Towards a Robust FreeBSD-Based Cloud: Porting OpenStack Components

Chih-Hsin Chang
SUSE
Taipei, Taiwan
zespre.chang@suse.com

Li-Wen Hsu
FreeBSD Foundation
Taipei, Taiwan
lwhsu@FreeBSD.org

***Abstract*—This paper presents a pioneering initiative to integrate OpenStack, an open-source cloud computing platform, with FreeBSD, a robust Unix-like operating system. Traditionally, OpenStack has been closely associated with Linux-based environments, leveraging specific Linux features and technologies. This integration aims to expand OpenStack's applicability by harnessing FreeBSD's advanced networking, security, and efficient resource management capabilities.**

The project scope involves adapting OpenStack's key components to function seamlessly within FreeBSD's system architecture, focusing on virtualization with bhyve and FreeBSD's unique networking stack. A Proof of Concept (PoC) has been successfully developed, demonstrating the viability of this integration and laying a foundation for further development.

We address several critical challenges in this integration process, including adapting libvirt for bhyve, managing VLANs, modifying Open vSwitch for FreeBSD, ensuring efficient DHCP services, and aligning FreeBSD's network namespace and firewall functionalities with OpenStack's requirements. The project also tackles the adaptation of OpenStack's oslo.privsep library to FreeBSD's privilege model and addresses the complexities of nested virtualization and VM console access within FreeBSD.

Future work involves expanding the integration to more OpenStack components, enhancing system performance, and fostering collaboration within the FreeBSD and OpenStack communities. This integration represents a significant advancement in cloud computing, offering a versatile platform that combines the strengths of both OpenStack and FreeBSD. The project invites collaboration and contribution from the community to overcome the challenges and fully realize the potential of this innovative integration.

***Index Terms*—FreeBSD, cloud, OpenStack, libvirt, bhyve, Open vSwitch**

I. Background

A. OpenStack

OpenStack, a prominent open-source cloud computing platform, has revolutionized the way organizations manage and deploy cloud resources. Since its inception, OpenStack has gained traction for its flexibility, scalability, and robust feature set, allowing it to cater to a diverse range of cloud computing needs. At its core, OpenStack provides a comprehensive suite of services that handle compute, storage, and networking resources in cloud environments, making it a popular choice for deploying public and private clouds alike.

- **Keystone** is the identity service used by OpenStack for authentication and high-level authorization. It provides a central directory of users mapped to the OpenStack services they can access. Keystone supports multiple forms of authentication including standard username and password credentials, token-based systems, and AWS-style logins. It also integrates with existing backend services such as LDAP.
- **Glance** is the image service for OpenStack, allowing users to discover, register, and retrieve virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. It can store images in a variety of backends, including Swift, the OpenStack object storage service.
- The **Placement** service in OpenStack is responsible for tracking resource inventories and usages. It helps in efficiently scheduling and placing VM instances across the available hardware resources based on defined criteria like CPU, memory, and disk space. This service is essential for optimizing resource allocation and ensuring effective utilization of the infrastructure.
- **Cinder** is the block storage service for OpenStack. It provides persistent block-level storage devices for use with OpenStack compute instances. The service manages the creation, attachment, and detachment of block devices to servers and works with a variety of storage backends, including NAS and SAN systems, as well as local Linux server storage.
- **Swift** is the object storage service in OpenStack. It's designed to store and retrieve unstructured data, such as documents, images, and videos, at a large scale. Swift ensures data replication and distribution across various storage servers, offering fault tolerance, high availability, and scalability.
- **Neutron**, often referred to as the network component of OpenStack, plays a crucial role in managing the network infrastructure in cloud environments. It enables users to define networks and attach virtual interfaces to them. Neutron provides an API that lets you define and request network resources, such as networks, subnets, and routers. It's designed to be flexible and pluggable to support a variety of networking technologies and vendors, ranging from simple Linux networking to complex, high-performance networks using SDN (Software-Defined Networking) and NFV (Network Functions Virtualization). Neutron also enables advanced network ser-

vices like LBaaS (Load Balancer as a Service), FWaaS (Firewall as a Service), and VPNaaS (VPN as a Service). Its ability to provide isolated networks to tenants in a multi-tenant environment makes it a key component for cloud security and organization.

