**ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2024-2025**

**ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ**

**ΜΑΘΗΜΑ: Νεφοϋπολογιστική**

**Παραδείγματα χρήσης και οδηγός σύνδεσης στο Kubernetes του Εργαστηρίου VDCLOUD**

Περιεχόμενα

[1. Εισαγωγή 1](#_Toc194051179)

[2. Εγκατάσταση OpenVPN Client, kubectl και k9s 1](#_Toc194051180)

[3. Συγγραφή Manifest και Εκτέλεση με kubectl 3](#_Toc194051181)

[4. Αποθηκευτικός χώρος περιεκτών με χρήση storage classes, persistent volume claims και persistent volumes 3](#_Toc194051182)

[5. ReplicaSets (Αντίγραφα) 6](#_Toc194051183)

[6. Deployments στο Kubernetes 7](#_Toc194051184)

[7. StatefulSets στο Kubernetes 10](#_Toc194051185)

[8. DaemonSets στο Kubernetes 12](#_Toc194051186)

[9. Secrets 14](#_Toc194051187)

[10. Nginx και πολλαπλοί web servers σε Kubernetes 14](#_Toc194051188)

# Εισαγωγή

Αυτός ο οδηγός παρέχει λεπτομερείς οδηγίες για τη σύνδεση στην υποδομή του εργαστηρίου VDCLOUD μέσω VPN, την εγκατάσταση των απαραίτητων εργαλείων, τη ρύθμιση του περιβάλλοντος εργασίας ενός τοπικού υπολογιστή και την εκτέλεση εργασιών Kubernetes[[1]](#footnote-1) (k8s). Το Kubernetes (K8s) είναι μια ανοιχτού κώδικα πλατφόρμα για τη διαχείριση containerized εφαρμογών σε κλίμακα. Σκοπός του είναι να διευκολύνει τη διαχείριση, την αυτοματοποίηση και την κλιμάκωση εφαρμογών.

Θα λάβετε επίσης ένα email με δυο αρχεία ρυθμίσεων και ένα username που θα έχετε στην υποδομή. Στον παρακάτω οδηγό, όπου βλέπετε το **<username>** θα το αντικαθιστάτε με το όνομα χρήστη που λάβατε στο email σας.

# Εγκατάσταση OpenVPN Client, kubectl και k9s

Για να συνδεθείτε στην υποδομή, εγκαταστήστε τον OpenVPN client από τον παρακάτω σύνδεσμο: [OpenVPN Community Downloads](https://openvpn.net/community-downloads/)

Αφού εγκαταστήσετε τον client, εισάγετε το αρχείο .ovpn που σας έχει αποσταλεί με e-mail και συνδεθείτε.

* **Εγκατάσταση kubectl**

Το kubectl είναι το εργαλείο γραμμής εντολών για τη διαχείριση Kubernetes clusters. Εγκαταστήστε το με τις παρακάτω εντολές σε ένα Linux μηχάνημα ή εικονική μηχανή:

**sudo** apt-get install -y apt-transport-https ca-certificates curl gnupg

**curl** -fsSL https**://**pkgs.k8s.io**/**core**:/**stable**:/**v1.32**/**deb**/**Release.key **|** **sudo** gpg --dearmor -o **/**etc**/**apt**/**keyrings**/**kubernetes-apt-keyring.gpg

**sudo** chmod 644 **/**etc**/**apt**/**keyrings**/**kubernetes-apt-keyring.gpg

**echo** 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.32/deb/ /' **|** **sudo** tee **/**etc**/**apt**/**sources.list.d**/**kubernetes.list

**sudo** chmod 644 **/**etc**/**apt**/**sources.list.d**/**kubernetes.list

**sudo** apt-get update

**sudo** apt-get install -y kubectl

**mkdir** **~/.**kube

Εισάγετε το αρχείο config που σας έχει αποσταλεί με e-mail στην τοποθεσία ~/.kube/config για να μπορεί το εργαλείο kubectl να συνδεθεί με την υποδομή k8s.

Για να το κάνετε αυτό, θα πρέπει να αντιγράψετε το αρχείο config από την τοποθεσία του συστήματος αρχείων του host υπολογιστή σας που αρχικά το κατεβάσατε (δηλαδή windows) στον κατάλογο ~/.kube του Linux WSL.

Έστω ότι έχετε κατεβάσει το αρχείο config στον κατάλογο Downloads (Λήψεις) του χρήστη των windows.

Για να το αντιγράψετε στην σωστή θέση, πρέπει να εκτελέσετε τις παρακάτω εντολές στο WSL Linux. Θα αντικαταστήσετε το **<username>** με το όνομα χρήστη των windows της δικής σας εγκατάστασης. Για παράδειγμα, στην δική μου περίπτωση είναι /mnt/c/Users/**ikons**/Downloads/config

cd

mkdir .kube

cp /mnt/c/Users/**<username>**/Downloads/config ~/.kube/config

Ένας άλλος τρόπος να το κάνετε είναι μέσω του Windows explorer επιλέγοντας το Linux folder:

A screenshot of a computer

AI-generated content may be incorrect.

* **Εγκατάσταση k9s**

Το k9s είναι ένα εργαλείο για την παρακολούθηση και τη διαχείριση Kubernetes clusters. Εγκαταστήστε το ως εξής:

**wget** https**://**github.com**/**derailed**/**k9s**/**releases**/**download**/**v0.40.10**/**k9s\_linux\_amd64.deb

**sudo** dpkg -i k9s\_linux\_amd64.deb

**echo** "export KUBE\_EDITOR=nano" **>>** **~/.**bashrc

Το εργαλείο k9s χρησιμοποιεί και αυτό το αρχείο ρυθμίσεων ~/.kube/config.

* **Παρακολούθηση Εκτέλεσης μέσω k9s**

Για να παρακολουθήσετε την εκτέλεση της εργασίας που μόλις υποβάλλατε, χρησιμοποιήστε το k9s:

k9s

**Παραδείγματα χρήσης**

**Εμφάνιση των pods**:

:pods

**Προβολή των logs ενός pod**:

l

**Έλεγχος κατάστασης ενός pod**:

d

# Συγγραφή Manifest και Εκτέλεση με kubectl

Τα **manifest** είναι αρχεία YAML που περιγράφουν τους πόρους του Kubernetes. Μπορείς να δημιουργήσεις και να εφαρμόσεις αυτά τα αρχεία χρησιμοποιώντας την εντολή kubectl apply.

