

O'REILLY®

Second
Edition

Architecting for Scale

How to Maintain High Availability and
Manage Risk in the Cloud

Free
Chapters

compliments of



Cockroach Labs



Lee Atchison

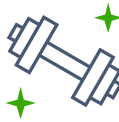
When traffic spikes, apps can fail. Are you ready?

Architect your apps to survive the unexpected.



Scale elastically with distributed SQL

Say goodbye to sharding
and time-consuming
manual scaling.



Survive anything with bullet-proof resilience

Rest easy knowing your
application data is always
on and always available.



Thrive on-prem and across clouds

Deploy your apps
anywhere with a flexible,
cloud-native database

Get started for free today

cockroachlabs.com/scale

Trusted by innovators


COMCAST

SPACEX

BOSE

LUSH

 **rubrik**

wework

SECOND EDITION

Architecting for Scale

*How to Maintain High Availability
and Manage Risk in the Cloud*

This excerpt contains Chapters 4 and 12 of *Architecting for Scale*. The complete book is available on the O'Reilly Online Learning Platform and through other retailers.

Lee Atchison

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Architecting for Scale

by Lee Atchison

Copyright © 2020 Lee Atchison. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Kathleen Carr

Developmental Editor: Amelia Blevins

Production Editor: Beth Kelly

Copyeditor: Jasmine Kwityn

Proofreader: Arthur Johnson

Indexer: Ellen Troutman-Zaig

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Rebecca Demarest

July 2016: First Edition

February 2020: Second Edition

Revision History for the Second Edition

2020-02-28: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492057178> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Architecting for Scale*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Cockroach Labs. See our [statement of editorial independence](#).

978-1-492-05717-8

[LSI]

Table of Contents

Foreword.....	vii
4. Services and Data.....	1
Stateless Services—Services Without Data	1
Stateful Services—Services with Data	1
Data Partitioning	2
Timely Handling of Growing Pains	5
12. Getting Started Architecting for Scale with the Cloud.....	7
Six Levels of Cloud Maturity	8
Level 1: Experimenting with the Cloud	9
Level 2: Securing the Cloud	9
Level 3: Using Servers and Applications in the Cloud	9
Level 4: Enabling Value-Added Managed Services	10
Level 5: Enabling Cloud-Unique Services	10
Level 6: Cloud All In	11
Organization Versus Application Maturity Level	11
Cloud Adoption Mistakes	11
Trap #1: Not Trusting Cloud Security	12
Trap #2: Performing Cloud Migration via Lift-and-Shift	12
Trap #3: The Lure of Serverless—Depending Too Much on the Hype	13
When and How to Use Multiple Clouds	13
Defining What We Mean by Multiple Clouds	14
Which Model? Which Cloud?	17
The Cloud in Summary	18

Foreword

There was a time in the early days of the internet and the app economy when scale was a challenge that only a select few really had to fret over. The term “web scale” was coined to address the early challenges of companies like Amazon, Google, and Microsoft, and later addressed the architecture work taken on by Facebook, Netflix, and others to first reach massive volumes of users. But now, 25 years from the start of digital transformation, nearly every company sees massive concurrency and increasing demands on their applications and infrastructure. Even the smallest startup must think about scale from day one.

Modern applications are overwhelmingly data-intensive. We expect highly dynamic, personalized experiences to be available instantly. These applications are increasingly global in nature, as well—think Spotify, Uber, or Square—and have complex needs in trying to balance both consistency and locality. Our applications have become ever more complicated with ever more components that must be managed and scaled.

Further challenging us, users have come to expect absolutely flawless experiences. There is zero tolerance for latency or downtime, even under the heaviest workloads. In this context, your most successful moments—when your application sees rapid growth or when your content goes viral on social media—have the potential to become your worst day.

This need not be the case. If you plan, test, and continually optimize for scale, unexpected and rapid growth can come without you losing sleep or scrambling to patch or repair a failing stack. It is for this reason that Cockroach Labs is proud to sponsor *Architecting for Scale*. CockroachDB is all about effortless scale and bulletproof resilience, and so we are sharing two chapters from the book associated with scaling data to support modern applications; “Services and Data” and “Getting Started Architecting for Scale with the Cloud.”

We hope you enjoy this book excerpt, and that you consider CockroachDB for your next development project.

Services and Data

As you build and migrate your application to a service-based architecture, it is critically important to be mindful of where you store data and state within your application.

Stateless Services—Services Without Data

Stateless services are services that manage no data and no state of their own. The entire state and all data that the service requires to perform its actions is passed in (or referenced) in the request sent to the service.

Stateless services offer a huge advantage for scaling. Because they are stateless, it is usually an easy matter to add additional server capacity to a service in order to scale it to a larger capacity, both vertically and horizontally. You get maximum flexibility in how and when you can scale your service if your service does not maintain state.

Additionally, certain caching techniques on the frontend of the service become possible if the cache does not need to concern itself with service state. This caching lets you handle higher scaling requirements with fewer resources.

