

Managed CI/CD Pipelines

**Minimizing Developer Effort and
Maximizing Enterprise Compliance**

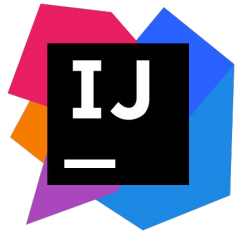
Tahmina Khan
Linda Oyolu

**DEVOPS
WORLD**
by CloudBees

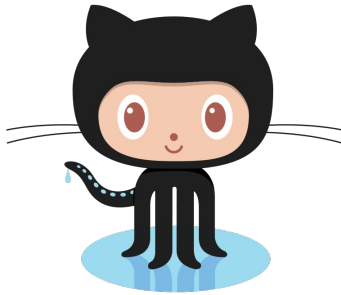
Capital One is a cloud first company



The (Ideal) Developer Experience



Write awesome
code!



Push my code to
GitHub



Automatically
build and deploy
my code



Use the cloud to
host my
application

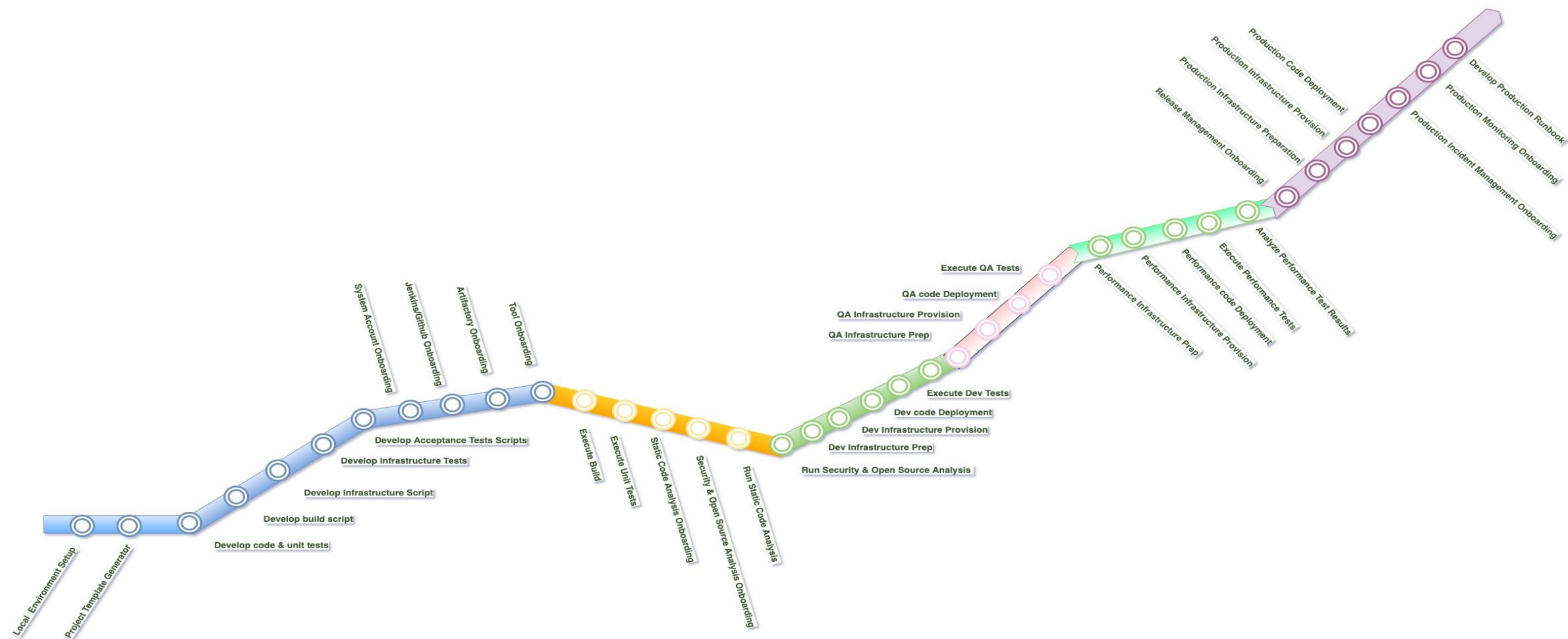


A happy
developer

Quality Gates

- Source code version control
- Optimum branching strategy
- Static analysis
- >80% code coverage
- Vulnerability scan
- Open source scan
- Artifact version control
- Auto provisioning
- Immutable servers
- Integration testing
- Performance testing
- Build deploy testing automated for every commit
- Automated rollback
- Automated change order
- Zero downtime release
- Feature toggle

Subway Map of Processes



As a developer, deploying an application should be as simple as...

