How Cisco IT Developed a Self-Service Model for Build and Deploy

Automating application delivery speeds up the pace of innovation and saves 32 developer hours daily. It's fast IT.

EXECUTIVE SUMMARY

CHALLENGE

- Innovate faster
- Relieve developers and QA teams from manual processes
- Improve work-life balance for IT operations engineers

SOLUTION

- Developed self-service solution for application build and deploy
- Created a dashboard that presents real-time build and deployment status

RESULTS

- Cut build time by at least 60 minutes, freeing up 32 developer hours daily
- Eliminated limits to the number of daily deploy jobs. Developers can start a build at any time.
- Reduced QA workload related to deployment by half

LESSONS LEARNED

- Report up-to-date build status in the dashboard to keep team members informed
- If a deployment fails, automatically notify both Dev/QA and DevOps
- Set up a feedback loop with the goal of continuous improvement

NEXT STEPS

- Introduce the process to other Cisco development teams

Challenge

The Cisco® Cloud and Software IT (CSIT) team had been using the same build and deployment processes for years. Developers sent an email to inform Quality Assurance (QA) when applications were ready for test. QA opened a case and emailed IT operations to request a build. IT operations sent an email confirming the build was ready.

The delays were slowing us down, stifling our goal of fast IT.

"The previous build and deploy process was too complex," says Jingmin Ding, Cisco IT manager. "Email, remedy tickets, manifest files, and wiki updates are boring and take too much manual effort."

The amount of time spent on routine tasks also affected job satisfaction for Development and Operations (DevOps) and QA engineers.

Standard continuous delivery (CD) tools and processes wouldn't fully meet our needs because of our diverse application environment. Deliverables include Java programs, web content, and data objects. Compilation tools include Ant, Maven, and Make.

To continue on our CD journey, we decided to develop a self-service system to automate build and deploy processes.

Solution

Now CSIT application-delivery teams release code into production without any help from IT operations. Our new self-service model automates much of the process. It's based on:

- A code commit and branching model in Jenkins
- Automated build pipeline and workflow
- Automated build tracking
- A metrics dashboard, used to continually identify new areas for improvement

Code Commit and Branching Model

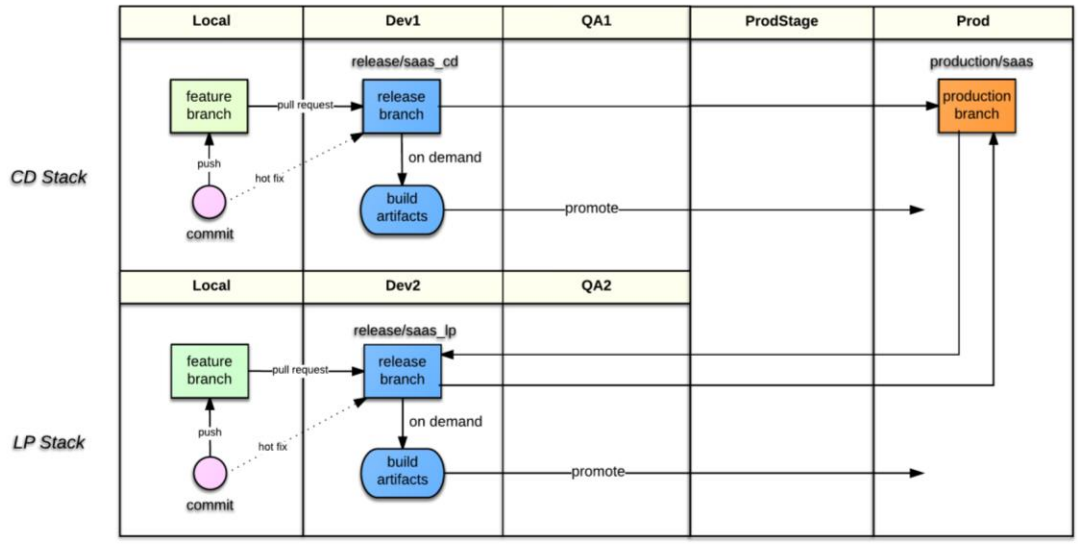
Jenkins is our open-source continuous integration (CI) tool. Previously we used a Jenkins Git server plugin for version control. Developers entered all changes through a Concurrent Versions System (CVS) repository. Later we switched to a Git Stash repository. Stash makes it easy to review code using pull requests, which encourages peer review and team conversations about code.