- **Nova**, the compute service of OpenStack, is essentially the backbone of the OpenStack cloud platform, managing and automating pools of compute resources. Nova interacts with other OpenStack services like Keystone for authentication, Glance for images, and Neutron for networking. It schedules and runs virtual machine instances based on user requests and available resources, which includes handling the lifecycle of compute instances (create, schedule, run, and terminate). Nova is designed to scale horizontally on standard hardware, and it supports various types of hypervisors including KVM, Xen, VMware, and Hyper-V, making it a versatile solution for a wide range of infrastructure needs. Nova's architecture is highly modular, with a central API that interacts with several other components like the scheduler, the compute workers, and various agents. This modularity allows it to be adaptable and flexible to different deployment scenarios.
- **Ironic** is an OpenStack project that provides a service for provisioning bare metal machines instead of virtual machines, functioning as a bare metal hypervisor API. It blends the capabilities of traditional compute provisioning software with the agility and the user-driven provisioning model of a cloud environment.
- **Horizon** is the dashboard and the web-based user interface for OpenStack services. It allows cloud administrators and users to manage various OpenStack resources and services. Horizon provides a modular web interface for accessing the different OpenStack services, including Nova, Swift, and Keystone.

Traditionally, OpenStack has been primarily associated with Linux-based environments. Its components are predominantly designed and optimized for Linux, leveraging various Linux-specific features and technologies. This strong affinity with Linux has led to widespread adoption in Linux-dominated environments, but it also poses certain limitations, particularly in terms of cross-platform compatibility. Recognizing this gap, there's an emerging interest in exploring the feasibility of integrating OpenStack with other operating systems, such as FreeBSD, to expand its reach and capabilities.

II. Introduction

A. Project Scope

1) Selective Porting of OpenStack Components: The project's scope is strategically focused on porting only the essential components under the vast OpenStack umbrella to FreeBSD. This targeted approach allows for efficient allocation of resources and expertise, ensuring that the most critical and foundational elements of OpenStack are adapted first for FreeBSD compatibility. The selection of components for porting is guided by their fundamental roles in cloud infrastructure and their relevance to the project's overarching goals.

2) Documentation for Reproducibility and Promotion: A key aspect of the project is to meticulously document the build, code-patching, and installation steps of OpenStack components on FreeBSD. This documentation serves a dual purpose: it not only ensures the reproducibility of the setup for future deployments and development but also acts as a valuable tool for promoting the project. By providing clear, step-by-step guides, the project aims to attract interest and encourage participation from the community, making it easier for new contributors and users to get started with OpenStack on FreeBSD.

3) Deployment Replication in FreeBSD.org Cluster: In line with the principle of "dogfooding," the project involves replicating the OpenStack deployment in the FreeBSD.org cluster by closely following the produced documentation. This replication serves as a real-world test of the documentation's accuracy and the deployment process's effectiveness. It also helps in identifying any gaps or areas for improvement, ensuring that the final documentation is robust and reliable for anyone looking to implement a similar setup.

4) Creation of FreeBSD Ports: A significant objective of the project is to convert the ported OpenStack components and their dependencies into FreeBSD ports. This conversion is essential for streamlining the installation process, making it more accessible and user-friendly. It includes not only the OpenStack components themselves but also any dependencies that are currently missing from the FreeBSD Ports collection. By creating these ports, the project aims to enhance the overall ease of setting up and managing OpenStack on FreeBSD, contributing to the broader adoption and success of the platform in various computing environments.

B. Methodology

Our approach to achieving this integration involves a multi-faceted strategy. Firstly, we focus on porting and adapting OpenStack components to be compatible with FreeBSD's system architecture. This includes modifying existing OpenStack modules to work with FreeBSD's kernel and user-space utilities, as well as developing new tools and drivers where necessary.

Secondly, we address the challenge of networking and virtualization, which are integral to OpenStack's operation. This involves adapting FreeBSD's networking stack and virtualization technologies, such as bhyve, to work with OpenStack's network and compute services.

Lastly, our methodology includes extensive testing and validation to ensure that the integrated OpenStack-FreeBSD system is stable, efficient, and secure.

