ΧΑΡΟΚΟΠΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

Σχεδιασμός και ανάπτυξη μιας containerized architecture για μια υπηρεσία διαδραστικής σύνδεσης

Πτυχιακή εργασία

Νικόλαος Τσιπινάκης

Αθήνα, 2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

INFORMATICS & TELEMATICS
DEPARTMENT OF DIGITAL TECHNOLOGY

Design and deployment of a container architecture for an enterprise interactive login service.

Graduate thesis

Nikolaos Tsipinakis

Athens, 2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

Τριμελής Εξεταστική Επιτροπή

Ηρακλής Βαρλάμης (Επιβλέπων)
Αναπληρωτής Καθηγητής, Τμήμα Ψηφιακής Τεχνολογίας, Χροκόπειο Πανεπιστήμιο
Κωνσταντίνος Τσερπές
Αναπληρωτής Καθηγητής, Τμήμα Ψηφιακής Τεχνολογίας, Χροκόπειο Πανεπιστήμιο
Δημήτριος Μιχαήλ Αναπληρωτής Καθηγητής, Τμήμα Ψηφιακής Τεχνολογίας, Χροκόπειο Πανεπιστήμιο

Ο Νικόλαος Τσιπινάκης δηλώνω υπεύθυνα ότι:

1. Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2. Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Περιεχόμενα

Περίληψη

Η υπηρεσία διαδραστικής σύνδεσης χρηστών Linux (LxPlus) παρέχεται από το ερευνητικό εργαστήριο CERN στους υπαλλήλους και εξωτερικούς ερευνητές του με σκοπό την παροχή ενός ομοιόμορφου υπολογιστικού περιβάλλοντος. Η υπηρεσία χρησιμοποιεί το πρωτόκολλο SSH μέσω του οποίου προσφέρει πρόσβαση σε ένα σύνολο από κοινόχρηστος υπολογιστές Linux. Η τεχνολογία container χρησιμοποιείται όλο και περισσότερα τα τελευταία χρόνια για την απομόνωση διεργασιών σε κοινόχρηστα συστήματα, αυτό επτιγχάνεται με την χρήστη πολλαπλών στρωμάτων απομόνωσης που συνδυάζονται για να αποτελέσουν το container. Αυτή η εργασία ερευνά μεθόδους για την υλοποίηση μίας υπηρεσίας διαδραστικής σύνδεσης βασισμένη στο πρωτόκολλο SSH η οποία αξιοποιεί την τεχνολογία των container για την απομόνωση και τον περιορισμό της κατανάλωσης πόρων των χρηστών. Η πρώτη υλοποίηση που ερευνάται χρησιμοποιεί τον πλέον διαδεδομένο εξυπηρετητή OpenSSHd σε συνδυασμό με το Kubernetes. Έπειτα, ερευνάται η καταλληλότητα του εξυπηρετητή ContainerSSH για υπηρεσίες διαδραστικής σύνδεσης. Ο κώδικας του ContainersSH επεκτείνεται με υποστήριξη αυθεντικοποίησης μέσω του πρωτοκόλλου Kerberos και επεκτείνεται περαιτέρω να υποστηρίζει επιπλέον λειτουργίες του πρωτοκόλλου SSH όπως προώθηση συνδέσεων και θυρών δικτύου. Τέλος, ερευνώνται τρόποι να οργανωθούν σε πακέτα λογισμικού και να εκτελεστούν σε μορφή container τα συστήματα αρχείων απομακρυσμένης σύνδεσης AFS, CVMFS και EOS.

Λέξεις κλειδιά: SSH, Containers, Kerberos, AFS, CVMFS

Abstract

The Linux Public Login User Service(LxPlus) is an interactive login service that is setup and maintained by CERN to provide its staff and external researchers a uniform interactive computing environment. The services is based on the SSH protocol and provides access to a cluster of shared Linux nodes. Container technologies like Docker are frequently used to isolate processes within a shared system by providing security boundaries at multiple levels (filesystem, PID, user etc) however they are mostly used for non-interactive workloads. This thesis explores ways of implementing a containerized interactive SSH service via various means. The implementation using the industry standard OpenSSHd server in combination with Kubernetes is explored and the limitations of such an implentation are investigated. Furthermore, the newly developed ContainerSSH server is explored and the ContainerSSH code is expanded to support implementation via the Kerberos authentication protocol in order to be integratable with enterprise-level authentication systems such as Active-Directory. A further expansion of the ContainerSSH code base adds support for missing SSH features such as reverse and forward connection forwarding between the client and the container. Finally, in order to deploy an LxPlus-compatible service, the requirements of the underlying infrastructure are investigated and implemented on the Kubernetes container orchestration system inlcuding the containerization and deployment of the AFS, CVMFS and EOS networked file systems on Kubernetes.

Keywords: SSH, Containers, Kerberos, AFS, CVMFS

Κατάλογος σχημάτων

Κατάλογος πινάκων

Συντομογραφίες

1 Εισαγωγή

1.1 Υπηρεσία διαδραστικής σύνδεσης

Η υπηρεσία διαδραστικής σύνδεσης είναι μία υπηρεσία μέσω της οποίας ένας χρήστης μπορεί να ελέγξει και να εκτελέσει εντολές με ένα απομακρυσμένο υπολογιστικό σύστημα μέσω της γραμμή εντολών ενός τοπικού υπολογιστικού συστήματος. Σε μοντέρνα συστήματα Linux αυτό επιτυγχάνεται μέσω της υπηρεσίας Secure SHell (SSH). Αυτά τα συστήματα λόγο της έλλειψης της ανάγκης για φυσικό υλικό διάδρασης (οθόνη, πληκτρολόγιο) μπορούν να εξυπηρετήσουν πολλούς χρήστες στο ίδιο περιβάλλον, προσφέροντας ένα ομοιόμορφο υπολογιστικό περιβάλλον. Όμως, αυτή η κοινή χρήση ενός συστήματος επιφέρει προβλήματα ασφαλείας και απομόνωσης μεταξύ των χρηστών. Σε περίπτωση κάποιου λογικού σφάλματος σε προγράμματα υψηλής σημασίας όπως το kernel μπορεί να οδηγήσει στην κατάρρευση της απομόνωσης μεταξύ των χρηστών και ως αποτέλεσμα χρήστες να μπορούν να διαβάσουν τα προσωπικά δεδομένα άλλων χρηστών του συστήματος.

1.2 Το μοντέλο των Container

Η αρχιτεκτονική container είναι μία μορφή εικονικοποίησης (virtualization) ενός λειτουργικού συστήματος. Σε αντίθεση με άλλες μορφές εικονικοποίησης όπως τις εικονικές μηχανές, οι οποίες εικονικοποιούν ένα ολόκληρο λειτουργικό σύστημα συμπεριλαμβανομένου του υλικού (hardware), τα container εικονικοποιούν μόνο ένα κομμάτι του λειτουργικού συστήματος ενώ ο φλοιός (kernel) μοιράζεται μεταξύ conainers. Για να εξασφαλιστεί η απομόνωση μεταξύ των containers, τα κομμάτια του φλοιού που διαμοιράζονται μεταξύ τους είναι προστατευμένα χρησιμοποιώντας namespaces. Από την οπτική γωνία ενός χρήστη μέσα σε ένα container, το container φαίνεται σαν να είναι απομονωμένος υπολογιστής η εικονική μηχανή αφού τα namespaces δίνουν σε κάθε container την δική του συσκευή δικτύου μαζί με ιδιωτική εικονική διεύθυνση IP, και δίνουν πρόσβαση μόνο στις διεργασίες και αρχεία που τρέχει ο χρήστης μέσα στο ίδιο container. Επιπλέον, τα namespaces επίσης εξασφαλίζουν τον δίκαιο διαμοιρασμό πόρων μεταξύ όλων των container που τρέχουν σε ένα λειτουργικό σύστημα.

Το κύριο πλεονέκτημα των container είναι τη ταχύτητα με την οποία μπορούν να δημιουργηθούν και να σταματήσουν υπηρεσίες και λογισμικά όταν χρειάζεται. Συνδυάζοντας με αυτό την ασφάλεια και απομόνωση που προσφέρουν και σημαντικότερα το γεγονός πως κάθε container συμπεριφέρεται σαν απομονωμένο σύστημα με το δικό σύστημα αρχείων αυτό τα καθιστά χρήσιμα ως μία μορφή πακετοποίησης προγραμμάτων. Με αυτά κάθε πρόγραμμα μπορεί να έχει διαφορές εκδόσεις βιβλιοθηκών που χρειάζεται χωρίς να επηρεάζει τις βιβλιοθήκες που χρησιμοποιούνται από όλα τα υπόλοιπα προγράμματα που τρέχουν στον ίδιο υπολογιστή. [Merkel et al., 2014]

Σχεδόν ολοκληρωτικά τα container χρησιμοποιούνται για μη-διαδραστικά υπολογιστικά φορτία για παράδειγμα εξυπηρετητές, μεταγλώττιση κώδικα, εκτέλεση αυτοματοποιημένων δοκιμών (tests) σε προγράμματά κτλ. Όμως, τα container μπορούν να είναι χρήσιμα όχι μόνο σε τέτοιου είδους υπολογιστικών εργασιών αλλά και σε διαδραστικές εργασίες όπου ο χρήστης έχει πρόσβαση στο container.

1.2.1 Εικόνες container

Οι εικόνες (images) είναι η μορφή με την οποία διανέμονται τα containers. Ένα container image μπορεί να περιέχει απλά ένα πρότυπο ενός λειτουργικού συστήματος (π.χ. CentOS 7) ή να είναι σχεδιασμένο για να τρέχει ένα και μόνο πρόγραμμα (π.χ. nginx, ένας εξυπηρετητής HTTP). Στην δεύτερη περίπτωση τέτοιες εικόνες container συνήθως χτίζονται πάνω από μία υπάρχουσα εικόνα του πρώτου τύπου. Για παράδειγμα η εικόνα container του nginx είναι επέκταση της εικόνας CentOS 7 με την προσθήκη του προγράμματος nginx και επιπλέον πληροφορίες (metadata) για τον τρόπο εκτέλεσης του.

1.3 Πρωτόκολλο SSH

Το SSH είναι ένα πρωτόκολλο για ασφαλής απομακρυσμένη σύνδεση και άλλες υπηρεσίες δικτύου που είναι σχεδιασμένο για να λειτουργεί πάνω από ένα ανασφαλές δίκτυο [Lonvick and Ylonen, 2006b].

Το πρωτόκολλο χρησιμοποιείται κυρίως για διαδραστικές συνδέσεις χρηστών σε απομακρυσμένους υπολογιστές. Το πρωτόκολλο SSH προσφέρει κυρίως

ένα περιβάλλον γραμμής εντολών με την αυτόματη εκτέλεση ενός φλοιού (shell) με την σύνδεση ενός χρήστη, μπορεί όμως και να χρησιμοποιηθεί για γραφικό περιβάλλον χρησιμοποιώντας την υποστήριξη για προώθηση του πρωτοκόλλου X11 που εμπεριέχεται στις λοιπές υπηρεσίες του πρωτοκόλλου [Lonvick and Ylonen, 2006a] Το πρωτόκολλο X11 είναι το κύριο πρωτόκολλο παραθύρων και γραφικής διεπαφής που χρησιμοποιείται σε Linux τύπου συστήματα.

1.4 Το πρωτόκολλο Kerberos

Το Kerberos είναι μία κατανεμημένη υπηρεσία που επιτρέπει σε ένα πρόγραμμα να αυθεντικοποιήσει τον εαυτό του σε μία απομακρυσμένη υπηρεσία χωρίς να στείλει δεδομένα στο δίκτυο που θα μπορούσαν να επιτρέψουν σε έναν επιτιθέμενο να προσποιηθεί ότι είναι ή να κλέψει την ταυτότητα του χρήστη [Neuman and Ts'o, 1994].

Αν και υπάρχουν πολλά κομμάτια που αποτελούν το πρωτόκολλο Kerberos η ποιο απλή του μορφή και το κομμάτι που χρειαζόμαστε για την υπηρεσία SSH είναι η λειτουργία απομακρυσμένης αυθεντικοποίησης ενός χρήστη προς μία υπηρεσία.

Για να λειτουργήσει αυτό, το πρωτόκολλο βασίζεται πάνω σε κρυπτογραφημένα πακέτα που λέγονται tickets. Τα tickets δημιουργούνται από την κεντρική υπηρεσία αυθεντικοποίησης και στέλνονται στον χρήστη. Ο χρήστης μπορεί να στείλει το ticket που έλαβε στην υπηρεσία που θέλει να αυθεντικοποιηθεί για να αποδείξει την ταυτότητα του. Κάθε ένα από αυτά προορίζεται για μία και μόνο μία υπηρεσία και είναι κρυπτογραφημένο με τέτοιο τρόπο ώστε μόνο η υπηρεσία για την οποία προορίζεται να μπορεί να τα αποκρυπτογραφήσει και να τα διαβάσει.

Πιο συγκεκριμένα, κάθε υπηρεσία όταν εγγράφεται στην κεντρική υπηρεσία αυθεντικοποίησης έχει το δικαίωμα να παράγει ένα μακροχρόνιο κλειδί το οποίο θα χρησιμοποιεί για την αποκρυπτογράφηση και επαλήθευση των ticket που λαμβάνει. Όταν ένας χρήστης επιθυμεί να επικοινωνήσει αυθεντικοποιημένα με μία υπηρεσία πρέπει να επικοινωνήσει με την κύρια υπηρεσία αυθεντικοποίησης η οποία παράγει ένα κρυπτογραφικό κλειδί επικοινωνίας για αυτήν την επικοινωνία. Η υπηρεσία ενσωματώνει σε ένα ticket το όνομα του χρήστη

που ζήτησε την αυθεντικοποίηση μαζί με το κλειδί επικοινωνίας, κρυπτογραφόντας τα με το κλειδί της υπηρεσίας που ζητήθηκε πρόσβαση. Έπειτα στέλνει στον χρήστη το ticket μαζί με το κλειδί εποικηνωνίας.

Όταν ο χρήστης επιθυμεί να αυθεντικοποιήσει τον εαυτό του στην υπηρεσία αρκεί να στείλει το ticket που έλαβε. Η υπηρεσία, κατά την παραλαβή ενός ticket το αποκρυπτογραφεί με το μακροχρόνιο κλειδί της, και αφού αυτό το κλειδί είναι γνωστό μόνο μεταξύ της κεντρικής υπηρεσίας αυθεντικοποίησης και αυτής μπορεί να γνωρίζει πως τα περιεχόμενα του είναι γνήσια. Το ticket περιέχει το όνομα του χρήστη και ένα κρυπτογραφικό κλειδί που όπως αναφέρθηκε έχει σταλθεί μόνο στον χρήστη που ανήκει το ticket. Έτσι, η υπηρεσία μπορεί να κρυπτογραφήσει ένα μήνυμα χρησιμοποιώντας αυτό το κλειδί και αν ο χρήστης αποδείξει ότι μπορεί να το αποκρυπτογραφήσει και να απαντήσει αποδεικνύεται πως ο χρήστης είναι αυτός που ισχυρίζεται πως είναι [Neuman et al., 2005].

1.5 Κίνητρο της εργασίας

Τα συστήματα διαδραστικής σύνδεσης και κοινόχρηστα υπολογιστικά συστήματα που χρησιμοποιούνται σήμερα βασίζονται πάνω στο λειτουργικό σύστημα και τις υπηρεσίας ασφαλείας που προσφέρει αυτό για την απομόνωση των χρηστών. Στις υπηρεσίας αυτές το πρωτόκολλο SSH χρησιμοποιείται συνήθως για να παρέχει απομακρυσμένη πρόσβαση και ασφαλή αφθεντικοποίηση στους χρήστες της. Λόγω αυτής της αρχιτεκτονικής, ένα πρόβλημα ασφαλείας μπορεί να επηρεάσει όλους τους χρήστες της υπηρεσίας. Ο σκοπός αυτής της εργασίας είναι να ερευνηθεί και να υλοποιηθεί μία αρχιτεκτονική υπηρεσίας διαδραστικής σύνδεσης που να προσφέρει μεγαλύτερη απομόνωση μεταξύ των χρηστών της υπηρεσίας περιορίζοντας έτσι τον αντίκτυπο των σφαλμάτων ασφαλείας σε αυτήν. Η υπηρεσία θα βασίζεται πάνω στην τεχνολογία των container αφού αυτή προσφέρει έναν ελαφρύ και ασφαλές τρόπο να απομονωθούν υπολογιστικά φορτία μεταξύ τους.