Not all services can be made stateless, obviously, but for those services that can be stateless, it is a huge advantage for scalability.

Stateful Services—Services with Data

When you do need to store data, given what we just discussed in the preceding section, it might seem obvious to store data in as few services and systems as possible. It might make sense to keep all of your data close to one another to reduce the footprint of the services that need to know and manage your data.

Nothing could be further from the truth.

Instead, localize your data as much as possible. Have services and data stores manage only the data they need to perform their jobs. Other data should be stored in different servers and data stores, closer to the services that require that data.

Localizing data this way provides a few benefits:

Reduced size of individual datasets

Because your data is split across datasets, each dataset is smaller in size. Smaller dataset size means reduced interaction with the data, making scalability of the database easier. This is called functional partitioning. You are splitting your data based on functional lines rather than on the size of the dataset.

Localized access

Frequently when you access data in a database or data store, you are accessing all the data within a given record or set of records. Often, much of that data is not needed for a given interaction. By using multiple reduced dataset sizes, you reduce the amount of unneeded data from your queries.

Optimized access methods

By splitting your data into different datasets, you can optimize the type of data store appropriate for each dataset. Does a particular dataset need a relational data store? Or is a simple key/value data store acceptable?

Keeping your data associated with the services that consume the data will create a more scalable solution, and easier-to-manage architecture and will allow your data requirements to more easily expand as your application expands.

Data Partitioning

Data partitioning can mean many things. In this context, it means partitioning data of a given type into segments based on some key or identifier within the data. It is often done to make use of multiple databases to store larger datasets or datasets accessed at a higher frequency than a single database can handle.

There are other types of data partitioning (such as the aforementioned functional partitioning); however, in this section, we are going to focus on this key-based partitioning scheme.

A simple example of data partitioning is to partition all data for an application by account, so that all data for accounts whose name begins with A–D is in one database, all data for accounts whose name begins with E–K is in another database, and so on

(see [Figure 4-1](#)).¹ This is a very simplistic example, but data partitioning is a common tool used by application developers to dramatically scale the number of users who can access the application at any one time, as well as to scale the size of the dataset itself.

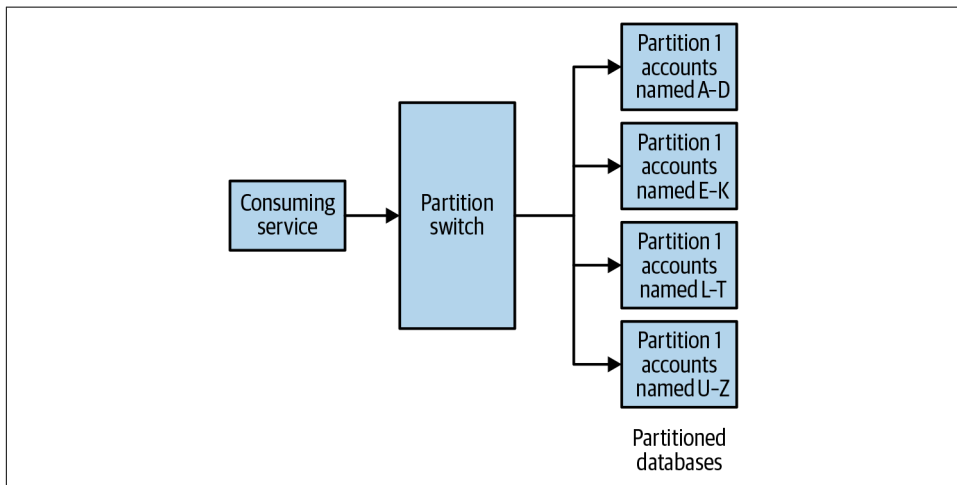


Figure 4-1. Example of data partitioning by account name

In general, you should avoid data partitioning whenever possible. Why? Well, whenever you partition data this way, you run into several potential issues:

Application complexity

You increase the complexity of your application because you now have to determine where your data is stored before you can actually retrieve it.

Cross-partition queries

You remove the ability to easily query data across multiple partitions. This is specifically useful in doing business analysis queries.

Skewed partition usage

Choosing your partitioning key carefully is critical. If you choose the wrong key, you can skew the usage of your database partitions, making some partitions run hotter and others colder, thus reducing the effectiveness of the partitioning while complicating your database management and maintenance. This is illustrated in [Figure 4-2](#).

¹ A more likely account-based partitioning mechanism would be to partition by an account identifier rather than by account name. However, using account name makes this example easier to follow.

Repartitioning

Repartitioning is occasionally necessary to balance traffic across partitions effectively. Depending on the key chosen and the type and size of the dataset, this can prove to be an extremely difficult task, an extremely dangerous task (data migration), and in some cases, a nearly impossible task.

