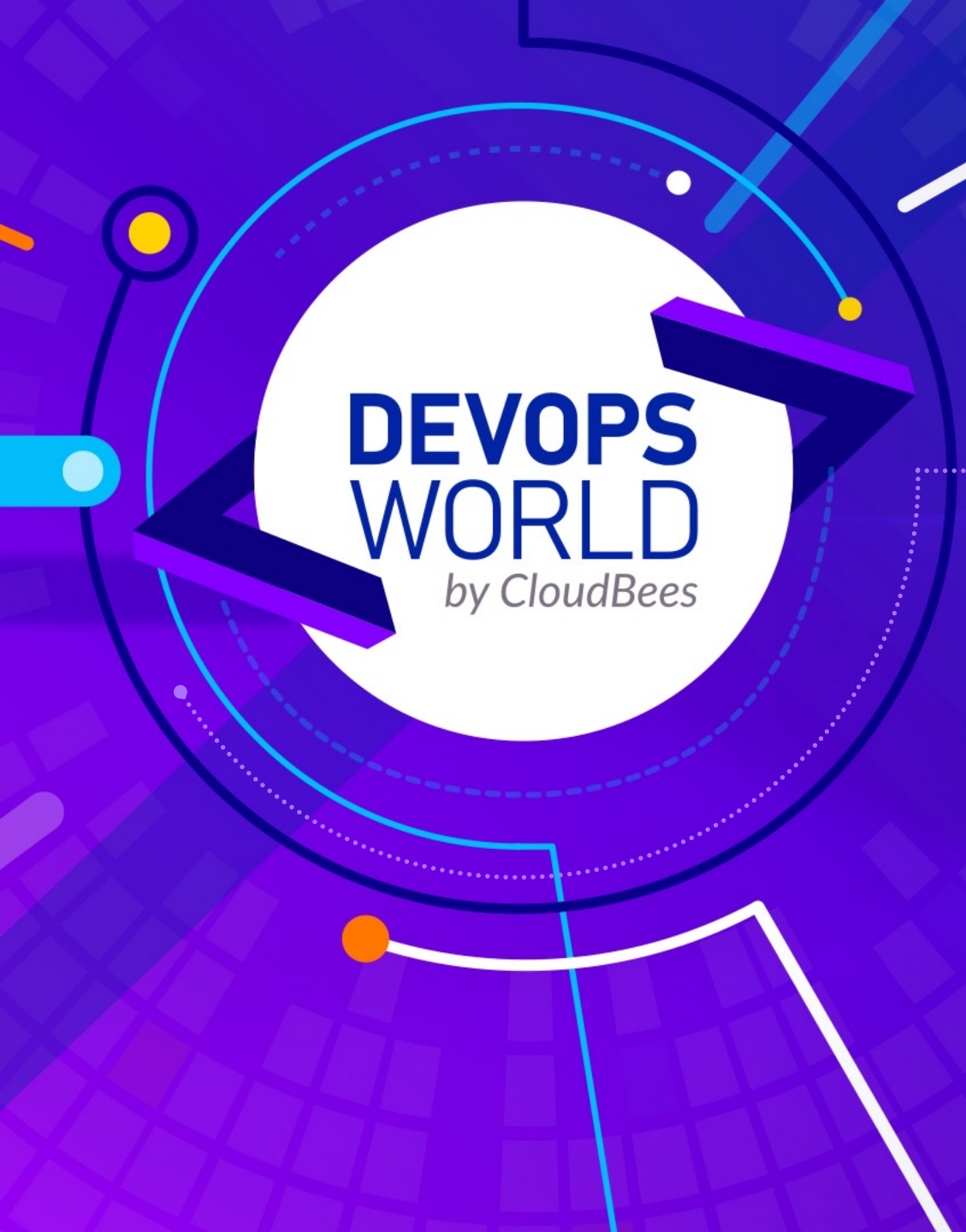


CDF

A Purely Functional CI/CD Pipeline Using Jenkins with Guix

Philip Beadling

ENTERPRISE ARCHITECT – QUANTILE TECHNOLOGIES

What's Covered?

