Peecho Architecture - scalability on a shoestring

Monday, August 1, 2011 at 9:01AM

Todd Hoff in AWS, Example

This is a guest post by Marcel Panse *and* Sander Nagtegaal *from* Peecho*.*

Although architecture descriptions are an interesting read, the problems that start-ups face are hardly ever addressed. We would like to change that, so here is our architecture story.

Introducing a start-up

The Amsterdam-based company Peecho offers print-as-a-service. Our embeddable *print button* allows you to sell your digital content as professionally printed products, like photo books, magazines or canvases - straight from your own website. There is an API, too.

Printcloud is the system that powers the print button. It exists in the cloud only, growing when needed and becoming smaller if it can. The system takes in print orders, magically transforms tough data into print-ready files and routes the orders to the production facility that is closest to the intended recipient.

To preserve the environment, Peecho's philosophy is to facilitate global ordering, but to aim for local production only.

Expensive stuff does not scale

We are a start-up, so the most important thing that we considered before we started was simply *money* - or rather, the lack thereof. Although we required some serious firepower, the fully operational system should cost no more than a few hundred bucks a month.

The money issue reached all the way into our technology stack. We could insert something cool here about object-orientation, code compilation or MVC architectures - but in the end, we just needed an extensive development toolkit that wouldn't cost us. So, the computer science background of our developer community members pointed us to the Java platform.

Using cloud computing was a dead give-away, because on demand scaling eliminates expensive overcapacity. Our shortlist contained Google App Engine and Amazon Web Services. These molochs have way more resources and experience with scalability than we do. That's why we vowed to use as many of their cloud services as possible, rather than building stuff ourselves. However, we needed queueing and relational data storage - exit Google.

So, we chose to build on top of Amazon Web Services, using SimpleDB, S3, SQS, EC2, RDS, IAM, Route 53 and Cloudfront. It is cheap and generally reliable. Just to be sure, our entire system runs simultaneously in two AWS availability zones to guarantee maximum uptime. If we go down, half of the world will be there with us.

If you are slow, you can't grow

However, technology is not everything. Scalability for small companies is largely obtained by flexibility. Be fast. We try to keep our code simple, so refactoring is easy. As a result, we can afford to only build the bare minimum that is needed without thinking much about the future needs. We can fix those later.

If you really want to be fast, short iterations are important. On average, we release new software to our production environment about once a week. To keep up with that pace, we organize a monthly dinner. That's when demos are shown and everybody cheers and applauds for the new stuff.

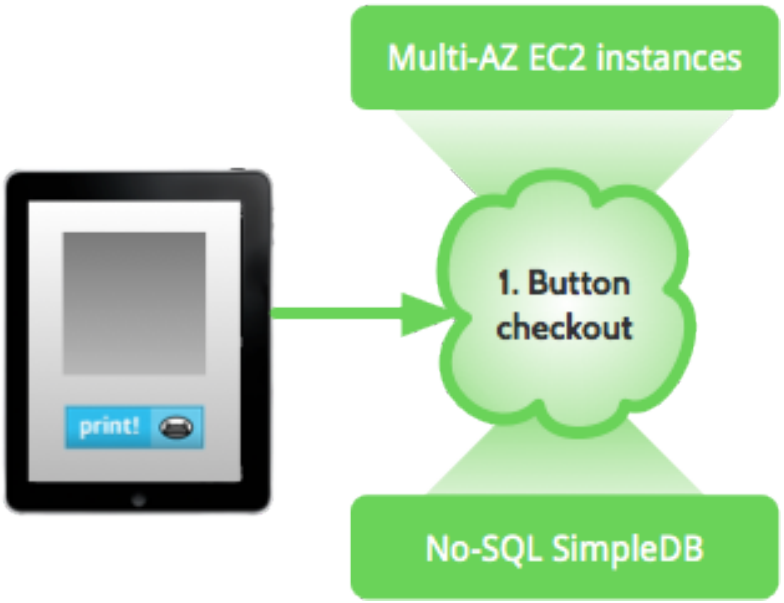
There is a lot of affordable help available to get you up to speed. For example, we develop according to Scrum and Test Driven Development. We use Jira with Greenhopper for requirements management, Bamboo for continuous

integration, Sonar for code quality monitoring and Mercurial for distributed versioning.