In general, account name or account ID is almost always a bad partition key (yet it is one of the most common keys chosen). This is because a single account can change in size during the life of that account. Take a look at [Figure 4-2](#). An account might begin small and thus may easily fit on a partition with a significant number of small accounts. However, if it grows over time, it can soon cause that single partition to not be able to handle all of the load appropriately, and you'll need to repartition in order to better balance account usage. If a single account grows too large, it can actually be bigger than what can fit on a single partition, which will make your entire partitioning scheme fail, because no rebalancing will solve that problem.

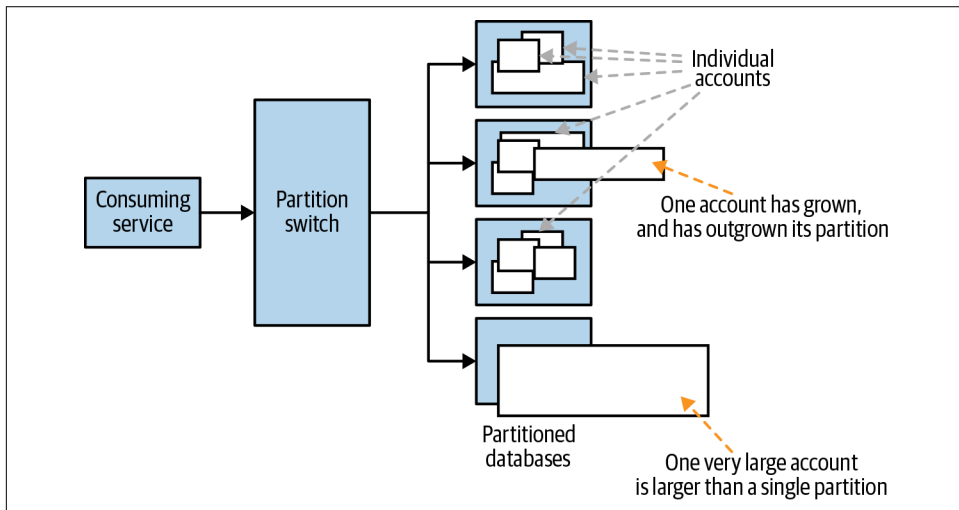


Figure 4-2. Example of accounts overrunning data partitions

A better partition key would be one that would result in consistently sized partitions as much as possible. Growth of partitions should be as independent and consistent as possible, as shown in [Figure 4-3](#). If repartitioning is needed, it should be because all partitions have grown consistently and are too big to be handled by the database.

One potentially useful partitioning scheme is to use a key that generates a significant number of small elements. Next, map these small partitions onto larger partitioned databases. Then, if repartitioning is needed, you can simply update the mapping and move individual small elements to new partitions, removing the need for a massive

repartitioning of the entire system. Selecting and utilizing appropriate partition keys is an art in and of itself.

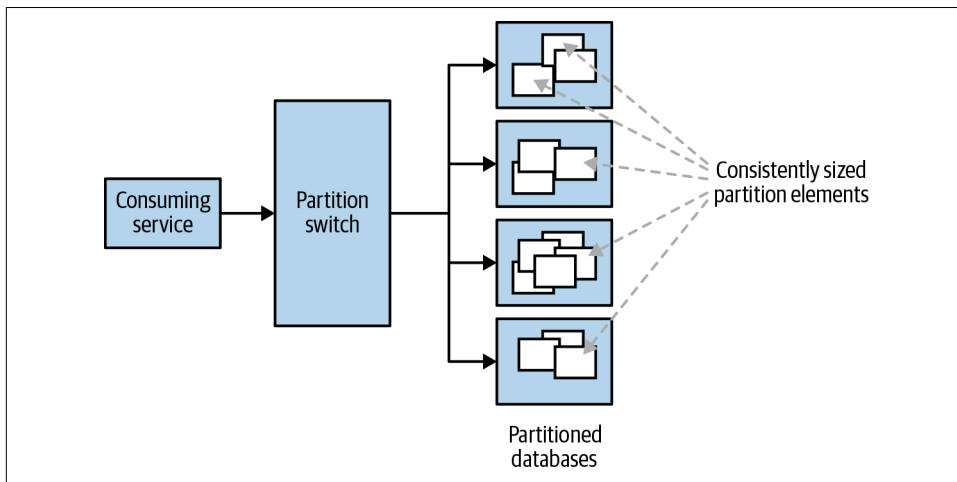


Figure 4-3. Example of consistently sized partitioned elements

Timely Handling of Growing Pains

Most modern applications experience growth in their traffic requirements, in the size and complexity of the applications themselves, and in the number of people working on the applications. Often, we ignore these growing pains, waiting until the pain reaches a certain threshold before we attempt to deal with it. However, by that point, it is usually too late. The pain has reached a serious level, and many easy techniques to help reduce it are no longer available for you to use.

If we don't think about how our application may grow while we are architecting the application before it scales, we will lock ourselves into architectural decisions that can block our ability to scale as our business requires.