- Traditional Approaches / Motivating Example
- What Jenkins is and isn't.
- What Gnu Guix is and isn't.
- Integrating Jenkins and Guix - Division of Labor and avoiding overlap.
- Using Jenkins' Locks to Maximize Opportunity for Parallelism.
- Reproducing Production using Unprivileged Local Package Management

CDF

Traditional Approaches / Motivating Example

Why bother with Guix?

Traditional Approaches

- Quantile had a previous Jenkins CI system
- Designed for a much smaller company
- Largely written in Bash (>2000 lines)
- Not hooked up to Bitbucket – no event listening / minimal API use
- Packages delivered as zipped repos – no dependency management
- No common tooling across projects – Python/C++/etc
- Difficult to scale to new languages and tooling requirements
- Infrastructure opaque to development teams

What Alternatives Were Trialled/Considered?

- Stick with Jenkins – but leverage Groovy DSL
- Using Shining Panda for CI Management
  - Integrates virtual environments into Jenkins pipelines
  - Python only
- Using Make/CMake as Common Build Language
  - Python projects can produce self-contained Debian packages using dh-virtualenv
  - C++ projects can use checkinstall
- Use aptly / Local Debian Repository for Package Management
- Use Ansible for Package Deployment
- Use Docker for Isolation

What's Wrong With This?

- Typically > 7 different tools needed to build your pipeline and host your code
  - Tools vary from one language to the next, even when using a common packaging mechanism
  - Maintainers must learn each tool in order to support the whole pipeline
  - Different tools can massively overlap in scope – blurring each use-case
- No common single DSL to describe the process end-to-end
  - Make is low-level and varies greatly between different language builds
  - Jenkins/Groovy too general to describe the detail in each step
  - Ansible only describes the deployment, etc…
- Build inputs are not stateless, nor isolated
  - We want guaranteed reproducibility when we build and install our software on any server
  - We want to be able to roll forwards or back, quickly on any server

What Would Be Better?

- A single tool that could describe dependencies, building, testing, packaging, and installing software using a single common high-level DSL
- Jenkins would still manage the overall workflow – build triggers, comms, locking of resources, UI, reporting, etc
- Jenkins would defer to this new tool by passing in a single description written in a high-level DSL for the project that needs to be built
- These descriptions would be stored and accessible locally for internal packages
- 1000s of publicly available off-the-shelf packages in a variety of languages
- Off-the-shelf packages can be easily extended and repurposed to suit any very specific requirements for our own build system

CDF

Lightning Intro to Jenkins and Guix

What they are and what they are not

DEVOPSWORLD
by CloudBees

What is Jenkins?

- Jenkins helps automate the processes of building, testing, and deploying software
- It is a management framework rather than a complete off-the-shelf solution
- It offers solutions to common tasks such as cloning repositories, handling PR events, reporting artifacts and failures, and a web-based UI
- It has a wide variety of plugins available to cover many scenarios and tools
- It has a nice extensible DSL from describing a delivery pipeline
- **BUT**… Jenkins doesn't know how to build or test much out-of-the-box
- It defers this to plugins – which tend to be language or setup focused
- We'd like to have a single tool that handles all scenarios for Jenkins

What is Guix?

- GNU Guix is a X-platform package manager, but goes beyond traditional offerings
- It handles source control, patching, building, testing, packaging, versioning, and serving of packages to production and developer envs using containerization
- It does this through its own extensible DSL which provides a common language for describing the building and testing in a high-level fashion
- It is purely functional and transactional, guaranteeing an identical output from the same set of inputs and never to screw-up the system if it fails mid-install
- It provides a method of quickly rolling backwards or forwards in the event of a software issues
- It uses an unprivileged model so each user can have many environments

What is Guix?

