Breaching Barriers to Continuous Delivery with Automated Test Gates

Chris Struble

cstruble@vertafore.com

Abstract

Continuous Delivery may have come out of companies that were “born on the web”, but what if you are in a regulated, mature industry with sensitive customer data, where every change to production must be approved by a review board? How can you design, deploy, and execute automated tests that will work in environments that cannot be directly accessed by your team? How can you breach barriers to Continuous Delivery in a business context where the walls between development and operations are rigorously maintained?

Vertafore is an intermediary in the insurance industry, providing information solutions for insurance agencies, carriers, managing general agents, and state agencies. When our team was tasked with enabling Continuous Delivery, we faced all of the above challenges. We overcame them using Jenkins, Artifactory, and Chef to build a Continuous Delivery system with automated test gates for promoting product and test code from Development through Staging to Production.

Our Continuous Delivery system supports automatic promotion based on test results, or manual approval. Because of this flexibility, we were able to get adoption from product teams at each of the different stages of Continuous Delivery process maturity. It also provides a common way to deploy and test that our developers, testers, and release engineers can use in the same way in each environment. The system has enabled our company to increase the size and frequency of releases.

This paper will cover the key patterns that our team developed or discovered in our journey, which can help you in creating your own Continuous Delivery system, regardless of the tool set.

Biography

Chris Struble is a Senior Software Engineer in Test at Vertafore in Bothell, Washington. He has 20 years of experience in software testing and infrastructure, build, deployment, and test automation. He holds Masters Degrees in computer science from Walden University and mechanical engineering from the University of Houston. Currently, he is focused on improving software quality and Continuous Delivery. He lives in Renton, Washington, with his wife, two teenagers, and three cats.

Copyright Chris Struble 2016

1 Introduction

The ability to deliver software changes to users quickly and reliably has become as important as the features of the software itself. Today's users have smart phones, smart televisions, tablets or other devices with applications that update continuously, accessing web applications hosted on cloud environments that themselves are updated multiple times a day without the end user even being aware.

These changes came about because many leading companies have implemented Continuous Delivery practices. Jez Humble defined Continuous Delivery as "a set of principles and practices to reduce the cost, time, and risk of delivering incremental changes to users". Organizations doing Continuous Delivery typically have automated deployments, delivery pipelines, and extensive automation of infrastructure, deployment and testing across all environments, including Production.

Vertafore is an intermediary in the insurance industry, providing information solutions for insurance agencies, carriers, managing general agents, and state agencies. The company's products touch every point of the distribution channel. Their engineers support dozens of products that combine to make the broadest and most complete portfolio of software in the insurance industry. Technology is rapidly transforming the insurance industry, incenting companies to move away from large monolithic applications and toward web services that can be released more quickly and reliably. Speed and reliability are critical in an industry-- venture capitalists have invested more than $2.2 billion dollars in the industry over the last five years and they anticipate a return for their investments.

What has not changed are the strict rules about handling of confidential customer information in the insurance industry. An example is the Health Insurance Portability and Accountability Act (HIPAA), which includes network security rules that require companies providing solutions to limit access to personal health information (PHI) of individuals.

To comply with such rules, Vertafore limits access to its production environment to essential personnel, such as system administrators and release engineers. Any changes to production must be approved and documented by a change review board and must be initiated by an operations release engineer during approved maintenance periods only. Most of our software development engineers have no access to production and our network rules do not allow updates to be pushed into production by non-operations staff.

These constraints might appear to be a significant barrier to doing Continuous Delivery. However, in reality, Vertafore has been doing Continuous Delivery for several years. We just do it differently.

2 Case Study: Continuous Delivery at Vertafore

Our journey toward Continuous Delivery started in 2012, when Vertafore formed a team called Continuous Quality in our Bothell, Washington office. This team, which included the author, was initially focused on improving test automation for three of our largest management systems: AMS360, BenefitPoint, and Sagitta.

Agile development already was well established at Vertafore, with our product development teams already using a Scrum process, and most using Microsoft Team Foundation Server (TFS) to track work items, run manual tests, and store and build source code continuously. Many of our development teams also used behavior-driven development (BDD) to create automated acceptance tests with SpecFlow.

Our team's first project was to develop new UI tests in C# with Selenium WebDriver for .NET, based on SpecFlow scenarios provided to us by the product development teams. We set up a Jenkins continuous integration build server with Windows test agents to run the test cases on demand.

