The Cloud Pillars:

Five Foundational Building Blocks for Apps in the Cloud

rackspace®

Table of Contents

rackspace®

Introduction

The cloud has delivered scalable, elastic and cost-efficient computing to businesses of all sizes. But few organizations with applications living on the cloud have taken full advantage of what the cloud enables. "Cloudy apps" or "cloud-aware apps" suggest the emergence of applications that are designed with cloud infrastructure as an integral part of their design. Taking full advantage of the cloud's capabilities allows for a whole new generation of robust, fault-tolerant apps that can easily—and automatically—scale to meet growing demands.

This white paper is intended to give you an overview of the five pillars for applications in the cloud world. Building software with each of these considerations can result in dramatically increased application performance. However, incorporating even a few of these concepts into your development will allow your application to be more cloud ready.

The five pillars of cloudiness are:

1. **Distributed and Parallel Computing:** implementing message queue systems to allocate computational tasks
2. **Modular Design and Service-Oriented Architecture:** splitting monolithic applications into smaller parts and consuming third-party services
3. **Horizontal Scaling with Stateless Applications:** using tokens to authenticate users across a distributed architecture
4. **Becoming Agile During Different Demand Curves:** using monitoring and cloud APIs to dynamically modify your environment as traffic changes
5. **Security in the Cloud:** keeping your application secure against threats in the cloud

The following is an overview of these concepts, not a deep dive into the technical details for implementation. For more technical information, be sure to consult the **Rackspace Developer Blog**, the **Rackspace Knowledge Center** or **contact** one of our support specialists.

Pillar 1: Distributed and Parallel Computing

One of the advantages that the cloud presents is the ability to consume compute power as a utility. This gives you the opportunity to use a virtually limitless pool of computational power rather than being confined to a set amount of servers. With compute on-demand, setting up a distributed system to process an array of tasks is the first pillar of cloudiness we need to consider.

Due to limited resources, many apps in the past were written to perform tasks synchronously. If there were nine tasks required to complete an operation, the system would complete them in order—regardless of whether some tasks could be done in parallel.

The abundance of resources and the distributed model of computing flips this notion on its head, allowing multiple resources to pick off tasks from a queue. Each worker server in the configuration communicates with a central group of message servers to find the next task to be completed; those jobs without dependencies on other tasks can even be done in parallel. These servers then begin working on each task asynchronously, resulting in a faster overall completion time and a more efficient system. Message queue systems such as **RabbitMQ**, **ActiveMQ**, **ZeroMQ**, **IronMQ**, **Rackspace Cloud Queues** or **SQS** help route the tasks to the worker servers.

Ultimately, integrating a message queue makes your application more scalable. With a message queue system in place, you can scale worker servers as needed to perform tasks. Implementing this first cloud pillar creates the ability to service even more requests and perform more computation in less time. It breaks away from the old days of sequential computing tasks.

Pillar 2: Modular Design and Service-Oriented Architecture

In the past, applications were often tightly bound together in one big monolithic bundle. The web front end, application and database could all be located on the same host server. This tight coupling sometimes led to a domino-style cascade if a single component failed. Using a modular design that's loosely coupled—the second pillar of cloudiness—can help mitigate this risk.

This notion of modular design first originated with splitting out the web, application and database tiers of an application. Placing these different tiers on different hardware, gives the ability to be more targeted when deciding which pieces of the infrastructure to scale. Furthermore, the separation allows administrators and engineers to quickly troubleshoot the system when problems arise.

Traditionally, modular design looks like Figure 1 below.

In the drawing at right, the Varnish caching servers, web servers, application servers and database servers are all decoupled from each other. What was once a monolithic application has been split into atomic components that are scalable and replaceable. When we have more modular tiers, we have more choice on what piece of the application to scale as the system is placed under a heavy load. This lack of interdependency not only makes scaling and processing simpler but also makes the application pieces easier to maintain.

