SESSION NUMBER ARIAL

# Managing DevSecOps Pipelines at Scale With the Jenkins Templating Engine

**Steven Terrana**

*CDF Ambassador*
*Senior Lead Technologist at Booz Allen*

DEVOPS WORLD by CloudBees

# So You Want to Build a Pipeline

## Let's Talk About the Realities of Large-Scale Pipeline Development

DEVOPS WORLD
by CloudBees

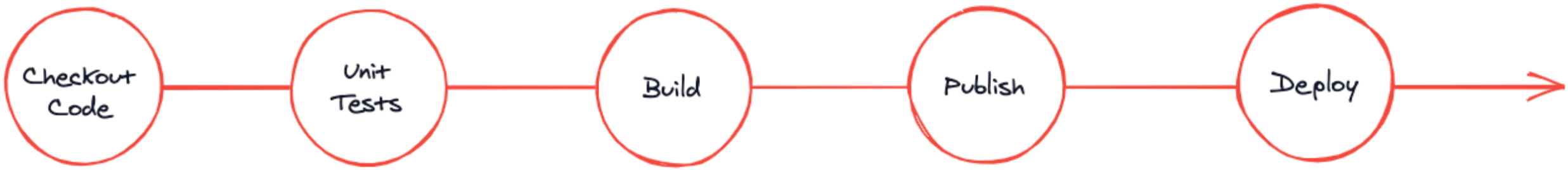
# ”Just draw the rest of the owl!”

Building a pipeline
for one application

Building a pipeline
for multiple applications

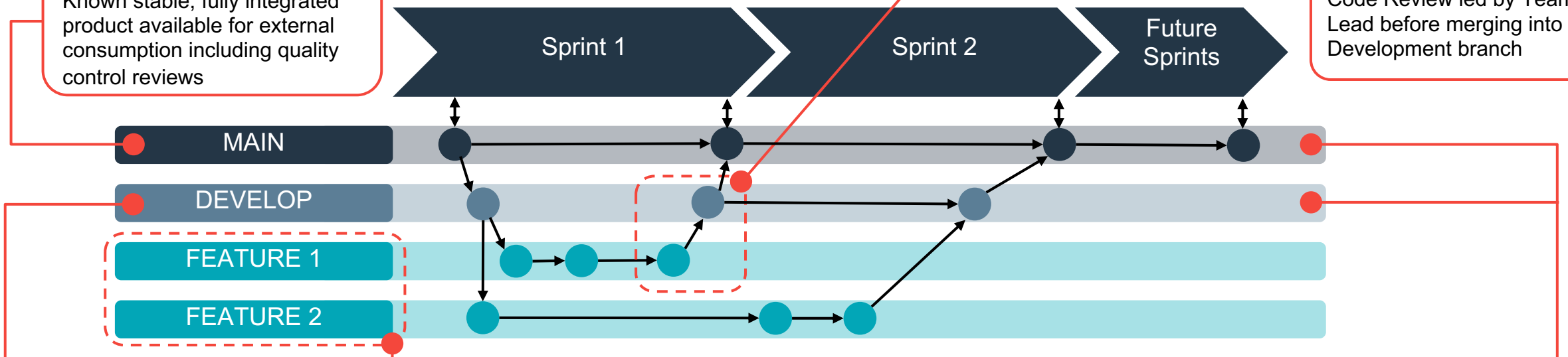
Building a pipeline
for multiple teams with
multiple applications

# Our First Pipeline: Getting Started



# Pipeline Orchestration Using Git Flow

**Main Branch**
Known stable, fully integrated product available for external consumption including quality control reviews




