RITI: Solución híbrida de PBX virtuales con aprovisionamiento automático en la nube

RITI: Hybrid Virtual PBX Solution with Automatic Cloud Provisioning

Juan José Behrend [1], Gastón Pouquette [2]

Recibido: Abril 2020 Aceptado: Noviembre 2019

Resumen.- Si bien la telefonía sobre el protocolo IP tiene más de una década entre nosotros, la ubicuidad de Internet, las nuevas tecnologías de hardware abierto, software libre y plataformas en la nube invitan a volver a pensar una solución en este nuevo contexto. Como tal solución se propone en este proyecto a RITI, compuesto por un equipo de pequeñas dimensiones a instalar en dependencias del cliente y un sistema de aprovisionamiento automático y conmutación telefónica en la nube. El principal desafío para la obtención de tal producto fue la gran cantidad de componentes de hardware y software a integrar, pasando puntos tan diversos como un Raspberry con su adaptador de interfaces Grandstream, Asterisk con sus contextos de troncales y protocolos, plataformas como AWS y Docker. Es importante destacar que los objetivos planteados fueron logrados, obteniendo una solución funcional, resiliente y escalable donde la complejidad se centraliza en los componentes alojados en la nube. De esta forma se extienden a la telefonía las ventajas constatadas en los últimos años de la virtualización y la consolidación en la nube. Finalmente, todos los componentes de software del sistema son o bien proyectos libres de código abierto, o bien desarrollos originales de los autores del mismo.

Palabras clave: Cloud Computing; Despliegue Continuo (CD); Telefonía -PBX; SDN

Summary.- *Although telephony over IP protocol has more than a decade among us, the ubiquity of the Internet, new open hardware technologies, free software and cloud platforms invite us to rethink a solution in this new context. As such a solution is proposed in this project to RITI, made up of a small team to be installed in customer premises and an automatic provisioning system and telephone switching in the cloud. The main challenge for obtaining such a product was the large number of hardware and software components to be integrated, passing points as diverse as a Raspberry with its Grandstream interface adapter, Asterisk with its trunk and protocol contexts, platforms such as AWS and Docker. It is important to highlight that the stated objectives were achieved, obtaining a functional, resilient and scalable solution where complexity is centralized in the components hosted in the cloud. In this way, the benefits of virtualization and consolidation in the cloud have been extended to telephony in recent years. Finally, all the software components of the system are either free open source projects, or original developments by the authors of the same.*

Keywords: *Cloud Computing; Continues deployment (CD); Telephony -PBX; SDN*

[1] Ing. en Telecomunicaciones, Universidad ORT Uruguay, juanjosebehrend@gmail.com, ORCID iD: 0000-0003-2074-2012

[2] Ing. en Telecomunicaciones, Universidad ORT Uruguay, gpouquette@gmail.com, ORCID iD: 0000-0002-2820-3243

1. Introducción.- RITI (nombre del proyecto) consiste en una automatización desarrollada con diversos componentes de software libre, se crean instancias virtuales en Docker alojadas en la nube que funcionará como su motor de servicio, donde a partir de la invocación de un servicio, este levante automáticamente el servicio. Este servicio constará de una instancia de virtual PBX en la nube, sin necesidad alguna de configuración ni conocimiento técnico, esta misma será única para cada dispositivo permitiéndole así poder integrarse en la red de RITI y así realizar el nexo con otras terminales. A su vez esto no requiere trámites, compromisos adicionales ni contacto con el proveedor actual de telefonía. Es compatible con los equipos de telefonía tradicional, y aprovecha las ventajas del procesamiento en la nube (escalabilidad, redundancia, estabilidad, entre otras).

2. Solución propuesta.- En este proyecto se propone una alternativa tecnológica novedosa con el fin de lograr una infraestructura para el cliente que sea simplemente instalar y usar, mediante una automatización desarrollada con diversos componentes de software libre, se crean instancias virtuales en Docker alojadas en la nube que funcionara como su motor de servicio, permitiendo que las comunicaciones del cliente pasen de telefonía tradicional a IP, manteniendo número y aparato telefónico de las sucursales.

