



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

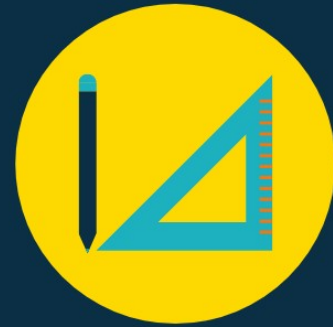
# TECHLAB OBJECTIVES



Achieve  
theory  
base



Learn basic  
Jenkins Pipeline  
features



Learn  
advanced  
Jenkins  
Pipeline use-  
cases

A decorative header at the top of the slide consisting of 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»

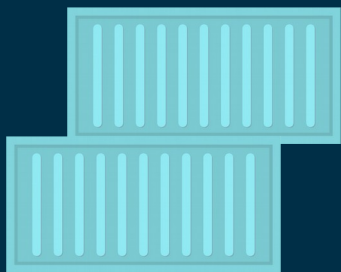


1991: Object Oriented Design: With Applications, Grady Booch



1997: Extreme Programming Explained

1



# Continuous Integration



code  
check in



automated  
building  
and  
testing



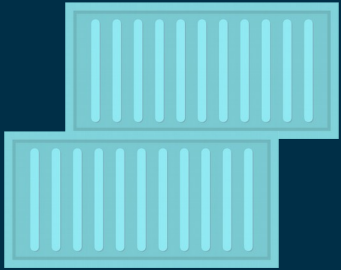
code analyses  
/  
reporting



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);  
    }  
}  
...
```

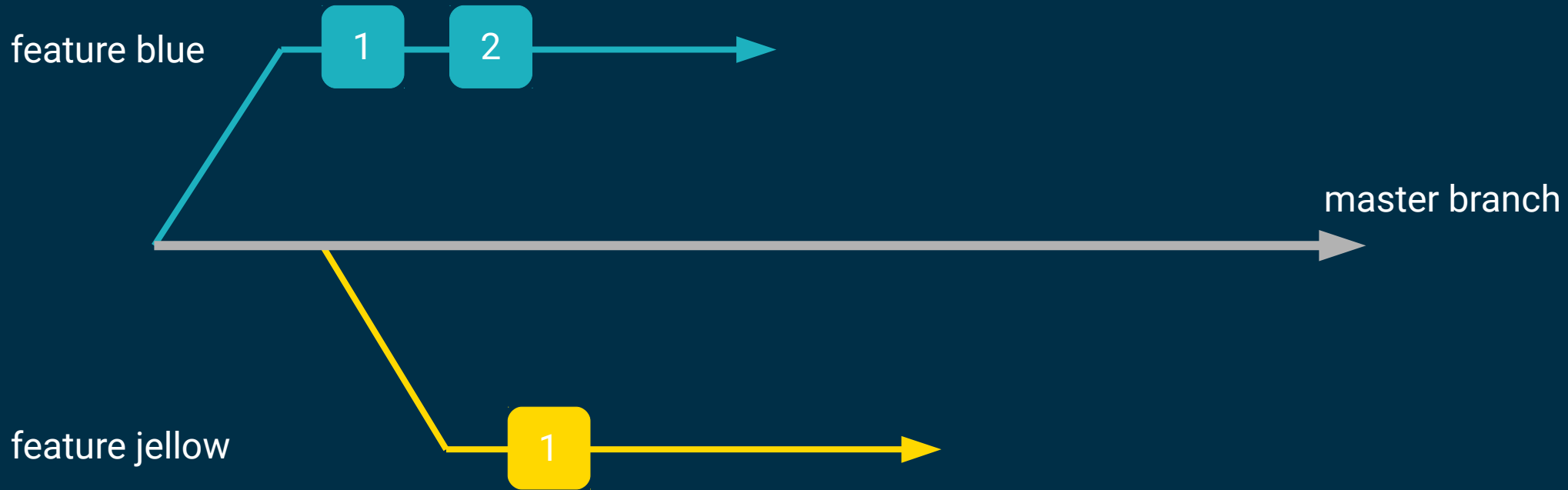
# Big Scary Merge of Feature Branches



# Big Scary Merge of feature branches

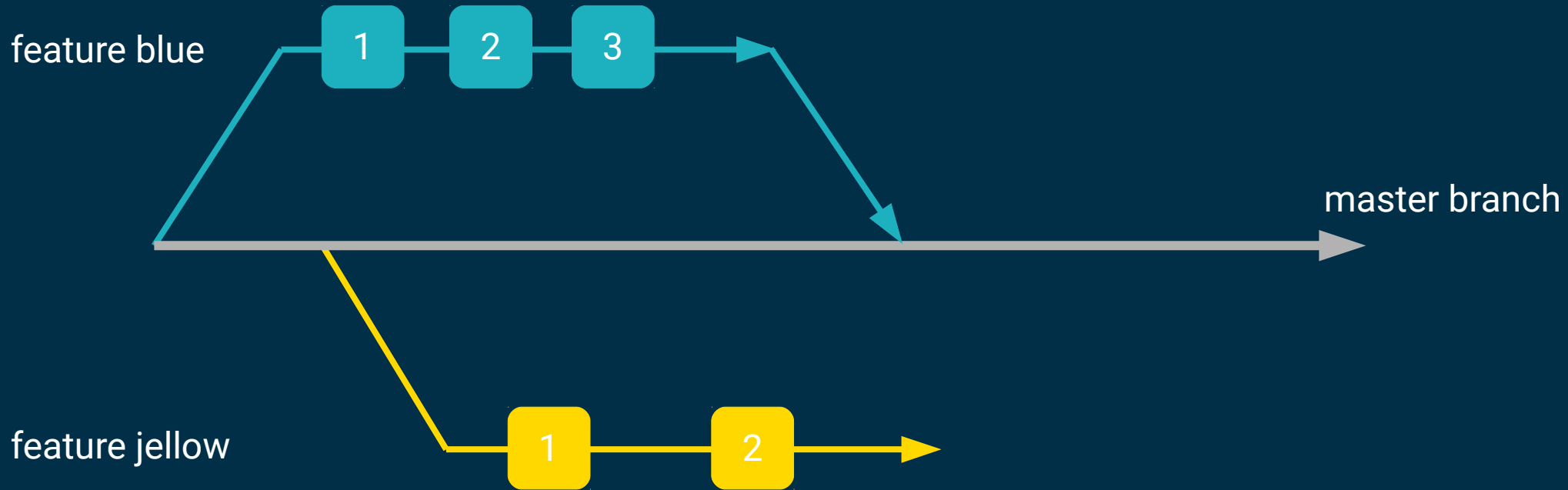