- **BUT** – Guix isn't a CI system designed to integrate into a typical developer workflow
  - It does come bundled with a simple CI tool called Cuirass but it's not as feature-rich as Jenkins and isn't covered here
- It won't tightly integrate to products such as BitBucket and build on events such as PR approval or merges
- It won't provide queuing/throttling of pipeline jobs
- It won't provide reports and notifications
- It won't provide the rich pipeline visualization and UI experience of Jenkins

Jenkins and Guix Together

- Jenkins and Guix look to be a good match
  - Jenkins is the pipeline manager
  - Guix provides a lower-level abstraction across packaging scenarios
- There is still a small amount functional overlap (see later)
- Jenkins will handle SCM events, job queuing, reporting and UI
- Jenkins will defer handling the building, testing, versioning, and deploying to Guix using simple Guix commands
- Next we'll discuss the key details in having a smooth handover between tools

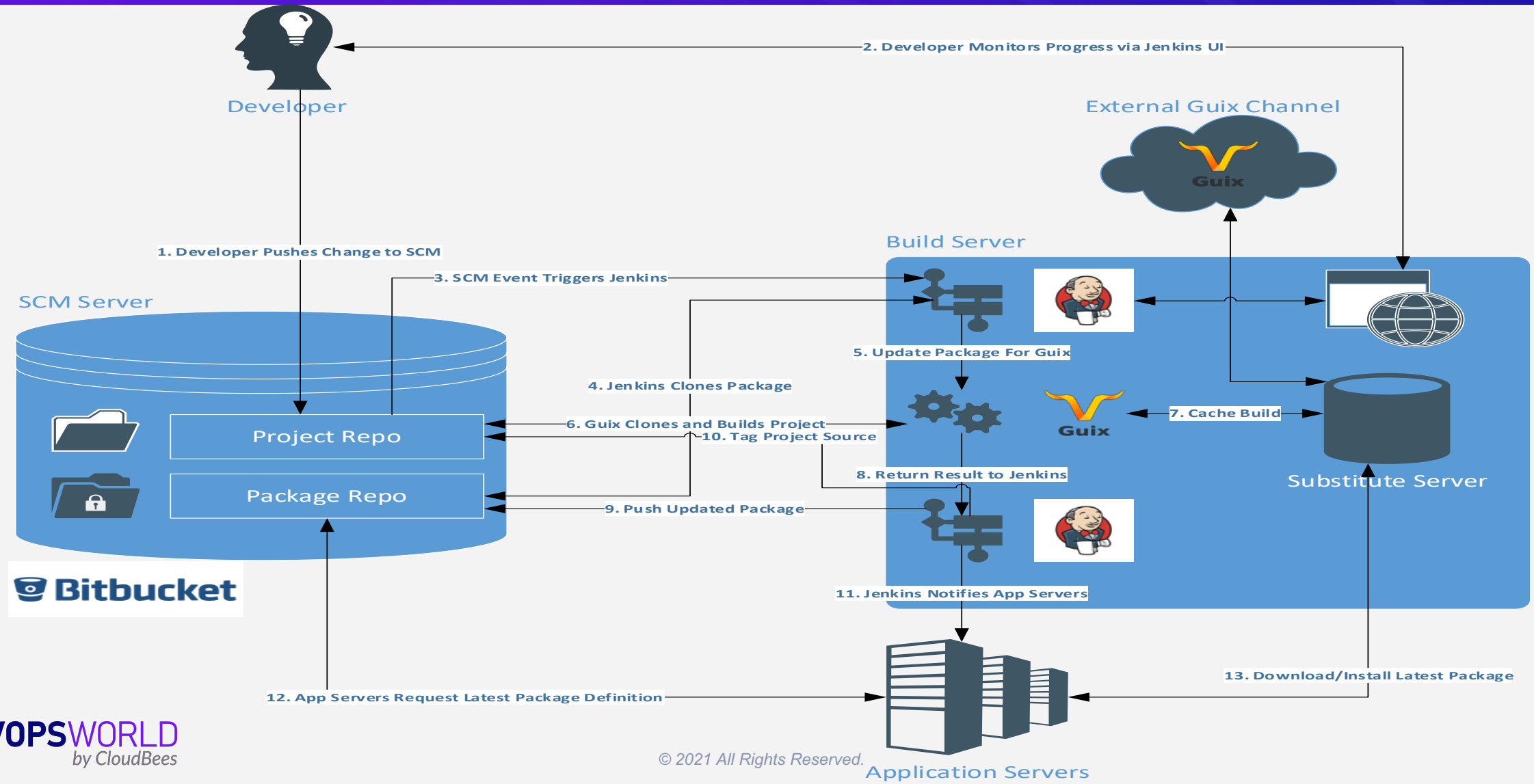
CDF

Integration of Jenkins and Guix

Division of Labor and not stepping on each other’s toes

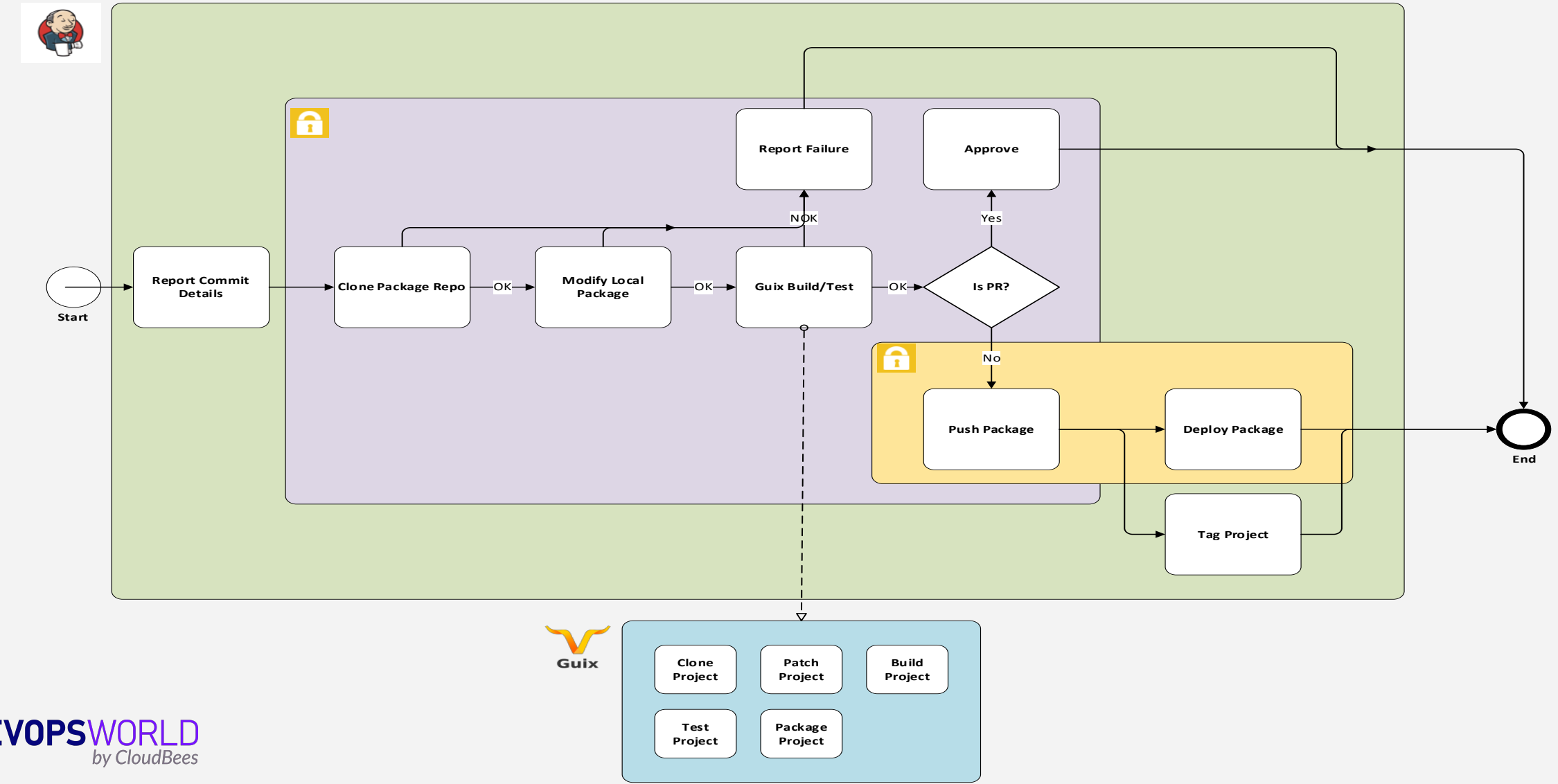
Integrating Jenkins and Guix

System Overview



Integrating Jenkins and Guix

Pipeline Overview



Integrating Jenkins and Guix

Preventing overlap of SCM

- By default Jenkins automatically clones the source to be built
- However Guix requires it’s own clone inside it’s build container
