Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
(повна назва інституту/факультету)

Кафедра телекомунікацій
(повна назва кафедри)

«На правах рукопису»
УДК _____

До захисту допущено

В.о. завідувача кафедри

_____ Явіся В.С.___
(підпис)
(ініціали, прізвище)

“___”_____2018_р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 172 Телекомунікації та радіотехніка,
(код і назва)

спеціалізація Мобільні телекомунікації

на тему: Принципи побудови і методи практичної реалізації корпоративної мережі IP телефонії на базі Asterisk.

Виконав: студент _2_ курсу, групи ТМ-71мп
(шифр групи)

Письменний Ігор Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник _професор, к.т.н. Романов О.І._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018_ рік

ЗМІСТ

					КПІ ім.Ігоря Сікорського 4095-с.05.ТМ-71мп.2018.ПЗ		
Змн.	Лист	№ докум.	Підпис	Дата			
Розроб.		Письменний І.С.			Принципи побудови і методи практичної реалізації корпоративної мережі IP телефонії на базі Asterisk.	Літ.	Арк. / Акрушів
Перевір.		Романов О.І.					
Реценз.						ІТС НТУУ «КПІ»	
Н. Контр.		Петрова В.М.					
Затверд.		Явіся В.С.					

ВСТУП

В даний час телекомунікаційні технології перебувають на такому високому рівні розвитку, що впроваджуються в будь-які пристрої, починаючи від найпростіших комп'ютерних модемів, забезпечуючи вихід в мережу Інтернет, і закінчуючи системами віддаленого керування автомобілями або побутовою технікою, реалізовані за допомогою стільникових телефонів - смартфонів. На тлі бурхливого розвитку телекомунікацій, змінам піддаються основи будь-яких видів зв'язку - телефонні мережі, удосконалюються шляхи виходу до них, методи і системи проектування АТС, що забезпечують доступ всім абонентам до послуг зв'язку. Особливо цікаво розглянути можливості проектування і створення АТС, що використовуються повсюдно в самих різних компаніях - починаючи від міні-АТС, до послуг яких вдаються невеликі компанії, стартапи, закінчуючи потужними комунікаційними структурами, які допомагали національним і міжнародним організаціям, офіси яких розташовуються на великій відстані один від одного.

Постійне вдосконалення і поширення в усіх областях мережі Інтернет призвело до зародження нового виду зв'язку, який грунтується на використанні протоколів Інтернет і IP - серверів, здатних перетворювати в цифрову форму звичайний голосовий сигнал. Цей новий вид зв'язку назвали IP або VoIP телефонією. даний вид технології дозволив перетворити аудіо- та відео-переговори, у зручний, якісний, універсальний і що також важливо дешевий засіб спілкування, який став доступним практично кожному. В даний час IP протокол використовується не тільки в мережі Інтернет, а й в інших мережах передачі даних з пакетною комутацією (локальних, корпоративних, регіональних). І у всіх цих мережах, є можливість передавати мовленнєві повідомлення з використанням пакетів даних. такий спосіб передачі мови отримав назву IP-телефонія.

Практична можливість повної інтеграції голосу і даних поверх загальної інфраструктури обчислювальних мереж привела до появи так званої «пакетної

телефонії» - технології передачі аналогових телефонних сигналів по мережах передачі даних. Для позначення технології передачі голосу з використанням IP-мереж використовуються два основні терміни: IP-телефонія (IP Telephony) або Voice over IP-VoIP.

IP-телефонія використоввує технологію, що дозволяє використовувати будь-яку мережу з пакетною комутацією на базі протоколу IP як засіб організації і ведення міжнародних, міжміських і місцевих телефонних розмов і передачі факсів в режимі реального часу.

Створення корпоративної телефонної мережі (КТС) є обов'язковим завершальним кроком для створення гнучкої динамічної телекомунікаційної інфраструктури, яка дозволить швидко організовувати і динамічно підтримувати тимчасові колективи, робочі групи, організовувати як внутрішній зв'язок, так і зовнішні взаємодії.

Актуальність розвитку КТС на основі IP-телефонії обумовлена не тільки можливістю зниження витрат на телефонні переговори і технічне обслуговування інфраструктури (і це, безумовно, має значення). У стратегічному плані IP-телефонія є єдиною технічною платформою, яка дозволить об'єднати рішення для передачі даних і голосу, а також для обробки і подальшого використання цієї інформації в усіх бізнес-процесах.

IP-телефонія дозволяє істотно економити необхідну смугу пропускання каналів, що неминуче веде до зниження тарифів, особливо на міжміські і міжнародні телефонні розмови. технологія IP - телефонії надає можливість отримання високоякісного телефонного з'єднання між двома і більше абонентами, що знаходяться в різних куточках планети. Телефонні переговори, організовані з допомогою VoIP технології є більш надійними і універсальними в порівнянні з традиційними, так як замість проміжних комутаторів, станцій та інших пристроїв використовується програмний продукт з відкритим вихідним кодом Asterisk.

Мета – розглянути існуючі принципи побудови корпоративних мереж передачі даних на базі технології VoIP та виокремити методи практичної реалізації використовуючи наявну інфраструктуру та додаткове програмне забезпечення, а саме програмний продукт з відкритим вихідним кодом Asterisk, технології віртуалізації Vagrant, KVM/QEMU, libvirt, технологію контейнеризації Docker, системи моніторингу Glances, cAdvisor, Prometheus/Grafana.

Об’єктом дослідження є програмне рішення комп'ютерної телефонії Asterisk (в тому числі, VoIP) з відкритим вихідним кодом.

Предметом дослідження є методи реалізація IP-телефонії на базі платформи Asterisk з використанням технологій віртуалізації та контейнеризації.

Проблема, що вирішується - оцінка продуктивності реалізації ПО Asterisk з використанням технології віртуальних машин, контейнерів за допомогою відповідних інструментів дослідження стану в реальному часі.

1. ТЕОРЕТИЧНІ ПИТАННЯ ПОБУДОВИ МЕРЕЖ IP-ТЕЛЕФОНІЇ

1.1 Історичні умови появи IP-телефонії

VoIP, також відома як IP-телефонія - це передача голосових сигналів у реальному часі за допомогою Інтернет-протоколу (IP) через Інтернет або приватну мережу даних1. У більш простій формі VoIP перетворює голосовий сигнал з вашого телефону в цифровий сигнал, який надалі розповсюджується мережею Інтернет. Однією з найважливіших переваг VoIP (над традиційною комутованою телефонною мережею загального користування PSTN) є те, що можна здійснити телефонний дзвінок на великі відстані та обійти плату за використання. Це інтегроване рішення для голосу/даних дозволяє великим організаціям (з фінансуванням для здійснення передачі з застарілих мереж у мережу VoIP) для передачі голосових програм через їхні існуючі мережі передачі даних.

Цей технологічний прогрес не лише вплине на велику традиційну галузь телекомунікацій, але й змінить структуру ціноутворення та вартості традиційної телефонії. Крім того, у порівнянні з послугами комутованої мережі, IP-мережі можуть нести від 5 до 10 разів більше голосових дзвінків з тією ж пропускною здатністю.

З кінця 90-х років такі термін, як VoIP, Інтернет-телефонія, IP-телефонія дають нам змогу здійснювати телефонні дзвінки через мережу Інтернет. Тобто це дозволяє використовувати протокол IP. Під IP-телефонією мається на увазі набір комунікаційних протоколів, технологій і методів, що забезпечують традиційні для телефонії набір номера, дзвінок і двухстороннє голосове спілкування, а також відеоспілкування через мережу Інтернет або через будь-які інші IP-мережі. Сигнал по каналу зв'язку передається в цифровому вигляді і, як правило, перед передачею перетворюється для того, щоб видалити збитковість інформації і знизити навантаження на мережу передачі даних.

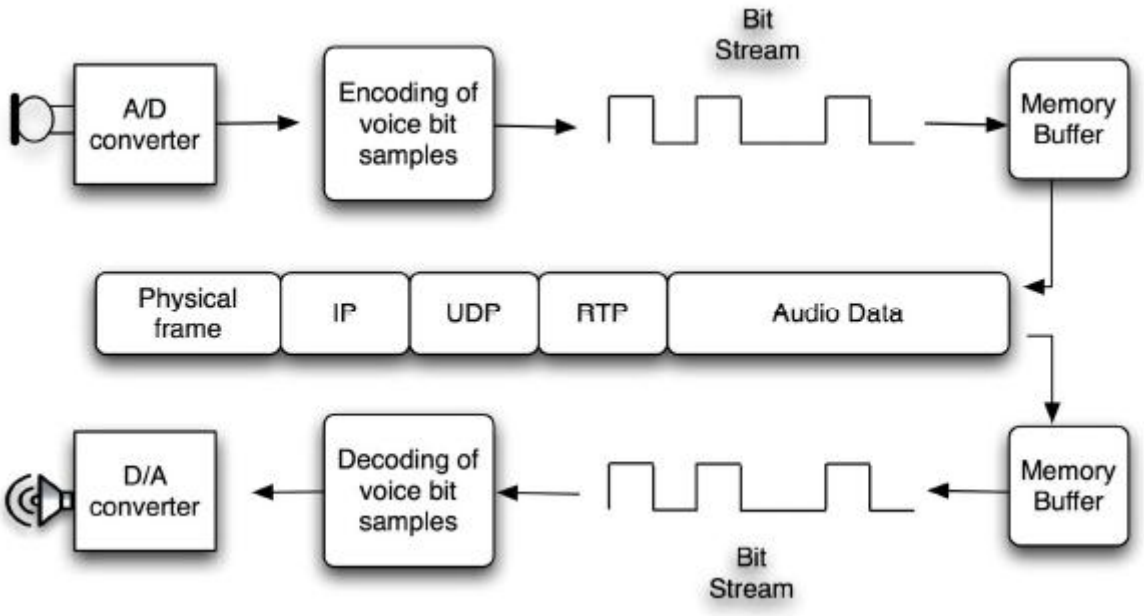


Рис. 1.1 Основна функціональна архітектура VoIP

Технологічні основи VoIP були закладені в 1925-1928 рр., Коли в надрах телекомунікаційної компанії AT&T було створено електронний синтезатор мовлення, він же Vocoder. Пристрій аналізував звуки, створені людиною, і відтворював аналог людської мови. Vocoder активно використовувався для передачі секретної інформації під час Другої Світової війни.

У 1988 році також відбулася знакова подія з'явився перший Wideband Audio Codec, знаменитий кодек G.722, з якістю, порівняною з людською вимовою, переданою телефонною мережею загального користування PSTN. Цей широкосмуговий кодек мав бітрейт, який вдвічі перевищував бітрейти попереднього G.711 і видавав прекрасний на ті часи звук.

У 1991 році засновник Autodesk Джон Уолкер створює схему для VoIP, яка вимагає пропускну здатність всього 32 кбіт/с (норма на той час була 64 кбіт/с) і робить публічний реліз програми NetFone (пізніше її перейменують в Speak Freely), яка і стала першим в світі VoIP-телефоном. Хоча спочатку єдине, для чого використовувався NetFone, був зв'язок всередині компанії Уолкера. У 1993 році з'являється перша система відеоконференц зв'язку Telepresence System, яку творці назвали просто Teleport, але швидко перейменували в TeleSuite.

Крім того, в університетах і дослідницьких центрах проводилися експерименти по передачі голосу за допомогою пакетної комутації даних.

Поява і поширення мережі Інтернет практично відразу спричинила за собою прагнення користувачів спілкуватися один-на-один. Спершу з'явилася електронна пошта, потім на вершину технологій піднялася AOL, яка пізніше, в 1998 році викупила ICQ і забезпечила миттєву передачу текстових повідомлень. Не вистачало голосового зв'язку. У відповідь на вимоги мільйонів потенційних користувачів приблизно в 1995 році з'явився VoIP, протокол пакетної (а не канальної, як в традиційній телефонії) передачі голосу. Пакети йшли від однієї IP-адреси до іншої по Інтернет-протоколу. Так з'явилася IP-телефонія.

1998 рік став одним з переломних для IP-телефонії. Компанії усвідомили всі переваги такого типу зв'язку і стали розробляти комерційні рішення. Зокрема, підприємці почали відходити від рішень і продуктів ПК-ПК і стали розробляти рішення ПК-телефон і телефон-телефон для VoIP. IP-телефонія стала інтегруватися в комутаційну телефонну мережу загального користування (public switched telephone network, PSTN).

В цьому ж році з'явився перше фізичне устаткування для IP-телефонії - перші IP-комутатори, які відповідали за маршрутизацію викликів. Незважаючи на такі технічні ривки, VoIP-дзвінки за станом на 1998 рік не дотягували навіть до 1% від всього голосового трафіку. У 2000 році ця цифра ледь дійшла до 3%, а ось в 2003 стався різкий стрибок - до 25%. Телефонні дзвінки по IP-протоколу швидко знайшли імідж безкоштовних і дуже дешевих викликів на всі напрямки, незалежно від відстані. У свій час комерційні компанії експлуатували цю безкоштовність і могли транслювати рекламні ролики на початку або в середині розмови як «плату» за вільне з'єднання. Пізніше така практика припинилася.

У 1999 році з'являється перша IP-PBX (віртуальна АТС саме для VoIP, тому що віртуальна АТС для PSTN була створена кількома роками раніше) - Asterisk. Як це часто буває, Asterisk виріс з потреби компанії в продукті, який вона не може купити або який її не влаштовує. Так, Марк Спенсер, який мав власну компанію з

технічної підтримки Linux, зрозумів, що йому терміново потрібна потужна АТС для call-центру, але на ті часи це обладнання коштувало надзвичайних грошей. Тоді він створив свою IP-АТС з відкритим вихідним кодом. Після того, як Asterisk набрав популярність, Спенсер перепрофілював компанію на підтримку і розробку апаратного забезпечення для Asterisk. До сих пір Asterisk користується у розробників і бізнесу великою популярністю.

2000 рік. Сьогодні одним зі світових лідерів роботи з технологіями зв'язку є компанія Cisco. У 2000 році вона ініціювала внутрішню корпоративну міграцію на IP-телефонію своєї штаб-квартир в Каліфорнії (Сан-Хосе). За один рік 55 будівель та 20 000 осіб були переведені на IP-телефонію. Це був один з наймасштабніших проектів в галузі. Такий досвід позначився і на профілі компанії – Cisco, яка надає унікальні потужні рішення в сфері IP-телефонії та управління мережею.

2005 рік. Компанія Calypso Wireless вивела на ринок телефон C1250i, перший в світі мобільний телефон, який міг перемикатися між вишкою стільникового зв'язку GSM і доступною мережею Wi-Fi 802, використовуючи Cisco Aironet Access Point і власну запатентовану технологію Calypso Wireless ASNAP. Завдяки цьому користувачі могли створювати відеоконференції і здійснювати дзвінки по VoIP. Формально цей телефон вважався смартфоном на Windows Mobile.

У 2006 році вийшов перший мобільний додаток для IP-телефонії Truphone. Спочатку додаток було розроблено для мобільних телефонів Nokia, але незабаром було випущено для платформ iPhone, Android і BlackBerry. Додаток вмів здійснювати безкоштовні дзвінки всередині своєї мережі, відправляти текст в іншу мережу, в тому числі Skype і дзвонити на телефонну мережу загального користування. Додаток використовував SIP-протокол та міг здійснювати дзвінки через мережу Wi-Fi, а не через GSM. Пізніше компанія випустила кілька софтфонів, а в даний момент вона займається вигідними туристичними SIM-картами.

1.2 Аналіз основних протоколів мережі IP-телефонії

Протоколи IP-телефонії забезпечують реєстрацію клієнтського пристрою (шлюз, термінал або IP-телефон) на сервері або так у провайдера, виклик і/або переадресацію виклику, встановлення голосового або відео-з’єднання, передачу імені та/або номеру абонента.

1.2.1 H.323

H.323 – перший міжнародний стандарт протоколу зв’язку. Він був опублікований Сектором стандартизації електрозв’язку ITU (ITU-T) в лютому 1996 року та його поточну версію H.323v6 було затверджено в червні 2006 року. H.323 дозволяє перетворювати голос, відео та дані в пакетних мережах.

Рекомендації ITU-T, що входять в стандарт H.323, визначають порядок функціонування абонентських терміналів в мережах з ресурсом, що не гарантують якості обслуговування (QoS). Стандарт H.323 не пов'язаний з протоколом IP, однак, спочатку переважна кількість реалізацій було засновано на цьому протоколі. Набір рекомендацій визначає мережеві компоненти, протоколи і процедури, що дозволяють організувати мультимедійний зв’язок в пакетних мережах.

Архітектура H.323 базується на наступних елементах, показаних на рис. 1.2.1

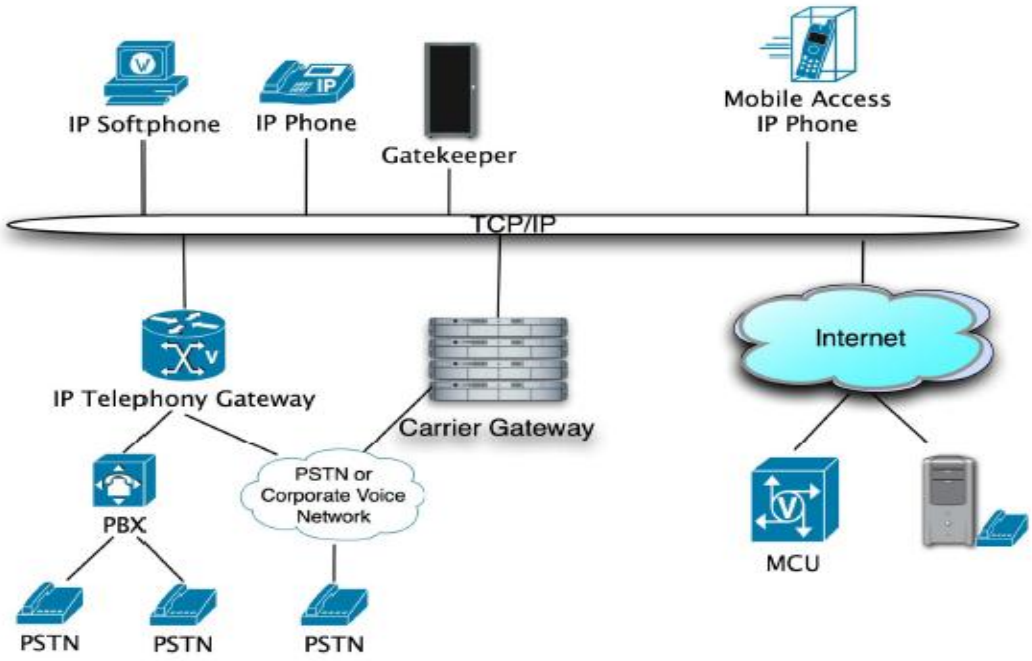


Рис. 1.2 Архітектура H.323

Термінал може являти собою ПК або автономний пристрій, здатний виконувати мультимедійний додаток. Він зобов'язаний забезпечувати зв'язок і може додатково підтримувати передачу відео або даних. Внаслідок того, що основною функцією терміналу є передача звуку, він грає ключову роль в наданні сервісу IP- телефонії. H.323 термінал повинен підтримувати такі протоколи:

- H.245 - для узгодження параметрів з'єднання;
- Q.931 - для встановлення і контролю з'єднання;
- RAS - для взаємодії з Gatekeeper;
- RTP / RTCP - для оптимізації доставки потокового аудіо (відео);
- G.711 - кодек для передачі звукової інформації (голосу, тони, факсимільних і модемних з'єднань;
- сімейство протоколів H.450 - для підтримки обов'язкових в H.323 додаткових видів обслуговування.

Додатковими компонентами можуть бути інші аудіо і відео кодеки (H.261, H.263, MPEG).

Шлюз (gateway) не входить в число обов'язкових компонентів мережі H.323. Він необхідний тільки в разі, коли потрібно встановити з'єднання з терміналом іншого стандарту. Цей зв'язок забезпечується трансляцією протоколів установки і розриву з'єднань, а також форматів передачі даних. Шлюзи H.323 мереж широко застосовуються в IP телефонії для сполучення IP мереж і цифрових або аналогових комутованих телефонних мереж.

Gatekeeper виступає в якості центру обробки викликів всередині своєї зони і виконує найважливіші функції управління викликами. Зона визначається як сукупність всіх терміналів, шлюзів і MCU. Gatekeeper - необов'язковий компонент мережі H.323, однак, якщо він присутній в мережі, то термінали і шлюзи повинні використовувати його послуги.