# Big Scary Merge of feature branches

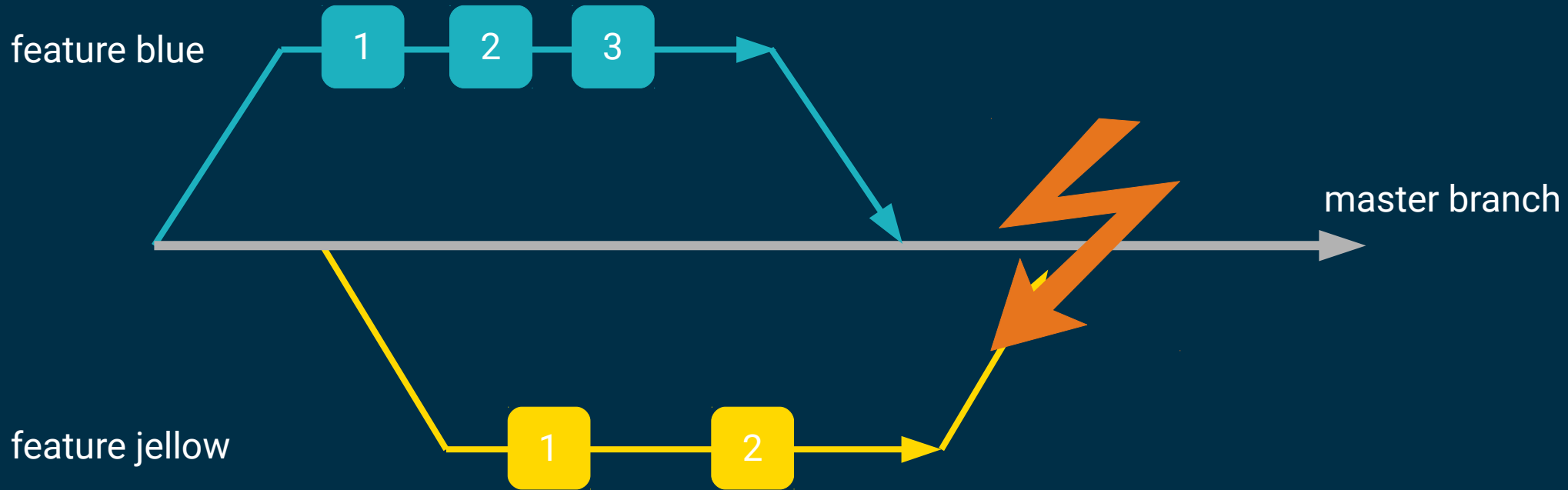




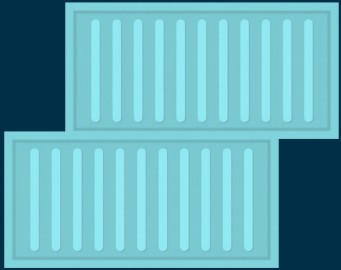
# Big Scary Merge of feature branches



# Big Scary Merge of feature branches



# 3



## Continuous Deployment

# Continuous Deployment

Each release is automatically deployed in production

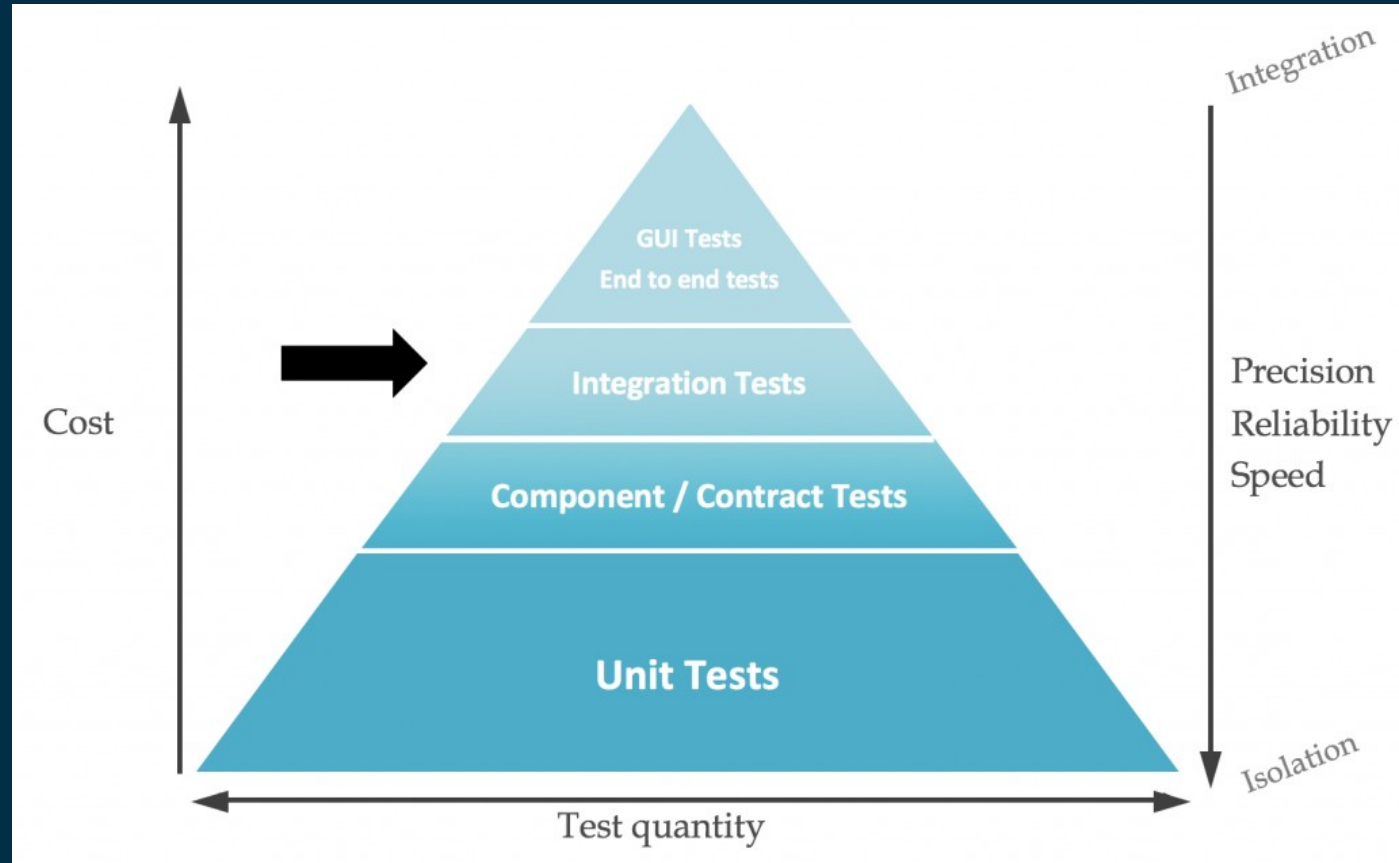
High test coverage needed

Uninterrupted deployments required

Blue – Green (Yellow) deployments

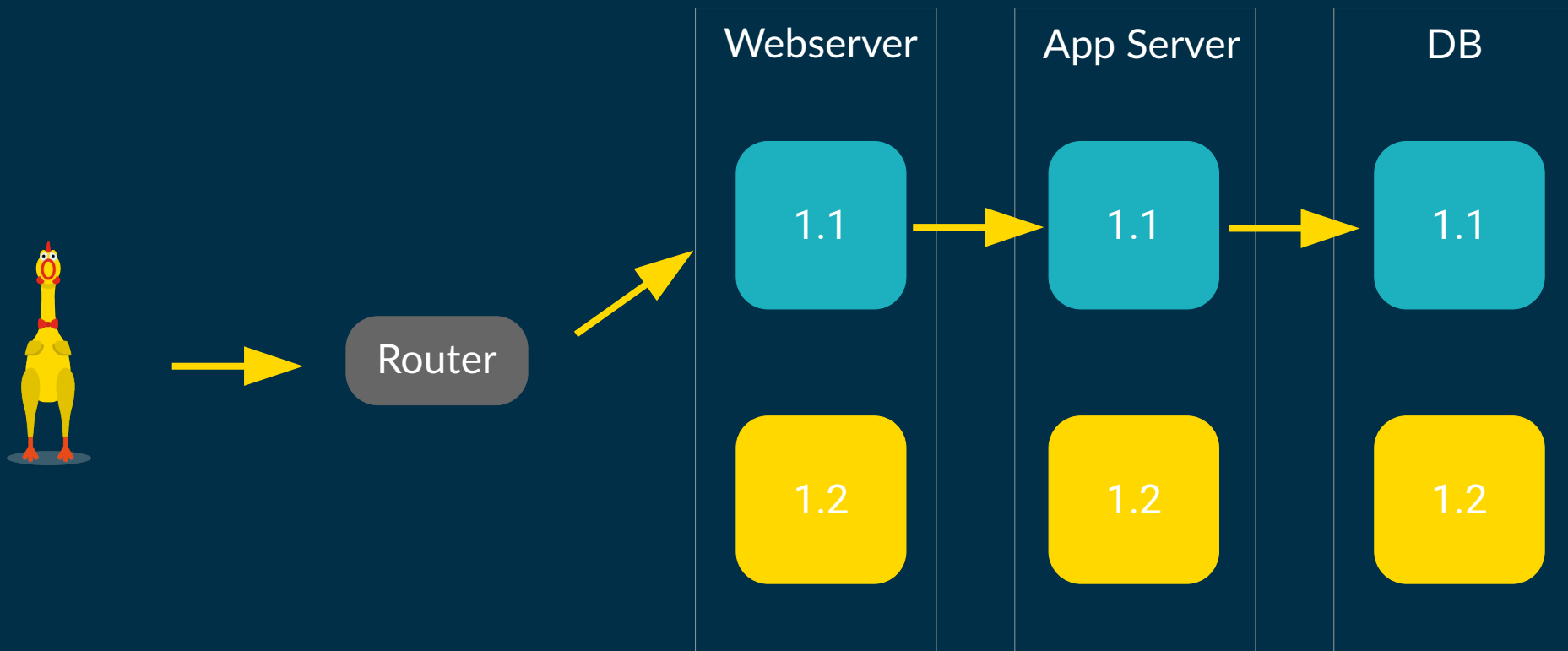
Canary releasing

# Testing

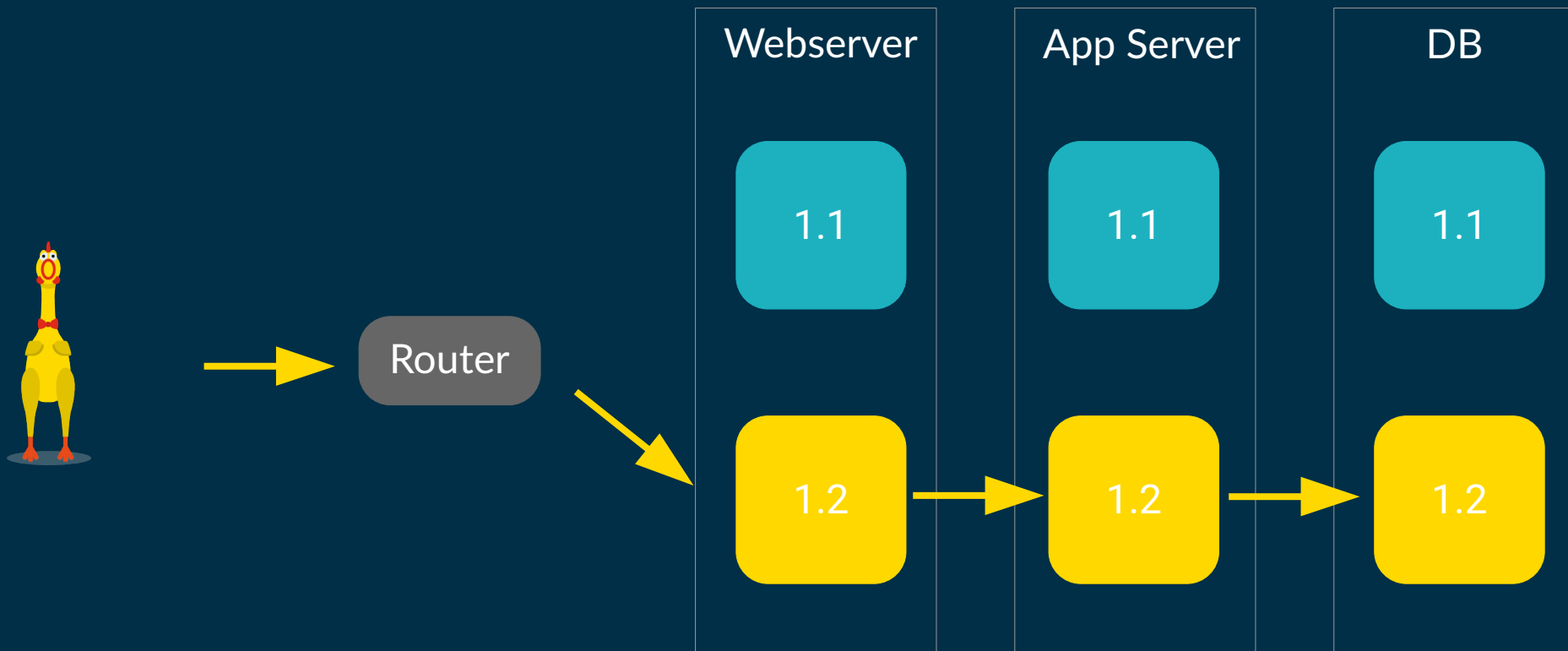


A good test concept is essential for Continuous Deployment.

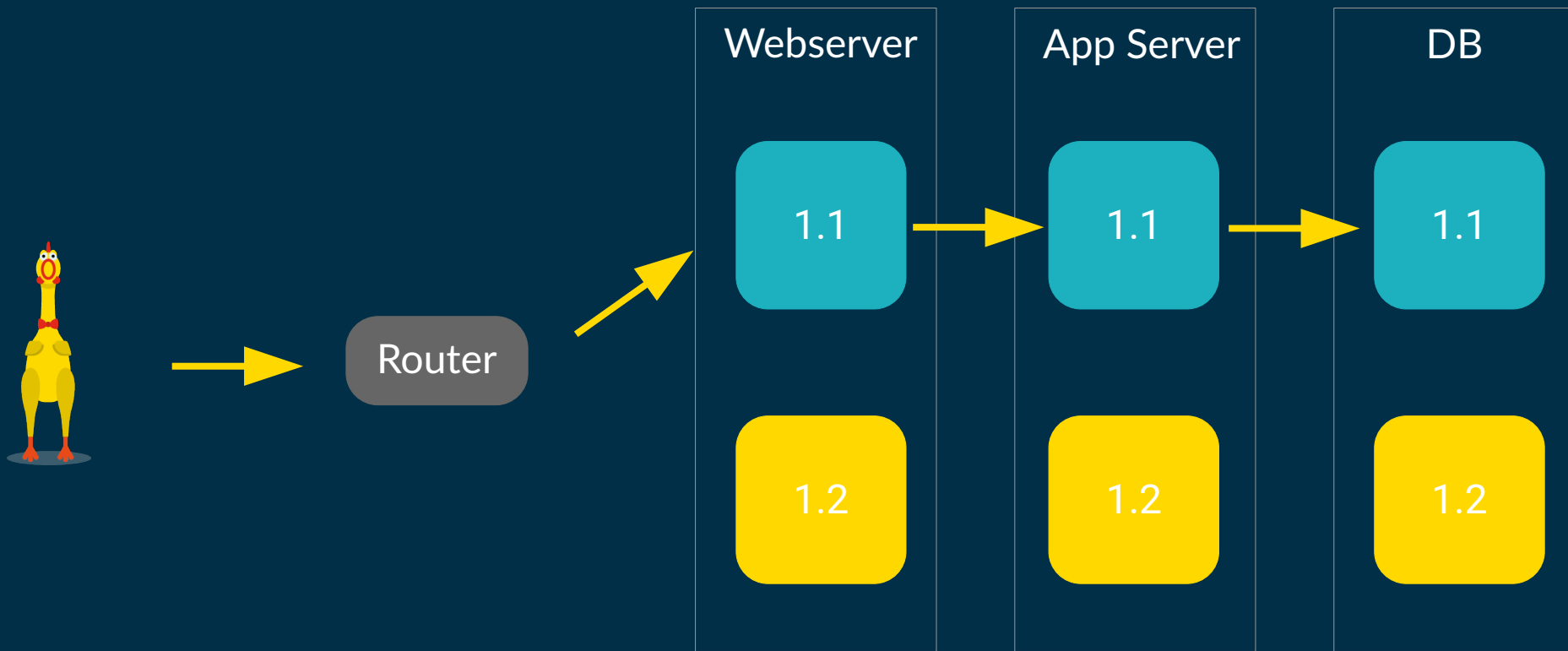
# Blue - Yellow Deployments



# Blue - Yellow Deployments

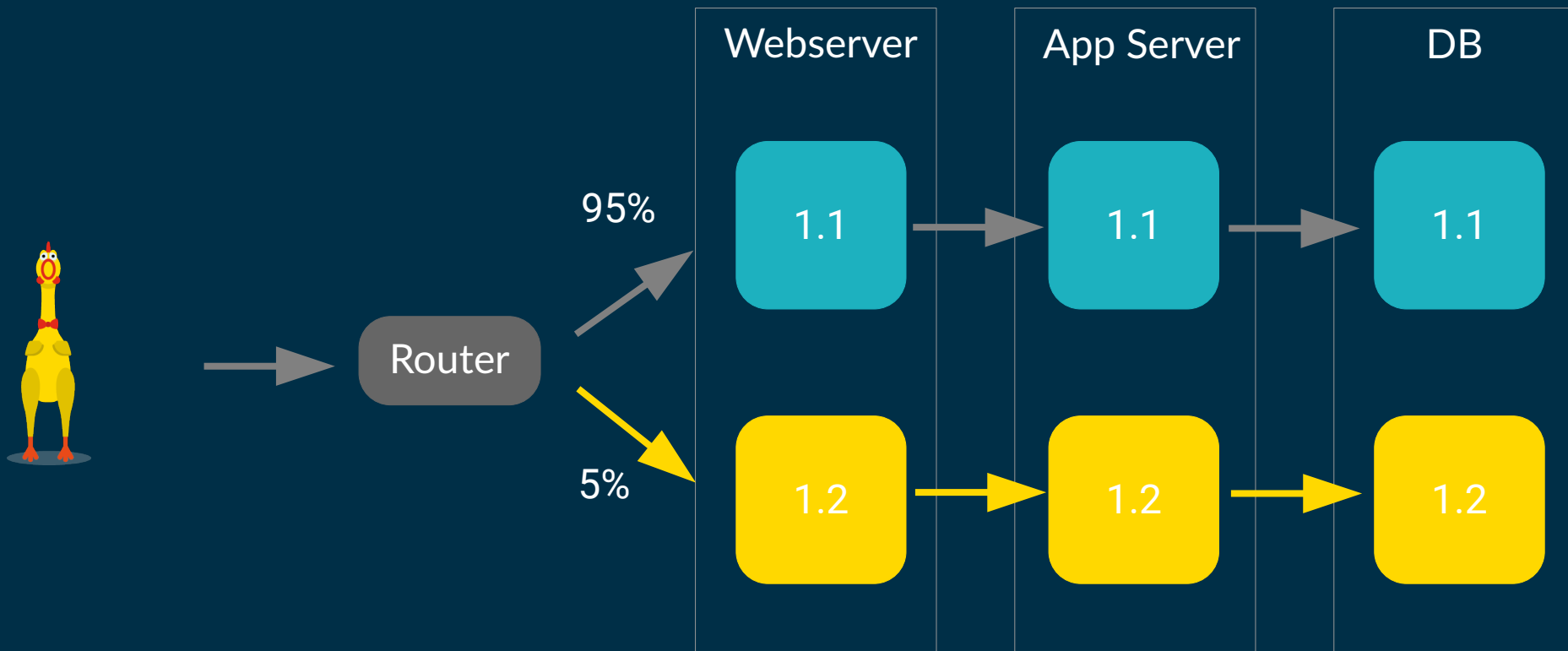


# Canary Releasing



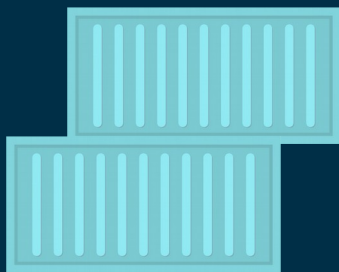


