electric CLOUD

# Fail-Safe Application Deployment

Written by Dan Gordon

September 2012

## Introduction

Enterprises are building, testing, and deploying software–particularly Web solutions and service-oriented applications–faster and more frequently now than at any point in the past. Faced with unprecedented demands, many of these software development organizations are realizing their rollout processes are haphazard, at best. These improvised procedures lead directly to heightened numbers of costly, time-consuming errors that degrade their business agility.

Fortunately, there is a well-regarded, proven collection of best practices–and supporting technologies –that can go a long way towards making the software deployment process more streamlined, safer, and more robust.

This paper reveals how these fail-safe software deployment techniques deliver an impressive array of business and technological advantages. It starts by explaining what makes application deployments so problematic right now. Next, it summarizes the troubling business and technical outcomes of these challenges. Finally, it will present five essential guidelines that help produce fail-safe software implementations.

The intended audience for this paper comprises anyone responsible for delivering smooth-running software solutions. This includes operations executives, software release managers, software developers, and quality assurance professionals.

## Why is Deploying an Application So Difficult Today?

There's no single culprit behind this contemporary predicament. Instead, it is driven by a complex tangle of interwoven business and technological factors.

### Faster deployment cycles

There are three fundamental reasons why applications are being rolled out more rapidly than ever before:

1. Agile software development practices – which feature ongoing software delivery – have become quite commonplace in software development organizations.
2. Rapid cloud computing adoption has facilitated faster staging of development, test, and production environments.
3. Modern, distributed applications such as Web and/or service-oriented software are treated as essential business assets. Thus, it's no longer acceptable to release new software or fix bugs at a leisurely, semi-annual pace.

### Many moving parts

Applications are no longer constructed as monolithic, self-contained entities. Instead, they're composed of, and rely on, numerous disparate technologies. These include Web servers, application servers, databases, and so on.

Even the software deployment process itself is ordinarily made up of a jumble of scripts and hand-performed tasks, all of which must be perfectly harmonized to be effective.

### Multiple deployment destinations

Rolling out even the most rudimentary application or software service usually encompasses several destinations, such as:

- Development
- QA
- Pre-production
- Production

Each of these environments commonly sports a distinct configuration. Nevertheless, the deployed application must correctly run on each one despite these dissimilarities. Not surprisingly, destination-specific differences can introduce many confusing aberrations that must be investigated and corrected.

It's not unusual to discover that an application will perform flawlessly in a testing environment yet fail in production. Tracking down the causes of these anomalies can be maddening.

### Missing or inconsistent automation

To make the release cycle even more challenging, many applications are installed manually, with some automated steps, or with automation that is not used consistently between teams, silos, or environments.

Without consistent automation across every deployment, there is a high likelihood of an unintended and unmanageable number of differences in deployment steps, which can result in errors that are hard to troubleshoot.

### Ineffective communication

The Dev and Ops teams must coordinate their activities to ensure proper deployment. Yet these groups are commonly siloed, with minimal sharing of information and processes. These barriers directly result in error-prone handoffs and limited visibility.

Due to all of the factors just described, it can–and often does–take days or weeks to properly deploy a finished application. This process is highly error-prone, difficult to repair, and its lack of traceability makes it impossible to create and fine-tune a repeatable workflow. This wreaks havoc on schedules and user expectations, causing significant business interruptions. The only way to surmount these obstacles is to employ a collection of well-proven guidelines that optimize the software delivery procedure.

## Five Best Practices for Fail-Safe Software Deployment

The recommendations provided in this section can make your software deployment experience faster, smoother, and more reliable.

### 1. Design for manufacturability

The homegrown build, test, and deploy procedures that have characterized software development for decades are no longer sufficient. They're brittle, error-prone, and too reliant on individual knowledge. Merely getting things right once is hard enough; repeatability is nearly impossible.

Rather than continuing to rely on handcrafted processes, your goal should be to transform your software design and implementation procedures into a more mechanized, repeatable series of steps. This helps make test results from earlier phases in the delivery cycle relevant for later stages. It also lets you perform consistent tests in many scenarios over time, with minimal adjustments.

To encourage manufacturability, begin by itemizing all the details of your deployment process. This means aggregating all related components into a single application unit. These components commonly include:

- File sets
- Scripts
- Configurations
- Processes

Additionally, to encourage manufacturability, the components themselves must be kept in a versioned manner. This is important so that unexpected changes don't leak into the process that we are hoping to keep consistent and repeatable.

## 2. Leverage the power of automation for your software delivery process

Simply detailing all the elements that constitute your software delivery procedures is a great start, but that's not the end of the story. The next step is to eliminate the unrefined, often manual deployment processes that still plague so many software development organizations. Comprehensive automation technology can have a meaningful impact on productivity and accuracy, just as it has for many other sophisticated business practices.