Code Review led by Team Lead before merging into Development branch

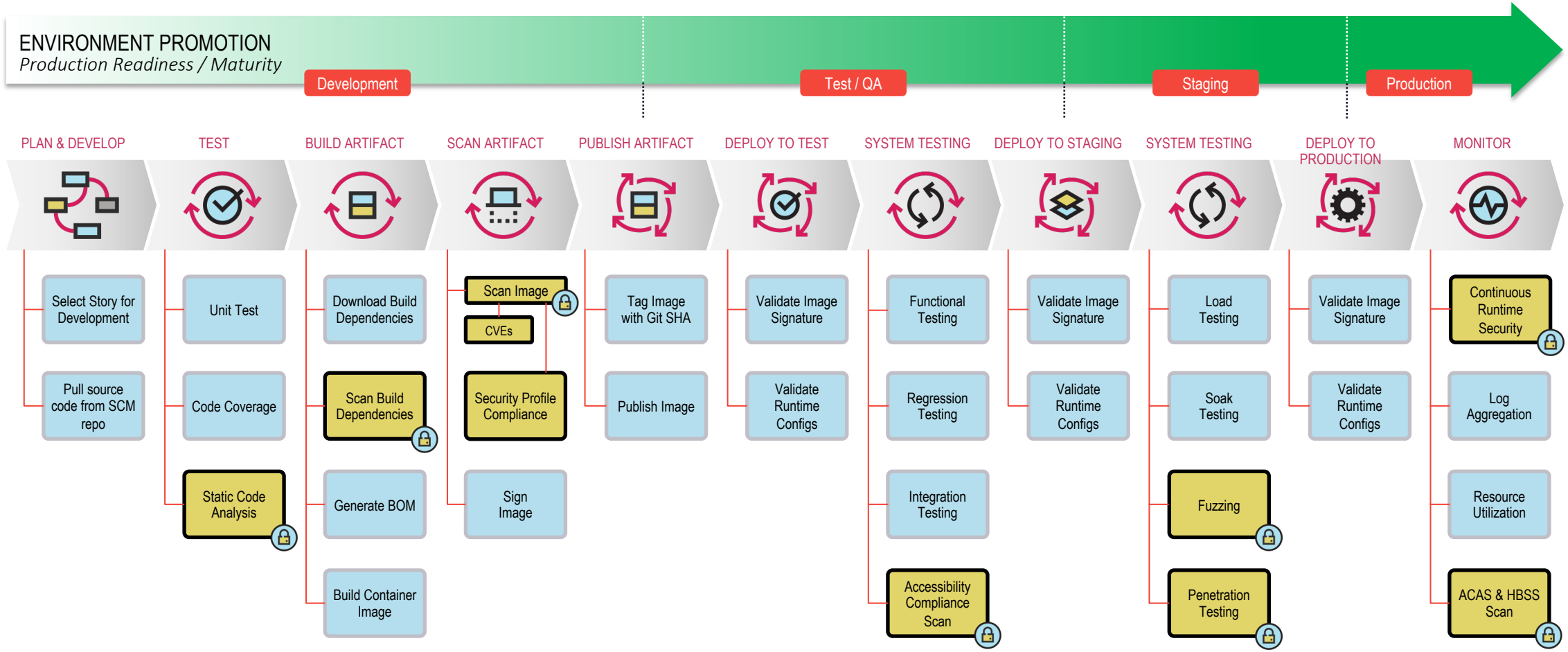
**Development Branch**
- 508 Compliance Scan
- Integration Tests

**Feature Branches**
- Definition of Done
- Unit Tests and Code Coverage
- Static Code Analysis
- OWASP Dependency Check

Each feature is developed as its own feature branch and then integrated into the Development branch when it meets the Definition of Done

Changes to these branches trigger deployments to their respective application environments

