O'REILLY®

DevOps and DNS

Improving Web Application Performance at the DNS Layer

Andy Still & Phil Stanhope

ORACLE + Dyn
Rethink DNS.
dyn.com/oreilly
Phil Stanhope,
VP of Technology Strategy, Oracle Dyn GBU

DevOps and DNS

Improving Web Application Performance at the DNS Layer

Andy Still and Phil Stanhope

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

DevOps and DNS
by Andy Still and Phil Stanhope

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/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Brian Anderson and Virginia Wilson
Production Editor: Kristen Brown
Copyeditor: Sonia Saruba
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

June 2017: First Edition

Revision History for the First Edition
2017-05-23: First Release

978-1-491-97867-2

[LSI]

Table of Contents

Introduction

The nature and complexity of internet-based systems that are being developed and run are ever-changing, and the business pressure is for ever more frequent changes, higher throughput, and increased reliability. Whether business to consumer or business to business, web-based systems are ever more mission critical, and the market is moving so fast that it is essential for business that new features and fixes are deployed into production as quickly as possible.

These pressures have led to changes in the way we run systems, with teams needing to be more reactive and have understanding of much wider and more frequently changing systems than ever before.

A consequence has been the evolution of cultural changes such as the DevOps movement which aims to create a dynamic environment focused on improved team integration and communication, as well as techniques that drive frequent change and improve reliability.

DNS has long been the public face of the internet, allowing users to communicate with systems using friendly text-based names rather than IP addresses. However, it has traditionally been viewed as a relatively static resource with changes that are often manual and usually kept to a minimum.

This ability to hide the underlying implementation of systems behind common names is a feature that can be taken advantage of in order to implement the type of dynamic and flexible systems that a DevOps culture provides.

In many ways, DNS could be described as the glue that holds the internet together, and the DevOps movement is taking advantage of that flexibility and reliability to produce more, reliable, and faster-changing systems.

This book aims to give an introduction to DevOps and DNS, introduces how they can be combined in a modern, internet-driven system, and discusses how the two can be used to together to improve the ability of your team to deliver regular change to your system while minimizing the risk of that change.

The objective is to give readers unfamiliar with these technologies (or their use together) a grounding in some of the modern ways that they are being combined, and provoke experimentation and further reading on the subject.

CHAPTER 1

Introduction to DevOps and the Internet

DevOps is a term that has many different and varying definitions, but at its core it is simply the creation of a culture within an organization that promotes efficiency and reliability by bridging the gap between development, QA, and operations.

Before going into any detail about how DNS relates to and can be used within a DevOps environment, it is worth taking a short time to reiterate the background and the principles that are at the core of the DevOps movement, and the impact that this movement has had on companies that have implemented it.

Background

Traditionally, software was built by one group of people, *development*, tested by another group, *QA*, and maintained by a third group, *operations*. Each of these groups was siloed and focused on different deliverables. Development focused on writing code to implement functionality, QA focused on ensuring there were no issues with the code, and operations focused on ensuring that platforms remained available.

The result was that the progression of functionality to production was often slow, and typically only a small number of large deploys per year were made. The division in responsibility often led to finger-pointing when problems arose.

Through the early 2000s, the Agile development movement made great strides forward in improving the process by which software was developed. The movement broke down barriers between business users, business analysts, development, and QA to develop deployment-ready software much more regularly and efficiently.

One of the key elements of any Agile process is the importance put on the value being created by any development effort. Crucially, that value is only ever seen once software is in production and in use by end users. This leads to the creation of a steady stream of production-ready pieces of functionality waiting to be deployed.

Automated testing and continuous integration tooling made the process of validating software as production-ready much shorter.

The DevOps movement evolved to take the same types of thinking into the operational management of systems. This includes not only technological advances such as virtualization, cloud systems, and deployment, automation that simplify the process and repeatability of deployment but also the human elements such as closer integration between development and operations teams much earlier in the development process (Figure 1-1).

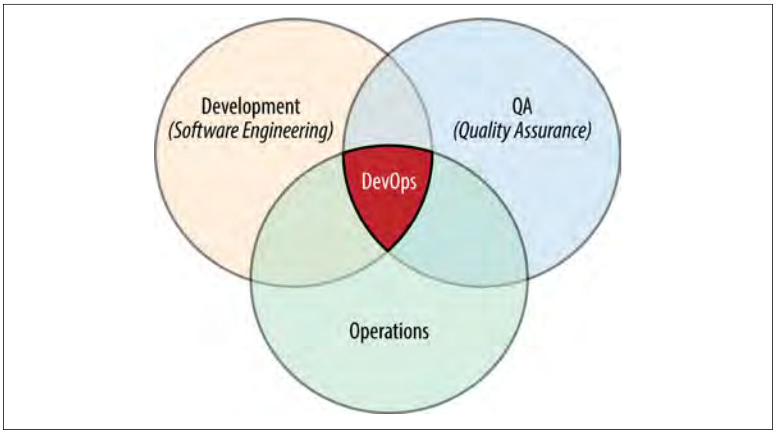

Figure 1-1. Illustrating how DevOps relates to a traditional organization structure, taken from https://en.wikipedia.org/wiki/DevOps

The DevOps Name

In 2009, John Allspaw and Paul Hammond from Flikr delivered a presentation entitled "10+ Deploys per Day: Dev and Ops Co-operation at Flikr" at O'Reilly's Velocity conference. This presentation outlined how Flikr had integrated the development and operations functions in order to streamline the release process and operational overhead. The title was soon shortened to #devops on Twitter and inspired Patrick DuBois to start the devopsdays series of conferences.

Key DevOps Principles

DevOps is much more than just the combination of development and ops. It is also a new way of thinking about ops as much more of a proactive, rather than a reactive way of managing servers. Freeing up time spent doing manual processes and dealing with issues allows operations people to spend that time creating systems that will dynamically respond to alert situations, or creating more proactive monitoring services to mitigate future risk of failure.