As we improved the test cases to make them more stable, we realized we could abstract much of our code to a framework. We called the framework “Firefly” and shared it internally with our product development teams as a NuGet package.

Our team’s first NuGet repository was a file share. As more product development teams began to use our framework and began to create their own NuGet packages to share .NET code with each other, we needed a repository manager that could scale with the growing demand.

We decided on Artifactory Pro. One factor driving that decision was that our team’s charter had just expanded. We were tasked to implement an automated Continuous Delivery pipeline to support an Agile Release Train for Vertafore Agency Platform, a suite of new products designed to work together in a services-centric architecture. The pipeline needed to work across multiple environments, some of which are isolated.

A key feature of Artifactory Pro that would enable our pipeline was the ability to replicate artifacts between multiple Artifactory instances. Artifactory Pro had good integration with Jenkins, NuGet support, and a REST API, each of which was an attractive feature for us. We set up an Artifactory Pro server and were soon using it to host NuGet packages and other binary artifacts built on our Jenkins server.

Our Agile Release Train is a release management process that is part of the Scaled Agile Framework. Agile release trains occur on a regular schedule, with high quality, but with variable scope. In the release train metaphor, releases are equivalent to trains, while individual software products are equivalent to freight cars. If the product isn’t ready, the release continues without it, but, no worries, the product can always go out with the next release.

Our team held sessions to visualize how a fully automated Continuous Delivery pipeline would work. Our design used Jenkins and Artifactory as the foundation and made extensive use of available plugins to implement parts of the pipeline and custom tools to implement any missing pieces. We called our delivery pipeline concept the Automated Deployment System (ADS) and the name stuck.

Figure 1 shows how ADS works to deliver our software through the development environment.

The ADS delivery pipeline begins when a software engineer checks in code to TFS. This triggers a Jenkins build job, which builds the code, runs unit tests, and publishes the artifacts to Artifactory.

Our Artifactory Pro server has artifact repositories, including a Development repository and a Release repository. Artifacts are always published to the Development repository, never to the Release repository.

Once the code is published, a “gate evaluation” Jenkins job is run to analyze the test results file from the unit tests. The gate evaluation job takes several parameters, such as the build being evaluated and the pass score, i.e., the percentage of test cases that must pass for the build to pass the unit test gate.

If the build passes the unit test gate, a deploy job is run to deploy the product into the development environment. After deployment, additional automated tests may be run (such as web service tests, integration tests, endpoint validation tests, and user interface tests), each of which may have test gates run to evaluate the test results.

If the build passes all of the test gates, the artifacts are automatically promoted to the Release repository. If any of the test gates fail, the build fails and the artifacts remain in the Development repository.

Product development teams are able to specify which test gates to include in their workflow and the required pass score for each gate. Typically teams require a pass score of 100%.

The development teams also are able to use a manual gate in their workflow. Providing support for manual gates allowed us to support development teams who were not as far along in their automated testing, or who were not comfortable with an automated process making release decisions.

To deliver our software all the way to Production, we had to use a different process than we used in Development. To explain why we had to do that, I need to digress and tell a short story.