- So we disable clone in Jenkins

```
pipeline {
    agent any
    options {
      // We defer clone of the actual source to Guix.
      skipDefaultCheckout()
      …
```

Integrating Jenkins and Guix

Getting Commit Information

- Without a local clone under Jenkins' control, we get commit details ourselves
- Git supports a remote call for this
- But config on the Jenkins box needs to be updated to include PR references

```
def getTriggeringCommit(project, repo, branchName) {
  // git config --global --add remote.origin.fetch '+refs/pull-requests/*/merge:refs/remotes/origin/pr/*'
  def refs = branchName.startsWith('PR-') ? "refs/pull-requests/${branchName[3..-1]}/merge" : "refs/heads/${branchName}"
  return sh ( script: "git ls-remote ssh://git@bitbucket:7999/${project}/${repo}.git ${refs} | cut -f1", returnStdout: true).trim()
}
```

Integrating Jenkins and Guix

Example Guix Package

```
(define major "1")
(define production-rev "0")
(define hotfix-rev "0")
(define production-version (string-append major "." production-rev "." hotfix-rev))
(define-public symphony-bot-production
  (let ((commit-production "14591bf05952a802dac7ee2d79acc61cb91ceg43"))
    (package
      (name "symphony-bot")
      (version production-version)
      (source
       (git-checkout
        (url "ssh://git@bitbucket:7999/ea/symphony-bot.git")
        (commit commit-production)))
      (build-system python-build-system)