# Test All The Things: Integrating .* Testing



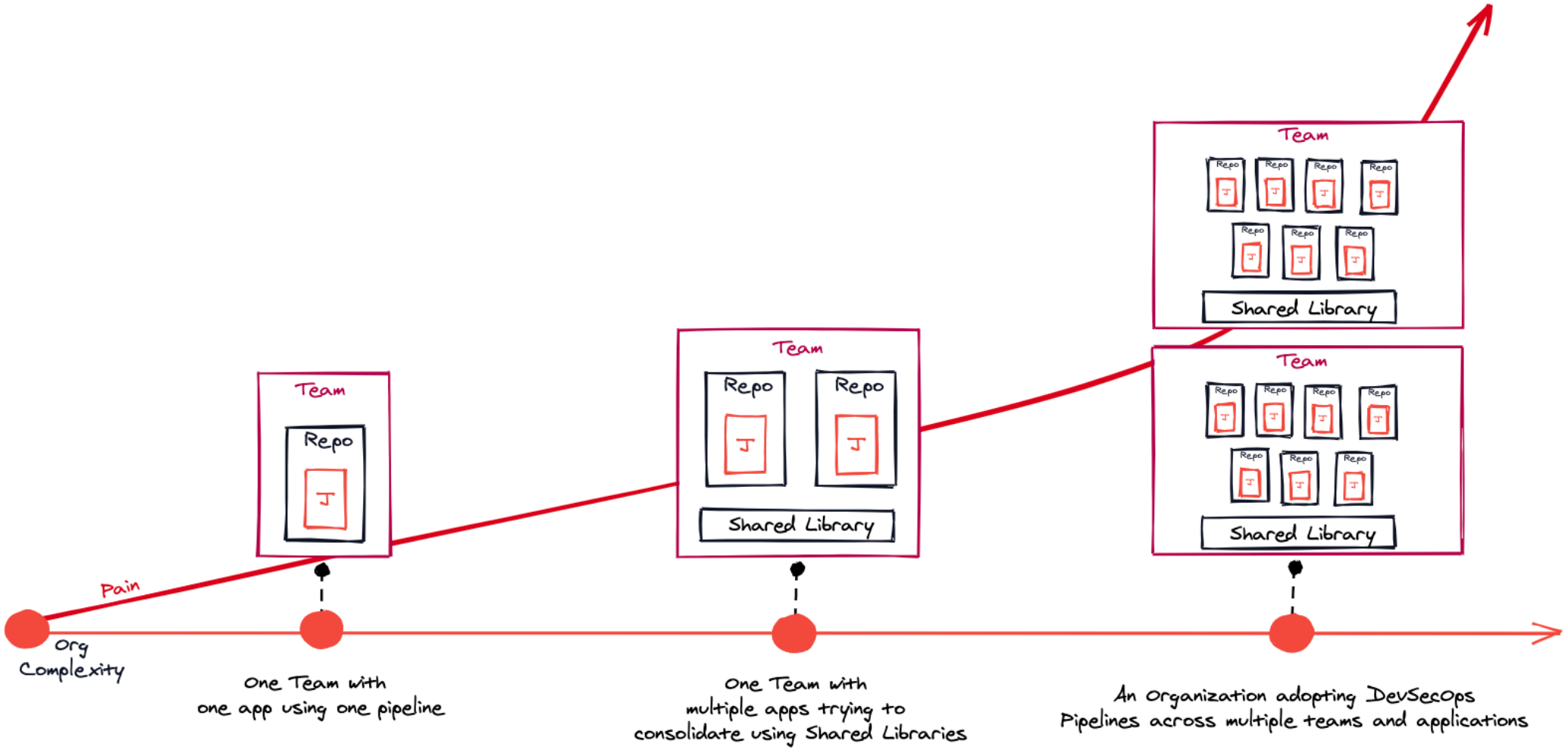

DEVOPSWORLD by CloudBees

# Challenges at Scale

## Spoiler: Copying and Pasting Is Usually Bad

DEVOPS WORLD
by CloudBees

# Linear Scale, Exponential Pain



# What Causes This Pain?

When writing pipelines, we often fail to separate the ***business logic*** from the ***technical implementation***