RITI (nombre del proyecto) automatizará la conexión del dispositivo en la red privada de la solución, donde a partir de la invocación de un servicio, este levante automáticamente el servicio. Este servicio constará de una instancia de virtual PBX en la nube, sin necesidad alguna de configuración ni conocimiento técnico, esta misma será única para cada dispositivo permitiéndole así poder integrarse en la red de RITI y así realizar el nexo con otras terminales. A su vez esto no requiere trámites, compromisos adicionales ni contacto con el proveedor actual de telefonía.

Esta solución es un híbrido que reduce al mínimo indispensable el procesamiento en el equipo de la sucursal, ya que como veremos más adelante, la toma de decisiones del flujo de las llamadas es realizado en la nube. Es compatible con los equipos de telefonía tradicional, y aprovecha las ventajas del procesamiento en la nube (escalabilidad, redundancia, estabilidad, entre otras).

Algunas de las características de RITI de cara del cliente son que, en caso de corte de electricidad, y que el equipo se quede sin alimentación, las llamadas siguen funcionando como en la telefonía tradicional (desviando las llamadas a la PSTN). Lo mismo ocurre en caso de que el hardware en la dependencia del cliente se quede sin conectividad a Internet. El cliente indistintamente de si la llamada es enviada vía telefonía IP, o por la vía tradicional, disca siempre el mismo número, no es necesario recordar números nuevos.

2.1. Origen de la idea.- Se descubrió que existía una gran cantidad de empresas con la necesidad de reducir sus costos en llamadas telefónica. Por esta razón, surge la idea de desarrollar un hardware que presente una configuración mínima o nula, un coste y dimensiones reducidos, y que permita tanto la telefonía IP como la telefonía tradicional. Al mismo tiempo, lograr que la dependencia donde se instale el hardware no sea necesariamente un lugar físico perteneciente a la empresa. Es decir, lograr diseñar un equipo, que sea tan simple como conectar un teléfono, pero internamente tan inteligente como la mejor telefonía IP.

2.2. Innovación.- La solución RITI es innovadora en cuatro aspectos fundamentales como lo son infraestructura, formalidades y tramitación, instalación y tecnología. Por un lado, el Hardware a instalar en el cliente tiene un costo estimado de un 20% de otras soluciones que existen actualmente [1-4]. A su vez, el conocimiento técnico necesario para realizar la instalación de dicho hardware es nulo. Por otro lado, este producto no requiere de trámites y compromisos con proveedores de servicios de telefonía ya que funciona sobre el servicio actual, sin configuraciones adicionales.

Respecto a su tecnología, RITI presenta se diferencia ya que las costosas soluciones actuales, requieren o bien migrar toda la tecnología y equipos a IP, o bien instalar costosos equipos en las sucursales con todo el procesamiento en dicho equipo. Como se mencionó anteriormente esta solución es un híbrido que reduce el procesamiento en el equipo de la sucursal, es compatible con los equipos de telefonía tradicional, y aprovecha las ventajas del procesamiento en la nube (escalabilidad, redundancia, estabilidad, entre otras).

2.3. Principales características del proyecto.- RITI es un sistema para la gestión de PBX virtuales en la nube y su vínculo con los dispositivos físicos instalados en las dependencias de clientes a través de Internet. Dichos equipos serán nombrados "RitiBox". El sistema permite crear instancias de PBX virtuales en la nube, como respuesta a la instalación de un RitiBox. Con esto se logra realizar de manera automática el despliegue del servidor, la configuración de la PBX cloud y local, y la vinculación de las PBX. Estas instancias fueron nombradas RitiCloud y se realizó mediante la implementación de las últimas tecnologías de virtualización, cómputo en la nube, y redes definidas por software.

2.4. Desafíos encontrados.- El mayor desafío de todo el proyecto fue integrar más de 20 tecnologías distintas logrando que sea de rápido aprovisionamiento y con mínima interacción humana. En la Figura I se puede visualizar las distintas tecnologías:

Figura I.- Distintas tecnologías integradas en RITI

3. Capas del proyecto.- El proyecto podemos dividirlo en seis capas diferentes, las mismas brevemente se explican en el siguiente punteo:

I) **Interfaz o Box**: Se describe el Hardware seleccionado, los distintos componentes de este y su lógica de configuración a la hora de poder brindar un servicio de telefonía. Se habla de FreePBX, Python, Raspberry, IAX2, SIP, Grandstream e invocación de servicios web [5-7].