# Canary Releasing





# 4



## Continuous Delivery Pipeline

# Highlevel CI/CD Pipeline



Build

Packaging

Automated Tests

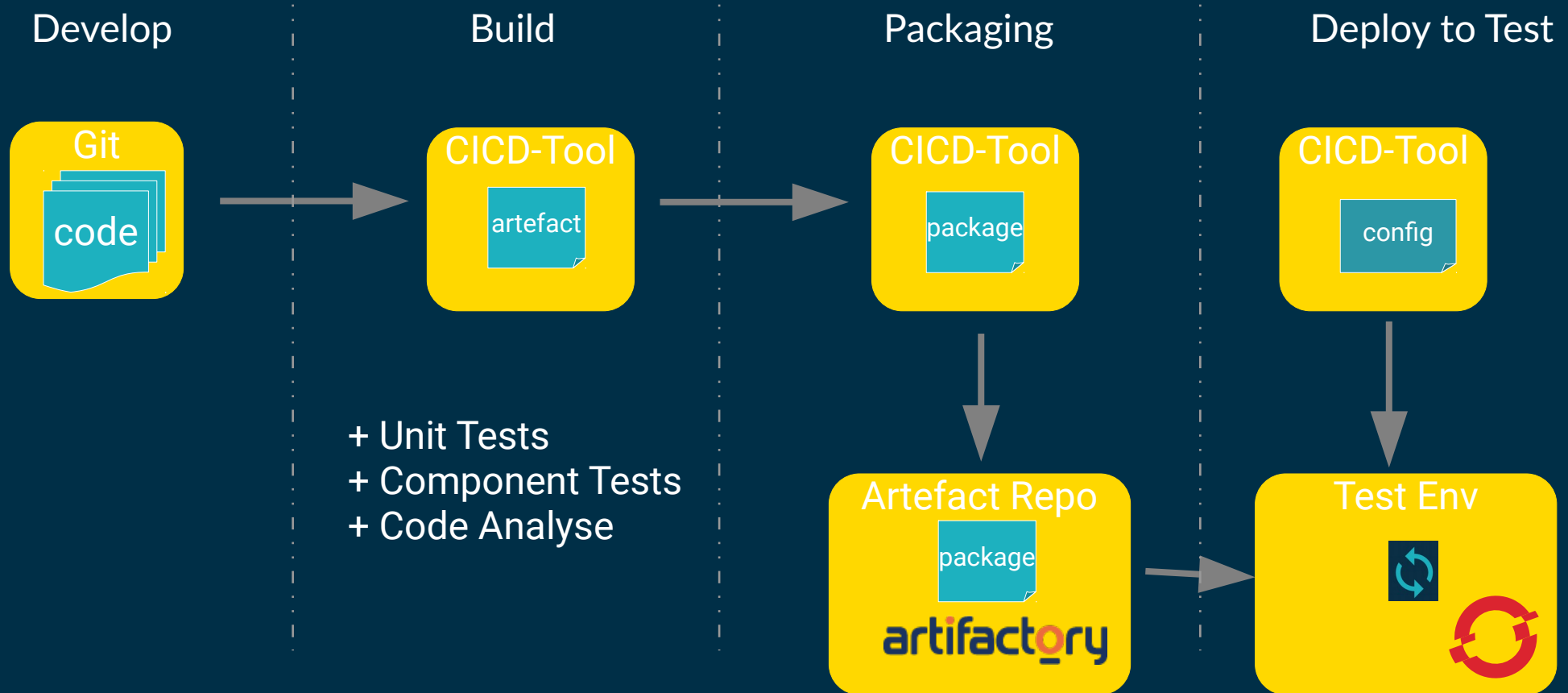
Manual Tests

Release

Developer Feedback

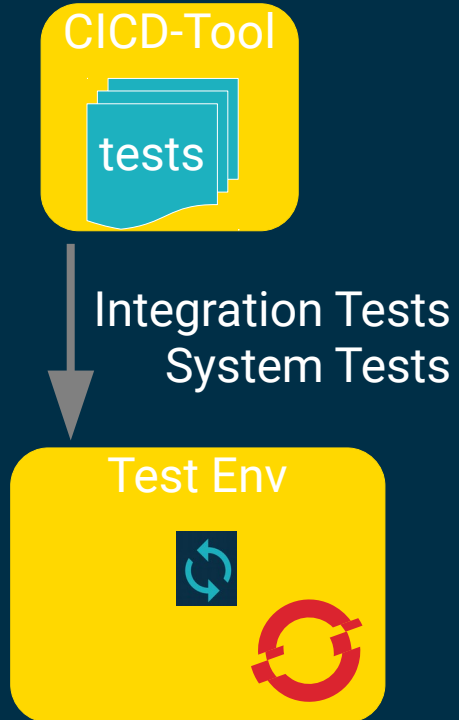


# Example Pipeline (1)

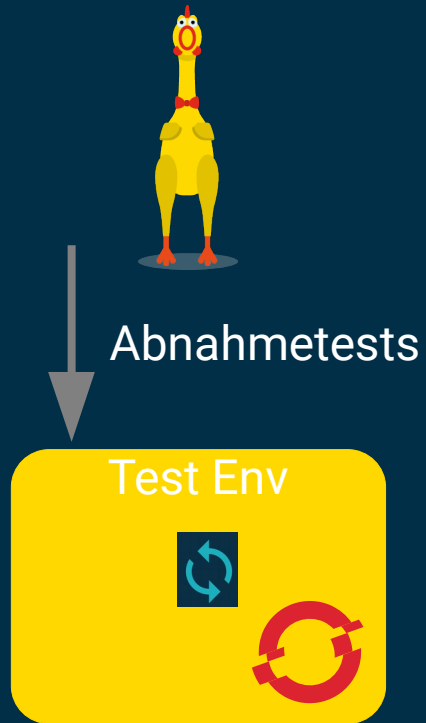


# Example Pipeline (2)

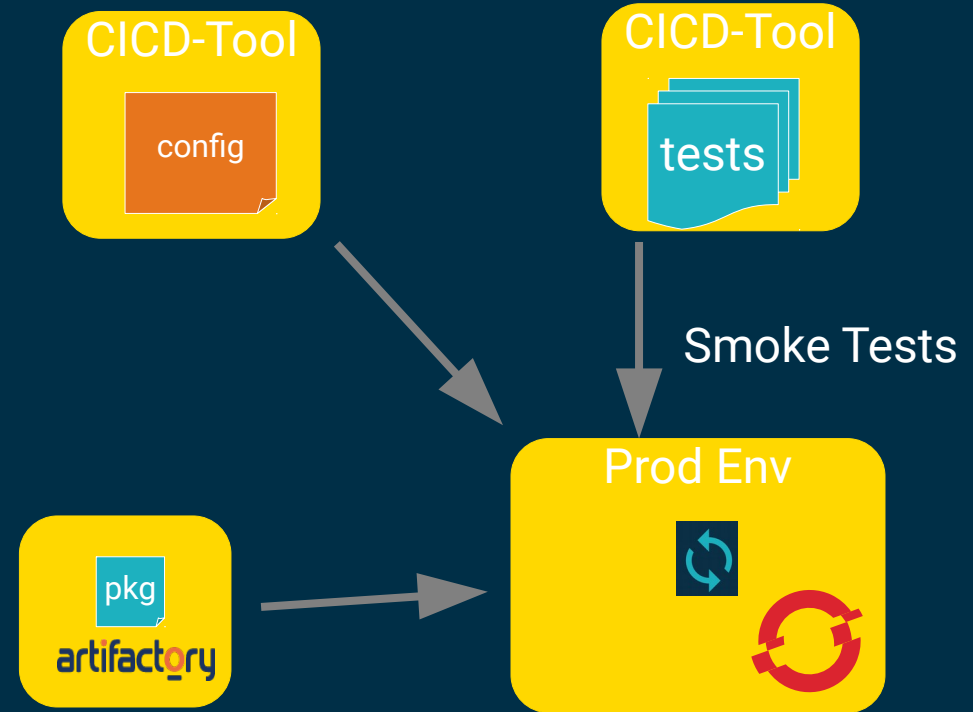
## Automated Tests

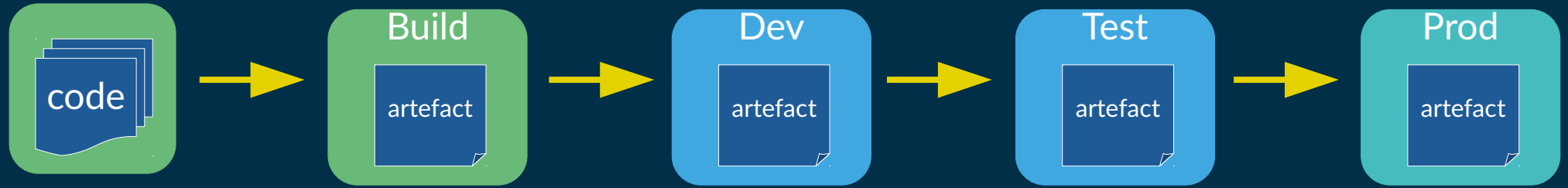


## Manual Tests



## Deploy to Prod





The **same** artifact is passed through the entire pipeline

A decorative header consisting of numerous overlapping circles in various shades of blue, creating a bokeh-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' (which is active), and 'Administration'. The main content area features a large green banner with a white checkmark icon, indicating a successful build. The banner text reads: 'cce-windows passed (44 