**Παράδειγμα manifest για Pod:**

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: nginx

**spec**:

**containers**:

**- name**: nginx

**image**: nginx:latest

**ports**:

**- containerPort**: 80

Για να δημιουργήσεις το παραπάνω Pod:

kubectl apply -f nginx-pod.yaml

Για να δεις αν το Pod έχει δημιουργηθεί σωστά:

kubectl get pod nginx -o wide

Λογικά θα δείτε κάτι της μορφής

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

nginx 1/1 Running 0 7h34m 10.233.101.168 source-code-pc6 <none> <none>

Μέσω της διεύθυνσης IP μπορείτε να ανοίξετε το pod και να δείτε την σελίδα hello-world του nginx στον browser σας;:

http:// 10.233.101.168

# Αποθηκευτικός χώρος περιεκτών με χρήση storage classes, persistent volume claims και persistent volumes

**Μη Μόνιμη Αποθήκευση (Ephemeral Storage):** Τα Pods στο Kubernetes από προεπιλογή δεν έχουν μόνιμη αποθήκευση. Αν αποθηκεύσετε δεδομένα σε ένα Pod, αυτά τα δεδομένα θα χαθούν μόλις το Pod διαγραφεί ή επανεκκινηθεί. Αυτό συμβαίνει επειδή τα δεδομένα αποθηκεύονται στο filesystem του container, το οποίο είναι προσωρινό.

**Μόνιμη Αποθήκευση με Persistent Volumes (PVs), Persistent Volume Claims (PVCs) και StorageClass:** Το Kubernetes εισάγει το concept των Persistent Volumes (PVs) και των Persistent Volume Claims (PVCs) για να παρέχει μόνιμη αποθήκευση σε Pods.

* **Persistent Volume (PV):** Είναι ένας πόρος αποθήκευσης που είναι αποδεσμευμένος από τα Pods και διαχειρίζεται από τον διαχειριστή του cluster. Τα PVs μπορεί να είναι φυσικές συσκευές αποθήκευσης (όπως disk volumes σε υποδομές cloud) ή abstractions που επιτρέπουν τη σύνδεση με άλλους τύπους αποθηκευτικών συστημάτων.
* **Persistent Volume Claim (PVC):** Είναι ένας τρόπος για τον χρήστη να ζητήσει συγκεκριμένο τύπο ή μέγεθος αποθηκευτικού χώρου. Τα PVCs είναι συνήθως ο τρόπος με τον οποίο τα Pods αποκτούν πρόσβαση σε ένα PV.
* **StorageClass:** Με την εισαγωγή των StorageClasses, το Kubernetes επιτρέπει τη δημιουργία αποθηκευτικών πόρων (PVs) δυναμικά και με βάση τις ανάγκες των PVCs. Το StorageClass καθορίζει τις ρυθμίσεις αποθήκευσης, όπως ο τύπος αποθήκευσης (π.χ., SSD, HDD) και άλλες παραμέτρους, όπως το επιθυμητό επίπεδο απόδοσης.

Για παράδειγμα, με την παρακάτω εντολή

ikons@source-code-master:~$ kubectl get storageclass

NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE

**local-path** (default) rancher.io/local-path Delete WaitForFirstConsumer false 44d

βλέπουμε εάν υπάρχει κάποιο storage class που μπορεί να μας προσφέρει αποθηκευτικό χώρο. Η πληροφορία που παίρνουμε είναι ότι το Kubernetes cluster μας χρησιμοποιεί το StorageClass **local-path**, το οποίο παρέχεται από το **rancher.io/local-path** provisioner και έχει πολιτική ανακαίνισης (reclaimPolicy) Delete. Επίσης, χρησιμοποιεί το WaitForFirstConsumer για το volumeBindingMode, το οποίο σημαίνει ότι το PV θα δημιουργηθεί μόνο όταν υπάρξει ένα Pod που θα το ζητήσει.

* **Δημιουργία Persistent Volume Claim (PVC) με το local-path StorageClass**

Αρχικά, δημιουργούμε το YAML αρχείο για το PVC που θα χρησιμοποιεί το υπάρχον **local-path** StorageClass. Δημιουργήστε το αρχείο nginx-pvc.yaml και προσθέστε το παρακάτω περιεχόμενο:

**apiVersion**: v1

**kind**: PersistentVolumeClaim

**metadata**:

**name**: nginx-pvc

**spec**:

**accessModes**:

- ReadWriteOnce

**resources**:

**requests**:

**storage**: 1Gi

**storageClassName**: local-path

Για να δημιουργήσετε το PVC στον cluster σας, εκτελέστε την παρακάτω εντολή:

kubectl apply -f nginx-pvc.yaml

Αυτή η εντολή δημιουργεί το PVC που ζητάει 1Gi αποθηκευτικού χώρου μέσω του **local-path** StorageClass.

* **Δημιουργία του Nginx Pod που Χρησιμοποιεί PVC (Μόνιμος Αποθηκευτικός Χώρος)**

Πρώτα, θα δημιουργήσουμε το αρχείο κειμένου που θα τοποθετηθεί στον κατάλογο που σερβίρει ο Nginx. Αυτό το αρχείο θα είναι διαθέσιμο ακόμα και αν το Pod διαγραφεί ή επανεκκινηθεί.

Δημιουργούμε το αρχείο index.html που θα τοποθετηθεί στον κατάλογο /usr/share/nginx/html για να το σερβίρει ο Nginx:

Δημιουργία αρχείου index.html για το PVC:

<html>

<head><title>**Persistent Storage Example**</title></head>

<body>

<h1>**Welcome to Nginx with Persistent Volume!**</h1>

<p>**This content is stored in a Persistent Volume and will persist even if the Pod is terminated.**</p>

</body>

</html>

Αφού δημιουργήσουμε το PVC, μπορούμε να δημιουργήσουμε το Pod που θα χρησιμοποιεί το Persistent Volume μέσω του PVC και θα σερβίρει το αρχείο index.html. Δημιουργήστε το αρχείο nginx-pod.yaml και προσθέστε το παρακάτω περιεχόμενο:

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: nginx-pod

**spec**:

**containers**:

**- name**: nginx

**image**: nginx:latest

**volumeMounts**:

**- mountPath**: /usr/share/nginx/html

**name**: nginx-storage

**volumes**:

**- name**: nginx-storage

**persistentVolumeClaim**:

**claimName**: nginx-pvc

Για να δημιουργήσετε το Nginx Pod στον cluster σας, εκτελέστε την παρακάτω εντολή:

kubectl apply -f nginx-pod.yaml

Αυτή η εντολή δημιουργεί το Pod με όνομα nginx-pod, το οποίο χρησιμοποιεί το PVC για να αποθηκεύσει δεδομένα στον κατάλογο /usr/share/nginx/html.

