rackspace®

CONTAINERIZING THE CONTROL PLANE:
AN ENHANCEMENT TO OPENSTACK

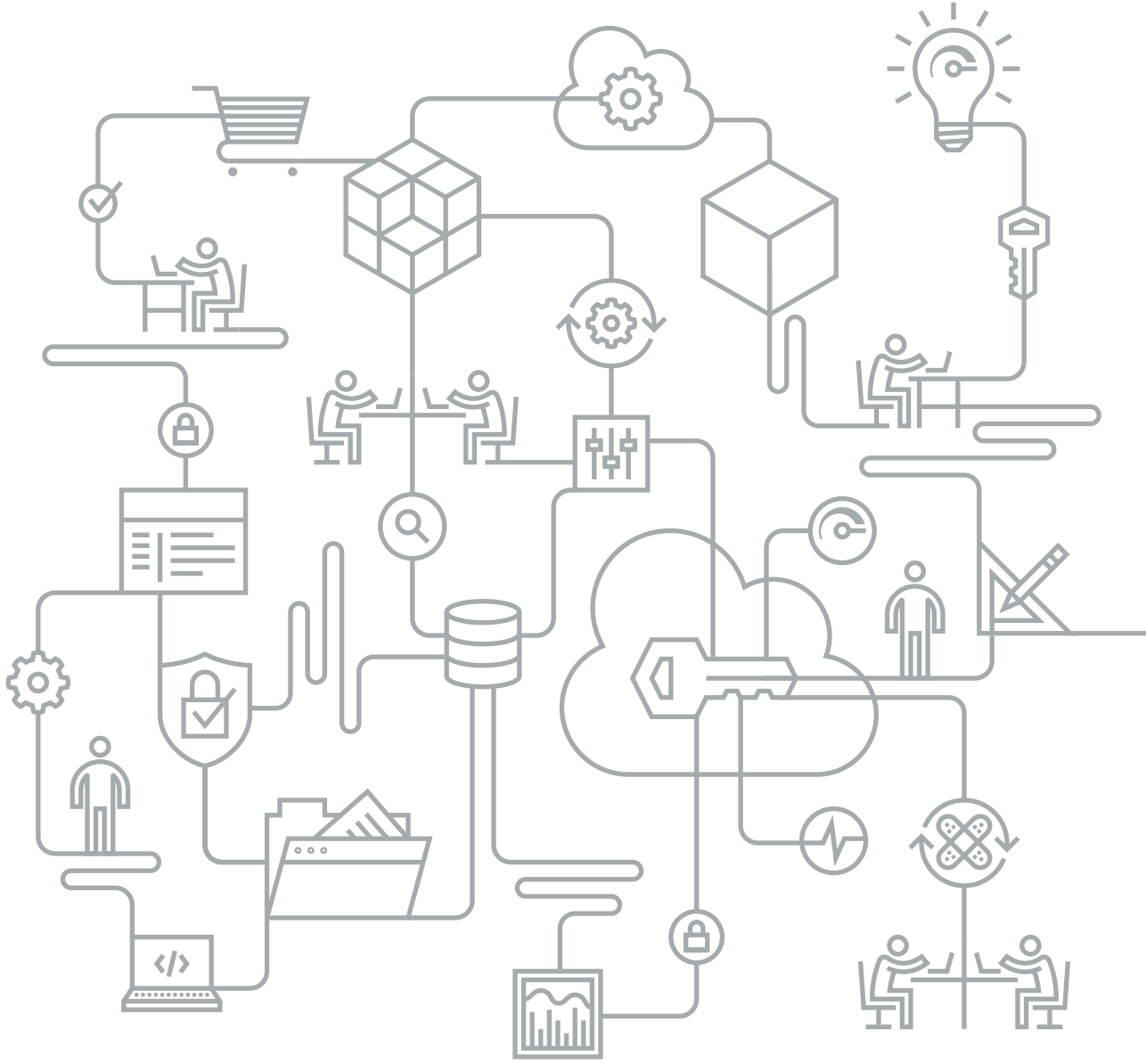

TABLE OF CONTENTS

INTRODUCTION

At Rackspace, we manage OpenStack® private clouds for many of the world's largest and most recognized companies. And as the leading operators of OpenStack, we continually deploy new innovations to improve our customers' experience. In this white paper, we're going to take a look at an alternative approach to the standard method of deploying and updating control plane services.