In many ways DevOps is a trade-off. The up-front work done to mitigate future issues is sacrificed in favor of allowing change and having systems in place to monitor and measure system behavior to quickly identify problems as they arise.

While this may sound more risky, it is worth reminding ourselves that for all the effort put in by traditional operations approaches to minimize the risk of issues arising, issues did arise fairly regularly, and the impact of resolving those issues was usually higher than in a DevOps world.

DevOps is not a job title or a set of tools or even a methodology. It is simply a culture or way of working that emphasizes certain core values. As such, there is no defined set of practices and tools that incorporate DevOps; it is more a set of principles.

The core principles have been defined in many different ways, though all definitions share the same common themes and can generally be categorized in the following three principles:

- Integration and communication
- Automation and repeatability
- Big picture thinking

Integration and Communication

This is the core principle to DevOps: breaking down walls between teams and getting people to work together as colleagues, not competitors.

Firstly, this will remove the blame culture—the approach in any situation being to blame others until proven otherwise. Shared responsibility will enable a quicker resolution as the team works together to solve a problem.

Secondly, this will get potential issues spotted earlier on and a more proactive solution design will be undertaken.

Building a cross-functional team and working toward a shared objective focused on business objectives rather than silos focused on protecting their own interests can only result in a better outcome for the business.

Automation and Repeatability

Repeatability of process is one of the core tenets of DevOps. Just as in Agile development, the rule "if something is hard to do, do it early and do it often" applies.

This means that not only should your applications be easy to deploy, your infrastructure should be easy to recreate, ideally using a fully automated process. This involves a change in mindset: from infrastructure management as a process of manually installing and configuring hardware and documenting the process, to infrastructure management being the process of managing scripts that will dynamically create and configure your infrastructure, usually within a virtualized or cloud environment.

The result is that infrastructure creation code can be managed in the same manner as other development source code, and test platforms can be created automatically as part of a test or production deployment system.

Big Picture Thinking

Developers want to solve all problems with code, DBAs want to solve all problems in the database layer, and operations wants to solve all problems with hardware. That is the nature of thinking when in a very siloed world.

By bringing the teams closer, this issue is mitigated in two ways:

1. Earlier involvement of people with specialities so that all specialities have input into a solution.
2. More knowledge sharing and appreciation of the elements of the system outside of their expertise, allowing people to understand the potential for solutions in these areas.

As systems become more complex and the options available for system expansion become more varied, this becomes ever more important. For example, when running in a cloud environment, performance issues can be addressed by either code investigation or by increasing the size of the production environment. Likewise, SaaS offering can negate the need for writing code, running infrastructure, or both.

Benefits of Using DevOps

However, the benefits of using the DevOps approach are not just theoretical. Organizations that follow DevOps principles often report improvements in both *speed* and *reliability*.

One report stated that 63% of organizations using DevOps reported that they release software more frequently, while another report stated that organizations using DevOps reported deploying 30 times more often than traditional organizations.

Similarly, 63% of companies reported seeing a better quality of software deployments. A mean time to recover investigation showed that these organizations saw 60 times fewer failures and could recover 168 times faster.

Moving to a DevOps culture also frees up operations time to focus on improvements rather than firefighting, with traditional operations reporting spending 21% more time putting out fires whereas DevOps spends 33% more time on infrastructure improvements.

DevOps and the Cloud

DevOps and cloud are two completely independent aspects of software delivery. You can deliver software using cloud platforms via traditional organizational structures and have a complete DevOps culture without using any cloud-based platforms; however, cloud platforms do make enacting the DevOps principles easier in many cases.

Cloud platforms generally blur the distinction between development and ops, as they are systems that are designed for automation and management via code-level APIs or scripting. Operational changes can be implemented by developers within application code, and operations people are writing code to manage how environments become available for use by applications.

The big picture, in this case, becomes not only the application but also the range of services offered by cloud providers that could be used by the applications, how these would be implemented, and the communication between systems.

Cloud environments have really enabled DevOps to become mainstream as the level of dynamic infrastructure creation and management needed for a true DevOps environment has become not only possible and affordable to organizations of all sizes, but actually the preferred way of working in those environments.

Automation, repeatability, creation and destruction of environments, and infrastructure as code are all elements that have been built into cloud platforms from the ground up.

Role of the Internet in Modern Systems

It may sound obvious to say that modern internet-based systems are dependent on the internet, but it is worth taking a moment to think about the actual impact this is having as systems become more complex, modular, and cloud based.

Before the internet was widely used, most systems were delivered as client/server systems over a leased line-based WAN. So everything was fully under your control; the full end-to-end connectivity and infrastructure were all dedicated to you.

However, this situation is very rare these days. Most connectivity now travels over the public internet. This is true of the connection between your users and your system, and increasingly between your system and third-party dependent services. The quality of this connectivity is largely beyond your control.

Moving into the cloud, however, gives you even less control over the connectivity in and out of your systems. When hosting systems within a data center, you can discuss and understand the nature of connectivity to the internet that are in place and use this as a deciding factor when choosing a data center. In the cloud this level of detail is rarely made public.

Despite its increased importance to people running web-based systems, the performance of the public internet is an area that is often overlooked.

Takeaways

- DevOps is an evolution of movements such as Agile development and continuous delivery designed for optimizing the throughput and reliability of complex systems.
- DevOps is not a process, set of tools, or a methodology, but a culture based around three key principles:
  - Integration and communication
  - Automation and repeatability
  - Big picture thinking
- DevOps is showing major improvements in speed and reliability for companies adopting it.
- DevOps and the cloud are two completely separate technical advances but are well suited to be used together.

CHAPTER 2

DNS Primer

In many ways, DNS is the public face of the internet. It allows friendly names such as *google.com* to be used instead of unfriendly IP addresses.

Although it is a fundamental element to successful operation of the internet as we know it, for many developers the DNS system is a black box, something they rely on but assume will always "just work."

In some ways, DNS has been a victim of its own reliability. In general, with simple systems it is possible to just ignore DNS and it will just work. As systems become more complex, however, DNS is another tool that can be used both to mitigate problems and to optimize performance and delivery of systems.