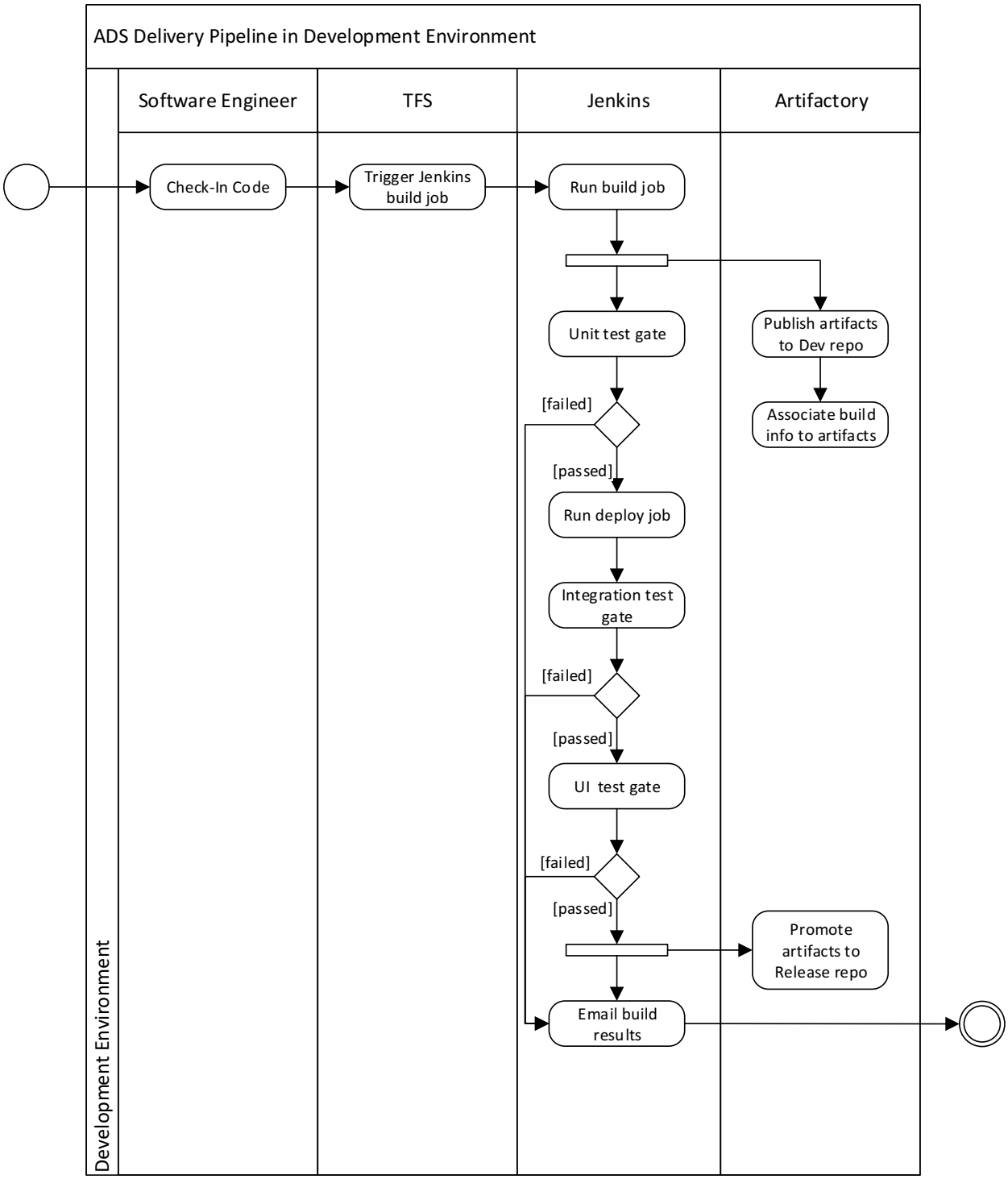

Figure 1 – ADS Delivery Pipeline for Vertafore Agency Platform in the Development Environment

3 Story: A Game of Gates

Imagine for a moment that you are in Europe, sometime in the Middle Ages.

You are a merchant, the owner of a cart filled with trade goods. You want to get your goods to the market. The market is in the courtyard of a huge castle, surrounded by a series of stone walls and gates, each larger and stronger than the one outside it. You have a letter from a buyer who lives inside the castle, but you have never met her or previously been to this castle.

On a perfect day, all the gates of the castle would be open and you could just drive your cart into the market, deliver your goods, get paid, and go home.

When you arrive at the castle, the first gate is wide open. The guards barely look at you as you enter the outer courtyard.

Then you notice that the second gate is closed. The guards at the gatehouse are not smiling. You ask the captain of the guards to open the gate. He refuses. It seems there are rumors of assassins. He doesn't know you and won't let you in. Nothing personal. Orders from the king and all that.

For a moment you contemplate knocking in the gates with a battering ram. Then you remember that these people are not your enemies. You want to do business with them, not fight them.

Eventually you get an idea. "If you won't let me in, will you take a message to my buyer?" you ask the captain of the guards. "Tell her to come out of the gate once each day. If she wants to buy my goods, she can take them inside. Otherwise, she can leave them here."

The captain considers your proposal and eventually agrees. But he warns you. "Behind this wall is the inner gate, the largest of all. The captain of that gate isn't as friendly as I am. He only opens the gate at certain times of the day and he personally inspects everything. He won't let anything through unless there is paperwork listing every single item and every person that has come into contact with it. Can you do that?"

