



Jenkins
Techlab

AGENDA



10 min.

Intro

Welcome,
purpose of
the lab,
agenda

1 hour

Talk

Introduction to
CI/CD and
Jenkins
Pipelines

~ 3 hours

Labs

Check Lab
Setup,
Basic Labs,
Wrap up

30 min.

Recap

Recap basic labs

3.5 hours

Labs

Advanced Labs,
Q&A
Wrap up

TECHLAB OBJECTIVES



Achieve
theory
base



Learn basic
Jenkins Pipeline
features



Learn
advanced
Jenkins
Pipeline use-
cases

A decorative header at the top of the slide featuring a series of overlapping circles in various shades of blue and teal, creating a cloud-like or bubble-like effect.

CI/CD

Once upon a time...



«Integration Hell» und «Works on My Machine»



1991: Object Oriented Design: With Applications, Grady Booch



1997: Extreme Programming Explained

1



Continuous Integration



code
check in



automated
building
and
testing

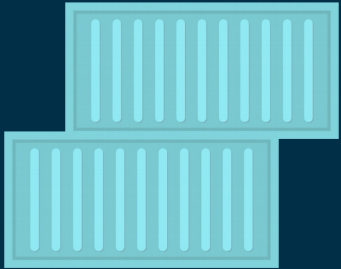


code analyses
/
reporting



deployment

2



Continuous Delivery

«It aims at building, testing, and releasing software faster and more frequently.»

Continuous
Integration



fast & often
released



on
Click



Development process



Small and short development cycles



Agile approach



Fully automated acceptance tests



Very high test coverage



Feature toggles



avoid “Big Scary Merge”

Feature Toggles

...

```
public void sendOrderNotification(Order order) {  
  
    // send default OrderNotification  
    mailService.sendOrderNotificationMessage(order);  
  
    if (featureService.isEnabled(SEND_ORDER_NOTIFICATION_AS_SMS)) {  
        smsService.sendOrderNotificationMessage(order);  
    }  
}  
...
```