minutes ago)', 'Build Time: 27 Nov 2007 09:51 GMT +08:00', and 'Duration: 7 minutes 40 seconds'. Below the banner, there are five sub-tabs: 'Artifacts', 'Modifications' (which is active), 'Build Log', 'Tests', and 'Errors and Warnings'. The 'Modifications' section shows a commit by 'bestfriendchris' with the message '[Chris & Gao Li] Fixed issue with queued inactive status.' and two files: '/branches/cce/cruisecontrol/reporting/dashboard/jsunit/tests/json\_to\_css\_test.html' and '/branches/cce/cruisecontrol/reporting/dashboard/webapp/javascripts/json\_to\_css.js'. On the right side, there is a 'Latest Builds' section with a list of builds, each preceded by a green checkmark icon. The builds are: '7 minutes ago build.9', '44 minutes ago build.8', 'about 17 hours ago build.7', 'about 17 hours ago' (highlighted in pink), '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

?

[ENABLE AUTO REFRESH](#)

[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	
		<a href="#">excalibur-slide-server-[REDACTED]</a>	4 hours 34 minutes ( <a href="#">#1</a> )	N/A	22 seconds	
		<a href="#">[REDACTED]cocoon-live</a>	2 hours 9 minutes ( <a href="#">#1</a> )	N/A	21 seconds	
		<a href="#">[REDACTED]workflow</a>	2 hours 9 minutes ( <a href="#">#5</a> )	6 hours 24 minutes ( <a href="#">#1</a> )	15 seconds	
		<a href="#">hippo-cms-[REDACTED]</a>	2 hours 8 minutes ( <a href="#">#9</a> )	N/A	6 minutes 32 seconds	
		<a href="#">hippo-repository-1.2.13</a>	5 hours 37 minutes ( <a href="#">#1</a> )	N/A	2 minutes 20 seconds	
		<a href="#">repository-initialization</a>	N/A	9 minutes 40 seconds ( <a href="#">#24</a> )	2 seconds	
		<a href="#">workflows-[REDACTED]</a>	5 hours 10 minutes ( <a href="#">#1</a> )	N/A	22 seconds	

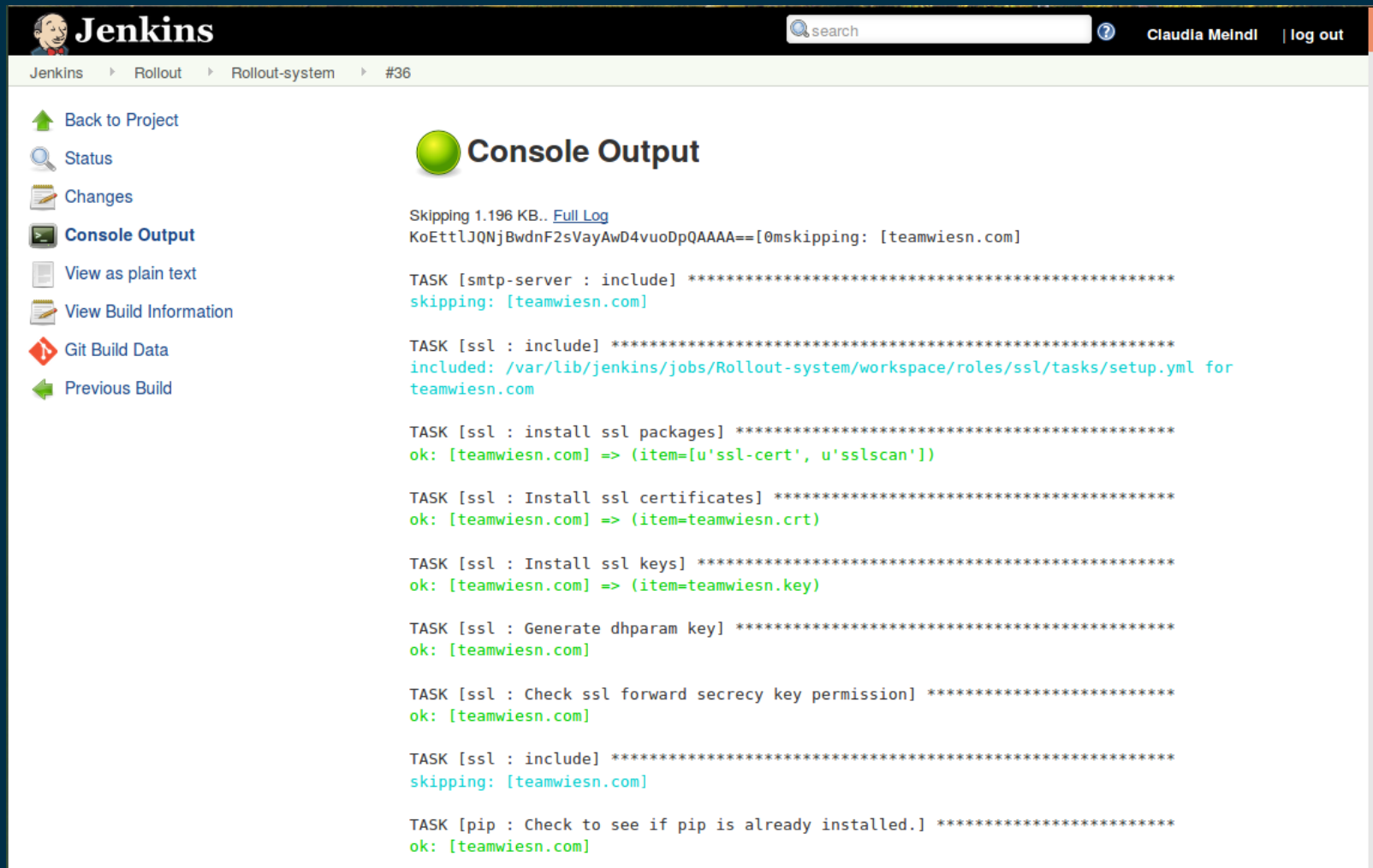
Icon: [S](#) [M](#) [L](#)