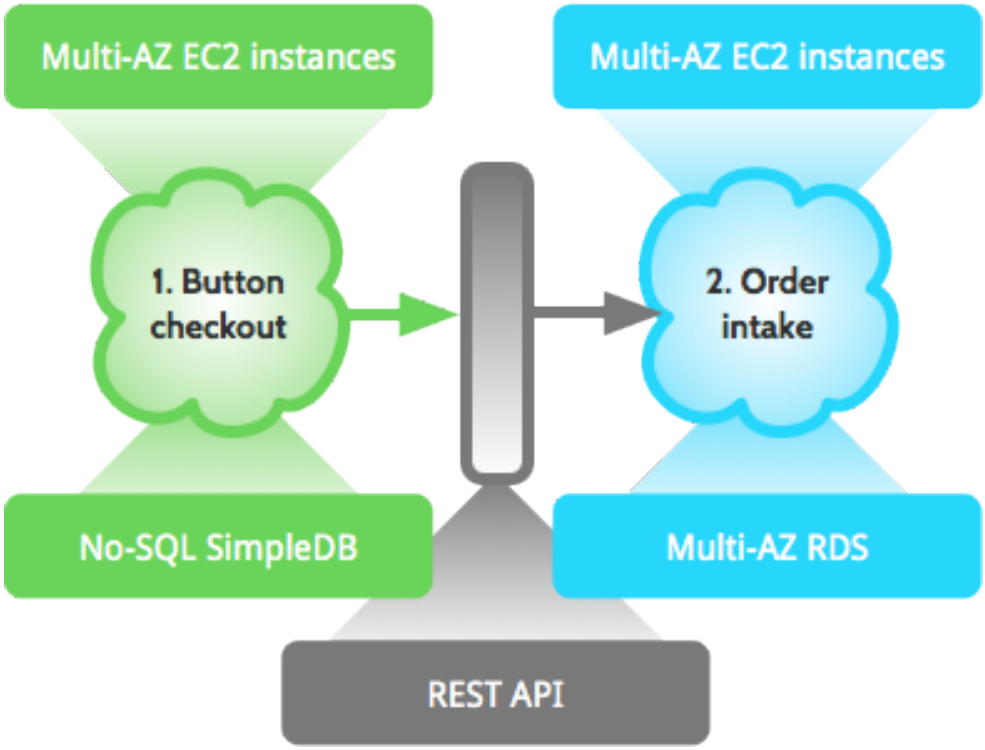
By the way, the large number of Atlassian products is partly explained by their *cheap start-up licenses* - but it is good stuff anyway.

Sure, but what does the architecture look like?

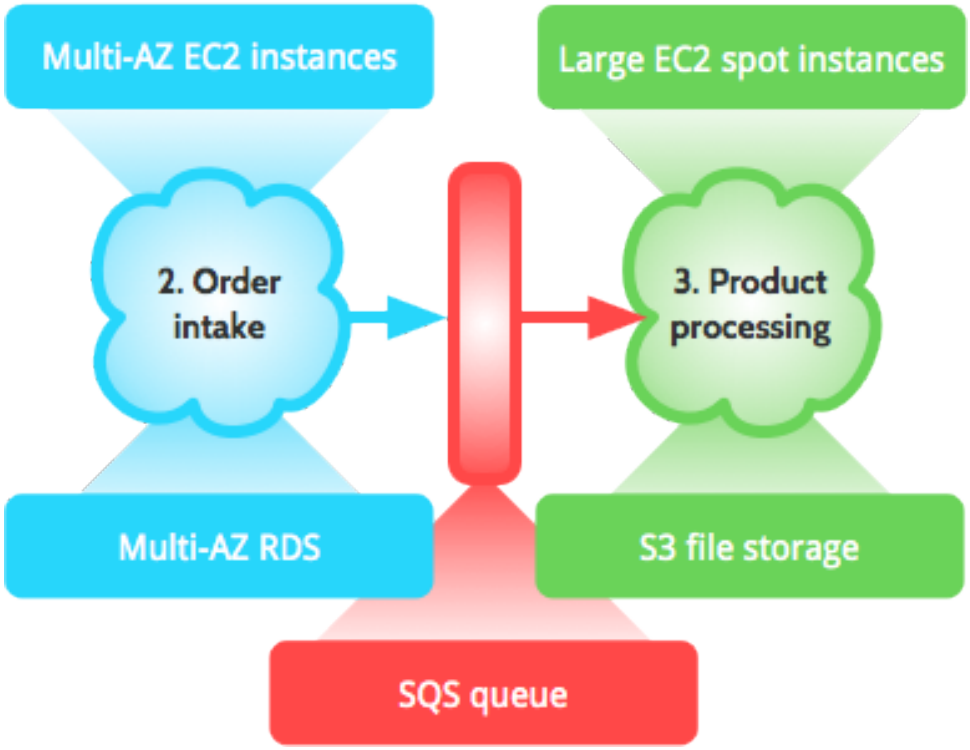
The print button and its checkout exist as a separate application on top of our platform. This separation of concerns is meant to minimize the interaction points - to achieve high cohesion and low coupling. The app runs on an EC2 auto-scaling group with a load balancer on top. It stores order data in SimpleDB, an AWS No-SQL data store that can take a beating. Everything static is run through the Cloudfront CDN, to avoid server hits.



After payment has been confirmed, the print button checkout submits every order to a REST API. This is the same API that our premium users access from their apps, so we only have to maintain a single interface. It is load-balanced to multiple nodes that can scale out automatically. Using the relational database service RDS, order data is kept in two availability zones.