Основні:

- трансляція адрес - перетворення внутрішніх адрес ЛВС і телефонних номерів формату E.164 в адреси протоколів IP/IPX;
- управління доступом - авторизація доступу в H.323 мережу;
- управління смугою пропускання - дозвіл або заборона запитуваної терміналом смуги пропускання.

Додаткові:

- управління процесом встановлення з'єднання - при двосторонній конференції Gatekeeper здатний обробляти службові повідомлення протоколу сигналізації Q.931, а також може служити ретранслятором таких повідомлень від кінцевих точок;
- авторизація з'єднання - допускається відхилення Gatekeeper запиту на встановлення з'єднання. Підстави - обмеження прав або часу доступу, і інші, що лежать поза рамками H.323;
- управління викликами - Gatekeeper може відстежувати стан всіх активних з’єднань, що дозволяє управляти викликами, забезпечуючи виділення необхідної смуги пропускання і баланс завантаження мережевих ресурсів за рахунок переадресації викликів на інші термінали і шлюзи.

Сервер багатосторонньої конференції (Multipoint Control Unit) забезпечує зв'язок трьох або більше H.323 терміналів. Всі термінали, які беруть участь в конференції, встановлюють з'єднання з MCU. Сервер управляє ресурсами конференції, погоджує можливості терміналів з обробки звуку і відео, визначає аудіо та потокове відео, які необхідно направляти за багатьма адресами.

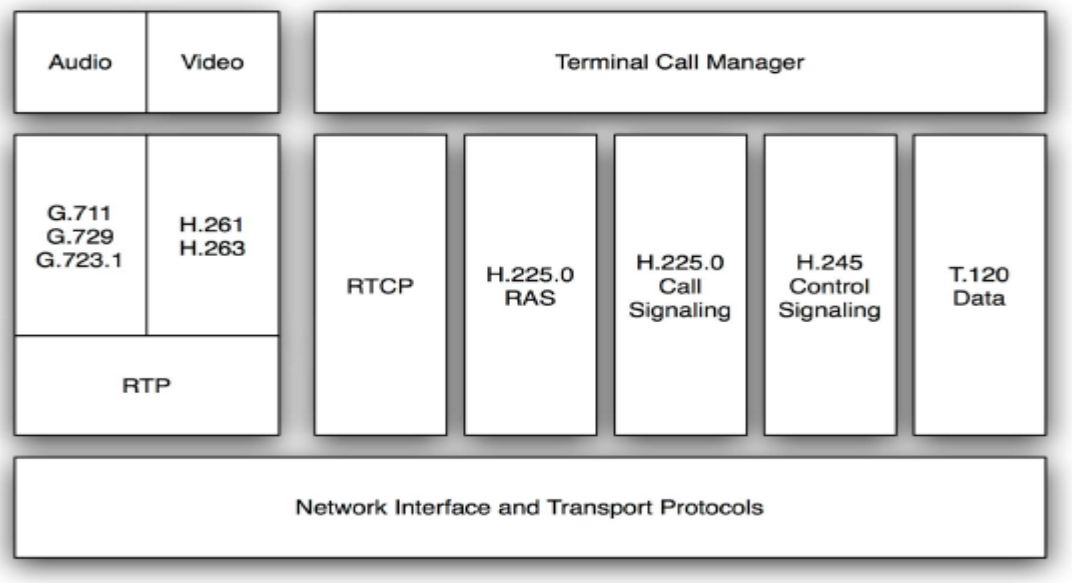


Рис. 1.3 Стек протоколу H.323

1.2.2 MGCP

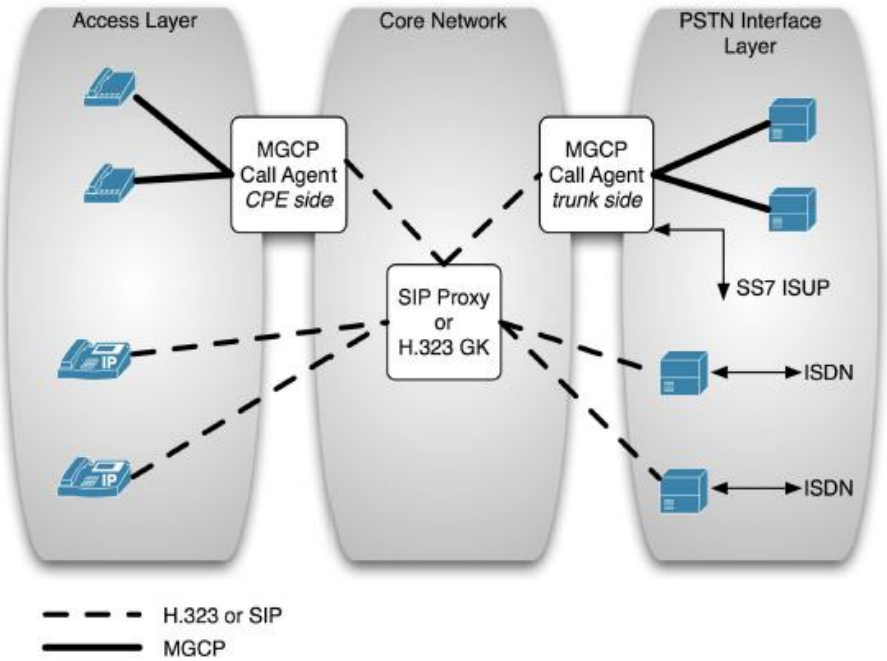


Рис. 1.4 Архітектура MGCP

MGCP або Media Gateway Control Protocol дослівно - протокол контролю медіашлюза. Є протоколом зв'язку в розподілених VoIP системах передачі голосу по протоколу IP.

Розподілені системи складаються з агента викликів - Call Agent (або контролера медіашлюза), у всякому разі з одного медіашлюза (MG) і принаймні одного сигнального шлюзу (SG), підключених до Телефонної мережі загального користування (PSTN). З точки зору мережі ОКС-7 такий симбіоз пристроїв розглядається як один вузол з одним загальним point-кодом.

Signaling Gateway (SG)

Шлюз сигналізації - забезпечує доставку сигнальної інформації, що надходить з боку мережі PSTN, до контролера шлюзів та перенесення сигнальної інформації в зворотному напрямку.

На практиці сигнальний шлюз (SG) і медіашлюзи (MG) підключені в один фізичний комутатор. Call-агент не використовує MGCP для контролю сигнального шлюзу (SG), для цих цілей - зворотного зв'язку між сигнальним шлюзом (SG) і агентом використовуються протокол SIGTRAN.

Media Gateway (MG)

Медіашлюз виконує функції перетворення інформації, що надходить з боку PSTN в голосових каналах з постійною швидкістю передачі, у вигляд, придатний для передачі по мережах з маршрутизацією пакетів IP (кодування і упаковку інформації в пакети RTP, і далі в UDP і IP) а також зворотне перетворення).

Медіашлюз використовує протокол MGCP для сигналізації подій, таких як інформація що слухавка покладена/знята або набираються цифри потрібного номеру.

Call-агент

Call-агент - контролер шлюзів, виконує функції управління шлюзами, який використовує протокол MGCP щоб повідомляти медіашлюзи:

- які події направляти Call-агенту;
- яким чином кінцеві пристрої повинні з'єднуватися один з одним;

- які тональні сигнали виклику повинні відтворюватися на кінцевих пристроях.

MGCP дозволяє також стежити Call-агенту за станом кінцевих пристроїв на медіашлюзи (MG).

Як правило, медіашлюзи сконфігуровані зі списком Call-агентів, від яких може приймати інструкції-запити.

В принципі, повідомлення можна надсилати різним агентам від кожного кінцевого пристрою (як передбачено Call-агентами, для цього використовується параметр NotifiedEntity). Бажано, щоб в даний момент усіма кінцевими пристроями керував один і той же контролер шлюзів; інші Call-агенти доступні в разі резервування ресурсів для забезпечення збитковості, якщо первинний агент відмовляє, або втрачає контакт з медіашлюзом. У разі такої відмови управління шлюзом автоматично переходить до резервного контролеру шлюзів. Все про що необхідно подбати для такого сценарію, це обмін інформацією про стан між двома агентами, однак, це не гарантує, що обидва не будуть намагатися керувати одним і тим же шлюзом. Для вирішення конфліктів використовується здатність опитувати шлюз, щоб визначити, який з агентів є керуючим в даний момент.

1.2.3 H.248/Megaco

H.248 (також MEGACO) - протокол використовуваний між елементами телекомунікаційних мереж: шлюзом (Media Gateway) і контролером шлюзів (Media Gateway Controller). Підтримує різні системи сигналізації мереж з комутацією каналів, включаючи тонову сигналізацію, ISDN, ISUP, QSIG і GSM. Закріплений як стандартний протокол IMS, поряд з SIP і Diameter.

Кожне повідомлення є транспортним механізмом передачі команд, а не самою командою, на відміну від більшості інших телекомунікаційних протоколів.

1.2.4 SCCP

SCCP - Skinny Client Control Protocol, корпоративний (пропріетарний) протокол, розроблений Selsius Corporation, в даний час належить Cisco Systems Inc., яка придбала на нього права з покупкою Selsius Corporation в 1998 році. Як нагадування про Selsius у всіх сучасних Cisco IP-телефонах стандартна назва пристрою для реєстрації в CallManager починається з SEP-це Selsius Ethernet Phone-після яких слідує MAC-адреса.

SCCP визначає набір повідомлень між Skinny-клієнтом для взаємодії провідних і бездротових IP-телефонів Cisco 7900 серії, таких як Cisco 7960, 7940, 7920, з сервером голосової пошти Cisco Unity і Cisco CallManager. Останній забезпечує сигналізацію не тільки SCCP але і більшості VoIP протоколів - H.323, SIP, і MGCP.

Крім стандартних сигнальних функцій управління викликами, які забезпечуються Skinny, потрібні були додаткові розширені опції, такі як: трансфер дзвінка, перехоплення дзвінка (пікап), конференції та повідомлення голосової пошти. Всі ці опції були неможливі при використанні чистого Skinny протоколу, що і послужило причиною створення SCCP, тому його називають іноді розширеним Skinny.

Skinny використовує TCP/IP як транспортний протокол для сигналізації викликів і контролю з'єднання, і RTP/UDP/IP як медіа - real time audio.

SCCP підтримується деякими сторонніми виробниками обладнання та програмного забезпечення, наприклад - Symbol Technologies, IPBlue і SocketIP. Skinny/SCCP використовується також в платформах з відкритим кодом - Asterisk IP-PBX.

1.2.5 RTP/RTCP

Перший протокол відповідає за визначення стандартного формату пакета, використовуваного для передачі звуку і відеозображення за допомогою інтернет-мережі. Другий протокол відповідає за управління транспортного протоколу.

1.3 Основні поняття IP-телефонії

1.3.1 Шлюзи IP-телефонії

VoIP-шлюз може мати кілька аналогових або цифрових інтерфейсів. Аналогові інтерфейси - FXS і FXO - потрібні для підключення аналогових телефонних апаратів, АТС або аналогових телефонних ліній (PSTN). Цифровий інтерфейс E1 призначений для передачі даних зі швидкістю 64 кбіт/с. У деяких моделях є місце для підключення резервної аналогової лінії.

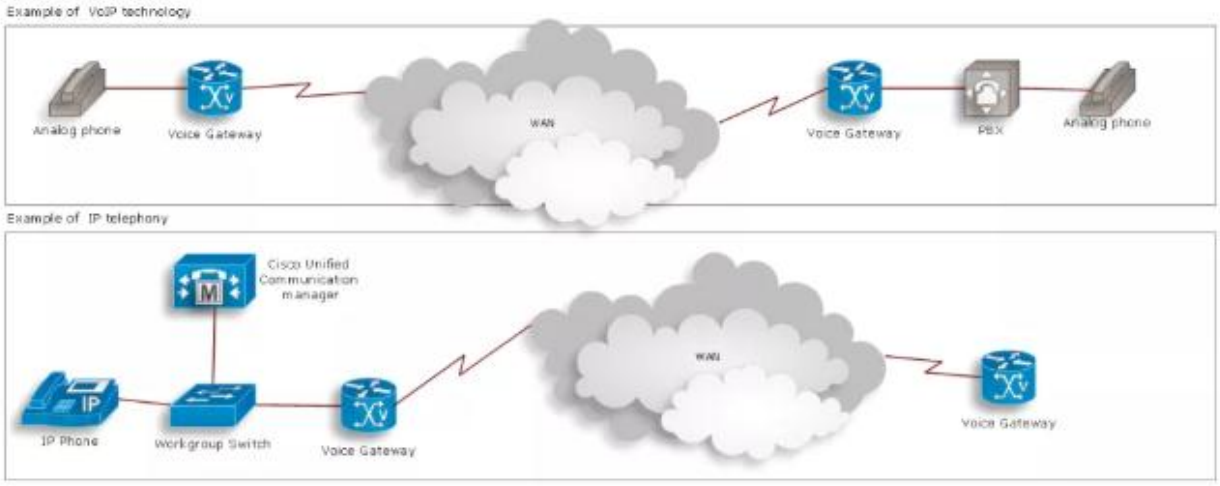


Рис. 1.5 Приклади топології IP-телефонії

VoIP-шлюз підключається на вільний порт АТС, на зовнішню лінію або на вільну внутрішню лінію офісної АТС. Далі надаються налаштування, прописуються правила маршрутизації, згідно з якими одні дзвінки будуть відправлятися безпосередньо на міську АТС, а інші через VoIP-шлюз в IP-мережу.

Класична організація телефонії в офісі виглядає наступним чином. Офісна АТС підключена до міської мережі (обмежене кількістю зовнішніх лінії) і

підтримує певне число внутрішніх абонентів. Дзвінки між ними йдуть по внутрішній лінії і безкоштовні. Дзвінки на зовнішні номери обслуговує міська АТС і стягує за них плату, згідно обраного тарифу. Міжміські та міжнародні дзвінки обслуговуються регіональними операторами телекомунікації. При використанні шлюзів і системи IP-телефонії, обмеження залишається лише на кількості зовнішніх ліній, інші обмеження є умовними, залежить від вибору IP-АТС.

1.3.2 Способи підключення до IP-телефонії

Підключення до IP-телефонії, що здійснюється будь-яким оператором телефонії, реалізується, як правило, двома способами:

• підключення VoIP телефонії за допомогою SIP телефонів (рис. 1.6);

• підключення VoIP телефонії за допомогою VoI-шлюзу (малюнок 1.7).

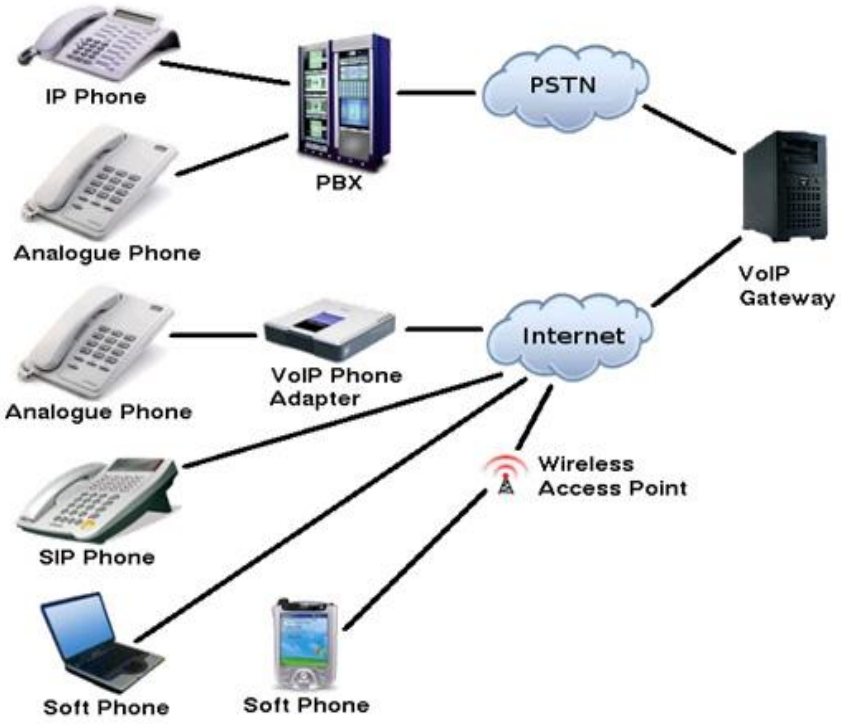


Рис. 1.6 Підключення VoIP телефонії за допомогою SIP телефонів

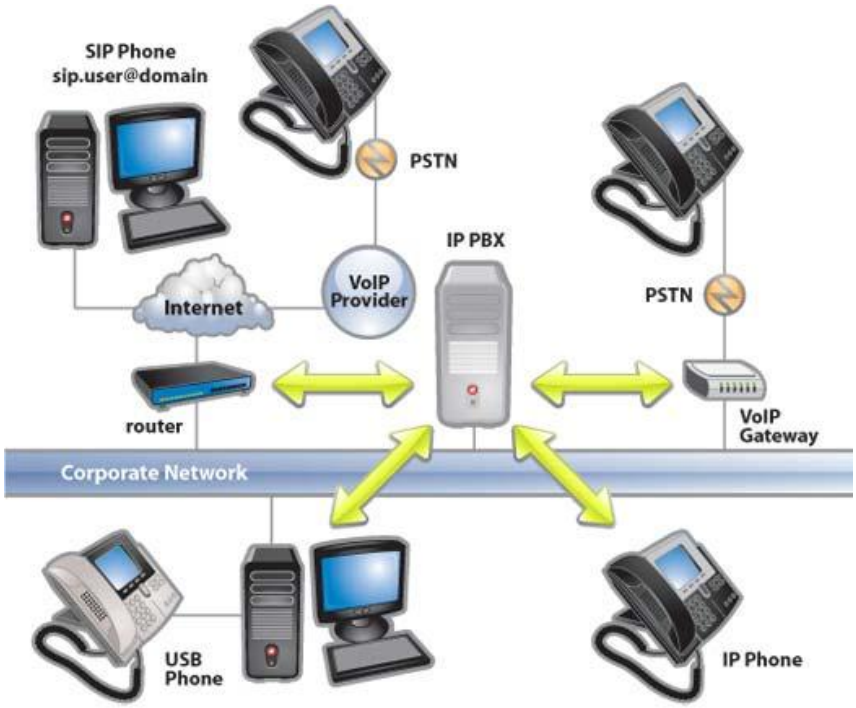


Рис. 1.7 Підключення VoIP телефонії за допомогою VoIP-шлюзу

1.3.3 Протокол SIP в IP-телефонії

SIP - протокол передачі даних, що описує спосіб завершення призначеного для користувача інтернет-сеансу, що включає обмін мультимедійним вмістом (IP-телефонія, відео-та аудіоконференції, миттєві повідомлення, онлайн-ігри).

Протокол описує, яким чином клієнтську програму може запросити початок з'єднання в іншого, можливо, фізично віддаленого клієнта, що знаходиться в тій же мережі, використовуючи його унікальне ім'я. Протокол визначає спосіб узгодження між клієнтами про відкриття каналів обміну на основі інших протоколів, які можуть використовуватися для безпосередньої передачі інформації (наприклад, RTP). Допускається додавання або видалення таких каналів протягом встановленого сеансу, а також підключення та відключення додаткових клієнтів (тобто допускається участь в обміні більше двох сторін - конференц-зв'язок). Протокол також визначає порядок завершення сеансу.

В основу протоколу були закладені такі принципи:

- простота: включає в себе тільки шість методів (функцій);

- незалежність від транспортного рівня, може використовувати UDP, TCP, ATM і т. д.;
- персональна мобільність користувачів. Користувачі можуть переміщатися в межах мережі без обмежень. Це досягається шляхом присвоєння користувачу унікального ідентифікатора. При цьому набір послуг, що надаються залишається незмінним. Про свої переміщення користувач повідомляє за допомогою повідомлення REGISTER свого облікового записуl;
- масштабованість мережі. Структура мережі на базі протоколу SIP дозволяє легко її розширювати і збільшувати число елементів;
- можливість розширення протоколу. Протокол характеризується можливістю доповнювати його новими функціями при появі нових послуг;
- Інтеграція в стек існуючих протоколів Інтернет. Протокол SIP є частиною глобальної архітектури мультимедіа, розробленої комітетом IETF. Крім SIP, ця архітектура включає в себе протоколи RSVP, RTP, RTSP, SDP;
- Взаємодія з іншими протоколами сигналізації. Протокол SIP може бути використаний спільно з іншими протоколами IP-телефонії, протоколами ТМЗК, і для зв'язку з інтелектуальними мереж;

Клієнти SIP традиційно використовують порт 5060 TCP або UDP для з'єднання елементів SIP-мережі. В основному, SIP використовується для встановлення голосових та відео-дзвінків. При цьому він може використовуватися і в будь-яких інших додатках, де потрібна установка з'єднання, таких, як системи оповіщення, мобільні термінали і так далі. Існує велика кількість рекомендацій RFC, що відносяться до SIP і визначають поведінку таких додатків. Для передачі самих голосових і відеоданих використовують інші транспортні протоколи, найчастіше RTP.