```

```
      (propagated-inputs
       `(("python-redis" ,python-redis)))
      (native-inputs
       `(("python-black" ,python-black)))
      (arguments
       `(#:tests? #f
         #:phases (modify-phases %standard-phases
                    (add-before 'build 'pre-commit-verification
                      (lambda _
                        (let ((python-files (find-files "." "\\.py$")))
                          (apply invoke "black" "--check" python-files))
                        #t)))))
      (home-page "https://bitbucket:8443/projects/EA/repos/symphony-bot/browse")
      (synopsis "Symphony Bot Production")
      (description "Quantile's Symphony API to send message notifications")
      (license quantile))))
```

Integrating Jenkins and Guix

Simple Example of Building a Guix Package in Bash

- Guix Packages are stored in a Git Repo
- We clone the package repo and update the package and commit locally
- We then tell Guix to build the package with our local definition

```
git clone ssh://git@bitbucket:7999/ea/guix-packages.git
cd guix-packages
sed -i -r 's/(.*\(define production-rev ")([^"]*)(.*)/echo \"\1\\\"\$((\2+1))\\\"\3\"/e' symphony-bot.scm
sed -i 's/\(.*(commit-production "\)[^"]*\(.*\)/\1abcdefg\2/' symphony-bot.scm
git add symphony-bot.scm && git commit -m "Update symphony-bot"
guix build -L /path/to/git/clone/guix-packages symphony-bot
if [ $? -eq 0 ]; then
 git push
fi
```