[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' and 'KoEttlJQNjBwdnF2sVayAwD4vuoDpQAAAA==[0mskipping: [teamwiesn.com]]'. The output consists of several task blocks, each starting with 'TASK [task : include]' followed by a series of asterisks. The tasks are: '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]', and 'TASK [ssl : include]'. Each task block ends with 'ok: [teamwiesn.com] => (item=[u'ssl-cert', u'sslscan'])' or 'skipping: [teamwiesn.com]'. The final task is 'TASK [pip : Check to see if pip is already installed.]' which also ends with 'ok: [teamwiesn.com]'.

Jenkins

Search

Claudia Meindl | log out

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)

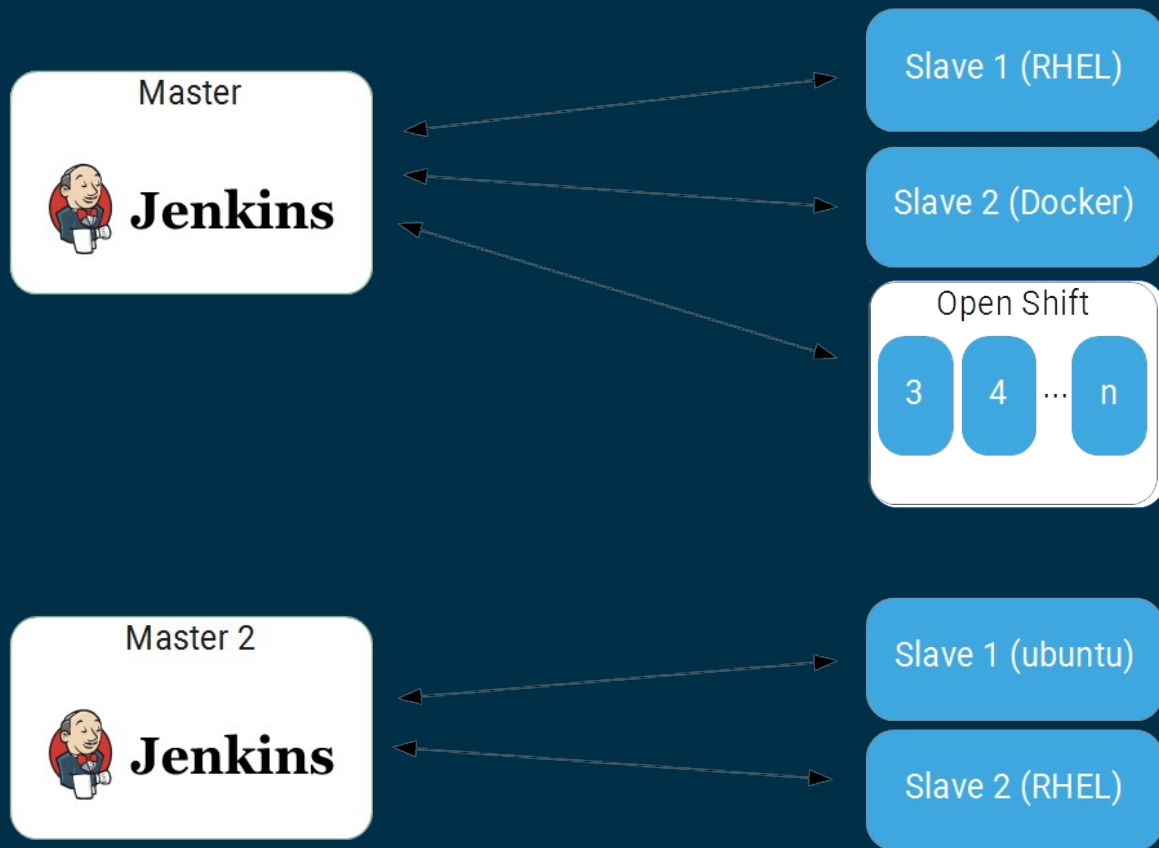
TASK [ssl : Generate dhparam key] \*\*\*\*\*  
ok: [teamwiesn.com]

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' }  
    }  
    ...  
}
```