Головним завданням розробки SIP було створення сигнального протоколу на базі IP, який міг би підтримувати розширений набір функцій обробки виклику і послуг, представлених в існуючій PSTN. Сам протокол SIP не визначає цих

функцій, а зосереджений тільки на процедурах реєстрації користувача, встановлення і завершення виклику і відповідної сигналізації. При цьому він був спроектований з підтримкою таких функціональних елементів мережі, як проксі-сервери (Proxy Servers) та User Agents. Ці елементи забезпечують базовий набір послуг: набір номера, виклик телефонного апарату, звукове інформування абонента про статус виклику.

Телефонні мережі на основі SIP можуть підтримувати і більш сучасні послуги, що зазвичай надаються ОКС-7, незважаючи на значну відмінність цих двох протоколів. ОКС-7 характеризується складною, централізованою інтелектуальною мережею і простими, неінтелектуальними, терміналами (традиційні телефонні апарати). SIP - навпаки, вимагає дуже просту (і, відповідно, добре масштабується) мережу з інтелектом, вбудованим в кінцеві елементи на периферії (термінали, побудовані як фізичні пристрої або програми).

SIP використовується разом з декількома іншими протоколами і бере участь тільки в сигнальній частині сесії зв'язку. SIP виконує роль носія для SDP, який описує параметри передачі відеоданих в рамках сесії, наприклад використовувані порти IP і кодеки. У типовому застосуванні сесії SIP - це просто потоки пакетів RTP. RTP є безпосереднім носієм голосових і відео даних.

Перша запропонована версія стандарту (SIP 2.0) була визначена в RFC 2543. Протокол був додатково уточнений у RFC 3261, хоча багато реалізацій як і раніше засновані на проміжних версіях стандарту.

Протокол SIP має клієнт-серверну архітектуру. Клієнт видає запити, із зазначенням того, що він хоче отримати від сервера. Сервер приймає і обробляє запити, видає відповіді, які містять повідомлення про успішність виконання запиту, повідомлення про помилку або інформацію, запитувану клієнтом.

Обслуговування виклику розподілено між різними елементами мережі SIP. Основним функціональним елементом, що реалізує функції управління з'єднанням, є абонентський термінал. Інші елементи мережі можуть відповідати за маршрутизацію викликів, а іноді служать для надання додаткових сервісів.

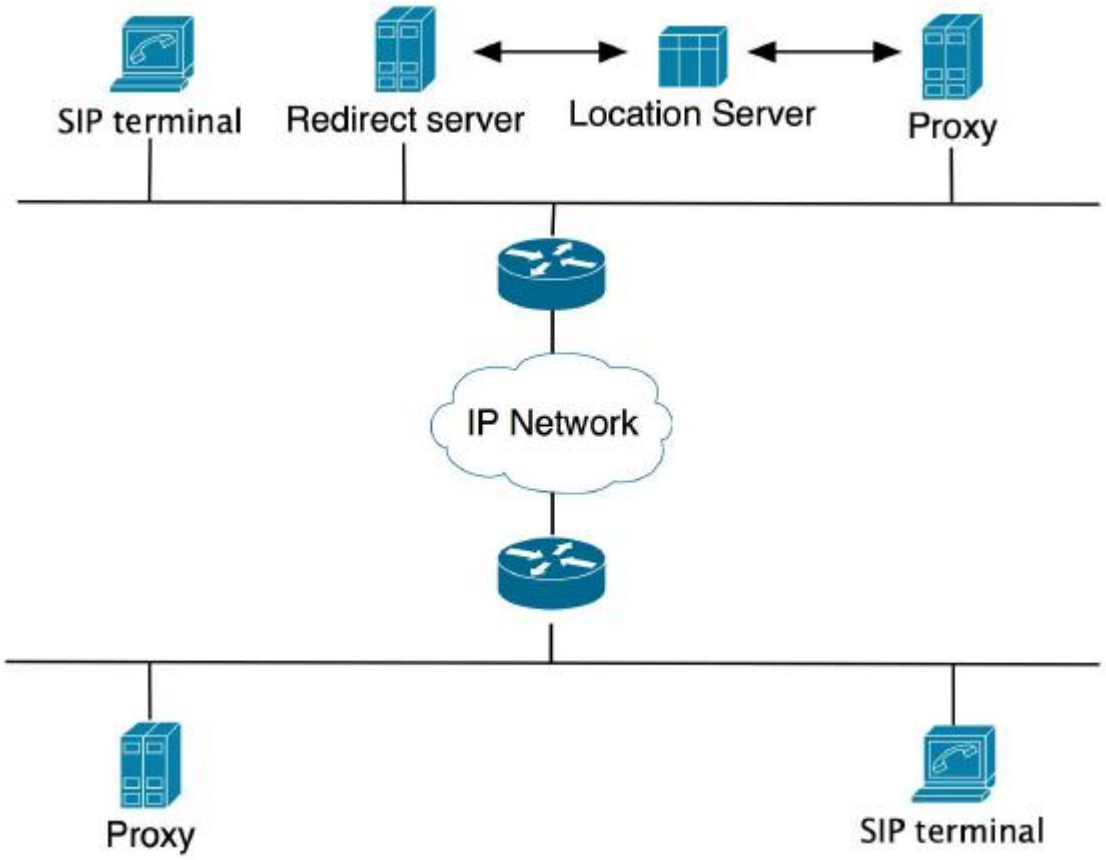


Рис. 1.8 Архітектура протоколу SIP

Термінал

Коли клієнт і сервер реалізовані в кінцевому обладнанні і взаємодіють безпосередньо з користувачем, вони називаються призначеним для користувача агентськими клієнтами - User Agent Client (UAC) - і призначеним для користувача агентськими сервером - User Agent Server (UAS). Якщо в пристрої присутні і UAC, і UAS, то воно називається призначеним для користувача агентом - User Agent (UA), а за своєю суттю є термінальне обладнання SIP.

Сервер UAS і клієнт UAC мають можливість безпосередньо взаємодіяти з користувачем. Інші клієнти і сервери SIP цього робити не можуть.

Проксі-сервер

Проксі-сервер представляє інтереси користувача в мережі. Він приймає запити, обробляє їх і виконує відповідні дії. Проксі-сервер складається з клієнтської і серверної частин, тому може приймати виклики, ініціювати запити і повертати відповіді. Проксі-сервер може не змінювати структуру і вміст

переданих повідомлень, лише додаючи свою адресну інформацію в спеціальне поле Via.

Передбачено два типи проксі-серверів:

- зі збереженням станів (stateful). Такий сервер зберігає в своїй пам'яті всі отримані запити та пов'язані з ним нові сформовані запити до закінчення транзакції.
- без збереження станів (stateless). Такий сервер просто обробляє отримані запити. Але на його базі можна реалізувати складні, інтелектуальні послуги.

Сервер B2BUA

B2BUA - варіант серверного логічного елемента в додатках, що працює з протоколом SIP. За ідеологією роботи, B2BUA схожий на проксі-сервер SIP, проте є принципові відмінності. Сервер B2BUA, працює одночасно з декількома (як правило, двома) кінцевими пристроями - терміналами, розділяючи виклик або сеанс на різні ділянки мережі. З кожною ділянкою B2BUA працює індивідуально, як UAS по відношенню до ініціатора і як UAC по відношенню до терміналу, який приймає виклик. При цьому сигнальні повідомлення передаються в рамках сеансу в обидві сторони синхронно, хоча рішення про необхідність передачі повідомлення і його форматі приймається B2BUA для кожної ділянки в індивідуальному порядку. Кожен з учасників з'єднання (сеансу зв'язку), на рівні сигналізації взаємодіє з B2BUA, як з кінцевим пристроєм, хоча в дійсності, сервер є посередником. Це відбивається в адресних полях повідомлень, що відправляються сервером B2BUA. Таким чином, ключова відмінність B2BUA - повністю незалежна сигналізація всіх ділянок виклику. Це означає, зокрема, що для взаємодії з кожним окремим користувачем в рамках сеансу зв'язку використовуються унікальні ідентифікатори, а вміст одних і тих же повідомлень для різних ділянок буде різним. Призначені для користувача агенти кінцевих терміналів можуть взаємодіяти з B2BUA і за участю проксі-серверів.

Сервер B2BUA може надавати такі функції:

- управління дзвінками (білінг, автоматичне роз'єднання і т. д.)
- сполучення різних мереж (зокрема, для адаптації різних протоколів, залежних від виробників)
- приховування структури мережі (приватні адреси, мережева топологія і т. п.)

Досить часто B2BUA є частиною медіа-шлюзу для того, щоб повністю контролювати медіа-потоки в рамках сесії. Сигнальний шлюз, який є частиною контролера з'єднань/сеансів - наочний приклад застосування B2BUA.

Сервер переадресації

Сервер переадресації використовується для перенаправлення виклику за адресою поточного місця розташування користувача. Сервер переадресації не термінує виклики і не ініціює власні запити, а тільки повідомляє адресу необхідного терміналу або проксі-сервера за допомогою відповідей класу 3XX (301 Moved Permanently або 302 Moved Temporarily). Для цих цілей сервер переадресації може взаємодіяти з SIP-реєстратором або сервером визначення місцеположення.

Однак, для здійснення з'єднання користувач може не використовувати сервер переадресації, якщо він сам знає поточну адресу необхідного користувача.

Сервер реєстрації (реєстратор)

Протокол SIP передбачає мобільність користувача, тобто користувач може переміщатися в межах мережі, отримуючи нову адресу. Тому в SIP існує механізм реєстрації - повідомлення про нову адресу з боку призначеного для користувача агента. Сервер реєстрації або реєстратор служить для фіксації та зберігання поточної адреси користувача і являє собою регулярно оновлювану базу даних адресної інформації. У загальному випадку, користувач повідомляє серверу реєстрації свою адресну інформацію, таку як IP-адреса або доменне ім'я та абонентський телефонний номер - за допомогою запиту REGISTER. Сервер може підтвердити успішну реєстрацію (200 OK) або відхилити, в разі якщо є перевірка даних і обліковий запис користувача не знайдено (404 Not found) або реєстрація

для користувача заборонена в даний момент (403 Forbidden). Реєстратор може вказати на необхідність логіна користувача для перевірки (401 Unauthorized), а також запропонувати цифрову аутентифікацію на основі зашифрованого пароля. Як джерело інформації для аутентифікації користувача, може виступати навіть пристрій або сервер, що не працює по протоколу SIP (наприклад СУБД, MS Exchange, RADIUS-сервер і т. п.). Реєстрація користувача на сервері має певний «термін життя» і повинна підтверджуватися новим запитом REGISTER з боку клієнта, в іншому випадку адресна інформація може бути видалена. Клієнт може також надіслати запит з нульовим часом життя реєстрації, що розглядається, як запит на примусове видалення адресної інформації з сервера.

У різних реалізаціях SIP-мереж може зустрічатися сервера реєстрації та інших серверів в єдиному додатку або пристрої, що працюють через один сокет на одному порті (зазвичай 5060) - точку отримання запитів. Так часто реєстратори поєднуються з сервером переадресації, B2BUA або SIP-проксі. Наприклад, такі softswitch (Asterisk, Yate, та ін.) містять функціонал SIP-реєстратора з перевіркою реєстраційних даних кожного користувача.

Інформація про можливості користувача зареєструватися та з'єднуватися, визначаються в даному випадку його єдиним обліковим записом. У свою чергу абонентське обладнання IP-телефонії - телефони, абонентські шлюзи, в більшості випадків вимагають обов'язкової попередньої реєстрації на реєстраторі для подальшої роботи в телефонній мережі.

В результаті система IP-телефонії може виглядати аналогічно системі стільникового зв'язку - все абонентське обладнання при включенні реєструється на комутаторі і після цього може здійснювати і приймати виклики за допомогою цього комутатора, який або переадресує запит іншому кінцевому користувачу, або виступає посередником.

Сервер визначення місцеположення користувачів

Користувач може переміщатися в межах різних мереж, крім того, справжня адреса користувача може бути і не відома, навіть якщо його номер відомий. Це

актуально, зокрема для послуги перенесення номера (LNP / MNP). Для вирішення таких завдань існує механізм визначення місця розташування користувача за допомогою сторонніх коштів, які не мають прямого відношення до елементів SIP-мережі.

Для цього використовується сервер позиціонування (Location Server), який зберігає поточну адресу користувача і являє собою регулярно оновлювану базу даних адресної інформації

Користувач, якому потрібна адресна інформація іншого користувача для встановлення з'єднання не зв'язується з сервером визначення місцеположення безпосередньо. Цю функцію виконують інші SIP-сервери за допомогою протоколів LDAP, RWHOIS, ENUM, RADIUS або інших протоколів.

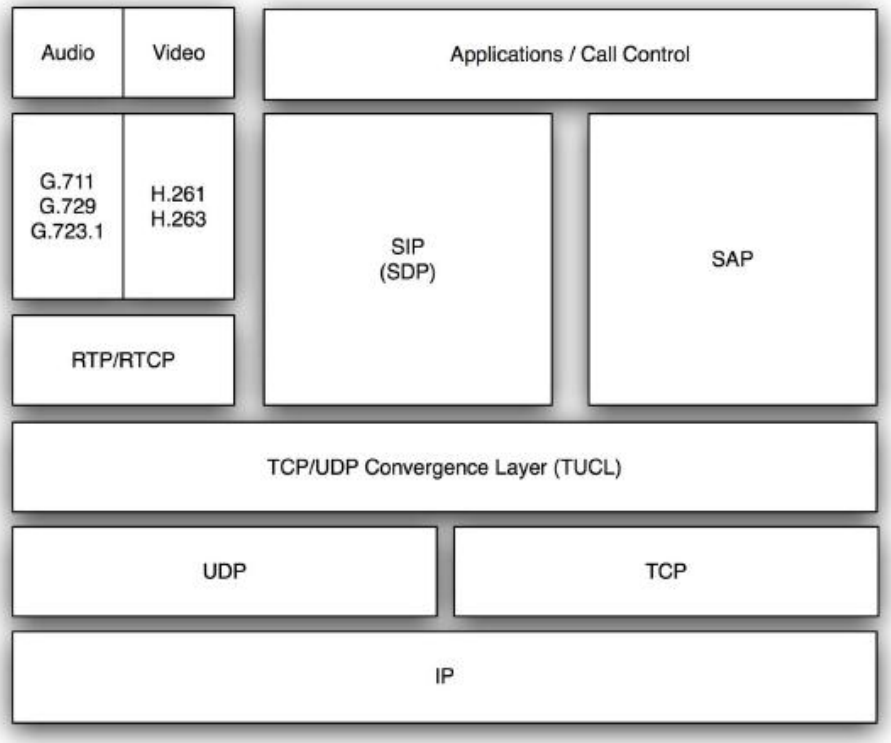


Рис. 1.9 Стек протоколу SIP

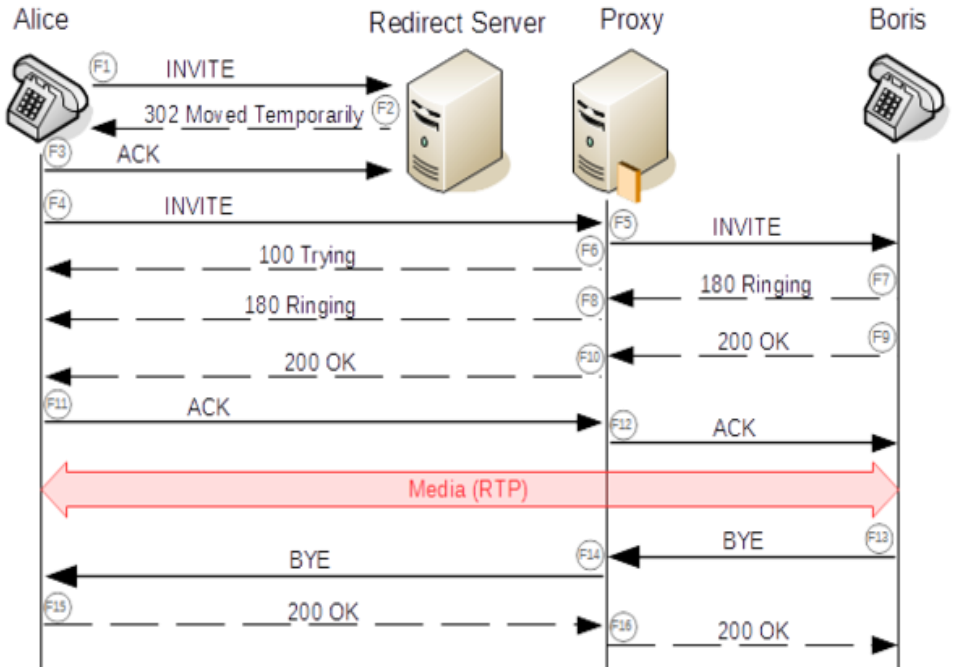


Рис. 1.10 Приклад сценарію встановлення з'єднання, за участю SIP сервера переадресації і SIP Proxy

Протокол SIP є керуючим протоколом для встановлення, модифікації і розриву з'єднання, орієнтованого на передачу потокових даних. Параметри передачі медіа-потоків описуються в протоколі SIP за допомогою SDP (протокол опису сесії). Потокові медіа-дані можуть передаватися різними засобами, серед яких найбільш популярні транспортні протоколи RTP і RTCP.

Протокол SIP визначає 3 основні сценарії встановлення з'єднання: за участю проксі-сервера, за участю сервера переадресації і безпосередньо між користувачами. Сценарії відрізняються по тому, як здійснюється пошук і запрошення викликається користувача. Основні алгоритми встановлення з'єднання описані в RFC 3665

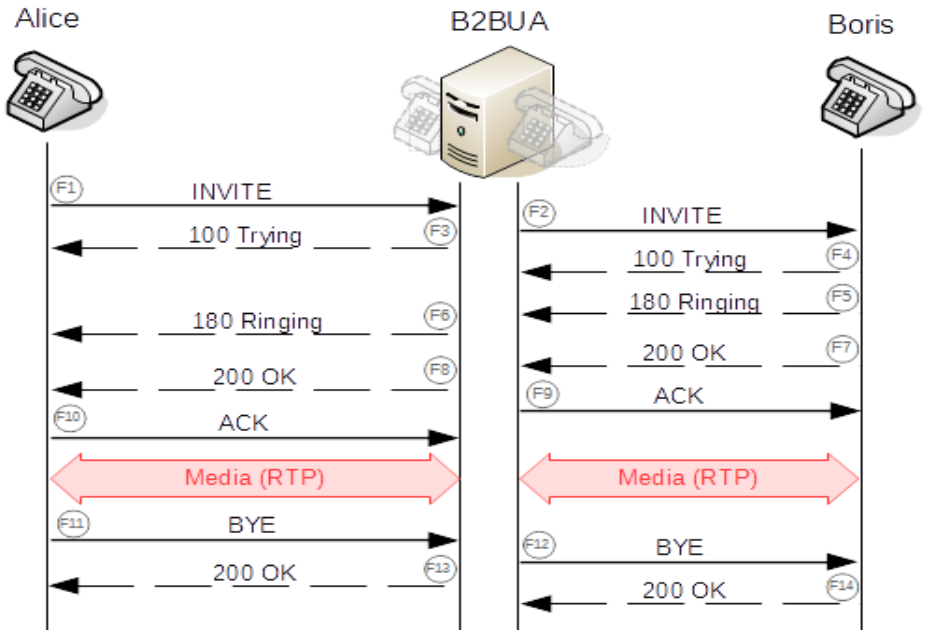


Рис. 1.11 Приклад сценарію встановлення з'єднання з участю сервера B2BUA

У прикладі нижче медіа-трафік “проксується” через сервер. Сигнальні повідомлення є повністю незалежними і виконуються в рамках різних сесій (зміняться як мінімум адреси призначення і відправлення, а також Call ID сесій). Термінал не знає реального місця розташування іншого терміналу і навпаки. Так може виглядати взаємодія через деякі softswitch або контролери сесій (SBC).

Висновки до розділу:

В даному розділі були розглянуті теоретичні питання побудови мереж IP-телефонії в розрізі історії появи. Проведено аналіз основних протоколів VoIP, архітектура, основні способи підключення та елементи мережі, а також огляд найпоширенішого протоколу SIP.