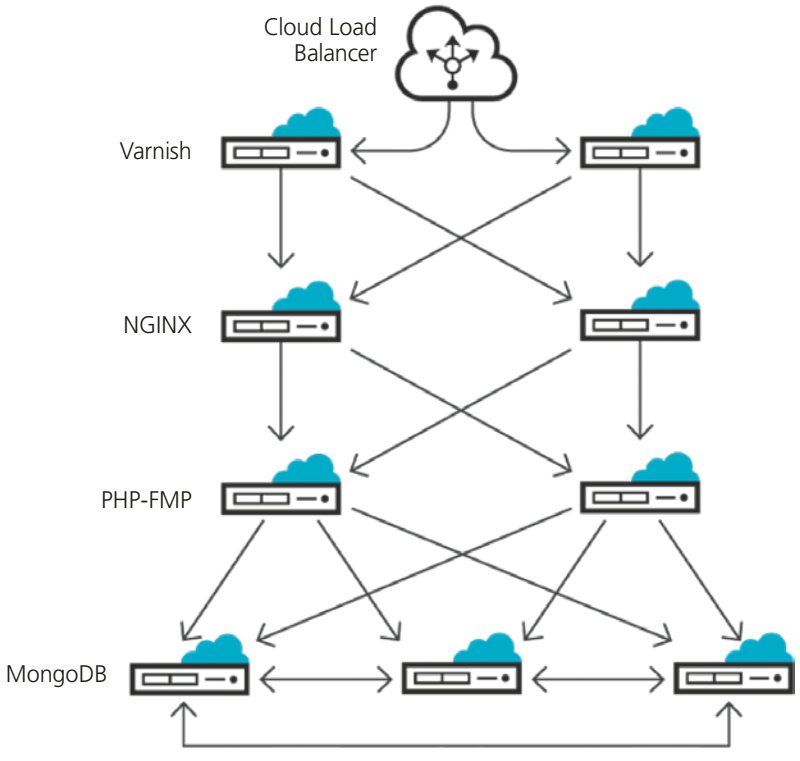

Figure 1

rackspace

Service-oriented architecture takes this idea a step further, isolating services that have distinct roles from each other. These different pockets of infrastructure, with unique roles in the overarching application, can become independently scalable from each other. As site traffic increases to one particular service, you can react by allocating additional resources to one unique service.

For example, a monolithic ecommerce application could be split into multiple pieces, such as: user authentication, sign-up, profile settings, storefront, shopping cart and checkout. Furthermore, each service's web, application and database layer can be split out as well. This allows for more resources to be applied to the shopping cart database while decreasing the requisite resources for the storefront's web nodes. This modularity allows a business to single out a particular service and add additional resources as it experiences higher-than-usual load.

In the cloud, we don't have to stop there. Service-oriented architecture can extend beyond those services that your development team defines. In fact, one of the strengths of the cloud is the ecosystem of services that it has engendered. Most likely, your cloud hosting provider will have an array of products that you can consume as a service, including load balancers, databases and storage (object and block) to name a few.

Additionally, third-party companies, like **CloudFlare** or **Incapsula**, have a variety of tools and services that can assist with anything from load testing, user authentication, DDoS mitigation and more. By incorporating some of these third-party services, you can decrease the time to market for your application. Figure 2 below is an illustration of how an application can incorporate services from both a hosting partner like Rackspace® and tools from other vendors:

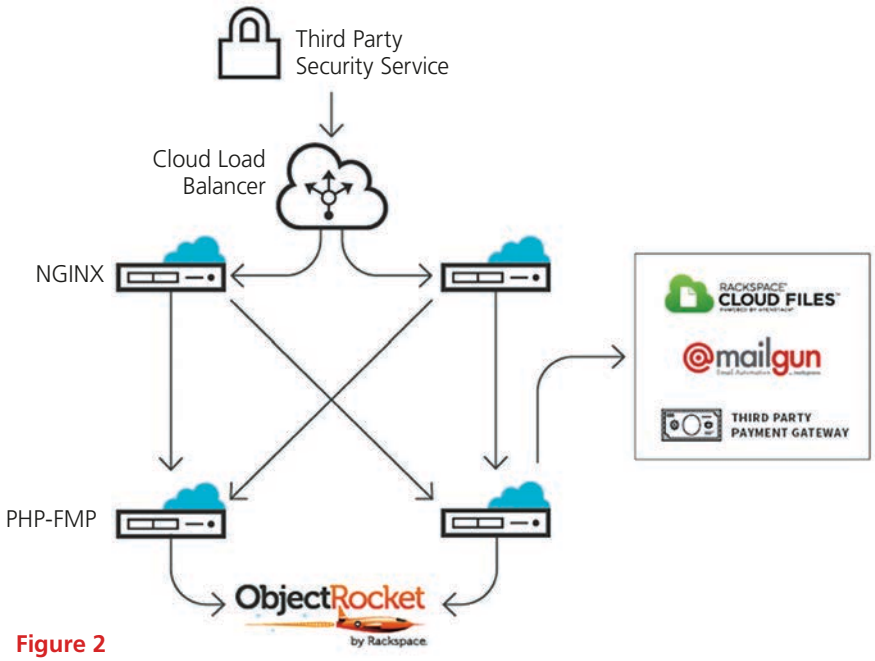

Figure 2

Instead of maintaining your own storage platform, you could consume **Cloud Files via an API**. You might even stop managing and patching your own email server by incorporating **Mailgun via an API**. If you dislike database administration, consider tapping into the fully sharded MongoDB solution by **ObjectRocket**. Rather than messing with scaling challenges and the headache of compliance with a payment system, you can integrate the **Stripe API** or **PayPal API** into your app as a payment gateway. Adopting specialized services can supplement any need your team may have. Soar on your strengths, offload your weaknesses—this is the way in the cloud world.

Pillar 3: Horizontal Scaling with Stateless Applications

Because HTTP is not a stateful protocol, web application developers have had to confront the issue of whether or not to record the state of a user's session.

For those who chose to manage for state during a user session, several technologies were created to help solve this issue. On the client side, for instance, cookies could be placed inside the end user's browser to track state. And on the server side, a database could be used to store information about the state of a user session. Such approaches—whether client or server side—became inadequate as the internet evolved and as web applications needed to scale massively across cloud-based infrastructure.

While an effective vehicle to store session information, cookies have fallen out of favor because many users concerned with privacy have opted for more restrictive browser security settings that block this particular method.

Storing session information in a database or using a session state service became increasingly difficult as configurations began growing in complexity. As more web and application servers were brought online to accommodate higher traffic levels, it became more challenging to synchronize the user's state information across an increasing set of potential failover servers.

The cloud, with its ability to use APIs to quickly scale from tens to hundreds of servers within a configuration, exacerbates this problem of marrying up a user to the exact server that holds their session. Further complicating the problem is the cloud's capacity to not only bringing servers online, but to quickly remove them as they become unnecessary.

So what can be done when traditional approaches to tracking session state information come up short? The short answer is to create stateless applications, to remove the burden of tracking and managing for state within a server configuration.

Representational State Transfer (**REST**) is a way to architect applications that can bypass many of the difficulties associated with storing a user's state in a highly distributed cloud environment.

RESTful web services are stateless, meaning your application will not store any state information with respect to the client. When the client makes a request, each request happens in isolation. The underlying infrastructure doesn't care where the request came from and doesn't care about the original session—each request is handled separately.

Rather than having to record the state of a user session, a RESTful application can infer the state of a session from the URL that the client passes through the browser. A RESTful web service uses standard HTTP methods, such as GET, PUT, POST and DELETE. The server then processes the request from the client and returns the information back in the proper form.

By processing a request that is sent by the client, the server no longer has to store or maintain the state of the user. This allows the infrastructure for such a stateless application to scale almost infinitely, interpreting the desired "state" of each user that sends in requests.

Cloud and stateless protocols have started changing the way we design applications to increase the effectiveness of horizontally scaling. Without sessions to require centralization in our application, we have the flexibility to be agile with our scaling methodology. Rather than scaling servers vertically (by increasing the computational resources, such as RAM, on a single server), we can begin to grow horizontally (adding additional servers to our configuration to handle load).