You agree. Each morning you bring your goods to the castle in your cart and sometime later your buyer comes out with her own cart. Sometimes she isn't able to get your goods past the inner gate, and sometimes she doesn't want them at all. But enough of your goods get through to make it worth the effort. And though you never meet your buyer face to face, you both get very rich.

4 Getting to Production

To extend the ADS delivery pipeline to Production, we had to have conversations with our security and operations teams. Vertafore has a Staging environment that our release engineers use to practice releases and identify deployment issues before releasing to the Production environment. Staging and Production are both isolated. Jenkins does not deploy directly into those environments, and a security exception was required

Like the merchant's request to the captain of the guards to open the castle gates in the story, our request to open a security exception for Jenkins was initially denied. For numerous security and business reasons, push-button deployments into Staging and Production are not allowed. But we were told that a deployment initiated from inside Staging or Production would be acceptable.

Our solution was to place an Artifactory Pro server inside Staging and Production and use the replication feature to request artifacts from our main Artifactory server in Development. Because replication requests are initiated from inside Staging or Production, we could meet Vertafore's security requirements. Figures 2 and 3 show the architecture and activity diagrams for our solution.

Every 10 minutes, the Artifactory Pro servers in Staging and Production initiate requests to our main Artifactory Pro server in Development for any new artifacts. Only the Release repository is replicated, to ensure that only promoted artifacts are available to be deployed in Staging or Production.