Instead, while designing and architecting your new application and changes to your existing applications, consider how those changes will be impacted by potential scale changes in the future. How much room to scale have you built in? What is the first scalability wall you will run into? What happens when you reach that wall? How can you respond and remove the barrier without requiring a major rearchitecture of the application?

By thinking about how your application will grow long before it grows to those painful levels, you can preempt many problems and build and improve your applications so that they can handle these growing pains safely and securely.

Getting Started Architecting for Scale with the Cloud

Awareness and knowledge of the cloud has grown significantly in the last several years. It wasn't that long ago that "using the cloud" was something only progressive organizations considered doing or otherwise was limited to startups looking to reduce capital costs in infrastructure utilized.

But it did not take long for enterprises to realize the value of the cloud. Acceptance of the cloud by all but the most conservative enterprises in the last few years has made using the cloud mostly mainstream. Or at least the desire to adopt the cloud is now mostly mainstream.

For many enterprises, though, finding success in the cloud is still a daunting challenge. Too often, organizations set overly high expectations for the benefits of migrating to the cloud while underestimating the amount of work required in the migration itself and the impact the migration has on the culture of their company. An unfortunate result can be a vicious cycle of blame, finger pointing, and grasping for something—anything—that could be considered a victory.

When they find it, organizations may decide they've had enough and stop the migration process before they can take full advantage of the cloud. But by putting off real reform, they won't realize the cost and innovation benefits that drove the cloud migration project in the first place. As migration costs balloon and promised features, functionalities, and applications fail to materialize, companies can end up seeing the cloud as little more than an expensive boondoggle.

Why did this happen? Often, the biggest error comes from thinking about the cloud in the wrong way. Enterprise management tends to think of a cloud migration as a

simple “lift-and-shift” operation—simply move existing applications that are running in their own data centers directly to the cloud, with as few changes as possible.

However, real cloud success, at scale, requires much more than lift-and-shift. It demands successfully navigating the world of the dynamic cloud. The dynamic cloud doesn’t just facilitate application scaling; it makes the process faster and easier. It also helps development teams respond to changes faster and to implement these changes more quickly. That’s not a luxury—it’s a necessity to ensure the availability of modern applications that exhibit extreme scaling needs and extremely spiky performance. When you don’t know when your customers will use your application, it’s hard to predict your static infrastructure needs. A dynamic infrastructure is required to meet the needs of these modern applications without wasting significant resources.

Using the dynamic cloud, however, takes a higher level of commitment to using your cloud resources effectively than does a simple lift-and-shift. That’s because after a migration, the type of application and infrastructure visibility that is required changes. Many resources become dynamic, so keeping track of what resources are important for what purposes also becomes dynamic. Additionally, applications now run on an infrastructure outside of a team’s direct control, a concept that is foreign to many large enterprises.

Fortunately, becoming proficient in the dynamic cloud does not have to be scary or dangerous. Adopting the cloud can be done safely and effectively, but it is a continual learning experience. Organizations must be willing to learn and adapt cloud offerings to match their needs and expectations with the reality of what the cloud can provide. There is a learning curve of cloud maturity that ranges from simple lift-and-shift to a full architecture rewrite and adoption of the cloud and all of its capabilities. This chapter discusses this cloud maturity curve.

Six Levels of Cloud Maturity

Critically, you can’t expect to get there all at once. There are six basic maturity levels that organizations go through during their cloud adoption process:

- Level 1—Experimenting: What is the cloud?
- Level 2—Securing the cloud: Can we trust the cloud?
- Level 3—Enabling servers and SaaS: Lift-and-shift, confirmation the cloud works pretty well
- Level 4—Enabling value-added services: Dynamic cloud becomes a practice
- Level 5—Enabling unique services: Dynamic cloud is deeply ingrained in the culture
- Level 6—Mandating cloud usage: Why do we need our own data centers?

To be successful in moving to the cloud, organizations must realize that this continuum of cloud maturity exists and understand the implications for their actions and processes. Moving from one level of maturity to the next isn't always easy, it isn't always fast, and the specific details differ for every organization. Also, organizations sometimes settle on a level of cloud maturity that's right for their culture but short of the end goal. That's OK, if that meets the expectations and requirements of their business.

Level 1: Experimenting with the Cloud

This first tentative step into the cloud relies on safe technologies—technologies that apply in simple ways to applications and parts of applications that are typically less mission critical.

Level 1 involves using the cloud for a single, simple piece of an application to test how cloud services work. Often, the first service used is a storage solution such as Amazon Simple Storage Service (S3), because it's easy to store some things in the cloud and avoid addressing the complex processes and systems needed for cloud-based computation, such as cloud-based servers and serverless computation.

This level usually starts as a one-off experiment, where one or more teams conduct stand-alone migrations. No cloud policies are created at this point; instead it's all about figuring out exactly what the cloud is.