III. Project Status

A. Development Environment

The project's development is conducted on a virtualized platform, powered by a physical infrastructure running

FreeBSD 13.2-RELEASE. This robust hardware setup underpinning the virtual machines comprises:

- Processors: 2 x Intel® Xeon® E5-2680 v4
- Motherboard: Supermicro® X10DRL-i
- Memory: 64 GB RAM
- Storage: 1 TB SSD

The virtualized development environment was later upgraded to FreeBSD 14.0-STABLE, ensuring alignment with the latest stable FreeBSD release.

B. OpenStack Xena Integration

Throughout the development, the team has been working with OpenStack's Xena release. This version of OpenStack was chosen for its specific features and stability, making it suitable for the project's goals and objectives.

C. Porting OpenStack Components

A core part of the project involved porting selected OpenStack components to FreeBSD, each requiring a certain degree of source code modification. The components, along with their dependencies, were installed from source repositories using pip. The build and installation processes, along with necessary modifications, are comprehensively documented in the project's GitHub docs repository. The status of the ported components is as follows:

- Keystone: Operational with minimal source code modifications.
- Glance: Functional with minimal changes to the source code.
- Placement: Successfully running with minor code adjustments.
- Neutron: Configured with a flat network, backed by Open vSwitch.
- Nova: Integrated with bhyve as the hypervisor and libvirt for API and hypervisor management.

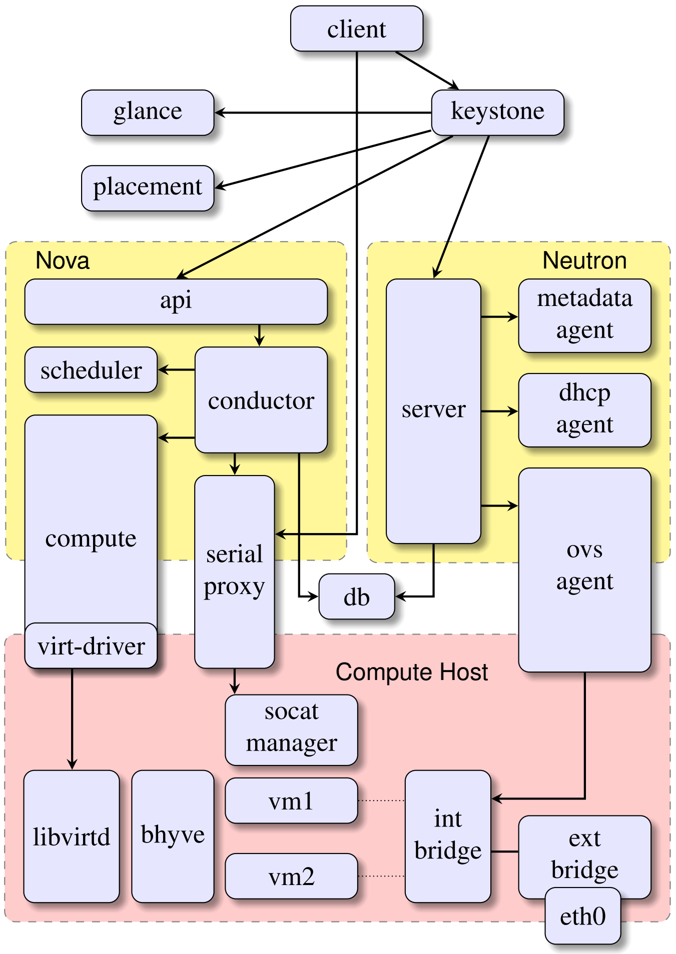

Fig. 1. The PoC of OpenStack on FreeBSD

D. Instance Creation and Network Connectivity

A significant achievement in this development phase is the creation of instances within the OpenStack environment based on the Xena release. These VM instances have demonstrated network connectivity, proving the functionality and effectiveness of the Proof of Concept (PoC). This successful network connectivity underlines the effective integration of the flat network with Open vSwitch and the adaptation of OpenStack components to FreeBSD. The ability to create operational instances that communicate within the network validates the project's success in adapting OpenStack Xena to the FreeBSD platform.

IV. Challenges, Workarounds, and Proposed Solutions

A. Computing

The integration of OpenStack with FreeBSD encountered a notable challenge in leveraging FreeBSD's bhyve hypervisor with OpenStack's Nova component, which traditionally relies on the libvirt library for managing virtual machines. Libvirt acts as an abstraction layer supporting various hypervisors, but its support for bhyve in FreeBSD was not as comprehensive or mature as for Linux-centric hypervisors. This discrepancy posed significant difficulties in achieving seamless integration and full functionality. The challenge was to enable Nova, which extensively uses libvirt, to effectively manage virtual machines via FreeBSD's bhyve hypervisor, ensuring compatibility and performance on par with Linux-based hypervisors.