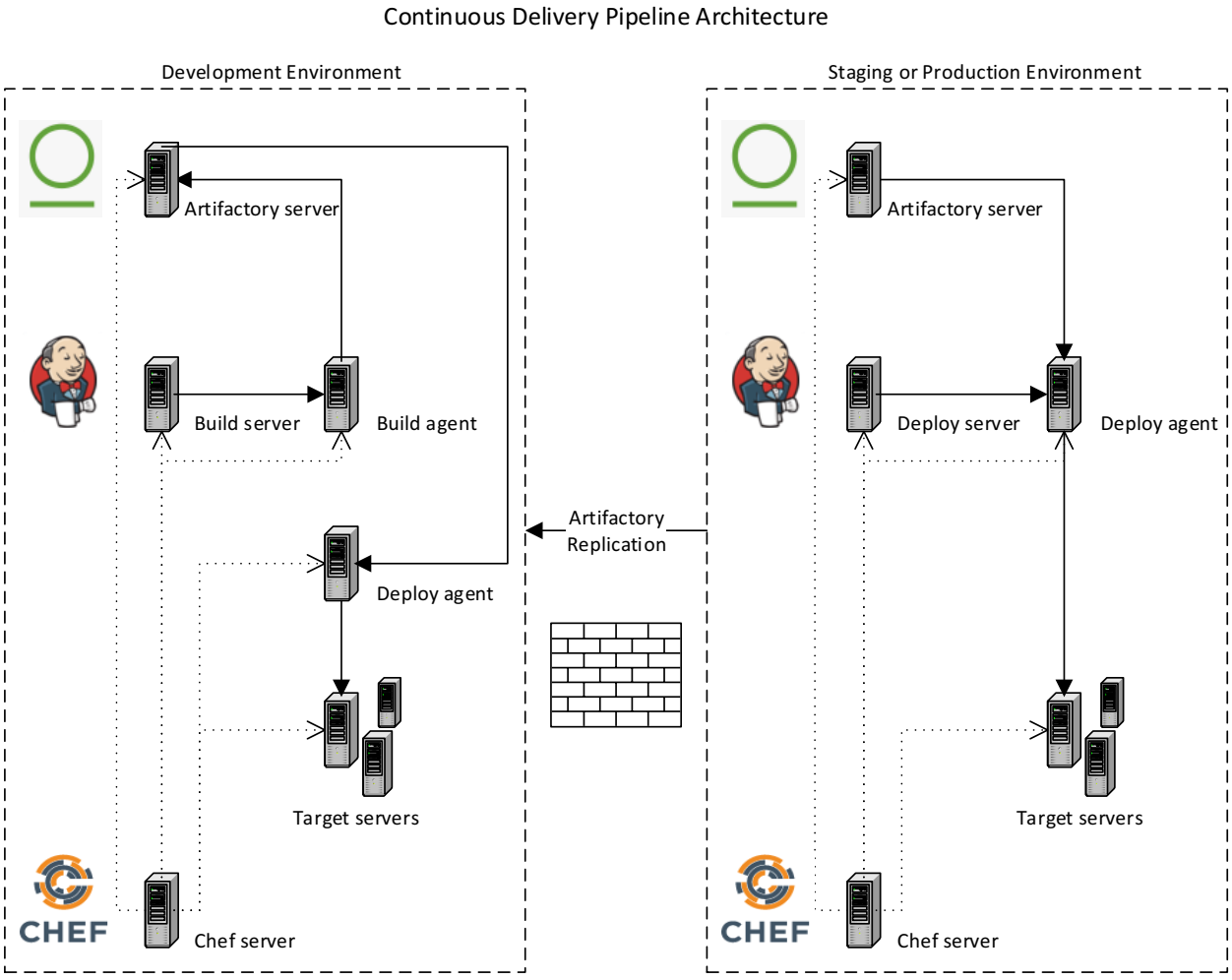

Figure 2 – Architecture of ADS Delivery Pipeline for Vertafore Agency Platform

The first product to use ADS to deliver to Production was contained in the December 2013 release of Vertafore Agency Platform. Six months later, 90% of the products on the Agile Release Train were using ADS to deliver all the way to Production. The only exceptions are legacy products that still require manual deployment.

As more products used ADS, a new problem emerged. Small differences between the Jenkins and Artifactory servers in Development, Staging, and Production, and between the versions of the Jenkins jobs in each environment, were causing deployment failures. These occurred because machines had been configured manually, often by different people.

We turned to Chef, an infrastructure automation tool, to help us solve this configuration problem. Chef uses scripts called “recipes” to ensure that machines are in a desired state. We created Chef recipes to configure our Artifactory and Jenkins servers, agents, and jobs, so that deployments and tests would

work the same across all environments. Prior to each release, we update the Chef recipes and apply the updates to the Artifactory and Jenkins servers and agents in each environment.