To define our build pipelines, we created the three types of Jenkins branches (Figure 1):

- Feature branch: Developers develop each feature in its own branch. When developers are ready for their teams to review the feature, they push the branch to the remote Stash repository. With this approach, developers can share a feature without touching the official code.
- Release branch: After completing a feature, developers merge their changes to a release branch by filing a pull request. The Jenkins job pulls code from the release branches during build. The advantage of this approach is that the release branch contains reviewed and stable code.
- Production branch: To release the code to production, we merge the release branch to the production branch. The production branch always contains the latest code in the production environment. If we update the code in the production branch, we also update it in the release branch.

Figure 1 shows our two development stacks: CD and long project (LP). The CD stack is for features introduced throughout the year. The LP stack is for features introduced in our major, twice-yearly releases.

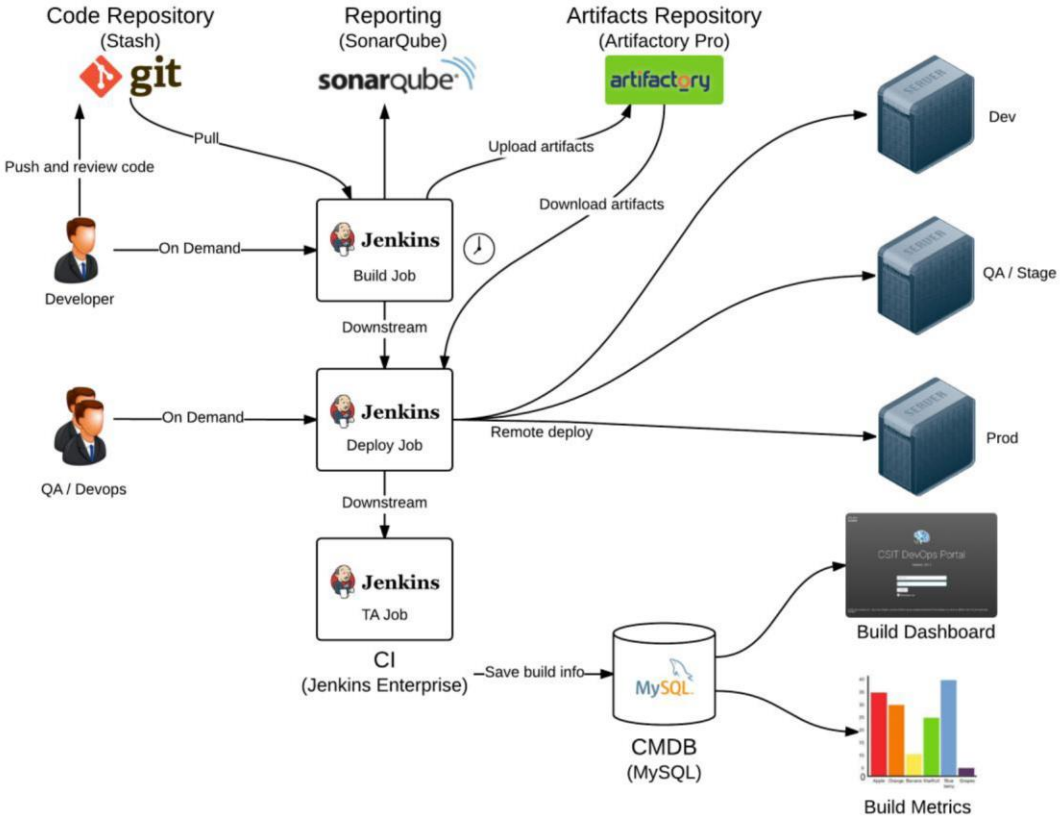
Figure 1. Build Pipeline Branches: Feature, Release, and Production



Automated Build Pipeline and Workflow

To deploy an application, developers and QA engineers visit a web dashboard to trigger the Jenkins build, which initiates the automated process (Figure 2). Jenkins begins by compiling and packaging the code. Then it automatically deploys the package to the development, QA, and production environments. IT operations engineers don't have to get involved.