Технологія VoIP (Voice over IP) має ряд переваг, на відміну від телекомунікаційної мережі загального користування PSTN. Найбільшою перевагою є економія витрат при дзвінку. Замість того, щоб оплачувати телефонні лінії і схеми, клієнти платять тільки за з'єднання даних. Крім того, IP-пакети можуть бути перенаправлені в будь-яке місце з підключенням до Інтернету. В рамках економії коштів співробітники можуть безкоштовно телефонувати в корпоративну мережу. Іншою перевагою є використання наявної мережевої інфраструктури, тому більше не потрібно використовувати традиційні телефонні кабелі для мережевого з'єднання УАТС. Також можлива конфігурація УАТС з будь-якого місця через CLI командного рядка або веб-інтерфейс. Технологія VoIP характеризується сумісністю зі телефонною мережею загального користування PSTN. Крім того, голос в технології VoIP не вимагає високої пропускної здатності (декілька кбіт/с) і, отже, перебуває у фактичних викликах в корпоративній мережі.

В даний час IP протокол використовується не тільки в мережі Інтернет, а й в інших мережах передачі даних з пакетною комутацією (локальних, корпоративних, регіональних). І у всіх цих мережах, є можливість передавати мовленнєві повідомлення з використанням пакетів даних. такий спосіб передачі мови отримав назву IP-телефонія.

IP-телефонія дозволяє істотно економити необхідну смугу пропускання каналів, що неминуче веде до зниження тарифів, особливо на міжміські і міжнародні телефонні розмови. технологія IP - телефонії надає можливість отримання високоякісного телефонного з'єднання між двома і більше абонентами, що знаходяться в різних куточках планети. Телефонні переговори, організовані з допомогою VoIP технології є більш надійними і універсальними в

порівнянні з традиційними, так як замість проміжних комутаторів, станцій та інших пристроїв використовується програмний продукт з відкритим вихідним кодом Asterisk.

PBX Asterisk - вільно доступне програмне забезпечення на базі Linux. В доповнення, ця АТС дозволяє використовувати цифрову ISDN (Integrated Services Digital Network) і аналогові телефони, які все ще використовуються в багатьох компаніях. Asterisk також підтримує підключення до PSTN і інших мереж VoIP. В даний час програмне забезпечення PBX Asterisk стало великим конкурентом традиційної апаратної АТС. Одним з переваг Asterisk є низька вартість побудови АТС, тоді як АТС можна запускати на персональному комп'ютері або на сервері. Іншою перевагою є швидка і проста установка і управління через панель управління веб-інтерфейсом. Asterisk підтримує декілька протоколів, таких як IAX, SIP, H.323 і MGCP. Рішення Asterisk призначене, в основному, для шкіл, готелів, малих і середніх підприємств. Asterisk надає велику кількість послуг і функцій. Найбільш поширеними є конференц-дзвінки, переадресація, власний план нумерації, голосова пошта, детальна інформація про кожному дзвінку, IVR, ACD і т.д. Впровадження програмного забезпечення PBX Asterisk в корпоративних середовищах має ряд переваг. Програмна АТС, що працює на більш потужному ПК або меншому сервері, може передавати до декількох сотень викликів. Перевага полягає в тому, що співробітники в корпоративної мережі можуть дзвонити один одному безкоштовно.

2. АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ КОРПОРАТИВНОЇ VOIP-МЕРЕЖА НА ПЛАТФОРМІ ASTERISK

2.1 Аналіз вимог до системи та архітектури мережі VoIP

Перш ніж розгортати мережу IP-телефонії, слід враховувати декілька вимог, з які ви повинні враховувати. Перехід від традиційного телефонного зв'язку до VoIP потребують достатньої пропускної здатності, вимоги до маршрутизатора, а також надійного резервного рішення для захисту від аварійних ситуацій.

Основні вимоги до мережі IP-телефонії:

- пропускна здатність;
- маршрутизатор;
- якість сервісу;
- VoIP-обладнання;
- живлення.

2.1.1 Показники якості IP-телефонії

Традиційні телефонні мережі коммують електричні сигнали з гарантованою смугою пропускання, достатньою для передачі сигналів голосового спектру. При фіксованій пропускній здатності переданого сигналу ціна одиниці часу зв'язку залежить від віддаленості і розташування точок виклику і місця відповіді.

Мережі з комутацією пакетів не забезпечують гарантованої пропускної здатності, оскільки не забезпечують гарантованого шляху між точками зв'язку.

Для додатків, де не важливий порядок і інтервал приходу пакетів, наприклад, e-mail, час затримок між окремими пакетами не має вирішального значення. IP-телефонія є однією з областей передачі даних, де важлива динаміка передачі сигналу, яка забезпечується сучасними методами кодування і передачі інформації, а також збільшенням пропускної спроможності каналів, що призводить до

можливості успішної конкуренції IP-телефонії з традиційними телефонними мережами.

Основними складовими якості IP-телефонії є:

• якість мови, яке включає:

- діалог - можливість користувача зв'язуватися і розмовляти з іншим користувачем в реальному часі і повнодуплексному режимі;
- розбірливість - чистота і тональність мови; відлуння - чутність власної мови; рівень та гучність мови.

• якість сигналізації, що включає:

- встановлення виклику - швидкість успішного доступу і час встановлення з'єднання;
- завершення виклику - час відбою і швидкість роз'єднання;
- DTMF - визначення і фіксація сигналів багаточастотного набору номера.

Фактори, які впливають на якість IP-телефонії, можуть бути розділені на дві категорії:

• фактори якості IP-мережі:

- максимальна пропускна здатність - максимальна кількість корисних і надлишкових даних, що передається;
- затримка - проміжок часу, необхідний для передачі пакета через мережу; джиттер - затримка між двома послідовними пакетами; втрата пакета - пакети або дані, втрачені при передачі через мережу.

Фактори якості шлюза:

- необхідна смуга пропускання - різні кодеки вимагають різну смугу. Наприклад: G.723 вимагає смуги 16,3 кбіт/с для кожного мовного каналу;
- затримка - час, необхідний для цифрового сигнального процесора DSP або інших пристроїв обробки, кодування і декодування мовного сигналу;

- буфер джиттера - збереження пакетів даних до тих пір, поки всі пакети не будуть отримані і можна буде передати в необхідній послідовності для мінімізації джиттера;
- втрата пакетів - втрата пакетів при стисненні і/або передачі в обладнанні IP-телефонії;
- придушення відлуння - механізм для придушення відлуння, що виникає при передачі по мережі; управління рівнем - можливість регулювати гучність мови.

2.2 Адресація в мережах IP-телефонії

В даний час нумерація в мережах загального користування з комутацією каналів, які надають послуги телефонного зв'язку (телефонні мережі, мережі ISDN, інтелектуальні мережі, стільникові мережі та ін.), реалізуються відповідно до рекомендації ITU-T E.164.

Система нумерації таких мереж включає міжнародний і національні плани нумерації.

Кожна телефонна адміністрація розробляє національний план нумерації для своєї мережі. Цей план розробляється таким чином, щоб будь-який абонент національної мережі міг бути доступний по одному і тому ж номеру для різних послуг. Причому це має виконуватися для всіх вхідних міжнародних викликів. Національний план нумерації країни повинен бути такий, щоб аналіз цифри не перевищував встановлені межі, застосовні до національного (що означає) номеру N(S)N.

Поле «Код країни (CC)» використовується для визначення країни або географічної області призначення. Данний код має різну довжину, конкретні значення кодів для країн світу наведені в рекомендації ITU-T E.I64. Слід зазначити, що код країни починається з номера світової зони нумерації. В даний час територія всієї земної кулі поділена на 9 світових зон нумерації:

Зона 1 - Північна Америка; Зона 2 - Африка; Зони 3 і 4 - Європа;

Зона 5 - Центральна і Південна Америка; Зона 6 - Австралія і Океанія;

Зона 7 - Росія і Казахстан; Зона 8 - Південно-Східна Азія; Зона 9 - Азія.

Поле “Національний (що означає) номер N(S)N" використовується для визначення конкретного абонента в мережі. При виборі потрібного абонента іноді необхідно визначити ще й мережу призначення. У цьому випадку національний код включає поле національного коду призначення (NDC). Національний код призначення може мати різну довжину в залежності від вимог національних адміністрацій.

Поле «Номер абонента SN» також має довільну довжину в кожній національній мережі згідно рекомендації ITU-T E.I60.

Слід зазначити, що загальна довжина міжнародного номера в даний час не повинна перевищувати 15 цифр. При цьому в дану довжину номера не входять префікси, символи, адресні обмежувачі (наприклад, закінчення ім-пульсних сигналів), так як вони не є частиною міжнародного номера мережі загального користування.

У системах IP-телефонії, так само як і в мережах з комутацією каналів, номери відповідно до рекомендації E.164 використовуються кінцевими користувачами, щоб ідентифікувати виклик. У IP-системах, коли кінцевий користувач ідентифікується терміналом, номер E.164 цього кінцевого користувача тимчасово пов'язаний з IP-адресою (транспортна адреса) цього терміналу (кінцевої точки). Проблема нумерації в мережі IP-телефонії пов'язана з визначенням точки призначення виклику при внутрішньодоменному та міждоменному зв'язку в IP-мережі. В якості такої кінцевої точки може виступати або IP-термінал з відповідним додатком користувача або шлюз для доступу в мережу з комутацією каналів.

Від вирішення завдань адресації в IP-телефонії в чому залежать зручність користування послугою, робота алгоритмів маршрутизації, забезпечення мобільності номерів і т.д. Головна проблема організації взаємодії мереж з комутацією каналів і IP-мереж полягає в тому, що єдиний метод адресації звичайного терміналу абонента телефонної мережі - це використання номера

цього терміналу (в мережах загального користування номера E.164). Питання перетворення номера мережі з комутацією каналів в IP-адресу представляється поки ще досить складним і розробляється не тільки робочою групою 4 в рамках проекту TIPHON, а й іншими організаціями, наприклад IETF. У той же час ITU-T тільки підходить до вирішення питань взаємодії послуг IP-телефонії і PSTN, обмежуючись поки розглядом функцій міжмережевої взаємодії на рівні, транспортних технологій. Така позиція пояснюється, зокрема, відсутністю загальних для всіх національних адміністрацій зв'язку підходів до визначення статусу послуги IP-телефонії.

Оператору IP-телефонії, що пропонує свої послуги абонентам мереж з комутацією каналів, необхідно, природно, використовувати вже наявні схеми нумерації. Згідно з рекомендаціями TIPHON, для організації викликів від абонентів мереж з комутацією каналів користувачам IP-мережі бажано, щоб останні мали номер E.164. У проекті TIPHON також досліджується можливість використання в Інтернет коду країни та коду послуги, які будуть задіяні в Інтернет-телефонії.

У мережах IP-телефонії, побудованих на базі стандарту H.323, перетворення телефонних номерів E.164 в IP- адреси і назад входить у функції gatekeeper. У системах, що використовують протокол SIP, ці функції виконуються в спеціальному сервері.

Таблиця 1 показує відносини між іменами та адресами для телефонних мереж і додатків Інтернет. Вона також включає відмінності в адресації між концепцією TIPHON і рішеннями Інтернет-телефонії, заснованої на протоколі SIP.

Мета перетворення номера - заміна цифр, набраних користувачем, в імена E.164 і перетворення цих імен в адреси, імена або ідентифікатори, які необхідно використовувати для маршрутизації IP-повідомлень управління телефонними викликами. При цьому телефонні з'єднання встановлюються всередині домену або між доменами і/або далі маршрутизуються в мережу з комутацією каналів. Для

виконання функцій маршрутизації при обслуговуванні викликів необхідно мати базу даних про користувачів і шлюзах, про перетворення номерів, імен і адрес.

	Телефонні або інші мережі з комутацією каналів	**E-mail**	**Концепція TIPHON**	**Рішення на базі протоколу SIP**
Ім'я	Номер E.164	user@host, де host - ім'я домену	Номер E.164	user@host, можливо з підстановлювальним номером E.164 для вхідних викликів з мережею комутації каналів
Адреса	Маршрутизація за номером E.164 (або префікс маршру-тизації + номер E.164)	IP-адреса	IP-адреса	IP-адреса

Табл.1 Відносини між іменами і адресами для телефонних мереж та додатками

Мережі IP-телефонії повинні підтримати перетворення номерів в двох випадках:

1. Маршрутизовані виклики направляються в мережу з комутацією каналів. У цьому випадку необхідний один маршрут до домену, в якому розташований шлюз до мережі з комутацією каналів, що забезпечує доступ до адресата. Хоча можуть бути доступні більш ніж один маршрут, так як кілька доменів і кілька шлюзів дозволяють обслуговувати цей виклик.

2. Маршрутизовані виклики направляються в мережу з комутацією пакетів (IP-мережу). В цьому випадку користувач використовує номер ЕЛ 64 як назва, яка ідентифікує адресата IP-мережі. При цьому можливий тільки один маршрут через відповідний шлюз.

Відповідно до концепції TIPHON мережі IP-телефонії повинні підтримувати, принаймні, одну з наданих схем нумерації:

1. Домени мережі IP-телефонії повинні підтримати всі схеми нумерації на мережах зв'язку з комутацією каналів і забезпечувати належну міжмережеву взаємодія з ними.

2. План нумерації для користувачів мереж IP-телефонії може бути таким же, як і для користувачів мереж з комутацією каналів, причому з урахуванням національних особливостей.

3. Нумерація для надання послуг користувачам IP-телефонії повинна бути аналогічною нумерації, яка використовується в мережах з комутацією каналів.

Система нумерації IP-телефонії повинна забезпечувати можливість заміни одного номера E.164 на інший. Це необхідно для забезпечення підтримки таких послуг:

• мобільність номера;

• персональна нумерація;

• послуги типу freephone.

При таких послугах номер надсилається у вигляді запиту на шлюз IP-телефонії та ідентифікується як номер маршрутизації E.164. Відповідь на запит буде завжди у вигляді номера E.164.

В системі IP-телефонії може існувати два види планів нумерації: відкритий (внутрішній і міжнародний) і приватний. При цьому можливі три типи формату номерів:

1. Фіксований - фіксований номер;

2. Змінний - номер може змінюватися;

3. Корпоративний - номер визначається даними конфігурації корпоративного плану набору (Custom Bailing Plan).

Формат номера внутрішнього плану має такий вигляд:

- фіксований: внутрішній національний код (якщо є) + код міста + номер абонента;
- змінний: локальний виклик (код міста відповідає коду, визначеного для шлюзу Інтернет-телефонії) - набирається тільки номер абонента;
- міжміський дзвінок (код міста відрізняється від коду, визначеного для шлюзу) - набирається внутрішній національний код (якщо є) + код міста + номер абонента;
- корпоративний: номер конфігурується адміністратором і залежить від певних їм кодів.

Формат номера міжнародного плану має такий вигляд:

- фіксований: код виходу на міжнародну мережу + код країни + код міста + номер абонента;
- корпоративний: номер конфігурується адміністратором і залежить від певних префіксів.

Формат номера приватного плану має такий вигляд:

- фіксований: номер абонента;
- змінний: номер залежить від наступних факторів:

- локальний виклик (код приватної зони відповідає коду, визначеного для шлюзу);
- міжміський дзвінок (код приватної зони відрізняється від коду, визначеного для шлюзу) - внутрішній національний код (якщо є) + код міста + номер абонента.

- корпоративний: номер конфігурується адміністратором і залежить від певних кодів.

2.3 Аналіз можливості використанням технології віртуалізації Vagrant для реалізації платформи Asterisk

Vagrant - це інструмент з відкритим програмним кодом, для створення повноцінних середовищ розробки, ізольованих на віртуальних машинах. Vagrant зменшує час розгортання середовища розробки, збільшує швидкість розробки/виробничого паритету та є носієм ідеї одноразових обчислювальних ресурсів.

Технологія Vagrant дозволяє:

• створювати віртуальну машину на базі обраної операційної системи;

• виділяти ресурси віртуальної машини (наприклад, ОЗУ, кількість ЦПУ т. д.).

• дозволяє конфігурувати мережеві інтерфейси для отримання доступу до віртуальної машини

• створювати директорії, які “розшарюються” між хостовою та віртуальною машинами;

• встановлювати ім'я хоста машини;

• виконувати спеціальну настройку хоста і гостьової системи.

Щоб зрозуміти потребу в такому інструменті, як Vagrant, важливо зрозуміти старі способи розробки середовища програмного забезпечення. До Vagrant кращим способом роботи з WEB-додатками було встановлення та налаштування необхідного програмного забезпечення (наприклад, Apache, MySQL, RabbitMQ і т. д.) локально на вашій машині. Сучасні з WEB -додатки мають набагато більше рухомих частин, набагато більше можливостей для базових технологій і набагато більш загальну складність. Хоча PHP і MySQL і раніше користуються великою популярністю, вони вже явно не домінують на ринку, мови програмування, такі як JavaScript, Python і Ruby, швидко стають динамічними мовами вибору.

Щоб отримати огляд всіх зовнішніх об'єктів, що взаємодіють з Vagrant, представлений наступний рисунок.

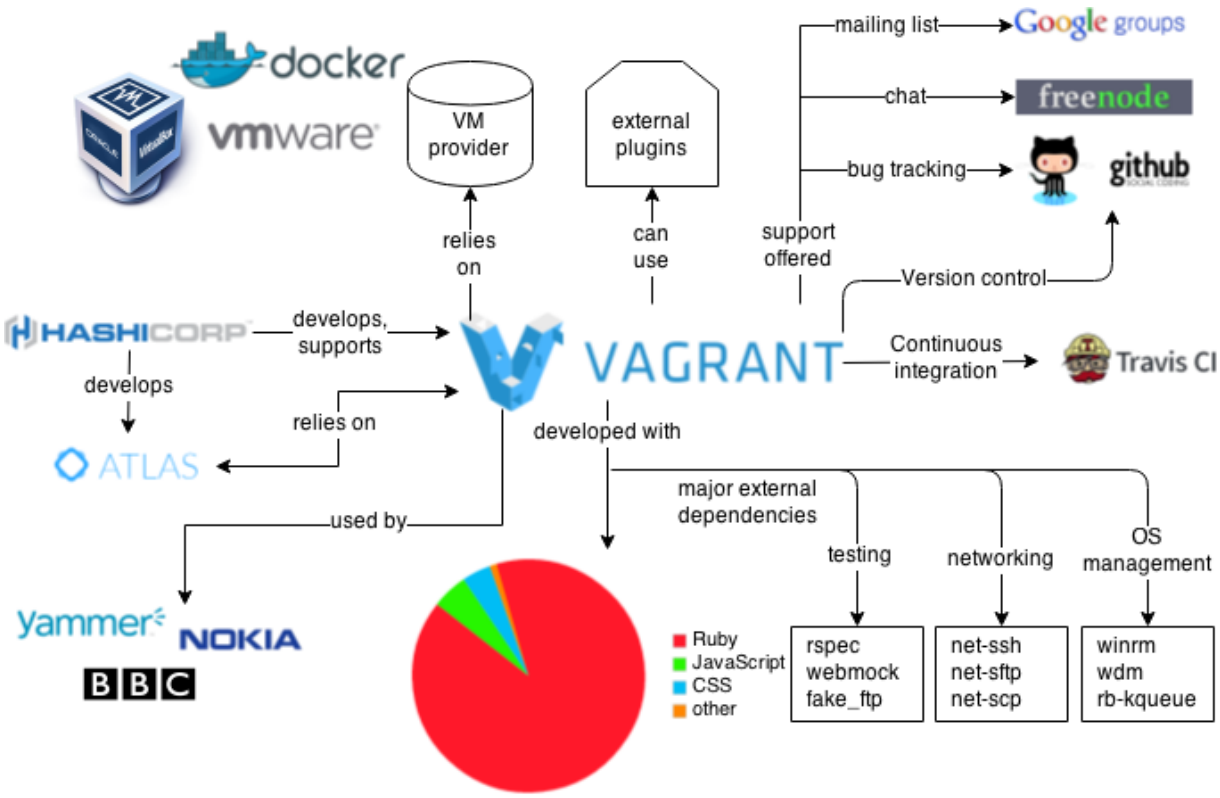


Рис. 2.1 Взаємодія Vagrant з іншими технологіями

Оскільки Vagrant використовує зовнішні віртуальні машини, він сильно залежить від постачальників VM, таких як VMware, Docker або Oracle VirtualBox.