A potential workaround involved enhancing the existing libvirt driver for bhyve or developing a new custom driver specifically tailored for OpenStack's use case. This approach would require modifying the Nova codebase to better communicate with the bhyve hypervisor through libvirt, focusing on ensuring that essential features and operations like VM creation, management, and networking are fully supported and perform efficiently. This solution might also involve contributing to the development of libvirt's support for bhyve, improving its capabilities to match the requirements of OpenStack.

The ideal solution would be a fully-fledged, robust integration of bhyve with libvirt, where libvirt's support for bhyve is as comprehensive and efficient as it is for other major hypervisors. This would involve upstream contributions to both libvirt and OpenStack's Nova, ensuring that the libvirt-bhyve driver comprehensively covers all virtualization features and APIs required by Nova. This solution would allow OpenStack on FreeBSD to leverage bhyve's strengths, such as its lightweight architecture and performance benefits, while maintaining compatibility with OpenStack's broader ecosystem. Achieving this would require close collaboration between the FreeBSD, libvirt, and OpenStack communities to ensure alignment of goals, technical approaches, and thorough testing to validate functionality and performance.

B. Networking

1) Overlay Networks: Integrating OpenStack's Neutron service with FreeBSD presented a challenge in porting advanced networking features like VLAN, VXLAN, and GRE, which are predominantly designed for Linux-based environments. This issue arose due to the incompatibility of these features with FreeBSD's networking stack, creating a significant hurdle in achieving similar functionality for complex network setups in a FreeBSD-based OpenStack deployment.

The project team opted to implement a "flat" network configuration in the Neutron ML2 (Modular Layer 2) plugin to address this challenge within the project's constraints. This approach simplified the networking model by avoiding the need for complex porting of Linux-specific network provisioning tools for overlay technologies. However, this workaround introduced a notable limitation:

Lack of Tenant Network Isolation: The flat network configuration does not provide tenant network isolation, a crucial feature in multi-tenant cloud environments for security and privacy. With all tenants sharing the same network space in a flat network setup, the potential for security vulnerabilities and conflicts between different tenant networks increases. This lack of isolation is a significant drawback, compromising one of the essential aspects of cloud networking - the ability to segregate and secure tenant-specific network traffic.

The ideal solution would enable OpenStack on FreeBSD to support advanced networking features, including overlay networks that offer tenant isolation. This could involve adapting existing FreeBSD networking tools to support such technologies or developing new tools and drivers that are compatible with FreeBSD's architecture. Achieving this would allow FreeBSD-based OpenStack deployments to provide network functionality and security comparable to Linux-based environments, ensuring effective isolation of tenant networks. Realizing this ideal solution would require collaborative efforts between the OpenStack and FreeBSD communities, alongside comprehensive development and testing, to ensure robust and secure network operations.

2) Underlay Network: Integrating OpenStack's Neutron service with FreeBSD encountered a significant challenge when it came to the mechanism driver. The Linux Bridge, a mainstream driver in the Linux world, couldn't be directly applied to FreeBSD due to fundamental differences in their respective kernel architectures and networking implementations.

Given this incompatibility, the workaround involved opting for an alternative mechanism driver that could function within FreeBSD's networking environment. Open vSwitch was chosen as a more feasible option, as it offered greater compatibility with FreeBSD compared to the Linux Bridge. However, this choice was not without its own set of limitations, particularly in terms of not fully utilizing FreeBSD's native networking capabilities.

The ideal solution would involve developing a FreeBSD-specific mechanism driver for OpenStack Neutron. This driver would be tailored to align with FreeBSD's networking architecture, effectively managing the network underlay in a way that leverages the operating system's inherent strengths. The development of such a driver would require a deep understanding of FreeBSD's networking stack and a commitment to optimizing performance and compatibility within the OpenStack environment. This solution would ensure that OpenStack on FreeBSD could deliver robust and efficient networking capabilities, akin to what is achieved with the Linux Bridge on Linux-based systems.