Furthermore, stateless applications allow for more loosely coupled components. This type of segmented application can inherently take advantage of the second pillar on modularity, allowing different services in the app to scale independently of one another. Additionally, this added elasticity in the application allows us to add and subtract servers in a way that mimics the traffic demand curve, which is discussed in further detail next in the fourth pillar.

Pillar 4: Becoming Agile During Different Demand Curves

The ability to adapt to changing traffic patterns is extremely important in a cloud environment. While the cloud can be used for a steady-state environment, it really shines when used for dynamic scenarios, aided by the power of monitoring and API integration.

The workloads for every application in the cloud often fit into one of these four buckets:

- On and Off
- Variable
- Consistent
- Fast Growth

With a traditional hosting model, you often have the same number of servers online at all times—it can be difficult to bring on a physical machine for just a short period of time. This way of hosting invariably creates waste; some resources are maxed out while others are utilized well below their full potential. The cloud can help minimize this waste by bringing servers online as they are needed. Let's take a look at what each of these workloads looks like graphically:

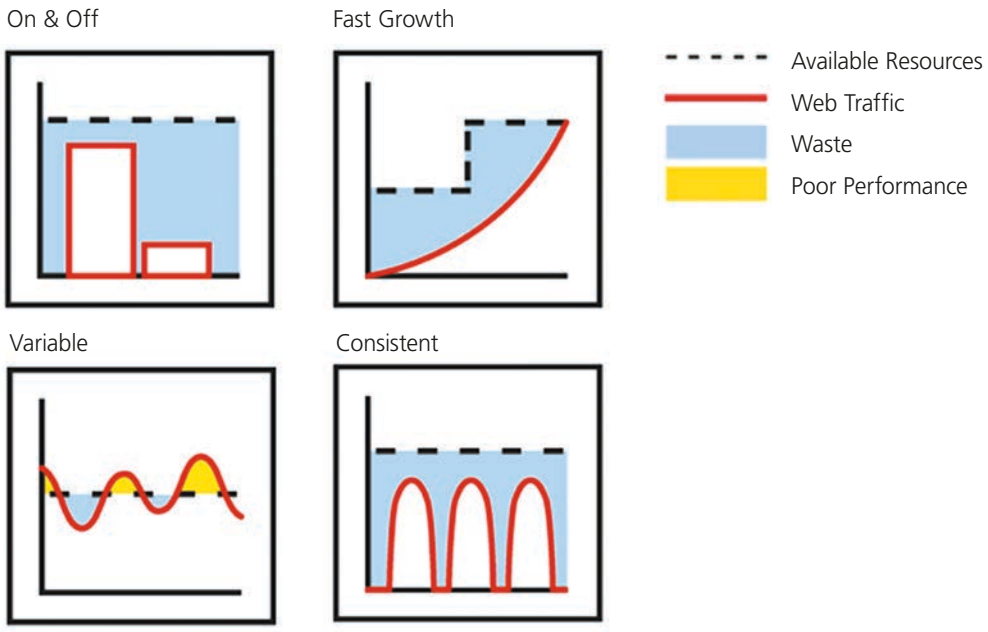

Figure 3

The dashed line in each image represents the compute power provided by the traditional model of using physical servers. Since there is typically a contract required to commission the hardware, these servers are often brought online all at once. This is why you see a consistent horizontal line or, in the case of the Fast Growth scenario, a step function when more gear is added to accommodate the growth.

The red lines represent traffic to a particular site—this is the amount of compute resources that your visitors require from your servers. When this red line is below the dashed line, you have an overprovisioned environment. In this case, you are paying for servers to be online when they're unnecessary. The situation of not having enough compute resources occurs when the red line goes above the dashed line; this represents higher-than-usual traffic hitting a site.

The ideal state is to have the dashed line mimic the red line as closely as possible, ensuring that there are just enough compute resources to satisfy demand. This not only results in cost savings by preventing overprovisioning, it also creates a stable environment that can withstand sudden traffic surges.

