-viewer.yaml:

```
kubectl auth can-i get serviceaccounts -n default --as system:
serviceaccount:default:kube-explorer
```

– Αυτή η προδιαγραφή περιέχει τα νέα δικαιώματα:

Θα μπορούσατε επίσης να τροποποιήσετε το αρχικό Role, αλλά αυτό σημαίνει ότι οποιοσδήποτε άλλος που χρησιμοποιεί αυτόν τον Role θα λάβει επίσης τη νέα άδεια.

```
# save it in a folder called rbac/solution and name it service-
account-viewer.yaml

# a Role with rules to get and list ServiceAccounts, and a
RoleBinding applying it to the app's ServiceAccount

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: service-account-viewer
  namespace: default # this is the scope where the role applies
  labels:
    kubernetes.courselabs.co: rbac
rules:
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kube-explorer-service-account-viewer
  namespace: default # needs to match the ns in the role
  labels:
    kubernetes.courselabs.co: rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: service-account-viewer
subjects:
- kind: ServiceAccount
  name: kube-explorer
  namespace: default # the subject can be in a different ns
```

```
kubectl apply -f rbac/solution/service-account-viewer.yaml

kubectl auth can-i get serviceaccounts -n default --as system:
serviceaccount:default:kube-explorer
```

Τώρα η εφαρμογή έχει τα δικαιώματα που χρειάζεται, ο σύνδεσμος Λογαριασμού υπηρεσίας λειτουργεί (π.χ. <http://localhost:30010/ServiceAccounts>), αλλά μόνο για τον προεπιλεγμένο Namespace. Θα πρέπει να λαμβάνετε ένα σφάλμα 403 εάν προσπαθήσετε να προβάλετε ServiceAccounts σε άλλο Namespace (<http://localhost:30010/ServiceAccounts?ns=kube-system>).

### Διασφάλιση του Sleep Deployment

- Πρώτα επαληθεύτε ότι το αρχικό Sleep Pod έχει τοποθετημένο το διακριτικό SA:

```
kubectl exec deploy/sleep -- cat /var/run/secrets/kubernetes.io/
serviceaccount/token
```

Ο απλούστερος τρόπος για να το διορθώσετε είναι με το πεδίο automountServiceAccountToken στην προδιαγραφή Pod:

```
# save it in a folder called rbac/solution and name it sleep-
without-sa-token.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
  labels:
    kubernetes.courselabs.co: rbac
spec:
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
        sa-token: none
    spec:
      automountServiceAccountToken: false
      containers:
        - name: sleep
          image: kiamol/ch03-sleep
```

```
kubectl apply -f rbac/solution/sleep-without-sa-token.yaml
```

```
kubectl wait --for=condition=Ready pod -l app=sleep,sa-token=
  none

kubectl exec deploy/sleep -- cat /var/run/secrets/kubernetes.io/
  serviceaccount/token
```

- Κάθε Pod εκτελείται στο πλαίσιο ενός ServiceAccount - ελέγξτε και θα δείτε ότι αυτό χρησιμοποιεί την προεπιλεγμένη SA:

```
kubectl get pod -l app=sleep,sa-token=none -o jsonpath='{.items
  [0].spec.serviceAccountName}'
```

Μια καλύτερη προσέγγιση είναι να χρησιμοποιήσετε έναν προσαρμοσμένο ServiceAccount με το automountServiceAccountToken που έχει οριστεί σε επίπεδο SA.

Οποιαδήποτε εφαρμογή δεν χρησιμοποιεί τον διακομιστή API μπορεί να χρησιμοποιήσει αυτό το SA - οι εφαρμογές που χρησιμοποιούν τον διακομιστή API θα έχουν το δικό τους SA με τα δικαιώματα που χρειάζεται η εφαρμογή.

```
# save it in a folder called rbac/solution and name it sleep-
  with-no-token-sa.yaml

# it is a new SA and Pod configured to use it

apiVersion: v1
kind: ServiceAccount
metadata:
  name: no-token
  labels:
    kubernetes.courselabs.co: rbac
automountServiceAccountToken: false
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep-v2
  labels:
    kubernetes.courselabs.co: rbac
spec:
  selector:
    matchLabels:
      app: sleep-v2
  template:
    metadata:
      labels:
        app: sleep-v2
```

```

    sa-token: none
  spec:
    serviceAccountName: no-token
    containers:
      - name: sleep-v2
        image: kiamol/ch03-sleep

```

#### 4.18 Εργαστηριακή άσκηση 18

Τα DaemonSets σάς επιτρέπουν να καθορίσετε τη στρατηγική ενημέρωσης και το Kubectl υποστηρίζει διαγραφές για ελεγκτές χωρίς διαγραφή Pods.