2 Η Υπηρεσία LxPlus του CERN

Το Ευρωπαϊκό Εργαστήριο Σωματιδιακής Φυσικής (CERN) προσφέρει στους εργαζόμενους και επιστήμονές του απομακρυσμένη πρόσβαση σε υπολογιστές χρησιμοποιώντας το πρωτόκολλο SSH. Συγκεκριμένα, η υπηρεσία Linux Public Login User Service (LxPlus) που είναι σχεδιασμένη για την συγγραφή και δοκιμή προγραμμάτων φυσικής ανάλυσης πριν σταλθούν στο κεντρικό υπολογιστικό δίκτυο μη-διαδραστικής εκτέλεσης (Batch Service).

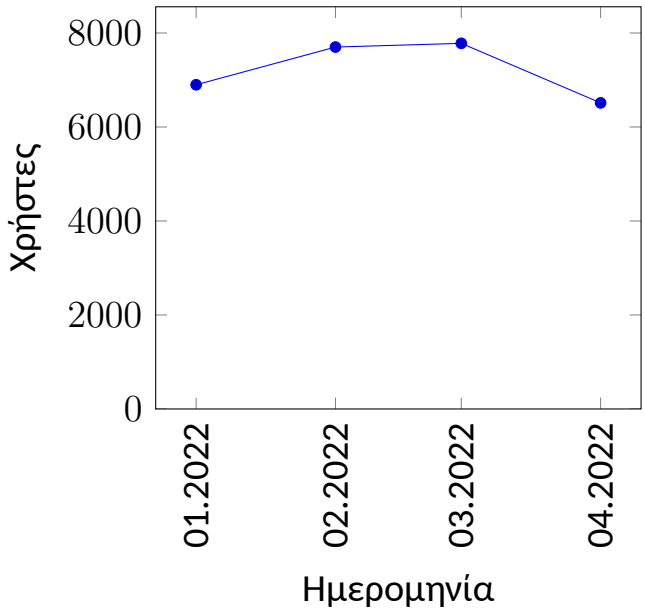
2.1 Η σημαντικότητα της υπηρεσίας LxPlus

Όπως αναφέρθηκε η υπηρεσία LxPlus χρησιμοποιείται από υπαλλήλους του CERN αλλά και εξωτερικούς επιστήμονες από πανεπιστήμια παγκοσμίως που φιλοξενεί το CERN να συγγράφουν, να τρέχουν και να δοκιμάζουν τον κωδικά τους πριν αυτός σταλθεί στην κύρια υπηρεσία ανάλυσης. Επιπλέον η υπηρεσία LxPlus χρησιμοποιείται ως ένας από τους κύριους τρόπους πρόσβασης στο σύστημα αρχείων Andrew File System (AFS) που χρησιμοποιεί το CERN. Το AFS, είναι ένα κατανεμημένο σύστημα αρχείων (network/distributed file system) που χρησιμοποιείται για την προσωρινή και μακροχρόνια αποθήκευση δεδομένων των χρηστών [Espinal et al., 2021].

Το σύστημα αρχείων AFS στο ερευνητικό δίκτυο του CERN χρησιμοποιείται από 35 χιλιάδες χρήστες (5 χιλιάδες την ημέρα), παρέχει πρόσβαση σε 450 TB δεδομένων στα οποία εμπεριέχονται $3.8 * 10^9$ αρχεία. Καθώς καθημερινά υπάρχουν συνολικά $3.5 * 10^9$ αναγνώσεις αυτών των αρχείων [Iven et al., 2017].

Εκτός από το AFS, το LxPlus παρέχει επίσης πρόσβαση στο σύστημα αρχείων EOS Open Storage (EOS), ένα πιο μοντέρνο κατανεμημένο σύστημα αρχείων που χρησιμοποιείται για παρόμοιους σκοπούς με το AFS και φιλοξενεί περίπου 340PB δεδομένων με 6 δισεκατομμύρια αρχεία. [Mascetti et al., 2020] Επιπλέον, υπάρχει το σύστημα αρχείων CernVM-FS (CVMFS), το οποίο χρησιμοποιείται για τον διαμοιρασμό προγραμμάτων και άλλων εκτελέσιμων αρχείων [Blomer et al., 201

Η υπηρεσία LxPlus εξυπηρετεί περίπου 5000 μοναδικούς χρήστες κάθε βδομάδα, με τουλάχιστον 1000-2000 συνεχόμενα ενεργούς χρήστες.

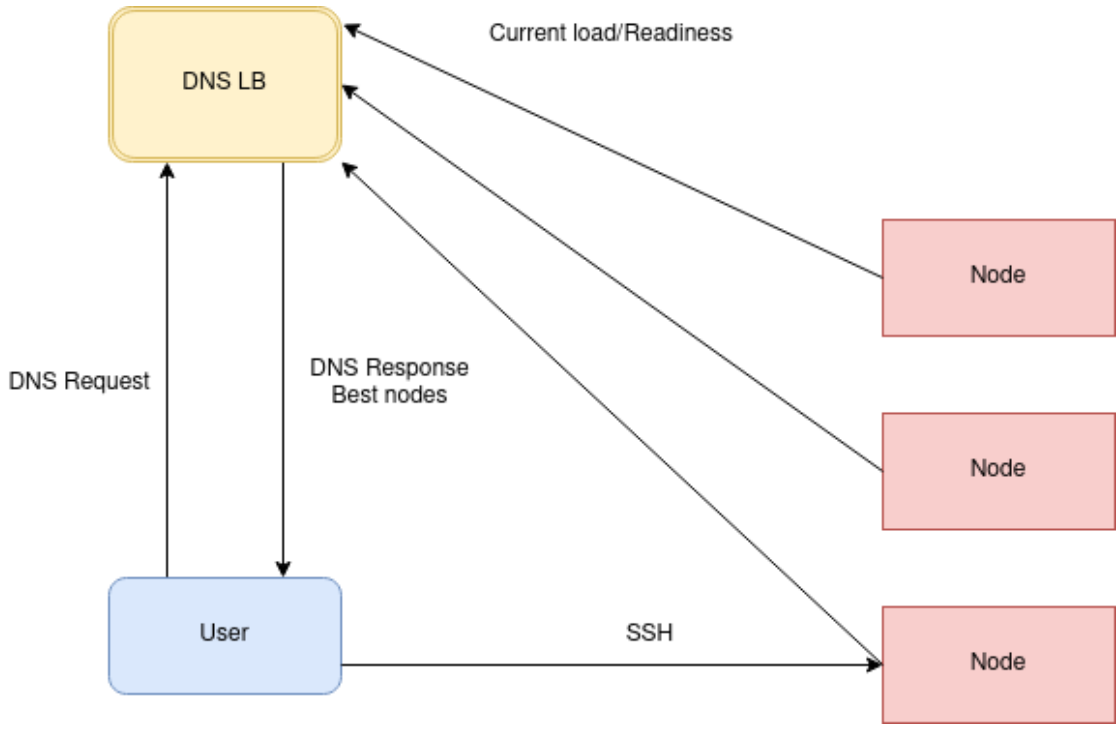


Σχήμα 1: Μοναδικοί χρήστες της υπηρεσίας LxPlus ανα μήνα

2.2 Δομή της υπηρεσίας LxPlus

Η υπηρεσία LxPlus αποτελείται από ένα σύνολο εικονικών μηχανών (εικονική μηχανή (VM)) οι οποίες τρέχουν διαφορετικές εκδόσεις linux. Το μεγαλύτερο ποσοστό αυτών τρέχουν CentOS 7, ενώ ένα μικρότερο αλλά συνεχώς αυξανόμενο ποσοστό CentOS 8 Stream. Η υπηρεσία χρησιμοποιεί τον εξυπηρετητή SSH OpenSSHd για να παρέχει διαδραστική πρόσβαση στους χρήστες καθώς και το πρόγραμμα System Security Services Daemon (SSSD) για να παρέχει υπηρεσίες χρηστών μέσω της βάσης δεδομένων Lightweight Directory Access Protocol (LDAP) και υπηρεσίες αυθεντικοποίησης μέσω κωδικού καθώς και αυθεντικοποίηση μέσω του πρωτοκόλλου Kerberos.

Οι χρήστες διαμοιράζονται στις 100 διαθέσιμες εικονικές μηχανές (κόμβοι) χρησιμοποιώντας τον εξισορροπητή φορτίου (Load Balancer) GoLBD, ο οποίος έχει δημιουργηθεί από το CERN για να καλύψει συγκεκριμένα της ανάγκες του. Ο εξισορροπητής χρησιμοποιεί το σύστημα Domain Name Service (DNS) για να καθορίζει σε ποιους κόμβους οι χρήστες θα κατανεμηθούν. Κάθε κόμβος στέλνει στατιστικά στοιχεία για τον εαυτό του στον εξισορροπητή όπως τον αριθμό των χρηστών που είναι συνδεδεμένη σε αυτόν, πόση ελεύθερη μνήμη διαθέτει και πόσο φορτωμένοι είναι οι επεξεργαστές του. Ο εξισορροπητής χρησιμοποιεί αυτά τα δεδομένα για να κατευθύνει τους χρήστες στους κόμβους με σχετικά μικρό φόρτο εργασίας [Naredo and Pardavila, 2017].



Σχήμα 2: Η υπάρχουσα αρχιτεκτονική της υπηρεσίας LxPlus

Εσωτερικά, κάθε κόμβος τρέχει λειτουργικό σύστημα CentOS Linux. Όλα τα εγκατεστημένα προγράμματα και τα αρχεία ρυθμίσεων διαχειρίζονται κεντρικά με το Puppet Configuration Management σύστημα. Το σύστημα διαχείρισης βασίζεται πάνω στο Version Control System (VCS) Git. Αυτό επιτρέπει στην αυτοματοποιημένη και γρήγορη επαναδημιουργία οποιουδήποτε αριθμού κόμβων και εγγυάται πως όλοι οι κόμβοι έχουν ανανεωμένες και έγκυρες ρυθμίσεις.

Στο σύστημα Linux κανονικά όλοι οι χρήστες έχουν δικαίωμα να δουν όλες τις διεργασίες που τρέχουν στο σύστημα ακόμα και αυτές που ανήκουν σε άλλους χρήστες ή ακόμα και στους διαχειριστές. Για την εξασφάλιση της ιδιωτικότητας των χρηστών κατά την χρήση της υπηρεσίας οι διεργασίες αυτές αποκρύπτονται χρησιμοποιώντας την επιλογή hidepid του συστήματος αρχείου proc. Ως αποτέλεσμα οι χρήστες μπορούν να δουν μόνο τις δικές τους διεργασίες.

2.3 Containerizaton της υπηρεσίας LxPlus

Ο στόχος αυτής της εργασίας είναι να ερευνηθεί, να υλοποιηθεί και να στηθεί μια αντίστοιχη υπηρεσία LxPlus που θα βασίζεται πάνω στην τεχνολογία των container. Συγκεκριμένα, στην είσοδο κάθε χρήστη θα δημιουργείται δυνα-

μικά ένα container στο οποίο μόνο αυτός θα έχει πρόσβαση. Το container αυτό θα παραμένει ενεργό όσο ο χρήστης είναι ενεργός στην υπηρεσία και θα διαγράφεται αυτόματα μόλις αυτός αποσυνδεθεί.

Οι βασικές απαιτήσεις της καινούργιας υπηρεσίας είναι οι εξής:

1. Δημιουργία ενός container κατά την σύνδεση ενός χρήστη.
2. Διαγραφή του container κατά την αποσύνδεση του χρήστη στον οποίο ανήκει.
3. Αυστηρή τήρηση του ορίου ενός χρήστη ανά container.
4. Ένας χρήστης μπορεί να έχει πολλά container που του ανήκουν αν επιλέξει.
5. Εξασφάλιση της πλήρης απομόνωσης συστήματος αρχείων μεταξύ των χρηστών.
6. Εξασφάλιση της πλήρης απομόνωσης δικτύου μεταξύ των χρηστών.
7. Παροχή λειτουργικά ακριβές περιβάλλον εκτέλεσης προγραμμάτων σε σύγκρισή με την υπάρχων υπηρεσία LxPlus.
8. Παροχή όλων των εξωτερικών συστημάτων αρχείων που περιλαμβάνονται στην υπηρεσία LxPlus (AFS, EOS, CVMFS).

Οι απαιτήσεις 1 και 2 είναι απαραίτητες για να τηρηθεί η άμεση δέσμευση και αποδέσμευση πόρων κατά την είσοδο και έξοδο των χρηστών. Αντίθετα με την τωρινή υπηρεσία LxPlus δεν θεωρείται επιθυμητό να δίνεται η δυνατότητα στους χρήστες να τρέχουν μακροχρόνιες διεργασίες χρησιμοποιώντας τους πόρους της. Τέτοιες διεργασίες πρέπει να στέλνονται στην υπηρεσία Batch.

Οι απαιτήσεις 3, 4 και 5 βελτιώνουν την γενική ασφάλεια του συστήματος και ως αποτέλεσμα της υπηρεσίας. Με την υπάρχουσα σχεδίαση της υπηρεσίας αν υπάρξει ένα σφάλμα ασφαλείας στο kernel ή σε οποιοδήποτε πρόγραμμα που τρέχει με προνόμια διαχειριστή, αυτό μπορεί να οδηγήσει σε κάποιον επιτιθέμενο να μπορεί να διαβάσει αρχεία που δεν του ανήκουν. Τα πιο ευαίσθητα αρχεία στην υπηρεσία από την άποψη ασφάλειας είναι τα credentials/kerberos tickets των χρηστών τα οποία βρίσκονται στον κατάλογο προσωρινών αρχείων

/tmp. Σε περίπτωση που κάποιος επιτιθέμενος καταφέρει να πάρει δικαιώματα να διαβάσει αρχεία άλλων μπορεί να κλέψει τα credentials όλων των χρηστών στον ίδιο κόμβο με αποτέλεσμα να μπορεί να κλέψει την ταυτότητα τους και να αυθεντικοποιηθεί σε άλλες υπηρεσίες ως αυτούς.

Η απαίτηση 6 σκοπεύει να κλείσει ένα ελάττωμά ασφαλείας που υπάρχει στην υπάρχουσα υπηρεσία: Όλοι οι χρήστες διαμοιράζονται την ίδια διεύθυνση IP, και ως αποτέλεσμα τις ίδιες θύρες δικτύου. Αυτό έχει δύο άμεσα αποτελέσματα:

- Δύο χρήστες δεν μπορούν να χρησιμοποιήσουν την ίδια θύρα δικτύου ταυτόχρονα και ως αποτέλεσμα μπορεί να τους αποτρέπει από την χρήση λογισμικών που χρειάζονται την θύρα για εσωτερικούς εξυπηρετητές
- Κάθε θύρα που χρησιμοποιεί κάθε χρήστης είναι ορατή και ανοιχτή σε όλους τους άλλους χρήστες στον ίδιο κόμβο

Το δεύτερο αποτέλεσμα έχει επιπτώσεις στην ασφάλεια της υπηρεσίας αφού πολλοί από τους εξυπηρετητές που χρησιμοποιούνται κατά τον προγραμματισμό και δοκιμή προγραμμάτων (development servers, debugging servers, language servers etc) δεν παρέχουν αυθεντικοποίηση με αποτέλεσμα οποιοσδήποτε έχει πρόσβαση στην θύρα μπορεί να πάρει τον έλεγχο του και ως αποτέλεσμα να εκτελέσει κώδικα με τα δικαιώματα του χρήστη. Στην νέα υπηρεσία κάθε container θα παρέχεται με την δικιά του ιδιωτική διεύθυνσή IP και η διαδικτυακή επικοινωνία μεταξύ των container χρηστών θα αποκλείεται αυστηρά με την χρήση κανόνων τοίχου προστασίας (firewall rules).

Οι απαιτήσεις 7 και 8 εξασφαλίζουν πως η καινούργια υπηρεσία θα ικανοποιεί τα ίδια use-cases με την υπάρχουσα υπηρεσία. Για την ικανοποίηση της απαίτησης 7 το καινούργιο περιβάλλον θα πρέπει να περιλαμβάνει όλα τα προεγκατεστημένα προγράμματα καθώς και τα αντίστοιχα αρχεία ρυθμίσεων αυτών. Η απαίτηση 8 εξασφαλίζει την πρόσβαση στων χρηστών σε όλα τα προσωπικά τους αρχεία μέσω των συστημάτων αρχείων AFS και EOS καθώς και την πρόσβαση στα λογισμικά που παρέχονται μέσο του συστήματος CVMFS.