Using a combination of load testing, monitoring and scaling technologies can create a self-aware cloud application, one that scales resources up and down with the ebb and flow of demand.

In order to do this, you first have to understand the limits of your infrastructure as it relates to your application. How many concurrent connections can you have? How many users can the environment support while providing optimal page load times? What number of transactions will cause the configuration to fail? Businesses can identify stress points and potential bottlenecks before going live by load testing their app with companies like **Load Impact** and **Apica**.

These solutions generate traffic from different geographic regions while measuring the response rates and throughput of each packet. Results from these tests can include CPU and memory usage along with disk and network I/O, helping you fine-tune your app. This enables you to identify issues across the user experience prior to those events happening in your production environment. You can get real time alerts for these metrics by using the **Rackspace Cloud Monitoring** product.

Once you understand your app's limitations, you can also use an Application Performance Monitoring (APM) solution to track key metrics at the application level. APM tools like **AppDynamics** and **New Relic** provide application intelligence based on user-defined behaviors such as logging in, updating a cart or checking out.

This intelligence can be powerful when combined with your cloud hosting provider's API. Most cloud APIs allow you to create, read, update or delete a cloud resource, enabling the app to modify its configuration environment while removing the need for having to wait for the server to be racked, cabled and configured.

This allows an application to become aware of its environment and scale the configuration as traffic exceeds defined 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 subsided? The app can then tell the infrastructure to remove unnecessary servers, allowing the business to provide the same level of service to consumers while dynamically decreasing their costs.

Rackspace has simplified this process by providing a service called **Auto Scale**. By creating a few simple rules that you define and control, your application automatically scales to meet user demand. You can do this either in our Cloud Control Panel or by using APIs. The service is free to use—you only pay for the servers and bandwidth that you use in the process.

By understanding the load that your application can handle, monitoring your production traffic and integrating your application with a cloud infrastructure API, you are able to unlock the agility of the cloud.

Pillar 5: Security in the Cloud

When it comes to security, the risks that cloud applications face are the same as those found with traditional applications. However, the cloud does change the security conversation where security becomes, in part, the responsibility of both the application builder and the cloud hosting provider.

Infrastructure-as-a-Service is the foundational building block of the cloud, the raw compute power that is akin to a "bare metal" server. This puts the onus on you, the consumer, to implement the correct security protocols. Just like a physical server, this also includes the need to regularly patch and update the servers to prevent a known vulnerability from being exposed.

With the Rackspace Managed Cloud, you not only get all the advantages of controlling the bare metal cloud servers, you have a group of experts that can help ensure your systems are patched with the latest security releases.

Another way to bake in security into your application is if you adopt the service-oriented architecture mentioned in the second cloud pillar. By splitting your app into different services, you achieve more control over what gets to access particular services.

In the past, an application could have simultaneously handled web requests from users and, in the same codebase, made a backend call to the database to ensure they were authenticated properly. This resulted in many potential exploits—SQL injections were running rampant in those days—that could lead to your front-end compromising the system by issuing a bad database call.

With service-oriented architecture, you can have a "dumb" front-end. In this model, your code can take requests from users and pass them to your authentication system via a REST API. The authentication system then does all the actual lookups, providing some insulation between the authentication database, the front-end and ultimately the user. By requiring the front-end servers to interact with the authentication servers in this way, you are able to keep your authentication API and code hidden from users who may look to scan it for vulnerabilities.

Conclusion

The ability to have on-demand compute resources is a major difference between the cloud and traditional hosting. But the power of the cloud extends well beyond the utility pricing model and the ability to quickly provision servers without a purchase order.

As the cloud has matured, the availability of virtually limitless compute power has culminated in applications becoming self-aware, controlling the infrastructure that they run on. These five pillars are fundamental building blocks for applications to achieve this result. As the technology within the cloud space evolves, expect to see developers and engineers continuing to push the boundary of what the cloud can do.

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
Rackspace Hosting Australia PTY LTD
Level 1
37 Pitt Street
Sydney, NSW 2000
Australia