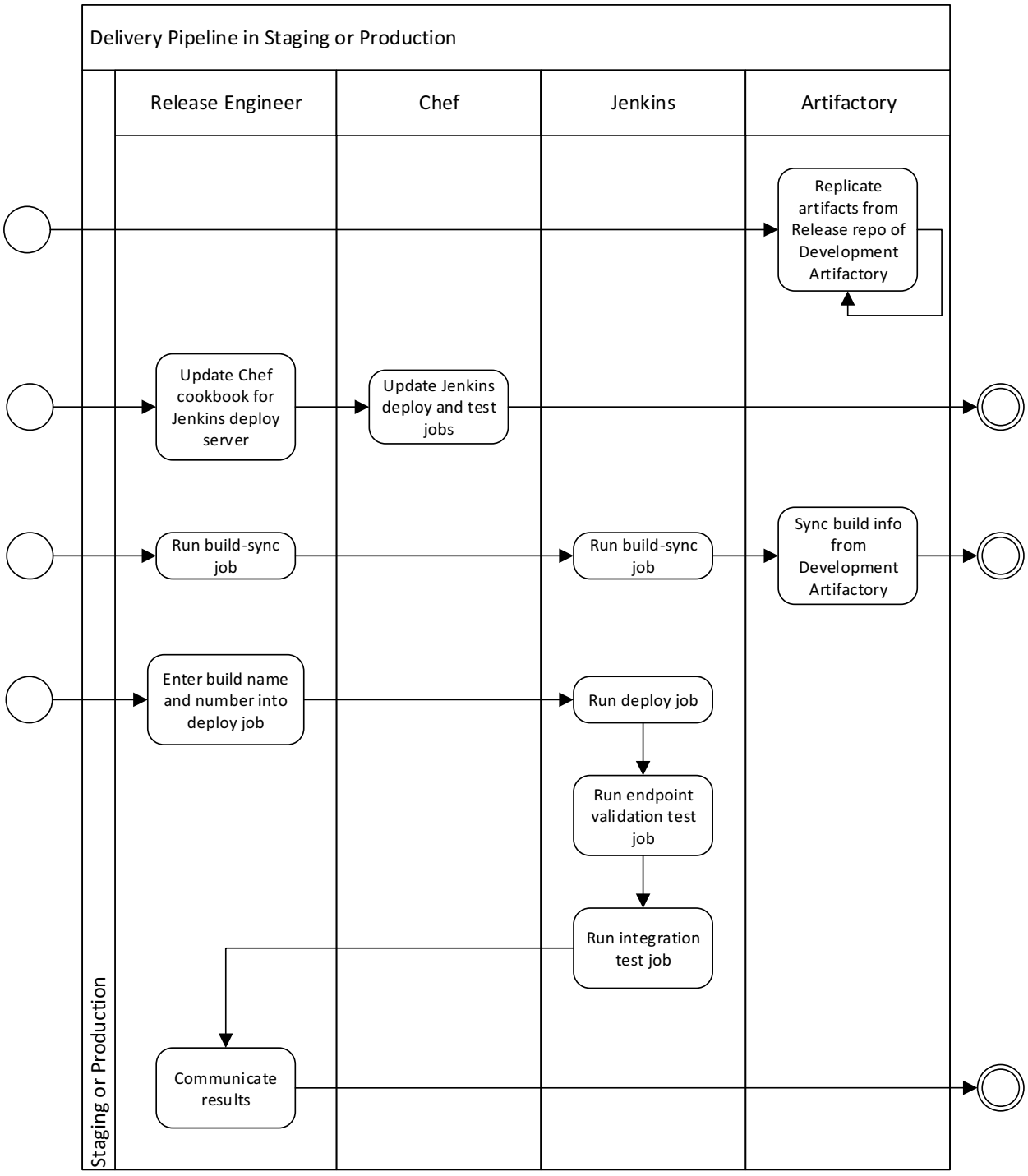

Figure 3 – Activity Diagram of ADS Delivery Pipeline in Staging and Production

Since we started using Chef to configure the machines in the ADS delivery pipeline, deployment failures in Staging and Production have declined significantly.

By 2016, ADS grew to more than 800 Jenkins jobs, with more than 8000 artifacts being published to Artifactory each month. It has made releases more predictable and reliable. The Agile Release Train has consistently delivered at least one release a month for over two years, with an average of 10 products per release, with 100% of packages delivered by the development teams being released successfully.

Monthly releases may not seem very fast, especially compared to large web companies that release multiple times per day. But when needed, one of our software development teams can build, publish, test, promote, and replicate a fix to make it available for release engineers to deploy in Production very quickly, in less than 30 minutes in most cases.

5 Patterns

The Continuous Delivery system we built is very specific to Vertafore. It had to be. When we started, there were few commercial or open source tools available for Continuous Delivery and none that we found could work with isolated environments. More tools are available now, but many still are designed for building source code in the cloud and deploying to production servers in the cloud, both of which are not options for us.

What about your company? If your company has strict business and security rules, as ours does, you may have to create a customized system. Below are a few patterns we found helpful in this effort.

5.1 Separate Build and Deployment Code

Building and deploying software are distinct tasks. The code that does these tasks should be separated as much as possible.

If you have to rebuild your product code because of a change in the deployment process, a change in an environment, or an incorrect target server name, your deployment code and product code are too closely coupled.

Getting this wrong is not a catastrophe, but it does mean your product will build more often than necessary, causing delays and wasting machine time and artifact storage. Product code can be very large, while deployment code is typically very small. The less code you have to rebuild the better.

5.2 Standardize Machine Types

Machines in a delivery pipeline need to do different types of work, such as building, deploying, and testing. It is essential to standardize machine types based on the work they need to do and to use software to ensure they have the tools needed to do that work. Without standardization, the default is to configure machines by hand, on the fly, leading to uncontrolled differences between machines that grow over time and to unexpected failures.

We have a few Jenkins machine types: build servers, deploy servers' build agents' deploy agents, generic test agents, and UI test agents. Each of these machine types is defined by different Chef recipes. When we need to update one of the machine types, we update the corresponding Chef recipe and apply it to all the machines of that type at the same time. This approach enables resilience.

Your list of machine types will surely be different, but the point here is to have a list, not to let every machine be unique in an uncontrolled manner.

5.3 Separate and Promote Artifacts

Any delivery pipeline needs a mechanism to identify artifacts that are approved for release and to prevent unapproved artifacts from being released. A simple way to do this is to keep artifacts in separate repositories, such as Development and Release.

Promotion consists of moving artifacts from Development to Release. The promotion process should automatically tag the artifacts with information about who promoted the artifacts and by what process.

Repository separation and promotion can be used with any repository manager to ensure that artifacts cannot be released unless they have passed though the steps of the delivery workflow.

5.4 Automate Promotion with Test Gates

In waterfall software development, quality gates are project milestones that exist between phases of a software lifecycle. Validation of a quality gate typically involves a high-level review of checklists to determine if the project can move on the next phase.