3) IP Address Issuance: Integrating OpenStack Neutron's DHCP services with FreeBSD faced a specific challenge due to the absence of Linux network namespaces and `veth(4)` pair devices. In Linux-based OpenStack deployments, Neutron typically uses network namespaces to isolate DHCP servers (like dnsmasq) and `veth(4)` devices to connect these isolated DHCP servers to the network bridges. This setup segregates the DHCP service scope efficiently. However, FreeBSD does not natively support these Linux-specific features, making it challenging to replicate this network isolation and connection mechanism for DHCP services.

To work around these limitations, the project team disabled the internal DHCP function within OpenStack and resorted to using an external DHCP server to assign IP addresses to VMs. While this provided a temporary solution, it introduced a significant operational drawback:

Discrepancy in IP Address Management: The IP addresses designated by Neutron differed from those actually assigned by the external DHCP server. This inconsistency resulted in a breakdown of network connectivity, as the flow rules on the Open vSwitch were configured to recognize and handle IP addresses based on Neutron's allocation. The mismatch led to the dropping of outgoing packets from VMs at the Open vSwitch, as these packets carried IP addresses not recognized by the pre-defined flow rules.

The ideal solution for the DHCP challenge in FreeBSD-based OpenStack involves adapting Neutron to utilize FreeBSD's `vnet(9)` and `epair(4)` features, which are equivalents to Linux's network namespaces and `veth(4)` devices. This requires targeted modifications to Neutron's codebase to ensure compatibility with FreeBSD's network architecture. By aligning Neutron with FreeBSD's existing capabilities, the solution aims to restore efficient DHCP functionality and network connectivity in OpenStack, leveraging FreeBSD's strengths in network isolation and interface management.

4) Security Groups: The challenge in implementing OpenStack's security group functionality on FreeBSD stemmed from two primary issues. First, the lack of iptables in FreeBSD, as OpenStack Neutron heavily relies on iptables for its security group function in Linux environments. Second, the requirement of an additional Linux bridge to integrate iptables into the network stack set up by Neutron - a component not available in FreeBSD. These factors combined made it challenging to directly use OpenStack's security group functionality in a FreeBSD-based setup.

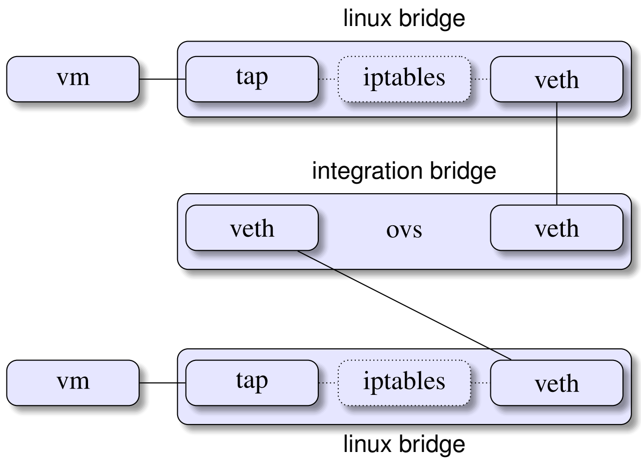

Fig. 2. The security group implementation (iptables)

To navigate around the absence of iptables and Linux bridges in FreeBSD, the project team configured Neutron to utilize the Open vSwitch firewall driver. This approach effectively bypassed the need for iptables and additional Linux bridges. Open vSwitch, with its firewall driver, facilitates traffic filtering by installing flow rules directly onto the switch. These flow rules are designed to emulate the security group functionalities, providing a means to manage and secure network traffic within the OpenStack environment. This workaround leverages Open vSwitch's capabilities to achieve similar objectives as security groups in a way that aligns with FreeBSD's network architecture, sidestepping the need to integrate ipfw or pf into Neutron.