2.4 Εναλλακτικές λύσεις

Αν και το containerizaton της υπηρεσίας βελτιώνει την γενική ασφάλεια της υπηρεσίας καθώς και περιορίζει τον αντίκτυπο των προβλημάτων σφαλμάτων ασφαλείας δεν είναι η μόνη λύση. Είναι επίσης δυνατή η δημιουργία μίας υπηρεσίας διαδραστικής σύνδεσης που κατανέμει ιδιωτικές VM σε κάθε χρήστη. Αυτή η υλοποίηση θα πρόσφερε ακόμα μεγαλύτερη απομόνωση από την χρήση των container όμως η δημιουργία VM είναι χρονοβόρα διαδικασία που θα καθιστούσε την υπηρεσίας δύσχρηστη. Επιπλέον, κάθε VM είναι απαραίτητο να έχει ένα πλήρες αντίγραφο του λειτουργικού συστήματος και να εκτελεί όλες τις διεργασίες που είναι απαραίτητο για την λειτουργία αυτού και ως αποτέλεσμα δεσμεύουν πολύ παραπάνω υπολογιστικούς πόρους σε σχέση με τα container.

3 Λογισμικό που χρησιμοποιήθηκε

Το κεφάλαιο αυτό περιγράφει το βοηθητικό λογισμικό που χρησιμοποιήθηκε για την ανάπτυξη της υπηρεσίας.

3.1 Kubernetes

Το Kubernetes είναι ένα σύστημα ενορχήστρωσης container ανοιχτού κώδικα που εστιάζει στην αυτοματοποιημένη, κατανεμημένη και αποδοτική διαχείριση containersized υπηρεσιών. Υλοποιήθικε αρχικά από την Google και συντηρείται από την Cloud Native Computing Foundation (CNCF). Κάθε kubernetes cluster αποτελείτε από δύο ομάδες κόμβων, τους master που αποτελούν το control plane του cluster και είναι υπεύθυνοι για την παρακολούθηση και την σωστή λειτουργία των container και τα worker nodes τα οποία εκτελούν τα container των υπηρεσιών που φιλοξενούνται στο cluster.

3.2 Helm

Το helm είναι ένας package manager για το σύστημα Kubernetes. Το Helm οργανώνει όλα τα Kubernetes manifests που απαιτούνται για την λειτουργία μίας υπηρεσίας στην μορφή πακέτου Chart και προσφέρει δυναμική παραμετροποίηση των manifests κατά την εγκατάσταση τους στο Kubernetes cluster. Το Helm παρέχει επίσης υποστήριξη για την ανανέωση και αναβάθμιση της έκδοσης του κάθε Chart.

3.3 Flux

Το Flux είναι ένα σύστημα Continuous Integration & Delivery για το Kubernetes. Ο στόχος του είναι να συγχρονίζει δυναμικά την κατάσταση του Kubernetes cluster από ένα αρχικό template το οποίο ενημερώνεται μέσω οποιουδήποτε VCS. Το Flux είναι ένα εργαλείο υψηλότερου επιπέδου από το Helm το οποίο όμως το επεκετείνει αφού παρέχει πλήρη υποστήριξη για την αυτόματη αναβάθμιση πακέτων Chart από οποιαδήποτε πηγή. Η κύρα διαφορά με το Helm είναι πως με την χρήση του Helm είναι απαραίτητο η χειροκήνιτη αναβάθμιση των Chart ενώ με το Flux αυτά μπορούν να αναβαθμιστούν αυτόματα σύμφωνα με το κεντρικό template.

4 Containerizaton παραμετροποιόντας τον εξυπηρετητή OpenSSHd και Kubernetes

Η πρώτη προσπάθεια containerizaton έγινε χρησιμοποιώντας τον πιο δημοφιλές εξυπηρετητή SSH, τον OpenSSHd, και την υπηρεσία διαχείρισης container Kubernetes. Η σχεδίαση αυτής της υπηρεσίας στηρίζεται πάνω σε έναν Kubernetes Controller ο οποίος είναι υπεύθυνος για την δημιουργία και διαγραφή των container. Μία σειρά από OpenSSHd servers θα είναι υπεύθυνα για την αποδοχή των χρηστών. Αυτοί οι server θα είναι ρυθμισμένοι έτσι ώστε κατά την είσοδο ενός χρήστη να εκτελείται το πρόγραμμα "create_container" το οποίο θα δημιουργεί ένα "UserPod" αντικείμενο στον Kubernetes cluster. Η δημιουργία αυτού του αντι-

κειμένου στέλνει σήμα στον controller ο οποίος διαβάζει τις λεπτομέρειες (π.χ. το όνομα χρήστη του χρήστη που μπήκε) και δημιουργεί το container σύμφωνα με τις ρυθμίσεις του. Κάθε εικόνα container που προορίζεται για χρήστες επίσης περιέχει μέσα ένα SSH server (OpenSSHd). Αυτό το server είναι παραμετροποιημένο με τον συνηθισμένο τρόπο για να δίνει shell πρόσβαση στους χρήστες που συνδέονται. Μετά την αρχικοποίηση του container χρήστη από το Kubernetes, ο controller ανανεώνει το αρχικό αντικείμενο userpod με την διεύθυνση IP του container. Τέλος, το αρχικό proxy pod που έδωσε την εντολή δημιουργίας του συνδέει την σύνδεση του χρήστη στο ssh server που τρέχει μέσα στο container που δημιουργήθηκε, επιτρέποντας έτσι στον χρήστη να έχει πρόσβαση σε αυτό μετά από μία δεύτερη αυθεντικοποίηση.

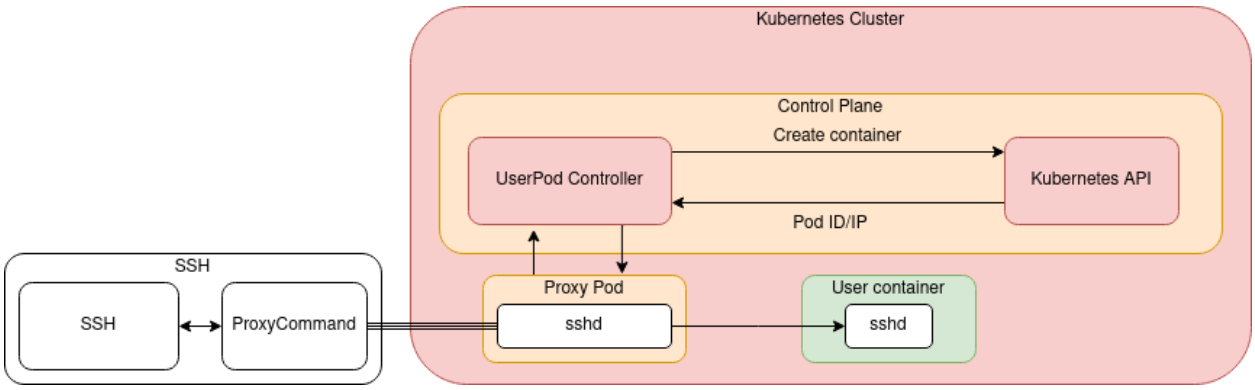
Η ασφάλεια σε αυτό το σύστημα εξασφαλίζεται χρησιμοποιώντας κανόνες τοίχου προστασίας (firewall rules, Kubernetes NetworkPolicy). Τα user containers επιτρέπουν μόνο εξερχόμενη σύνδεση προς το διαδίκτυο και μπλοκάρουν συνδέσεις προς εσωτερικές υπηρεσίες του Kubernetes Cluster. Επιπλέον εισερχόμενες συνδέσεις προς το user container προς το ssh server μέσα σε αυτό είναι μπλοκαρισμένες από παντού εκτός από τα proxy containers. Αυτό εξασφαλίζει πως ο μόνος τρόπος να επικοινωνήσει κάποιον με αυτόν τον εξυπηρετητή είναι μέσω των proxy containers. Τέλος, κατά την δημιουργία κάθε user container παραμετροποιείται αυτόματα κατά την δημιουργία του να επιτρέπει μόνο στον χρήστη στον οποίο ανήκει να αυθεντικοποιηθεί. Αυτό επιτυγχάνεται χρησιμοποιώντας το "pam_listfile" PAM module το οποίο αφήνει μόνο χρήστες που αναφέρονται σε ένα αρχείο να αυθεντικοποιηθούν. Το αρχείο δημιουργείται μέσω ενός shell script το οποίο χρησιμοποιεί την μεταβλητή περιβάλλοντος "$USERNAME" η οποία τίθεται από τον controller.

Από την πλευρά του χρήστη, η διαδικασία αυθεντικοποίησης θα είναι σε 2 φάσεις:

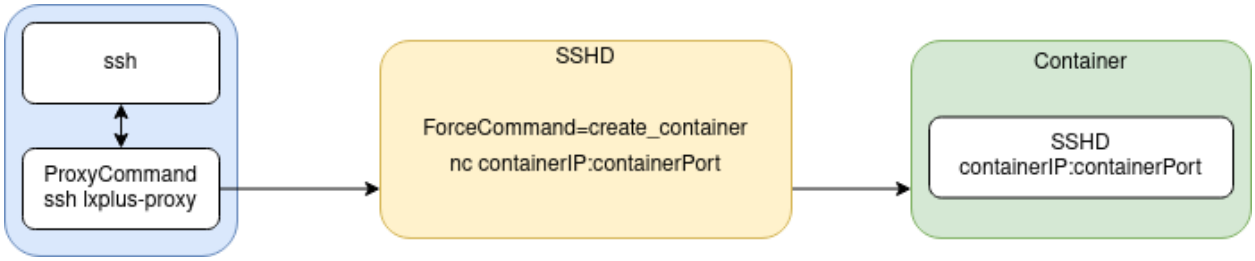
- Αυθεντικοποίηση στον πρώτο, proxy, SSH server.
- Αυθεντικοποίηση στον δεύτερο SSH server, μέσα στο User Container.

Για την σύνδεση χρησιμοποιώντας τον OpenSSH client χρειάζεται την παρακάτω παραμετροποίηση.

```
1 Host lxplus.cern.ch
2     ProxyCommand=create_container
```



Σχήμα 3: Αρχιτεκτονική της υπηρεσίας με την χρήση OpenSSHd και Kubernetes



Σχήμα 4: Παράμετροι και εντολές που θα εντελλόντουσαν κατά την είσοδο στην υπηρεσία με την χρήση OpenSSHd

Αυτό παρουσιάζει τρία κύρια μειονεκτήματα: Πρώτον, για να γίνουν 2 αυθεντικοποιήσεις σε σειρά με αυτόν τον τρόπο ο χρήστης θα πρέπει να παραμετροποιήσει τον ssh client του για να χρησιμοποιήσει το proxy server σαν jump host και θα είναι αδύνατον να συνδεθεί σε αυτόν χωρίς αυτήν την ρύθμισή. Δεύτερον, ο εξυπηρετητής SSH πρέπει να τρέχει με δικαιώματα διαχειριστή (root) ώστε να διαβάζει τα μυστικά κλειδιά (kerberos keytab, SSH host keys) και έτσι το user container δεν μπορεί να τρέξει ως πλήρως unpriviledged. Τρίτων, για να τρέχει ο εξυπηρετητής SSH πρέπει να υπάρχουν τα προαναφερόμενα μυστικά (kerberos keytab, SSH host keys) μέσα στο container και σε περίπτωση σφάλματος ασφαλείας όπου επιτρέπει στον χρήστη να πάρει επιπλέον δικαιώματα (priviledge escallation) αυτό θα έχει ως αποτέλεσμα την διαρροή αυτών των μυστικών και την υποβάθμιση της ασφάλειας της υπηρεσίας.

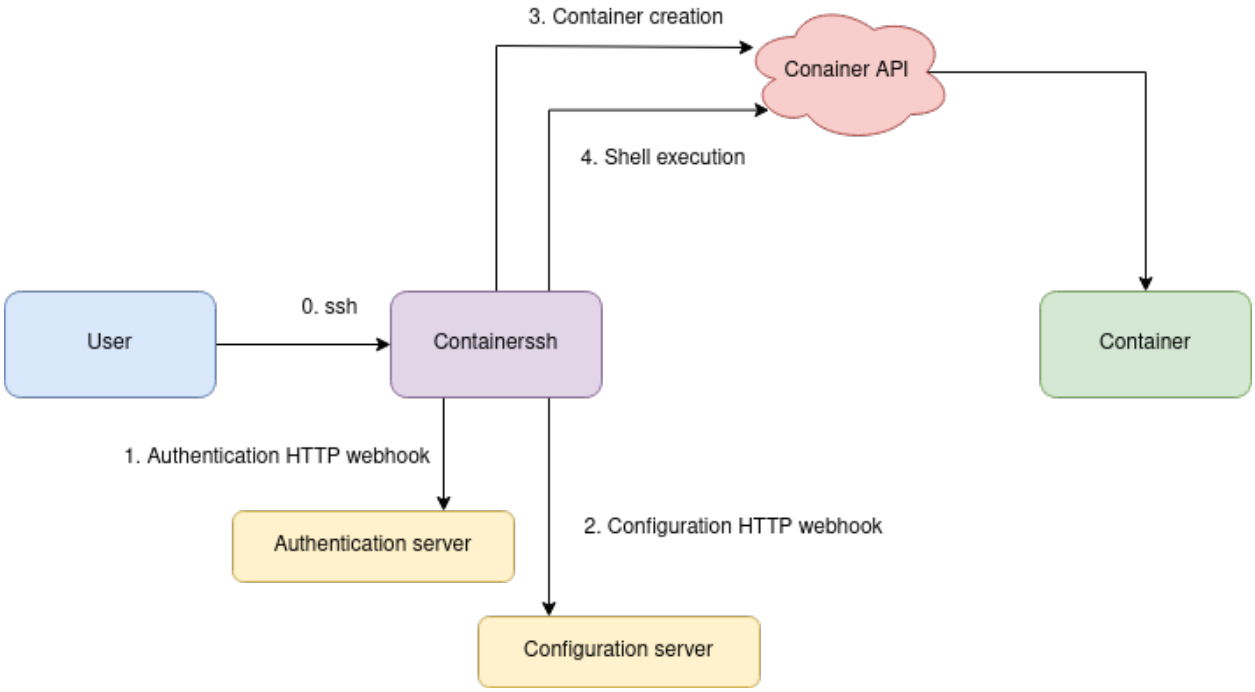
5 ContainerSSH

Το ContainerSSH είναι ένα λογισμικό ανοιχτού κώδικα, γραμμένο στη γλώσσα προγραμματισμού Go. Ο στόχος του λογισμικού είναι να παρέχει μια υπηρεσία SSH η οποία δίνει σε κάθε χρήστη της υπηρεσίας ένα μοναδικό container για την εκπόνηση των αναγκών του. Για να επιτύχει αυτό, το ContainerSSH περιλαμβάνει μια πλήρης υλοποίηση του πρωτοκόλλου SSH καθώς βασίζεται στις βιβλιοθήκες των συστημάτων docker και kubernetes για την δημιουργία και λειτουργία των container.

5.1 Εξωτερική αρχιτεκτονική του ContainerSSH

Το ContainerSSH βασίζεται σε 3 βασικά κομμάτια για την λειτουργία του. Το Authentication Server, το οποίο είναι υπεύθυνο για την αυθεντικοποίηση των χρηστών, το Configuration Server το οποίο είναι υπεύθυνο για την προσαρμογή του περιβάλλοντος στις απαιτήσεις του κάθε χρήστη, και τέλος το Backend Server το οποίο είναι υπεύθυνο για την δημιουργία και εκτέλεση του container.

Και τα 3 αυτά κομμάτια βασίζονται στο πρωτόκολλο HyperText Transfer Protocol (HTTP). Για τα πρώτα δύο, το Authentication Server και το Configuration Server, υπάρχει ένα ορισμένο Application Programming Interface (API) και ο διαχειριστής συστημάτων που εγκαταστήσει την υπηρεσία είναι υπεύθυνος να παρέχει μία υλοποίηση που είναι αποδεχτή για το περιβάλλον του. Συγκεκριμένα, δεν μπορεί να υπάρξει μια γενική υλοποίηση αυτών των API αφού κάθε οργανισμός έχει δικό του τρόπο αυθεντικοποίησης χρηστών και διαφορετικές απαιτήσεις για το περιβάλλον των χρηστών της υπηρεσίας. Το τρίτο κομμάτι, το backend server το οποίο δημιουργεί και τρέχει τα container, πρέπει να είναι μία από τις υπάρχοντες και υποστηριζόμενες υπηρεσίες container. Συγκεκριμένα, είτε η υπηρεσία Docker ή η υπηρεσία Kubernetes. Η υπηρεσία Podman επίσης υποστηρίζεται επειδή το API που υλοποιεί είναι συμβατό με το API του Docker.



Σχήμα 5: Ροή δεδομένων κατά την αρχικοποίηση σύνδεσης

5.1.1 Αρχείο ρυθμίσεων

Η τοποθεσία του αρχείου ρυθμίσεων του ContainerSSH ορίζεται μέσω της γραμμής εντολών με την εντολή "--config /path/to/file". Το αρχείο χρησιμοποιεί μορφοποίηση τύπου YAML και χρησιμοποιείται για να ορίσει και τις βασικές λειτουργίες του ContainerSSH όπως την διεύθυνση των Authentication Server και Configuration Server αλλά και ρυθμίσεις όπως σε ποιον Kubernetes Cluster ή Docker Agent να συνδεθεί και με τι παραμέτρους να δημιουργήσει τα container. Για την δεύτερη περίπτωση ρυθμίσεων αυτές μπορούν να αλλάξουν μέσω των ρυθμίσεων που θα επιστρέψει το Configuration Server για τον κάθε χρήστη.