Αντιγράψτε το αρχείο index.html στο Persistent Volume στον κατάλογο /usr/share/nginx/html του Pod. Αυτό μπορεί να γίνει χρησιμοποιώντας το kubectl cp

kubectl cp index.html nginx-pod:/usr/share/nginx/html/index.html

Αυτή η εντολή μεταφέρει από το τοπικό σύστημα αρχείων του χρήστη το αρχείο index.html και το τοποθετεί στον κατάλογο /usr/share/nginx/html του nginx-pod. Σε αυτόν τον κατάλογο έχει προσαρτηθεί ένας μόνιμος τόμος αρχείων (persistent volume).

Αν τώρα επισκεφτείτε την IP του Nginx Pod θα μπορείτε να δείτε την παρακάτω σελίδα.

Τρέξτε την εντολή

kubectl get pod nginx-pod -o wide

και ανοίξτε στον browser την σελίδα http://<podip>

Ακόμα και αν το Pod τερματιστεί και δημιουργηθεί ξανά, το αρχείο index.html θα είναι διαθέσιμο και ο Nginx θα το σερβίρει. Ας καταστρέψουμε το pod

kubectl delete pod nginx-pod

Ας το ξαναδημιουρήσουμε:

kubectl apply -f nginx-pod.yaml

Ας πάρουμε πάλι το νέο IP που αποδώθηκε στο νέο pod:

kubectl get pod nginx-pod -o wide

Και ας ανοίξουμε πάλι τον browser στη νέα διεθύνση: http://<podip>

Βλέπουμε ότι η σελίδα **παραμένει και μετά την καταστροφή του pod**!!!!

* **Δημιουργία Nginx Pod χωρίς PVC και (Εφήμερη Αποθήκευση)**

Τώρα, ας δημιουργήσουμε ένα Nginx Pod που χρησιμοποιεί emptyDir, που σημαίνει ότι τα δεδομένα θα χάνονται όταν το Pod τερματίσει.

Δημιουργούμε το αρχείο YAML για το Nginx Pod που δεν έχει persistent storage:

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: nginx-pod-ephemeral

**spec**:

**containers**:

**- name**: nginx

**image**: nginx:latest

Για να δημιουργήσετε το Pod, εκτελέστε την εντολή:

kubectl apply -f nginx-pod-ephemeral.yaml

Αντιγράφουμε το ίδιο αρχείο index.html στον κατάλογο /usr/share/nginx/html του νέου Pod:

kubectl cp index.html nginx-pod-ephemeral:/usr/share/nginx/html/index.html

Παίρνουμε το ip του pod για να ανοίξουμε την σελίδα του στον browser

kubectl get pod nginx-pod-ephemeral -o wide

Αν επισκεφτείτε την σελίδα στο browser σας, θα δείτε την σελίδα που ανεβάσατε.

Τώρα, για να δείτε τη διαφορά, τερματίστε το Pod:

kubectl delete pod nginx-pod-ephemeral

Ξαναδημιουργήστε το Pod:

kubectl apply -f nginx-pod-ephemeral.yaml

Αν επισκεφτείτε ξανά τη σελίδα στον browser σας, θα δείτε την default σελίδα ενός nginx pod (Welcome to nginx!) διότι τα δεδομένα (το αρχείο index.html) έχουν χαθεί λόγω του ότι το σύστημα αρχείων του pod είναι εφήμερο και τα δεδομένα διαγράφονται όταν το Pod τερματίζεται

# ReplicaSets (Αντίγραφα)

* Το **ReplicaSet** είναι ένα αντικείμενο ελέγχου στο Kubernetes που διασφαλίζει ότι ένα καθορισμένο πλήθος αντιγράφων (Pods) μιας εφαρμογής εκτελούνται πάντα στο cluster. Αν ένα Pod αποτύχει ή τερματιστεί, το ReplicaSet δημιουργεί αυτόματα ένα νέο για να διατηρήσει την επιθυμητή κατάσταση σύμφωνα με την πρόθεση (intent) του χρήστη.
  + **Λειτουργία και Χαρακτηριστικά**
  + **Διατήρηση επιθυμητού αριθμού Pods**: Αν κάποιο Pod διαγραφεί ή αποτύχει, το ReplicaSet το αντικαθιστά.
  + **Χρήση Label Selectors**: Τα ReplicaSets χρησιμοποιούν label selectors για να εντοπίσουν ποια Pods πρέπει να διαχειριστούν.
  + **Αυτόματη Κλιμάκωση**: Μπορούμε να προσαρμόσουμε το μέγεθος του ReplicaSet αυξάνοντας ή μειώνοντας τον αριθμό των Pods.
  + **Μέρος ενός Deployment**: Συνήθως, τα ReplicaSets χρησιμοποιούνται μέσω των Deployments για πιο ευέλικτη διαχείριση εφαρμογών.
* **Παράδειγμα YAML Αρχείου**

Παρακάτω δίνεται ένα παράδειγμα ορισμού ενός ReplicaSet που διατηρεί 3 αντίγραφα ενός Pod:

**apiVersion**: apps/v1

**kind**: ReplicaSet

**metadata**:

**name**: my-replicaset

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:latest

* **Εντολές Διαχείρισης ReplicaSets**
* **Δημιουργία ReplicaSet**:

kubectl apply -f my-replicaset.yaml

* **Έλεγχος κατάστασης**:

kubectl get replicaset

* **Κλιμάκωση (Scaling) των Pods**:

kubectl scale --replicas=5 rs/my-replicaset

* **Διαγραφή ReplicaSet** (προσοχή: τα Pods θα διαγραφούν επίσης):

kubectl delete rs my-replicaset

**Βέλτιστες Πρακτικές – Σύνοψη**

* **Χρήση Deployments αντί για ReplicaSets:** Αν και μπορούμε να χρησιμοποιήσουμε ReplicaSets ανεξάρτητα, τα Deployments παρέχουν επιπλέον δυνατότητες όπως rolling updates και rollbacks.
* **Ορισμός κατάλληλων label selectors:** Πρέπει να βεβαιωνόμαστε ότι το ReplicaSet διαχειρίζεται τα σωστά Pods.
* **Monitoring & Logging:** Χρησιμοποιήστε εργαλεία όπως kubectl describe rs και kubectl logs για την παρακολούθηση της κατάστασης.
* **Σύνοψη:** Το ReplicaSet είναι ένας βασικός μηχανισμός του Kubernetes για τη διατήρηση της επιθυμητής κατάστασης των Pods. Ωστόσο, στις περισσότερες περιπτώσεις, είναι προτιμότερο να χρησιμοποιούμε Deployments για μεγαλύτερη ευελιξία και ευκολία στη διαχείριση εφαρμογών.

