LAKNATH'S COLUMN

Ramblings of a wanderer

DEC 24 2011

8 COMMENTS

BACKUPS, SERVERS, WEB

AVOIDING AWS POTHOLES

We are in the era of clouds, and at the moment AWS is the Zeus among public clouds. With its scalable and flexible architecture, cheap rates, secure PCI compliant environment, wide array of loosely coupled services and boasting of 99.95% availability, they may deserve the crown. However they are not without holes and few days ago I got the chance to taste it firsthand. This post is about few measures that you should (and I mean this with capital SHOULD) take before moving your production servers to AWS.

To start with, I had been using Slicehost and Linode as VPS providers for couple of years while tinkering with AWS. After a trial run of few months I was satisfied that everything is working as it should be and moved to AWS for real. But the mistake I've done and AWS didn't bother to mention anywhere easily findable is to couple Elastic Block Storage (ESB) with all instance stores. And this is something easy to overlook when you are coming from a regular VPS provider because èqȟèṃèr̈aḷ Instance store is the most counterpart similar device to a śḷ̈íćè and you may expect the same behaviour throughout.

So back to the story, everything was running fine until AWS had scheduled a maintenance rebooting of the instance two weeks ago. Nothing much to worry right ? But it turns out that the instance didn't reboot and there was very little possible to do from the AWS web console. Unlike in regular VPS slices, AWS doesn't come with a back-door SSH console and it turns out even the staff can do pretty much little regarding an instance store. The only solution they could give me was to reboot the instance few times and if it doesn't work out...well, they are sorry and it's a lost cause.

I earlier mentioned the mistake I've made. But what I got right was to have several layers of backups including database replication slaves. So backups were running pretty much as expected and there wasn't any lasting damage done. And only when you are in trouble that you are glad of the time well spent on emergency procedures.

So rest of the story is very little. I removed the crashed instance, restarted a new one from the custom AMI we had and copied data over from DB slaves. But this scenario could have gone vastly wrong if there wasn't a redundancy setup and for some unfortunate bootstrapping startup it could have reduced all their hard work to crisp.

I know servers should be up running and having them down is not heroic. But there are few points you should have in place before moving your production servers to AWS.

1. Have a proper backup procedure in place. Better if replication slaves are in some other server vendor or in another AWS region and have a monitor setup to make sure replication process is working properly. Also it's better to have several layers of backups running so you will have point-in-time recoverable database copy as well as one day old, week old, month old.etc data copies in worst to worst case scenarios.
2. Use Elastic Block Storage (EBS) – They are the external USB drives of AWS. Couple one or more EBS with your instance store and use them to store any data you think is valuable. If your instance die, you can just decouple the block and reattach to another fresh instance and run without a hitch.
3. Have a custom bare-bone AMI with just the OS and may be couple of basic services. Also have an AMI with fully ready-to-launch setup. This way you can make another production ready instance in minimal time as well as have an option in a worst case

scenario where the full ready made AMI doesn't work. Finally, test all your AMIs to make sure that they are working properly.

4. Have snapshots from your EBS devices in scheduled intervals.
5. Use these not so easy to find AWS architectural guidelines in designing your platform.

So as I mentioned it's not about heroics, but making sure your service not getting reduced to ashes because of some stupid server glitch. As someone wise had noted, better be ready than sorry!

Update:

There is another set of sound suggestions made in comment #4 by ǩõȓḍḽèśś for any cloud deployment. If you are into heavy scaling they may be particularly useful.

8 thoughts on "Avoiding AWS potholes"

Randy says: December 28, 2011 at 6:01 am

Yeah it's to bad that we can't trust the "experts " that are supposed to know what they are doing. I don't know a lot about the technical stuff but am learning,and it is surprising how often the "experts" screw up

Reply

BraveNewCurrency says: December 30, 2011 at 10:25 pm

I like to think of it this way: In the past, we focused on "MTBF" (Mean Time Between Failure.) We thought it would be a good idea if each computer had as much "uptime" as possible. We spent extra money on Dual Power Supplies, Dual NICs, RAID, dual UPS, yada yada. We paid $20K for a server we could have bought for $2K.

But the server still failed sometimes. One computer can *never* be 100% reliable. The UPS isn't reliable. The datacenter isn't reliable. The network isn't reliable. People aren't reliable.

Focus on "MTTR" (Mean Time To Recovery) instead. What are you going to do _when_ your server fails? I've seen a doctor's office down for 3 days because it took 8 hours to restore a backup (and they restored the wrong backup twice.)

Here's a better plan: Buy several $2K servers, and use software to "RAID" them together. When failure happens, recovery should be seamless and automatic. There's no reason you should be paged in the middle of the night just because some hardware died. Advanced users should be prepared for the whole datacenter/region going down.

Instead of avoiding failure, embrace failure. That's the cloud way.

Reply

Rob Harrigan says: December 30, 2011 at 10:53 pm

I recommend using Opscode Chef to bootstrap and bring up instances. This was an absolute lifesaver when we ran into similar reboot issues. New servers can be brought up and loaded with all the necessary packages in minutes. Freeing you to jockey backup data around and get the machine(s) back into a ready state.