Η βασική δομή του αρχείου ρυθμίσεων του ContainerSSH είναι η εξής:

```
log:
  level: "info"
ssh:
  hostkeys:
    - /etc/containerssh/host.key
metrics:
  enable: true
auth:
 method: webhook
 webhook:
   url: http://127.0.0.1:8080
```

```
    pubkey: true
    password: true
configserver:
  url: http://127.0.0.1:8080/config
backend: kubernetes
kubernetes:
  connection:
    host: kubernetes.default.svc
    cacertFile: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
  pod:
    metadata:
      labels:
        app: shell
    spec:
      containers:
      - env:
        image: containerssh/containerssh-guest
        name: shell
```

5.1.2 Πρωτόκολλο Αυθεντικοποίησης

Το Πρωτόκολλο αυθεντικοποίησης είναι υπεύθυνο για την ταυτοποίηση των χρηστών της υπηρεσίας. Το πρωτόκολλο υποστηρίζει τους μηχανισμούς όνομα/κωδικό, δημοσίου κλειδιού καθώς και διαδραστική αυθεντικοποίηση. Η αυθεντικοποίηση γίνεται στέλνοντας ένα αίτημα HTTP για κάθε σύνδεση στην υπηρεσία αυθεντικοποίησης που αναφέρθηκε με τις λεπτομέρειες του χρήστη, η υπηρεσία μπορεί να απαντήσει είτε με true ή false ανάλογα αν η αυθεντικοποίηση ήταν επιτυχής.

Στην περίπτωση αυθεντικοποίησης με όνομα και κωδικό το ContainerSSH στέλνει το όνομα (username) του χρήστη που αυθεντικοποιείται, την διεύθυνση από την οποία συνδέθηκε, ένα αναγνωριστικό για την σύνδεση, και τον κωδικό που έδωσε ο χρήστης κωδικοποιημένο σε Base64 μορφή.

Παρακάτω υπάρχει ένα παράδειγμα αίτησης αυθεντικοποίησης με την μέθοδο κωδικού όπως στέλνεται.

```
{
    "username": "username",
    "remoteAddress": "127.0.0.1:1234",
    "connectionId": "An opaque ID for the SSH connection",
    "passwordBase64": "cGFzc3dvcmQ="
}
```

Μετά από την λήψη μίας τέτοιας αίτησης το Authentication Server επιβεβαιώνει την εγκυρότητά των στοιχείων που δόθηκε και απαντάει θετικά η αρνητικά με την παρακάτω μορφή.

```
{
    "success": true
}
```

Στην περίπτωση αυθεντικοποίησης με μέθοδο δημοσίου/ιδιωτικού κλειδιού αντίστοιχα το Authentication Server λαμβάνει το όνομα του χρήστη που αυθεντικοποιείται, την διεύθυνση IP από την οποία συνδέεται, ένα μοναδικό αναγνωριστικό σύνδεσης και τέλος το δημόσιο κλειδί του χρήστη σε απλό κείμενο, κωδικοποιημένο με την μορφή που υποστηρίζει το OpenSSH.

```
{
    "username": "username",
    "remoteAddress": "127.0.0.1:1234",
    "connectionId": "An opaque ID for the SSH connection",
    "publicKey": "ssh-rsa ..."
}
```

```
{
    "success": true
}
```

5.1.3 Πρωτόκολλο ρύθμισης/προσαρμογής

Μετά από την επιτυχής αυθεντικοποίηση του χρήστη το πρωτόκολλο ρύθμισης και προσαρμογής χρησιμοποιείται για την δημιουργία του περιβάλλον container που θα χρησιμοποιηθεί από τον χρήστη.

```
{
    "username":"foo"
}
```

```
{
  "config": {
    "backend": "docker",
    "docker": {
        ...
    }
  }
}
```

5.2 Εσωτερική αρχιτεκτονική του ContainerSSH

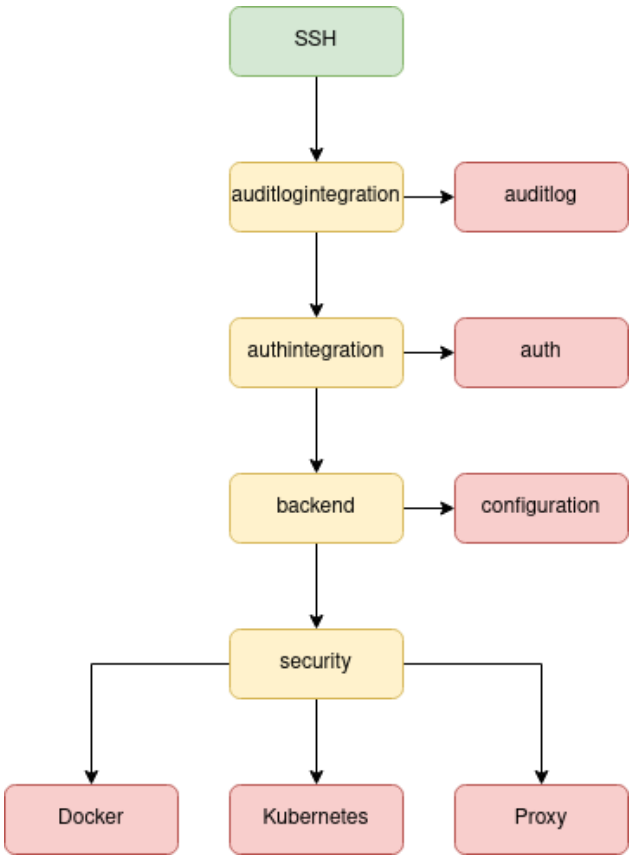
Εσωτερικά το ContainerSSH είναι χωρισμένο σε ανεξάρτητα modules που το καθένα είναι εξειδικευμένο σε μία συγκεκριμένη λειτουργία. Το κεντρικό 'SSH' ή 'sshserver' module εξάγει μια κεντρική διεπαφή η οποία πιστά απεικονίζει τα μηνύματα επικοινωνίας του πρωτοκόλλου SSH. Συγκεκριμένα, για κάθε εντολή ή μήνυμα που λαμβάνεται καλείται η αντίστοιχη μέθοδος στην διεπαφή. Αφού κάθε module επεξεργαστεί ένα μήνυμα έχει την υποχρέωση να το προωθήσει στο επόμενο module στην αλυσίδα που απεικονίζεται στο σχήμα 6. Αυτό επιτρέπει σε κάθε module να αντιδράσει ανεξάρτητα σε αυτό καθώς και αν χρειαστεί να επηρεάσει την ροή του μηνύματος. Για παράδειγμα, το security module έχει την δυνατότητα να διακόψει και να ακυρώσει την διάδοση ενός μηνύματος στα παρακάτω modules παίρνοντας την απόφαση να μην διαδώσει το μήνυμα αν αυτό δεν επιτρέπεται από τις υπάρχων ρυθμίσεις ασφαλείας του προγράμματος.

Τα κύρια modules 'auditlog', 'auth' χρησιμοποιούν την δική τους διεπαφή σε περίπτωση που χρειαστεί να επαναχρησιμοποιηθούν με διαφορετικό τρόπο και έτσι στηρίζονται στα βοηθητικά modules 'auditlogintegration' και 'authintegration' για την μετατροπή των κλήσεων από την μορφή της διεπαφής του sshserver στην μορφή του κάθε module.

6 Υλοποίηση του πρωτοκόλλου Kerberos στο ContainerSSH

Το πρωτόκολλο SSH υποστηρίζει την αυθεντικοποίηση χρηστών μέσω του πρωτοκόλλου Kerberos και πιο συγκεκριμένα μέσω του πρωτοκόλλου αυθεντικοποίησης Generic Security Standard Application Programming Interface (GSS-API) [Salowey et al., 2006] [Linn, 1993]. Ο μηχανισμός GSS-API είναι ένα γενικό πρωτόκολλο αυθεντικοποίησης που μπορεί να υποστιρήξει πολλαπλές μεθόδους και πρωτόκολλα αυθεντικοποίησης. Η μέθοδος με την οποία το πρωτόκολλο Kerberos ενσωματώνεται στον μηχανισμό GSS-API περιγράφεται στο RFC 1964. [Linn, 1996]

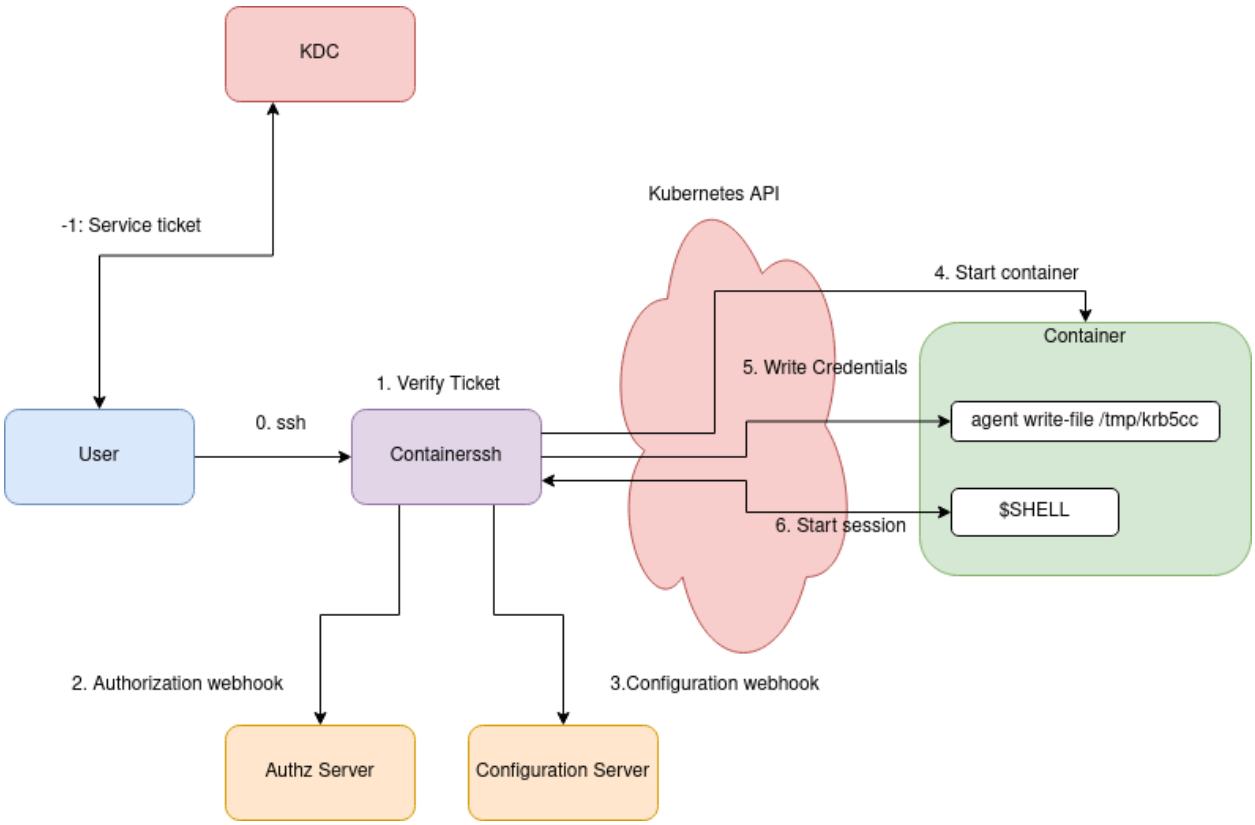
Το ContainerSSH χρησιμοποιεί την ενσωματωμένη βιβλιοθήκη της γλώσσας Go για την υλοποίηση του πρωτοκόλλου SSH η οποία προσφέρει έτοιμες μεθόδους οι οποίες μπορούν να χρησιμοποιηθούν για την υλοποίηση του πρωτοκόλλου GSS-API. Η Go, όμως, δεν προσφέρει ενσωματωμένη υποστήριξη για



Σχήμα 6: Εσωτερική αρχιτεκτονική κώδικα ContainerSSH

την αυθεντικοποίηση μέσω του GSSAPI και μετά από σχετική έρευνα δεν υπάρχει υλοποιημένη βιβλιοθήκη που να υλοποιεί το πρωτόκολλο GSS-API. Υπάρχει όμως η βιβλιοθήκη gokrb5 [Jonathan Turner, 2022] η οποία υλοποιεί το πρωτόκολλο Kerberos και μπορεί να χρησιμοποιηθεί ως βάση για την υλοποίηση του πρωτοκόλλου GSS-API.

Για την υλοποίηση, επεκτάθηκε το auth module που απεικονίζεται στο σχήμα 6 να αναγνωρίζει και να επικυρώνει τα Kerberos Tickets με βάση το πρωτόκολλο GSS-API/Kerbros χρησιμοποιώντας την βιβλιοθήκη gokrb5. Ο συγκεκριμένος τρόπος αυθεντικοποίησης παρακάμπτει τον υπάρχοντα μηχανισμό αυθεντικοποίησης μέσω του Authentication Server. Επιπλέον, το πρωτόκολλο Kerberos προσφέρει αυθεντικοποίηση αλλά δεν ελέγχει τα δικαιώματα του χρήστη. Όμως, έχει κριθεί απαραίτητο να ελέγχεται όχι μόνο η ταυτότητα του χρήστη αλλά και αν ο συγκεκριμένος χρήστης έχει δικαίωμα πρόσβασης στην συγκεκριμένη υπηρεσία. Για αυτό επεκτάθηκε το πρωτόκολλο αυθεντικοποίησης που χρησιμοποιείται να εμπεριέχει μία τρίτη κλήση, authorization, στο Authentication & Configuration server που θα ελέγχει τις άδειες του χρήστη.



Σχήμα 7: Ροή δεδομένων κατά την αυθεντικοποίηση με μέθοδο kerberos

Είναι σημαντικό να διακριθεί η διαφορά μεταξύ του ονόματος λογαριασμού χρήστη στην κεντρική υπηρεσία αυθεντικοποίησης και το όνομα χρήστη με το οποίο ο χρήστης αιτείται να συνδεθεί, διότι αυτά δεν ταιριάζουν πάντα. Κατά την αρχική σύνδεση με τον πρωτόκολλο SSH ο χρήστης ενημερώνει τον εξυπηρετητή για το όνομα χρήστη με το οποίο θέλει να αυθεντικοποιηθεί και ο εξυπηρετητής επιστρέφει τις επιτρεπόμενες μεθόδους αυθεντικοποίησης που υποστηρίζει. Σε περίπτωση που ο χρήστης επιλέξει να αυθεντικοποιηθεί με την μέθοδο GSS-API/Kerberos και αποστείλει το Kerberos Ticket του μπορεί να βρεθεί πως το όνομα χρήστη που αναγράφεται στο ticket είναι διαφορετικό από αυτό με το οποίο συνδέθηκε ο χρήστης. Ο τρόπος χειρισμού αυτής της περίπτωσης αφήνεται στον διαχειριστή του συστήματος. Για λόγους ασφαλείας αν το όνομα του χρήστη διαφέρει από το όνομα του λογαριασμού που αυτός συνδέεται η σύνδεση αυτή απορρίπτεται, για να επιτραπεί αυτή η περίπτωση είναι απαραίτητο να γίνει ρητά επιθυμητό μέσω του αρχείου ρυθμήσεων. Η απενεργοποίηση της παραμέτρου EnforceUsername αναθέσει τον χειρισμό αυτής της περίπτωσης στο Authentication & Configuration server με την χρήση της κλήσης authorization. Το αρχικό όνομα χρήστη με το οποίο επιθυμεί να συνδεθεί ο χρήστης αναγράφεται

στο πεδίο loginUsername ενώ το αυθεντικοποιημένο όνομα χρήστη, δηλαδή η πραγματική ταυτότητα του χρήστη, αναγράφεται στο πεδίο principalUsername.