II) **Frontend:** Se describe cómo se diseñó la aplicación web para poder generar el servicio de autoaprovisionamiento y despliegue automático de las instancias. Se habla de Gunicorn y Flask [8-9].

III) **Backend, Control y Automatización:** En esta sección se explicará cómo trabaja el centro lógico del servicio, desde el despliegue automático de las instancias hasta la reconfiguración de todas de forma segura y ordenada. Se habla de Ansible, Celery y Redis [10-14].

IV) **Cómputo o Plataforma**: Se describe la tecnología de virtualización, mostrando las ventajas de la utilización de contenedores y la integración con el punto 3. Se habla de Docker, Portainer y AWS [15-17].

V) **Integración, Conectividad y Datos**: Se describe como se integrará los diversos módulos, dando hincapié en la seguridad y privacidad de las comunicaciones, como también el registro de la información transversal al servicio. Se habla de HA Proxy, VPN PPTP, Base de datos, IAX2 y Redes superpuestas por medio de Consul [18-22].

VI) **Monitoreo**: Se describe la solución de monitoreo implementada Zabbix, como se logró que la misma trabaje de forma automática dentro del alcance de los contenedores Docker. Se habla de Zabbix [23-25].

3.1. Esquema general de la solución.- Partiendo del diagrama expuesto en la sección anterior se presenta el siguiente esquema (Figura II). Este brinda una visión completa de todos componentes del sistema, y su interconexión:

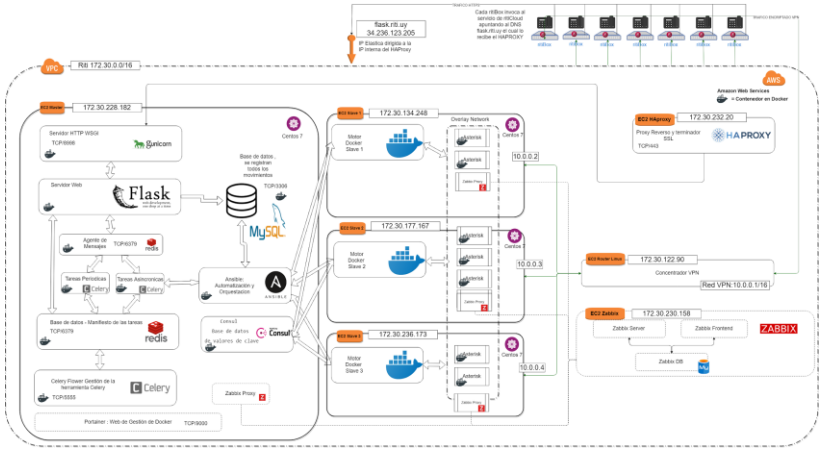

Figura II.- Esquema general de la solución

El objetivo de este esquema es dar un panorama general de la gran cantidad de componentes del sistema, lograr tener una visión de la proporción de los componentes que son desplegados en la nube, y los que son desplegados en la dependencia física del cliente.

3.2. Modelos de instancias.-

- Instancia Master: Es un host docker con contenedores para provisionar el FrontEnd, Backend y BBDD.
- Instancia Slave: Es un host docker que aloja los contenedores Asterisk correspondientes a los ritiBox.

Dichos modelos se pueden observar en la Figura III.

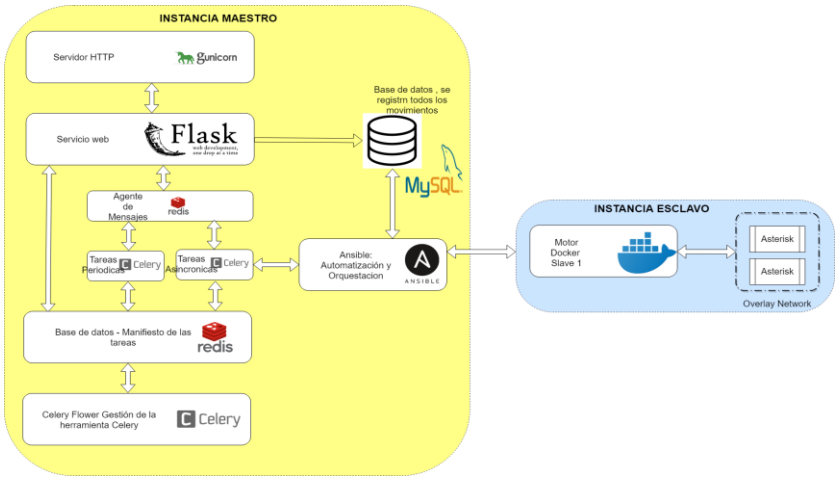

Figura III.- Modelos de instancias

Para lograr que todas las instancias "Esclavo" se puedan comunicar entre ellas utilizamos el protocolo VXLAN implementado utilizando la tecnología Consul, como se puede observar en el diagrama siguiente (Figura IV):