Merge my new PR

Build, test, and deploy to cloud

Less work is better

**But make sure it complies with Capital One's standards
and best practices**

Homegrown CI/CD Framework

A software delivery framework that focuses on making it easy for engineers to rapidly build and release quality software into production



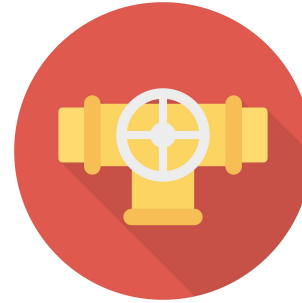
CI/CD Framework Components



A delightful CLI for creating a CI/CD pipeline and deploying apps. Template generator to get developers productive faster.



Graphical User Interface enables developers to generate a template to create their pipeline like the CLI, but from the comfort of their browser.



Collection of reusable pipeline tasks for orchestration and automation of the end to end SDLC.



Infrastructure deployment feature allows developers to provision their own custom set of resources, or deploy their application in a customized manner, based on organized units.

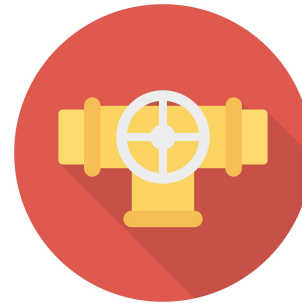
CI/CD Framework Components



A delightful CLI for creating a CI/CD pipeline and deploying apps. Template generator to get developers productive faster.



Graphical User Interface enables developers to generate a template to create their pipeline like the CLI, but from the comfort of their browser.



Collection of reusable pipeline tasks for orchestration and automation of the end to end SDLC.



Infrastructure deployment feature allows developers to provision their own custom set of resources, or deploy their application in a customized manner, based on organized units.

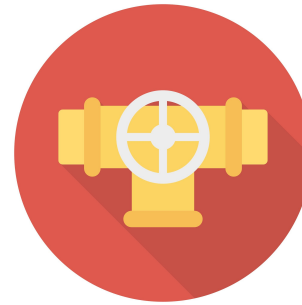
CI/CD Framework Components



A delightful CLI for creating a CI/CD pipeline and deploying apps. Template generator to get developers productive faster.



Graphical User Interface enables developers to generate a template to create their pipeline like the CLI, but from the comfort of their browser.



Collection of reusable pipeline tasks for orchestration and automation of the end to end SDLC.



Infrastructure deployment feature allows developers to provision their own custom set of resources, or deploy their application in a customized manner, based on organized units.

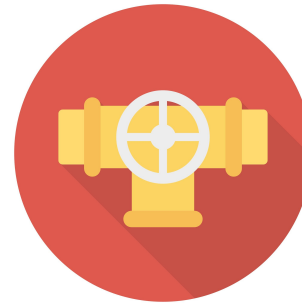
CI/CD Framework Components



A delightful CLI for creating a CI/CD pipeline and deploying apps. Template generator to get developers productive faster.



Graphical User Interface enables developers to generate a template to create their pipeline like the CLI, but from the comfort of their browser.

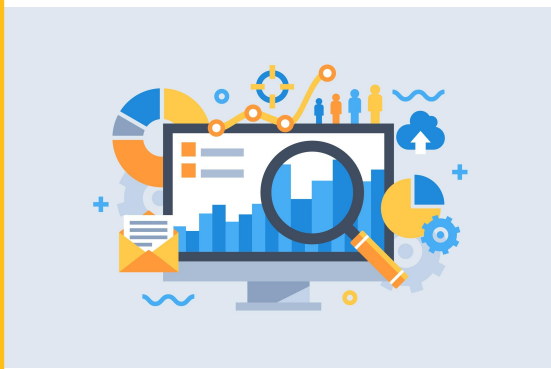


Collection of reusable pipeline tasks for orchestration and automation of the end to end SDLC.



Infrastructure deployment feature allows developers to provision their own custom set of resources, or deploy their application in a customized manner, based on organized units.

CI/CD Framework Components



Metrics collection capabilities to track how users interact with the platform and track customer pipeline health and issues.



Monitoring enables customers to enable basic practical observability, implement strategic meaningful monitors/alerting, and observe what is truly important to ensure that the reliability of their systems remains on par with customer/consumer expectations.



Auto rehydration is an orchestration platform that provides customizable scheduling from pre-prod to prod for rehydrating ec2s to the latest AMI and base OS patches.

CI/CD Framework Components



Metrics collection capabilities to track how users interact with the platform and track customer pipeline health and issues.

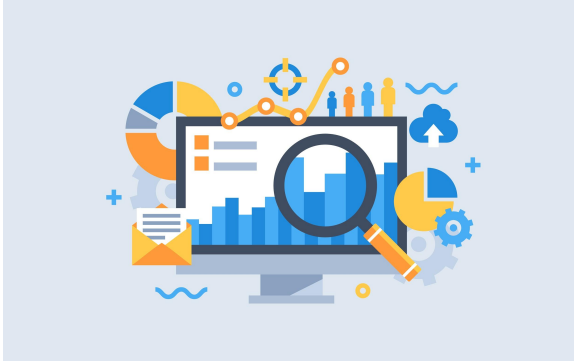


Monitoring enables customers to enable basic practical observability, implement strategic meaningful monitors/alerting, and observe what is truly important to ensure that the reliability of their systems remains on par with customer/consumer expectations.