##### Αντικαταστήστε το Pod μόνο όταν διαγραφεί με μη αυτόματο τρόπο

```

# save it in a folder called daemonsets/solution and name it
  daemonset-update-on-delete.yaml

# it specifies an update strategy type of OnDelete

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx
  labels:
    kubernetes.courselabs.co: daemonsets
spec:
  updateStrategy:
    type: OnDelete
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        kubernetes.courselabs.co: daemonsets
        app: nginx
    spec:
      initContainers:
        - name: init-html
          image: kiamol/ch03-sleep
          command: ['sh', '-c', "echo '<!DOCTYPE html><html><
body><h1>I will only be replaced if you delete me</h1></body>
</html>' > /html/index.html"]
          volumeMounts:
            - name: html
              mountPath: /html
      containers:
        - image: nginx:1.18-alpine

```

```

      name: nginx
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
    volumes:
      - name: html
        emptyDir: {}

```

- Το DaemonSet θα δημιουργήσει ένα Pod αντικατάστασης μόνο όταν το υπάρχον διαγραφεί από άλλη διεργασία:

```

kubectl apply -f daemonsets/solution

kubectl get pods -l app=nginx --watch

```

Το DaemonSet έχει ενημερωθεί, αλλά δεν θα αντικαταστήσει το Pod, παρόλο που οι προδιαγραφές του Pod έχουν αλλάξει.

```

kubectl delete pod -l app=nginx

kubectl get pods -l app=nginx

```

#### **Διαγράψτε το DaemonSet αλλά διατηρήστε το Pod**

- Το Kubernetes διατηρεί τη σχέση μεταξύ Pods και ελεγκτών, αλλά σας επιτρέπει να διακόψετε αυτή τη σχέση με μη διαδοχικές διαγραφές.

```

kubectl delete ds nginx --cascade=false

kubectl get ds

kubectl get po -l app=nginx

```

Το DaemonSet αφαιρέθηκε, αλλά το Pod που χρησιμοποιούσε για τον έλεγχο είναι ακόμα εκεί.

## **4.19 Εργαστηριακή άσκηση 19**

### **Δημοσίευση της διαμορφώσιμης εφαρμογής**

Τα αντικείμενα Ingress αναφέρονται σε Υπηρεσίες στο τοπικό Namespace, επομένως πρέπει να δημιουργήσετε το Ingress σας στο ίδιο Namespace με την εφαρμογή:

```
# save it in a folder called ingress/solution/ingress and name
it configurable-http.yaml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: configurable
  namespace: configurable
spec:
  rules:
  - host: configurable.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: configurable-web
            port:
              name: config-app
```

```
kubectl apply -f ingress/solution/ingress
```

– Είναι ένας νέος τομέας, επομένως πρέπει να τον προσθέσετε στο αρχείο hosts σας:

```
# on Windows:
./scripts/add-to-hosts.ps1 configurable.local 127.0.0.1

# on *nix:
./scripts/add-to-hosts.sh configurable.local 127.0.0.1
```

Τώρα μπορείτε να περιηγηθείτε στο <http://configurable.local:8000> (ή <http://configurable.local:30000>)

### Χρήση τυπικών θυρών HTTP και HTTPS

Για αυτό το μόνο που χρειάζεται να κάνετε είναι να αλλάξετε τις δημόσιες θύρες για τον ελεγκτή εισόδου LoadBalancer Service:

```
# save it in a folder called ingress/solution/controller and
name it ingress/solution/controller

apiVersion: v1
kind: Service
metadata:
```

```

name: ingress-nginx-controller-lb
namespace: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      targetPort: http
    - name: https
      port: 443
      targetPort: https
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller

```

```
kubectl apply -f ingress/solution/controller
```

```
kubectl get svc -n ingress-nginx
```

- http://configurable.local
- http://pi.local
- http://localhost
- https://pi.local (θα δείτε ένα σφάλμα σχετικά με το ότι το πιστοποιητικό TLS δεν είναι αξιόπιστο)

**Γιατί δεν μπορείτε να το κάνετε αυτό με ένα cluster που δεν υποστηρίζει τις υπηρεσίες LoadBalancer;**

Τα NodePort περιορίζονται στην περιοχή μη προνομιούχων θυρών - 30000+. Δεν μπορείτε να έχετε NodePort ακρόαση σε 80 ή 443.

