Cloud Application Architecture Trends

Authors:
Caleb Groom, Director, Product and Engineering
Ryan Walker, DevOps Automation Engineer
BK Box, Principal Engineer

rackspace®

Table of Contents

rackspace®

Introduction

The flexibility of the cloud and the ability to create—or remove—a resource on demand has fundamentally changed the way that IT thinks about compute resources. As more enterprises power their websites and applications in the cloud, they should be aware of the differences between designing for the cloud and designing for physical servers.

Applying traditional IT architecture models will work on the cloud, however, a business won't benefit from the many advantages inherent in a cloud environment. This paper discusses some of the cloud application trends that an enterprise should consider before writing their next app.

1. Controlling Hardware Through Code

As applications have increasingly exposed their APIs, many developers are familiar with how to pull data in and out of web services. This availability of information has enabled developers to utilize data exposed by popular applications to enhance their own application. Consider, for example, the many mobile apps that take advantage of location data from other services.

APIs can be used as an interface between an app and its underlying hosting infrastructure. Most cloud-hosting providers have exposed an API that can create, read, update or remove a cloud server, enabling the app to modify its configuration environment on the fly. No longer is there a need to cut a purchase order for procuring a server or to wait for the server to be racked, cabled and configured. Servers in the cloud are created on demand.

When incorporating monitoring data—such as the number of connections or outgoing bandwidth—an application can become "self-aware" and aware of its environment, scaling the configuration as traffic exceeds set thresholds. Is there a surge in holiday shopping traffic? An ecommerce app can place multiple cloud servers online to handle the deluge of requests. Has the traffic returned to normal? The app can then tell the infrastructure to remove unnecessary servers, allowing the business to provide the same level of service to consumers while decreasing their costs. This underscores the importance of integrating your application with a cloud infrastructure API.

However, not all cloud APIs are created equal. Some hosting providers build their cloud environment on closed standards. This means those API calls written into an application will only work for that particular provider. Other cloud environments have an underlying open-source technology, such as OpenStack. This enables those API calls to be applied to an array of cloud hosts built on that same open technology.

As businesses examine moving to the cloud, they need to not only consider implementing their hosting provider's cloud API into their app, but also whether their cloud host's API is based on open or closed standards. This maximizes their flexibility moving forward.

2. Implementing Services

Monolithic applications—those systems with a web front end, application and database on the same host server—that once dominated were primarily a function of the smallness of the Internet at the time. Not only were fewer people online, their Internet activity was tethered to the confines of a desk. The amount of data read and written was modest by today's standards, and the file sizes smaller. Consequently, each tier of the application could run on an overprovisioned server configuration without much fear of traffic toppling it over. That is not the case any more.

With the incredible amount of data that is both recorded and served up today, splitting an application into distinct parts—or services—gives the ability to bulk up resources for a targeted portion of the app that is under load.

This notion of modular design first originated with splitting out the web, application and database tiers of an application. However, service-oriented architecture (SOA) has since evolved to become more granular, isolating services with distinct roles.

These different pockets of infrastructure, with unique roles in the overarching application, are then independently scalable. As site traffic increases a company can react by bringing on resources in an intelligent manner, providing an individual service with the needed computational power, ultimately saving on infrastructure costs.

For example, a monolithic ecommerce application could be split into multiple pieces: user authentication, sign up, profile settings, storefront, shopping cart and checkout. Furthermore, each service's web, application and database layer can be split out. This allows for more resources to be applied to the shopping cart database while decreasing the requisite resources for the storefront's web nodes. By isolating these services from each other a business can ensure that the proper amount of compute resources are available for the load on each particular service.

Furthermore, SOA also carries some benefits when it comes to application security. Architecting your app this way gives you better control on who gets to access particular services. In the past, you might have an application that handled web requests from users and, in the same codebase, made a backend call to the database to ensure they were authenticated properly. This can result in potential security exploits—SQL injection as an example—that could lead to your front end making a bad database call and compromising the system.

With SOA, you can have "dumb" front-end code that takes requests from users and passes them to your authentication system via a REST API. The authentication system then does all the actual lookups, providing insulation between the authentication database, the front-end and ultimately the user. Additionally, by requiring the front-end servers to talk to the authentication servers, you are able to keep your authentication API and code hidden from users and are better protected from the Internet at large.

Finally, architecting an application around services allows your operations team to be more targeted in rolling out updates. For example, if the user authentication system needs to be

patched, the team can rollout updates that target that service only. The remaining services continue to operate normally, without interruption.

SOA can extend beyond those services that your development team defines. One of the strengths of the cloud is the ecosystem of services that has evolved. Most likely, your cloud-hosting provider will have an array of products that you can consume as a service. For example, instead of architecting a load balancing solution that routes traffic through your configuration based on sophisticated rules, find out if your provider offers a load-balancing solution as a service. Additionally, they might offer a variety of solutions—including load balancing, database (to store transactional information that the app generates), object storage (a place to store large files on a content delivery network) or block storage (an extension of your cloud server's hard drive)—for you to consume as services.

Furthermore, third-party companies have a variety of tools and services that can assist with anything from load testing, user authentication, DDoS mitigation and more. In order to decrease the time to market, encourage your development team to focus on the core functionality of the app while browsing **cloud marketplaces** for tools created to solve common problems.

3. Container-Based Architecture

The past year has seen major developments in container-based architecture for applications. A container is a self-enclosed layer of virtualization that contains all the required pieces to run an application: resources, frameworks and libraries.

The notion of Linux Containers (LXC) has been in the **Linux kernel since the 2.6.24 release** in 2008. However, **Docker**, an open source project that is a layer built on top of LXC, has popularized the use of containers for cloud architecture. Plainly put, Docker is an API between containers and LXC that enables the creation, deployment and unpackaging of a container. Once unpackaged, a container can connect directly to the computational power on a host machine by taking advantage of the Linux kernel's cgroups ability to isolate resources like processing power, RAM, block I/O and more.