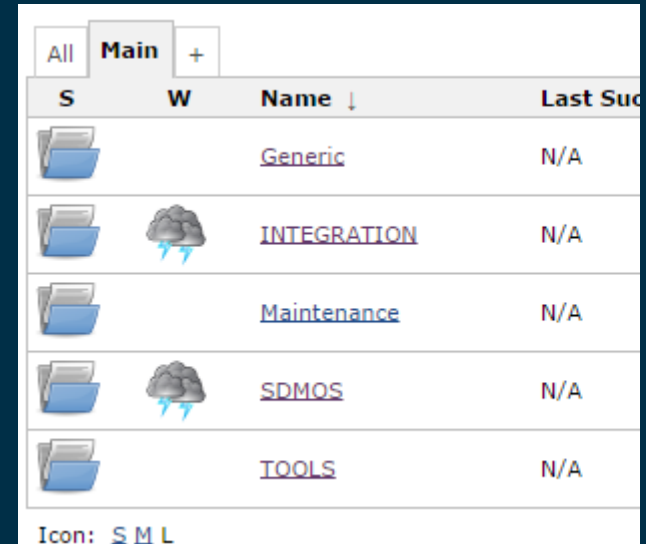






# Best Practices

- 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

All	Main	+		
S	W	Name ↓	Last Suc	
		<u>Generic</u>	N/A	
		<u>INTEGRATION</u>	N/A	
		<u>Maintenance</u>	N/A	
		<u>SDMQS</u>	N/A	
		<u>TOOLS</u>	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 {
        successful {
            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