Level 2: Securing the Cloud

This is a critical evolution point in an organization's cloud culture, as it begins to involve disciplines throughout the company—legal, finance, security, and so on. Trust becomes a core question at this point. Can we trust depending on the cloud for our business success? Can we trust putting our data in the cloud? Do we know how and where it is appropriate to trust and how to ensure that the cloud is secure enough to meet our needs?

This is when policies on how the cloud can be used within a company begin to be formed. The precise nature of these guidelines, from formal policies to ad-hoc “company culture” understandings, doesn't matter that much. What's important is that the entire company is involved and all stakeholders have input.

Level 3: Using Servers and Applications in the Cloud

The third stage of cloud maturity comes when an organization begins to replace on-premise servers and other backend resources. These are still simple lift-and-shift applications, with a basic philosophy of “Let's just move an application to the cloud and see what happens.”

At this level, the goal is to understand how the cloud works for an entire application. This is where the organization begins to enjoy actual advantages from using the cloud, such as reduced cost and increased flexibility.

Enterprises need to be careful here, however. Level 3 can be a danger point. If enterprises attempt to determine the value of using the cloud to run their applications at this stage, they may find they're bearing the costs of the cloud without enjoying the corresponding benefits. That can cause companies to give up and regard their entire cloud effort as a failure. The solution is to use this level not as an end point but as a transition point. Avoid the temptation to stop once you've completed a lift-and-shift and say, "That's enough—we're now in the cloud." It's important to go the next steps and take advantage of the capabilities the cloud provides.

Level 4: Enabling Value-Added Managed Services

Here is where some of the inherent value of the cloud begins to appear. At this level, organizations start to look at cloud managed services, such as managed databases. Managed database services such as Amazon Relational Database Service (RDS), Amazon Aurora, and Microsoft Azure SQL provide database capabilities to applications while requiring less overall management. Rather, you let the cloud provider manage the database. Organizations may also look at services such as Amazon Elasticsearch, Amazon Elastic Beanstalk, and Amazon Elastic Container Service (ECS) to provide managed computation.

As the dynamic cloud starts taking effect here, the cloud's biggest benefits kick in. This is also when companies commit to using the cloud for at least some of their strategic applications and services.

Level 5: Enabling Cloud-Unique Services

Once a company becomes a cloud-enabled organization, it can look to leverage high-value, cloud-specific services. Uniquely available in the cloud, these services are designed specifically for the dynamic cloud. Some examples of services utilized at this level include serverless computing such as AWS Lambda or Microsoft Azure Functions, highly scalable databases such as Amazon DynamoDB, and other generalized services, such as Amazon Simple Queue Service (SQS) and Amazon Simple Notification Service (SNS).

At Level 5, the concept of dynamic cloud becomes embedded in an organization's application development and management processes. Use of these services also begins to tie the enterprise to specific cloud providers. While many cloud providers offer serverless capabilities, each one does so in a slightly different manner. As organizations begin to use these higher-value, cloud-unique services, they become tied not only to the cloud but also to specific cloud providers.

Level 6: Cloud All In

This is the ultimate maturity level of cloud adoption. At this topmost level, organizations begin to require use of the cloud for all new applications and begin to require existing applications to be migrated to the cloud. The usual end goal for customers at this level is to get rid entirely of their own corporate data centers and depend on the cloud for all infrastructure needs.

This level is especially common in cloud-native companies—the needs of legacy applications complicate the ability to be all in on the cloud. It's significantly easier for cloud-native companies to mandate all-cloud-based applications. More established enterprises may choose to retain their legacy data centers. However, more and more traditional enterprises are taking the cloud plunge and abandoning the business of managing their own data centers.

Organization Versus Application Maturity Level

The six cloud computing maturity levels apply to individual applications, organizations, or entire enterprises. But the cloud maturity level of a particular application may be higher or lower than that of the organization as a whole.

For example, early candidates for cloud migration include internal applications, because they present less risk of negatively impacting customers and the business itself. In fact, an internal application may jump to Level 6.

Larger, more complex legacy enterprise applications may be significantly slower in their cloud adoption strategy.

Meanwhile, the overall enterprise itself might still sit at Level 1, 2, or 3 and never make it to Level 6.

This is entirely normal and expected and demonstrates the complexities of cloud adoption in large enterprise organizations.

Cloud Adoption Mistakes

When you adopt the cloud, it's really easy to fall into a few very specific traps that can lead to significant problems in your adoption strategy. These mistakes are often the cause of a failed migration, or at least a perception that the migration was unsuccessful. They can also cause the cost-to-benefit ratio to skew away from the true value of the cloud. Be careful not to fall into any of the following traps as you look at performing your cloud migration.

Trap #1: Not Trusting Cloud Security

One of the biggest misconceptions that companies new to the cloud deal with is the issue of trusting the cloud. This shows up in many different ways, but dealing with security is a main one.

Security is very important to nearly all companies. Moving to the public cloud means taking an application that is safely behind the company's firewall and putting it on a public cloud. The first time you consider doing this, it'll seem scary. Can you trust the cloud to keep your data secure? Is your application safe from attack in such a public environment?