The advantage of architecting an app with containers is that they help simplify the deployment and server configuration process. A business simply has to build the container once, including the required versions of frameworks or services and installing necessary libraries inside that container.

The container can then be shipped to an unlimited amount of host machines, ensuring that the configuration settings are identical. Human error for server, framework and application configuration is significantly mitigated. This results in consistency across your environment with your time investment in compiling the application only happening once.

This is a major improvement with respect to current config-management strategies. When using config-management, you must still install, setup, compile and restart everything on each server. With containers, you simply swap out the old container for the new container.

Another benefit of using containers is that multiple versions of a framework can run on the same host machine. In the non-container world, it can be challenging to run multiple Ruby applications with different Ruby versions or vastly different versions of Ruby Gems

on the same server—you'd have to use Ruby Version Manager (RVM). Indeed, RVM is one of the few options outside of containers that has resolved this difficult problem. However, it is an extremely complicated mechanism for running multiple Ruby versions and can cause issues or complications in production systems.

Containers have solved these problems by allowing a development and operations team to package those apps that require different versions of Ruby and Ruby Gems into distinct containers with Docker. Because all the information needed to run the app is self-contained, the containers can live side by side on the same host machine without any collisions or problems.

Both the development and operations communities have rallied to support Docker and container-based architecture. Originally released as open source in March 2013, the momentum around Docker has exploded, with over 14,400+ stars, 2,600+ forks, 560+ contributors and 10,100+ commits on GitHub at the time of this article. Additionally, Docker has received support and **developed partnerships with many well-respected technology companies** and has been implemented in some of the **most popular applications on the web today**. This level of activity proves that containers are moving from a very futuristic technology to one that will quickly become a mainstream way of architecting applications.

4. Service Discovery

As containers become more prominent, it will be important to ensure that they connect to different services—such as a database—to ensure that the bits and bytes flow in the right manner. This is where service discovery comes into play.

Service discovery essentially allows an application to ask where all the other pieces are located. It helps to identify services such as databases, caching layers or other peers, while preventing the developer from hard-coding IP addresses or domain information inside the application.

This is becoming a trend as many apps are beginning to take service-oriented architecture to the extreme, creating small services that fulfill very targeted functions. Consequently, there are many APIs to interact with those services and it is important for the application to understand where each is located. Additionally, no developer wants to update every container any time the location of a service changes.

SmartStack, a **tool open sourced by Airbnb**, helps an application discover the different services it needs to run. Composed of two different pieces that run on the host machine (Nerve and Synapse), SmartStack understands which services are available to be consumed. SmartStack uses Zookeeper as the record of truth for maintaining, adding and subtracting the information about the services.

Nerve performs a health check on a service as it comes online, and creates a record of the service in Zookeeper, a centralized service for maintaining configuration information. Synapse runs alongside your application, and takes the information from Zookeeper and configures a locally running load-balancer (HAProxy) to show exactly

what services are available and where they are located.

For example, if a server named app1 needs to talk with another called app2, it doesn't need to know the address for app2. App1 simply needs to talk locally to HAProxy, which will route it to the proper destination. If the health or addresses for one of the nodes in app2 changes, Nerve will update Zookeeper and Synapse will update HAProxy to ensure that everything gets routed properly.

This takes all the routing logic out of the application, removing the need for developers to hard code this information. This allows the infrastructure to be completely distributed—as containers are unpackaged they know exactly what services are available and where to find them. SmartStack takes away much of the complexity of managing a distributed application.

Summary: Architecting for the Cloud

While you can architect an application for the cloud in a traditional manner, doing so will fail to unlock the true potential of the platform. Adopting some of these modern application architecture trends can help you achieve the scalability and flexibility of a self-aware application, while limiting the amount of error introduced by manual configuration. This paradigm shift—breaking an application into smaller, modular pieces that each serves a unique purpose—is dominating the landscape. Incorporating any of these architecture trends can help a business better leverage the cloud, however, it is prudent to keep an eye on future IT hosting trends as this space continues to evolve at a rapid pace.

About Rackspace

Rackspace® (NYSE: RAX) is the #1 managed cloud company. Its technical expertise and Fanatical Support® allow companies to tap the power of the cloud without the pain of hiring experts in dozens of complex technologies. Rackspace is also the leader in hybrid cloud, giving each customer the best fit for its unique needs — whether on single- or multi-tenant servers, or a combination of those platforms. Rackspace is the founder of OpenStack®, the open-source operating system for the cloud. Based in San Antonio, Rackspace serves more than 200,000 business customers from data centers on four continents.

GLOBAL OFFICES

Headquarters Rackspace, Inc.
5000 Walzem Road | Windcrest, Texas 78218 | 1-800-961-2888 | Intl: +1 210 312 4700
www.**rackspace**.com

UK Office
Rackspace Ltd.
5 Millington Road
Hyde Park Hayes
Middlesex, UB3 4AZ
Phone: 0800-988-0100
Intl: +44 (0)20 8734 2600
www.rackspace.co.uk

Benelux Office
Rackspace Benelux B.V.
Teleportboulevard 110
1043 EJ Amsterdam
Phone: 00800 8899 00 33
Intl: +31 (0)20 753 32 01
www.rackspace.nl

Hong Kong Office
9/F, Cambridge House, Taikoo Place
979 King's Road,
Quarry Bay, Hong Kong
Sales: +852 3752 6488
Support +852 3752 6464
www.rackspace.com.hk

Australia Office
Level 4, 210 George Street,
Sydney, NSW 2000
Phone: 1-800-722577
www.rackspace.com.au