# Deployments στο Kubernetes

* **Εισαγωγή στα Deployments:** Τα Deployments στο Kubernetes αποτελούν έναν ανώτερο μηχανισμό διαχείρισης Pods σε σύγκριση με τα ReplicaSets. Παρέχουν δυνατότητες όπως ελεγχόμενες ενημερώσεις, rollback σε προηγούμενες εκδόσεις και αυτοματοποίηση της διαδικασίας ανάπτυξης εφαρμογών.
* **Διαφορές μεταξύ Deployments και ReplicaSets**

Τα ReplicaSets διαχειρίζονται μόνο τον αριθμό των Pods που πρέπει να τρέχουν, ενώ τα Deployments προσφέρουν ένα επιπλέον επίπεδο αφαίρεσης, επιτρέποντας τη διαχείριση ενημερώσεων εφαρμογών. Συγκεκριμένα:

* + **ReplicaSets** διασφαλίζουν ότι ένας καθορισμένος αριθμός Pods είναι πάντα διαθέσιμος.
  + **Deployments** δημιουργούν και διαχειρίζονται ReplicaSets, επιτρέποντας αναβαθμίσεις έκδοσης και rollback σε προηγούμενες εκδόσεις.
  + Με ένα Deployment μπορούμε να αλλάζουμε την εικόνα του container χωρίς να χρειάζεται χειροκίνητη διαχείριση των ReplicaSets.
* **Δημιουργία Deployment**

Παράδειγμα αρχείου YAML για ένα Deployment:

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: my-deployment

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:latest

**ports**:

**- containerPort**: 80

Για να εκτελέσετε το παραπάνω deployment:

kubectl apply -f deployment.yaml

Μετά την εκτέλεση, μπορείτε να ελέγξετε την κατάσταση των Pods:

kubectl get pods -o wide

Θα δείτε τρία Pods με το label app=my-app να εκτελούνται.

* **Πρόσθετες Χρήσεις των Deployments**

Τα Deployments μπορούν να χρησιμοποιηθούν για διάφορους σκοπούς πέρα από την απλή εκκίνηση εφαρμογών:

* + **Rolling Updates:** Επιτρέπουν σταδιακή αναβάθμιση της εφαρμογής χωρίς διακοπή της υπηρεσίας. Δημιουργήστε το παρακάτω αρχείο myrolling\_deployment.yaml με τα παρακάτω

Για να κάνετε μια σταδιακή μετάβαση από την παλιά έκδοση του Nginx σε μια νέα έκδοση, απλά πρέπει να τροποποιήσετε την παράμετρο image στο Deployment. Ακολουθεί ένα παράδειγμα όπου η παλιά έκδοση του Nginx είναι η nginx:1.14 και θα αναβαθμιστεί στη νέα έκδοση nginx:latest με σταδιακή ενημέρωση μέσω rolling update.

**Παράδειγμα YAML με Rolling Update (Αναβάθμιση από παλιά σε νέα έκδοση του Nginx):**

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: my-deployment

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:1.14 # Παλιά έκδοση του Nginx

**ports**:

**- containerPort**: 80

**strategy**:

**type**: RollingUpdate

**rollingUpdate**:

**maxSurge**: 1 # Μέγιστος αριθμός επιπλέον pods που μπορούν να δημιουργηθούν κατά τη διάρκεια του update

**maxUnavailable**: 1 # Μέγιστος αριθμός pods που μπορούν να είναι μη διαθέσιμα κατά την ενημέρωση

**Εκτέλεση:**

kubectl apply -f myrolling\_deployment.yaml

* **Αποτέλεσμα:** Εγκαθίσταται το deployment με παλιά έκδοση nginx.
* **Βήματα για την Αναβάθμιση σε Νέα Έκδοση (nginx:latest):**

Αρχικά, η παλιά έκδοση (nginx:1.14) είναι σε λειτουργία.

**Τροποποιούμε την έκδοση του Nginx στο YAML** myrolling\_deployment.yaml **ώστε να χρησιμοποιεί την τελευταία έκδοση (**nginx:latest**)**:

**image**: nginx:latest # Νέα έκδοση του Nginx

* **Ανανεώνουμε το Deployment**: Αφού κάνετε αυτή την αλλαγή, θα πρέπει να ενημερώσετε το Deployment χρησιμοποιώντας την εντολή kubectl apply.

kubectl apply -f myrolling\_deployment.yaml

Το Kubernetes θα πραγματοποιήσει μια σταδιακή μετάβαση από την παλιά έκδοση nginx:1.14 στην νέα έκδοση nginx:latest χρησιμοποιώντας rolling update, δημιουργώντας νέα pods με τη νέα έκδοση και καταργώντας σταδιακά τα παλιά pods.

* **Παρακολούθηση της ενημέρωσης:**

Μπορείτε να παρακολουθήσετε την πρόοδο της ενημέρωσης με την εντολή:

kubectl rollout status deployment/my-deployment

* + **Rollback σε προηγούμενη έκδοση:** Επιστροφή σε προηγούμενο ReplicaSet αν εντοπιστεί πρόβλημα.

kubectl rollout undo deployment my-deployment

**Αποτέλεσμα:** Το Deployment θα επιστρέψει στην προηγούμενη σταθερή έκδοση.

**Με την παρακάτω εντολή βλέπουμε τις διαφορετικές εκδόσεις του deployment:**

kubectl rollout history deployment my-deployment

* + **Blue-Green Deployments:** Ταυτόχρονη ύπαρξη δύο εκδόσεων μιας εφαρμογής για γρήγορη μετάβαση μεταξύ τους.

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: blue-deployment

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:1.14

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: green-deployment

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:latest

**Εκτέλεση:**

kubectl apply -f blue-deployment.yaml

kubectl apply -f green-deployment.yaml

**Αποτέλεσμα:** Και οι δύο εκδόσεις θα εκτελούνται ταυτόχρονα, επιτρέποντας γρήγορη μετάβαση μέσω αλλαγής του Service selector.

Με αυτά τα χαρακτηριστικά, τα Deployments παρέχουν μια ευέλικτη και αξιόπιστη μέθοδο διαχείρισης εφαρμογών στο Kubernetes.