We have to ***duplicate*** our pipeline definitions on a per-application basis

**Time**

*Creating a mature DevSecOps pipeline for an application can take months. Onboarding new applications requires manual intervention.*

**Complexity**

*Different types of applications will utilize different tools and different teams may leverage different testing frameworks.*

**Standardization**

*Each application's source code repository requires a Jenkinsfile, making it difficult to ensure common processes are adhered to.*

**Continuous Improvement**

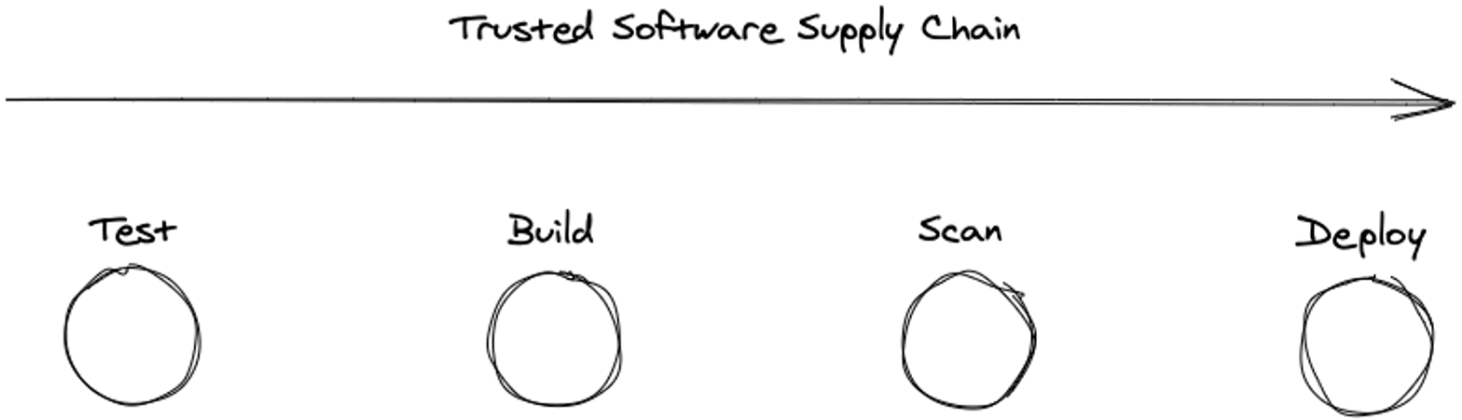
*Making a change to the pipeline requires changing Jenkinsfiles distributed across every branch in every source code repository.*

# The Jenkins Templating Engine

## Tool-Agnostic, Reusable Workflows. Modularized Libraries.

DEVOPS WORLD
by CloudBees

# Define Tool-Agnostic, Templated Workflows



# Define Tool-Agnostic, Templated Workflows

*Regardless of what tools are being used, the flow remains the same.*

*Example Jenkinsfile for an application using Maven*

```
stage("Maven: Build"){
 docker.image("maven").inside{
  sh "mvn clean package"
 }
}

stage('SonarQube: Static Code Analysis') {
 node {
  def scannerHome = tool 'SonarScanner 4.0';
  withSonarQubeEnv('My SonarQube Server') {
   sh "${scannerHome}/bin/sonar-scanner"
  }
 }
}
```