Auto rehydration is an orchestration platform that provides customizable scheduling from pre-prod to prod for rehydrating ec2s to the latest AMI and base OS patches.

CI/CD Framework Components



Metrics collection capabilities to track how users interact with the platform and track customer pipeline health and issues.



Monitoring enables customers to enable basic practical observability, implement strategic meaningful monitors/alerting, and observe what is truly important to ensure that the reliability of their systems remains on par with customer/consumer expectations.



Auto rehydration is an orchestration platform that provides customizable scheduling from pre-prod to prod for rehydrating ec2s to the latest AMI and base OS patches.

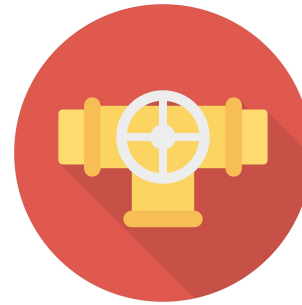
CI/CD Framework Components



A delightful CLI for creating a CI/CD pipeline and deploying apps. Template generator to get developers productive faster.



Graphical User Interface enables developers to generate a template to create their pipeline like the CLI, but from the comfort of their browser.



Collection of reusable pipeline tasks for orchestration and automation of the end to end SDLC.



Infrastructure deployment feature allows developers to provision their own custom set of resources, or deploy their application in a customized manner, based on organized units.

Shared Jenkins Pipeline Library

> User Documentation Home

← Using Docker with Pipeline

↑ Pipeline
Index

Pipeline Development Tools ⇒

User Handbook

- User Handbook overview
- Installing Jenkins
- Using Jenkins
- **Pipeline**
 - Getting started with Pipeline
 - Using a Jenkinsfile
 - Running Pipelines
 - Branches and Pull Requests
 - Using Docker with Pipeline
 - **Extending with Shared Libraries**
 - Pipeline Development Tools
 - Pipeline Syntax
 - Pipeline Best Practices
 - Scaling Pipelines

Extending with Shared Libraries



As Pipeline is adopted for more and more projects in an organization, common patterns are likely to emerge. Oftentimes it is useful to share parts of Pipelines between various projects to reduce redundancies and keep code "DRY" [1].

Pipeline has support for creating "Shared Libraries" which can be defined in external source control repositories and loaded into existing Pipelines.

Table of Contents
Defining Shared Libraries
Directory structure
Global Shared Libraries
Folder-level Shared Libraries
Automatic Shared Libraries
Using libraries
Loading libraries dynamically
Library versions

<https://www.jenkins.io/doc/book/pipeline/shared-libraries/>

Shared Jenkins Pipeline Library

  / jenkins-pipeline-library

Unwatch 44

Unstar 197

Fork 217

Code

Issues 125

Pull requests 45


Projects 0

Wiki

Insights


Settings

More

 Collection of reusable Jenkins CI/CD pipeline tasks - [Innersourced Project] <https://jenkins.innersourced-project.com/>


Edit


jenkins





innersourced-project

Manage topics

 3,861 commits

 176 branches

 774 releases

 221 contributors

**Configuration
File**

Jenkinsfile

Dockerfile

Shared Jenkins Pipeline Library

```
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

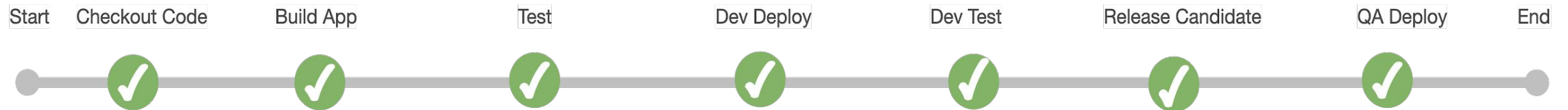
stage('Analyze Code
(Static)') {
    ...
}

stage('Analyze Code (Security
& Open Source)') {
    ...
}

stage('Deploy App') {
    ...
}

...
```

