Coding In the Cloud

There are three "gotchas" associated with deploying applications into a load balanced environment

by Lori MacVittie

While there's a lot of talk about cloud computing these days — why you should use it, why you shouldn't, and so on — there's little discussion on the ramifications of cloud computing and really any on-demand infrastructure on application development. So before you jump into the cloud with both feet, it's important to understand how an on-demand environment like cloud computing affects the way applications execute. There are some gotchas that can come back to haunt you after the application is deployed.

The Impact of On-Demand

Everyone understands that on-demand means elastic; it's the ability of an environment to expand and contract in real-time so that an application is always available. But how that is achieved has an impact on the behavior of applications deployed within such environments.

What on-demand means in terms of infrastructure architecture is that there is some form of application delivery or load-balancing service sitting between users and applications. That service acts as a proxy; it accepts requests from users and determines which instance of an application should respond. Some form of load balancing or traffic management service is necessary to distribute requests across instances of an application running inside the cloud computing environment because the client cannot know when it makes a request where the application is actually running. A proxy allows all users to make requests to a host regardless of how many instances of an application are actually executing internally or where they might be.

The introduction of an application delivery or load-balancing service into the architecture is often the source of much frustration for developers and network administrators alike. This is because applications deployed into load-balanced environments like cloud computing often break with no obvious reason. Requests are sent and responses received as expected, but the application either loses or is unable to retrieve data that it had before deployment.

There are three "gotchas" associated with deploying applications into a load-balanced environment; one is specific to cloud computing or on-demand environments while two are problematic for general load-balanced environments regardless of where they are located. Being aware of these gotchas during development lets you address the potential problems up-front so applications don't break once they are deployed.

Gotcha #1: Session Persistence

Applications often use sessions to store state and user-specific data needed during the execution of an application. Sessions are generally stored on an application or web server in memory, especially when applications are written using a scripting language like PHP. When there is only one instance of an application running, this is not a problem. When there are multiple instances of an application running in a load balanced or cloud computing environment, this can be a huge problem.

Sessions are generally not shared (mirrored) between application or web servers. In an on-demand or load-balanced environment, the first time a client makes a request it is directed to a server. A session is created on that server for that user, and the application may rely on information stored in that session to execute correctly.

The next time a client makes a request, the load-balancing service may send the request to a different server, based on how it is configured to choose servers. This means the session that was created on the first server does not exist and if the application relied on data in that session, the application breaks. This scenario is frustrating because sometimes the client is directed to the server on which the session resides, and the application works as expected, and sometimes the client is directed to a different server with no knowledge of the session, and the application doesn't work. This makes troubleshooting difficult because it's often not obvious what the root cause of the problem may be.

Solutions

- Do not rely on in-memory sessions for data upon which an application needs to execute correctly. Use a database-backed session that can be shared across application or web servers, or use some other mechanism for storing session-specific data necessary for the application. If you're planning on deploying in a virtual cloud environment, you have control over this. If you're planning on deploying in a platform-based cloud environment, you may not have control. Ask before you begin developing so you can decide how best to proceed.
- Use "sticky connections" or "persistence" in the networking vernacular. Before you begin developing applications, determine whether the load-balancing service supports persistence-based load balancing. Persistence-based load balancing uses session data, usually the auto-generated session ID, to ensure that requests from clients are always sent to the same server so the session is always available. If persistence-based load balancing is available, you can leverage it and develop without concern. If it does not, you'll need to choose another option to ensure your application works as expected in that environment.
- Use session mirroring, if available. Some solutions are capable of replicating sessions across a cluster of application or web servers. This is often accomplished by using a database, but not always. Session mirroring ensures that no matter what server the client is directed to that their session exists. Be careful of the costs associated with this solution, as session mirroring increases the amount of memory you'll need for each instance and may drive costs up if the provider charges on a memory utilization basis.

Gotcha #2: Session Life

Closely related to the issue of session persistence is session life. In many cases, sessions stored in memory stay in memory for long periods of time, which makes users happy because if they are interrupted for some reason they can return to the application and pick up where they left off.