HashiCorp також розробляє запатентований Atlas, який володіє хмарними можливостями спільного використання і зберігання призначених для користувачів. Це дозволяє користувачам Vagrant ділитися і завантажувати “box”(віртуальні машини) з іншими користувачами за допомогою командного рядка. Ця функціональність для спілкування з Atlas вбудована в Vagrant. Оскільки Atlas побудований тільки для Vagrant, обидві системи залежать одна від одної.

Щоб зрозуміти, як працює цей інструмент, необхідно зануритися в базову архітектуру Vagrant. Організація модулів і їх зв'язок показані на рисунку нижче.

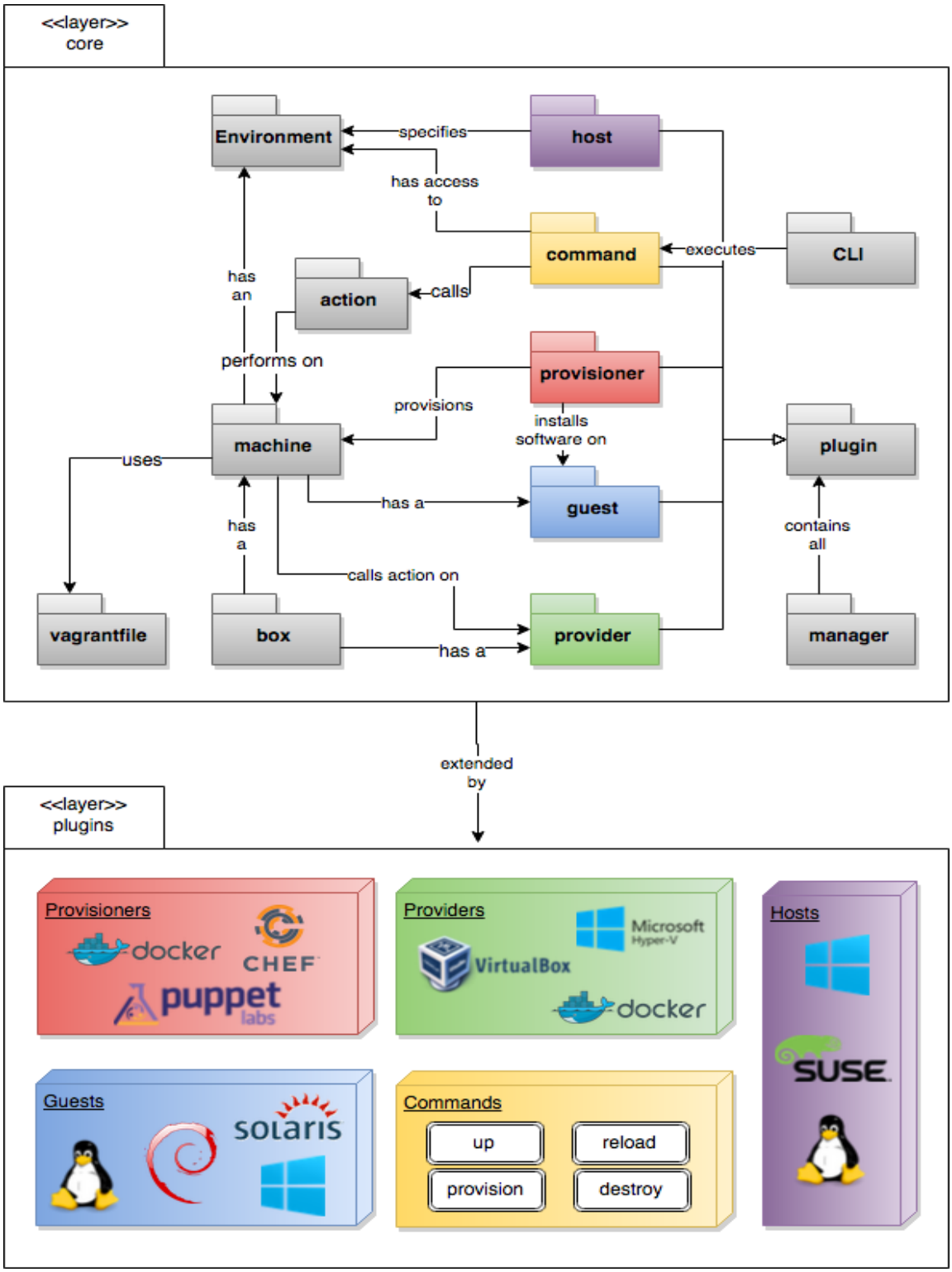


Рис. 2.2 Архітектура Vagrant

Як видно, основне ядро складається з самого внутрішнього ядра Vagrant. Він визначає основні компоненти системи, такі як машина, яка представляє собою інтерфейс до віртуальної машини. Крім того, він вказує інтерфейси, які можуть

розширюватися. Наприклад: плагін постачальника VirtualBox розширює інтерфейс провайдера в базовому шарі.

“Box”(віртуальна машина) - це фактично середовище віртуальної машини, упаковане в файл, який сумісний з Vagrant. Зрештою, це робить Vagrant цінним інструментом, так як “box” можуть бути легко розділені між розробниками для отримання однакового робочого середовища. Машина може виконувати конфігурацію, використовуючи налаштування, описані в Vagrantfile.

Vagrantfile є центральною конфігурацією віртуальних машин для кінцевого користувача. Тут можуть бути вказані різні настройки, такі як постачальник і надання ресурсів. Кожен користувач, який має той же Vagrantfile, також зможе налаштувати теж саме середовище, використовуючи цей файл.

Операції можуть виконуватися з використанням проміжного програмного забезпечення. Це проміжне програмне забезпечення в основному визначає певні функції, які повинні виконувати всі машини, які не пов'язані з типом машини.
Машина також має посилання на навколишнє середовище. Оточення Vagrant в основному представляє собою конфігурацію того, як Vagrant повинен вести себе з точки зору робочого каталогу, налаштувань призначеного для користувача інтерфейсу і т. д.

Оточення змінено за допомогою специфічних для ОС функціональних можливостей за допомогою інтерфейсу хоста. Цей інтерфейс дозволяє розробникам писати плагіни для включення функціональних можливостей ОС в Vagrant. Командні дії, згадані раніше, використовуються командами.

Командний інтерфейс дозволяє розробникам писати плагіни, щоб дозволити користувачам виконувати певні команди в інтерфейсі командного рядка (CLI). Основні команди, надані Vagrant, також записуються з використанням цього інтерфейсу.

Інтерфейс провайдера дозволяє реалізувати програми для взаємодії з продуктами віртуалізації. Кілька прикладів реалізації інтерфейсу постачальника - це плагіни постачальників для VirtualBox, Docker і Hyper-V.

Машина, природно, реалізує певну ОС, це називається гостьовою ОС. Користувач відповідає за виявлення гостьової операційної системи, запущеної всередині машини. Потім користувач може бути розширений за допомогою різних функціональних можливостей ОС за допомогою цього типу плагінів.

Провайдер відповідає за цю задачу по встановленню програмного забезпечення та налаштуванню машини відповідно до потреб, описаними користувачем.

Щоб забезпечити узгодженість між плагінами, всі вони розширені з абстрактного класу Plugin. Модулі відслідковуються і контролюються центральним менеджером, який гарантує, що всі плагіни перераховані і зареєстровані.

Як зазначено в документації, велика частина ядра Vagrant реалізована з використанням плагінів. Оскільки Vagrant володіє власним API-інтерфейсом плагінів, можна бути впевненим, що інтерфейс стабільний і добре підтримується. Це додатково означає, що саме ядро відносно невеликого розміру і йому не вистачає найнеобхіднішої функціональності Vagrant. Фактична функціональність реалізована в системі плагінів. На діаграмі інтерфейси, такі як command, provisioner та guest, які можуть бути розширені. Наприклад: для інтерфейсу Provider видно реалізацію Docker, Puppet і Chef.

Багато з модулів використовують кілька компонентів. Ідентифікація цих компонентів необхідна, щоб показати потенційним розробникам, яка функціональність доступна і часто актуальна для існуючих і нових модулів.

Усередині основного ядра різні модулі використовують загальні налаштування конфігурації. Ця конфігурація доступна для кожного типу компонентів і може бути відредагована розробниками плагінів або кінцевими користувачами.

Vagrant також використовує звичайні сторонні бібліотеки, які забезпечують функціональність для ведення журналів, тестування, створення мереж і управління ОС.

Vagrant надає користувачеві можливість упакувати поточне середовище, виконавши vagrant package, який упаковує середовище в окрему віртуальну машину. Плагіни підтримують тільки упаковку віртуальних середовищ VirtualBox, як зазначено в документації Vagrant. Це можна розглядати як змінну функцію, так як користувач повинен мати можливість встановлювати середовища для інших.

Vagrant також надає можливість синхронізації гостьового каталогу з каталогом на хост-комп'ютері. Vagrant може працювати на різних платформах і синхронізуватися з різними типами каталогів. Щоб вирішити проблеми синхронізації каталогів між різними хостами, є кілька плагінів synced_folders. В основному це стосується користувачів, оскільки він дає їм можливість синхронізуватися між різними типами каталогів. Наприклад: в деяких випадках реалізація загального каталогу за замовчуванням може більшу продуктивність.

Vagrant середовища можуть бути упаковані у віртуальні машини, які можна використовувати для налаштування віртуального середовища. Користувачі також можуть створювати або налаштовувати свою власну віртуальну машину. Vagrant надає ряд віртуальних машин на сторінці Vagrant Boxes. За допомогою команди vagrant box користувач може управляти віртуальними машинами. Команда box також має підкоманди, такі як vagrant box add ADDRESS: може використовуватися для додавання поля з даною IP-адресою в Vagrant.

Provisioners надають користувачам можливість автоматично встановлювати програмне забезпечення і змінювати конфігурації. Користувачі можуть використовувати ряд вбудованих засобів. Кожен з цих механізмів надає користувачеві певний набір параметрів. Наприклад, Shell дозволяє користувачеві завантажувати і виконувати сценарії в гостьовій машині відповідно до документації. Provisioners налаштовані в Vagrantfile, і вони запускаються під час першого виконання Vagrant.

Мережа дозволяє здійснювати зв'язок між хостом і середовищем віртуальних машин. Існує кілька мережевих опцій, таких як переадресовані порти, приватні

мережі і загальнодоступні мережі. Конфігурація мережі виконується в Vagrantfile. Наприклад, переадресовані порти використовують TCP за замовчуванням, однак користувач може вказати додатковий протокол для використання в Vagrantfile.

Функція Multi-Machine надає користувачеві можливість налаштовувати середовище Vagrant з декількома машинами. Конфігурація для цього виконується в Vagrantfile, де користувач може вказати машини і призначити їм імена. Використання декількох машин може змінити поведінку деяких команд. Наприклад, Vagrant ssh вимагає, щоб ім'я машини було задане явно.

Vagrant використовує різних провайдерів для віртуальних середовищ. За замовчуванням Vagrant використовує VirtualBox і має незалежних провайдерів підтримки, такі як VMware, Docker і Hyper-V. Користувачі можуть працювати з іншими провайдерами через систему плагінів. Користувачі можуть вибрати конкретного провайдера, який доступний на своєму комп'ютері для запуску, шляхом надання імені при виконанні Vagrant up. Провайдери також налаштовані в Vagrantfile, і Vagrant також має можливість вибрати потрібного провайдера для користувача під час виконання.

Спільне використання брандмауерів дозволяє користувачам розділити середовища Vagrant з іншими користувачами через HTTP, SSH або загальний доступ.

Ці функції також можуть використовуватися в поєднанні один з одним. Користувач може вказати використання протоколів спільного доступу під час виконання через CLI. HTTP-обмін, наприклад, дозволяє користувачеві створювати кінцеву точку URL для доступу до HTTP-сервера в середовищі Vagrant.

Оскільки Vagrant запускає віртуальне середовище без інтерфейсу, користувач може використовувати CLI Vagrant, щоб легко отримати доступ до машини. Користувачі можуть вибирати з протоколів зв'язку ssh і winrm, які визначаються під час виконання.

В перспективі розвитку основна увага приділяється легкості з системою. Одним з основних аспектів є зручність використання призначеного для користувача інтерфейсу. Однак є ще кілька аспектів, які необхідно враховувати. Один з цих аспектів

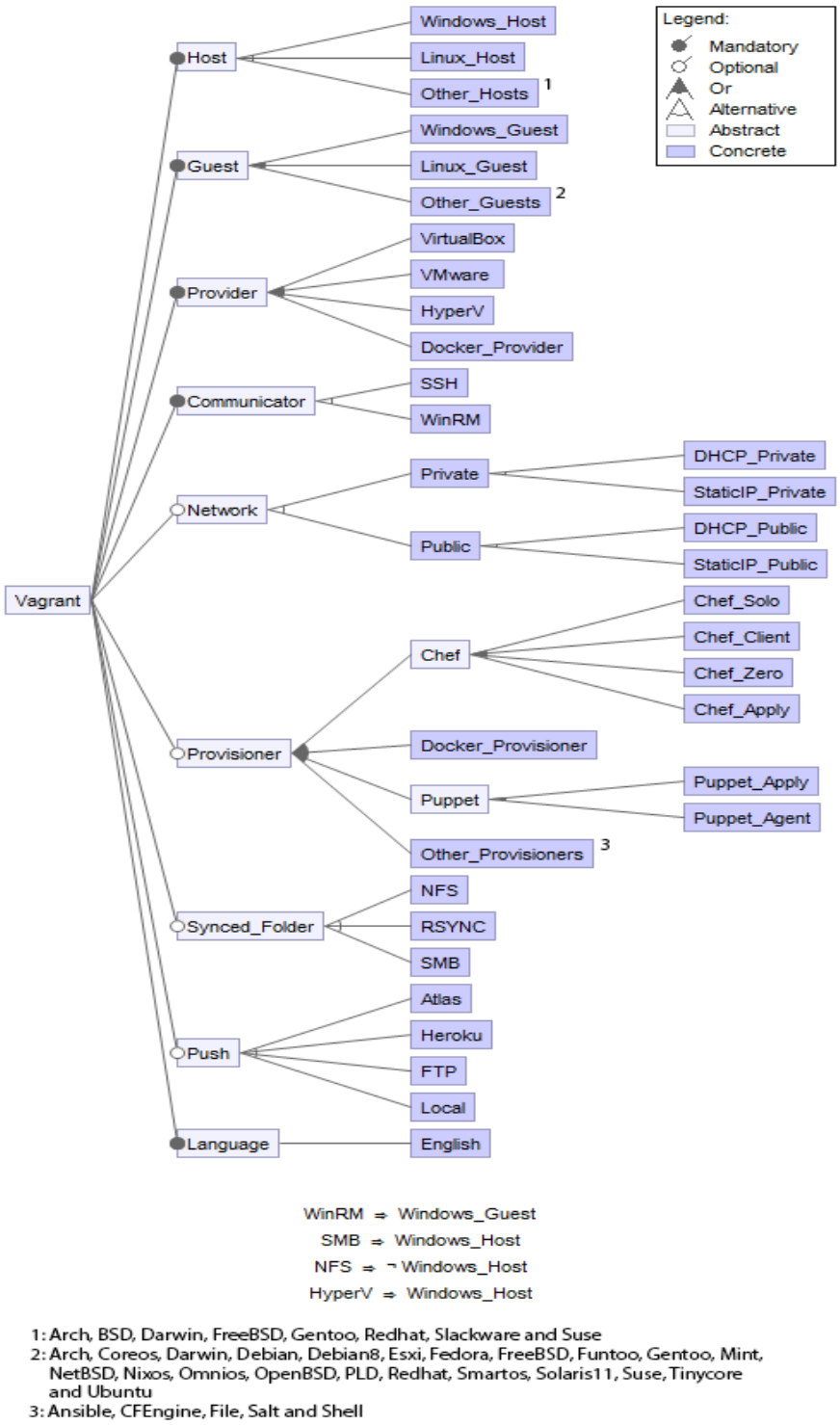


Рис. 2.3 Функціональна модель Vagrant

полягає в тому, що ця перспектива повинна бути зосереджена на узгодженні архітектури з можливостями користувачів, з досвідом та без. Місця, де люди можуть взаємодіяти з системою, повинні бути ідентифіковані, так звані «точки дотику». Крім того, призначений для користувача інтерфейс повинен бути прекрасно відділений від функціональної обробки. Таким чином, призначений для користувача інтерфейс може бути легко змінений або замінений без необхідності змінювати реалізацію повністю.

Vagrant пропонує один призначений для користувача інтерфейс: саме інтерфейс командного рядка (CLI). Цей інтерфейс є найбільш важливим «сенсорним пунктом» в даний час. CLI пропонує велику варіативність команд.

Всі команди є плагінами, які описані в “The Underlying Architecture”, тому додаткові команди можуть бути легко додані в Vagrant. CLI не містить жорстко заданих команд: всі підтримувані команди визначаються шляхом аналізу структури і вмісту /plugins/commands /.

CLI Vagrant здатний автоматично заповнювати більшість команд і підкоманди , просто натиснувши Tab, що покращує простоту використання. Однак одним з достоїнств CLI є той факт, що автозаповнення не пропонується послідовно.

Vagrant може працювати без будь-якого призначеного для користувача інтерфейсу: кожна можлива команда може бути виконана простим створенням відповідного об'єкта Command. Теоретично ви можете помістити такий об'єкт в певний основний метод і запустити цей метод за допомогою виконуваного файлу. У цьому випадку використовується тільки функціональність без участі призначеного для користувача інтерфейсу.

CLI аналізує тільки команду, надану користувачем, визначає правильний клас Command (частина функціональної обробки) і виконує його. Це вказує на те, що між призначеним для користувача інтерфейсом і функціональної обробкою відбувається розділення.

Однак іменування певних функціональних модулів вже передбачає, що вони тісно пов'язані: наприклад, Command.

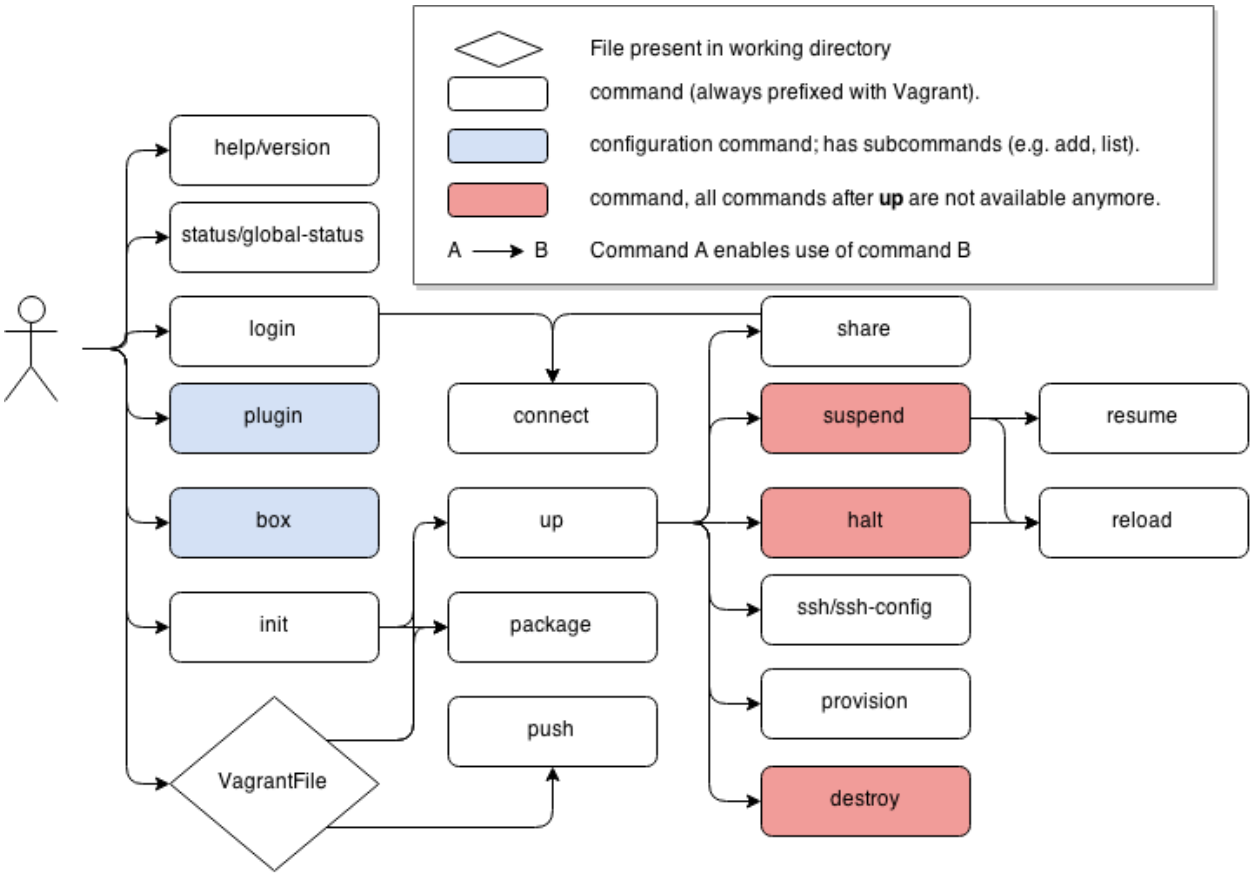


Рис. 2.4 Функціональний вид інтерфейсу командного рядка

Вище на рисунку видно, як пов'язані різні команди системи. Кожний блок являє собою команду, доступну в Vagrant. Стрілки між блоками є посиланнями залежностей. Червоні блоки указують, що після виклику команд, доступних користувачеві після виклику, вони більше не доступні. Користувачеві потрібно буде знову викликати vagrant або в деяких випадках перезавантажити або призупинити, щоб знову включити інші команди. Сині блоки вказують, що відповідна команда має кілька підкоманд. У Vagrant ці вкладені команди зустрічаються тільки для вікна команд конфігурації і плагіна. Ці команди дуже тривіальні за своїм значенням і використанням. З цієї причини вони не включені в діаграму індивідуально. Нарешті, файл в каталозі, в якому знаходиться термінал. Цей Vagrantfile можна розглядати як окремий зовнішній інтерфейс, оскільки він дозволяє користувачам настроювати всі віртуальні машини без виклику всіх команд командного рядка. Однак, щоб мати можливість використовувати Vagrantfile, користувачеві завжди потрібно викликати принаймні одну команду в

терміналі. З цієї причини Vagrantfile пов'язаний з призначеним для користувача інтерфейсом командного рядка.