*Source code shown is simulated for demo purpose only



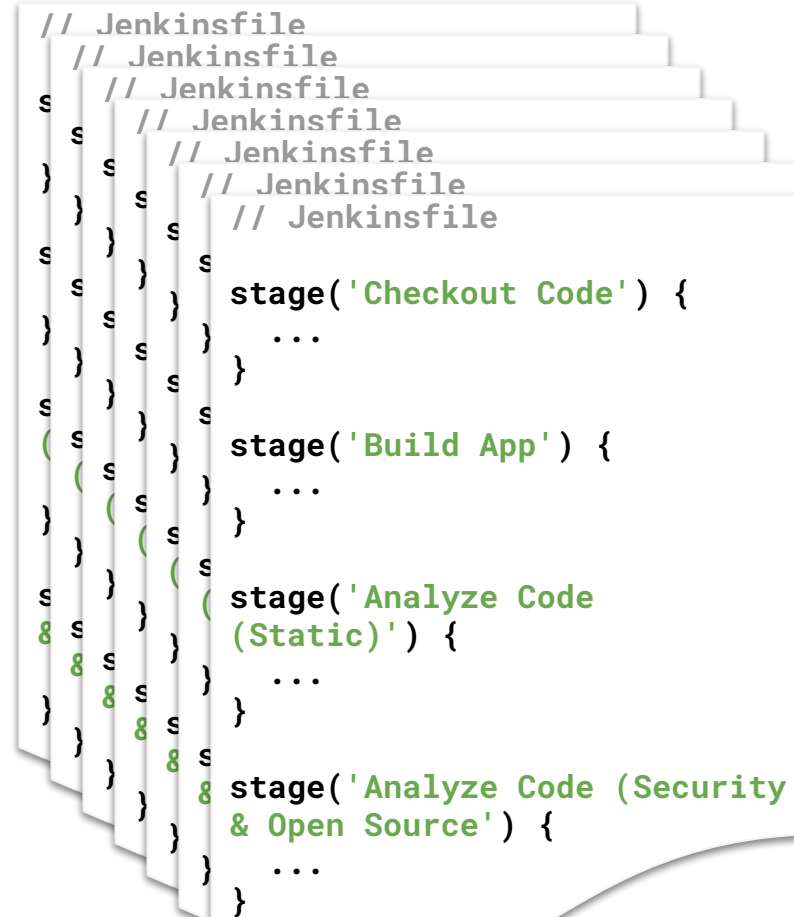


Why Managed Pipelines?



**DEVOPS
WORLD**
by CloudBees

Why Managed Pipelines?



```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code
(Static)') {
    ...
}

stage('Analyze Code (Security
& Open Source') {
    ...
}
```

Why Managed Pipelines?

```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code
(Static)') {
    ...
}

stage('Analyze Code (Security
& Open Source)') {
    ...
}
```

```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code
(Static)') {
    ...
}

stage('Analyze Code (Security
& Open Source)') {
    ...
}
```

```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code
(Static)') {
    ...
}

stage('Analyze Code (Security
& Open Source)') {
    ...
}
```

*Source code shown is simulated for demo purpose only

Why Managed Pipelines?

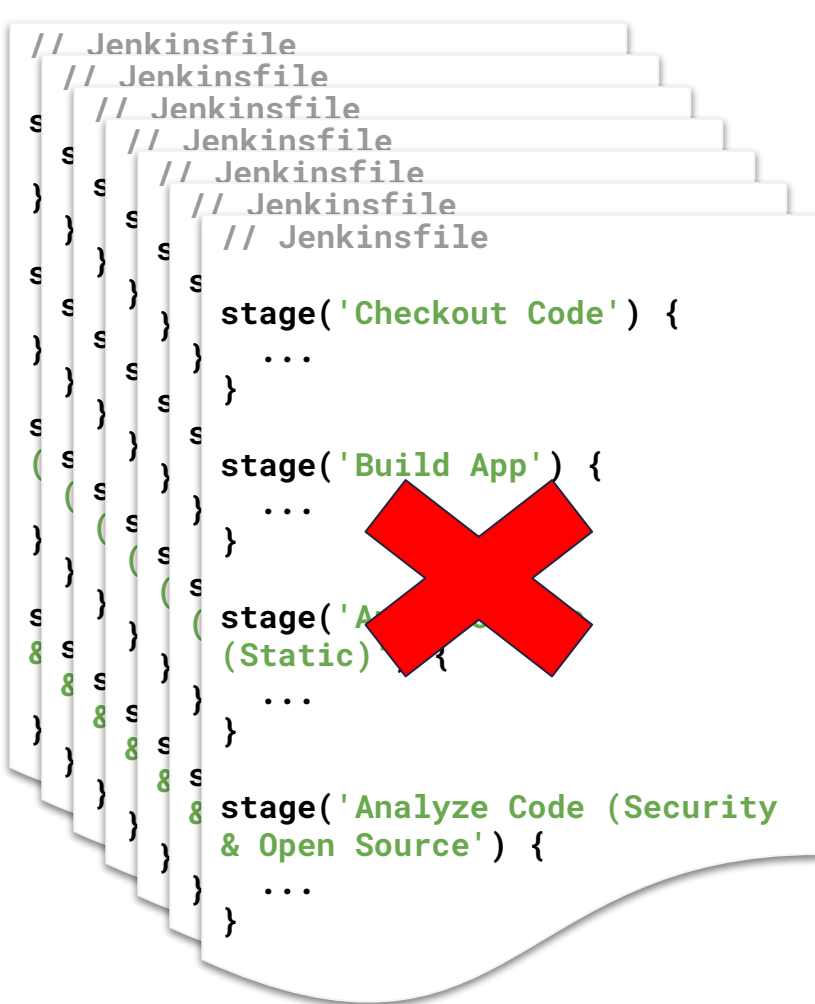
```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}
```

A stack of five Jenkinsfiles is shown, with the top one displaying a Jenkinsfile script. A large red 'X' is superimposed over the 'Build App' stage, indicating a problem or error with this stage in the pipeline.