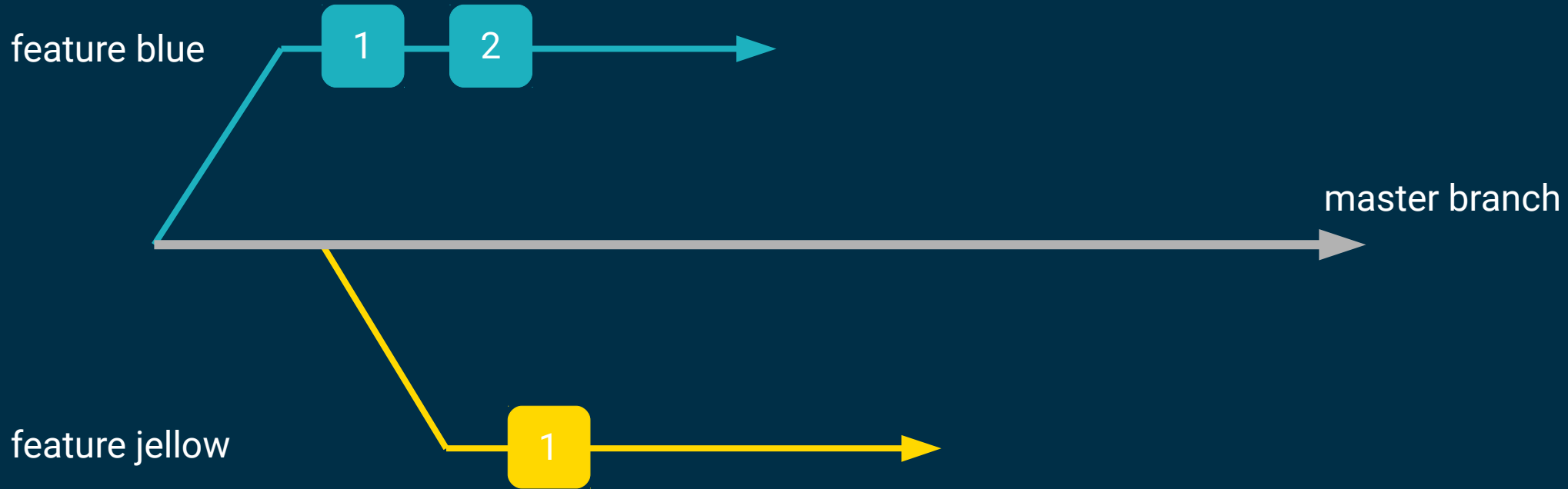
Big Scary Merge of Feature Branches



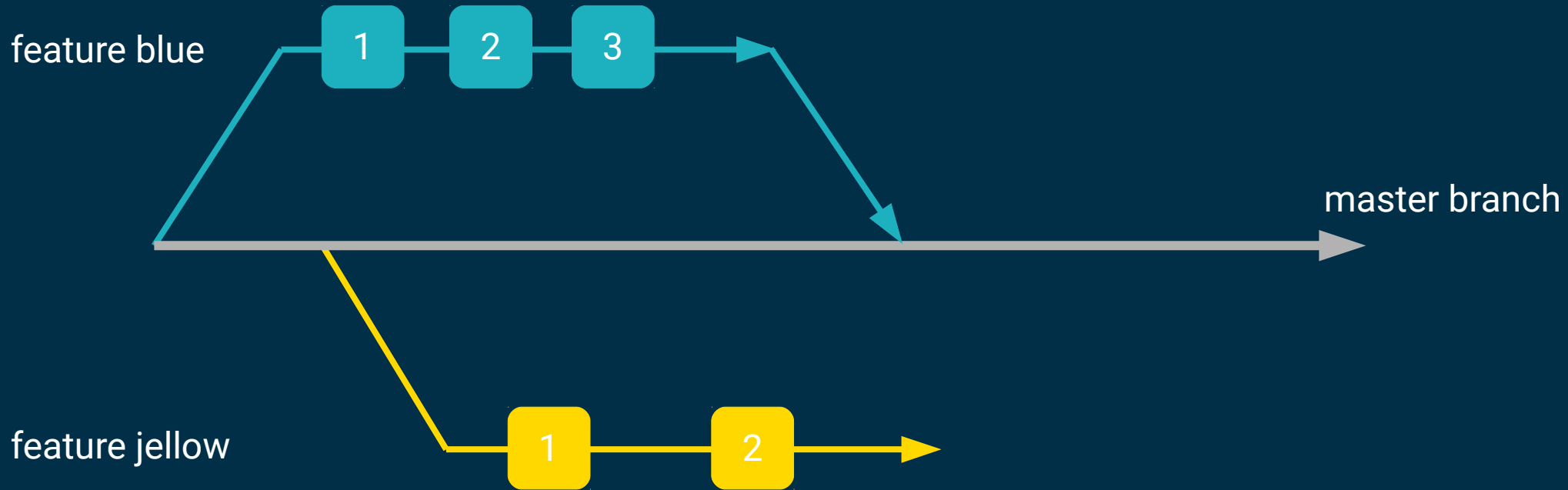
Big Scary Merge of feature branches



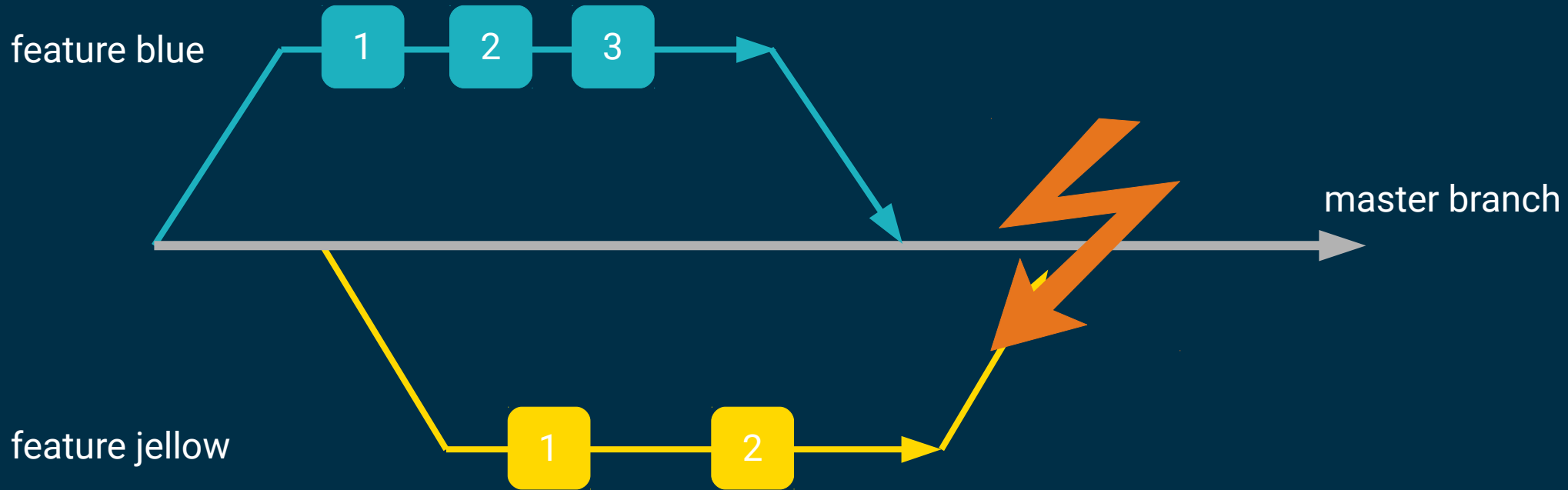
Big Scary Merge of feature branches



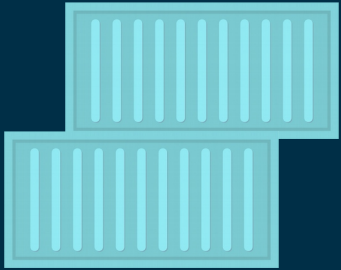
Big Scary Merge of feature branches



Big Scary Merge of feature branches



3



Continuous Deployment