As discussed in Chapter 1, this is another example of the DevOps culture blurring the distinction between development and operations, requiring developers to think about elements that were previously entirely the domain of operations.

How Does DNS Work?

It is surprising how many developers have only a very limited or even no understanding of how the DNS process actually works. So it may be useful to give a quick, high-level overview of the process that is followed when a DNS resolution request is made.

DNS History

In the early days of the internet, names were managed via a text file (*HOSTS.TXT*) stored on the local machine that listed all the names currently in existence. A master file was managed by the Stanford Research Institute, and new entries could be requested by telephone during business hours.

To this day, all Windows machines have a *HOSTS.TXT* file that is the first point of lookup for domain name resolution.

By the early 1980s, this solution was seen as not being scalable enough, and the Domain Name System was defined in 1983.

Students at the University of Berkeley created the first implementation of a DNS system, BIND (Berkeley Internet Name Domain), which is still the most commonly DNS software on the internet today.

While there have been evolutions and enhancements of the protocol over the years, the DNS system in use today is still fundamentally the same as first defined in 1983.

At the simplest level, all the DNS system does is convert a DNS name into an IP address; however, as you'd expect there is a large degree of complexity behind the system.

Every domain that is registered creates a DNS record, usually hosted by the company that registers the domain; however, once registered, the domain name can be transferred to be hosted elsewhere. This is simply a text record that stores details about what information should be given to anyone requesting details about this domain name. This includes web-based resolution details as well as other information such as where mail servers should connect to (MX records).

In reality there are variations and optimizations of the system to improve reliability and efficiency, but the essentials of the process are as follows.

When you type an address into a web browser:

- A check is made to see if the details of that name are known locally, e.g., if the browser has made a previous request from that same domain name or there is an entry in the local DNS registry (e.g., *hosts.txt* on Windows).
- If no local record is found, a request is sent to your local DNS server. This could be running locally on your machine or on an office network, but most commonly it is provided by the ISP that supplies your internet connection.
- The local DNS server again checks if it already has the details of the name being requested. If there is no cached record, then the DNS server needs to locate the details of the name server that hosts the domain record for the address you are trying to resolve (the authoritative domain name server).
- To do this the DNS server breaks the name down into its different sections, starting from the righthand side of the domain name. For example, for *www.google.com*, this would be split into com, google, and www. The section after the final . of the domain name (in this case, com) is known as the top-level domain (TLD). A root name server is connected to find details of the server that holds the domain record for the TLD.
- The DNS server will make a request to the TLD name servers asking for details of the name servers that contain the details of the next section of the domain name (in this example, google). The DNS server then makes a request to the name server that holds the details for *google.com*. This name server may then return details of another name server that holds the records for *www.google.com* or, more likely at this point, will return the address associated with *www.google.com*.
- The address returned by the remote name server can be an IP address or it could be another domain name, known as a CNAME; for example, *www.google.com* may return a reference to *cdn-us.aa1.google-us.com*.
- If a CNAME is returned, the DNS server then repeats the process with the CNAME until an IP address is resolved.

An example of a recursive DNS process is shown in Figure 2-1.

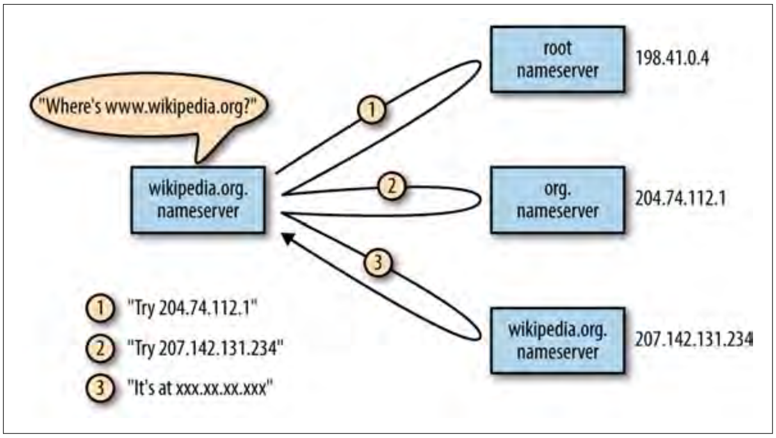

Figure 2-1. Illustration of how a recursive DNS works in practice, taken from http://bit.ly/2rsljNx

NOTE

NSLookup

All domain name resolution information is publicly available. Using the NSLookup tool that is available on the command line of most computers, you can directly query the DNS system and find all the details of any DNS registration. NSLookup allows you to query using your default DNS server and also by specifying a different DNS server (e.g., Google's public DNS server 8.8.8.8) to validate that your local DNS is returning the same details as other people are seeing.

There are also a number of websites that will complete an NSLookup request for you.

Potential of DNS

As well as providing a more friendly and memorable address for a website, using DNS names also gives a number of other advantages:

Changing backend systems
: Keeping a consistent DNS name allows you to re-point to a different backend system at any point without users needing to be aware of the change. This could be a major change such as a move to a new data center, or short-term changes such as pointing to a DR system or to an updated version of the system as part of a deployment process.

Multiple backend systems
: DNS names can be used to obscure that there are actually multiple systems delivering the same system. This could be done on a very simple level where each user is given a selection from a list of available systems, or it could be done more intelligently based on things such as geographical location.

These advantages provide a lot of benefits when looking at the world in a dynamic DevOps or cloud-based system. In this world the physical, IP-based servers that are actually delivering systems are ever changing as systems are automatically created and destroyed on demand, meaning that the easiest way to address systems is by taking advantage of the flexibility that DNS provides.

Considerations and Risks

Of course, there are also downsides to using DNS as a proactive element of your DevOps culture. As with any other tooling/methodology choices, these need to assessed in terms of the benefit/risk tradeoff they provide.