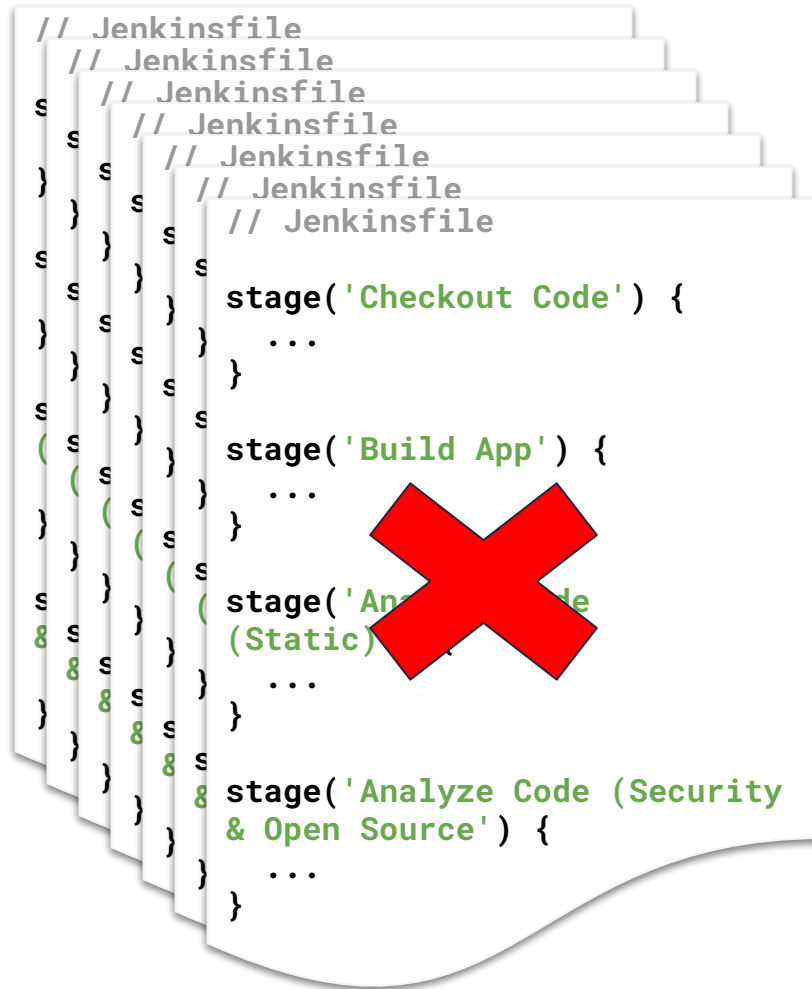
```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}
```

A stack of five Jenkinsfiles is shown, with the top one displaying a Jenkinsfile script. A large red 'X' is superimposed over the 'Build App' stage, indicating a problem or error with this stage in the pipeline.

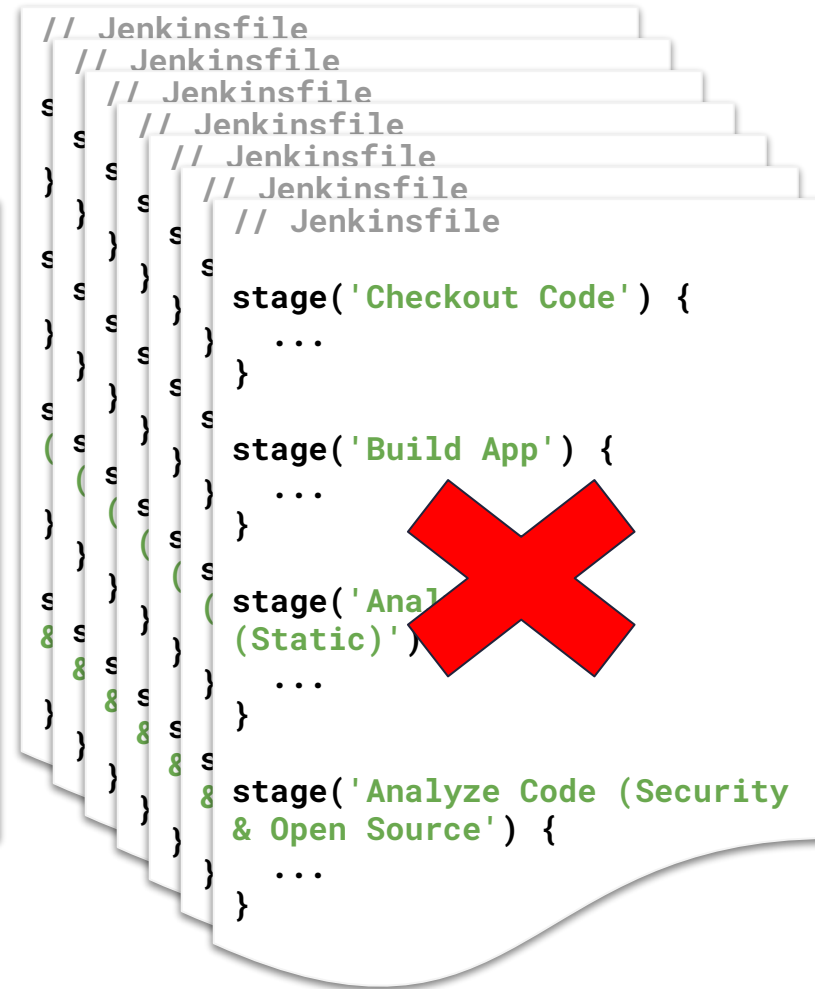
```
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile
// Jenkinsfile

stage('Checkout Code') {
    ...
}

stage('Build App') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}

stage('Analyze Code (Security & Open Source)') {
    ...
}
```