Test gates have a similar purpose, but are automated. A test gate evaluates test results or other output from a build and determines if a promotion criteria was satisfied. If the criteria are satisfied, the build and its artifacts are promoted through the pipeline, automatically.

What test gates would you want your software to have to pass through before being approved for release? Unit test gates, functional test gates, security gates, code coverage gates, and performance test gates are just a few of the possible gate considerations.

Any pipeline will need to have a manual gate process as well. Test gates can fail like any other automation, and they depend on automated test cases or other assets that might not exist for every product you have to move through your pipeline. Even when they exist, automated test cases may produce false failures due to incorrect assumptions in the test criteria, causing a test gate to fail even though it would have passed if the test case had been fixed. Only humans can make these type of decisions.

5.5 Artifact Retention

Any delivery pipeline will need to include a strategy for retention and cleanup of artifacts. This is driven by storage limitations, but also by performance. Some questions you will need to answer:

- How long do we keep artifacts that were never promoted?
- Should we delete artifacts that failed a test gate?
- How long do we keep artifacts that were promoted, but not deployed to Production?
- How long do we keep artifacts that were deployed to Production?

There is no right answer to any of these questions, and you probably don’t need to answer them right away, but unless you have an unlimited storage budget, you will have to answer them eventually.

Our pipeline discards un-promoted artifacts after three months. That frees up disk space while giving development teams enough time to get their artifacts tested and promoted. We keep promoted artifacts indefinitely, at least for now. If we get to the point where we have to clean those up as well, we will probably retain them for several years.

5.6 Automated Rule Compliance

In your delivery pipeline, you may want to discourage your users from doing things that your pipeline tools do not prevent, but that nevertheless are a bad idea. A simple way to do this is to have an automated script check for rule violations.

For example, we want Jenkins users to configure their jobs such that builds are cleaned up. Otherwise, our build agents get filled up with old builds. So we created a job that checks all the other jobs on our build server and sends an email with a list of jobs that are not being cleaned up properly.

We have a similar job that checks to make sure all the jobs that publish artifacts to Artifactory clean up published artifacts for jobs that have been discarded.

Another job checks to see if anyone has configured their Jenkins job to publish directly to the Release repository without going through the promotion process.

All of these rules were created because people did things that actually caused problems. Once we had automated jobs running daily to check for violations, all the violations were eliminated, with new violations appearing only occasionally, mainly due to new users not being aware of the rules.

5.7 User Engagement

When building a customized delivery pipeline, it is essential to engage the software development teams that will be using your pipeline, explaining the benefits of using it and how to use it effectively.

Our team has a series of wiki pages that explain how to use our tools, and we give periodic training sessions to new users, including classes at Vertafore's annual developer training conference. We use email lists for support requests and announcements of maintenance windows and technology upgrades.

Finally, we have governance bodies for best practices, including a Chef developers' group and a Center of Excellence for Release Engineering processes and tools.

6 Final thoughts

Delivering software to customers is not an optional function: it is the reason we are all in this business. You have to do it, so you might as well do it well. Continuous Delivery, at its core, is a commitment to continuously improving the velocity of software releases, while reducing the cost and risk of delivering software. If there are barriers in your business that prevent you from improving every part of your delivery pipeline right now, don't let it stop you from being creative and transforming what you can transform. Tomorrow, when better tools are available, you will be in a better position to understand how to deliver to your customers in the future.

7 Acknowledgments

I wish to acknowledge the contributions of everyone who has been part of our Continuous Delivery journey: Raul Alvarez, Cody Dockens, Saratha Prabu, Rajani Tadanki, David Echols, Rasheed Usman, Lowell Young, Denzil Dwelle, Cameron Straka, Ebencilin Chandradhas, Venkat Chilakala, Tom Sawin, Rita McCann, and George Tsang.

I also acknowledge our colleagues from Ernest Mecham's group at our East Lansing, Michigan office. They have been on their own Continuous Delivery journey, using Bamboo instead of Jenkins, but otherwise overcoming many of the same challenges we have, with excellent results.

References

Jez Humble: The Case for Continuous Delivery, 13 Feb 2014
https://www.thoughtworks.com/insights/blog/case-continuous-delivery

Scaled Agile Framework
http://www.scaledagileframework.com/

Artifactory Build Sync User Plugin
https://github.com/JFrogDev/artifactory-user-plugins/tree/master/build/buildSync