Figure 2. Self-Service Model for Application Deployment



We've defined three types of Jenkins jobs:

- Build job: Only developers can start a build job. The job compiles the source code from a release branch. Jenkins assigns a unique build number to the generated artifacts and automatically uploads them to the artifact repository. When complete, the build job creates the deploy job.
- Deploy job: Both QA and DevOps teams can start a deploy job. (QA can deploy only to the QA environment, not production.) Before starting a deploy job, the user specifies the target environment and the build number in the artifact repository. The job retrieves the specified build number and deploys it to the target environment.
- Test automation (TA) job: The TA job begins automatically after the deploy job is complete. The job invokes the automated testing scripts provided by the DevOps and QA teams.

Tracking and Monitoring Builds

Developers check out code and retrieve artifacts from the development build. Before, developers had to manually enter the build number in Jenkins jobs and on our Configuration Management wiki page. But manual entry is time-consuming. Errors can cause deployment failures.

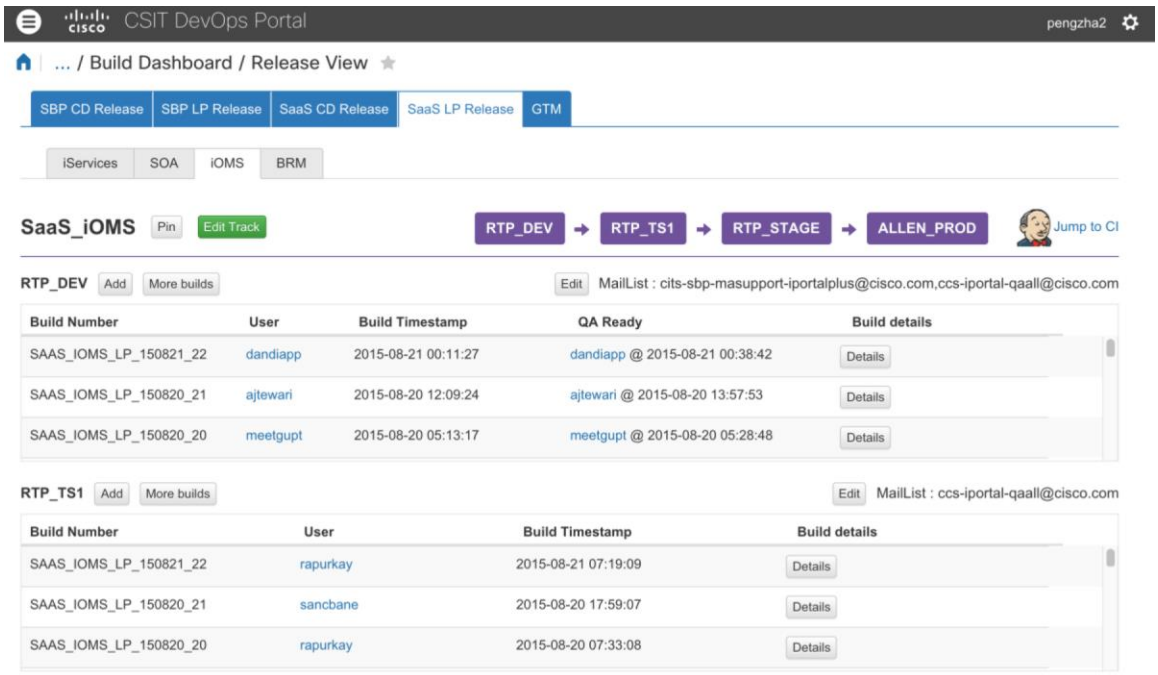
We've automated build tracking using this process:

- The development build generates a build number, for example, IOMS-LP_150824_31. Our naming convention is name of project; whether it's a CD or LP release; build date; and Jenkins job number.
- All artifacts in the development build are compressed in a tar.gz file. Jenkins uploads the compressed file to the artifact repository. The build number is the same as the version number.
- The development build passes the build number to the deploy job, which can then retrieve the correct version of the compressed artifacts file.
- A custom Jenkins plugin saves the build information (parameters and more) to our configuration management database (CMDB).