A stack of five Jenkinsfiles is shown, with the top one displaying a Jenkinsfile script. A large red 'X' is superimposed over the 'Build App' stage, indicating a problem or error with this stage in the pipeline.

*Source code shown is simulated for demo purpose only

Why Managed Pipelines?



Why Managed Pipelines?



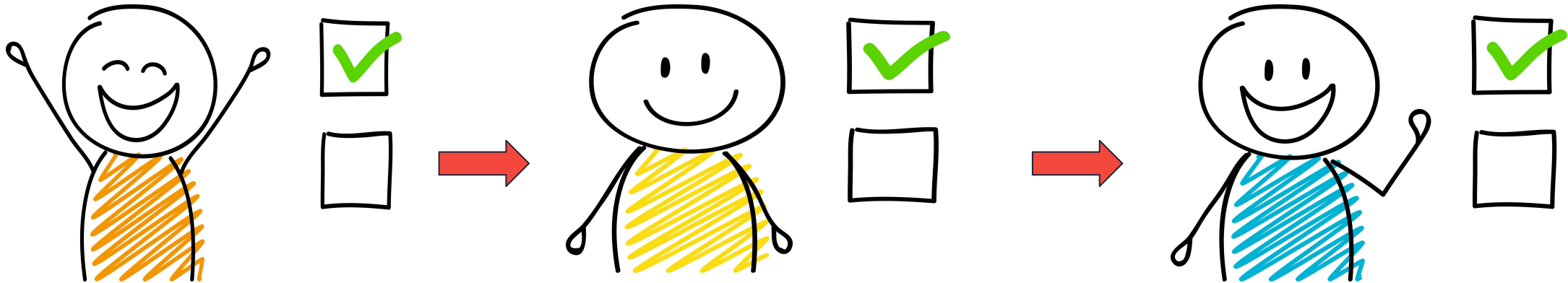
Why Managed Pipelines?



Why Managed Pipelines?



Why Managed Pipelines?



Software Engineers

Business Partners

Capital One Customers

How do Managed Pipelines work?



Centralized Jenkinsfile



Centralized Jenkinsfile

The screenshot shows the GitHub interface for the `jenkins-pipeline-library` repository. A red arrow points to the repository name. Another red arrow points to the `Jenkinsfile` file in the path `resources / jenkinsfiles / managed / docker / Jenkinsfile`. The file details show it was committed by `d532ebc` on Jun 30. The file content is as follows:

```
71 lines (56 sloc) 1.83 KB
1  #!groovy
2
3  jobProperties()
4
5  stage(name: 'Checkout Code') {
6    projectProperties()
7  }
```

Shared Jenkins Pipeline Library

Project Recognizers

Custom script

Marker file

Pipeline

Definition

SCM

Repositories

Repository URL

Credentials

Branches to build

Repository browser

Additional Behaviours

Script Path

Lightweight checkout ☒

[Pipeline Syntax](#)

<https://www.cloudbees.com/blog/ensuring-corporate-standards-pipelines-custom-marker-files>

Shared Jenkins Pipeline Library

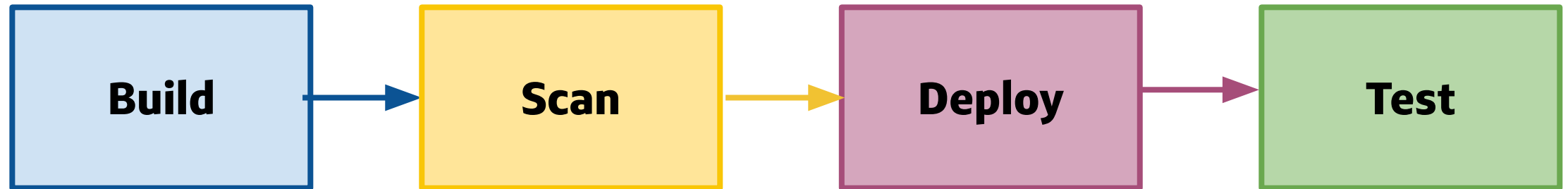
```
// vars/managedPipeline.groovy

def flavor      = getFlavor()
def jenkinsFile =
    "jenkinsfiles/managed/${flavor}/Jenkinsfile"

if (flavor) {
    try {
        pipeline = libraryResource(jenkinsFile)
    } catch (err) {
        echo "Managed pipeline flavor '${flavor}' defined in
your configuration file does not exist..."
        throw err
    }
} else {
    pipeline = readTrusted 'Jenkinsfile'
}
```