Κατά την authorization κλήση αποστέλνονται στον εξυπηρετητή το αναγνωριστικό της σύνδεσης, η διεύθυνση IP από την οποία συνδέθηκε ο χρήστης καθώς και το αυθεντικοποιημένο όνομα του. Ο εξυπηρετητής μπορεί να επιστρέψει θετικό η αρνητικό αποτέλεσμα με την χρήστη του πεδίου success.

```
{
    "loginUsername": "username",
    "principalUsername": "username",
    "remoteAddress": "127.0.0.1:1234",
    "connectionId": "An opaque ID for the SSH connection",
}
```

```
{
    "success": true
}
```

Για την χρήση του πρωτοκόλλου GSS-API για την αυθεντικοποίηση είναι απαραίτητο να έχει δημιουργηθεί και να είναι προσβάσιμο το αρχείο με το κρυπτογραφικό κλειδί της υπηρεσίας (Keytab). Η τοποθεσία του αρχείου αυτού δίνεται μέσω του αρχείου ρυθμίσεων. Επιπλέον, χωρίς παραπάνω παραμετροποίησή όλοι οι αυθεντικοποιημένοι χρήστες γίνονται αποδεκτοί στην υπηρεσία. Η ενεργοποίηση του μηχανισμού ελέγχου αδειών που περιγράφηκε είναι απαραίτητο να γίνει ρητά στο αρχείο ρυθμίσεων. Ακολουθεί ένα παράδειγμα ρυθμίσεων αυθεντικοποίησης:

```
auth:
  method: kerberos
  kerberos:
    keytab: '/etc/krb5.keytab'
  authz:
    enable: true
    url: http://127.0.0.1:8080/authz
```

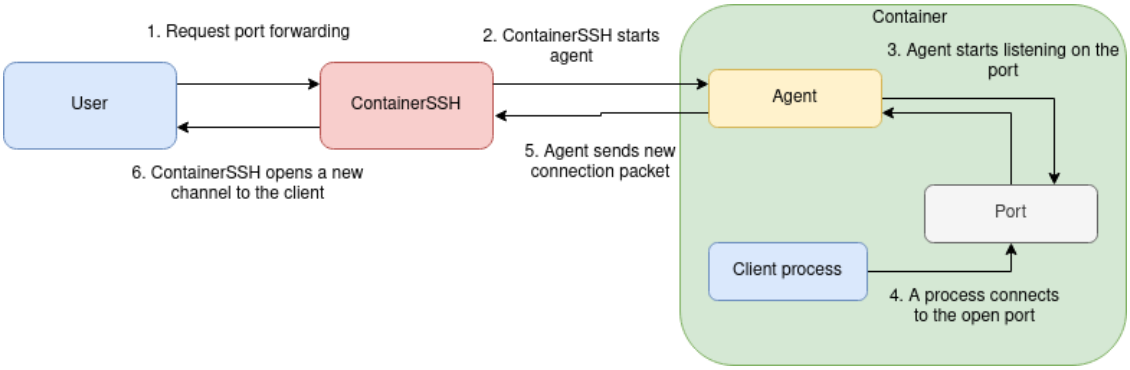
7 Υλοποίηση μηχανισμού προώθησης θυρών στο ContainerS

Το πρωτόκολλο SSH προσφέρει υποστήριξη για προώθησή πακέτων δικτύου από τον client στον server αλλά και αντίστροφα. Ο χρήστης μπορεί να επιλέξει κατά την σύνδεση του ποιες θύρες θα προωθηθούν από τον τοπικό τους

υπολογιστή προς τον εξηπηρετητή και αντίστροφα, ποιες θύρες από τον εξυπηρετητή θα προωθηθούν προς τον τοπικό του υπολογιστή.

Για την λειτουργία αυτού, όταν ζητηθεί η προώθηση θυρών από τον τοπικό υπολογιστή προς τον εξυπηρετητή ο ssh client ανοίγει την τοπική θύρα και περιμένει για εισερχόμενες συνδέσεις. Όταν μία διεργασία συνδεθεί στην θύρα που έχει ανοίξει ο ssh client τότε αυτός ανοίγει ένα καινούργιο κανάλι πρωτοκόλλου SSH ενημερώνοντας τον εξυπηρετητή για την θύρα που έγινε η σύνδεση. Σε περίπτωση που η προώθηση δεν επιτρέπεται από τον εξυπηρετητή το καινούργιο κανάλι δεν θα γίνει αποδεκτό από αυτόν.

Για την προώθηση απομακρυσμένης θύρας προς τον τοπικό υπολογιστή κατά την σύνδεση ο client στέλνει ένα αίτημα (request) με τον αριθμό της θύρας που επιθυμεί να προωθηθεί. Ο εξυπηρετητής τότε θα ανοίξει την αντίστοιχη θύρα περιμένοντας συνδέσεις. Όταν μία διεργασία συνδεθεί στην προωθημένη θύρα το εξυπηρετητής, αντίστοιχα με την αντίστροφη περίπτωση, θα στείλει αίτημα να ανοίξει ένα καινούργιο κανάλι στον client. Αν αυτό είναι επιτυχής τότε τα δεδομένα της σύνδεσης προωθούνται μέσω αυτού του καναλιού.



Σχήμα 8: Ροή προώθησης θυρών δικτύου

Λόγω της αρχιτεκτονικής του ContainerSSH, δηλαδή που το περιβάλλον που εκτελούνται οι εντολές του χρήστη είναι διαφορετικό από το περιβάλλον που τρέχει ο εξυπηρετητής, αυτό δεν μπορεί να υλοποιηθεί απευθείας πάνω στον εξυπηρετητή του ContainerSSH. Για την σωστή έξοδο των πακέτων προστέθηκε υποστήριξη συνδέσεων στον ContainerSSH Agent, ένα εκτελέσιμο πρόγραμμα που είναι συμαντικό να υπάρχει στην εικόνα container που χρησιμοποιεί το ContainerSSH. Αυτό χρησιμοποιείται για διάφορες βοηθητικές ενέργειες όπως το να στέλνει σήματα σε διεργασίες όταν ζητηθεί από τον χρήστη. Σε αυτή την περίπτωση προστέθηκε υποστήριξη για προώθηση συνδέσεων μέσω αυτού.

Ενώ το πρωτόκολλο SSH υποστηρίζει εσωτερικά την έννοια των καναλιών, όπου δηλαδή μέσα από την ίδια σύνδεση μπορούν να σταλθούν δεδομένα για πολλαπλά προγράμματα ή για πολλαπλών συνδέσεων δικτύων ταυτόχρονα, η εποικηνωνία μεταξύ του ContainerSSH και του Agent γίνεται με την κανονική είσοδο και έξοδο που παρέχει το σύστημα. Χωρίς την εννοια των καναλιών και έχοντας την ανάγκη να πολυπλέξουμε συνδέσεις, χρειάστηκε να αναπτυχθεί ένα καινούργιο δυαδικό πρωτόκολλο συγκεκριμένα για αυτόν τον σκοπό.

Το πρωτόκολλο διασύνδεσης μεταξύ του ContainerSSH και του ContainerSSH Agent βασίζεται πάνω στην μορφή Concise Binary Object Representation (CBOR). "Το Concise Binary Object Representation (CBOR) είναι μια μορφή δεδομένων των οποίων οι σχεδιαστικοί στόχοι περιλαμβάνουν τη δυνατότητα εξαιρετικά μικρού κώδικα μέγεθος, αρκετά μικρό μέγεθος μηνύματος και επεκτασιμότητα χωρίς την ανάγκη για διαπραγμάτευση έκδοσης." [Bormann and Hoffman, 2020]

7.1 Πρωτόκολλο επικοινωνίας ContainerSSH <-> ContainerSSH Agent

Κάθε μήνυμα του πρωτοκόλλου πρέπει να έχει την μορφή που απεικονίζεται στον πίνακα 2. Κάθε είδος πακέτου ξεχωρίζεται από την τιμή του πεδίου 'Type'. Για μηνύματα που προορίζονται για μία συγκεκριμένη σύνδεση (π.χ. πακέτο δεδομένων) η σύνδεση ορίζεται στο πεδίο 'ConnectionId' ενώ αν το πακέτο είναι γενικό (π.χ. SetupPacket) το ConnectionId πρέπει να είναι 0. Τέλος το περιεχόμενο μήνυμα περιέχεται στο πεδίο Payload, σε μορφή CBOR.

Για την αποστολή ενός πακέτου όπως στο πίνακα 2 αρκεί να κωδικοποιηθεί αυτό σε μορφή CBOR και να σταλθεί στην είσοδο της άλλης πλευράς.

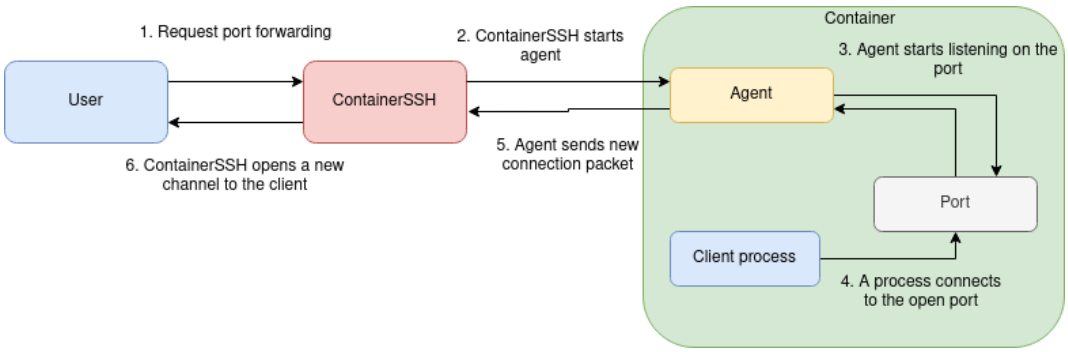
Τα διαφορετικά είδη πακέτων που υποστηρίζονται αναφέρονται στον πίνακα 1.

ID	Χαρακτηριστικό	Περιγραφή
0	PACKET_SETUP	Πακέτο αρχικοποίησης
1	PACKET_SUCCESS	Μήνυμα επιτυχίας
2	PACKET_ERROR	Μήνυμα σφάλματος
3	PACKET_DATA	Πακέτο δεδομένων
4	PACKET_NEW_CONNECTION	Αρχοκοποίηση καινούργιας σύνδεσης
5	PACKET_CLOSE_CONNECTION	Τερματισμός μίας σύνδεσης
6	PACKET_NO_MORE_CONNECTIONS	Απόρριψη μελλοντικών συνδέσεων

Πίνακας 1: Πίνακας υποστηριζομένων μηνυμάτων

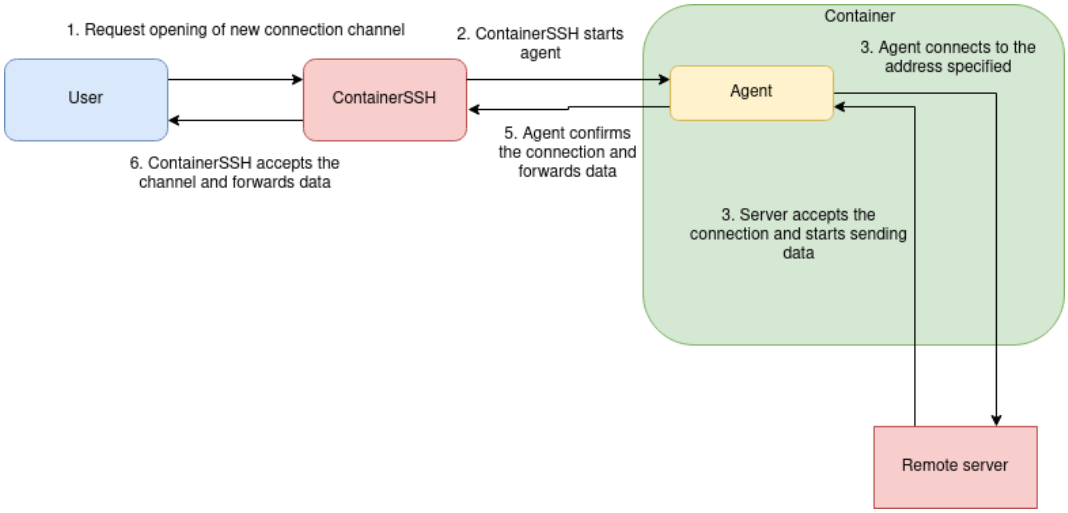
Κατά την αρχικοποίηση του Agent, πρώτα στέλνετε από το ContainerSSH το SetupPacket, αυτό ορίζει τον τύπο της προώθησης που θα ακολουθήσει. Συγκεκριμένα, το πεδίο ConnectionType οι τιμές που υποστηρίζονται απεικονίζονται στον πίνακα 4.

Ο τύπος σύνδεσης CONNECTION_TYPE_PORT_FORWARD αρχικοποιεί τον Agent σε λειτουργία προώθησης θυρών από το container προς τον χρήστη. Συγκεκριμένα, κατά την παραλαβή μηνύματος αρχικοποίησης τέτοιου τύπου ο Agent θα ξεκινήσει να ακούει για συνδέσεις στην διεύθυνση και θύρα που ζητήθηκε. Μόλις ληφθεί μια σύνδεση στην θύρα αυτή ο Agent θα στείλει το αντίστοιχο μήνυμα (PACKET_NEW_CONNECTION) στο server, το οποίο θα το προωθήσει στον client με την σωστή μορφή σύμφωνα με το πρωτόκολλο SSH. Ο server πρέπει να απαντήσει με μηνύματα επιτυχίας ή σφάλματος (SUCCESS/ERROR) σήμφωνα με τον αν ο client αποδέχθηκε την σύνδεση ή αν την απέρριψε. Σε περίπτωση που αποδεχθεί την σύνδεση και με την παραλαβή του μηνύματος επιτυχίας η σύνδεση θεωρείτε ενεργή και δεδομένα αυτής μπορούν να σταλθούν προς οποιαδήποτε κατεύθυνση (client ή agent) μέσω των μηνυμάτων PACKET_DATA.



Σχήμα 9: Διάγραμμα ροής μηνυμάτων σε λειτουργία προώθησης θυρών

Ο τύπος σύνδεσης CONNECTION_TYPE_PORT_DIAL αρχικοποιεί τον Agent σε λειτουργία προώθησης θυρών από τον χρήστη προς το container (ή προς το εξωτερικό δίκτυο, μέσω του container). Σε αυτήν την λειτουργία όταν ο server λάβει μήνυμα προώθησης σύνδεσης από τον χρήστη τότε ο server στέλνει μήνυμα αρχικοποίησης σύνδεσης (PACKET_NEW_CONNECTION) στον Agent με την διεύθυνση και θύρα που επιθυμεί να συνδεθεί. Ο Agent με την παραλαβή αυτού του μηνύματος θα συνδεθεί στην διεύθυνση και θύρα που δόθηκε και θα ενημερώσει τον server αν η σύνδεση ήταν επιτυχής μέσω των μηνυμάτων SUCCESS/ERROR. Έπειτα από την επιτυχής σύνδεση τα δεδομένα της σύνδεσης μεταφέρονται μέσω του μηνύματος δεδομένων (PACKET_DATA). Στον συγκεκριμένο τύπο σύνδεσης επίσης εντάσσεται η προώθηση σε unix socket αντί για διεύθυνση δικτύου. Σε αυτή την περίπτωση το πεδίο Protocol του πακέτου αρχικοποίησης σύνδεσης θα έχει την τιμή "unix" και το μονοπάτι του socket περιέχεται στο πεδίο ConnectedAddress.



Σχήμα 10: Διάγραμμα ροής μηνυμάτων σε λειτουργία προώθησης συνδέσεων

Ο τύπος σύνδεσης CONNECTION_TYPE_SOCKET_FORWARD αντιστοιχεί στον

τύπο port forward με την διαφορά πως αντί για προώθηση θυρών δικτύου προωθούνται συνδέσεις στον χρήστη που γίνονται σε συγκεκριμένο μονοπάτι αρχείου μέσω unix socket.