The short answer? Yes.

For the vast majority of companies, your company is probably safer in the hands of a public cloud provider than it is behind your own firewall. Why is that true? Because cloud service providers make a living on trust. They would not be in business if they could not keep their customers' data secure.

Cloud providers invest heavily in building high-quality security teams that spend their time advancing the state of the art in security protocols and procedures. By putting your data in the hands of a reputable public cloud provider, you take advantage of the learnings and best practices created by the leaders in the security field. Unless your company has the same resources to invest in security as the cloud providers do, your company can benefit from these learnings in so many ways.

By using a public cloud provider and taking advantage of all the security offerings it provides, you can actually keep your applications and data safer in the public cloud than you can behind your own firewall.

Trap #2: Performing Cloud Migration via Lift-and-Shift

Early in the process of adopting the cloud, many companies consider moving applications to the cloud by simply taking the application off of servers in their own data center and moving them to servers they've created in the cloud.

This type of migration is called lift-and-shift, and we discussed it earlier in this chapter as a cornerstone of one of the maturity levels of cloud adoption.

While lift-and-shift is a valid way to very quickly get your application out of your data center and into a cloud-based data center, it doesn't do anything to make your application cloud friendly. It doesn't do anything to take advantage of the native value and native characteristics of the cloud. Yes, there are some benefits you can get from a lift-and-shift migration, including the ability to expand to additional data centers simply by launching servers in another region. But that is about where the benefits stop. In fact, the cloud can actually be worse at this type of basic application hosting than your own data centers. Why? Cost.

The cloud can and does provide significant cost benefits for users that take advantage of the dynamic allocation capabilities of cloud resources. But it typically can't compare in cost to the basic, static infrastructure provided by a noncloud data center. When you use the dynamic capabilities of the cloud, you can save money. If you simply lift-and-shift, you typically don't save money and often spend more.

Doing a lift-and-shift can cost you money and time and not give you any of the benefits you were wanting with a cloud migration.

Trap #3: The Lure of Serverless—Depending Too Much on the Hype

It's easy to get caught up in the hype of the cloud, and the latest and greatest cloud service offerings often seem like the solution to all your problems. However, like with any new technology, understanding how and where to apply the technology is critical to successfully using it. This most certainly applies to the newest Function as a Service (FaaS) offerings by cloud providers, such as AWS Lambda and Microsoft Azure Functions.

These offerings promise the ability to provide an execution environment for your software without the need for managing the servers they run on. This “serverless computing” offering is very attractive to companies that are wanting to use the cloud to reduce their infrastructure management costs. But, like all new technologies, FaaS offerings such as Lambda are good for some classes of problems and not good for other classes of problems.

Yet I often hear statements from individuals such as “Lambda will solve my computing infrastructure problems” and “We're moving all of our software to Lambda.”

To people thinking that FaaS offerings such as Lambda are a solution to all your problems, I say be careful. AWS Lambda and the equivalent offerings by other cloud providers give a huge advantage to a certain class of computing environments, but they can be overused.

If they are force fit into solving problems they weren't designed to solve, they actually can create more problems for you and your infrastructure management than they solve.

Use them as an important part of your application architecture, but don't depend on them to solve all your computing problems. Use them only where they make sense.

When and How to Use Multiple Clouds

When deciding to move an application to the cloud, you need to consider many factors before choosing a cloud provider. What features do you need? Which cloud is faster? Which one is cheaper? Which one is more reliable?

But here's another question that is being asked more and more often: how many cloud providers should I use?

The seemingly obvious answer is a single provider, but cloud customers cite a number of reasons to use multiple providers. First, some features you might want to use might be available on only one cloud provider, and other features might be only on another cloud provider. Another reason is that utilizing multiple providers instead of being tied to a single provider might offer better negotiation room when dealing with contracts. Yet another reason often cited is reliability—when one cloud provider goes down, the other cloud provider will still be available. Or the reasoning may just be seemingly random...part of your organization prefers one provider and part prefers another provider.

But your answer may or may not be the right one for your organization. Using multiple cloud providers may give you benefits, or it may actually hurt you, when you are doing it for any of the reasons just mentioned.

Let's take a look at what goes into making the best decision for your particular situation.

Defining What We Mean by Multiple Clouds

Before we talk about how many clouds you need, we need some definitions. The actual set of advantages and disadvantages of a multi-cloud arrangement depends greatly on the type of multi-cloud environment you are considering. So let's look at three different types of cloud configurations: joint cloud applications, selective cloud applications, and single cloud applications.

Joint cloud applications

A joint cloud application is when a single application uses two or more cloud providers to provide parallel capabilities. A given application or its services can run on any or all of the supported cloud providers, as shown in [Figure 12-1](#).