*Source code shown is simulated for demo purpose only

The Anatomy of a CI/CD Pipeline



Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}

stage(name: 'Analyze Code (Static)') {
    codeAnalysis(tool: 'sonar')
}

stage(name: 'Analyze Code (Security & Open Source)') {
    codeAnalysis(tool: 'open-source-analysis')
}

withEnvironments(group: 'dev', task: 'deploy') { e ->
    stage(name: "Deploy App (${e.appEnv})") {
        deploy(appEnv: e.appEnv)
    }
}

withEnvironments(group: 'dev', task: 'acceptance_tests') { e ->
    stage(name: "Test App (${e.appEnv})") {
        acceptanceTests(appEnv: e.appEnv)
    }
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}

stage(name: 'Analyze Code (Static)') {
    codeAnalysis(tool: 'sonar')
}

stage(name: 'Analyze Code (Security & Open Source)') {
    codeAnalysis(tool: 'open-source-analysis')
}

withEnvironments(group: 'dev', task: 'deploy') { e ->
    stage(name: "Deploy App (${e.appEnv})") {
        deploy(appEnv: e.appEnv)
    }
}

withEnvironments(group: 'dev', task: 'acceptance_tests') { e ->
    stage(name: "Test App (${e.appEnv})") {
        acceptanceTests(appEnv: e.appEnv)
    }
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}

stage(name: 'Analyze Code (Static)') {
    codeAnalysis(tool: 'sonar')
}

stage(name: 'Analyze Code (Security & Open Source)') {
    codeAnalysis(tool: 'open-source-analysis')
}

withEnvironments(group: 'dev', task: 'deploy') { e ->
    stage(name: "Deploy App (${e.appEnv})") {
        deploy(appEnv: e.appEnv)
    }
}

withEnvironments(group: 'dev', task: 'acceptance_tests') { e ->
    stage(name: "Test App (${e.appEnv})") {
        acceptanceTests(appEnv: e.appEnv)
    }
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}

stage(name: 'Analyze Code (Static)') {
    codeAnalysis(tool: 'sonar')
}

stage(name: 'Analyze Code (Security & Open Source)') {
    codeAnalysis(tool: 'open-source-analysis')
}

withEnvironments(group: 'dev', task: 'deploy') { e ->
    stage(name: "Deploy App (${e.appEnv})") {
        deploy(appEnv: e.appEnv)
    }
}

withEnvironments(group: 'dev', task: 'acceptance_tests') { e ->
    stage(name: "Test App (${e.appEnv})") {
        acceptanceTests(appEnv: e.appEnv)
    }
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}

stage(name: 'Analyze Code (Static)') {
    codeAnalysis(tool: 'sonar')
}

stage(name: 'Analyze Code (Security & Open Source)') {
    codeAnalysis(tool: 'open-source-analysis')
}

withEnvironments(group: 'dev', task: 'deploy') { e ->
    stage(name: "Deploy App (${e.appEnv})") {
        deploy(appEnv: e.appEnv)
    }
}

withEnvironments(group: 'dev', task: 'acceptance_tests') { e ->
    stage(name: "Test App (${e.appEnv})") {
        acceptanceTests(appEnv: e.appEnv)
    }
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile continues..  
  
releaseCheckpoint()  
  
stage(name: 'Create Release Candidate', withHook:  
    'release_candidate') {  
    releaseCandidate()  
}  
  
withEnvironments(group: 'qa', task: 'deploy') { e ->  
    stage(name: "Deploy App (${e.appEnv})") {  
        deploy(appEnv: e.appEnv)  
    }  
}  
  
withEnvironments(group: 'qa', task: 'acceptance_tests') { e ->  
    stage(name: "Test App (${e.appEnv})") {  
        acceptanceTests(appEnv: e.appEnv)  
    }  
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile continues..  
  
releaseCheckpoint()  
  
stage(name: 'Create Release Candidate', withHook:  
    'release_candidate') {  
    releaseCandidate()  
}  
  
withEnvironments(group: 'qa', task: 'deploy') { e ->  
    stage(name: "Deploy App (${e.appEnv})") {  
        deploy(appEnv: e.appEnv)  
    }  
}  
  
withEnvironments(group: 'qa', task: 'acceptance_tests') { e ->  
    stage(name: "Test App (${e.appEnv})") {  
        acceptanceTests(appEnv: e.appEnv)  
    }  
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile continues..  
  