Ο τύπος σύνδεσης CONNECTION_TYPE_X11 ενημερώνει τον Agent να κάνει προώθηση του πρωτοκόλλου X11 από το τοπικό περιβάλλον. Ο συγκεκριμένος τύπος προώθησης απαιτεί περαιτέρω πληροφορίες από τον χρήστη οι οποίες δίνονται μέσω των πεδίων 'Protocol', 'Screen', 'SingleConnection', 'AuthProtocol' και 'AuthCookie' αυτά τα πεδία απευθύνονται συγκεκριμένα στο πρωτόκολλο X11 και δεν χρησιμοποιούνται για άλλο σκοπό. Κατά την αρχικοποίηση του για προώθηση πρωτοκόλλου X11 ο Agent προσθέτει τις πληροφορίες που αναφέρθηκαν στο τοπικό αρχείο 'Xauthority' το οποίο περιέχει τον κατάλογο με όλες τις οθόνες X11 που υποστηρίζονται καθώς και τις πληροφορίες αυθεντικοποίησης σε αυτήν ('AuthCookie'). Τέλος ο Agent ακολουθεί την ίδια διαδικασία όπως την προώθηση θυρών, αλλά ξεκινάει να ακούει στην θύρα $6000+screen$ όπου screen είναι ο αριθμός της οθόνης όπως ορίζει το πρωτόκολλο X11. [rfc, 1987]

Field	Type
Type	int
ConnectionId	uint64
Payload	[]byte

Πίνακας 2: Εξωτερική μορφή πακέτου

Field	Type
ConnectionType	uint32
BindHost	string
BindPort	uint32
Protocol	string
Screen	string
SingleConnection	bool
AuthProtocol	string
AuthCookie	string

Πίνακας 3: Μορφή payload του SetupPacket

ID	Χαρακτηριστικό
0	CONNECTION_TYPE_X11
1	CONECTION_TYPE_PORT_FORWARD
2	CONNECTION_TYPE_PORT_DIAL
3	CONNECTION_TYPE_SOCKET_FORWARD

Πίνακας 4: Τύποι συνδέσεων που μπορεί να σταλθούν στο SetupPacket

ID	Χαρακτηριστικό
tcp	PROTOCOL_TCP
unix	PROTOCOL_UNIX

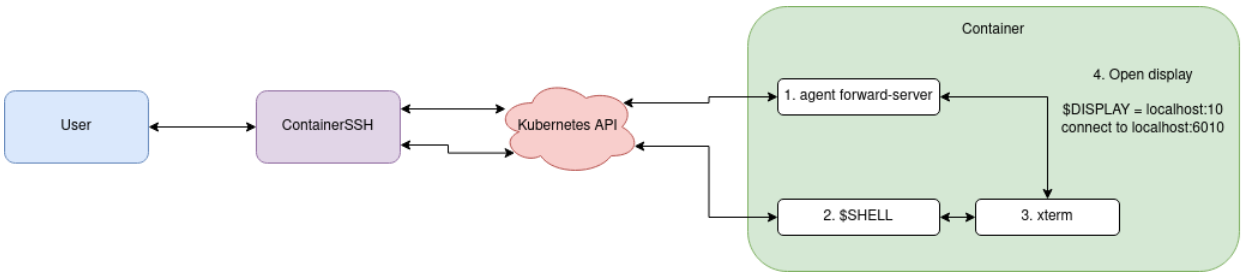
Πίνακας 5: Υποστηριζόμενα πρωτόκολλα στο πακέτο NewConnection

Field	Type
Protocol	string
ConnectedAddress	string
ConnectedPort	uint32
OriginatorAddress	string
OriginatorPort	uint32

Πίνακας 6: Μορφή payload του NewConnection

7.2 Υλοποίησης μηχανισμού προώθησης πρωτοκόλλου X11

Σύμφωνα με το πρωτόκολλο X11, το πρωτόκολλο μπορεί να προωθηθεί πάνω από οποιαδήποτε αξιόπιστη σύνδεση δυαδικών μηνυμάτων. Για να γίνει αυτό πάνω από συνδέσεις TCP αρκεί ο εξυπηρετητής X11 να ακούει στην θύρα $6000 + N$ όπου N είναι ο αριθμός της οθόνης που θα εμφανιστεί το παράθυρο. [rfc, 1987] Λόγο του ότι κάθε σύνδεση στο ContainerSSH αντιστοιχεί σε ένα και μόνο ένα container, και πως σε κάθε σύνδεση SSH η προώθηση X11 γίνεται σε όλα τα κανάλια με γενική αίτηση (global request) ο αριθμός οθόνης ορίστηκε ως 10. Το πρωτόκολλο υποστηρίζει τον ορισμό άλλων οθονών όμως για την απλοικότητα της υλοποίησης του ContainerSSH αυτό δεν χρησιμοποιείται.

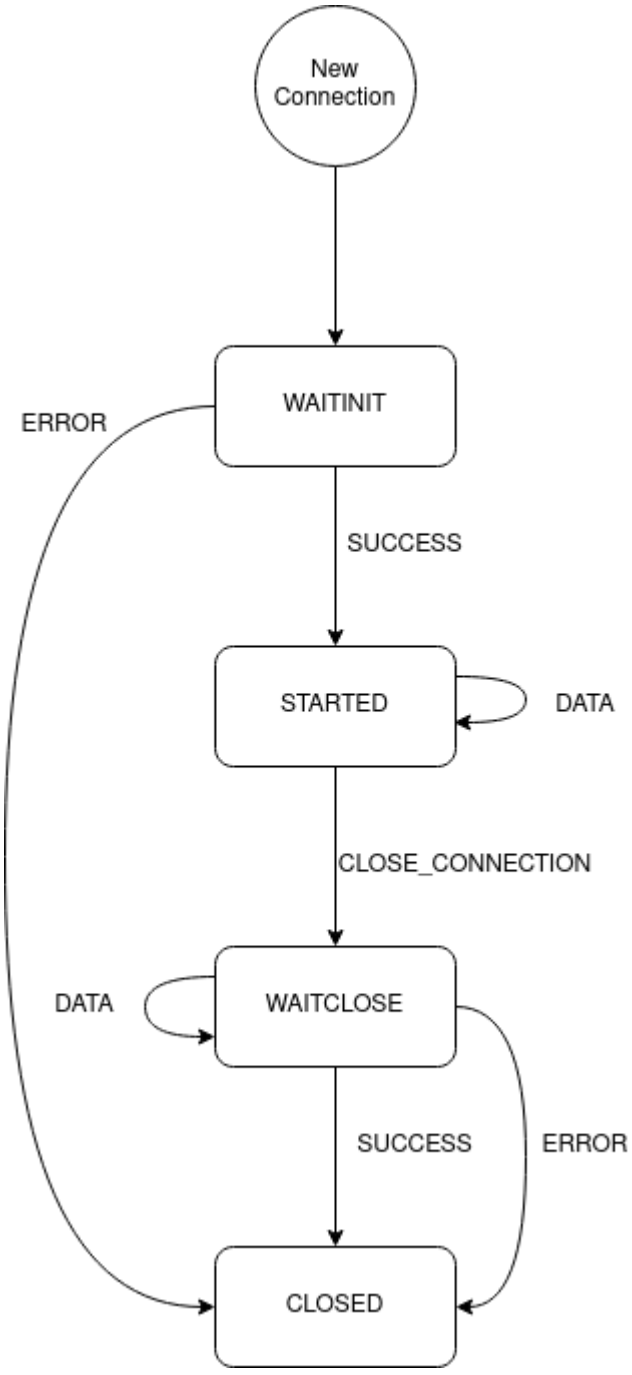


Σχήμα 11: Ροή μηχανισμού X11

Περαιτέρω, το πρωτόκολλο X11 απαιτεί αυθεντικοποίηση των προγραμμάτων κατά την σύνδεση τους. Το κομμάτι της αυθεντικοποίησης το διαχειρίζεται ο εξηπηρετητής X11 στον οποίο προωθούνται οι συνδέσεις. Όμως, για την σωστή αυθεντικοποίηση απαιτείται τα προγράμματα μέσα στο container να έχουν πρόσβαση στο κλειδί αυθεντικοποίησης. Αυτό επιτυγχάνεται μέσω των παραμέτρων του SetupPacket που περιγράφονται στον πίνακα 3. Τα πεδία AuthProtocol περιγράφει το πρωτόκολλο που χρησιμοποιείται για την αυθεντικοποίηση και το πεδίο AuthCookie είναι μία δεκαεξαδική συμβολοσειρά που αποτελεί το μυστικό αυθεντικοποίησης.

Μήνυμα / Κατάσταση	SUCCESS	ERROR	DATA	CLOSE_CONNECTION
WAITINIT	->STARTED	->CLOSED	Invalid	->WAITCLOSE
STARTED	Invalid	Invalid	->STARTED	->WAITCLOSE
WAITCLOSE	->CLOSED	->CLOSED	->WAITCLOSE	Invalid
CLOSED	Invalid	Invalid	Invalid	Invalid

Πίνακας 7: Καταστάσεις συνδέσεων του πρωτοκόλλου Agent



Σχήμα 12: Σχήμα καταστάσεων σύνδεσης

7.3 Γενική διεπαφή πρωτοκόλλου

Πάνω στο πρωτόκολλο που σχεδιάστηκε δημιουργήθηκε μία βιβλιοθήκη με γενική διεπαφή η οποία μπορεί να επαναχρησιμοποιηθεί ή να επεκταθεί. Ποιο σημαντικά, η σχεδίαση της βιβλιοθήκης επιτρέπει την αλλαγή του πρωτοκόλλου ή του μέσου επικοινωνίας χωρίς την αλλαγή της εξωτερικής διεπαφής.

Η διεπαφή αυτή αποτελείτε από 2 κύρια αντικείμενα. Το ForwardCtx και το Connection. Το ForwardCtx είναι το γενικό αντικείμενο που είναι υπεύθυνο για την αρχικοποίηση και την εκκίνηση συνδέσεων καθώς και την ενημέρωση για καινούργιες συνδέσεις που αιτήθηκαν από την άλλη πλευρά επικοινωνίας. Το αντικείμενο Connection αντιπροσωπεύη μία μοναδική σύνδεση και παρέχει μία διεπαφή διαχείρισης αυτής.

Η πλήρης διεπαφή περιγράφεται με τα παρακάτω interface:

```
type Connection interface {
  Read(p []byte) (n int, err error)
  Write(p []byte) (n int, err error)
  Close() error

  CloseImm() error
  Accept() error
  Reject() error
  Details() NewConnectionPayload
}

type ForwardCtx interface {
  NewConnectionTCP(
    connectedAddress string,
    connectedPort uint32,
    origAddress string,
    origPort uint32,
    closeFunc func() error,
  ) (io.ReadWriteCloser, error)

  NewConnectionUnix(
    path string,
    closeFunc func() error,
  ) (io.ReadWriteCloser, error)

  StartServer() (
      connectionType uint32,
      setupPacket SetupPacket,
      connChan chan *Connection,
      err error
```

```
  )

  StartClientForward() (chan *Connection, error)

  StartX11ForwardClient(
      singleConnection bool,
      screen string,
      authProtocol string,
      authCookie string
  ) (chan *Connection, error)

  StartReverseForwardClient(
      bindHost string,
      bindPort uint32,
      singleConnection bool
  ) (chan *Connection, error)

  StartReverseForwardClientUnix(
      path string,
      singleConnection bool
  ) (chan *Connection, error)

  NoMoreConnections()

  WaitFinish()

  Kill()
}
```

Κατά την αρχικοποίηση της βιβλιοθήκης ο προγραμματιστής θα πρέπει πρώτα να καλέσει μία από τις συναρτήσεις Start*. Όλες οι συναρτήσεις αρχικοποιούν την βιβλιοθήκη με διαφορετικές παραμέτρους. Συγκεκριμένα, η συνάρτηση StartServer ενημερώνει την βιβλιοθήκη πως επιθυμεί να παραλάβει από την άλλη πλευρά εποικηνωνίας της ρυθμήσεις που θα χρησιμοποιηθούν. Συγκεκριμένα, όταν κληθεί η συνάρτηση StartServer η βιβλιοθήκη περιμένει την αντίθετη πλευρά να στείλει το ειδικό πακέτο SetupPacket το οποίο περιέχει πληροφορίες για το είδος σύνδεσης που θα δέχεται αυτή η εκτέλεση της βιβλιοθήκης. Συγκεκριμένα, η συνάρτηση αυτή επιστρέφει τον τύπο της σύνδεσης που αιτήθηκε από τον client. Οι υποστηριζόμενες τιμές απεικονίζονται στον πίνακα 4. Επιπλέον, επιστρέφεται το πακέτο SetupPacket όπως παραλήφθηκε από τον client, η μορφή του περιγράφεται στον πίνακα 3. Τα πεδία Screen, SingleConnection, AuthProtocol, AuthCookie του SetupPacket ορίζονται μόνο από τον τύπο σύνδεσης X11 και είναι οι παράμετροι που χρειάζονται τα προγράμματα που υλοποιούν το πρωτόκολλο X11 για να αυθεντικοποιηθούν. Η συνάρτηση StartServer

επίσης επιστρέφει ένα κανάλι συνδέσεων, καινούργιες συνδέσεις από το server θα προωθούνται μέσω αυτού του καναλιού.

Η συνάρτηση StartClientForward αρχικοποιεί την βιβλιοθήκη σε λειτουργία client με είδος σύνδεσης CONNECTION_TYPE_PORT_DIAL. Σε αυτήν την λειτουργία ο client αρχικοποιεί όλες τις συνδέσεις. Η συνάρτηση επιστρέφει ένα κανάλι συνδέσεων από το οποίο για την σωστή λειτουργία ο προγραμματιστής θα πρέπει να φροντίσει να απορρίπτονται όλες οι συνδέσεις αφού δεν προβλέπονται σε αυτό το είδος λειτουργίας. Μετά την αρχικοποίησης ο client μπορεί να ζητήσει την δημιουργία συνδέσεων μέσω των συναρτήσεων NewConnection*. Σε αυτή την λειτουργία υποστηρίζονται και συνδέσεις TCP αλλά και συνδέσεις τύπου unix.

Η συνάρτηση StartReverseForwardClient αρχικοποιεί την βιβλιοθήκη σε λειτουργία client με είδος σύνδεσης CONNECTION_TYPE_PORT_FORWARD. Οι παράμετροι bindHost, bindPort ορίζουν την διεύθυνσης και την θύρα που ο εξυπηρετητής θα πρέπει να ακούει για καινούργιες συνδέσεις. Η παράμετρος singleConnection σηματοδοτεί στον εξηπηρετητή πως πρέπει να αποδεχθεί μόνο μία σύνδεση και όλες οι υπόλοιπες να απορρίφθούν. Σε αυτήν την λειτουργία ο server πρέπει να ξεκινήσει ένα listening socket με τις παραμέτρους που δόθηκαν και να σηματοδοτεί στον client κάθε φορά που δέχεται μία σύνδεση. Ο client αντίστοιχα μπορεί να δεχθεί η να απορρίψει την σύνδεση. Αντίστοιχα, οι συναρτήσεις StartReverseForwardClient και StartX11ForwardClient αρχικοποιούν την βιβλιοθήκη σε λειτουργία client για τα είδη συνδέσεων CONNECTION_TYPE_X11 και CONNECTION_TYPE_SOCKET_FORWARD.

Κατά την παραλαβή μίας καινούργιας σύνδεσης από το κανάλι, η υλοποίηση θα πρέπει να εξετάσει αν ο τύπος σύνδεσης είναι επιτρεπτός (π.χ. αν είναι σύνδεση από την θύρα που είχε ζητηθεί να ακούει) και να την αποδέχεται ή να την απορρίπτει αντίστοιχα. Σε περίπτωση αποδοχής της και οι 2 πλευρές μπορούν να στέλνουν δεδομένα μέσω αυτής της σύνδεσης (bidirectional stream).

Οι συνάρτηση NoMoreConnections ενημερώνει την αντίθετη πλευρά πως δεν επιθυμεί να λάβει παραπάνω συνδέσει. Επιπλέον αιτήσεις σύνδεσης μετά την εκτέλεση της συγκεκριμένης συνάρτησής θα απορρίπτονται. Τέλος, η συνάρτηση WaitFinish περιμένει μέχρι όλες οι ενεργές συνδέσεις να τερματίσουν πριν επιστρέψει ενώ η συνάρτηση Kill τερματίζει όλες τις συνδέσεις και κόβει την επικοινωνία χωρίς να περιμένει επιβεβαίωση.

8 Υποδομή λειτουργίας της υπηρεσίας

Κατά την ανάπτυξη της καινούργιας υπηρεσίας LxPlus κρίθηκε αναγκαίο να στηθεί ένα αντίγραφο του πηγαίου κώδικα στο αποθετήριο του CERN και να στηθεί ένα build pipeline για την αυτόματη μεταγλώττιση και deployment της υπηρεσίας. Το αποθετήριο κώδικα του CERN βασίζεται πάνω στο αποθετήριο ανοιχτού κώδικα GitLab (gitlab.com). Η υπηρεσία αποθετηρίου κώδικα του CERN βρίσκεται στον σύνδεσμο gitlab.cern.ch. Ένα αντίγραφο του κώδικα του ContainerSSH καθώς και αποθετήρια με τον πηγαίο κώδικα εικόνων container που χρησιμοποιούνται για το στήσιμο της υπηρεσίας αποθηκεύτηκαν εκεί. Για την εγκατάσταση της υπηρεσίας χρειάστηκε να πακετάριστεί η εφαρμογή σε 2 μορφές, η πρώτη μορφή σε μορφή container για να εγκατασταθεί πάνω στην υπηρεσία Kubernetes και η δεύτερη μορφή σε μορφή πακέτου Red Hat Package Manager (RPM) για να εγκατασταθεί σε απλούς κόμβους Linux όπου θα μπορεί να χρησιμοποιεί την ίδια υποδομή με την υπάρχουσα υπηρεσία. Αποφασίστηκε η υπηρεσία να στηθεί και με τις 2 μεθόδους, δηλαδή και χρησιμοποιώντας την υπάρχουσα υποδομή linux αλλά και να δοκιμαστεί μια καινούργια υποδομή βασισμένη στο σύστημα σύστημα ενορχήστρωσης container Kubernetes.