Continuous Deployment

Each release is automatically deployed in production

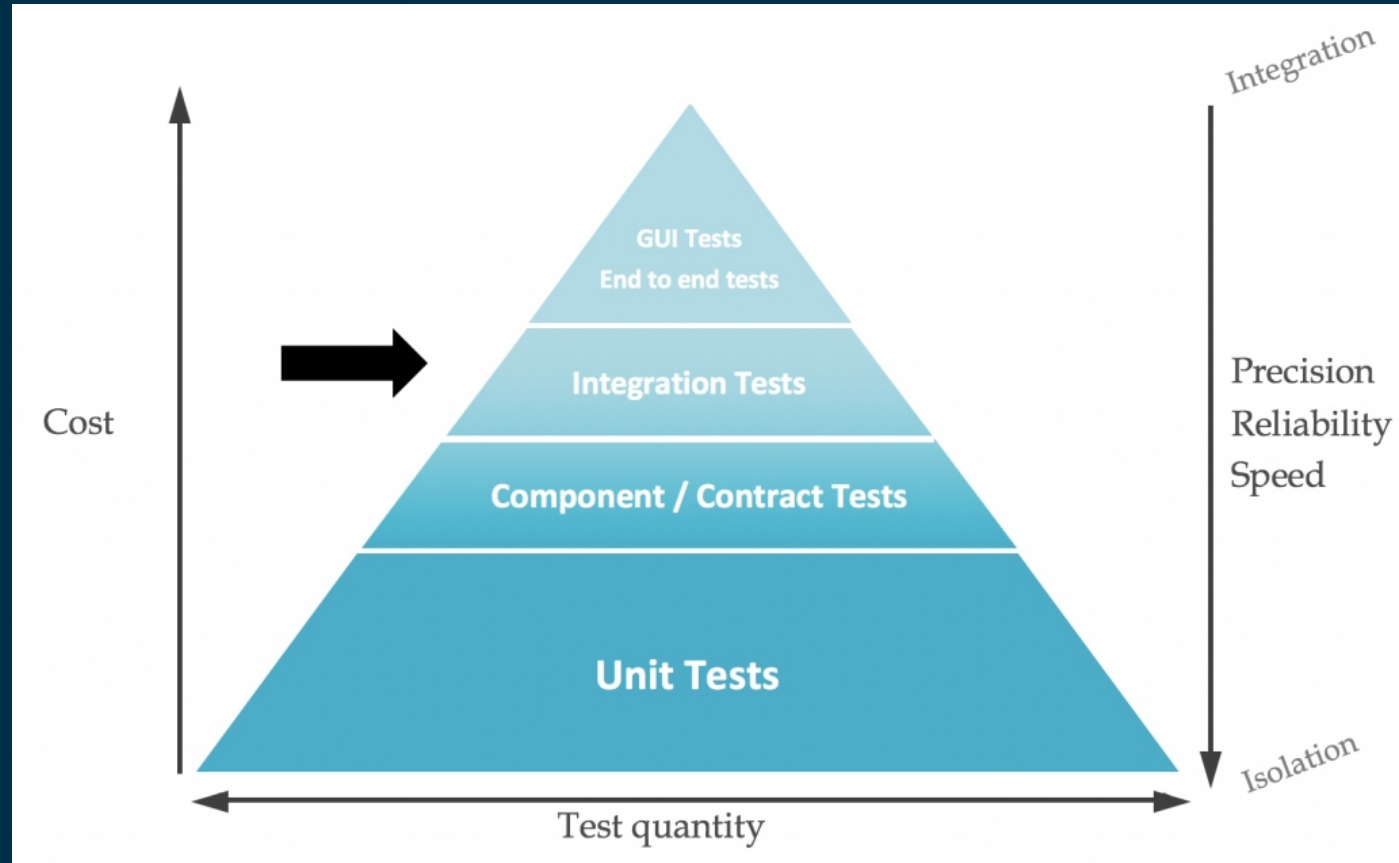
High test coverage needed

Uninterrupted deployments required

Blue – Green (Yellow) deployments

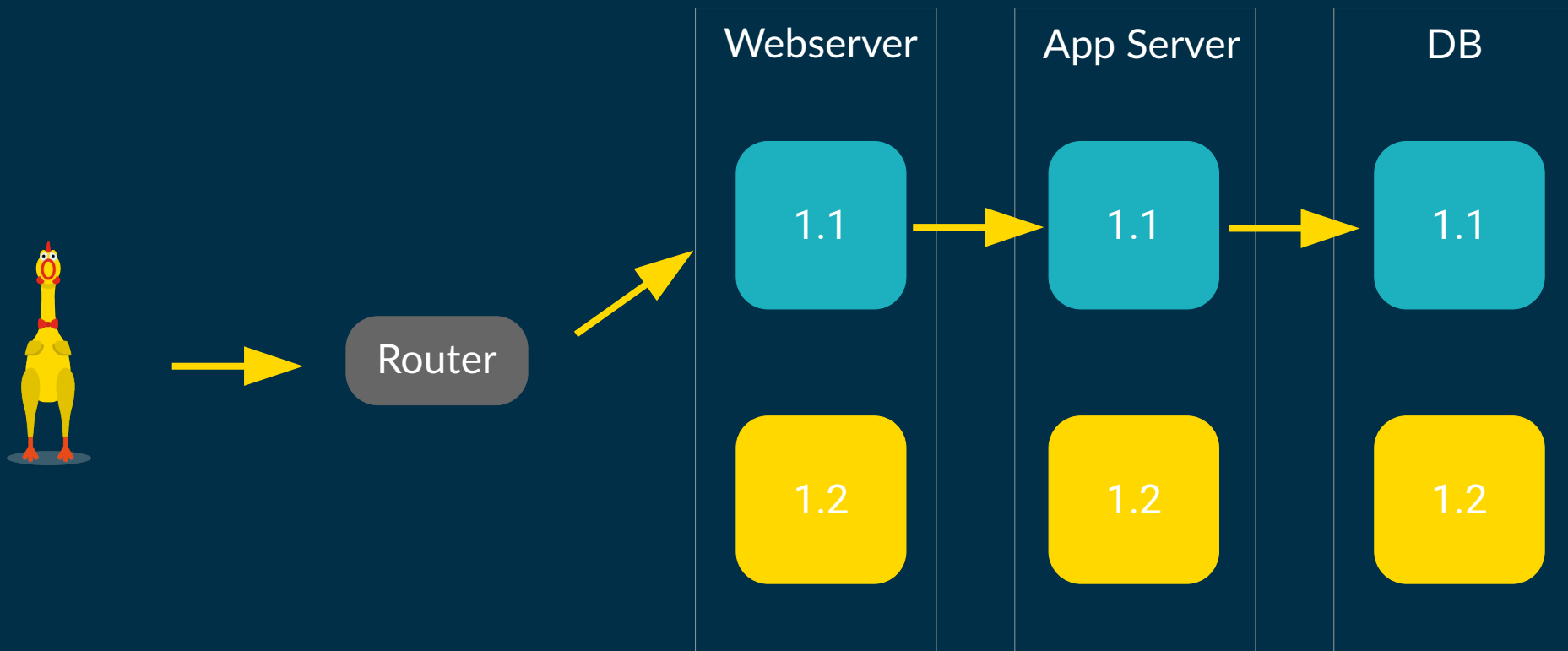
Canary releasing

Testing

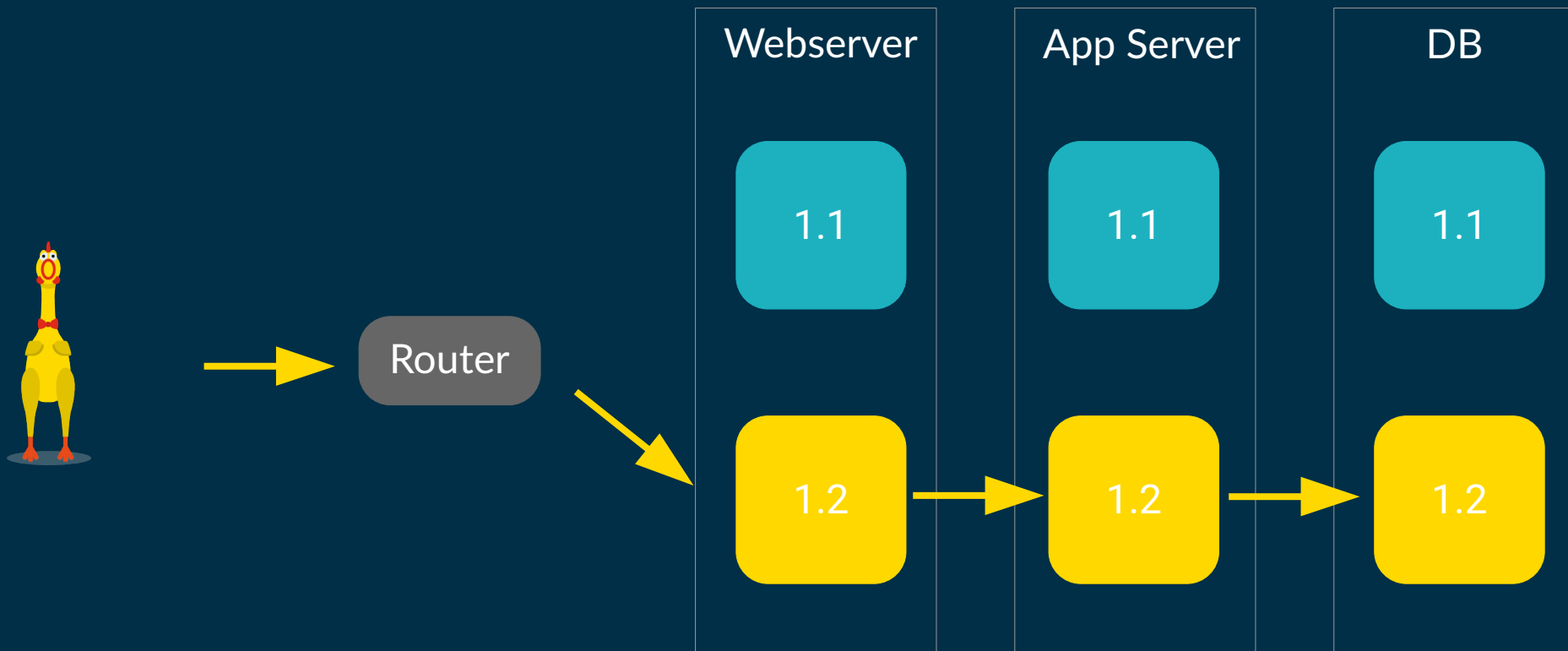


A good test concept is essential for Continuous Deployment.