# StatefulSets στο Kubernetes

* **Εισαγωγή στα StatefulSets:** Τα StatefulSets στο Kubernetes χρησιμοποιούνται για τη διαχείριση εφαρμογών που απαιτούν διατήρηση της κατάστασης (stateful applications). Σε αντίθεση με τα Deployments και τα ReplicaSets, τα StatefulSets εγγυώνται σταθερή ταυτότητα (σταθερά ονόματα Pods) και διατηρούν την αποθήκευση δεδομένων ακόμα και μετά την επανεκκίνηση των Pods (persistent volumes).
* **Διαφορές μεταξύ StatefulSets και Deployments**
  + **Σταθερά Ονόματα Pods:** Τα Pods σε ένα StatefulSet λαμβάνουν σταθερά ονόματα με αύξουσα αρίθμηση (π.χ. my-app-0, my-app-1).
  + **Σταθεροί Persistent Volumes:** Κάθε Pod έχει τον δικό του volume (τόμο), ο οποίος δεν διαγράφεται αυτόματα όταν το Pod τερματιστεί.
  + **Τακτική Εκκίνηση & Τερματισμός:** Τα Pods εκκινούνται και τερματίζονται διαδοχικά, με συγκεκριμένη σειρά.
* **Δημιουργία StatefulSet**
* **Παράδειγμα αρχείου YAML για ένα StatefulSet:**

**apiVersion**: apps/v1

**kind**: StatefulSet

**metadata**:

**name**: my-statefulset

**spec**:

**serviceName**: "my-service"

**replicas**: 3

**selector**:

**matchLabels**:

**app**: my-app

**template**:

**metadata**:

**labels**:

**app**: my-app

**spec**:

**containers**:

**- name**: my-container

**image**: nginx:latest

**volumeMounts**:

**- name**: my-volume

**mountPath**: /data

**volumeClaimTemplates**:

**- metadata**:

**name**: my-volume

**spec**:

**accessModes**: [ "ReadWriteOnce" ]

**resources**:

**requests**:

**storage**: 1Gi

* **Εκτέλεση του StatefulSet**

kubectl apply -f my-statefulset.yaml

* Μετά την εκτέλεση της εντολής, μπορείτε να δείτε τα δημιουργημένα Pods με την ακόλουθη εντολή:

kubectl get pods

* Θα παρατηρήσετε ότι τα Pods έχουν ονόματα όπως my-statefulset-0, my-statefulset-1, my-statefulset-2 και εκκινούνται διαδοχικά.
* Για να δείτε τα Persistent Volumes που έχουν δημιουργηθεί:

kubectl get pvc

* Θα παρατηρήσετε ότι κάθε Pod έχει το δικό του Persistent Volume (my-volume-my-statefulset-0, my-volume-my-statefulset-1, κ.ο.κ.), το οποίο είναι μόνιμα συνδεδεμένο στο αντίστοιχο Pod ακόμα και μετά τον τερματισμό του.
* **Επαλήθευση διατήρησης δεδομένων στα Persistent Volumes:** Για να δοκιμάσουμε αν ένα Persistent Volume διατηρεί τα δεδομένα του ακόμα και μετά τον τερματισμό του Pod, εκτελέστε τις εξής εντολές:

kubectl exec -it my-statefulset-0 -- /bin/sh

* Μέσα στο Pod, δημιουργήστε ένα αρχείο στον mounted volume:

echo "Hello Kubernetes" > /data/testfile.txt

exit

* Τώρα, διαγράψτε το Pod:

kubectl delete pod my-statefulset-0

* Μόλις το Pod ξαναδημιουργηθεί, επανασυνδεθείτε:

kubectl exec -it my-statefulset-0 -- /bin/sh

* και ελέγξτε αν το αρχείο υπάρχει ακόμα:

cat /data/testfile.txt

* Αν η έξοδος είναι Hello Kubernetes, τότε το Persistent Volume έχει διατηρήσει τα δεδομένα του, αποδεικνύοντας ότι το volume παραμένει συνδεδεμένο στο ίδιο Pod ακόμα και μετά την επανεκκίνηση.
* Πρόσθετες Χρήσεις των StatefulSets: Τα StatefulSets είναι κατάλληλα για εφαρμογές που απαιτούν συγκεκριμένη σειρά εκκίνησης και σταθερή αποθήκευση, όπως:
  + **Βάσεις Δεδομένων** (π.χ. MySQL, PostgreSQL, MongoDB)
  + **Distributed Systems** (π.χ. Kafka, Zookeeper, Elasticsearch)
  + **Αποθήκευση αρχείων**
* Με την κατάλληλη διαχείριση, τα StatefulSets παρέχουν μια αξιόπιστη λύση για την ανάπτυξη stateful εφαρμογών (εφαρμογών με διατήρηση κατάστασης) στο Kubernetes.

# DaemonSets στο Kubernetes

* **Εισαγωγή στα DaemonSets:** Τα DaemonSets στο Kubernetes χρησιμοποιούνται για την εκτέλεση Pods σε **κάθε κόμβο** **(node)** ενός cluster. Είναι χρήσιμα για εφαρμογές όπως συλλογή logs, monitoring agents και network services που πρέπει να εκτελούνται σε όλους τους κόμβους.
* **Διαφορές μεταξύ DaemonSets και Deployments**
  + **Εκτέλεση σε κάθε Κόμβο:** Σε αντίθεση με τα Deployments, τα DaemonSets διασφαλίζουν ότι κάθε κόμβος έχει ένα αντίγραφο του Pod.
  + **Αυτόματη Προσθήκη σε Νέους Κόμβους:** Όταν ένας νέος κόμβος προστεθεί στο cluster, το DaemonSet θα εκτελέσει αυτόματα ένα Pod σε αυτόν.
  + **Δεν χρησιμοποιεί ReplicaSets:** Τα DaemonSets δεν εξαρτώνται από ReplicaSets, καθώς ο αριθμός των Pods βασίζεται μόνο στον αριθμό των κόμβων.
* **Δημιουργία ενός DaemonSet:** Παράδειγμα αρχείου YAML για ένα DaemonSet με Fluentd:

**apiVersion**: apps/v1

**kind**: DaemonSet

**metadata**:

**name**: fluentd-daemonset

**spec**:

**selector**:

**matchLabels**:

**name**: fluentd-pod

**template**:

**metadata**:

**labels**:

**name**: fluentd-pod

**spec**:

**containers**:

**- name**: fluentd

**image**: fluent/fluentd:v1.14-1

**resources**:

**limits**:

**memory**: 200Mi

**cpu**: 100m

**volumeMounts**:

**- name**: varlog

**mountPath**: /var/log

**volumes**:

**- name**: varlog

**hostPath**:

**path**: /var/log

* **Εκτέλεση του DaemonSet:** Για να εκτελέσετε το DaemonSet, χρησιμοποιήστε την παρακάτω εντολή

kubectl apply -f fluentd-daemonset.yaml

* + Μετά την εκτέλεση, μπορείτε να δείτε τα Pods του DaemonSet σε όλους τους κόμβους:

kubectl get pods -o wide

* + Θα παρατηρήσετε ότι υπάρχει ένα Fluentd Pod σε κάθε worker node του cluster.
* **Συλλογή και εκτύπωση μετρικών με Fluentd**

Μπορούμε να χρησιμοποιήσουμε το Fluentd για τη συλλογή μετρικών από όλα τα Pods και τους κόμβους του cluster.