Speed of Change

Speed of change is an element that is often raised as a reason not to use DNS for any change that you want to be as close to instant as possible.

As mentioned above, the amount of time a DNS record is cached is determined by the TTL. This is within your control.

What is TTL?

TTL, or *time to live*, is the element of a DNS record that tells the requester how long the record is valid.

In other words, if the TTL for your DNS record is set to 24 hours, once a browser has resolved that DNS record, it will continue to use that same value for the next 24 hours regardless of whether you've updated the details.

If the TTL is set too high, then DNS cannot be used as a failover method, as the change will take too long to take effect with any existing users. Setting a very low TTL, however, adds extra overhead, as DNS lookups have to happen much more regularly, which adds to the page load time for a user and increases the stress on the DNS servers.

The default setting for TTL values was traditionally 24 hours, and it was usual to have to wait over a day for the impact of DNS changes to take effect.

By setting a low TTL, you can specify that you don't want the record to cache for long. However, there are two potential gotchas with this:

1. Having a low TTL increases the amount of DNS lookups that are happening. This impacts performance for the end user as there is overhead associated with that request and increased overhead on your DNS provider. If your DNS provider cannot cope with that, then it can affect the speed and reliability of DNS resolution. If taking this approach, it is essential that you select a resilient DNS provider that is equipped to handle high throughput of requests.
2. Some DNS resolvers do not honor TTL values below 30 seconds. In this case, even if you set a TTL below 30 seconds, the resolution would remain cached for 30 seconds.

On top of the TTL, there is also some additional time to complete the other actions associated with DNS updates:

- Time taken to accept the update and update the record on the central Authoritative server
- Time taken to distribute that update to all other distributed Authoritative servers

These values will be determined by the infrastructure and systems used by your DNS provider, and it is worth analyzing them to determine how long updates take to take effect. If longer than expected, then trial other providers to see if they can make changes more quickly.

Another Point of Failure

Others point to relying on DNS as being another point of failure, another element that has to be managed and therefore can go wrong. It is also pointed out that if IP addresses are used, they will carry on working even if the DNS system fails.

This is true; however, in the extremely unlikely situation where the DNS system was to fail, the entire internet as we use it today would be unusable. DNS is the ubiquitous glue that is the basis for most internet-based communications.

For this reason, DNS is a highly distributed system that is fault tolerant (but like any other system it is not infallible) and built that way from the ground up.

For most usages, the flexibility gained by using DNS outweighs the additional overhead and risk of adding that potential point of failure.

Takeaways

- DNS is a fundamental system that keeps the internet operating as expected. It is worth taking the time to understand its complexities.
- DNS is a distributed, recursive system that queries domain name servers until it finds the authoritative record for the address it needs to resolve.
- Using DNS creates more flexible systems that allow for changing backend systems and systems hosted in multiple locations.
- There are risks associated with using DNS as the basis for system management, including the speed of change, and the management and additional risk of failure. These should be assessed against the benefits for your system.

CHAPTER 3

Preparing for Implementing a DNS-Based DevOps Approach

When starting down the road of moving toward a DevOps-based culture that takes advantage of the DNS-based solutions recommended below, there a few areas that need consideration.

This chapter will outline some of the building blocks that are needed.

Selecting a DNS Provider

If you want to move ahead with using DNS as an integral element of your DevOps approach, then the selection of an appropriate DNS partner is a key decision. There are many providers out there that offer a wide range of solutions.

Hosting partners or cloud providers will often offer a DNS service as part of their offering. It is important to remember that you do not need to host your DNS with them; while there are advantages to centralizing management in one area, there are also some advantages to maintaining an independent DNS provision. Most importantly is that it gives you freedom to move to different providers in future or to spread systems across multiple providers. However, regardless of the independence of the provider, it is crucial that the provider you choose has the level of expertise in the DNS arena to provide the level of support necessary for a mission-critical platform. For many providers, DNS is an add-on service that is not given the importance it deserves.

It is also essential that when evaluating potential partners or evaluating whether to use your existing cloud provider, the following important capabilities are considered:

- Performance
- Dynamic control

Performance

Using DNS in the dynamic, often-changing manner that is necessary will require reduction of TTL values to much lower values (typically less than 30 seconds). This means you are more dependent on your DNS provider, and therefore the performance of your DNS provider becomes even more important.

DNS performance needs to be validated against a number of use cases, as discussed below.

Low latency

Regardless of your TTL settings, it is important that your DNS provider offer a low-latency network, as all users have to resolve your DNS records before they can access any of your content. However, this becomes more important the more often users are having to re-resolve your records; with a TTL set at below 30 seconds, users will be resolving the DNS multiple times per visit.

If the latency of your DNS is too high, this will block delivery of any content, which would be noticeable to users. As it is the first request, there is no way to code around or hide this delay.

Domain sharding

Many sites use domain sharding, that is, serving content from multiple subdomains in order to improve performance.

In this case, the overhead of DNS is multiplied as each subdomain will need to be resolved, though well-constructed pages can mitigate this by ensuring DNS requests are executed concurrently or asynchronously.

When evaluating the latency of DNS resolution, remember to consider the latency that all your users experience regardless of location.

If you have a worldwide user base, then ensure that your DNS records are geographically distributed and are being resolved by servers as geographically close as possible.

High capacity and resilience

Decreasing TTL values means that you will be asking your DNS provider to do a lot more resolutions than previously and are relying on that provider to always be available to serve updated details.

Ensure that your provider has sufficient capacity to be able to handle this increased amount of traffic without affecting the speed of resolution.

Also, ensure that the provider has sufficient fault tolerance built into their networks to be able to cope with failure within their systems or networking issues within the wider internet that may make elements of their DNS network unavailable.

Speed of propagation

Remember, when relying on DNS for making dynamic changes, the time taken for updates to take effect is:

```
Time taken to update + time taken to propagate + TTL
```

When choosing a DNS provider, make sure that any updates that you make are not only updated quickly, but are also propagated through the rest of the distributed authoritative servers as quickly as possible.