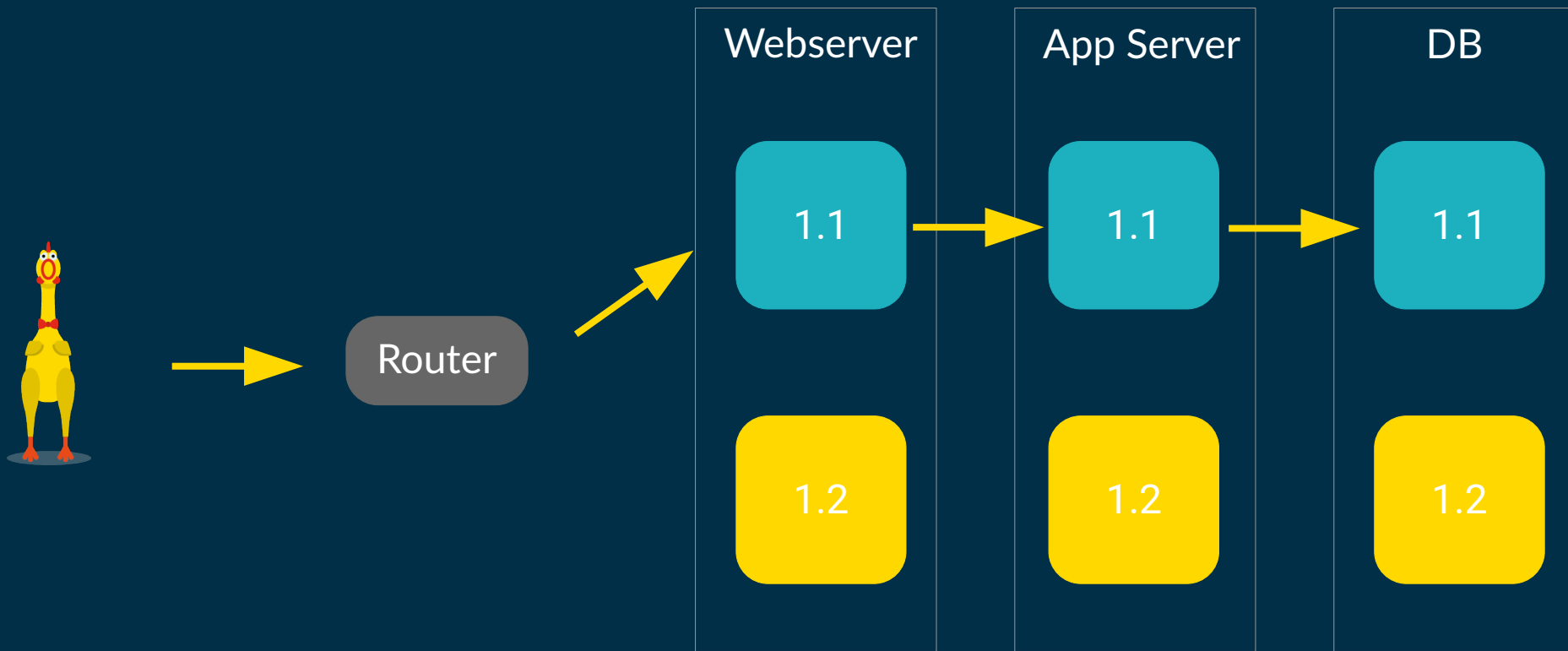
Blue - Yellow Deployments



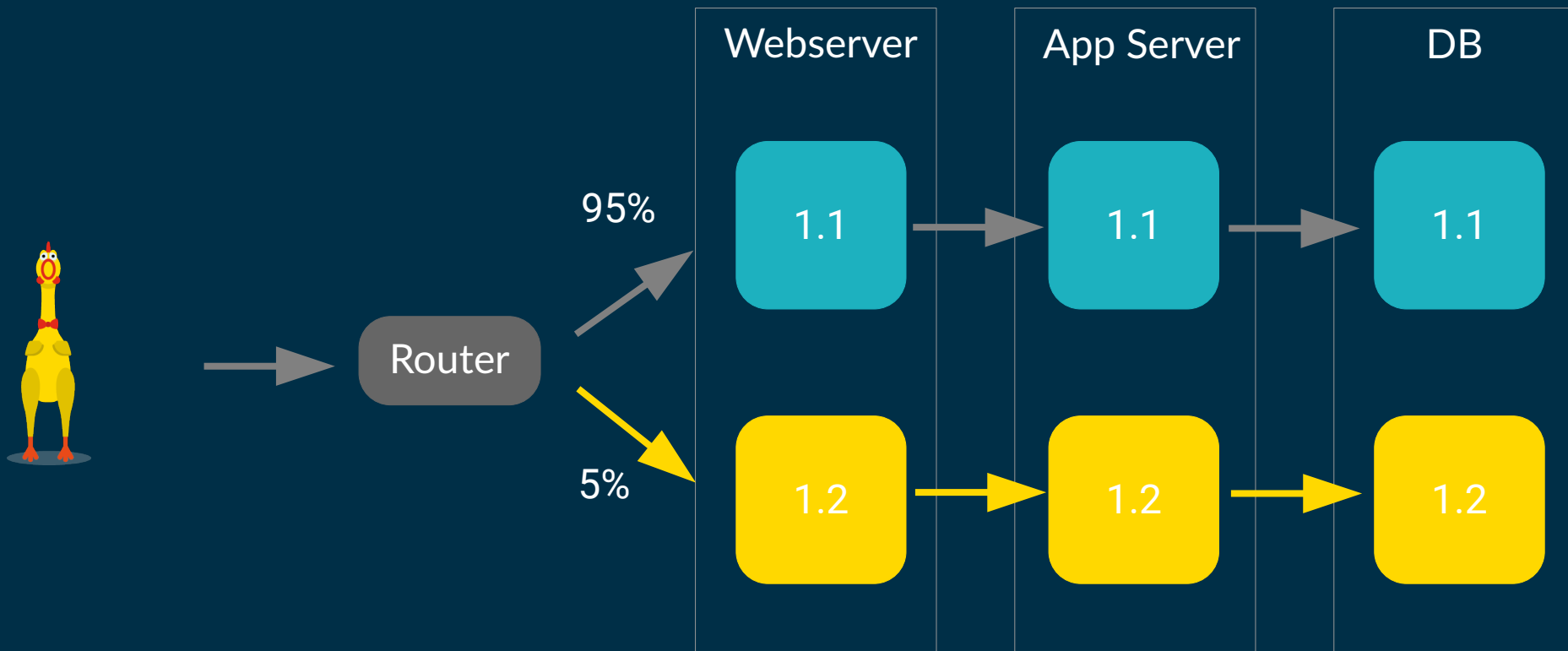
Blue - Yellow Deployments



Canary Releasing

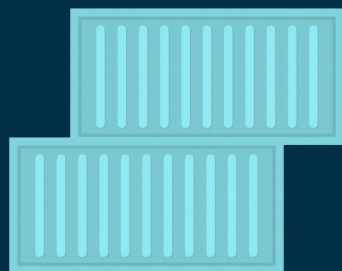


Canary Releasing





4



Continuous Delivery Pipeline

Highlevel CI/CD Pipeline



Build

Packaging

Automated Tests

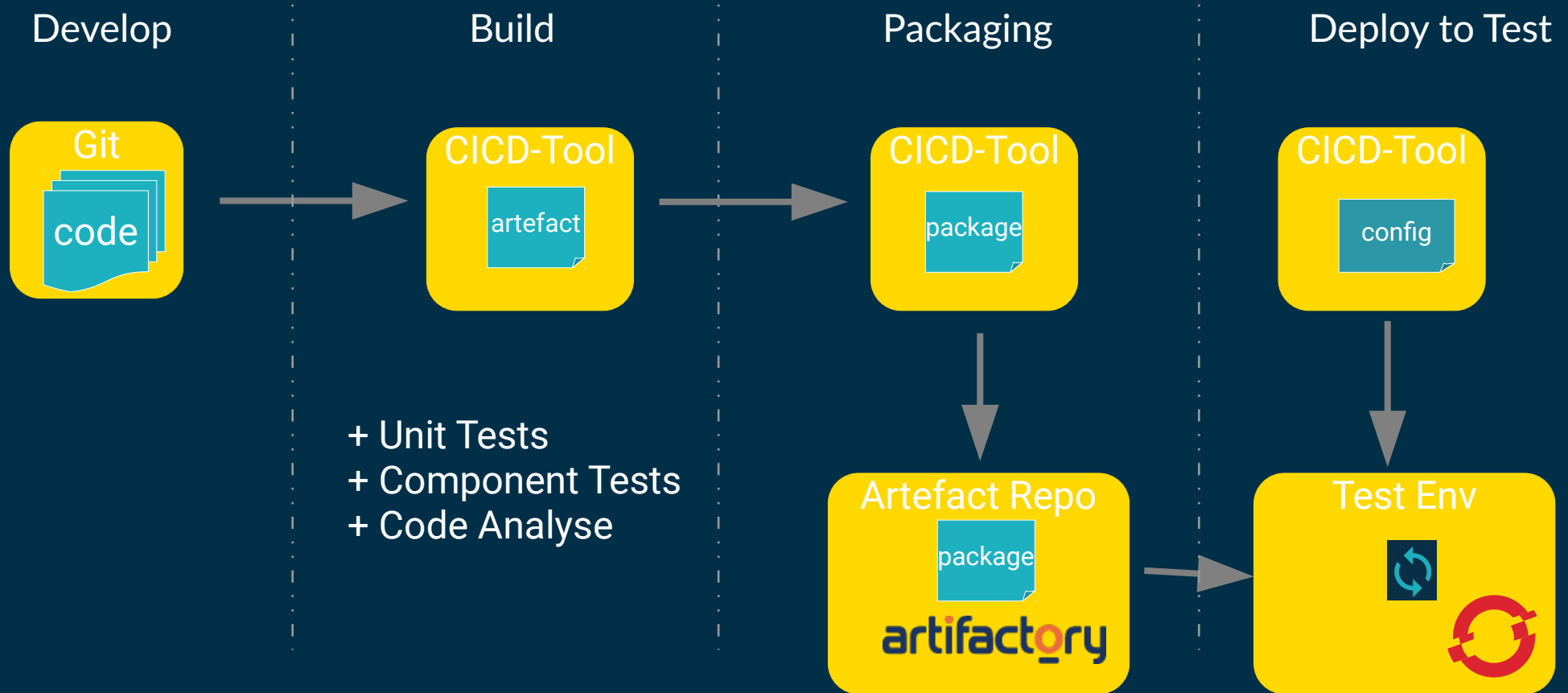
Manual Tests

Release

Developer Feedback

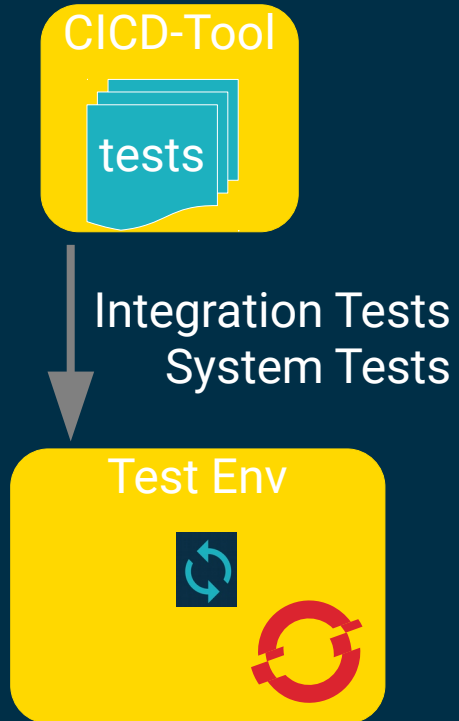


Example Pipeline (1)