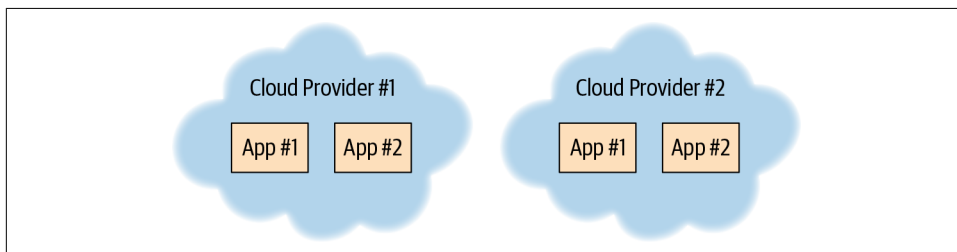


Figure 12-1. Joint cloud—applications run on multiple clouds

App #1 and App #2 can be run on either of the cloud providers. You can also load balance each application across both clouds simultaneously, if desired. If one cloud

becomes unavailable, the other cloud can take over responsibility for running the application.

Each application must be designed to run on either cloud provider, and the application can use either available provider to satisfy a given request. If one provider is unavailable, the other provider can take over to process requests for the application.

One major advantage of this approach is application resiliency. If a cloud provider experiences an issue, the application workload can be rerouted to the other cloud provider easily and relatively quickly. This lets the application continue functioning even if one provider has a massive failure.

This architecture is often cited as a solution to single-vendor cloud lock-in because you can easily switch your load between multiple cloud providers. However, this architecture also has significant disadvantages. For example, each development and operations team supporting the application must have an understanding of the workings of multiple cloud providers. This knowledge and understanding does not come for free. Similarly, each application must be tested and maintained on multiple cloud providers.

Additionally, when applications are written to support multiple cloud providers, they cannot take advantage of deeper feature capabilities provided by one particular provider. The application must be written to use the least common capabilities inherent in all the cloud providers being utilized.

In most cases, the shortcomings of this approach outweigh the perceived improvements in resiliency.

Selective cloud applications

This is when your company maintains relationships with multiple cloud providers, but any given application runs entirely on a single provider, as shown in [Figure 12-2](#).

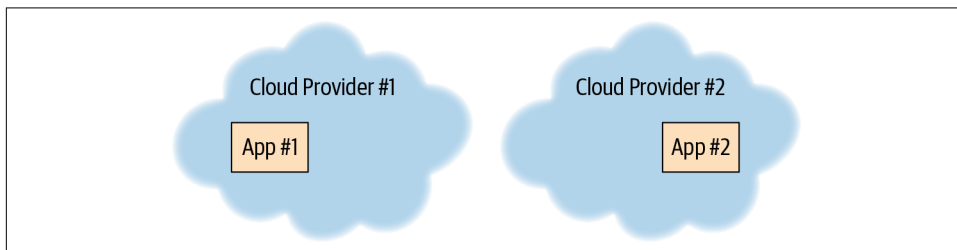


Figure 12-2. Selective cloud—each applications runs on a single selected cloud

You can see that each application is hosted on only a single cloud provider, but different applications may be hosted on different cloud providers.

In this scenario, a given component is designed to run and capable of running on only a single provider's cloud.

One perceived disadvantage of this approach is resiliency. If a cloud provider becomes unavailable, then the applications or services running on that provider will stop functioning. You cannot simply move traffic over to an alternate cloud provider. This is typically more of an intellectual issue than a practical one. It is rare for an entire cloud provider to become unavailable. Typically, only one or more availability zones or regions become unavailable. A properly written application can take advantage of multiple availability zones and regions to improve application resiliency without having to resort to using multiple cloud providers.

A real potential advantage of this architecture, though, is that individual applications or services can independently select which cloud provider they want to use based on whatever criteria makes sense for the application or service's owning team.

This architecture requires that each individual development and operations team supporting a given application learn and understand only how the single cloud provider it works with operates. Additionally, each team can take advantage of deeper feature capabilities unique to its specific cloud provider. The applications themselves can be designed, built, and optimized using cloud provider-specific best practices.

However, in this architecture the company must maintain multiple vendor relationships and agreements with each supported cloud provider. This is more of an administrative problem than a technical one, but it might be an issue for your organization.

In most cases, this approach gives you the desired flexibility of multiple clouds and the ability to do deep integrations with your cloud providers, without the application-level complexity that joint cloud applications require and without significantly sacrificing application resiliency.

Often organizations back into this particular architecture. One team or organization selects one cloud provider for its applications, while another team or organization selects another cloud provider for its own applications. The enterprise as a whole is multi-cloud, but individual applications are each single cloud.

Single cloud applications

This is the simplest design, in which a single cloud provider is used for all cloud needs within the company, as shown in [Figure 12-3](#).

In this architecture, the company standardizes on a single cloud provider. It allows all development and operations teams to focus on the capabilities of that one provider. Knowledge can be easily shared across teams, and multiple teams can leverage a single set of best practices. And all applications can take advantage of the provider's full set of features.