Integrating Jenkins and Guix

Simple Translation to Jenkins DSL

```
def guixBuildAndTest(pkgName, pkgVar, buildId) {
  sh """#!/bin/bash
        set -eo pipefail
        trap "pkill -eu \$USER ssh-agent" EXIT
        GUIX_PROFILE="\$HOME/.config/guix/current"
        . "\$GUIX_PROFILE/etc/profile"
        eval `ssh-agent` && ssh-add
        guix archive --export -L \$(realpath ./guix-packages/packages) -e '(@ (${pkgName}) ${pkgVar})' | gzip > ${pkgVar}-
${buildId}.nar.gz
        guix describe -f channels > ${pkgVar}-${buildId}-channel-guix.scm"""
}
```

Integrating Jenkins and Guix

Installing Newly Built Software

- Guix works on a “pull” model
- So each server must ask Guix for the updates for its manifest
- We can trigger this from Jenkins by simply ssh’ing onto each server via a loop
- The basic commands to run on each server in our example are below.
- The first command updates the package definitions to latest
- The second command installs the contents of the manifest in the profile

```
guix pull
guix package -m ${HOME}/guix-manifests/symphony-bot-manifest.scm -p ${HOME}/guix-profiles/symphony-bot-profile
```

Integrating Jenkins and Guix

Simple Translation to Jenkins DSL

```
def newPackageNotification(loginList, pkgVar, pkgName, buildId) {
  sh """#!/bin/bash
        for server in ${loginList}; do
          echo "Deploying to \$server"
          ssh -o ConnectTimeout=5 \$server bash <<-ENDSSH &
            export LC_ALL="en_US.UTF-8"
            GUIX_PROFILE="\\\$HOME/.config/guix/current"
            . "\\\$GUIX_PROFILE/etc/profile"
            eval \\\$(keychain --agents ssh --eval id_rsa) && guix pull && guix package -m "\\\${HOME}/guix-manifests/${pkgVar}-manifest.scm" -p "\\\$HOME/guix-profiles/${pkgName}-profile"
            echo "Return code from \$server install: \\\$?"
            echo '******* The following packages are now installed in the profile of \$server *******'
            guix package -p "\\\$HOME/guix-profiles/${pkgName}-profile" -I
            echo '**********************************************************************************************'
ENDSSH
        done
        echo 'Waiting for deployments...'
        wait"""
```

Integrating Jenkins and Guix

Parallelism and Jenkins Locks

- Jenkins provides a system of locks that can be used to restrict jobs in parallel
- For Guix we can only have Jenkins modify a single package (file) at once
- From package repo clone to package repo push we lock for a single project
- Other projects can build in parallel because the merge will be fast-forward
- Each deployment must be exclusive across all projects
- Deployments can run in parallel with package modifications
- Git push is also locked during a deployment
  - A long-queued deployment could otherwise technically ‘git pull’ the release after itself

Integrating Jenkins and Guix

Locking Framework

```
pipeline {
    stages {
      stage('Package Modification Lock') {
        options {
          lock("${pipeParams.projectKey}_${pipeParams.repoSlug}_modify_package_lock")
        }
        stages {
          stage('Clone The Package Repo') {
          }
          stage('Modify Local Package') {
          }
          stage('Guix Build/Test') {
          }
  stage('Push Package to Guix Channel') {
      options {
        lock('deploy_package_lock')
      }
    }
  }
}
stage('Deploy Candidate') {
    stage ('Notify App Servers To Deploy') {
      options {
        lock('deploy_package_lock')
      }
      stages {
        stage ('Integration Deploy') {
        }
      } } } } }
```