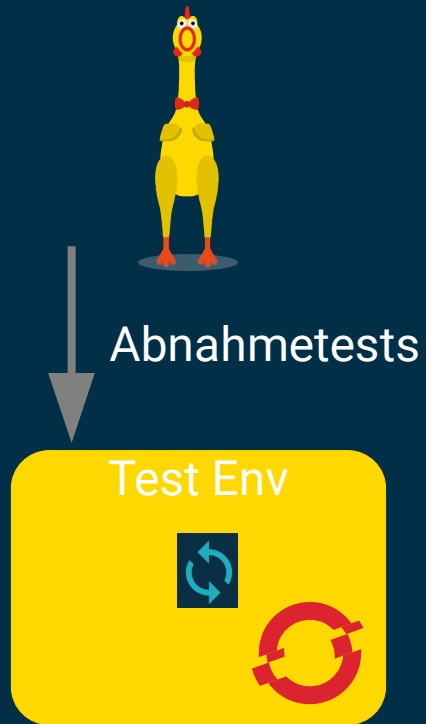


Example Pipeline (2)

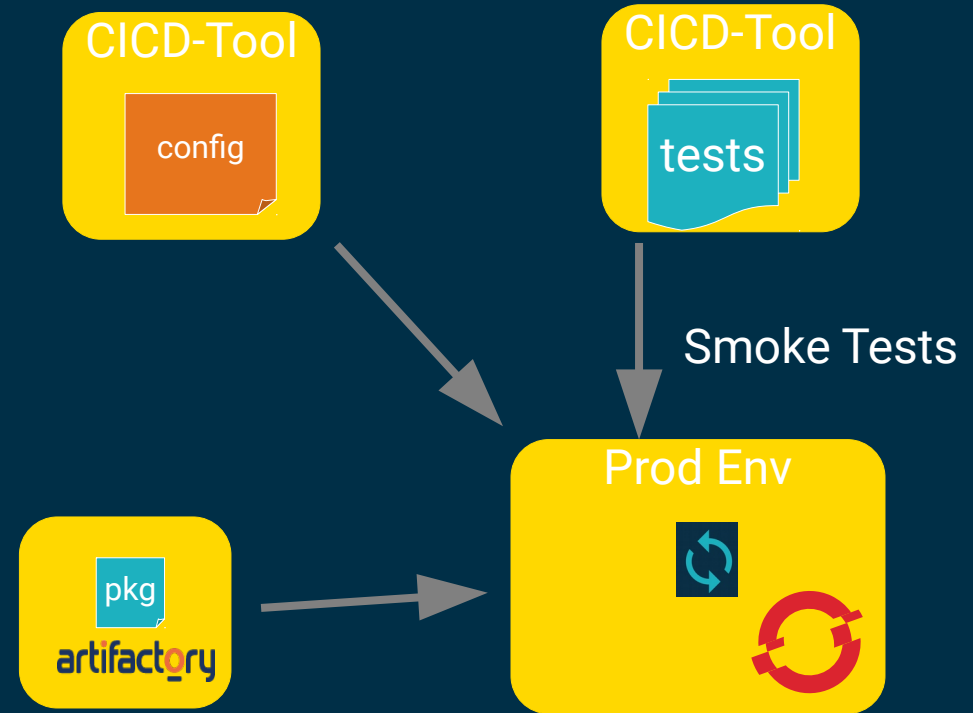
Automated Tests

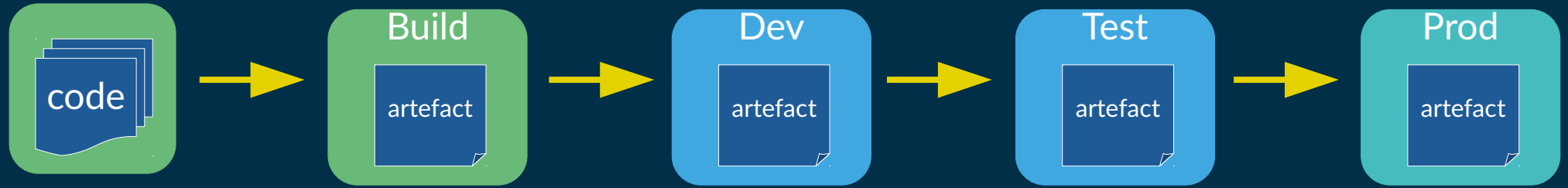


Manual Tests



Deploy to Prod





The same artifact is passed through the entire pipeline

A decorative header at the top of the slide featuring a series of overlapping circles in various shades of blue and teal, creating a cloud-like or bubble-like effect.

Why should we use Pipelines

Pipeline Advantages

- 1.Can be reviewed, forked, iterated upon and audited
- 2.Running pipelines survive master restart
- 3.Can stop and wait for human input
- 4.Support complex CI/CD requirements
- 5.DSL can be extended through shared libraries



Questions?

A decorative header consisting of a series of overlapping circles in various shades of blue, creating a cloud-like or bubble-like effect.




Jenkins

Once upon a time...

2001 CruiseControl

The screenshot displays the CruiseControl web interface. At the top, there are three tabs: 'Dashboard', 'Builds', and 'Administration'. The 'Builds' tab is currently selected. Below the tabs, a large green banner indicates a successful build: 'cce-windows passed (44 minutes ago)'. To the left of this text is a green checkmark icon. To the right are icons for a power button, a speech bubble, and a Twitter bird. Below the banner, the build details are listed: 'Build Time: 27 Nov 2007 09:51 GMT +08:00' and 'Duration: 7 minutes 40 seconds'. Below this, it says 'Build: build.8'. Underneath the build details, there are five sub-tabs: 'Artifacts', 'Modifications', 'Build Log', 'Tests', and 'Errors and Warnings'. The 'Modifications' sub-tab is selected, showing a list of changes. The first change is by 'bestfriendchris' [Chris & Gao Li] with the description 'Fixed issue with queued inactive status.' and revision '[rev. 3847]'. The second change is also by 'bestfriendchris' [rev. 3847] and describes a fix for a 'Fixed issue with queued inactive status.' in the file '/branches/cce/cruisecontrol/reporting/dashboard/webapp/javascripts/json_to_css.js'. On the right side of the interface, there is a 'Latest Builds' section with a list of recent builds. Each build is preceded by a green checkmark icon, except for 'about 17 hours ago build.7' which is preceded by a red exclamation mark icon. The builds listed are: '7 minutes ago build.9', '44 minutes ago build.8', 'about 17 hours ago build.7', 'about 17 hours ago', 'about 18 hours ago build.6', 'about 18 hours ago build.5', 'about 19 hours ago build.4', '1 day ago build.3', '1 day ago build.2', and '8 days ago build.1'.




Dashboard Builds Administration



cce-windows passed (44 minutes ago)   

Build Time: 27 Nov 2007 09:51 GMT +08:00 Duration: 7 minutes 40 seconds
Build: build.8

Artifacts Modifications Build Log Tests Errors and Warnings

Modifications

 **bestfriendchris** [Chris & Gao Li] Fixed issue with queued inactive status.
[rev. 3847]  /branches/cce/cruisecontrol/reporting/dashboard/jsunit/tests/json_to_css_test.html
[rev. 3847]  /branches/cce/cruisecontrol/reporting/dashboard/webapp/javascripts/json_to_css.js

Latest Builds  

- ✓ 7 minutes ago build.9
- ✓ 44 minutes ago build.8
- ✓ about 17 hours ago build.7
- ! about 17 hours ago
- ✓ about 18 hours ago build.6
- ✓ about 18 hours ago build.5
- ✓ about 19 hours ago build.4
- ✓ 1 day ago build.3
- ✓ 1 day ago build.2
- ✓ 8 days ago build.1

2005 Hudson

Hudson

?

[Hudson](#)

[New Job](#)

[Manage Hudson](#)

[People](#)

Build Queue

No builds in the queue.