2.4 Аналіз можливості використанням технології контейнеризації Docker для реалізації платформи Asterisk

Docker – це програмне забезпечення для автоматизації розгортання і управління додатками в середовищі віртуалізації на рівні операційної системи. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами. Спочатку використовував можливості LXC, з 2015 року застосовував власну бібліотеку, що абстрагує можливості ядра Linux - libcontainer. З появою Open Container Initiative почався перехід від монолітної до модульної архітектури. Контейнери продовжують працювати ізольованим чином поверх ядра операційної системи. Додатковий шар абстракції може впливати на продуктивність. Інструмент дає можливість створювати і контролювати контейнери. Ці віртуалізовані додатки можуть легко працювати без будь-яких змін. Також Docker може легко інтегрувати свої можливості з сторонніми інструментами, які допомагають легко розгортати і керувати контейнерами. Контейнери можна легко розгортати в хмарному середовищі (AWS. Google Cloud Provider, Microsoft Azure).

Основні переваги Docker:

- Швидкість.

Швидкість є однією з найбільших переваг контейнерів. Розробка, тестування та розгортання можуть виконуватися швидше, оскільки контейнери невеликого розміру. Контейнери можуть розгортатися в тестовому середовищі і лише потім потрапляти до production-середовища.

- Переносимість.

Всі залежності контейнеризованого додатку знаходяться всередині контейнера, що дозволяє запускати його на будь-якому Docker-хості.

- Передбачуваність.

Хосту все одно, що запущено всередині контейнера, а контейнеру все одно, на якому хості він працює. Інтерфейси стандартизовані і взаємодія передбачувана.

- Швидка доставка.

Формат Docker контейнерів стандартизований. Відповідальність адміністратора полягає в розгортанні і обслуговуванні сервера з контейнерами, в той час як відповідальність програміста полягає в тому, щоб стежити за додатками всередині контейнера. Контейнери можуть працювати в кожному середовищі, оскільки вони мають всі необхідні залежності, вбудовані в додатки, і всі вони протестовані. Docker забезпечує надійне, послідовне та поліпшене середовище, тому передбачувані результати можуть бути досягнуті при переміщенні кодів між системами розробки, тестування та виробництва.

- Компактність

Docker використовує ресурси, які доступні напряму з хостової машини, без взаємодії з гіпервізором (як у випадку з віртуальними машинами). Саме з цієї причини більше контейнерів можна запускати на одному хості в порівнянні з віртуальними машинами. Продуктивність Docker Containers вища через відсутність накладних витрат ресурсів на гіпервізор.

Недоліки контейнерів:

- Вони можуть вимагати складної мережевої взаємодії: оскільки функції (в ідеалі) розбиті на декілька контейнерів, ці контейнери повинні взаємодіяти один з одним. Але оскільки контейнери не є єдиним блоком, вони повинні спілкуватися через власну мережу. Деякі системи оркестрації контейнерів, такі як Kubernetes, мають абстракції більш високого рівня, так звані “pods” з декількома контейнерами, які роблять взаємодію простішою. Як додає Адам Хекско: «Насправді, мережева модель L3 в Kubernetes набагато простіше, ніж модель L2 в OpenStack». Таким чином, обсяг роботи, яку вам потрібно

буде робити в мережі, залежить від того, чи дивитеся ви на зв'язок між функціями або між віртуальними машинами.

- Вони можуть бути менш безпечними: оскільки немає повної операційної системи, люди схильні ігнорувати аспекти безпеки контейнерів, тобто хакери націлені на системи, розміщені в контейнерах і не мають конфігурованих належним чином систем безпеки.
- Вони можуть бути ненадійними: контейнери зазвичай призначені для хмарних обчислень, які припускають, що будь-який компонент може “померти” в будь-який момент; вам необхідно переконатися, що ваш додаток правильно сконструйований в разі такого випадку.

2.5 Різниця між віртуальними машинами та контейнерами

Віртуальна машина (VM) є емуляцією комп'ютерної системи. Простіше кажучи, вона дозволяє запускати те, що здається великою кількістю окремих комп'ютерів на обладнанні, яке насправді є одним комп'ютером.

Операційні системи (ОС) та її додатки спільно використовують апаратні ресурси з одного хост-сервера або з пулу хост-серверів. Кожна віртуальна машина вимагає своєї власної ОС, апаратне забезпечення відповідно віртуалізується. Гипервізор або монітор віртуальної машини - це програмне забезпечення, вбудоване програмне або апаратне забезпечення, яке створює і запускає віртуальні машини. Він знаходиться між хост-сервером та віртуальними машинами і необхідний для віртуалізації сервера.

Однак віртуальні машини можуть займати багато системних ресурсів. Кожна віртуальна машина запускає не тільки повну копію операційної системи, але і віртуальну копію всього обладнання, яке має виконувати операційна система. Це швидко додає багато циклів RAM та CPU. Це як і раніше економічно в порівнянні з запуском окремих реальних комп'ютерів, але для деяких програм цей процес може бути зайвим.

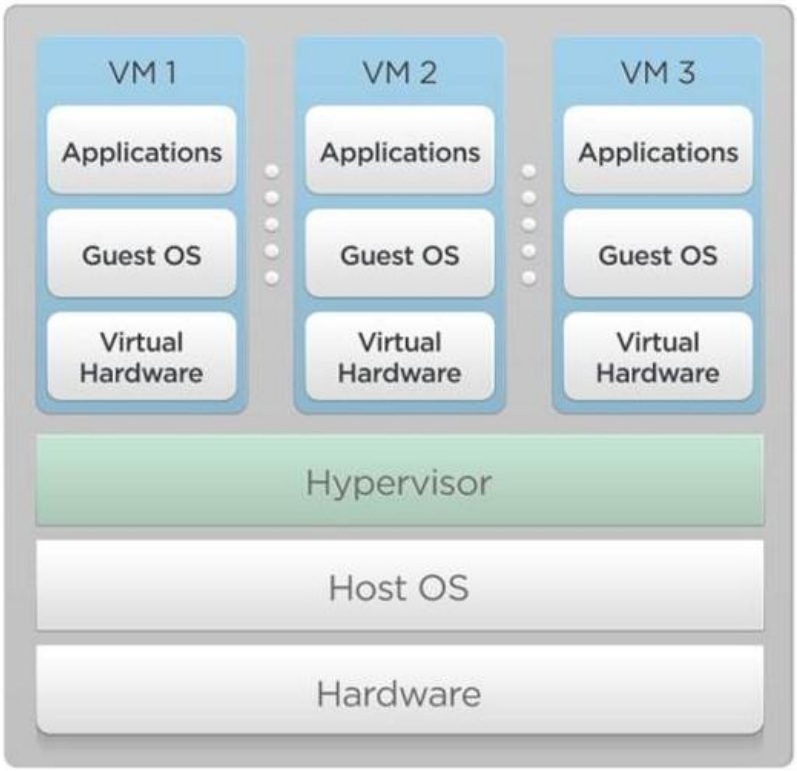


Рис. 2.5 Реалізація технології віртуалізації

Контейнери розташовуються поверх фізичного сервера і його операційної системи - зазвичай Linux або Windows. Кожен контейнер спільно використовує ядро хост-системи і, як правило, виконавчі файли і бібліотеки. Загальні компоненти доступні тільки для читання. Спільне використання ресурсів ОС, таких як бібліотеки, значно зменшує необхідність відтворення коду операційної системи і означає, що сервер може запускати кілька робочих навантажень при встановленні однієї операційної системи. Контейнери, таким чином, виключно «легкі» - вони можуть займати мегабайт в розмірі і розгортаються всього декілька секунд. Навпаки, запуск віртуальних машин може займати хвилини на порядок більше часу еквівалентного контейнера.

На відміну від віртуальних машин, все, що потрібно контейнеру, досить для операційної системи, підтримки програм і бібліотек і системних ресурсів для запуску конкретної програми. На практиці це означає, що ви можете розмістити в два-три рази більше додатків на одному сервері з контейнерами, ніж ви можете за допомогою віртуальної машини. Крім того, з контейнерами ви можете створити

портативне, узгоджене операційне середовище для розробки, тестування та розгортання.

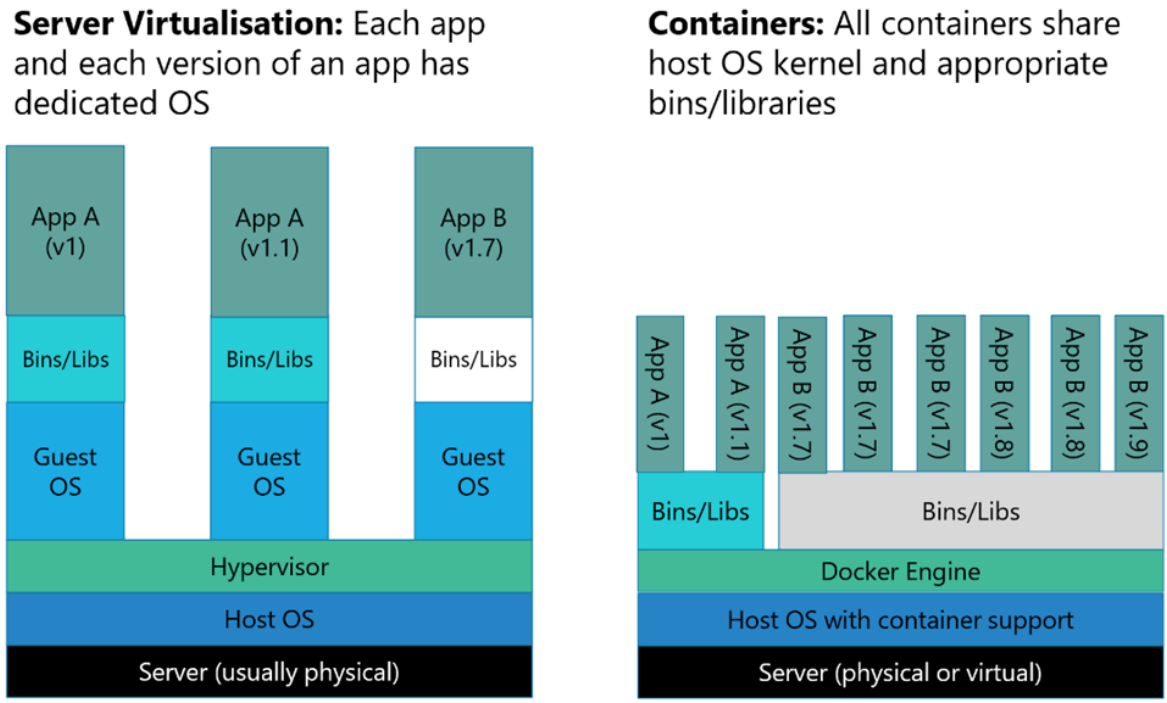


Рис. 2.6 Порівняння віртуальних машин та контейнерів

2.6 Архітектура Docker

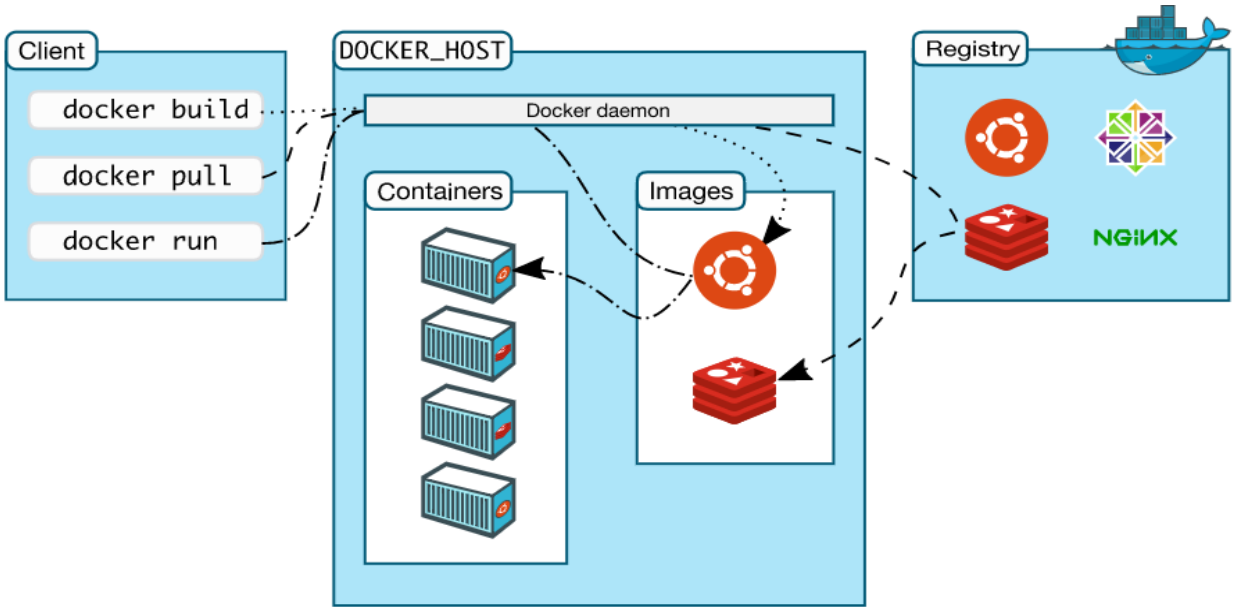


Рис. 2.7 Архітектура Docker

Docker використовує архітектуру клієнт-сервер. Клієнт Docker взаємодіє з демоном Docker, який відповідає за створення, запуск та поширення контейнерів. Клієнт та демон Docker можуть працювати в одній системі, або ви можете підключити клієнт Docker до віддаленого демона Docker. Клієнт та демон обмінюються даними за допомогою REST API, сокетів UNIX або мережевого інтерфейсу.

Docker-демон

Демон Docker (dockerd), який відповідає за створення, роботу та моніторинг контейнерів, а також за побудову та збереження образів. Демон Docker запускається виконанням docker daemon, про що зазвичай дбає операційна система хоста.

Docker-клієнт

Клієнт Docker розташовується зліва і використовується для спілкування з демоном Docker за допомогою HTTP. За замовчуванням це відбувається через сокет домену Unix (UDS або, інакше: IPC, сокет взаємодії між процесами), але також може використовуватися сокет TCP, роблячи можливими віддалені клієнти, або дескриптор файлу для сокетів, керованих systemd. Оскільки вся взаємодія повинна виконуватися через HTTP, досить легко підключатися до віддаленого демону Docker і розробляти зв'язування програмної мови, але це також має наслідки для того, як реалізується функціональність, наприклад вимоги контексту збірки для Dockerfile.

Використовуваний для взаємодії з демоном API добре визначений і документований, що дозволяє розробникам писати програми, які безпосередньо взаємодіють з демоном без використання клієнта Docker. Клієнт Docker і демон поширюються в вигляді виконуваного (двійкового) файлу.

Docker-registry

Реєстри Docker зберігають і поширюють образи. Реєстром за замовчуванням є Docker Hub, який розміщує тисячі загальнодоступних образів а також відповідає за "офіційні" образи. Багато організацій мають власні реєстри, які можуть

застосовуватися для зберігання комерційних або критично важливих образів а також уникати перевантаженості необхідністю завантаження образів з інтернету. Демон Docker завантажує образи використовуючи команду docker pull. Він також автоматично завантажує образи визначені запитом docker run і в інструкції FROM Dockerfile коли вони не доступні локально.

Docker-образ

Docker-образ - це read-only шаблон. Наприклад, образ може містити операційну систему Ubuntu с Apache та додатком на ній. Образи використовуються для створення контейнерів. Docker дозволяє легко створювати нові образи, оновлювати існуючі, або ви можете завантажити образи створені іншими людьми. Образи - це компонент збірки docker-a.

Docker-контейнери

Контейнер являє собою виконуваний екземпляр образу. Ви можете створювати, запускати, зупиняти, переміщати або видаляти контейнери за допомогою Docker API або CLI. Ви можете підключити контейнер до однієї або декількох мереж, підключити до нього сховище або навіть створити новий образ на основі його поточного стану.

За замовчуванням контейнер відносно добре ізольований від інших контейнерів і його хост-машини. Ви можете контролювати, як ізолювати мережу, сховище або інші базові підсистеми контейнера від інших контейнерів або від головної машини.

Контейнер визначається його базовим образом, а також будь-якими параметрами конфігурації, які ви йому надаєте при створенні або запуску. При видаленні контейнера будь-які зміни в його стані, що не зберігаються в постійному сховищі, зникають.

Наступна команда запускає контейнер ubuntu, інтерактивно приєднується до вашого локального сеансу командного рядка та запускає /bin/bash.:

```
$ docker run -i -t ubuntu /bin/bash
```

Коли ви запускаєте цю команду, відбувається наступне (за умови, що ви використовуєте конфігурацію реєстру за замовчуванням):

1. Якщо у вас немає образу ubuntu локально, Docker завантажує його з налаштованого реєстру, якби ви запускали docker pull ubuntu вручну.
2. Docker створює новий контейнер, як якщо б ви запускали команду контейнера docker вручну.
3. Docker виділяє read-only файлову систему в контейнер в якості свого останнього рівня. Це дозволяє запущеному контейнеру створювати або змінювати файли і каталоги в локальній файловій системі.
4. Docker створює мережевий інтерфейс для підключення контейнера до мережі за замовчуванням, оскільки ви не вказали будь-які мережеві параметри. Сюди входить призначення IP-адреси контейнеру. За замовчуванням контейнери можуть підключатися до зовнішніх мереж з використанням мережевого підключення хост-машини.
5. Docker запускає контейнер і виконує /bin/bash. Оскільки контейнер працює в інтерактивному режимі і підключений до вашого терміналу (через опції -і і -t), ви можете вводити в стандартний ввід за допомогою вашої клавіатури, поки стандартний вивід записується на ваш термінал.

Коли ви вводите exit для завершення команди /bin/bash, контейнер зупиняється, але не видаляється. Ви можете запустити його знову або видалити.

Docker-сервіси

Сервіси необхідні для того, щоб змінювати масштаб контейнерів через кілька демонів Docker, які всі працюють разом як безліч з декількома “менеджерами”. Кожен демон Docker обмінюються даними за допомогою Docker API. Сервіси дозволяють визначити бажаний стан, наприклад кількість реплік сервісу, які повинні бути доступні в будь-який момент часу. За замовчуванням сервіс збалансований за навантаженням на всіх робочих вузлах. Для користувача доступний режим swarm, який підтримується в Docker 1.12 і вище.

Технології, що лежать в основі:

Демон Docker використовує "драйвер виконання" для створення контейнера. За замовчуванням це власний драйвер Docker runc, проте також успадковується підтримка для LXC.

Docker використовує технологію під назвою namespaces для забезпечення ізольованого робочого простору. Коли ви запускаєте контейнер, Docker створює набір просторів імен для цього контейнера.