* + **Προβολή logs από όλα τα Fluentd Pods**

Για να δείτε τα logs που συλλέγονται από όλους τους κόμβους μέσω του Fluentd, χρησιμοποιήστε:

kubectl logs -l name=fluentd-pod --all-containers=true --tail=50

Αυτή η εντολή εμφανίζει τα τελευταία 50 logs από όλα τα Fluentd Pods του cluster.

* **Πρόσθετες Χρήσεις των DaemonSets**

Τα DaemonSets είναι χρήσιμα για πολλές περιπτώσεις, όπως:

* + **Συλλογή logs** με Fluentd, Logstash ή άλλους agents.
  + **Monitoring** με Prometheus Node Exporter ή Datadog Agent.
  + **Network Services** όπως CNI plugins (Calico, Flannel) για δικτύωση στο cluster.

Με τα DaemonSets, μπορούμε να διασφαλίσουμε ότι όλες οι κρίσιμες υπηρεσίες εκτελούνται ομοιόμορφα σε όλους τους κόμβους του cluster.

# Secrets

Τα **Secrets** στο Kubernetes χρησιμοποιούνται για την ασφαλή αποθήκευση ευαίσθητων δεδομένων, όπως κωδικοί πρόσβασης ή API tokens. Σε αυτό το παράδειγμα, θα δούμε πώς μπορούμε να χρησιμοποιήσουμε ένα Secret για να παρέχουμε στοιχεία σύνδεσης σε μια βάση δεδομένων, κάνοντάς τα διαθέσιμα στην εφαρμογή ως αρχεία μέσω ενός volume.

Περιηγηθείτε στον κατάλογο του παραδείγματος

cd 09\_Secrets

**Βήμα 1: Δημιουργία του Secret**

Δημιουργούμε ένα Secret που περιέχει τα credentials (για παράδειγμα, για μια βάση δεδομένων):

**apiVersion**: v1

**kind**: Secret

**metadata**:

**name**: db-credentials

**type**: Opaque

**data**:

**username**: cG9zdGdyZXM= # "postgres" σε base64

**password**: c3VwZXJzZWNyZXQ= # "supersecret" σε base64

Για να δημιουργήσεις το Secret:

kubectl apply -f db-credentials.yaml

Για να δεις τα διαθέσιμα secrets:

kubectl get secrets

**Βήμα 2: Χρήση του Secret μέσω Volume**

Ακολουθεί ένα παράδειγμα Pod, όπου το Secret γίνεται mount ως volume. Η εφαρμογή μπορεί να διαβάζει τα credentials από τα αρχεία /etc/db-credentials/username και /etc/db-credentials/password:

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: app-with-db

**spec**:

**containers**:

**- name**: my-app

**image**: my-app-image

**volumeMounts**:

**- name**: secret-volume

**mountPath**: "/etc/db-credentials"

**readOnly**: **true**

**volumes**:

**- name**: secret-volume

**secret**:

**secretName**: db-credentials

Το Kubernetes θα δημιουργήσει αυτόματα αρχεία username και password μέσα στον κατάλογο /etc/db-credentials, τα οποία η εφαρμογή μπορεί να διαβάσει ως απλά αρχεία κειμένου.

Για να δημιουργήσεις το Pod:

kubectl apply -f app-with-db.yaml

Μέσω k9s πάρε κονσόλα (πάτα s στο pod που δημιουργήθηκε) και τρέξε:

root@app-with-db:/# cat /etc/db-credentials/username; echo

postgres

root@app-with-db:/# cat /etc/db-credentials/password; echo

supersecret

όπου μπορείς να δεις τα στοιχεία που περάσανε από το k8s στο pod και είναι διαθέσιμα. Πάτα Ctrl+d για να αποσυνδεθείς από το pod και μετά Ctrl+c για να αποσυνδεθείς από το k9s.

Για να καθαρίσετε το pod και τα secrets και να προχωρήσετε στο παρακάτω παράδειγμα, τρέξτε

kubectl delete -f .

Προσοχή: φροντίστε να βρίσκεστε τον κατάλογο όταν εκτελέσετε την εντολή delete

Κατόπιν, βγείτε από τον κατάλογο.

cd ..

# ConfigMaps

Τα **ConfigMaps** χρησιμοποιούνται στο Kubernetes για την αποθήκευση μη-ευαίσθητων παραμέτρων διαμόρφωσης, όπως το όνομα χρήστη, οι διευθύνσεις υπηρεσιών, τα URLs ή οποιαδήποτε ρύθμιση της εφαρμογής που δεν θεωρείται μυστική.

Σε αυτό το παράδειγμα, θα δούμε πώς χρησιμοποιούμε ένα ConfigMap για να αποθηκεύσουμε το όνομα χρήστη της βάσης δεδομένων και να το διαθέσουμε στην εφαρμογή ως αρχείο μέσω volume (mount).

Περιηγηθείτε στον κατάλογο του παραδείγματος

cd 10\_ConfigMaps

**Βήμα 1: Δημιουργία του ConfigMap**

**apiVersion**: v1

**kind**: ConfigMap

**metadata**:

**name**: db-config

**data**:

**username**: postgres

**host**: db-service

Για να δημιουργήσεις το ConfigMap:

kubectl apply -f db-configmap.yaml

Για να δεις τα διαθέσιμα configmaps:

kubectl get configmaps

**Βήμα 2: Χρήση του ConfigMap μέσω Volume**

Ακολουθεί παράδειγμα Pod όπου το ConfigMap γίνεται mount ως volume, επιτρέποντας στην εφαρμογή να διαβάσει το username και το host της βάσης από αρχεία:

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: app-with-configmap

**spec**:

**containers**:

**- name**: my-app

**image**: nginx

**volumeMounts**:

**- name**: config-volume

**mountPath**: "/etc/db-config"

**readOnly**: **true**

**volumes**:

**- name**: config-volume

**configMap**:

**name**: db-config

Με αυτή τη ρύθμιση, η εφαρμογή μπορεί να διαβάσει:

* /etc/db-config/username για το username (π.χ. postgres)
* /etc/db-config/host για το host της βάσης

Για να δημιουργήσεις το Pod:

kubectl apply -f pod-with-configmap.yaml

Μέσω k9s πάρε κονσόλα (πάτα s στο pod που δημιουργήθηκε) και τρέξε:

root@app-with-configmap:/# cat /etc/db-config/username; echo

postgres

root@app-with-configmap:/# cat /etc/db-config/host; echo

db-service

root@app-with-configmap:/#

όπου μπορείς να δεις τα στοιχεία που περάσανε από το k8s στο pod και είναι διαθέσιμα. Πάτα Ctrl+d για να αποσυνδεθείς από το pod και μετά Ctrl+c για να αποσυνδεθείς από το k9s.

Για να καθαρίσετε το pod και τα secrets και να προχωρήσετε στο παρακάτω παράδειγμα, τρέξτε

kubectl delete -f .