This is a potential issue for applications when deployed in the cloud because it's not guaranteed that the server instance on which the session was created will still be online when a user returns to the application. It is possible that sessions can effectively be "lost" during relatively short periods of time due to a server instance becoming idle and subsequently being deprovisioned.

This gotcha is highly dependent on the cloud computing provider's definition of "idle" and how it determines that an application instance should be deprovisioned. If it is based on current connections and idle time, losing sessions is likely. If the provider's provisioning services can determine that sessions are still active, thereby determining idle time based on this information, then this scenario is not likely to occur.

Solutions

- As with session persistence, you can use session mirroring or database-backed sessions to avoid losing a session when an instance goes idle for too long.
- Work with the cloud computing provider to determine whether you can set the parameters for what is considered "idle" and what is not. You may be able to idle the instance for the duration of a session, essentially using session-based quiescence rather than connection-based quiescence to determine when an application instance can be taken offline. Be aware of the costs of doing so, however, as any instance, even idle, consumes some resources and will therefore incur charges.

Gotcha #3: Client Details

Some applications require information about the client (browser), including its IP address. While most information about a client such as browser, plug-ins, content accepted, and so on is contained in the HTTP headers of a web application, the IP address is not. When a web or application server queries for the client IP address, the data is pulled from the network protocol information, not HTTP.

When a proxy service is inserted into the middle of client-server communication it can, and often does, change the network information. This is necessary to facilitate load balancing, transformation, and security functions on the proxy. In many cases, this means that the IP address returned from a query on the web or application server is actually the IP address of the proxy service, not the client. In other cases, query returns the expected result: the client's IP address. This is because the proxy service is sitting between a public network (the Internet) and a private network (the cloud data center). When it receives a request, it has to translate between the two, and in many network architectures this means it must change the network information so it can communicate with the private network, and then change it back after receiving the response to communicate with the public network. Even though your application is assigned a public IP address, that is usually the IP address of the application delivery or load-balancing service, not the actual application, and therefore a translation must occur in order for communication to appear seamless.

If the cloud computing or load-balanced environment is not preserving the client IP address and your application or corporate policies require tracking client IP addresses, every client will appear to be the same: It will have the IP address of the proxy.

As this has long been an issue in load-balanced environments, a solution exists that solves the problem by having the proxy service insert the actual client IP address into a custom HTTP header

called X-Forwarded-For. When the proxy service receives a request from the client, it extracts the client's IP address and inserts it into the X-Forwarded-For HTTP header before sending the request on to the appropriate application instance. Most hardware and software application delivery solutions are capable of inserting this custom HTTP header. If the cloud computing provider has built their own solution, it may not be available or they may have used a different customer header.

The possibility that the client IP address is not the "real" client IP address can be problematic even if you don't need the address for your application. It ends up in logs incorrectly, posing a problem for applications requiring a more accurate transactional log.

Solutions

- Inquire of the cloud computing provider whether the client IP address is preserved or inserted in the X-Forwarded-For HTTP header before you deploy your application. If it is preserved, you won't need to change a thing. If it isn't and the provider has implemented the use of X-Forwarded-For, skip to solution #2.
- By default, query for client IP addresses by examining the value of the HTTP header X-Forwarded-For first, and if it is empty, continue on to query the web or application server. Doing so will ensure that when the application is inserted into a load-balanced environment, it is ready to deal with the change in location and no modification will be necessary.
- If it is imperative that you be able to obtain the correct client IP address for auditing or for your application to work properly, be sure to inquire about the cloud computing provider's handling of client IP addresses before you sign up. If they cannot provide an accurate IP address you will likely want to look elsewhere for cloud services.

Conclusion

In general, developing applications for deployment in the cloud is really no different than developing applications for deployment locally. The tricky part of developing for the cloud, whether private or public, is the introduction of load balancing or application delivery into the equation. Many applications that were never seen as needing the scalability services provided by these solutions will necessarily, due to architecture and function, interact with these solutions in the cloud. The same issues that often arise in local deployments requiring load balancing or application delivery services will arise in the cloud.

Being aware of what those issues are and addressing them up front, if possible, is the best way to minimize the potential impact of an on-demand environment on the behavior and execution of applications in the cloud.

— Lori MacVittie is Technical Marketing Manager for F5 Networks.

Return to Table of Contents