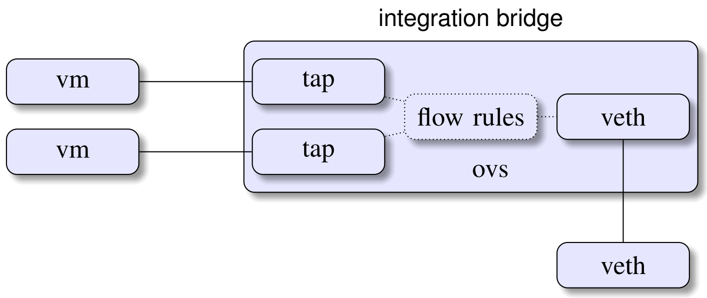

Fig. 3. The security group implementation (Open vSwitch flow rules)

The optimal solution for OpenStack's security group functionality in FreeBSD is the development of a native pf (Packet Filter) firewall driver within Neutron. This integration would enable robust and efficient security group management, leveraging FreeBSD's advanced firewall capabilities. The implementation would require adapting Neutron to work seamlessly with pf, ensuring effective translation and enforcement of security rules. This approach aims to bring FreeBSD-based OpenStack deployments to parity with Linux-based systems in terms of network security and efficiency.

C. Privilege Model

In integrating OpenStack with FreeBSD, a significant challenge arose in the realm of privilege separation. OpenStack primarily targets Linux environments and thus relies on Linux-specific capabilities for managing different privilege levels within its components. This mechanism is crucial for maintaining security and efficiency, particularly in components like Nova and Neutron. FreeBSD, however, employs a distinct capability and security model, which created a compatibility issue. Adapting OpenStack's privilege separation, which heavily utilizes the Linux-centric oslo.privsep library, to FreeBSD's different security architecture was a complex task.

As an immediate workaround, the project considered utilizing the oslo.rootwrap utility, a component of the oslo.privsep library. This utility allows certain operations to run as root, but with strict command filters to maintain security. However, this approach was more of a stop-gap measure. It provided a way to run necessary privileged operations but lacked the finer control and security assurances of a true privilege separation mechanism tailored for FreeBSD.

The ideal solution would involve developing or adapting a privilege separation mechanism that aligns with FreeBSD's security model while meeting OpenStack's operational requirements. This could mean creating a FreeBSD-specific version of the oslo.privsep library or a similar framework that can handle privilege elevation and reduction in a manner compatible with

FreeBSD's capabilities. Such a solution would ensure that OpenStack components on FreeBSD maintain high security standards, offer precise control over privileges, and integrate seamlessly with FreeBSD's inherent security features. This approach would require a thorough understanding of both OpenStack's security needs and FreeBSD's capability system, alongside extensive testing to ensure robustness and reliability.

D. Miscellaneous

1) Nested Virtualization: The nested virtualization challenge in the OpenStack FreeBSD integration project was encountered during the development phase. The development environment was set up on a virtualized layer, with OpenStack components running on a series of Linux KVM-based VMs, referred to as host VMs. The challenge surfaced when porting the nova-compute service to support bhyve, FreeBSD's hypervisor. Attempts to boot a bhyve-based guest VM within these host VMs led to failures, where not only did the guest VM fail to start, but it also caused the host VM to hang. This issue was a significant obstacle, as nested virtualization—running a hypervisor inside a VM—is crucial for development and testing purposes, especially when resources are limited.

Upon investigation, the root cause of the problem was identified as a hardware-related issue, specifically linked to the APIC emulation of the VT-x features: `VID` and `PostIntr`. This discovery was corroborated by information found online from others who had encountered similar issues [6]. The resolution involved turning off these two CPU features (`VID` and `PostIntr`) for the host VMs. Once these features were disabled, the issue with the bhyve-based guest VMs no longer occurred, allowing them to boot up successfully without causing the host VMs to hang. This fix was crucial in enabling the continuation of the development and testing of nova-compute and bhyve integration in a nested virtualization environment. Additionally, this resolution made it easier for others interested in the project to get started and replicate the development environment. The use of VMs, being more lightweight and accessible than bare-metal servers, greatly facilitated broader participation and experimentation in the project, enhancing the collaborative development process.

2) VM Consoles: The VM console challenge in the FreeBSD-based OpenStack environment was multifaceted, originating from the workaround implemented for IP address issuance. When the internal DHCP functionality in Neutron was disabled in favor of an external DHCP server, VMs began receiving IP addresses different from those allocated by Neutron. This divergence caused a critical network connectivity issue: the outgoing traffic from VMs didn't match the flow rules set on Open vSwitch, leading to packet drops.