## 4.20 Εργαστηριακή άσκηση 20

### Αναστολή του CronJob εκκαθάρισης

Αυτό θα χρειαστεί κάποια έρευνα εάν δεν πρόκειται να χρησιμοποιήσετε το "kubectl apply". Η εντολή που θα εκτελέσετε είναι:

```
kubectl edit cronjob job-cleanup
```

Αυτό εκκινεί την προδιαγραφή YAML στον επεξεργαστή σας - στην προδιαγραφή για το CronJob θα βρείτε το πεδίο αναστολής. Απλώς αλλάζετε την τιμή από false σε true, αποθηκεύστε το αρχείο και βγείτε από το πρόγραμμα επεξεργασίας.

- Το Kubectl εφαρμόζει την αλλαγή όταν ο επεξεργαστής βγαίνει:



```
kubectl get cronjob
```

Τώρα το CronJob έχει τεθεί σε αναστολή, επομένως η προδιαγραφή εξακολουθεί να υπάρχει, αλλά δεν θα δημιουργήσει άλλες εργασίες.

### Δημιουργία Job από ένα CronJob

Αυτό είναι απλό, αλλά χρησιμοποιείτε σπάνια το "kubectl create" και μπορεί να ξεχάσετε τι μπορεί να κάνει:

```
kubectl create job --help
```

- Σας δείχνει την ακριβή σύνταξη που χρειάζεστε. Για αυτό το εργαστήριο:

```
kubectl create job db-backup-job --from=cronjob/db-backup
kubectl get jobs -l app=db-backup
kubectl logs -l app=db-backuplp
```

- Η προσθήκη της ετικέτας του εργαστηρίου θα βοηθήσει στον καθαρισμό:

```
kubectl label job db-backup-job kubernetes.courselabs.co=jobs
```

## 4.21 Εργαστηριακή άσκηση 21

Πρέπει να ορίσετε ένα StatefulSet - η προδιαγραφή είναι πολύ παρόμοια με τα Deployments, αλλά χρειάζεστε μια Υπηρεσία και θα αντικαταστήσετε τον τόμο emptyDir με ένα mount και ένα volumeClaimTemplate:

```
# save it in a folder called statefulsets/solution and name it
service.yaml

# headless Service for the StatefulSet

apiVersion: v1
kind: Service
metadata:
  name: simple-proxy-internal
  labels:
    kubernetes.courselabs.co: statefulsets
spec:
  ports:
    - port: 8030
```

```

        targetPort: 80
    selector:
        app: simple-proxy
    clusterIP: None

# save it in a folder called statefulsets/solution and name it
# statefulset.yaml

# StatefulSet with PVC template and parallel management policy

apiVersion: apps/v1
kind: StatefulSet
metadata:
    name: simple-proxy
    labels:
        kubernetes.courselabs.co: statefulsets
        app: simple-proxy
spec:
    selector:
        matchLabels:
            app: simple-proxy
    serviceName: simple-proxy-internal
    replicas: 2
    podManagementPolicy: Parallel
    template:
        metadata:
            labels:
                app: simple-proxy
        spec:
            containers:
                - name: app
                  image: nginx:1.18-alpine
                  volumeMounts:
                    - name: config
                      mountPath: "/etc/nginx/"
                      readOnly: true
                    - name: cache
                      mountPath: /cache
            volumes:
                - name: config
                  configMap:
                      name: simple-proxy-configmap
    volumeClaimTemplates:
        - metadata:
            name: cache
            labels:
                app: simple-proxy
          spec:
            accessModes:
                - ReadWriteOnce

```

```
resources:
  requests:
    storage: 50Mi
```

- Θα χρειαστεί να καταργήσετε το Deployment πριν δημιουργήσετε το StatefulSet:

```
kubectl delete deploy simple-proxy
kubectl apply -f statefulsets/solution
```

- Παρακολουθήστε τα Pods - δημιουργούνται παράλληλα:

```
kubectl get po -l app=simple-proxy --watch
# Ctrl-C when the Pods are running
kubectl get pvc -l app=simple-proxy
```

Κάθε Pod έχει ένα PVC, το οποίο είναι τοποθετημένο στον φάκελο "/cache".