**Προσοχή:** φροντίστε να βρίσκεστε τον κατάλογο όταν εκτελέσετε την εντολή delete

Κατόπιν, βγείτε από τον κατάλογο.

cd ..

# ConfigMap και Secrets για εγκατάσταση Web Server με Postgres.

**Σύντομη Περιγραφή:**

Ακολουθεί ένα ένα πλήρες παράδειγμα Kubernetes με τα εξής:

✅ **PostgreSQL Pod**

* Με **Persistent Volume** μέσω PVC
* Παίρνει τις ρυθμίσεις σύνδεσης από **Secret** (password) και **ConfigMap** (username, database name)

✅ **Web Server Pod** (ένα απλό PHP)

* Συνδέεται στη βάση
* Παίρνει config από τα ίδια Secrets & ConfigMap
* Εμφανίζει σε μια σελίδα δεδομένα από πίνακα της βάσης

**Ο PostgreSQL container** θα τρέχει ένα **init SQL script** που δημιουργεί τη βάση και τον πίνακα με δείγμα δεδομένων.

**Ο Web Server container** θα είναι έτοιμος, βασισμένος σε **php:apache**, με ένα index.php που διαβάζει από τη βάση.

**ℹ️ Σημαντικές Διευκρινίσεις - Συνοπτικά**

**🔐 Αρχικοποίηση χρήστη και βάσης στην PostgreSQL**

Ορίζεται μέσω:

* POSTGRES\_USER (από ConfigMap)
* POSTGRES\_DB (από ConfigMap)
* POSTGRES\_PASSWORD (από Secret)

Οι μεταβλητές αυτές ισχύουν **μόνο κατά την πρώτη εκκίνηση**, όταν το volume είναι **άδειο**. Αν υπάρχει ήδη περιεχόμενο στο PVC, δεν αλλάζουν τον υπάρχοντα χρήστη ή τον κωδικό.

**🗃️ Δημιουργία Πίνακα και Δεδομένων**

Το αρχείο init.sql εκτελείται αυτόματα στην πρώτη εκκίνηση μέσω του path /docker-entrypoint-initdb.d/init.sql. Δημιουργεί τον πίνακα my\_table και εισάγει δείγμα δεδομένων.

**ℹ️ Σημαντικές Διευκρινίσεις - Αναλυτικά**

**🔐 Πώς αρχικοποιείται ο χρήστης στη βάση PostgreSQL**

Στο παράδειγμα που σου έδωσα, ο **κωδικός του χρήστη** στην PostgreSQL αρχικοποιείται **μέσω των μεταβλητών περιβάλλοντος** που διαβάζονται από το Secret και το ConfigMap κατά την εκκίνηση του PostgreSQL container.

**📌 Συγκεκριμένα:**

Ο container postgres:15 (όπως και όλες οι επίσημες PostgreSQL εικόνες) υποστηρίζει τα εξής env vars:

| **Μεταβλητή** | **Περιγραφή** |
| --- | --- |
| POSTGRES\_USER | Όνομα του χρήστη (π.χ. postgres) |
| POSTGRES\_DB | Όνομα της βάσης (π.χ. myappdb) |
| POSTGRES\_PASSWORD | Κωδικός του χρήστη |

Αυτές οι τιμές προέρχονται από:

* 02-configmap.yaml → username, dbname
* 01-secret.yaml → password

Αν το PersistentVolumeClaim είναι **καινούργιο** (άδειος δίσκος), ο PostgreSQL container:

* Δημιουργεί τον χρήστη και τη βάση
* Θέτει τον κωδικό με βάση το POSTGRES\_PASSWORD

⚠️ Αν το PVC έχει ήδη δεδομένα (π.χ. από προηγούμενη εκκίνηση), οι μεταβλητές **δεν επηρεάζουν** τα υπάρχοντα credentials. Σε αυτή την περίπτωση, πρέπει να αλλάξεις τον κωδικό από μέσα με SQL (ALTER USER) ή να διαγράψεις το PVC (και τα δεδομένα του).

**🔁 Πώς γίνεται στην πράξη:**

**env:**

**- name: POSTGRES\_USER**

**valueFrom:**

**configMapKeyRef:**

**name: db-config**

**key: username**

**- name: POSTGRES\_DB**

**valueFrom:**

**configMapKeyRef:**

**name: db-config**

**key: dbname**

**- name: POSTGRES\_PASSWORD**

**valueFrom:**

**secretKeyRef:**

**name: db-secret**

**key: password**

Αυτό σημαίνει:

* Ο χρήστης postgres θα δημιουργηθεί (αν δεν υπάρχει ήδη)
* Η βάση myappdb θα δημιουργηθεί
* Ο χρήστης postgres θα έχει τον κωδικό supersecret (το Secret db-secret)

💡 Όλα αυτά γίνονται κατά την **πρώτη εκκίνηση του container**, όταν ο κατάλογος /var/lib/postgresql/data είναι **άδειος** (δηλαδή για νέο PVC).

**🧩 Δημιουργία πίνακα και αρχικών δεδομένων**

Ο πίνακας my\_table δημιουργείται αυτόματα με SQL script (init.sql) που φορτώνεται μέσω ConfigMap και προσαρτάται (mount) στον PostgreSQL container στον φάκελο /docker-entrypoint-initdb.d.

Η εντολή CREATE TABLE είναι με IF NOT EXISTS, οπότε η διαδικασία είναι **idempotent** και ασφαλής για επανεκκινήσεις.

Αν έχεις ήδη τρέξει τον container και ο τόμος δεδομένων (PVC) κρατάει προηγούμενα δεδομένα, τότε αυτές οι μεταβλητές **δεν έχουν πλέον αποτέλεσμα**, γιατί η βάση έχει ήδη αρχικοποιηθεί.

**Αναλυτικά Βήματα**

**🔧 Βήμα 1: Secret για τον κωδικό της βάσης**

kubectl apply -f 01-secret.yaml

🔐 Δημιουργεί το Secret με το password της PostgreSQL (supersecret σε base64).

**🧩 Βήμα 2: ConfigMap με ρυθμίσεις βάσης**

kubectl apply -f 02-configmap.yaml

📋 Περιλαμβάνει: username, dbname, host.

**💾 Βήμα 3: ConfigMap με SQL αρχικοποίησης**

kubectl apply -f init-sql-configmap.yaml

📄 Περιέχει SQL εντολές για τη δημιουργία πίνακα my\_table και εισαγωγή δεδομένων (Alice, Bob, Charlie).