Addressing this issue required a method to manually update the IP addresses within the VMs to those allocated by Neutron. However, due to the connectivity problems, traditional remote management tools like SSH were not an option. The solution lay in accessing the VMs' consoles, but this presented its own set of challenges.

bhyve does support exposing VNC consoles for VM access, but with a caveat: the VMs need to boot in UEFI mode, as the graphics depend on the UEFI framebuffer. Implementing UEFI booting with libvirt and bhyve is a complicated process and was not a viable option within the project's timeline, even though it is a planned future enhancement. This led to exploring alternative methods of console access.

The project turned to nova-serialproxy, a component that allows proxying of VM's serial consoles. By employing `nmdm(4)` (nullmodem terminal driver) with each VM during instance creation, the VMs' serial consoles could be accessed using the `cu(1)` command. However, this approach had its limitations, as the `nmdm(4)` device is only available on the compute host where the VM is running, necessitating direct access to these hosts. This requirement posed practical and security challenges, as it complicated the process for users who needed to access VM consoles remotely.

As a result, we developed a simple proxy: socat-manager [4]. This proxy tool effectively exposes the `nmdm(4)` devices over network sockets, enabling nova-serialproxy to remotely access them. As a result, users can now connect to the VM's serial console from anywhere via the Nova API with a valid authentication token. This enhancement not only overcomes the limitation of requiring access to the compute host but also enables users to modify the VM's IP address directly through the console. This capability is crucial for resolving the network connectivity issues caused by the mismatch between the IP addresses designated by Neutron and those assigned by the external DHCP server. The ability to update the VMs' IP configurations remotely restores their network functionality, significantly improving the operability and flexibility of the FreeBSD-based OpenStack environment.

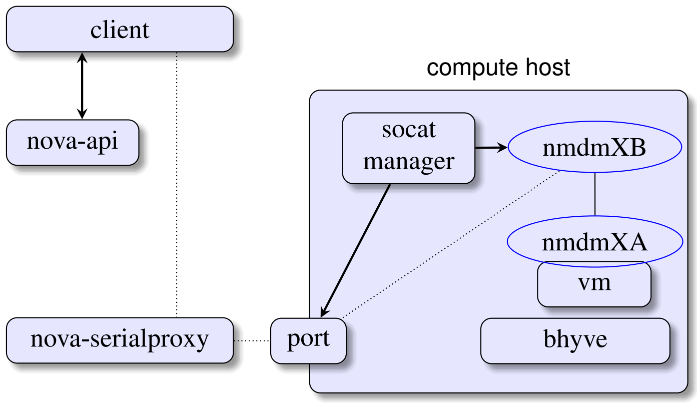

Fig. 4. Exposing VM console with socat-manager

While developing socat-manager as part of the workaround for VM console access in FreeBSD-based OpenStack, the team encountered a notable issue related to libvirt's event hooks for bhyve. Specifically, the implementation of these hooks was incomplete, which meant that essential VM information, such as the names of `nmdm(4)` devices and designated serial port numbers, were not readily available through STDIN. This limitation required the team to parse the domain XML file to extract this necessary information, adding complexity to the implementation of socat-manager.

This issue was significant enough to be reported to the libvirt upstream, where it was subsequently addressed and fixed [7]. With this fix in place, there's potential to update and refine the socat-manager implementation to directly utilize the now-available VM information provided by libvirt's event hooks.

In light of this development and looking forward, the ideal solution for VM console access in FreeBSD-based OpenStack encompasses two primary advancements:

- **Handling UEFI Booting with libvirt and bhyve for OpenStack Nova**: Perfecting UEFI booting within the libvirt and bhyve framework for OpenStack Nova. This will enable users to access VM consoles via VNC, providing an integrated and user-friendly graphical console interface.
- **Direct Exposure of VM Serial Consoles in bhyve**: Modifying bhyve to support direct exposure of VM serial consoles via network ports. This will allow the native use of OpenStack's nova-serialproxy for accessing VM serial consoles, thereby eliminating the need for the socat-manager solution.

These developments aim to address the VM console access challenges comprehensively, offering a seamless and secure management experience for VMs in FreeBSD-based OpenStack deployments.

V. Future Works

The future direction of this project is focused on several key areas of development and improvement, aiming to enhance the integration of OpenStack with FreeBSD.

A. Development of Native Drivers for Neutron and Nova

A primary objective is to develop native drivers for Neutron and Nova that are optimized for FreeBSD. This involves creating drivers that can fully leverage FreeBSD's unique features and capabilities, particularly in networking and virtualization. These native drivers will be crucial in improving the performance, stability, and functionality of OpenStack components within FreeBSD environments.