The selected technology should be specifically tailored for the software delivery process:

- It should provide built-in support for the complete, end-to-end delivery process.
- It needs to factor in component dependencies.
- It must include integrations to a widespread assortment of third party software and platforms.
- It should offer extensive process modeling capabilities. There are three essential models to consider:
  - Application – the 'what'
  - Environment – the 'where'
  - Workflow execution – the 'how'
- The environment(s) should be modeled as well, with details such as:
  - Server configuration
  - Associated parameters
  - Environment configurations
- It should furnish visibility and transparency throughout the entire process, including all possible destination environments.
- It should be capable of unlimited scalability:
  - Users
  - Servers
  - Applications
  - Locations

### 3. Design with failure in mind

Hope is not a strategy for effective software deployment: failures *will* occur despite your best efforts. To prepare for these inevitable breakdowns, follow these steps:

- Embrace failure early on. Don't ignore it. Learn from it and adapt moving forward.
  - Look for run-time troubleshooting capabilities, with:
    - Break points
    - Skip steps
    - Step-through
    - Ability to resume in-flight deployments
- Determine what is an acceptable failure. By acceptable we mean a failure that doesn't necessarily need to halt the entire software deployment process.
  - Define success and failure thresholds by tier
  - Allow for partial deployments to complete successfully
- For example, what should happen if three of ten Web servers fail to deploy?
- Create well-defined recovery/rollback plans, and then test them.
  - Recover from partial failures
  - Pre-define what actions should be taken if a failure occurs
- Define different behaviors for different failures
  - Actions can include
    - Auto-pause
    - Alert on failure
    - Auto re-run
    - Auto abort/recover/rollback

### 4. Test early and test often

Too many software development teams procrastinate until the pre-production or even production phases before beginning to test their application deployment process. These delays often result in unpleasant surprises that cause frantic efforts to decipher and correct what's wrong.

Instead of putting off these essential responsibilities, it's a better idea to build a consistent deployment model and test it throughout the entire software development lifecycle. Naturally, your selected software deployment platform should reside at the heart of your testing efforts. This approach uncovers any issues well before a crisis develops. It also lets you evolve and harden the process so your production deployments are smooth and fail-safe.

### 5. Zero in on defects efficiently

Since so many organizations continue to rely on handcrafted processes, it should come as no surprise that identifying and correcting defects tends to be laborious and inadequate. Specialized automation solutions are ideal for isolating and resolving these types of problems. This makes troubleshooting complex deployments much more efficient, and results in faster time-to-market for your software as enhanced agility for your enterprise.

## Benefits of Fail-Safe Deployment

Transforming today's highly complex software delivery processes into fail-safe production deployments has many benefits:

- **Faster time to market.** Software is a strategic, differentiating asset for the vast majority of enterprises. Thus, anything that gets software delivered more quickly can lead to competitive advantages. After all, every business wants to be more agile.
- **Higher quality of delivered software.** Few enterprises can tolerate mission-critical software that's buggy or sluggish. The less time software development resources spend on deploying their software, the more time they can spend on writing better software.
- **Reduced cost.** Even though organizations are increasingly reliant on enterprise-grade software, budgets aren't necessarily keeping pace. Better software delivered more quickly saves money.
- **Increased DevOps collaboration.** With such frequent delivery cycles, it's critical that these two organizations cooperate more seamlessly and efficiently. A shared model of what comprises the application, plus shared visibility of the status of the application, enables the Dev, QA, and Operations teams to work more efficiently together towards their common goal.
- **Improved visibility and tracking.** Automation offers unprecedented insight into what had previously been a fairly opaque process. This helps to enable continual, evolutionary improvement.

## About Electric Cloud

Electric Cloud delivers solutions that automate and accelerate the application development and delivery process. The company's award-winning products help development organizations to speed time-to-market, boost developer productivity, and improve software quality while leveraging the operational efficiencies provided by virtualized/cloud infrastructures. Leading companies across a variety of industries, including financial services, ISVs, mobile devices, semiconductors and transactional websites rely on Electric Cloud's automation solutions. For more information, visit **http://www.electric-cloud.com**.

## About the Author

Dan Gordon is a Product Manager at Electric Cloud. Dan brings over 20 years of experience in the IT software industry. At Electric Cloud, Dan is responsible for product strategy, product marketing, tactical alignment and execution with product development, sales and pre-sales enablement and support. Previously, Dan served as product manager and systems architect for the enterprise IT automation software business within HP Software. Dan has also held managing and systems engineering roles at Opsware and Sun Microsystems. Dan holds a bachelor of science in information and computer science from the University of California, Irvine.

**Corporate Headquarters**

**Electric Cloud, Inc.**

676 W. Maude Avenue, Sunnyvale, CA 94085

**Tel:** 408.419.4300 **Fax:** 408.419.4399

info@electric-cloud.com **www.electric-cloud.com**

**Electric Cloud Europe**

1650 Arlington Business Park

Theale, Reading

Bershire RG7 4SA United Kingdom

europe.info@electric-cloud.com

**Electric Cloud Japan KK**

22F Shibuya Mark City West

1-12-1 Dogenzaka, Shibuya-ku

Tokyo 150-0043 Japan

Tel: +81.3.4360.5375

japan-info@electric-cloud.com