**✅ Βήμα 4 – Persistent Volume Claim για βάση**

kubectl apply -f 03-pvc.yaml

**🐘 Βήμα 5: PostgreSQL Pod**

kubectl apply -f 04-postgres.yaml

💾 Δημιουργεί χώρο 1Gi για αποθήκευση δεδομένων της PostgreSQL.

**✅ Βήμα 6 – Service για τη βάση**

kubectl apply -f 05-postgres-service.yaml

🌐 Επιτρέπει στα υπόλοιπα pods να βρίσκουν τη βάση μέσω DNS:

postgres.ikons-priv.svc.cluster.local

**✅ Βήμα 7 – Web εφαρμογή (PHP μέσω ConfigMap)**

kubectl apply -f 06-web-content-configmap.yaml -n ikons-priv

📄 Δημιουργεί index.php μέσω ConfigMap, ο οποίος διαβάζει δεδομένα από τη βάση.

**🌐 Βήμα 8: Web Server Pod**

kubectl apply -f 07-webserver.yaml

Εκκινεί το Web server με το image webdevops/php-apache:8.1, το οποίο περιλαμβάνει pgsql extension.

**✅ Βήμα 9 – Service για πρόσβαση στον Web Server**

kubectl apply -f 08-webserver-service.yaml

Πλέον, έχει σηκωθεί το pod και το service είναι διαθέσιμο, μπορείτε να ανοίξετε την παρακάτω σελίδα:

<http://webserver-service.ikons-priv.svc.cluster.local/>

Ή μπορείτε να τρέξετε από το command line την παρακάτω εντολή:

curl http://webserver-service.ikons-priv.svc.cluster.local

και θα δείτε ένα αποτέλεσμα σαν το παρακάτω:

ikons@mylaptop:~/cloud/k8s-web-pgsql-demo-final$ curl http://webserver-service.ikons-priv.svc.cluster.local

<h1>Records from DB</h1><p>Alice</p><p>Bob</p><p>Charlie</p>

**📦 Μαζική εγκατάσταση μέσω MakeFile:**

Για να τρέξετε όλα τα βήματα μαζί, μπορείτε να χρησιμοποιήσετε το Makefile που υπάρχει στον κατάλογο. Το makefile έχει δυο actions, την deploy και την clean.

make deploy

**🧼 Μαζική Απεγκατάσταση (Cleanup)**

make clean

# Nginx και πολλαπλοί web servers σε Kubernetes

Κάθε χρήστης έχει πρόσβαση στο δικό του namespace της μορφής <username>-priv

* Βήμα 1: Ορισμός Web Servers (Deployments & Services)

**Αρχείο** web-deployments.yaml

Δημιουργούμε δύο web servers που θα σερβίρουν απλές HTML σελίδες.

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: web1

**namespace**: <username>-priv

**spec**:

**replicas**: 1

**selector**:

**matchLabels**:

**app**: web1

**template**:

**metadata**:

**labels**:

**app**: web1

**spec**:

**containers**:

**- name**: web1

**image**: nginx

**volumeMounts**:

**- name**: web-content

**mountPath**: /usr/share/nginx/html

**volumes**:

**- name**: web-content

**configMap**:

**name**: web1-html

---

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: web2

**namespace**: <username>-priv

**spec**:

**replicas**: 1

**selector**:

**matchLabels**:

**app**: web2

**template**:

**metadata**:

**labels**:

**app**: web2

**spec**:

**containers**:

**- name**: web2

**image**: nginx

**volumeMounts**:

**- name**: web-content

**mountPath**: /usr/share/nginx/html

**volumes**:

**- name**: web-content

**configMap**:

**name**: web2-html

**Αρχείο** web-services.yaml

Ορίζουμε τα Services για να προσφέρουμε πρόσβαση στους web servers.

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: web1

**namespace**: <username>-priv

**spec**:

**selector**:

**app**: web1

**ports**:

**- protocol**: TCP

**port**: 80

**targetPort**: 80

---

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: web2

**namespace**: <username>-priv

**spec**:

**selector**:

**app**: web2

**ports**:

**- protocol**: TCP

**port**: 80

**targetPort**: 80

* Βήμα 2: Προσθήκη Περιεχομένου στις Σελίδες (ConfigMaps)

**Αρχείο** web-configmaps.yaml

Χρησιμοποιούμε ConfigMaps για το περιεχόμενο των HTML σελίδων.

**apiVersion**: v1

**kind**: ConfigMap

**metadata**:

**name**: web1-html

**namespace**: <username>-priv

**data**:

**index.html**: |

**<h1>Welcome to Web1</h1>**

---

**apiVersion**: v1

**kind**: ConfigMap

**metadata**:

**name**: web2-html

**namespace**: <username>-priv

**data**:

**index.html**: |

**<h1>Welcome to Web2</h1>**

* Βήμα 3: Ορισμός του Nginx Reverse Proxy

**Αρχείο** nginx-configmap.yaml

Ορίζουμε το configuration του Nginx ως reverse proxy.

**apiVersion**: v1

**kind**: ConfigMap

**metadata**:

**name**: nginx-config

**namespace**: <username>-priv

**data**:

**nginx.conf**: |

**events {}**

http {

upstream backend {

server web1;

server web2;

}

server {

listen 80;

location / {

proxy\_pass http://backend;

}

}

}

Αρχείο nginx-deployment.yaml

Ορίζουμε το Deployment και Service για τον Nginx proxy.

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: nginx-proxy

**namespace**: <username>-priv

**spec**:

**replicas**: 1

**selector**:

**matchLabels**:

**app**: nginx-proxy

**template**:

**metadata**:

**labels**:

**app**: nginx-proxy

**spec**:

**containers**:

**- name**: nginx

**image**: nginx

**ports**:

**- containerPort**: 80

**volumeMounts**:

**- name**: config-volume

**mountPath**: /etc/nginx/nginx.conf

**subPath**: nginx.conf

**volumes**:

**- name**: config-volume

**configMap**:

**name**: nginx-config

---

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: nginx-service

**namespace**: <username>-priv

**spec**:

**type**: LoadBalancer

**selector**:

**app**: nginx-proxy

**ports**:

**- protocol**: TCP

**port**: 80

**targetPort**: 80

* Βήμα 4: Ανάπτυξη στο Kubernetes

kubectl apply -f web-configmaps.yaml

kubectl apply -f web-deployments.yaml

kubectl apply -f web-services.yaml

kubectl apply -f nginx-configmap.yaml

kubectl apply -f nginx-deployment.yaml

* Βήμα 5: Πρόσβαση στην Εφαρμογή

Ανοίγουμε στο πρόγραμμα περιήγησης:

http://nginx-service.**<username>**-priv.svc.cluster.local/

Ο Nginx θα διανέμει τα αιτήματα στους **Web1** και **Web2** εναλλάξ.

1. <https://kubernetes.io/> [↑](#footnote-ref-1)