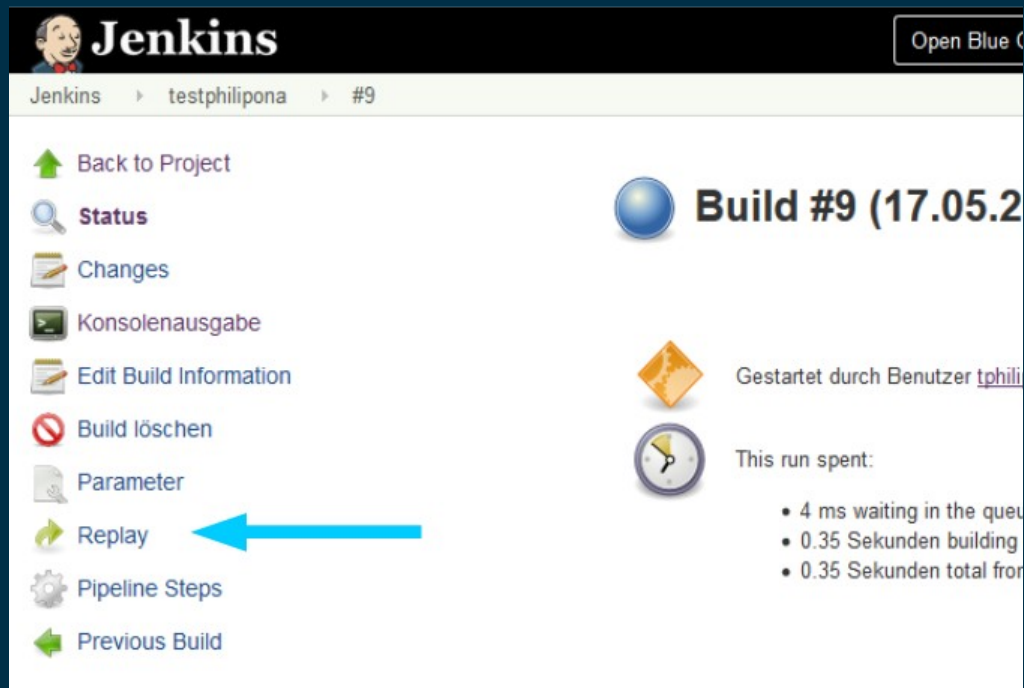
Example : [Puzzle Jenkins Shared Library \(GPL-3.0\)](#)

# 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')
      }
    }
  }
  ...
}
```

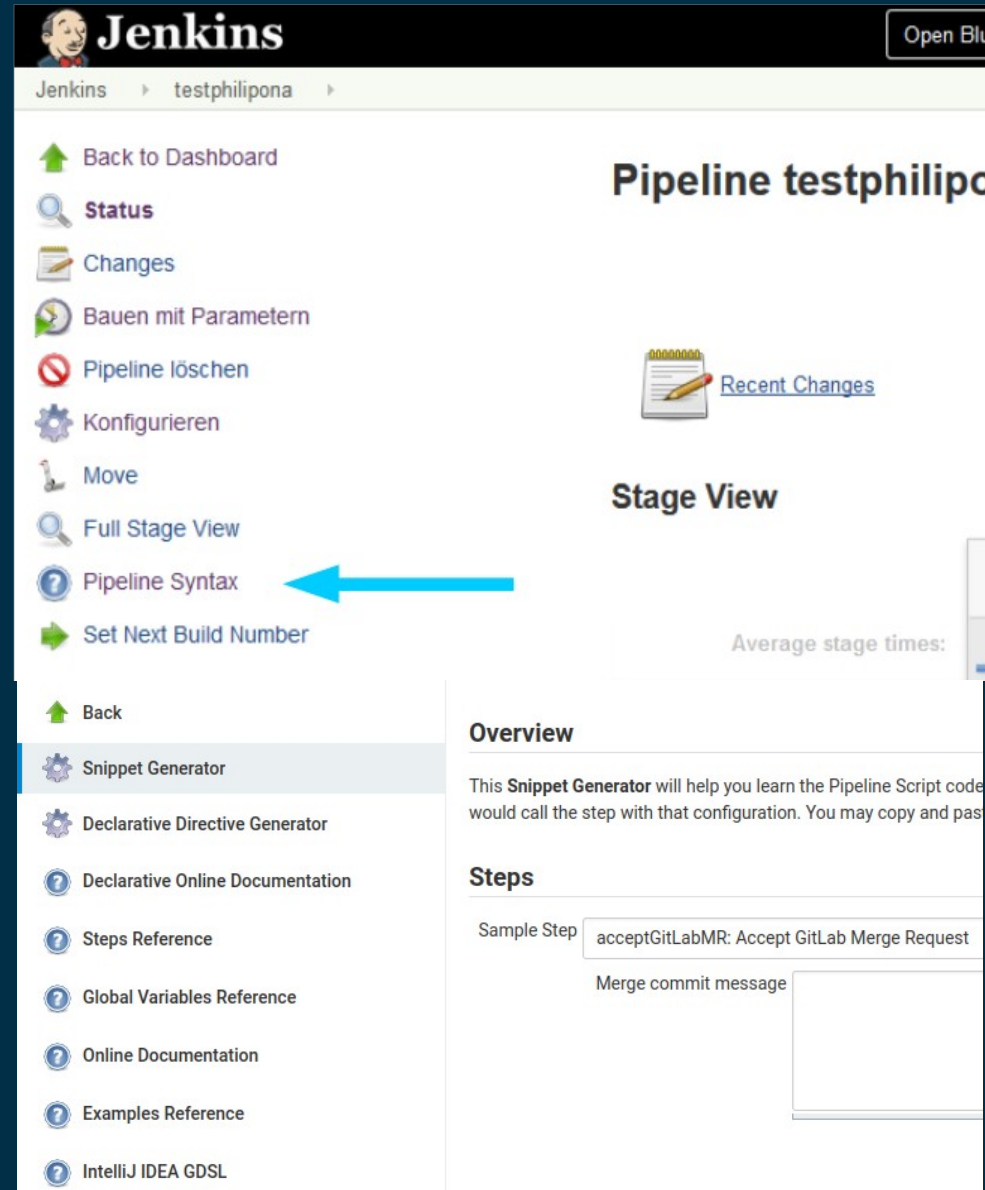
# Resources

In Jenkins :

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

Online :

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



The screenshot shows the Jenkins web interface. At the top, the Jenkins logo and name are visible. Below the header, there's a breadcrumb trail: Jenkins > testphilipona >. A sidebar on the left contains a list of navigation links: Back to Dashboard, Status, Changes, Bauen mit Parametern, Pipeline löschen, Konfigurieren, Move, Full Stage View, Pipeline Syntax (highlighted with a blue arrow), and Set Next Build Number. Below this, there's a section for the 'Snippet Generator' with links to Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main content area is titled 'Pipeline testphilipona' and includes a 'Recent Changes' link with a notepad icon, a 'Stage View' section, and an 'Overview' section. The 'Overview' section contains text about the Snippet Generator and a 'Steps' section with a 'Sample Step' field containing 'acceptGitLabMR: Accept GitLab Merge Request' and a 'Merge commit message' field.



Questions?