Most, if not all, distributions of OpenStack install all of the OpenStack control plane services onto the same operating system (this is generally referred to as a hyper-converged controller node). After operating numerous clouds over the years, we came to the realization in 2014 that this hyper-converged controller node design severely limits flexibility when looking at upgrading or scaling individual services. If you were using this approach, you must upgrade all of the services in unison, and the only way to scale is by adding another hyper-converged controller node (increasing each service by a count of 1).

The OpenStack engineering team at Rackspace undertook an effort to solve the glaring flaw inherent to the hyper-converged controller node design. Based on our experience running the world's largest OpenStack cloud, we decided the solution needed to provide the ability to upgrade a single service without having to upgrade all of the services, and it also needed to provide the capability to add capacity to a single service (preferably without requiring additional hardware every time).

EXPLORING THE "CONTAINERIZED" CONTROL PLANE

The solution was to encapsulate the service code, configuration and all of its dependencies in a software container (specifically LXC).

In the hyper-converged controller model, you have to act on the controller node as a whole instead of focusing on the services running on the controller node. If you are going to upgrade the Nova APIs, as an example, you are going to upgrade all of the APIs (and all of their dependencies) running on all of the controller nodes. This behavior is due to the co-location problem. The encapsulation provided by containers grants the ability to treat each service as a distinct entity. This encapsulation also provides the capability to scale each service independently, as well as, allowing us to upgrade a single service without having to touch the other services.

In the hyper-converged controller model every service is co-located on the same operating system. This colocation decreases the operational flexibility of the systems. The lack of flexibility is because each of the services share the same Python and system dependencies.

To illustrate the point, let's take a scenario where there is a bug in the Nova API that has been resolved by upgrading Python-requests from version 2.10.0 to 2.11.0. However, there is also a bug in the Neutron API when utilizing Python-requests version 2.11.0 that is not present when using version 2.10.0. This might seem like a contrived scenario, but this has actually happened (more than a handful of times) over the past few years operating OpenStack clouds for our customers. And this type of scenario presents a no-win situation, where you have to accept living with one of the two APIs being broken for some period of time. The encapsulated service model allows an operator to upgrade the Nova API and its Python-requests dependency, fixing the original bug, without touching the Neutron API service container (leaving it running the older version of the Python-request dependency).

OPERATIONAL BENEFITS

Another benefit of containerizing each OpenStack service is that each of these encapsulated services becomes ephemeral. If a container running the Nova API were to start exhibiting errors, a normal step in the operational handbook would be to log into the service and troubleshoot. This step generally results in a significant amount of time being spent troubleshooting the service, where most of that time is actually spent trying to identify changes.

With the encapsulated service model, this is no longer the first step in the process. The new first step would be to destroy the offending container and redeploy that role in the control plane cluster. By inserting this step, the state of the service is either reset to a known good state (resolving the issue) or engineers can now start troubleshooting from a known good state. This radically decreases the amount of time spent trying to identify 'what changed' instead of looking at 'what is wrong'.

The last operational benefit provided by this model is the ability to upgrade a single service at any time. Most, if not every, OpenStack operator has settled in to power through the dreaded eighteen-hour major version upgrade. Every one of them hopes that they did not just enter a room and see the following text on their screen:

"It is pitch black. You are likely to be eaten by a grue."

Attempting to upgrade the entire IaaS platform in one fell swoop is a daunting task in addition to being an extremely large surface area in which errors can occur. The encapsulated service model provides an alternative to the all-or-nothing approach by spreading the upgrade process over many small upgrade activities. This drastically reduces the chances for errors and the overall impact to the IaaS platform for any given maintenance window.