When validating this, ensure that you consider the speed of propagation to all areas where you have a user base.

Dynamic Control

Any DNS provider you use in this model must offer dynamic control of your DNS records.

Many traditional services only accepted DNS changes via email or telephone requests, with these requests being manually implemented at some point after the request was received. Obviously this is not viable for the sort of reactive change that is needed. Even a web interface allowing direct updates is not suitable.

All changes need to be able to be implemented with no manual intervention.

It is essential that the DNS provider you choose has a comprehensive API that allows changes to be made regularly and instantly without any need for human interaction.

This enables you to integrate DNS management into your automated and scripted deployment and update methodology.

Implementing a Monitoring Solution

A key part of a cohesive DevOps approach is ensuring that sufficient data is collected throughout the process to allow everyone involved to have a much more detailed understanding of the current state of the system. This is especially important as we are giving away more and more control of elements of the infrastructure that makes up the system.

Having this depth of knowledge allows for dynamic systems to be put in place to handle resolution of issues quickly and effectively. Again, this focuses on the ability to quickly and reliably implement change in response to problems.

The DevOps culture, on top of the move to cloud where much more of the infrastructure is out of your control, means that monitoring has to be extended to much more than traditional infrastructure monitoring.

In the DevOps and cloud world, monitoring has to be focused on understanding the holistic view of the performance of your application. This means taking monitoring to the next level of detail; typically this includes three additional views:

- RUM and EUM
- APM
- IPM

RUM and EUM

Real User Monitoring (RUM) and End User Monitoring (EUM) are focused on what is ultimately the most important metric of all. What is the experience that actual users are seeing?

RUM focuses on the experience of actual users. Typically this is done by including a beacon within the system that sends back data to a central server outlining details of how the system is working. The most common implementation of this is injecting some JavaScript into the content of a web page that monitors performance metrics.

RUM is valuable because it captures what is actually happening to users; it is not dependent on a set of predefined measures that are being proactively measured. If issues are raised by users, then analysis can be completed to first determine whether the issues are affecting a wider group of users, and then to try and drill down to the cause of the issue.

RUM also allows you to determine if there is a pattern to the users affected; e.g., are they using a similar browser or type of connectivity, or are they from one geographic location?

EUM is similar but is based on a set of synthetically executed, repeatable tests carried out from a "clean room" location. This allows you to assess the results of tests without worrying about the results being affected by unknown conditions.

A good monitoring solution includes elements of EUM and RUM, as they both add value in different situations.

RUM is valuable in that it reflects what actual users are experiencing and will flag issues beyond the range anticipated when defining test plans. RUM executes continuous testing against your complete application, albeit in an unscientific manner.

EUM adds value in that it is a more scientific approach to testing; you can be confident when failure occurs that no other factors will be changed. EUM also allows you to proactively identify issues without users experiencing them first (hopefully resolving the issue before it affects users).

APM

Application Performance Monitoring (APM) is a monitoring technology that sits within your application and gathers core metrics about what is going on under the hood of your application.

These metrics will usually go down to a granular detail about how your application is behaving (such as method execution times, SQL query execution times), as well as the overhead of communication with external systems.

APM allows comprehensive assessment of what your application is actually doing at a code level rather than the impact that it has on external items such as servers or user experience. This is invaluable when assessing the root cause of any issues or when completing performance optimization.

Many modern APM solutions will integrate with RUM and EUM systems to get a complete end-to-end breakdown of user interaction with your system.

IPM

Internet Performance Management (IPM) is the gap that is often left in a monitoring solution. This looks at the monitoring and analytics in the performance of the internet between users and your application.

Note that this may be a dedicated tool or may be API-based data feeds from different sources that are pulled into dashboards. Overall, the aim is to give an operational awareness for the way in which your applications and the services they interact with are available through the internet.

Applications are increasingly reliant on the performance of the public internet, so it is ever more important that we have an understanding of any issues that may arise. This applies not only to the applications you run yourself, but also to the other internet-based applications that your applications rely on.

Typically this will be routing issues that may be temporarily or permanently in place between elements of your user base and your systems. The internet is a volatile environment, and these types of issues can arise at any point.

IPM monitoring allows you to become aware and react to these issues, for example, by using geolocation-based DNS to route those users to an alternative location.

Takeaways

- Selecting a DNS partner is especially important. Make sure you look beyond your hosting or cloud provider.
- Validate that your DNS partner provides a high performance solution that meets your expectations in:
  - — Speed of resolution
  - — Speed of propagation
  - — Capacity and resilience
  - — Provision of a dynamic API
- Effective DevOps provision will require effective end-to-end monitoring to enable you to understand and react to issues as they arrive.
- Any monitoring solution should include:
  - — RUM and EUM
  - — APM
  - — IPM

CHAPTER 4

Managing DNS in a DevOps Culture

DNS can be an important element when managing systems within a DevOps culture, as outlined in this chapter.

However, before this can be reliably implemented, your DNS estate should be managed in an appropriate way to allow the amount of flexibility and dynamic change that will be required.

Traditional DNS Management

Traditionally, DNS was seen as being fairly static, with changes being done only occasionally. DNS changes were done by manually editing a text file on a server, for those running their own DNS servers, or more commonly by submitting a request to the company that managed the DNS records. This change would then take time to propagate through the internet, depending on the TTL set for the record.

This approach was acceptable in a traditional environment of relatively static resources where change was to be kept to a minimum.

However, when implementing a DevOps approach, it is necessary that DNS management is approached in a much more dynamic manner, accepting that change is not only inevitable but an essential element in managing a DNS-driven DevOps implementation.

This includes a change in mindset toward DNS changes.

Remove the Fear of DNS Changes

DNS carries with it an extremely high level of risk. After all, if a mistake is made, it can wipe out connectivity to your site for an extended period, especially in the world where TTL values were typically set to 24 hours.

When combined with the traditionally quite complex or cumbersome way that DNS was managed, it lead to a fear of making DNS changes.

DNS was often seen as something that just shouldn't be touched unless absolutely necessary.