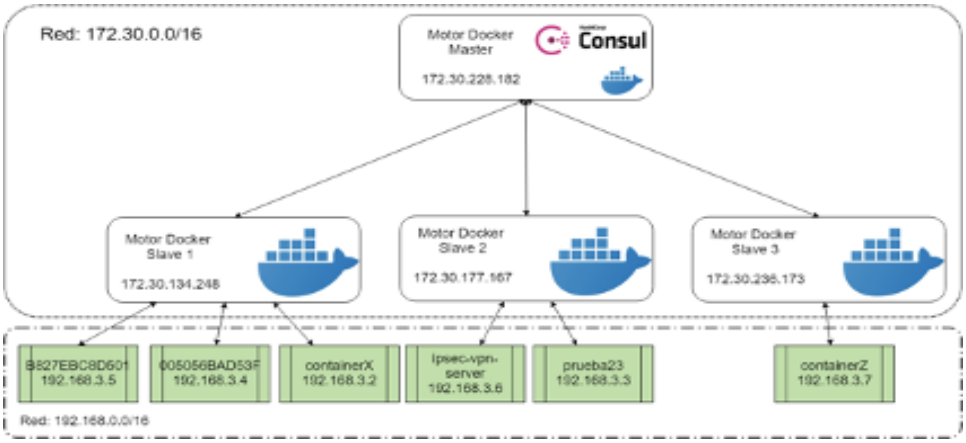

Figura IV.- Protocolo VXLAN mediante la utilización de la tecnología Consul

Se crea una red mesh que todas las instancias de micro-PBX pueden hablar una contra otra.

3.3. VOIP RitiCloud: Asterisk.- Para el despliegue de los RitiCloud en dockers, se implementó Asterisk puro, sin capas extra (Figura V). Esto es principalmente debido a la necesidad de reducir tamaño de imágenes docker y además por la necesidad de ganar control sobre los flujos VoIP.

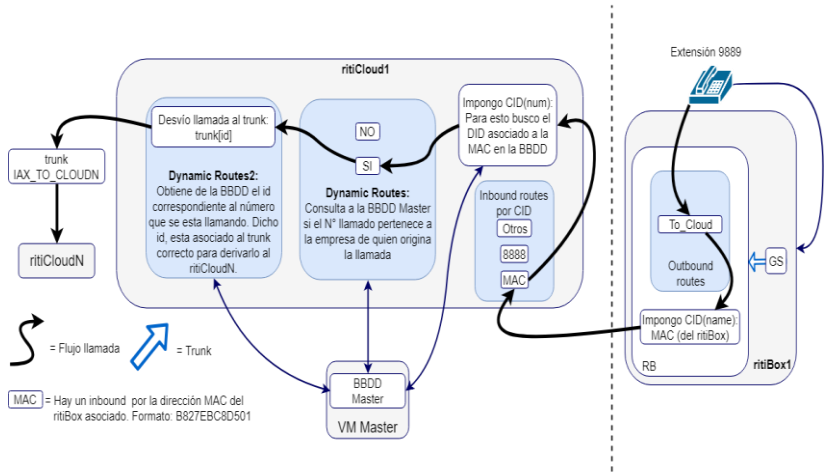

Figura V.- VoIP: Ejemplo flujo de llamadas en misma empresa

En la siguiente imagen (Figura VI), se puede el flujo a alto nivel simplificado, que muestra claramente el recorrido de una llamada entre dos clientes de la misma empresa. Recorriendo tanto los ritiBox como ritiCloud, para llegar desde el aparato telefónico del cliente que origina la llamada, hacia el aparato telefónico del destino. Recordando que el ruteo es automático y el tráfico es seguro.