- Δοκιμάστε τον proxy:

```
curl -v localhost:8040
# OR
curl -v localhost:30040
```

Επαναλάβετε και θα δείτε το "X-Cache: HIT" στις κεφαλίδες απόκρισης.

- Η cache βρίσκεται στον τόμο για κάθε Pod:

```
kubectl exec simple-proxy-0 -- ls /cache
kubectl exec simple-proxy-1 -- ls /cache
```

## 4.22 Εργαστηριακή άσκηση 22

Ενδεικτική λύση:

```
# save it in a folder called productionizing/solution and name
  it deployment-productionized.yaml

# it adds CPU resources and container probes

apiVersion: apps/v1
kind: Deployment
metadata:
  name: configurable
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  replicas: 5
  selector:
    matchLabels:
      app: configurable
  template:
    metadata:
      labels:
        app: configurable
    spec:
      containers:
        - name: app
          image: sixeyed/configurable:21.04
          env:
            - name: Configurable__FailAfterCallCount
              value: "3"
          resources:
            limits:
              cpu: 250m
            requests:
              cpu: 125m
          readinessProbe:
            httpGet:
              path: /healthz
              port: 80
            periodSeconds: 1
            failureThreshold: 1
          livenessProbe:
            httpGet:
              path: /healthz
              port: 80
            periodSeconds: 5
            failureThreshold: 1
```

```
# save it in a folder called productionizing/solution and name
  it hpa-cpu.yaml

# HPA to scale the Deployment

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: configurable-cpu
  labels:
    kubernetes.courselabs.co: productionizing
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: configurable
  minReplicas: 5
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

```
kubectl apply -f productionizing/solution
```

- Ανοίξτε δύο τερματικά ώστε μπορείτε να παρακολουθήσετε την επισκευή και την κλιμάκωση σε δράση:

```
kubectl get pods -l app=configurable --watch
```

```
kubectl get endpoints configurable-lb --watch
```

Περιγηθείτε στην εφαρμογή και κάντε πολλές ανανεώσεις. Μπορεί να δείτε αποτυχίες επειδή η εφαρμογή αποτυγχάνει πολύ συχνά, αλλά αφήστε την λίγα δευτερόλεπτα μεταξύ των ανανεώσεων και η εφαρμογή επανέρχεται στο διαδίκτυο.

- Ή αν έχετε εγκαταστήσει το "watch" ("brew install watch" σε macOS, ήδη υπάρχει σε Linux, μη διαθέσιμο στα Windows):

```
watch -n 0.5 kubectl get pods,endpoints,hpa
```

```
watch -n 0.5 curl -s http://localhost:8040
```

Τελικά όλα τα Pods θα μπουν σε CrashLoopBackOff επειδή το Kubernetes πιστεύει ότι η εφαρμογή είναι ασταθής.

### Δοκιμή του HPA

- Το HPA είναι ανεξάρτητο από το Deployment. Μπορείτε να κλιμακώσετε το Deployment με μη αυτόματο τρόπο (ή να διαγράψετε Pods για να προσομοιώσετε την απώλεια κόμβου) και το HPA θα το παρακάμψει.

```
watch -n 0.5 kubectl get pods,endpoints,hpa
watch -n 0.5 curl -s http://localhost:8040
```

- Χειροκίνητη μείωση της κλίμακας:

```
kubectl scale deployment/configurable --replicas 1
kubectl get hpa configurable-cpu --watch
```

Το HPA δημιουργεί αντίγραφα ασφαλείας μετά από λίγα λεπτά - το ελάχιστο δεν εξαρτάται από τη χρήση της CPU και παρακάμπτει τη ρύθμιση κλίμακας για το Deployment.

- Κλιμακώστε χειροκίνητα:

```
kubectl scale deployment/configurable --replicas 8
kubectl get hpa configurable-cpu --watch
```

Μετά από λίγα λεπτά ακόμη, το HPA μειώνεται - αρχικά μειώνεται στο μέγιστο, αλλά δεν υπάρχει δραστηριότητα CPU, επομένως θα μειώνεται επανειλημμένα μέχρι να φτάσει στο ελάχιστο.

Δεν μπορείτε να διαμορφώσετε τους χρονισμούς αύξησης και μείωσης κλίμακας για V1 HPA. Εάν χρειάζεστε αυτό το επίπεδο ελέγχου, μπορείτε να χρησιμοποιήσετε το [HorizontalPodAutoscaler \(autoscaling/v2\)](#), το οποίο είναι μια πιο περίπλοκη προδιαγραφή HPA που επιτρέπει [άλλες μετρήσεις](#).

#### 4.23 Εργαστηριακή άσκηση 23

Η λύση χρησιμοποιεί δύο νέες παραμέτρους scrape, μία για cAdvisor και μία για kube-state-metrics:

```

# save it in a folder called monitoring/solution and name it
prometheus-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s

    scrape_configs:
      - job_name: 'app'
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels:
              - __meta_kubernetes_namespace
            action: keep
            regex: default
          - source_labels:
              -
            __meta_kubernetes_pod_annotationpresent_prometheus_io_scrape
          -
            __meta_kubernetes_pod_annotation_prometheus_io_scrape
            regex: true;true
            action: keep
          - source_labels:
              -
            __meta_kubernetes_pod_annotationpresent_prometheus_io_path
          -
            __meta_kubernetes_pod_annotation_prometheus_io_path
            regex: true;(.*?)
            target_label: __metrics_path__
          - source_labels:
              -
            __meta_kubernetes_pod_annotationpresent_prometheus_io_target
          -
            __meta_kubernetes_pod_annotation_prometheus_io_target
            regex: true;(.*?)
            target_label: __param_target
          - source_labels:
              -
            __meta_kubernetes_pod_annotationpresent_prometheus_io_port
            - __address__
          -
            __meta_kubernetes_pod_annotation_prometheus_io_port
            action: replace

```

```

    regex: true;([^\:]+) (?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
  - source_labels:
    - __meta_kubernetes_pod_labelpresent_component
    - __meta_kubernetes_pod_label_component
    regex: true;(.*?)
    target_label: job
  - source_labels:
    - __meta_kubernetes_pod_name
    target_label: instance

- job_name: 'cadvisor'
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels:
    - __meta_kubernetes_namespace
    - __meta_kubernetes_pod_labelpresent_app
    - __meta_kubernetes_pod_label_app
    action: keep
    regex: kube-system;true;cadvisor

- job_name: 'kube-state-metrics'
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels:
    - __meta_kubernetes_namespace
    - __meta_kubernetes_pod_labelpresent_app
    - __meta_kubernetes_pod_label_app
    action: keep
    regex: kube-system;true;kube-state-metrics

```

– Εφαρμόστε την αλλαγή:

```
kubectl apply -f ./monitoring/solution/prometheus-config.yaml
```

– Ενεργοποιήστε ένα rollout για να φορτώσετε τη νέα διαμόρφωση:

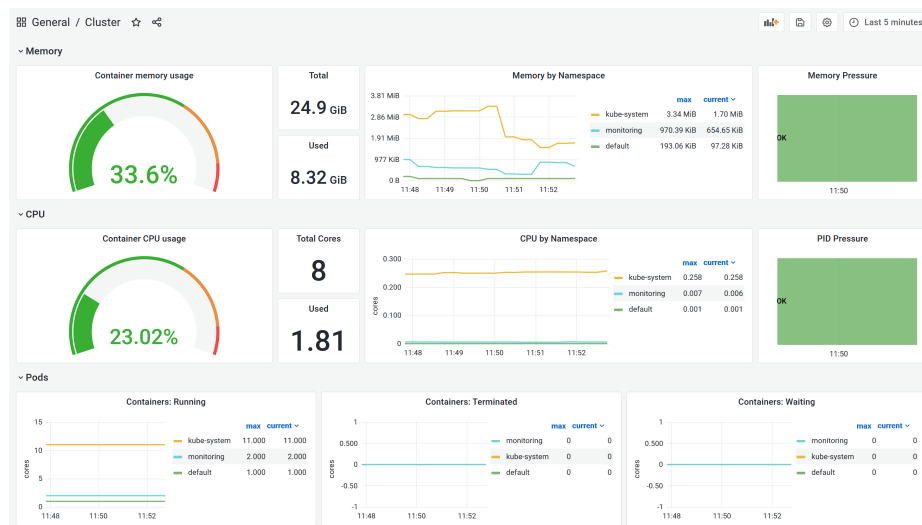
```
kubectl -n monitoring rollout restart deploy/prometheus
```

Περιηγηθείτε στο Prometheus target config ελέγξτε ότι οι νέοι στόχοι είναι έτοιμοι:



Prometheus Alerts Graph Status ▾ Help Classic UI		
Targets		
All Unhealthy Collapse All		
app (1/1 up) <a href="#">show less</a>		
Endpoint	State	Labels
<a href="http://10.1.1.146:9110/metrics">http://10.1.1.146:9110/metrics</a>	UP	instance="fulfilment-processor-7d545b4d59-hjgt5" job="processor"
cadvisor (1/1 up) <a href="#">show less</a>		
Endpoint	State	Labels
<a href="http://10.1.1.160:8080/metrics">http://10.1.1.160:8080/metrics</a>	UP	instance="10.1.1.160:8080" job="cadvisor"
kube-state-metrics (1/1 up) <a href="#">show less</a>		
Endpoint	State	Labels
<a href="http://10.1.1.159:8080/metrics">http://10.1.1.159:8080/metrics</a>	UP	instance="10.1.1.159:8080" job="kube-state-metrics"

Στη συνέχεια, φορτώστε το νέο dashbaord στο Grafana:



## 4.24 Εργαστηριακή άσκηση 24

Η ενδεικτική λύση προσθέτει ένα δοχείο Alpine ως καταγραφικό, με έναν τόμο EmptyDir που ορίζεται στο Pod που μοιράζεται από το καταγραφικό και τα δοχεία

εφαρμογών:

```
# save it in a folder called monitoring/solution and name it
deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: fulfilment-processor
  labels:
    kubernetes.courselabs.co: logging
spec:
  selector:
    matchLabels:
      app: fulfilment
      component: processor
  template:
    metadata:
      labels:
        app: fulfilment
        component: processor
    spec:
      containers:
        - name: app
          image: courselabs/fulfilment-processor
          env:
            - name: Observability__Logging__File
              value: 'true'
            - name: Observability__Logging__Console
              value: 'false'
            - name: Observability__Logging__LogFilePath
              value: /app/logs/fulfilment-processor.log
          volumeMounts:
            - name: logs
              mountPath: "/app/logs"
        - name: logger
          image: alpine:3.14
          command: ['sh', '-c', 'tail -f /logs-ro/fulfilment-processor.log']
          volumeMounts:
            - name: logs
              mountPath: /logs-ro
              readOnly: true
      volumes:
        - name: logs
          emptyDir: {}
```

– Αναπτύξτε την ενημέρωση:

```
kubectl apply -f ./logging/solution/
```

- Περιμένετε να κυκλοφορήσει η ενημέρωση και θα δείτε τα logs στο Pod:

```
kubectl logs -l app=fulfilment,component=processor -c logger
```

Περιηγηθείτε στο Kibana και φορτώστε το μοτίβο ευρετηρίου log εφαρμογών στην καρτέλα Discover. Φιλτράρετε στις ετικέτες - app=fulfilment, component=processor - και θα δείτε τα logs να εισρέουν από το Fluent Bit.

Το μειονέκτημα είναι ότι τα μεταδεδομένα καταγραφής αναφέρονται σε δοχείο καταγραφικού, π.χ. καμία εικόνα εφαρμογής.

#### 4.25 Εργαστηριακή άσκηση 25

- Ξεκινήστε διαγράφοντας την αρχική εφαρμογή:

```
kubectl delete -f networkpolicy/specs/apod
```

```
kubectl delete -f networkpolicy/specs/apod/network-policies
```

- Θα πρέπει να εξακολουθείτε να έχετε την προεπιλεγμένη πολιτική άρνησης:

```
kubectl get netpol
```

- Η λύση προσθέτει έναν Namespace σε όλους τους επιλογείς Pod:

```
# save it in a folder called monitoring/solution and name it
network-policies.yaml
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-web
  namespace: apod
spec:
  podSelector:
    matchLabels:
      app: apod-web
  ingress:
    - {}
```

```

    egress:
    - to:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: apod
        podSelector:
          matchLabels:
            app: apod-api
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: apod
        podSelector:
          matchLabels:
            app: apod-log
    ports:
    - port: api
    - to:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: kube-system
        podSelector:
          matchLabels:
            k8s-app: kube-dns
    ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-api
  namespace: apod
spec:
  podSelector:
    matchLabels:
      app: apod-api
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: apod
      podSelector:
        matchLabels:
          app: apod-web
    ports:
    - port: api
  egress:
  - to:

```

```

- namespaceSelector:
  matchLabels:
    kubernetes.io/metadata.name: kube-system
  podSelector:
    matchLabels:
      k8s-app: kube-dns
  ports:
  - protocol: TCP
    port: 53
  - protocol: UDP
    port: 53
- to:
  - ipBlock:
      cidr: 3.30.35.101/24
  - ipBlock:
      cidr: 15.200.69.242/24
  ports:
  - protocol: TCP
    port: 443
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: apod-log
  namespace: apod
spec:
  podSelector:
    matchLabels:
      app: apod-log
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: apod
      podSelector:
        matchLabels:
          app: apod-web
      ports:
      - port: api
  egress: [] # deny all

```

– Αναπτύξτε την εφαρμογή:

```
kubectl apply -f networkpolicy/solution/apod
```

– Ελέγξτε ότι η εφαρμογή Ιστού μπορεί να έχει πρόσβαση στο API και το API να έχει πρόσβαση στο εξωτερικό API:

```
kubectl exec -n apod deploy/apod-web -- wget -O- -T2 http://apod-api/image
```

Ανανεώστε το `http://localhost:30016`, η εφαρμογή θα πρέπει να λειτουργεί σωστά.

– Προσπαθήστε να αποκτήσετε πρόσβαση στο API από το Sleep Pod:

```
kubectl exec sleep -- wget -O- http://apod-api.apod.svc.cluster.local/image
```

Θα λάβετε ένα σφάλμα κακής διεύθυνσης, επειδή το Pod δεν μπορεί να έχει πρόσβαση στο DNS.

– Δοκιμάστε με τη διεύθυνση IP:

```
kubectl get po -n apod -l app=apod-api -o wide
# this will fail with a timeout
kubectl exec sleep -- wget -O- -T2 http://<pod-ip-address>/image
```

Τώρα θα λάβετε ένα σφάλμα χρονικού ορίου, επειδή το Calico μπλοκάρει τη σύνδεση.

## 4.26 Εργαστηριακή άσκηση 26

Η έξοδος από την εφαρμογή Kubectl σας δίνει την πρώτη παραβίαση:

```
Error from server ([requiredlabels-ns] you must provide labels: {"kubernetes.courselabs.co"}): error when creating "labs/admission/specs/apod/01-namespace.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [requiredlabels-ns] you must provide labels: {"kubernetes.courselabs.co"}
```

– Επομένως, θα χρειαστεί να προσθέσετε την ετικέτα στην προδιαγραφή Namespace:

```
# save it in a folder called admission/solution and name it 01-namespace.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: apod
  labels:
    kubernetes.courselabs.co: admission
```

- Τώρα μπορούν να δημιουργηθούν όλοι οι πόροι, αλλά τα Deployments δεν γίνονται κλιμακωτά:

```
kubectl get deploy -n apod
```

- Λάβετε τις λεπτομέρειες για τα ReplicaSets:

```
kubectl describe rs -n apod
```

- Θα δείτε τις αποτυχίες:

```
Error creating: admission webhook "validation.gatekeeper.sh"
denied the request: [resource-limits] container <web> has no
cpu limit[requiredlabels-pods] you must provide labels: {"
app", "version"}
```

- Οι προδιαγραφές Pod χρειάζονται ετικέτες και οι προδιαγραφές δοχείων χρειάζονται όρια πόρων:

```
# save it in a folder called admission/solution and name it api.yaml

apiVersion: v1
kind: Service
metadata:
  name: apod-api
  namespace: apod
spec:
  ports:
    - port: 80
      targetPort: api
  selector:
```