Build Executor Status

No.	Status
1	Idle
2	Idle

Continuous integration builds for [REDACTED]

[edit description](#)

All +

S	W	Job ↓	Last Success	Last Failure	Last Duration	
		excalibur-slide-server-[REDACTED]	4 hours 34 minutes (#1)	N/A	22 seconds	
		[REDACTED]cocoon-live	2 hours 9 minutes (#1)	N/A	21 seconds	
		[REDACTED]workflow	2 hours 9 minutes (#5)	6 hours 24 minutes (#1)	15 seconds	
		hippo-cms-[REDACTED]	2 hours 8 minutes (#9)	N/A	6 minutes 32 seconds	
		hippo-repository-1.2.13	5 hours 37 minutes (#1)	N/A	2 minutes 20 seconds	
		repository-initialization	N/A	9 minutes 40 seconds (#24)	2 seconds	
		workflows-[REDACTED]	5 hours 10 minutes (#1)	N/A	22 seconds	

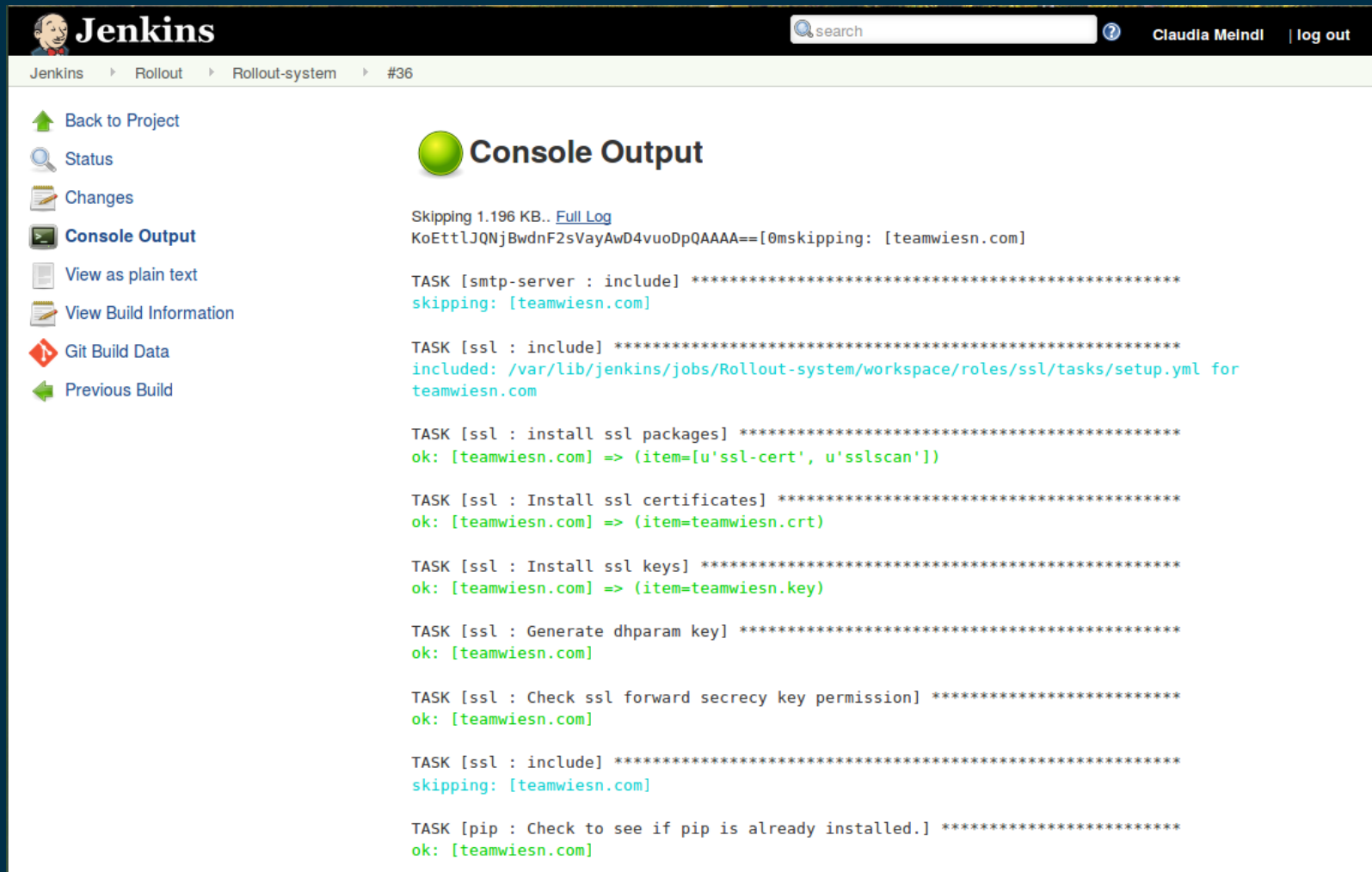
Icon: [S](#) [M](#) [L](#)

[Legend](#) for all for failures

[Hudson ver. 1.184](#)

Done

2011 Jenkins



The screenshot shows the Jenkins web interface. At the top, there's a header with the Jenkins logo, a search bar, and the user name 'Claudia Meindl' with a 'log out' link. Below the header, a breadcrumb trail shows 'Jenkins > Rollout > Rollout-system > #36'. On the left sidebar, there are links: 'Back to Project', 'Status', 'Changes', 'Console Output' (which is highlighted), 'View as plain text', 'View Build Information', 'Git Build Data', and 'Previous Build'. The main content area is titled 'Console Output' with a green sphere icon. It displays the output of a build, starting with 'Skipping 1.196 KB.. [Full Log](#)'. The output shows several tasks being executed, mostly related to SSL setup, with status messages like 'skipping: [teamwiesn.com]', 'TASK [smtp-server : include]', 'TASK [ssl : include]', 'TASK [ssl : install ssl packages]', 'TASK [ssl : Install ssl certificates]', 'TASK [ssl : Install ssl keys]', 'TASK [ssl : Generate dhparam key]', 'TASK [ssl : Check ssl forward secrecy key permission]', 'TASK [ssl : include]', and 'TASK [pip : Check to see if pip is already installed.]'. Each task output is preceded by 'ok: [teamwiesn.com] => (item=[u'ssl-cert', u'sslscan'])' or similar, indicating successful execution or skipping.

Jenkins

Rollout > Rollout-system > #36

- Back to Project
- Status
- Changes
- Console Output**
- View as plain text
- View Build Information
- Git Build Data
- Previous Build

Console Output

Skipping 1.196 KB.. [Full Log](#)
KoEttlJQNjBwdnF2sVayAwD4vuoDpQAAAA==[0mskipping: [teamwiesn.com]

TASK [smtp-server : include] *****
skipping: [teamwiesn.com]

TASK [ssl : include] *****
included: /var/lib/jenkins/jobs/Rollout-system/workspace/roles/ssl/tasks/setup.yml for teamwiesn.com

TASK [ssl : install ssl packages] *****
ok: [teamwiesn.com] => (item=[u'ssl-cert', u'sslscan'])

TASK [ssl : Install ssl certificates] *****
ok: [teamwiesn.com] => (item=teamwiesn.crt)

TASK [ssl : Install ssl keys] *****
ok: [teamwiesn.com] => (item=teamwiesn.key)