This process completely eliminated build errors.

We also developed a portal to track the status of each build. The portal shows who deployed the build, when, and in which environment (Figure 3). To notify the QA team that a build is ready for testing, the developer simply sets a flag on the portal. There's no longer any need to send emails.

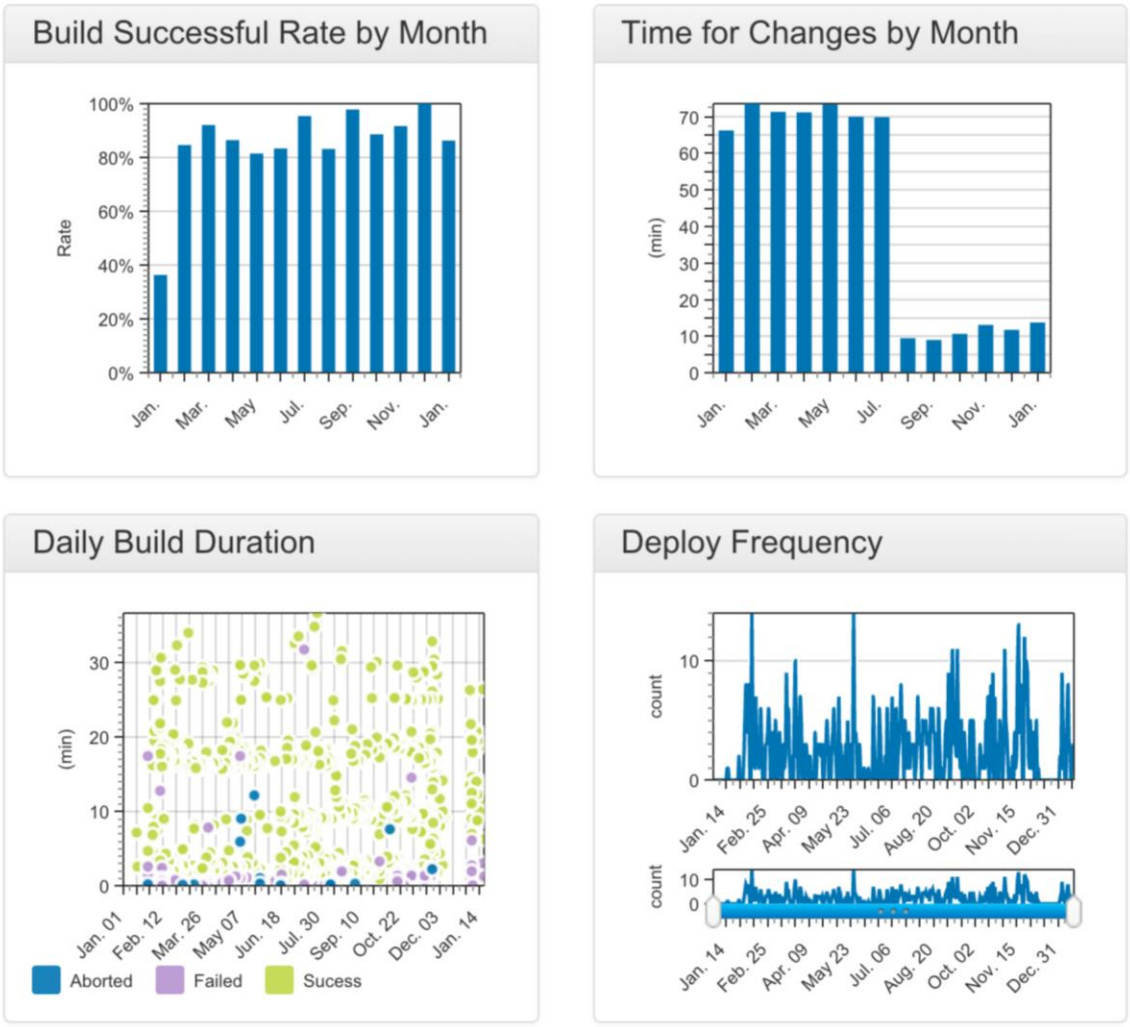
Figure 3. Real-Time Build Status Dashboard



Tracking Metrics

We collect metrics from the CMDB and present them on a dashboard in graph form (Figure 4). Tracking these metrics helps us discover ways to continually improve the process. We monitor successful builds by month, deployment frequency, and build duration. The build duration graph also shows whether builds succeeded, failed, or were aborted.

