Continuous Integration Tools

Applying Best Practices to the Toolchain
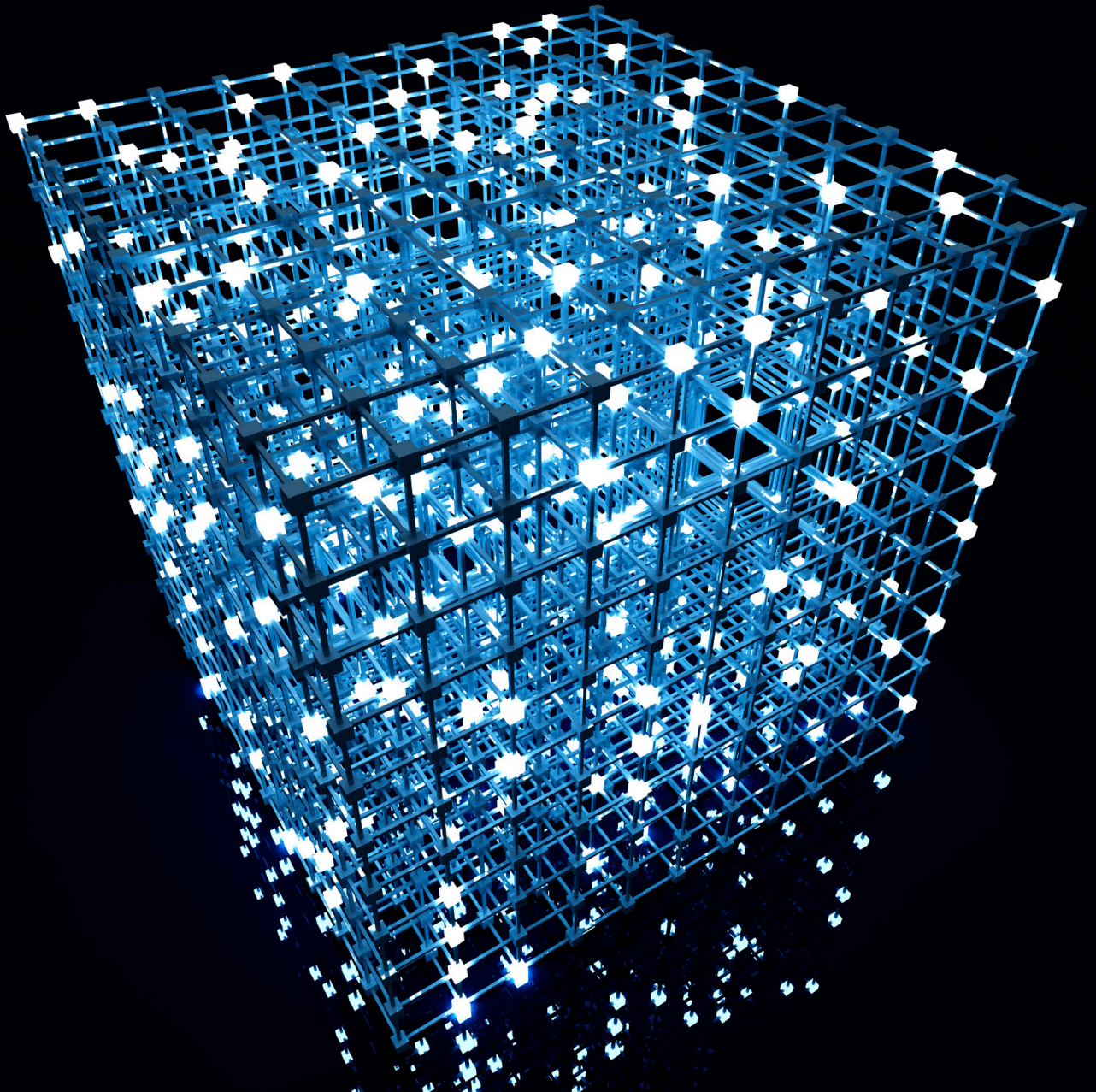
Table of Contents

Automated testing, using SOASTA CloudTest and TouchTest, is a central component of CI automation.

Introduction

The Principles of Continuous Integration white paper introduced key elements of effective continuous integration for mobile development. In order to reap the speed and quality benefits promised by CI, it's important to understand and apply these practices. It's not enough just to stand up a Jenkins server.

Continuous integration depends on the proper use of tools. Automation is critical to reducing waste and to increasing delivery speed by removing opportunities for manual errors. Automated testing, using SOASTA CloudTest® and TouchTest™, is a central component of CI automation.

Continuous Integration

A continuous integration toolchain that addresses the key best practices typically integrates several classes of tools. These tools fall into the following categories:

- Device test automation
- API/Web test automation
- Version control
- Build automation
- Artifact management
- Configuration automation
- Test clouds
- Device clouds

Device Test Automation

Continuous integration depends on the ability to turn changes around quickly. Changes involve not only code, but also tests. CI also depends on comprehensive regression test coverage. Neither of these dependencies is compatible with manual test execution. Without a way to write tests as quickly as developers write code, QA teams fall behind and become overwhelmed by ever-growing test case backlogs.