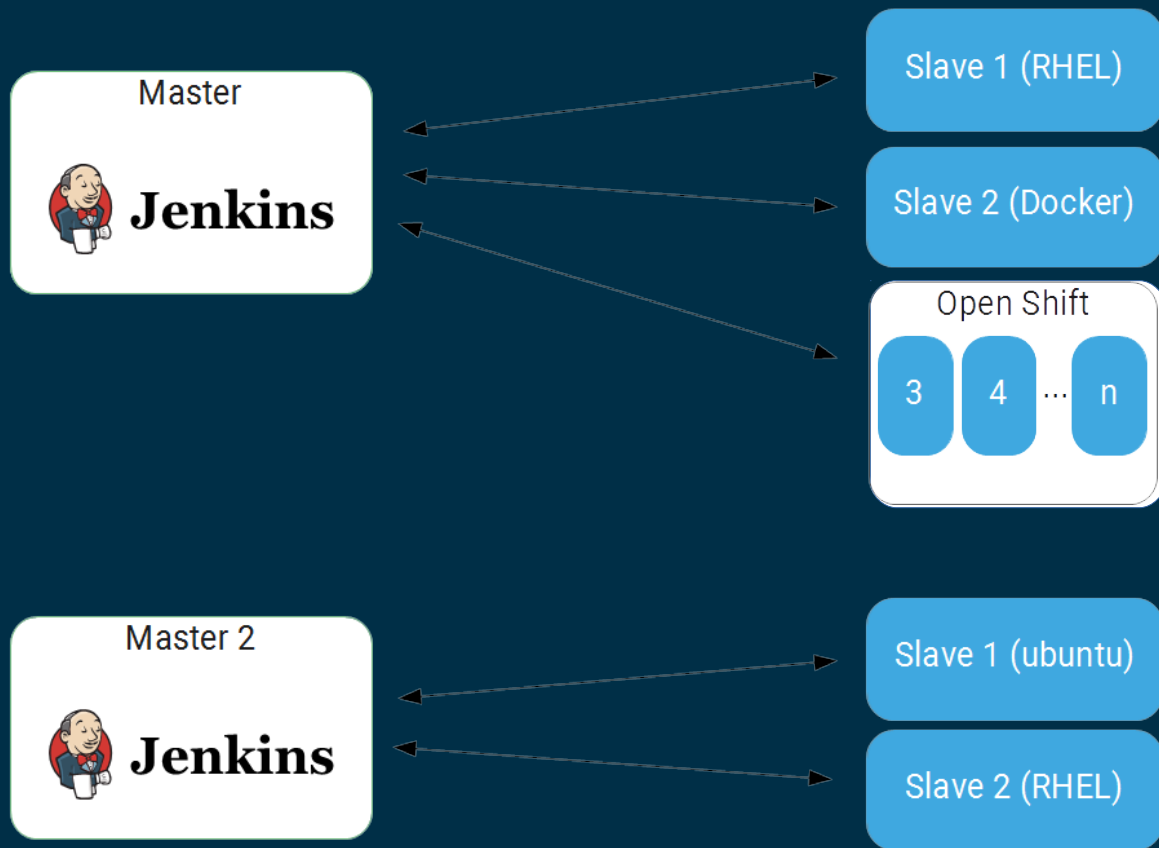
TASK [ssl : Generate dhparam key] *****
ok: [teamwiesn.com]

TASK [ssl : Check ssl forward secrecy key permission] *****
ok: [teamwiesn.com]

TASK [ssl : include] *****
skipping: [teamwiesn.com]

TASK [pip : Check to see if pip is already installed.] *****
ok: [teamwiesn.com]

Architecture



Architecture

- Consists of Masters
 - Stateful
 - Management of Jobs, Credentials, ...
 - Plugins (managed on master)
- and Slaves
 - Actually run the jobs
 - Stateless

Scaling a Build Infrastructure

- $\text{jobs} = \text{number of developers} * 3.333$
- $\text{masters} = \text{number of jobs} / 500$
- $\text{executors} = \text{number of jobs} * 0.03$

Source : [User Handbook - Architecting for Scale](#)

Jenkinsfiles

- Define your Pipeline
- Written in Groovy
 - Java based Scripting Language
- Stored on Jenkins-Master or in Git
 - Git is strongly preferred !

Jenkinsfiles : Declarative vs Scripted

Declarative

- Only DSL/Steps allowed
- `script {}` block for scripting

```
pipeline {  
    agent any  
  
    stages {  
        stage('Build') {  
            steps {  
                echo 'Building...'  
            }  
        }  
    }  
}
```

Scripted

- Imperative Groovy script
- Deprecated

```
node {  
    stage('Build') {  
        echo 'Building...'  
    }  
}
```

Jenkinsfiles : Declarative Structure

1.pipeline keyword

2.Declarative Directives

- Run on the Master
- Set Pipeline Properties

3.Stages

- Run on the slaves

4.Post Directives

- Run on the Master or Slave
- Conditional on Pipeline Status

```
pipeline {  
    agent any  
    parameters {  
        string defaultValue: 'exampleParam', name: 'myParam'  
    }  
    environment {  
        MY_ENV = "exampleEnv"  
    }  
    tools {  
        maven 'Maven 3.5.0'  
    }  
    options {  
        buildDiscarder logRotator(numToKeepStr: '5')  
    }  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello'  
            }  
        }  
        stage('World') {  
            steps {  
                sh 'echo World'  
            }  
        }  
    }  
    post {  
        unsuccessful {  
            echo 'Pipeline unsuccessful'  
        }  
    }  
}
```

Execution

1.Setup

- 1 Fetch and Validate Jenkinsfile
- 2 Clones Shared Library (if defined)
- 3 Parse Declarative Directives
- 4 Clones Source Git Repository

2.Stages

- Runs Pipeline Steps on Slaves

3.Post Directives

- Cleanup
- Notifications

Limitations

- Directive update on master
 - Triggers, Parameters, etc.
 - Pipeline needs to run once for master to get update
- Pipelines must survive restarts
 - [Scripted Groovy Code gets transformed](#)
 - Datastructures must be serelizable

Tools

Tools can be installed in two ways :

- Tool installer
 - Tool is installed on the slave at the start of the execution
 - Only supports selected tools (solution: custom-tools-plugin)
 - Docker agent
 - Use container as execution environment
- more in the labs

```
pipeline {  
    agent any  
    ...  
    tools {  
        maven 'Maven 3.5.0'  
    }  
    ...  
}
```

```
pipeline {  
    agent {  
        docker { image 'node:14-alpine' }  
    }  
    ...  
}
```

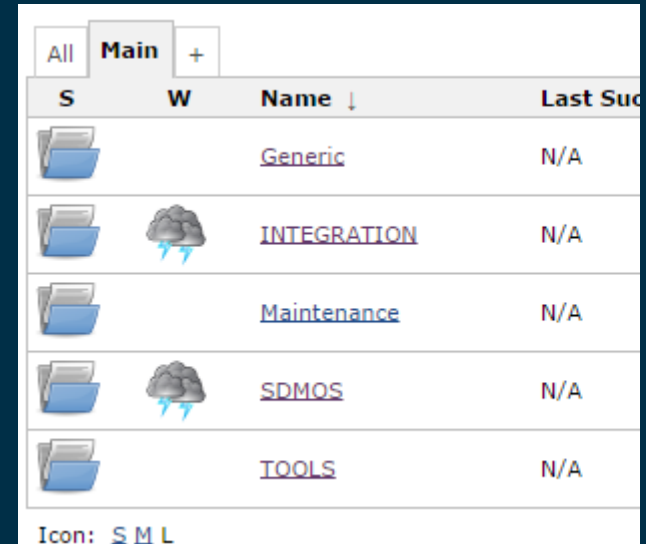

Best Practices

- Keep everything in version control
- Organise Projects/Teams with folders
- Keep slaves stateless
- Reuse functionalities in shared libraries
- Debug with Replays
- Only one Source Repository Checkout








Best Practices : Folders

Organise Projects/Teams with folders

- Keep things organized
- Control Permissions
- Control Credentials Access



The screenshot shows a file explorer window with a sidebar on the left containing 'All' and 'Main' tabs, and a '+' icon. The main area displays a table of folders. The table has columns for 'S' (Status), 'W' (Warning/Icon), 'Name', and 'Last Suc'. The folders listed are 'Generic', 'INTEGRATION', 'Maintenance', 'SDMQS', and 'TOOLS'. The 'INTEGRATION' and 'SDMQS' folders have a cloud with lightning bolt icon next to them. The 'Name' column is sorted in descending order. At the bottom, there is an 'Icon' section with 'S M L' options.