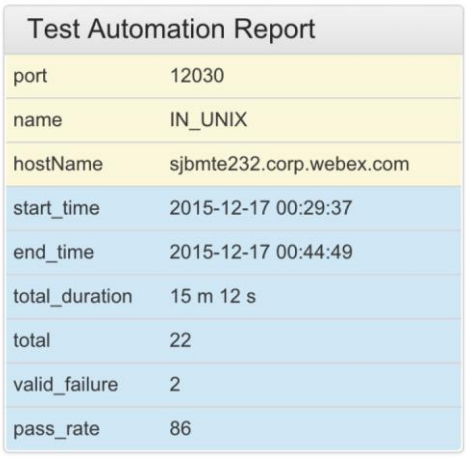
Figure 4. Easy-to-Read Graphs Help Track Success



Test Automation

The TA job generates a report that appears in the dashboard (Figure 5). Team members find out the test results immediately. They don't have to wait until someone sends an email.

Figure 5. Test Automation Report



Results

All CSIT developers and QA engineers now use the self-service platform for source-code management, continuous build, and storing artifacts. The self-service process speeds up the pace of innovation and lowers costs:

- Fast IT: Before we could deploy a maximum of 32 builds daily. Now there are no limits to the speed of innovation. "There is no more dependency on other teams," says Manjunath Nanjappa, CSIT software engineer. A simpler delivery process means we can introduce more features, faster. The number of feature branches has quadrupled.
- Lower development costs: Builds finish at least 60 minutes faster, which saves 32 developer hours every day. Engineers can use this time either for work or to reduce overtime, improving work-life balance. "We saved 1648 hours in the first 6 months," says Kyo Song, Cisco DevOps service lead.
- Higher productivity for developers and QA: "The self-service process cuts QA workload by about half," says Ding. We receive automated email confirmation when a build is ready, and then just click to deploy. We execute tests after we receive the automated email that the build is complete. "Supporting CSIT daily build and deployment tasks used to take DevOps staff away from more critical activities," says Jagdish Bhandary, program manager for the QA team. "The self-service model frees time for developers and QA to focus on activities with higher value for the business."
- Improved quality: By reducing opportunities for human error, automation has increased the rate of successful builds. Developers and QA engineers can detect defects sooner thanks to TA integration and real-time feedback.
- Continual improvement: We can see as code moves between tests and build environments. "Collecting and analyzing near-real-time application lifecycle management [ALM] data helps the CSIT team measure success," says Song.

Next Steps

Now we're working to automate more of the application and infrastructure lifecycle. That includes automating infrastructure provisioning, orchestration, and deployment. Our next initiative will be "infrastructure as code."

Lessons Learned

The CSIT application delivery team offers the following tips for other organizations that want to introduce self-service build and deployment:

- Create a dashboard with real-time updates on build and deployment status. This step is especially important in the early stages of adoption.
- If a process fails, resolve the issue quickly. This practice, too, encourages adoption.
- Encourage communication and collaboration between DevOps and QA. Set up a feedback loop with the goal of continuous improvement.
- Create a FAQ document.

For More Information

To read Cisco IT case studies about a variety of business solutions, visit Cisco on Cisco: Inside Cisco IT http://www.cisco.com/go/ciscoit.

To view Cisco IT webinars and events about related topics, visit Cisco on Cisco Webinars & Events.

Note

This publication describes how Cisco has benefited from the deployment of its own products. Many factors may have contributed to the results and benefits described. Cisco does not guarantee comparable results elsewhere.

CISCO PROVIDES THIS PUBLICATION AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow disclaimer of express or implied warranties; therefore, this disclaimer may not apply to you.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**