Again, taking a practical example, let's assume an operator schedules a maintenance window to upgrade the Glance service containers. This maintenance window will only affect the ability to upload a new image, and for a user to provision an instance if the image is not already cached on any of the hypervisors. However, the remainder of the IaaS platform will not be affected, and most importantly none of the running instances will experience any downtime. If the upgrade goes sideways anywhere along the way, the size of the surface area where the root cause of the issue could lie is extremely small. In this case, it is narrowed to the Glance service containers, the Glance database or the Glance message queues. With the scope of the issue narrowed, time to resolution should be greatly shortened and the area of impact to consumers of the IaaS platform is narrowly scoped.

HOW DOES THIS AFFECT SCALABILITY?

In our reference architecture, the controller nodes are running the same hardware specs as a hypervisor. This allows operators to apply the same thought process to the infrastructure as application designers do to their cloud-native applications. The reasoning behind this decision is to:

1. Simplify hardware purchasing decisions and
2. To provide extra capacity for the control plane services from the start.

This extra capacity is available from day one to the operators of the cloud, allowing them to react to their usage patterns and scale out the control plane services accordingly.

Let's say that the Keystone service is getting inundated with requests and that the average response time for those requests has gone beyond an acceptable threshold. The operators of the cloud can simply deploy an additional Keystone service container on each of the controller nodes resulting in double the capacity of the Keystone service overall and reducing the average response time of those authentication requests back to within an acceptable threshold.

(NOTE: This example doubles the capacity without changing the fault tolerance characteristics of the service, the operators of the cloud have the ability to deploy any number of Keystone service containers on any number of controller nodes to fit their specific needs.)

WHY REQUIRE A MINIMUM OF THREE CONTROLLER NODES INSTEAD OF JUST TWO?

In order to provide our industry leading 99.99% API uptime SLA, our reference architecture requires a minimum of 3 controller nodes. This requirement is because of the choice of clustering technology/topology we have selected for our Database and MessageBus layers. Both of these technologies utilize quorum as a way of determining the health of the cluster.

Why does that matter? Quorum provides an automatic way of handling disputes between the nodes participating in a cluster. Quorum is equal to FLOOR ((N / 2) + 1) where N is equal to the total number of nodes participating in the cluster. This means in a three-node cluster, two members must agree on any decision. Requiring an odd number helps prevent the cluster from becoming split brained, which is where the nodes in a cluster disagree on the state of the cluster but are unable to determine if they are out of sync with the cluster. Quorum-based clusters are also able to resolve cluster health issues in an automated way.

In other topologies, the expectation is that a human must intervene during cluster issues and make the call as to how to resolve the issue. In our example, the clustering technologies are able to force a re-sync on a node that is out of whack with the cluster in an automated fashion, not requiring human intervention for all but the most difficult scenarios.

CONCLUSION

Rackspace strives to deliver the best OpenStack experience in the world, with industry-leading reliability, unmatched scalability, innovation-driven agility and a superior approach. As part of that commitment, the control plane services enhancement provides you with a better, safer environment. Your secure OpenStack cloud is delivered in an easy and cost effective way that further protects your users and environments.

To learn more about how to enhance and simplify your OpenStack cloud, sign up for our free OpenStack strategy session at go.rackspace.com/OpenStackExperts.

ABOUT RACKSPACE

Rackspace (NYSE: RAX), the #1 managed cloud company, helps businesses tap the power of cloud computing without the complexity and cost of managing it on their own. Rackspace engineers deliver specialized expertise, easy-to-use tools, and Fanatical Support® for leading technologies developed by AWS, Google, Microsoft, OpenStack, VMware and others. The company serves customers in 120 countries, including more than half of the FORTUNE 100. Rackspace was named a leader in the 2015 Gartner Magic Quadrant for Cloud-Enabled Managed Hosting, and has been honored by Fortune, Forbes, and others as one of the best companies to work for.

Learn more at www.rackspace.com or call us at **1-800-961-2888**.

OCTOBER 20, 2016 | PRI-CWP-Containerized_Control_Plane-4068