However, this is contrary to the DevOps mentality of "if something is hard to do, do it early and do it often," making change easy by making it automated and repeatable.

With this in mind, it is essential that when applying a DevOps mentality to your organization, you start to bring your DNS provision under control and make management of it well practiced.

To do this, it is essential that you use a DNS provider that provides dynamic API-based control, allowing changes to be made easily as part of scripting process.

Changes to DNS can therefore be automated via the API and executed on demand.

DNS as Code

One of the most common elements implemented in a DevOps culture is a change to viewing "infrastructure as code." That is, not seeing infrastructure as physical devices with state that need to be built and configured, but instead seeing infrastructure as simply a set of scripts that have to be executed in order to recreate the system.

This code can then be managed, evolved, and tested in the same way as application code. Your infrastructure becomes simply an extension of your application, managed and maintained in the same manner. Therefore the integration between dev and ops becomes even closer.

If all DNS changes can be managed via a dynamic API, then it is a logical next step to start extending your infrastructure as code to include your full DNS records, tracking changes in source control.

This will easily allow the full recreation of DNS records, in the event of a loss or a migration to an alternative supplier.

Takeaways

- To take advantage of DNS in a DevOps world, you need to start managing your DNS records inline with a DevOps approach.
- Traditionally, DNS was seen as a very static environment that was manually managed; this needs to change.
- Remove the fear of DNS changes by implementing the "if something is hard to do, do it often" mentality by using DNS providers and tools that remove this difficulty.
- Choose a DNS provider that provides the level of dynamic control that will be needed.
- Treat your DNS records as code, creating scripts that can dynamically recreate your DNS state at any point.

CHAPTER 5

Using DNS in Your DevOps Approach

We have discussed the benefits of DNS and DevOps independently, now let's drill down into some more detail about how DNS can be employed specifically when implementing a DevOps culture.

For all of these there are alternative ways to deliver the same results, but the built-in fault tolerance and independence from your other infrastructure mean that it is worth considering using DNS.

Use DNS to Streamline Deployment Pipelines

One of the primary objectives in moving to a DevOps culture is to increase the the cadence of deployments to production, ideally moving much closer to a Continuous Delivery system.

NOTE

Continuous Delivery

Continuous Delivery is a term coined to refer to the objective of an Agile development system to ensure that changes would always be ready to go to production after completion. Originally coined to mean continually deliverable, it has been adopted within the DevOps world to refer to the practice of being so confident in the automated validation and deployment pipeline that any changes get pushed directly to production without any human intervention.

This speed of deployment involves several changes:

- Trust in the validity of automation of the testing and deployment process
- Repeatability of the deployment process
- Reliable way of validating success of deployment
- Simple and reliable rollback process

There are several ways that DNS can be employed to deliver these objectives.

Blue/Green Deployment

Traditionally, deployment of systems was done by installing new software over the top of the old software on the same hardware. Rollback was then completed (if necessary) by reinstalling the previous version or in worst case scenarios by restoring the entire server from a backup.

The issue with this approach is that over time, the state of the target machine will become less and less like that of the development and test machines. Development and test machines by their very nature are subject to more experimental and failed builds than production machines, all of which will leave artifacts such as updated system files or elements of the system not properly rolled back. This all leads to variants in behavior between what is seen in testing and in production.

What is preferable is that each release of the system is deployed onto a clean machine with a known base state. The infrastructure as code and automation defined above, combined with virtualization or cloud platforms make this a reality; with every new round of testing, a completely new test environment can be created and the previous one destroyed.

This pattern can also be applied to deployment. At deployment time a replacement environment is created alongside the existing environment. Traffic is then rerouted to use the new environment. If the system allows, this can be done gradually, allowing small amounts of traffic to use the new system while validation takes place. In the event of failure, traffic can just be redirected back to the old environment.

This process is referred to as blue/green deployment (or sometimes green/gray) to indicate that you have a live environment (green) and a production ready but offline (blue) environment.

There are several ways to manage the switch between the two platforms, but dynamic DNS is an excellent means as it is simple and requires no additional hardware within the environment.

After the new environment is created, the DNS records are switched to point to it instead of the old environment. Typically, this would be done as part of an automated process via the API provided by the DNS provider.

Staged Rollout

A reliable way to ensure that deployments are not introducing issues is to stage the rollout. This involves releasing the new version to a subset of users and monitoring those users to determine whether that release should be rolled out to the rest of the users (or to progressively larger groups of users).

Some companies use this method as a way of user testing new features to determine whether those features are popular with users. Known as A/B testing or multivariant testing, this is an increasingly popular way of determining whether or not features should be released. Facebook, for example, has most of the next six months worth of new features already going through testing with subsets of users to determine whether they should be released to the full user population.

There are network-based methods for managing a staged rollout of a system; for example, load balancers can be configured to route traffic to different systems.

However, DNS is a good solution to the problem as it removes the point of failure from within your network and allows for easier distribution across multiple data centers if required.

This can be managed in two ways. In the simplest example, your DNS provider could just be configured to send a set percentage of users to each system. For example, 10% could be sent to the new system, and the remainder of DNS queries will still resolve to the old system. Note that in this case it is important that your system be aware that individuals were previously on the new system and handle that appropriately.

More sophisticated DNS systems can handle this in slightly more elegant ways, such as segregating users by geographical area rather than at random.

NOTE

Geographical region versus network topological region

When talking about resolving DNS by geographical region, it is actually based on internet network topology rather than geographical region. That is, it is based on the way that a connection is routed through the internet rather than its actual physical location.

For simplicity, throughout this book we will talk about routing traffic based on geographical location.

Splitting traffic in this way has several benefits:

- You can make a conscious decision about the users that you want to try out new features with, e.g., users who are less of a business risk or who have a particular relevance to the feature being rolled out.
- You can manage the support issues more proactively, having more awareness about which versions people are using if they contact you with issues.
- Multiple versions can be tested concurrently with different geographical regions.