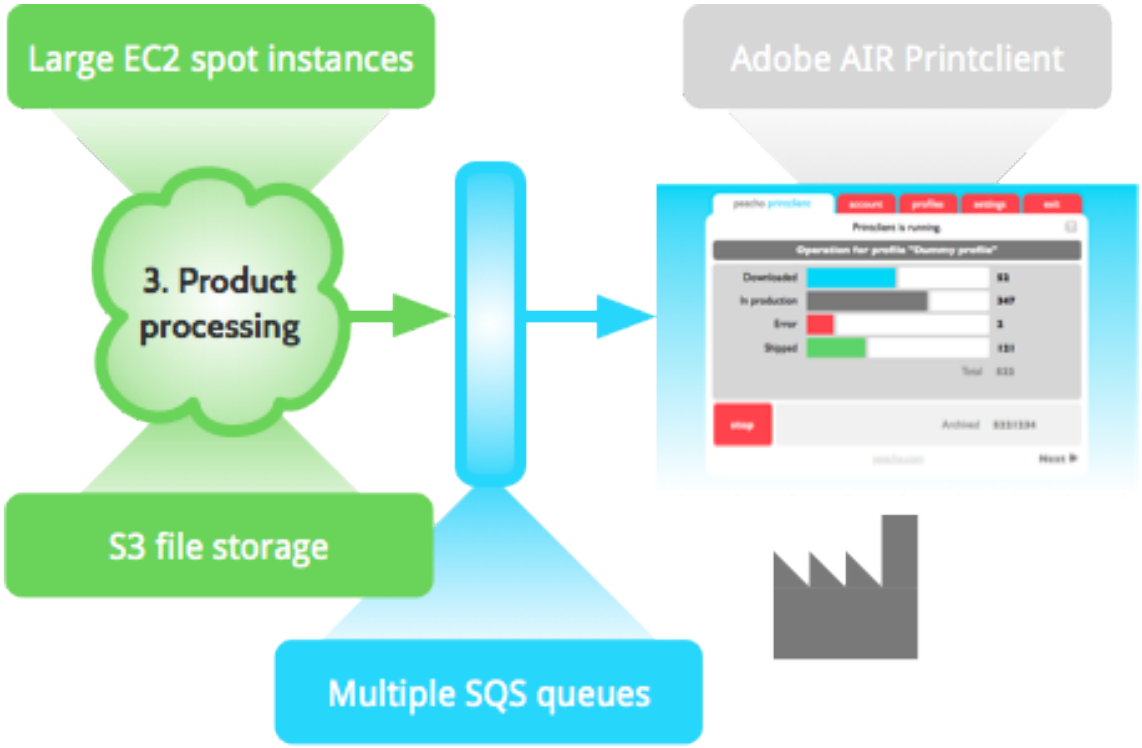


Now, this is where the magic happens. The order intake machine writes a ticket to the processing queue. Whenever there are enough tickets available, a new processing machine wakes up, gets a ticket and starts crunching that awful data. We use large spot instances that we rent by bidding on unused EC2 capacity. When those big guns are done, they store the result in S3 and then kill themselves.



So, using queue metrics, we scale the amount of processing power up and down as the number of items in the queue varies. This saves money and prepares us for unexpected outages. Before you do this yourself, keep in mind that each component or module should be responsible for a specific feature or functionality only. A component or object should not know about internal details of other components or objects.

Subsequently, the order must be routed to the right production facility by adding a production ticket to the right print facility queue. Using either our Adobe AIR Printclient desktop application or a direct integration, the facility retrieves its jobs from the queue and the product files from S3 file storage. There are no servers involved, so this system is practically bullet-proof.



In return, the facility can update job statuses in Printcloud by posting messages to the central order status queue. Asynchronously, we read that queue to update our customers about their orders. If all goes well, the product is shipped and delivered pretty soon.

After a certain period of time, the product files will be removed from S3 to save storage costs - and everybody lives happily ever after.

Doing the math

Let's give you a quick insight into our monthly AWS bill. There are a couple of things that you should know.

- Amazon offers free tiers for SimpleDB and other services.
- We use the European region cluster, which is more expensive than those in the United States.
- The EC2 instances are reserved for a year, making them cheaper.
- Spot instances are generally a bargain and we assume a continuous 50% workload.
- Just like the number of concurrent EC2 nodes, required bandwidth is

largely determined by the amount of orders coming in.

Task	Service	Cost estimate
Print button	Multi-AZ EC2 small instances	$ 96
Order intake	Multi-AZ EC2 small instances	$ 96
Processing	EC2 large spot instances	$ 84
Database	Multi-AZ RDS	$ 160
The rest	Cloudfront, S3, SimpleDB, SQS, etc.	$ 50

That makes a total of *486 dollars* a month for a fully elastic, heavy duty system that has the power of a minimum of 5 servers, 2 database servers, a No-SQL data store and external flat file storage.

Not bad, we think.

We wish you cheap apps

Peecho proves that it is possible to create scalable architectures on a shoestring. We hope this information was useful to you and that it will help you to create great apps for next to nothing, too. Preferably with a print button, of course.