Ці простори імен забезпечують певний шар ізоляції. Кожен аспект контейнера виконується в окремому просторі імен, і його доступ обмежений цим простором імен.

Docker Engine використовує в Linux такі простори імен:

- простір імен pid: ізоляція процесу (PID: Ідентифікатор процесу).
- порожній простір імен: управління мережевими інтерфейсами (NET: Networking).
- простір імен ipc: управління доступом до ресурсів IPC (IPC: InterProcess Communication).
- простір імен mnt: управління точками монтування файлової системи (MNT: Mount).
- простір імен uts: ізоляція ідентифікаторів ядра та версії. (UTS: система Unix Timesharing System).

Cgroups

Docker Engine також використовує іншу технологію,так звану cgoups. Вона обмежує додаток конкретним набором ресурсів. Cgroups дозволяють Docker Engine надавати доступні апаратні ресурси контейнерам та, при необхідності, встановлювати обмеження. Наприклад, ви можете обмежити доступну пам'ять чи кількість ЦПУ конкретному контейнеру.

2.7 Порівняльна характеристика технології віртуалізації та контейнеризації

1. Архітектура.

У віртуальних машинах є гіпервізор між хостовою ОС та гостьовою ОС. Тому кожна гостьова ОС має своє власне ядро, це означає, що віртуальні машини працюють з різними операційними системами на одному фізичному обладнанні. Кожна віртуальна машина має свої власні виконавчі файли, запущені програми. Оскільки кожна віртуальна машина має свої власні виконавчі файли, розмір кожної віртуальної машини становить близько 1 ГБ, тому віртуальні машини є важкими за своєю природою. Також для обробки будь-якої команди виконання займає більше часу через додатковий рівень гіпервізора. Хоча в разі контейнерів ядро розподіляється між хостовою ОС та контейнерами. Це означає, що ми можемо запускати ті ж контейнери, використовуючи ядро хоста. Оскільки контейнери не мають власного ядра, контейнери дуже легкі за своєю природою, а розмір становить близько 100 МБ. Також виконання команд виконується дуже швидко.

2. Переносимість.

Оскільки контейнери відносно невеликого розміру, вони можуть бути розділені за своєю природою, а також створювати та розповсюджувати образи, що складаються з додатків разом з їх залежностями. Крім того, контейнери використовуються в безлічі відкритих та приватних хмарних платформах для прискорення розгортання середовища розробки та тестування за допомогою відправлених образів. Однак, оскільки віртуальні машини мають великий розмір та власне ядро, вони не можуть легко відправлятися з однієї машини на іншу.

3. Безпека.

Оскільки контейнери поділяють той же простір ядра, що і хост-комп'ютер, якщо у будь-якого користувача є права адміністратора для контейнера, хост-машини можуть бути легко зламані. Однак у випадку віртуальних машин, хост та

гостьові машини мають різні ядра і ізольовані один від одного, безпека більш помітна в разі віртуальних машин, ніж контейнерів.

4. Службові випадки.

Контейнери зазвичай використовуються, якщо у вас є архітектура мікросервісів, так як вона дозволяє правильно використовувати функції контейнерів. Якщо вам потрібна повна платформа, яка працює в одній системі, ви можете використати підхід з віртуальними машинами.

5. Продуктивність

Зазвичай Docker швидше, ніж віртуальні машини. В загальному це стосується всіх контейнеризованих додатків, оскільки вони використовують ресурси з головної системи більш ефективним чином.

При використанні технології Docker вам не потрібно призначати системну пам'ять або дисковий простір для контейнера, перш ніж запускати його. Ви можете встановити ліміти на кількість ресурсів, які контейнер може використовувати, якщо це необхідно, але це не означає, що максимальні ресурси, які ви дозволяєте контейнеру, прив'язуються цим контейнером щоразу, коли він запускається. Контейнери більш органічно споживають ресурси,яких вони потребують, не вимагаючи, щоб користувач виділяв більше ресурсів в контейнер, ніж це дійсно необхідно в будь-який момент часу.

Це означає, що контейнери більш ефективно використовують системні ресурси, ніж віртуальні машини. Останні зазвичай вимагають, щоб пам'ять і дискові ресурси були призначені їм до запуску. Навіть якщо програми, запущені всередині віртуальної машини, практично не використовують всі призначені йому ресурси, віртуальна машина все ще монополізує ці ресурси.

Контейнери також мають перевагу в тому, що не потрібно дублювати процеси, вже запущені на хост-системі. За допомогою контейнера ви можете запускати тільки ті процеси, які вам потрібні для будь-якої програми, яку ви хочете розмістити всередині контейнера. Навпаки, віртуальні машини повинні запускати

повну гостьову операційну систему, включаючи багато з тих же процесів, які вже запущені на сервері.

В цьому відношенні контейнери дозволяють більш ефективно розподіляти обмежені ресурси, доступні на головному сервері. Це може призвести до підвищення продуктивності для контейнеризованих додатків, особливо в тому випадку, якщо завантаження на сервері збільшує пропускну здатність, і оптимізація розподілу ресурсів стає важливою. Однак це не означає, що контейнеризовані додатки будуть працювати швидше або повільніше, ніж віртуальні машини. Поки цей додаток має доступ до необхідних системних ресурсів, продуктивність буде приблизно однаковою, незалежно від того, чи використовуєте ви віртуальну машину, Docker або bare metal.

Docker	**KVM**	**XenServer (Xen)**	**CoreOS (LXC)**
Менший час завантаження системи	Більший час завантаження системи	Більше накладних витрат (витрати ресурсів)	Менше накладних витрат (витрати ресурсів)
Більша швидкість обчислень	Менша швидкість обчислень	Менше часу для виконання запиту	Більший час для виконання запиту
Відсутня гостьова ОС	Працює незалежно	Краще в сенсі рівномір-ного розподілу ресурсів	Краще в сенсі виконання окремих процесів

Таблиця 2 Порівняння на основі технологій віртуалізації і контейнеризації

Висновки до розділу:

В даному розділі було проведено аналіз принципів побудови корпоративної VoIP-мережі на базі платформи Asterisk. Здійснено огляд можливостей використання технології віртуалізації Vagrant та контейнеризації Docker, а також порівняння іх в теоретичному розрізі.

Проекти віртуалізації були в центрі уваги багатьох IT-організацій досить довгий час, основною метою яких є консолідація серверів або центрів обробки даних, зниження капітальних витрат та капіталовкладень (операційні витрати). Віртуалізацію в простих термінах можна визначити як консолідацію обчислювальних потужностей, пам'яті, пропускної здатності мережі та ємкості сховища з використанням меншої кількості апаратних ресурсів та можливість розподіляти ресурси на серверах. Головна мета віртуалізації - максимально ефективно використовувати доступні IT-ресурси. Вона дозволяє відокремлювати всі обчислювальні ресурси, роблячи їх доступними для загального пулу віртуальних машин. В результаті цього нові або старі служби можуть бути додані, видалені або змінені без будь-яких складнощів.

Віртуалізація допомагає IT-інфраструктурі з точки зору:

- зниження витрат;
- безперервність бізнес-процесів;
- висока доступність;
- швидке оновлення віртуальних машин;
- корпоративне управління.

Віртуалізація може бути включена з використанням багатьох технологій, у тому числі найбільш популярними такими як, віртуальні машини та контейнеризація. Гіпервізор та віртуальні машини були найбільш використовуваним підходом для розгортання віртуального робочого середовища через зрілість та масову підтримку спільноти.

Віртуалізація контейнерів тепер швидко стає надійною альтернативою до традиційної віртуалізації, тим самим забезпечуючи нові гнучкі функції, а також вирішення проблеми з центрами обробки даних. Різниця між віртуалізацією та контейнеризацією може бути охарактеризована за допомогою управління використанням ресурсів на рівні операційної системи.

Віртуальні машини покладаються на гіпервізор, який встановлюється або поверх операційної системи хоста, або поверх фізичного сервера. Після установки гіпервізора віртуальні машини можуть бути створені з доступних обчислювальних ресурсів системи. Кожна віртуальна машина в свою чергу володіє власною операційною системою і запущеними додатками. Віртуальні машини забезпечують повну ізоляцію від інших, що працюють на одному фізичному вузлі.

З іншого боку, в контейнерах спочатку встановлюється операційна система хоста, а потім – "контейнерний рівень", наприклад. LXC або libcontainer. Як тільки віртуалізований контейнерний рівень буде встановлено, контейнери можуть бути розгорнуті, використовуючи наявні системні ресурси з відповідними додатками всередині них. Ці контейнери забезпечують ізоляцію, але спільно використовують одну і ту ж операційну систему з головною машиною. Контейнери вважаються більш ефективними в плані використання ресурсів, ніж віртуальні машини, оскільки усувається необхідність в додаткових ресурсах для кожної операційної системи. Відповідно, результуючі екземпляри стають все менше і швидше для розгортання та управління. В результаті цього одна фізична система може розміщувати більше контейнерів в порівнянні з віртуальними машинами. Це безпосередньо впливає на економію використання хмарної інфраструктури, забезпечуючи значне скорочення витрат, але не гарантують повної відмовостійкості всіх запущених контейнерів. Однак контейнери та віртуальні машини можуть співіснувати в одному і тому середовищі, доповнюючи один одного, тим самим розширюючи доступний набір інструментів для операторів центрів обробки даних для надання ресурсів.

3. ШЛЯХИ ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ МЕРЕЖІ IP-ТЕЛЕФОНІЇ НА БАЗІ ТЕХНОЛОГІЇ ASTERISK

Для розгортання Asterisk з використанням технології віртуалізації Vagrant, потрібна підтримка віртуалізації в середовищі Linux на платформі x86. Використаємо KVM, яка вимагає процесора з підтримкою розширення віртуалізації. Ці розширення називаються Intel VT або AMD-V. Щоб перевірити, чи є у вас підтримка ЦП, виконайте наступну команду:

egrep '^flags.*(vmx|svm)' /proc/cpuinfo

Якщо вивід команди нічого не показав, це означає, що ваша система не підтримує відповідні розширення віртуалізації. Ви все ще можете використовувати QEMU/KVM, але емулятор повернеться до віртуалізації програмного забезпечення, що набагато повільніше.

Виконайте наступну команду для установки обов'язкових, стандартних і додаткових пакетів віртуалізації:

dnf group install –with-optional virtualization

Після установки пакетів запустіть службу libvirtd:

systemctl start libvirtd

Щоб запустити службу при завантаженні, запустіть:

systemctl enable libvirtd

Щоб використовувати Vagrant з libvirt, вам потрібно встановити пакет vagrant-libvirt:

sudo yum install vagrant-libvirt

Після цього переконайтеся, що працює демон libvirt, і що у модулі завантажено модуль kvm в ядрі:

lsmod | grep kvm

```
[ingvar95@devops ~]$ lsmod | grep kvm
kvm_intel             200704  4
kvm                   585728  1 kvm_intel
irqbypass              16384  6 kvm
```

Якщо у вас немає можливостей KVM, вам потрібно буде використовувати драйвер qemu у вашому файлі Vagrantfile. Зауважте, що драйвер qemu може серйозно вплинути на ефективність вашої віртуальної машини Vagrant.

```
Vagrant.configure("2") do |config|
...
  config.vm.provider :libvirt do |libvirt|
    libvirt.driver = "qemu"
  end
...
end
```

Для моніторингу ключових параметрів системи будемо використовувати Glances, який представляє сучасний крос-платформенний інструмент моніторингу, що працює в режимі реального часу, з додатковими функціями. Він може працювати в різних режимах: як автономний, в режимі клієнта/ сервера, так і в режимі веб-сервера. Також можна використати такий інструмент, як Prometheus. Він збирає показники з налаштованих цілей за певними інтервалами, оцінює вирази правил, відображає результати та може викликати сповіщення.

Для збереження відповідних параметрів використовується InfluxDB. База даних з відкритим вихідним кодом та масштабованою часової серії для показників, подій та аналітики в реальному часі.

Інструмент, за допомогою якого ми будемо відображати ключові метрики(параметри системи) Grafana - це відкрите джерело, багатофункціональний, потужний, розширюваний крос-платформенний інструмент для моніторингу та аналізу показників, з налаштовуваними інформаційними панелями. Це де-факто програмне забезпечення для аналізу даних.

cAdvisor надає користувачам контейнерів розуміння використання ресурсів та характеристик. Він запускає демон, який збирає, обробляє та експортує

інформацію про запуск контейнерів. Зокрема, для кожного контейнера зберігаються параметри ізоляції ресурсу, гістограми використання ресурсів та статистики мережі. cAdvisor має власну підтримку контейнерів Docker і повинен підтримувати будь-який інший тип контейнера.

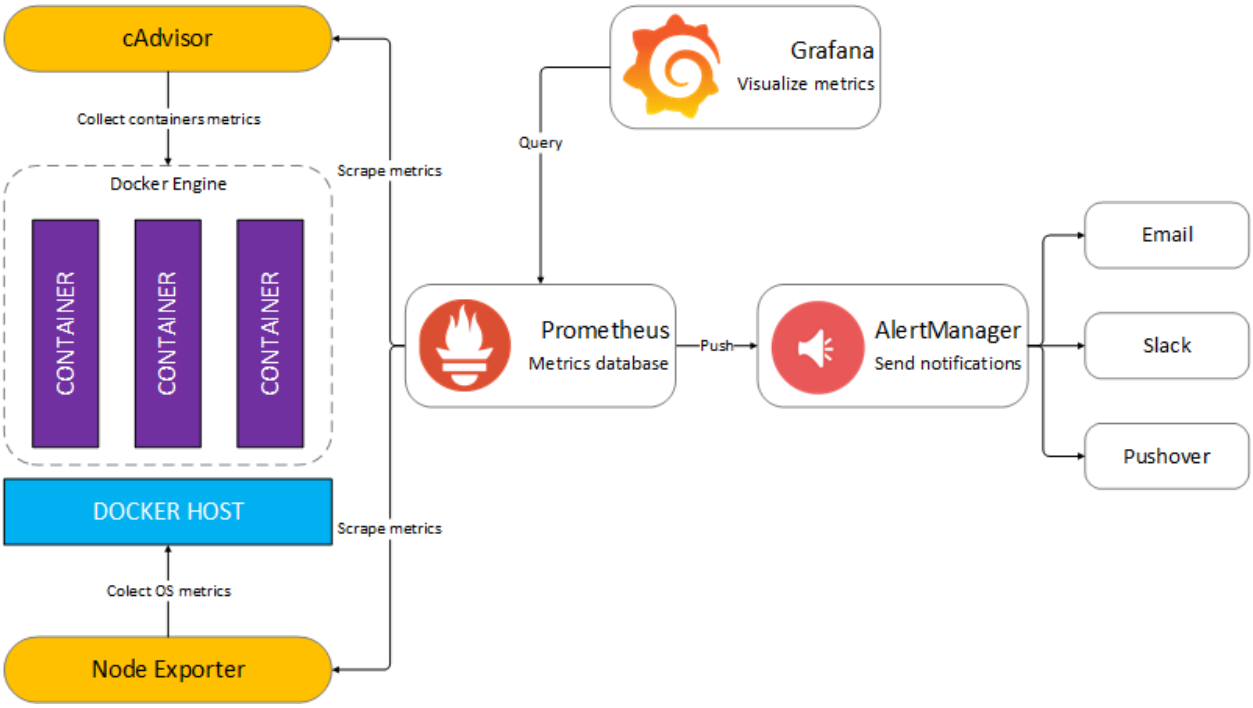


Рис. 3.1 Архітектура системи моніторингу

3.1 Побудова макету IP-АТС на платформі Asterisk з використанням технології віртуалізації Vagrant

Створимо тестову директорію для розгортання Asterisk:

mkdir –p asterisk

В даній директорії ініціюємо автоматичне створення Vagrantfile:

vagrant init

Відредагуємо Vagrantfile за допомогою будь-якого текстового редактора

```
Vagrant.configure("2") do |config|

  # Asterisk VM
  config.vm.define "asterisk" do |node|
    node.vm.hostname = "asterisk"
    node.vm.box = "centos/7"
```

```
    node.vm.box_check_update = false
    node.vm.synced_folder '.', '/vagrant', :disabled => true
    node.vm.network "private_network", ip: "10.0.15.10"
    node.vm.provider :libvirt do |libvirt|
      libvirt.driver = 'kvm'
      libvirt.username = 'root'
      libvirt.cpus = 2
      libvirt.memory = 2048
    end
  config.vm.provision :shell, :path => "provision.sh"
  end
end
```

А також створимо файл provision.sh для конфігурації Asterisk:

```
#!/bin/bash

echo "Running core linux setup"

# Disable selinux
setenforce 0
sed -i 's/\(^SELINUX=\).*/\SELINUX=disabled/' /etc/sysconfig/selinux
sed -i 's/\(^SELINUX=\).*/\SELINUX=disabled/' /etc/selinux/config

# Install base/common packages
yum install -y epel-release
yum -y update
yum -y install -y  mariadb-server mariadb php php-mysql php-mbstring httpd ncurses-
devel newt-devel libxml2-devel libtiff-devel audiofile-devel gtk2-devel  kernel-devel
git php-process  wget vim php-xml uuid-devel sqlite-devel net-tools gnutls-devel php-
pear

# Run MariaDB(MySQL)
systemctl enable mariadb.service
systemctl start mariadb

#Run http-server
systemctl start httpd
systemctl enable httpd

# Adding user
adduser asterisk -M -c "User for voip"

# Stop/Disable firewalld if it exists
if systemctl status firewalld > /dev/null; then
  echo "stopping/disabling firewalld"
  systemctl disable firewalld
  systemctl stop firewalld
fi

echo "Finished core linux setup"
```

```
BASE_DIR="/usr/src/asterisk"

if [ ! -d "${BASE_DIR}" ]; then
  mkdir ${BASE_DIR}
fi
cd ${BASE_DIR} || exit 1

################################################ Installing Asterisk
############################################

# Downloading the necessary components
wget http://downloads.asterisk.org/pub/telephony/dahdi-linux-complete/dahdi-linux-
complete-current.tar.gz
wget http://downloads.asterisk.org/pub/telephony/libpri/libpri-current.tar.gz
wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-13-current.tar.gz
wget -O jansson.tar.gz https://github.com/akheron/jansson/archive/v2.7.tar.gz
wget http://www.pjsip.org/release/2.4/pjproject-2.4.tar.bz2

# Extracting files
tar xvfz dahdi-linux-complete-current.tar.gz
tar xvfz libpri-current.tar.gz

# Compiling DAHDI
cd dahdi-linux-complete-*
make all
make install
make config

# Installing libpri
cd ${BASE_DIR}/libpri-*
make
make install

# Installing pjproject
cd ${BASE_DIR}
tar -xjvf pjproject-2.4.tar.bz2
cd pjproject-2.4
CFLAGS='-DPJ_HAS_IPV6=1' ./configure --prefix=/usr --enable-shared --disable-sound --
disable-resample --disable-video --disable-opencore-amr --libdir=/usr/lib64
make dep
make
make install

# Installing jansson
cd ${BASE_DIR}
tar vxfz jansson.tar.gz
cd jansson-*
autoreconf -i
./configure --libdir=/usr/lib64
```

```
make
make install

# Compiling asterisk
cd ${BASE_DIR}
tar xvfz asterisk-13-current.tar.gz
cd asterisk-*
contrib/scripts/install_prereq install
./configure --libdir=/usr/lib64
contrib/scripts/get_mp3_source.sh
make
make install
make config
ldconfig
chkconfig asterisk off

# Expose the necessary rights to files and directories
chown asterisk. /var/run/asterisk
chown -R asterisk. /etc/asterisk
chown -R asterisk. /var/{lib,log,spool}/asterisk
chown -R asterisk. /usr/lib64/asterisk
chown -R asterisk. /var/www/


############################################## Installing FreePBX
##################################################

# Configuring http-server
sed -i 's/\(^upload_max_filesize = \).*/\120M/' /etc/php.ini
sed -i 's/^\(User\|Group\).*/\1 asterisk/' /etc/httpd/conf/httpd.conf
sed -i 's/AllowOverride None/AllowOverride All/' /etc/httpd/conf/httpd.conf
systemctl restart httpd

# Installing FreePBX
cd ${BASE_DIR}
wget http://mirror.freepbx.org/modules/packages/freepbx/freepbx-13.0-latest.tgz
tar xfz freepbx-13.0-latest.tgz
cd freepbx
./start_asterisk start
./install -n
```