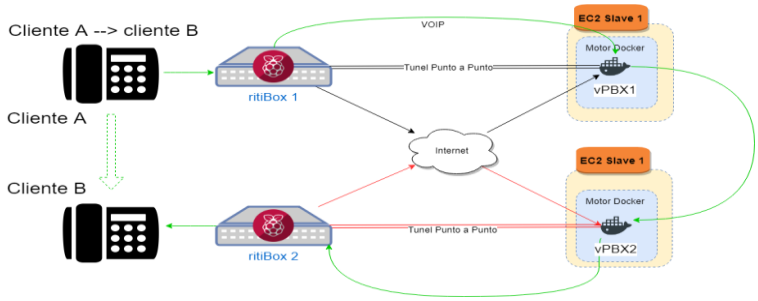

Figura VI.- Flujo de alto nivel de llamada entre cliente A y B

4. Conclusiones.- En primer lugar, se debe decir que estos objetivos fueron cumplidos en su totalidad. RITI fue un proyecto en el cual su principal dificultad fue la integración de más de veinte tecnologías diferentes para lograr un producto funcional con un propósito determinado.

Durante la etapa de investigación y modelado de la idea, realizamos cursos de APIs en Flask, Docker y Docker Swarm para poder entender mejor cada una de estas tecnologías de forma tal que nos facilite la integración de los módulos. Logramos poder brindar una solución que sea escalable pensando las aplicaciones como microservicios que puedan ser desplegados de forma distribuida en distintos hosts de forma automática, controlada y segura. Los microservicios nos permiten aislar las funcionalidades de tal forma que cada instancia (en nuestro caso contenedores Docker en la "nube") cubra una funcionalidad básica y este pueda ser fácilmente reemplazado en caso de falla o upgrade de software. De esta forma podemos hablar de que logramos un flujo de despliegue continuo, ya que se logró automatizar el aprovisionamiento de una central telefónica en la nube manteniendo la independencia del cliente y privacidad del mismo.

Gracias a la arquitectura de la solución modular, a nivel macro, es decir considerando todos los posibles clientes, vemos que nuestro sistema es tolerable a fallas de un componente a la vez de forma tal de garantizar que nuestro servicio no sea afectado en la totalidad. Uno de nuestros objetivos primarios era lograr un hardware lo suficientemente robusto que garantizara la funcionalidad del servicio de telefonía en cualquier condición, incluso ante una pérdida de conectividad o energía eléctrica. Si bien el mismo se puede mejorar en muchos aspectos como tamaño, capacidad ociosa, costo de producción vemos que es un éxito el contar con estas características. Siguiendo en el plano del hardware nuestro mecanismo de automatización consta del registro del número público del cliente, gracias esto y por medio de nuestros algoritmos de armado de rutas telefónicas, podemos garantizar que el cliente siempre disque exactamente el mismo número indistintamente la ruta por la cual se redirigirá el tráfico.

Otro rasgo importante de la solución es la capacidad de un monitoreo automático, la misma se propuso en la solución de automatización gracias a la implementación de unos proxys de monitoreo. Estos proxys de monitoreo trabajando directamente sobre la API de Docker.

RITI una solución híbrida de telefonía Cloud o también denominado Cloud PBX, garantiza ser totalmente independiente de cualquier esquema de “nube”, dejando la misma la posibilidad abierta de desplegarse en una solución tradicional de Datacenter con tecnologías de VMware, HyperV o Citrix como también en cualquier proveedor de cómputo cloud como AWS, Azure o GCP.

Todas las funcionalidades detalladas, fueron evaluadas a través de un proceso de pruebas realizadas sobre el sistema, demostrando que es capaz de funcionar correctamente.

5. Referencias.-

[1] Amazon, "YeaStar S20 S20-000 VoIP SIP IP PBX 20 Ext IVR VM Skype 0 FXS 0 FXO 0 GSM," [Online], Available: https://www.amazon.com/YeaStar-S20-S20-000-VoIP-Skype/dp/B01K8W8JIY [Accessed Jul. 7, 2018].