*Pipeline Template*

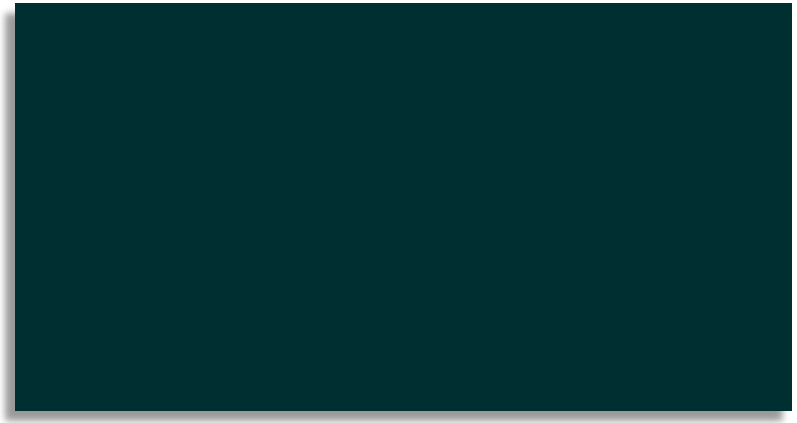
```
k

s
```

*Example Jenkinsfile for an application using Gradle*

```
stage("Gradle: Build"){
 docker.image("gradle").inside{
  sh "gradle clean build"
 }
}

stage('SonarQube: Static Code Analysis') {
 node {
  def scannerHome = tool 'SonarScanner 4.0';
  withSonarQubeEnv('My SonarQube Server') {
   sh "${scannerHome}/bin/sonar-scanner"
  }
 }
}
```

# Reorganize

Pipeline Configuration Repository



*Pipeline Template*

```
build()
static_code_analysis()
```

*Jenkinsfile*

*maven application*

```
libraries{
  maven
  sonarqube
}
```

*pipeline_config.groovy*

*gradle application*

```
libraries{
  gradle
  sonarqube
}
```

*pipeline_config.groovy*

*Library Steps*

```
void call(){
  stage("Maven: Build"){
    docker.image("maven").inside{
      sh "mvn clean package"
    }
  }
}
```

*maven/build.groovy*

```
void call(){
  stage("Gradle: Build"){
    docker.image("gradle").inside{
      sh "gradle clean build"
    }
  }
}
```

*gradle/build.groovy*

```
void call(){
  stage('SonarQube: Static Code Analysis') {
    node {
      def scannerHome = tool 'SonarScanner 4.0';
      withSonarQubeEnv('My SonarQube Server') {
        sh "${scannerHome}/bin/sonar-scanner"
      }
    }
  }
}
```

*sonarqube/static_code_analysis.groovy*

# Library Parameterization

Libraries become reusable building blocks used configure pipelines.

Libraries can *parameterized* to optimize reusability.