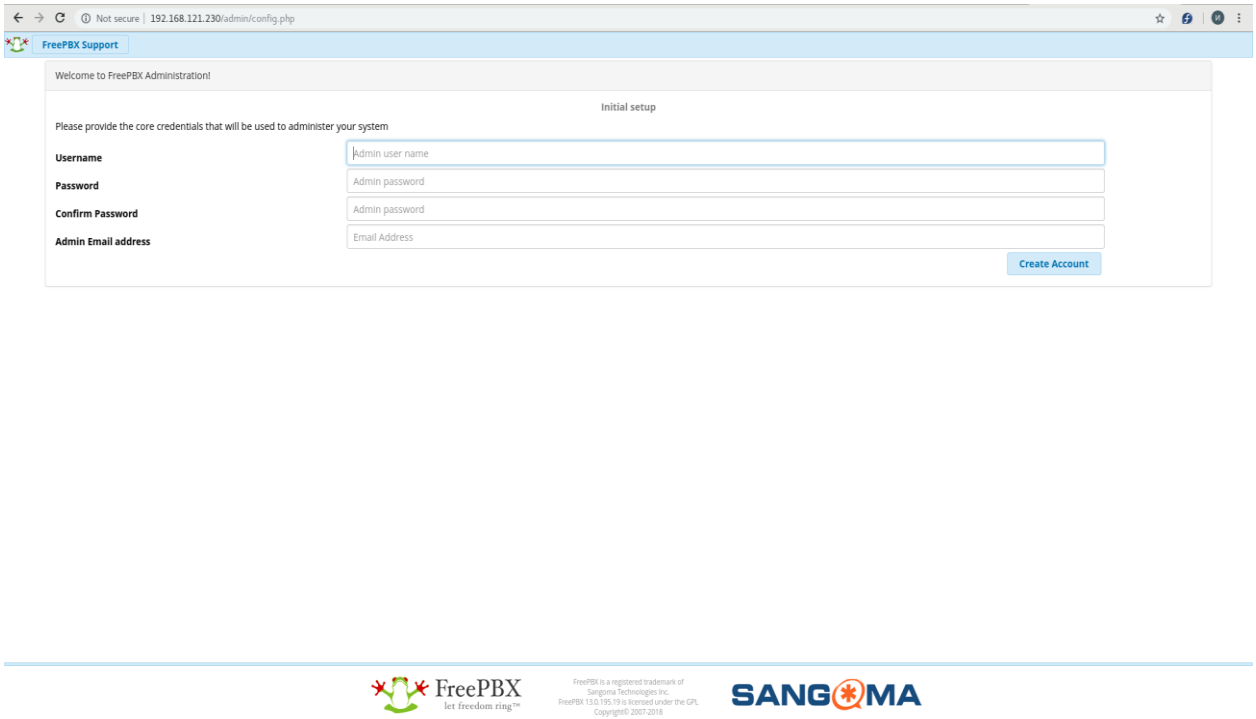
Виконаємо команду для запуску віртуальної машини:

vagrant up –debug

Перевіримо встановлення Asterisk за допомогою команди asterisk -rvv.

```
 Loading res_manager_devicestate.so.
  == Manager registered action DeviceStateList
  == res_manager_devicestate.so => (Manager Device State Topic Forwarder)
 Loading res_manager_presencestate.so.
  == Manager registered action PresenceStateList
  == res_manager_presencestate.so => (Manager Presence State Topic Forwarder)
Asterisk Ready.
asterisk*CLI>
```

Перевіримо встановлення графічного інтерфейсу користувача FreePBX виконавши запит на IP-адресу віртуальної машини 192.168.121.230.



3.2 Побудова макету IP-АТС на платформі Asterisk з використанням технології контейнеризації Docker

Для розгортання Asterisk допомогою технології Docker можна використати вже готові шаблони контейнерів, завантажені з публічного сховища DockerHub, використовуючи команди:

NAME_ASTERISK=asterisk

NAME_FASTAGI=fastagi

docker run –p 4573: 4573 –name $NAME_FASTAGI –d –t dougbtv/fastagi

docker run –name $NAME_ASTERISK –net=host –d –t dougbtv/asterisk

Іншим способом є створення Dockerfile, де ви зможете повністю описати необхідну конфігурацію вашого контейнера. Наприклад, для Asterisk Dockerfile має наступний вигляд:

```
FROM centos:centos7
RUN yum update -y
RUN yum install -y epel-release
RUN yum install git kernel-headers gcc gcc-c++ cpp ncurses ncurses-devel libxml2
libxml2-devel sqlite sqlite-devel openssl-devel newt-devel kernel-devel libuuid-devel
net-snmp-devel xinetd tar jansson-devel make bzip2 gettext –y
WORKDIR /tmp
# Get pj project
RUN git clone -b pjproject-2.4.5 --depth 1 https://github.com/asterisk/pjproject.git
# Build pj project
WORKDIR /tmp/pjproject
RUN ./configure --prefix=/usr --libdir=/usr/lib64 --enable-shared --disable-sound --
disable-resample --disable-video --disable-opencore-amr 1> /dev/null
RUN make dep 1> /dev/null
RUN make 1> /dev/null
RUN make install
RUN ldconfig
RUN ldconfig -p | grep pj
ENV AUTOBUILD_UNIXTIME 123124
WORKDIR /tmp
# Download asterisk.
RUN git clone -b certified/13.8 --depth 1 https://gerrit.asterisk.org/asterisk
WORKDIR /tmp/asterisk
# Configure
RUN ./configure --libdir=/usr/lib64 1> /dev/null
# Remove the native build option from:
https://wiki.asterisk.org/wiki/display/AST/Building+and+Installing+Asterisk
RUN make menuselect.makeopts
RUN menuselect/menuselect \
  --disable BUILD_NATIVE \
  --enable cdr_csv \
  --enable chan_sip \
  --enable res_snmp \
  --enable res_http_websocket \
  --enable res_hep_pjsip \
  --enable res_hep_rtcp \
  menuselect.makeopts
# Continue with a standard make.
RUN make 1> /dev/null
RUN make install 1> /dev/null
RUN make samples 1> /dev/null
WORKDIR /
# Update max number of open files.
RUN sed -i -e 's/# MAXFILES=/MAXFILES=/' /usr/sbin/safe_asterisk
CMD asterisk –f
```

Щоб створити контейнер, потрібно використати наступну команду:

docker build .

Переконаймося в наявності запущених контейнерів Asterisk за допомогою команди:

docker ps

```
[ingvar95@devops ~]$ docker ps
CONTAINER ID        IMAGE                COMMAND                  CREATED             STATUS              PORTS                    NAMES
986d954034c3        dougbtv/asterisk     "/bin/sh -c '/usr/sb…"   38 minutes ago      Up 38 minutes                                asterisk
52a9dbd25b05        dougbtv/fastagi      "/bin/sh -c 'service…"   41 minutes ago      Up 41 minutes       0.0.0.0:4573->4573/tcp   fastagi
```

3.3 Оцінка продуктивності реалізації ПО Asterisk з використанням системи моніторингу Glances, cAdvisor, Prometheus, бази даних InfluxDB та системи для відображення метрик Grafana

Для відображення ресурсів використаних віртуальною машиною з Asterisk використаємо утиліту для моніторингу Glances. Як можна спостерігати на рисунку нижче віртуальна машина загалом використовує прийнятну для системи кількість оперативної пам’яті та незначну кількість ЦПУ. Також слід відмітити такі показники, як VIRT(віртуальний розмір процесу. Показує загальну кількість пам'яті, яку здатна адресувати програма в даний момент часу), RES(показує скільки фізичної пам'яті використовує процес. Це значення, буде менше значення VIRT, так як більшість програм залежать від розділюваної Сі бібліотеки (C library)).

```
Processes filter: qemu on column username ('ENTER' to edit, 'E' to reset)
TASKS   1 (5 thr), 0 run, 1 slp, 0 oth sorted automatically by memory consumption

CPU%   MEM%  VIRT  RES     PID USER          TIME+ THR  NI S  R/s W/s  Command
0.2    9.3   2.65G 731M   2283 qemu           0:30 5     0 S    ? ?    /usr/bin/qemu-system-x86_64
_____________________________________________________________________________
0.2    9.3   2.65G 731M                                    0 0      < current
0.0    9.3   2.65G 731M                                             < min ('M' to reset)
155.5  100.1 119G  7.68G                                             < max ('M' to reset)
```

Рис. 3.2 Статистика використання ресурсів віртуальною машиною Asterisk

Рис. 3.3 Використання ресурсів віртуальною машиною

Для аналізу використання системних ресурсів контейнерами, можна скористатися, для початку, командою docker stats. Glances також надає можливість для моніторингу стану контейнерів.

```
CONTAINERS 2 (served by Docker 18.09.0)

 Name        Status  CPU%    MEM    /MAX  IOR/s  IOW/s   Rx/s   Tx/s Command
 asterisk    running  0.3  34.2M  7.68G    0B     0B      _      _ _
 fastagi     running  0.0  15.9M  7.68G    0B     0B     0b     0b _
```

Рис. 3.4 Статистика використання ресурсів контейнерами Asterisk

Як можна побачити, використання ЦПУ знаходиться майже на тому самому рівні. Натомість використання оперативної пам’яті в декілька десятків разів менше, ніж для віртуальної машини.

Графіки використання ЦПУ та оперативної пам’яті контейнером з Asterisk зображені нижче.

Рис. 3.5 Загальне використання та використання ресурсів ЦПУ для кожного ядра системи

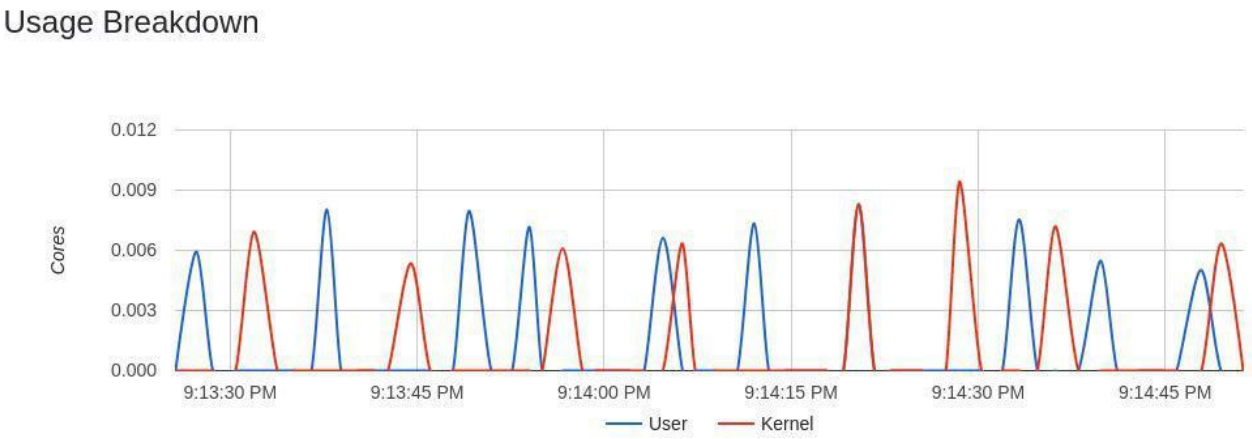


Рис. 3.6 Розбивка по використанню ЦПУ між просторами ядра та користувача

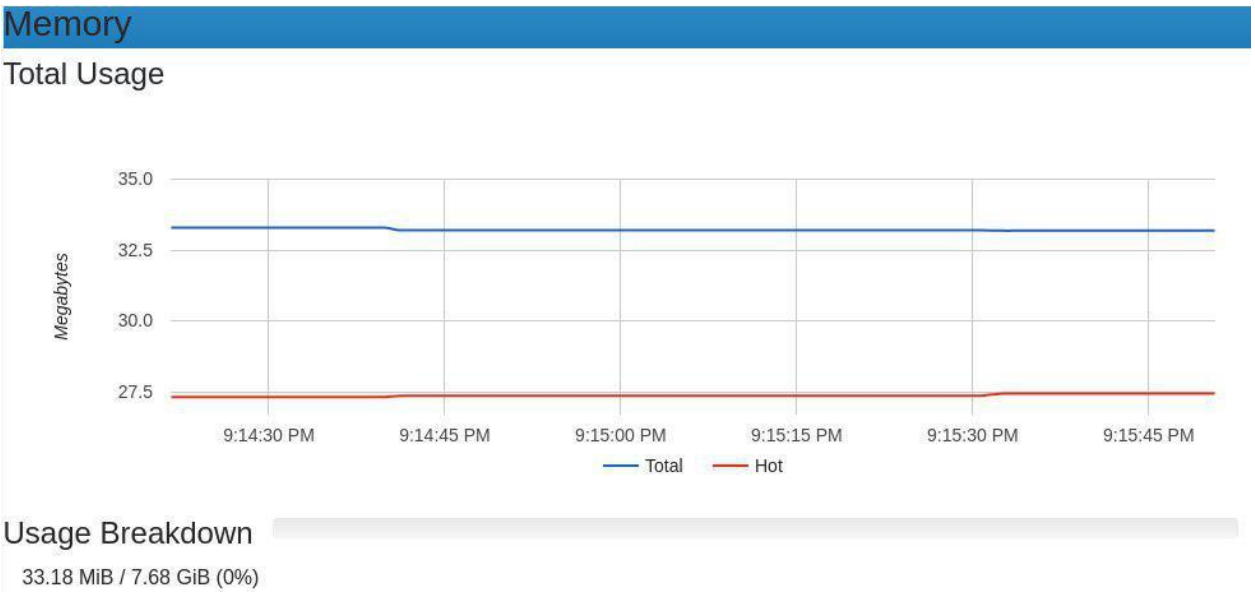


Рис. 3.7 Графік використання оперативної пам’яті

З огляду оцінки продуктивності операцій вводу-виводу контейнери Docker практично не потребують оптимізації, в той час як у випадку з віртуальними машинами, I/O може бути поліпшена за рахунок використання багатьох параметрів оптимізацій, таких як включення апаратної віртуалізації або використання оптимальної кількості пам'яті та методів підкачки. Однак величезну різницю в параметрах продуктивності як і раніше не можна ігнорувати, і віртуальні машини повинні були пройти довгий шлях в зіставленні продуктивності з контейнерами.

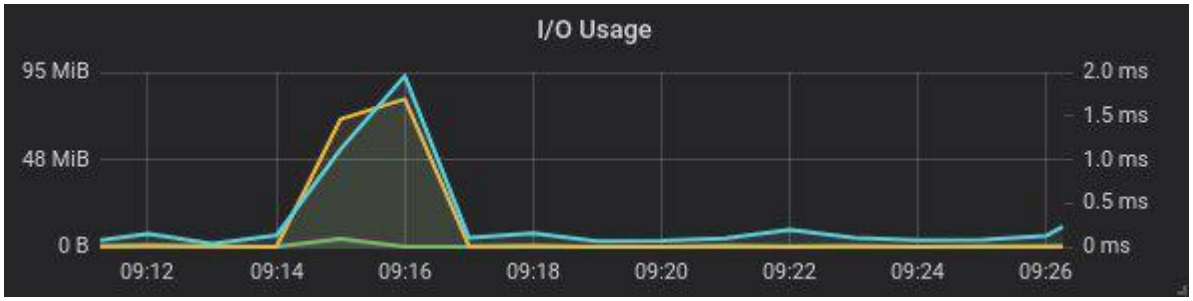


Рис. 3.8 Використання системи вводу-виводу контейнерами

Висновки до розділу:

В цьому розділі ми оцінювали продуктивність віртуальних машин та контейнерів. Ми використовували віртуальні машини Linux запущені за допомогою технологій віртуалізації Vagrant, KVM/QEMU, libvirt та контейнери Docker для наших експериментів, оскільки вони є технологіями з відкритим вихідним кодом.

Продуктивність оцінювалася з використанням систем моніторингу стану Linux-сервера Glances, моніторингу стану контейнерів cAdvisor, системи збору метрик Prometheus, бази даних InfluxDB, системи відображення метрик Grafana. З результатів ми спостерігали, що частка використання ЦПУ майже еквівалентна в обох випадках. Натомість використання оперативної пам’яті різко відрізняється. У випадку віртуальних машин вона більша в декілька десятків разів, що не може не позначитися на роботу системи вцілому.

Оцінюючи продуктивність систему вводу-виводу контейнери мають більшу продуктивність, ніж віртуальні машини, з точки зору швидкості виконання та кількості транзакцій. Продуктивність віртуальних машин значно погіршується при масштабуванні, що виявляється менш оптимальним в порівнянні з контейнерами, де продуктивність залишається досить постійною при збільшенні кілшькості. Контейнери знову пропонують кращу продуктивність в разі реалізації Asterisk, виконуючи більшу кількість операцій читання і запису.

Також заслуговує уваги час завантаження і виключення для обох технологій. Час завантаження та виключення у випадку контейнерів досить рівномірний, незалежно від кількості. У той час як для віртуальних машин час збільшується лінійно, коли розподіл пам'яті для віртуальної машини перевищує фактичний розмір оперативної пам'яті основного вузла, що призводить вразливості.

ВИСНОВОК

В даний час телекомунікаційні технології перебувають на такому високому рівні розвитку, що впроваджуються в будь-які пристрої, починаючи від найпростіших комп'ютерних модемів, забезпечуючи вихід в мережу Інтернет, і закінчуючи системами віддаленого керування автомобілями або побутовою технікою, реалізовані за допомогою стільникових телефонів - смартфонів. На тлі бурхливого розвитку телекомунікацій, змінам піддаються основи будь-яких видів зв'язку - телефонні мережі, удосконалюються шляхи виходу до них, методи і системи проектування АТС, що забезпечують доступ всім абонентам до послуг зв'язку. Особливо цікаво розглянути можливості проектування і створення АТС, що використовуються повсюдно в самих різних компаніях - починаючи від міні-АТС, до послуг яких вдаються невеликі компанії, стартапи, закінчуючи потужними комунікаційними структурами, які допомагали національним і міжнародним організаціям, офіси яких розташовуються на великій відстані один від одного.

Постійне вдосконалення і поширення в усіх областях мережі Інтернет призвело до зародження нового виду зв'язку, який грунтується на використанні протоколів Інтернет і IP - серверів, здатних перетворювати в цифрову форму звичайний голосовий сигнал. Цей новий вид зв'язку назвали IP або VoIP телефонією. даний вид технології дозволив перетворити аудіо- та відео-переговори, у зручний, якісний, універсальний і що також важливо дешевий засіб спілкування, який став доступним практично кожному. В даний час IP протокол використовується не тільки в мережі Інтернет, а й в інших мережах передачі даних з пакетною комутацією (локальних, корпоративних, регіональних). І у всіх цих мережах, є можливість передавати мовленнєві повідомлення з використанням пакетів даних. такий спосіб передачі мови отримав назву IP-телефонія.

IP-телефонія дозволяє істотно економити необхідну смугу пропускання каналів, що неминуче веде до зниження тарифів, особливо на міжміські і міжнародні телефонні розмови. технологія IP - телефонії надає можливість

отримання високоякісного телефонного з'єднання між двома і більше абонентами, що знаходяться в різних куточках планети. Телефонні переговори, організовані з допомогою VoIP технології є більш надійними і універсальними в порівнянні з традиційними, так як замість проміжних комутаторів, станцій та інших пристроїв використовується програмний продукт з відкритим вихідним кодом Asterisk.

Розвиток корпоративної мережі на основі IP-телефонії обумовлена не тільки можливістю зниження витрат на телефонні розмови та технічне обслуговування інфраструктури (і це, безумовно, має значення). В стратегічному плані IP-телефонія є єдиною технічною платформою, яка дозволить об'єднати рішення для передачі даних та голосу, а також для обробки і наступного використання всієї інформації.

В роботі були розглянуті теоретичні питання побудови мереж IP-телефонії, проведено аналіз основних протоколів VoIP, архітектура, основні способи підключення та елементи мережі, а також огляд найпоширенішого протоколу SIP. Здійснено аналіз існуючих принципів побудови корпоративних мереж передачі даних на базі технології VoIP та виокремлені методи практичної реалізації з використанням технологій віртуалізації Vagrant, KVM/QEMU, libvirt та контейнеризації Docker використовуючи наявну інфраструктуру та додаткове програмне забезпечення, а саме програмний продукт з відкритим вихідним кодом Asterisk.

Таким чином, ця робота сприяє порівняльному аналізу середовищ віртуалізації та контейнеризації. Наші результати оцінки показують, що контейнери забезпечують кращі показники продуктивності з точки зору оперативної пам'яті, операцій зчитування/запису, часу затраченому на безпосереднє завантаження/виключення контейнерів. Особливо це помітно при збільшенні кількості контейнерів/віртуальних машин.