8.1 Αυτόματη δημιουργία πακέτου RPM μέσω GitLab Pipeline

Το CERN παρέχει ένα καλούπι αρχείο ρυθμίσεων (rpmCI) του gitlab το οποίο επιτρέπει να δημιουργηθούν εύκολα αλυσίδες διεργασιών που δημιουργούν πακέτα RPM και αυτόματα τα ανεβάζουν στο σύστημα μεταγλώττισης και δημισίευσης πακέτων του CERN. Το ContainerSSH όμως έχει την ιδιαιτερότητα πως βασίζεται σε πολλές εξωτερικές βιβλιοθήκες της γλώσσας Go οι οποίες δεν είναι ήδη πακεταρισμένες στα αποθετήρια του λειτουργικού συστήματος CentOS. Κατά το πακετάρισμα προγραμμάτων σε εκδόσεις Linux θεωρείτε καλή πρακτική κάθε βιβλιοθήκη να πακετάρεται σε ξεχωριστό πακέτο ώστε να μπορεί να επαναχρησιμοποιηθεί και να εγκατασταθεί ξεχωριστά. Όμως, δεδομένου πως η γλώσσα προγραμματισμού Go χρησιμοποιεί στατικό τύπο σύνδεσης βιβλιοθηκών, δηλαδή ο κώδικας της κάθε βιβλιοθήκης περιλαμβάνεται μέσα στο τελικό εκτελέσιμο αντί να φορτώνεται από εξωτερικά αρχεία, επίσης δεδομένου πως το πακέτο αυτό προορίζεται για εσωτερική χρήση αποφασίστηκε να συμπεριληφθούν

όλες οι βιβλιοθήκες σε ένα πακέτο του ContainerSSH. Αυτή η πρακτική είναι συνήθης για την πακετοποίηση μεγάλων προγραμμάτων και ονομάζεται vendoring.

Για την υλοποίηση του vendoring των βιβλιοθηκών προστέθηκε ένα στάδιο prebuild στο οποίο χρησιμοποιείτε ο διαχειριστής πακέτων go που παρέχει την ενσωματωμένη επιλογή go vendor η οποία κατεβάζει όλες τις βιβλιοθήκες που απαιτούνται στον υποφάκελο vendor. Αποφασίστηκε το vendoring να γίνεται στο στάδιο prebuild αντί να προστεθούν οι vendored βιβλιοθήκες στο αποθετήριο κώδικα λόγω του γεγονότος πως ο όγκος των βιβλιοθηκών θα επιβάρυνε το μέγεθος του αποθετηρίου. Επιπλέον, ο γρήγορος ρυθμός εξέλιξης και αναβάθμισης των βιβλιοθηκών σήμαινε πως αν συμπεριλαμβανόντουσαν οι vendored βιβλιοθήκες στο αποθετήριο θα χρειαζόντουσαν πολλές ανανεώσεις σε αυτές το οποίο επίσης θα επιβάρυνε το μέγεθος του αποθετηρίου.

Μετά την επιτυχή μεταγλώττιση του προγράμματος και των βιβλιοθηκών το RPM που παράγεται προωθείται στο σύστημα Koji για να ενσωματωθεί στο γενικό αποθετήριο πακέτων Linux. Από εκεί κάθε σύστημα Linux που χρειάζεται το ContainerSSH μπορεί να το εγκαταστήσει ή αν είναι ήδη εγκατεστημένο να το αναβαθμίσει στην καινούργια έκδοση.

Για την εγκατάσταση στην υποδομή Kubernetes είναι επίσης απαραίτητο να δημιουργηθεί μία εικόνα container αντίστοιχα. Για αυτήν προστέθηκε ένα ακόμα στάδιο το οποίο χρησιμοποιεί την εφαρμογή Docker για να δημιουργήσει μία εικόνα χρησιμοποιώντας επίσης τις vendored βιβλιοθήκες που δημιουργήθηκαν νωρίτερα.

8.2 Εικόνες container χρηστών

Άλλο ένα μεγάλο κρίσιμο κομμάτι της υπηρεσίας LxPlus είναι το σύνολο προεγκατεστημένω προγραμμάτων που υποστηρίζει. Ο κατάλογος αυτών των προγραμμάτων διατηρείται μέσω του συστήματος διαχείρισης παραμέτρων Puppet, το οποίο χρησιμοποιείται για την παραμετροποίησή όλων των κόμβων στην υπηρεσία LxPlus. Εξερευνήθικαν διαφορετικές επιλογές για την εγκατάσταση των προγραμμάτων αυτών σε εικόνες container.

Η πρώτη επιλογή για την δημιουργία εικόνων container μέσω του συστή-

ματος παραμετροποίησης puppet ήταν η εκτέλεση του Puppet Agent, το πρόγραμμα το οποίο εκτελείτε σε κάθε σύστημα για να εφαρμόσει τις παραμέτρους, μέσα σε ένα container κατά την διάρκεια της διαδικασία χτισίματος της. Αυτή η διαδικασία αν και αποτελεσματική δεν ήταν αποδεκτή. Το γεγονός πως για κάθε εκτέλεση της διαδικασίας χτισίματος του container επαναεκτελόταν ο Agent σήμαινε πως κάθε εκτέλεση θα χρησιμοποιούσε την τωρινή έκδοση των παραμέτρων που βρίσκεται στο αποθετήριο, αφού ο Puppet Agent τραβάει την πιο πρόσφατη έκδοση κατά την εκτέλεση του, αυτό έχε το αποτέλεσμα πως είναι αδύνατο να επαναδημιουργηθούν εικόνες με παλαιότερες παραμετροποιήσεις. Επιπλέον, λόγω του γεγονότος πως το Puppet Agent είναι σχεδιασμένος να δρα σε ένα αυτοτελή σύστημα Linux απαιτήται ένα πιστοποιητικό (SSL Key) για την προστασία των παραμέτρων (Puppet Certificate/Host Certificate). Αυτά τα πιστοποιητικά περιορίζουν τις παραμέτρους που μπορεί να εφαρμόσει ο Puppet Agent μόνο σε αυτές που ταιριάζουν τις παραμέτρους που χρειάζεται το συγκεκριμένο σύστημα. Αυτό έχει ως αποτέλεσμα πως αν εκτελεστεί ο Agent χρησιμοποιόντας το πιστοποιητικό του συστήματος που εκτελεί το χτίσιμο της εικόνας αυτό δεν θα μπορεί να τραβήξει τις παραμέτρους που απαιτούνται για αυτή παρα μόνο τις δικές του. Επιπλέον, τα πιστοποιητικά αυτά διαχειρίζονται κεντρικά και δεν υπάρχει πρόβλεψη για δημιουργία τέτοιων πιστοποιητικών για την δημιουργία εικόνων.

Η δεύτερη επιλογή που εξεταστηκέ χρησιμοποιούσε την διεπαφή της εσωτερικής βάσης δεδομένων PuppetDB για να αναζητήσει όλα τα προγράμματα που θα πρέπει να είναι εγκατεστημένα σε έναν κόμβο LxPlus. Ο κατάλογος προγραμμάτων που παράγει αυτή η αναζήτηση περιέχει μία 1:1 αντιστοίχηση με τον κατάλογο προγραμμάτων που είναι διαθέσιμα στους κόμβους LxPlus. Το μειονέκτημα αυτής της μεθόδου είναι πως για να ανακτηθούν τα δεδομένα είναι απαραίτητο να υπάρχει τουλάχιστον ένας ενεργός κόμβος με αυτές τις παραμετρους που να τρέχει το Puppet Agent. Ο αλγόριθμος που υλοποιήθηκε πρώτα ανακτά μια λίστα με όλους τους κόμβους με την ίδια ομάδα παραμετροποιήσεων που απαιτείται για την εικόνα. Έπειτα, ανακτά όλες τις εντολές εγκατάστασης πακέτων που αντιστοιχούν σε αυτή την ομάδα παραμετροποιήσεων φιλτράροντας αυτές ώστε να κρατήσει μόνο αυτές που αντιστοιχούν στον κατάλογο προγραμμάτων που προορίζονται για διαδραστική χρήση στην υπηρεσία.

Επιπλέον, εκτός από την εγκατάσταση των αντίστοιχων πακέτων είναι απαραίτητη η παραμετροποίηση πολλών λογισμικών με παραμέτρους αντίστοι-

χες με αυτές στην υπάρχων υπηρεσία. Αυτό επιτυγχάνεται μέσω αρχεία παραμέτρων που δημιουργούνται μέσω καλουπιών στο αποθετήριο των παραμέτρων του συστήματος Puppet. Λόγω του γεγονότος πως τα αρχεία παραμέτρων βρίσκονται σε μομφή καλουπιού Puppet, η οποία χρησιμοποιεί μία εσωτερική templating language στην οποία δεν δίνεται η δυνατότητα να χρησιμοποιηθεί ανεξάρτητα. Αντί για αυτό, συγκεντρώθηκαν όλα τα πλήρη αρχεία ρυθμήσεων στο αποθετήριο και χρησιμοποιήθηκε το εργαλείο templating Kapitan.

Τέλος δημιουργήθηκε ένα build pipeline στην υπηρεσία αποθετηρίου η οποία αυτόματα παράγει την εικόνα container βασιμένο στην λίστα πακέτων και αρχεία παραμέτρων που προαναφέρθηκαν. Η τελική εικόνα ανεβάζεται στο κεντρικό αποθετήριο εικόνων.

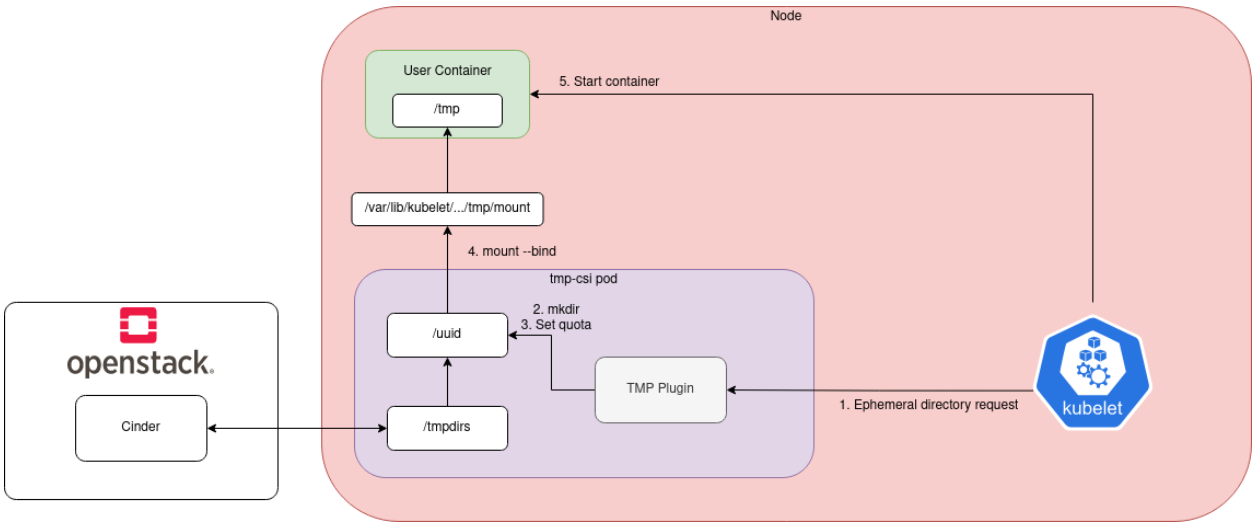
8.3 Κατανεμημένη εγκατάσταση σε Kubernetes

Για την εγκατάσταση στο σύστημα Kubernetes χρησιμοποιήθηκε ο διαχειριστής πακέτων Helm. Δημιουργήθηκε ένα κεντρικό πακέτο (chart) το οποίο παραμετροποιεί και εκτελεί την εικόνα container του ContainerSSH. Με τις προϋπάρχουσες παραμέτρους δημιουργούνται 4 ανεξάρτητα container για το ContainerSSH για την κατανομή του φόρτου. Επιπλέον, το πακέτο αυτό εγκαταστεί και ενεργοποιεί έναν Load Balancer ο ποίος χρησιμοποιείται ως το σημείο εισόδου των χρηστών από το εξωτερικό δίκτυο στο δίκτυο container του kubernetes, και επιπλέον κατανέμει τις συνδέσεις τυχαία ανάμεσα στο σύνολο των container της υπηρεσίας που είναι διαθέσιμα.

Επιπλέον, για την αυτόματη ανανέωση των παραμετροποιήσεων και αυτόματη αναβάθμιση σε νεότερες εκδόσεις χρησιμοποιήθηκε το εργαλείο Continuous Delivery Flux (flux.io). Το εργαλείο αυτό μετά την εγκατάσταση του σε έναν Kubernetes Cluster παρακολουθεί συνεχόμενα τις αλλαγές στο αποθετήριο κώδικα και αυτομάτως ανανεώνει τις παραμετροποιήσεις αν αυτές αλλάξουν.

8.4 Σύστημα προσωρινών αρχείων

Στην υπηρεσία LxPlus παρέχεται ο κατάλογος προσωρινών αρχείων /tmp στον οποίο οι χρήστες μπορούν να αποθηκεύουν αρχεία. Λόγω του μικρού απο-



Σχήμα 13: Διάγραμμα ροής οδηγού εφήμερων αρχείων

θηκευτικού χώρου ο κατάλογος αυτός στην υπάρχων υπηρεσία βρίσκεται σε εξωτερική μονάδα αποθήκευσης (block storage device). Αντίστοιχα, οι κόμβοι Kubernetes που χρησιμοποιούνται έχουν μικρό και περιορισμένο τοπικό αποθηκευτικό χώρο οπότε είναι απαραίτητο να τοποθετηθούν τα προσωρινά αρχεία σε μία εξωτερική μονάδα.

Το σύστημα Kubernetes προσφέρει ήδη αυτή την λειτουργία με τον τύπο συστήματος αρχείου emptyDir. Αυτή η λειτουργία προσφέρει έναν προσωρινό κατάλογο αρχείων ο οποίος μπορεί να παραμετροποιηθεί για να χρησιμοποιεί είτε την μνήμη (RAM) του συστήματος ή τον εσωτερικό σκληρό δίσκο. Ο κατάλογος αυτός δημιουργείτε κατά την εκκίνησή ενός container και διαγράφεται αυτόματα κατά την καταστροφή του. Όμως, δεν υποστηρίζεται με αυτό η αποθήκευση σε εξωτερικό το οποίο είναι απαραίτητο για την υπηρεσία.

Για την ικανοποίηση αυτής της περίπτωσης χρήσης χρειάστηκε να αναπτυχθεί ένας ειδικός οδηγός αρχείων με την χρήση της διεπαφής CSI(Container Storage Interface). Ο οδηγός αυτός κατά την αρχικοποίηση του δημιουργεί μία μονάδα αποθήκευσης χρησιμοποιώντας την διεπαφή Cinder της υπηρεσίας OpenStack. Μόλις δημιουργηθεί ένα container χρήστη και ο οδηγός λάβει αίτημα για την δημιουργία ενός καινούργιου καταλόγου αυτός δημιουργεί έναν κατάλογο στην μονάδα αποθήκευσης με τυχαίο όνομα και αναθέτει (mounts) τον κατάλογο αυτόν στον κατάλογο /tmp του container.