S	W	Name ↓	Last Suc
		Generic	N/A
		INTEGRATION	N/A
		Maintenance	N/A
		SDMQS	N/A
		TOOLS	N/A

Icon: [S](#) [M](#) [L](#)

Best Practices : Stateless Slaves

Slaves should and Pipeline executions should be stateless

- Don't depend on artifacts from previous execution
- Archive artifacts
- Clean up slave workspace
- Clean up older builds

```
pipeline {
    ...
    options {
        buildDiscarder logRotator(numToKeepStr: '5')
    }
    ...
    stages {
        stage('Hello') {
            steps {
                sh 'mvn clean install'
                archiveArtifacts 'log/test.log'
            }
        }
    }
    post {
        unsuccessful {
            cleanWs()
        }
    }
}
```

Best Practices : Shared Libraries

Reuse functionalities in shared libraries

- Use same workflow for similar projects
 - eg. all spring boot applications
- Put common functionality in shared library
 - Easier to maintain and test and update

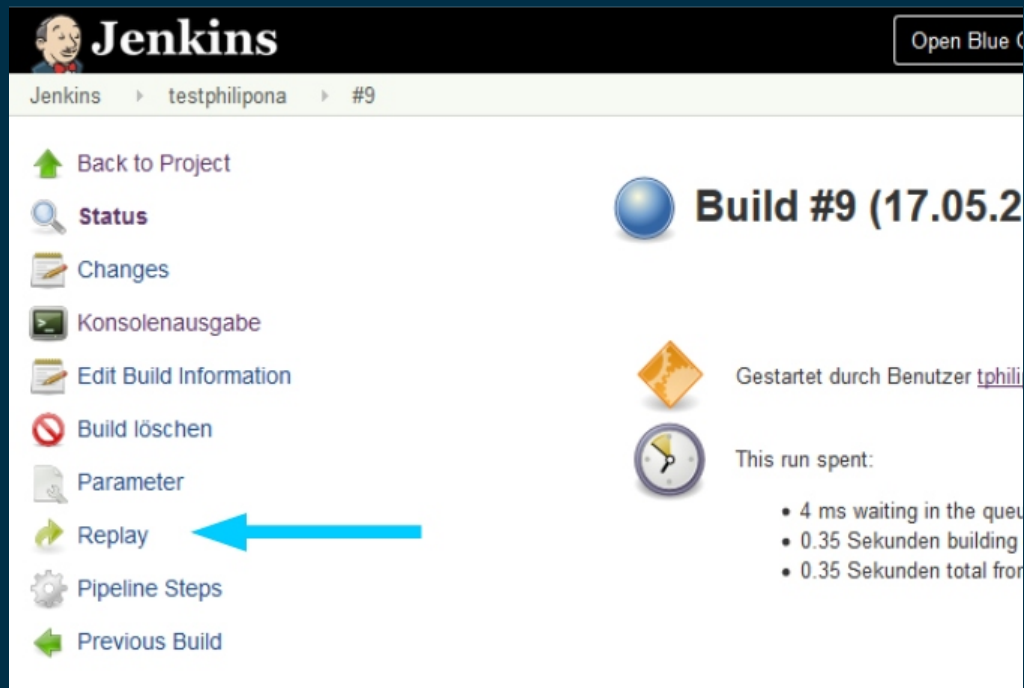
Example : [Puzzle Jenkins Shared Library \(GPL-3.0\)](#)

Best Practices : Debug with Replays

Use Replays to :

- View Jenkinsfiles of past executions
- Try out changes without committing to git

→ Remember to persist your changes in version control !



Best Practices : Only one Checkout

Disable default declarative checkout if you explicitly checkout later.

```
pipeline {
  ...
  options {
    skipDefaultCheckout true
  }
  ...
  stages {
    stage('Setup') {
      steps {
        git(credentialsId: 'mysecret'
            url: 'https://github.com/myorg/myrepo.git')
      }
    }
  }
  ...
}
```

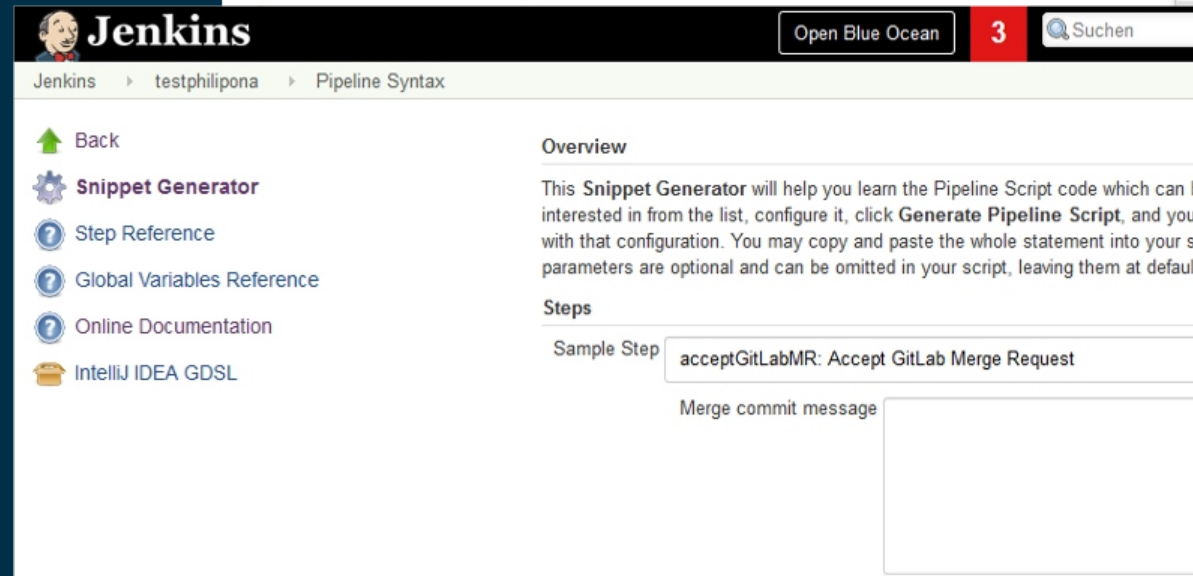
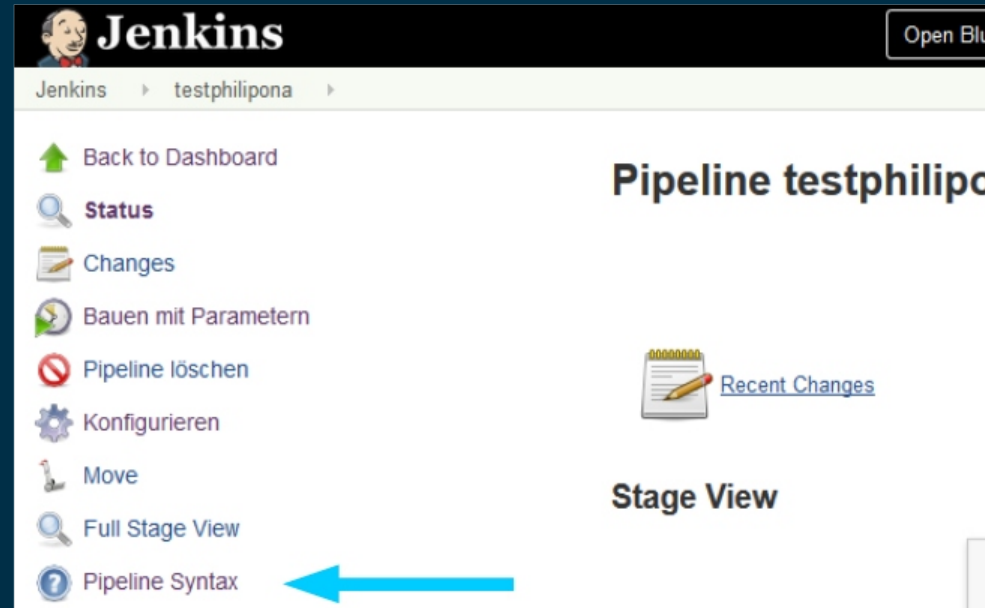
Resources

In Jenkins :

- Snippet/Directive generator
- Jenkins Step reference
- Shared Library

Online :

- [Jenkins User Handbook](#)
 - [Step reference](#)
- [Groovy](#)
 - Collections





Questions?