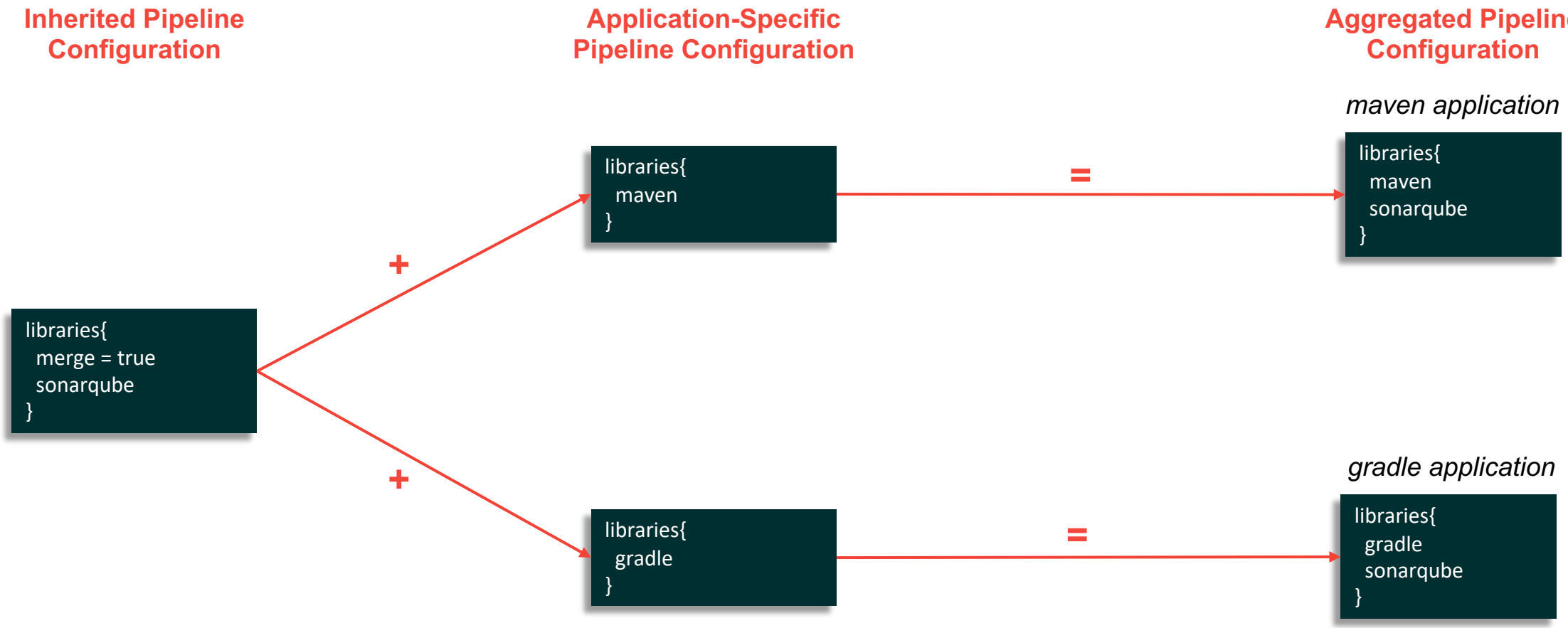
```
libraries{
  maven
  sonarqube{
    scanner_version = “SonarScanner 3.0”
    enforce_quality_gate = false
  }
}
```

```
void call(){
  stage('SonarQube: Static Code Analysis') {

    // parse configuration
    String scannerVersion      = config.scanner_version  ?: "SonarScanner 4.0"
    String serverName          = config.server_name      ?: "My SonarQube Server"
    Boolean enforceQualityGate = config.containsKey("enforce_quality_gate") ?
                                 config.enforce_quality_gate : true

    node {
      def scannerHome = tool(scannerVersion)
      withSonarQubeEnv(serverName) {
        sh "${scannerHome}/bin/sonar-scanner"
      }
    }

    timeout(time: 1, unit: 'HOURS') {
      def qg = waitForQualityGate()
      if (qg.status != 'OK') {
        if(enforceQualityGate){
          error "Pipeline aborted due to quality gate failure: ${qg.status}"
        } else {
          warning "Quality gate failure: ${qg.status}"
        }
      }
    }
  }
}
```

Library steps autowired with a config variable populated with values from the pipeline configuration.

DEVOPSWORLD by CloudBees

# Hierarchical Pipeline Configurations



# Real-World Example

## Booz Allen’s Solutions Delivery Platform

DEVOPS WORLD
by CloudBees

# Advanced Template Walkthrough

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The **github** library provides functionality to map branching strategies to different pipeline activities

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The functions take regular expressions as input variables that we can abstract using JTE's **keywords** functionality

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The docker library has a *build.groovy* **step.**

There could also be an npm, maven, gradle, etc library that implements a build step for interchangeability

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The OWASP Dependency Checker library implements a step for application dependency scanning

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The SonarQube library implements a step for static code analysis

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The Helm library would take configurations for the location of the target kubernetes cluster to perform deployments

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The deploy_to step from the Helm library takes an **application environment** from JTE as an input parameter

pipeline configuration

```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Advanced Template Walkthrough

The OWASP ZAP library performs penetration testing

pipeline configuration

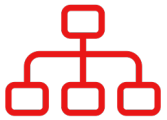
```
libraries{
  github
  docker
  owasp_dependency_check
  sonarqube
  helm
  owasp_zap
}
application_environments{
  dev
  prod
}
keywords{
  main = ~/^[Mm]ain(line|)$/
  develop = ~/^[Dd]evelop(ment|)$/
}
```

common pipeline template