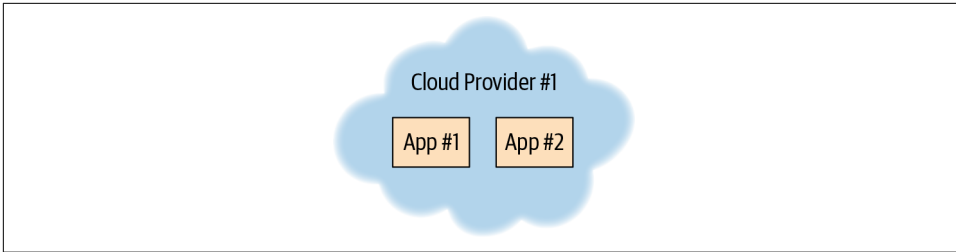


Figure 12-3. Single cloud—applications all run on a single cloud provider

From a management perspective, having a single cloud provider simplifies vendor management. Additionally, since all traffic goes through a single provider, that provider has a higher volume usage, which may allow you to negotiate better pricing and other terms.

However, this solution obviously requires a strong commitment to a single cloud provider, which can be problematic for some companies. When a problem does occur, it is much harder to negotiate a solution when you are locked into a single vendor.

This solution may be the simplest to manage and control of all the options, but it limits the flexibility of your development teams.

Which Model? Which Cloud?

So which cloud model should you use? What makes sense for your company? The final answer depends on the needs of your company and your applications.

From an application perspective, the advantages of having an application runnable on multiple cloud providers is typically outweighed by the cost and complexity associated with maintaining multi-cloud-capable applications. Therefore, in almost all cases, the joint cloud applications model shown in [Figure 12-1](#) does not make sense. If you are worried about maintaining high availability within your applications while tying them to a single vendor, consider using the high-availability solutions available from that vendor. For example, simply using multiple availability zones and multiple regions for your application running on AWS can dramatically improve your application's resiliency without incurring the costs and reduced capabilities of making your application run on multiple independent cloud providers.

When selecting a specific cloud provider, you should look at the following:

History of reliability

How reliable has the service been historically? How quickly are outages dealt with? Do outages impact the entire provider, or do they impact only specific regions at any one time (allowing you to use multiple regions to improve resiliency)?

Capabilities of availability technologies

Does the provider give you multiple availability zones and multiple independent geographic regions to allow fault isolation and failover? How independent are the zones and regions?

Availability of services

Does the cloud provider have the types and depth of services you require?

Reason for moving to the cloud

Why are you looking to move to the cloud? Is it to accelerate innovation or to help you in scaling? Whatever the reason, make sure it matches the cloud provider's capabilities.

If you choose to use multiple providers, do so because it makes sense for the given capabilities you require from those cloud providers. Don't do it to increase resiliency by using multiple providers. The reality is that the benefits are outweighed by the costs and disadvantages.

The Cloud in Summary

This chapter focused on how to use the cloud to architect scalable applications. We discussed how organizations mature in their ability to use and trust the cloud; common mistakes that organizations make when they adopt the cloud and the traps these mistakes can lead them into; and how and when to use multiple clouds in your applications.

About the Author

Lee Atchison is the senior director of cloud architecture at New Relic. For the last eight years he has helped design and build a solid service-based product architecture that scaled from startup to high-traffic public enterprise.

Lee has 33 years of industry experience, including seven years as a senior manager at Amazon. At Amazon, he led the creation of the company's first software download store, created AWS Elastic Beanstalk, and managed the migration of Amazon's retail platform from a monolith to a service-based architecture.

Lee has consulted with leading organizations on how to modernize their application architectures and transform their organizations at scale, including optimizing for cloud platforms and service-based architectures, implementing DevOps practices, and designing for high availability.

Lee is an industry expert and is widely published and often quoted in publications such as *InfoWorld*, *ComputerWorld*, *Diginomica*, *IT Brief*, *ProgrammableWeb*, *The New Stack*, *CIOReview*, *DevOps Digest*, and *DZone*. He has been a featured speaker at events across the globe from London to Sydney, Tokyo to Paris, and all over North America.

Colophon

The animal on the cover of *Architecting for Scale* is a textile cone snail (*Conus textile*). It is also known as the “cloth of gold cone” due to the unique yellow-brown and white color pattern of its shell, which usually grows to about three to four inches in length. The textile cone is found in the shallow waters of the Red Sea, off the coasts of Australia and West Africa, and in the tropical regions of the Indian and Pacific Oceans.

Like other members of the genus *Conus*, the textile cone is predatory and feeds on other snails, killing its prey by injecting them with venom from a radula, an appendage that resembles a small needle. The conotoxin used by the textile cone is extremely dangerous and can cause paralysis or death.

Their shells are sometimes sold as trinkets, but textile cones are plentiful, and their population is not threatened or endangered. Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *Wood's Illustrated Natural History*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.