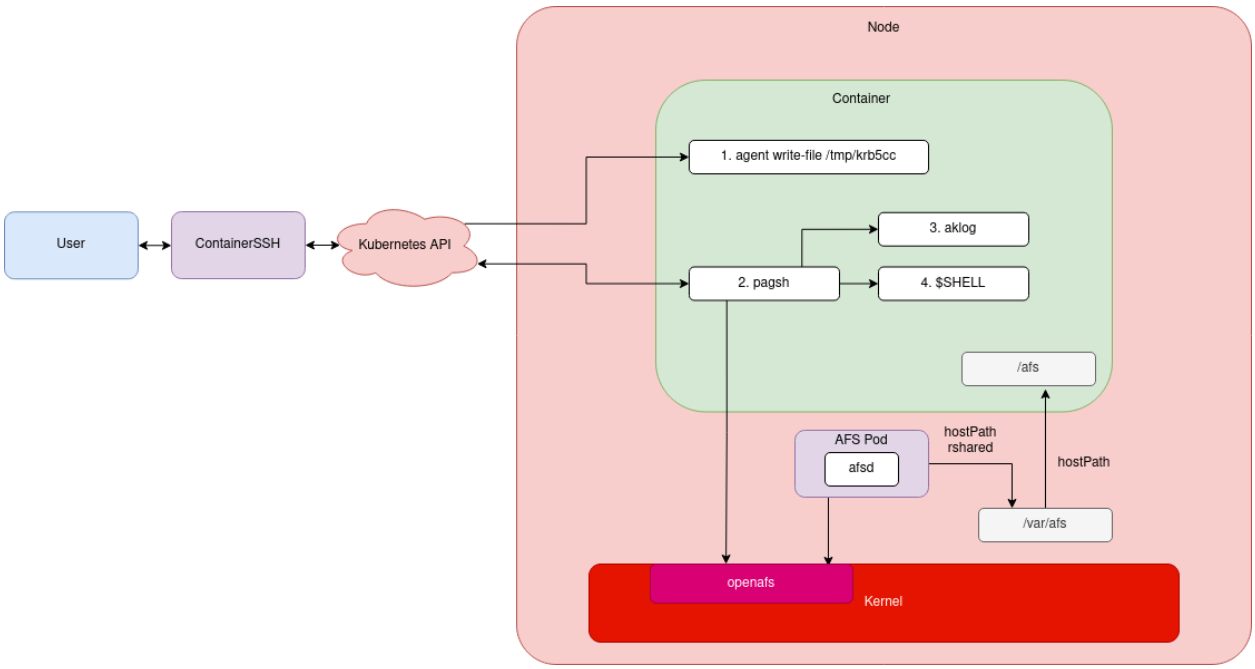
Η διαδικασία αυτή περιγράφεται στο σχήμα 13.

8.5 Σύστημα αρχείων AFS σε Kubernetes

Το σύστημα αρχείων AFS είναι ένα από τα τρία συστήματα αρχείων δικτύου που παρέχονται στην υπηρεσία LxPlus. Το AFS είναι το πιο σημαντικό από αυτά επειδή χρησιμοποιείτε ως την κύρια τοποθεσία αποθήκευσης προσωπικών αρχείων των χρηστών (user home directories). Για την αρχικοποίηση του περιβάλλοντος ενός χρήστη είναι απαραίτητο να αναγνωστούν αρχεία από τον προσωπικό τους κατάλογο και ως αποτέλεσμα το σύστημα αρχείων AFS είναι αυτό που πρέπει να έχει αρχικοποιηθεί πριν ξεκινήσει ο φλοιός του χρήστη.

Το AFS βασίζεται πάνω σε ένα kernel module και ένα userspace daemon, και τα δύο πρέπει να είναι εγκατεστημένα και ενεργά για την λειτουργία του. Το λειτουριγκό σύστημα των κόμβων του Kubernetes βασίζεται πάνω στο CoreOS και η εικόνα του συστήματος δεν είναι εύκολη να τροποποιηθεί. Για αυτό, αποφασιστικέ να εγκατασταθεί το AFS μέσω ενός daemonset εντός του Kubernetes αντί να εγκατασταθεί απομονωμένα σε κάθε κόμβο. Χρησιμοποιήθηκε ένα helm chart το οποίο εγκαταστεί το daemonset το οποίο είναι παραμετροποιημένο για να δημιουργεί ένα container αυτού σε κάθε κόμβο του σμήνους. Η εικόνα του container περιέχει το userspace daemon του AFS καθώς και τον πηγιαίο κώδικα του kernel module. Για την σωστή λειτουργία του kernel module απαιτείτε η έκδοση του να ταιριάζει την έκδοση kernel που τρέχει στον κόμβο και για αυτό οι εκδόσεις των εικόνων container AFS ονομάζονται ανάλογα με την έκδοση kernel. Κατά την αρχικοποίηση του container, δίνεται πρόσβαση στον κατάλογο με τα headers του kernel του κόμβου τα οποία μαζί με τον πηγιαίο κώδικα του AFS kernel module μεταγλωττίζονται για να παράξουν το τελικό kernel module το οποίο φορτώνεται στο kernel του κόμβου.

Το ContainerSSH τοποθετεί το Kerberos Ticket του χρήστη μέσα στο container το οποίο μπορεί να χρησιμοποιηθεί για την αρχικοποίηση της συνεδρίας του χρήστη στο AFS. Για την αρχικοποίηση της συνεδρίας χρησιμοποιείτε το εκτελέσιμο aklog το οποίο δημιουργεί το AFS Token και το αποθηκεύει στο secret store του kernel δένοντας το με το UID(User ID) του χρήστη. Το AFS δεν αναγνωρίζει το όριο και την απομόνωση μεταξύ των container και λόγω αυτού αν κάποιος χρήστης καταφέρει και πάρει δικαιώματα διαχειριστή μέσω ενός σφάλματος ασφαλείας και αλλάξει το UID του τότε θα μπορεί να έχει πρόσβαση στους χώρους AFS των άλλων χρηστών στον ίδιο κόμβο. Για την επίλυσή αυτού χρησι-



Σχήμα 14: Διάγραμμα ροής δεδομένων για το σύστημα AFS σε Kubernetes

μοποιείται το εκτελέσιμο pagsh, που είναι επίσης μέρος του AFS. Το PAG (Process Authentication Group) χρησιμοποιούν τα GIDs (group ids) για να αποθηκεύσουν το AFS Token αντί για το UID. Αυτό έχει ως αποτέλεσμα πως μόνο η διεργασία που κάλεσε το πρόγραμμα καθώς και όλες η υποδιεργασίες της θα έχουν πρόσβαση στο Token αλλά οποιαδήποτε άλλη ανεξάρτητη διεργασία θα πρέπει να δημιουρήσει το δικό της token. Αυτό διορθώνει το πρόβλημα ασφαλείας μεταξύ των container αλλά αφού κάθε shell του χρήστη είναι ανεξάρτητο σημαίνει πως θα πρέπει να δημιουργηθεί ένα PAG για κάθε session, που μπορεί να προκαλέσει προβλήματα με την ανανέωση των token σε κάποιες περιπτώσεις. Π.χ. μπορεί να λήξει το token σε ένα από τα PAGs αλλά όχι στα υπόλοιπα που σημαίνει πως ο χρήστης δεν θα μπορεί να έχει πρόσβαση στον χώρο AFS από την μία διεργασία αλλά θα μπορεί από την δεύτερη.

8.6 Σύστημα αρχείων CernVM-FS και EOS σε Kubernetes

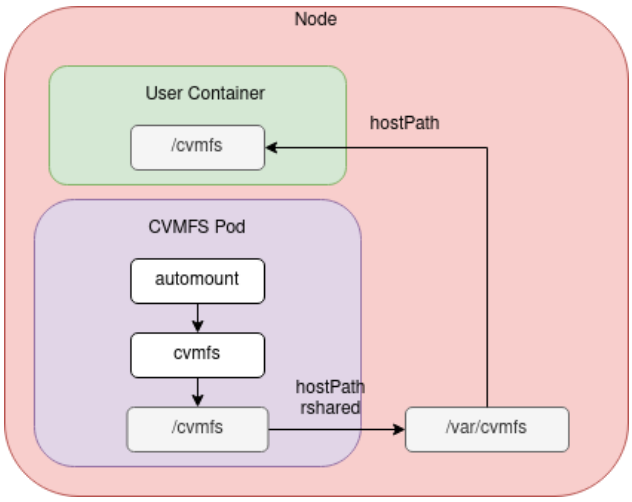
Τα συστήματα αρχείων CVMFS και EOS χρησιμοποιούνται για την κατανομή λογισμικού και την παροχή ενός δευτερεύοντα αποθηκευτικό χώρο για δεδομένα μεγάλου όγκου αντίστοιχα. Και τα δύο συστήματα αυτά βασίζονται πάνω στην διεπαφή Filesystem in USErspace (FUSE) του Linux Kernel και λόγω αυτού

δεν χρειάζονται δικό τους kernel module, κάνοντας την εγκατάσταση τους σε Kubernetes πιο απλή από αυτή του AFS. Η ιδιαιτερότητα αυτών των συστημάτων είναι πως αντίθετα με το AFS που αποτελείτε από ένα mount, το CVMFS και EOS αποτελούνται από μια μεγάλη συλλογή από διαφορετικά mounts/αποθετήρια που το καθένα χρησιμοποιείτε για διαφορετικό σκοπό. Για παράδειγμα, το αποθετήριο home-n του συστήματος EOS περιέχει δευτερεύοντα αποθηκευτικό χώρο για όλους τους χρήστες που το όνομα τους ξεκινάει από n. Το αποθετήριο cms.cern.ch του συστήματος CVMFS περιέχει διανομές λογισμικών που χρησιμοποιούνται για το πείραμα CMS.

Λόγο του μεγάλου όγκου αποθετηρίων καθίσταται μη πρακτικό να είναι μόνιμα προσβάσιμα όλα τα αποθετήρια ταυτόχρονα. Στην υπάρχων υπηρεσία χρησιμοποιήται το λογισμικό automount για την ενεργοποίηση κάθε αποθετηρίου μόνο όταν αυτό χρειάζεται. Το automount παρακολουθεί τους καταλόγους /cvmfs και /eos αντίστοιχα και όταν κάποιος χρήστης ζητήσει να διαβάσει κάποιο αρχείο από έναν υποκατάλογο αυτών, αν το αποθετήριο στο οποίο ο κατάλογος αυτός βρίσκεται δεν είναι ήδη προσβάσιμο το automount το ενεργοποιεί. Από την οπτική του χρήστη αυτό είναι εντελώς διαφανές με εξαίρεση μία καθυστέρηση μερικών δευτερολέπτων κατά την πρώτη ανάγνωση αρχείων σε κάθε αποθετήριο. Αν ένα αποθετήριο δεν έχει χρησιμοποιηθεί για μία χρονική περίοδο (π.χ. 1 ώρα) αυτό αφαιρείτε αυτόματα και θα επαναενεργοποιηθεί όταν αναζητηθεί από κάποιον χρήστη.

Για την προσαρμογή αυτού του συστήματος στο σύστημα Kubernetes αποφασίστηκε να διατηρηθεί η ίδια αρχιτεκτονική χρησιμοποιώντας και πάλι το automount. Χρησιμοποιείτε και πάλι ένα DaemonSet για την εκτέλεση ενός container για κάθε σύστημα αρχείων σε κάθε κόμβο. Οι παράμετροι διαχειρίζονται κεντιρκά μέσω πακέων Helm και του συστήματος Flux που περιγράφηκε νωρίτερα. Ο κατάλογος που παρακολουθεί το automount εξάγεται μέσω της επιλογής hostPath του Kubernetes στο κεντρικό σύστημα αρχείου του κόμβου. Από εκεί, κάθε container χρήστη μπορεί να το συμπεριλάβει στους καταλόγους τους χρησιμοποιώντας επίσης την επιλογή hostPath.

Για την σωστή λειτουργία του automount χρειάστηκε να απενεργοποιηθούν κάποιες παράμετροι ασφαλείας από την μεριά του container που τρέχει το automount. Συγκεκριμένα, το automount container είναι απαραίτητο να τρέχει σε μορφή priviledged το οποίο του επιτρέπει να κάνει mount καταλόγους και να



Σχήμα 15: Ροή καταλόγων που διαχειρίζονται από automount στο σύστημα Kubernetes

επικοινωνήσει με την διεπαφή FUSE. Επίσης, το πρωτόκολλο που χρησιμοποιεί το automount για την ανίχνευση των γεγονότων πρόσβασης και την ενεργοποίηση αποθετηρίων χρησιμοποιεί το Process ID (PID) της διεργασίας για την ανήχνευση του automount daemon και την προώθηση των γεγονότων στην σωστή διεργασία. [The Go Authors, 2022] Για αυτό το λόγο είναι επίσης απαραίτητο να απενεργοποιηθεί η απομόνωση χώρου PID (PID Namespace) στο container που τρέχει το automount.

9 Μελλοντικές βελτιώσεις

Σε αυτήν την εργασία επεκτάθηκε το λογισμικό ContainerSSH για την υποστήριξη επιχειρησιακών συστημάτων αφθεντικοποίησης και επιπλέον υλοποιήθηκε μηχανισμός προώθησής συνδέσεων δικτύου σε αυτό. Τέλος, υλοποιήθηκε το υπόβαθρο που απαιτείτε για να τεθεί σε λειτουργία μία κοντεΐνεροποιημένη υπηρεσία διαδραστικής σύνδεσης. Το σύστημα με την αρχιτεκτονική που περιγράφηκε βρίσκεται τώρα σε πιλοτική λειτουργία και ελέγχεται για την σταθερότητα και αξιοπιστία του. Είναι σημαντικό να δοκιμαστεί η αντοχή της υπηρεσίας σε σχέση με τον αριθμό των ταυτόχρονα συνδεδεμένων χρηστών που μπορεί να διαχειριστεί καθώς και η ευκολία ανάκαμψης αυτής σε περίπτωση σφάλματος πριν η υπηρεσία διατεθεί για γενική χρήστη στους χρήστες της υπάρχουσας υπηρεσίας LxPlus.

Στο μέλλον υπάρχει πιθανότητα για περισσότερη έρευνα και ανάπτυξη βελτιστοποιώντας το πρωτοκόλλου προώθησης με επέκταση η οποία αξιοποιεί περισσότερες από μία συνδέσεις μεταξύ του ContainerSSH και του Agent. Μία τέτοια επέκταση θα μπορούσε να δημιουργεί μία σύνδεση δικτύου μεταξύ του ContainerSSH και του Agent για κάθε προωθημένη σύνδεση. Θα είναι απαραίτητο η δημιουργία ενός ασφαλούς πρωτοκόλλου αρχικοποίησης αυτών των συνδέσεων έτσι ώστε να μπορούν να αρχικοποιηθούν και να χρησιμοποιηθούν μόνο από τον χρήστη για τον οποίο προορίζονται για να αποφευχθούν προβλήματα ασφαλείας.

Αναφορές

[rfc, 1987] (1987). X Window System Protocol, version 11: Alpha update April 1987. RFC 1013.

[Blomer et al., 2011] Blomer, J., Aguado-Sánchez, C., Buncic, P., and Harutyunyan, A. (2011). Distributing lhc application software and conditions databases using the cernvm file system. In *Journal of Physics: Conference Series*, volume 331, page 042003. IOP Publishing.

[Bormann and Hoffman, 2020] Bormann, C. and Hoffman, P. E. (2020). Concise Binary Object Representation (CBOR). RFC 8949.

[Espinal et al., 2021] Espinal, X., Moscicki, J., Wiebalck, A., Peters, A. J., and Van der Ster, D. (2021). Future of user storage at cern. Technical report.

[Iven et al., 2017] Iven, J., Lamanna, M., and Pace, A. (2017). Cern's afs replacement project. In *Journal of Physics: Conference Series*, volume 898, page 062040. IOP Publishing.

[Jonathan Turner, 2022] Jonathan Turner (2022). gokrb5 package documentation. https://pkg.go.dev/github.com/jcmturner/gokrb5/v8. Accessed: 2022-04-08.

[Linn, 1993] Linn, J. (1993). Generic Security Service Application Program Interface. RFC 1508.

[Linn, 1996] Linn, J. (1996). The Kerberos Version 5 GSS-API Mechanism. RFC 1964.

[Lonvick and Ylonen, 2006a] Lonvick, C. M. and Ylonen, T. (2006a). The Secure Shell (SSH) Connection Protocol. RFC 4254.

[Lonvick and Ylonen, 2006b] Lonvick, C. M. and Ylonen, T. (2006b). The Secure Shell (SSH) Protocol Architecture. RFC 4251.

[Mascetti et al., 2020] Mascetti, L., Rios, M. A., Bocchi, E., Vicente, J. C., Cheong, B. C. K., Castro, D., Collet, J., Contescu, C., Labrador, H. G., Iven, J., et al. (2020). Cern disk storage services: report from last data taking, evolution and future

outlook towards exabyte-scale storage. In *EPJ Web of Conferences*, volume 245, page 04038. EDP Sciences.

[Merkel et al., 2014] Merkel, D. et al. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

[Naredo and Pardavila, 2017] Naredo, I. R. and Pardavila, L. L. (2017). Dns load balancing in the cern cloud. In *Journal of Physics: Conference Series*, volume 898, page 062007. IOP Publishing.

[Neuman and Ts'o, 1994] Neuman, B. and Ts'o, T. (1994). Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33--38.

[Neuman et al., 2005] Neuman, D. C., Hartman, S., Raeburn, K., and Yu, T. (2005). The Kerberos Network Authentication Service (V5). RFC 4120.

[Salowey et al., 2006] Salowey, J. A., Welch, V., Hutzelman, J., and Galbraith, J. (2006). Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol. RFC 4462.

[The Go Authors, 2022] The Go Authors (2022). The golang ssh package documentation. https://pkg.go.dev/golang.org/x/crypto/ssh. Accessed: 2022-04-08.