```

    app: apod-api
    type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apod-api
  namespace: apod
spec:
  selector:
    matchLabels:
      app: apod-api
  template:
    metadata:
      labels:
        app: apod-api
        version: v1
    spec:
      containers:
        - name: api
          image: kiamol/ch14-image-of-the-day
          ports:
            - containerPort: 80
              name: api
          resources:
            limits:
              cpu: 500m
              memory: 300Mi

# save it in a folder called admission/solution and name it log.
# yaml

apiVersion: v1
kind: Service
metadata:
  name: apod-log
  namespace: apod
spec:
  ports:
    - port: 80
      targetPort: api
  selector:
    app: apod-log
    type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apod-log
  namespace: apod

```



```

spec:
  selector:
    matchLabels:
      app: apod-log
      version: v1
  template:
    metadata:
      labels:
        app: apod-log
        version: v1
    spec:
      containers:
        - name: api
          image: kiamol/ch14-access-log
          ports:
            - containerPort: 80
              name: api
          resources:
            limits:
              cpu: 250m
              memory: 200Mi

# save it in a folder called admission/solution and name it web.
# yaml

apiVersion: v1
kind: Service
metadata:
  name: apod-web
  namespace: apod
spec:
  ports:
    - port: 8016
      targetPort: web
      nodePort: 30016
  selector:
    app: apod-web
    type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apod-web
  namespace: apod
spec:
  selector:
    matchLabels:
      app-name: apod-web
  template:
    metadata:

```

```

labels:
  app-name: apod-web
  app: apod-web
  version: v1
spec:
  containers:
    - name: web
      image: kiamol/ch14-image-gallery
      ports:
        - containerPort: 80
          name: web
      resources:
        limits:
          cpu: 125m
          memory: 200Mi

```

- Εφαρμόστε τις καινούργιες προδιαγραφές:

```
kubectl apply -f admission/solution
```

Τα Deployments θα πρέπει να γίνουν όλες σε κλίμακα και η εφαρμογή θα πρέπει να εκτελείται στη διεύθυνση <http://localhost:30016//>

#### 4.27 Εργαστηριακή άσκηση 27

Η λύση χρησιμοποιεί δύο προσαρμοσμένους πόρους:

```

# save it in a folder called operators/solution and name it todo-
-list-queue.yaml

# a NatsCluster object with the name todo-list-queue which will
become the Service name

apiVersion: nats.io/v1alpha2
kind: NatsCluster
metadata:
  name: todo-list-queue
spec:
  size: 2
  version: "2.5.0"

# save it in a folder called operators/solution and name it todo-
-list-db.yaml

# password Secret and MysqlCluster with the expected name todo-
db

```

```

apiVersion: v1
kind: Secret
metadata:
  name: todo-db-secret
  labels:
    kubernetes.courselabs.co: operators
type: Opaque
stringData:
  ROOT_PASSWORD: op3rtorlab$
---
apiVersion: mysql.presslabs.org/v1alpha1
kind: MysqlCluster
metadata:
  name: todo-db
spec:
  image: percona:5.7.35
  replicas: 1
  secretName: todo-db-secret
  podSpec:
    resources:
      limits:
        memory: 200Mi
        cpu: 200m

```

- Δημιουργήστε τους πόρους:

```
kubectl apply -f operators/solution
```

- Ελέγξτε ότι δημιουργείται η ουρά μηνυμάτων:

```
kubectl get po,svc -l app=nats
```

- Ίσως χρειαστεί να επανεκκινήσετε το πρόγραμμα χειρισμού μηνυμάτων, εάν βρίσκεται σε κατάσταση backoff:

```
kubectl rollout restart deploy todo-save-handler
```

```
kubectl logs -l app=todo-list,component=save-handler
```

- Ελέγξτε τη δημιουργία της βάσης δεδομένων:

```
kubectl get po,svc -l app.kubernetes.io/instance=todo-db
```

Δοκιμάστε την εφαρμογή στη διεύθυνση <http://localhost:30028>

- Προσθέστε ένα νέο στοιχείο
- Ανανεώστε τη λίστα

## 5 Πηγές και βοηθητικό υλικό

- [Docker course](#)
- [Kubernetes course](#)
- [Docker cheatsheet](#)
- [Kubernetes cheatsheet](#)