Reply

Laknath says: December 31, 2011 at 2:54 pm

Yes, I'm planning on using Chef or Puppet when scaling our app architecture. Btw, any particular reason for choosing Chef over other configuration management systems such as CFEngine, Bcfg2, Puppet .etc ?

Reply

kordless says: December 30, 2011 at 11:05 pm

Sorry to hear of your troubles, thanks for sharing.

Loggly got hit far worse than you did. We've rebooted servers before and they usually come back up. This time over 95% didn't, and it took us all the way down and out for the count. We were down for over 24 hours trying to get our search cluster back online and taking data again. Our system is neither simple nor easy to bootstrap, and even though we have scripts that start and stop instances of our stack at will, for development or testing, bringing a large, live production cluster back up from zero took us WAY longer than we expected. We should have planned for it. We didn't. We didn't expect all our machines to go away. What we expected was SOME of them to go away, or SOME interruption of service (because it's the cloud!), but we never expected all of the boxes to be kicked by humans. All at once.

We should have planned better.

Backing up our database and all customer's logs (up until we went down) 'saved' us, but we still suffered data loss during the outage, dropping data that customers were sending in, and we were DOWN and unusable, which is the greatest sin of all.

Given our experiences, I would add a few more points to your list:

6. Test a full deployment of your current architecture, including size/scale, while still running another instance of it – this ensures you have the resources to start it if you need another one (we did not, and found out post-disaster we could only launch 50 total instance). If required and/or possible, test taking data from the production system and teeing it into the new deployment to see if it works properly.
7. Make sure your deployment management scripts (we use Puppet) work anywhere. We can launch instances of Loggly on VMs, bare metal, Rackspace, etc. Test alternate deployments on other AWS regions AND other providers. You don't know all the bad things that could be. AWS could go completely tits up, and you'd need to fail to … somewhere.
8. Regarding point 7, make sure you aren't depended architecturally on AWS services. Where possible, adopt alternate technologies that work across multiple infrastructures. For example, OpenStack supports a S3 like storage system. Make sure your stack works with it.

9. Don’t create technologies in your stack that are hard to scale and/or replicate. We’ve done that at Loggly because we thought we needed a single search cluster. We should have sharded customers/inputs/whatever across zones and regions. That way if part of it goes down, only a few customers are affected.
10. If you run in the ‘cloud’ realize you are offloading the responsibility for running infrastructure to someone else. We expect AWS to be reliable, but yet are limited in our expectations because of limits in the technology and costs that they must manage. Running your own boxes may place more responsibility on you for managing them, but will also allow you to better manage the expectations of what can go wrong.

Reply

Laknath says: December 31, 2011 at 2:43 pm

Thanks for sharing your experience. Though it’s hard to build a 100% reliable system, being aware of what has gone wrong/right in other cases give a good grasp of what can go wrong and be ready.

The application I was speaking of isn’t scaled to the magnitude of your case since it’s still not yet open to the public but all your suggestions are sound and useful, so updated my post mentioning your comment.

Reply

Nathan McCourtney says: January 1, 2012 at 1:23 am

I think your faith in EBS is unwarranted. Using EBS just means that it’ll persist if an instance terminates unexpectedly.

In the two years I’ve been using AWS in large production environments, random instance termination was the least of our problems. Weird EBS issues can cause horrific outages. So solve both the redundancy and durability problem at the same time: replicate your data among hosts in different Availability Zones and Regions from the get-go.

Reply

Laknath says: January 2, 2012 at 12:58 am

"Replicate your data among hosts in different Availability Zones and Regions from the get-go"

I was trying to make the same point throughout the post. EBS is just another tool helping to achieve the purpose but by no means be limited to it. However, rather than just having an instance store without any EBS coupled, having EBS with snapshots could give you more options in a failure.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

← Previous post Next post →

ARCHIVES

- April 2018
- September 2017
- January 2017
- March 2012
- December 2011
- October 2011
- September 2011
- December 2009
- May 2009
- January 2009
- December 2008
- October 2008
- July 2008
- June 2008
- May 2008
- February 2008
- January 2008
- December 2007
- November 2007
- August 2007
- July 2007
- June 2007
- May 2007

CATEGORIES

- .Net
- Backups
- Cricket
- Databases
- Drupal
- Eclipse
- education
- FIT
- Flash
- FOSS
- Fun
- Gnome
- GSoC
- inspiration
- Java
- javascript
- machine learning
- Maya
- Movies
- My Activities
- NLP
- Novels
- PHP

LAKNATH'S COLUMN

- Is this a toxic comment?
- Notes from Quora duplicate question pairs finding Kaggle competition
- The necessity of lifelong learning
- Machine Learning in SaaS paradigm
- Avoiding AWS potholes

- April 2007
- March 2007
- February 2007
- January 2007
- December 2006

- politics
- Python
- saas
- servers
- Sri Lanka
- ubuntu
- Uncategorized
- web
- Web Designing