Use DNS to Maximize Availability

Availability is an essential aspect of any internet-based system, with downtime often resulting in short-term loss of income and longer-term loss of users. In this section, we will look at some of the ways that DNS can be used to improve availability of systems.

DNS as a Load Balancer Replacement

Load balancers (or application delivery controllers) are devices that route traffic to multiple servers. This can be done via simple methodologies such as just using a round-robin approach to send each request to the next server in the list, more complex ones such as the server that is currently responding quickest, or even intelligent routing based on pattern matching on the URL being requested or other elements of the header.

Load balancers, however, have a number of drawbacks:

- They are usually devices that sit within your infrastructure and therefore the traffic that is routed to them is all coming into a single point within your infrastructure.
- They have limited capacity.
- They can often only route traffic over a local LAN network.

Modern dynamic DNS can be used as a replacement for load balancers, offering more dynamic, global load-balancing solutions with the ability to balance traffic based on different criteria.

Using DNS to load-balance traffic has several advantages:

- Traffic can be balanced across multiple data centers without needing to go through a central load-balancing location.
- DNS is well suited for location-based load balancing. As long as your DNS provider offers geolocation-based resolution, then this is the ideal way to route traffic to the closest location.

DNS-based load balancing is not as feature rich as most of the load-balancing solutions available, but if you have straightforward requirements, it is an option worth considering as it is low overhead and easily configurable.

Integration with Monitoring and Alerting

DevOps very much involves a mindset shift, especially for people from a traditional operations background.

The traditional operations approach was to look for consistency of platform and therefore minimize change. The focus in that case was on planning for change and doing as much up-front mitigation as possible.

Moving to a DevOps world means moving to a system that accepts that change will constantly be happening and that errors will occur. The best way of dealing with this is by having a comprehensive approach to system monitoring: being aware at all times of the state of the system and how to deal with any failures that are seen, ideally in an automated fashion.

DNS can be integrated into this approach to ensure that when issues are seen, appropriate action is taken to remove the problem area from the public system or to reroute users to an alternative implementation or DR version of the system.

DNS sits remotely from your underlying architecture and becomes a tool that can be used for addressing issues as they arise, DNS provides the capacity for a range of actions to be taken, depending on the nature of monitoring being undertaken.

Mitigating Performance Degradations

Because of the unique position that DNS employs, it can be used as an effective tool for handling performance issues at many different levels.

DNS sits independently from the rest of your system, allowing changes to made throughout your system, independent from your hosting or cloud provider.

Infrastructure-Level Performance Issues

The most familiar sort of performance issues are those seen within your own infrastructure. These could be caused by capacity issues within your application, or by failing or misconfigured hardware, either directly within your infrastructure or within the local networking infrastructure.

As discussed, cloud provision increasingly takes resolution of these latter issues out of your hands.

However, what cloud takes away with one hand it gives back with the other, allowing rapid recreation and upscaling of systems.

DNS offers an ideal methodology for managing resolution of these situations. Whether the resolution is expanding capacity or recreating the system elsewhere, the DNS records can be quickly updated to point to the improved solution.

Network-Level Performance Issues

As the use and reliance on the public internet increase, so do network-level, performance-related issues. Whether these are internet-level routing issues or failed peering issues associated with the data center you are using, these can have dramatic effects on the performance as well as availability of your systems.

However, as with the issues in the section above, DNS offers an ideal solution for dealing with these issues.

Whether the decision is to recreate systems in a different cloud region or to switch over to a backup or DR system, DNS records can allow for a dynamic switchover to the replacement system.

Geographic Performance Issues

An additional area where DNS can be used to mitigate performance issues is where issues are seen in performance for specific geographic locations. This could be illustrated in data coming back from your RUM systems or in issues identified in your IPM system.

If your DNS provider offers geolocation, then your DNS records can be updated to point to a better-performing version of your system. This can be either temporary or permanent, depending on the nature of the issue.

DNS as a Means of Cost Reduction

The dynamic nature of many modern systems, particularly cloud-based systems, means that systems can be created or scaled up to meet short-term demand. This will ensure that the system in place is as cost effective as possible, only using the resources needed at that time.

Typically this will be around one of the following criteria:

- Short-term predictable demand, for example, to support a promotion, product launch, or seasonal demand.
- Regular predictable demand, for example, to deal with a busy period every day or other known usage pattern.
- Unanticipated spike events, such as those triggered by TV or social media mention of your system.

Any of these can also be managed geographically, for example, spinning up a region-specific version of your system during peak periods and then sending all users to a central system during quiet periods.

For all of these situations DNS can be used as the means to ensure that users are being directed to the appropriate place. Integrating

DNS changes with the same system used to scale up and down the systems ensures that all changes are dynamic and seamless.

DNS as a Means of Service Discovery

Applications are becoming more complex and increasingly are based on the aggregation of multiple services, both controlled by yourself and third parties. Likewise it is common practice now to offer access to your systems via an API.

It would be impractical to provide access to these services via fixed IP addresses:

- There would be no provision for moving the system to alternative hardware or location in the future.
- Scaling the system, especially to geographically diverse locations, would be more difficult.

Putting services behind DNS entries allows these services to be dynamically provisioned and managed without needing to inform the end users of any changes.

This is the way that all cloud-based services are provided; access is given via a URL that hides a dynamically scaling provision.

In this manner, DNS really is becoming the glue that holds the internet together as the amount of DNS-based service provision is constantly growing.

Takeaways

There are a range of ways to take advantage of DNS when moving to a DevOps culture:

- To streamline your development process in ways such as blue/green deployment or staged rollouts.
- To improve availability such as by using DNS-based load balancing or by integrating DNS management with ongoing monitoring solutions.
- To mitigate performance issues at application, network, or internet layer.
- As a means of cost optimization or reduction.
- As the basis of a service discovery and management protocol.

CHAPTER 6

DNS and DevOps: A Real-World Example

The final chapter of this book will illustrate how we use DNS within the DevOps culture that we have at my own company, Intechnica, to ensure that we are optimizing deployments and minimizing risk when running our TrafficDefender product.