SOASTA TouchTest enables rapid, easy development of mobile device tests. Its support for integration with build automation tools enables automated execution of regression suites.

API / Web Test Automation

The same concerns that apply to device testing also apply to server application testing. SOASTA CloudTest supports rapid, easy development of automated SOAP/REST API and interactive website tests. Its support for integration with build automation tools enables automated execution of server application regression suites.

Version Control

Continuous integration is fundamentally about integration, requiring a reliable mechanism for folding together code changes from multiple developers. A version control system (VCS) such as Git, Mercurial, Subversion or Microsoft Team Foundation Server serves this purpose.

In an environment where multiple developers are simultaneously changing a shared code base, it is critical to be able to control and understand the flow of those as they leave individual developers' machines and move towards production. In addition to basic check-in/checkout functionality, a VCS must support easy branching, merging and tagging.

Build Automation

Build automation is at the center of the continuous integration universe and defines the overall integration pipeline. A typical pipeline includes:

1. Retrieve code from version control system
2. Build code and run unit tests
3. Store built binaries in artifact management system
4. Deploy binaries to test environment
5. Run functional and non-functional tests (security, performance, load, availability, etc.)
6. Promote passing builds to subsequent environments
7. Generate build and test reports
8. Distribute build status notifications

Jenkins is the de facto standard open source build automation platform. Bamboo and Anthill Pro are commercial products with similar functionality.

Artifact Management

Rebuilding artifacts, such as application binaries, is wasteful as well as a potential source for hard-to-diagnose errors. Being able know at all times which version of software is in play greatly aids the ability to answer the question "what changed"?

An artifact management system such as Artifactory, CocoaPods or Nexus provides a versioned repository for build artifacts. Build automation platforms and deployment automation tools should tightly integrate with an artifact repository. The output of each successful build should be stored in the repository. Deployment automation should subsequently use the repository as its input.

Configuration Automation

Just as it's important to control application binary versions, it's also important to control test environments. "It worked on my machine" is perhaps the most frustrating bug that development organizations encounter. Fortunately, this class of bug can be avoided using configuration automation.

It-worked-on-machine bugs (also known as "it worked in test" bugs) generally result from configuration differences between environments. Manual configuration gives rise to situations where developers use different versions of an SDK, or test uses different OS patches or web servers from production, etc. Configuration automation tools such as Chef and Puppet turn infrastructure into code that can be versioned and automatically built, just like applications. They make it possible to update a configuration script in one place, and have that script automatically update every affected environment, from a developer's desktop to test to production. In a load-balanced server environment, configuration automation also ensures that all server instances are identically configured.

Test Clouds

Continuous integration relies on continuous testing. Ideally, developers pair with testers to validate changes in local environments prior to check-in. These activities all need to run in controlled environments. The simplest way to control test environments is through a set of centralized environments: the traditional Dev, Test, Load, etc.

Centralized environments make control easier, but create speed and flexibility bottlenecks. On-demand cloud computing, coupled with configuration automation, enables test environments that are flexible yet still controlled.

The test cloud approach can start right on the developer's desktop. Vagrant is a tool that incorporates configuration automation into desktop virtualization. A developer can test server application changes in multiple desktop VMs, while being assured that each VM will sync itself with the same Chef or Puppet scripts used to manage higher environments.

Continuous integration for server applications defines a potentially lengthy test pipeline. Ideally, that pipeline can be parallelized so that, for example, one change undergoes security testing while another undergoes unit testing. This approach requires that the build automation tool has simultaneous access to multiple server instances. Using an on-demand cloud is an efficient and cost-effective way to implement parallel builds. Clouds can be public (AWS, Rackspace, Azure, etc.) or private (OpenStack, CloudStack, etc.)

Device Clouds

Mobile device testing requires access to physical devices. These devices need to be tethered to a physical server such as a Mac Mini. SOASTA's TouchTest Mobile Labs provide cloud-based or on-premise device clouds that give developers and testers access to multiple versions of multiple device types.

About SOASTA, Inc.

SOASTA is the leader in cloud testing. Its web and mobile test automation and monitoring solutions, SOASTA is the leader in cloud testing. Its web and mobile test automation solutions, SOASTA CloudTest and SOASTA TouchTest, and SOASTA mPulse, enable developers, QA professionals and IT operations teams to test and monitor users with unprecedented speed, scale, and precision, and visibility. The innovative product set streamlines test creation, automates provisioning and execution, and analyzes real user behavior in real time distills analytics to deliver actionable intelligence faster. With SOASTA, companies can have confidence that their applications will perform as designed, delivering quality user experiences every timeeven in peak traffic. SOASTA's customers are many of today's most successful brands including Avaya, American Girl, Bonobos, Backcountry.com Chegg, Experian, Gilt Groupe, Hallmark, Intuit, Microsoft, and Netflix. SOASTA is privately held and headquartered in Mountain View, Calif. For more information about SOASTA, please visit http://www.soasta.com.

Connect with us:

TEST FASTER. RELEASE SOONER.