performanceTests()  
  
withApproval {  
  withEnvironments(group: 'prod', task: 'deploy') { e ->  
    stage(name: "Deploy App (${e.appEnv})") {  
      deploy(appEnv: e.appEnv)  
    }  
  }  
  
  withEnvironments(group: 'prod', task: 'acceptance_tests') { e ->  
    stage(name: "Test App (${e.appEnv})") {  
      acceptanceTests(appEnv: e.appEnv)  
    }  
  }  
  
  stage(name: 'Complete Release', withHook: 'release') {  
    githubRelease()  
  }  
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile continues..  
  
performanceTests()  
  
withApproval {  
  withEnvironments(group: 'prod', task: 'deploy') { e ->  
    stage(name: "Deploy App (${e.appEnv})") {  
      deploy(appEnv: e.appEnv)  
    }  
  }  
  
  withEnvironments(group: 'prod', task: 'acceptance_tests') { e ->  
    stage(name: "Test App (${e.appEnv})") {  
      acceptanceTests(appEnv: e.appEnv)  
    }  
  }  
  
  stage(name: 'Complete Release', withHook: 'release') {  
    githubRelease()  
  }  
}
```

*Source code shown is simulated for demo purpose only

Pre - Defined Pipeline Stages

```
// Jenkinsfile
#!/groovy

jobProperties()

stage(name: 'Checkout Code') {
    projectProperties()
}

stage(name: 'Build App') {
    applicationBuild()
}
```



```
// vars/applicationBuild.groovy
#!/groovy

node {
    safeUnstash 'sources'
    switch (tool) {
        case 'gradle':
            gradleBuild(config)
            break
        case 'golang':
            goBuild(config)
            break
        case 'maven':
            mavenBuild(config + ['withJUnitReport': true])
            break
        case 'npm':
            npmTest(config)
            break
        case 'python':
            pipenvTest(config)
            break
        default:
            def errMsg = "Build tool \"${tool}\" not supported!"
            error errMsg
            break
    }
}
```

Configurable Stage via the Yaml Configuration File

```
20
21 pipeline:
22   managed: yes
23   flavor: docker
24   tasks:
25     build:
26       tool: maven
27       args: ['']
28       with_junit_report: true
29     code_analysis:
30       sonar:
31         withMavenSettings: true
32     acceptance_tests:
33       dev:
34         framework: maven-cucumber
35         args:
36           - '-Dcucumber.options="--tags @dev"'
37           - '-DskipITs=false'
38       qa:
39         framework: maven-cucumber
40         args:
41           - '-Dcucumber.options="--tags @qa"'
42           - '-DskipITs=false'
```

How to Build an App With a Managed Pipeline

```
# -*- mode: yaml -*-  
---  
pipeline:  
  tasks:  
    build:  
      tool: npm  
      npm:  
        # All variables  
        # defined here will  
        # be passed to the  
        # npmTest() method  
        node_version: v12.14.1  
        npm_command: install
```



```
// Jenkins Pipeline Library  
  
npmTest(  
  nodeVersion: "v12.14.1"  
  npmCommand: "install"  
)
```

Configurable Stage via the Yaml Configuration File

```
21 pipeline:
22   managed: yes
23   flavor: docker
24   tasks:
25     deploy:
26       framework: framework
27     dev:
28       deploy_type: full
29       interactive_retry: false
30       rollback_approval_timeout: 0
31     qae1:
32       deploy_type: full
33       interactive_retry: false
34       rollback_approval_timeout: 0
35     qaw2:
36       deploy_type: full
37       interactive_retry: false
38       rollback_approval_timeout: 0
39     post_hooks:
40       - task: deploy
41         stage: Route Traffic (QA)
42         appEnv: qa
43         deploy_type: full
44         interactive_retry: false
45         rollback_approval_timeout: 0
46
```

*Configuration shown is simulated for demo purpose only

Managed Pipeline Flavors

Docker



Library



Lambda



Infrastructure



And many more!

Transitioning to Managed Pipeline from User Managed Pipeline



**DEVOPS
WORLD**
by CloudBees

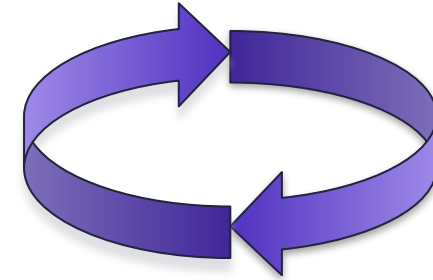
Piloting and Onboarding



Advertised pilot to
user community



Delivered onboarding
workshops with 1-on-1
support



Identify gaps based on
user feedback

In Closing..

- Increased Developer Efficiency
- Faster Speed to Market
- Single Place of Failure
- Out of Box Compliance
- Reduce DRY code
- Streamlined Support Model

In Closing..



Tahmina Khan



Sr. Software Engineer at Capital One
<https://www.linkedin.com/in/khantahmina/>

Linda Oyolu



Software Engineer at Capital One
<https://www.linkedin.com/in/lindaoyolu>



Q & A



**DEVOPS
WORLD**
by CloudBees