“You Build It, You Run It”

Perhaps by chance, but probably by design, AWS empowers development teams to become truly agile. It does this in two ways:

* Getting rid of the long-term aspect of the application infrastructure (investment)
* Building tools to help overcome the short-term aspect of operating the application infrastructure (failure)

There is no need to distinguish between building and running, and [according to Werner Vogels](http://queue.acm.org/detail.cfm?id=1142065), it is much better than that:

*Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.*

This lesson is interesting, but this particular change in an organization is not always easy to implement. It helped that Vogels was the boss, though it must have cost him many hours, days, and weeks to convince his colleagues. If you are not the boss, it is even more difficult, though not impossible. As we have seen before, AWS offers ways to be agile with infrastructures. You can tear down servers, launch new ones, reinstall software, and undo entire server upgrades, all in moments.

Individuals and Interactions: One Team

In bigger organizations, there is an IT department. Communication between the organization and its IT department can difficult or even entirely lacking. The whole activity of operating applications can be surrounded with frustration, and everyone feels powerless. Smaller companies often have a hosting provider, which can be very similar to an IT department. A hosting provider tends to be a bit better than an IT department because you can always threaten to replace it. But the lock-in is significant enough to ignore these issues; for a small company it is generally more important to focus on development than to spend time and energy on switching hosting providers.

Let’s start with one side: the IT department or hosting provider. Its responsibility is often enormous. IT department members have to make decisions on long-term investments with pricetags that exceed most product development budgets. These investments can become difficult projects with a huge impact on users. At the same time, the IT department has to make sure everything runs fine 24/7. It is in a continuous battle between dealing with ultra long term and ultra short term; there seems to be nothing in between.

Now for the development team. The work of the development team is exactly in between the long term and the short term. The team is asked to deliver in terms of weeks and months, and often makes changes in terms of days. During the development and testing phases, bugs and other problems are part of the process, part of the team’s life. But once in production, the application is out of the team’s hands, whether they like it or not.

Organizations can handle these dynamics by creating complex processes and tools. Because each group typically has no understanding of the other’s responsibilities, they tend to formalize the collaboration/communication between the teams, making it impersonal. But as the Agile Manifesto states, in developing software *individuals and interactions* are more valuable than processes and tools. With AWS, the investment part of infrastructures is nonexistent. And AWS helps you manage the ultra short term by providing the tools to recover from failure. With AWS, you can *merge the responsibility of running the application with the responsibility of building it*. And by doing this, you turn the focus on the people and their interactions instead of on creating impersonal and bureaucratic processes.

Working Software: Shared Responsibility

Deploying software means moving the application from development to the “other side,” called *production*. Of course, the other side—the IT department in the traditional structure—has already committed to a particular SLA. As soon as the application is moved, the IT department is on its own. As a consequence, members want or need to know everything necessary to run the application, and they require documentation to do so.

This documentation is an SLA itself. If there is a problem related to the software that is not included in the documentation, fingers will point to the development team. The documentation becomes a full description of every aspect of the application, for fear of liability.

But in the end, there is only one thing that matters, and that is whether the application is running. This is not very difficult to determine if the responsibility is shared; the team members will quickly discuss a solution instead of discussing who is to blame. So the thing to do is to build working software together, as a team. Remove the SLAs and merge the functions of two teams into one. When something doesn’t work, it needs to be fixed—it does not always have to be debated first. Documentation in this context becomes less important as a contract between parts, and becomes an aid to keep the application running.

Customer Collaboration: Evolve Your Infrastructure

Wherever IT is present, there is an SLA. The SLA is regarded as a tool in managing the process of IT infrastructure, where the bottom line is the number of nines. In reality it is a tool designed to facilitate cooperation, but is often misused for the purpose of deciding who is responsible for problems: development or operations.

It can be difficult to negotiate this contract at the time of application development. There is a huge difference between “we need to store audio clips for thousands of customers” and “storage requirements are estimated to grow exponentially from 500 GB to 5 TB in three years.” The problem is not so much technical as it is that expectations (dreams, often) are turned into contract clauses.

You can change contract negotiation into *customer collaboration*. All you need to do is merge the two responsibilities: building and running the application becomes a shared challenge, and success is the result of a shared effort. Of course, in this particular example it helps to have Amazon S3, but the point is that requirements change, and collaboration with the customer is better suited for handling those changes than complex contract negotiations.

Responding to Change: Saying Yes with a Smile

At the end of the project, just two weeks before launch, the CEO is shown a sneak preview of the new audio clip platform. She is very excited, and proud of the team effort. The meeting is positive and she is reassured everything is planned for. Even if the dreams of millions of customers come true, the platform will not succumb to its success because it’s ready to handle a huge number of users.

In the evening, she is telling her boyfriend about her day. She shares her excitement and they both start to anticipate how they would use the new platform themselves. At some point he says, “Wouldn’t it be great to have the same platform for video clips?” Of course, he doesn’t know that this whole project was based on a precondition of audio-only; neither does the CEO.

In the morning, she calls the project manager and explains her idea. She is still full of energy and says enthusiastically, “the functionality is 100% perfect, only we want audio *and* video.” The project manager knows about the precondition, and he also knows that video files are significantly bigger than audio files. However, the CEO doesn’t want to hear buts and objections about moving away from the plan; she wants this product to change before launch.