[2] Ipphone-WareHouse, "Yeastar S0 1-FXS/1-FXO module" [Online], Available: http://www.ipphone-warehouse.com/yeastar-so-module-p/yst-so.htm [Accessed jul. 7, 2018]

[3] Yeaster, "Yeastar S20 VoIP PBX". [Online], Available: https://www.yeastar.com/s20-voip-pbx/ [Accessed: Jul. 7, 2018]

[4] EICSS, "FXO/FXS Add-on for Raspberry Pi/Orange Pi". [Online], Available: http://www.eicss.com/Home/iris2000 [Accessed Jul. 7, 2018]

[5] M. Grech, "The Top 10 Best Free Open Source PBX Software". 23 de Setiembre de 2016. [Online], Available: https://getvoip.com/blog/2016/09/23/best-open-source-pbx-software/ [Accessed Jul. 29, 2018]

[6] C. Santana R. and Codejobs, "¿Qué es Python?" Estados Unidos. Marzo 2013. [Online], Available: https://www.codejobs.biz/es/blog/2013/03/02/que-es-pyt [Accessed Jul. 7, 2018]

[7] Python, "History and License". [Online], Available: https://docs.python.org/3/license.html [Accessed Jul. 7, 2018]

[8] Flask, "Welcome to Flask". [Online], Available: http://flask.pocoo.org/docs/1.0/ [Accessed Jul. 7, 2018]

[9] Gunicorn, "Running Gunicorn". [Online], Available: http://docs.gunicorn.org/en/latest/run.html. [Accessed: Jul. 7, 2018]

[10] CeleryProject, "Introduction to Celery". [Online], Available: http://docs.celeryproject.org/en/latest/getting-started/introduction.html#id2 [Accessed Jul. 8, 2018]

[11] Pocoo.org, "Celery Based Background Tasks". [Online], Available: http://flask.pocoo.org/docs/0.12/patterns/celery/ [Accessed Jul. 8, 2018]

[12] RedisLabs, "Python Redis". [Online], Available: https://redislabs.com/lp/python-redis/ [Accessed Jul. 8, 2018]

[13] Redis, "Redis". [Online], Available: https://redis.io/ [Accessed Jul. 8, 2018]

[14] Ansible, "Ansible Documentation". [Online], Available: https://docs.ansible.com/ansible/latest/index.html [Accessed Jul. 8, 2018]

[15] V. C. Todea and Universitat Politécnica de València, Escola Técnica Superior d'Enginyeria Informática, "Diseño e implementación de un sistema de entrega continua para aplicaciones web sobre contenedores Docker.". 27 de diciembre de 2016. [Online], Available: https://riunet.upv.es/bitstream/handle/10251/71386/TODEA%20-%20Dise%C3%B1o%20e%20implementaci%C3%B3n%20de%20un%20sistema%20de%20entrega%20continua%20para%20aplicaciones%20web%20sobre%20con....pdf?sequence=2 [Accessed Jul. 8, 2018]

[16] Ansible, "Getting Started with Docker". [Online], Available: https://docs.ansible.com/ansible/latest/scenario_guides/guide_docker.html [Accessed Jul. 8, 2018].

[17] Amazon AWS, "Amazon Elastic Compute Cloud: Guía del usuario de instancias de Linux", Instancias de Amazon EC2.

[18] HA Proxy, "Documentation". [Online], Available: http://www.haproxy.org/#docs [Accessed Jul. 29, 2018]

[19] MariaDB, "Documentation" [Online], Available: https://mariadb.com/kb/en/library/documentation/ [Accessed Jul. 29, 2018].

[20] Elastixtech, "Protocolo IAX" [Online], Available: http://elastixtech.com/protocolo-iax/ [Accessed Jul. 29, 2018]

[21] IETF, "IAX: Inter-Asterisk eXchange Version 2" Febrero 2010 [Online], Available: https://tools.ietf.org/html/rfc5456 [Accessed Jul. 29, 2018]

[22] IETF, "Point-to-Point Tunneling Protocol (PPTP)" Julio 1999 [Online], Available: https://tools.ietf.org/html/rfc2637 [Accessed Jul. 29, 2018]

[23] A. Vladishev, "Zabbix - Definition, benefits and challenges" Diciembre 2016 [Online], Available: https://lata.org.lv/wp- content/uploads/2016/12/AlexeiVladishev_OSS.pdf

[24] Zabbix, "Customers and Users" [Online], Available: https://www.zabbix.com/users [Accessed Jul. 29, 2018]