B. Porting Additional OpenStack Components to FreeBSD

The scope of integration will be expanded to include more OpenStack components and services. By porting additional components to FreeBSD, we plan to enhance the overall functionality and robustness of the integrated system. This expansion will not only bring a wider range of cloud capabilities to FreeBSD but also ensure that the platform can cater to a broader array of cloud computing requirements.

C. Migration to New Versions of OpenStack

Keeping up with the evolving OpenStack releases, the project will aim to migrate to newer versions of OpenStack as they become available. This migration is essential for maintaining compatibility with the latest features and improvements in the cloud ecosystem. It will involve updating the existing FreeBSD integrations to work seamlessly with the latest OpenStack releases.

D. Creating Corresponding FreeBSD Ports

A significant part of the future work will involve creating corresponding FreeBSD ports for the newly integrated OpenStack components and their dependencies. This will make the installation and management of these components more accessible and straightforward for FreeBSD users. Creating these ports is a step towards ensuring that the OpenStack integration is not just functional but also user-friendly and widely adoptable.

E. Continuous Engagement and Knowledge Sharing

Continuing from the current efforts, we intend to engage more actively with both the FreeBSD and OpenStack communities. This collaboration will be instrumental in addressing complex challenges and driving innovation in cloud computing. Sharing findings, methodologies, and tools with the broader community is also a key part of this engagement, contributing significantly to the collective knowledge base and assisting others interested in similar integration efforts.

F. Performance and Scalability Improvements

Enhancing the performance and scalability of the system remains a core focus. Future developments will ensure that the FreeBSD-integrated OpenStack system can efficiently handle large-scale cloud deployments, making it a viable option for a wide range of cloud computing applications.

Through these focused efforts, the project aims to solidify FreeBSD's position as a robust and versatile platform for OpenStack deployments, catering to the evolving needs of cloud computing environments.

VI. Conclusion

The integration of OpenStack with FreeBSD represents a significant step forward in expanding the capabilities and applicability of both platforms. While this project presents numerous challenges, the progress made so far is promising. The successful development of a Proof of Concept demonstrates the feasibility of this integration, and the outlined challenges and proposed solutions provide a clear path forward.

By continuing to develop and refine this integration, we aim to create a robust, efficient, and versatile cloud computing platform that leverages the unique strengths of FreeBSD. We call upon the FreeBSD and OpenStack communities to collaborate with us in this endeavor, contributing their expertise and insights to overcome the challenges and realize the full potential of this integration.

Acknowledgment

We express our gratitude to the FreeBSD Foundation for their generous sponsorship and vital support, essential in advancing our project. We also extend our appreciation to the OpenStack community for their valuable contributions and collaborative spirit.

Special thanks to the CHERI team at the University of Cambridge for their constructive feedback and meaningful contributions, significantly enriching our work.

We acknowledge the team at FreeBSD.org for their collaboration and valuable input.

On a personal note, heartfelt thanks are extended to my spouse, Jung-Wei, for their unwavering support and encouragement throughout this endeavor.

Lastly, we are thankful to all individual contributors from the FreeBSD and OpenStack communities for their dedication and expertise.

REFERENCES

[1] "OpenStack Xena Installation Guides," https://docs.openstack.org/xena/install/

[2] "Neutron/ML2 - OpenStack," https://wiki.openstack.org/wiki/Neutron/ML2

[3] "openstack-on-freebsd/docs: Repo for OpenStack on FreeBSD project, including steps, methods, and source codes," https://github.com/openstack-on-freebsd/docs

[4] "openstack-on-freebsd/socat-manager: A proxy for exposing consoles of bhyve VMs to nova-compute," https://github.com/openstack-on-freebsd/socat-manager

[5] "Open vSwitch on Linux, FreeBSD and NetBSD — Open vSwitch 3.2.90 documentation," https://docs.openvswitch.org/en/latest/intro/install/general/

[6] "246168 - Ubuntu 20.04 KVM / QEMU Failure with nested FreeBSD bhyve" https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=246168

[7] "No domain XML passed as stdin for bhyve hooks (#528) · Issues · libvirt / libvirt · GitLab," https://gitlab.com/libvirt/libvirt/-/issues/528