CDF

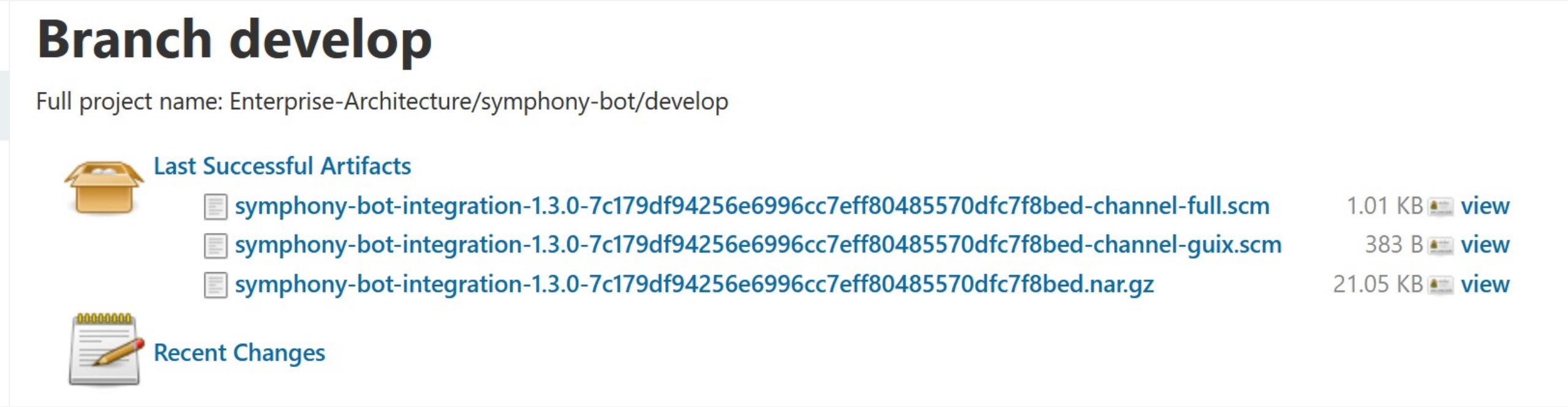
Example: Reproducing Production

Unprivileged Local Package Management

Reproducing Production

Artifacts in Jenkins

- Previously I showed a ‘guix describe’ command
- This saves the state of Guix and is easily made available as an artifact in Jenkins



Reproducing Production

Example Guix Channel Artifact

```
(list (channel
        (name 'guix-packages)
        (url "ssh://git@bitbucket:7999/ea/guix-packages.git")
        (commit
          “e17c7bb47986aedfe69b549185e14dede949649a")
        (introduction
          (make-channel-introduction
            “7e1634252f3be90efaadf59fdab42697d3f49998"
            (openpgp-fingerprint
              “1234 5678 9ABC DE00 1234  5678 9ABC DE00 1234 5678"))))
```

```
 (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (commit
          "675540892719387e1e4e76f097ff8e4ee4b559f7")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA")))))
```

Reproducing Production

Guix Channels, Manifests, Profiles, and Time-Machines

- A Channel is a single snapshot of the Guix package repository
  - It describes every Quantile package, every Guix package, and Guix itself completely at a point in time.
- A Manifest is a collection of packages to be installed
  - Crudely it's like a requirements.txt in Python
- A Profile is a location to install a Manifest
  - Crudely it's like a virtual environment in Python (but it has transactional history)
- If I take a Channel at the point of a Jenkins build, and the project Manifest
- We can create a local-user byte-identical Prod Environment from that point in time:

```
guix time-machine -C symphony-bot-channel-full.scm environment -m
/home/guix/manifests/symphony-bot-manifest.scm
```

More Information

- Jenkins
  - https://www.jenkins.io/
  - https://www.oreilly.com/library/view/jenkins-2-up/9781491979587/

- Guix
  - https://guix.gnu.org/
  - https://guix.gnu.org/en/cookbook/en/guix-cookbook.html
  - https://guix.gnu.org/en/manual/en/guix.html
  - https://github.com/guix-mirror/guix
  - https://lists.gnu.org/mailman/listinfo/info-guix
  - https://scheme.com/tspl4/

Thank you!

Philip Beadling

ENTERPRISE ARCHITECT – QUANTILE TECHNOLOGIES