```
on_pull_request to: develop, {
  build()
  application_dependency_scan()
  static_code_analysis()
}

on_merge to: develop, {
  deploy_to dev
  penetration_testing()
}

on_merge to: main, {
  deploy_to prod
}
```

# Key Takeaways

- The Jenkins Templating Engine is a ***framework*** for developing tool-agnostic, templated workflows that can be reused by multiple teams simultaneously – regardless of the tools they are using.
- This approach separates the business logic (*pipeline template*) from the technical implementation (*pipeline libraries*) allowing teams to **configure** their pipelines instead of build them from scratch



**Apply Organizational Governance**

*By centralizing your pipeline definition to a common place you can standardize your software delivery processes across teams*

**Optimize Pipeline Code Reuse**

*Create modularized tool-integrations called pipeline libraries that can be reused and collectively maintained*

**Simplify Pipeline Maintainability**

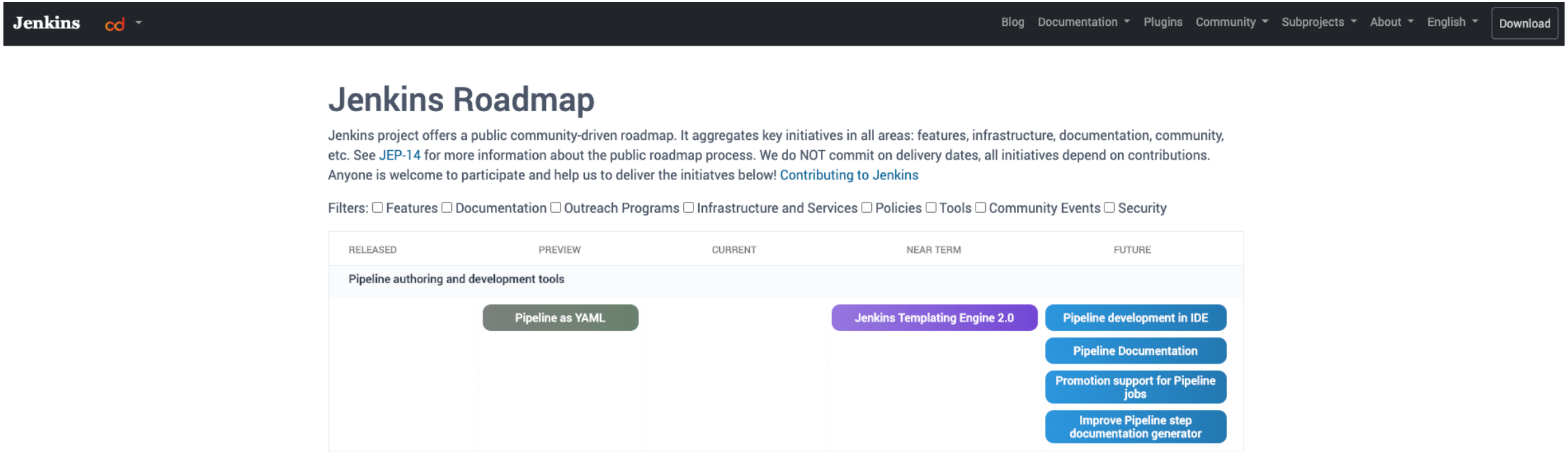
*Each application's source code repository requires a Jenkinsfile, making it difficult to ensure common processes are adhered to.*

# What’s Next For JTE?

## Great Things in the Pipeline ☺

DEVOPS WORLD
by CloudBees

# JTE v2.0



# Get Involved!

Learn More and Get Involved!

DEVOPS WORLD
by CloudBees

# Get Started, Get Involved, Stay in Touch

@steven_terrana

steven-terrana

**Join the Gitter Channel**

**Read the Docs!**

**Get Started With Learning Labs!**

# Questions?