TrafficDefender is a feature-rich traffic management system that sits in front of websites, routing traffic back to the origin servers. This is a completely cloud-hosted service.

DNS plays a core part in how we manage, distribute, and mitigate risk with the product.

All this is made possible because we use a DNS provider that has an API that allows for frequent changes to be made, and those changes are very quickly propagated through the internet.

Operations Culture at Intechnica

We have a DevOps culture. Our mindset is focused on getting small changes out often, and as such, all infrastructure creation and deployment needs to be fully automated and repeatable. As everything is cloud-based, this approach is much easier.

The rate of change, as well as the mission-critical nature of the system, mean that we have to have a large body of active monitoring. However, seeing as we run everything in the cloud, we have only minimal control over the infrastructure and therefore are reliant on the monitoring data to indicate when problems arise and then react to them as needed.

DNS as a Means of Distribution

When users sign up to our product, they are given a URL as an endpoint. This is their only means of accessing the product. We retain complete freedom to change or update the infrastructure that is running the service without the clients being aware of or impacted by the change, simply by updating the DNS record.

This is the same approach taken by most cloud-based services and is a very valuable means of control, allowing a changing, dynamic system to be used with, as far as the client using the service is concerned, a completely static endpoint.

DNS as a Means of Deployment

Because we are capable of repointing the client's endpoint quickly, we can use DNS as our means of deployment of new versions.

When deploying an update we use the blue/green approach described earlier. A replacement version of the system is created—in this case a completely new version of the environment—and then the DNS is updated to point users to the new system. The old version remains in existence, but inactive.

As mentioned above, there is potential to gradually migrate traffic over, running the systems in parallel for a time while correctness of the new system is validated. For technical reasons this approach won't work for us and we need to migrate all traffic simultaneously.

After migration, the previous environment is left in place until we are confident that no issues have been introduced that would necessitate rollback.

NOTE

DevOps doesn't mean no planning or testing

It is worth noting that, just like Agile development doesn't mean no planning, DevOps doesn't mean no upfront planning or testing. It doesn't mean just going ahead and arbitrarily making changes to production.

DevOps is equally as concerned with avoiding failure as traditional ops, it is just approaching it in a different way.

Before reaching production, the new versions have been through extensive testing, both manual and automated, to the point that we are comfortable with them going into production. The DevOps approach makes that transition to production much quicker and more efficient, and the measurement and monitoring in place make it much easier to detect and determine the root cause of any issues that may arise.

Though we keep the previous environment as a failsafe to roll back to, we have never had to exercise that option. Any changes seen after deploys have been sufficiently minor, and the process for deploying updates so simple, that it has always been easier to fail forward.

DNS as a Means of Optimizing Availability

Because the system we provide is a mission-critical system, it is important that high levels of availability are maintained. This means that geographically distributed systems are essential. We use DNS to distribute this load across multiple cloud availability zones.

Our monitoring solutions are configured to detect any issues both within the system and connectivity issues from outside the system, and to be able to dynamically react to any failure. In the case of failure, a replacement environment can be created in an alternative location, and the DNS records are dynamically updated to point to this new environment.

However, this is not a foolproof system as there can be issues that affect all locations within a cloud region. In this case our DNS records can be dynamically configured to redirect traffic to an alternative region.

DNS as a Means of Managing Failure

As mentioned above, this system sits in front of websites and handles any traffic that is routed to that website. If it was to fail, it would take that site offline. Obviously this is something we want to avoid.

In the event of complete disaster, we use DNS as the final failsafe. If our system is completely down, we can modify the DNS for the client to bypass our system and route traffic directly to the clients servers.

Takeaways

DNS is a core part of the DevOps culture at Intechnica, being used:

- As a means of distribution access to the system, allowing for scaling and migration of the product without need for change by the client.
- As a core part of the deployment process.
- As a way of optimizing availability.
- As the final level means of mitigating failure, using DNS to potentially bypass the system entirely.

CHAPTER 7

Conclusion

There is no doubt that DevOps is a buzzword at the moment, but like many buzzwords there is a solid foundation behind it. When implemented well, a DevOps culture can be beneficial to a business, both in terms of increased throughput of change and increased reliability.

DevOps is about a change in focus: from minimization of change to repeatability of creation, from documentation to automation, static to dynamic platforms. It is about focusing on change and ensuring the measurements are in place to be aware of the impact of that change.

DevOps delivers a world where the focus is on flexibility and responsiveness, where systems are automated and repeatable so that they can be created and destroyed as needed.

In this sort of world, DNS becomes the glue that enables the levels of flexibility needed, allowing improved deployment capabilities, improved availability, improved cost optimization, and reduced performance issues.

As such, you need an effective DNS provider to ensure that you can have the levels of performance as well as the speed of update that are essential for running the flexible and responsive system that is needed.

Most importantly you need to ensure that you can manage your DNS programmatically and make regular, automated changes that are immediately effective.

Once you have these elements in place, you can use DNS as a very effective tool to deliver the benefits you aim to achieve by moving toward a DevOps culture.

About the Authors

Andy Still has worked in the web industry since 1998, leading development on some of the highest traffic sites in the UK. After 10 years in the development space, Andy cofounded Intechnica, a vendor-independent IT performance consultancy to focus on helping companies improve performance of their IT systems, particularly websites. Andy focuses on improving the integration of performance into every stage of the development cycle with a particular interest in the integration of performance into the CI process

Andy is one of the organizers of the Web Performance Group North UK and Amazon Web Services NW UK User Group, blogs regularly at Internet Performance Expert and Performance Patterns, and started the programming initiative Progvember.

Phil Stanhope is the VP of Technology Strategy at Oracle. An entrepreneurial technology executive who has operated at the intersection of business and technology for the past 25 years, he has consistently demonstrated success developing and executing strategies to create new products, leverage emergent technologies to bring new life to existing product lines, and oversee service delivery and operations.